ENSEM

ÉCOLE DE L'ÉNERGIE

École National Supérieur d'électricité et de Mécanique

Contrôle d'un agent via apprentissage par renforcement

Auteur:
Makdoud Nizam

Tuteurs:
Jerome Kodjabachian
Marc Schoenauer
Alexandre Kazmierowski

September 1, 2017

THALES

Inria
INVENTEURS DU MONDE NUMÉRIQUE

Remerciements

Je tiens à remercier chaleureusement toutes les personnes qui m'ont accompagnées durant ce stage pour faire de ces six mois une expérience profitable sur le plan professionnel et agréable sur le plan personnel.

Tout d'abord, je remercie Jérome Kodjabachian, Marc Shoenauer et Christophe Meyer pour m'avoir donné l'opportunité d'effectuer ce stage, de m'avoir conseillé et permis de m'épanouir durant ces six mois.

Je remercie également Alexandre kazmierowski pour son soutien, ses conseils assidus et sa disponibilité sans laquelle je n'aurais pas pu avancer sur certains points de ce stage.

Enfin, je tiens à remercier l'ensemble de l'équipe AS&BSIM, pour sa sympathie et sa disponibilité.

Abstract

Artificial intelligence is a growing field which aims is to improve task automation. Through this internship, a subfield of artificial intelligence named reinforcement learning has been explored.

Reinforcement learning deals with issues governed by a set of rewards. For example, the classical problem in reinforcement learning is the control of agent through one place to another. The agent receives a reward when it arrives to the final place. The goal of our control is to find the optimal sequence of actions that maximises the cumulated rewards, which is, for our example, to reach to goal within the shortest time.

THALES is involved in the infrastructure security design, and aims to ensure the security of infrastructures by created plans to scenarios generated by SE-STAR. SE-STAR is a simulation capable of created multiple scenarios which may involve a large number of agents. SE-STAR uses internals drives (like hunger, fear, stress ...) to control the mass of agents but these internal drives has to be carefully design to accomplish a scenario (like a fight in a gare).

We choose another way of controlling agents that uses only rewards to automatically find the optimal behaviour that correspond to a given scenario. Notably, we have worked on the control of an agent in a simulated environnement created by the thales' internal biomimetic simulation SE-STAR.

We will propose various architectures and reinforcement learning algorithms to allow an agent to move in its simulated environnement by its own internal motivations to find our specified goal. The novelty of this approch is to find a policy with a weakly informative input like the partial vision of the agent which banned the use of other traditionnal control algorithms.

Non-exhaustively, the internship has been divided in five periods:

- Bibliographic and litterature search.
- Application of main algorithms on simple environnements.
- Development of reinforcement learning algorithms on 2D and 3D environnements used by researchers to benchmark their controllers.
- Design of a wrapper around the simulation SE-STAR to support the use of our external algorithm.

- Research of algorithms based on intrinsic motivations (or curiosity) to help the agent to find the optimal policy in difficult 3D environnements like those created by SE-STAR.

Résumé

L'apprentissage automatique est un domaine subissant actuellement une forte expansion, permettant une automatisation de nombreuses tâches. A travers ce stage, nous avons exploré une sous partie de l'apprentissage automatique qui se nomme l'apprentissage par renforcement profond.

Ce domaine vise à résoudre des problèmes qui sont régis par des récompenses. Par exemple, nous pouvons imaginer le problème d'un agent ayant pour but d'aller d'un point A à un point B, une récompense serait donné à l'agent dès qu'il attendra le point B. Notre objectif sera de trouver la politique optimale de l'agent pour maximiser la quantité de récompense reçue. Dans notre exemple, l'agent aura à apprendre la séquence d'actions qui permettra d'aller jusqu'au point B, et ainsi d'obtenir le maximum de récompenses. En particulier, nous travaillerons à l'aide d'un logiciel de simulation bio-inspiré qui se nomme SE-STAR.

THALES est impliqué dans le design d'infrastructures critiques. Elle a pour objectif de garantir la sécurité des infrastructures. Pour cela, elle utilise le logiciel SE-STAR qui est une simulation ayant pour but de créer des plans répondant à des scénarios possibles dans l'infrastructures concernées. SE-STAR a la capacité de gérer un grand nombre d'agents simultanément. Pour contrôler ces individu, SE-STAR utilise des motivations (faim, stress, peur ...), néanmoins, pour contrôler une masse d'individus, il faut être précis dans la conception de ces motivations pour accomplir un scénario.

Nous avons choisi une autre voie pour contrôler les agents qui n'utilise que la spécification de récompenses pour générer un comportement correspondant à celui souhaité pour un scénario. Nous avons donc travaillé sur le contrôle d'un agent dans une simulation crée par SE-STAR.

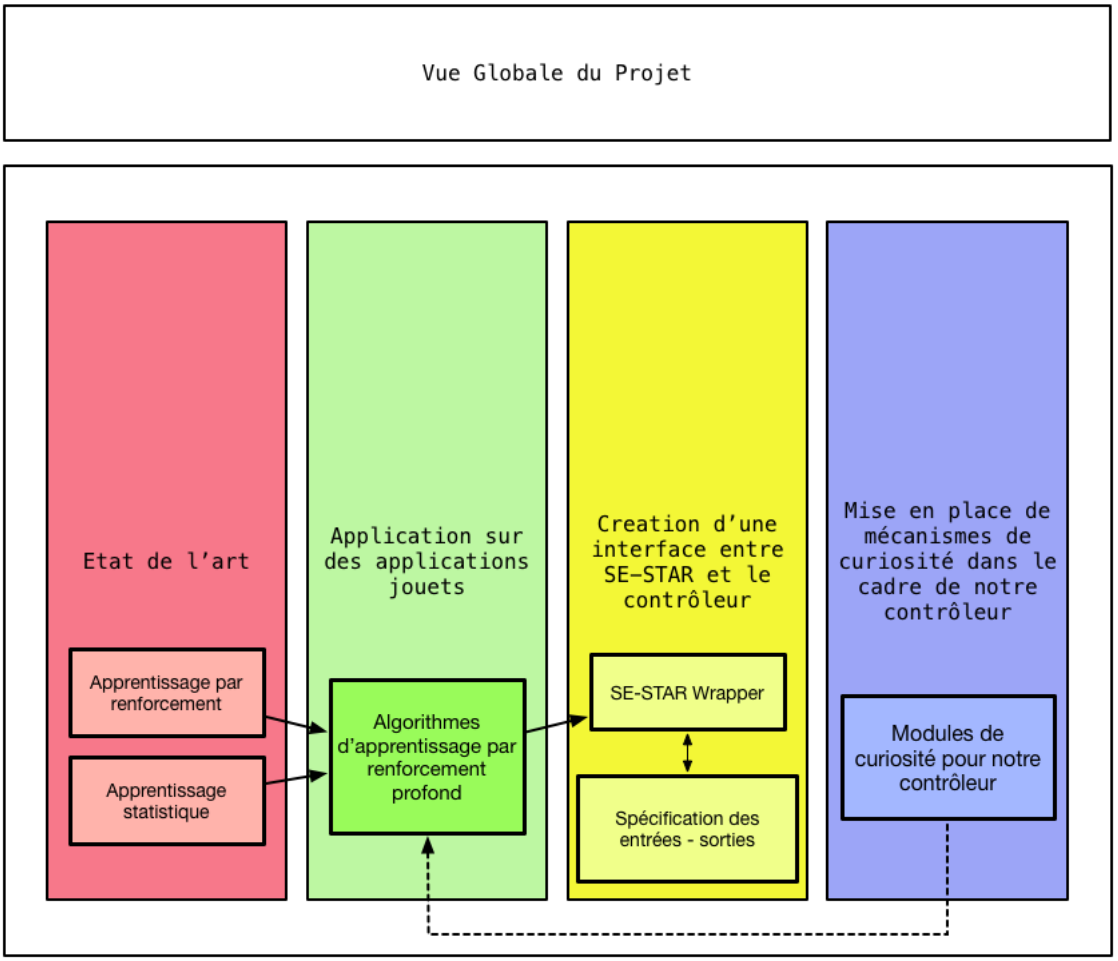
Nous proposerons une architecture et des algorithmes d'apprentissage par renforcement permettant à un agent de se déplacer dans l'environnement simulé à la recherche de la sortie de façon non supervisée. L'intérêt de notre approche est qu'il utilise des entrées extremment bruitées et partielles (la vision de l'agent) ce qui empêche toutes stratégies de contrôles usuellement appliquées.

De manière non exhaustive, nous pouvons définir ce stage en cinq périodes:

- Découverte et recherche bibliographique.
- Application des principaux algorithmes sur des cas simples.

- Élaboration d'algorithmes de renforcements profonds sur des environnements 2D et 3D utilisés par les chercheurs pour comparer la performance de leurs méthodes
- Conception d'un outil autour de SE-STAR pour supporter les algorithmes d'apprentissage par renforcement.
- Recherche autour d'une solution adaptée spécifiquement à l'apprentissage par renforcement profond dans le cadre problématique des environnements labyrinthiques.

Ci dessous une représentation des grandes périodes pendant ce stage:



Liste des figures

Table des matières

1 Introduction

Dans le cadre de ma formation d'ingénieur à l'*École Nationale Supérieure d'électricité et de Mécanique*, j'ai réalisé un stage du 2 février 2017 au 2 août 2017 au sein du laboratoire ThereSIS de THALES Services à Palaiseau, dans le campus de Saclay. J'ai eu l'occasion pendant ce stage d'effectuer de nombreux séjours à l'INRIA Tao pendant lesquels j'ai pu m'entretenir avec des chercheurs en apprentissage par renforcement et me tenir au courant des avancées dans le domaine de l'intelligence artificielle.

J'ai réalisé ce stage au sein de l'équipe ASBSIM (Adaptative Systems and Biomimetic Simulation), qui travaille principalement sur le logiciel **SE-STAR**. Il s'agit d'un environnement synthétique où évoluent de nombreux agents virtuels, chacun disposant d'un modèle cognitif.

Dans le cadre de ces simulations, je me suis intéressé à la navigation d'un agent en utilisant l'apprentissage par renforcement profond pour contrôler l'agent dans son environnement. En particulier, la contrainte imposée sur le contrôle de l'agent est qu'en entrée du contrôleur, nous n'avions accès qu'à une image 2D d'un environnement 3D (en l'occurrence SE-STAR).

1.1 Contexte du stage et de l'entreprise

Le groupe THALES est une multinationale française employant 67 000 salariés dans 56 pays. Si l'entreprise est connue auprès du grand public pour sa présence sur le marché de la défense, elle opère aussi sur plusieurs marchés duaux (à la fois civils et militaires) : l'aéronautique, l'espace et la sécurité, ainsi que sur le marché civil du transport terrestre. Les problématiques de traitement et interprétation des données ainsi que d'aide à la prise de décision sont au cœur de l'activité du groupe. Afin de renforcer la position du groupe dans l'innovation autour des nouvelles technologies, les activités de recherche et développement sont au cœur de l'entreprise, représentant 20% de ses revenus. THALES possède un réseau international de laboratoires qui coopèrent avec des universités et des laboratoires de recherche publics.

THALES Services est une division de THALES, dont l'activité se situe autour des systèmes d'information et de communications sécurisés. ThereSIS (THALES European Research for E-Government Secured Information Systems) est le laboratoire de recherche en systèmes d'information de THALES Services. Il est situé sur le site

de THALES Research and Technology à Palaiseau. Les équipes en place ont pour mission de tester des technologies innovantes, afin de déterminer si elles peuvent être exploitées au profit du groupe THALES et de ses clients. Des démonstrateurs sont mis en place pour présenter aux clients des cas concrets d'utilisation de ces technologies novatrices, parmi lesquelles on peut trouver le cloud computing, les systèmes de vidéosurveillances, les simulateurs.

Depuis quelques années, ThereSIS a mis l'accent sur les technologies autour de l'intelligence artificielle, et propose une expertise dans ce domaine. Cela permet à THALES d'être un acteur de poids dans les domaines de de la cybersécurité et de la sécurité des infrastructures critiques (gares, aéroports, ...).

ThereSIS propose des projets de recherche ou répond à des appels d'offres émanant d'entreprises ou de groupes finançant la recherche industrielle. L'intérêt des clients pour les prototypes présentés peut mener à une phase d'industrialisation et de commercialisation prise en charge par les départements de production de THALES Services.

En quelques chiffres, voici une vue de la taille de THALES et de l'importance donnée à la RD:

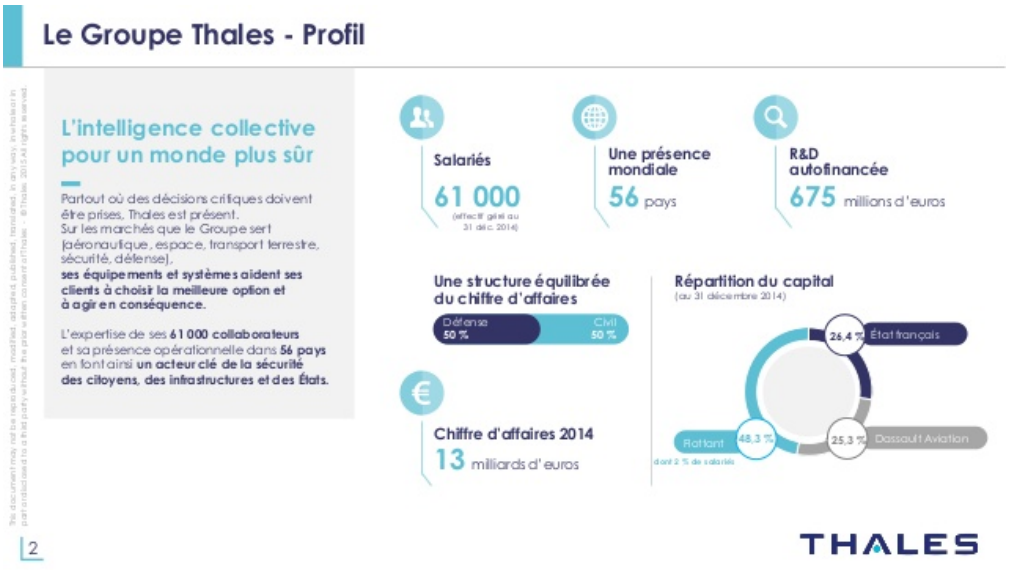


Figure 1: THALES en quelques chiffres

1.2 Description du sujet de stage

1.2.1 L'environnement de simulation: SE-STAR

SE-STAR est un logiciel de simulation dans des environnements modélisés à partir de lieux réels. Dans l'environnement virtuel interagissent en temps réel des agents représentant des personnes. Il est conçu en particulier pour les infrastructures critiques (stations de métro, gares, aéroports). Il peut gérer en temps réel un grand nombre d'agents (plus de 10 000), permettant de nombreuses interactions avec l'environnement qui peut être facilement modifié par l'utilisateur (ajout d'objets tels que des caméras de surveillance, des distributeurs de boissons, des bancs, etc.). La simulation cherche à représenter de manière réaliste le comportement des usagers de ces infrastructures.

Dans le contexte de ce stage, nous allons restreindre SE-STAR à la modélisation d'un environnement simple contenant un seul agent.

1.2.2 Objectif du stage et contraintes identifiées

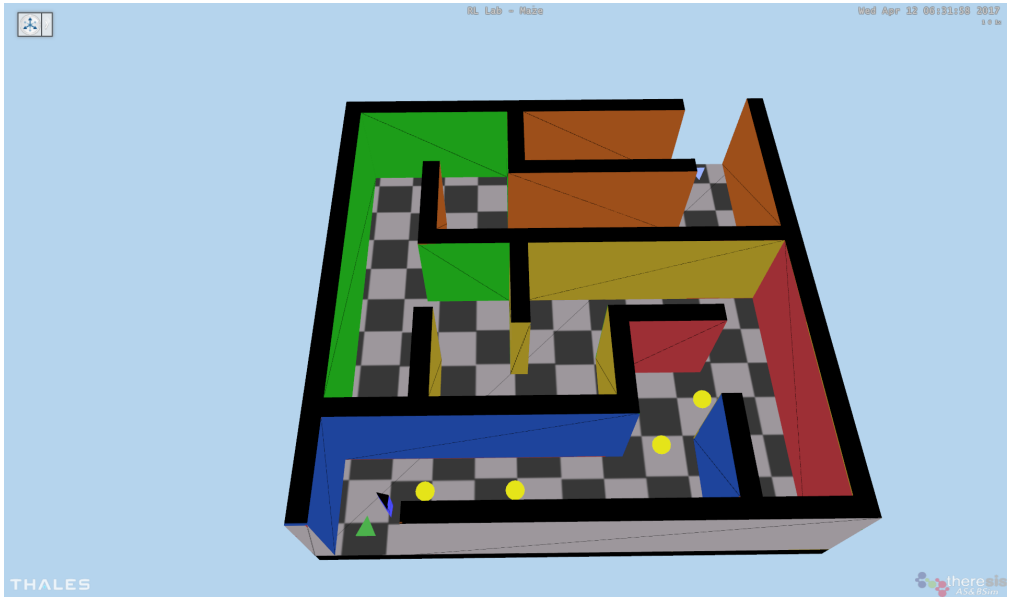
L'objectif de ce stage est de proposer un contrôle d'un agent de l'entrée de l'environnement à la sortie de l'environnement en évitant des zones considérées comme dangereuses. Néanmoins, nous possédons des contraintes sur notre contrôle:

1. **L'environnement est inconnu de l'agent:**

   La première contrainte de notre objectif est que notre environnement est inconnu. En effet nous souhaitons développer un algorithme de contrôle que soit capable de proposer un asservissement fiable pour tout environnement proposé par SE-STAR. L'impératif de généralisation est une contrainte forte pesant sur le choix technologique pour réaliser ce contrôle. En effet, un asservissement classique repose sur la connaissance de la dynamique du système qui n'est pas disponible dans notre cas et , pire encore, est fluctuante en fonction des environnements.

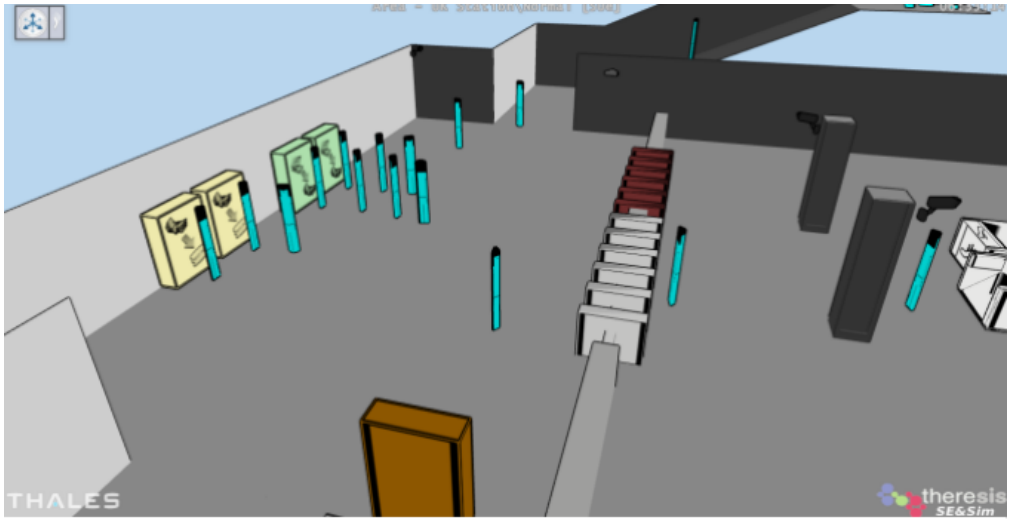
2. **Le contrôle doit être robuste aux changements d'environnements:**

   Pour être utile, il faut que notre agent apprenne à se mouvoir d'un point A à un point B. De plus, notre algorithme d'apprentissage ne doit pas se suffire à apprendre par coeur la séquence d'action pour rejoindre la sortie mais doit apprendre à se repérer et voir les points d'intérêts permettant de trouver la



.5

Figure 2: Environnement créé pour le contrôle d'un agent



.5

Figure 3: Simulation d'une station de métro

Figure 4: Exemple d'utilisation de SE-STAR

sortie quelque soit l'environnement. Nous recherchons une approche analogue à la façon humaine pour sortir d'un environnement (longer les murs, chercher des indices visuels indiquant la sortie ...).

3. **Les entrées de la commande sont données par la vision de l'agent:**

Une des difficultés qui empêche l'utilisation des outils de la théorie du contrôle, vient du format des entrées. Nous devons bâtir un algorithme de contrôle se basant sur des images partielles d'un environnement. Comme un humain lâché dans un labyrinthe, il n'aura pas accès à une image complète du labyrinthe mais uniquement une vue partielle possiblement bruitée. Notre agent devra, à partir d'une information incomplète, réussir à déterminer une séquence d'actions lui permettant de trouver la sortie.

1.2.3 Défis et choix technologiques

Nous cherchons un algorithme de contrôle assez générique pour ne nécessiter en entrée que des images (tableaux de pixels). Un des défi de ce stage est de permettre à l'agent d'être capable de trouver les repères les plus adaptés pour réussir sa mission quelque soit l'environnement.

Cela implique que nous ne recherchons pas seulement une solution adaptée à un environnement particulier. Nous souhaitons un agent capable de généraliser à un ensemble d'environnements similaires.

Les nombreuses contraintes et les défis technologiques impliquent une décomposition de l'algorithme choisi en deux grandes étapes:

1. L'agent explore l'environnement et apprend simultanément une politique optimale pour arriver à la sortie.

2. Nous nous assurons que l'agent est capable de trouver une bonne politique sur des environnements similaires pour vérifier que l'agent n'a pas juste *appris par coeur une solution*[1].

Se pose naturellement la question du choix de la technologie pour réaliser cet apprentissage. La contrainte de généralisation nous pousse à utiliser l'apprentissage profond (ou *deep learning*). L'apprentissage profond a prouvé son efficacité dans sa capacité à être pertinent à un grand nombre d'entrées possibles. La distinction entre généralisation et apprentissage par coeur des données est particulièrement importante dans ce domaine. Les méthodes qui en sont issues ont prouvé leurs capacités à trouver des solutions à des problèmes nouveaux (jamais utilisés en entrainement).

Il nous faut un domaine s'attaquant à des environnements compliqués, observés de manières partielles, et possiblement changeant au cours du temps. L'apprentissage

[1]Nous reviendrons plus tard sur la signification de la généralisation qui est un problème central

par renforcement propose une manière d'aborder ces d'environnements complexes.

Ainsi, nous avons proposé une méthode alliant l'apprentissage profond pour gérer les entrées du système avec l'apprentissage par renforcement pour réaliser un agent capable de naviguer dans un environnement pour y découvrir une sortie en ayant comme donnée unique sa vision.

2 Revue de la littérature et état de l'art

Dans cette partie, nous allons passer en revue les principales explications du fonctionnement de notre contrôleur. En particulier, nous expliquerons comment fonctionne l'apprentissage par renforcement, les intérêts de cette technologie et ses limites.

Nous mettrons en relation l'apprentissage par renforcement avec l'apprentissage profond (deep learning). Enfin, nous discuterons des moyens mis en place pour tester nos algorithmes sur des cas moins complexes pour s'assurer du fonctionnement de nos algorithmes.

2.1 Fonctionnement de l'apprentissage par renforcement et intérêt

L'apprentissage par renforcement est une sous partie de l'apprentissage automatique. Il fait référence à un apprentissage se basant sur des des expériences de façon à optimiser la quantité de récompenses récue. A chaque action effectuée par l'agent, l'environnement attribut à celui ci une récompense. L'objectif est de trouver la séquence d'actions qui mène à la plus grande accumulation de récompenses. L'agent va être uniquement guider par son expérience et devra apprendre de ses erreurs. Imaginons le cas suivant: nous devons apprendre un nouveau tour à notre animal néanmoins, il est impossible de lui exprimer clairement ce que nous souhaitons de lui. Notre seul levier est la quantité de récompenses ou au contraire de punitions que va recevoir notre animal. Ce problème est connu sous le nom de: *problème d'assignement du crédit*.

Une des particularités de l'apprentissage par renforcement réside dans le fait que notre politique d'actions influence grandement les états rencontrés. Prenons l'exemple d'un agent autonome dans un labyrinthe à la jonction entre deux salles différentes. Si la séquence d'actions fait se retrouver notre agent dans une des salles, il n'aura accès qu'a la vision de cette salle.

2.1.1 Formalisation de l'approche par renforcement

Nous allons proposer un formalisme permettant d'introduire les notions importantes de l'apprentissage par renforcement. Nous commençerons par expliquer de façon simple les notions indispensables puis nous nous servirons des processus de

décision markoviens pour avoir un cadre solide théorique.

Nous considérerons:

- Un ensemble d'états noté $\mathbb{S}$

  Cette ensemble d'états regroupe l'ensemble des états possibles pouvant être fournis par l'environnement. Dans le cas du jeux d'échecs, cela correspondrait au nombre de positions possibles atteignables ($\sim 10^{43}$ soit un nombre supérieur au nombre d'atomes dans l'univers)

- Un ensemble d'actions noté $\mathbb{A}$

  Cette ensemble d'actions regroupe les actions possibles par l'agent en fonction de l'état dans lequel il se trouve. Il est possible d'imaginer des états pour lesquels l'ensemble des actions soit différent. Dans le cas des échecs les actions possibles peuvent différer en fonction de l'état de la partie (exemple: clouage, roi en échec ...).

- Un ensemble de récompenses noté $\mathbb{R}$

  Cette ensemble regroupe les récompenses que peut fournir l'environnement. Ces récompenses peuvent être nul, négative ou positive (nous parlerons de récompense quelque soit la valeur de celle-ci). Les récompenses dépendent à la fois de l'état mais aussi de l'action effectuée par l'agent.

  Selon le cas, il peut être à l'expérimentateur de définir la spécification des récompenses mais dans de nombreux cas l'ensemble des récompenses se fixe facilement. Dans le cas de notre labyrinthe l'agent reçoit +1 s'il trouve la sortie 0 sinon.

- La dynamique de l'environnement $\mathbb{T}$

  La dynamique de l'environnement représente la probabilité, sachant que j'effectue une action $\boldsymbol{a} \in \mathbb{A}$ et que je suis d'en un certain état $\boldsymbol{s_t} \in \mathbb{S}$, d'atterrir dans un nouvelle état $\boldsymbol{s_{t+1}} \in \mathbb{S}$. C'est une connaissance sur le modèle, qui dans bien des cas, nous fait défaut. En l'occurrence pendant ce stage, la dynamique est une donné non connue. Il est à noter que ce qui est précédemment énoncé n'est vrai que sous l'hypothèse de Markov qui nous assure que notre dynamique est conditionnellement indépendante de des états et actions précédents. Cette hypothèse est fausse en pratique et nous discuterons des stratégies utilisées pour pallier à ce problème.

- La politique d'actions $\boldsymbol{\Pi}$

La politique d'actions peut être définie comme la probabilité, sachant que je suis dans un certain état $\boldsymbol{s_t} \in \mathbb{S}$, d'effectuer une action $\boldsymbol{a} \in \mathbb{A}(\sim_{\approx})$. Dans un soucis de généralité, nous considérons une politique stochastique néanmoins la politique peut être déterministe. En apprentissage par renforcement nous cherchons à optimiser notre politique d'action dans le but de maximiser la quantité de récompenses reçus (pondérée de manière à un mettre un poid plus important aux récompenses obtenues récemment).

Ainsi un problème d'apprentissage par renforcement profond classique suit le schéma suivant. A chaque pas de temps t, l'agent va percevoir un état $\boldsymbol{s_t} \in \mathbb{S}$. L'agent va choisir l'action à effectuer $\boldsymbol{a} \in \boldsymbol{A(s_t)}$ et va recevoir une récompense $\boldsymbol{r_{t+1}}$. Notre objectif va donc être de maximiser la somme pondérée des récompenses obtenues par notre séquence d'actions.

On définira notre problème sous la forme d'un processus décisionnel de Markov, soit un 5-tuple $\boldsymbol{(\mathbb{S}, \mathbb{A}, \mathbb{T}, \mathbb{R}, \gamma)}$. Le paramètre γ qui intervient dans la définition de notre problème est important car c'est ce paramètre qui va pondérer l'importance d'un récompense au cours du temps. Il existe de multitudes choix pour la spécification de ce paramètre, néanmois usuellement γ sera égale à .99. Cela pour deux raisons, la première est que nous souhaitons que les récompenses tardives soient prisent en compte mais de façon plus faible que celle qui sont immédiates. Deuxièmement, pour des raisons de stabilité, il est important théoriquement que le paramètre γ soit inférieur à 1.

Notre objectif est de trouver la politique $\boldsymbol{\Pi}$ tel que la somme pondérée des récompenses (notée $\boldsymbol{G_t}$) soit maximale avec:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad \text{sachant} \quad a_t = \Pi(s_t) \tag{1}$$

Pour déterminer la politique optimal, il est utile de définir la fonction d'états $\boldsymbol{V(s_t)}$ qui représente le gain moyen à partir d'un certain état en suivant une politique $\boldsymbol{\Pi}$ et la fonction d'états-actions $\boldsymbol{Q(s_t, a_t)}$ qui représente le gain moyen à partir d'un certain état et en effectuant une certaine action puis en suivant une politique $\boldsymbol{\Pi}$.

$$V(s_t) = \mathbb{E}_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, s_t = s \right) \tag{2}$$

$$Q(s_t, a_t) = \mathbb{E}_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, s_t = s, a_t = a \right) \tag{3}$$

Une propriété fondamentale des fonctions d'état et d'état-action est qu'ils respectent une relation de récursivité bien connu: l'équation de Bellman (pour V et Q):

$$V_\pi(s_t) = \sum_a \underbrace{\pi(x\ ,\ a)}_{p\ (a\ |\ s)} \sum_{s',r} \overbrace{T(s',r,s,a)}^{p(s',r|s,a)} \Big[\ r + \gamma V_\pi(s')\ \Big] \tag{4}$$

$$Q_\pi(s_t, a_t) = \sum_{s',r} \overbrace{T(s',r,s,a)}^{p(s',r|s,a)} \Big[\ r + \gamma V_\pi(s')\ \Big] \tag{5}$$

les relations précédentes sont particulièrement intéressantes car elles nous permettent d'exprimer la valeur d'un état à partir de la valeur des états successifs à celui ci. Ces deux relations sont à l'origine de nombreuses méthodes en apprentissage par renforcement. Notamment, elles seront à l'origine des méthodes appliquées pendant ce stage dans le contrôle d'un agent dans la simulation de Thales SE-STAR.

Pourtant, nous n'avons toujours pas explicité la forme de la politique optimale. Nous allons nous aider des fonction d'état, et d'état-action qui fournissent un indicateur sur la pertinence de la politique choisie. Ainsi, on pourra dire qu'une politique π est meilleurs qu'une politique μ si $V_\pi(s) > V_\mu(s), \forall s$. Il parait donc intéressant de rechercher la politique maximisant la fonction d'état $V_* = \max_\pi V_\pi(s)$. On peut faire la même chose avec la fonction d'état-action: $Q_* = \max_\pi Q_\pi(s,a)$.

Nous pouvons expliciter Q^* et V^* correspondant aux équations optimales de Bellman respectives:

$$V^*(s) = \max_a \sum_{s',r} p(s',r|s,a) \Big[\ r + \gamma V_\pi(s')\ \Big] \tag{6}$$

$$Q^*(s,a) = \sum_{s',r} p(s',r|s,a) \Big[\ r + \gamma \max_{a'} Q_\pi(s',a')\ \Big] \tag{7}$$

A partir des fonctions état et état-action, il est relativement aisé de déterminer la politique optimale. Une fois qu'on a déterminé V^* nous pouvons déterminer

l'action optimale en faisant une recherche à un pas dans le temps. Ce qui est remarquable est qu'en apprentissage par renforcement, il y a un compromis fondamental entre un objectif à court et long terme pourtant nous venons d'affirmer qu'en exploitant V^* et en faisant notre recherche à un pas nous avons pris en compte des effets à long termes. Avec la fonction d'état-action c'est encore plus simple car il n'y a plus besoin de recherche auxiliaire car la fonction d'état-action encode déjà cette recherche, il suffit donc d'exploiter la fonction d'état-action optimale à chaque état pour obtenir la politique optimale: $\pi^*(s) = \max_a Q^*(s, a)$

Malheureusement, il est, dans la plupart des cas, impossible de déterminer les fonctions d'état ou d'état-action optimales car cela reviendrait à une recherche exhaustive de toutes les possibilités avec la connaissance de leurs probabilités d'apparaître et la connaissance du gain espéré associés. Néanmoins, les fonctions d'état et d'état-action sont utiles à l'apprentissage et au contrôle de notre agent. Dans la partie suivante nous allons expliquer comment approximer ces fonctions pour pouvoir commencer à établir des algorithmes de contrôle efficaces.

En résumé, l'apprentissage par renforcement propose de contrôler un agent grâce à des stimulis (récompenses) données par l'environnement ou l'expérimentateur.

L'objectif est de déterminer une politique (soit un fonction $\mathbb{S} \rightarrow \mathbb{A}$) qui maximise le gain cumulé pondéré (pondération utile pour favoriser l'importance des récompenses à court termes tout en prenant en compte les récompenses à long termes).

Pour déterminer la politique optimale, nous nous aidons de deux fonctions (les fonctions d'état et d'état-action) qui sont une mesure de la quantité de récompenses pouvant être espéré à un certain état (ou à un état en effectuant une certaine action) en suivant une politique.

On peut dès lors en trouvant une fonction d'état (ou d'état-action) optimale dérivé une politique optimale.

Sauf cas jouet, les fonctions d'état et d'état-action sont inconnues, notre objectif va être de trouver une façon de les approximer pour permettre l'établissement d'algorithmes de contrôle par apprentissage par renforcement.

Enfin, nous montrerons que nous ne sommes pas obliger de passer par ce intermédiare pour déterminer la politique optimale tout en insistant sur l'importance de la connaissance des fonctions d'états qui sont un atout quelques soit les approches utilisés.

2.1.2 Introduction des principales approches en apprentissage par renforcement

Après avoir introduit les principaux concepts de l'apprentissage par renforcement, il convient d'en extraire des algorithmes permettant de trouver notre politique optimale. Un difficulté majeure est que nous n'avons ni connaissance des fonctions d'états (et d'états-actions) ni connaissance de la dynamique du modèle (noté T).

La question est donc de savoir s'il est possible d'approximer ces fonctions, ou alors de s'en passer.

Avant de répondre à ces questions, il est important de faire la différence entre deux types d'approches en apprentissage par renforcement:

1. **Model Free** (Apprentissage sans modèle):
   Dans ce type d'approche, nous n'utilisons pas un modèle de l'environnement pour déterminer la politique optimale, uniquement une recherche basée sur notre expérience. Intuitivement, nous pourrions penser cette approche inférieure à l'approche *Model Based* pourtant nous verrons que l'approche *Model Free* est plus simple à mettre en place et est plus performante dans la plupart des cas que nous avons rencontrés. Cela s'explique par la difficulté d'apprentissage de la dynamique de l'environnement (qui dans bien est cas n'est pas stationnaire).

2. **Model Based** (Apprentissage avec modèle):
   Dans ce type d'approche, nous allons utiliser la connaissance du modèle (ou l'approximation interne du modèle par l'agent) pour déterminer la politique optimale. L'apprentissage avec modèle apparaît comme plus puissant mais est encore un problème non résolu (tout comme en Model Free mais des algorithmes ont déjà montré leur capacité à résoudre des environnements difficiles). La difficulté est que, bien souvent, l'entrée n'est pas très informative sur la dynamique de l'environnement. Ainsi, une image 2D représentant un monde 3D est une entrée dont il est difficile de tirer un modèle.

Dans cette partie, nous allons nous concentrer sur l'établissement d'algorithmes *Model Free*.

Précédemment, nous avons établi que nous n'avions aucune connaissances des fonctions d'états, ni de la dynamique de l'environnement. Notre objectif est de trouver une approximation des fonctions optimales d'états qui nous permettrait d'établir une politique optimale. Pour ce faire nous devons trouver un moyen d'approximer la fonction d'état et de trouver un moyen d'établir à quel point notre approxima-

tion est loin de la fonction optimale.

Nous expliquerons plus tard le mécanisme d'approximation de fonctions utilisé (réseau de neurones). Pour cette partie, nous considérerons juste la fonction $\omega, s \rightarrow \widetilde{V}_\omega(s)$ qui associe une matrice ω et un état à une approximation de la fonction d'état.

Nous pouvons ainsi formaliser, en un problème d'optimisation, notre objectif:

$$\omega^* = \min_\omega \left\| V^\pi - \widetilde{V}_\omega \right\|_2$$

Nous pouvons donc, en utilisant une méthode d'optimisation telle que la descente de gradient stochastique, déterminer un minimum local:

$$\begin{aligned} \omega_{t+1} &= \omega_t - \tfrac{1}{2}\,\alpha \nabla_\omega \left\| V^\pi - \widetilde{V}_\omega \right\|_2 \\ \omega_{t+1} &= \omega_t + \tfrac{1}{2}\,\alpha (V^\pi - \widetilde{V}_\omega) \nabla_\omega \widetilde{V}_\omega \end{aligned} \tag{8}$$

Cela nous pose un problème de taille car nous n'avons pas connaissance de la fonction d'état. Or, notre précédante équation repose sur la connaissance de celle-ci et il ne serait pas pertinant de chercher une approximation d'une fonction déjà connue. Il existe plusieurs méthodes pour l'approximer, nous allons en donner deux. Une méthode basée sur la méthode de Monte Carlo, et l'autre basée sur du *bootstrapping* (le bootstrapping étant basé sur des processus itératifs convergeant vers la valeur qui est souhaitée)

Approximations possibles de la fonction d'état	
Monte-Carlo	$\widetilde{v}(s_t) = \mathbb{E}_\pi\big[G_t \vert S_t = s\big] \sim \frac{1}{N} \sum_{k=0}^{N} \gamma r_{t+k}$
Bootstrapping	$v_{k+1} = \sum_a \pi(a,s) \sum_{s',r} p(s', r \mid s, a)\,\big[r + \gamma\, v_k(s')\,\big]$

Nous commencerons par envisager l'approche par bootstrapping. La première remarque est que le bootstrapping requiert la connaissance du modèle de l'environnement ce qui est particulièrement handicapant dans notre cas. Néanmoins, nous avons des garantis sur la convergence de notre approximation vers la fonction d'état. Ainsi, si nous sommes en capacité de connaître le modèle, il nous suffit d'itérer

pour connaître la véritable fonction d'état. Pourtant, le véritable problème que nous avons avec cette méthode est un problème qui apparaîtra souvent et qui est dénommé *la malédiction de la dimensionalité*. Plus la dimensionnalité de l'état est grande plus le coût de cette approche augmente (de façon exponentielle). Dans notre cas d'usage, l'état sera un tableau de pixels ce qui rend impossible l'utilisation de ce genre d'algorithmes.

Si l'approche précédente reste inapplicable, la philosophie du bootstrapping n'a pas à être jetée. Il n'est pas pertinent de rechercher à tendre vers la fonction d'état en itérant selon la formule de bellman. Cependant il sera intéressant de chercher à itérer d'une autre manière de façon à s'approcher de la véritable fonction d'état. Toute la question est de trouver une formule qui permet d'itérer (à faible coût) et qui garantie de tendre vers une approximation de la fonction d'état pertinente.

Passons maintenant à la méthode via *Monte Carlo*. Cette méthode s'intègre parfaitement dans l'esprit de l'apprentissage par renforcement. En effet, un agent va jouer un épisode et récupérer un ensemble d'actions, états et récompenses. A partir de cet ensemble, nous sommes en capacité d'approximer la fonction d'état tel que l'approximation de la fonction d'état pour un état s correspond aux récompenses obtenues dès la première visite de l'état s (*First visite Monte Carlo*). Dans le cas où nous allons jouer plusieurs épisodes, nous pourrons affiner notre approximation en approchant notre fonction d'état pour un état s comme étant la moyenne des récompenses obtenues dès la première visite de l'état s sur l'ensemble des épisodes joués. Les méthodes d'approximation de la fonction d'état possèdent quelques avantages par rapport au bootstrapping:

- Insensible à la dimension de l'état ce qui est intéressant dans notre cas.
- Nous pouvons générer des épisodes en commençant par les états nous intéressant pour estimer leur fonction d'état ce qui n'est pas possible.

Néanmoins, l'approche par Monte Carlo possède aussi des désavantages tels que le fait d'être obligé de jouer un épisode entier pour pouvoir estimer la fonction d'état. Cela pourra être problématique dans le cas où l'épisode est long . De plus, il parait pertinent de pouvoir itérer à partir d'estimations antérieures ce qui n'est pas le cas avec les méthodes de Monte Carlo.

2.1.3 TD Learning: la base des algorithmes pour le contrôleur

En se basant sur l'approximation décrite précédemment en utilisant l'approche de Monte-Carlo, pour un environnement non stationnaire (qui change en fonction du

temps), nous pouvons trouver une formule approximant la fonction d'état:

$$\widetilde{V}_\pi(S_t) \leftarrow \widetilde{V}_\pi(S_t) + \alpha\Big(G_t - \widetilde{V}_\pi(S_t)\Big) \tag{9}$$

Le problème de la formule ci-dessus est que, pour mettre à jour notre croyance sur la fonction d'état, nous sommes obligé d'attendre la fin d'un épisode.

Nous allons voir dans cette partie qu'il est possible de combiner les approches de Monte-Carlo et Bootstrapping pour garder les avantages dès deux méthodes tout en gommant leurs défauts. Pour cela, rappelons la formule de la fonction d'état:

$$\begin{aligned} V_\pi &= \mathbb{E}_\pi\Big(\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\,\Big|\, s_t = s\Big) \\ &= \mathbb{E}_\pi\big[G_t|S_t = s\big] \\ &= \mathbb{E}_\pi\big[r_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s\big] \end{aligned}$$

Nous pouvons en déduire une nouvelle formule pour estimer la fonction d'état:

$$\widetilde{V}_\pi(S_t) \leftarrow \widetilde{V}_\pi(S_t) + \alpha\Big(r_{t+1} + \gamma V_\pi(S_{t+1}) - \widetilde{V}_\pi(S_t)\Big) \tag{10}$$

Nous obtenons ainsi une formule se basant sur du bootstrapping car nous nous servons de la fonction d'état à S_{t+1} et nous n'avons toujours pas besoin de la connaissance du modèle (Monte-Carlo).

La formule précédemment constitue la base sur laquelle nous allons nous appuyer pour créer notre contrôleur. Néanmoins, si la fonction d'état est importante, il est plus intéressant d'approximer la fonction d'état action dans le contexte d'un contrôle.

Pour le contrôle, c'est donc la formule ci-dessous, qui se nomme SARSA (pour *state action reward state action* [2], qui sera utilisée:

$$\widetilde{Q}_\pi(S_t, A_t) \leftarrow \widetilde{Q}_\pi(S_t, A_t) + \alpha\Big(r_{t+1} + \gamma\widetilde{Q}_\pi(S_{t+1}, A_{t+1}) - \widetilde{Q}_\pi(S_t, A_t)\Big) \tag{11}$$

[2] voir la formule ci-dessous pour comprendre pourquoi

Il existe dans la littérature une pléthore de variations de SARSA en particulier, une des variations qui va nous intéresser est celle qui s'appelle Q-Learning [24]:

$$\widetilde{Q}_\pi(S_t, A_t) \leftarrow \widetilde{Q}_\pi(S_t, A_t) + \alpha\left(r_{t+1} + \gamma \max_{a'} \widetilde{Q}_\pi(S_{t+1}, a') - \widetilde{Q}_\pi(S_t, A_t)\right) \tag{12}$$

Malgré leur relative proximité, nous privilégierons le Q-Learning pour dès raison que nous expliquerons plus tard.

Nous allons finir cette partie sur le TD Learning par expliquer la relation entre l'approximation de la fonction d'état par optimisation d'une fonction de perte et le Q learning. Cette introduction est importante car c'est l'algorithme le plus simple qui a été utilisé durant ce stage sur différents environnements. Nous allons expliquer comment fonctionne le Deep Q Learning [14] [15] (soit le premier algorithme d'apprentissage par renforcement profond).

Néanmoins, nous n'expliquerons pas totalement cet algorithme car il requiert quelques connaissances en deep learning pour la compréhension du mécanisme d'approximation de la fonction. Pourtant, il est important de le mentionner maintenant car il découle du TD Learning et en particulier de l'algorithme du Q-Learning et qu'il est fondamental pour la compréhension des autres algorithmes utilisés durant ce stage.

2.1.4 Du TD Learning au Deep Q Learning

Un problème en apprentissage par renforcement est la représentation de la fonction d'état (d'état action). Naivement, nous pourrions imaginer un dictionnaire qui, à chaque état, garderait en mémoire la fonction d'état action pour chacune des actions. Néanmoins, dans le cas d'états représentés par des pixels, il faudrait une capacité astronomique de mémoire.

Nous préférons donc utiliser des fonctions paramétrées par un ensemble de poids (noté θ) tel que $f_\theta = \widetilde{Q} \sim Q$.

Précédemment, nous avions introduit une méthode par minimisation d'une fonction de perte pour permettre de trouver l'ensemble θ tel que $f_\theta = \widetilde{Q} \sim Q$.

Imaginons maintenant que notre agent joue dans l'environnement et récolte des transitions (un ensemble composé de l'état courant, de l'état précédent, de l'action

jouée et de la récompense obtenu que l'on notera e_i pour la ième transition). L'agent stockera ces transitions dans une mémoire que l'on notera $\mathcal{D}$. Soit $\mathcal{D} = \big(e_0, ..., e_N\big)$

La fonction de perte utilisée est:

$$L_i(\theta_i) = \mathbb{E}_{\mathrm{s,a}\sim\rho(.)}\Big[\big(y_i - f_\theta(s, a; \theta)\big)^2\Big] \tag{13}$$

$$y_i = \mathbb{E}_{\mathrm{s'}\sim}\big[r + \max_{a'} f_{\theta_{i-1}}(s', a')\big] \tag{14}$$

Avec $\rho(s, a)$ la distribution sur la séquence d'états et d'actions.

2.1.5 Architecture de base A2C/A3C utilisée pour le contrôle

Après avoir représenté le *Deep-Q-Learning*, nous allons introduire une architecture classique en apprentissage par renforcement profond qui se dénome A3C[13][3]. Pour arriver à expliquer le fonctionnement de l'A3C, il convient de partir du constat qu'il serait aisé de trouver la politique optimale (paramétrisée par un ensemble de poids ω) en maximisant une fonction de performance (notée η). Formellement cela reviendrais à mettre à jour les poids ω selon ;a formule suivante:

$$\omega_{t+1} = \omega_t + \alpha\nabla\eta(\omega_t) \tag{15}$$

Si l'approche présentée est naturelle, elle reste néanmoins incomplète car nous n'avons pas encore explicité notre fonction de performance. Dans notre cas, la fonction de performance sera la fonction d'état v_{π_ω}(qui donne un estimé de la récompense obtenue en étant à un certain état et en utilisant une certaine politique). Il est facile de voir que plus la politique est bonne plus la fonction d'état (pour un état s) est bonne. Cela correspond donc à une bonne métrique de la qualité de la politque.

Le problème réside maintenant dans notre capacité à expliciter $\nabla\eta(\omega_t)$. Ce problème a été résolue dans notre définition de η. En utilisant la méthode de Monte-Carlo REINFORCE [25] (qui se base sur le théorème du gradient de la politque [21] qui explicite le gradient de la fonction d'état selon les poids ω)

[3]Asynchronous Methods for Deep Reinforcement Learning, dans la suite de ce rapport nous utiliserons le terme A3C

On obtient alors:

$$\omega_{t+1} = \omega_t + \alpha\gamma^t G_t \nabla \log\big(\pi(A_t \mid S_t, \omega)\big) \tag{16}$$

Un dès default de la méthode REINFORCE[25] est que l'estimateur du gradient qu'elle génére est hautement bruité. Pour pallier à cela, plusieurs méthodes ont été mise au points. La façon la plus simple de réduire la variance du gradient estimé, on utilise la formule suivante qui utilise la fonction d'état:

$$\omega_{t+1} = \omega_t + \alpha\gamma^t (G_t - \hat{v}(S_t, \theta_t)) \nabla \log\big(\pi(A_t \mid S_t, \omega)\big)^4 \tag{17}$$

Comme nous l'avons précédemment décrit, les méthodes issuent de la famille de Monte-Carlo sont connues pour être lente à converger (malgré l'introduction de notre stratégie pour réduire la variance de l'estimateur du gradient). Pour accélerer la vitesse de convergence, nous allons donc nous servir du TD Learning pour introduire une nouvelle stratégie qui se base sur le gradient de la politique (comme pour REINFORCE) et qui utilise à la place du gain cumulé, une mise à jour se basant sur celle du TD-Learning. Soit:

$$G_t \longleftrightarrow r_t + \gamma^t \hat{v}(S_{t+1}, \theta_t)$$

Donnant l'algorithme Actor Advantage Critic (A2C)[5]

$$\omega_{t+1} = \omega_t + \alpha\gamma^t (r_t + \hat{v}(S_{t+1}, \theta_t) - \hat{v}(S_t, \theta_t)) \nabla \log\big(\pi(A_t \mid S_t, \omega)\big) \tag{18}$$

2.1.6 de l'A2C à A3C

l'asynchronous advantage actor-critic (A3C) augmente A2C en utilisant n agents en parallèles (et n environnements) qui apprennent de manière asynchrone. Le réseau est copié et donné à chaque agent ce qui permet de n'avoir qu'un seul réseau et non n différents. A chaque mise à jour, les gradients sont envoyés au réseau principales qui à chaque nouvel épisode se copie pour les n agents.

[4] La mise à jour de θ pour la fonction d'état a été omise pour la clarté

[5] Advantage car r_t: est un estimateur de la fonction d'état-action(Q), on utilise donc la fonction d'avantage($A = Q - V$)

L'avantage de cette méthode est donc d'accéléré l'apprentissage de l'agent (en utlisant les n agents) mais le principal avantage de cette méthode est la décorrellation des gradients car les gradients sont issues des n agents. Cette problèmatique est la même que celle pour le deep-q-learning (DQN), contrairement à la mémoire utilisée dans le DQN qui utilise une partie des ressources de l'ordinateur, s'il est possible de générer n environnements en parallèle alors cette dernière méthode permet de sauvegarder des ressources. Néanmois, avec cette méthode sans module, il n'y a pas de réutilisation des résultats possibles, cela implique qu'il faudra un grand nombre d'episodes pour stabiliser un apprentissage. Un autre avantage qui vient des n agents qui apprennent en parallèle est que cela encourage l'exploration de l'environnement car les agents auront un comportement différents les uns des autres.

L'A3C sera donc l'algorithme de base pour le contrôle d'un agent dans SE-STAR. Il sera par la suite augmenté de plusieurs modules décrits au cours de ce rapport.

2.1.7 Platforme utilisables pour garantir le fonctionnement de nos algorithmes

Malgré que nos algorithmes aient pour but d'être déployer sur l'environnement interne de Thales SE-STAR, il nous a fallut trouvé des environnement suffisamment complexe pour tester la robustesse de nos agent mais aussi suffisamment simple pour ne pas devoir un temps infiniment long pour avoir notre réponse. De plus, la connection de nos algorithmes avec SE-STAR est un enjeu essentielle et a apporté son lot de difficulté. Il n'était donc pas raisonnable d'attendre ni de tester nos agents sur cette plateforme.

Il existe de nombreuses plateformes dédiées à l'apprentissage par renforcement. Nous avons choisis de nous concentrer sur des environnements proches de SE-STAR. Ainsi, de nombreuses études en RL utilise des jeux Atari en 2D et 3D. Nous allons donc nous basé en partie sur ce type d'environnement.

Vous pouvez voir page suivante plusieurs types d'environnements sur des plateformes différentes:

1. **Gym[3] - OpenAI**
   Gym est une plateforme qui met à disposition des jeux Atari et des simulations typiques en contrôle (Cartpole, MountainCar ...). La plateforme fournit une API permettant de réutiliser nos codes sur l'ensemble des simulations qui sont fournis par Gym.

L'API de Gym étant simple, nous avons décider de nous basé sur cette API pour la construction d'un wrapper autour de la simulation interne de Thales SE-STAR.

2. **Universe - OpenAI**
   Universe est une autre plateforme mettant à disposition une multitude de jeux très différents les uns des autres (2D, 3D, textes ...). Certains sont récents et demande une certaines réflexions (Portal, age of empire, ...). l'organisation de Universe est plus complexe que celle de gym et impose la prise en compte de plus d'éléments (réseau, possible lag, infrastructure ...).

   Universe propose donc des environnements plus difficiles à tout point de vue que Gym néanmois cette plateforme n'est pas encore utilisé par les chercheurs et on peut douter de la maintenance du projet Universe (OpenAI semble mettre en avant Gym)

3. **DeepmindLab**[1] **- Deepmind / Google**
   Le deepmindlab est une seul simulation de labyrinthe 3D très proche de notre application. Le problème est la difficulté d'utilisation de cette plateforme. Ainsi, nous n'avons pas utilisé cette ressource néanmoins elle est pertinant et est une cible pour tester nos algorithmes.

4. **Doom - Vizdoom**[11]
   Doom est un jeu PC de tir à la première personne. Cet environnement est devenu un classique dans la communauté RL pour tester les algorithmes. A tel point que l'université de Poznan à créer une compétition d'IA basée sur Doom. L'intérêt de ce environnement contrairement au précédant et qu'il est plus complet avec un ensemble d'actions possibles conséquent. De plus, l'environnement de Doom met en lumière des difficulté pour les IA en matière d'exploration de l'environnement et est donc un terrain de jeu pour résoudre ou proposer des algorithmes résolvant ce type de problème.

2.2 Utilisation et intérêt de l'apprentissage statistique - Deep Learning

Dans cette partie, nous allons nous intéresser aux différentes motivations qui nous ont poussé à utiliser l'apprentissage statistique (ou Deep Learning) dans notre contrôleur. Dans notre introduction, nous avons énoncé quelques points qui nous ont influencé pour choisir cette technologie. Nous préciserons en quoi le "Deep

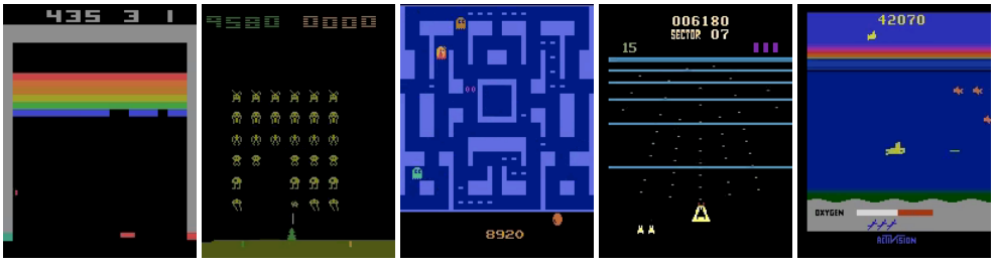


Figure 5: Exemple d'environnements GYM (Open AI)

Figure 6: Exemple d'environnements Universe (Open AI)

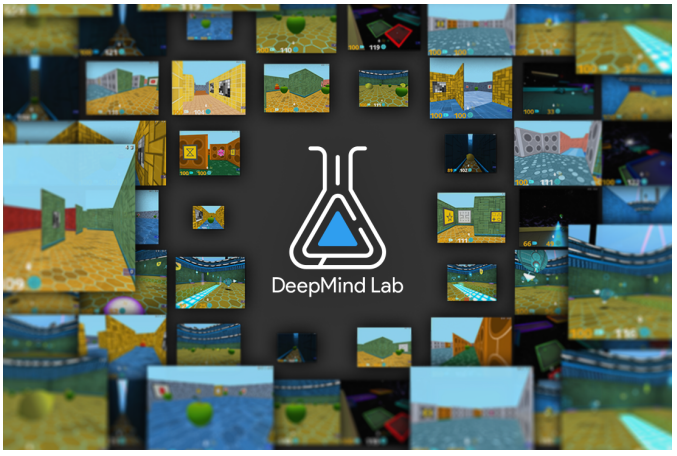


Figure 7: Exemple d'environnements DeepmindLab (Deepmind (Google))

Learning" est pertinent dans notre contexte et introduirons son fonctionnement et sa mise en relaction avec l'apprentissage par renforcement.

2.2.1 Motivations et objectifs de l'apprentissage statistique

Nous avons, en introduction, énoncé les principales contraintes qui reposées sur notre asservissement. De façon synthétique, nous avions trouvé:

- Environnement est partiellement observable (possiblement bruité).
- L'agent doit être capable, à partir d'un apprentissage sur une environnement, de réussir sur un environnement assez proche (généralisation).
- Les entrées seront basées sur la vision de l'agent.

Nous verrons que ces problématiques sont abordées par l'apprentissage statistique. La force du deep learning réside dans sa capacité à extraire de l'information à partir d'entrées bruitées à hautes dimensions. Néanmoins, le deep learning possède son lot de restrictions parmi lesquelles:

- Cela nécessite de très nombreux exemples pour réussir son apprentissage.
  nous verrons qu'en RL[6] il faudra parfois plus de $80 * 10^6$ images pour finir un apprentissage)
- L'apprentissage est difficile et parfois non stable
  nous approfondirons ce point plus tard, mais il est à noter que selon l'architecture choisit et nos fonctions de pertes, la stabilité du réseau n'est pas garanti. De façon plus claire, il y a énormément de facteurs jouant sur la stabilité du réseau et il peut être difficile de régler ces facteurs pour obtenir un bon résultat.
- Le réseau peut sur apprendre (apprentissage par coeur des résultats) impliquant l'impossibilité de généraliser. Cela veut dire que le réseau sera incapable d'extraire une bonne politique d'un environnement diférent.

Malgré les difficultés énoncées, le deep learning rend possible notre contrôle basé sur la vision en temps réel d'un agent.

Définissons le contexte dans lequel le deep learning s'inscrit. A chaque pas de temps, l'agent va recevoir l'image partielle de l'environnement (soit un tableau de

[6]Durant ce rapport, l'acronyme RL fera référence à *reinforcement learning* (ou apprentissage par renforcement en Français).

pixels). Nous souhaitons en sortie avoir la probabilité d'actions selon l'état (soit la politique pour l'état rencontré).

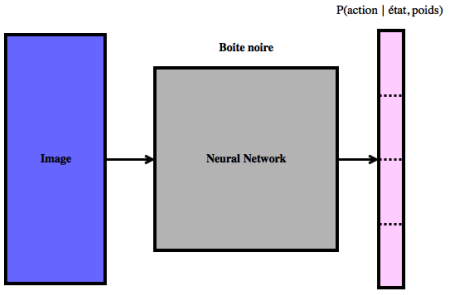


Figure 8: Représentation haut niveau de l'architecture classique en deep learning

Bien évidemment, le deep learning peut être utilisé pour bien d'autres choses mais nous nous restreindrons à son utilisation dans le cadre de l'apprentissage par renforcement. Néanmoins la figure ci dessus reste d'actualité dans la cadre d'un apprentissage supervisé (exemple en classification: déterminer le nombre représenté par une image correspondrait à avoir en sortie du réseau $P(\text{nombre} \mid \text{état}, \text{poids})$.

Un problème est maintenant de définir comment va être généré $P(\text{action} \mid \text{état}, \text{poids})$ et c'est là qu'apparait la véritable force du deep learning. On peut façonner automatiquement les poids du réseau de neurones dans le but d'optimiser une fonction de perte (ou de gain).

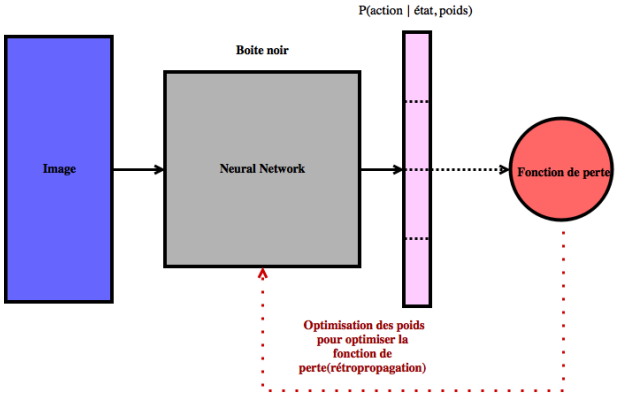


Figure 9: Représentation haut niveau de l'architecture classique en deep learning avec optimisation de la fonction de perte

Dans la partie suivante, nous expliquerons les mécanismes responsables de l'optimisation

du de la fonction de perte pour véritable définir ce que veut dire un apprentissage dans le contexte du deep learning. Nous commencerons par introduire ce qu'est un réseau de neurones dans sa forme la plus simple et nous expliquerons succinctement les évolutions utilisées (qui sont le réseau de neurones à convolution et les réseaux récurrents).

2.2.2 Architecture en apprentissage statistique et réseau de neurones linéaire

Nous allons commencer par expliquer la structure sur laquelle se base les réseaux de neurones (du moins historiquement) avec *le modèle du perceptron.*

On définira un ensemble de poids w sous la forme d'un vecteur de poid W, l'entrée sera un vecteur X de même dimension que les poids. On appelle f la fonction d'activation, qui est un hyperparamètre, c'est à dire que c'est à l'expérimentateur de la définir. On retrouve néanmoins quelques fonctions d'activations très utilisées, nous reviendrons sur l'intérêt des fonctions d'activations un peu plus tard. Dans le cas du perceptron, nous utiliserons la fonction d'activation $f(x) = \begin{cases} +1 \text{ si } \mathrm{W}^T X > \theta \\ \text{-1} \end{cases}$

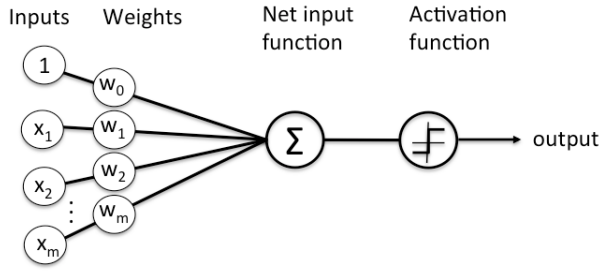


Figure 10: Représentation haut niveau de l'architecture classique en deep learning avec optimisation de la fonction de perte [20]

Pour se conformer à la **??**, il nous reste à définir quelle est la fonction de coût. Pour cela, nous avons encore besoin de quelques définitions. Plaçons dans un problème à deux classes (chien et chat par exemple) on définira la classe d'une entrée comme état la classe à laquelle l'entrée appartient ainsi on donnera $c(x) = \begin{cases} 1 \text{ si x} \in \text{Chien} \\ \text{-1 si x} \in \text{Chat} \end{cases}$, cela nous permet donc de définir la façon dont on va optimiser les poids:

$$W_{t+1} = W_t - \alpha\big(c(x) - f(x)\big)x$$

2.2.3 Réseau de neurones dense

Dans cette partie, nous allons expliquer l'architecture de base utilisée dans le contrôle de l'agent.

Notre objectif est d'approximer une certaines fonction $y = f(x)$. Nous avons accès à un ensemble de entrées et sorties associées que l'on notera $\mathcal{D} = \Big\{(x_1, y_1), ..., (x_N, y_N)\Big\}$. Nous souhaitons déterminer la fonction $\widetilde{f}_w$ tel que $\widetilde{f}_w \sim f$, la fonction $\widetilde{f}_w$ est défini comme $\widetilde{f}_w(x) = Wx + b$. Nous pouvons alors de façon défnir notre objectif comme un problème d'optimisation où l'objectif est de minimiser l'erreur quadratique moyenne $E = \sum_{p=1}^{N} \big(\widetilde{f}(x) - f(x)\big)^2$. Nous pouvons dès lors changer les poids dans le but de minimiser cette erreur (ou fonction de perte). La difficulté est que nous souhaitons être capable de *généraliser*, c'est à dire que si nous prenons un nouvelle ensemble $X = \big\{x'_1, ..., x'_P\ \big\}$, nous souhaitons que notre fonction $\widetilde{f}_w$ soit toujours aussi proche de f. Un problème en apprentissage statistique qui peut survenir est que le réseau ait appris l'ensemble $\mathcal{D}$ mais que l'approximation soit mauvaise dès lors que l'on présente un ensemble d'entrée non vu durant l'entrainement. De nombreuses stratégies ont été mise en place pour éviter ce problème.

Pourtant, en l'état nous sommes incapables d'approximer des fonctions non linéaires. Or notre souhait est d'utiliser un réseau de neurone pour approximer la fonction qui associe des états à l'action optimale pour obtenir le maximum de récompenses, on peut conjoncturer la non linéarité de cette fonction.

Pour parvenir à approximer des fonctions non linéaires, nous allons donc empiler des réseaux denses suivis de fonction non linéaires (appelées *fonctions d'activation*). En pratique, on utilisera les fonctions tangeant hyperbolique, sigmoide, relu ...

En notant: g^l pour la l ième fonction d'activation, la formule donnant la dynamique du l-ième module est:

$$h^{(l)} = g^{(l)}\big(W^{(l)}h^{(l-1)} + b^{(l)}\big)$$

Les réseaux denses bien qu'extrêmment utiles ont deux défauts majeurs. Le premier est l'incapacité des réseau de neurones denses à prendre en compte la temporalité. Or, dans notre cas, les actions précédentes sont importantes pour déterminer l'action à effectuer. Deuxièmement, les réseaux denses ne sont pas les plus adaptés pour des tableaux de pixels (images) en entrée. Les réseaux de convolutions utilisent certaines spécificités du format image que n'utilise pas le réseau dense. Nous reviendrons sur les architectures palliant aux deux défauts

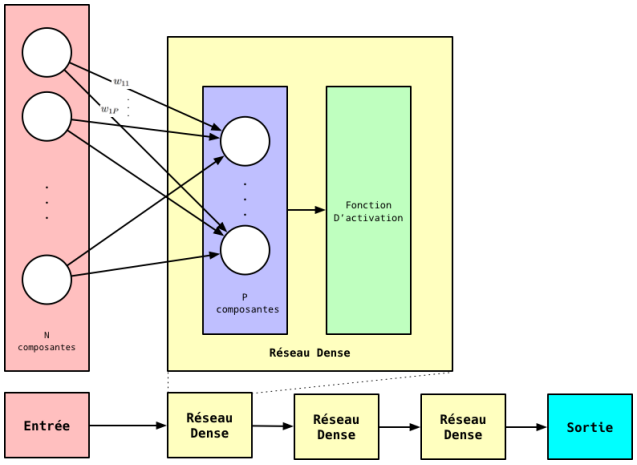


Figure 11: Schéma d'un modèle dense et de la réalisation d'un réseau dense

cités précédemment.

2.2.4 Réseau de convolution et récurrent

le réseaux de neurones à convolutions utilise l'opération de convolution pour déterminer la sortie du réseau.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

Avec K le noyau et I l'entrée. On peut considérer l'opération de convolution comme une moyenne locale mouvante. Ainsi, elle permet d'être plus robuste aux bruits mais à de nombreux autres atouts.

Les réseaux de convolutions utilisent en entrée des blocs de dimensions 3 (en pratique des tenseurs à 4 dimensions mais par simplicité, nous considerons simplement l'entrée comme un bloc 3D. On pourra penser une image RGB comme un bloc de dimension 3). Le réseau de convolutions s'est bati autour de trois idées. La première est des **intéractions locales**, ce qui veut dire que chaque entrée va intéragir avec seulement un sous ensemble de la sortie. La deuxième est le **partage des poids** , car dans le réseau a convolution, le noyau est commun à tout les pixels d'entrée (alors qu'on pourrait le changer pour chaque pixel). Cela a pour effet d'utiliser bien moins de paramètres qu'un réseau dense à taille équivalente. Enfin, troisièmment, le réseau possède **une invariance par translation**, cela implique que si un objet est translaté en sortie de réseau le résultat sera le même. Cela permet d'être plus robuste dans de nombreux cas.

Le réseau de convolutions est composé d'un noyau qui est l'équivalent des poids dans le réseau de neurones denses. Il est utilisé dans l'opération de convolution. C'est le noyau qui est optimisé dans le but de minimiser une certaine fonction de coût.

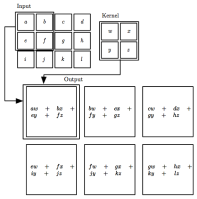


Figure 12: Opération de convolution dans un CNN

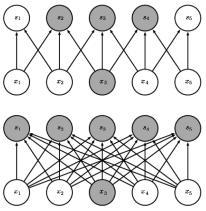
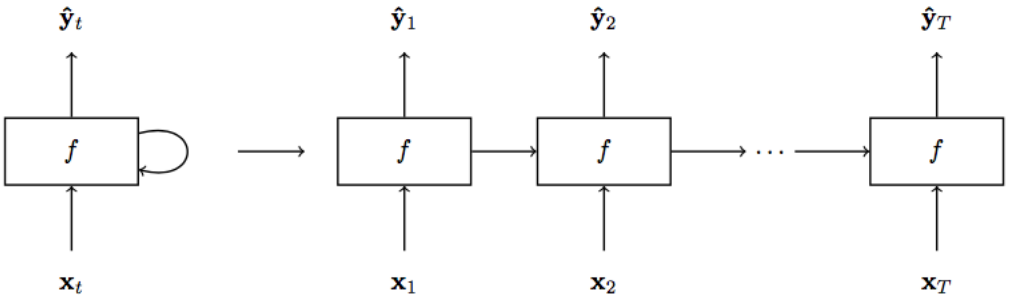


Figure 13: Activation de x_3

Les réseaux de neurones récurrent ont été créés dans le but de pouvoir être utilisés avec des données séquentielles. La dynamique d'un réseau récurrent est donnée pour un niveau L par la formule suivante:

$$\begin{cases} \mathrm{h}_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \\ \mathrm{y}_t = W_{hy}h_t + b_y \end{cases}$$

La formule ci dessous donne la dynamique du réseau récurrent le plus simple. Il souffre de nombreux problème, des alternatives existent notamment le réseau récurrent *Long-Short term memory*[7] .



2.2.5 Utilisation du deep learning dans le cadre de l'apprentissage par renforcement

Dans la partie précédente, nous avons expliqué dans quel cadre il est pertinant d'utiliser l'apprentissage statistique (ou deep learning). Maintenant, nous allons voir comment intervient précisement cette technologie dans notre contrôle par apprentissage par renforcement.

Comme nous l'avons vu, les principaux algorithmes d'apprentissage par renforcement utilisent soit la fonction d'état (V), soit la fonction d'état action (Q). Ces fonctions sont une mesure de la qualité d'un état ou d'une action selon une politique. Ainsi, il joue un rôle centrale dans le contrôle de notre agent. Plus encore, ces deux fonctions sont inconnues, et la réussite du contrôle de l'agent repose en grande partie sur notre capacité à approximer ces fonctions. Compte tenu de la complexité supposée de ces fonctions, nous avons décidé d'utiliser l'apprentissage statistique car elle a montré sa capacité à apprendre des fonctions très complexe (non linéaires).

Dans la partie précédente, nous avons introduit l'apprentissage statistique supervisé (car nous connaissons les sorties souhaitées). Or, nous serons dans un cas dit non supervisé car nous n'avons aucun signal sur lequel travaillé pour avoir la Q ou V fonction. Notre réseau devra approximer la fonction d'état avec un signal peu intéressant qui est la récompense. Cela n'est pas suffisant, nous recherchons un réseau capable d'extraire d'un état qui lui est présenté les éléments qui sont suffisamment discrimants pour juger de la qualité d'une politique (via la V ou Q fonction).

Nous pouvons donc décomposer notre architecture d'apprentissage en deux parties. La première aura pour but d'extraire de l'état les élements importants pour caractériser la qualité de celui-ci (la V ou Q fonction). La deuxième aura pour but d'utiliser les éléments qui sont sorties de la première partie (soit des élements plus ou moins représentatifs de la qualité) et devra les utiliser pour approximer la Q (ou V) fonction.

Voici une vue schématique de l'architecture réalisée dans le papier *Human-level control through deep reinforcement learning*[15]

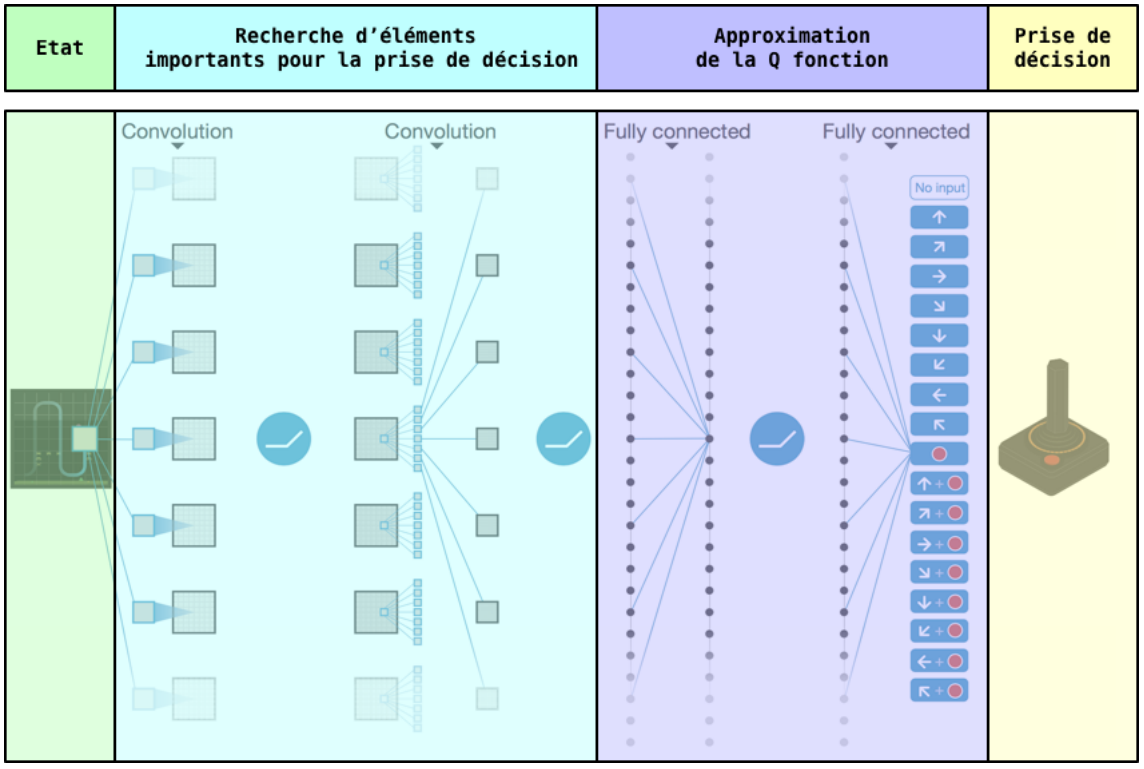


Figure 14: Schéma récapitulatif des différentes sous parties dans un réseau de neurones utilisé pour de l'apprentissage par renforcement

3 Interface entre SE-STAR et le contrôleur

Un grande partie de ce stage a tourné autour de l'interaction des algorithmes d'apprentissage par renforcement et SE-STAR (cf **??**). Cette partie sera consacrée à l'explication de l'architecture mise en place autour de SE-STAR pour la connection du contrôleur et de SE-STAR. L'architecture proposée est composée de la création d'une API simple pour communiquer avec SE-STAR et d'une interface bas niveau pour intégrer l'API haut niveau (réalisée par Alexandre kazmierowski).

Nous discuterons des divers difficultés que nous avons rencontré, des limites de notre architecture et des améliorations possibles.

SE-STAR est à la base un logiciel de simulation interne à THALES. SE-STAR a été prévu pour permettre une communication via l'API de SE-STAR néanmoins le contrôle d'un seul agent par apprentissage par renforcement. Pourtant, l'API n'a pas été pensé pour un pilotae d'un agent par renforcement. .Le contrôle des agents dans SE-STAR se base sur des stimulis internes motivationnels qui peuvent être configurés en fonction du rôle que l'on souhaite donner à un agent. Nous ne rentrerais pas dans les détails du fonctionnement original qui est complexe et orthogonal à notre objectif. Notre but est justement de créer un contrôle automatique se basant sur des stimulis extrinsèques (donnés par l'environnement) reposant sur la théorie de l'apprentissage par renforcement profond.

SE-STAR est un logiciel complexe. Il a donc fallu créer une interface avec le contrôleur en étant le moins intrusif possible sur le code source de SE-STAR. L'objectif était de produire un code réutilisable quelques soit l'environnement. Nous nous rendrons compte que ce choix implique une perte en performance.

3.1 Analyse fonctionnelle de l'API choisie

Nous allons profiter de la simplicité de notre API pour l'expliquer. En apprentissage par renforcement, un agent reçoit l'état dans lequel il se trouve. A partir de celui-ci, il interagit avec l'environnement qui lui répond en envoyant le prochain état et la récompense associée à l'état. Nous allons nous baser sur ce fonctionnement pour créer notre API.

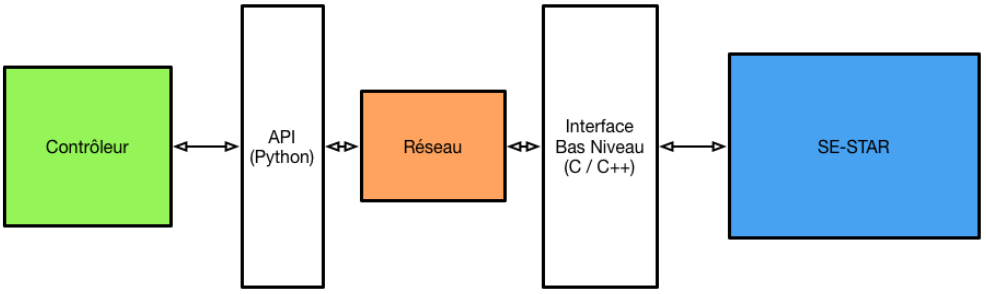


Figure 15: Vue globale de l'interaction entre le contrôleur et SE-STAR

API Client	
Fonction	Explication
Reset	Reset a pour but de redémarrer la simulation (obligatoire en cas de fin de partie ou si un certain temps est dépassé)
Step	1. Entrée: l'action effectuée 2. Sortie: une liste contenant l'état suivant, la récompense reçue, un booléen indiquant si la simulation est terminée et un dictionnaire donnant une multitude d'informations sur la simulation.
Connect	Une fonction nous permettant d'établir une connection à distance avec un Windows distant. Nous ne parlerons pas de la méthode de connection à distance qui est assez complexe et reposant sur des outils tiers (Winexe, ...)

L'API est simple mais correspond exactement à la boucle classique en renforce-

ment. Alexandre Kazmierowski a construit le serveur répondant à ces appels. L'outil créé par M. Kazmierowski permet l'interconnexion entre SE-STAR et le client .

3.2 Principales limites et contraintes

La simulation SE-STAR fonctionne actuellement uniquement sur Windows. Cela pose un problème car les différents outils, pour créer des algorithmes de Deep Learning, utilisent une distribution linux. Les échanges réseaux entre Linux et Windows limitent drastiquement le nombre d'images par seconde que peut recevoir l'agent (définissant la vitesse de simulation) car l'envoi d'un tableau de pixels sur le réseau est intrinsèquement coûteux.

De plus, nous avons été confronté à de nombreux problèmes de déconnections impromptues qui ont nécessité la mise en place de mécanismes de reconnections automatiques. Néanmoins l'architecture mise en place manque encore de stabilité et n'est pas robuste à tous les types d'erreurs pouvant être rencontrés.

L'algorithme utilisé pour contrôler l'agent dans SE-STAR a été l' qui repose sur de multiples agents qui jouent dans l'environnement de façon asynchrone. Cela multiplie le nombre d'appels réseau et joue sur la stabilité de celui-ci, cela accroît la latence des appels agent (SE-STAR) / algorithme (Contrôle). De façon empirique, nous nous apercevons qu'il y a des problèmes dès que le nombre d'agents est supérieur à six. Cela s'explique en partie par le fait que SE-STAR est un logiciel demandant beaucoup de ressources, Or celui-ci n'a jamais été prévu pour une utilisation demandant de faibles ressources.Ainsi la multiplication des instances de SE-STAR peut causer des problèmes sur la machine cotée serveur si elle n'a pas assez de puissance. Alexandre Kazmierowski nous a permis d'alléger les ressources demandées, permettant d'augmenter le nombre d'agents.

Nous pouvons dès lors nous demander si la stratégie asynchrone d'entraînement est viable. Une problématique inhérente aux algorithmes de renforcement, qui découle de l'utilisation du deep learning, est le besoin de décorréler les états lors de l'entraînement . Or dans le cadre de nos algorithmes, cela s'avère difficile. La stratégie d'entraînement asynchrone permet la décorrélation des états lors de l'apprentissage.

En ajoutant les défaults de l'A3C, nous pouvons nous demander si d'autres algorithmes ne seraient pas plus pertinents.

3.3 Compromis et solutions envisagées

Dans cette partie, nous allons expliquer les solutions envisagées pour éviter de surcharger le réseau. Avant cela, il convient de revenir sur certaines contraintes sur le contrôle. En particulier, nous souhaitions avoir un contrôle qui est le moins intrusif possible sur l'environnement en l'occurrence SE-STAR.

Nous allons cependant proposer une méthode qui reste intrusive mais permet de résoudre en partie les problèmes liés à la multiplication des instances de SE-STAR et de réduire les appels réseaux.

La méthode qui sera développée se nomme: Efficient Parallel Methods for Deep Reinforcement Learning [4]. Nous ne rentrerons pas dans les détails de l'algorithme car il est similaire à l'A3C [13]. Néanmoins,il propose une approche synchrone permettant l'utilisation du GPU. L'idée de ce papier est de proposer une architecture synchrone permettant de décorréler les états utilisés pour l'entrainement. A chaque pas, le contrôleur va décider quelles actions vont effectuer un ensemble d'agents. Le contrôleur va récupérer l'ensemble des états et des récompenses pour l'apprentissage et le choix de la nouvelle action. Quand le contrôleur rentre en phase d'apprentissage, il a un ensemble de transitions (état précédant, état actuel, action, récompense) décorrélées (car venant d'agents différents). Les premiers essais sont encourageants mais l'apprentissage n'est pas assez stable.

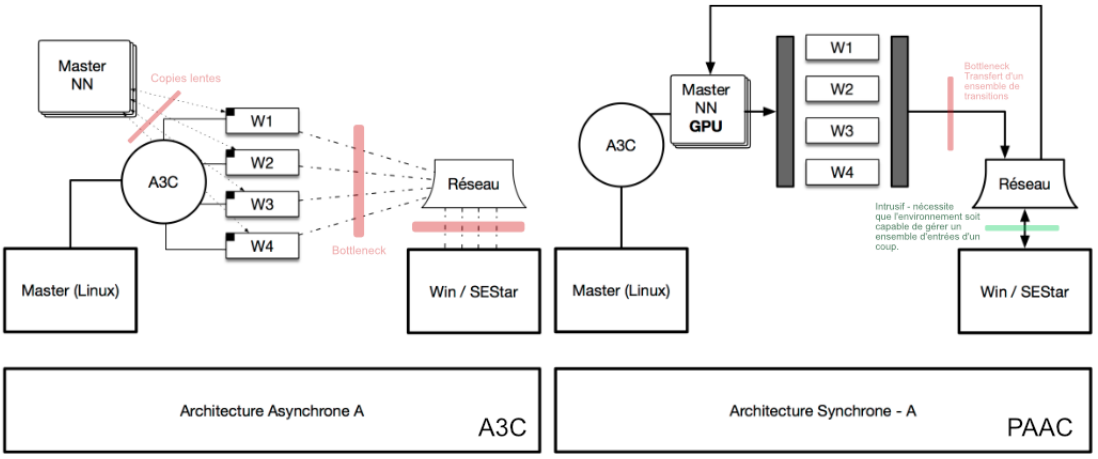


Figure 16: Comparatif des algorithmes PAAC / A3C en mettant l'accent sur les problématiques réseaux

Nous remarquons que les principaux problèmes qui découlent de la stratégie asyn-

chrone d'entrainement (A3C) sont liés à la communication réseau mais pas seulement. Un stratégie asynchrone exclue l'utilisation du GPU. La stratégie synchrone de la PAAC permet l'utilisation du GPU et impose que la simulation soit capable de gérér un ensemble d'actions en entrée. De plus, la question des transferts sur le réseau n'est pas résolue par cette approche.

La véritable question est de savoir s'il n'existe pas d'autres approches qui suppriment la nécessité du réseau ou qui reposent sur des outils spécialement crées pour gérer les difficultés inhérentes aux réseaux. Nous verrons rapidement deux approches idéales se basant sur un outil Docker [12] et sur l'utilisation de Wine (qui permet de faire fonctionner des applications Windows sur Linux)

3.4 Solutions idéales

3.4.1 SE-STAR sous linux via Wine

Notre objectif a été d'utiliser SE-STAR nativement sous Linux. Pour cela, il y a deux solutions. Premièrement, nous pouvons compiler directement SE-STAR pour Linux. Le problème est que le logiciel a été pensé pour une utilisation Windows et utilise des certaines librairies uniquement disponibles sur Windows. Nous ne savons pas si cela est possible même si cela aura le mérite d'être essayé. Deuxièmement, nous pouvons passer sous Wine pour utiliser une application Windows. Après une phase de configuration compliquée pour utiliser Wine, nous avons été en capacité d'utiliser SE-STAR sous Linux et d'utiliser l'. Nous avons remarqué des interruptions complètes de SE-STAR au bout d'une dizaine de minutes. Pour l'instant, il est difficile de savoir ce qui est responsable de ces interruptions. Nous pensons toujours que Wine est une solution viable. Il est reste néanmoins à comprendre l'origine des problèmes. Une des problématiques qui survient avec l'utilsation de Wine est que les fonctionnalités permises par SE-STAR ne sont pas entièremenet prise en charge par Wine. Cela nous oblige donc à passer par une version obsolète de Wine et possiblement moins stable.

3.4.2 Une architecture scalable d'entrainement via Docker

Nous allons présupposer que nous avons un moyen d'utiliser SE-STAR sous Linux. Nous exploiterons Docker (qui est un système de conteneurisation, qui groupe une application avec un environnement Linux léger configuré au préalable). Cela permet, en spécifiant les dépendances de SE-STAR, de pouvoir déployer dans un

réseau n instances de SE-STAR et de simplement utilser le *swarm mode* de docker pour avoir une gestion du réseau (load balancing, ...). Nous pourrions imaginer déployer dans un cluster de cpu (ou gpu) plus de 1000 instances de SE-STAR gérées par le swarm mode de Docker. Ce genre d'architecture a déjà été utilisée.Cela pourra être mis en place dès que SE-STAR sera disponible sous Linux.

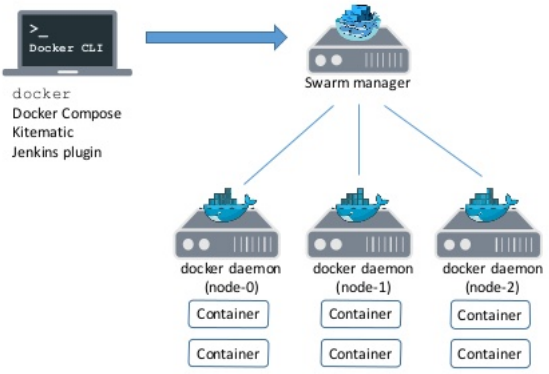


Figure 17: Simple vue d'ensemble de l'utilisation de Docker en swarm mode

Le swarm mode de Docker est une architecture composée d'un (ou plusieurs) noeud manageur qui a pour but de contrôler les noeuds enfants, de gérer le load balancing, la gestion des erreurs des noeuds enfants ... En ayant des conteneurs avec les dépendances requises par SE-STAR et l'application SE-STAR dans chaque conteneurs enfants, cela permet d'utiliser les algorithmes classiques d'apprentissage par renforcement profond à un échelle industrielle.

3.5 Conclusion sur les questions d'interfaces et de réseaux

A travers ce chapitre, j'ai essayé de vous expliquer les principaux enjeux dans le cadre de l'interface entre SE-STAR et les algorithmes d'apprentissage. Sur des cas jouets, ces problématiques ne font pas surfaces néanmoins, dans des applications industrielles nous sommes obligé de réfléchir à l'architecture dans laquelle nous allons déployer nos algorithmes et nos applications. Or, les d'apprentissage et les applications peuvent être destinées à des environnements différents ou pire avoir des dépendances contradictoires. Il est donc essentiel d'utiliser des applications capables d'isoler une application afin que celle-ci puisse être disponible sur le réseau dans le but de pouvoir permettre la communication entre les applications.

Ci-dessous, un schéma récapitulatif de l'architecture idéal (qui a été partiellement mise en place durant le stage).

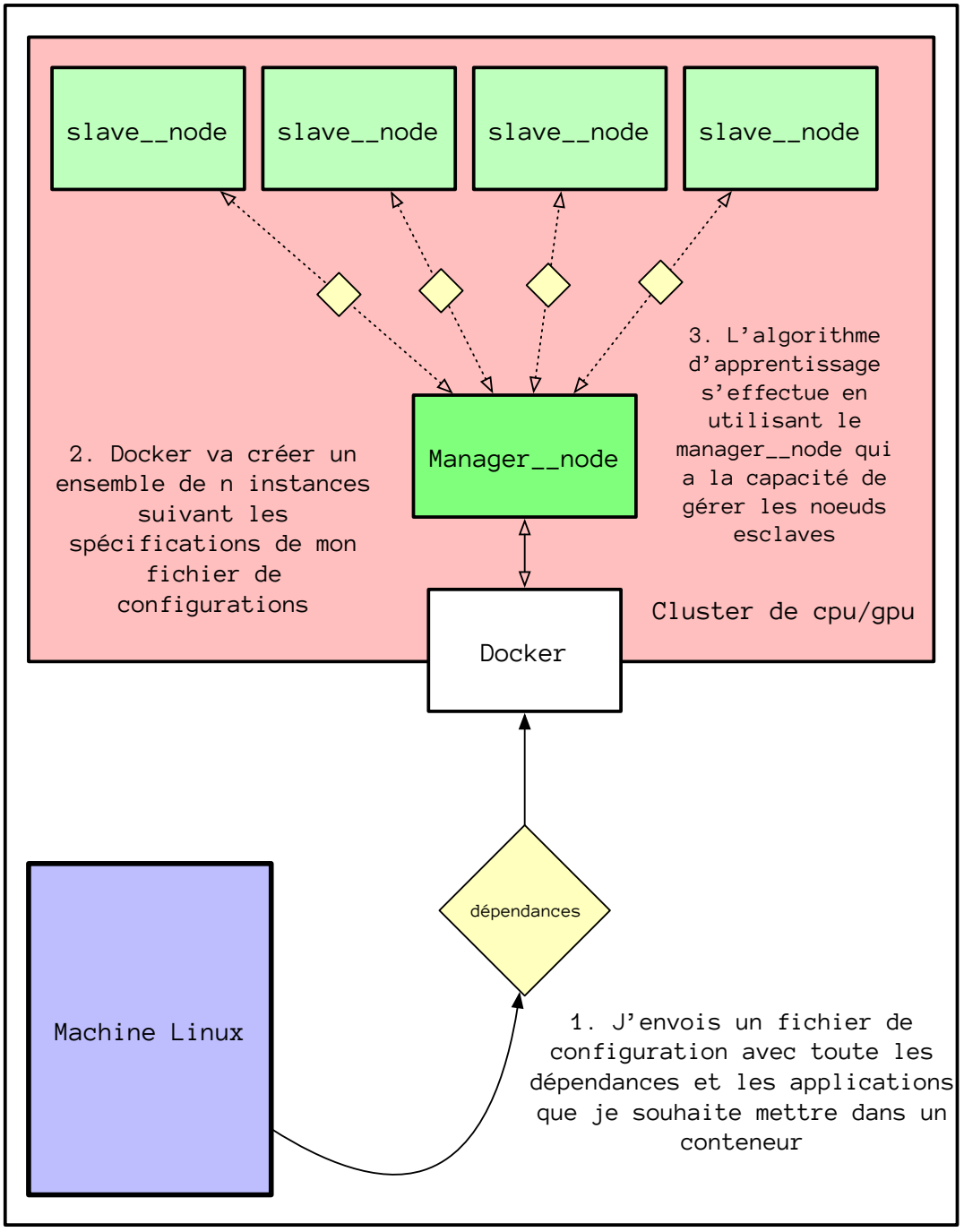


Figure 18: Vue d'ensemble de l'architecture partiellement mise en place dans le cadre de l'apprentissage de l'algorithme

4 Contrôle d'un agent via la simulation SE-STAR par renforcement profond

Cette partie sera consacrée au fonctionnement du contrôle appliqué à SE-STAR. Dans le but de comprendre les choix entrepris, nous commencerons par expliquer comment fonctionne l'algorithme d'apprentissage par renforcement utilisé pour SE-STAR. Puis, nous expliquerons les faiblesses de cet algorithme (dans le contexte de SE-STAR). Enfin, nous introduirons un module de curiosité qui a pour but d'améliorer l'apprentissage et donc le contrôle de l'agent.

Nous considérons SE-STAR comme un environnement sans faire mention des problématiques liées à l'interface contrôleur / SE-STAR, ou des problématiques liés aux réseaux qui ont été développé dans la partie précédente.

4.1 Le choix d'un algorithme efficace pour le contrôle d'un agent dans SE-STAR.

Il existe dans la littérature un panel très large d'algorithmes qui sont potentiellement puissants. Or, pour des questions de temps, nous n'avons pas été capable d'essayer tous les algorithmes. Nous avons donc dû faire un choix.

Nous allons à travers cette partie, donner les principaux réseaux qui nous ont poussé à nous diriger vers un apprentissage par renforcement profond.

Tout d'abord, les spécificités de SE-STAR impliquent le non-usage des algorihtmes *model based* (basés sur un modèle, soit la connaissance de la fonction de transition). Or, il est évident que dans le cas de SE-STAR, cette fonction est inconnue. Nous pouvons même aller plus loin, la fonction de transition dans le cas d'états sous forme d'un tableau de pixels est bien souvent insoluble. Cela reviendrai à avoir la connaissance d'une fonction qui, sachant l'etat courant et l'action donne la probabilité d'être dans un nouvel état. Voici le cheminement pour déterminer le nombre de combinaisons possibles d'états pour une image de 82x82 pixels. Un pixel est composé de trois nombres de 0 à 255. Chaque pixel a la capacité de produire théoriquement $256^3 = 16\ 777\ 216$ couleurs. Ainsi, il y a $82 * 82 * 256^3 = 112\ 810\ 000\ 384$ combinaisons d'images différentes possibles. Nous comprenons vite l'impossibilité d'utiliser un algorithme basé sur la fonction de transition. Malgré le fait que pendant ce stage nous n'avons pas utilisé d'algorithmes purement *model based*, nous pourrions toujours approximer cette fonction en utilisant une architecture de Deep Learning et en trouvant une fonc-

tion de perte adéquate.Cela est possible, mais ne sera pas testé durant ce stage car l'approche *model based* est difficilement applicable dans le cas d'états sous forme de pixels (nous passons toujours par une phase de compression de l'état pour permettre ce genre d'algorithmes). Il y a peu d'algorithmes suivant ce principe qui ont montré de bon résultats sur des environnements difficiles.

- Notre algorithme d'apprentissage par renforcement appartiendra à la famille des **algorithmes sans modèle (*Model Free*)**

Il existe beaucoup d'algorithmes dans la famille *Model Free*. Nous souhaitons donc utiliser en premier lieu un algorithme qui est robuste et performant selon la littérature. De plus, nous aimerions un algorithme qui soit extensible. En effet, comme nous le verrons plus tard avec le module de curiosité, la plupart des environnements 3D sont difficiles pour les algorithmes d'apprentissage par renforcement.

Au vue de la littérature, il y a deux méthodes qui ressortent:

1. **Deep Q Networks (DQN [15])**
2. **Asynchronous advantage critic (A3C)**[7] **[13]**

Les principes sur lesquels reposent ces deux algorithmes ont été développé précédemment dans ce rapport. Bien que nous avons testé ces deux algorithmes sur SE-STAR, nous avons déjà commencé par comparer ces deux algorithmes sur des cas plus simples. Néanmois, il est à noter que sur les expérimentations, que nous avons effectué, ont été utilisées des versions améliorées du DQN de l'A3C (non indiquées dans ce rapport). En effet, ces méthodes ont trouvé un echo dans la communauté scientifique et ont été améliorées de nombreuses manières. Il serait trop long d'expliciter toutes les améliorations existantes pour ces algorithmes mais nous allons tout de même les citer et donner une référence pour les lecteurs les plus curieux.

[7] Nous utiliserons A3C ou DQN dans la suite de ce rapport

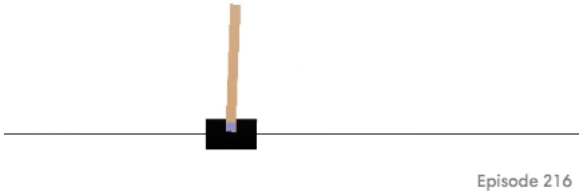
Algorithmes testés pendant le stage	
Améliorations du Deep-Q-Network	
Double-Q-Learning (DDQN [22])	Résout un problème intrinsèque du Deep Q Network qui est l'introduction d'un biais au niveau de l'estimation de la fonction d'état action.
Dueling-Q-Learning (D-DQN[23])	Au lieu d'approximer la fonction d'état action ($\rightarrow$ DQN), nous préférons approximer *la fonction d'avantage* qui se défini comme la différence de la fonction d'état et de la fonction d'état action. Cela a de nombreuses qualités car la fonction avantage permet de déterminer si une action mène à un gain positif par rapport à la moyenne des gains espérée ce qui est plus utile que simplement la fonction d'état action dans le cadre du contrôle (la fonction avantage donne une information plus fine et intéressante que la fonction d'état action)
Prioritized Replay[18]	Pour décoreller l'entraînement dans le cadre de l'apprentissage en utilisant le DQN, nous sommes obligé de nous servir d'une mémoire pour stocker des transitions et pouvoir ensuite échantillonner celles-ci. La méthode de *favorisation de la mémoire* consiste à échantillonner les transitions qui ont mené à la plus forte erreur (cf fonction de perte), ce qui correspond à des cas particulièrement intéressants dans le cadre de l'apprentissage.
Améliorations de l'A3C	
GAE[8][19]	La méthode GAE (voir note de bas de page pour avoir la référence) est une méthode qui n'est pas directement affiliée à l'A3C mais qui a été utilisée dans celle-ci et est devenu quasiment standard. L'idée est d'utiliser un estimateur de la fonction d'avantage qui est biaisé mais qui possède une variance plus faible. Il a été montré empiriquement que son utilisation améliore grandement l'apprentissage.
UNREAL	A3C-GAE avec en plus des objectifs auxiliaires pour améliorer la stabilité du réseau et pousser l'agent à explorer son environnement.

Comparons empiriquement les méthodes dérivées du DQN et de l'A3C.

4.2 Comparaison entre les méthodes issues de la famille du DQN et de l'A3C

4.2.1 Les algorithmes issus de la famille du Deep Q Learning

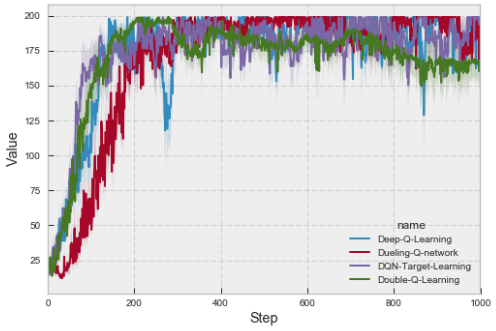
Les résultats sont issus d'expérimentations sur l'environnement de contrôle cart-pole, où le but est de maintenir une barre sur un ensemble mobile le plus longtemps possible. L'ensembles des actions est composé de LEFT pour aller à gauche et RIGHT pour aller à droite. L'agent reçoit une récompense de +1 à chaque pas de temps. L'environnement est réussi lorsque que l'agent a 200 points (nombre de points maximum).



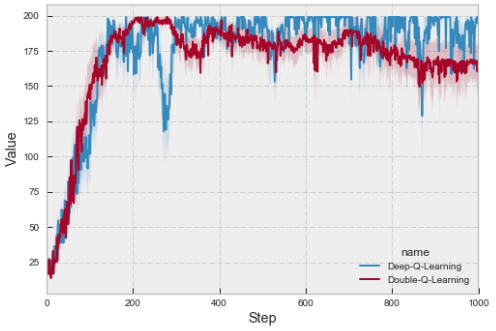
A travers ces premiers résultats, nous pouvons comparer les différents algorithmes issues de la famille du Q Learning.

Nous pouvons remarquer que, quelque soit l'algorithme issu de la fammile du Q Learning, l'apprentissage est dans l'ensemble stable. En effet, Nous ne voyons pas, pour la majorité des algorithmes testés, de chutes de performance durant l'apprentissage. Cependant, il y a de faibles différences concernant la dynamique d'apprentissage. L'algorithme le plus simple (*le Deep-Q-Learning* a la dynamique la plus rapide pour arriver de façon quasiment stable à 200 points avec le Deep-Target-Learning[9]. Ceci s'explique car les dérivés du DQN stabilisent l'entraînement au prix d'un apprentissage plus long. Il est à noter que tous les autres algorihtmes issus du Q learning ont été testés dans la version avec *Target*. Nous pouvons ainsi voir que, globalement, le DQN est un peu moins stable que les autres algo-

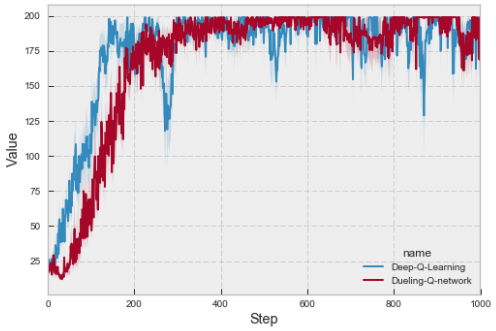
[9]une version du DQN visant à stabiliser l'apprentissage



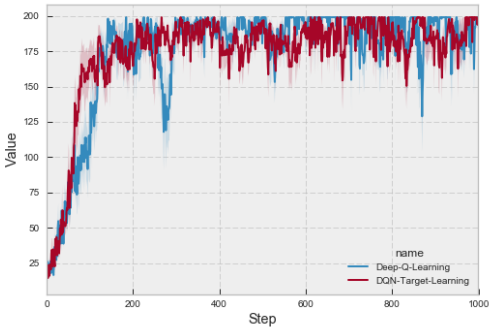
(a) Comparaison: Famille DQN



(b) Comparaison: DQN / DDQN



(c) Comparaison: DQN / Dueling



(d) Comparaison: DQN / DQN-Target

Figure 19: Résultats du Deep Q learning sur CartPole

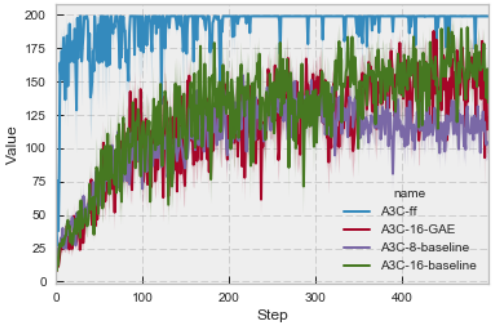
rithmes (présence de pics avec un score faible). Enfin, nous pouvons constater que *le dueling-Q-network* et le DQN avec target sont les plus stables et donc les plus adéquats.

Ce constat est corrélé avec d'autres expérimentations faites qui ne sont pas présentent dans ce rapport.

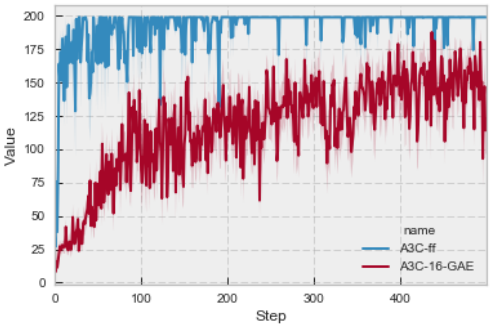
4.2.2 Les algorithmes issus de la famille de l'A3C

Nous avons réalisé les mêmes expérimentations avec des algorithmes issus de la famille de l'A3C. Nous avons testé en particulier deux variantes de l'A3C.

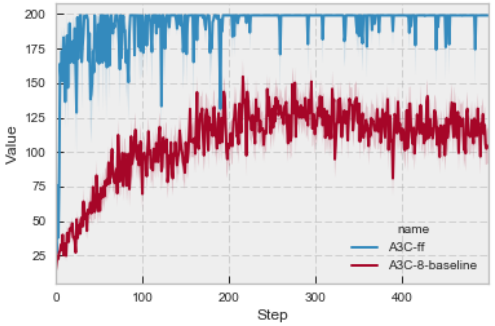
- A3C-FF
  Pour A3C Feedforward, cela fait référence au type de réseau de neurones



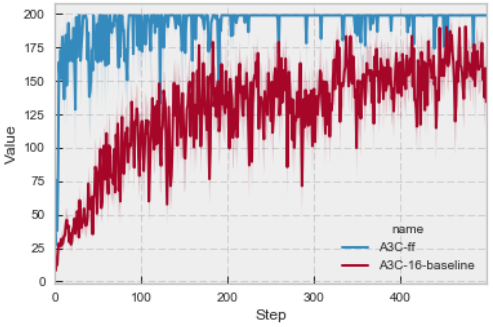
(a) Comparaison : Famille A3C



(b) Comparaison A3C-LSTM / A3C-LSTM-GAE avec 16 processus



(c) Comparaison A3C-LSTM / A3C-FF avec 8 processus



(d) Comparaison A3C-LSTM / A3C-FF avec 16 processus

Figure 20: Résultats de l'A3C sur CartPole

utilisé en sortie du réseau de convolution. En particulier, le réseau de neurones correspond au modèle ldense précédement décrit.

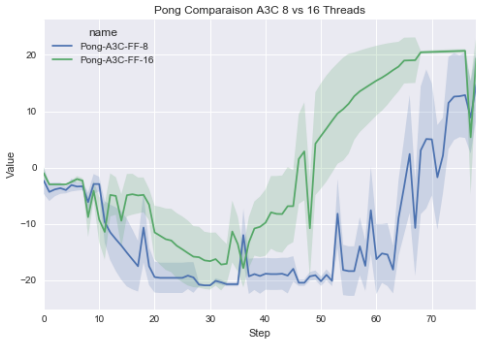
- A3C-LSTM
  LSTM correspond à un réseau de neurones récurrent qui a la particularité par rapport au réseau dense de prendre en compte les états précédents. Ce type de réseau est pertinant dans le cas où nous avons seulement une image partielle de l'environnement.

De plus, nous avons comparé ces types d'algorithmes avec un nombre différent de processus. Rappelons que l'A3C est un algorithme asynchrone se basant sur de multiples agents (dans la littérature entre 4 et 32 processus sont habituellement utilisés) Nous remarquons que l'A3C est quasiment deux fois plus rapide que le DQN (et ses variantes) pour arriver au score maximum de 200 points dans sa

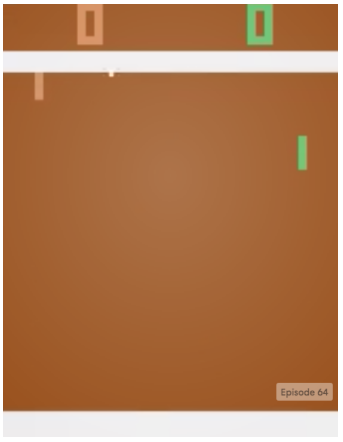
verison Feedforward (FF). Cela s'explique car un réseau récurrent est bien plus long à entrainer. Pour des applications complexes nous préférerons la version LSTM de l'A3C. Ainsi, dans le cadre de SE-STAR, nous utiliserons la version LSTM.

Le nombre de processus est aussi déterminant dans la dynamique d'apprentissage. En regardant la figure c qui compare la version FF avec 16 processus (en bleu) et la version avec 8 processus, nous nous rendons compte qu'un nombre élevé de processus est bénéfique pour l'entraînement.

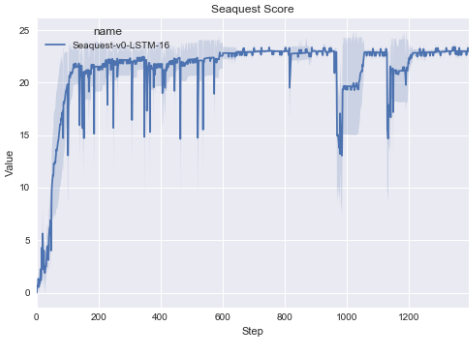
Cartpole est un environnement relativement simple à résoudre. L'environnement est éloigné de notre cas d'usage qui sera SE-STAR. Nous avons donc testé les différents algorithmes sur des environnements plus complexes. Ci dessous, les résultats pour différents environnements 2D dans lesquels l'état est un tableau de pixels (82*82). Ces environnements sont plus complexes que Cartpole pour au moins trois raisons. La première est qu'il y a un nombre élevé d'actions (douze en moyenne), la deuxième est que l'état possède une haute dimensionnalité. Enfin, les récompenses sont données seulement à la fin d'un épisode ou plus rarement que dans Cartpole. Cela ralenti l'entraînement assez fortement et posera de nombreux problèmes.



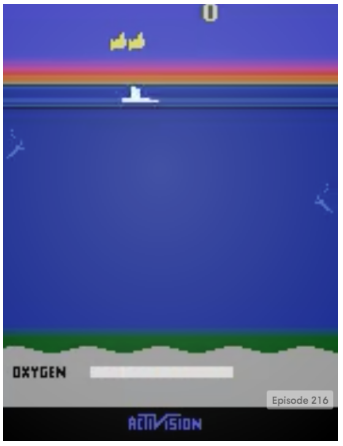
(a) Résultat de l'A3C sur PONG



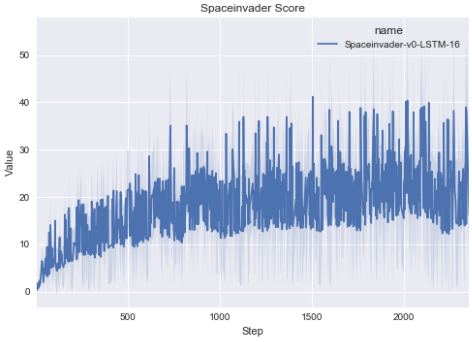
(b) PONG



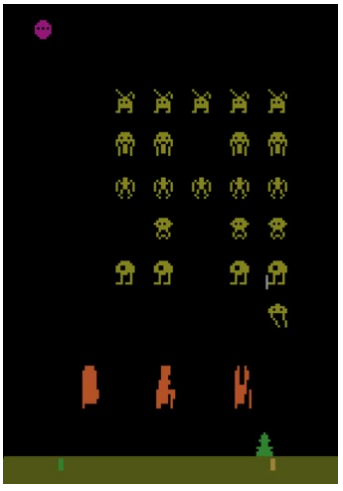
(c) Résultat de l'A3C sur SEAQUEST



(d) SEAQUEST



(e) Résultat de l'A3C sur SPACEINVADERS



(f) SPACE INVADERSE

Figure 21: Résultats de l'A3C sur des environnements ATARI

Nous avons utilisé l'A3C-LSTM-GAE pour l'apprentissage car il a montré de meilleurs résultats d'après nos tests. Nous n'avons pas utilisé un algorithme issu de la famille du Q-Learning car un des défault de cette famille est la lenteur pour converger vers une bonne politique. Nous avons déjà mentionner qu'intrinsèquement le Deep-Q-Learning avait des problèmes de stabilité, or les stratégies pour le stabiliser ont aussi pour effet d'entraver la dynamique d'apprentissage de celui-ci. Dans la suite de ce rapport, nous privilégierons donc les méthodes se basant sur la famille de l'A3C.

4.3 Spécification des différents environnements créés par SE-STAR

Dans cette partie, nous introduirons les environnements qui ont été utilisés avec notre architecture de contrôle. Nous avons créé plusieurs environnements et plusieurs représentations (images 2D vue de haut, vue FPS[10]). Nous montrerons qu' à l'heure actuelle, certains environnements sont difficiles à utiliser avec de l'apprentissage par renforcement pour des raisons que nous verrons ultérieurement.

4.3.1 Environnement labyrinthique simple créé par SE-STAR

L'objectif de ce stage fut de contrôler un agent à travers un environnement labyrinthique dans le but de trouver la sortie de celui-ci. Notre premier environnement fut extrêmment simple.

Ci-dessous, une image de l'environnement créé et de la representation associée à celui-ci.

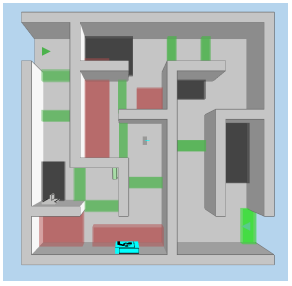


Figure 22: Environnement A en 3D

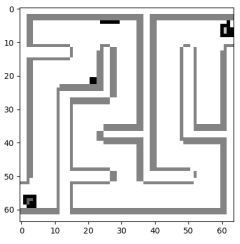


Figure 23: Environnement A par l'agent

[10]Vue classique des jeux vidéo d'action, où l'on voit seulement ce que l'agent voit

[inline]Transformer en tiré ou tableau Nous pouvons voir en rouge les coins où l'agent va recevoir une pénalité de -0.1, et en vert foncés les coins où l'agent va recevoir une récompense de +0.1. Le coin en vert clair représente l'arrivé souhaitée (avec une récompense de +1). Les lieux en noir sont des lieux où l'agent est très fortement pénalisé (par une pénalité de -1) et une interruption de l'environnement (équivalent à un game over[11]). Quand l'agent passe dans une zone où il reçoit une récompense, toute la zone disparait pour éviter des stratégies où l'agent tournerait en rond dans une zone à récompenses.

Ci-dessous un tableau récapitulatif des spécifications de l'environnement:

Spécifications	
Gestion des actions	De 4 à 9 actions (modulables) [12]
Durée des actions	Quatres pas de temps
Format de l'état	2D
Taille de l'image	42 * 42
Prise en compte de l'orientation	Oui[13]

Nous avons noté un apprentissage très difficile avec ce type d'environnement. Cela est principalement dû à des problèmes réseaux qui ont ralenti et complexifié l'entraînement. Néanmoins, nous avons noté que l'état donné à l'agent était difficilement exploitable. Un problème avec la représentation qu'on a adopté (en 2D) est l'absence de vision du placement des récompenses. Or, comme nous l'avons déjà noté, pour réussir un apprentissage, l'agent doit être capable de récupérer des informations pertinantes de l'état. Cela n'est pas possible avec la représentation actuelle.

Nous avons donc utilisé le même environnement avec une représentation. Ci-dessous, la véritable image sortie du simulateur et l'image donnée à l'agent pour l'apprentissage.

[11] partie perdue dans un jeux video

[13] On utilisera les actions suivantes: HAUT, BAS, DROITE, GAUCHE, DIAGONALES (4 actions), et NO-OP (ne rien faire)

[13] Les actions ont un effet différent en fonction de l'orientation

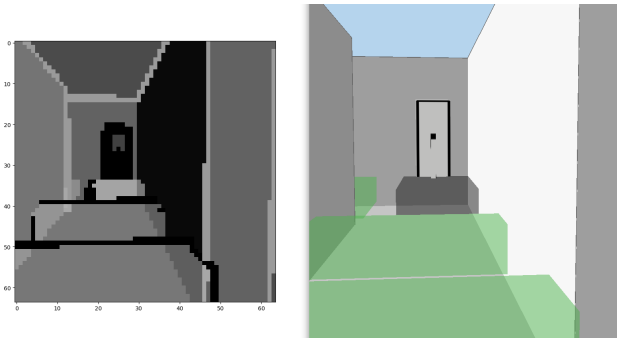


Figure 24: Représentation 3D de l'environnement

Comme le montre les figures au dessus, la vue est partielle et correspond à ce que voit l'agent. Les zones de récompenses sont maintenant visibles mais le résultat n'est pas satisfaisant. En effet, avec la nouvelle représentation qui est une vue 2D partielle est particulièrement difficile pour notre agent. Les environnements 3D avec une vue 2D donnés à un agent sont complexes pour des agents entraintés via A3C (ou DQN). Cela s'explique car ce genre d'algorithmes utilise une architecture qui cherche des éléments saillants dans l'environnement pour pouvoir approximer la Q fonction et trouver la politique optimale. Or dans notre environnement il y a peu d'élements possiblement interessants pour approximer la Q fonction. De plus, la Q fonction (ou fonction d'état-action) donne une estimation de la valeur d'une action suivant une politque. Le problème est que nous utilisons des actions qui dépendent fortement de l'orientation de l'agent. Si l'agent choisit d'avancer, selon l'orientation de celui-ci, l'action AVANCER aura un sens radicalement différent du point de vuede l'agent. Cela implique que l'approximation recherchée est quasiment impossible.

Pour régler un maximum de problèmes, nous avons créé un environnement similaire qui possède plus d'éléments caractéristiques pour aider l'agent. Nous avons supprimé la dépendance par rapport à l'orientation de l'agent.

Comme nous le voyons, l'environnement est plus coloré pour permettre à l'agent un apprentissage plus aisé. La spécification reste globalement la même sauf pour la dépendance par rapport à l'orientation.

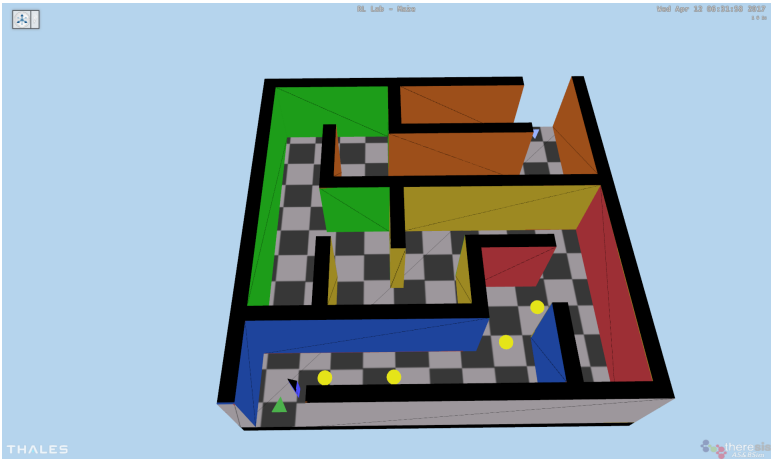


Figure 25: Représentation 3D de l'environnement B

Spécifications	
Gestion des actions	4 actions (modulables) [14]
Durée des actions	Quatres pas de temps (modulables)
Format de l'état	3D
Taille de l'image	42 * 42
Prise en compte de l'orientation	Non

Avec cet environnement, nous avons une bonne base d'expérimentation. Malheureusement, nous n'avons pas pu expérimenter autant de temps que nous l'aurions voulu. Nous avons pu démontrer avec cet environnement qu'il était possible de faire intéragir des algorithmes d'apprentissages par renforcement avec un logiciel de simulation qui n'était pas adapté à cela. Compte tenu des résultats que nous avons eu, nous avons décidé d'intégrer des mécanismes de curiosité dans notre contrôle pour permettre à l'agent d'explorer l'envrionnement de façon beaucoup plus efficace.

Le manque d'exploration a été une difficulté majeure pour l'agent. Des stratégies utilisant des motivations auxiliaires ont été mises au point pour répondre à ce problème. Nous explorerons cette idée dans la partie suivante.

4.4 Module de curiosité

Notre contrôleur mis en place, notre agent avait bien du mal à atteindre la moitié de l'environnement. Il est évident que l'apprentissage dans un environnement 3D était bien plus difficile. Pour nous convaincre de la difficulté d'un apprentissage sur un environnement labyrinthique, nous avons testé notre contrôle sur l'environnement de test Vizdoom (voir TODO section ?). Même constant, il est difficile pour l'agent d'explorer l'environnement et de dépasser la moitié de la carte.

Il y a deux explications naturelles à ce manque d'exploration:

1. L'agent ne reçoit qu'un état partiel de l'envrionnement et qui est déformé (passage 2D / 3D).

2. Notre algorithme de contrôle ne contient aucun mécanisme poussant l'agent à explorer son environnment. Empiriquement, l'agent tourne en rond car l'A3C-LSTM-GAE favorise une uniformisation de la distribution de la politique. Dans notre cas, cela a pour effet d'avoir un agent qui va essayer toutes les actions (droite, gauche, haut, bas), et donc tourner en rond.

Dans cette partie, nous allons expliquer comment fonctionne le module de curiosité qui a été mis en place. Nous commencerons par introduire succintement le dilemme exploration-exploitation, les différentes solutions dans le cadre de la théorie des Bandits à plusieurs armes et nous finirons pas discuter de la possibilité d'étendre les résultats au cadre de l'apprentissage par renforcement.

4.4.1 La théorie des bandits à plusieurs bras

Dans la théorie des bandits à plusieurs bras, un agent a un ensemble de K actions. A chaque action est associé une distribution P_a de moyenne μ_a. A chaque fois que l'agent choisit l'action a, il reçoit une récompense samplée depuis la distribution P_a soit $r_a^t \sim P_a$. L'objectif de l'agent est de maximiser la somme des récompenses obtenues.

$$\text{Maximiser:} \quad \sum_{t=1}^{T} r_{a_i}^t \text{ avec } r_{a_i}^t \sim P_a$$

Il y a donc un paralèlle à faire entre le contexte de la théorie des bandits et celui de l'apprentissage par renforcement. Une dès grande différences est qu'en

apprentissage par renforcement la distribution des récompenses est dépendante de la séquence d'action et possiblement dépendantes du temps.

Nous nous intéressons dans cette partie à un contexte pour simple pour illustrer théoriquement le recours à un module de curiosité.

La notion de **regret** est centrale, et est défini comme la différence entre le choix optimal des actions et le choix de l'agent (Regret $= \sum_{t=1}^{T} \left({r_{a*}}^{t} - r_{a_i}^{t} \right)$).

L'idéal serait d'avoir un algorithme minimisant le regret obtenu par l'agent (trouver l'action qui mène à obtenir le maximum de récompenses). Au début, l'agent n'a évidemment pas idée des distributions P_a. Pour pouvoir estimer les dites distributions, il sera obliger d'effectituer des actions plus ou moins au hasard (d'**exploration** l'espace des actions). Toute la difficulté de la théorie des bandits et de savoir comment effectuer des actions de manière à explorer leur espace et savoir quand on a assez exploré pour exploiter la connaissance des estimés des distributions.

La litérature concernant les bandits est riche néanmoins nous souhaitons justifier l'utilisation de mécanisme de curiosité via la théorie des bandits en montrant en quoi les stratégie naive son inéfficace. Ainsi, nous nous attarderons sur quatres stratégies populaires: les méthodes **Aléatoire**, **greedy**, **ϵ-greedy**, *et* **UCB**.

1. **La sélection d'actions aléatoire.**
   C'est la stratégie la plus simple qui consiste à aléatoirement choisir une action. Cette stratégie est inefficace pour plusieurs raisons, la plus importante est qu'elle ne prend en compte les récompenses obtenues antérieurement. Le regret espéré avec la méthode aléatoire est égale à: $T\left(\mu^* - \widetilde{\mu} \right)$. Le regret croit de manière linéaire avec cette méthode, ce qui indique l'incapacité de cette méthode à trouver l'action optimale (ce qui est évident compte tenu de la méthode)

2. **La sélection d'actions selon la méthode greedy**
   La méthode greedy (pour avide) est une méthode de selection qui se contente de selectionner l'action qui à l'instant t, selon l'estimation de l'agent, donne la meilleur espérence de gain (de façon plus formelle $a = \underset{a}{\text{argmax}} Q[a]$, avec Q le gain espéré estimé). Le problème est donc que l'agent va choisir continuellement un action dès lors qu'il n'aura plus d'égalité en espérence de gain selon les actions. Or, il est fort problème que l'estimation de l'agent soit mauvaise et donc qu'il se trompe d'action optimale. Avec cette méthode, le regret espéré est de: $T\left(\mu^* - \widetilde{\mu_{a_i'}} \right)$, avec i' l'action choisie en premier. Sans

surprise, le reget crois de manière linéaire.

3. **La sélection d'actions selon la méthode ϵ-greedy**
   La méthode ϵ-greedy est une variante de la méthode greedy dans laquelle l'action est choisie de façon greedy avec une probabilité de 1-ϵ et de façon aléatoire avec une probabilité ϵ. Nous avons une borne inférieur de regré espéré qui est de $T\epsilon\big(\ \mu^* - \widetilde{\mu}\ \big)$. Dans ce cas, le regret croit aussi de manière linéaire néanmoins nous voyant que si nous pouvons faire décroitre ϵ alors il serait théoriquement possible de faire croitre le regret seulement de façon logarithmique. C'est la méthode de sélection utilisé dans le **Deep-Q-Learning** et ses variantes.

4. **La selection d'actions selon la stratégie UCB**
   pour *Upper-Confidence-Bound* repose sur le principe d'optimisme face à l'incertitude. Ce principe stipule que plus nous sommes incertain de notre estimation sur le gain espéré plus il faut explorer cette action. On peut montrer théoriquement que dans le contexte des bandits simples, cela correspond à choisir de façon greedy plus un facteur de motivation à l'exploration. Soit formellement:
   $$\text{action} = \underset{a}{\text{argmax}}\Big[Q[a] + \text{Curiosité}\Big]$$
   avec Curiosité $= \sqrt{\frac{2\log t}{N(a)}}$. On peut montrer que le regret croit de façon logarithmique ce qui est mieux que les stratégies précédantes.

On pourrait donc imaginer utilisé une variante de l' dans le cadre de notre projet de contrôle. Néanmois, les formules données ci dessus ne sont valables que dans le cadre restreint au bandits. Sans pour autant utiliser , on pourra partir de cette idée de nécessité une motivation auxiliaire poussant l'agent à explorer l'environnement. Dans la partie suivante, nous allons tacher de définir plus précisement ce que veut dire d'explorer un environnement et nous allons proposer un module de curiosité pour arriver au contrôle de notre agent.

4.4.2 Exploration et Motivation auxiliaire

Le besoin d'exploration vient du simple fait que pour estimer la fonction d'état (ou d'état action), il faut avoir déjà recontré l'état visé. Or, dans le cas d'un environnement où l'agent n'a accès qu'a une vue partielle de l'environnement il est difficile d'estimer ces fonctions pour chaque état. Il apparait donc nécessaire de pousser l'agent à explorer en partie l'envrionnement pour déterminer au mieux une politique. Commne nous l'avons vu, dans la partie précédente, l'utilisation d'une

récompense auxiliaire poussant à explorer l'environnement à du sens. Néanmoins, dans le contexte de l'apprentissage par renforcement les stratégies du style UCB (ou Bayesienne avec le sampling de Thomson) sont difficilement réalisable.

Pourtant, ce n'est pas la seule difficulté, l'état que reçoit notre agent est une représentation partielle de l'environnement. En effet, l'agent ne voit qu'un représentation 2D d'un environnement 3D. La question est donc de savoir comment explorer à partir d'une représentation compréssée de l'état. Cette problématique est au centre de la récherche en apprentissage par renforcement. Il y a plusieurs piste envisagées.

1. Objectifs secondaires aidant l'agent dans ça prise de décision. Par exemple, nous pourrions optimiser notre réseau de neurones pour qu'il maximise la réduction de profondeur ($\text{depth}_{s1} > \text{depth}_{s2}$). Le problème de cette approche est qu'elle est dépendante de l'environnement.Or l'utilisation de l'apprentissage par renforcement est intéressante car elle permet d'être applicable à un grand nombre d'environnement sans grand changement.

2. Formaliser le concepte d'exploration de façon générique. Nous chercherons une manière d'explorer un environnement à partir d'une représentation latente (compréssée) de celui ci. Il n'y a pas de consensus sur ce sujet actuellement.

3. Utiliser des objectifs secondaires génériques. Par exemple, si nous pouvions formaliser de manière même simpliste la notion de curiosité, on peut faire l'hypothèse aurait pour effet une amélioration des performences l'algorithme car il en découlerait une exploration.

4.5 Motivations auxiliaires et motivations

Les motivations auxiliaires ont pu but d'aider l'agent à explorer son environnement. En effet, plus l'état est dimensionnelle élevée plus cela nécessite des approches particulières pour garantir que l'agent va être pousser à explorer celui-ci. Sans module auxiliaire, il est possible que l'agent reste dans un périmêtre restreint ce qui n'est pas souhaitable.

4.5.1 Motivations auxiliaires spécifiques au environnements 3D

Dans un premiers temps, nous allons explorer un module de curiosité simple que nous permettra d'introduire plus tard le module de curiosité qui a été integré dans

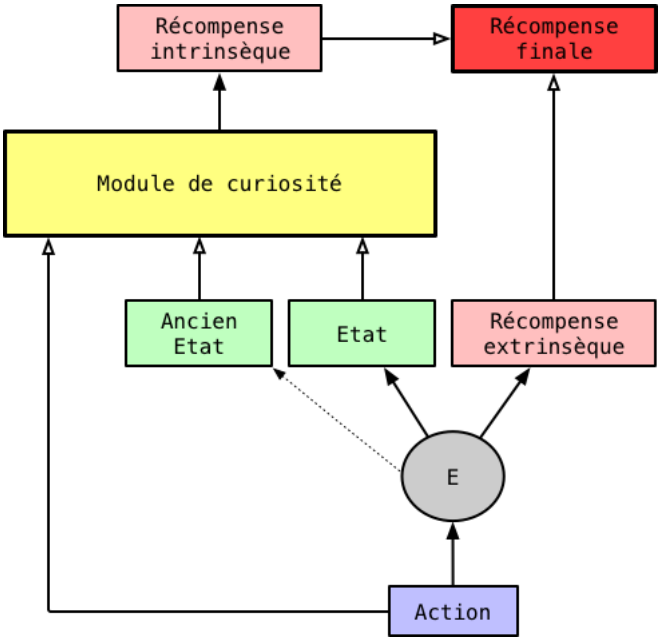


Figure 26: Structure d'un module de curiosité

le contrôle. Ce module par d'un constat simple,en poussant l'agent à changer en permanance le flux d'entrée qu'il perçoit, l'agent devra explorer l'environnement. En effet, bien souvent un manque d'exploration implique que les états que voient l'agent resteront similaire. Si l'agent recherche des états qui sont différents les uns des autres alors on peu faire l'hypothèse que l'agent sera pousser à explorer en profondeur l'environnements pour continuer de changer son flux d'entrée.

Formellement sela consiste à prendre la différence entre deux états sucessifs et considérer leurs normes comme un signal pouvant aider l'agent.

$$\text{récompense intrinsèque} = \lambda \parallel s_1 - s_2 \parallel^2 \tag{19}$$

L'équation Equation 19 est trop simple pour être utiliser. En effet, prenons le cas d'un agent qui tourne en rond, alors selon l'équation 19 l'agent serait encouragé à tourner. Ce comportememt exclu donc l'utilisation de ce type d'approche naive. Nous allons décrire une méthodequi se base sur l'intuition que nous avons developpé en utilisant une approche hiérarchique. L'idée est que l'agent dévelope un mécanisme d'attention lui permettant de savoir qu'elle sous-partie de l'état est intéressante et avec laquelle on peut employer la stratégie simpliste. Formellement

cela revient à apprendre la séquence des k tel quel:

$$\text{récompense intrinsèque} = \tau \frac{\| h_k \odot [s_t - s_{t-1}] \|^2}{\| s_t - s_{t-1} \|^2} \tag{20}$$

L'équation (**??**) montre comment utilisé seulement une sous partie de l'entrée pour calculer la récompense intrinsèque. Cette formule est utilisé dans le papier *Feature Control as Intrinsic Motivation for Hierarchical Reinforcement Learning*[5] Nous ne détaillerons pas la méthode pour déterminer la séquence de k car elle est extrêmment similaire à la façon de determiner la séquence d'action à effectuer. Nous avons aussi créer une méthode qui généralise la précédente qui au lieu de choisir un k, pondère par un facteur choisi par l'agent l'ensemble des sous ensembles de l'état pour calculer la récompense intrinsèque.

Nous avions peur de déstabiliser le réseau avec ce genre de recompense, le choix des hyperparamètres (λ, τ) nécessite du soin et de nombreux essais.

Les tests préliminaires ont montrés que cela ne destabilisé pas l'apprentissage, néanmoins cela mériterais plus de test pour être convaincu de cela.

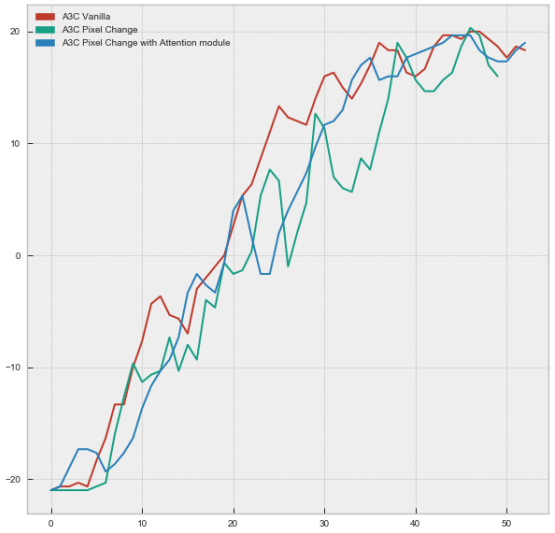


Figure 27: Résultats des motivations auxiliaires sur Pong (Gym-OpenAI)

Comme vous pouvez le voir le module a été testé avec comme contrôleur l'A3C. Les résultats indiquent que la recherche de la séquence de k pour un cas simple (Pong) n'augmente pas significativement l'entrainement. Il est reste maintenant a testé ce module avec SE-STAR. Les premiers résultats du contrôleur ne sont

pas disponible avec ce module, il est noté que ce type de module n'a jamais été utilisé dans un cas aussi complexe qu'un environnement 3D crée par SE-STAR mais a été utilisé pour permettre à un agent d'explorer un environnement 2D qui nécessite une forte exploration et qui ne donne pas ou peu de signal intéressante pour l'apprentissage d'une politique optimale.

La solution présenté a été envisagé mais n'a pas été retenu dans le cadre du contrôle de l'agent. Nous avons utlisé un module de curiosité qui est plus générique.

4.6 Module de curiosité générique intégré au contrôle

Nous allons maintenant décrire la solution qui a été retenue dans le contrôle de SE-STAR. La solution proposée par des principes motivationnelles expliquées précédemment. La seule différence vient du faite que nous ne faisons plus un différence sur l'espace des pixels mais sur l'espace latent (construit par le réseau de neurone). L'avantage de cette méthode est de réussir à compresser l'état pour en extraire uniquement ce qui est indispensable. De plus, le module de curiosité utilise une méthode de compression supplémentaire qui a pour but de supprimer les parties qui ne sont pas influencé par l'action. Ce module est très fortement inspiré du papier *Curiosity-driven Exploration in Deep Reinforcement Learning via Bayesian Neural Networks*[8]

Le module de curiosité est composé de plusieurs sous-modules:

- **sous-module inverse**
  Le sous module inverse a pour but de trouver une compression permettant de réduire la dimensionnalité de l'état tout en étant capable de retrouver l'action qu'a effectué l'agent. Cela a pour but de travailler avec en entrée un nouvelle état qui est le plus pertinant possible. Imaginons le cas où l'agent se trouve devant un arbre, celui-ci aura les feuilles bougeants en permanence. Avec la stratégie de curiosité descrite précédemment il est alors aiséde voir que l'agent sera encouragé à ne pas bouger. Or, avec ce nouveau module de curiosité, comme l'agent ne peut agir sur les feuilles, ce sous module supprimer les feuilles et ainsi le module de curiosité aura un effet bien plus stable et ne sera pas sujet à des scénarios défavorables.

- **sous-module avant**
  Le sous module avant à pour but de déterminer la récompense intrinsèque qui sera donné à l'agent. En utilisant les compressions des états à l'instant t et t+1 que l'on nommera $\phi(s_t)$ et $\phi(s_{t+1})$, on peut appliquer la même

stratégie que décrite dans le chapitre précédent. La récompense intrinsèques est donnée par la formule suivante:

$$\text{récompense intrinsèque} = \tau \parallel \phi(s_t) - \phi(s_{t+1}) \parallel^2 \tag{21}$$

L'association permet d'utiliser notre stratégie naive tout en évitant les principales de faiblesse de celle ci.

Voici un schéma récapitulatif de la méthode de curiosité ci dessus. C'est une proposition du schéma de curiosité naif en proposant une méthode de compression qui supprime les parties de l'états qui ne sont pas intéressant dans la cadre de la curiosité de l'agent.

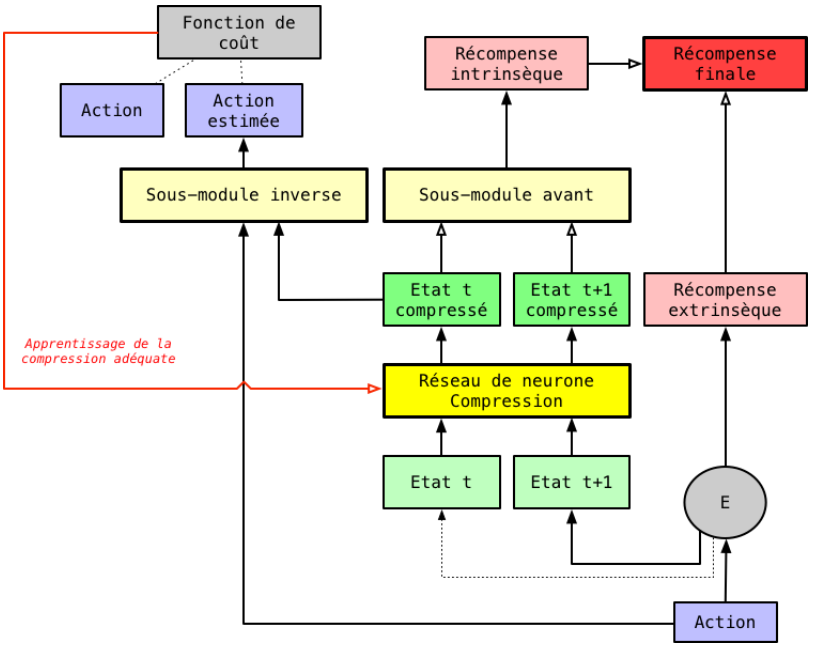


Figure 28: Schéma finale du module de curiosité mise en place

Le module de curiosité est plus complexe que celui présenté ci-dessus néanmoins le module présenté représente l'idée globale de l'algorithme. Ce module de curiosité a été testé conjointenement avec l'A3C-LSTM-GAE avec les environnements Vizdoom (Doom) et SE-STAR. Les premiers résultats sont encourageants mais il faudra rêgler les quelques problèmes d'interface entre SE-STAR et notre contrôleur pour véritable tester la conjonction des deux algorithmes sont des environnements 3D plus complexe sur SE-STAR.

5 Conclusion

5.1 Contributions

Durant ce stage, nous avons mis au point une interface entre SE-STAR et les principaux algorithmes d'apprentissage profond. Un asservissement d'un agent, naviguant de l'environnement simulé par SE-STAR grâce à notre interface, a été construit. Notre asservissement à comme particularité d'être doté d'un module de curiosité exploitant des mécanismes *"assimilable de façon lointaine à la curiosité humaine"* dans le but de résoudre le dilemme exploitation / exploration.

5.2 Discussions et travail futur

Plusieurs autres directions pourraient être intéressantes à explorer. Nous parlerons exclusivement des pistes pour améliorer notre contrôleur dans cette partie. La première piste à explorer serait l'introduction de mécanismes utilisants la dynamique de l'environnement dans notre contrôle. En effet, actuellement, notre apprentissage repose sur des algorithmes ne requierant pas de modèle. Pourtant, il existe des algorithmes utilisants la dynamque du modèle pour créer des modules similaires à notre module de curiosité. Nous pouvons d'ailleurs à bien des égards voir notre module de curiosité comme un module se servant d'une partie de la dynamque du modèle.

En se basant sur l'idée d'utiliser le modèle pour y extraire des entrées auxiliaires intéressantes, il en ressort deux grandes voies empruntables:

- **Théorie de l'information: Empowerment, entropie relative:**
  A travers ce stage, il en ressort une tentative de formalisation du concept de curiosité dans l'espoir qu'il aide l'agent à parcourir l'environnement pour en extraire une politique optimale. Les approches de constructions de motivations auxiliaires basées sur la théorie de l'information utilisent des notions mesurant l'incertude d'une variable aléatoire sachant la connaissance d'une autre variable aléatoire (entropie mutuelle). Se basant sur ces mesures, nous pouvons utiliser comme récompenses auxiliaires l'information donnée par une séquence d'action sur un état futur. L'idée sous jacente étant non seulement de pousser l'agent à explorer l'environnement mais aussi d'encourager l'environnement à prendre contrôle de l'environnement (reposant sur le concept d'empowerment [10], [16], [17]). Les principaux contrôles basés sur ces notions sont utilisés sur des cas simples, il est impossible de calculer l'entropie

mutuelle (ou l'empowerment) dans les cas où la dimensionnalité de l'état est haute (ou continue). Des approches basées sur des approximations variationnelles se mettent en place ([6]). C'est donc une voie à ne pas négliger dans la conception d'un module de curiosité qui soit le plus bénéfique pour l'agent possible.

- **Probabilité bayésienne, incertitude et curiosité**
  Une façon simple de définir l'incertitude est de la considérer comme étant une mesure de la réduction de l'incertitude de l'agent sur un certain espace (d'état futur, du modèle ...). Pourtant, les réseaux de neurones sont pour la plupart déterministiques (à une entrée A correspond une sortie B). Or, ils serait interessant de pouvoir encoder directement dans le réseau de neurones l'incertitude (sur un espace) de l'agent. Pour cela, nous pouvons faire appel aux probabilités bayésiennes qui donnent un cadre formel parfait pour prendre en compte l'incertitude. L'intégration de réseaux de neurones bayésiens est un champ de la recherche très actif ([2]). Une récompense auxilaire pourrait être proportionnelle à la réduction d'incertitude concernant la dynamique du modèle (à supposer que l'agent garde en mémoire une estimée de la dynamique) [9]

Il y aurait beaucoup d'approches à explorer, et ces approches sont bien moins découpées qu'il semblerait. L'approche du contrôle utilisant des mécanismes motivationnelles intrinsèques est prometteuse. Particulièrement, dans des cas où les approches usuelles ne fonctionnent pas.

Ce stage m'aura donc permis d'explorer à travers un objectif concrêt, l'asservissement d'un agent simulé via des motivations intrinsèques intégrées à un contrôle par apprentissage par renforcement profond.

Glossaire

API En anglais: *Application program interface*, c'est un ensemble de protocoles, ou de contrats spécifiant le fonctionnant d'un programme. Dans le cas de ce rapport, cela correspond à un contrat entre l'application client et serveur (SE-STAR). vii, 30, 31

GPU En anglais: *Graphics Processing Unit*, est un composant normalement utilisé dans la gestion des graphismes qui permet d'effectuer des calcules hautement parallèlisable de façon extrêmement rapide. 34

PAAC Référence à une méthode en apprentissage par renforcement nommée *Efficient Parallel Methods for Deep Reinforcement Learning* [4]. Elle se repose sur la même stratégie que l'A3C en utilisant une architecture synchrone permettant l'utilisation du GPU. 34

RL RL est l'acronyme de reinforcement learning qui en francais se traduit par apprentissage par renforcement.. 22

Wine En anglais: *Wine Is Not an Emulator*, est une application permettant l'utilisation d'application Windows sous Linux. Son nom vient du faite que de nombreuses personnes pensent a tort que Wine est un 'emulateur de Windows' or Wine a réimplémenté l'ensemble des appels systèmes dans le user space linux permettant à l'utilisateur de faire fonctionner une application Windows sous une distriubtion Linux. vii, 34

References

[1] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, J. Schrittwieser, K. Anderson, S. York, M. Cant, A. Cain, A. Bolton, S. Gaffney, H. King, D. Hassabis, S. Legg, and S. Petersen, *Deepmind lab*, CoRR, abs/1612.03801 (2016).

[2] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, *Weight Uncertainty in Neural Networks*, ArXiv e-prints, (2015).

[3] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *Openai gym*, 2016.

[4] A. V. Clemente, H. N. Castejón, and A. Chandra, *Efficient Parallel Methods for Deep Reinforcement Learning*, ArXiv e-prints, (2017).

[5] N. Dilokthanakul, C. Kaplanis, N. Pawlowski, and M. Shanahan, *Feature control as intrinsic motivation for hierarchical reinforcement learning*, CoRR, abs/1705.06769 (2017).

[6] K. Gregor, D. J. Rezende, and D. Wierstra, *Variational intrinsic control*, CoRR, abs/1611.07507 (2016).

[7] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural Comput., 9 (1997), pp. 1735–1780.

[8] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel, *Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks.*, CoRR, abs/1605.09674 (2016).

[9] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel, *Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks*, CoRR, abs/1605.09674 (2016).

[10] T. Jung, D. Polani, and P. Stone, *Empowerment for continuous agent-environment systems*, CoRR, abs/1201.6583 (2012).

[11] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski, *Vizdoom: A doom-based AI research platform for visual reinforcement learning*, CoRR, abs/1605.02097 (2016).

[12] D. Merkel, *Docker: Lightweight linux containers for consistent development*

and deployment, Linux J., 2014 (2014).

[13] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, *Asynchronous methods for deep reinforcement learning*, CoRR, abs/1602.01783 (2016).

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing Atari with Deep Reinforcement Learning*, ArXiv e-prints, (2013).

[15] 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, 518 (2015), pp. 529–533.

[16] S. Mohamed and D. Jimenez Rezende, *Variational Information Maximisation for Intrinsically Motivated Reinforcement Learning*, ArXiv e-prints, (2015).

[17] C. Salge, C. Glackin, and D. Polani, *Empowerment – an Introduction*, ArXiv e-prints, (2013).

[18] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, *Prioritized experience replay*, CoRR, abs/1511.05952 (2015).

[19] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, *High-dimensional continuous control using generalized advantage estimation*, CoRR, abs/1506.02438 (2015).

[20] sebastian raschka, *Single-layer neural networks and gradient descent*, 2015.

[21] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour, *Policy gradient methods for reinforcement learning with function approximation*, in In Advances in Neural Information Processing Systems 12, MIT Press, 2000, pp. 1057–1063.

[22] H. van Hasselt, A. Guez, and D. Silver, *Deep reinforcement learning with double q-learning*, CoRR, abs/1509.06461 (2015).

[23] Z. Wang, N. de Freitas, and M. Lanctot, *Dueling network architectures for deep reinforcement learning*, CoRR, abs/1511.06581 (2015).

[24] C. J. C. H. Watkins and P. Dayan, *Q-learning*, in Machine Learning, 1992, pp. 279–292.

[25] R. J. Williams, *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, Machine Learning, 8 (1992), pp. 229–256.