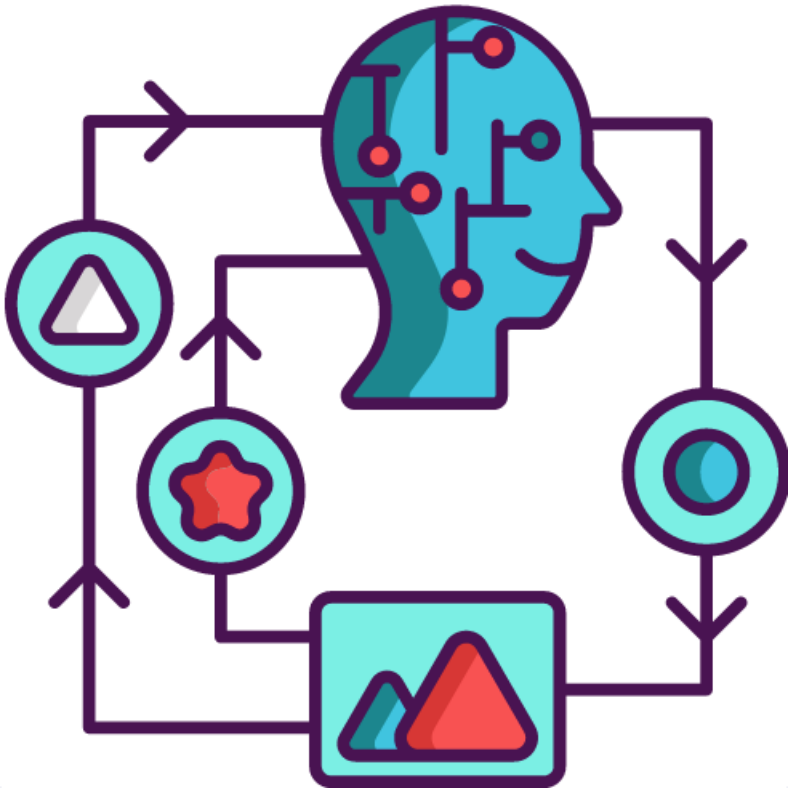
TX

P24

Sacha SZENDROVICS
Théo DUBUS

Notes d'apprentissage TX

Apprentissage par renforcement



Projet

ALLIANCE SORBONNE UNIVERSITÉ
utc

Sommaire

Introduction

Nous avons été amenés, dans le cadre d'une TX réalisée au cours du semestre de printemps 2024, à étudier l'apprentissage par renforcement. L'objectif de ce document est de transmettre les différentes connaissances acquises lors de notre apprentissage, ainsi que de faire un point sur la mise en pratique que nous avons faite de ces dernières, au travers une implémentation simple des concepts clés de l'apprentissage par renforcement. Notre apprentissage se base fortement sur la formation disponible sur le site Hugging Face disponible sur ce lien([2]).

La formation que nous avons suivi est largement basé sur ce livre d'apprentisssage par renforcement ([1]).

1 Qu'est-ce que l'apprentissage par renforcement ?

Dans cette section, nous allons définir l'apprentissage par renforcement et ses différentes composantes. Nous allons expliquer les différentes notions fondamentales de l'apprentissage par renforcement, les différentes politiques d'action, ainsi que les difficultés pouvant être rencontrées et les solutions qui les accompagnent.

1.1 Définition formelle et principes de bases

L'apprentissage par renforcement, ou reinforcement learning (RL), est un cadre d'apprentissage automatique qui permet à un agent d'apprendre à prendre des décisions dans un environnement incertain et dynamique. L'agent apprend à partir de ses interactions avec l'environnement, en recevant des signaux de renforcement (ou récompenses) qui l'aident à ajuster son comportement pour atteindre un objectif spécifique.

Plus précisément, l'apprentissage par renforcement peut être défini comme suit : il s'agit d'un processus d'apprentissage par essais et erreurs, dans lequel un agent apprend à prendre des décisions en interagissant avec un environnement et en recevant des récompenses (positives ou négatives) sous forme de rétroaction unique.

Un agent est défini comme une entité autonome capable d'observer son environnement et de choisir des actions basées sur ses observations. L'environnement est défini comme le cadre dans lequel évolue l'agent. Ce dernier est influencé et évolue en fonction des actions de l'agent.

Le principe de fonctionnement de l'apprentissage par renforcement est le suivant :

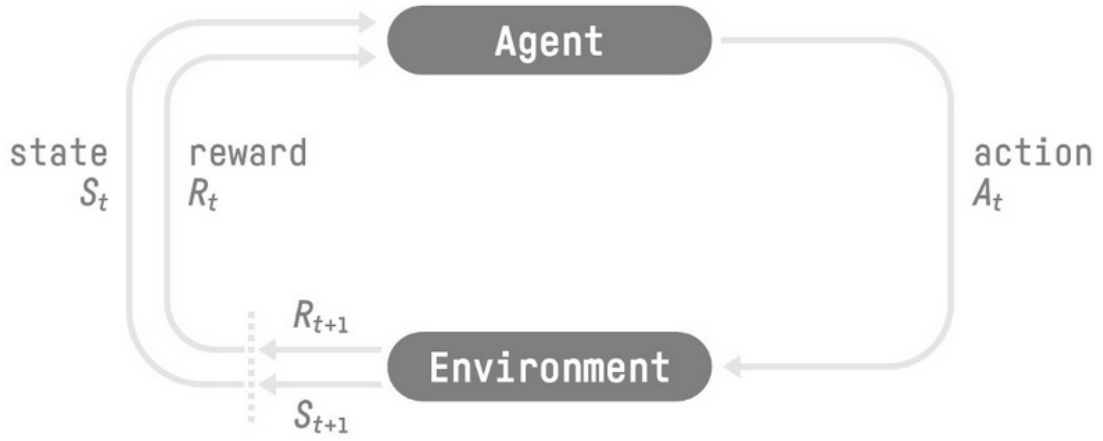


FIGURE 1 – Cycle de fonctionnement du RL

Dans ce processus, l'agent reçoit l'état S_0. En fonction de sa politique d'action, il choisit une action A_0 à effectuer dans l'environnement. Cette action modifie l'environnement, qui passe alors dans un nouvel état S_1. Cette transition d'état produit également une récompense R_1 (positive ou négative), qui est envoyée à l'agent pour l'aider à ajuster son comportement. L'agent reçoit alors ces deux valeurs (S_1 et R_1) et utilise cette rétroaction pour mettre à jour sa politique d'apprentissage et choisir une nouvelle action à effectuer.

1.1.1 Description de l'espace d'état et d'action

L'espace d'état est l'ensemble de toutes les configurations possibles de l'environnement. Il peut être représenté de différentes manières, en fonction de la nature de l'environnement et des capteurs de l'agent. Dans certains cas, l'état de l'environnement peut être complètement observable par l'agent, ce qui signifie que toutes les informations pertinentes sont disponibles pour l'agent (exemple : une partie d'échecs). Dans d'autres cas, l'état peut être partiellement observable, ce qui signifie que l'agent ne peut accéder qu'à une partie des informations pertinentes (exemple : le champ de vision d'un joueur pour un jeu de course). Dans ce cas, on parle d'observation plutôt que d'état.

L'espace d'action est l'ensemble de toutes les actions possibles que l'agent peut effectuer dans l'environnement. Comme pour l'espace d'état, il peut être représenté de différentes manières, en fonction de la nature de l'environnement et des capacités de l'agent. Les actions peuvent être discrètes, ce qui signifie qu'il y a un nombre fini d'options possibles (par exemple, se déplacer vers la gauche, vers la droite, vers le haut ou vers le bas). Elles peuvent également être continues, ce qui signifie qu'il y a un nombre infini d'options possibles (par exemple, la vitesse à laquelle un véhicule se déplace ou l'angle avec lequel on tourne le volant).

1.1.2 Calcul de la récompense

Le calcul de la récompense est fondamental, car c'est le seul retour que l'agent va recevoir vis-à-vis d'une action. On peut voir la récompense de la façon suivante : une récompense positive est associée à une bonne action et une récompense négative est associée à une mauvaise action choisie par l'agent. Ainsi, améliorer notre agent revient donc à maximiser sa récompense. L'agent doit donc effectuer ses actions en prenant en compte des trajectoires, des suites d'actions et d'états potentiels pour lesquels il estimera une récompense cumulée probable en résultant.

Une première approche naïve serait de calculer cette récompense cumulée ainsi :

$$R(\tau) = r_{t+1} + r_{t+2} + r_{t+3} + \cdots = \sum_{k=0}^{\infty} r_{t+k+1}$$

Où :
- t l'étape actuelle
- τ la trajectoire de l'agent, la séquence des états S_{t+k+1} et actions A_{t+k+1}
- r_{t+k+1} la récompense résultante de l'état suite à la $t + k + 1^{eme}$ action de l'agent

Cependant, cette manière de calculer donne la même importance aux récompenses proches et aux récompenses plus lointaines, ce qui n'est pas le toujours pertinent. En effet, une récompense proche est plus susceptible de se produire qu'une récompense plus lointaine. Nous ajoutons alors un taux d'actualisation $\gamma \in]0; 1]$, que l'on peut voir comme un facteur de décroissance d'importance des récompenses lointaines. Ainsi, un γ élevé donnera de l'importance aux récompenses lointaines, et un γ faible donnera plus d'importance aux récompenses proches. Nous pouvons

donc redéfinir le calcul de la récompense cumulée de la manière suivante :

$$R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

1.1.3 Types de tâches

Une tâche est une instance d'un problème d'apprentissage par renforcement. Il y a deux types de tâches : épisodiques et continues. Les tâches épisodiques sont des tâches pour lesquelles on a un point de départ et un point d'arrivée. (Exemple : un niveau de Super Mario Bros). Les tâches continues, quant à elles, sont des tâches qui durent éternellement, qui n'ont pas de point de départ ni de point d'arrivée. L'agent continue alors jusqu'à ce qu'on l'arrête. (Exemple : agent de trading en bourse).

Nous nous focaliserons ici sur les tâches épisodiques.

1.1.4 Compromis exploration / exploitation

La relation exploration-exploitation est un défi majeur dans l'apprentissage par renforcement. Comme nous l'avons évoqué, au fur et à mesure des essais et erreurs de l'agent, ce dernier en apprend sur la manière de se comporter avec son environnement, en essayant une grande variété d'actions et d'états. Il va ensuite se servir de son expérience pour augmenter sa récompense cumulée. Cette expérience peut parfois empêcher l'agent d'en apprendre davantage, ce dernier risque de répéter les meilleures actions qu'il a pu faire, sans tenter de nouvelles actions menant à des états encore non explorés, mais potentiellement meilleurs. Plaçons-nous dans ce cas de figure :

FIGURE 2 – Illustration du problème exploration / exploitation

Dans cet exemple, on peut imaginer sans problème que l'agent rencontrera tôt dans son apprentissage les récompenses autour de lui, et donc risque de rester indéfiniment dans ces états, sans aller chercher des récompenses potentiellement plus élevées dans d'autres états. Ces autres états ne sont pas associés à une forte récompense aux yeux de l'agent, car ce dernier ne les a pas explorés et donc n'a jamais fait l'expérience de la récompense qu'ils apportent.

Pour résoudre ce problème, on utilise souvent des stratégies d'exploration-exploitation qui font réaliser des actions aléatoirement à l'agent au début de la découverte de

l'environnement. Cela permet à l'agent d'explorer un peu plus son environnement et de découvrir des récompenses qu'il n'aurait pas trouvées autrement. Au fur et à mesure que l'agent découvre son environnement, on réduit progressivement la quantité d'exploration aléatoire et on se concentre davantage sur l'exploitation des récompenses les plus élevées.

1.2 Les différentes politiques d'action

Dans cette section, nous allons étudier les différentes stratégies que les agents peuvent utiliser pour choisir leurs actions et maximiser leur récompense. Nous appellerons par la suite π la politique d'action qui permet à l'agent de maximiser son rendement attendu en fonction de l'état dans lequel il se trouve.

Il existe deux approches pour cela : le choix de l'action peut être basé sur une politique de conduite, ou alors sur la maximisation d'une valeur associée à un état.

1.2.1 Choix d'action basé sur une politique

Il existe deux principaux types de politiques d'apprentissage : les politiques déterministes et les politiques stochastiques.

Les politiques déterministes sont des fonctions qui associent à chaque état une action unique à effectuer. Autrement dit, l'agent sait exactement quelle action effectuer en fonction de l'état dans lequel il se trouve. Ce type de politique est souvent utilisé dans les environnements simples où il est facile de déterminer la meilleure action à effectuer en fonction de l'état.

Ce type de politique est représenté par l'équation suivante :

$$\pi(s) = a$$

Où :
- s : l'état considéré
- a : l'action choisie

Les politiques stochastiques, en revanche, sont des fonctions qui associent à chaque état une distribution de probabilités sur les actions possibles. Autrement dit, l'agent ne sait pas exactement quelle action effectuer en fonction de l'état dans lequel il se trouve, mais il a une idée de la probabilité que chaque action soit la meilleure. Ce type de politique est souvent utilisé dans les environnements plus complexes où il est difficile de déterminer la meilleure action à effectuer en fonction de l'état.

Ce type de politique est représenté par l'équation suivante :

$$\pi_a(s) = P[A|s; a]$$

Où :
- $P[A|s]$: la distribution de probabilité de l'ensemble des actions étant donné l'état s
- $P[A|s; a]$: la valeur pour a dans $P[A|s]$
- A : l'ensemble d'action

Exemple

$P[A|s] = [Gauche : 0.1, Droite : 0.6, Haut : 0.2, Bas : 0.1]$
$A = \{Gauche, Droite, Haut, Bas\}$
On a alors $\pi_{Droite}(s) = P[A|s; Droite] = 0.6$

Il convient de noter que même si les politiques stochastiques peuvent sembler moins efficaces que les politiques déterministes, elles peuvent en fait être plus utiles dans certains cas. Par exemple, si l'environnement est très incertain ou s'il y a plusieurs bonnes actions possibles pour un état donné, une politique stochastique peut aider l'agent à explorer toutes les options et à trouver la meilleure solution à long terme. Une politique stochastique offre également une solution au compromis exploration / exploitation. En effet, pour l'entraînement, on peut choisir l'action suivante à réaliser directement à partir de la distribution de probabilité actuelle de l'état courant, n'excluant ainsi aucune action.

1.2.2 Choix d'action basé sur la valeur

Les méthodes de choix d'action basées sur la valeur sont une approche pour trouver la politique optimale de manière indirecte. Au lieu d'apprendre directement la politique optimale, les agents apprennent à évaluer la valeur de chaque état dans l'environnement. Cette valeur correspond au rendement actualisé attendu que l'agent peut obtenir en partant de cet état et en suivant la politique actuelle. La "politique actuelle" est une politique à définir manuellement en fonction de la valeur et dépend de l'algorithme considéré.

La fonction de valeur, notée $V_\pi(s)$, est utilisée pour associer une valeur à chaque état s dans l'environnement. Cette fonction est définie comme le rendement attendu en agissant selon la politique π à partir de l'état s. On la note ainsi :

$$V_\pi(s) = E_\pi[G_t|S_t = s]$$

avec :
— $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$

Il est également possible d'utiliser une fonction "Action-Valeur", notée $Q_\pi(s, a)$, qui associe une valeur à chaque paire (état, action) dans l'environnement. Cette fonction correspond au rendement attendu en partant de l'état s, en effectuant l'action a, puis en suivant la politique , on la note ainsi :

$$Q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a]$$

La fonction $V_\pi(s)$ mesure donc la valeur d'un état, tandis que $Q_\pi(s, a)$ mesure la valeur (l'intérêt) d'effectuer une action dans un certain état.

En résumé, les méthodes basées sur la valeur apprennent à l'agent à évaluer la valeur de chaque état ou de chaque paire (état, action) dans l'environnement, ce qui lui fournit les informations nécessaires pour prendre des décisions éclairées pour maximiser sa récompense à long terme.

2 Fonctionnement d'un agent

Maintenant que nous avons vu les différents concepts qui composent l'apprentissage par renforcement, nous allons étudier des techniques d'apprentissage concrètes.

2.1 Q-Learning

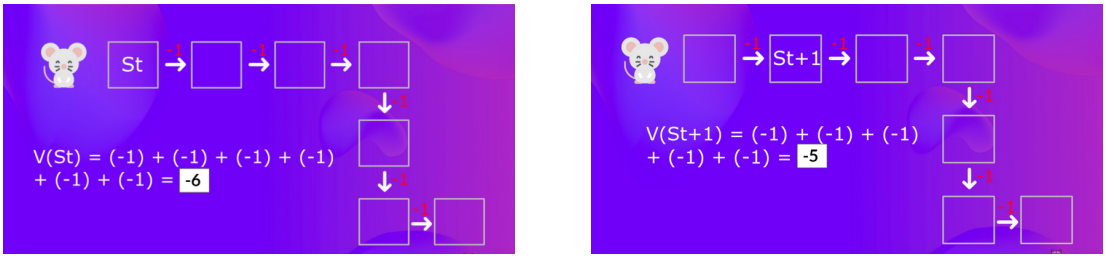
Nous approfondissons ici les méthodes de résolution de problèmes basées sur la valeur. Nous allons ainsi étudier un algorithme d'apprentissage par renforcement utilisant cette méthode de résolution, le Q-Learning.

Dans les méthodes basées sur les politiques, on n'a pas à définir manuellement de politique, car celle-ci s'apprend directement par entraînement. Dans les méthodes basées sur des valeurs, on doit en définir une à partir de notre fonction valeur. On peut par exemple choisir une politique qui choisit des actions menant à la plus grande récompense, on parle de politique gloutonne :

$$\pi(s) = arg\ \max_{a}\ Q_\pi(s,a)$$

2.1.1 Équation de Bellman

Le calcul de l'estimation E_π des récompenses G_t pour chaque $V(S_t)$ peut s'avérer coûteux, d'autant plus qu'il est répétitif :



(a) Calcul la valeur de l'état S_t (estimation de la récompense)

(b) Calcul la valeur de l'état S_{t+1} (estimation de la récompense)

FIGURE 3 – Illustration de la répétition du calcul de la valeur

Une grande partie du calcul de la valeur de deux états données est répété, ce qui est fastidieux si on doit le faire pour chaque valeur d'état ou chaque paire action état. C'est alors qu'on utilise l'équation de Bellman. On considère ainsi la valeur d'un état comme :

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma * V_\pi(S_{t+1})|S_t = s]$$

On utilise ainsi la récursivité pour, au lieu de calculer chaque valeur comme la somme du rendement attendu, la calculer comme la somme entre la récompense immédiate et la valeur actualisée de l'état qui suit. Cela épargne beaucoup de temps de calcul, car on n'a plus à sommer les récompenses à chaque itération.

2.1.2 Stratégies d'apprentissage

Il existe deux stratégies d'apprentissage pour mettre à jour les valeurs. La méthode Monte Carlo consiste à faire tourner un épisode entier en suivant la stratégie

actuelle, calculer G_t et s'en servir pour mettre à jour $V(S_t)$. Les valeurs sont ainsi mises à jour :

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Avec α le taux d'apprentissage.

La méthode par différence temporelle, quant à elle, consiste à n'attendre qu'une seule interaction, de S_t à S_{t+1} pour mettre à jour $V(S_t)$. On met ainsi à jour $V(S_t)$ à chaque étape. Cependant, comme on n'a pas encore vécu d'épisode entier, nous ne pouvons calculer G_t, il faut donc l'estimer à partir de la récompense immédiate et de la valeur actualisée de l'état suivant :

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Cette méthode est appelée TD(0), ou différence temporelle en une étape (mettre à jour la fonction de valeur après chaque étape individuelle).

2.1.3 L'algorithme Q-Learning

Q-learning est une méthode basée sur des valeurs qui utilise une approche de différence temporelle pour entraîner sa fonction de valeur d'action. On dit que Q-learning est une méthode "Hors politique", cela vient du fait qu'on n'utilise pas la même politique pour agir et pour mettre à jour la politique, nous verrons cela plus bas avec l'algorithme.

Dans le cadre du Q-Learning, nous utilisons la fonction de valeur $Q_\pi(s, a)$. Cette fonction $Q\pi$ est souvent codée comme un tableau, avec pour colonnes les actions possibles pour ligne les différents états. Voici un exemple de ce à quoi peut ressembler ce tableau avant l'entraînement :

	↑	↓	←	→
[0,0]	0	0	0	0
[0,1]	0	0	0	0
[1,0]	0	0	0	0
[1,1]	0	0	0	0

FIGURE 4 – Table Q au moment de son initialisation

Voici la définition de l'algorithme :

Algorithm 14: Sarsamax (Q-Learning)

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do** Step 1
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  $t \leftarrow 0$
  repeat
    Choose action A_t using policy derived from Q (e.g., ϵ-greedy) Step 2
    Take action A_t and observe R_{t+1}, S_{t+1} Step 3
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$ Step 4
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return Q

FIGURE 5 – Algorithme Q-Learning

Comme nous avons dit plus haut, nous pouvons voir qu'on n'utilise pas la même politique pour la mise à jour et pour l'inférence. En effet, on utilise ici la politique optimale pour la mise à jour, qui est définie comme suit :

$$\pi^*(s) = arg\ \max_a\ Q^*(s,a)$$

On utilise une politique différente pour l'inférence, introduisant une part d'aléatoire pour gérer le compromis exploration-exploitation. Cette politique s'appelle epsilon-greedy et consiste à prendre une action aléatoire avec une probabilité epsilon, la politique optimale sinon. Généralement, on fixe epsilon haut au début de l'apprentissage et on le réduit progressivement au fur et à mesure de l'apprentissage. Formellement, cette politique est définie comme suit :

Soit $\epsilon \in [0,1]$ un paramètre de la politique epsilon-greedy. Considérons une variable aléatoire X qui suit une distribution uniforme sur $[0,1]$, c'est-à-dire $X \sim \mathcal{U}(0,1)$. Notons x une réalisation de cette variable aléatoire.
La fonction $\pi(s)$ est définie comme suit :

$$\pi(s) = \begin{cases} \pi^*(s) & \text{si } x \geq \epsilon, \\ a \in A & \text{sinon, avec } a \text{ choisi uniformément au hasard dans } A. \end{cases}$$

Notons également le fait que comme on utilise ici une fonction de valeur basée sur une paire état-action et non juste un état, la formule de mise à jour de la fonction de valeur :

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Devient donc :

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

$R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ est la valeur cible pour $Q(S_t, A_t)$ vers laquelle on se rapproche avec un pas de α.

On peut effectivement constater ici que l'on utilise la politique gloutonne (ou optimale) pour la mise à jour, en utilisant $\gamma \max_a Q(S_{t+1}, a)$, on ne considère que la

meilleure paire état action de l'état suivant.

On parle d'algorithme sur politique quand les politiques d'inférence et de formation sont les mêmes.

2.2 Deep Q-Learning

Nous avons vu comment résoudre des problèmes d'apprentissage par renforcement grâce au Q-learning. Cet algorithme a cependant certaines limites, notamment le fait que l'espace d'état doit être discret et petit. Cela pose un problème quand on a un espace d'état infini ou très grand, une table gigantesque devient alors bien moins pertinente.

C'est alors qu'on passe de l'apprentissage par renforcement à l'apprentissage par renforcement profond. Au lieu d'utiliser une table Q, Deep Q-Learning utilise un réseau de neurones, qui prendra en paramètre un état et retournera une valeur par action. L'idée est d'estimer ce qu'aurait indiqué la table Q si on l'avait construite.

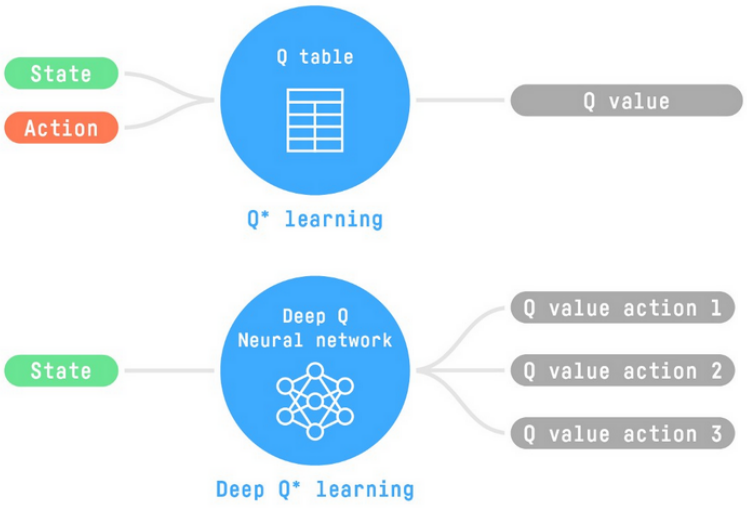


FIGURE 6 – Différence entre Q Learning et Deep Q Learning

Le Deep Q Learning permet alors de traiter des problèmes ayant des espaces d'état arbitrairement grands. Du fait que cette méthode consiste toujours à simuler la table Q, elle retourne pour un état donné toujours une valeur par action possible, et par conséquent n'est cependant pas capable de traiter des problèmes avec de grands espaces d'action.

L'algorithme est défini comme suit :

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

FIGURE 7 – Algorithme Deep Q-Learning

Si on met à part les changements de notation, cet algorithme est très similaire à celui que nous avons vu pour le Q-learning à quelques différences près. On peut dans un premier temps constater que la mise à jour avec un pas α d'un élément du tableau Q vers une valeur cible a été remplacée par un pas de descente de gradient d'une fonction de perte, définie comme l'écart entre notre fonction Q et la fonction Q-cible. Nous verrons par la suite l'utilité de cette fonction Q-cible.

Cet algorithme comporte deux phases, une phase d'échantillonnage où on effectue des actions et où on stocke les tuples d'expérience observés, et une phase d'entraînement où on sélectionne un petit lot de tuples au hasard et où on s'en sert pour faire une étape de mise à jour de descente de gradient.

L'entraînement du Deep Q-learning peut souffrir d'instabilité, différentes solutions sont mises en place pour le stabiliser, telles que l'experience replay, la correction de l'entraînement avec un nouveau réseau Q-cible, ainsi qu'un réseau double Q.

2.2.1 Correction 1 : Experience replay

Jusqu'ici, quand l'agent interagissait avec l'environnement, on obtenait un tuple d'expérience (état, action, récompense, état suivant), qu'on utilisait pour mettre à jour Q, puisqu'on rejetait, ce qui n'est pas efficace. De plus, cela fait que l'agent a tendance à oublier les expériences précédentes à mesure qu'il en acquiert de nouvelles. Un agent qui s'entraîne au niveau 1 d'un jeu puis passe au niveau 2 oubliera progressivement à jouer au niveau 1 à mesure qu'il apprend le niveau 2.

La solution consiste à créer un tampon de relecture qui stocke les tuples d'expériences tout en interagissant avec l'environnement, puis à échantillonner un petit lot de tuples pour l'entraînement. Cela empêche le réseau de neurones d'apprendre uniquement ce qu'il a fait immédiatement auparavant. Ce tampon est noté D dans l'algorithme.

2.2.2 Correction 2 : Correction de Q-cible

Pour calculer l'erreur, on calcule la différence entre la valeur cible et la valeur Q actuelle. Comme nous l'avons vu auparavant, comme nous n'avons aucune idée de l'objectif réel, nous utilisons l'équation de Bellman pour estimer cette valeur cible :

$$y = R_{t+1} + \gamma \underset{a}{max} Q(S_{t+1}, a)$$

On définit la fonction de perte comme :

$$y - Q(S_t, A_t)$$

Cependant, on voit ici qu'on utilise les mêmes paramètres (poids) pour estimer la valeur cible y et la nouvelle valeur de Q, qui est calculée en utilisant la fonction de perte. En effet, la fonction Q apparaît dans les deux expressions. Il existe donc une forte corrélation entre les deux valeurs, à chaque étape de la formation, les valeurs de Q et les valeurs cibles changent, on se rapproche de l'objectif, mais ce dernier change également.

Pour pallier cela, on utilise un réseau distinct $\hat{Q}$ avec des paramètres fixes pour estimer la cible, réseau qu'on actualise occasionnellement toutes les étapes C avec les paramètres de Q. Cette modification a tendance à fortement stabiliser l'entraînement.

2.2.3 Correction 3 : Double DQN (Deep Q-Network)

Cette méthode vise à résoudre le problème de la surestimation des valeurs de Q. Rappelons que la cible est : $R_{t+1} + \gamma \underset{a}{max} Q(S_{t+1}, a)$. Comment être sûr que la meilleure action pour l'état suivant est l'action avec la valeur Q la plus élevée ? L'exactitude des valeurs Q dépend des actions que nous avons tentées et des voisins que nous avons explorés. Au début de la formation, on ne dispose pas de suffisamment d'informations sur la meilleure action à entreprendre. Prendre la valeur Q maximale comme meilleure action peut mener à des faux positifs. Si les actions non optimales reçoivent régulièrement une valeur Q supérieure à la réelle meilleure action optimale, l'apprentissage sera compliqué.

La solution consiste à utiliser deux réseaux de neurones différents afin de découpler la sélection de l'action et la génération de la cible. On utilise alors :

- Le réseau Q pour sélectionner la meilleure action à entreprendre pour l'état suivant (l'action avec la valeur Q la plus élevée)
- Le réseau cible $\hat{Q}$ pour calculer la valeur cible de cette action à l'état suivant

Cette solution a déjà été mise en place par la solution au problème précédent. Cela aide à réduire la surestimation des valeurs Q, ce qui aide à un entraînement plus rapide avec un apprentissage plus stable.

2.2.4 Exemple de possibilité du Deep Q Learning

Un exemple de possibilité que nous offre le Deep Q Learning est celle de pouvoir avoir des images comme observation pour l'agent. En effet, cela impliquera généralement un grand espace d'état, mais pas nécessairement un grand espace d'action. Prenons un exemple avec un space shooter :

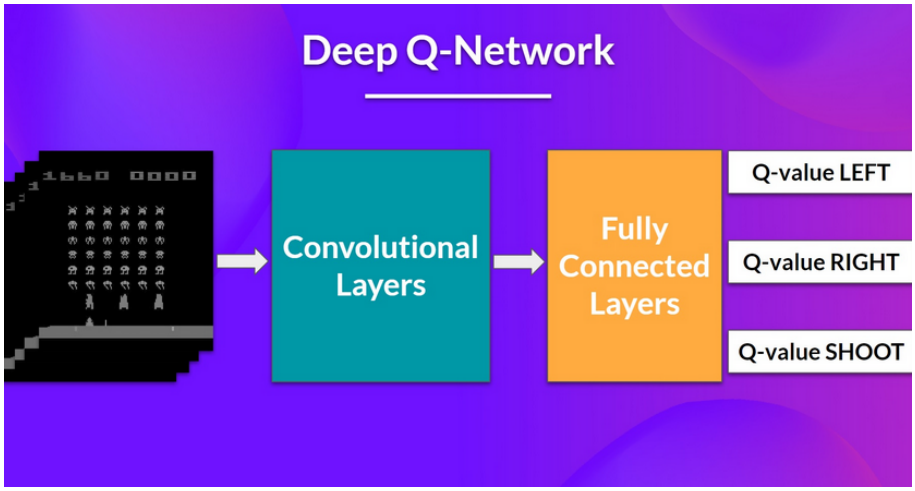


FIGURE 8 – Application du Deep Q learning aux images

Un space shooter contient généralement un nombre limité d'actions possibles. En général, on ne fournit pas à l'agent des images telles quelles du jeu. En effet, il est courant d'effectuer un recadrage de l'image pour enlever les parties ne contenant pas d'informations pertinentes, ainsi que de convertir l'image en niveaux de gris si la couleur n'apporte aucune information utile au jeu.

Un état est généralement représenté sous la forme de plusieurs trames et non une seule, permettant de retranscrire les informations temporelles que l'on perd en utilisant une seule trame (vitesse des objets, direction, etc).

2.3 Gradient de politique

Nous étudions les méthodes de résolution basées sur la politique, et non plus les méthodes basées sur la valeur. Nous allons donc optimiser directement une politique sans passer par une étape intermédiaire d'apprentissage d'une fonction de valeur.

Il est ici question d'un sous-ensemble des méthodes basées sur les politiques appelé gradient de politique. L'idée est de paramétrer la politique, en utilisant par exemple un réseau de neurones π_θ qui produira une distribution de probabilité sur les actions (politique stochastique) :

$$\pi_\theta(s) = P[A|s;\theta]$$

L'objectif est alors de maximiser les performances de la politique paramétrée en utilisant la montée de gradient. On contrôle ainsi θ ce qui affectera la répartition des actions sur un état.

On définit $J(\theta)$ comme la réponse cumulée attendue, notre objectif est de trouver la valeur de θ qui maximise cette fonction objectif.

2.3.1 Différence entre méthodes à gradient de politique et méthodes basées sur les politiques

La plupart des méthodes basées sur les politiques sont "sur politique" (et non "hors politique"), on utilise uniquement la politique de notre version la plus récente

de π_θ.

Dans les méthodes basées sur des politiques, on cherche directement la politique optimale, mais on peut optimiser le paramètre θ indirectement en maximisant l'approximation locale de la fonction objectif avec des techniques comme "hill climbing", "simulated annealing", ou des stratégies d'évolution.

Dans les méthodes à gradient de politique, on recherche également directement la politique optimale, mais on optimise θ directement en effectuant la montée de gradient sur $J(\theta)$.

2.3.2 Avantages des méthodes du gradient de politique

Les méthodes basées sur le gradient de politique possèdent de nombreux avantages. Tout d'abord, elles offrent généralement plus de simplicité d'intégration, on estime directement la politique sans stocker de données supplémentaires (valeurs d'actions). Ensuite, il y a également la possibilité d'apprendre une politique stochastique, avec notamment un espace d'action fini ou infini, ce qui est impossible avec les méthodes basées sur la valeur. Cela règle de nombreux problèmes comme celui de l'exploration-exploitation ainsi que celui de l'alias perpétuel (quand deux états semblent identiques, mais nécessitent deux actions différentes).
En plus de cela, les méthodes basées sur le gradient de politique offrent généralement de meilleures propriétés de convergence que les méthodes basées sur les valeurs. Pour ces dernières, nous mettions à jour la fonction de valeur en prenant le maximum des Q-estimation. L'opérateur "max" est un opérateur agressif faisant que les probabilités d'actions peuvent changer considérablement pour un changement arbitrairement faible des valeurs des actions. Les préférences stochastiques en matière de politique d'action, quant à elles, évoluent plus progressivement au fil du temps.

2.3.3 Inconvénients des méthodes du gradient de politique

Les méthodes basées sur le gradient de politique ont néanmoins quelques inconvénients. Il y a notamment le fait qu'on a généralement tendance à converger vers un maximum local et non global. Le gradient de politique se déroule également plus lentement, ce qui prend plus de temps à l'exécution. Pour finir, le gradient de politique présente généralement une grande variance, ce qui peut parfois poser un problème.

2.3.4 Principe de fonctionnement

L'objectif avec le gradient de politique est de contrôler la distribution de probabilité des actions en ajustant la politique pour que les actions qui maximisent le rendement soient échantillonnées plus fréquemment à l'avenir.

Pour cela, on va appliquer une stratégie Monte Carlo, qui, on le rappelle, consiste à laisser l'agent agir au cours d'un épisode entier avant de le mettre à jour. Si on "gagne" l'épisode, alors on considère chaque action entreprise comme "bonne" et devra être davantage échantillonnée dans le futur. Ainsi, en fonction de l'issue et pour chaque paire état-action, on augmente ou diminue la probabilité $P(a|s)$ d'entreprendre l'action a dans l'état s.

Voici à quoi ressemble l'algorithme simplifié du gradient de politique :

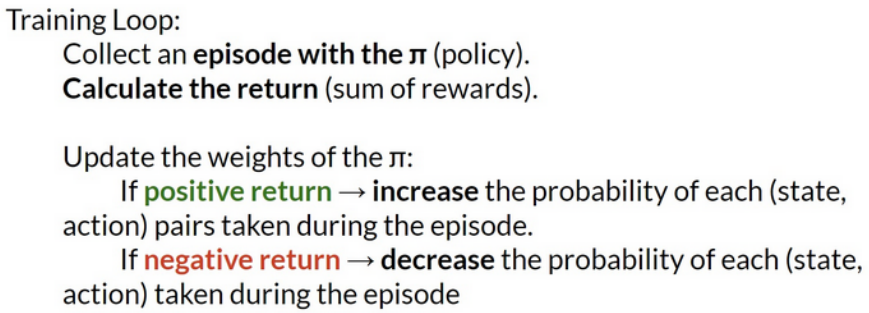


FIGURE 9 – Algorithme du gradient de politique

2.3.5 La fonction objectif $J(\theta)$

La fonction objectif nous donne la performance de l'agent compte tenu d'une trajectoire τ (état d'une séquence d'action sans tenir compte de la récompense, contrairement à un épisode), et produit la récompense cumulée attendue, définie comme la moyenne pondérée de toutes les valeurs possibles de $R(\tau)$:

$$J(\theta) = E_{\tau\sim\pi}[R(\tau)] = \sum_{\tau} P(\tau;\theta)R(\tau)$$

Avec :

- $R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ (rappel), le retour d'une trajectoire τ arbitraire.
- $P(\tau;\theta)$ la pondération de la récompense cumulée attendue $R(\tau)$, correspondant à la probabilité d'emprunter la trajectoire τ, probabilité découlant directement de θ car il définit la politique qui sélectionne les actions de la trajectoire.

$P(\tau;\theta)$ se définit comme :

$$P(\tau;\theta) = [\prod_{t=0} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)]$$

Avec :

- $P(s_{t+1}|s_t, a_t)$ la distribution des états
- $\pi_\theta(a_t|s_t)$ La probabilité de choisir a_t dans l'état s_t

Notre objectif est de trouver la valeur de θ qui maximise $J(\theta)$ ce qui, formellement, s'écrit ainsi :

$$\max_{\theta} J(\theta) = E_{\tau\sim\pi_\theta}[R(\tau)]$$

2.3.6 Montée de gradient et théorème du gradient de politique

Pour optimiser θ, on utilise la montée de gradient sur $J(\theta)$. Une étape de mise à jour peut ainsi être écrite comme :

$$\theta \leftarrow \theta + \alpha * \nabla_\theta J(\theta)$$

On ne peut cependant pas calculer le gradient réel, car cela nécessite de calculer la probabilité de chaque trajectoire possible, ce qui serait très coûteux en calcul, on doit donc estimer ce gradient à partir d'un échantillon. De plus, pour différencier $J(\theta)$, on doit différencier la distribution des états, ce qu'on ne peut pas faire, parce qu'on ne la connait pas toujours.

On utilise alors le théorème du gradient de politique (policy gradient theorem) pour reformuler la fonction objectif en une fonction différenciable qui n'implique pas la différenciation de la distribution des états. Ce théorème dit que :

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log(\pi_\theta(a_t|s_t))R(\tau)]$$

On peut alors estimer le gradient comme suit :

$$\nabla_\theta J(\theta) \approx \hat{g} = \sum_{t=0} \nabla_\theta log(\pi_\theta(a_t|s_t))R(\tau)$$

Avec :

- $\nabla_\theta log(\pi_\theta(a_t|s_t))$ la direction de l'augmentation la plus forte de la log-probabilité de sélectionner l'action a_t à partir de l'état s_t. Cela nous dit comment on devrait changer les poids de la politique si on veut augmenter ou diminuer cette log-probabilité.
- $R(\tau)$ le rendement, qui augmentera la probabilité de la combinaison (s_t, a_t) s'il est haut, la baissera sinon.

Cette estimation ne prend qu'une seule trajectoire pour estimer le gradient, mais on peut également en considérer plusieurs en faisant la moyenne des estimations :

$$\nabla_\theta J(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0} \nabla_\theta log(\pi_\theta(a_t^{(i)}|s_t^{(i)}))R(\tau^{(i)})$$

Vous pouvez consulter ici le détail du calcul pour le passage de la formule du théorème du gradient de politique aux estimations que nous venons de faire.

L'algorithme par renforcement Monte Carlo consiste alors simplement à mettre à jour θ avec l'estimation $\hat{g}$, pondérée par le pas d'apprentissage α :

$$\theta \leftarrow \theta + \alpha\hat{g}$$

2.3.7 PPO : Optimisation de la politique de proximité

Nous savons empiriquement que de petites mises à jour de politiques pendant la formation sont plus susceptibles de converger vers une solution optimale. De plus, une mise à jour trop importante dans une étape de mise à jour de politique peut entraîner une "chute du précipice", un grand pas dans une mauvaise direction qui mettra un temps long pour se rétablir, voire un non-rétablissement.

Ainsi, avec la PPO, nous regardons à quel point la nouvelle politique s'éloigne de l'ancienne, en calculant un ratio. Si ce ratio n'est pas dans la place $[1 - \epsilon, 1 + \epsilon]$, alors nous ignorons la mise à jour pour éviter de trop nous éloigner de l'ancienne.

2.4 Acteur Critique

L'algorithme par renforcement Monte Carlo vu dans la section précédente possède certains défauts, notamment celui d'avoir une grande variance de rendement. Or, l'estimation du gradient de politique correspond à l'augmentation la plus forte de rendement, une grande variance est alors problématique et il faut prendre en compte un grand nombre de trajectoires pour le calcul du gradient, ce qui mène à un apprentissage plus long.

C'est alors qu'interviennent les méthodes Acteur Critique, une architecture hybride entre méthodes basées sur les valeurs et méthodes basées sur les politiques qui réduisent la variance et stabilisent la formation en utilisant :

— Un acteur qui contrôle le comportement de l'agent (méthode basée sur Politique)
— Un critique qui mesure la qualité de l'action entreprise (méthode basée sur Valeur)

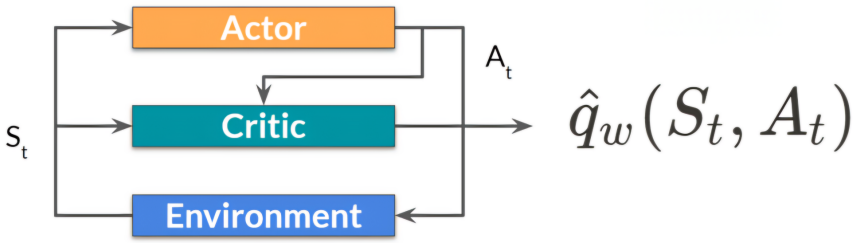
Nous étudierons ici un algorithme utilisant cette architecture hybride, Advantage Actor Critic (A2C). Pour cette méthode, il y a donc deux fonctions à apprendre :

— La politique $\pi_\theta(s)$ qui contrôle les actions de l'agent, "l'acteur", paramétrée par θ
— La fonction de valeur $\hat{q}_w(s, a)$, "le critique", paramétrée par w

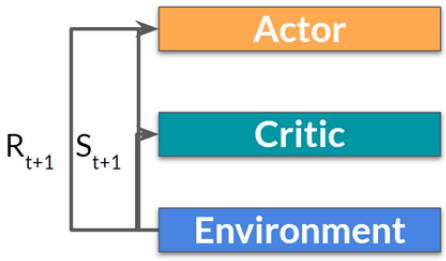
2.4.1 Processus d'entraînement

Voici alors le processus acteur critique :

1. À chaque pas de temps t, on obtient l'état actuel S_t de l'environnement, et on le fournit en entrée à l'acteur et au critique.
2. L'acteur produit une action A_t
3. Le critique calcule $\hat{q}_w(S_t, A_t)$ la valeur de l'action A_t dans l'état S_t



4. L'action A_t effectuée dans l'environnement génère un nouvel état S_{t+1} et une récompense R_{t+1}.



5. L'acteur met à jour ses paramètres de sa fonction de politique en utilisant la valeur Q :
$$\Delta\theta = \alpha\nabla_\theta(log(\pi_\theta(s,a))\hat{q}_w(s,a)$$
6. L'acteur produit la prochaine action à entreprendre A_{t+1} étant donné le nouvel état S_{t+1}.
7. Le critique met à jour ses paramètres de sa fonction de valeur avec A_t, A_{t+1}, S_t, et S_{t+1} :
$$\Delta w = \beta(R(s,a) + \gamma\hat{q}_w(s_{t+1},a_{t+1}) - \hat{q}_w(s_t,a_t))\nabla_w\hat{q}_w(s_t,a_t)$$
Avec :
— $R(s,a) = R_{t+1}$
— β le taux d'apprentissage, ce dernier n'a pas forcément à être le même que pour celui de la fonction de politique

2.4.2 Introduction de la fonction Advantage

Il est possible de stabiliser encore plus l'entrainement en remplaçant la fonction de valeur d'action comme critique par la fonction avantage. L'idée de la fonction avantage est de calculer l'avantage relatif d'une action par rapport aux autres possibles dans le même état. Cette fonction est définie comme suit :

$$A(s,a) = Q(s,a) - V(s)$$

Avec :
— $Q(s,a)$ la valeur de choisir l'action a dans l'état q
— $V(s)$ la valeur moyenne de l'état s

On calcule ainsi la récompense supplémentaire que l'on obtient si on effectue cette action par rapport à la valeur moyenne de l'état, cette récompense supplémentaire correspond à ce qui dépasse de la valeur attendue pour cet état.

— Si $A(s,a) > 0$, la récompense est supérieure à celle attendue, alors le gradient est poussé dans cette direction
— Si $A(s,a) < 0$, la récompense est inférieure à celle attendue, le gradient est alors poussé dans la direction inverse

Cette fonction avantage nécessite cependant deux fonctions de valeurs pour être calculée, $Q(s,a)$ et $V(s)$. On utilise alors l'erreur par différence temporelle pour estimer $Q(s,a)$:

$$Q(s,a) \approx r + \gamma V(s')$$

Avec $V(s')$ la valeur du nouvel état s' après avoir effectué l'action a depuis l'état s. La fonction avantage devient donc :

$$A(s,a) = r + \gamma V(s') - V(s)$$

2.5 Online / Offline RL

Dans le cadre de l'apprentissage par renforcement, il existe deux types d'agents : les agents en ligne et les agents hors ligne.

Un agent en ligne apprend en interagissant directement avec l'environnement. Il sélectionne une action à partir de sa politique actuelle, observe l'état suivant et la récompense associée, puis met à jour sa politique en conséquence. L'apprentissage en ligne est souvent utilisé dans les situations dans lesquelles l'environnement est dynamique et change constamment, car l'agent peut s'adapter rapidement aux nouveaux états et récompenses. Cependant, cela peut également rendre l'apprentissage plus instable, car l'agent peut être influencé par des récompenses aléatoires ou des états inhabituels.

Un agent hors ligne, en revanche, apprend à partir d'un ensemble de données préexistantes. Cet ensemble de données peut être constitué de trajectoires d'états, d'actions et de récompenses précédemment observées par l'agent ou par d'autres agents. L'agent hors ligne utilise alors ces données pour apprendre une politique optimale, sans interagir directement avec l'environnement. L'apprentissage hors ligne est souvent utilisé dans les situations où l'interaction avec l'environnement est coûteuse ou dangereuse, ou lorsque l'on dispose de grandes quantités de données préexistantes. Cependant, l'agent hors ligne peut également être limité par la qualité et la pertinence des données d'apprentissage.

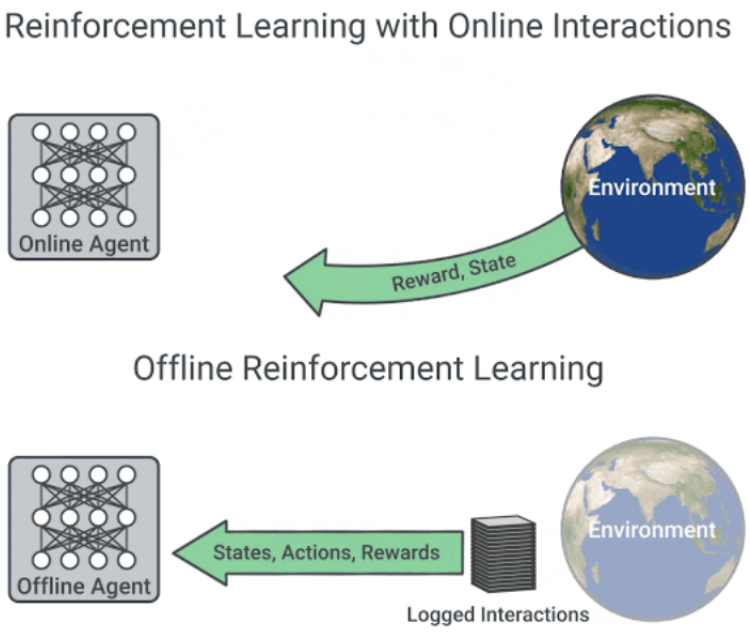


FIGURE 10 – Online and Offline Reinforcement learning

3 Environnement MARL : multi-agent

3.1 Introduction

Dans cette section, nous allons étudier les systèmes d'apprentissage par renforcement multi-agents (MARL), dans lesquels plusieurs agents interagissent dans un même environnement.

Il existe plusieurs types d'environnements multi-agents :
- Environnements coopératifs : les agents doivent maximiser les avantages communs (ex : entrepôt avec des robots de transports autonomes)
- Environnements compétitifs / adversaires : les agents cherchent à maximiser ses bénéfices en minimisant ceux de son adversaire (ex : jeu de tennis)
- Mélange antagoniste et coopératif (ex : jeu de foot, les agents coopèrent avec les autres agents de leur équipe, et minimisent les bénéfices de l'équipe adverse)

Dans un environnement multi-agents, il existe deux principales façons de gérer l'interaction entre les agents : les architectures décentralisées et les architectures centralisées.

3.2 Système décentralisé

Dans un système décentralisé, les agents interagissent avec l'environnement indépendamment les uns des autres, sans se soucier des actions des autres agents. L'avantage d'un tel système est sa simplicité. En effet, comme les agents ne partagent aucune information, ces derniers peuvent être entraînés comme des agents individuels, ce qui simplifie leur intégration.

Le gros point faible d'un système décentralisé est la non-coopération des agents. En effet, les agents considèrent ici les actions des autres agents comme faisant partie de la dynamique de l'environnement. Sauf qu'un agent suppose également qu'il est le seul à modifier son environnement (environnement stationnaire), ce qui n'est ici pas vrai, car une partie de l'environnement (les autres agents) évolue. Chaque agent essaye donc de s'adapter au comportement des autres agents en modifiant le sien, et comme le comportement de chaque agent change en permanence, on peut arriver à des situations de non-convergence, ou encore où on n'atteint pas d'optimal global.

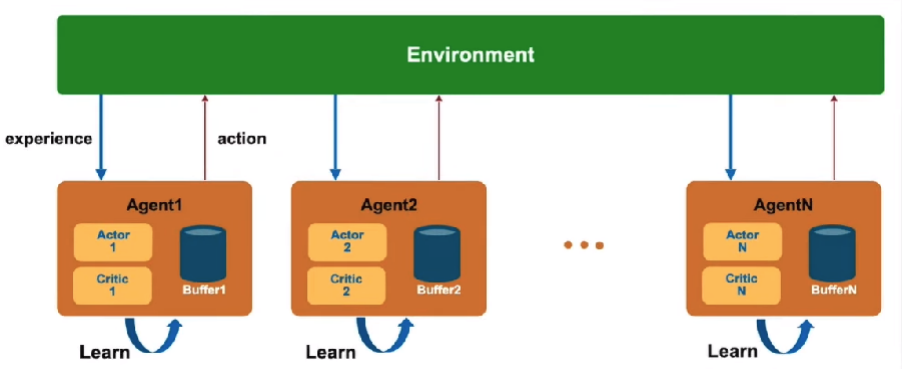


FIGURE 11 – Illustration d'un système décentralisé

3.3 Système centralisé

Dans un système centralisé, nous avons un processus de haut niveau qui collecte les expériences de chaque agent : le tampon d'expérience. Ces expériences sont utilisées pour apprendre une politique commune. Le gros inconvénient de cette architecture est alors le gain fort en complexité, qui s'accompagne cependant de la résolution du problème concernant les systèmes décentralisés.

L'expérience commune issue du tampon d'expérience suite à la collecte de chaque expérience individuelle pourra par exemple être la position de chaque agent. Cette expérience collective est alors utilisée pour former une politique commune qui fera agir les agents de la manière la plus bénéfique dans son ensemble (les agents reçoivent tous la même récompense). Chaque agent apprendra ainsi de l'expérience commune.

Cela revient ici à revenir à un environnement stationnaire, car tous les agents sont traités comme une entité plus grande, et tous les agents connaissent la politique des autres agents, car c'est la même que la leur.

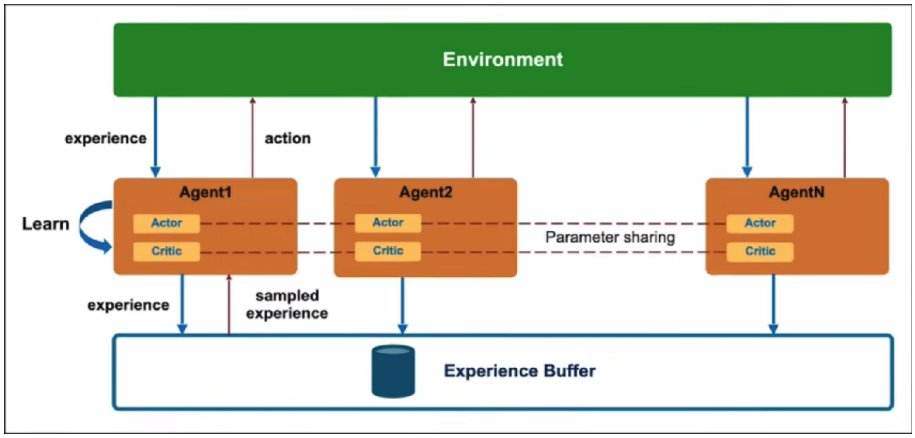


FIGURE 12 – Illustration d'un système centralisé

Cependant, la représentation centralisée peut également présenter des défis. Tout d'abord, elle nécessite une communication et une coordination importantes entre les agents et le super-agent, ce qui peut être difficile à mettre en œuvre dans des systèmes complexes. De plus, elle peut être vulnérable aux défaillances du super-agent, car si le super-agent tombe en panne, tous les agents peuvent être affectés. Enfin, elle peut ne pas être évolutive, car il peut être difficile d'ajouter de nouveaux agents au système sans modifier la conception du super-agent.

3.4 Détail sur les environnements compétitifs

Comme nous l'avons évoqué, dans les environnements compétitifs, les agents cherchent à maximiser leurs propres récompenses en minimisant celles de leurs adversaires. Les exemples classiques de ce type d'environnement sont les jeux à deux joueurs, tels que le tennis, les échecs ou le go.

Pour un entraînement pour ce genre d'environnement, former des agents adversaire peut être assez complexe. Si l'adversaire est trop bien entraîné, l'agent risque

de ne jamais arriver dans des situations pour lesquelles il peut faire des bonnes actions, et donc n'apprendra jamais. Si l'adversaire est faible, l'agent n'apprendra pas comment battre un adversaire fort. La solution consiste à avoir un adversaire au même niveau que l'agent, et qui améliorera son niveau au fur et à mesure que l'agent améliore le sien.

Pour cela, on utilise le jeu autonome ou self play, l'agent utilise d'anciennes copies de lui-même comme adversaire.

Pour ce genre de problème (les jeux antagonistes), la récompense cumulée n'est pas toujours la meilleure métrique pour mesurer la progression de l'apprentissage, car elle dépend grandement de l'adversaire. On peut alors utiliser le score ELO, qui calcule le niveau de compétence relatif entre deux joueurs d'une population donnée dans un jeu à somme nulle.

À la fin de la partie :
— Si le joueur au plus haut ELO gagne, il prend quelques points à l'autre joueur
— Si le joueur au plus bas ELO gagne, il prend beaucoup de points à l'autre joueur
— Si égalité, le joueur au plus bas ELO prend quelques points à l'autre joueur

Les résultats attendus, entre 0 et 1 (probabilité d'un joueur de gagner), sont définis pour le joueur A et le joueur B comme suit :

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}}$$

$$E_B = \frac{1}{1 + 10^{(R_A - R_B)/400}}$$

On définit également un taux d'ajustement maximal K :
— $K = 16$ pour les bons joueurs
— $K = 32$ pour les plus faibles

Si le joueur A marque S_A points (1 si il gagne 0 si il perd), on met ainsi à jour son classement :

$$R'_A = R_A + K(S_A - E_A)$$

4 Mise en pratique

Nous avons choisi de réaliser un projet consistant à implémenter un algorithme de Q-Learning entièrement sans aucune librairie. L'objectif est de tout faire manuellement, autant l'environnement que l'algorithme, afin de nous assurer de notre bonne compréhension des notions apprises.

4.1 Environnement

Afin de rendre le projet ludique, nous avons choisi de nous inspirer d'un jeu simple à comprendre, à savoir Helltaker. Nous avons légèrement adapté les règles à notre problème. Ainsi, les environnements auxquels l'agent est exposé sont des grilles sur lequel ce dernier peut se déplacer. Chaque case de la grille peut contenir un mur, de la lave, un mur cassable, une clé, un coffre, du vide ou l'agent lui-même.

4.2 Agent

L'agent est représenté par une case bleue dans l'environnement. Ses actions possibles sont : {haut, bas, gauche, droite}. Si l'agent essaye d'aller dans une direction et qu'il y a du vide ou une clé dans cette direction, cela sera interprété comme un déplacement. Si l'agent essaye d'aller dans une direction et qu'il y a un mur cassable dans cette direction, alors cela sera interprété comme le fait de casser le mur. Dans toute autre situation, l'action sera rejetée.

4.3 Fonctionnement

L'objectif de l'agent est de récupérer la clé en atteignant la case sur laquelle cette dernière se trouve, puis d'atteindre le coffre, le tout en prenant le moins de pénalités possibles. Les pénalités / récompenses sont définies comme suit :

- Essayer d'avancer contre un mur : -10
- Récupérer la clé : 100
- Essayer d'avancer contre le coffre sans la clé : -10
- Atteindre le coffre avec la clé : 100
- Aller dans la lave : -100
- Essayer de sortir de la carte : -10
- Effectuer une action : -1
- Casser un mur : -5 ou -20

Un épisode se termine soit par l'agent qui va dans la lave, soit l'agent qui atteint le coffre avec la clé, soit l'agent qui fait 250 actions sans finir par une des deux issues précédentes.

Mettre une pénalité de -1 à chaque action permet de pousser l'agent à terminer le niveau avec le moins d'actions possibles, car résoudre le niveau avec plus d'actions réduit la récompense cumulée.

4.4 Résultats

Nous avons créé différentes cartes que nous avons toutes fait tester à l'agent. L'agent arrive alors toujours à trouver la série d'actions optimale lorsqu'il a été suffisamment entraîné. Nous avons également pu tester l'influence des valeurs des

pénalités, en testant l'agent sur une carte ou l'agent a le choix entre emprunter un chemin nécessitant de casser beaucoup de murs, mais assez court, ou un chemin qui ne demande de casser aucun mur, mais plus long. En fonction de la valeur de la pénalité liée au fait de casser un mur (-5 ou -20), l'agent ne fait pas le même choix, car la valeur de la pénalité définit directement quel chemin est le plus optimal à prendre au vu de la récompense cumulée.

Conclusion

Dans ce rapport, nous avons exploré en profondeur les différentes facettes de l'apprentissage par renforcement (RL), un domaine crucial de l'intelligence artificielle. Nous avons abordé les concepts fondamentaux, les méthodes, et les algorithmes spécifiques qui permettent à un agent de prendre des décisions optimales dans un environnement donné.

L'apprentissage par renforcement se distingue par sa capacité à apprendre à partir des interactions avec l'environnement, en recevant des récompenses ou des pénalités qui guident l'agent vers des comportements optimaux. Nous avons expliqué en détail le cycle de fonctionnement du RL, les différentes notions fondamentales dans ce domaine, ainsi que les divers défis rencontrés.

Nous avons ensuite vu différentes stratégies d'implémentations de ces concepts, telles que le Q-Learning, le Deep Q-Learning, les méthodes à gradient de politique ainsi que l'approche acteur-critique. Ces méthodes peuvent être appliquées dans des contextes multi-agents (MARL), où la coordination et la compétition entre agents ajoutent des niveaux de complexité supplémentaires.

Dans la section pratique, nous avons développé un environnement de test et formé un agent à naviguer efficacement en utilisant différentes stratégies et paramètres de pénalités. Les résultats montrent que l'agent peut apprendre à choisir des chemins optimaux en fonction des pénalités associées à certaines actions, démontrant ainsi l'efficacité de notre implémentation.

En conclusion, ce rapport met en lumière l'importance et l'efficacité de l'apprentissage par renforcement dans le développement d'agents intelligents capables de prendre des décisions autonomes et optimales. Les techniques et approches présentées offrent une base solide pour des applications futures dans divers domaines, de la robotique à la gestion des ressources, en passant par les jeux et les systèmes autonomes.

Références

[1] R. S. Sutton and A. G. Barto. *Reinforcement Learning - An Introduction - second edition*. MIT press, 19-10-2018.

[2] T. Simonini and O. Sanseviero. The hugging face deep reinforcement learning class. https://github.com/huggingface/deep-rl-class, 2023.