Apprentissage par renforcement

Rapport écrit par
Pierre-André Mikem, Nawel Arab et Ramy Merabet

Présentation d'un article de recherche

Institut Polytechnique de Paris

Rapport rendu à Erwan Le Pennec
Date de remise : 01/03/2021

Table des matières

1 Introduction

L'article que nous nous proposons de vous présenter dans ce rapport est le suivant : **Deep Reinforcement Learning for Multi-Agent Systems : A Review of Challenges, Solutions and Applications**. Dans cet article, les auteurs proposent un aperçu de l'historique de l'apprentissage par renforcement, en posent le cadre et les bases théoriques, expliquent l'utilisation de l'apprentissage profond en apprentissage par renforcement, introduisent le cadre Multi-Agents et listent les principales problématiques, solutions et applications associées.

Le parti-pris dans notre présentation est le suivant : dans un premier temps, nous nous proposons de reprendre *linéairement* l'article proposé, en vous en exposant notre compréhension, et en détaillant, notamment au niveau mathématique, certains points rapidement survolés par les auteurs. Nous avons ensuite choisi de centrer notre attention sur une problématique apparaissant dans le cadre Multi-Agents : la non-stationnarité intrinsèque aux intéractions entre les différents agents. Cette problématique sera détaillée plus en profondeur, notamment grâce à l'étude des références laissées par les auteurs en fin d'article.

Notre présentation suivra donc le plan suivant :

— Préliminaires sur l'apprentissage par renforcement Mono-Agent
— Apprentissage par renforcement profond
— Cadre Multi-Agents et non-stationnarité

Enfin, le présent rapport sera accompagné d'un Notebook dans lequel nous vous proposons une petite implémentation d'un jeu collaboratif illustrant notre travail, et notamment une autre problématique mentionnée dans l'article : celle de l'observabilité partielle.

2 Préliminaires sur l'apprentissage par renforcement Mono-Agent

2.1 Un domaine de recherche à part

Les algorithmes d'apprentissage par renforcement sont utilisés depuis de nombreuses années maintenant pour résoudre des problèmes de décision séquentiels, mais deviennent difficilement exploitables lorsque l'environnement dans lequel évolue l'agent est de grande dimension. L'utilisation du Deep Learning permet depuis quelques années de faire face à ce problème. Cet article s'intéresse à la situation dans laquelle non pas un mais de multiples agents doivent coopérer pour résoudre une tâche complexe dans un espace de grande dimension : c'est le cadre du Multi-Agent Deep Reinforcement Learning (MADRL).

Il nous a paru important, dans un premier temps, d'expliquer en quoi l'apprentissage par renforcement se distingue des autres domaines de l'intelligence artificielle.

L'apprentissage par renforcement est à distinguer de l'apprentissage supervisé : l'apprentissage par renforcement consiste en l'apprentissage d'un ou plusieurs agents par intéraction (du type essai/erreur) avec son ou leur environnement tandis que l'apprentissage supervisé consiste en l'apprentissage à partir d'une base de données labellisées (i.e. définies par des entrées et les sorties correspondantes) fournie par un intervenant extérieur.

L'apprentissage par renforcement est aussi à distinguer de l'apprentissage non-supervisé : prenons l'exemple du clustering. On suppose que l'on dispose d'une base de données d'apprentissage partitionnée en classes caractérisées par un label et pour laquelle les labels de sorties ne sont pas disponibles. Dans ce cadre, l'apprentissage non-supervisé consiste à explorer la structure cachée des données pour essayer de rendre compte du partitionnement intrinsèque aux données ; le critère que l'on optimise n'est pas directement lié à l'erreur de partitionnement mais rend plutôt compte de l'*homogénéité* au sein des différents clusters. A contrario, l'apprentissage par renforcement est un apprentissage "goal-directed" dans le sens où le modèle considéré définit clairement un critère, rendant compte de l'erreur effectuée (ou plutôt du gain perçu), à optimiser.

Enfin, l'apprentissage par renforcement profond est à distinguer de l'apprentissage profond : dans le second cas, le réseau de neurones apprend différents niveaux d'abstraction du problème (par exemple, dans un CNN, chaque couche cachée extrait des features de plus en plus complexes dans l'image fournie en entrée) tandis que dans le premier, un réseau de neurones n'est utilisé que comme un outil d'approximation permettant l'apprentissage en grande dimension.

Toutefois, étant donnée la grande difficulté des problèmes réels, dans de nombreuses situations, un agent unique, entrainé par un algorithme d'apprentissage par renforcement profond, ne fournit souvent pas des résultats très satisfaisants. Nous sommes alors amenés à considérer des systèmes Multi-Agents. Dans ce cadre, on considère plusieurs agents et ceux-ci sont mis en compétition ou en collaboration pour obtenir de meilleurs résultats.

L'introduction de systèmes Multi-Agents nécessite alors un cadre spécifique et fait apparaître de nombreuses problématiques que nous considérerons dans la suite.

2.2 Le cadre de l'apprentissage par renforcement Mono-Agent

L'apprentissage par renforcement est, comme nous l'avons dit, une procédure d'apprentissage de type essai/erreur.

Dans le cadre de l'apprentissage par renforcement Mono-Agent, on définit deux acteurs fondamentaux : un agent, apprenant une tâche bien définie, et son environnement, fournissant à l'agent une récompense selon l'action qu'il effectue à un instant t du processus. Les intéractions agent/environnement à un instant t du processus sont décrites par l'intermédiaire de 3 éléments caractéristiques : un état s_t appartenant à un espace d'états S, une action a_t appartenant à un espace d'actions A, et une récompense r_t à valeurs réelles.

À un instant t du processus, l'agent examine l'état s_t de l'environnement, et choisit en conséquence une action a_t. Cette action a_t produit alors un effet sur l'environnement, qui modifie son état de s_t à s_{t+1} et fournit une récompense r_{t+1} à l'agent.

Ainsi, les intéractions agent/environnement peuvent être décrites par une séquence, ou trajectoire, d'états/actions/récompenses : $s_0, a_0, r_1, s_1, a_1, ..., r_n, s_n$. En pratique, on définit un état terminal $s_n := s_T$. Une séquence d'états/actions/récompenses de l'instant initial à l'état terminal est alors appelée un épisode.

On définit alors une politique, que l'on notera de manière générique π dans la suite, comme un moyen permettant à l'agent de choisir une action, étant donné un état. Une politique π déterministe est une application de l'espace d'états S dans l'espace d'actions A. Une politique π stochastique est un noyau de probabilités conditionnelles indexé par S : étant donné un état $s \in S$, $\pi(a|s)$ est la probabilité pour l'agent de choisir l'action a sachant que l'environnement est dans l'état s.

Remarque : En fait, une politique est toujours définie comme un noyau de probabilités conditionnelles indexé par S, et on parle de politique stochastique lorsqu'il existe un état s et une action a tels que $\pi(a|s) < 1$. Dans la suite, dans le cas d'une politique déterministe, on ne fera pas de distinction entre application au sens classique et noyau de probabilités conditionnelles de support de cardinal 1.

Les politiques déterministes sont désirables dans le sens où leur comportement est prédictible, ce qui est un facteur crucial pour produire des algorithmes performants. En pratique, on peut définir une politique déterministe π_d à partir d'une politique stochastique π_s de la manière suivante : étant donné un état s, on associe à s l'action la plus probable pour la politique π_s, i.e. $\pi_d(s) = a_{i_0}$, où $i_0 = argmax_i \pi_s(a_i|s)$.

Ainsi, pour chaque $s \in S$, $\pi(s)$ est déterminé par l'ensemble : $\Psi(s) = \{p\,(a_i \mid s) \mid a_i \in \Delta_\pi\}$, où Δ_π représente l'ensemble des actions candidates pour la politique π.

Dans la suite, on fera les hypothèses suivantes, pour rester dans le cadre développé par les auteurs :

Hypothèses :

— On suppose dans la suite que l'espace d'actions, qui désigne maintenant l'ensemble des actions candidates pour la politique π, soit Δ_π, est discret. Cette hypothèse est pratique puisqu'elle simplifie les notations, mais peut être aisément relachée en remplaçant $\sum_a$ par $\int_A$ dans la suite.

— On suppose de plus que l'espace d'états est discret. Cette hypothèse est plus fondamentale et une problématique étudiée par les auteurs concerne justement l'extension des algorithmes au cas d'un espace d'états continu.

— On suppose enfin que (s_{t+1}, r_{t+1}) est entièrement déterminé par (s_t, a_t), et est donc indépendant des états et actions passés. Pour être plus précis au sens mathématique, on suppose que la loi de (s_{t+1}, r_{t+1}) est entièrement déterminée par (s_t, a_t). On se place alors dans le cadre des Processus de Décision Markoviens. Dans ce cadre, la dynamique est entièrement déterminée par les probabilités de transition $p(s', r'|s, a)$. La principale différence avec le cadre des Processus de Décision Markoviens "classiques" est que ces probabilités sont inconnues et sont donc à apprendre par l'agent, par l'intermédiaire d'un processus d'exploration de l'environnement.

L'apprentissage par renforcement se déroule alors selon le paradigme de l'apprentissage par essai/erreur, de la manière suivante :

— On attribue à l'agent une politique aléatoire π_0 (déterministe ou stochastique).

— L'agent "joue" selon la politique π_0 au sens où ses actions succesives sont déterminées par la politique π_0. Cette phase peut être vue comme une phase d'exploration dans laquelle l'agent "apprend" son environnement.

— L'agent améliore alors sa politique en prenant en compte les résultats de son exploration.

Cette procédure est appelée *policy improvement*. En l'appliquant jusqu'à obtention d'une politique optimale, on aboutit à une suite de politiques de qualité croissante : $\pi_0, \pi_1, ..., \pi_{t+1}, ..., \pi^*$.

Tout le jeu de l'apprentissage par renforcement consiste alors en la mise en place d'un **compromis exploration/exploitation** optimal : exploration pour apprendre l'environnement, exploitation pour utiliser les informations déjà acquises sur l'environnement pour améliorer la politique courante.

Cette procédure nécessite toutefois la définition d'un moyen d'évaluer la qualité d'une politique et de comparer des politiques entre elles. Cela fait l'objet de la section suivante.

2.3 Les équations de Bellman pour les fonctions valeur et q-valeur

On définit le retour à l'instant t par : $R_t = \sum_{i=0}^{T-t-1} \gamma^i r_{t+i+1}$, où $0 \leq \gamma < 1$ est un facteur d'atténuation permettant, mathématiquement, de garantir la convergence de la somme considérée lorsque $T = +\infty$.

Cette quantité modélise le "retour sur investissement" lorsqu'à l'instant t l'agent choisit l'action a_t. Le choix de γ permet de contrôler la longueur de la fenêtre temporelle prise en compte par l'agent pour l'optimisation de sa politique : lorsque γ est proche de 1, l'agent est "clairvoyant" au sens où il optimise sa politique en prenant en compte son gain en temps long. À l'inverse, lorsque γ est proche de 0, l'agent optimise sa politique en ne prenant en compte que son gain immédiat.

Remarque : Considérer le "retour sur investissement" plutôt que le "gain immédiat" est tout à fait naturel dans le sens où, dans le cadre des PDMs, le choix d'une action a à un instant t à une incidence sur r_{t+1}, mais aussi sur s_{t+1}.

On définit ensuite la fonction valeur permettant, étant donné un état initial s, d'évaluer la qualité d'une politique π partant de l'état s. Cette fonction est définie par la formule : $V_\pi(s) = \mathbb{E}[R_t \mid s_t = s, \pi]$. De même, on définit la fonction q-valeur permettant, étant donnés un état initial s et une action initiale a, d'évaluer la qualité d'une politique π partant de l'état s en choisissant l'action a à l'instant initial, ou encore la qualité du choix de l'action a dans l'état s. Cette fonction est définie par la formule : $Q_\pi(s, a) = \mathbb{E}[R_t \mid s_t = s, a_t = a, \pi]$.

Remarque : Dans les formules définissant les fonctions valeur et q-valeur, c'est l'hypothèse markovienne qui permet d'assurer l'indépendance en t.

On a alors les formules suivantes :

$$V_\pi(s) = \sum_a \pi(s, a) \sum_{s'} p(s' \mid s, a) \left(\mathbb{W}_{s \to s' | a} + \gamma V_\pi(s') \right), \text{et}$$

$$Q_\pi(s, a) = \sum_{s'} p(s' \mid s, a) \left(\mathbb{W}_{s \to s' | a} + \gamma V_\pi(s') \right),$$

où $\mathbb{W}_{s \to s' | a} = \mathbb{E}[r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s']$.

Ces formules s'obtiennent simplement en appliquant la formule des probabilités totales.

Plus précisément, on a :

$$\begin{aligned}
V_\pi(s) &\doteq \mathbb{E}_\pi\left[R_t \mid s_t = s\right] \\
&= \mathbb{E}_\pi\left[r_{t+1} + \gamma R_{t+1} \mid s_t = s\right] \\
&= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p\left(s', r \mid s, a\right)\left[r + \gamma \mathbb{E}_\pi\left[R_{t+1} \mid s_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a \mid s) \sum_{s', r} p\left(s', r \mid s, a\right)\left[r + \gamma V_\pi\left(s'\right)\right] \\
&= \sum_a \pi(s, a) \sum_{s'} p\left(s' \mid s, a\right)\left(W_{s \to s' \mid a} + \gamma V_\pi\left(s'\right)\right).
\end{aligned}$$

La deuxième formule se démontre de la même manière.

Ces équations se réécrivent de manière équivalente :

$$\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi\left[R_t \mid s_t = s\right] \\
&= \mathbb{E}_\pi\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \mid s_t = s\right] \\
&= \mathbb{E}_\pi\left[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + ...) \mid s_t = s\right] \\
&= \mathbb{E}_\pi\left[r_{t+1} + \gamma R_{t+1} \mid s_t = s\right] \\
&= \mathbb{E}_\pi\left[r_{t+1} + \gamma V_\pi(s_{t+1}) \mid s_t = s\right], \text{et}
\end{aligned}$$

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a\right]$$

.

Ces deux équations sont appelées équations de Bellman et sont largement utilisées dans le cadre de l'amélioration de politiques.

On peut en effet définir une relation d'ordre sur l'ensemble des politiques via la fonction valeur et la fonction q-valeur par :

$\pi \leq \pi' \iff [V_\pi(s) \leq V_{\pi'}(s) \forall s] \vee [Q_\pi(s,a) \leq Q_{\pi'}(s,a) \forall (s,a)]$.

On peut alors en déduire un algorithme glouton d'amélioration de politiques (déterministes) : il suffit d'explorer tous les couples (s,a), ou de manière plus naturelle, pour chaque état s, d'explorer toutes les actions possibles a étant donné l'état s, puis de sélectionner celle maximisant la fonction q-valeur.

Une autre façon de faire est de résoudre directement les équations d'optimalité de Bellman :

$V_{\pi^*}(s) = \max_a \sum_{s'} p\left(s' \mid s, a\right) \left(\mathbb{W}_{s \to s'|a} + \gamma V_{\pi^*}\left(s'\right)\right)$, et

$Q_{\pi^*}(s, a) = \sum_{s'} p\left(s' \mid s, a\right) \left(\mathbb{W}_{s \to s'|a} + \gamma \max_{a'} Q_{\pi^*}\left(s', a'\right)\right)$.

Ces équations sont des équations de point fixe qui peut être résolue par les techniques classiques d'analyse numérique (en pratique, on préfère résoudre l'équation de q-valeur car il est plus commode de travailler avec un opérateur du type $E[max...]$ qu'avec un opérateur du type $maxE[...]$).

Ces équations permettent de déterminer les fonctions valeur et q-valeur optimales. On peut alors leur associer une politique déterministe optimale avec la même règle que précédemment : à chaque état s, on associe l'action maximisant la fonction q-valeur.

Toutefois, la résolution des équations de Bellman nécessite la connaissance complète de l'environnement de l'agent, puisqu'elles font intervenir les probabilités de transition $p(s'|s,a)$. Or, dans le cadre de l'apprentissage par renforcement, la connaissance de l'environnement n'est possible qu'au moyen de l'intéraction avec ce dernier. De plus, même dans le cas où l'on se donne des hypothèses fortes sur l'environnement, l'explosion combinatoire rend impossible le traitement des environnements de grande dimension.

La suite de l'article développe l'étude des méthodes *Model-free*, i.e. des méthodes ne requérant aucune connaissance préalable sur l'environnement de l'agent.

2.4 Les méthodes *Model-free*

2.4.1 Les méthodes de Monte-Carlo

D'une manière très générale, les méthodes de Monte-Carlo mettent en application l'intuition contenue dans la loi des grands nombres qui est que l'on peut approcher une espérance, i.e. une intégrale calculée contre une certaine loi, à partir d'un grand nombre de réalisations d'une variable aléatoire suivant cette loi.

Dans le cadre de l'apprentissage par renforcement, l'application des méthodes de Monte-Carlo consiste ainsi en l'estimation des fonctions valeur et q-valeur par génération de nombreux épisodes et moyennage des retours observés.

Par exemple, l'estimateur de Monte-Carlo de $Q_{pi}(s,a)$ est donné par :

$Q_\pi^{MC}(s, a) = \frac{1}{n} \sum_{i=1}^{n} R^i(s,a)$, où n est le nombre d'épisodes générés et $R^i(s,a)$ est le retour observé au i-ème épisode en partant de l'état s et de l'action a (dans le cas de la variante *first visit* vue en cours).

Les méthodes de Monte-Carlo se basent ainsi uniquement sur la simulation du Processus de Décision Markovien sous-jacent et ne nécessite aucune hypothèse sur le modèle. Pour assurer la convergence des méthodes de Monte-Carlo, qui est en pratique quadratique, deux hypothèses fondamentales sont nécessaires :

— Le nombre d'épisodes générés est grand.
— Chaque état et chaque action doit avoir été visité un nombre significatif de fois, ce qui justifie notamment l'introduction de politiques stochastiques (cette hypothèse ne peut pas être vérifiée avec des politiques déterministes).

Pour garantir la propriété précédente d'"exploration" de l'environnement, on utilise en pratique la règle suivante d'amélioration de politiques : pour chaque état s, on donne une probabilité $1 - \epsilon + \frac{\epsilon}{|\Delta_\pi(s)|}$ à l'action maximisant la fonction de q-valeur, et une probabilité $\frac{\epsilon}{|\Delta_\pi(s)|}$ aux autres actions. Cette règle d'amélioration de politiques rend la politique stochastique, et permet de laisser la possibilité à l'agent de choisir une action autre que celle maximisant la fonction q-valeur, dans une optique d'exploration. En pratique, le paramètre ϵ est réduit au cours des itérations pour se rapprocher d'une politique déterministe.

Dans la littérature, on divise généralement les méthodes de Monte-Carlo en 2 sous-catégories : les méthodes *on-policy*, et les méthodes *off-policy*. Dans le premier cas, la même politique π est utilisée pour l'évaluation et l'exploration et est donc nécessairement stochastique comme vu dans la règle d'amélioration de politiques précédente. Dans le second cas en revanche, on utilise une politique π pour l'évaluation, et une politique π' pour l'exploration, ce qui permet notamment de faire en sorte que la politique π reste déterministe, et donc prédictible. Les méthodes *off-policy* sont en général plus complexes et plus puissantes que les méthodes *on-policy*. Toutefois, les méthodes *on-policy* présentent l'avantage d'être plus stables dans le cas d'un espace d'états continu et lorsqu'elles sont utilisées conjointement à un outil d'approximation de fonctions comme un réseau de neurones.

2.4.2 Les méthodes *Temporal-Difference*

Les méthodes *Temporal-Difference* sont une combinaison des méthodes de Monte-Carlo et des méthodes de programmation dynamique. De même que les méthodes de Monte-Carlo, les méthodes *Temporal-Difference* sont aussi des méthodes d'apprentissage par simulation c'est-à-dire apprenant à partir de l'expérience directe de l'agent, sans modèle de la dynamique de l'environnement. Mais contrairement aux méthodes de Monte-Carlo, elles ne nécessitent pas d'attendre la fin d'un épisode pour mettre à jour les estimées. De même que les méthodes de programmation dynamique, les méthodes *Temporal-Difference* utilise des estimées précédentes pour mettre à jour des estimées courantes. En cela, les méthodes *Temporal-Différence* sont des méthodes de *Bootstrapping*.

Les méthodes *Temporal-Difference* se fondent sur l'équation de Bellman : $V_\pi(s) = \mathbb{E}_\pi \left[r_{t+1} + \gamma V_\pi(s_{t+1}) \mid s_t = s \right]$, qui permet d'affirmer que, conditionnellement à $s_t = s$, $r_{t+1} + \gamma V_\pi(s_{t+1})$ est un estimateur non-biaisé de $V_\pi(s)$. C'est cette observation qui motive l'algorithme TD(0) utilisé pour l'estimation de la fonction valeur.

L'idée consiste alors à initialiser un tableau (d'où le nom de méthodes tabulaires) V arbitrairement, puis à le mettre à jour à chaque pas de temps via la formule :

$$V(s_t) \leftarrow V(s_t) + \alpha(\underbrace{r_{t+1} + \gamma V(s_{t+1}) - V(s_t)}_{\text{TD erreur}})$$

où $0 < \alpha < 1$ désigne le pas, ou taux d'apprentissage.

On distingue en général deux types de méthodes *Temporal-Difference* : les méthodes *on-policy TD control* connues sous le nom de Sarsa et *off-policy TD control* connues sous le nom de Q-learning.

2.4.3 Les méthodes d'apprentissage *on-policy*

Nous présentons dans cette partie une méthode *Temporal-Difference on-policy*, appelée Sarsa (pour State-Action-Reward-State-Action), basée sur l'algorithme TD(0) présenté précédemment.

L'algorithme Sarsa est le suivant :

Initialiser $Q(s,a)$ arbitrairement pour tout état s non-terminal, et $Q(s,a) = 0$ pour tout état s terminal ;
$i \leftarrow 1$;
tant que $i < N$, *avec* N *le nombre d'épisodes* **faire**
    Initialiser s ;
    Choisir a selon Q (déterministe ou ϵ-greedy) ;
    tant que s *non-terminal* **faire**
        Jouer l'action a ;
        Observer r et s' ;
        Choisir a' selon Q (déterministe ou ϵ-greedy) ;
        $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$;
        $s \leftarrow s'$;
        $a \leftarrow a'$;
    fin
fin

On parle de méthode *on-policy* : la politique apprise est la même que celle utilisée pour faire les choix des actions à exécuter (choix de a').

Nous avons vu en cours que cet algorithme converge dans le cas ϵ_t-greedy avec $\epsilon_t \to 0$.

2.4.4 Les méthodes d'apprentissage *off-policy*

La méthode *Q-learning* est quant à elle une méthode *off-policy* : la mise à jour de la fonction q-valeur est faite indépendamment de la politique suivie à l'étape courante.

L'algorithme *Q-learn* est le suivant :

Initialiser $Q(s,a)$ arbitrairement pour tout état s non-terminal, et $Q(s,a) = 0$ pour tout état s terminal ;
$i \leftarrow 1$;
tant que $i < N$, *avec* N *le nombre d'épisodes* **faire**
    Initialiser s ;
    Choisir a selon Q (déterministe ou ϵ-greedy) ;
    tant que s *non-terminal* **faire**
        Jouer l'action a ;
        Observer r et s' ;
        $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$;
        $s \leftarrow s'$;
        $a \leftarrow a'$;
    fin
fin

L'algorithme *Q-learn* converge si le taux d'apprentissage tend vers 0, mais pas trop vite (au sens où $\sum_t \alpha_t$ diverge mais $\sum_t \alpha_t^2$ converge), et si chaque couple (s,a) est visité infiniment souvent (au sens où $\pi(s,a) > 0$ pour chaque couple (s,a)).

3 Apprentissage par renforcement profond

3.1 Les Deep Q-Networks (DQN)

L'idée principale derrière l'apprentissage par renforcement profond est d'estimer la fonction action-valeur optimale $Q^\star$ à l'aide de réseaux de neurones. Formellement, il s'agit de trouver $\theta^\star$ tel que $Q(s,a;\theta^*) \approx Q^\star(s,a)$. Pour cela, une structure efficace souvent utilisée est le *deep Q-network* (DQN) développé par Mnih et al. en 2015. Il est essentiellement basé sur le Q-learning et permet de gérer les environnements de grande dimension.

Un Q-network est entraîné pour ajuster les paramètres θ_i à l'itération i pour réduire le MSE (Mean squarred error) dans l'équation de Bellman. Les valeurs optimales cibles $r+\gamma \max_{a'} Q^\star(s',a')$ sont subtituées avec les approximations $r+\gamma \max_{a'} Q(s',a';\theta_i^-)$ où les θ_i^- sont les paramètres de l'itération précédente. On obtient ainsi un séquence de fonctions pertes $L_i(\theta_i)$ qui change à chaque itération i :

$$
\begin{aligned}
L_i(\theta_i) =& \mathbb{E}_{s,a,r}\left[\left(\mathbb{E}_{s'}[r+\gamma \max_{a'} Q(s',a';\theta_i^-)] - Q(s,a;\theta_i)\right)^2\right] \\
L_i(\theta_i) =& \mathbb{E}_{s,a,r,s'}\left[\left(r+\gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right] + \\
& \mathbb{E}_{s,a,r}\left[\mathbb{V}_{s'}[r+\gamma \max_{a'} Q(s',a';\theta_i^-)]\right].
\end{aligned}
$$

Ainsi, chaque étape de l'optimisation correspond à la minimisation d'une fonction de perte $L_i(\theta_i)$. Le dernier terme de la somme (dans l'expression de $L_i(\theta_i)$) correspondant à la variance des cibles (action-valeurs optimales) ne dépendant des paramètres θ_i en cours, peut être ignoré. L'algorithme d'entraînement d'un Q-network modifie l'algorithme de Q-learning standard afin de s'adapter à de grands réseaux de neurones.

Tout d'abord, l'agent choisit et exécute des actions selon une politique ϵ-greedy basé sur Q. Une technique appelé *relecture d'expérience* est utilisé afin de stocker les expériences de l'agent à chaque pas de temps, $e_t = (s_t,a_t,r_t,s_{t+1})$ dans un jeu de données $D_t = \{e_1,...,e_t\}$ et tout ceci regroupé par épisode dans une mémoire de relecture. Ces échantillons stockés sont ensuite tirés aléaoiterement pour être utilisés pendant l'étape d'optimisation des paramètres θ_i. Le fait de les tirer aléatoirement à un certain avantage par rapport à l'algorithme de Q-learning classique. Apprendre directement d'échantillons consécutifs s'avère être peu efficace à cause des fortes corrélations présentes entre les étapes d'une expérience. Rendre aléatoires ces échantillons permet d'éviter cela et de réduire la variance de la mise à jour des paramètres. Ainsi les oscillations de paramètres menant à une divergence peuvent être évitées.

Tout comme un algorithme de Q-learning classique, un algorithme de Q-learning profond est *off-policy*. Il utilise 2 politiques, une politique cible qui est la politique optimale que l'on veut apprendre et une autre plus exploratoire permettant de générer différents comportements. Un Q-network permet ainsi d'estimer les fonctions action-valeur de ces 2 politiques. Cependant, cette approche pose un problème. Une mise à jour qui augmente $Q(s_t,a_t)$ augmente aussi souvent $Q(s_{t+1},a)$ pour tout action a. Il peut en résulter des oscillations de la fonction action-valeur cible et une divergence de la politique cible. Pour résoudre ce problème, Mnih et al. (2015) propose de générer les valeurs cibles avec d'anciennes mises à jour de paramètres. Formellement, toutes les C itérations le Q-network est clôné pour obtenir un $\hat{Q}$-network pour estimer les valeurs cibles pour les C prochaines itérations. Ceci engendre un décalage entre le moment où Q est mis à jour et le moment où cette mise à jour affecte les valeurs cibles, rendant ainsi des oscillations et une divergence moins probables.

3.2 Une variante : Les Double Deep Q-Networks (DDQN)

Tous les algorithmes mentionnés jusque là nécessitent des opérations de maximisation afin de construire les polices cibles. Dans ces algorithmes, l'estimation de valeurs maximales est faite en prenant des maximums d'autres valeurs estimées. Cette approche pose un problème. Le fait qu'il y ait une incertitude sur chaque estimation introduit un biais lorsqu'on prend le maximum et entraîne une surestimation des valeurs optimales. Une façon de résoudre cela est de séparer la sélection de la meilleure action de l'évaluation des actions. Cette approche consiste à conserver deux estimations indépendantes Q_1 et Q_2 de la fonction action-valeur. On peut ainsi par exemple utiliser Q_1 pour déterminer la meilleure action $a^\star = \text{argmax}_a\, Q_1(a)$ et Q_2 pour estimer sa valeur, $Q_2(a^\star) = Q_2(\text{argmax}_a\, Q_1(a))$. Cette méthode est appelé le *double learning*. Lorsqu'on fusionne cette dernière et l'algorithme Q-learning on obtient un algorithme appelé Double Q-learning. Ce dernier consiste à jouer à pile ou face à chaque pas de temps pour décider quelle fonction acton-valeur mettre à jour. Si pile est obtenu par exemple, alors la mise à jour se fait ainsi :

$$Q_1(S_t,A_t) = Q_1(S_t,A_t) + \alpha[R_{t+1} + \gamma Q_2(S_{t+1}, \text{argmax}_a\, Q_1(S_{t+1},a)) - Q_1(S_t,A_t)]$$

Si face est obtenu, alors la même mise à jour est faite en échangeant les rôles de Q_1 et Q_2.

L'architecture du DQN présenté plus haut permet d'appliquer la méthode double learning sans avoir à créer de nouveaux réseaux de neurones. En effet, nous disposons d'un réseau de neurones original suivant une politique exploratoire et d'un réseau de neurones cible pour estimer la politique cible. Hasselt propose ainsi de sélectionner l'action optimale avec le réseau de neurones original et d'estimer sa valeur avec le réseau de neurones cible. L'architecture obtenue est celle d'un *double deep Q-network* (DDQN).

Dans une telle architecture, la fonction perte à minimiser à chaque étape est :

$$L_i(\theta_i) = \mathbb{E}\left[(r + \gamma Q(s', \text{argmax}_{a'}\, Q(s',a';\theta_i); \theta_i^-) - Q(s,a;\theta_i))^2\right]$$

Pour rappel, θ_i^- correspond à une ancienne mise à jour des paramètres. DDQN est la variante la plus simple de DQN. Il existe d'autres variantes plus sophistiquées dans la littérature.

4 Cadre Multi-Agents et non-stationnarité

4.1 La spécificité du cadre Multi-Agents

Nous avons développé dans les parties précédentes le cadre de l'apprentissage par renforcement, ses principaux algorithmes, et l'utilisation des réseaux de neurones en apprentissage par renforcement. Toutes ces considérations ont été faites dans le cadre Mono-Agent.

Toutefois, de nombreuses situations réelles se prêtent naturellement davantage à l'introduction de systèmes Multi-Agents. L'introduction d'un cadre Multi-Agents peut aussi être motivée par une volonté d'amélioration des performances en mettant en collaboration ou compétition de multiples agents.

Plusieurs communautés se sont intéressées à l'apprentissage par renforcement dans le cadre Multi-Agents. Dans la suite, nous adoptons un point de vue issu de la théorie des jeux, car il nous a semblé qu'il s'agissait du point de vue le plus naturel sur la question.

Le processus de décision de Markov (PDM) introduit précédemment est un problème de décision à agent unique. Dans le cadre de l'apprentissage par renforcement Multi-Agents, le PDM est généralisé à un jeu stochastique, ou un jeu de Markov, qui peut être défini comme un n-agent PDM. Plus formellement, un jeu de Markov à n agents est défini par un tuple $(S,\{A^i\}_{i\leq n},p,\{r^i\}_{i\leq n})$ où S désigne l'espace, toujours supposé, discret des états observés par les agents.

On définit l'espace joint des actions par $A = A^1 \times A^2 \times ... \times A^n$. L'application $p : S \times A \to [0,1]$ est alors la probabilité de transition, et les applications $r^i : S \times A \times S \to \mathbb{R}$ sont les fonctions de gain qui détermine le gain perçu par le i-ème agent lors de sa transition de (s,a) à s'. On notera que différents agents peuvent recevoir différentes récompenses pour la même transition. On suppose à nouveau que les transitions entre états vérifient la propriété de Markov.

Tout comme dans le modèle des PDMs Mono-Agent, chaque agent essaie d'optimiser son gain propre. La différence fondamentale par rapport au cas précédent est que ces gains dépendent maintenant aussi des politiques des autres agents. En effet, l'objectif de chaque agent est de maximiser son gain propre à long terme et ce en déterminant une politique $\pi^i : S \to \Delta(A^i)$ (où $\Delta(A^i)$ est l'espace des lois de probabilité sur A^i) telle que $a_t^i \sim \pi^i(.|s_t)$. Dans le cas d'un paramétrage Multi-Agents, les transitions d'état sont le résultat de l'action de tous les agents ensemble, de sorte que les "récompenses" des agents dépendent de la politique jointe $\pi(a|s) = \prod_{i=1}^n \pi^i(a^i|s)$:

$$R_\pi^i = \mathbb{E}\left[R_{t+1} \mid s_t = s, a_t^i = a, \pi\right].$$

De même, les équations de Bellman de chaque agent dépendent de la politique jointe π :

$$V_\pi^i(s) = \mathbb{E}^i \left[R_{t+1} + \gamma V_\pi^i(s_{t+1}) \mid s_t = s, \pi \right], \text{et}$$

$$Q_\pi^i(s,a) = \mathbb{E} \left[R_{t+1} + \gamma Q_\pi^i(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a, \pi \right].$$

C'est cette dépendance en la politique jointe qui est la source du problème de non-stationnarité développé plus loin.

On distingue plusieurs sortes de jeux de Markov, des jeux entièrement coopératifs, des jeux entièrement compétitifs, et des jeux mixtes.

Jeux coopératifs

Dans un jeu stochastique entièrement coopératif, la fonction de récompense est la même pour tous les agents, $R^1 = R^2 = ... = R^n$, et le but des agents est de maximiser leurs récompenses communes. L'objectif peut être exprimé en apprenant les valeurs optimales de l'action conjointe et on peut utiliser le Q$-$*learning* pour l'apprentissage :

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1},a) - V(s_t,a_t))$$

, où chaque agent utilise une politique dite *greedy* pour maximiser la récompense commune :

$$\pi^i(s) = \arg\max_{a^i} \max_{a^1,...,a_{i-1},a_{i+1},...,a^n} Q^*(s,a)$$

Toutefois, dans certains états, plusieurs actions jointes peuvent être optimales.

Jeux compétitifs

Dans un jeu stochastique enitèrement concurrentiel, par exemple, si $n = 2$, $R^1 = -R^2$, et les deux agents ont des objectifs opposés. Le principe du minimax peut être appliqué : maximiser son gain dans l'hypothèse la plus défavorable où l'adversaire s'efforcera toujours de le minimiser. L'algorithme $minimax - Q$ utilise le principe minimax pour calculer les politiques et les valeurs des jeux de scène, et une règle de différence temporelle similaire au Q-learning pour propager les valeurs à travers les transitions d'état. L'algorithme pour l'agent 1 est donné ci-dessous :

$$h_{1,t}(s_t,.) = \arg m1(Q_t,s_t)$$
$$Q_{t+1}(s_t,a_t^1,a_t^2) = Q_t(s_t,a_t^1,a_t^2) + \alpha(r_{t+1} + \gamma m1(Q_t,s_{t+1}) - Q_t(s_t,a_t^1,a_t^2))$$

où $m1$ est le *minimax* de l'agent 1 défini comme :

$$m1(Q,s) = \max_{h1(s,.)} \min_{a^2} \sum_{a^1} h1(s,a^1) Q(s,a^1,a^2)$$

La politique stochastique de l'agent 1 dans l'état s à l'instant t est désignée par $h_{1,t}(s,.)$. Le problème d'optimisation de $m1$ peut être résolu par des méthodes de programmation linéaire. La table Q n'est pas indexée par l'indice de l'agent, car les équations font l'hypothèse implicite que Q = Q1 = -Q2 ; ceci découle de l'égalité $R^1 = R^{-2}$. La méthode *minimax* est bien indépendante des adversaires. En effet, l'optimisation minimax admet plusieurs solutions, elles permettent d'obtenir le résultat *minimax* indépendamment de ce que fait l'adversaire. Cependant si l'adversaire ne prend pas toujours l'action qui est la plus mauvaise pour l'apprenant et que l'apprenant a un modèle de la politique de l'adversaire, il peut faire mieux que le minimax.

Jeux mixtes

Dans les jeux mixtes, aucune contrainte n'est imposée aux fonctions de récompense des agents. Les agents y sont intéressés par le propre gain/récompenses mais pas de manières nécessairement concurrentes.

4.2 La problématique de la non-stationnarité

La considération de multiples agents fait apparaître de nombreux problèmes. Citons par exemple l'observabilité partielle, qui correspond au cas où les différents agents n'ont pas une information complète sur leur environnement commun. Cette problématique se formalise naturellement dans le cadre des Processus de Décision Markoviens Partiellement Observés. Citons aussi la mise au point de schéma d'entraînement efficaces, l'extension aux espace d'états continus, ou encore l'utilisation de l'apprentissage par transfert.

Parmi ces problématiques figure celle de la non-stationnarité.

En effet, dans le cas Mono-Agent, observant un état de l'environnement, un agent choisit une action et observe le changement d'état de l'environnement qui est la conséquence de sa seule action. Dans le cas Multi-Agents en revanche, chaque agent observe l'environnement commun et choisit en conséquence une action, puis observe le changement d'état de l'environnement résultat de l'ensemble des actions des agents. Pour chaque agent, la propriété de Markov n'est donc plus vérifiée. Il y a des effets d'intéraction entre les agents et d'apprentissage simultané ; c'est dans ce cadre qu'apparaît le problème de non-stationnarité : l'objectif d'un agent donné est d'apprendre la meilleure politique en réponse aux politiques des autres agents. Si les autres agents adaptent leur politique aussi, alors la politique *cible* de l'agent considéré varie dans le même temps (i.e. le critère à optimiser par un agent donné dépend de la politique jointe des agents comme vu précédemment). Pour cette raison, le problème de non-stationnarité est aussi connu sous le nom de *moving target problem*.

Formellement, considérons un jeu stochastique $(S,\mathcal{N},A,T,R)$ où S est un ensemble fini d'états, $\mathcal{N}$ un ensemble fini de $\mathcal{I}$ joueurs, $A = A_1 \times \cdots \times A_{\mathcal{I}}$ où A_i est un ensemble d'actions pour le joueur i, $T : S \times A \times S \leftarrow \mathbb{R}$ la fonction de probabilité de transition après une action, $R = \{r_1, \ldots, r_{\mathcal{I}}\}$ où $r_i : S \times A \leftarrow \mathbb{R}$ est la fonction de gain pour le joueur i. Dans le cadre multi-agents, la fonction valeur V_i^{π} d'un agent dépend de l'action jointe $a = (a_i,a_{-i})$ et de la politique jointe $\pi(a,s) = \prod_j \pi_j(s,a_j)$:

$$V_i^{\pi}(s) = \sum_{a \in A} \pi(s,a) \sum_{s' \in S} T(s,a_i,a_{-i},s')[R(s,a_i,a_{-i},s') + \gamma V_i^{\pi}(s')]$$

Par conséquent, la politique optimale d'un agent dépend des politiques des autres agents :

$$\pi_i^{\star}(s,a_i,\pi_{-i}) = \text{argmax}_{\pi_i} V_i^{(\pi_i,\pi_{-i})}(s)$$

Plusieurs approches, issues de l'apprentissage par renforcement mais aussi de la théorie des jeux, ont été proposées pour prendre en compte ce phénomène de non-stationnarité. Les auteurs ont notamment listé de nombreux algorithmes de l'état de l'art. Nous proposons quant à nous de nous attarder sur les différentes approches théoriques pour contrer ce phénomène.

Nous nous plaçons dans la suite dans le cadre de la théorie des jeux.

Un jeu sous forme normale est formellement défini comme un tuple $(\mathcal{N}, A, u)$, où $\mathcal{N}$ est un ensemble fini de $\mathcal{I}$ joueurs, indexé par i, $A = A_1 \times ... \times A_{\mathcal{I}}$, où A_i est l'ensemble fini des actions possibles pour le joueur i, et $u = (u_1, ..., u_{\mathcal{I}})$, où $u_i : A \to \mathbb{R}$ est la fonction d'utilité du joueur i. Pour $(\mathcal{N}, A, u)$ un jeu sous forme normale, on définit l'ensemble des stratégies mixtes pour le joueur i comme $\mathcal{S}_i = \Delta(A_i)$ (cette notion coïncide avec celle de politique stochastique introduite précédemment). Ensuite, étant donné un profil de stratégies mixtes s_{-i} des joueurs concurrents du joueur i, la meilleure réponse du joueur i est la stratégie mixte s_i^* telle que $\forall s_i \in \mathcal{S}_i \; u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i})$. On la note $\mathcal{BR}(s_{-i})$ (pour *Best Response*) dans la suite. Ce cadre étant posé, le problème de non-stationnarité apparaît maintenant tout naturellement : par définition même, $\mathcal{BR}(s_{-i})$ dépend de s_{-i}, qui est voué à changer durant l'apprentissage des joueurs concurrents. Par ailleurs, une remarque importante est que la politique optimale d'un agent peut être interprétée comme la meilleure réponse aux politiques des autres agents : $\pi_i^{\star}(s,a_i,a_{-i}) = \mathcal{BR}_i(\pi_{-i})$.

Un concept central en théorie des jeux est celui d'équilibre de Nash : une famille de stratégies $s = (s_1, ..., s_n)$ est un équilibre de Nash si $\forall i \in \{1, ..., n\} \; s_i = \mathcal{BR}(s_{s-i})$. Cette situation correspond à la situation de meilleures réponses mutuelles et constitue un objectif à atteindre.

Plusieurs approches ont été mises au point pour cela :

— Ignore,
— Forget,
— Respond to target opponents,
— Learn opponents models,
— Theory of mind.

Plaçons-nous dans le cadre d'un jeu à deux joueurs entièrement concurrentiel.

Dans l'approche **Ignore**, les joueurs ignorent le caractère non-stationnaire de la situation : chaque joueur suppose que son concurrent joue une stratégie mixte fixée et les fréquences des actions observées sont utilisées pour estimer cette stratégie mixte. Si l'un des deux joueurs ne joue pas une stratégie stationnaire, alors cette approche ne mène pas à une meilleure réponse (il n'y a aucune garantie théorique).

Une approche moins naïve est l'approche **Forget**, qui consiste à ne tenir que de l'information récente dans l'amélioration de la politique courante. Cette approche mène à des algorithmes nécéssitant souvent des temps longs d'entraînement pour converger.

Dans l'approche **Respond to target opponents**, pour chaque agent i, un groupe de stratégies concurrentes est pré-défini et l'agent i apprend la meilleure réponse à la stratégie dans ce groupe se rapprochant le plus de la stratégie concurrente observée. On reproche en général à ce type d'approche son manque d'adaptabilité (les algorithmes en découlant mènent à de très mauvais résultats dès que la stratégie concurrente ne se trouve pas dans le groupe de stratégies pré-défini).

L'approche précédente fait des hypothèses très fortes : on présuppose que la stratégie du joueur concurrent appartient à un groupe de stratégies pré-défini. Dans l'approche **Learn opponents models** au contraire, chaque agent tente d'apprendre le comportement de son concurrent, sans a priori. Un désavantage de ce type de techniques est notamment qu'elles nécéssitent de supposer que le joueur concurrent garde une stratégie suffisamment longtemps pour qu'elle puisse être apprise par son concurrent, ce qui peut être irréaliste dans certains cas.

Enfin, dans l'approche **Theory of mind**, chaque joueur suppose que son concurrent raisonne sur sa propre stratégie : le joueur 1 suppose que le joueur 2 modélise sa stratégie et en déduit une meilleure réponse ; le joueur 1 cherche alors une meilleure réponse à cette meilleure réponse ; le joueur 1 suppose ensuite que le joueur 2 a réussi à modéliser cette meilleure réponse à une meilleure réponse et en a déduit une meilleure réponse ; le joueur 1 ... Cette approche est très sophistiquée et peut mener à d'excellents résultats dans des jeux de stratégie. Elle est néanmoins très lourde au niveau computationnel.

Ces différentes approches de la non-stationnarité ont donné lieu à de (très) nombreux algorithmes dont une liste a été dressée par les auteurs.

5 Conclusion

L'article que nous avons présenté, **Deep Reinforcement Learning for Multi-Agent Systems : A Review of Challenges, Solutions and Applications**, donnait une vue d'ensemble sur le cadre générique de l'apprentissage par renforcement Mono-Agent, et les différentes méthodes associées (Monte-Carlo, Sarsa, Q-learn), sur l'utilisation qui pouvait être faite des réseaux de neurones en apprentissage par renforcement, en tant qu'approximateurs de fonctions valeur et q-valeur, et enfin sur l'introduction du cadre Multi-Agents, des problématiques et des algorithmes associés.

L'objectif de cet article était simplement de donner à son lecteur un point de vue global sur l'état de l'art, et sur les différents (et nombreux) algorithmes existants.

Nous avons tenté au travers de notre présentation de reprendre ce cadre, en y ajoutant une rigueur mathématique certaine, notamment en nous appuyant sur des résultats vus en cours ou lus dans les références laissées par les auteurs en fin d'article.

Merci pour votre lecture !