CentraleSupélec

Reinforcement Learning

Codage d'un agent de Puissance 4

Auteur :
Guilhem Prince

En groupe avec :
Saad Chtouki
Ibrahim Rakib

Encadrants :
Hédi Hadiji

Avril 2023

CentraleSupélec

Table des matières

1 Introduction

Pour ce projet, j'ai décidé d'explorer la piste d'un contrôleur Monte Carlo, comme vu lors du TD numéro 3 à propos du BlackJack. Cette approche vient compléter le travail de Saad et Ibrahim, qui eux ont implementé un agent qui apprend par QLearning, méthode qu'ils ont pu étudier plus en profondeur.

Mon objectif était de voir si cette methode pouvait vaincre les deux agents naïfs donnés dans le notebook starter, à savoir *RandomPlayer* et *PlayLeftmostLegal*.

Le code ainsi que tous les documents relatifs à notre projet sont disponibles en open source sur notre dépot github.

2 Implémentation

2.1 Méthode utilisée

Cette méthode de contrôle par Monte Carlo est une méthode d'évaluation des valeurs de q suivant une politique (policy), avec comme amélioration de politique l'ϵ-cupidité (greedyfication).

Même si l'un des problèmes de cette méthode de prédiction de Monte-Carlo est que l'on doit attendre un episode complet pour commencer à apprendre, et que cela est parfois inefficace, il était tout de même intéressant de tester cette méthode pour la comparer au Q-Learning, dans leur dualité de méthodes on-policy, off-policy. En effet, avec le controlleur Monte Carlo (on-policy), nous apprenons les valeurs des états à partir de la politique que nous jouons (dans notre cas au puissance 4, le hasard), contrairement au Q-Learning, qui estime les valeurs **indépendamment** de la politique utilisée, sûrement plus efficace.

2.2 En pratique

Pour l'implementation, j'ai repris en grande partie du code du TD numéro 3 (celui du Blackjack). Il a fallu adapter l'agent à l'environnement PettingZoo, différent de l'environnement Gym habituellement utilisé. Il a donc fallu notamment écrire manuellement les *action_space* et *observation_space* :

[0,1,2,3,4,5,6] pour l'espace d'actions et convertir les arrays numpy de l'espace des observations (tous de shape (6, 7, 2)) en des tuples, à l'aide d'une fonction récursive, disponible dans le fichier utils.py. Il a aussi fallu inclure la gestion des actions illégales : ceci est fait au niveau de la methode *get_action*, où l'on intersecte le masque des actions légales avec l'espace des actions pour faire notre choix.

J'ai aussi réimplémenté une méthode *run_N_episodes* qui entraîne mon agent en se basant sur l'environnement Gym simulé *EnvAgainstPolicy* fourni dans le notebook de démarrage.

2.3 Résultats

Les résultats ainsi que les analyses de mon agent contrôlé par méthode de Monte Carlo sont détaillés dans le notebook "monte_carlo_controller.py". En résumé, celui-ci bat 100% des fois le *PlayLeftmostLegal()*, en jouant en premier ou en deuxième (bien qu'un seul entraînement ne suffise pas à le faire gagner tout le temps dans les deux cas). Par contre, l'agent n'obtient aucun résultat contre le joueur aléatoire. Mon hypothèse est que l'agent apprend à ne pas perdre contre une stratégie bien spécifique (jouer toujours à gauche), mais n'a pas le temps d'apprendre à gagner dans le nombre d'épisodes donnés. Ainsi, contre un joueur aléatoire, l'agent joue globalement aléatoirement, et a un résultat similaire qu'un agent aléatoire contre un autre agent aléatoire.

Peut-être qu'un nombre d'épisodes plus grand réussirait à faire gagner mon agent contre un joueur aléatoire. Néanmoins, les résultats me font dire qu'une approche "on-policy" n'était sûrement pas la plus optimale pour un MDP avec un espace d'observations si grand.

3 Pistes d'améliorations et autres méthodes

D'autres methodes ont été envisagées, notamment des méthodes d'exploration d'arbre, telles que le MCTS ou le minimax. Un papier était notamment fourni sur Édunao pour la Monte Carlo Tree Search, ce qui m'a fait hésiter à implémenter cette méthode, avant que je réalise qu'elle n'incluait pas réellement d'apprentissage. C'est pourquoi je me suis finalement tourné vers un algorithme vu en cours.

En étudiant la MCTS, l'autre approche commune d'exploration d'arbre qu'est la méthode Minimax avec découpage α - β, m'est apparue intéressante. Cette méthode fonctionne bien avec une bonne fonction heuristique et tant que le facteur de branchement est raisonnable. Ici, une fonction heuristique admissible pourrait etre de compter les suites de 2 jetons d'affilés, de 3 jetons d'affilés, pour notre agent et l'agent adverse, et d'affecter des poids à ce comptage. Dans le Puissance 4, le facteur de branchement n'est pas trop élevé car il n'y a que, au plus, 6 actions possibles à chaque tour (bien plus faible que le Jeu d'Échecs ou le Go par exemple). Cela dit, cette méthode sortait du cadre de l'étude car elle n'inclut pas d'apprentissage.

Pour améliorer nos approches d'apprentissage par renforcement (Monte Carlo contrôleur ou QLearner), il serait intéressant d'implémenter une méthode de Deep QLearning, qui évalue les valeurs d'états à l'aide de réseaux de neurones, car le Puissance 4 peut être un espace d'états trop grand pour les méthodes tabulaires.

4 Conclusion

Bien que cette méthode n'ai pas eu de résultats très concluants, battant seulement les bots les plus idiots au Puissance 4, et étant loin de battre un humain, celle-ci avait le mérite d'être comparée à la méthode de Q-Learning, qui aura de bien meilleurs résultats contre le joueur aléatoire (Voir les rapports de Saad et Ibrahim).