Rapport de TP : Implémentation d'algorithmes de Reinforcement Learning en Java

Robin Courault, Julien Floyd

Sommaire

Introduction

Ce projet avait pour objectif d'introduire les concepts fondamentaux du *reinforcement learning* (apprentissage par renforcement) à travers l'implémentation et la comparaison de plusieurs algorithmes classiques dans des environnements variés.

Nous avons développé en Java plusieurs stratégies d'apprentissage automatique, allant des méthodes simples d'exploration/exploitation (multi-armed bandits) à des méthodes plus complexes de planification de politique (Policy et Value Iteration) adaptées aux processus de décision markoviens (MDP).

Choix d'implémentations

Généralités

L'ensemble des algorithmes a été implémenté avec une structure modulaire pour faciliter la comparaison, le réemploi de code et l'expérimentation.
Les paramètres globaux suivants ont été appliqués :

- **Nombre d'itérations maximal** : 10 000
- **Seuil de convergence** :
    - 0.00001 pour les algorithmes de type bandit,
    - 0.001 pour les algorithmes MDP (Policy/Value Iteration)

Le critère d'arrêt repose sur la stabilité des valeurs ou de la politique, selon l'algorithme.

ϵ-Greedy

L'algorithme ϵ-greedy repose sur un équilibre entre exploitation des meilleures actions connues et exploration d'actions alternatives.

Notre implémentation utilise les paramètres suivants : - **Taux d'apprentissage** : 0.2 - **Facteur de réduction (discount)** : 0.9 - **Taux d'exploration** ϵ : 0.2

Ce choix de configuration vise à assurer une convergence rapide tout en laissant place à la découverte de nouvelles stratégies au début de l'apprentissage.

UCB (Upper Confidence Bound)

UCB affine la stratégie d'exploration en intégrant une borne de confiance qui privilégie les actions moins explorées. Son principal avantage est une meilleure gestion du compromis exploration/exploitation que ϵ-greedy, sans nécessiter de paramètre arbitraire comme ϵ.
La réutilisation de la structure de ϵ-greedy a permis une implémentation efficace, centrée sur la redéfinition des stratégies de sélection et de mise à jour des récompenses.

Bandit Gradient

L'algorithme Bandit Gradient adopte une approche probabiliste : les préférences sont modifiées via une mise à jour stochastique basée sur la récompense obtenue et une référence (*baseline*).
Nous avons utilisé : - **Taux d'apprentissage** : 0.2 - **Récompense de base** : initialisée à 0

Ce type d'algorithme est particulièrement efficace dans des environnements non-stationnaires.

Policy Iteration

Policy Iteration est un algorithme central dans les MDPs. Il alterne entre : 1. L'évaluation de la politique courante (calcul des valeurs d'état), 2. L'amélioration de cette politique à partir des nouvelles valeurs.

Dans notre cas, la nature **déterministe** de l'environnement a permis de simplifier considérablement les calculs. Nous avons optimisé l'implémentation en arrêtant le processus dès que la politique ne varie que faiblement.

Value Iteration

Value Iteration se concentre sur la mise à jour des valeurs d'état sans évaluation explicite d'une politique. La politique optimale est extraite une fois que les valeurs convergent.
C'est un algorithme simple, mais très efficace, notamment dans des environnements de taille modérée comme notre implémentation du jeu de Tic-Tac-Toe.

Analyse des algorithmes

Premier cas : environnement simple

Dans ce scénario, l'utilisateur sélectionne toujours le même item (par exemple, un film favori).
L'environnement est donc **stationnaire** et **prévisible**.
Parmi les trois algorithmes de bandits, ϵ-Greedy est le plus efficace : - Convergence rapide, - Simplicité de mise en œuvre, - Résultats fiables dans des contextes stables.

UCB et Bandit Gradient sont plus complexes, sans offrir d'avantage réel tangible ici.

Deuxième cas : YouTube

Ce scénario simule un utilisateur dont les goûts changent avec le temps.
L'environnement devient **non-stationnaire**, ce qui complexifie la tâche d'apprentissage.
Dans ce contexte : - **Bandit Gradient** tire son épingle du jeu, grâce à sa capacité d'adaptation, - ϵ-Greedy et UCB, bien que performants initialement, ne s'ajustent pas bien aux changements.

Cela montre l'intérêt des algorithmes adaptatifs dans les systèmes de recommandation en ligne.

Troisième cas : Tic-Tac-Toe

Le jeu de Tic-Tac-Toe présente une **structure séquentielle** avec un nombre d'états non négligeable, mais encore **traitable**.
Les algorithmes **Policy Iteration** et **Value Iteration** permettent de converger vers une stratégie optimale.
Puisque l'environnement est assez petit, la politique **Value Iteration** est plus efficace, car plus simple.

Conclusion

Ce projet nous a permis d'expérimenter plusieurs méthodes d'apprentissage par renforcement à travers des cas concrets et variés.
Les principales leçons que nous en tirons sont les suivantes :

- Les algorithmes doivent être choisis en fonction de la **nature de l'environnement** (stationnaire vs. non-stationnaire, simple vs. complexe).
- Les approches simples comme ϵ-greedy sont **rapides et efficaces** dans des contextes prévisibles.
- Des algorithmes comme **Bandit Gradient** sont mieux adaptés à des environnements évolutifs ou instables.
- Dans les environnements à **états multiples**, les algorithmes de type **MDP** (Policy et Value Iteration) permettent d'atteindre des stratégies optimales.

Enfin, ce travail nous a aussi sensibilisés à l'importance : - du **choix des paramètres** (taux d'apprentissage, discount, etc.), - de la **structure logicielle** pour la lisibilité et la réutilisabilité du code, - et de l'**observation expérimentale** pour valider les comportements théoriques.