CentraleSupélec | université PARIS-SACLAY

Projet 2023 / **Lien du github**

Connect4 agent

Reinforcement Learning

Étudiants :
Benitah Noam

Enseignants :
Hadiji Hédi

24 avril 2023

I Introduction au problème

I.1 Contexte

L'apprentissage par renforcement est une sous-discipline du machine learning qui se concentre sur l'optimisation des actions d'un agent pour maximiser une certaine récompense dans un environnement donné. Cette approche a montré des résultats prometteurs dans diverses applications, notamment dans les jeux. Dans le domaine des jeux, l'apprentissage par renforcement a été utilisé pour développer des agents capables de rivaliser, voire de surpasser, les compétences humaines. Des exemples notables incluent AlphaGo de DeepMind, qui a battu les meilleurs joueurs mondiaux de Go, ainsi que l'agent OpenAI Five, qui a réussi à vaincre des équipes professionnelles de Dota 2. Ces succès montrent que l'apprentissage par renforcement peut être utilisé pour résoudre des problèmes complexes et créer des agents intelligents capables d'affronter des adversaires humains.

I.2 Formalisation du problème

Le jeu de Connect Four est un jeu de société classique qui représente un problème intéressant pour les algorithmes d'apprentissage par renforcement. Le but du jeu est simple : aligner quatre jetons de sa propre couleur verticalement, horizontalement ou en diagonale sur une grille 6x7. Voici une formalisation plus précise de notre problème :

- **Espace d'états (S)** : L'espace d'états est constitué de toutes les configurations possibles du plateau de jeu. Le plateau de jeu de Connect Four est une grille 2D de dimensions 6x7, où chaque case peut être vide, occupée par un jeton du joueur 1 ou occupée par un jeton du joueur 2. Ainsi, il y a 3**(6x7) états possibles, bien que beaucoup d'entre eux ne soient pas atteignables en raison des règles du jeu.
- **Espace d'actions (A)** : L'espace d'actions représente les actions que les joueurs peuvent effectuer à chaque tour. Dans Connect Four, les joueurs peuvent choisir de placer un jeton dans l'une des 7 colonnes du plateau, à condition que la colonne ne soit pas déjà remplie. Ainsi, l'espace d'actions est un ensemble discret de 7 actions possibles.
- **Règles de transition** : Les règles de transition définissent comment le plateau de jeu évolue en fonction des actions des joueurs. Lorsqu'un joueur choisit une colonne, son jeton est placé dans la case la plus basse disponible de cette colonne. Les joueurs alternent les tours jusqu'à ce que l'un des joueurs gagne ou que le plateau soit rempli, entraînant une égalité.
- **Fonction de récompense (R)** : La fonction de récompense attribue une valeur numérique à chaque transition d'état en fonction de l'action choisie et du résultat. Dans le contexte de Connect Four, la récompense peut être définie comme suit :
  - Si l'action mène à une victoire, attribuer une récompense de 1
  - Si l'action mène à une défaite, attribuer une récompense de -1
  - Si l'action mène à une égalité, attribuer une récompense nulle
  - Si l'action ne mène pas à la fin du jeu, attribuer une récompense nulle
- **Objectif** : L'objectif du jeu pour chaque joueur est de maximiser sa fonction de récompense cumulée en plaçant ses jetons de manière à aligner quatre jetons de sa couleur, soit horizontalement, verticalement ou en diagonale. Les agents d'apprentissage par renforcement sont entraînés pour apprendre des stratégies optimales en explorant l'espace d'états et en adaptant leur comportement en fonction des récompenses reçues.

II Algorithme

Dans ce projet, nous avons développé et évalué plusieurs agents d'apprentissage par renforcement pour le jeu de Connect Four en utilisant différentes techniques telles que les arbres de recherche Monte Carlo (MCTS) et les réseaux de neurones profonds avec Advantage Actor-Critic (A2C). Notre objectif était de créer un agent capable de rivaliser avec des joueurs humains et d'approfondir notre compréhension des défis et des opportunités associés à l'application de l'apprentissage par renforcement aux jeux.

II.1 Design et structure du code

Le code de ce projet est organisé en différentes parties pour faciliter la compréhension et la maintenance. Voici un aperçu des principales composantes du code :

- **Agent de base (base-agent.py)** : Ce fichier contient la classe de base Agent, qui définit l'interface commune pour tous les agents utilisés dans le projet. Les agents spécifiques hériteront de cette classe de base et implémenteront des méthodes supplémentaires. Voici les différents agents qui héritent de cette classe :
  - **Agent MCTS (mcts.py)** : L'agent MCTS implémente l'algorithme Monte Carlo Tree Search (MCTS) pour sélectionner la meilleure action à prendre dans un jeu de Connect Four. Il effectue plusieurs simulations (contrôlées par n-simulations) pour explorer les états possibles du jeu. Il utilise un facteur c-puct pour équilibrer l'exploration et l'exploitation lors de la sélection des actions. L'arbre MCTS est constitué de MCTSNode, qui stockent les informations sur les actions, les récompenses et les visites.
  - **Agent A2C (actor-critic.py)** : L'agent Advantage Actor-Critic utilise une approche d'apprentissage par renforcement appelée "Actor-Critic". Il a deux réseaux neuronaux, l'acteur et le critique. L'acteur est responsable de la sélection des actions, tandis que le critique estime la valeur de l'état actuel.L'agent A2C possède des méthodes pour mettre à jour les réseaux neuronaux, sauvegarder et charger les modèles, et sélectionner la meilleure action en fonction de l'état actuel.
  - **Agent Random (random.py)** : L'agent Random sélectionne une action aléatoire à chaque étape. Il n'apprend pas de ses actions et n'a pas de mémoire ou de stratégie pour s'améliorer au fil du temps. Toutes les méthodes de cette classe sont des opérations simples ou vides, car il n'y a pas de mise à jour ni de mécanisme de sauvegarde et de chargement pour cet agent.
- **Humain (human.py)** : Cette classe représente un joueur humain dans un jeu. Elle est conçue pour permettre à un utilisateur d'interagir avec un environnement de jeu et un agent en fournissant des actions.
- **Environnement de jeu (game.py)** : La classe Game gère le déroulement du jeu et l'interaction entre les agents et l'environnement. Elle prend en charge l'entraînement, l'évaluation et la visualisation des parties de jeu. La classe Game utilise les instances d'agents et d'environnement pour gérer les interactions et les mises à jour des agents. Les agents sont des instances de la classe Agent ou Human.
- **Script principal (main.py)** : Ce fichier est le point d'entrée du projet. Il contient le code pour initialiser l'environnement, créer les agents, et exécuter le jeu en faisant interagir les agents avec l'environnement. Le script principal gère également l'affichage des résultats et la sauvegarde des données d'apprentissage des agents.

III Agents

Nous avons implémentés différents agents afin de comparer leurs performances. Certains sont plus performant en terme de vitesse d'execution alors que d'autres prennent de meilleures décisions pour remporter la partie. Nous pouvons aussi les différencier sur la phase d'entrainement. En effet, A2C necessite un entrainement car il est basé sur les réseaux de neurones contrairement à MCTS. Je vais ici présenter l'agent MCTS sur lequel j'ai davantage travaillé.

III.1 Monte Carlo Tree Search (MCTS)

Contrairement aux agents qui utilisent des réseaux de neurones pour l'apprentissage par renforcement, un agent MCTS n'a pas besoin d'être entraîné avant l'évaluation. Le MCTS est un algorithme de recherche en ligne qui explore l'espace des états et des actions en construisant un arbre de recherche à partir de l'état actuel du jeu. L'exploration de l'arbre est effectuée en utilisant des simulations de Monte Carlo et en suivant une stratégie d'équilibrage entre l'exploration et l'exploitation.

L'agent MCTS utilise les informations accumulées lors de la recherche pour choisir la meilleure action à prendre dans l'état actuel. Comme le processus de recherche est effectué en temps réel pendant l'évaluation, il n'y a pas de phase d'entraînement distincte pour un agent MCTS. Cependant, il est possible d'améliorer les performances d'un agent MCTS en ajustant les paramètres de l'algorithme, comme le nombre de simulations, le facteur d'exploration (Cpuct). Voici une description des points centraux de l'agent MCTS et de son fonctionnement :

- **MCTSNode** : La classe MCTSNode représente un noeud dans l'arbre de recherche. Chaque noeud contient les attributs suivants : parent, children, action, visit-count, total-reward, et average-reward. Le parent est un lien vers le noeud parent, tandis que les enfants sont une liste de noeuds enfants. Le noeud stocke également l'action menant à cet état, ainsi que le nombre de visites, la récompense totale et la récompense moyenne accumulée lors des simulations.
- **get-best-child** : Cette méthode prend en entrée un noeud et retourne le meilleur enfant en fonction de la récompense moyenne.
- **get-best-action** : Cette méthode effectue des simulations MCTS. Après n-simulations, elle sélectionne le meilleur enfant en fonction de la récompense moyenne et retourne l'action correspondante. Le fonctionnement de la selection est assuré par 4 étapes principales : Sélection, Expansion, Rollout et Backpropagation. Voici une description de ces étapes :
  - **select** : Cette méthode prend en entrée un noeud et un état, puis effectue la phase de sélection en parcourant l'arbre de recherche jusqu'à ce qu'un noeud feuille soit atteint. Si le noeud actuel n'a pas d'enfants, il est étendu en utilisant la méthode expand. Sinon, la méthode choisit le meilleur enfant en fonction de l'UCT (Upper Confidence Bound for Trees) et effectue une action pour atteindre l'état suivant.
  - **expand** : Cette méthode prend en entrée un noeud et un environnement, puis crée de nouveaux noeuds enfants pour toutes les actions légales possibles à partir de l'état actuel.
  - **rollout** : Cette méthode prend en entrée un état et effectue une simulation aléatoire à partir de cet état jusqu'à ce qu'un état terminal soit atteint. Elle

retourne la récompense obtenue à la fin de la simulation.

- **backpropagate** : Cette méthode prend en entrée un noeud et une récompense, puis met à jour les statistiques des noeuds en remontant l'arbre jusqu'à la racine.

Ces étapes sont répétées un certain nombre de fois, et à la fin, l'action du noeud enfant de la racine ayant la meilleure valeur estimée est choisie comme action optimale.

IV Hyperparamètres MCTS

Le choix des hyperparamètres dans l'algorithme MCTS peut avoir un impact significatif sur les performances et la qualité des résultats. Voici les hyperparamètres clés que nous avons utilisés :

- **Nombre de simulations (n-simulations)** : Il s'agit du nombre de fois que l'algorithme MCTS parcourt les quatre étapes (sélection, expansion, simulation et rétropropagation). Un nombre plus élevé de simulations permet d'explorer davantage l'arbre de recherche, ce qui peut améliorer la qualité de l'action choisie. Cependant, cela augmente également le temps de calcul. Il est donc important de trouver un équilibre entre le nombre de simulations et le temps disponible pour prendre une décision.
- **Constante d'exploration (c-puct)** : Cette constante détermine l'équilibre entre l'exploration (essayer de nouvelles actions) et l'exploitation (choisir des actions déjà évaluées comme bonnes) lors de la sélection des nœuds. Une valeur élevée de c-puct favorise l'exploration, tandis qu'une faible valeur favorise l'exploitation. Le choix de cette constante dépend du problème à résoudre et de l'expérience préalable. Une approche courante consiste à tester plusieurs valeurs et à sélectionner celle qui donne les meilleures performances.

Pour choisir les hyperparamètres, nous pouvons utiliser des méthodes d'optimisation des hyperparamètres, comme la recherche par grille ou la recherche aléatoire. L'impact des hyperparamètres sur les performances de l'algorithme MCTS peut varier en fonction du problème, et il est important de les ajuster en conséquence pour obtenir les meilleurs résultats possibles. Cependant,nous n'avons pas eu le temps de se concentrer sur le problème d'optimisation des hyperparamètres. Nous avons donc testés plusieurs hyperparamètres et avons choisis les suivant en fonction de la performance du modèle : **n-simulations = 100 et c-puct = 1.0.**

V Travail personnel

Nous avons travaillés en groupe sur l'étendu du projet. Nous sommes partie de 0 et avons avancé étapes par étapes en trouvant des solutions ensemble pour faire fonctionner nos agents. Si je devais mentionner une partie du projet sur laquelle j'ai plus travaillé je mentionnerais la création de l'agent MCTS et c'est donc pour cela que j'ai détaillé son fonctionnement dans mon rapport.