CentraleSupélec | université PARIS-SACLAY

Projet 2023

Connect4 agent

Reinforcement Learning

Étudiants :
Gabison Yoan

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 Analyse et résultats

Nous allons vous présenter différents résultats qui permettront de comparer les performances de nos différents agents. Une modification de certains hyperparamètres peut influencée la performance de nos agents positivement comme négativement.

II.1 Hyperparamètres

- **A2C** : Les hyperparamètres clés d'A2C sont :
  - **nb_epoch** : Le nombre d'itérations d'apprentissage effectuées sur les données.
  - **gamma** : Le facteur de remise, qui détermine l'importance accordée aux récompenses futures par rapport aux récompenses immédiates.
  - **optimiser** : Le choix de l'optimiseur pour mettre à jour les poids du réseau.
  - **lr_actor** : Le taux d'apprentissage pour mettre à jour les poids du réseau de politiques.
  - **lr_critics** : Le taux d'apprentissage pour mettre à jour les poids du réseau d'évaluation de la valeur de l'état.
  - **eps_init** : La probabilité initiale d'effectuer une action aléatoire, pour l'exploration.
  - **eps_min** : La probabilité minimale d'effectuer une action aléatoire.
  - **eps_step** : Le pas de réduction de la probabilité d'effectuer une action aléatoire, pour la recherche par descente de gradient stochastique.

  Nous avons choisi les hyperparamètres suivants de manière empirique : **nb_epoch** $= 10^4$, **gamma** $= 0.99$, **optimiser** $=$ **Adam**, **lr_actor** $= 10^{-5}$, **lr_critics** $= 10^{-5}$, **eps_init** $= 0.5$, **eps_min** $= 10^{-5}$, **eps_step** $= 10^{-3}$
- **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 tout en gardant un temps convenable : **n-simulations = 100 et c-puct = 1.0.**

II.2 Qualité de l'agent

La qualité de nos agents (MCTS et A2C) a été mesuré en les faisant jouer 1000 parties contre l'agent Random. En effet, un agent performant doit être en mesure de gagner pratiquement tout le temps contre un agent aléatoire. Voici un tableau qui résume les performances de nos agents :

		Nb of victory	Nb of losses	% of victory*	% of victory (wo draw)*
Player 1	A2C	912	88	91.2%	91.2%
	A2C Conv	996	4	99.6%	99.6%
	MCTS	785	215	78.5%	78.5%
Player 2	A2C	901	99	90.1%	90.1%
	A2C Conv	956	44	95.6%	95.6%
	MCTS	622	378	62.2%	62.2%
Mean	A2C	906.5	93.5	90.5%	90.5%
	A2C Conv	976	24	97.6%	97.6%
	MCTS	703.5	296.5	70.3%	70.3%

*Les parties invalides ne sont pas prises en compte

III Environnements complexes

La méthode MCTS peut être appliquée à des environnements plus complexes tels que le backgammon, les échecs, le jeu de Go et Starcraft. Cependant, la performance de l'algorithme dépendra de plusieurs facteurs.

Dans des jeux comme le go et les échecs, le nombre de coups possibles et la complexité des situations augmentent considérablement, ce qui rend l'exploration de l'arbre de recherche plus difficile. Dans ces cas, l'utilisation de méthodes avancées, telles que les réseaux de neurones pour l'évaluation des positions, peut être bénéfique. Par exemple, l'algorithme AlphaGo de DeepMind utilise MCTS avec des réseaux de neurones pour l'évaluation des positions et des politiques.

Pour des environnements comme Starcraft, la situation est encore plus complexe en raison de la nature temps réel et de la dimensionnalité élevée de l'espace d'action. Dans ce cas, l'approche MCTS pourrait être moins efficace, et d'autres méthodes basées sur l'apprentissage par renforcement profond pourraient être plus appropriées en raison du temps de calcul nécessaire au bon fonctionnement de MCTS.

Dans l'ensemble,il pourrait être nécessaire de combiner MCTS avec d'autres techniques, d'adapter les hyperparamètres ou d'utiliser des heuristiques spécifiques au domaine pour améliorer les performances.

IV Travail personnel

Nous avons travaillés en groupe sur l'étendu du projet. Nous nous donnions rendez-vous deux fois par semaine afin de tous pouvoir avancer sur le projet. Il n'y a pas réellement de partie qui a été étudié par un membre de l'équipe plus que d'autres. Nous sommes partie de 0 ensemble et avons avancé étapes par étapes en trouvant des solutions ensemble pour faire fonctionner nos agents.

V Conclusion

En conclusion, ce projet de reinforcement learning sur le jeu du Puissance 4 nous a permis d'explorer et de mettre en pratique différents algorithmes tels que A2C et MCTS. Nous avons fait plusieurs choix de conception lors du développement de notre algorithme, notamment en ce qui concerne la structure du code et les hyperparamètres.

La procédure d'entraînement de notre algorithme impliquait un processus itératif où l'agent jouait contre lui-même pour accumuler des données d'entraînement. Ces données étaient ensuite utilisées pour mettre à jour les poids du réseau de neurones de l'agent à l'aide de l'optimiseur Adam. Nous avons également utilisé une exploration stochastique basée sur la politique pour encourager l'agent à explorer différents mouvements pendant l'entraînement.

Le choix des hyperparamètres a été crucial pour optimiser les performances de notre algorithme. Nous avons effectué des expérimentations pour trouver les meilleures valeurs pour des hyperparamètres tels que le taux d'apprentissage, la taille du réseau de neurones, le nombre d'itérations d'entraînement, etc. Nous avons analysé l'impact de ces hyperparamètres sur les performances de l'agent et avons ajusté les valeurs en conséquence.

Pour évaluer la qualité de notre agent, nous avons utilisé des mesures de performance telles que le taux de victoires contre un agent random. Ces mesures nous ont permis de quantifier les performances de notre agent et de comparer ses performances avec d'autres approches.

Bien que notre approche ait montré de bons résultats sur le jeu du Puissance 4, il est difficile de dire avec certitude comment elle se comporterait sur des environnements plus complexes tels que le backgammon, les échecs, le go ou Starcraft. Ces jeux présentent des défis supplémentaires tels que des espaces d'états et d'actions beaucoup plus vastes, ainsi que des niveaux de complexité stratégique plus élevés.

Si nous avions plus de temps ou de puissance de calcul, nous pourrions améliorer notre approche de plusieurs manières. Tout d'abord, nous pourrions effectuer une recherche plus exhaustive des hyperparamètres pour optimiser davantage les performances de l'agent. Nous pourrions également explorer d'autres techniques d'apprentissage par renforcement avancées, telles que les réseaux neuronaux profonds convolutionnels (CNN) pour l'apprentissage des états du jeu. De plus, nous pourrions étendre notre approche à d'autres jeux complexes pour évaluer sa généralisation et son adaptation à différents environnements.