TP3 - Reinforcement Learning

Briac Six - Baptiste Bellamy

November 18, 2024

Abstract

Dans ce projet, nous avons implémenté un model de Deep Q-Learning, pour résoudre le problème du jeu Pong d'Atari. Le but du jeu consiste à déplacer un taxi sur une grille 5x5, ramasser un passager à un emplacement donné, et le déposer à une destination spécifique tout en optimisant le nombre de mouvements. Le problème est un environnement de contrôle discret, idéal pour explorer des méthodes d'apprentissage tabulaire. Notre Implémentation suit le papier 'Playing Atari with Deep Reinforcement Learning' de DeepMind Technologies sortie en Décembre.

Contents

1 Choix d'implémentation

Pour implémenter ces classes, nous nous sommes basés sur les indications fournies dans le papier de recherche. Cependant, nous avons décidé de créer deux classes distinctes pour une meilleure modularité :

- **DeepQLearningAgent** : Cette classe regroupe tous les paramètres et les fonctions nécessaires à l'agent, notamment pour le choix des actions et pour son entraînement.
- **DeepQNetwork** : Cette classe représente le modèle de Deep Q-learning. Elle est composée de trois couches de convolution et de deux couches fully connected (denses). Nous avons utilisé la bibliothèque PyTorch pour sa conception.

Nous avons entraîné notre modèle et apporté certaines modifications aux approches décrites dans le papier. Voici les principales adaptations :

- L'optimiseur original a été remplacé par **Adam**, qui a montré de meilleurs résultats lors des expérimentations.
- La mémoire de l'agent a été fixée à une taille de 10 000 transitions, car une mémoire plus grande entraînait des problèmes de stabilité durant l'entraînement.
- Pour améliorer les performances de notre modèle, nous avons intégré un **target model**, conformément à l'approche du Double Q-Learning. Cette méthode, largement documentée dans plusieurs recherches, vise à stabiliser l'entraînement en utilisant un second réseau de neurones, appelé réseau cible (target network). Cette stratégie permet de réduire les oscillations et les biais introduits par des estimations de valeur imprécises lors de l'entraînement

Concernant les paramètres de l'agent et du modèle, nous avons retenu les valeurs suivantes pour les hyperparamètres de l'agent :

- **Facteur de discount** (γ) : 0.99
- **Taille des lots** ($batch\_size$) : 32
- **Nombre d'actions possibles** ($n\_actions$) : `env.action_space.n` (6)
- **Nombre d'époques d'entraînement** ($epochs$) : 500

2 Résultats

Malgré nos nombreuses tentatives et ajustements, nous n'avons pas réussi à obtenir les résultats escomptés. Lors de nos expérimentations, le meilleur score de récompense atteint par l'agent a été de -9, bien en deçà des performances attendues. Malgré une analyse approfondie des paramètres et de l'implémentation, nous n'avons pas réussi à identifier l'origine exacte du problème.

De plus, le temps d'exécution particulièrement long de l'algorithme a fortement limité nos itérations et analyses. En effet, il a fallu attendre environ 200 époques avant d'observer une diminution notable de la perte, et l'entraînement complet nécessite plusieurs heures pour s'exécuter. Cela a rendu les ajustements et les tests plus complexes, empêchant une exploration rapide et approfondie des causes potentielles des résultats insatisfaisants.