Reinforcement Learning

- Environnement Highway & Racetrack -
CentraleSupélec - Université Paris-Saclay, Gif-Sur-Yvette
22 avril 2025

Project Report

DAVID Erwan

Objectif du projet

Ce rapport présente le développement et l'amélioration d'agents intelligents pour deux environnements de simulation : Highway-Env et Racetrack, basés sur la bibliothèque Gym. L'objectif est d'entraîner des agents capables de naviguer efficacement dans des environnements complexes, en évitant les collisions et en optimisant leur comportement. Pour Highway-Env, un algorithme DQN a été utilisé avec des améliorations comme le Prioritized Experience Replay et une grille d'observation redéfinie. Pour Racetrack, un algorithme PPO a été implémenté, intégrant des actions continue. Ce rapport détaille les architectures, les hyperparamètres, les récompenses et les améliorations spécifiques à chaque environnement.

Lien du Github :
https://github.com/guillfay/RL.git

Table des matières

Configuration N°1 : Highway

I. Présentation de l'environnement

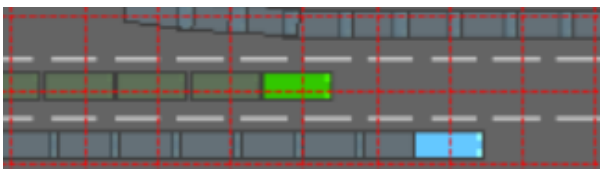
Etats et actions

L'environnement Highway-Env simule une autoroute à 4 voies où un agent (véhicule) doit naviguer tout en évitant les collisions et en respectant certaines contraintes. Les états sont représentés sous forme de grille, où chaque case contient des informations sur les véhicules environnants :

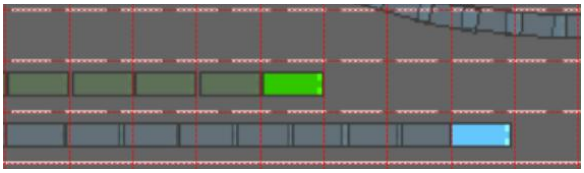
["presence", "x", "y", "vx", "vy", "cos_h", "sin_h"]

Dans ce projet, la grille a été redéfinie pour améliorer la perception de l'agent :

- Les cases de la grille ont été ajustées pour correspondre exactement à la taille du véhicule.
- L'agent est centré dans une case au centre de sa perception, avec un nombre égal de cases devant et derrière lui (4), ce qui améliore la symétrie de la perception. Il ne voit que la voit au-dessus et en dessous de lui afin de simplifier le problème.
  Cf schéma ci-dessous :



Perception initiale



Perception améliorée

- Les actions disponibles pour l'agent sont des commandes discrètes : aller sur la voie au-dessus ou en dessous, accélérer ou ralentir.

L'espace d'action est donc discret contrairement à l'espace des états qui est continue, ce qui convient bien à l'algorithme DQN utilisé.

Rewards

Un custom wrapper a été implémenté pour redéfinir les récompenses, offrant un contrôle total sur leur fonctionnement. Les récompenses incluent :

- Pénalité de collision : une forte pénalité est appliquée en cas de collision (-16 rare)
- Récompense pour rester sur la bonne voie : l'agent est encouragé à rester sur la voie de droite (0.05 à chaque pas de temps)
- Récompense pour une vitesse élevée : une récompense proportionnelle à la vitesse du véhicule, dans une plage définie (entre 0 et 0.9 à chaque pas de temps)

Ces récompenses ont été conçues pour guider l'agent vers un comportement sûr et efficace sur l'autoroute. Trouver les bonnes valeurs est tout un art et est loin d'être évident. Il faut faire attention à ne pas mettre trop de récompenses différentes pour ne pas perturber l'agent à excès.

II. Présentation de l'algorithme

Architecture MLP et présentation de l'algorithme

L'algorithme utilisé est une version modifiée du Deep Q-Network (DQN). L'architecture du réseau neuronal est un MLP (Multi-Layer Perceptron), adapté à l'espace d'observation discret de l'environnement. Le MLP est composé de :

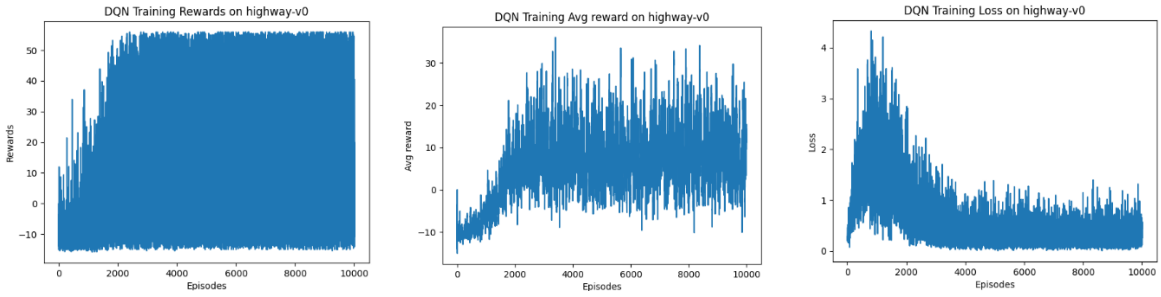
- Une couche d'entrée qui aplatit les observations (grille) en un vecteur.
- Deux couches cachées avec 512 et 128 neurones, utilisant des fonctions d'activation ReLU.
- Une couche de sortie qui prédit les valeurs Q pour chaque action possible.

Pour stabiliser l'apprentissage, une séparation entre le réseau principal (pour sélectionner les actions) et le réseau cible (pour évaluer les actions) a été mise en place. Cela évite que le réseau "poursuive sa propre queue" en ajustant ses prédictions sur des valeurs instables.
L'algorithme utilise une décroissance linéaire de l'exploration (epsilon), où l'agent commence par explorer largement (epsilon = 1.0) et réduit progressivement cette exploration jusqu'à un minimum de 0.1, garantissant une exploitation majoritaire tout en conservant une petite probabilité d'exploration.

Hyperparamètres

Plusieurs améliorations ont été apportées pour optimiser l'apprentissage :

- **Prioritized Experience Replay (PER) :**
  - Les transitions avec une grande erreur TD (Temporal Difference) sont priorisées, permettant à l'agent de se concentrer sur les expériences les plus informatives.
  - Une priorisation manuelle a été ajoutée pour accorder plus d'importance aux transitions avec des récompenses très faibles (par exemple, en cas de collision), afin d'apprendre efficacement des erreurs.
- **Normalisation des entrées :** Les observations (états) sont normalisées pour être comprises entre 0 et 1, ce qui améliore la stabilité de l'apprentissage et assure des premiers épisodes plus stables.
- **Normalisation progressive des récompenses** : Les récompenses sont centrées et réduites progressivement à l'aide d'un taux de décroissance (decay rate), garantissant une distribution stable des récompenses au fil du temps. On rappelle que les récompenses sont en plus pleinement maitrisées avec le wrapper définit plus haut.
- **Fonction de perte Smooth L1 Loss** : Cette fonction est utilisée pour limiter l'impact des grandes erreurs sur les gradients. Elle agit comme une MSE pour les petites erreurs et comme une L1 pour les grandes erreurs, ce qui stabilise l'apprentissage. La loss étaient parfois sinon exponentiellement croissante …
- **Clipping des gradients :** Les gradients sont limités entre -5 et +5 pour éviter des mises à jour excessives des poids du réseau, ce qui pourrait déstabiliser l'apprentissage.



Graphiques des rewards, rewards moyennés de manière glissante et loss de Huber

L'agent apprend effectivement à circuler dans son environnement en évitant les voitures. Les récompenses restent très disparates car l'agent se retrouvent souvent dans des situations où le choc est inévitable. Un exemple : il roule déjà à vitesse minimum, et les voitures devants lui roulent légèrement moins vite formant un « mur », le choc est inévitable.

III. Amélioration possible

- **Utilisation d'un CNN** : Bien qu'un CNN ait été testé sans succès en raison du faible nombre d'états, il pourrait être réévalué avec une architecture plus adaptée ou des états enrichis (par exemple, en ajoutant des informations spaciales).
- **Augmentation des données** : Jouez beaucoup de 3000 épisodes avec un cluster de calcul par exemple
- **Enrichir sa vision** : lui permettre de voir plus de voies.
- **Exploration plus sophistiquée :** Remplacer la décroissance linéaire d'epsilon par une stratégie plus avancée, comme epsilon-greedy adaptatif ou Boltzmann exploration, pour mieux équilibrer exploration et exploitation.
- **Enrichissement des récompenses** : Ajouter des récompenses pour des comportements plus complexes, comme maintenir une distance de sécurité ou anticiper les mouvements des autres véhicules.

Configuration N°2 : Racetrack

IV. Présentation de l'environnement

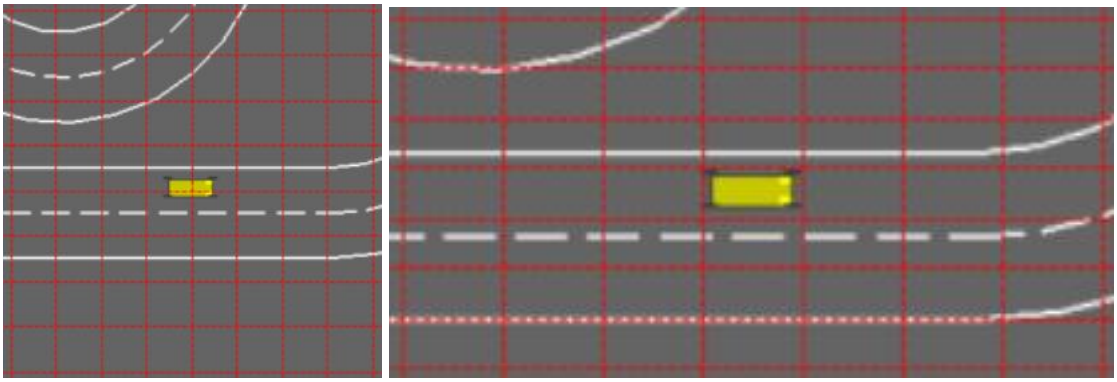
Etats et actions

L'environnement Racetrack simule une piste où un véhicule contrôlé par l'agent doit naviguer tout en évitant les collisions avec d'autres véhicules. Contrairement à l'environnement précédent, les actions sont continues, ce qui permet un contrôle plus précis du véhicule (ajuster l'angle de direction). Les états sont représentés sous forme de grille discrète, où chaque case contient des informations booléennes sur la présence d'autres véhicules et si la case est sur la route :

['presence', 'on_road']

Des modifications ont été apportées pour améliorer la perception de l'agent :

- La grille a été redéfinie pour que les cases aient exactement la taille du véhicule.
- L'agent est centré dans une case, avec un nombre égal de cases devant et derrière lui.



Perception initiale *Perception améliorée*

- Plusieurs véhicules (4) ont été ajoutés sur la piste pour rendre les rencontres avec d'autres voitures plus fréquentes, aidant ainsi notre agent à apprendre à esquiver les autres voitures.

Rewards

Un custom wrapper a été implémenté pour redéfinir les récompenses, offrant un contrôle total sur leur fonctionnement. Les récompenses incluent :

- Récompense positive pour éviter les collisions : un bonus est attribué à la fin de l'épisode si aucune collision n'a eu lieu (10, rare)
- Pénalité pour collision : une forte pénalité est appliquée en cas de collision (-30, rare)
- Pénalité pour actions trop significatives : des actions trop importantes (par exemple, des changements brusques de direction) entraînent une pénalité, encourageant des comportements plus fluides (-0.5 x le carré de l'action, à chaque pas de temps)
- Récompense pour rester sur la route : l'agent est encouragé à rester sur la piste (0.5, à chaque pas de temps)

Ces récompenses sont globalement positives pour guider l'algorithme, sauf en cas de collision ou d'actions excessives

V. Présentation de l'algorithme

Architecture MLP et présentation de l'algorithme

L'algorithme utilisé est une version modifiée de PPO (Proximal Policy Optimization), adaptée aux actions continues. Deux réseaux neuronaux distincts sont utilisés :

- **Actor :** génère les actions en sortie sous forme de moyenne et d'écart-type d'une distribution normale.
- **Critic :** estime la valeur des états pour calculer les avantages.

Les deux réseaux sont des MLP (Multi-Layer Perceptrons) avec la même architecture :

- Une couche d'entrée qui prend les observations aplaties.
- Deux couches cachées avec 128 neurones chacune, utilisant des fonctions d'activation ReLU.
- Une couche de sortie adaptée à chaque rôle :
    - L'actor produit une moyenne (et un écart-type appris séparément) pour chaque dimension de l'action.
    - Le critic produit une estimation scalaire de la valeur de l'état.

Pour stabiliser l'apprentissage, l'actor apprend plus lentement que le critic grâce à des **taux d'apprentissage différents** :

- **Actor : 1e-4** (réduit pour éviter des mises à jour trop rapides de la politique, ce qui pourrait déstabiliser l'optimisation).
- **Critic : 5e-4** (plus élevé pour permettre une estimation rapide et précise des valeurs des états).

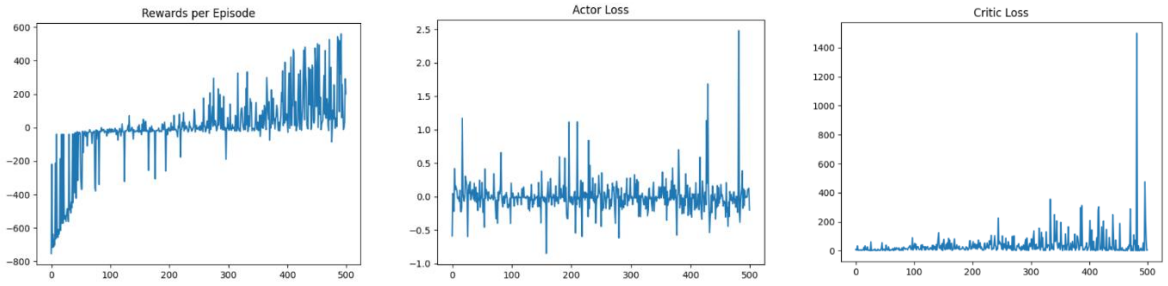
De plus, le **clipping des gradients** (entre -5 et +5) est utilisé pour éviter des mises à jour excessives des poids, ce qui pourrait entraîner des oscillations ou une divergence. Enfin, l'utilisation de **mini-batches** (taille 64) permet de réduire la variance des gradients et d'améliorer la convergence, tout en rendant l'entraînement plus efficace sur des ensembles de données volumineux. Ces choix garantissent un apprentissage stable et progressif dans un environnement complexe comme Racetrack.

Hyperparamètres

Plusieurs techniques ont été utilisées pour améliorer la stabilité et l'efficacité de l'apprentissage :

- **Loss MSE pour le critic** : La fonction de perte MSE (Mean Squared Error) est utilisée pour le critic, car elle est relativement stable dans notre simulation racetrack et adaptée à la régression. Elle permet simplement de minimiser le carré de l'écart entre les valeurs prédites et les retours calculés.
- **Mini-batches** : Les données sont divisées en mini-batches pour l'entraînement, ce qui réduit la variance des gradients et améliore la convergence. Chaque batch contient 64 transitions.
- **Clipping des gradients** : Les gradients sont limités entre -5 et +5 pour éviter des mises à jour excessives des poids, ce qui pourrait déstabiliser l'apprentissage.
- **Eps_clip** : Une contrainte sur la mise à jour des politiques est appliquée avec un facteur eps_clip = 0.1, empêchant des changements trop importants dans la politique entre deux itérations.

- **Espaces d'observation** : Bien que l'espace d'observation soit discret, il n'est pas encore assez grand pour justifier l'utilisation d'un CNN. Un MLP reste donc le choix optimal pour cet environnement.



Graphiques des rewards, loss de l'acteur et loss du critique

L'agent apprend effectivement à circuler dans son environnement en évitant toute sortie de route. Cependant, probablement par manque d'exemples durant son entrainement, il a du mal à éviter à chaque fois les collisions avec les autres voitures de la course. Les récompenses et la loss de l'acteur s'améliorent petit à petit, contrairement à la loss du critique qui semble être légèrement instable sur ces 500 épisodes. Cette instabilité peut s'expliquer par le fait que le critique doit prédire des valeurs de retour à long terme, qui sont plus difficiles à estimer précisément, surtout avec des données limitées et bruitées. 500 épisodes est faible, l'algorithme PPO est très coûteux en temps car il nécessite plusieurs passes sur des mini-batches pour optimiser à la fois l'acteur et le critique, contrairement à DQN qui effectue une seule mise à jour par batch.

VI. Amélioration possible

- **Augmentation de la grille d'observation** : Étendre la grille pour inclure davantage d'informations sur les véhicules environnants pourrait améliorer la prise de décision de l'agent.
- **Utilisation d'un CNN** : Si l'espace d'observation est enrichi, un CNN pourrait être testé pour exploiter les relations spatiales dans la grille.
- **Augmentation des données** : Jouez plus de 1000 épisodes avec un cluster de calcul par exemple
- **Récompenses plus complexes** : Ajouter des récompenses pour des comportements plus avancés, comme maintenir une distance de sécurité ou anticiper les mouvements des autres véhicules.
- **Exploration plus sophistiquée** : utiliser des techniques de curriculum learning pour complexifier progressivement l'algorithme en le faisant d'abord commencer sur une piste rectiligne avec d'autres véhicules.