Apprentissage par Renforcement avec DQN pour la Conduite Automatisée

Ammar Mariem , Bouhadida Malek , El Barhichi Mohammed

April 24, 2025

Abstract

Ce rapport présente l'implémentation d'un agent de conduite autonome dans un environnement autoroutier simulé en utilisant un Deep Q-Network (DQN). Le projet est divisé en plusieurs tâches, et ce rapport se concentre principalement sur la première tâche (Task 1), qui consiste à implémenter un DQN dans un environnement prédéfini. L'agent doit apprendre à naviguer sur une autoroute en évitant les collisions avec d'autres véhicules.

1 Introduction

L'objectif principal de ce projet est d'entraîner un agent de conduite autonome dans un environnement simulé à l'aide d'un algorithme de Reinforcement Learning (apprentissage par renforcement). Le projet a été divisé en plusieurs tâches, et l'objectif principal est de développer un agent capable de conduire en toute sécurité en apprenant par essais et erreurs.

L'agent interagit avec l'environnement pour maximiser les récompenses obtenues en fonction de son comportement. Il apprend à éviter les collisions, à maintenir une vitesse optimale et à changer de voie pour éviter les obstacles. Ce projet utilise l'algorithme Deep Q-Network (DQN) pour approximier la fonction de valeur d'action.

2 Task 1 - Implémentation d'un DQN dans un environnement autoroutier

2.1 Objectifs de la Task 1

La première tâche de ce projet était d'implémenter un DQN dans un environnement autoroutier simulé. L'agent devait apprendre à prendre des décisions pour éviter les collisions avec d'autres véhicules tout en maximisant la vitesse. L'environnement utilisé est une simulation où plusieurs véhicules circulent de manière autonome, et l'agent doit interagir avec cet environnement pour éviter les collisions tout en respectant les règles de circulation.

2.2 Étapes de l'implémentation

Pour implémenter l'agent avec le Deep Q-Network (DQN), j'ai suivi ces étapes principales :

1. **Création de l'environnement** : L'agent évolue dans un environnement autoroutier simulé, avec des informations sur la position, la vitesse et l'orientation des véhicules. Ces observations servent à déterminer les actions à prendre, telles que accélérer, changer de voie ou ralentir.

2. **Implémentation du DQN** : J'ai conçu un réseau de neurones pour générer des Q-values représentant la qualité de chaque action dans un état donné, en utilisant les observations de l'environnement.

3. **Mise à jour de la fonction Q par Q-learning** : Le DQN utilise le Q-learning pour mettre à jour les Q-values et maximiser les récompenses futures attendues. La mise à jour des Q-values suit la règle suivante :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right)$$

   Où :

   - s_t : état actuel
   - a_t : action actuelle
   - r_t : récompense obtenue après avoir pris l'action
   - s_{t+1} : nouvel état
   - γ : facteur de réduction des récompenses futures
   - α : taux d'apprentissage

   Cette règle permet à l'agent de corriger ses Q-values en fonction des récompenses immédiates et futures, afin qu'elles convergent vers les valeurs correctes au fil de l'entraînement.

4. **Mémoire de replay** : La mémoire de replay stocke les transitions (état, action, récompense, nouvel état). Cela permet à l'agent d'échantillonner des expériences passées de manière aléatoire pour stabiliser l'apprentissage.

5. **Stratégie epsilon-greedy** : L'agent commence par explorer (epsilon élevé), puis exploite ses connaissances au fur et à mesure de l'entraînement en réduisant epsilon.

6. **Entraînement et mise à jour des poids** : L'agent met à jour ses Q-values en fonction des récompenses obtenues et de la Q-value cible, calculée après chaque transition.

7. **Mise à jour du réseau cible** : Un réseau cible, copie du réseau principal, est mis à jour périodiquement pour éviter des changements trop rapides dans les Q-values cibles pendant l'entraînement.

3 Ajout de la visualisation pendant l'entraînement

Pour observer l'évolution de l'agent pendant l'entraînement, j'ai utilisé la méthode `render()` de l'environnement simulé. Cela permet de visualiser l'état de l'environnement à chaque étape de l'entraînement. L'agent (représenté par la voiture verte) interagit avec l'environnement et prend des décisions pour éviter les collisions avec d'autres véhicules (en bleu et en gris).

L'image ci-dessous montre une visualisation typique de l'environnement pendant l'entraînement, où l'agent est en train d'évoluer parmi les autres véhicules.

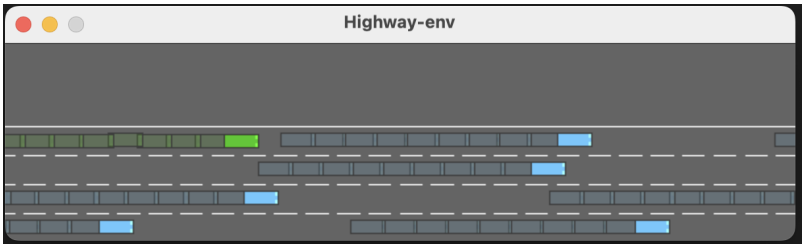

Figure 1: Visualisation de l'environnement pendant l'entraînement. L'agent (vert) évolue sur l'autoroute parmi d'autres véhicules (bleus).

4 Résultats de l'entraînement

L'agent a été entraîné pendant un total de 5000 épisodes. À chaque épisode, l'agent a sélectionné des actions en fonction de sa politique epsilon-greedy et mis à jour son modèle en fonction des récompenses reçues. Les résultats sont les suivants :

4.1 Graphiques des performances

Les performances de l'agent au fil des épisodes d'entraînement ont été suivies à l'aide de courbes montrant l'évolution de la récompense et de la perte. Voici un exemple des graphiques générés lors de l'évaluation des performances de l'agent après chaque période d'entraînement.

4.1.1 Évolution de la récompense

Au fur et à mesure de l'entraînement, l'agent a montré une amélioration significative de ses performances, avec des récompenses cumulées qui augmentent au fur et à mesure des épisodes. L'agent a appris à éviter les collisions et à rester sur la voie correcte tout en maintenant une vitesse optimale.

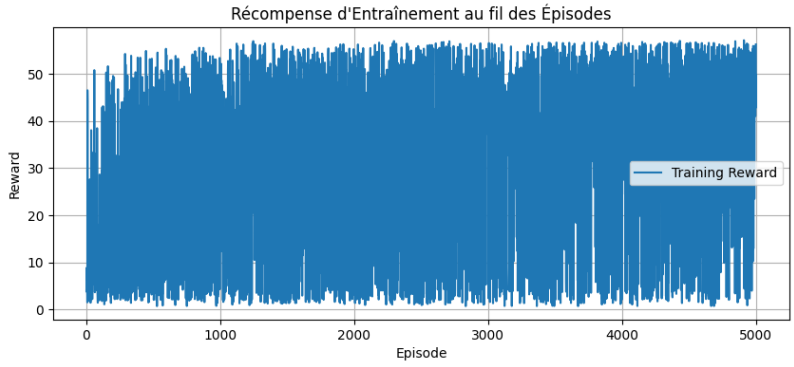

Figure 2: Récompense d'Entraînement au fil des Épisodes

4.1.2 Courbe de l'Epsilon

La première courbe montre l'évolution de l'epsilon, qui régule le taux d'exploration et d'exploitation de l'agent au fil du temps. Au début de l'entraînement, epsilon est élevé, ce qui favorise l'exploration. À mesure que l'agent apprend, epsilon diminue progressivement pour favoriser l'exploitation des connaissances acquises.

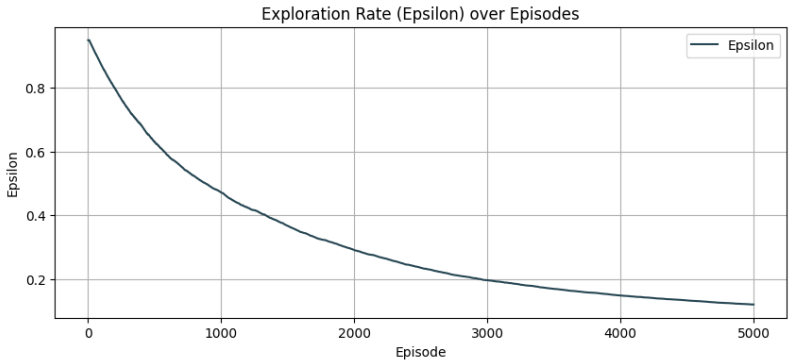

Figure 3: Taux d'exploration (Epsilon) au fil des épisodes

4.1.3 Courbe de la Perte

La deuxième courbe montre l'évolution de la perte pendant l'entraînement. La perte est élevée au début, mais elle diminue progressivement à mesure que l'agent apprend à mieux estimer les valeurs Q. Les pics observés dans les premiers épisodes peuvent être causés par des fluctuations dans les gradients pendant l'apprentissage.

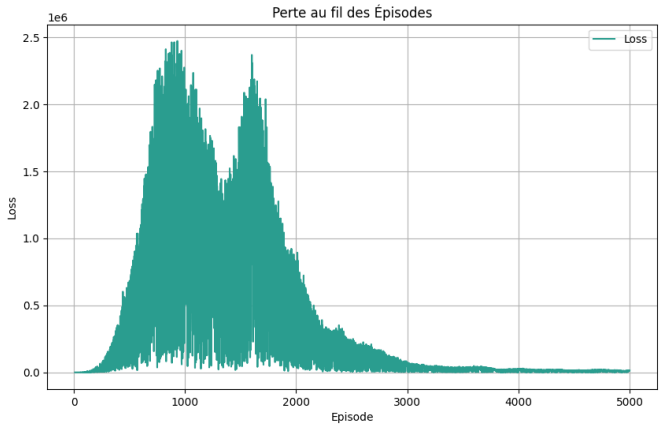

Figure 4: Perte au fil des épisodes

4.1.4 Courbe de la Récompense Moyenne en Évaluation

La dernière courbe montre l'évolution de la récompense moyenne pendant l'évaluation de l'agent. Ces évaluations sont effectuées tous les 10 épisodes pour suivre la performance de l'agent sur des épisodes non vus lors de l'entraînement.

Commentaire :

La courbe montre une performance relativement stable de l'agent, avec des fluctuations qui indiquent une robustesse partielle du modèle. Les oscillations suggèrent que bien que l'agent ait appris une politique efficace, il pourrait encore bénéficier d'améliorations dans l'ajustement des hyperparamètres ou par l'utilisation de techniques avancées comme Double DQN pour renforcer sa stabilité et sa robustesse.

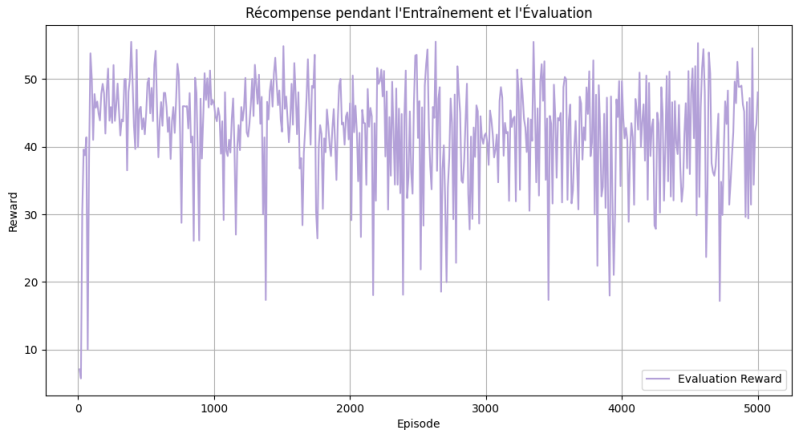

Figure 5: Récompense moyenne en évaluation au fil des épisodes

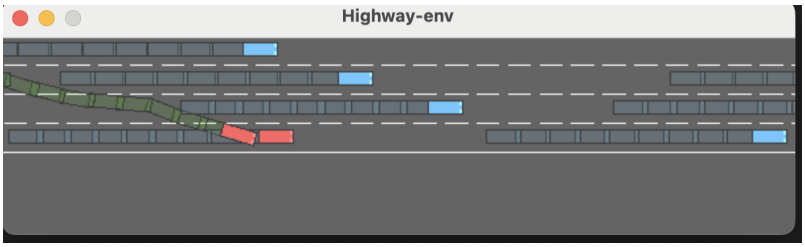

Figure 6: Visualisation de l'agent pendant l'entraînement, montrant une collision lors d'un changement de voie

Analyse supplémentaire :

Bien que l'agent montre une bonne robustesse générale, il n'est pas parfait. Une des limitations observées lors de l'entraînement est que l'agent rencontre parfois des collisions, surtout lorsqu'il change de voie pour doubler d'autres véhicules. Ces collisions surviennent lorsque l'agent n'anticipe pas correctement la position des véhicules voisins ou effectue un changement de voie trop brusque. Cela suggère que bien que l'agent ait appris une politique relativement robuste, il existe des situations où son comportement pourrait être amélioré. Par exemple, l'algorithme pourrait être affiné pour mieux gérer les moments critiques où un changement de voie est nécessaire tout en évitant les collisions.

5 Conclusion

Dans cette tâche, j'ai réussi à implémenter un agent de conduite autonome en utilisant l'algorithme Deep Q-Network. L'agent a montré une amélioration de ses performances au fil de l'entraînement, apprenant à éviter les collisions et à maintenir une vitesse optimale. Le projet peut désormais passer à l'étape suivante, où l'on pourrait explorer des environnements avec des actions continues ou utiliser des techniques plus avancées comme Double DQN ou Dueling DQN.