

# Rapport Apprentissage par Renforcement

Emile Descroix

Février 2026

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Environnement choisi et modifications</b>	<b>3</b>
<b>3</b>	<b>Deep Q-Learning</b>	<b>3</b>
3.1	Commentaires sur l'implémentation de l'algorithme . . . . .	4
3.2	Choix des hyperparamètres . . . . .	4
3.3	Résultats du modèle final . . . . .	5
<b>4</b>	<b>Proximal Policy Optimization</b>	<b>5</b>
4.1	Description de l'algorithme . . . . .	5
4.2	Choix des hyperparamètres . . . . .	6
<b>5</b>	<b>Résultats finaux</b>	<b>7</b>

## 1 Introduction

Ce rapport présente les algorithmes utilisés ainsi que les expériences réalisées pour implémenter l'entraînement d'un agent pour l'environnement "Car Racing" du module `gymnasium`. Les algorithmes ont été recodés "from scratch", le code ainsi qu'une vidéo présentant le résultat d'un agent entraîné sont disponibles sur le répertoire suivant : <https://github.com/Emiile9/ProjetRL>. Ce rapport se concentre sur les points d'attention de l'implémentation, les résultats empiriques pour différentes valeurs d'hyperparamètres et la discussion de la pertinence de l'utilisation des algorithmes dans ce cas précis.

## 2 Environnement choisi et modifications

L'environnement choisi est celui d'une voiture de course qui doit parcourir un circuit le plus rapidement possible. Les observations sont reçues sous la forme d'une image 96x96 au format RGB. Les actions peuvent être discrètes : 0 pour ne rien faire, 1 pour tourner vers la droite, 2 vers la gauche, 3 pour accélérer et 4 pour ralentir ou bien continues : une valeur entre -1 et 1 pour la direction 1 pour accélérer et 0 pour freiner. A chaque frame on retire 0.1 au reward et chaque nouvelle tuile du circuit visitée apporte  $\frac{1000}{N}$  où N est le nombre total de tuiles.

Pour faciliter l'apprentissage, les images sont modifiées grâce aux transformations suivantes :

- **MaxAndSkipObservation** permet de répéter une action et de diminuer le nombre de frames vu par le réseau, cela rend le mouvement plus compréhensible et réduit le temps de calcul nécessaire.
- **GrayscaleObservation** permet de convertir les images de la couleur vers une échelle de gris et simplifie l'information pour le réseau.
- **ResizeObservation** permet de conserver seulement la partie centrale de l'image, cela retire les informations superflues du bord de l'image et ne conserve que l'essentiel.
- **FrameStackObservation** permet d'envoyer plusieurs images au réseau et de traduire le mouvement de la voiture plutôt que d'envoyer des images fixes.

## 3 Deep Q-Learning

Le premier algorithme à implémenter et tester est celui de *Deep Q-Learning*, nous verrons dans un premier temps comment répondre aux différentes problématiques rencontrées lors de son implémentation puis nous comparerons les résultats pour plusieurs valeurs d'hyperparamètres. On utilise dans cette partie un ensemble d'actions discret décrit plus tôt.

### 3.1 Commentaires sur l'implémentation de l'algorithme

La politique suivie est une politique  $\epsilon$ -greedy où  $\epsilon$  décroît de manière linéaire, nous discuterons ensuite de la vitesse et de l'ampleur de cette décroissance.

Avant d'entraîner le réseau il faut d'abord regrouper assez d'exemples, on utilise une file dans laquelle on enregistre l'ensemble des observations : état, action, reward, état suivant et terminal ou non.

Pour assurer la stabilité lors des mises à jour, l'algorithme utilise deux modèles, un réseau cible qui sert d'objectif d'apprentissage et un réseau "policy" que l'on utilise pour sélectionner les actions pendant l'entraînement et que l'on met à jour après chaque action. Tous les  $n$  épisodes on copie les poids du réseau "policy" vers le réseau cible.

### 3.2 Choix des hyperparamètres

Dans un premier temps on explore les différentes possibilités pour différents learning rate. On obtient les résultats suivants. On constate que le learning rate le plus faible,  $lr = 0.001$  semble moins bien apprendre l'environnement. Les courbes de progression sont similaires mais le modèle est moins bon après les premiers épisodes. Pour les autres valeurs, les résultats sont très similaires, on conserve un learning rate de 0.0005 pour la suite des expérimentations. On

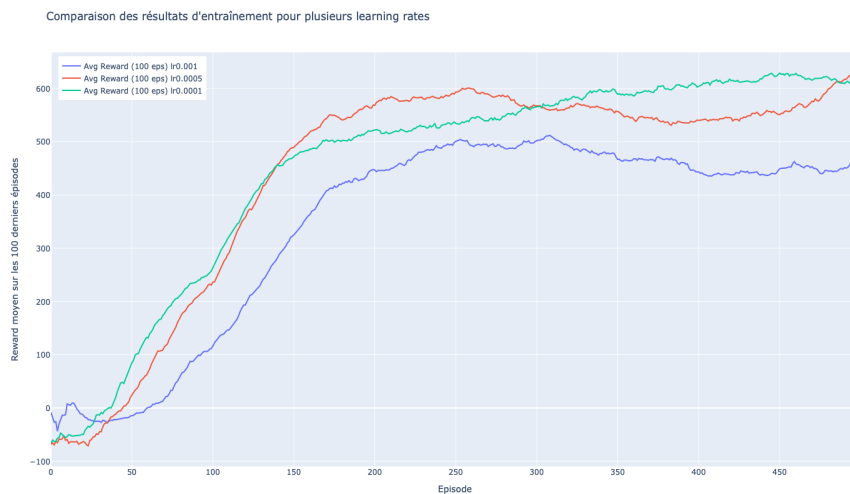


Figure 1: Evolution de la récompense moyenne sur les 100 derniers épisodes pour différentes valeurs de learning rate.

compare maintenant l'effet du paramètre  $\epsilon$ , qui contrôle l'exploration de l'agent. On conserve une mise à jour linéaire ou après chaque action on actualise la valeur d'epsilon de la manière suivante :  $\epsilon \leftarrow \epsilon - \frac{\epsilon_{max} - \epsilon_{min}}{k}$  où  $\epsilon_{max} = 0.9$  et  $\epsilon_{min} = 0.01$  et où l'on teste plusieurs valeurs de  $k$  : 1000, 10000 et 100000. Les résultats

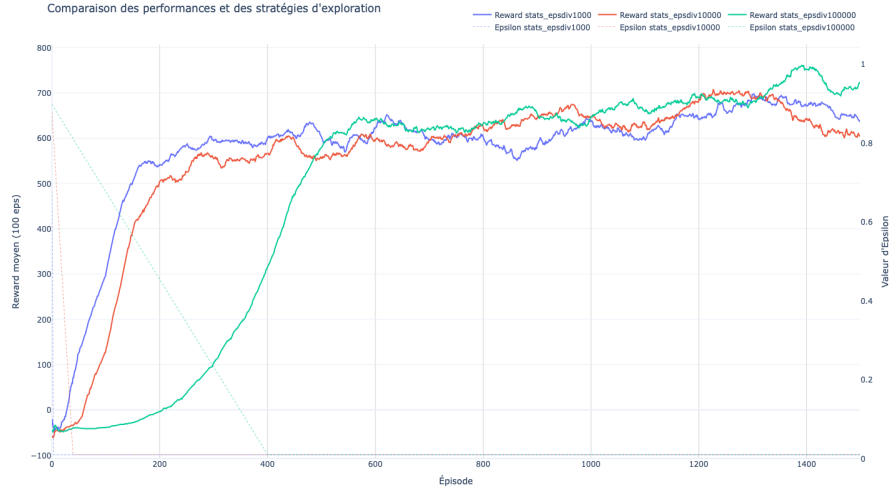


Figure 2: Evolution de la récompense moyenne sur les 100 derniers épisodes pour différentes stratégies d'exploration

sont ceux attendus, dans le cas où epsilon décroît plus rapidement l'agent apprend plus lentement mais semble légèrement mieux performer après un grand nombre d'épisodes. Il y a donc un choix à faire entre la performance et la durée de l'entraînement. Si nous voulons un agent très performant et que nous avons le temps et les ressources de calcul nous pouvons conserver un epsilon élevé plus longtemps. IL est également possible de modifier le nombre d'épisodes entre deux copies de poids vers le réseau cible ou encore de raffiner la taille du buffer qui contient les observations de l'agent mais ces paramètres ne semblent pas impacter significativement la performance ou le temps d'entraînement de l'agent.

### 3.3 Résultats du modèle final

Pour ce modèle final on remarque une forte progression pendant les 500 premiers épisodes d'entraînement avant de stagner. Un point d'amélioration pourrait-être la variance des récompenses, même après près de 1500 épisodes l'agent obtient parfois moins de 200 et peut performer jusqu'à 900.

## 4 Proximal Policy Optimization

### 4.1 Description de l'algorithme

La politique suivie est une politique stochastique directement paramétrée par le réseau de neurones : à chaque état, l'acteur produit une distribution de proba-

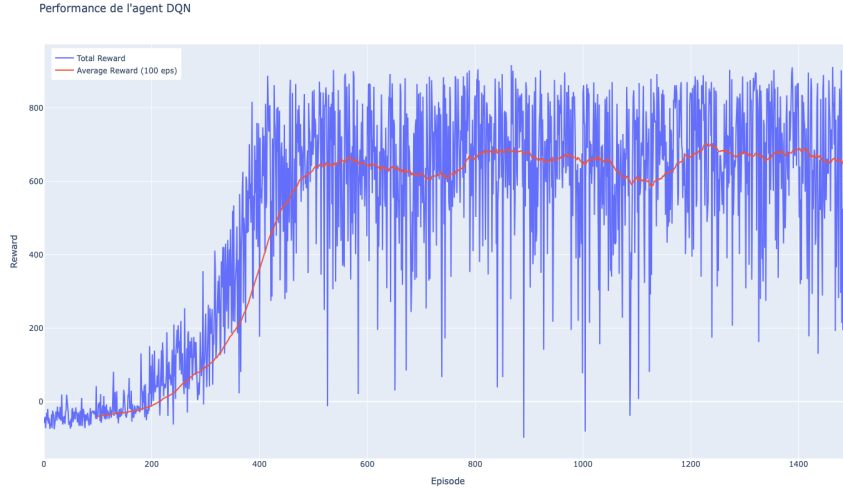


Figure 3: Evolution de la récompense par épisode durant l’entraînement d’un agent à partir de l’algorithme Deep Q-Network

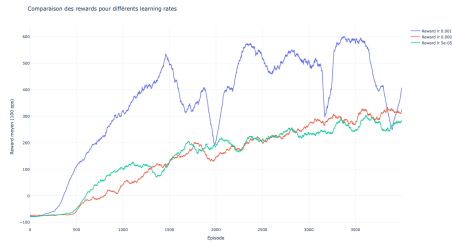
bilité sur les actions, depuis laquelle on échantillonne. L’exploration est assurée naturellement via le terme d’entropie dans la fonction objectif, dont le coefficient décroît progressivement pour passer d’une phase d’exploration vers une phase d’exploitation. Plutôt que d’apprendre action par action, PPO collecte des trajectoires complètes de longueur fixe  $n$  avant chaque mise à jour : pour chaque transition on enregistre l’état, l’action, la récompense, ainsi que la log-probabilité de l’action et l’estimation de valeur produite par le critique. Ces données servent à calculer les avantages via l’estimation GAE, qui combine les retours à court et long terme grâce aux paramètres  $\gamma$  et  $\lambda$ .

Pour assurer la stabilité des mises à jour, PPO ne maintient pas deux réseaux distincts mais contraint directement l’amplitude du changement de politique à chaque update : le ratio entre la nouvelle politique et l’ancienne est clippé dans l’intervalle  $[1-\epsilon, 1+\epsilon]$ , empêchant toute mise à jour trop agressive. Les données collectées sont réutilisées pendant  $K$  epochs consécutives avant d’être jetées, ce qui distingue PPO des méthodes on-policy pures.

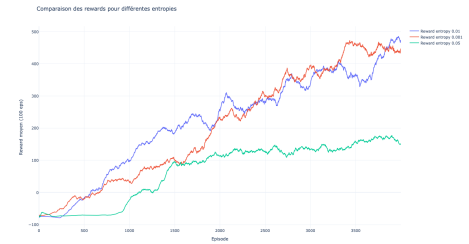
## 4.2 Choix des hyperparamètres

On teste d’abord plusieurs taux d’apprentissage différents (voir Fig. 4(a)). On remarque d’importantes différences, un learning rate plus faible semble apprendre plus rapidement et obtenir de meilleurs résultats. Néanmoins l’apprentissage est très instable du aux mises à jour plus brues et les performances peuvent chuter drastiquement.

On peu aussi agir sur l’exploration de l’agent, en changeant la valeur du coefficient d’entropie. En effet un coefficient d’entropie plus élevé pénalise les poli-



(a) Evolution de la récompense PPO.



(b) Comparaison des récompenses pour différents coefficients d'entropie.

Figure 4: Comparaison des récompenses pour différents hyperparamètres (moyenne sur 100 épisodes).

tiques certaines et pousse l'agent à explorer. Le graphique présente l'évolution de la récompense moyenne en fonction de la valeur de ce coefficient d'entropie (voir Fig.4 (b)).

Ce graphique illustre le dilemme exploration - exploitation, le coefficient d'entropie le plus haut, en vert, encourage fortement l'exploration et ne permet pas à l'agent de converger. A l'opposé le plus faible, en rouge ralentit l'apprentissage.

## 5 Résultats finaux

Enfin on peut comparer les deux méthodes, sur un entraînement de 8000 épisodes on remarque des progressions complètement différentes. L'agent entraîné avec DQN progresse très vite mais stagne après 500 épisodes. A contrario, l'agent ppo progresse de manière régulière et continue de s'améliorer même après plusieurs milliers d'épisodes.

Si l'on regarde les récompenses en détails on observe également une différence majeure. L'agent PPO semble plus régulier, la récompense à une variance plus faible. Comme évoqué plus tôt, lors de l'entraînement l'agent DQN continue d'obtenir des récompenses négative. A l'inverse avec le PPO la récompense est plus faible mais plus régulière.

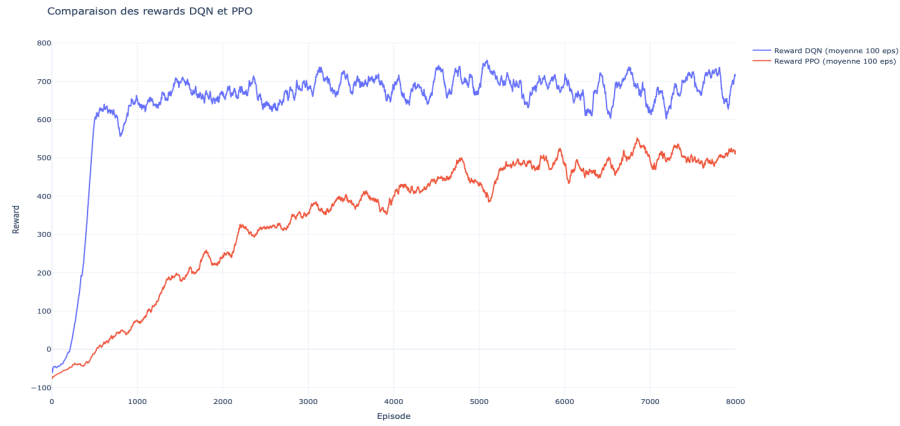
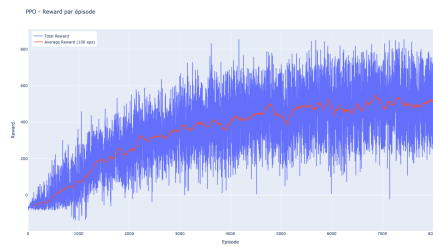
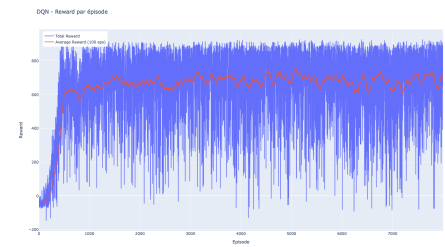


Figure 5: Comparaison des récompenses pour l'entraînement avec DQN et PPO



(a) Evolution de la récompense PPO.



(b) Evolution de la récompense DQN.

Figure 6: Comparaison des récompenses par épisode DQN vs PPO.