CentraleSupélec

Reinfocement Learning tâche 2 - Continuous action

Students:

Marc-César Garcia-Grenet
marc-cesar.garcia-grenet@student-cs.fr

April 2025

1 Description du problème `racetrack-v0`

L'environnement `racetrack-v0` est une simulation de conduite autonome proposée dans la suite `highway-env`. Il représente un circuit routier où un véhicule contrôlé par l'agent doit progresser le long d'une piste sans sortir des limites et en évitant des obstacles potentiels. Le but est d'atteindre la fin du circuit dans un laps de temps limité.

Le problème est formulé comme une tâche de contrôle continu : l'agent choisit à chaque pas de temps une action dans un espace d'action *continu* représentant principalement l'accélération latérale, et, dans certaines variantes, également l'accélération longitudinale. L'observation est constituée d'une grille spatiale de caractéristiques (comme la vitesse, la présence de voisins, etc.) qui encode la perception de l'environnement autour du véhicule.

Ce problème illustre bien les défis typiques du renforcement continu : nécessité d'exploration efficace, sensibilité aux instabilités dynamiques, rareté des récompenses (l'objectif n'est atteint qu'en fin de parcours), et nécessité de généraliser des stratégies sur des trajectoires variées et non triviales.

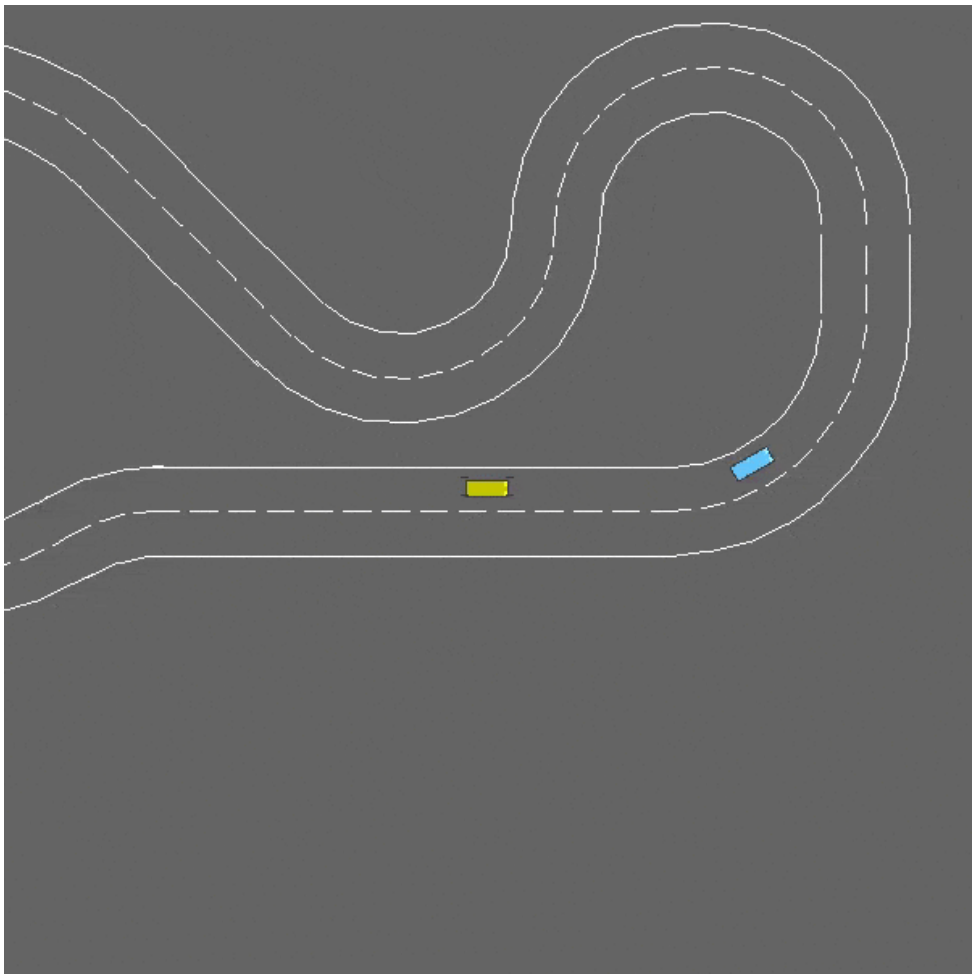


Figure 1: Environnement racetrack-v0 avec la configuration basique

1.1 Spécificités de la configuration utilisée

L'environnement a été configuré de manière précise pour maximiser la difficulté et refléter des conditions réalistes :

- **Contrôle latéral continu** : l'agent pilote directement l'accélération latérale.

- **Observation sous forme de grille d'occupation** : l'état est représenté par une grille 12×12, contenant trois canaux :

  1. Présence des véhicules voisins.
  2. Vitesse relative des véhicules.
  3. Accélération relative des véhicules.

  Cette observation est alignée avec l'axe du véhicule contrôlé pour simplifier l'apprentissage d'actions indépendantes de l'orientation absolue.

- **Durée d'épisode limitée** : chaque épisode est restreint à 120 étapes temporelles, imposant un délai pour atteindre l'objectif.

- **Rewards** :

  – Progression sur le circuit : $+8$ par unité avancée.
  – Coût d'écart au centre de la voie : pénalité de -2 ou -4 selon la configuration.
  – Pénalités pour collision : -1 ou -10.
  – Pénalité légère sur chaque action exécutée : -0.05.

- **Présence d'adversaires** : trois véhicules adverses sont générés aléatoirement.

- **Rendu graphique** : l'environnement est rendu en mode `rgb_array` pour la collecte de séquences visuelles, et en mode `human` pour la visualisation après entraînement.

- **Exploration initiale** : dans SAC, une phase d'exploration purement aléatoire est imposée durant les 5 000 premières étapes pour diversifier l'expérience enregistrée dans le buffer.

- **Curriculum learning** : la difficulté est progressivement augmentée en rajoutant progressivement plus de véhicules adverses au fil de l'entraînement pour stabiliser la montée en compétence de l'agent.

Cette configuration rend l'environnement particulièrement exigeant pour un agent d'apprentissage par renforcement et explique en partie les résultats assez médiocres de l'agent à la réalisation de la tâche.

2 Implémentation de l'algorithme PPO

Avant d'opter pour l'approche SAC, j'ai développé une solution basée sur l'algorithme **Proximal Policy Optimization** (PPO), un algorithme **on-policy** qui stabilise l'apprentissage en limitant la modification de la politique entre les mises à jour. PPO utilise un ratio de probabilité tronqué pour empêcher des sauts trop brutaux dans l'espace des politiques.

Mon agent PPO est entraîné avec les techniques suivantes :

- **Generalized Advantage Estimation (GAE)** pour réduire la variance des estimations d'avantages tout en contrôlant le biais,
- **Régularisation par entropie** pour encourager l'exploration,
- **Entraînement par mini-lots** et **plusieurs époques d'optimisation** à partir de la même trajectoire.

Hyperparamètres utilisés

- Facteur d'actualisation : $\gamma = 0.99$
- Coefficient GAE : $\lambda = 0.95$
- Ratio de clip : `clip_ratio` $= 0.2$
- Nombre d'époques de mise à jour : `update_epochs` $= 4$
- Taille des mini-lots : `minibatch_size` $= 32$
- Coefficient d'entropie : `entropy_coef` $= 0.02$
- Taux d'apprentissage : `lr_actor` $=$ `lr_critic` $= 3 \times 10^{-4}$
- Taille cachée des réseaux : `hidden_size` $= 512$
- Nombre d'épisodes d'entraînement : 150
- Fréquence d'évaluation : tous les 10 épisodes

Environnement de simulation

L'environnement utilisé est `racetrack-v0` du package `highway-env`, configuré spécifiquement :

- Actions continues latérales (accélération longitudinale désactivée),
- Observation basée sur une grille d'occupation (12×12) avec les attributs `presence`, `velocity` et `acceleration`,
- Progression récompensée (+8), centrage sur la voie pénalisé (-2), collisions sévèrement pénalisées (-10),
- Curriculum learning : augmentation progressive du nombre de véhicules adverses durant l'entraînement.

Architecture des réseaux

- **Acteur** :
  - Deux couches convolutives ($3 \to 16 \to 32$ canaux, noyaux 3×3) avec ReLU,
  - Deux couches fully-connected ($512 \to 512$) avec ReLU,
  - Deux sorties : moyenne et log-écart-type de la distribution normale multivariée des actions,
  - Clamping du log-écart-type dans l'intervalle $[-20, 2]$ pour éviter des extrêmes numériquement instables,
  - L'action est échantillonnée à partir de cette distribution normale puis bornée dans l'intervalle admissible.
- **Critique** :
  - Deux couches fully-connected ($512 \to 512$) avec ReLU et normalisation de couche (LayerNorm),
  - Une sortie scalaire représentant la valeur $V(s)$ de l'état.

Détails de l'entraînement

L'entraînement suit un protocole rigoureux :

- Collecte de rollouts de 1024 pas temporels,
- Calcul des avantages avec GAE,
- Normalisation des avantages,
- Optimisation séparée de l'acteur (objectif PPO clip + entropie) et du critique (perte MSE sur la valeur),
- Sauvegarde du meilleur modèle selon la moyenne de récompenses d'évaluation.

Avantages et limites observés

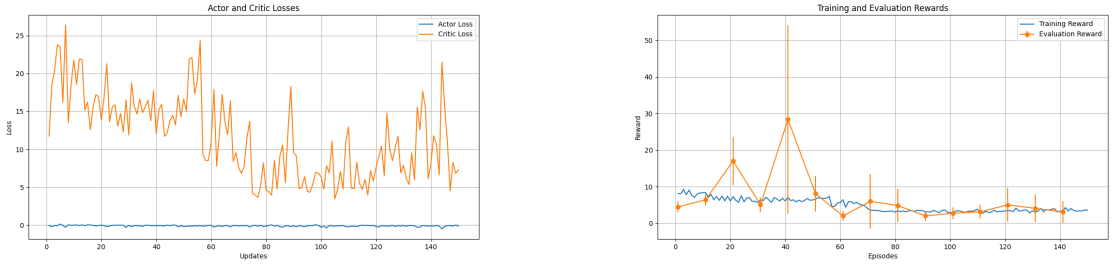
Malgré la robustesse théorique de PPO, plusieurs difficultés spécifiques sont apparues dans cet environnement :

- **Stabilité limitée** : l'agent tend à quitter la piste prématurément ou à effectuer des mouvements circulaires.
- **Lenteur de l'apprentissage** : Plusieurs minutes de calculs étaient nécessaires pour entrainer le modèle, on aurait pu envisager un apprentissage sur plusieurs environnement en parallèle.
- **Curriculum learning** : a permis d'améliorer progressivement la robustesse face aux véhicules adverses, sans résoudre entièrement l'ensemble des problèmes rencontrés.

Résultats PPO

La learning curve de notre modèle est assez spécifique, on voit que la plus grande erreur provient du critic à cause de l'utilisation de la MSE loss pour ce dernier. On voit que l'apprentissage est lent et bruité ce qui se ressent dans les résultats finaux.

On voit que les trainings rewards et evaluation rewards baissent légèrement au cours du temps ce qui signifie que notre modèle n'arrive pas à trouver des manières de répondre à la consigne qui lui sont données à travers les récompenses.



(a) Pertes d'entraînement pour PPO

(b) Rewards d'entraînement pour PPO

Figure 2: Évolution des métriques d'entraînement pour PPO

Conclusion PPO

PPO a démontré une capacité à compléter certaines trajectoires sans collisions. Les performances restaient extrêmement variables même après 150 épisodes, avec de nombreuses sorties de piste et collisions.

Malgré des améliorations progressives grâce au curriculum learning et au réglage d'hyperparamètres, l'agent n'était pas en mesure de terminer un tour. En fait on a un effondrement de la policy et si l'on essaie de visualiser le modèle final on voit que l'agent tourne et sort de la piste des les premiers instants de la simulations à chaque fois. Au contraire pour le meilleur modèle, les résultats sont très variables et l'agent arrive parfois à suivre une partie de la piste.

3 Implémentation de l'algorithme Soft Actor-Critic (SAC)

L'algorithme **Soft Actor-Critic** (SAC) est un algorithme off-policy moderne, conçu pour l'apprentissage dans des environnements continus complexes. Il repose sur l'optimisation simultanée de deux objectifs :

- Maximiser le retour espéré (expected return) via une politique stochastique.
- Maximiser l'entropie de la politique pour encourager une exploration persistante, réduisant ainsi le risque de convergence prématurée.

Ces caractéristiques font de SAC un choix particulièrement pertinent pour notre tâche de conduite autonome sur circuit.

Particularités du SAC utilisé

L'agent SAC implémente :

- Un **twin Q-network** pour l'évaluation critique, réduisant le biais positif de la fonction de valeur.
- Un **réglage automatique de l'entropie** (α), afin d'ajuster dynamiquement l'équilibre entre exploration et exploitation.
- Une **stratégie off-policy**, facilitée par un `replay buffer` pour l'apprentissage à partir d'expériences passées.
- Un **shaping des récompenses** et une **normalisation en ligne des observations** pour stabiliser l'entraînement.

Hyperparamètres utilisés

- $\gamma = 0.99$: facteur de discount valorisant les récompenses futures.
- $\tau = 0.005$: coefficient de mise à jour douce (soft update) des réseaux cibles.
- $\alpha = 0.2$: coefficient initial d'entropie (peut être adapté automatiquement).
- $learning\_rate = 3 \times 10^{-4}$: taux d'apprentissage partagé par l'acteur, les critiques et le réglage d'entropie.
- $batch\_size = 256$: taille des mini-lots pour l'entraînement.
- $buffer\_size = 10^6$: capacité du `ReplayBuffer`.
- $max\_episodes = 1500$: nombre maximal d'épisodes d'entraînement.
- $max\_steps = 100$: nombre maximal de pas temporels par épisode.
- $initial\_random\_steps = 5000$: nombre de pas initiaux d'exploration aléatoire avant apprentissage.
- $updates\_per\_step = 1$: une mise à jour par interaction environnementale.
- $eval\_interval = 50$: fréquence d'évaluation du modèle.

Architecture des réseaux

- **Acteur** :
  - Trois couches convolutionnelles ($3 \to 32 \to 64 \to 64$ canaux) avec ReLU.
  - Deux couches fully-connected de taille 256.
  - Production de la moyenne et du log-écart-type d'une distribution normale multivariée.
  - Application d'une fonction `tanh` sur l'échantillon pour borner les actions entre $[-1, 1]$.
  - Calcul du log-probabilité ajusté par la transformation `tanh`.
- **Critiques** (`Q1` et `Q2`) :
  - Deux réseaux jumeaux, chacun recevant la concaténation de l'état et de l'action.
  - Deux couches convolutionnelles pour encoder l'état.
  - Deux couches fully-connected avec normalisation de couche (LayerNorm) dans chaque réseau.
  - Prédiction indépendante des valeurs $Q_1(s, a)$ et $Q_2(s, a)$.

Apprentissage et stratégie d'entraînement

Le protocole d'apprentissage inclut :

- **Exploration initiale aléatoire** sur 5 000 étapes pour remplir le buffer.
- **Curriculum learning** : introduction progressive de véhicules adverses agressifs selon un calendrier d'épisodes.
- **Optimisation alternée** :
  - Mise à jour du critique par minimisation de la perte MSE entre prédiction Q et cible Bellman.
  - Mise à jour de l'acteur pour maximiser Q tout en tenant compte de l'entropie.
  - Ajustement automatique d'α par optimisation d'une fonction de perte spécifique si nécessaire.
- **Soft update** des réseaux cibles après chaque mise à jour.

Replay Buffer

L'agent utilise un **ReplayBuffer** FIFO (First-In, First-Out) contenant les tuples $(s, a, r, s', \mathit{done})$:

- Échantillonnage aléatoire des transitions pour la mise à jour des réseaux.
- Buffer de grande capacité (10^6 transitions) pour capturer une grande diversité d'expériences.

Une amélioration envisagée serait l'intégration d'un **Prioritized Experience Replay (PER)**.

Analyse des résultats

L'entraînement sur 1 500 épisodes révèle :

- Une **augmentation progressive** de la récompense moyenne par épisode.
- Une **hausse significative du taux de réussite** après introduction du curriculum learning.
- Une **stabilité d'apprentissage** même en présence d'agents adverses.

Néanmoins ces résultats restent à nuancer en effet l'agent apprend relativement lentement et garde un success_rate relativement faible. Il est possible que l'agent eut été meilleur en continuant davantages l'entrainement mais la vitesse très élevée de la voiture rend quasiment impossible la conduite sur la piste.

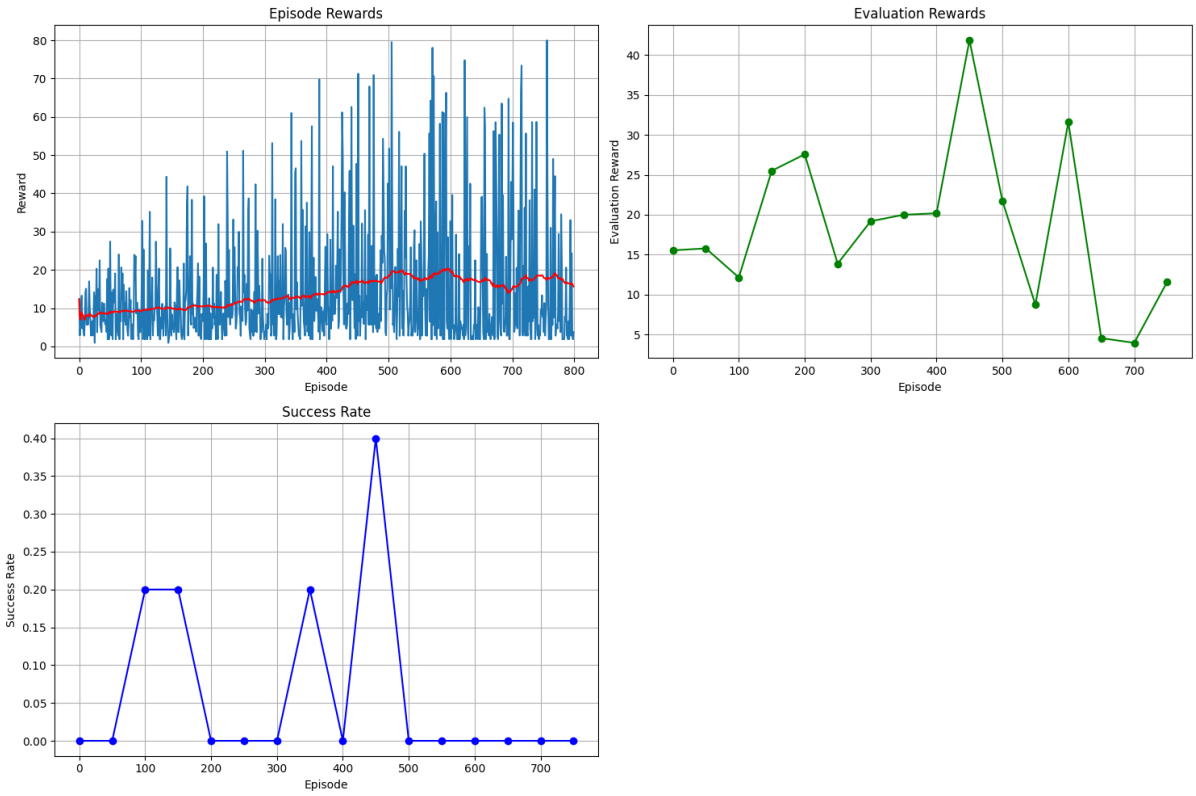


Figure 3: Évolution des différentes métriques d'entraînement pour SAC.

Conclusion SAC

L'agent SAC est bien meilleur que l'agent PPO codé manuellement, en effet il obtient un reward moyen plus élevé que le PPO et ses comportements sont plutôt bon. On peut néanmoin s'interroger sur la configuration choisie car il semble que les target_speeds soient trop élevées et que l'agent ne soit pas en mesure de contrôler sa vitesse.

Comparaison SAC vs PPO

Critère	**SAC**	**PPO**
Type d'algorithme	Off-policy	On-policy
Exploration	Stochastique avec entropie automatique	Stochastique avec entropie fixe
Mise à jour de la politique	Continue via buffer	Par lot après collecte
Stabilité de l'entraînement	Très stable	Moyenne, dépend du sampling
Vitesse de convergence	Très lente	Lente
Complexité d'implémentation	Élevée	Moyenne
Robustesse en environnement bruité	Excellente	Moyenne
Qualité des trajectoires	Correcte	Instables (dans ce cas)
Sensibilité à l'exploration initiale	Faible (grâce à l'entropie)	Élevée
Adapté à `racetrack-v0`	Oui	Oui
Résultats obtenus	Taux de succès plus élevé, trajectoires meilleures	Succès rares, sorties de piste fréquentes

Table 1: Comparaison synthétique entre SAC et PPO sur l'environnement `racetrack-v0`.

4 Utilisation de PPO via `Stable-Baselines3`

En complément de l'implémentation manuelle, nous avons évalué l'algorithme **Proximal Policy Optimization** (PPO) en utilisant la bibliothèque `Stable-Baselines3`, une référence robuste pour l'apprentissage par renforcement.

PPO est un algorithme **on-policy** qui optimise la politique en s'appuyant sur un ratio de probabilité tronqué, stabilisant ainsi l'apprentissage. Cette version intègre plusieurs optimisations pratiques : normalisation automatique, vectorisation d'environnements et gestion efficace du parallélisme.

Configuration de l'entraînement

Le modèle PPO a été entraîné avec la configuration suivante :

- Politique : `MlpPolicy` (réseau fully-connected).
- Architecture du réseau : deux couches de 256 neurones pour l'acteur (pi) et pour le critique (vf).
- Nombre d'environnements parallèles : $n = 6$ (`SubprocVecEnv`).
- Horizon temporel d'une mise à jour : $n\_steps = 128$ (par environnement).
- Taille des mini-lots : $batch\_size = 64$.
- Nombre d'époques d'optimisation par mise à jour : $n\_epochs = 10$.
- Taux d'apprentissage : 5×10^{-4}.
- Facteur de discount : $\gamma = 0.9$ (plus court-terme que les PPO/SAC précédents).
- Nombre total d'interactions : 10^5 pas temporels.

L'entraînement s'effectue en mode asynchrone sur six processus, ce qui accélère la collecte d'expériences et permet une meilleure diversité des trajectoires.

Gestion de l'environnement et enregistrement

Un wrapper `RecordVideo` est utilisé pour enregistrer les performances de l'agent, permettant une visualisation qualitative post-entraînement.

Observations

L'implémentation PPO fournie par `Stable-Baselines3` est très bonne et l'agent a un comportement tout a fait satisfaisant sans être parfait :

- Une politique parfois hésitante notamment pour dépasser les véhicules adverses
- Une vitesse assez lente globalement (bien moins rapide que SAC).

Cette approche présente l'avantage d'une très grande simplicité d'utilisation et de stabilité du code, permettant un prototypage rapide. Néanmoins il est assez compliqué d'y appliquer notre propre configuration de l'environnement c'est pourquoi je n'établis pas de comparaison supplémentaire avec les deux premiers algorithmes.

5 Conclusion

Dans ce projet, nous avons étudié différentes approches d'apprentissage par renforcement pour résoudre un problème de conduite autonome complexe dans l'environnement `racetrack-v0`.

L'implémentation manuelle de PPO a mis en lumière les défis liés aux algorithmes on-policy, notamment une grande sensibilité à l'échantillonnage, une convergence lente, et une forte variabilité des comportements appris. Malgré des efforts sur la régularisation et le curriculum learning, les résultats sont restés limités.

L'algorithme SAC, reposant sur une stratégie off-policy avec maximisation de l'entropie, a permis d'obtenir de meilleurs résultats. Il a offert une exploration plus efficace, une stabilité d'apprentissage supérieure, et une meilleure robustesse face aux environnements bruités et aux configurations difficiles. Néanmoins, l'apprentissage reste lent et sensible aux conditions de configuration initiale, notamment la vitesse cible des véhicules.

L'utilisation de la bibliothèque `Stable-Baselines3` pour entraîner PPO a démontré la puissance d'outils standards : une intégration rapide, un entraînement stable, et un comportement correct de l'agent. Toutefois, cette approche reste moins flexible pour une personnalisation poussée des environnements et configurations.

Au final, cette étude a souligné l'importance du choix de l'algorithme, du façonnage des récompenses, du réglage des hyperparamètres et du contrôle fin de l'environnement pour réussir l'apprentissage dans des tâches de contrôle continu réalistes. Des perspectives d'amélioration incluraient un entraînement plus long, un ajustement dynamique de la vitesse, ou encore l'implémentation de stratégies d'exploration plus avancées comme le Prioritized Experience Replay ou la planification multi-agents.