

# Reinforcement Learning

## Rapport technique : Algorithme PPO & SAC

---

IMDS5A - Polytech Clermont-Ferrand

Sous la direction de : Julien Hautot

Auteurs : Ayman ZEJLI & Loic MAGNAN

Date : 20 décembre 2025

---

## 1 Introduction et Problématique

L'Apprentissage par Renforcement (RL) s'est imposé comme une méthode incontournable pour la résolution de problèmes de contrôle continu complexes, tels que la robotique. Ce rapport se propose d'étudier et de comparer deux algorithmes majeurs de ce domaine, à savoir **Proximal Policy Optimization (PPO)** et **Soft Actor-Critic (SAC)**, appliqués à l'environnement de simulation **Hopper-v5**. Cet environnement, modélisant un robot monopode instable, constitue un banc d'essai rigoureux pour évaluer la capacité des algorithmes à acquérir une politique de contrôle robuste face à une dynamique chaotique.

### Problématique

Comment apprendre à un robot instable (Hopper) à sauter sans tomber ? Faut-il privilégier une méthode prudente et stable (PPO) ou une méthode rapide et exploratrice (SAC) ? Ce rapport compare ces deux approches pour identifier la plus performante.

## 2 Analyse Architecturale et Implémentation

L'analyse du code source développé pour cette étude met en lumière les divergences structurelles profondes entre les deux approches, chacune apportant une réponse technique différente à notre problématique.

## 2.1 PPO : Architecture et Stabilité

L'implémentation de l'algorithme PPO repose sur l'architecture suivante :

- Classes **Policy et Value** : Le code s'articule autour de deux réseaux de neurones distincts. La classe Policy modélise l'acteur et produit une distribution Gaussienne sur l'espace des actions. La classe Value estime la fonction de valeur  $V(s)$ . L'architecture utilisée est un **Perceptron Multicouche (MLP)** avec **2 couches cachées de 64 neurones** et des activations **Tanh**.
- **Estimation de l'Avantage (GAE)** : L'algorithme utilise l'Estimateur Généralisé de l'Avantage (GAE) pour réduire la variance des estimations de gradient.
- **écanisme de Clipping** : Le cœur de la stabilité de PPO réside dans sa fonction objectif "clippée". En limitant le ratio de probabilité entre la nouvelle et l'ancienne politique à l'intervalle  $[1 - \epsilon, 1 + \epsilon]$  (avec  $\epsilon = 0.2$ ), l'algorithme s'interdit les changements brusques de comportement.

Ce mécanisme de sécurité bride les mises à jour de politique trop importantes pour éviter les chutes de performance brutales. En restant conservateur, il empêche l'agent de désapprendre ses acquis et assure une progression stable, ce qui est indispensable pour maintenir l'équilibre d'un robot instable.

## 2.2 SAC : Maximisation de l'Entropie

L'algorithme SAC propose une architecture plus complexe, orchestrée par la classe **SACAgent**, visant à extraire le maximum d'information de chaque interaction.

- **ReplayBuffer et Off-Policy** : Contrairement à PPO, SAC stocke les transitions passées dans un **ReplayBuffer** pour les réutiliser, ce qui permet une efficacité d'échantillonnage bien supérieure.
- **GaussianPolicy et Entropie** : L'agent utilise une politique stochastique qui maximise non seulement la récompense, mais aussi l'entropie de la politique.
- **QNetwork (Twin Critics)** L'architecture repose sur l'emploi de deux réseaux critiques dont la valeur minimale est retenue afin de corriger les biais d'optimisme inhérents au Q-learning, tout en utilisant des réseaux plus profonds que PPO, composés de 256 neurones avec des activations ReLU pour capturer des dynamiques complexes, et des mises à jour douces (Soft Update) à un taux de 0,005 pour assurer une évolution fluide des cibles.

Cette architecture favorise la découverte rapide de solutions. L'entropie empêche l'agent de rester figé dans une posture statique (optimum local fréquent sur Hopper) et le pousse à trouver une marche dynamique.

## 3 Protocole Expérimental et Hyperparamètres

Les expériences ont été menées sur l'environnement Hopper-v5 avec des configurations d'hyperparamètres spécifiques. Voici le détail des paramètres utilisés pour chaque algorithme :

TABLE 1 – Tableau Comparatif des Hyperparamètres

Paramètre Clé	PPO (On-Policy)	SAC (Off-Policy)
Durée d’Entraînement	800 Époques ( $\approx 1.6M$ steps)	300 000 Steps
Architecture Réseau	MLP (2 couches de 64)	MLP (2 couches de 256)
Taille de Batch	64	256
Learning Rate	$3 \times 10^{-4}$	$3 \times 10^{-4}$
Mécanisme Spécifique	Clipping ( $\epsilon = 0.2$ ) + GAE	Entropie + Replay Buffer (1e6)

## 4 Analyse des Performances et Résultats

L’analyse des courbes de récompense permet de trancher sur l’efficacité relative des deux méthodes face à notre problématique.

### 4.1 Analyse de l’Algorithme PPO

L’évolution de la récompense moyenne obtenue via l’algorithme PPO, telle qu’illustrée sur la Figure 1, témoigne d’une dynamique d’apprentissage on-policy rigoureuse et sécurisée. La courbe se décompose en trois phases distinctes qui valident les fondements théoriques de cette approche.

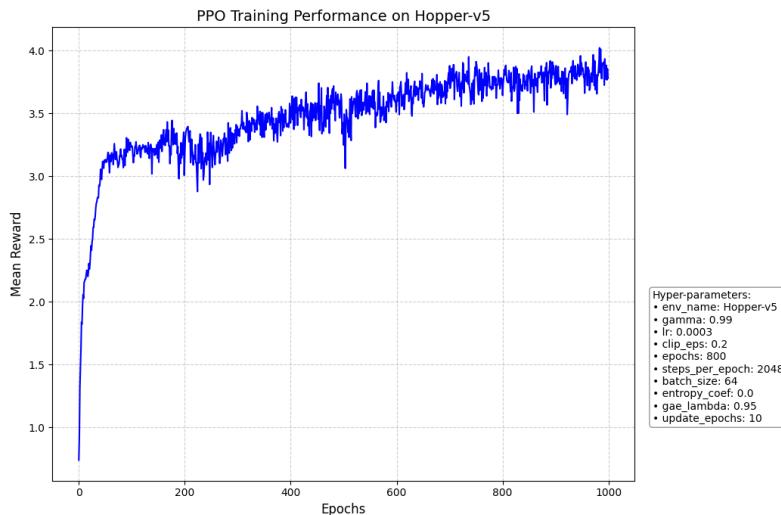


FIGURE 1 – Évolution de la récompense moyenne sur 800 époques pour PPO.

Dans un premier temps, on observe une phase d’ascension fulgurante durant les cent premières époques, passant d’un score initial de 0.75 à plus de 3.2. Ce saut de performance initial correspond à l’acquisition des réflexes moteurs de base permettant à l’agent de ne pas s’effondrer dès le début de l’épisode.

Entre les époques 100 et 1000, l’algorithme PPO confirme sa fiabilité à travers une phase d’exploration active (100-400) sécurisée par le clipping qui préserve les acquis malgré les

oscillations, suivie d'une stabilisation finale entre 3,8 et 4,0 qui garantit une politique de contrôle continu stable et reproductible.

## Effet de la normalisation des avantages

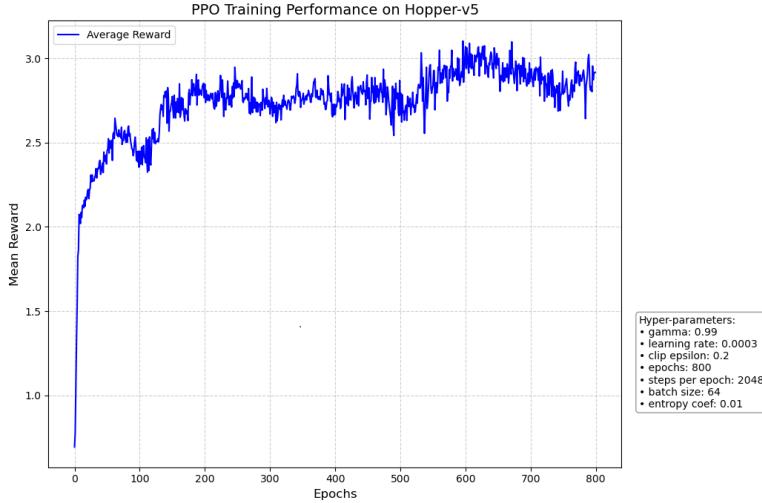


FIGURE 2 – Performance de PPO sans normalisation des avantages.

Sans normalisation, la courbe présente des oscillations majeures. La normalisation (centrage et division par l'écart-type) est essentielle pour stabiliser la magnitude du gradient.

## 4.2 Analyse de l’Algorithme SAC

Grâce à sa méthode off-policy et à l'utilisation d'un Replay Buffer, l'algorithme SAC affiche une efficacité d'échantillonnage nettement supérieure à celle de PPO. Comme l'illustre la Figure 3, l'agent dépasse le score de 1000 dès les 50 000 premiers pas, une rapidité d'apprentissage permise par la réutilisation intensive des données stockées qui assure une optimisation constante.

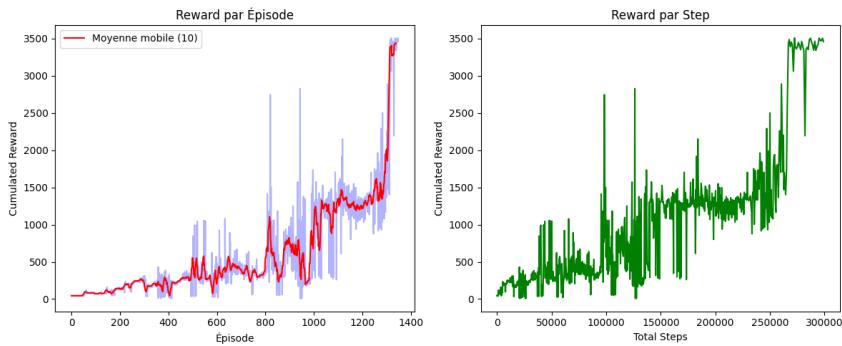


FIGURE 3 – Performance de SAC (Récompense par épisode et par steps).

Comme l'illustre la Figure 3, l'algorithme SAC connaît un véritable déclic aux alentours de 270 000 steps avec une explosion de la performance passant de 1500 à 3500 points, témoignant d'une consolidation efficace de la stratégie de course après une phase d'exploration intense, tout en conservant une forte variance inhérente à la recherche d'entropie

maximale qui, malgré des échecs ponctuels, surpassé nettement la stabilité plus lente de PPO en termes de vitesse et de coordination.

## 5 Défis Computationnels et Instabilité Algorithmique

L’implémentation et l’entraînement des algorithmes PPO et SAC ont nécessité une exploitation intensive des ressources matérielles, mobilisant principalement le GPU (CUDA) pour accélérer les opérations tensorielles des réseaux de neurones composés de couches cachées de 256 neurones. Nos expérimentations sur l’environnement Hopper-v5 ont révélé que le point de bascule entre exploration et exploitation est critique : un nombre restreint d’époques ou un volume de données trop faible empêche l’agent de consolider ses succès, provoquant une chute brutale des récompenses après une hausse éphémère.

### Analyse comparative des performances PPO

L’optimisation à 1000 époques avec normalisation des observations a permis d’atteindre un pic de récompense de 4,02 durant l’entraînement. Cependant, le phénomène d’effondrement (Catastrophic Forgetting) a été observé lors du test final avec une chute de la récompense à environ 30. Ce résultat prouve qu’un entraînement prolongé ne garantit pas la performance finale sans un équilibre parfait entre le clipping (fixé à 0,2) et la régularité des données pour éviter que l’agent n’oublie sa politique de marche stable.

### Complexité spécifique à SAC

La charge de calcul pour SAC est restée constante et lourde en raison de la mise à jour simultanée de cinq réseaux avec un batch size de 256. L’utilisation d’un paramètre de mise à jour douce ( $\tau$ ) de 0,005 et de l’Automatic Entropy Tuning a été indispensable pour prévenir une convergence prématuée vers une politique déterministe. En résumé, la réussite sur Hopper-v5 dépend autant de la puissance de calcul brute que d’une surveillance étroite du moment optimal de l’arrêt de l’apprentissage (early stopping) afin de prévenir toute divergence subite après avoir atteint l’optimum.

## 6 Conclusion

Ce projet a permis de répondre à la question centrale : comment apprendre à un robot instable comme le Hopper à sauter sans tomber ? L’étude montre que le choix de l’algorithme dépend de l’objectif visé : la sécurité ou la performance pure.

- **PPO, le choix de la prudence** : Idéal pour un apprentissage régulier et sans risque grâce au *clipping*, il évite les déviations brusques. Cependant, sa lenteur et sa difficulté à maintenir ses performances hors entraînement constituent des limites majeures.
- **SAC, le choix de la performance** : Nettement plus rapide et efficace, il utilise l’exploration par l’entropie pour consolider des stratégies dynamiques, atteignant un score de 3500 qui surpassé largement PPO.

**Bilan technique :** L’entraînement a démontré qu’une durée prolongée n’est pas gage de succès. Le phénomène d’effondrement (*catastrophic forgetting*) est une réalité technique : la réussite sur Hopper-v5 repose avant tout sur l’identification du moment optimal pour arrêter l’apprentissage.

## Références

- [1] Schulman, J., et al. (2017). *Proximal Policy Optimization Algorithms*. OpenAI. <https://arxiv.org/abs/1707.06347>
- [2] Haarnoja, T., et al. (2018). *Soft Actor-Critic : Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. UC Berkeley. <https://arxiv.org/abs/1801.01290>
- [3] Gymnasium Documentation. *Hopper Environment*. Farama Foundation. <https://gymnasium.farama.org/environments/mujoco/hopper/>
- [4] Todorov, E., et al. (2012). *MuJoCo : A physics engine for model-based control*. <http://www.mujoco.org/>