Applied Project 1: Deep RL

Jin Kilidjian
EPFL
jin.kilidjian@epfl.ch

Alessio Zazo
EPFL
alessio.zazo@epfl.ch

Joachim Despature
EPFL
joachim.despature@epfl.ch

Benjamin Bahurel
EPFL
benjamin.bahurel@epfl.ch

Abstract

This work presents a comparative evaluation of four deep reinforcement learning algorithms, DQN, PPO, SAC, and TD3, on three standard OpenAI Gym environments: CartPole-v1, Pendulum-v1, MoutainCar-v0 and MountainCarContinuous-v0. While previous studies often emphasize complex benchmarks, we focus on fundamental tasks to isolate implementation details, hyperparameter sensitivity, and algorithm-specific trade-offs. Our experiments confirm that off-policy actor-critic methods (SAC, TD3) outperform on sparse-reward and continuous control problems, whereas value-based and on-policy approaches (DQN, PPO) remain competitive in dense-reward settings with simpler dynamics. We further perform hyperparameter ablations to assess sensitivity across learning rate, network size, exploration noise, and entropy regularization. Results show significant performance variance across seeds and configurations, especially for TD3 in challenging exploration environments. By highlighting these practical considerations, we aim to bridge the gap between algorithmic theory and effective deployment in real-world scenarios. An interactive version of this study, including visualizations and training videos, is available at: https://huggingface.co/spaces/zazo2002/RL_Applied_1_Project

1 Introduction

Deep reinforcement learning algorithms exhibit significant performance variations across environments, yet comprehensive empirical comparisons remain limited Agarwal et al. [2020]. While theoretical analyses exist for individual methods Jin et al. [2019], Mei et al. [2020], practical implementation challenges and environment-specific behaviors are under-explored.

This work compares four state-of-the-art deep RL algorithms—DQN Mnih et al. [2013], PPO Schulman et al. [2017], SAC Haarnoja et al. [2018], and TD3 Fujimoto et al. [2018]—on CartPole-v1 and MountainCar-Continuous-v0. Unlike prior comparisons focusing on complex domains, we emphasize implementation details, hyperparameter sensitivity, and algorithmic trade-offs on fundamental control tasks.

Our empirical analysis reveals environment-dependent performance hierarchies, with off-policy actor-critic methods demonstrating superior sample efficiency on sparse reward tasks while both on-policy and value-based methods remain competitive on dense reward problems. We document critical implementation insights often omitted from algorithmic descriptions but essential for practical deployment.

Preprint. Under review.

2 Algorithms concept

2.1 Deep Q-Networks

Deep Q-Network (DQN) Mnih et al. [2013] is a value-based reinforcement-learning algorithm that uses a deep neural network to approximate the optimal action-value function. It learns by minimizing the temporal-difference (TD) error between its current Q-value estimates and targets computed via a periodically updated "target network," while sampling transitions uniformly from an experience replay buffer to break correlations. The training objective is the mean-squared Bellman error:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(r + \gamma \max_{a'} Q_{\theta^-}(s', a') - Q_\theta(s, a)\right)^2\right], \tag{1}$$

where Q_θ is the online network, Q_{θ^-} its lagged copy, γ the discount factor, and $\mathcal{D}$ the replay buffer of past transitions.

2.2 PPO

Proximal Policy Optimization (PPO) Schulman et al. [2017] is a policy gradient method designed to improve training stability while maintaining strong performance. PPO optimizes a clipped surrogate objective to constrain policy updates, avoiding destructive updates that can occur with large policy shifts. The central idea is to limit the difference between the new policy π_θ and the old policy $\pi_{\theta_{\text{old}}}$ using a clipped probability ratio:

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{2}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the ratio of the new and old policy probabilities and $\hat{A}_t$ is an estimate of the advantage function. PPO retains the simplicity of vanilla policy gradients while offering improved robustness and sample efficiency across a wide range of environments.

2.3 SAC

Soft Actor-Critic (SAC) Haarnoja et al. [2018] is an off-policy actor-critic method that maximizes both expected return and policy entropy for robust exploration. SAC optimizes a maximum entropy objective that encourages exploration while learning optimal policies:

$$J(\pi) = \mathbb{E}_{s_t\sim\rho_\pi}\left[\mathbb{E}_{a_t\sim\pi(\cdot|s_t)}\left[Q(s_t, a_t) - \alpha \log \pi(a_t|s_t)\right]\right] \tag{3}$$

where α is the temperature parameter controlling the exploration-exploitation trade-off. SAC uses twin Q-networks to mitigate overestimation bias and automatically tunes the temperature parameter α to maintain desired entropy levels. The off-policy nature allows efficient reuse of experience data through replay buffers, making SAC particularly effective for continuous control tasks with sparse rewards.

2.4 TD3

Twin Delayed Deep Deterministic Policy Gradient (TD3) Fujimoto et al. [2018] addresses critical issues in actor-critic methods, particularly overestimation bias and high variance in policy updates. TD3 introduces three key modifications to DDPG: (1) twin Q-networks that take the minimum value to reduce overestimation, (2) delayed policy updates that occur less frequently than critic updates to reduce variance, and (3) target policy smoothing that adds noise to target actions:

$$y = r + \gamma \min_{i=1,2} Q_{\phi_i'}(s', \tilde{a}'), \quad \tilde{a}' = \text{clip}(\mu_{\theta'}(s') + \epsilon, a_{low}, a_{high}) \tag{4}$$

where $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c)$ is clipped noise. These modifications significantly improve learning stability and sample efficiency in continuous control tasks, making TD3 particularly effective for environments requiring precise action selection.

3 Approach

3.1 OpenAI Gym environments

In **CartPole-v1**, the agent must balance a pole hinged on a moving cart by applying discrete forces to the left or right. The state is a 4-dimensional vector $(x, \dot{x}, \theta, \dot{\theta})$ capturing the cart's position and velocity as well as the pole's angle and angular velocity. At each timestep, the agent receives a reward of $+1$ for keeping the pole within bounds. An episode ends if the pole falls past a threshold angle, the cart moves too far from center, or after 500 steps. The task is considered solved when the agent achieves an average return of at least 475 over 100 consecutive episodes.

Pendulum-v1 requires the agent to swing up and stabilize an inverted pendulum using a continuous torque in $[-2, 2]$. The observation is a 3-vector $(\cos\theta,\ \sin\theta,\ \dot{\theta})$, encoding the pendulum's orientation and angular velocity. The reward is the negative of a weighted sum of squared angle error (distance from upright), angular velocity, and control effort, so that the optimal policy yields values close to zero (less negative). Each episode has a fixed horizon of 200 steps. There is no official "solved" threshold, but successful behavior keeps the pendulum upright with minimal torque.

In **MountainCar-v0**, a car stuck in a valley must build momentum to drive up a steep hill to reach a goal position. The 2-dimensional state $(x, \dot{x})$ gives position and velocity, and the action space is discrete: push left, no push, or push right. At every step the agent incurs a reward of -1 until it reaches the goal ($x \geq 0.5$) or until 200 steps elapse. Solving the task means obtaining an average return of at least -110 over 100 consecutive episodes, indicating the car consistently escapes the valley.

MountainCarContinuous-v0 is a continuous-action variant of the classic MountainCar. The state space remains 2-dimensional $(x, \dot{x})$, but the agent now selects a throttle $a \in [-1, 1]$ each step. The reward is $-0.1\, a^2$ at each timestep plus a one-time bonus of $+100$ upon reaching the goal ($x \geq 0.45$), with episodes lasting up to 999 steps. A policy is considered to have "solved" the environment when it achieves an average return of 90 or higher over 100 consecutive episodes, reflecting efficient climbs with minimal control effort.

3.2 Algorithms Implementation

We applied the **Deep Q-Network**(DQN) algorithm to the CartPole-v1 and MountainCar-v0 environments because both feature discrete action spaces. To encourage sufficient exploration early in training, we employed an ϵ-greedy policy: with probability ϵ the agent selects a random action, and with probability $1 - \epsilon$ it chooses the action with the highest predicted Q-value. By combining this strategy with experience replay (to decorrelate samples) and a periodically updated target network (to stabilize Q-value targets), our DQN implementation learns to balance the pole in CartPole and to build momentum and climb the hill in MountainCar.

We implemented **PPO** following Schulman et al. [2017] and evaluated on CartPole-v1 and MountainCarContinuous-v0. CartPole used basic actor-critic with categorical distributions, while MountainCarContinuous required Normal distributions, entropy regularization (0.01), and action std decay (0.995) for sparse reward exploration. Key practical deviations from theory include advantage normalization for stability, entropy regularization for exploration, and episode-based updates for implementation simplicity.

We implemented **TD3** following Fujimoto et al. [2018] and evaluated it on Pendulum-v1 and MountainCarContinuous-v0. For Pendulum-v1, standard TD3 architecture with twin critics, delayed actor updates, and target policy smoothing was sufficient for stable learning. MountainCarContinuous-v0, due to its sparse rewards, required reward shaping using velocity and position terms, warm-up steps, and decaying exploration noise to enable effective exploration. Key implementation details include advantage-like reward normalization in the replay buffer, per-episode CSV logging, and hyperparameter sweeps over learning rates, noise levels, and target update rates to analyze sensitivity. Practical deviations from the original paper include reward shaping and early stopping criteria based on smoothed reward thresholds for computational efficiency.

Soft Actor-Critic (SAC) was evaluated on Pendulum-v1 and MountainCarContinuous-v0 to enable direct comparison with TD3. For consistency, we retained the standard SAC algorithm when testing on the Pendulum-v0 environment. Given the relative simplicity of this environment, the algorithm

converged to a solution without requiring hyperparameter optimization, though we later tuned parameters to accelerate convergence. On MountainCarContinuous-v0, hyperparameter optimization alone proved insufficient for standard SAC convergence. Following the approach used with TD3, we implemented an identical reward shaping function, which immediately improved convergence performance. While implementing reward shaping complicated direct comparison with discrete algorithms, the resulting acceleration in simulation speed enabled more extensive hyperparameter exploration, which we determined to be more valuable for our analysis.

4 Results

4.1 Performance Across Environments of the Algorithms

4.1.1 CartPole-v1

Both DQN and PPO successfully solve CartPole-v1, achieving the target reward of 475+ (Figure 1, top right). DQN demonstrates faster initial convergence around episode 600, while PPO shows more gradual learning but achieves similar final performance. Both algorithms exhibit occasional instability after convergence, with the dense reward structure favoring both value-based and policy gradient approaches on this well-structured discrete control task.

4.1.2 MountainCarContinuous-v0

Off-policy actor-critic methods dramatically outperform PPO on MountainCarContinuous-v0 (Figure 1 bottom). SAC achieves rapid convergence around episode 200, reaching the solve threshold of 90+ reward, while TD3 shows similar performance with slightly more volatility. PPO struggles significantly, remaining near -40 reward throughout training and failing to discover the successful exploration strategy required for this sparse reward environment. The continuous action space and sparse reward structure clearly favor off-policy methods with experience replay.

4.1.3 Pendulum-v1

SAC and TD3 demonstrate comparable performance on Pendulum-v1, both converging to approximately -250 reward (Figure 1, top left). SAC shows slightly faster initial learning and more stable convergence, while TD3 exhibits similar final performance but with greater variance during training. Both algorithms successfully learn the swing-up and stabilization task, with their off-policy nature and continuous control capabilities well-suited to this environment's requirements.

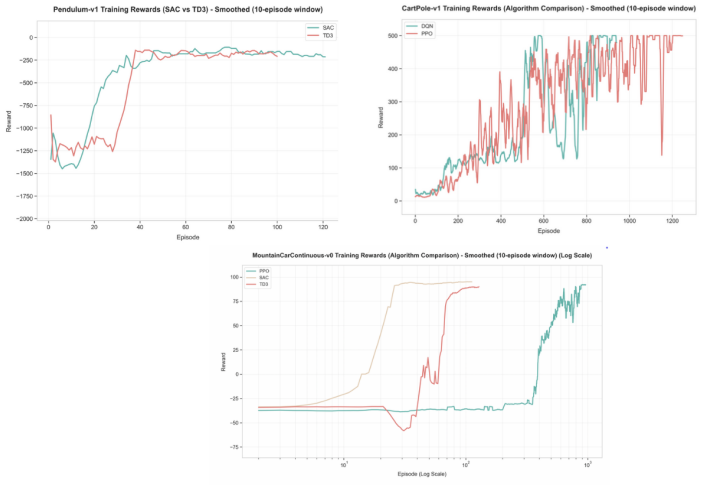

Figure 1: Training performance across all environments. Top left: Pendulum-v1 (SAC vs TD3), Top Right: Cartpole-v0 (PPO vs DQN), Bottom: MountainCarContinuous-v0 (PPO vs SAC vs TD3). All curves smoothed over 10-episode windows with log scale for episode axis where applicable.

4.1.4 Overall comparaison

Computational Constraints

Among the four algorithms evaluated, **SAC** emerges as the most computationally demanding due to its complex architecture involving dual Q-networks, target networks, policy networks, and entropy regularization mechanisms. This sophisticated design requires additional forward passes and sampling from stochastic policies, significantly increasing computational overhead. **TD3** follows as the second most resource-intensive algorithm, maintaining six separate networks including two critics, two target critics, an actor, and a target actor. The algorithm's computational burden is further amplified by its requirement for multiple forward passes per update to implement target smoothing and delayed policy updates. **PPO** occupies a middle ground in terms of computational expense, utilizing separate policy and value networks but offsetting some of this cost by reusing batches across several optimization epochs. **DQN** proves to be the least computationally demanding of the four, operating with a streamlined architecture consisting of a single Q-network and its corresponding target network, though this simplicity restricts its applicability to discrete action spaces only.

Policy Storage Efficiency

The algorithms exhibit distinct characteristics in terms of policy representation compactness. **DQN** achieves the most efficient storage by storing policies implicitly through Q-values, eliminating the need for explicit policy parameters entirely. **TD3** follows with a relatively compact representation using a deterministic actor network, which requires fewer parameters compared to stochastic policy methods. **PPO** increases storage requirements by modeling stochastic policies that output both mean and log standard deviation for each action dimension, necessitating additional parameters to capture policy uncertainty. **SAC** represents the most parameter-intensive approach, modeling stochastic policies with the added complexity of entropy tuning mechanisms. This comprehensive policy representation becomes particularly demanding in high-dimensional action spaces, where the parameter count scales significantly with the dimensionality of the action space.

Scaling for Continuous Actions

The algorithms demonstrate varying degrees of effectiveness when scaling to continuous action spaces. **SAC** exhibits superior scalability in continuous domains, leveraging entropy-regularized exploration that naturally adapts to higher-dimensional action spaces while maintaining robust performance. **TD3** handles continuous control tasks effectively by implementing mechanisms to mitigate overestimation bias, which helps preserve stability as action dimensionality increases. **PPO** demonstrates reasonable scaling capabilities but may experience performance degradation in higher-dimensional continuous spaces due to its relatively simpler policy structure compared to more sophisticated approaches. **DQN** faces significant challenges in continuous domains, as the discretization of continuous action spaces becomes computationally intractable and increasingly impractical as dimensionality grows.

Efficient Use of Off-Policy Data

The algorithms vary considerably in their ability to leverage historical data efficiently. **SAC** demonstrates exceptional off-policy data efficiency through its combination of replay buffer utilization and entropy regularization, which provides robustness against distribution shifts that commonly occur when reusing older experiences. **TD3** also achieves strong off-policy performance by combining experience replay with target smoothing techniques that stabilize the learning process when working with diverse historical data. **DQN** utilizes replay buffers for data reuse but lacks sophisticated mechanisms to address distributional mismatch between current and historical data, which can limit its sample efficiency on complex tasks. **PPO** operates fundamentally as an on-policy algorithm, discarding older samples after each policy update, which inherently reduces its ability to reuse data compared to the off-policy methods and may result in lower sample efficiency in data-scarce environments.

4.2 Hyperparameter Sensitivity

4.2.1 DQN

We performed controlled hyperparameter experiments on CartPole-v1, varying one parameter at a time (Figure 2 shows these results). Higher learning rates ($\times 10$) accelerated initial gains but led to large oscillations and lower asymptotic return, whereas lower rates ($\div 10$) were stable yet converged very slowly. Batch size exhibited a classic bias–variance tradeoff: very small minibatches ($\div 10$)

incurred high gradient noise and delayed learning, while very large ones (×10) smoothed updates but amplified occasional collapses; a mid-size of 64 struck the best balance. Network capacity was likewise sensitive: reducing to a single 128-unit layer still learned but plateaued below the two-layer baseline, and expanding to twenty layers failed entirely. Finally, replay-buffer size impacted data diversity: too small (10 K) starved the agent of varied experiences and capped performance, while too large (1 M) introduced stale transitions that increased instability, making 100 K the sweet spot for CartPole.

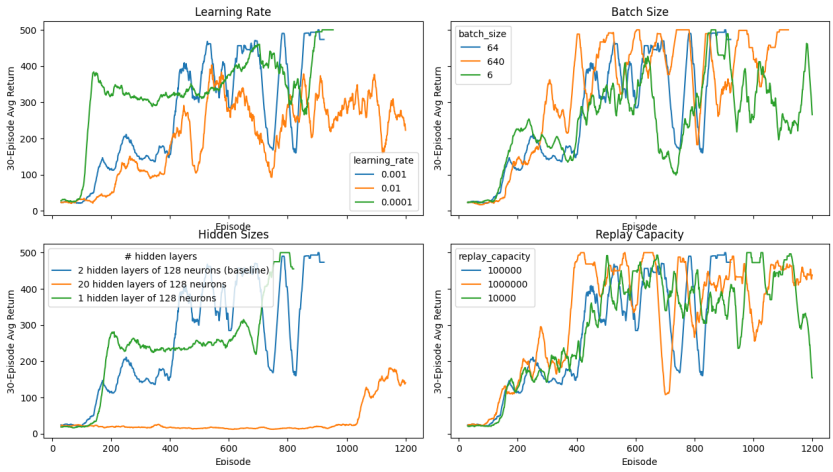

Figure 2: Effect of hyperparameters on DQN performance in Cartpole-v1. Top row: Learning rate and Batch size. Bottom row: Hidden sizes and Replay Capacity. Each plot shows performance averaged over multiple runs.

4.2.2 PPO

We performed controlled hyperparameter experiments on both environments, varying one parameter at a time (Figure 3 shows MountainCar results; CartPole exhibited similar trends). Higher learning rates accelerated convergence but caused instability, while lower discount factors ($\gamma < 0.99$) harmed long-term performance. Smaller clipping ranges ($\epsilon = 0.1$) provided stability at the cost of slower convergence, and excessive update epochs led to overfitting on stale rollouts. Action standard deviation proved critical for MountainCar exploration, requiring high initial values with gradual decay. PPO demonstrated strong performance when appropriately tuned, with particular sensitivity to learning rate and exploration parameters in sparse-reward environments.

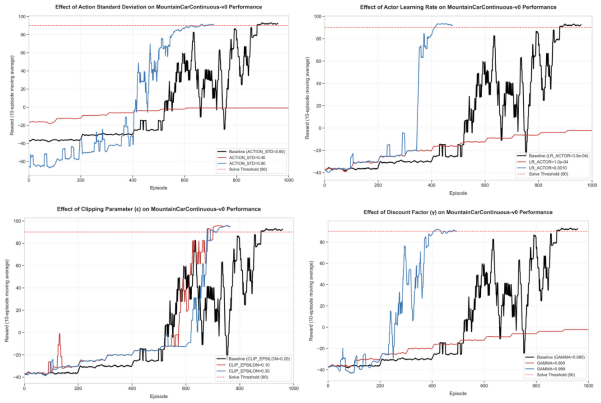

Figure 3: Effect of hyperparameters on PPO performance in MountainCarContinuous-v0. Top row: Action standard deviation and actor learning rate. Bottom row: Clipping parameter and discount factor. Each plot shows performance averaged over multiple runs.

4.2.3 SAC

Changing the hyperparameters on ‘MountainCart-Continuous‘ using **Soft Actor-Critic (SAC)**, gave us the possibility to analyze their effects on learning dynamics and final performance. The results, shown in Figure 4, reflect the average performance across multiple random seeds.

Learning Rate: Higher learning rates (e.g., ‘0.001‘) accelerated early learning but introduced instability, occasionally causing overshooting or performance dips before convergence. Mid-range values (e.g., ‘0.0006‘) balanced speed and stability effectively, while lower rates (e.g., ‘0.0003‘) offered stable but slower convergence.

Target Smoothing Coefficient (τ): This coefficient influences how quickly the target networks adapt. Lower values (e.g., ‘0.005‘) resulted in smoother, more stable learning curves, while higher values (e.g., ‘0.02‘) increased reactivity but sometimes caused sharp drops in performance. A value around ‘0.01‘ appeared to provide the best balance.

Entropy Coefficient (α): The entropy coefficient controls the degree of exploration. A higher value (e.g., ‘0.9‘) encouraged exploration but led to volatile learning. Moderate entropy (e.g., ‘0.5‘) showed strong and consistent performance, while a lower value (e.g., ‘0.1‘) led to early convergence but risked premature exploitation.

Discount Factor (γ): The discount factor determines how much future rewards are valued relative to immediate ones. A lower value (e.g., 0.95) led to faster early learning. A higher value (e.g., 0.995) prioritized long-term gains but introduced occasional instability, evident in sharp reward drops. The mid-range setting (0.99) provided a good trade-off, achieving consistently high rewards with stable convergence, making it a robust choice across episodes.

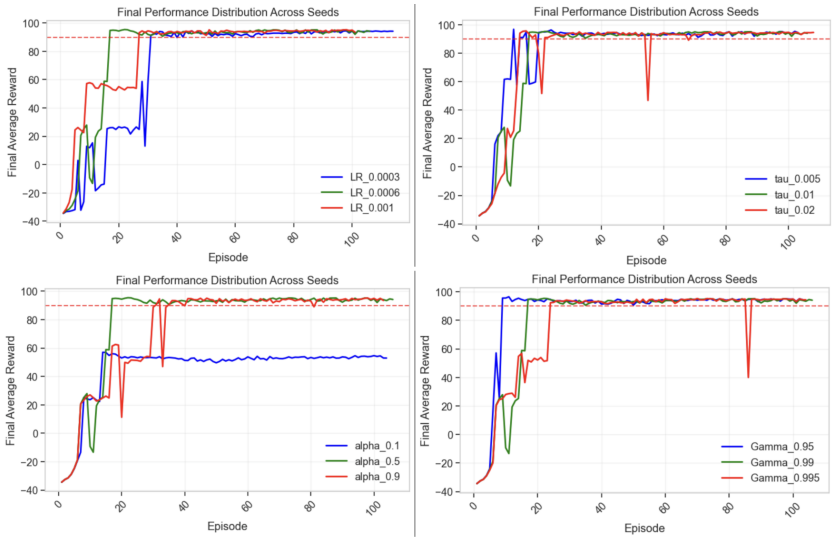

Figure 4: Effect of hyperparameters on SAC performance in MountainCarContinuous-v0. Top row: Learning Rate and Target Smoothing Coefficient. Bottom row: Entropy coefficient and discount factor. Each plot shows performance averaged over multiple runs.

4.2.4 TD3

We performed controlled hyperparameter experiments on both environments, varying one parameter at a time (Figure 5 shows Pendulum results; MoutainCarContinuous exhibited similar trends). Lowering policy noise (σ) was accelerating convergence and increasing stability. Too small τ values were leading to slower convergence, which is not a wished behavior. Actor and Critic learning rates had pretty much the same effect on the learning; some values were slightly better in terms of stability, but without making a huge difference at the end. TD3 performed greatly on Pendulum environment,

with a fast convergence and strong stability along episodes. TD3 on MoutainCarContinuous was harder to tune and pretty irregular. Sometimes the agent would simply not learn anything for 1000 episodes and on a different seeds, the objective was reached in 40-50 episodes. It seems like TD3 on this environment is really initialization-dependent, and is also really dependent on which types of actions the agent is learning at the beginning. If the agent goes in the bad direction at the beginning, there's really low chances that the agent finds the solution in a reasonable amount of time.

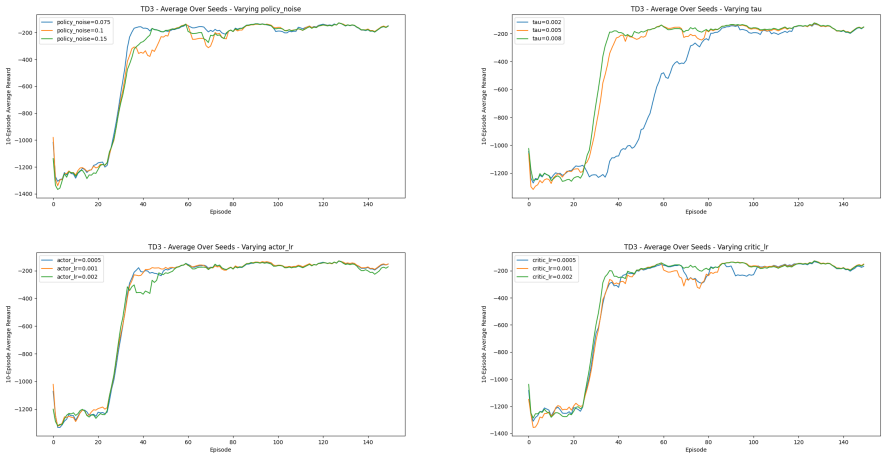

Figure 5: Effect of hyperparameters on TD3 performance in Pendulum-v1. Top row: Policy noise σ and τ. Bottom row: Actor and critic learning rates. Each plot shows performance averaged over multiple runs.

5 Conclusion

We conducted a comparative study of four widely used deep reinforcement learning algorithms: DQN, PPO, SAC, and TD3, across four Gym-AI control environments with varying levels of complexity and reward sparsity. Our experiments reveal that algorithmic performance is highly environment-dependent: DQN and PPO perform well on dense-reward tasks with discrete or low-dimensional action spaces, while SAC and TD3 excel in continuous control settings and sparse-reward domains.

Through controlled hyperparameter studies, we demonstrated that stability and convergence are sensitive to factors such as learning rate, replay buffer size, entropy regularization, and exploration noise. Notably, TD3's performance on MountainCarContinuous-v0 was particularly unstable across seeds, underlining the challenges of tuning off-policy methods in highly sensitive environments.

Despite using simplified implementations, our results align with established theoretical expectations and reinforce the importance of implementation details often omitted from standard algorithm descriptions. Limitations include the use of relatively simple environments and the exclusion of more complex domains where scalability and policy generalization could be further assessed.

Future work will extend this evaluation to larger-scale benchmarks, incorporate advanced exploration strategies, and examine transfer learning across related tasks. We also aim to explore robustness under environment perturbations and policy distillation between algorithms to better understand the generalization capabilities of deep RL methods.

For a full interactive demonstration including visual comparisons and videos of trained agents, visit: `https://huggingface.co/spaces/zazo2002/RL_Applied_1_Project`

References

Alekh Agarwal, Sham M Kakade, Jason D Lee, and Gaurav Mahajan. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. 2020.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning (ICML)*, 2018.

Chi Jin, Zhuoran Yang, Zhaoran Wang, and Michael I Jordan. Provably efficient reinforcement learning with linear function approximation. 2019.

Jincheng Mei, Chenjun Xiao, Csaba Szepesvari, and Dale Schuurmans. On the global convergence rates of softmax policy gradient methods. *International conference on machine learning*, pages 6820–6829, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 2013.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.