Comparative Analysis of Reinforcement Learning Algorithms in the Walker Environment

Rachit Agarwal, Vikhyat Bansal, Palmani Duraisamy
Amrita School of Artificial Intelligence
Amrita Vishwa Vidyapeetham, Coimbatore, India
d palmani@cb.amrita.edu

***Abstract*- In this paper, we present a comparative study of multiple reinforcement learning (RL) algorithms applied to the Walker environment of OpenAI Gym. The Walker environment, which requires an agent to learn efficient and stable locomotion, serves as an ideal testbed for evaluating the effectiveness of different RL techniques. We examine the performance of several prominent RL algorithms, including Twin Delayed DDPG(TD3), Advantage Actor Critic (A2C), Soft Actor Critic (SAC) and Proximal Policy Optimization (PPO). Our evaluation criteria focus on the learning efficiency, stability, and overall performance of each algorithm. Through rigorous experimentation, we identify the relative strengths and limitations of each method.**

Keywords: Reinforcement Learning, Algorithm, TD3, SAC and PPO.

I. Introduction

This section offers a concise synopsis of important concepts that set the stage for comprehending the field's growth and difficulties.

A. OpenAI Gym

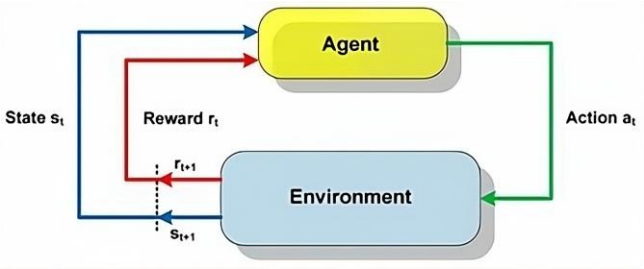
OpenAI Gym is a powerful toolkit for reinforcement learning, offering standardized environments for algorithm development and comparison. It fosters collaboration and accelerates progress in autonomous systems.

B. Reinforcement Learning

Reinforcement learning is a paradigm focused on discerning optimal actions in dynamic environments through a process of trial-and-error learning.

1) RL Algorithm: The RL algorithm, which coordinates the agent's interactions with its surroundings, is the fundamental component of reinforcement learning. The algorithm improves the agent's decision-making through recurrent trial-and-error.

2) RL problem elements: The Agent, Model, Policy, Reward Signal, Value Function, and State-Action Value Function are important parts of a reinforcement learning issue. Together, these components create the learning environment and direct the agent toward making the best decisions.



3) Model-free Reinforcement Learning: In contrast, model-free reinforcement learning algorithms interact with the environment directly, relying on trial and error to optimize decision-making processes. These algorithms, such as PP0, TD3, SAC excel in situations where an explicit model of the environment is unavailable [6].–

Types of policy we are dealing with are:

On Policy: Updates the policy based on actions taken by the same policy (e.g. PPO). It is generally more stable but less sample-efficient.

Off Policy: Updates the policy based on actions taken by a different policy or past experiences (e.g., TD3, SAC). It is typically more sample-efficient but can be less stable.

II. OBJECTIVES

A. Problem Statement

The Walker2d-v2 environment in OpenAI Gym presents a complex challenge for reinforcement learning (RL) algorithms, involving the task of controlling a two-dimensional walking agent to achieve stable and efficient locomotion. Despite significant advancements in RL, determining which algorithms perform best under various conditions remains an open question, particularly in continuous control tasks such as Walker2d-v2.

This study aims to address the following key issues:

Algorithm Performance Comparison:

Evaluate and compare the performance of multiple model-free RL algorithms, including Proximal Policy Optimization (PPO), Twin Delayed Deep Deterministic Policy Gradient (TD3), Soft Actor-Critic (SAC) in the Walker2d-v2 environment.
Metrics of interest include learning efficiency, stability of the learning process, convergence rate, and overall performance in terms of cumulative reward.

Sample Efficiency:

Determine which algorithms make the most effective use of data collected from the environment, considering the trade-offs between exploration and exploitation.

Stability and Robustness:

Assess the stability and robustness of the algorithms in learning and maintaining stable walking behavior.
Examine the algorithms' resilience to variations in initial conditions and hyperparameter settings.

Implementation Complexity:

Analyze the complexity of implementing each algorithm, including computational requirements and ease of tuning hyperparameters.
Provide insights into the practical considerations of deploying these algorithms in real-world applications.

B. Literature Review

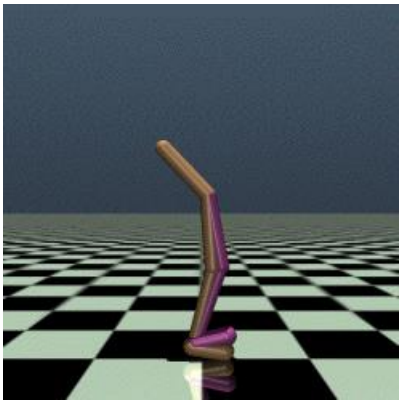
Many tasks, particularly those involving physical control, have high dimensionality and continuous action spaces [3]. The authors developed the Deep Deterministic Policy Gradient (DDPG) [3, 5] by incorporating benefits from the Actor-Critic [4], Deterministic Policy Gradient (DPG), and DQN. The DDPG performs admirably when it comes to continuous controlling. Nevertheless, Q values are overestimated in DDPG [6]. In order to adopt a minimal Q value and prevent underestimating, Fujimoto et al. employ clipped double Q learning methods [7].

The Soft Actor-Critic is an additional algorithm that bears resemblance to the TD3. It uses a maximum entropy framework, which can boost robustness and promote exploration [8]. Achieving the maximum cumulative reward while maintaining the policy's maximum entropy is the aim of maximum entropy reinforcement learning. All of the previously listed algorithms are off-policy algorithms. PPO is also appropriate for continuous system-controlling duties when it comes to on-policy algorithms. Its goal as an on-policy algorithm is to determine the optimal improvement step without going too far back in the past, which could lead to a collapse in performance.

The performance of algorithms in continuous system-controlling tasks is being compared in a number of ways. Previous research employs the v2 version of Mujoco in Gym, such as Walker2d-v2 and Ant-v1, to compare the performance of TD3, SAC, A2C, and PPO [8]. The author conducted experiments to demonstrate SAC's superiority over alternative algorithms. Furthermore, research utilizing the DDPG, TD3, and TAD3 is carried out without specifying which particular conditions were employed [6].

All things considered, this study demonstrates that the TD3, SAC, A2C, and PPO are highly capable of learning policies to regulate the continuous system. There are still a number of difficulties in this field, though. For instance, it's unclear which algorithms work best for what tasks and how many surroundings can impact how well an algorithm performs in a given situation. To address these problems and improve our comprehension of RL in Mujoco continuous system control, more research is required.

III. ABOUT ENVIRONMENT



This environment extends the hopper habitat, which was developed using the work of Erez, Tassa, and Todorov in "Infinite Horizon Model Predictive Control for Nonlinear Periodic Tasks," by including an additional pair of legs that

allow the robot to walk forward rather than hop. In comparison to traditional control settings, this environment, like previous Mujoco environments, seeks to expand the amount of independent state and control variables.
Two thighs are located in the middle below the torso, two legs are located in the bottom below the thighs, and two feet are attached to the legs on which the entire body rests. These four main body parts make up the two-dimensional, two-legged figure known as the walker. By applying torques to the six hinges that connect the six body parts, the
objective is to make both sets of feet, legs, and thighs cooperate to move forward (right).

Action Space	Box(-1.0, 1.0, (6,), float32)
Observation Shape	(17,)
Observation High	[inf inf inf inf inf inf inf inf inf inf inf inf inf inf inf inf]
Observation Low	[-inf -inf -inf -inf -inf -inf -inf -inf -inf -inf -inf -inf -inf -inf -inf -inf]

Figure 1(a)

Num	Action	Control Min	Control Max	Name (in corresponding XML file)	Joint	Unit
0	Torque applied on the thigh rotor	-1	1	thigh_joint	hinge	torque (N m)
1	Torque applied on the leg rotor	-1	1	leg_joint	hinge	torque (N m)
2	Torque applied on the foot rotor	-1	1	foot_joint	hinge	torque (N m)
3	Torque applied on the left thigh rotor	-1	1	thigh_left_joint	hinge	torque (N m)
4	Torque applied on the left leg rotor	-1	1	leg_left_joint	hinge	torque (N m)
5	Torque applied on the left foot rotor	-1	1	foot_left_joint	hinge	torque (N m)

Figure 1(b)

Num	Observation	Min	Max	Name (in corresponding XML file)	Joint	Unit
0	z-coordinate of the top (height of hopper)	-Inf	Inf	rootz (torso)	slide	position (m)
1	angle of the top	-Inf	Inf	rooty (torso)	hinge	angle (rad)
2	angle of the thigh joint	-Inf	Inf	thigh_joint	hinge	angle (rad)
3	angle of the leg joint	-Inf	Inf	leg_joint	hinge	angle (rad)
4	angle of the foot joint	-Inf	Inf	foot_joint	hinge	angle (rad)
5	angle of the left thigh joint	-Inf	Inf	thigh_left_joint	hinge	angle (rad)
6	angle of the left leg joint	-Inf	Inf	leg_left_joint	hinge	angle (rad)
7	angle of the left foot joint	-Inf	Inf	foot_left_joint	hinge	angle (rad)
8	velocity of the x-coordinate of the top	-Inf	Inf	rootx	slide	velocity (m/s)
9	velocity of the z-coordinate (height) of the top	-Inf	Inf	rootz	slide	velocity (m/s)
10	angular velocity of the angle of the top	-Inf	Inf	rooty	hinge	angular velocity (rad/s)
11	angular velocity of the thigh hinge	-Inf	Inf	thigh_joint	hinge	angular velocity (rad/s)
12	angular velocity of the leg hinge	-Inf	Inf	leg_joint	hinge	angular velocity (rad/s)
13	angular velocity of the foot hinge	-Inf	Inf	foot_joint	hinge	angular velocity (rad/s)
14	angular velocity of the thigh hinge	-Inf	Inf	thigh_left_joint	hinge	angular velocity (rad/s)
15	angular velocity of the leg hinge	-Inf	Inf	leg_left_joint	hinge	angular velocity (rad/s)
16	angular velocity of the foot hinge	-Inf	Inf	foot_left_joint	hinge	angular velocity (rad/s)

Figure 1(c)

Figure 1(a) gives general information about the environment,
Figure 1(b) tells about action space,
Figure 1(c) tells about observation space.

REWARDS

The reward consists of three parts:

healthy_reward: Every timestep that the walker is alive, it receives a fixed reward of value healthy_reward,

forward_reward: A reward of walking forward which is measured as forward_reward_weight * (x-coordinate before action - x-coordinate after action)/dt. dt is the time between actions and is dependeent on the frame_skip parameter (default is 4), where the frametime is 0.002 - making the default dt = 4 * 0.002 = 0.008. This reward would be positive if the walker walks forward (right) desired.

ctrl_cost: A cost for penalising the walker if it takes actions that are too large. It is measured as ctrl_cost_weight * sum(action2) where ctrl_cost_weight is a parameter set for the control and has a default value of 0.001

The total reward returned is reward = healthy_reward bonus + forward_reward - ctrl_cost and info will also contain the individual reward terms

Starting state

All observations start in state (0.0, 1.25, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) with a uniform noise in the range of [-0.005, 0.005] added to the values for stochasticity.

Episode Termination

If terminate_when_unhealthy=True is passed during construction (which is the default), the episode terminates when any of the following happens:

The episode duration reaches a 1000 timesteps

The walker is unhealthy.

IV. ALGORITHMS IN USE

1) PPO - Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a popular model-free reinforcement learning (RL) algorithm developed by OpenAI. It strikes a balance between ease of implementation, sample efficiency, and reliable performance, making it a go-to choice for many RL applications.

Policy Gradient Method:
PPO is a type of policy gradient method, which means it directly optimizes the policy that maps states to actions by estimating the gradient of the expected reward with respect to the policy parameters.

Clipped Objective Function:
One of the core innovations in PPO is the use of a clipped surrogate objective. This objective limits the size of policy updates to prevent drastic changes, which helps stabilize training. The clipped objective is defined as:

$$L(\theta) = \epsilon_\tau[min(r(\theta)\hat{A}\_t, clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}\_t)]$$

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_\theta \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t))\right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2,$$

typically via some gradient descent algorithm.
8: **end for**

2) TD3 – Twin Delayed DDPG

While DDPG can achieve great performance sometimes, it is frequently brittle with respect to hyperparameters and other kinds of tuning. A common failure mode for DDPG is that the learned Q-function begins to dramatically overestimate Q-values, which then leads to the policy breaking, because it exploits the errors in the Q-function. Twin Delayed DDPG (TD3) is an algorithm that addresses this issue by introducing three critical tricks:

Trick One: Clipped Double-Q Learning. TD3 learns two Q-functions instead of one (hence "twin"), and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions.

Trick Two: "Delayed" Policy Updates. TD3 updates the policy (and target networks) less frequently than the Q-function. The paper recommends one policy update for every two Q-function updates.

Trick Three: Target Policy Smoothing. TD3 adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along changes in action.

Together, these three tricks result in substantially improved performance over baseline DDPG.

Clipped double-Q learning. Both Q-functions use a single target, calculated using whichever of the two Q-functions gives a smaller target value:

$$y(r, s', d) = r + \gamma(1-d) \min_{i=1,2} Q_{\phi_{i,\text{targ}}}(s', a'(s')),$$

and then both are learned by regressing to this target:

$$L(\phi_1, \mathcal{D}) = \mathop{\mathrm{E}}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(Q_{\phi_1}(s,a) - y(r,s',d)\right)^2\right],$$

$$L(\phi_2, \mathcal{D}) = \mathop{\mathrm{E}}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(Q_{\phi_2}(s,a) - y(r,s',d)\right)^2\right].$$

Using the smaller Q-value for the target, and regressing towards that, helps fend off overestimation in the Q-function.

In TD3, the policy is updated less frequently than the Q-functions are. This helps damp the volatility that normally arises in DDPG because of how a policy update changes the target.

Algorithm 1 Twin Delayed DDPG

1: Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
3: **repeat**
4: Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
5: Execute a in the environment
6: Observe next state s', reward r, and done signal d to indicate whether s' is terminal
7: Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
8: If s' is terminal, reset environment state.
9: **if** it's time to update **then**
10: **for** j in range(however many updates) **do**
11: Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12: Compute target actions

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \quad \epsilon \sim \mathcal{N}(0, \sigma)$$

13: Compute targets

$$y(r, s', d) = r + \gamma(1-d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$

14: Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} (Q_{\phi_i}(s,a) - y(r,s',d))^2 \qquad \text{for } i = 1, 2$$

15: **if** j mod `policy_delay` $= 0$ **then**
16: Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} Q_{\phi_1}(s, \mu_\theta(s))$$

17: Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1-\rho)\phi_i \qquad \text{for } i = 1, 2$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta$$

18: **end if**
19: **end for**
20: **end if**
21: **until** convergence

3) SAC – Soft Actor Critic

Soft Actor Critic (SAC) is an algorithm that optimizes a stochastic policy in an off-policy way, forming a bridge between stochastic policy optimization and DDPG-style approaches. It isn't a direct successor to TD3 (having been published roughly concurrently), but it incorporates the clipped double-Q trick, and due to the inherent stochasticity of the policy in SAC, it also winds up benefiting from something like target policy smoothing.

A central feature of SAC is entropy regularization. The policy is trained to maximize a trade-off between expected return and entropy, a measure of randomness in the policy. This has a close connection to the exploration-exploitation trade-off: increasing entropy results in more exploration, which can accelerate learning later on. It can also prevent the policy from prematurely converging to a bad local optimum.

$$\mathop{\mathrm{E}}_{a\sim\pi_\theta}[Q^{\pi_\theta}(s,a) - \alpha \log \pi_\theta(a|s)] = \mathop{\mathrm{E}}_{\xi\sim\mathcal{N}}[Q^{\pi_\theta}(s,\tilde{a}_\theta(s,\xi)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s,\xi)|s)]$$

$$\max_\theta \mathop{\mathrm{E}}_{\substack{s\sim\mathcal{D}\\ \xi\sim\mathcal{N}}}\left[\min_{j=1,2} Q_{\phi_j}(s,\tilde{a}_\theta(s,\xi)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s,\xi)|s)\right]$$

SAC trains a stochastic policy with entropy regularization, and explores in an on-policy way. The entropy regularization coefficient \alpha explicitly controls the explore-exploit tradeoff, with higher \alpha corresponding to more exploration, and lower \alpha corresponding to more exploitation. The right coefficient (the one which leads to the stablest / highest-reward learning) may vary from environment to environment, and could require careful tuning.

Algorithm 1 Soft Actor-Critic

1: Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
3: **repeat**
4: Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
5: Execute a in the environment
6: Observe next state s', reward r, and done signal d to indicate whether s' is terminal
7: Store (s,a,r,s',d) in replay buffer $\mathcal{D}$
8: If s' is terminal, reset environment state.
9: **if** it's time to update **then**
10: **for** j in range(however many updates) **do**
11: Randomly sample a batch of transitions, $B = \{(s,a,r,s',d)\}$ from $\mathcal{D}$
12: Compute targets for the Q functions:

$$y(r,s',d) = r + \gamma(1-d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s',\tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

13: Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2 \qquad \text{for } i = 1,2$$

14: Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} \left(\min_{i=1,2} Q_{\phi_i}(s,\tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)|s\right)\right),$$

where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.
15: Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1-\rho)\phi_i \qquad \text{for } i = 1,2$$

16: **end for**
17: **end if**
18: **until** convergence

At test time, to see how well the policy exploits what it has learned, we remove stochasticity and use the mean action instead of a sample from the distribution. This tends to improve performance over the original stochastic policy.

For a detailed explanation behind working of soft actor critic algorithm please refer to the openAI blog. It gives mathematical proof behind working of this algorithm in various condition.

V. Results

Performance Metrics Comparison

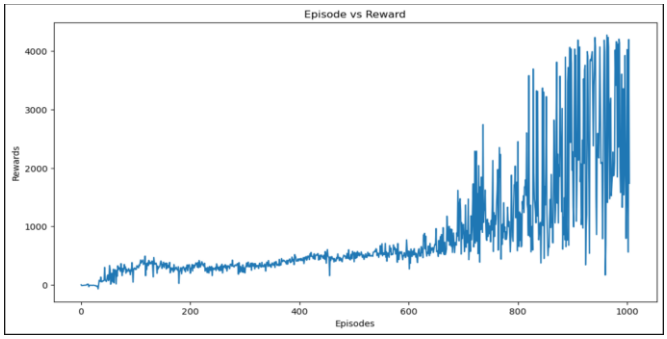
Metric	TD3	SAC	PPO
Average Return	3813.95	3116.83	1202.52
Standard Deviation of Return	576.61	664.44	674.11
Variance of Return	332480.03	441478.84	454425.07
Average Return under Perturbation	3647.41	3134.54	329.49
Standard Deviation under Perturbation	733.57	698.14	597.66
Early Performance	124.91	58.94	360.79
Variance of Late Performance	1349864.11	255100.31	666714.38
Total Training Time (seconds)	1068.33	2688.49	1809.78
Average Time per Episode (seconds)	1.06	2.06	1.81

Inferences

TD3 (Twin Delayed Deep Deterministic Policy Gradient):

1. Performance: TD3 shows the highest average return (3813.95), indicating strong performance in the Walker2d environment.

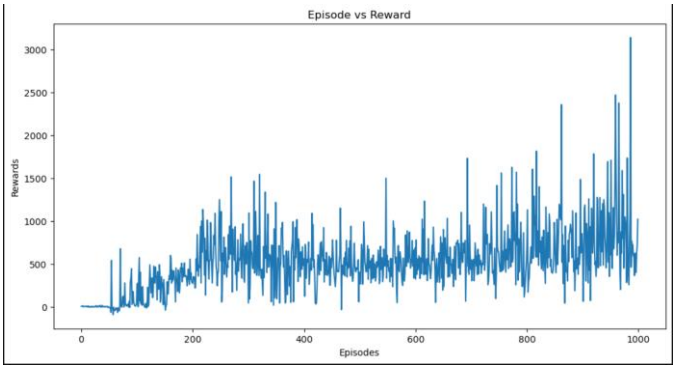
2. Stability: The relatively low standard deviation of return (576.61) suggests stable performance, though the variance of late performance is quite high (1349864.11), indicating some instability towards the end of training.
3. Robustness: TD3 maintains a good average return under perturbation (3647.41) with an increased standard deviation (733.57), showing robustness but slightly affected by perturbations.
4. Sample Efficiency: The early performance of 124.91 suggests slower initial learning compared to PPO but better than SAC.
5. Computational Efficiency: TD3 is the most computationally efficient, with a total training time of 1068.33 seconds and the fastest average time per episode (1.06 seconds).



TD3 learning curve

SAC (Soft Actor-Critic):

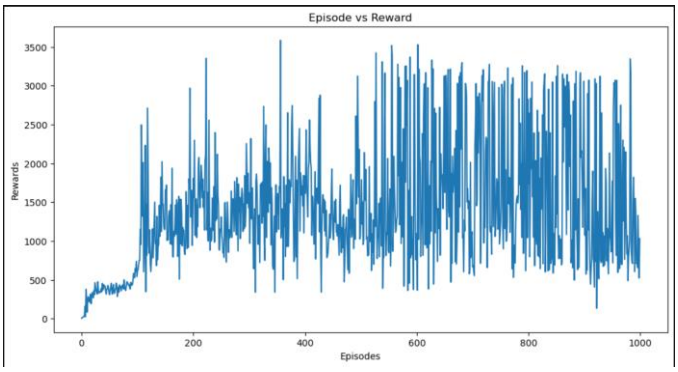
1. Performance: SAC has a lower average return (3116.83) than TD3 but higher than PPO, indicating decent performance.
2. Stability: The standard deviation of return (664.44) suggests more variability in performance compared to TD3, and the variance of late performance is the lowest (255100.31), indicating good stability in later stages.
3. Robustness: SAC's performance under perturbation is consistent (3134.54) with a slight increase in standard deviation (698.14), showing robustness similar to its normal performance.
4. Sample Efficiency: SAC has the lowest early performance (58.94), indicating slower initial learning.
5. Computational Efficiency: SAC is the least computationally efficient, with the highest total training time (2688.49 seconds) and the slowest Average time per episode (2.06 seconds)



SAC learning curve

PPO (Proximal Policy Optimization):

1. Performance: PPO has the lowest average return (1202.52), indicating weaker performance in the Walker2d environment compared to TD3 and SAC.
2. Stability: PPO has the highest standard deviation of return (674.11), suggesting significant variability in performance, and its variance of late performance is also relatively high (666714.38), indicating less stability.
3. Robustness: PPO's performance drops significantly under perturbation (329.49) with a lower standard deviation (597.66), indicating high sensitivity to changes.
4. Sample Efficiency: PPO shows the highest early performance (360.79), indicating faster initial learning.
5. Computational Efficiency: PPO is moderately efficient, with a total training time of 1809.78 seconds and an average time per episode of 1.81 seconds, falling between TD3 and SAC.



PPO learning curve

VI. CONCLUSION & FUTURE WORK

Conclusion

In this study, we evaluated and compared the performance of three reinforcement learning algorithms—Twin Delayed Deep Deterministic Policy Gradient (TD3), Soft Actor-Critic (SAC), and Proximal Policy Optimization (PPO)—in the Walker2d environment from MuJoCo. Our analysis was based on a set of comprehensive performance metrics, including average return, standard deviation and variance of return, performance under perturbation, early performance, variance of late performance, total training time, and average time per episode.

Our findings indicate that:

1. TD3 demonstrated the highest average return and robust performance under perturbation, making it the best overall performer in this environment. It also exhibited the fastest computational efficiency. However, its high variance of late performance suggests some instability towards the end of the training process.
2. SAC showed consistent and stable performance with a good average return and the lowest variance of late performance, indicating high stability in the later stages of training. Despite its robustness and stability, SAC was found to be computationally expensive and exhibited slower initial learning.
3. PPO achieved the fastest initial learning and moderate computational efficiency but had the lowest average return and highest sensitivity to perturbations. This indicates significant variability and less robustness compared to TD3 and SAC.

Based on these results, TD3 is recommended for scenarios where high performance and computational efficiency are prioritized. SAC is suitable for applications requiring stable performance in the later stages of training, despite its higher computational costs. PPO can be considered for situations demanding rapid initial learning, although its overall performance and robustness are relatively lower.

Future Work

Future research can extend this study in several directions:

1. Extended Evaluation: Evaluate the performance of TD3, SAC, and PPO across a broader range of environments and tasks to generalize the findings. This can include other continuous control tasks and different complexities to understand the versatility and robustness of each algorithm.
2. Parameter Tuning: Investigate the impact of different hyperparameter settings on the performance of these algorithms. Fine-tuning hyperparameters could potentially enhance the performance and stability of each algorithm.
3. Hybrid Approaches: Explore hybrid approaches that combine the strengths of TD3, SAC, and PPO. For instance, a method that integrates the stability of SAC with the efficiency of TD3 might yield improved results.
4. Advanced Metrics: Incorporate additional performance metrics such as energy efficiency, robustness to non-stationary environments, and real-world applicability. This can provide a more holistic evaluation of the algorithms' performance in practical applications.
5. Real-World Implementation: Test these algorithms in real-world robotic systems to assess their practical viability and performance in real-time applications. This will help in understanding the practical constraints and adaptations needed for real-world deployments.

By addressing these areas, future work can provide deeper insights into the strengths and limitations of each algorithm, paving the way for more robust, efficient, and versatile reinforcement learning solutions.

VII. REFERENCES

[1] Liu, Shijie. "An Evaluation of DDPG, TD3, SAC, and PPO: Deep Reinforcement Learning Algorithms for Controlling Continuous System." In *2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing (DAI 2023)*, pp. 15-24. Atlantis Press, 2024.

[2] Park, Inhee, and Teng-Sheng Moh. "Multi-agent deep reinforcement learning for walker systems." In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 490-495. IEEE, 2021.

[3] Mujoco. MuJoCo-Gym Documentation, URL: https://www.gymlibrary.dev/environments/mujoco/. Last accessed: Aug. 11, 2023.

[4] Konda, V., & Tsitsiklis, J. Actor-critic algorithms. Advances in neural information processing systems, 12, 1008-1014 (1999).

[5] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. Deterministic policy gradient algorithms. In International conference on machine learning. 387-395 (2014).

[6] Tiong, T., Saad, I., Teo, K. T. K., & bin Lago, H. Deep reinforcement learning with robust deep deterministic policy gradient. arXiv:2011.00001 (2020).

[7] Fujimoto, S., van Hoof, H., & Meger, D. Addressing function approximation error in actor-critic methods. arXiv preprint arXiv:1802.09477 (2018).

[8] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint arXiv:1801.01290 (2018).

[9] Haarnoja, Tuomas, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. "Learning to walk via deep reinforcement learning." *arXiv preprint arXiv:1812.11103* (2018).

[10] Reda, Daniele, Tianxin Tao, and Michiel van de Panne. "Learning to locomote: Understanding how environment design matters for deep reinforcement learning." In *Proceedings of the 13th ACM SIGGRAPH Conference on Motion, Interaction and Games*, pp. 1-10. 2020.

[11] Huang, Shengyi, Anssi Kanervisto, Antonin Raffin, Weixun Wang, Santiago Ontañón, and Rousslan Fernand Julien Dossa. "A2C is a special case of PPO." *arXiv preprint arXiv:2205.09123* (2022).

[12] Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms." *arXiv preprint arXiv:1707.06347* (2017).

[13] Gu, Yang, Yuhu Cheng, CL Philip Chen, and Xuesong Wang. "Proximal policy optimization with policy feedback." *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52, no. 7 (2021): 4600-4610.

[14] Dankwa, Stephen, and Wenfeng Zheng. "Twin-delayed ddpg: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent." In *Proceedings of the 3rd international conference on vision, image and signal processing*, pp. 1-5. 2019.