Comparative Analysis of Soft Actor-Critic and Proximal Policy Optimization Algorithms on the Walker2D Environment

Reinforcement Learning Final Project

By Rufina George and Shrivatsa Mudligiri

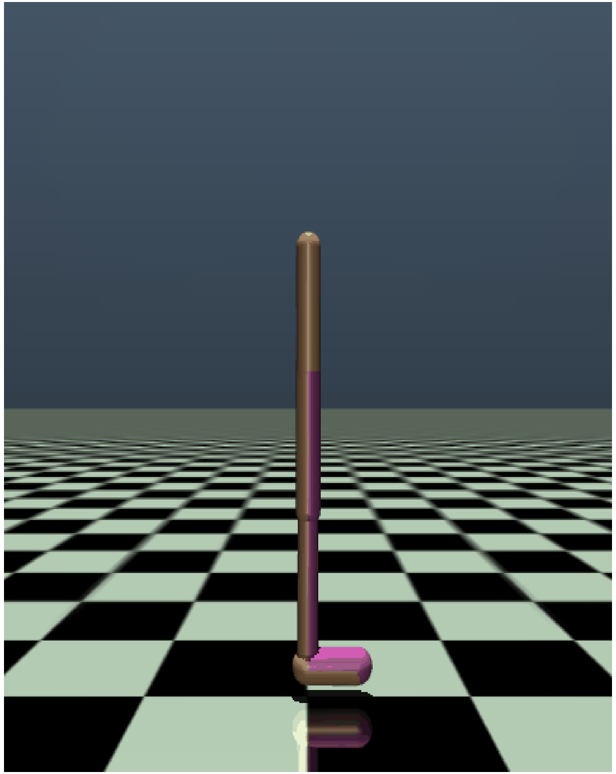
Introduction

Challenges in Robotics

- Traditional control methods in robotics struggle with complex and dynamic environments.
- These tasks demand precise and fine-grained control over robot actuators.
- Hard-coding behaviors for every task and environment becomes impractical.
- Dynamic and uncertain environments necessitate adaptive control strategies.

The Solution: Reinforcement Learning

- Reinforcement Learning (RL) enables goal-oriented learning through interaction with the environment.
- Robots can learn optimal behaviors by exploring actions and learning from consequences.
- RL offers flexibility and adaptability, crucial for continuous control tasks.
- Continuous learning from interactions improves performance and robustness.
- Algorithms like Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO) are effective in continuous control.



Introduction

Benefits of Reinforcement Learning in Robotics

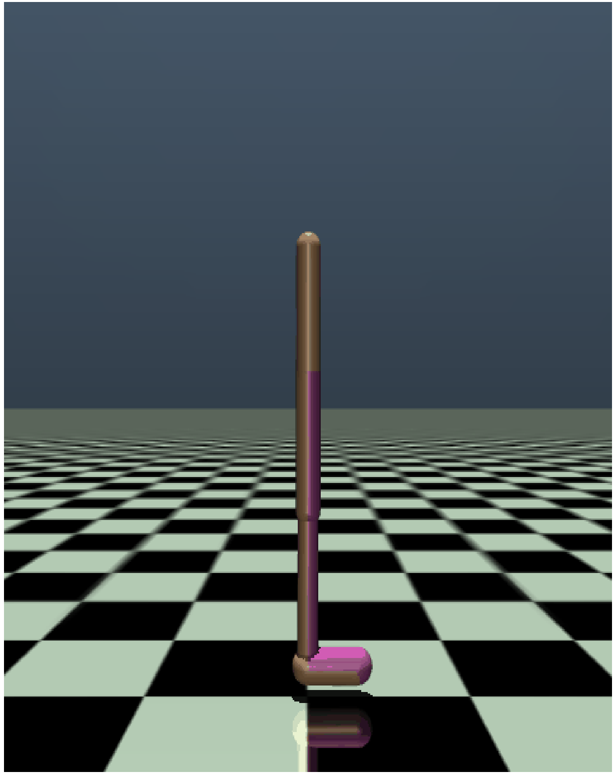
- Autonomous learning and adaptation without manual programming.
- Tackling complex and dynamic environments more effectively.

Conclusion

- RL offers a powerful paradigm for achieving adaptive control in robotics.
- Advancements in RL have the potential to unlock new capabilities in robotic systems.

Goal

- Evaluate and compare the performance of SAC and PPO on the Walker2D environment.
- Understand their strengths and weaknesses in the context of a complex control problem.
- Investigate factors such as training efficiency, stability, and robustness, which are important considerations in real-world applications of RL algorithms.



Actor-critic methods

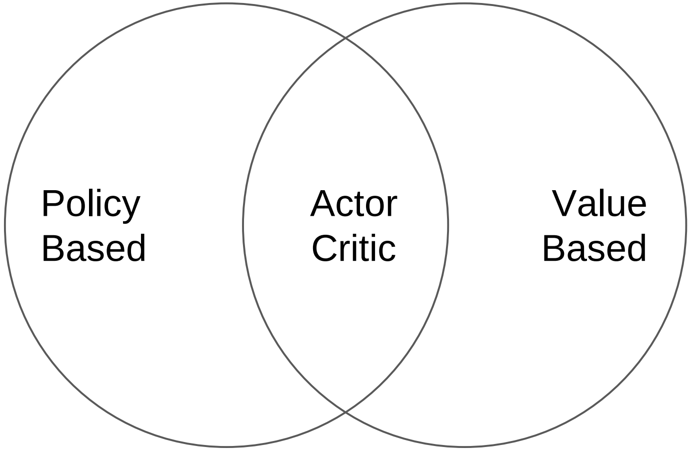
Actor-Critic methods combine value-based and policy-based approaches.

The “Critic” estimates the value function (action-value or state-value) and the “Actor” updates the policy distribution in the direction suggested by the Critic (such as with policy gradients).

Both the Critic and Actor functions are parameterized with neural networks.

Advantages

- Efficient Exploration
- High Sample Efficiency
- Policy Improvement
- Flexible



Soft Actor-Critic (SAC)

SAC combines (actor-critic + entropy regularization)

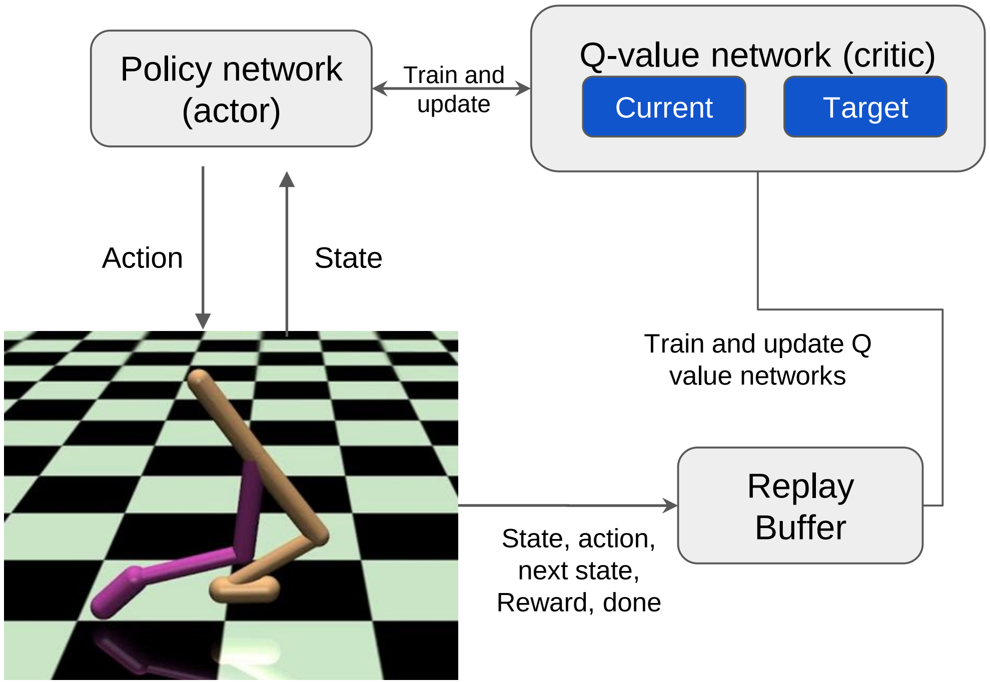
Key Features:

- Well-suited for tasks with continuous action spaces
- Directly optimizes the policy, learning the best actions to take in different situations.
- Estimates the value function to guide the learning process and evaluate the expected return or future rewards.
- Employs a soft value function update approach
- Soft Q-Learning and Soft Value Iteration use maximum entropy principles to encourage the learning of soft value functions.
- Policies achieve high rewards and capture the underlying uncertainty in the environment.

Applications:

- Successfully applied to a wide range of tasks, including robotic control, autonomous driving, and game playing.
- In robotics, SAC enables robots to perform precise and smooth movements, adapt to dynamic environments, and learn complex tasks.
- SAC has also shown promising results in complex game-playing domains, achieving impressive performance in games like Atari and Dota 2.

Soft Actor-Critic: How it works?



Loss function for Q-network in SAC:

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

The policy should, in each state, act to maximize the expected future return plus expected future entropy:

$$\begin{aligned} V^{\pi}(s) &= \underset{a \sim \pi}{\mathrm{E}} \left[Q^{\pi}(s,a) \right] + \alpha H \left(\pi(\cdot|s) \right) \\ &= \underset{a \sim \pi}{\mathrm{E}} \left[Q^{\pi}(s,a) - \alpha \log \pi(a|s) \right]. \end{aligned}$$

To get the policy loss, the final step is that we need to substitute Q^{\pi_{\theta}} with one of our function approximators

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

Proximal Policy Optimization (PPO)

PPO's combination of stability, sample efficiency, and compatibility with continuous action spaces has made it a popular choice for various reinforcement learning applications.

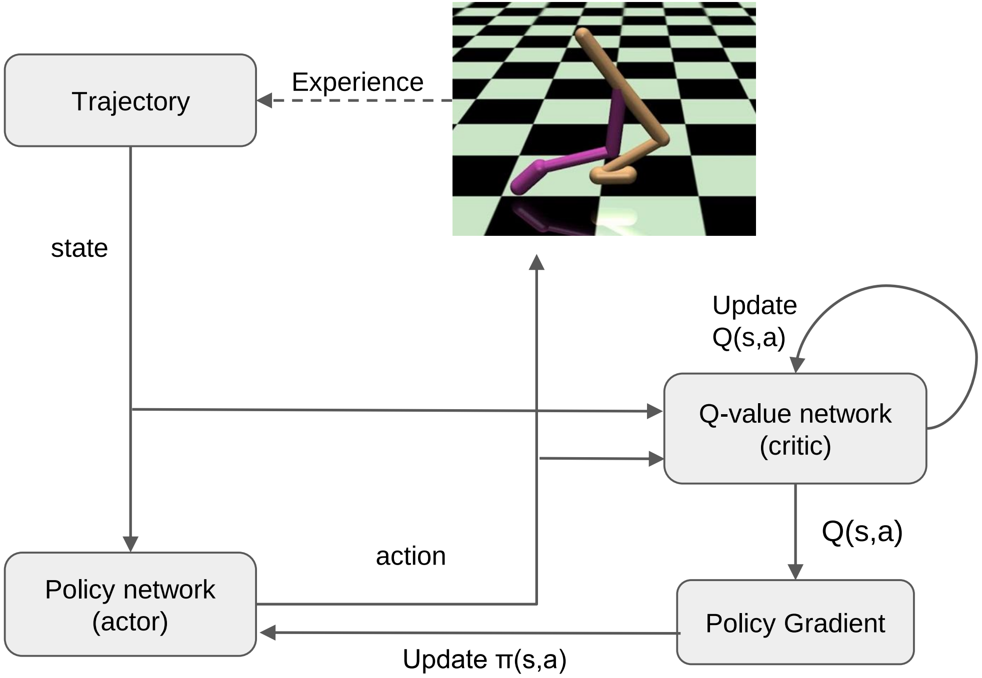
Key Features:

- Works well with continuous action spaces, ideal for tasks that require fine-grained control.
- Policy updates are based on the most recent and relevant experiences as it is an on-policy algorithm.
- Objective function trades off between maximizing expected rewards and controlling the policy update magnitude.
- Incorporates a trust region approach to policy updates.
- Achieves effective policy learning with fewer interactions with the environment.

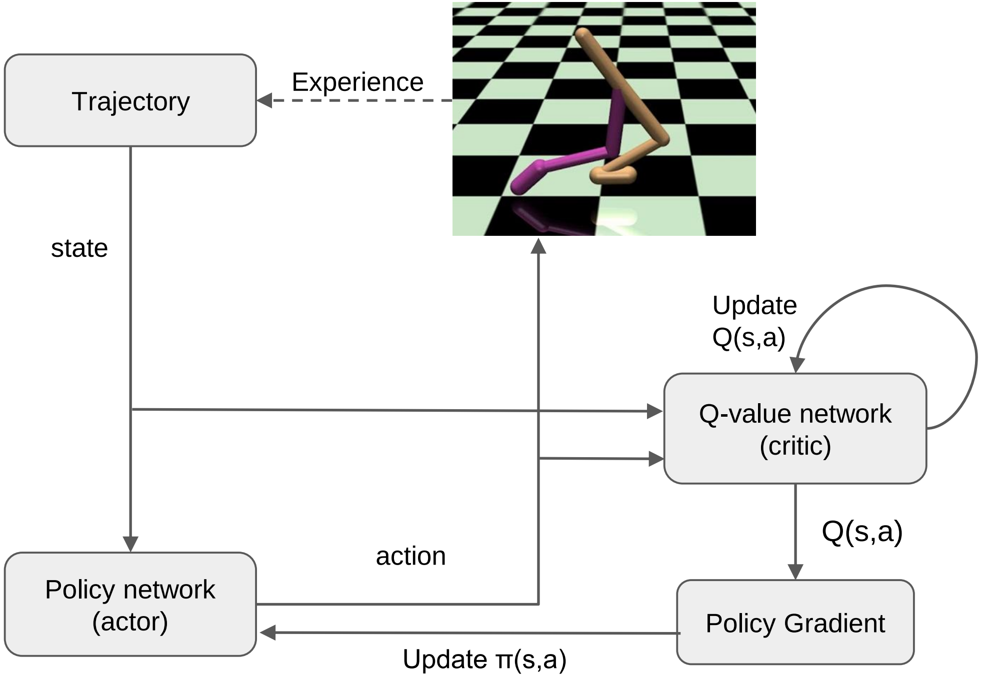
Applications:

- PPOs ability to handle continuous action spaces and stability during training makes it suitable for complex control tasks in robotics.
- It has been used to achieve high-performance agents in challenging game environments, such as Atari games and complex strategy games.
- PPO is commonly used in simulated environments like the OpenAI Gym and MuJoCo.
- PPO has also been widely used as a benchmark algorithm in reinforcement learning research.

Proximal Policy Optimization (PPO)



Proximal Policy Optimization (PPO)



Ratio between the new policy and old policy

$$r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_k}(a_t|s_t)$$

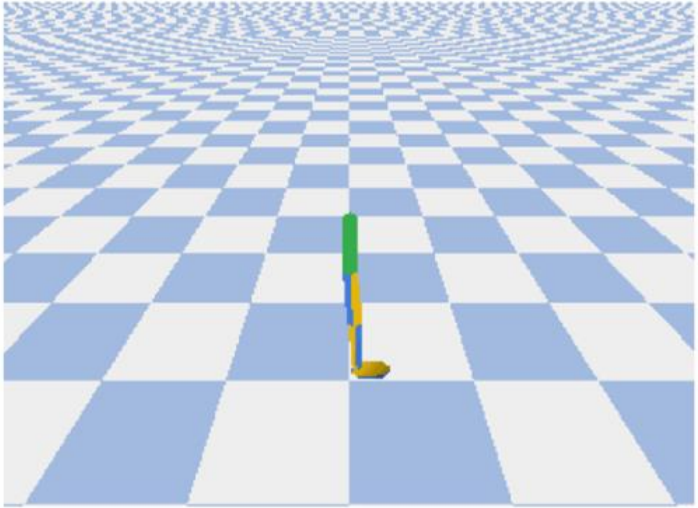
Clipped PPO Objective Function

$$\mathcal{L}^{CLIP}_{\theta_k}(\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k})\right]\right]$$

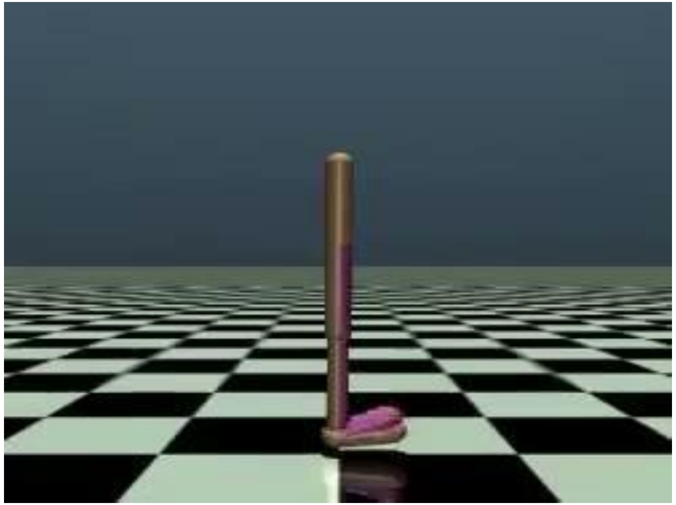
Policy Update

$$\theta_{k+1} = \arg\max_\theta \mathcal{L}^{CLIP}_{\theta_k}(\theta)$$

Results

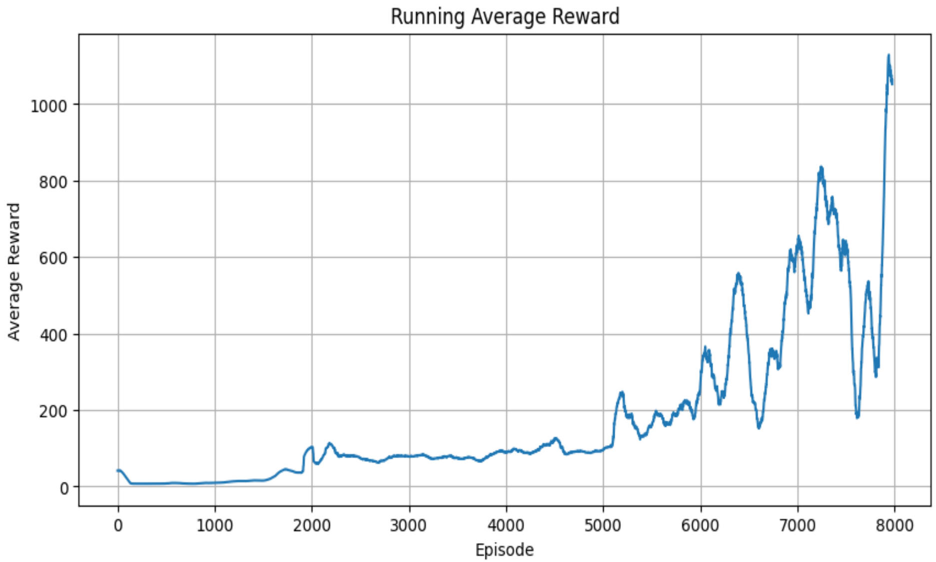


SAC Trained Walker2D Agent



PPO Trained Walker2D Agent

Results



SAC Running Reward Plot
(Training Time = 3476 seconds
1,500,000 timesteps)

PPO Running Reward Plot
(Training Time = 5787.92 seconds
1,000,000 timesteps)

Conclusion

- Both algorithms performed well on the environment, but PPO produced more stable results when compared to SAC based on the running rewards plot. Whereas the SAC experiment produced a much better result when compared based on the simulation generated.
- PPO being an on-policy algorithm, made it slower and less sample efficient compared to SAC, which is an off-policy algorithm.
- SAC, with its off-policy nature and soft value function updates, has a much better sample efficiency and faster learning on continuous control tasks like Mujoco Walker2d.
- PPO is generally regarded as a more stable algorithm and was easier to implement and tune compared to SAC, which has additional complexity due to the value function estimation.

Thank You