Reinforcement Learning for 2D Humanoid Bipedal Locomotion: A Comparative Study of PPO, SAC, and DDPG Algorithms

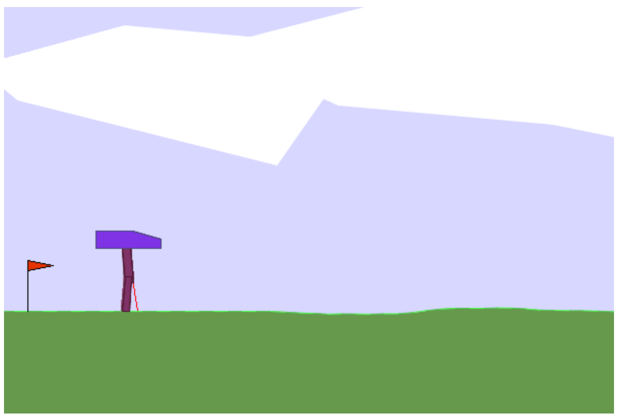
Gopi Sainath Mamindlapalli, Harin Kumar Nallaguntla

Introduction

In this project, we undertook the task of developing and comparing three distinct reinforcement learning algorithms—Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), and Deep Deterministic Policy Gradients (DDPG). Our primary objective was to address the formidable challenge of enabling a humanoid robot to achieve bipedal locomotion, specifically the complex task of walking forward, within a physics-based simulator. This endeavor required us to build these algorithms from the ground up, emphasizing their capacity to tackle the intricate problem of human-like locomotion. Our evaluation framework placed significant emphasis on key performance metrics, including training stability, sample efficiency, and the quality of the learned walking behavior, providing a comprehensive analysis of the algorithms' efficacy in addressing this challenging problem.

Environment/Simulator

The Bipedal Walker v3 environment is a challenging simulation featuring a 4-joint walker robot with an action space controlling motor speed values in the range of -1 to 1 for each of the 4 joints at the hips and knees. The observation space includes diverse state information such as hull angle speed, angular velocity, horizontal and vertical speed, joint positions, angular speeds, legs' contact with the ground, and data from 10 lidar rangefinder measurements, notably lacking explicit coordinates. Rewards are structured to encourage forward movement, with points accumulating up to 300 as the robot progresses. Falling incurs a -100 penalty, and applying motor torque results in a minor deduction. The optimal agent aims to achieve a higher score through efficient navigation. The starting state places the walker at the left end of the terrain, and episodes conclude upon hull-ground contact or if the walker surpasses the right end of the terrain length, introducing termination conditions that challenge the agent's adaptability and control capabilities in this dynamic bipedal locomotion setting.



Bipedal Walker environment

Overview of Algorithms Used

SAC: Soft Actor-Critic (SAC) is an advanced reinforcement learning algorithm designed for continuous control tasks. It operates in an off-policy manner, decoupling the actor and critic networks to enhance data efficiency. One distinctive aspect of SAC is its incorporation of entropy maximization as an essential component of the objective function, promoting a balance between exploration and exploitation. By encouraging the policy to exhibit high entropy, SAC facilitates robust exploration in complex environments. The algorithm's action space is characterized by motor speed values within the [-1, 1] range for each of the 4 joints, allowing for nuanced control over the bipedal walker in our scenario. SAC employs a temperature parameter, allowing users to regulate the trade-off between exploration and exploitation during training. Notably, SAC has demonstrated success in achieving sample efficiency and adaptability, making it well-suited for challenging tasks such as humanoid locomotion within the Bipedal Walker v3 environment.

Algorithm 1 Soft Actor-Critic

```
1: Input: initial policy parameters $\theta$, Q-function parameters $\phi_1$, $\phi_2$, 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
               $$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)\middle| s\right)\right),$$
            where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt $\theta$ 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
```

DDPG: Deep Deterministic Policy Gradients (DDPG) is a powerful reinforcement learning algorithm specifically designed to address continuous action spaces in complex environments. A key characteristic of DDPG is its combination of deep neural networks for both the actor and the critic. The actor network determines the optimal policy in a continuous action space, providing a deterministic action for a given state. The critic, on the other hand, evaluates the quality of the chosen actions by estimating the state-action value function. DDPG employs experience replay and target networks to stabilize training and improve sample efficiency. This algorithm leverages the insights of Q-learning and policy gradients, enabling it to handle high-dimensional state and action spaces effectively.

Algorithm 1 Deep Deterministic Policy Gradient

```
1: Input: initial policy parameters $\theta$, Q-function parameters $\phi$, empty replay buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
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 however many updates do
11:      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:      Compute targets
             $y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$
13:      Update Q-function by one step of gradient descent using
             $\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_\phi(s, a) - y(r, s', d))^2$
14:      Update policy by one step of gradient ascent using
             $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$
15:      Update target networks with
             $\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$
             $\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$
16:    end for
17:  end if
18: until convergence
```

PPO: Proximal Policy Optimization (PPO) stands out as a robust and widely utilized reinforcement learning algorithm tailored for addressing complex tasks, particularly in environments with continuous action spaces. PPO operates within an on-policy framework, focusing on improving stability and sample efficiency during training. A distinctive feature of PPO is its emphasis on maintaining a conservative policy update through a constrained optimization objective, preventing large policy changes that could destabilize learning. By incorporating a clipping mechanism, PPO bounds the ratio between new and old policies during optimization, thereby ensuring that the algorithm converges in a stable and controlled manner. This characteristic makes PPO particularly well-suited for scenarios where ensuring training stability is critical.

Algorithm 1 PPO-Clip

```
1: Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
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 $V_{\phi_k}$.
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
```

Implementation Details

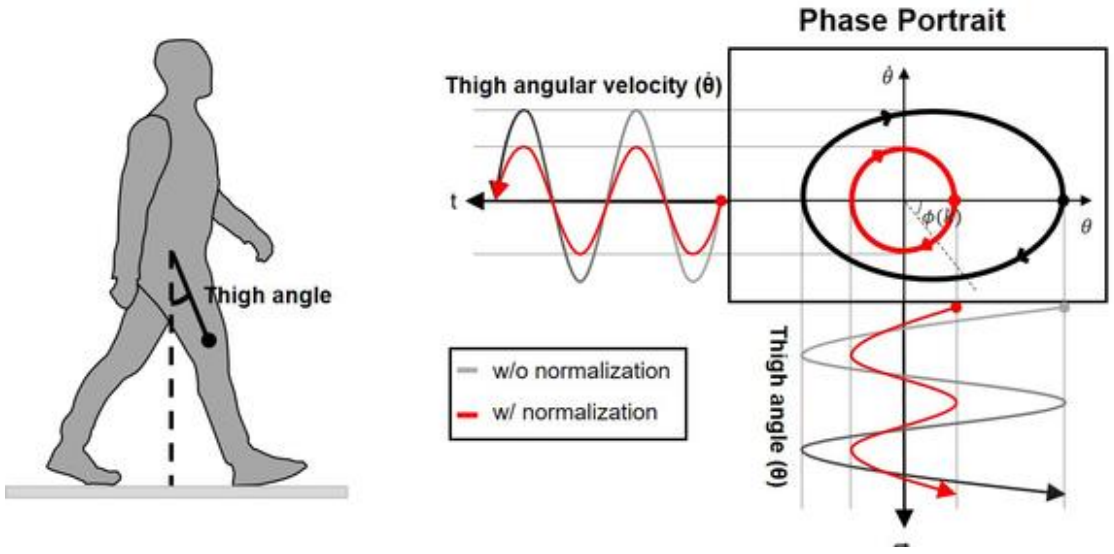
We implemented these three algorithms from scratch in PyTorch and these are the hyper-parameters we used for training agents.

PyTorch	**Soft Actor-Critic**	**Deep Deterministic Policy Gradients**	**Proximal Policy Optimization**
γ	*0.99*	*0.99*	*0.99*
α	*3e-4*	*1e-3*	*3e-4*
$batch\ size$	*256*	*256*	*128*
$train\ steps$	*1M*	*1M*	*1M*
	soft update=0.005 *initial entropy=0.1*	*soft update=0.005*	*clip=0.005* *update steps=1024*

Here, γ is discounting factor and α is learning rate. Soft update is the soft update value we used in updating critic network weights. Initial entropy is the initial value of entropy regularization we used in SAC. Clip value is the value we use to clip the Actor loss. Update steps is the number of updates we do in PPO at every update step.

Phase Portrait Analysis

Phase portrait analysis is a mathematical and graphical technique used to explore and understand the dynamic behavior of a system described by a set of differential equations. In this analysis, the state variables of the system are represented as coordinates in a multi-dimensional space, forming a phase space. The trajectories of these variables over time are then depicted as curves or paths within this space, offering valuable insights into the system's long-term behavior, stability, and potential attractors or repellers. The patterns and shapes observed in the phase portrait provide visual cues about the underlying dynamics of the system, aiding researchers and analysts in uncovering key characteristics such as periodicity, oscillations, or convergence to equilibrium points. Phase portrait analysis is widely employed in various scientific fields, including physics, biology, and control theory, as it provides an intuitive and insightful representation of the evolution of dynamic systems.

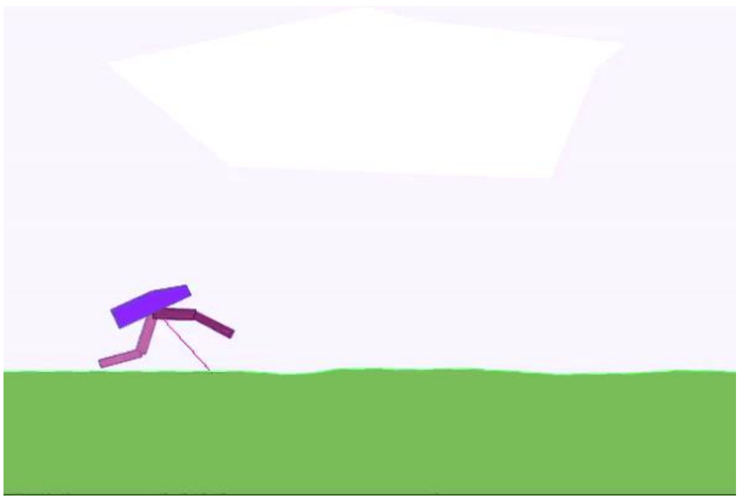


Phase Portrait of a Human Gait

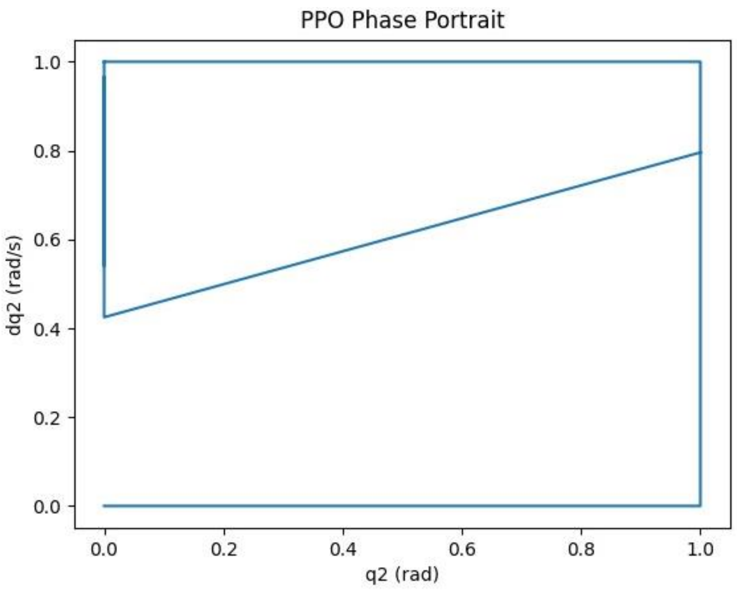
A bounded phase portrait is indicative of the stability of the gait, suggesting that the system remains within specific limits or boundaries during the entire walking cycle. In the context of human locomotion, a bounded phase portrait reflects the ability of the musculoskeletal and neuromuscular systems to maintain a consistent and controlled gait. Deviations from a stable walking pattern are constrained within defined parameters, demonstrating the system's resilience to external disturbances and adaptability to varying terrains. This graphical representation of gait parameters, such as joint angles and angular velocities, assures that the walking process remains within a predictable range, minimizing the risk of falls or erratic movements. Overall, a bounded phase portrait serves as a visual testament to the robust stability of the gait system.

Comparison of 2D Humanoid Bipedal Gaits

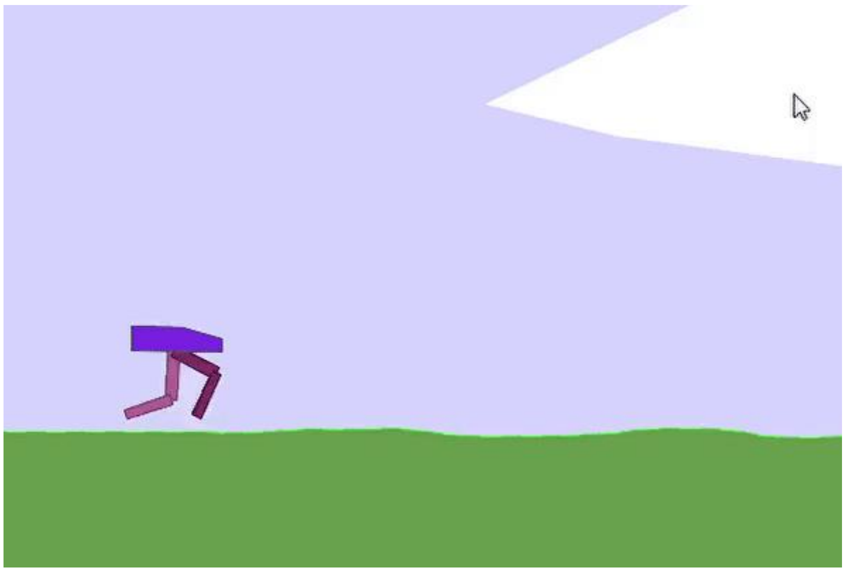
We trained all the three algorithms for 1 million timesteps and compared the final obtained performance. Our primary focus lies in comparing the gaits achieved after 1 million train steps. It is important to note that our intention is not to evaluate the algorithms based on extended training durations, as we acknowledge that with more training, all three algorithms would likely attain reasonable stability. To analyze the stability of the obtained gaits, we are generating a phase portrait of the left knee motor joint angle, referred to as "q2." This approach allows us to gain insights into the gait stability within the chosen timeframe of 1 million timesteps. Below, we show the final gaits obtained from training all three algorithms and their corresponding phase portraits.



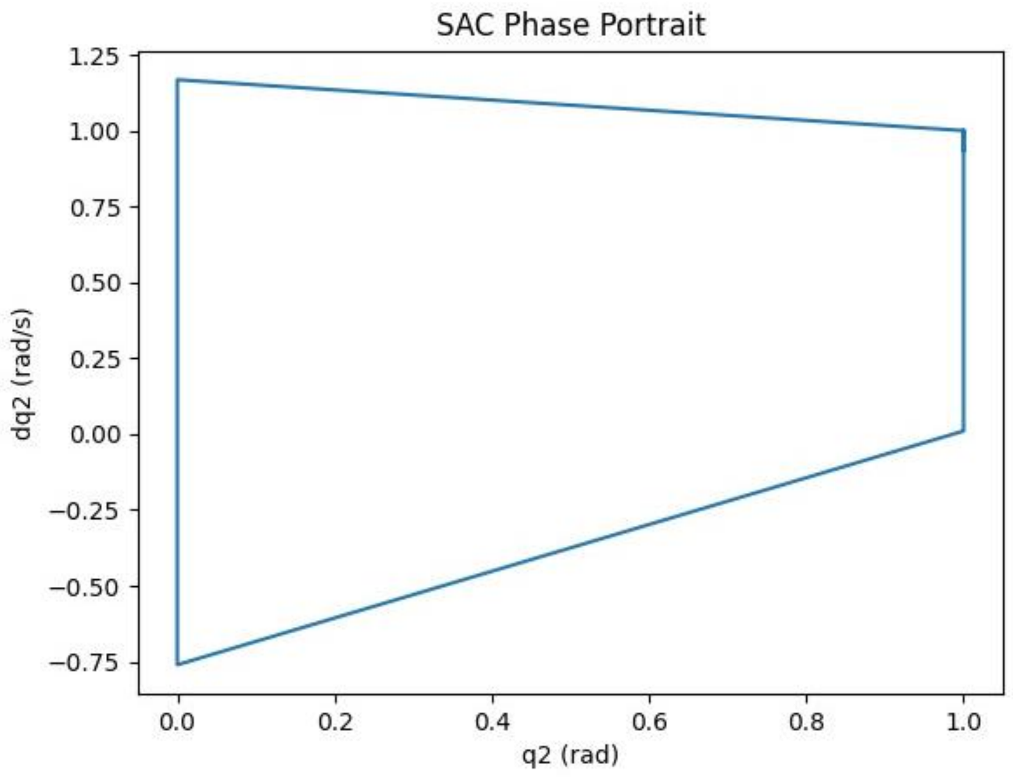
PPO: reward 291, length 1084



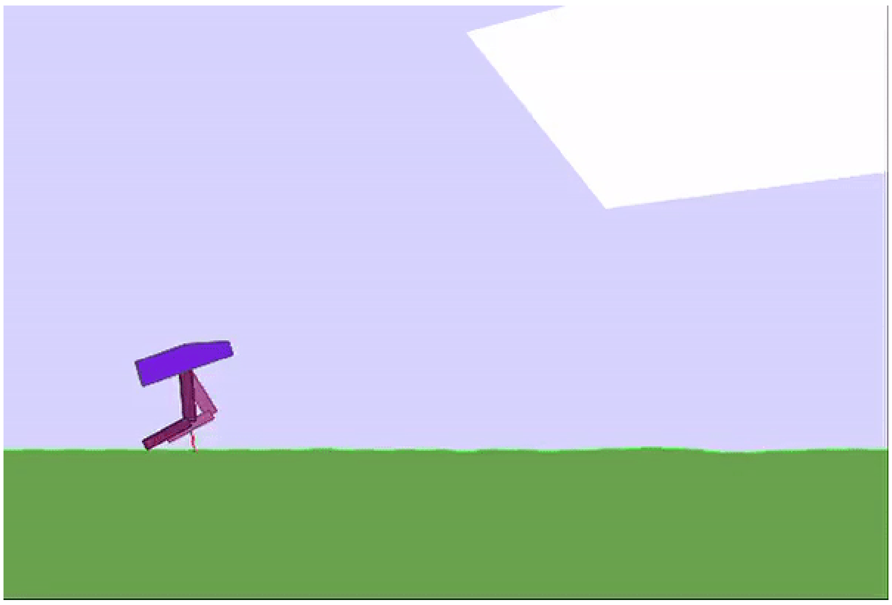
Observing the video, it is evident that PPO successfully completes the episode; however, the gait's stability is questionable as one of its legs is noticeably dragged across the ground for forward movement. The phase portrait analysis of this gait reveals an unbounded trajectory, indicating instability in the gait's motion.



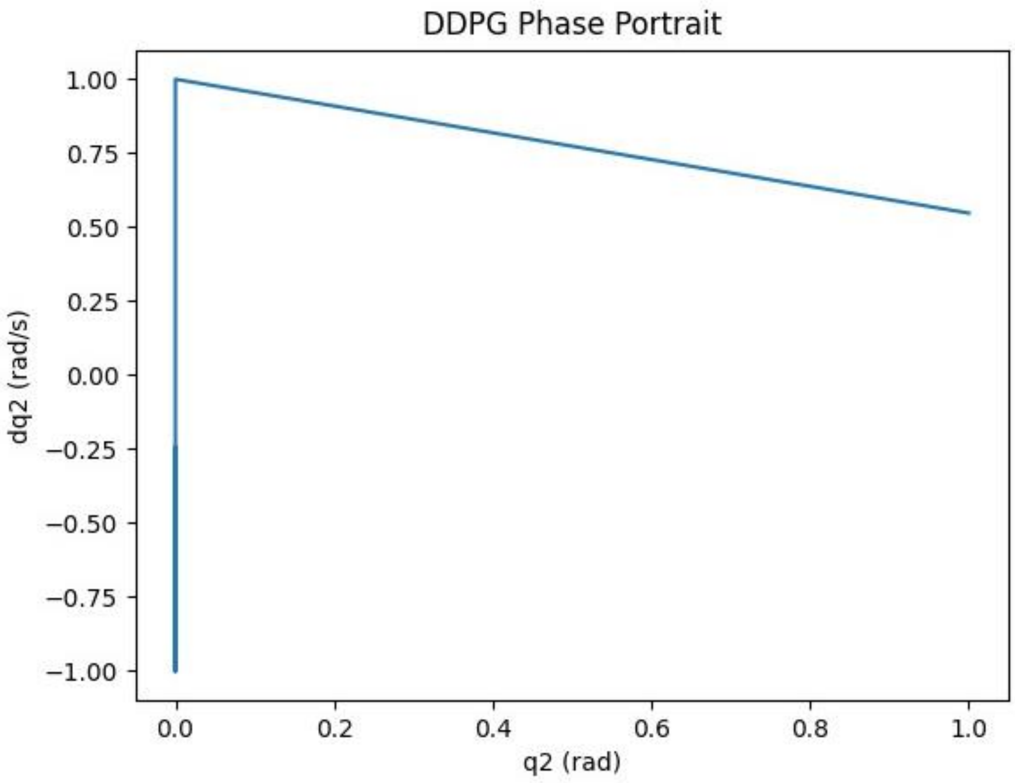
SAC: reward 315, length 755



SAC successfully completes the episode, exhibiting a notably natural and stable gait without the presence of artifacts observed in PPO. The absence of anomalies in the gait, such as dragging or awkward movements, contributes to a more authentic walking behavior. The stability of SAC's gait is further substantiated by the analysis of the bounded phase portrait plot, affirming that the system maintains a consistent and controlled trajectory throughout the walking cycle.

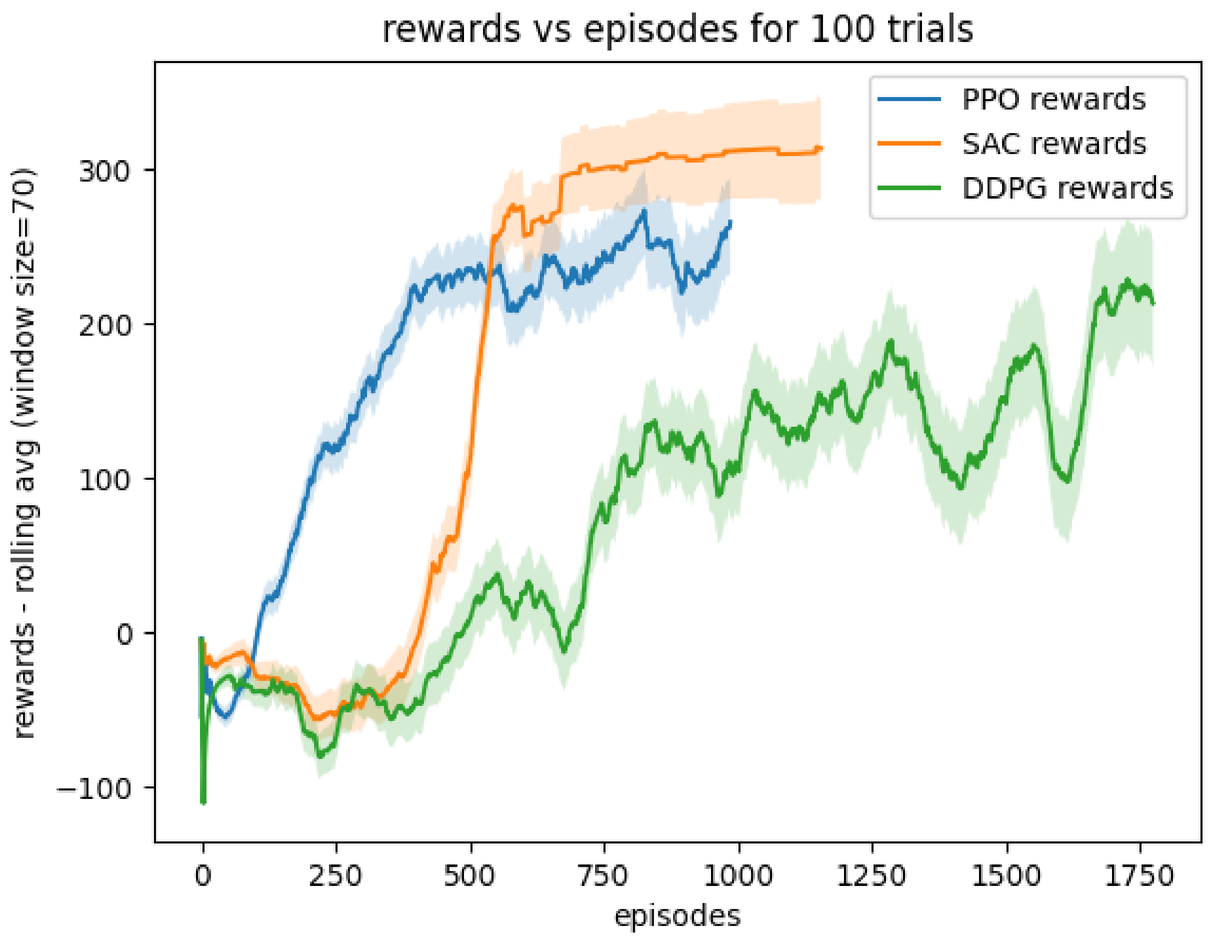


DDPG: reward 76, length 514

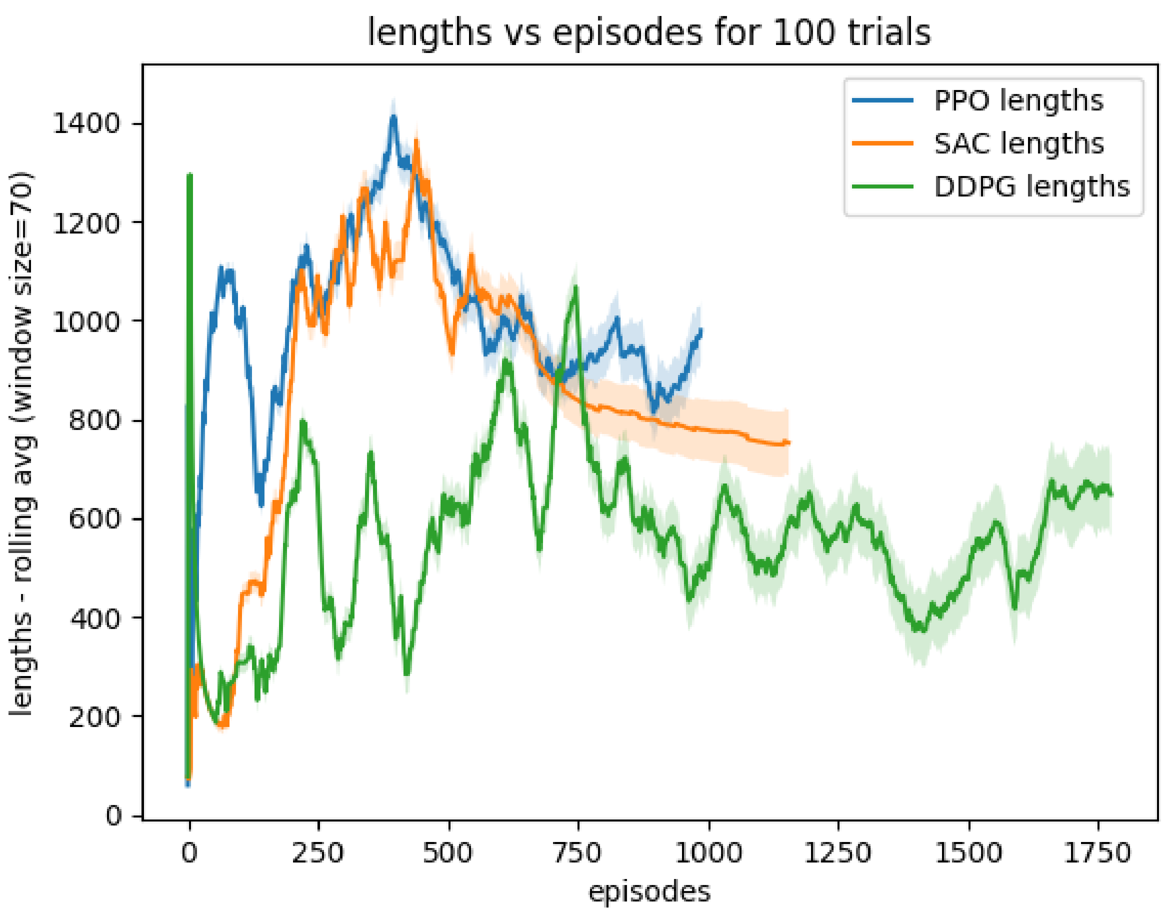


DDPG takes very small steps in order to move forward but can not finish the episode and falls down in the end. Although initially appearing stable, the gait proves to be inherently unstable as the episode progresses. The phase portrait plot reveals a zero-area, classifying it as a "conservative system," indicative of an attempt to conserve energy during movement. Being a conservative system does not inherently guarantee stability. The video makes it clear that the conservation of energy does not always correspond to a stable gait, highlighting the intricate challenges of balancing energy efficiency and gait stability concurrently.

Training Results



Analysing the rewards versus episodes curves for the three algorithms, it is evident that the SAC algorithm stands out as the most data-efficient. It attains the highest reward in the fewest number of episodes and demonstrates remarkable stability throughout training, characterized by a consistently increasing trend in reward values. Following SAC, PPO exhibits commendable data efficiency and training stability. Conversely, DDPG proves to be the least data-efficient, requiring an additional 700 episodes compared to SAC and PPO to approach a similar reward value. Moreover, DDPG exhibits the highest level of instability during training, marked by notable fluctuations in the reward values.



Examining the lengths versus episodes plot, it is discernible that SAC initially achieves a substantial length value of around 1400 but subsequently decreases to a smaller value of approximately 1000 towards the end. This suggests an initial adoption of a sub-optimal policy to reach the episode's end in the middle, followed by the discovery of a more effective policy enabling a quicker completion in fewer steps. In contrast, PPO exhibits a similar trend but fails to surpass SAC in reaching the end of the episode more rapidly. Notably, DDPG completes the episode at approximately 600 but, crucially, does not reach the episode's conclusion, signifying a fall at the end.

Conclusion

In conclusion, SAC emerged as the most successful algorithm, swiftly solving the environment with the shortest training time, presenting a remarkably stable and natural gait, showcasing high sample efficiency, and maintaining consistent stability during training. On the other hand, DDPG faced challenges in completing the episode, maintained a conservative gait, displayed the lowest sample efficiency, and exhibited the least stability during training. PPO struck a balance between the two, demonstrating quicker training times and achieving a relatively natural-looking gait. However, it did not escape some degree of instability in its walking pattern, highlighting the delicate trade-off between training efficiency and gait stability in reinforcement learning algorithms.

References

- https://www.gymlibrary.dev/environments/box2d/bipedal_walker/
- spinningup.openai.com
- Choi et al. “Walking-Speed-Adaptive Gait Phase Estimation for Wearable Robots” MDPI 2023
- Schulman et al. “Proximal Policy Optimization Algorithms” arXiv preprint 2017
- Lillicrap et al. “Continuous Control with Deep Reinforcement Learning” arXiv preprint 2016
- Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor” ICML 2018