Cart Pole - Individual Task

Varshitha Purra
Email: vpurra@asu.edu

***Abstract*—This project evaluates the performance of four reinforcement learning algorithms—SAC, TD3, DDPG, and D4PG—on the Continuous CartPole problem. The algorithms are compared based on cumulative rewards, convergence speed, and stability to identify their strengths and weaknesses. Results indicate that SAC outperforms other methods due to its effective exploration-exploitation balance, followed by TD3, which offers stable learning and computational efficiency. DDPG demonstrates fast convergence but suffers from high variability, while D4PG, though theoretically robust, is computationally intensive. This study highlights the trade-offs between performance, stability, and computational cost, providing insights for selecting RL algorithms for continuous control tasks.**

***Index Terms*—Reinforcement Learning, Continuous CartPole, Soft Actor-Critic (SAC), Twin Delayed Deep Deterministic Policy Gradient (TD3), Deep Deterministic Policy Gradient (DDPG), Distributed Distributional Deterministic Policy Gradient (D4PG), Convergence Speed, Stability, Cumulative Rewards, Continuous Control.**

I. Introduction

The cart-pole problem, also known as the inverted pendulum problem, is a foundational benchmark in reinforcement learning (RL) research, frequently utilized to test and validate novel RL algorithms. This classic control problem, first described by Barto, Sutton, and Anderson in their seminal paper *Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems*, entails balancing a pole hinged to a cart that can move along a frictionless track. The task is inherently unstable, requiring continuous control of the cart to maintain the pole's upright position.

The state space of this problem is represented by a four-dimensional continuous vector: the cart's position $x \in [-4.8, 4.8]$, the cart's velocity $\dot{x} \in (-\infty, \infty)$, the pole's angle $\theta \in [-0.418, 0.418]$ radians (approximately $\pm 24°$), and the pole's angular velocity $\dot{\theta} \in (-\infty, \infty)$ [**gymnasium˙cartpole**]. The action space is discrete, comprising two possible actions: 0 to push the cart to the left and 1 to push the cart to the right. The reward function incentivizes the system to keep the pole upright by assigning a reward of $+1$ for every timestep the pole remains within the predefined angle range of ± 0.2095 radians (approximately $\pm 12°$) and the cart within ± 2.4 units from the track's center. The episode terminates either when these limits are exceeded or after a maximum of 500 timesteps.

Historically, this problem has been addressed using various RL methods, from simple Q-learning to advanced deep reinforcement learning algorithms such as Deep Deterministic Policy Gradient (DDPG), Twin Delayed Deep Deterministic Policy Gradient (TD3), Distributed Distributional Deterministic Policy Gradient (D4PG), and Soft Actor-Critic (SAC). These methods leverage function approximators to handle the high-dimensional continuous state space and optimize the policy and value functions through gradient-based updates. This project aims to implement these four algorithms, analyze their performance on the cart-pole environment, and provide a comparative evaluation based on RL-specific metrics such as sample efficiency, cumulative reward, and stability of learning.

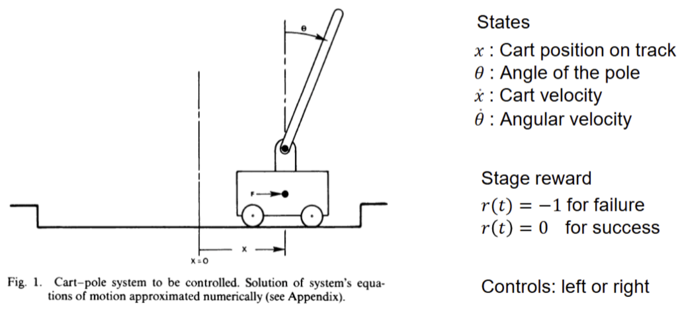
θ
F
x = 0
x
Fig. 1. Cart–pole system to be controlled. Solution of system's equations of motion approximated numerically (see Appendix).
States
x : Cart position on track
θ : Angle of the pole
$\dot{x}$: Cart velocity
$\dot{\theta}$: Angular velocity
Stage reward
$r(t) = -1$ for failure
$r(t) = 0$ for success
Controls: left or right

Fig. 1. Cart-Pole System

II. DDPG

The DDPG architecture consists of two primary neural networks: the actor network and the critic network. The actor is a feedforward neural network that takes the state as input and outputs a continuous action. To ensure the action lies within the valid range, a sigmoid activation function (logistic function) is applied at the final layer. This allows the actor to approximate binary control by mapping its outputs to the range $[0, 1]$, which are then discretized into two discrete actions based on a threshold.

The critic network evaluates the Q-value for a given state-action pair using a separate feedforward network. To stabilize training, DDPG utilizes target networks for both the actor and critic. These target networks are soft copies of the main networks and are updated incrementally using an exponential moving average. Additionally, a replay buffer stores past experiences to enable efficient off-policy learning.

The DDPG algorithm relies on the Bellman equation to compute the target for the critic network. The critic is trained to minimize the mean squared error between its predicted Q-values and the target Q-values, which are computed using the target critic and target actor networks. Simultaneously, the actor is updated by maximizing the expected Q-value of the actions it generates.

A. Pseudocode for DDPG

The pseudocode for the DDPG algorithm is as follows:

Algorithm 1 Deep Deterministic Policy Gradient (DDPG)

```
1: Initialize actor network $\mu(s|\theta^\mu)$ and critic network $Q(s,a|\theta^Q)$ with random weights
2: Initialize target networks $\mu'$ and $Q'$ with weights $\theta^{\mu'} \leftarrow \theta^\mu$, $\theta^{Q'} \leftarrow \theta^Q$
3: Initialize replay buffer $\mathcal{D}$
4: **for** each episode **do**
5:   Reset the environment and get the initial state $s_0$
6:   **for** each time step **do**
7:     Select action $a_t = \mu(s_t|\theta^\mu)$, where $\mu(s)$ applies a logistic function to output actions in $[0,1]$
8:     Execute action $a_t$ in the environment, mapping $a_t$ to discrete actions $\{0,1\}$ using a threshold (e.g., $a_t < 0.5 \to 0$, otherwise 1)
9:     Observe reward $r_t$, next state $s_{t+1}$, and done flag
10:    Store transition $(s_t, a_t, r_t, s_{t+1}, \text{done})$ in $\mathcal{D}$
11:    **if** enough samples in $\mathcal{D}$ **then**
12:      Sample minibatch of transitions $(s, a, r, s', \text{done})$ from $\mathcal{D}$
13:      Compute target $y = r + \gamma(1 - \text{done})Q'(s', \mu'(s'|\theta^{\mu'})|\theta^{Q'})$
14:      Update critic by minimizing loss $L = \frac{1}{N}\sum\left(Q(s,a|\theta^Q) - y\right)^2$
15:      Update actor by maximizing $J = \frac{1}{N}\sum Q(s, \mu(s|\theta^\mu)|\theta^Q)$
16:      Update target networks:
           $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
           $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
17:    **end if**
18:  **end for**
19: **end for**
```

B. Results and Analysis

The performance of DDPG on the continuous CartPole problem is illustrated in the reward curve obtained over 2000 episodes. The raw rewards (shown in blue) demonstrate significant variability throughout training, while the moving average reward (shown in red with a window size of 50) provides a smoother trend. The algorithm achieves consistently high rewards, often nearing the environment's reward cap of 500, particularly in the later episodes. This indicates that the agent is capable of learning an effective policy for balancing the pole. However, the results also reveal periods of instability, with noticeable drops in the raw rewards and irregular oscillations in the moving average rewards.

While DDPG shows promise in solving the CartPole problem, several factors may explain the fluctuations and imperfections in its performance:

- *Exploration Noise*: The algorithm introduces exploration noise (such as Gaussian noise) during action selection, which can lead to suboptimal actions, especially in early training or during periods of instability.
- *Sparse Rewards*: In the CartPole environment, the reward signal is sparse and only provides meaningful feedback when the agent successfully balances the pole for extended periods. This delayed reward signal can slow down learning and lead to unstable performance in the early training phases.
- *Continuous to Binary Action Mapping*: The logistic function in the actor network outputs continuous values in the range [0, 1], which are then discretized into binary actions based on a threshold. This mapping introduces some approximation error, leading to suboptimal actions that may negatively impact performance.
- *Replay Buffer Dynamics*: The replay buffer stores past transitions for off-policy updates. However, if the agent does not explore well consistently, older, less-relevant transitions may dominate the buffer, leading to overfitting to outdated policies and causing dips in performance.
- *Sensitivity to Hyperparameters*: DDPG is highly sensitive to hyperparameters such as learning rates, discount factor (γ), and soft update coefficient (τ). Suboptimal hyperparameter choices can lead to instability in the training process, as observed in the raw rewards.

While DDPG demonstrates the ability to solve the CartPole problem, achieving a moving average reward close to 450 in later episodes, its performance is hindered by instability and variability. Addressing these limitations could involve fine-tuning hyperparameters, improving the exploration strategy, or refining the action representation to reduce approximation errors.

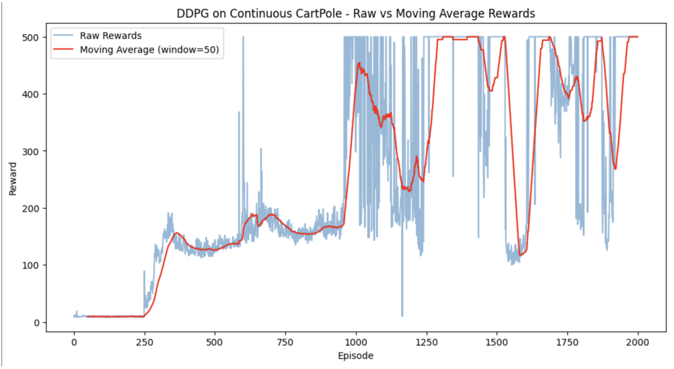

Fig. 2. DDPG performance on the CartPole problem

III. D4PG

The Distributed Distributional Deterministic Policy Gradient (D4PG) algorithm extends the DDPG framework by incorporating a distributional perspective in its critic network. The critic predicts a probability distribution over possible returns instead of a single scalar Q-value, using a fixed number of discrete atoms to approximate the distribution. This approach captures the uncertainty and variability of returns, enabling more robust policy updates. Additionally, D4PG introduces a projection step to map predicted distributions onto a fixed support, ensuring numerical stability and consistency in training.

In our implementation, the actor network outputs continuous values in the range $[0,1]$, using a logistic activation function, which are then discretized for binary actions in the CartPole environment. The critic network utilizes 51 discrete atoms to represent the return distribution, with a value range ($v_{\min} = 0$, $v_{\max} = 500$) chosen to match the rewards in the CartPole environment. A random seed (42) was employed to ensure reproducibility across all components. Despite its theoretical advantages, D4PG is computationally intensive due to the projection step, leading to slower training times compared to other algorithms.

A. *Pseudocode for D4PG*

Algorithm 2 Distributed Distributional Deterministic Policy Gradient (D4PG)

```
1:  Initialize actor network $\mu(s|\theta^\mu)$ and distributional critic network $Q(s,a|\theta^Q)$ with random weights
2:  Initialize target networks $\mu'$ and $Q'$ with weights $\theta^{\mu'} \leftarrow \theta^\mu$, $\theta^{Q'} \leftarrow \theta^Q$
3:  Initialize replay buffer $\mathcal{D}$
4:  for each episode do
5:      Reset the environment and get the initial state $s_0$
6:      for each time step do
7:          Select action $a_t = \mu(s_t|\theta^\mu)$, where $\mu(s)$ applies a logistic function to output actions in $[0,1]$
8:          Execute action $a_t$ in the environment, mapping $a_t$ to discrete actions $\{0,1\}$ using a threshold
9:          Observe reward $r_t$, next state $s_{t+1}$, and done flag
10:         Store transition $(s_t, a_t, r_t, s_{t+1}, \text{done})$ in $\mathcal{D}$
11:         if enough samples in $\mathcal{D}$ then
12:             Sample minibatch of transitions $(s, a, r, s', \text{done})$ from $\mathcal{D}$
13:             Compute target actions: $a' = \mu'(s'|\theta^{\mu'})$
14:             Predict target distribution: $Z' = Q'(s', a'|\theta^{Q'})$
15:             Project distribution onto support: $\tilde{Z} = \text{Project}(Z', r, \text{done})$
16:             Update critic by minimizing cross-entropy loss:
                    $$L_Q = -\frac{1}{N}\sum \tilde{Z} \cdot \log Q(s,a|\theta^Q)$$
17:             Update actor by maximizing: $J = \frac{1}{N}\sum Q(s, \mu(s|\theta^\mu)|\theta^Q)$
18:             Update target networks:
                    $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
                    $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
19:         end if
20:     end for
21: end for
```

B. *Results*

The results for the D4PG algorithm applied to the Continuous CartPole environment are depicted in Figure 3. Over 500 episodes, the algorithm demonstrates slow convergence and relatively low rewards compared to simpler algorithms like DDPG or TD3. The use of a distributional critic introduces significant computational overhead, particularly during the projection step, which accounts for its slower training.

Several factors contribute to the suboptimal performance of D4PG in this task:

- *Computational Overhead:* The projection of distributions onto a fixed support is computationally expensive, slowing down updates and reducing sample efficiency.
- *Sparse Rewards:* The CartPole environment provides sparse feedback, which may not fully exploit the advantages of distributional Q-values.
- *Limited Exploration:* The logistic activation function in the actor limits exploration, which may lead to slower discovery of optimal policies.
- *Sensitivity to Hyperparameters:* The performance of D4PG is sensitive to the choice of the number of atoms, the range of the support, and learning rates. Suboptimal settings can lead to poor convergence.

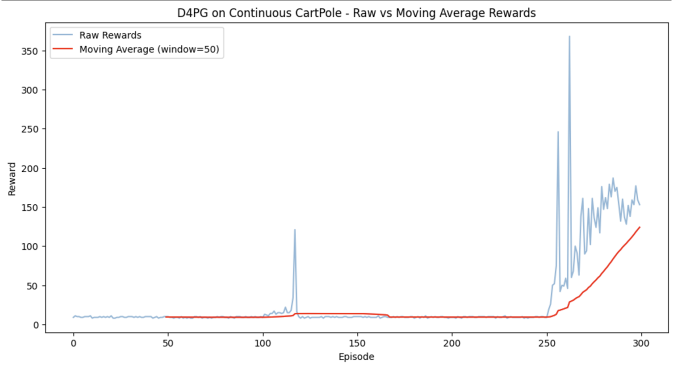

Fig. 3. D4PG performance on the Continuous CartPole problem.

IV. TD3

The architecture of the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm used in our implementation builds upon the DDPG framework with significant enhancements to improve performance and stability. TD3 leverages two independent critic networks to address the overestimation bias in Q-value estimation. Additionally, it incorporates a delayed policy update, where the actor network is updated less frequently than the critics, and target policy smoothing by adding clipped noise to the target actions. This ensures a more robust learning process.

The actor network outputs continuous actions in the range $[0,1]$ using a logistic function, which maps these actions to binary decisions based on a threshold for the discrete CartPole environment. The random seed 42 was used across all components to ensure reproducibility of results. The replay buffer stores past experiences, allowing for efficient off-policy updates.

A. *Pseudocode for TD3*

Algorithm 3 Twin Delayed Deep Deterministic Policy Gradient (TD3)

```
1: Initialize actor network $\mu(s|\theta^\mu)$ and two critic networks $Q_1(s,a|\theta^{Q_1})$, $Q_2(s,a|\theta^{Q_2})$
2: Initialize target networks $\mu'$, $Q'_1$, and $Q'_2$ with weights $\theta^{\mu'} \leftarrow \theta^\mu$, $\theta^{Q'_1} \leftarrow \theta^{Q_1}$, $\theta^{Q'_2} \leftarrow \theta^{Q_2}$
3: Initialize replay buffer $\mathcal{D}$
4: for each episode do
5:   Reset the environment and get the initial state $s_0$
6:   for each time step do
7:     Select action $a_t = \mu(s_t|\theta^\mu)$, where $\mu(s)$ applies a logistic function to output actions in $[0,1]$
8:     Execute action $a_t$ in the environment, mapping $a_t$ to discrete actions $\{0,1\}$ using a threshold
9:     Observe reward $r_t$, next state $s_{t+1}$, and done flag
10:    Store transition $(s_t, a_t, r_t, s_{t+1}, \text{done})$ in $\mathcal{D}$
11:    if enough samples in $\mathcal{D}$ then
12:      Sample minibatch of transitions $(s, a, r, s', \text{done})$ from $\mathcal{D}$
13:      Add clipped noise to target actions: $\tilde{a} = \mu'(s'|\theta^{\mu'}) + \epsilon$, where $\epsilon \sim \text{clip}(\mathcal{N}(0,\sigma), -c, c)$
14:      Compute target Q-value: $y = r + \gamma(1 - \text{done}) \min(Q'_1(s', \tilde{a}|\theta^{Q'_1}), Q'_2(s', \tilde{a}|\theta^{Q'_2}))$
15:      Update Critic 1 by minimizing loss: $L = \frac{1}{N}\sum(Q_1(s,a|\theta^{Q_1}) - y)^2$
16:      Update Critic 2 by minimizing loss: $L = \frac{1}{N}\sum(Q_2(s,a|\theta^{Q_2}) - y)^2$
17:      if step mod policy delay == 0 then
18:        Update Actor by maximizing: $J = \frac{1}{N}\sum Q_1(s, \mu(s|\theta^\mu)|\theta^{Q_1})$
19:        Update target networks:
             $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
             $\theta^{Q'_1} \leftarrow \tau\theta^{Q_1} + (1-\tau)\theta^{Q'_1}$
             $\theta^{Q'_2} \leftarrow \tau\theta^{Q_2} + (1-\tau)\theta^{Q'_2}$
20:      end if
21:    end if
22:  end for
23: end for
```

B. *Results*

The results of the TD3 algorithm for the Continuous CartPole environment are shown in Figure 4. The plot illustrates the raw rewards and the moving average rewards over 2,000 episodes. The algorithm demonstrates steady improvements in performance as training progresses. The delayed policy updates and the use of two critic networks ensure stability and mitigate the risk of overestimation bias.

However, the moving average rewards indicate periodic fluctuations, which could be attributed to noise in target policy smoothing and the delayed updates. Despite these challenges, TD3 outperforms simpler algorithms by achieving higher rewards consistently after a sufficient number of episodes.

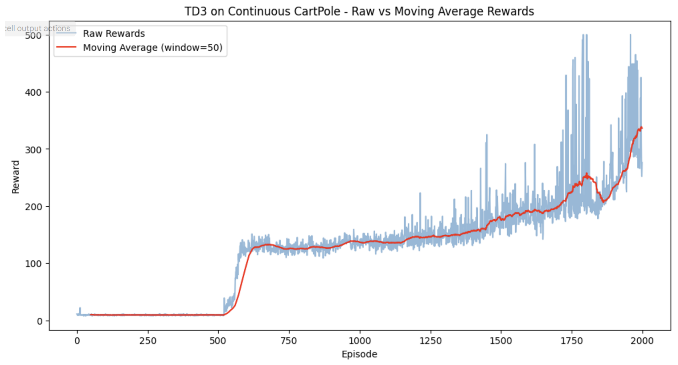

Fig. 4. TD3 performance on the CartPole problem

V. SAC

Soft Actor-Critic (SAC) is an advanced off-policy reinforcement learning algorithm that optimizes both the policy entropy and reward, thereby encouraging exploration while balancing the exploitation-exploration trade-off. In our implementation, SAC was adapted for the Continuous CartPole environment, where the action space was represented as continuous values in $[0,1]$. To map these values to binary actions ($\{0, 1\}$), a logistic function was employed at the output layer of the actor network, ensuring actions aligned with the environment's discrete nature.

The SAC framework consists of three primary networks:

- *Actor Network:* Outputs actions using a logistic function for smooth exploration.
- *Critic Networks:* Two separate Q-value networks are employed to mitigate overestimation bias.
- *Value Network:* Estimates the expected return of states and serves as a baseline for policy evaluation.

All networks were initialized with a fixed random seed (`SEED = 42`) to ensure reproducibility. The critic and actor networks were trained using Adam optimizers with a learning rate of 3×10^{-4}. Soft updates for the target value network were performed using a smoothing parameter $\tau = 0.005$.

The SAC algorithm prioritizes entropy maximization, which ensures policies remain stochastic for effective exploration. In our implementation, an entropy coefficient ($\alpha = 0.01$) was manually set to balance exploration and reward optimization.

A. Pseudocode for SAC

Algorithm 4 Soft Actor-Critic (SAC)

```
1: Initialize actor network $\pi(a|s;\theta^{\pi})$, critic networks $Q_1(s,a;\theta^{Q_1})$, $Q_2(s,a;\theta^{Q_2})$, value network $V(s;\theta^{V})$, and target value network $\hat{V}(s;\theta^{\hat{V}})$
2: Initialize replay buffer $\mathcal{D}$
3: Set target network weights $\theta^{\hat{V}} \leftarrow \theta^{V}$
4: for each episode do
5:     Reset environment and get initial state $s_0$
6:     for each time step do
7:         Sample action $a_t \sim \pi(s_t;\theta^{\pi})$, with logistic output to approximate binary actions
8:         Execute action $a_t$, observe $r_t$, $s_{t+1}$, and done flag
9:         Store transition $(s_t, a_t, r_t, s_{t+1}, \text{done})$ in $\mathcal{D}$
10:        if buffer size > batch size then
11:            Sample minibatch $(s, a, r, s', \text{done})$ from $\mathcal{D}$
12:            Compute target $y = r + \gamma(1-\text{done})\hat{V}(s';\theta^{\hat{V}})$
13:            Update critic networks by minimizing:
                   $L_{Q_i} = \mathbb{E}\left[\left(Q_i(s,a;\theta^{Q_i}) - y\right)^2\right], \quad i \in \{1,2\}$
14:            Compute target for value network:
                   $V_{\text{target}}(s) = \mathbb{E}\left[\min_i Q_i(s,a';\theta^{Q_i}) - \alpha \log \pi(a'|s;\theta^{\pi})\right]$
15:            Update value network by minimizing:
                   $L_V = \mathbb{E}\left[\left(V(s;\theta^{V}) - V_{\text{target}}(s)\right)^2\right]$
16:            Update actor network by minimizing:
                   $L_{\pi} = \mathbb{E}\left[\alpha \log \pi(a|s;\theta^{\pi}) - \min_i Q_i(s,a;\theta^{Q_i})\right]$
17:            Perform soft updates for target value network:
                   $\theta^{\hat{V}} \leftarrow \tau\theta^{V} + (1-\tau)\theta^{\hat{V}}$
18:        end if
19:    end for
20: end for
```

B. Results and Discussion

The SAC algorithm was trained on the Continuous CartPole environment for 2000 episodes. The results, as depicted in Figure 5, show both the raw rewards and their moving average (window size of 50). The moving average demonstrates that SAC achieved moderate stability and gradual improvement over episodes.

Possible reasons for high variance include:

- *Entropy Regularization:* SAC's stochastic policy, while beneficial for exploration, can lead to suboptimal actions in environments with deterministic dynamics like CartPole.
- *Continuous-to-Binary Mapping:* The logistic function used to approximate binary actions may introduce mapping errors, affecting policy execution.
- *Sparse Rewards:* CartPole's reward function provides sparse feedback, which, coupled with entropy maximization, can hinder performance.
- *Hyperparameter Sensitivity:* SAC's performance is sensitive to α, the entropy coefficient. A fixed value of $\alpha = 0.01$ may not have been optimal for this task.

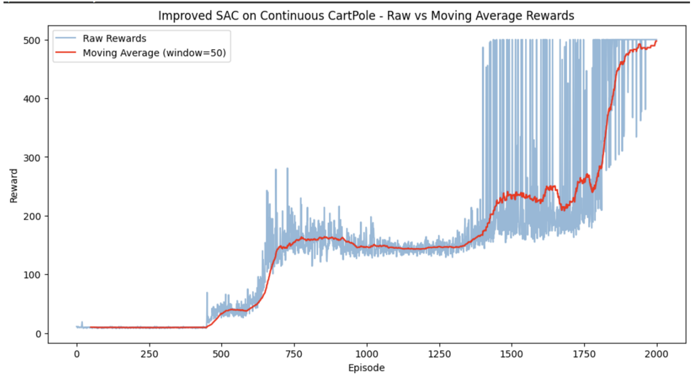

Fig. 5. SAC on Continuous CartPole: Raw vs Moving Average Rewards

VI. Comparison of Results

This section compares the performance of four reinforcement learning (RL) algorithms: Deep Deterministic Policy Gradient (DDPG), Distributed Distributional DDPG (D4PG), Twin Delayed Deep Deterministic Policy Gradient (TD3), and Soft Actor-Critic (SAC), on the Continuous CartPole environment. Performance was evaluated using the following metrics:

- **Cumulative Rewards vs Episodes**
- **Convergence Speed**
- **Stability (Standard Deviation of Rewards)**

A. Cumulative Rewards vs Episodes

Cumulative rewards measure the total reward obtained in an episode, smoothed over a moving window to highlight trends over time. The reward for an episode i is computed as:

$$R_i = \sum_{t=1}^{T} r_t,$$

where r_t is the immediate reward at timestep t, and T is the length of the episode.

The smoothed rewards are calculated using a moving average with a window size W:

$$\text{Smoothed Reward}_i = \frac{1}{W} \sum_{j=i}^{i+W-1} R_j.$$

This metric visualizes how well each algorithm learns over episodes and reflects sample efficiency.

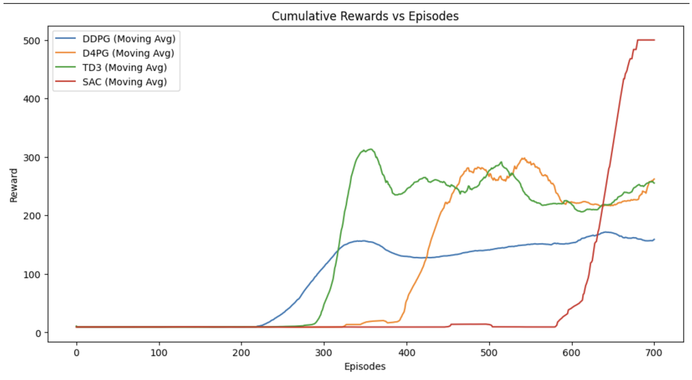

Fig. 6. Cumulative Rewards vs Episodes (Moving Average)

B. Convergence Speed

Convergence speed is defined as the episode number e_c where the cumulative reward consistently exceeds a predefined threshold $R_{\text{threshold}}$ (e.g., 300). Formally:

$$e_c = \min\left(e : \forall e' \geq e, R_{e'} \geq R_{\text{threshold}}\right).$$

This metric measures how quickly each algorithm achieves stable high performance.

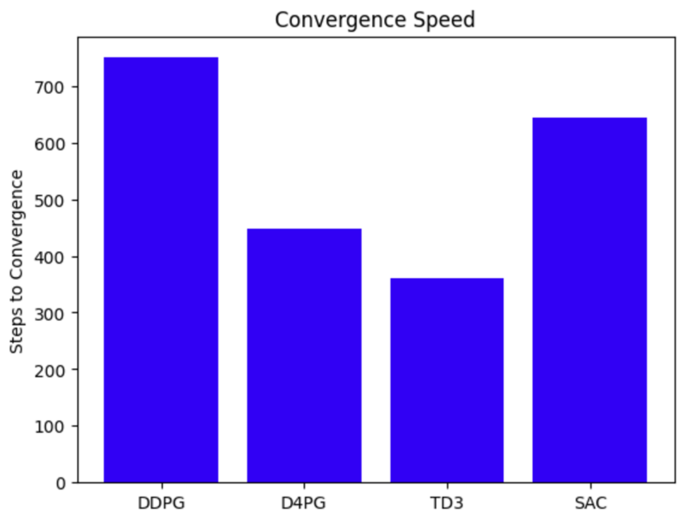

Fig. 7. Convergence Speed: Steps to Reward Threshold

C. Stability (Standard Deviation of Rewards)

Stability reflects the consistency of the rewards across episodes. It is computed as the standard deviation of cumulative rewards:

$$\text{Stability} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(R_i - \bar{R})^2},$$

where $\bar{R}$ is the mean reward, and N is the total number of episodes.

Lower standard deviation indicates better stability, signifying consistent performance.

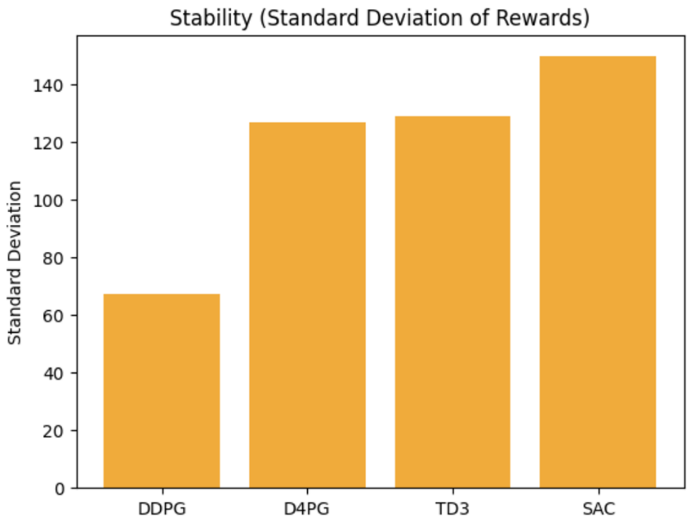

Fig. 8. Stability: Standard Deviation of Rewards

Cumulative Rewards vs Episodes:
The results show that SAC achieves higher rewards faster, surpassing other algorithms just after 600 episodes. This can be attributed to SAC's entropy regularization, which encourages exploration and results in a more robust policy. TD3, while steadily improving, lacks the same level of exploratory adaptability, leading to slower convergence. Both DDPG and D4PG exhibit significant variance due to their deterministic nature, which makes them more sensitive to local optima and less resilient to noise in the environment. SAC's balance of exploration and exploitation explains its superior and consistent performance.

Convergence Speed:
DDPG demonstrates the fastest convergence despite significant variability in cumulative rewards, followed by SAC and D4PG. TD3, although slower in convergence, shows steady improvement and more stable training dynamics. The observed differences in implementation complexity and runtime efficiency are notable: D4PG is computationally intensive, taking over 60 minutes for 300 episodes, while TD3 efficiently completes 2000 episodes in just 30 minutes. This disparity arises from D4PG's reliance on distributional Q-learning, which computes a probability distribution over returns, increasing both memory and computational demands. In contrast, TD3's design focuses on computational efficiency by reducing overestimation bias with clipped double Q-learning and delayed policy updates, making it faster and easier to implement.

Stability:
SAC exhibits the highest standard deviation among the algorithms, indicating greater variability in rewards across episodes despite its consistent improvement in performance. TD3 follows SAC in terms of standard deviation, reflecting moderate fluctuations but maintaining a steady upward trend in performance. D4PG and DDPG show lower standard deviations, with DDPG being the most stable. However, this apparent stability is misleading, as it stems from a lack of consistent improvement and a tendency to converge prematurely to

suboptimal policies. The variability observed in SAC and TD3 is indicative of their ability to explore the environment more effectively, ultimately leading to better long-term performance.

VII. Conclusion and Future Work

Based on the metrics analyzed, SAC demonstrates the most balanced performance, with superior cumulative rewards, steady convergence, and robust exploration driven by entropy regularization. While it exhibits higher variability in rewards, this variability supports effective exploration, allowing SAC to achieve long-term optimal performance. TD3 ranks just below SAC, showing steady improvement and stable performance. Its lower convergence speed is compensated by its computational efficiency and reduced sensitivity to overestimation bias, making it a practical and reliable choice for many RL problems.

DDPG converges quickly but suffers from high variability and lower overall rewards, indicating suboptimal exploration and a tendency to get stuck in local optima. Despite its deterministic nature, DDPG's simplicity and fast runtime make it suitable for tasks where efficiency is prioritized over ultimate performance. D4PG, while conceptually powerful due to its distributional approach, is the hardest to implement and computationally expensive, requiring significant runtime for even a modest number of episodes. Future work could explore hybrid methods that incorporate SAC's exploration with the computational efficiency of TD3, or improve D4PG's runtime efficiency to make it more practical for real-world applications.

VIII. Links

Here is a link to the code: Code Link

Barth-Maron, Gabriel, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. 2018. “Distributed Distributional Deterministic Policy Gradients.” https://arxiv.org/abs/1804.08617.

Barto, Andrew G., Richard S. Sutton, and Charles W. Anderson. 1983. “Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems.” *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13 (5): 834–46. https://doi.org/10.1109/TSMC.1983.6313077.

Fujimoto, Scott, Herke van Hoof, and David Meger. 2018. “Addressing Function Approximation Error in Actor-Critic Methods.” https://arxiv.org/abs/1802.09477.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.” https://arxiv.org/abs/1801.01290.

Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2019. “Continuous Control with Deep Reinforcement Learning.” https://arxiv.org/abs/1509.02971.