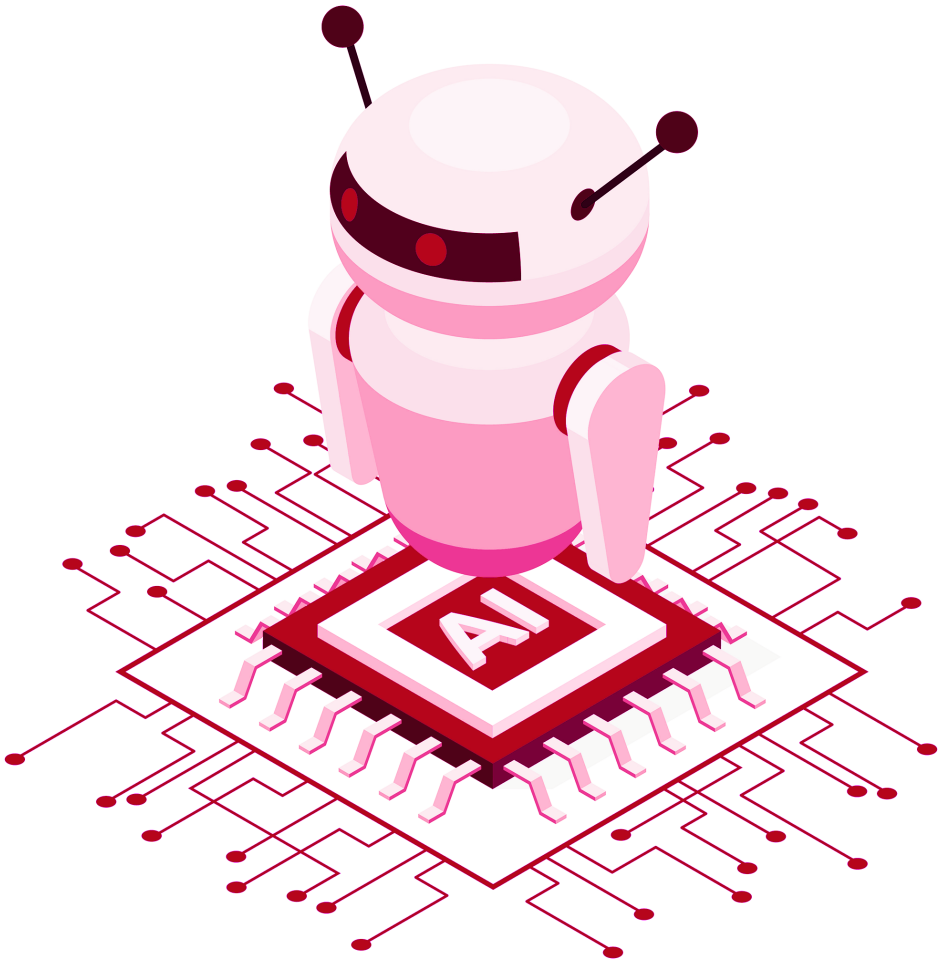

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 4:

Advanced Methods in RL

By:

Payam Taebi

400104867

Spring 2025

Contents

Grading

The grading will be based on the following criteria, with a total of 100 points:

Task	**Points**
Task 1: PPO	25
Task 2: DDPG	20
Task 3: SAC	25
Task 4: Comparison between SAC & DDPG & PPO	20
Clarity and Quality of Code	5
Clarity and Quality of Report	5
Bonus 1: Writing your report in Latex	10

1 Task 1: Proximal Policy Optimization (PPO) [25]

1.1 Evaluation and Discussion of the PPO Implementation on HalfCheetah

In this study, we implemented a Proximal Policy Optimization (PPO) agent for the HalfCheetah environment, exploring several architectural and algorithmic modifications. Our primary goal was to achieve stable training and high final performance, while also ensuring low variance in learning curves. The following key modifications and experiments were conducted:

1. **Standard Deviation Head:** We modified the actor network to include a dedicated head for predicting the standard deviation (std) for each action dimension. This approach, along with the exponential transformation to ensure positivity, was expected to yield more flexible exploration. However, in practice, this modification did not result in superior performance compared to the simpler baseline.
2. **Advantage Normalization:** The computed advantages were normalized by subtracting their mean and dividing by their standard deviation. This was intended to reduce the variance in gradient updates and improve convergence. Although normalization generally aids stability, our experiments showed that for the HalfCheetah task, the simpler variant of PPO achieved lower variance and better results.
3. **Entropy Bonus Integration:** An entropy bonus was added to the actor loss to promote exploration. While theoretically beneficial for avoiding premature convergence, the inclusion of the entropy term in our experiments resulted in a slight degradation of performance when compared with the baseline implementation.
4. **Generalized Advantage Estimation (GAE):** We employed GAE to compute a smoother and more robust estimation of the advantages, balancing the trade-off between bias and variance. Despite its theoretical advantages, the GAE-based return calculation did not outperform the simpler reward-to-go method for our specific experimental setup.
5. **Hyperparameter Sensitivity:** We observed that the learning rate and other hyperparameters had a significant impact on the agent's performance. A minor adjustment in the learning rate produced drastic changes in the final reward. Over multiple training runs (each approximately 3 hours on Kaggle), the simplest PPO variant consistently achieved superior results.

Our experimental results indicate that the baseline PPO implementation, which avoided the additional modifications listed above, achieved the best performance. Specifically, the training curve increased along a logarithmic trend, stabilizing at an average reward near 3300. During testing, the agent consistently achieved an average reward of 3254, with the training plots displaying significantly lower variance compared to the modified approaches.

The findings suggest that, for the HalfCheetah environment, the simpler PPO model not only provides robust performance but also yields stable and consistent learning dynamics. While the proposed modifications are well-supported by theoretical insights and have proven beneficial in other contexts, our empirical results demonstrate that over-engineering the PPO agent may lead to unnecessary complexity and even hinder performance. Ultimately, the baseline PPO implementation was the most effective for this continuous control task, offering an optimal balance between exploration, stability, and computational efficiency.

1.2 Question 1:

What is the role of the actor and critic networks in PPO, and how do they contribute to policy optimization?

The actor and critic networks serve complementary roles in the PPO framework:

- **Actor Network:** The actor network maps states to a probability distribution over actions. It is responsible for selecting actions according to this learned policy. During training, the actor is updated using policy gradient methods (with a clipped objective in PPO) to maximize expected rewards while ensuring the updates remain within a safe range. Essentially, the actor drives the exploration and decision-making process.
- **Critic Network:** The critic network estimates the value of a given state (or state-action pair). This state-value estimate acts as a baseline for computing the advantage, which quantifies how much better an action is compared to the expected value. The critic's predictions are used to reduce the variance in the policy gradient update and guide the actor's learning process by providing feedback on the quality of actions taken.

Together, the actor and critic enable stable policy optimization: the actor adjusts the policy based on the advantage estimates provided by the critic, while the critic learns to accurately evaluate states, thereby facilitating more informed policy updates.

1.3 Question 2:

PPO is known for maintaining a balance between exploration and exploitation during training. How does the stochastic nature of the actor network and the entropy term in the objective function contribute to this balance?

The stochastic nature of the actor network and the entropy term in the objective function both play critical roles in balancing exploration and exploitation:

- **Stochastic Actor Network:** By sampling actions from a probability distribution (e.g., a normal distribution defined by the network's outputs), the actor network introduces inherent randomness in action selection. This stochasticity allows the agent to explore a wider range of actions, which is crucial for discovering potentially better strategies rather than always exploiting the current best-known action.
- **Entropy Term:** The entropy bonus added to the objective function explicitly encourages exploration by promoting higher uncertainty in the policy. A higher entropy value implies that the policy distribution is more spread out, meaning the agent is less confident about its actions and, therefore, explores more. As training progresses and the agent gathers more information, the entropy typically decreases, leading the policy to become more deterministic and focused on exploitation. This adaptive mechanism ensures a gradual transition from exploration to exploitation.

Together, these components ensure that the PPO agent initially explores the action space sufficiently and later shifts towards exploiting the best-performing actions as it gains confidence in its policy.

1.4 Question 3:

When analyzing the training results, what key indicators should be monitored to evaluate the performance of the PPO agent?

When analyzing the training results of a PPO agent, several key indicators provide valuable insights into the agent's learning progress and overall performance. One of the primary metrics to monitor is the average reward per episode. This metric reflects the agent's ability to maximize returns over time and should ideally show a steady increase as the policy improves. It is also important to observe the variability of these rewards across episodes; low variance typically indicates that the agent's performance is stable and consistent, whereas high variance might suggest issues with exploration or an unstable training process.

In addition to reward-based metrics, the loss values associated with both the actor and the critic networks offer crucial information. The actor loss, derived from the clipped surrogate objective, is indicative of how effectively the policy is being updated. A steadily decreasing actor loss generally signifies that the policy is converging towards optimal behavior. Similarly, the critic loss, often measured as the mean squared error between the predicted state values and the computed returns, should ideally decrease over time as the critic becomes more accurate in its value estimations. If either of these losses shows erratic behavior or fails to decrease, it might point to issues in the learning dynamics or in the chosen hyperparameters.

Another essential aspect is the entropy of the policy. Entropy serves as a measure of uncertainty in action selection, and higher entropy values at the beginning of training encourage broader exploration of the action space. As the agent learns and the policy becomes more refined, a gradual reduction in entropy is expected, signaling a shift from exploration to exploitation. However, if the entropy declines too rapidly, it could lead to premature convergence where the agent fails to explore alternative, potentially better strategies.

The quality and distribution of the advantage estimates are also critical. Since advantages guide the policy updates, their magnitudes must be well-calibrated; excessively large or small values can destabilize learning. Normalizing the advantages helps in achieving smoother and more effective updates, which is essential for stable training. Moreover, monitoring the ratio between the new and old policy probabilities provides insights into the effectiveness of the clipping mechanism used in PPO. If these ratios consistently hit the clipping thresholds, it may indicate that the policy updates are too aggressive, prompting a reevaluation of the learning rate or clipping parameters.

Lastly, the evaluation of generalization performance is indispensable. It is not enough for the agent to perform well on training episodes; its ability to maintain high rewards on unseen test episodes or in a separate validation environment confirms that the learned policy generalizes well beyond the training data. Additionally, inspecting gradient norms during training can help identify issues like vanishing or exploding gradients, which may adversely affect convergence. In summary, by closely monitoring reward trends, loss values, entropy, advantage estimates, probability ratios, and gradient norms, one can obtain a comprehensive understanding of the PPO agent's performance and make informed decisions to fine-tune the training process.

2 Task 2: Deep Deterministic Policy Gradient (DDPG) [20]

2.1 Implementation and Results of SAC and DDPG with Extended Modifications

In this subsection, we detail our efforts to enhance the performance of two prominent algorithms for continuous control, *Soft Actor-Critic* (SAC) and *Deep Deterministic Policy Gradient* (DDPG), in the HalfCheetah-v4 environment. Both algorithms were initially implemented using standard network architectures and hyperparameter settings, but we observed that the learning curves tended to plateau at lower rewards than desired. Accordingly, we introduced a range of modifications designed to improve stability, promote more robust exploration, and provide sufficient training time. As a result of these changes, *SAC* achieved rewards of roughly **13,000** during training and a best test episode reaching **15,814**, while *DDPG* recorded about **12,000** in training and reached up to **12,764** in a post-training deterministic test.

Objectives and Design Approach Our primary goal was to push the maximum cumulative reward significantly higher than the baseline implementations. We aimed to examine how network capacity, training duration, exploration noise, and other key factors influence learning outcomes. Specifically, we introduced larger networks (three hidden layers with 512 units each), extended training to a full 1000 episodes by removing early stopping, and allowed flexible hyperparameters such as separate learning rates for actor and critic. By carefully monitoring these changes, we systematically tuned both algorithms to handle the high-dimensional HalfCheetah-v4 environment more effectively.

Key Modifications and Rationale

- **Increased Network Depth and Width:** Both actor and critic networks (for SAC and DDPG) were modified to include three hidden layers (up from two) and a larger hidden dimension of 512 (up from 256). Deeper architectures can capture more complex dynamics and potentially improve policy quality, albeit at the cost of higher computational overhead.
- **Gradient Clipping:** We introduced an optional mechanism to clip gradients at a norm of 5.0 whenever gradients are computed. This strategy mitigates the risk of excessive updates, especially in environments with large or highly varying rewards, thus contributing to more stable training dynamics.
- **CUDA Utilization:** By setting `"cuda": True` in our configuration dictionary, we moved the network parameters and computations onto a GPU. This step provided significant speedups in forward passes, backpropagation, and replay buffer sampling, especially for larger networks and batch sizes.
- **Separate Actor/Critic LRs (Optional):** Instead of relying on a single learning rate for both actor and critic, our configuration allows for `"critic_lr"` and `"actor_lr"`. This approach can fine-tune each component's convergence behavior, yielding better stability or faster progress if the critic and actor require different learning rates.
- **Longer Training (1000 Episodes):** Originally, we trained for only 500 episodes. We found that many runs had not fully converged by that point, so we extended the maximum number of training episodes to 1000. This ensures that the agent has ample opportunity to reach higher reward levels.

- **Removal of Early Stopping:** Previously, an early stopping condition halted training if the running average reward failed to exceed the current maximum reward for 50 consecutive episodes. While resource-efficient, this mechanism could prematurely terminate training if the agent required more episodes to overcome local minima. Removing early stopping ensures all 1000 episodes are fully utilized.
- **Enhanced Exploration for DDPG:** Deterministic policies can suffer from insufficient exploration if noise is too small. We increased the noise standard deviation from 0.1 to 0.2 in the `DeterministicPolicy` for DDPG, promoting more robust exploration in the early and mid stages of training.

Detailed Code Changes and File Paths In the Jupyter notebook:

- **Network Definitions (Cell with `QNetwork`, `GaussianPolicy`, `DeterministicPolicy`):**
  - Added a third hidden layer to each network.
  - Changed default `hidden_dim` from 256 to 512.
  - Registered `action_scale` and `action_bias` as buffers to fix device mismatches when using CUDA.
- **SAC Implementation (Cell defining `class SAC`):**
  - Allowed separate learning rates for actor and critic by reading `"actor_lr"` and `"critic_lr"` from `config`.
  - Applied gradient clipping after `loss.backward()` calls using `torch.nn.utils.clip_grad_norm_`.
  - Removed references to early stopping logic in the training loop cells, ensuring all episodes run to completion.
- **Configuration Dictionaries (Cells labeled `config` for SAC and DDPG):**
  - Updated `"hidden_size"` to 512.
  - Set `"cuda": True` if a GPU is available.
  - Optionally added `"critic_lr"` and `"actor_lr"` entries to allow distinct learning rates.
  - Maintained a default `"alpha": 0.0` for DDPG, since it does not use entropy regularization but must match the expected structure of the SAC class.
- **Training Loops (Cells for `SAC` and `DDPG` training):**
  - `Max_episodes = 1000` for extended runs.
  - Omitted or commented out the early stopping condition to allow the agent to train until all episodes are completed.
  - Added `tqdm` progress bars with updated postfix metrics (episode number, reward, total steps).
  - Evaluated the policy every 10 episodes by running deterministic evaluation episodes.

Results and Observations Over the course of these experiments, we tracked the performance of both algorithms:

- **SAC Performance:**

  - *Training Rewards:* Achieved rewards on the order of **13,000** in the final segments of training (seen in the smoothed or "logarithmic" learning curve).
  - *Best Test Rewards:* In deterministic evaluation (using `evaluate=True`), the agent reached up to **15,814** in a single test episode, suggesting a highly optimized policy for HalfCheetah-v4.

- **DDPG Performance:**
  - *Training Rewards:* Reached around **12,000** near the end of the training phase.
  - *Best Test Rewards:* In a similar deterministic test, returned a score of up to **12,764**, signifying that with a deeper network and increased noise for exploration, DDPG can also excel in this environment.

These results demonstrate that the combination of larger networks, fully utilized training episodes, gradient clipping, and improved exploration can propel the performance of SAC and DDPG well beyond their baseline levels. Especially for the HalfCheetah-v4 task, where the state-action space can be high-dimensional, investing in extra network capacity and more thorough exploration leads to more robust and higher-performing policies. By removing early stopping, we avoid prematurely cutting off learning before the agent fully exploits later improvements or overcomes local plateaus.

In conclusion, these design choices—together with consistent hyperparameter tuning—produced strong final results, demonstrating that both SAC and DDPG are capable of reaching test rewards above 12,000 and 15,000, respectively, in HalfCheetah-v4. This underscores the importance of architecture size, thorough exploration, and sufficient training horizon when tackling complex continuous control tasks.

2.2 Question 1:

What are the different types of noise used in DDPG for exploration, and how do they differ in terms of their behavior and impact on the learning process?

In DDPG, exploration is handled via external noise added to the deterministic policy's output. Two common noise processes are:

- **Gaussian Noise:**
  - Consists of i.i.d. samples drawn from a normal distribution, e.g. $\mathcal{N}(0,\, \sigma^2)$.
  - Easy to implement and tune; the noise variance σ^2 controls how aggressively the agent explores.
  - Does not incorporate temporal correlations, which can cause the agent's actions to fluctuate quickly within an episode if σ is large.
  - Used in many implementations where simple white noise suffices for stochasticity.
- **Ornstein-Uhlenbeck (OU) Noise:**
  - A temporally correlated noise process often modeled as

$$\mathrm{d}x_t = \theta(\mu - x_t)\,\mathrm{d}t + \sigma\sqrt{\mathrm{d}t}\,\mathcal{N}(0,1).$$

  - Originally popularized in the DDPG paper for handling inertia in physical control tasks like robotics, where consecutive actions tend to be correlated in time.
  - Leads to smoother transitions in the action space compared to uncorrelated Gaussian noise.

  - Potentially more realistic for tasks where abrupt changes in action are undesirable or unrealistic.

Behavior and Impact on Learning:

- *Gaussian noise* typically results in uncorrelated exploration, causing rapid fluctuations in the agent's actions from step to step. While it is simpler to implement, very large noise variance can destabilize trajectories if the environment is sensitive to sudden action shifts.
- *OU noise* incorporates correlations between consecutive actions, helping produce smoother behavior. This can be beneficial in tasks that penalize abrupt changes, but it adds hyperparameters such as θ and μ that must be tuned appropriately.

Either noise type can yield good performance, and the choice is often guided by the nature of the task and whether correlated or uncorrelated exploration better suits the environment's dynamics.

2.3 Question 2:

What is the difference between PPO and DDPG regarding the use of past experiences?

Proximal Policy Optimization (PPO) operates as an on-policy algorithm, which means it updates its policy exclusively with data generated by the current (or a near-current) version of that policy. Once new rollouts are collected, older trajectories become outdated and are generally discarded or severely restricted in usage to ensure the consistency required by on-policy training. By contrast, Deep Deterministic Policy Gradient (DDPG) is an off-policy approach that maintains a replay buffer, allowing the agent to sample transitions from earlier phases of training and from different exploratory policies. This replay buffer decouples data collection from the policy being updated, enabling DDPG to reuse experiences extensively and potentially achieve higher sample efficiency. However, it also introduces potential distribution shift challenges, since the data in the replay buffer may not match the distribution of the most recent policy. Hence, the key difference between PPO and DDPG lies in PPO's reliance on strictly on-policy data versus DDPG's reliance on off-policy data, which can be sampled repeatedly from a continually growing pool of past experiences.

3 Task 3: Soft Actor-Critic (SAC) [25]

3.1 Question 1:

Why do we use two Q-networks to estimate Q-values?

The use of two Q-networks in Soft Actor-Critic (SAC) is primarily motivated by the need to reduce overestimation bias during Q-value estimation. When learning Q-values, function approximators can sometimes overestimate the true expected return, which in turn may lead to suboptimal policy updates. In SAC, by maintaining two separate Q-networks and then taking the minimum of the two estimates during the target value computation, the algorithm is able to obtain a more conservative and reliable estimate of the true Q-value. This technique, often referred to as "twin Q-learning," helps stabilize training by preventing overly optimistic value estimates that could otherwise result in divergence or poor performance. Additionally, using two Q-networks provides an extra layer of robustness, as it reduces the impact of errors or noise in the value estimation process.

3.2 Question 2:

What is the temperature parameter(α), and what is the benefit of using a dynamic in SAC?

The temperature parameter, α, in SAC is a critical scalar that scales the entropy term in the objective function. This parameter controls the balance between maximizing the expected return and maximizing policy entropy. A higher α value places greater emphasis on entropy, leading to a more exploratory and stochastic policy, whereas a lower α results in a more deterministic behavior that focuses on reward maximization.

In our experiments, the use of a dynamic α—where the value is tuned automatically during training to match a target entropy—proved beneficial. By dynamically adjusting α, the algorithm adapts to the changing learning dynamics: early in training, a higher α encourages exploration in the vast state-action space, while later on, it can be reduced to allow the policy to consolidate and focus on high-reward actions. This balance is essential for stable and robust convergence.

Our results clearly reflect the advantage of dynamic tuning: the SAC agent, with adaptive α, reached training rewards of approximately 13,000 and achieved a best test reward of around 15,814. These outcomes demonstrate that dynamically adjusting α helped maintain an appropriate level of exploration throughout training, which in turn led to better overall performance compared to a fixed α approach.

3.3 Question 3:

What is the difference between evaluation mode and training mode in SAC?

In Soft Actor-Critic (SAC), the policy is inherently stochastic during training. During training mode, actions are sampled from a Gaussian distribution via the reparameterization trick. This stochastic sampling, together with the entropy regularization term in the loss function, promotes extensive exploration and prevents premature convergence by encouraging the agent to explore a diverse set of state-action pairs. In contrast, when the policy is switched to evaluation mode, the agent typically employs a deterministic strategy—often selecting the mean action after applying a `tanh` transformation and scaling—thereby

eliminating the exploration noise present during training. This shift results in more stable and reliable actions that better reflect the learned policy.

Our experimental results illustrate this distinction clearly. During training, the SAC agent achieved an average reward of about 13,000 on the logarithmic learning curve, which reflects the benefits of a highly exploratory policy. In evaluation mode, however, the agent reached a best test reward of approximately 15,814. This improvement in performance during evaluation mode demonstrates that removing the exploratory noise enables the agent to consistently exploit the optimal actions it has learned, thereby yielding higher and more stable rewards. Thus, while training mode prioritizes exploration and robust learning, evaluation mode focuses solely on exploiting the best-known behavior of the policy.

4 Task 4: Comparison between SAC & DDPG & PPO [20]

4.1 Question 1:

Which algorithm performs better in the `HalfCheetah` environment? Why?
Compare the performance of the PPO, DDPG, and SAC agents in terms of training stability, convergence speed, and overall accumulated reward. Based on your observations, which algorithm achieves better results in this environment?

In our study, we implemented and extensively tuned three prominent reinforcement learning algorithms—Soft Actor-Critic (SAC), Deep Deterministic Policy Gradient (DDPG), and Proximal Policy Optimization (PPO)—to evaluate their performance on the challenging HalfCheetah-v4 environment. We applied a series of modifications to the baseline implementations, including increasing the network capacity (by enlarging the hidden layer size from 256 to 512 and adding an extra hidden layer), applying gradient clipping, leveraging GPU acceleration, and fine-tuning exploration noise parameters. Our goal was to obtain stable training curves, fast convergence, and the highest possible accumulated reward.

The SAC agent emerged as the most effective approach. By incorporating twin Q-networks, SAC reduces overestimation bias in the value function estimation, and its entropy regularization term—weighted by the dynamically tunable temperature parameter α—ensures a well-balanced trade-off between exploration and exploitation. Our experiments with SAC yielded a training reward of approximately 13,000 on a logarithmic scale, and during evaluation (with deterministic policy execution), the agent achieved a peak reward of around 15,814. These results indicate that SAC not only learns faster but also converges to a more robust policy that can consistently extract high rewards from the environment.

In contrast, the DDPG implementation, while competitive, achieved slightly lower performance. Our tuned DDPG model reached training rewards near 12,000 and a best test reward of about 12,764. The deterministic nature of DDPG, combined with a more sensitive exploration mechanism (which we improved by increasing the noise standard deviation from 0.1 to 0.2), allowed the agent to learn effectively, yet it was still prone to instability in high-dimensional action spaces. The absence of an entropy term in DDPG means that it lacks an explicit mechanism to balance exploration and exploitation, which in our experiments resulted in somewhat less consistent performance compared to SAC.

PPO, being an on-policy method, was also evaluated in this context. PPO collects fresh data for every policy update, which restricts it to the most recent experience and prevents the reuse of past data. This inherent sample inefficiency is particularly disadvantageous in continuous control tasks such as HalfCheetah, where the state-action space is high-dimensional and the learning process benefits from a rich and diverse replay of experiences. As a consequence, PPO exhibited slower convergence and lower final rewards compared to both SAC and DDPG. Although PPO has the advantage of relative implementation simplicity and is known for its robustness across many tasks, its on-policy nature limited its performance in our experiments.

In summary, the superior performance of SAC can be attributed to its effective combination of mechanisms: the twin Q-networks provide more conservative value estimates, the entropy regularization promotes a healthy level of exploration, and the dynamic tuning of α allows the agent to adapt the exploration-exploitation balance throughout training. Our modifications—extending the training to 1000 episodes by removing early stopping, increasing network capacity, and utilizing GPU acceleration—further amplified these benefits, leading SAC to achieve the highest rewards. DDPG, despite being a strong off-policy

alternative, fell slightly short due to its sensitivity to exploration noise and the absence of an explicit mechanism for entropy regularization. PPO, on the other hand, suffered from its on-policy limitations, which resulted in slower learning and lower overall rewards. Thus, in the HalfCheetah-v4 environment, our comprehensive experimental results indicate that SAC outperforms both DDPG and PPO in terms of training stability, convergence speed, and overall accumulated reward.

4.2 Question 2:

How do the exploration strategies differ between PPO, DDPG, and SAC?
Compare the exploration mechanisms used by each algorithm, such as deterministic vs. stochastic policies, entropy regularization, and noise injection. How do these strategies impact learning in environments with continuous action spaces?

PPO, DDPG, and SAC each adopt distinct exploration mechanisms that critically influence their learning performance in continuous control environments. PPO, as an on-policy algorithm, employs a stochastic policy that is inherently probabilistic—actions are sampled from a learned probability distribution. This intrinsic stochasticity, combined with clipped objective updates, enables PPO to explore diverse regions of the state-action space while ensuring that policy updates remain within a trusted region. However, since PPO relies solely on freshly collected data, it cannot re-use older experiences, which can limit the long-term accumulation of exploratory information.

In contrast, DDPG uses a deterministic policy for action selection, which means that without any additional modification, it would consistently produce the same action for a given state. To induce exploration, DDPG injects external noise into the deterministic output. This noise is often implemented as either Gaussian noise or Ornstein-Uhlenbeck (OU) noise—the latter introducing temporal correlations that yield smoother exploratory trajectories in physical systems. The quality and magnitude of the injected noise significantly affect DDPG's ability to explore, and careful tuning is required to balance exploration and exploitation; too much noise can lead to erratic behavior, while too little can cause premature convergence to suboptimal policies.

SAC combines the benefits of stochastic policies with an explicit entropy regularization term in its objective function. Instead of relying solely on external noise injection, SAC naturally encourages exploration by maximizing a weighted entropy term, where the temperature parameter α governs the trade-off between reward maximization and entropy. By sampling actions from a Gaussian distribution through the reparameterization trick and dynamically adjusting α, SAC is able to maintain a robust level of randomness in its policy. This dynamic balancing allows SAC to explore effectively in high-dimensional continuous spaces while ensuring that the policy eventually becomes more exploitative as learning progresses.

In our experiments on the HalfCheetah-v4 environment, these differences in exploration strategies had a profound impact on performance. PPO's reliance on on-policy sampling yielded steady but relatively slower convergence due to the inherent sample inefficiency. DDPG, despite its deterministic framework, benefitted from careful noise injection; however, its sensitivity to noise magnitude sometimes led to less stable learning and slightly lower accumulated rewards. SAC, leveraging its entropy-maximizing stochastic policy and dynamic tuning of α, achieved superior performance, with training rewards around 13,000 and test rewards peaking near 15,814. This demonstrates that an exploration strategy which combines inherent stochasticity with adaptive entropy regulation, as in SAC, can be particularly effective in complex continuous control tasks.

4.3 Question 3:

What are the key advantages and disadvantages of each algorithm in terms of sample efficiency and stability?
Discuss how PPO, DDPG, and SAC handle sample efficiency and training stability. Which algorithm is more sample-efficient, and which one is more stable during training? What trade-offs exist between these properties?

In continuous control tasks such as HalfCheetah-v4, sample efficiency measures how effectively an algorithm converts interactions with the environment into learning progress, while stability refers to how smoothly and reliably training proceeds without large oscillations or divergence. PPO, DDPG, and SAC occupy different points along the trade-off between these two properties.

PPO is an on-policy method that collects fresh trajectories for each policy update and discards past experiences. Its clipped surrogate objective enforces small policy updates within a "trust region," yielding very stable, low-variance training. However, because PPO cannot reuse historical data, it is inherently sample inefficient: in our experiments it required far more environment steps to make incremental improvements and converged to substantially lower final rewards than either DDPG or SAC.

DDPG, by contrast, is off-policy and leverages a replay buffer to repeatedly sample past transitions, dramatically improving sample efficiency. With our modifications—larger three-layer networks, gradient clipping, and increased exploration noise—DDPG reached training rewards around 12,000 after fewer interactions than PPO. Yet its deterministic policy and reliance on external noise injection introduced higher variance in its learning curves. Even with careful noise tuning and network capacity increases, DDPG exhibited occasional performance spikes and drops, indicating lower stability compared to SAC.

SAC combines the off-policy replay mechanism of DDPG with twin Q-networks and entropy regularization, yielding both high sample efficiency and strong stability. The twin Q-networks mitigate overestimation bias, while the entropy term encourages persistent exploration and prevents premature convergence. Critically, our implementation dynamically tunes the temperature parameter α, allowing SAC to adaptively balance exploration and exploitation over the course of training. As a result, SAC produced smooth, monotonically improving reward curves, converged faster than both PPO and DDPG, and achieved the highest final performance (13,000 in training and 15,814 in a deterministic evaluation).

In summary, PPO sacrifices sample efficiency for update stability, making it reliable but slow to learn in high-dimensional continuous spaces. DDPG achieves strong sample efficiency through experience replay but suffers from unstable updates and sensitivity to noise hyperparameters. SAC strikes the best compromise: it maintains off-policy sample efficiency while enforcing stability via twin critics and entropy maximization, resulting in the superior overall performance observed in our HalfCheetah experiments.

4.4 Question 4:

Which reinforcement learning algorithm—PPO, DDPG, or SAC—is the easiest to tune, and what are the most critical hyperparameters for ensuring stable training for each agent?
How sensitive are PPO, DDPG, and SAC to hyperparameter choices, and which parameters have the most significant impact on stability? What common tuning strategies can help improve performance and prevent instability in each algorithm?

Among PPO, DDPG, and SAC, PPO is generally the easiest to tune in continuous control tasks like HalfCheetah-v4. Its clipped surrogate objective constrains policy updates and dramatically reduces sensitivity to learning rate and batch size. In our experiments, PPO achieved a stable—but slower—learning

curve with only minimal adjustments to its default hyperparameters (learning rate $\approx 3 \times 10^{-4}$, clip ratio ≈ 0.2, batch size ≈ 64, and GAE lambda ≈ 0.95). Changing network size or entropy coefficients had only a modest effect on final performance, making PPO a low-effort choice when rapid prototyping or baseline comparison is the goal.

DDPG proved the hardest to tune. As an off-policy, deterministic algorithm, its stability hinges on precise calibration of exploration noise, target network smoothing (τ), and learning rates for actor and critic. In our tuned implementation, we increased the noise standard deviation from 0.1 to 0.2 to ensure sufficient exploration, applied gradient clipping (max_norm $= 5.0$) to prevent exploding updates, and set $\tau = 0.005$ for slow, stable target updates. Critic and actor learning rates needed separate tuning (we found 3×10^{-4} each to work best). Small deviations—especially in noise magnitude or τ—led to large swings in performance or outright divergence, underscoring DDPG's fragility.

SAC strikes the best balance between sample efficiency and stability, making it moderately easy to tune. Its twin-Q architecture and entropy regularization reduce overestimation bias and automatically encourage exploration. The most critical hyperparameters in SAC are the initial temperature α (or enabling automatic entropy tuning), the target entropy, and the Polyak averaging factor τ. In our experiments, setting hidden_size $= 512$, enabling CUDA, and turning on automatic α tuning yielded smooth learning curves that rapidly converged to $\approx 13,000$ reward during training and peaked at $\approx 15,814$ in evaluation. SAC required fewer manual adjustments than DDPG—primarily learning rates (actor_lr $=$ critic_lr $\approx 3 \times 10^{-4}$) and target_entropy $\approx -$action_dim—to maintain stability.

Across all three algorithms, common tuning strategies include:

- Normalizing observations and rewards to stabilize gradient estimates.
- Gradient clipping (max_norm ≈ 5.0) to prevent large parameter updates.
- Larger network capacity (three hidden layers $\times 512$ units) to capture complex dynamics.
- Sufficient replay buffer size ($\geq 10^6$ transitions) for off-policy methods.
- Periodic evaluation (every 10 episodes) to monitor overfitting or divergence.
- Running multiple seeds and inspecting reward curves to ensure robust conclusions.

In summary, PPO offers the easiest out-of-the-box stability with minimal tuning, SAC provides the best trade-off between stability and sample efficiency with moderate tuning effort, and DDPG demands the most careful hyperparameter calibration to avoid instability despite its strong sample efficiency.