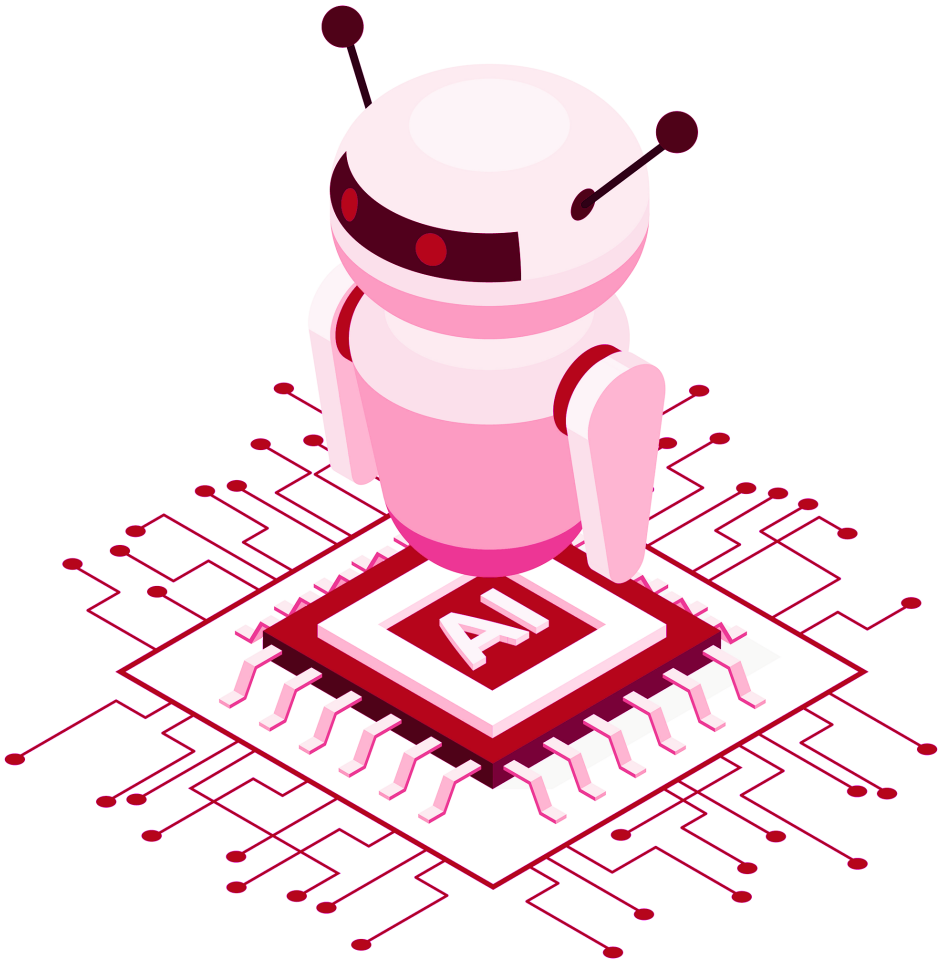

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 4:

Advanced Methods in RL

By:

Ali Najar

401102701

RIML

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 Question 1:

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

Actor Network:

The actor is the one making decisions. It takes in the current state and outputs a probability distribution $\pi_\theta(a|s)$ over possible actions. PPO updates this actor using a special trick (the clip objective) to prevent it from making wild updates that could break everything.

Critic Network:

The critic is like a judge. It evaluates how good a state (or action) is by estimating the value function $V_\phi(s)$ and instead of picking actions, it just tells the actor whether its choices were actually good. The value function $V_\phi(s)$ estimates the expected return from a given state. PPO uses this critic to calculate the advantage function, which helps the actor learn when to stick to its policy and when to change it.

Contribution to policy optimization:

The actor picks actions, then the environment gives feedback (reward). Then the critic estimates how valuable the state was compared to what it expected. PPO uses this to adjust the actor's policy gently so that it improves without making sharp updates. The use of both networks in the PPO objective function is vivid:

$$L(\theta) = \mathbb{E}\left[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio and A_t is the advantage estimate.

1.2 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 actor network in PPO doesn't just output a single action—it gives a probability distribution over all possible actions. This means the agent randomly picks actions based on these probabilities, instead of always choosing the one that looks best. Early in training, the policy is more random, helping the agent explore different strategies. Over time, as it learns, the randomness naturally decreases. To further enhance exploration, PPO includes an entropy bonus in the loss function to encourage randomness. As training progresses, entropy naturally decreases, making the policy more deterministic (agent sticks to what works). This helps prevent premature convergence to a suboptimal strategy.

$$L(\theta) = \mathbb{E}\left[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right] + \beta H(\pi_\theta)$$

where $H(\pi_\theta) = -\sum_a \pi_\theta(a|s)\log\pi_\theta(a|s)$ is the entropy term, and β controls the strength of the entropy regularization.

1.3 Question 3:

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

Episode Return:

It tells you how much reward the agent is getting per episode. During this experiment PPO reached an Episode return of 3000 after 500 episodes, which is very moderate.

Policy Loss:

It is good to check policy loss to see if it's decreasing. If it has many fluctuations or is not decreasing much it can be due to poor hyperparameter tuning such as learning rate or entropy coefficient. It can also be due to high clipping value which leads to sharp updates and unstable training.

$$L_{\text{policy}} = \mathbb{E}\left[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right]$$

where $r_t(\theta)$ is the probability ratio and A_t is the advantage function.

Value Loss:

Similar to policy loss this one should decrease too. If it's too high it means that the agent is struggling to estimate the true value of each action. This makes the training process unstable.

$$L_{\text{value}} = \mathbb{E}\left[(V_\phi(s_t) - R_t)^2\right]$$

where $V_\phi(s_t)$ is the critic's estimate and R_t is the actual return.

Advantage Estimates:

If it's small this means that the action is not that much better and if it's very high it can lead to unstable updates.

Entropy (Exploration Level):

If entropy drops too fast, the agent might be stuck in a bad policy.

$$H(\pi_\theta) = -\sum_a \pi_\theta(a|s) \log \pi_\theta(a|s)$$

KL Divergence:

The KL divergence measures how much the new policy is shifting away from the old one. If KL is too high, the policy is changing too fast (could cause instability). If it's too low, the updates might be too weak.

$$D_{\text{KL}}(\pi_{\theta_{\text{old}}} || \pi_\theta) = \sum_a \pi_{\theta_{\text{old}}}(a|s) \log \frac{\pi_{\theta_{\text{old}}}(a|s)}{\pi_\theta(a|s)}$$

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

2.1 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?

To make sure the agent explores different actions, we add noise to the action outputs. There are a few different types of noise used in DDPG.

Gaussian Noise:

Just adds random values from a Gaussian (normal) distribution to the action. It is smooth noise that slightly perturbs actions. It works fine but can be too random and might not explore efficiently in some environments.

$$a_t = \pi_\theta(s_t) + \mathcal{N}(0, \sigma^2)$$

Ornstein-Uhlenbeck (OU) Noise: A more correlated type of noise, which means the next noise value is based on the previous one. It is modeled using a differential equation:

$$dx_t = \theta(\mu - x_t)dt + \sigma dW_t$$

where θ controls the mean reversion speed, σ the noise magnitude, and dW_t is a Wiener process. For practical implementation in RL, we use a discrete version:

$$X_{t+1} = X_t + \theta(\mu - X_t)\Delta t + \sigma\sqrt{\Delta t}\mathcal{N}(0, 1)$$

2.2 Question 2:

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

PPO is an on-policy algorithm, meaning it only learns from the most recent batch of experiences collected using the current policy. It collects a batch of experiences, updates the policy using those experiences, and then throws them away.

DDPG is an off-policy algorithm, meaning it stores all past experiences in a replay buffer and can learn from old data. This means it can reuse past experiences multiple times, making learning more sample-efficient.

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

3.1 Question 1:

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

Overestimation bias:

In standard Q-learning, we estimate the Q-value as:

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$

The max operator can cause an overestimation bias because it picks the highest value, even if it's overestimated. This bias can cause the policy to favor bad actions, leading to instability.

In our QNetwork class, we define two separate Q-functions. When computing target Q-values, SAC takes the minimum of the two Q-values instead of the max. This prevents the policy from relying too much on an overestimated Q-value.

$$Q_{\text{target}}(s,a) = r + \gamma \min(Q_1(s',a'), Q_2(s',a'))$$

3.2 Question 2:

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

The temperature parameter α controls the balance between exploration and exploitation by adjusting how much entropy is added to the objective function.

$$J(\pi) = \mathbb{E}_{s_t \sim \mathcal{D}, a_t \sim \pi} \left[\alpha \log \pi(a_t|s_t) - Q(s_t,a_t) \right]$$

It can be seen that higher α leads to more exploration while lower α means more exploitation.

In short, α controls how random the agent's actions are. If it's too high, the agent explores too much and never settles on a good policy. If it's too low, the agent becomes greedy and might get stuck in a local optimum.

Instead of keeping α constant, SAC often learns α dynamically by optimizing the following loss function:

$$L(\alpha) = -\mathbb{E}_{a_t \sim \pi} \left[\alpha (\log \pi(a_t|s_t) + H_{\text{target}}) \right]$$

where H_{target} is the target entropy.

The benefits of using a dynamic α include:

- **Adaptive Exploration**: α starts high to encourage exploration and decreases over time to promote exploitation.
- **Stability**: Prevents the policy from being overly random or too deterministic early in training.
- **Automatic Tuning**: Removes the need for manual tuning of exploration noise, making SAC robust across different environments.

3.3 Question 3:

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

In Soft Actor-Critic (SAC), the agent operates in two distinct modes:

- **Training Mode**: The agent explores the environment, collects data, and updates its policy.
- **Evaluation Mode**: The agent executes actions based on its learned policy without exploration.

1. Training Mode

During training, the SAC agent needs to explore the environment effectively to learn an optimal policy. This requires the following:

- **Stochastic Action Selection**: Actions are sampled from the policy distribution:

$$a_t \sim \pi_\phi(a_t|s_t) = \mathcal{N}(\mu_\phi(s_t), \sigma_\phi(s_t))$$

  where π_ϕ is the learned stochastic policy.

- **Entropy Maximization**: The policy encourages high entropy to balance exploration and exploitation. The objective is:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}, a_t \sim \pi} \left[\alpha \log \pi_\phi(a_t|s_t) - Q_\theta(s_t, a_t) \right]$$

- **Exploration Noise**: Actions retain randomness due to the entropy term.
- **Parameter Updates**: The critic, actor, and temperature parameter α are updated iteratively.

2. Evaluation Mode

During evaluation, SAC selects the **mean action** instead of sampling from the Gaussian distribution:

$$a_t = \tanh(\mu_\phi(s_t)) \cdot \text{action scale} + \text{action bias}$$

This removes exploration noise and ensures consistent performance.

So the followings happen:

- **Deterministic Action Selection**: Instead of sampling, the action is chosen deterministically removing exploration noise.
- **Entropy Term Removal**: Since entropy is used for exploration, it is disabled during evaluation.
- **No Parameter Updates**: The policy is frozen, and no gradients are computed.

Overall Comparison

Aspect	Training Mode	Evaluation Mode
Action Selection	Stochastic ($a_t \sim \mathcal{N}(\mu, \sigma)$)	Deterministic ($a_t = \mu$)
Exploration	High (sampling from Gaussian)	Low (mean action only)
Entropy Term	Included ($\alpha \log \pi$)	Removed
Parameter Updates	Yes (gradient updates)	No (frozen policy)

Table 1: Comparison of Training and Evaluation Modes in SAC

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?

1. Training Stability

- **PPO**: Moderate stability. PPO uses on-policy learning, meaning it updates the policy using only the most recent batch of experiences. While stable, it suffers from sample inefficiency.
- **DDPG**: Low stability. DDPG is off-policy and highly sensitive to hyperparameters. It often suffers from high variance in training due to function approximation errors in the Q-function.
- **SAC**: High stability. SAC mitigates overestimation bias by using two Q-networks and includes entropy regularization, which helps maintain stability.

2. Convergence Speed

- **PPO**: Moderate convergence speed. Since PPO is on-policy, it requires frequent data collection, slowing down convergence.
- **DDPG**: DDPG requires careful tuning of noise and learning rates to avoid instability. But overall it can converge very fast in simple environments. Although, it is more instable than PPO or SAC.
- **SAC**: Fast convergence. SAC's off-policy nature allows efficient data reuse, leading to faster learning.

PPO updates the policy using a clipped objective:

$$L(\theta) = \mathbb{E}\left[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right]$$

This prevents drastic policy updates but makes learning slower.

DDPG relies on deterministic policy updates:

$$a_t = \mu_\theta(s_t) + \mathcal{N}(0, \sigma)$$

where noise $\mathcal{N}(0, \sigma)$ is required for exploration.

SAC improves upon DDPG by adding an entropy regularization term, ensuring smooth updates and faster convergence.

3. Overall Accumulated Reward

- **PPO**: Reaches moderate rewards but is limited by on-policy sample inefficiency.
- **DDPG**: Struggles with high variance and often does not achieve high rewards.
- **SAC**: Achieves the highest reward due to stable learning and efficient exploration.

Algorithm	Training Stability	Convergence Speed	Final Reward
PPO	Moderate	Slow	Medium
DDPG	Low	Fast but unstable	High but lower than SAC
SAC	High	Fast	High

Table 2: Comparison of PPO, DDPG, and SAC in HalfCheetah-v4

Conclusion

Based on our observations:

- **SAC performs best in the HalfCheetah environment**, achieving the highest rewards and fastest convergence.
- **PPO is a stable alternative but slower** due to on-policy updates.
- **DDPG reaches higher rewards than PPO but is unstable**, so it suffers from instability.

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?

1. PPO: Stochastic Policy with Implicit Exploration

PPO uses a **stochastic policy**, meaning it inherently explores by sampling actions from a learned probability distribution:

$$a_t \sim \pi_\theta(a_t|s_t)$$

The policy network outputs a Gaussian distribution:

$$\pi_\theta(a|s) = \mathcal{N}(\mu_\theta(s), \sigma_\theta(s))$$

Exploration Mechanism

- PPO **relies on stochasticity** in action sampling for exploration.
- There is no explicit noise injection, but an entropy term with coefficient of 0.02 is used to encourage more exploration.

Impact on Learning

- PPO's exploration depends on its learned variance $\sigma_\theta(s)$, which may shrink over time, reducing exploration.
- Works well in **high-dimensional** continuous action spaces but may require tuning of $\sigma_\theta(s)$.
- More stable than DDPG but less sample-efficient than SAC.

2. DDPG: Deterministic Policy with Noise Injection

DDPG uses a **deterministic policy**, meaning that given the same state, it always produces the same action:

$$a_t = \mu_\theta(s_t)$$

Exploration Mechanism

Since the policy is deterministic, DDPG must add noise externally to encourage exploration. Here we used gaussian noise to promote exploration. It just adds random values from a Gaussian (normal) distribution to the action. It is smooth noise that slightly perturbs actions.

$$a_t = \mu_\theta(s_t) + \mathcal{N}(0, \sigma^2)$$

Impact on Learning

- DDPG's exploration is **external** (via noise), making it highly sensitive to noise parameters.
- Works well in **low-dimensional action spaces** but struggles in high-dimensional environments.
- Exploration quality depends heavily on noise tuning, making training unstable.

3. SAC: Stochastic Policy with Entropy Regularization

SAC uses a **stochastic policy**, similar to PPO, and also uses **entropy regularization** to promote exploration:

$$a_t \sim \pi_\theta(a_t|s_t) = \mathcal{N}(\mu_\theta(s), \sigma_\theta(s))$$

Exploration Mechanism

SAC includes an **entropy bonus** to encourage randomness:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}, a_t \sim \pi} \left[\alpha \log \pi_\phi(a_t|s_t) - Q_\theta(s_t, a_t) \right]$$

where α controls the exploration-exploitation trade-off.

Impact on Learning

- SAC automatically adjusts exploration by tuning α dynamically.
- More stable than both PPO and DDPG.
- Works exceptionally well in **high-dimensional continuous spaces**.

Comparison of Exploration Strategies

Algorithm	Policy Type	Exploration Mechanism	Effectiveness
PPO	Stochastic	Implicit via learned variance $\sigma_\theta(s)$	Moderate
DDPG	Deterministic	Noise injection (OU noise)	Unstable, sensitive to noise
SAC	Stochastic	Entropy regularization $\alpha \log \pi$	High stability, best performance

Table 3: Comparison of Exploration Strategies in PPO, DDPG, and SAC

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?

1. PPO: On-Policy Learning with Stability but Low Sample Efficiency

Sample Efficiency

PPO is an on-policy algorithm. This limits sample efficiency because past experiences cannot be reused:

$$L(\theta) = \mathbb{E}\left[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right]$$

Overall:

- PPO throws away past experience requiring many interactions to learn.
- **Lower sample efficiency** compared to off-policy methods like DDPG and SAC.
- Works best when data collection is inexpensive (e.g., simulated environments).

Training Stability

PPO is designed for stability:

- Clipped objective prevents drastic policy updates, ensuring smooth training.
- More robust to hyperparameters than DDPG.
- Less prone to instability since it does not rely on a replay buffer.

Trade-off: PPO is **stable but inefficient**. It requires many samples, making it impractical for real-world problems with limited interactions.

2. DDPG: Off-Policy Learning with High Sample Efficiency but Instability

Sample Efficiency

DDPG is off-policy. This mean that:

- It is much more **sample-efficient** than PPO since past transitions are reused.
- It uses a replay buffer to improve data efficiency.
- It works well in settings where real-world interactions are expensive.

Training Stability

DDPG suffers from training instability due to:

- **Overestimation bias** from Q-function updates.
- **Exploration issues** (deterministic policy needs noise injection).
- **Hyperparameter sensitivity** (learning rate, noise scale, etc.).

Trade-off: DDPG is sample-efficient but unstable. It requires careful tuning and is prone to divergence.

3. SAC: Off-Policy Learning with High Sample Efficiency and Stability

Sample Efficiency

SAC, like DDPG, is off-policy, allowing it to reuse past experience efficiently. Overall, it has high **sample efficiency** similar to DDPG.

Training Stability

SAC is **the most stable algorithm** due to:

- **Twin Q-networks** reducing overestimation bias.

- Entropy regularization preventing premature convergence.
- Automatic temperature tuning ensuring a balance between exploration and exploitation.

Trade-off: SAC is both **sample-efficient and stable** but computationally more expensive due to two Q-networks and entropy tuning.

Comparison of Sample Efficiency and Stability

Algorithm	Sample Efficiency	Training Stability
PPO	Low (on-policy, no data reuse)	High (clipped updates, smooth learning)
DDPG	High (off-policy, replay buffer)	Low (prone to divergence)
SAC	High (off-policy, replay buffer)	High (entropy regularization, twin Q-networks)

Table 4: Comparison of PPO, DDPG, and SAC in Terms of Sample Efficiency and Stability

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?

1. PPO

Hyperparameter Sensitivity

PPO is generally easier to tune compared to the other two algorithms due to its built-in mechanisms for stability:

- **Clipped objective function** prevents policy updates from being too large.
- **On-policy nature** ensures more predictable updates but requires more samples.
- **Lower sensitivity** to learning rate and batch size compared to off-policy methods.

Critical Hyperparameters

The most important hyperparameters for stable PPO training are:

- **Clipping parameter** ϵ: Controls how much the policy can change per update.
- **Learning rate**: Too high leads to instability; too low slows learning.
- **Batch size**: Affects stability; larger batches smooth updates.
- **GAE (Generalized Advantage Estimation) parameter** λ: Balances bias and variance in advantage estimation.

Common Tuning Strategies

- Use a clipping parameter ϵ between 0.1 and 0.3 to prevent excessive updates.
- A small learning rate is a good starting point.
- Increase batch size if training is unstable.

- Reduce λ for less variance but more bias in advantage estimates.

2. DDPG

Hyperparameter Sensitivity

DDPG is highly sensitive to hyperparameters, making it the hardest algorithm to tune. Some reasons are:

- Overestimation bias in Q-learning can cause instability.
- Exploration depends on noise tuning, requiring manual adjustment.

Critical Hyperparameters

The most important hyperparameters for DDPG stability are:

- **Noise scale**: Controls exploration; too high leads to erratic behavior, too low prevents learning.
- **Replay buffer size**: Affects sample efficiency; too small leads to overfitting.
- **Learning rate**: Needs careful tuning to avoid divergence.
- **Batch size**: Affects stability; larger batches reduce variance.

Common Tuning Strategies

- Use Ornstein-Uhlenbeck (OU) noise for continuous control tasks.
- Set noise decay to reduce randomness over time.
- Use a small learning rate ($\approx 10^{-4}$) to prevent unstable updates.
- Increase replay buffer size ($\approx 10^6$ samples) to improve sample efficiency.

3. SAC

Hyperparameter Sensitivity

SAC is easier to tune than DDPG because it:

- Uses entropy regularization for exploration which is more stable than gaussian noise.
- Employs twin Q-networks to mitigate overestimation bias.
- Includes automatic entropy tuning, reducing the need for manual adjustments.

Critical Hyperparameters

The most important hyperparameters for SAC stability are:

- **Entropy coefficient** α: Determines exploration; can be fixed or learned.
- **Replay buffer size**: Large buffers improve sample efficiency.
- **Learning rate**: Lower values prevent divergence.
- **Batch size**: Influences learning smoothness.

Common Tuning Strategies

- Enable automatic entropy tuning to adjust exploration dynamically.
- Use large replay buffers ($\approx 10^6$ samples) to ensure diverse training data.

- Adjust α if training is too exploratory or too conservative.
- Use a small learning rate ($\approx 3 \times 10^{-4}$) for stable training.

Comparison of Hyperparameter Sensitivity and Tuning Difficulty

Algorithm	Tuning Difficulty	Most Sensitive Hyperparameters	Stability
PPO	Easy	ϵ, learning rate, batch size	High
DDPG	Hard	Noise scale, replay buffer, learning rate	Low
SAC	Moderate	α, replay buffer, learning rate	High

Table 5: Comparison of PPO, DDPG, and SAC in Terms of Hyperparameter Sensitivity and Stability

- PPO is the easiest to tune, with built-in stability mechanisms but lower sample efficiency.
- DDPG is the hardest to tune, as it is highly sensitive to noise and Q-function updates.
- SAC balances efficiency and stability, making it easier to tune than DDPG while being more stable.