Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Solution for Homework 4:

Advanced Methods in RL

Designed By:

Ahmad Karami

ahmad.karami77@yahoo.com

Hamidreza Ebrahimpour

ebrahimpour.7879@gmail.com

RIML

Spring 2025

Contents

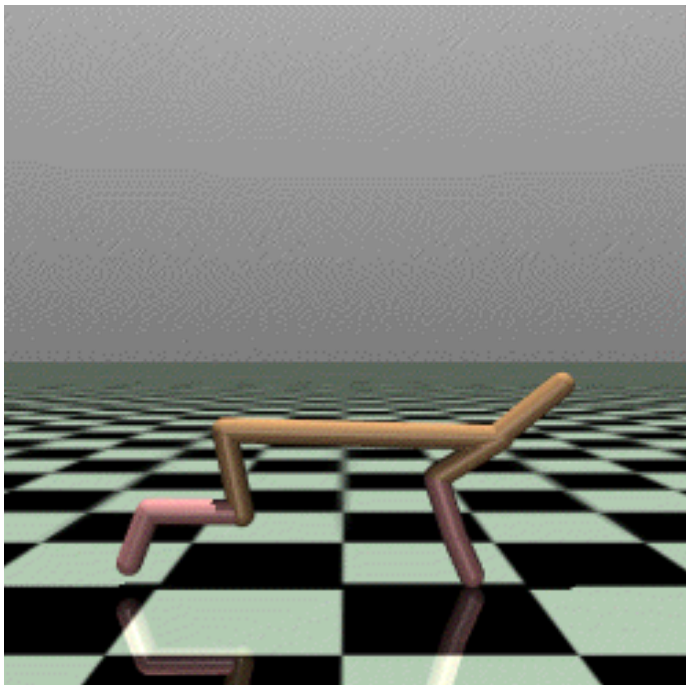
1 Part 2 (Problem Descriptions)

This assignment focuses on advanced policy-based reinforcement learning algorithms specifically designed for continuous action spaces. With step-by-step guidance, you will implement and analyze key algorithms, including Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), and Deep Deterministic Policy Gradient (DDPG). The tasks involve developing and fine-tuning these agents, understanding their optimization processes, and comparing their performance. Throughout the assignment, you are expected to complete the provided tasks, analyze the results, and answer conceptual questions related to the strengths and limitations of each method.

The `HalfCheetah` environment, part of the MuJoCo physics simulator, is a widely used benchmark for continuous control in reinforcement learning. It features a 2D robotic cheetah with **six controllable joints**, designed to move forward efficiently.

- **Action Space**: The environment has a **6-dimensional continuous action space**, where each dimension represents the torque applied to one of the six joints.
- **State Space**: The state is represented by a **17-dimensional vector**, including the position, velocity, and joint angles of the robot.
- **Reward Function**: The agent receives a reward based on its forward velocity while being penalized for excessive energy consumption, encouraging both speed and efficiency.
- **Challenges**: Learning a stable gait requires balancing exploration and exploitation, handling contact dynamics, and optimizing energy efficiency.

This environment is beneficial for evaluating reinforcement learning algorithms like PPO, SAC, and DDPG, as it requires precise control and policy optimization in a high-dimensional continuous action space.



You can also see more details about this environment in this link.

1.1 Task 1: Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a widely used reinforcement learning algorithm that belongs to the family of policy gradient methods. It improves policy learning stability by using a clipped surrogate objective function, which prevents excessively large policy updates that could destabilize training. Due to its simplicity, efficiency, and robustness, PPO is one of the most popular algorithms for solving reinforcement learning tasks in both continuous and discrete action spaces.

1.1.1 Task Overview

The provided notebook `PPO_continuous.ipynb` contains detailed explanations and guidance to help you complete this task. Additionally, you should give appropriate answers to a set of conceptual questions. However, you do not need to write your answers directly in the notebook due to space constraints. Instead, please submit them separately.

1.1.2 Questions

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

   Answer: The actor network is responsible for learning the policy by outputting the mean (μ) and standard deviation (σ) of a Gaussian distribution, from which actions are sampled. This enables stochastic action selection, which is crucial for exploration. The critic network estimates the state value function, providing a baseline for computing advantage estimates, which helps reduce variance in policy updates. PPO optimizes the policy by maximizing the expected advantage while constraining updates to ensure stability.

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?

   Answer: The actor network outputs a Gaussian distribution, ensuring stochastic action selection, which naturally promotes exploration. Additionally, including an entropy bonus in the loss function encourages the policy to maintain randomness during learning, especially in early stages, preventing premature convergence to suboptimal deterministic policies.

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

   Answer: Several key indicators should be monitored:

   - **Cumulative Reward per Episode:** Higher rewards indicate better policy learning.
   - **Policy Loss:** Measures how much the policy is changing; large fluctuations may indicate instability.
   - **Value Loss:** Evaluates how well the critic is approximating the value function; decreasing loss suggests better estimation.
   - **Entropy:** Higher entropy suggests more exploration, while lower entropy indicates more deterministic behavior; monitoring entropy ensures a balance between exploration and exploitation.

1.2 Task 2: Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is an off-policy reinforcement learning algorithm designed for continuous action-space environments. It combines the advantages of Deterministic Policy Gradient (DPG) and Deep Q-Networks (DQN) by using an actor-critic architecture, where the actor selects actions deterministically, and the critic estimates the action-value function. To encourage exploration, DDPG introduces noise into the actions during training. Additionally, it employs target networks and experience replay to improve learning stability. Due to its ability to handle high-dimensional continuous control tasks efficiently, DDPG is widely used in robotics and other real-world applications.

1.2.1 Task Overview

The notebook `SAC_DDPG_continuous.ipynb` offers detailed explanations and guidance to assist you in completing this task. You should also answer some conceptual questions based on your implementation and results.

1.2.2 Questions

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?

   Answer: In DDPG, exploration is essential for the agent to explore the environment and avoid getting stuck in suboptimal policies. To achieve exploration in continuous action spaces, noise is added to the action selected by the actor network. There are two commonly used types of noise in DDPG: **Ornstein-Uhlenbeck noise** and **Gaussian noise**.

   - **Ornstein-Uhlenbeck Noise: Behavior:** Ornstein-Uhlenbeck (OU) noise is a temporally correlated noise process, meaning that the noise depends on its previous value and tends to revert to a mean value over time. This type of noise is often used in physical systems and is designed to simulate more realistic perturbations in continuous control tasks (such as robotic movements). It is generated using the following equation:

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

     Impact on Learning: The temporally correlated nature of OU noise encourages the agent to explore in a smooth and continuous manner. This is particularly beneficial in environments where actions are more dependent on previous actions and the environment's state (e.g., robotic control). The noise does not introduce abrupt changes, thus allowing the agent to explore in a more structured way without straying too far from reasonable actions.

     Use Case: OU noise is ideal for tasks that involve smooth, continuous control, such as robotic arms, drones, or self-driving cars, where the actions tend to have some degree of momentum or continuity.

   - **Gaussian Noise: Behavior:** Gaussian noise is statistically independent and follows a normal distribution with a mean of zero and some variance. Unlike Ornstein-Uhlenbeck noise, Gaussian noise does not have temporal correlation; each noise sample is independent of the previous one and is sampled from a Gaussian distribution. It is often generated using the following equation:

     $$\epsilon_t \sim \mathcal{N}(0, \sigma)$$

     Impact on Learning: Gaussian noise encourages more random exploration because each action is perturbed independently of previous ones. While it can help in discovering diverse actions, it may

not always suit continuous control tasks as it can lead to more erratic behavior and instability in the agent's exploration process. Without the temporal correlation, the exploration might be too noisy and less structured, which could hurt convergence in certain types of environments.

Use Case: Gaussian noise is more appropriate for environments where exploration is less dependent on previous actions and where random exploration is more valuable, such as simple simulation environments or tasks where precise control is less critical.

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

   Answer: The key difference between PPO (Proximal Policy Optimization) and DDPG (Deep Deterministic Policy Gradient) regarding the use of past experiences lies in whether they are on-policy or off-policy algorithms.

   - PPO is an on-policy algorithm, meaning it updates the policy based on experiences generated by the current policy. Once the data is collected, it is used for a limited number of updates before new data is required. This means PPO relies only on recent experiences to update its policy.
   - DDPG, on the other hand, is an off-policy algorithm, meaning it can reuse past experiences stored in a replay buffer for training. This allows DDPG to sample random batches of experiences and update the policy more efficiently by learning from experiences generated by previous versions of the policy, not just the current one.

   In summary, PPO requires fresh, on-policy data for each update, while DDPG can reuse past experiences from a replay buffer to train its policy.

1.3 Task 3: Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) is an off-policy, model-free reinforcement learning algorithm that combines the benefits of both value-based and policy-based methods. SAC introduces the concept of entropy regularization, which encourages exploration by maximizing both expected reward and entropy (a measure of randomness in the policy). This allows SAC to balance exploration and exploitation more effectively than traditional methods. SAC is known for its sample efficiency, stability, and ability to handle complex tasks in environments with high-dimensional action spaces.

1.3.1 Task Overview

The notebook `SAC_DDPG_continuous.ipynb` offers detailed explanations and guidance to assist you in completing this task. You should also answer some conceptual questions based on your implementation and results.

1.3.2 Questions

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

   Answer: In SAC, two Q-networks (often called Q1 and Q2) are used to mitigate the problem of overestimation bias. By maintaining two independent Q-networks, the algorithm can take the minimum of the two Q-values to form the target during updates. This reduces the tendency of the critic to overestimate Q-values, leading to more stable and reliable value estimates, which improves the overall performance of the agent.

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

Answer: The temperature parameter in Soft Actor-Critic (SAC) controls the trade-off between exploration and exploitation by regulating the entropy term in the objective function. Specifically, α determines the weight of the entropy in the overall loss function, which influences how much randomness (or uncertainty) the policy should maintain in its action selection. When α is high, the agent is encouraged to explore more by taking actions with higher entropy (more randomness), which prevents the agent from prematurely exploiting a suboptimal policy. When α is low, the agent is more focused on exploiting what it has learned and making more deterministic, reward-maximizing decisions. Benefit of a dynamic α parameter: SAC dynamically adjusts α during training, which allows the algorithm to adaptively balance exploration and exploitation over time. Early in training, the agent may need more exploration (higher entropy) to discover a good policy, so α is higher. As training progresses, the agent can focus more on exploitation (lower entropy) and refine its policy, so α decreases. This dynamic adjustment helps SAC maintain efficient exploration in complex environments while ensuring that the agent eventually converges to an optimal policy. It also reduces the need for manually tuning the α parameter, making SAC more flexible and robust for various tasks.

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

   Answer: In SAC, evaluation mode and training mode are distinguished by how actions are selected:

   - In training mode, actions are sampled from the policy, which allows the agent to explore and improve its performance by interacting with the environment randomly (encouraging higher entropy).
   - In evaluation mode, the action is determined by taking the mean of the distribution output by the policy network. This ensures that the agent acts deterministically, using the learned policy to exploit what it has already learned without additional randomness, allowing for a fair assessment of the agent's performance.

1.4 Task 4: Comparison between SAC & DDPG & PPO

In this section, we will compare the key differences and similarities between three popular reinforcement learning algorithms: Soft Actor-Critic (SAC), Deep Deterministic Policy Gradient (DDPG), and Proximal Policy Optimization (PPO). We will focus on their approaches to policy learning, exploration strategies, and performance in continuous action spaces.

1.4.1 Questions

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?

   Answer: SAC is the best-performing algorithm in the HalfCheetah environment due to its superior training stability, faster convergence, and higher accumulated rewards. DDPG follows, offering better performance than PPO, but still falls short of SAC's efficiency and stability in continuous action tasks. PPO, while widely used in many environments, is less suited for tasks with continuous action spaces like HalfCheetah.

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?

   Answer: PPO uses a stochastic policy where actions are sampled from a Gaussian distribution, with

entropy regularization to encourage exploration. This makes it stable but less aggressive in exploring compared to SAC. DDPG, on the other hand, uses a deterministic policy and relies on noise injection (such as Ornstein-Uhlenbeck or Gaussian noise) to introduce exploration. While it is sample-efficient, it struggles in high-dimensional action spaces. SAC also uses a stochastic policy like PPO but explicitly maximizes entropy, encouraging diverse exploration and balancing reward-seeking with uncertainty. This makes SAC the most effective for exploration but also the most computationally expensive. Overall, SAC explores best, PPO is stable but explores less aggressively, and DDPG is efficient but weaker in high-dimensional settings.

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?

   Answer: DDPG is the most sample-efficient since it is an off-policy algorithm, meaning it can reuse past experiences for learning. However, it suffers from high instability, requiring careful tuning of exploration noise and hyperparameters. Small changes can lead to performance collapse.

   PPO is an on-policy method, making it less sample-efficient because it discards past experiences after each policy update. However, it is highly stable due to its clipped surrogate objective, preventing drastic policy updates that could destabilize learning. This makes PPO ideal for environments where robustness is more important than efficiency.

   SAC balances both aspects well. Being off-policy, it is more sample-efficient than PPO but slightly less efficient than DDPG. Its key advantage is its high stability, thanks to entropy maximization, which ensures continuous exploration and prevents premature convergence. However, this comes at the cost of higher computational complexity due to maintaining multiple critics and optimizing the entropy term.

   Trade-offs: DDPG: Highest sample efficiency but least stable. PPO: Most stable but least sample-efficient. SAC: A middle ground, offering a good balance of efficiency and stability, though computationally expensive.

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?**
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?

   Answer: Key Hyperparameters for Stability:

   - PPO: Clipping parameter, learning rate, batch size, number of epochs per batch.
   - DDPG: Learning rate, target update rate (τ), action noise (Ornstein-Uhlenbeck or Gaussian), replay buffer size.
   - SAC: Temperature parameter (α for entropy regularization), learning rate, target update rate, replay buffer size.

   Common Tuning Strategies: PPO benefits from moderate batch sizes and conservative policy updates. DDPG requires careful tuning of exploration noise and target network updates. SAC requires proper entropy tuning to balance exploration and exploitation.

2 References

[1] Cover image designed by freepik

[2] Half Cheetah environment

[3] Some other continuous action-space environments

[4] Continuous control with deep reinforcement learning

[5] Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

[6] Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

[7] Soft Actor-Critic Algorithms and Applications