CS449 HW4 REPORT

Yigit Turkmen / 22102518

1. Explanation of the SAC and HER Algorithms

Soft Actor-Critic (SAC):

SAC is an off-policy reinforcement learning algorithm that combines the benefits of entropy maximization and Q-learning. It aims to optimize a stochastic policy in an actor-critic framework while encouraging exploration by adding an entropy term to the reward objective. The algorithm consists of the following key components:

1. **Actor-Critic Architecture:** It uses two Q-functions (to address overestimation bias) and a stochastic policy network to generate actions.
2. **Soft Q-Function:** The critic learns to predict the expected return augmented with an entropy term.
3. **Policy Optimization:** The actor is trained to maximize the expected reward while ensuring high entropy in its actions.
4. **Entropy Regularization:** An automatic entropy coefficient adjustment mechanism is used to balance exploration and exploitation.

Hindsight Experience Replay (HER):

HER is a replay strategy designed to improve sample efficiency in sparse reward environments. It modifies the replay buffer by storing trajectories where the goals have been replaced with goals actually achieved during the episode. The algorithm can therefore learn from failed attempts by treating them as successes with respect to the adjusted goal. HER works particularly well in environments where achieving a sparse reward is rare without goal modification.

HER is typically integrated with an off-policy reinforcement learning algorithm like SAC, enabling the agent to learn robust policies even in challenging goal-based tasks.

---

2. Summary of Implementation

The code implements SAC combined with HER in the FetchReach environment. The following are the notable components:

1. **Environment Setup:**
    - The FetchReach environment from gymnasium_robotics is initialized.
    - Necessary libraries like PyTorch and MuJoCo are used for neural network implementation and physics simulation.
2. **Neural Network Architectures:**
    - Policies and Q-functions are implemented as deep neural networks with ReLU activations and fully connected layers.
    - The actor outputs a Gaussian policy for continuous actions.

3. **Replay Buffer:**
    - A custom replay buffer supports HER by storing transitions and modifying goals for effective learning.
4. **Training Loop:**
    - The agent interacts with the environment, storing transitions in the replay buffer.
    - The Q-functions and policy are optimized using gradients computed with PyTorch.
    - Evaluation is periodically performed to measure success rates across episodes.
5. **Multiple Seed Evaluation:**
    - The training process is run with multiple random seeds to assess the stability and reliability of the learned policy.

---

3. Analysis of Results

The results indicate that SAC with HER achieves a high success rate in the FetchReach task, even under sparse reward conditions. By using HER, the agent significantly improves sample efficiency, as it learns from unsuccessful episodes by redefining goals.

Key observations include:

1. **Stability:** The success rates demonstrate that the agent learns consistently across different runs.
2. **Policy Reliability:** SAC ensures that the learned policy is robust due to its entropy-maximizing objective, which reduces the likelihood of overfitting to specific states.

---

4. Discussion of Variations Observed Between Seeds and Potential Reasons

1. **Performance Variations:**
    - Success rates varied slightly across random seeds, reflecting differences in initial conditions and stochastic exploration.
    - Some seeds resulted in faster convergence, while others required more episodes to achieve similar success rates.
2. **Potential Reasons for Variations:**
    - **Exploration Patterns:** Different seeds result in unique exploration paths, influencing the diversity of transitions stored in the replay buffer.
    - **Gradient Noise:** Stochastic gradient updates can lead to different convergence behaviors.
    - **Goal Sampling in HER:** The randomness in selecting hindsight goals might also contribute to performance variability.

By averaging results over multiple seeds, these variations are accounted for, ensuring that the reported performance reflects the algorithm's general capabilities.

For reach.py I adjusted the positions as 0 0 0.

Related plot for the six different seeds and their performences: