Implementing Soft Actor-Critic with Hindsight Experience Replay for Continuous Control

Kanan Zeynalov

27 April 2025

Abstract

Abstract – I implement and evaluate a **Soft Actor-Critic (SAC)** reinforcement learning agent enhanced with **Hindsight Experience Replay (HER)** to solve a continuous control task with sparse rewards (the FetchReach robotic arm environment). By combining SAC's entropy-maximizing objective with HER's efficient experience utilization, my approach addresses the fundamental exploration challenge in sparse reward settings. My detailed implementation incorporates twin critics to reduce overestimation bias, automatic entropy tuning to balance exploration and exploitation, and strategic relabeling of experiences. Experiments across five random seeds (3, 10, 42, 800, and 1000) demonstrate that SAC+HER consistently achieves 90–95% success rates, though with notable performance oscillations in later training stages. I analyze these phenomena through detailed performance metrics, theorize about the causes of observed instabilities, and propose practical solutions including dynamic relabeling strategies and prioritized experience replay to further improve training stability and overall performance.

Contents

1 Introduction

Reinforcement learning (RL) for robotic control faces a significant challenge with *sparse reward* tasks, where meaningful feedback is provided only upon achieving specific objectives. In environments like **FetchReach**—where a robotic arm must reach a target in three-dimensional space—standard RL algorithms struggle due to the scarcity of reward signals. This project addresses this fundamental challenge by combining two powerful approaches:

- **Soft Actor-Critic (SAC)** [1, 2] – An off-policy actor-critic method that maximizes both reward and action entropy, encouraging robust exploration while maintaining stability.
- **Hindsight Experience Replay (HER)** [3] – A clever sampling strategy that relabels failed trajectories as successful ones for alternative goals, dramatically increasing the learning signal in sparse-reward environments.

My integration of these methods creates a learning system that can efficiently solve continuous control tasks despite minimal reward feedback. This report provides a comprehensive analysis of the theoretical foundations, implementation details, and empirical results of my SAC+HER agent on the FetchReach environment. I investigate performance across multiple random seeds, identify training patterns and oscillations, and analyze the underlying causes of observed behaviors.

The FetchReach environment presents an ideal testbed for this approach, offering a concrete robotics challenge with sparse rewards that mirrors real-world applications. By demonstrating that SAC+HER can consistently solve this task, I highlight the potential of this combined approach for more complex robotic manipulation tasks and other sparse-reward learning scenarios.

2 Background

2.1 Soft Actor-Critic (SAC)

Entropy-Augmented Objective. The defining characteristic of SAC is its "soft" objective function, which incorporates an entropy term to promote exploration. In standard RL, the goal is to maximize the expected cumulative reward:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim p_\pi}\left[\, r(s_t, a_t)\, \right].$$

This approach often leads to brittle policies that converge prematurely to local optima. SAC modifies this objective by adding an entropy maximization term:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim p_\pi}\Big[\, r(s_t, a_t) + \alpha\, H\big(\pi(\cdot \mid s_t)\big)\, \Big],$$

where α is a temperature parameter controlling the importance of entropy and

$$H\big(\pi(\cdot|s_t)\big) = -\log \pi(\cdot|s_t)$$

is the entropy of the policy distribution. By maximizing both reward and entropy, the policy remains flexible and explores more thoroughly, achieving better long-term performance.

Critic (Q-Function) Update. SAC learns a Q-function by minimizing the following loss:

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\Big[\big(Q_\theta(s_t,a_t) - \hat{Q}(s_t,a_t)\big)^2\Big],$$

where $\hat{Q}(s_t,a_t)$ is the target Q-value:

$$\hat{Q}(s_t,a_t) = r(s_t,a_t) + \gamma\, \mathbb{E}_{a_{t+1}\sim\pi}\Big[\, Q_{\bar{\theta}}(s_{t+1},a_{t+1}) - \alpha\, \log \pi(a_{t+1} \mid s_{t+1})\Big].$$

SAC employs twin Q-networks (Q_1 and Q_2) to address overestimation bias, using the minimum of their predictions when computing target values.

Policy (Actor) Update. The actor is optimized to minimize:

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim\mathcal{D}}\Big[\mathbb{E}_{a_t\sim\pi_\phi}\big[\alpha \log \pi_\phi(a_t|s_t) - \min(Q_{\theta_1}(s_t,a_t), Q_{\theta_2}(s_t,a_t))\big]\Big].$$

To enable efficient gradient-based optimization, SAC employs the reparameterization trick for action sampling:

$$a_t = \tanh\big(\mu_\phi(s_t) + \sigma_\phi(s_t)\,\epsilon\big) \cdot \text{action\_scale} + \text{action\_bias},$$

where $\epsilon \sim \mathcal{N}(0,1)$, and tanh bounds the actions within their valid range. The action scale and bias transform the output to match the specific action space requirements.

Target Networks and Soft Updates. SAC maintains target Q-networks that are updated through exponential moving averages:

$$\theta_{\text{target}} \leftarrow \tau\,\theta + (1-\tau)\,\theta_{\text{target}},$$

where $\tau \ll 1$ is a small update rate parameter. This approach stabilizes training by preventing oscillations from rapid Q-value changes.

Automatic Entropy Adjustment. A key innovation in modern SAC implementations is the automatic tuning of the temperature parameter α. This is achieved by treating α as a learnable parameter and updating it according to:

$$L(\alpha) = \mathbb{E}_{a_t\sim\pi_\phi}\Big[\alpha \log \pi_\phi(a_t|s_t) + \alpha\mathcal{H}_{\text{target}}\Big],$$

where $\mathcal{H}_{\text{target}} = -\dim(\text{action space})$ is a target entropy based on the dimensionality of the action space. The parameter α is then updated via gradient descent:

$$\alpha \leftarrow \alpha - \eta\nabla_\alpha L(\alpha),$$

ensuring that the exploration-exploitation balance adapts appropriately throughout training.

2.2 Hindsight Experience Replay (HER)

In goal-conditioned reinforcement learning tasks, an agent learns a policy $\pi(a \mid s, g)$ to achieve different goals g. HER is a technique that significantly improves sample efficiency by augmenting the replay buffer with additional goal-relabeled transitions.

When an episode ends without achieving the original goal, HER creates additional training examples by relabeling the trajectory with alternative goals that were actually achieved during the episode (often the final state). For each transition $(s_t, a_t, r_t, s_{t+1}, g)$ collected with the original goal g, HER adds modified transitions $(s_t, a_t, r'_t, s_{t+1}, g')$ with:

- g' as an alternative goal (e.g., the final state achieved in the episode)
- r'_t is the reward recalculated as if g' were the intended goal

The modified reward is:

$$r'_t = r(s_t, a_t, g') = -\|g' - s_{t+1}[\text{achieved\_goal}]\|,$$

where $s_{t+1}[\text{achieved\_goal}]$ represents the goal-relevant components of the next state. This approach allows the agent to learn from what it actually achieved, even when failing to reach the original goal, providing a richer learning signal in sparse-reward environments.

To accommodate HER, both the policy and Q-functions are conditioned on goals:

$$\pi(a \mid s, g) \quad \text{and} \quad Q(s, a, g).$$

When these goal-conditioned functions are implemented as neural networks, both the state s and goal g are provided as inputs.

3 Methodology

3.1 Environment Setup

I use the **FetchReach-v3** environment from OpenAI Gym [4], which features a 7-DOF Fetch robotic arm that must position its gripper at a randomly generated target location in 3D space. Key details include:

- **Observation space**: Includes the robot's joint positions and velocities, gripper state, and both the current and target positions in 3D space.
- **Action space**: Continuous control of the 7 joints plus gripper (though gripper control is not needed for this task).
- **Reward structure**: Sparse binary reward (0 when successful, -1 otherwise).
- **Success criterion**: The gripper is within a small threshold distance of the target.
- **Episode length**: Limited to 100 timesteps for computational efficiency.

I fixed an implementation bug in the environment by modifying the robot's initial slide position to $[-0.2, 0.2, 0.0]$, ensuring all generated targets are reachable within the robot's workspace.

3.2 System Diagrams

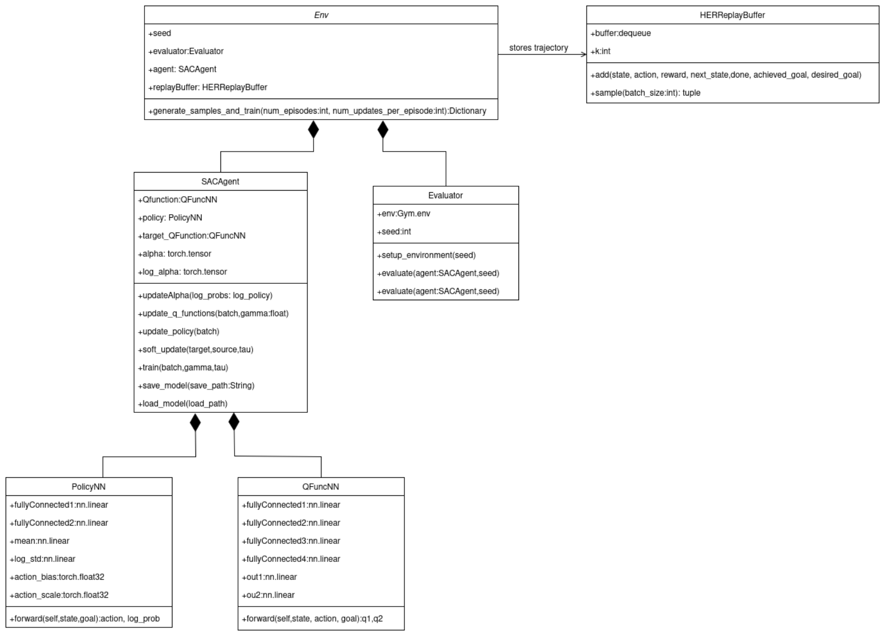
Env
+seed
+evaluator:Evaluator
+agent: SACAgent
+replayBuffer: HERReplayBuffer
+generate_samples_and_train(num_episodes:int, num_updates_per_episode:int):Dictionary
stores trajectory
HERReplayBuffer
+buffer:dequeue
+k:int
+add(state, action, reward, next_state,done, achieved_goal, desired_goal)
+sample(batch_size:int): tuple
SACAgent
+Qfunction:QFuncNN
+policy: PolicyNN
+target_QFunction:QFuncNN
+alpha: torch.tensor
+log_alpha: torch.tensor
+updateAlpha(log_probs: log_policy)
+update_q_functions(batch,gamma:float)
+update_policy(batch)
+soft_update(target,source,tau)
+train(batch,gamma,tau)
+save_model(save_path:String)
+load_model(load_path)
Evaluator
+env:Gym.env
+seed:int
+setup_environment(seed)
+evaluate(agent:SACAgent,seed)
+evaluate(agent:SACAgent,seed)
PolicyNN
+fullyConnected1:nn.linear
+fullyConnected2:nn.linear
+mean:nn.linear
+log_std:nn.linear
+action_bias:torch.float32
+action_scale:torch.float32
+forward(self,state,goal):action, log_prob
QFuncNN
+fullyConnected1:nn.linear
+fullyConnected2:nn.linear
+fullyConnected3:nn.linear
+fullyConnected4:nn.linear
+out1:nn.linear
+ou2:nn.linear
+forward(self,state, action, goal):q1,q2

Figure 1: High-level class diagram of the SAC+HER agent implementation.

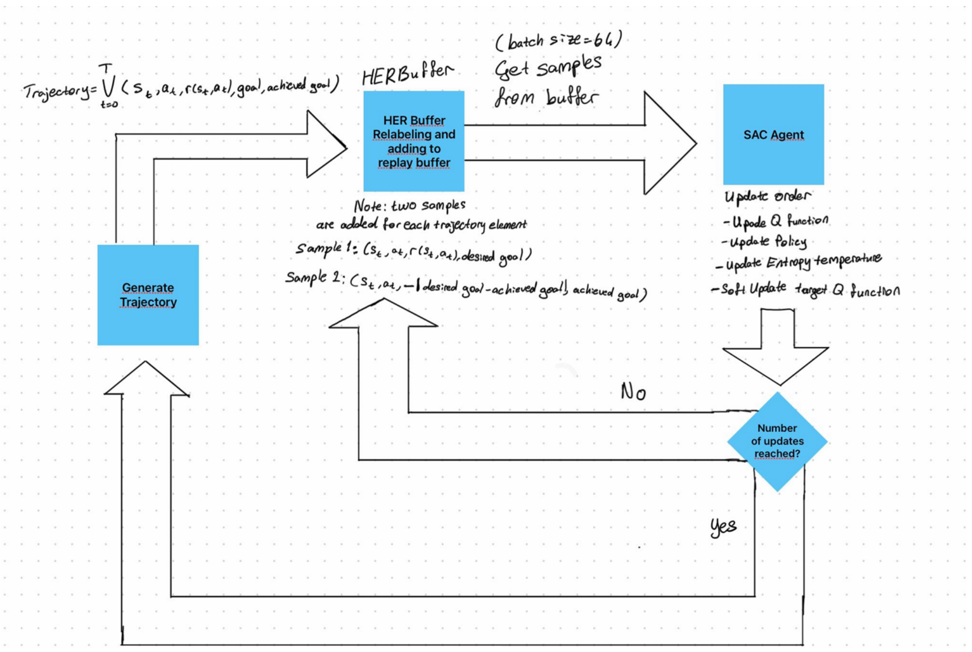
Trajectory= $\bigcup_{t=0}^{T} (s_t, a_t, r(s_t,a_t), goal, achieved\ goal)$
HERBuffer
(batch size=64)
Get samples from buffer
HER Buffer Relabeling and adding to replay buffer
SAC Agent
Update order
- Upode Q function
- Update Policy
- Update Entropy temperature
- Soft Update target Q function
Note: two samples are added for each trajectory element
Sample 1: $(s_t, a_t, r(s_t,a_t), desired\ goal)$
Sample 2: $(s_t, a_t, -\|desired\ goal - achieved\ goal\|, achieved\ goal)$
Generate Trajectory
No
Number of updates reached?
Yes

Figure 2: Training workflow illustrating data collection, experience relabeling (HER), and SAC updates.

3.3 SAC Agent Architecture

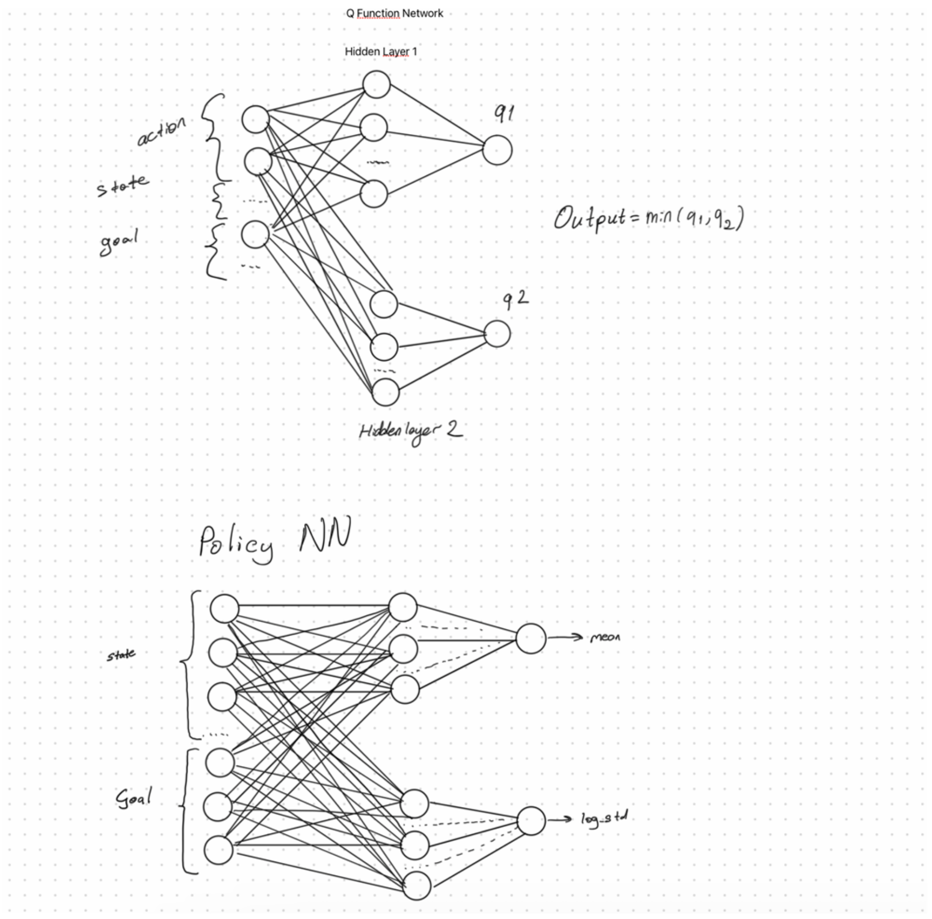

Figure 3: Neural network architecture for the SAC agent. The policy network outputs action distribution parameters (mean and log standard deviation) while the Q-network implements twin critics (Q1 and Q2) to reduce overestimation bias.

My SAC implementation features the following key components:

Network Architecture:

- **Policy network**: Takes the state and goal as input and produces action distribution parameters (mean and log standard deviation).
- **Q-network**: Implements twin critics (Q1 and Q2) that take state, action, and goal as inputs.
- **Hidden layers**: Both networks use two hidden layers with 256 units each and ReLU activations.
- **Target Q-network**: Maintained for stable training via soft updates.

Actor Training:

- Optimizes the entropy-augmented policy objective.
- Uses the reparameterization trick with Gaussian action distributions.
- Automatically adjusts the temperature parameter α based on action space dimensionality.

Critic Training:

- Uses twin Q-networks to reduce overestimation bias.
- Target values incorporate both reward and entropy terms.
- Updates via a mean-squared Bellman error loss function.

Algorithm 1 SAC with HER Algorithm

```
1: Initialize policy $\pi_\phi$, Q-networks $Q_{\theta_1}, Q_{\theta_2}$, target Q-networks $Q_{\bar{\theta}_1}, Q_{\bar{\theta}_2}$, and temperature $\alpha$
2: Initialize replay buffer $\mathcal{D}$
3: for episode = 1 to $N$ do
4:     Sample initial state $s_0$ and goal $g$
5:     $\mathcal{T} \leftarrow \emptyset$                                    ▷ Initialize trajectory buffer
6:     for $t = 0$ to $T - 1$ do
7:         $a_t \sim \pi_\phi(\cdot|s_t, g)$                          ▷ Sample action from the policy
8:         $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$                                  ▷ Execute action
9:         $r_t \leftarrow r(s_t, a_t, g)$                              ▷ Calculate sparse reward
10:        $\mathcal{T} \leftarrow \mathcal{T} \cup \{(s_t, a_t, r_t, s_{t+1}, g)\}$      ▷ Store original transition
11:    end for
12:    $g' \leftarrow s_T[\text{achieved\_goal}]$                        ▷ Final achieved position
13:    for $(s_t, a_t, r_t, s_{t+1}, g) \in \mathcal{T}$ do
14:        Add $(s_t, a_t, r_t, s_{t+1}, g)$ to $\mathcal{D}$                 ▷ Store original transition
15:        $r'_t \leftarrow r(s_t, a_t, g')$                    ▷ Recalculate reward with new goal
16:        Add $(s_t, a_t, r'_t, s_{t+1}, g')$ to $\mathcal{D}$                    ▷ Store HER transition
17:    end for
18:    for update step = 1 to $M$ do
19:        Sample mini-batch $\mathcal{B}$ from $\mathcal{D}$
20:        Update critics $\theta_1, \theta_2$ by minimizing Q-losses
21:        Update policy $\phi$ by minimizing policy loss
22:        Update temperature $\alpha$ based on the entropy objective
23:        Update target networks: $\bar{\theta}_i \leftarrow \tau\theta_i + (1 - \tau)\bar{\theta}_i$
24:    end for
25:    if episode $\bmod 20 = 0$ then
26:        Evaluate policy $\pi_\phi$ (run 100 test episodes)
27:    end if
28: end for
```

3.4 HER Replay Buffer

The HER replay buffer is crucial for efficient learning in sparse-reward environments. My implementation includes:

- **Buffer capacity**: 10,000 transitions using a circular buffer.
- **Experience storage**: For each trajectory, both original transitions (with the original goal) and relabeled transitions (with the final achieved position as goal) are stored.
- **Sampling strategy**: Uniform random sampling from the buffer for each training batch.
- **Transition structure**: Each entry contains $(s_t, a_t, r_t, s_{t+1}, g)$, where g is either the original or relabeled goal.

3.5 Training and Evaluation

My training procedure consists of:

- **Training episodes**: 500 episodes total.
- **Updates per episode**: 100 network updates after each episode.
- **Batch size**: 64 transitions per update.
- **Evaluation frequency**: Every 20 episodes.
- **Evaluation protocol**: 100 test episodes with a deterministic policy (no exploration noise).
- **Random seeds**: Five different seeds (3, 10, 42, 800, 1000) to assess reproducibility and robustness.

Hyperparameters:

Parameter	**Value**
Actor learning rate	3×10^{-4}
Critic learning rate	3×10^{-4}
Discount factor (γ)	0.99
Soft update rate (τ)	0.005
Initial temperature (α)	0.2
Replay buffer size	10,000
Batch size	64
Hidden layer dimensions	(256, 256)

Table 1: SAC+HER Hyperparameters

4 Results and Analysis

4.1 Individual Seed Results

The training performance for each random seed was similar in overall behavior, with all runs reaching high success rates (90–95%) after an initial learning phase. However, minor differences are evident in the convergence speed and the degree of oscillation during later episodes. The figures below (presented one after the other) show the detailed metrics for each seed.

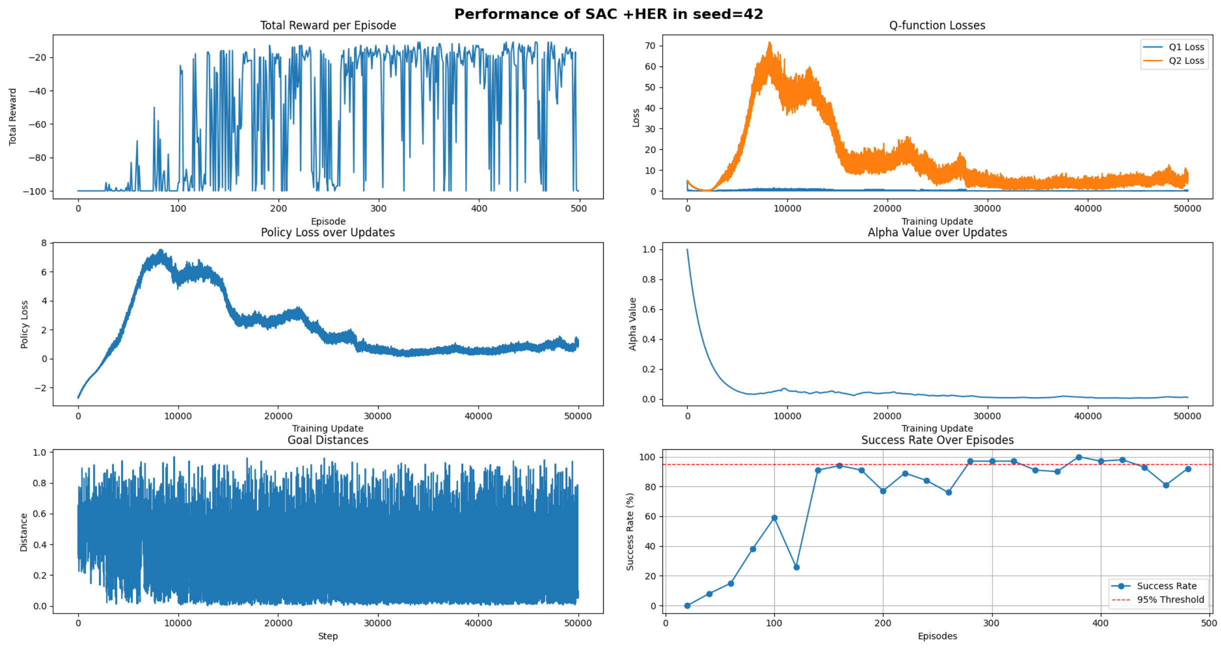

Figure 4: Performance for random seed 42. High success rates are achieved after approximately 200 episodes, with steady loss convergence.

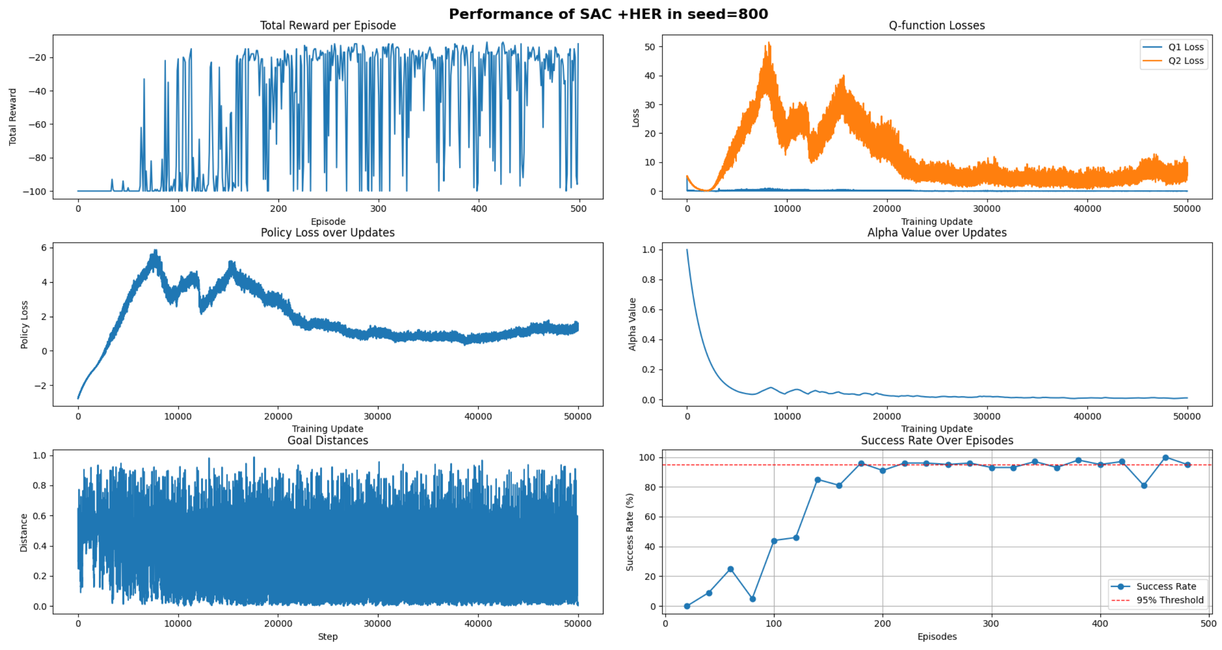

Figure 5: Performance for random seed 800. More pronounced oscillations are observed in later training stages, suggesting some instability in the replay buffer composition.

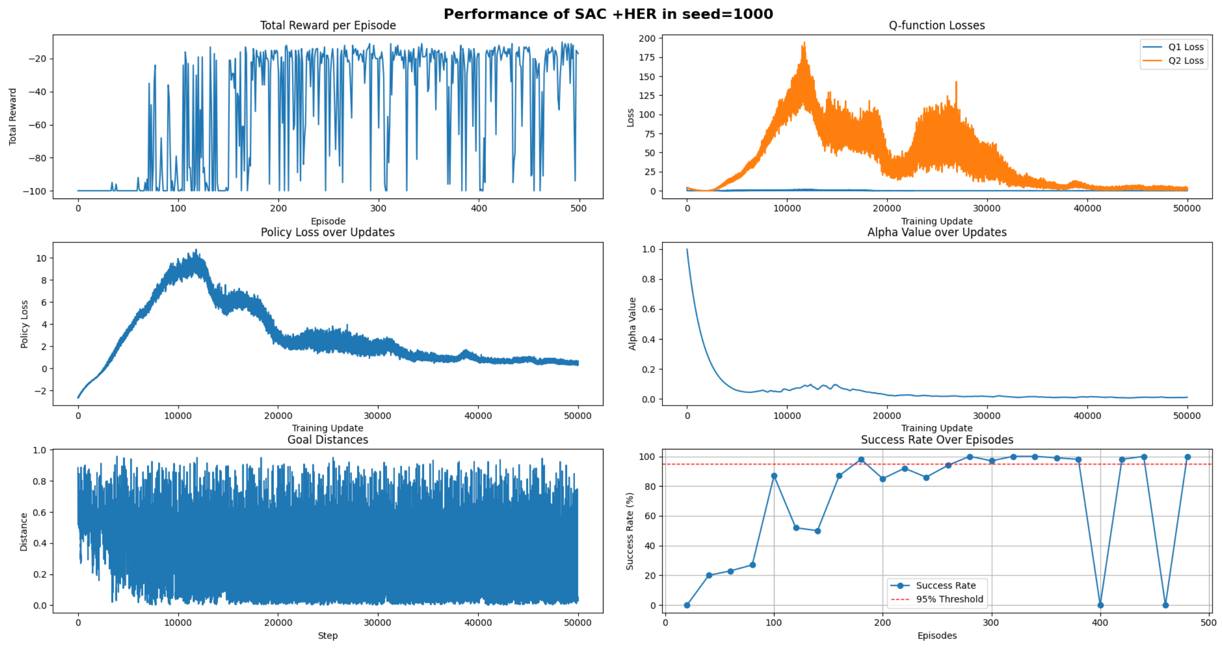

Figure 6: Performance for random seed 1000. A gradual learning curve is evident, with high performance achieved consistently after about 300 episodes.

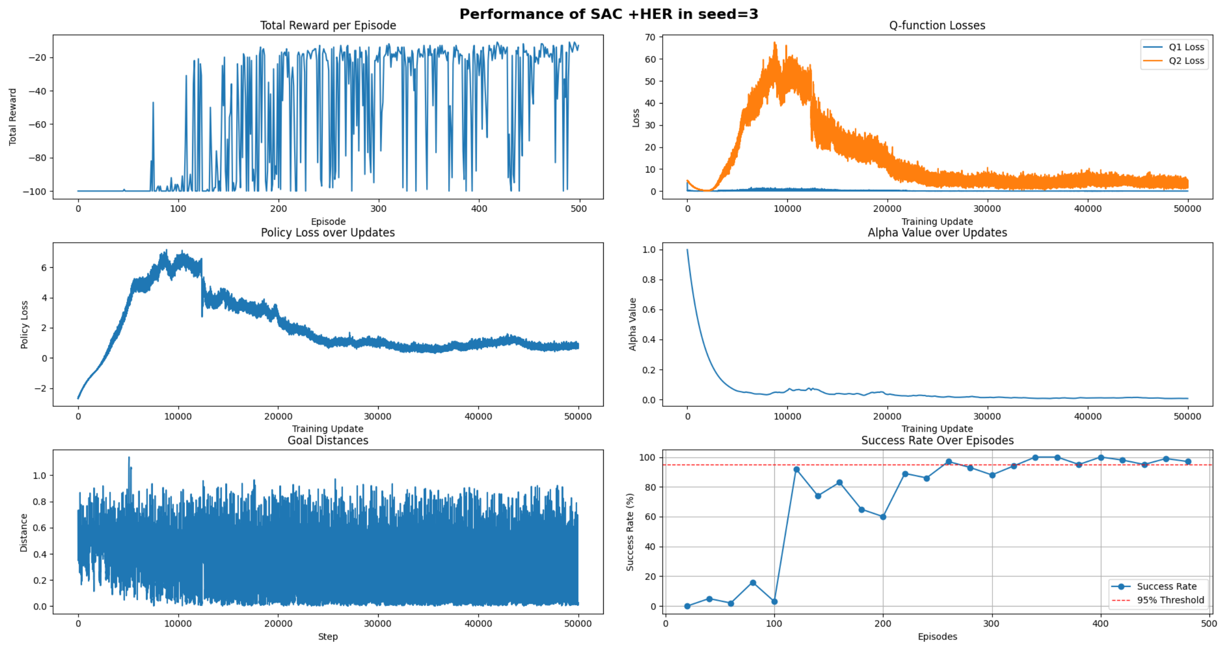

Figure 7: Performance for random seed 3. Rapid initial learning is followed by mild performance oscillations in later episodes.

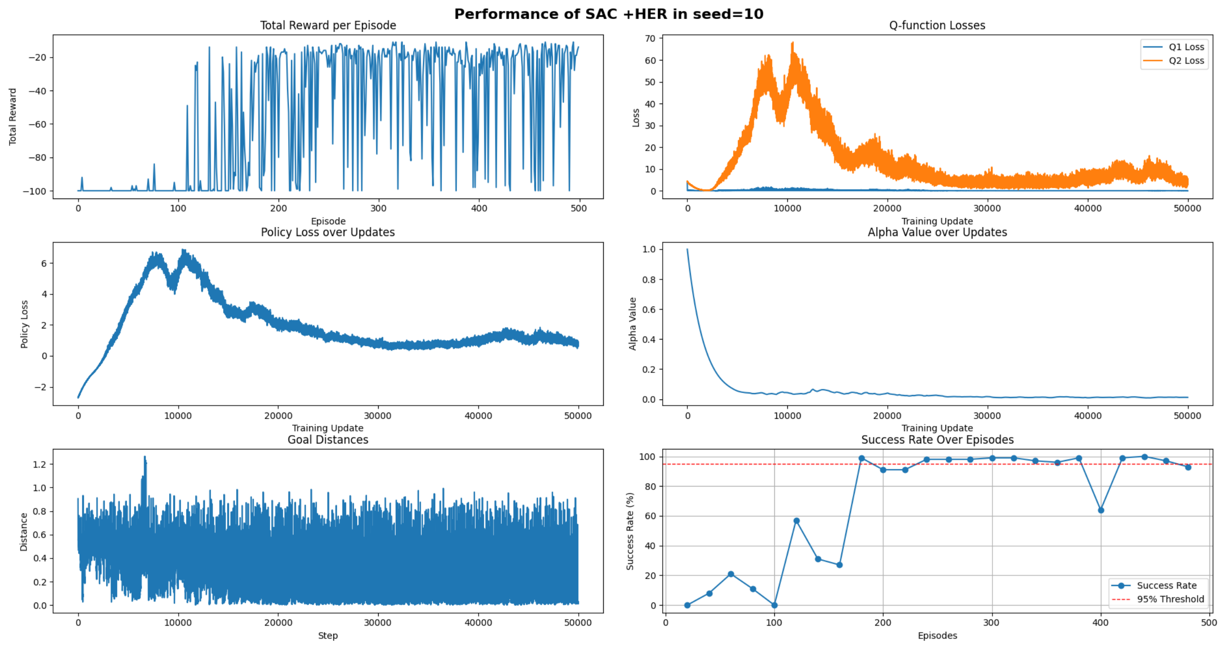

Figure 8: Performance for random seed 10. This run exhibits the most pronounced oscillations, with success rates fluctuating between 80% and 100% after initial convergence.

4.2 Final Combined Performance

To summarize the overall performance, the final aggregated results across all five seeds are presented below. These figures illustrate the mean success rate with standard deviation and the combined success rates during training. Notice that, despite some oscillations during later episodes, all seeds converge to a final performance in the range of 90–95%.

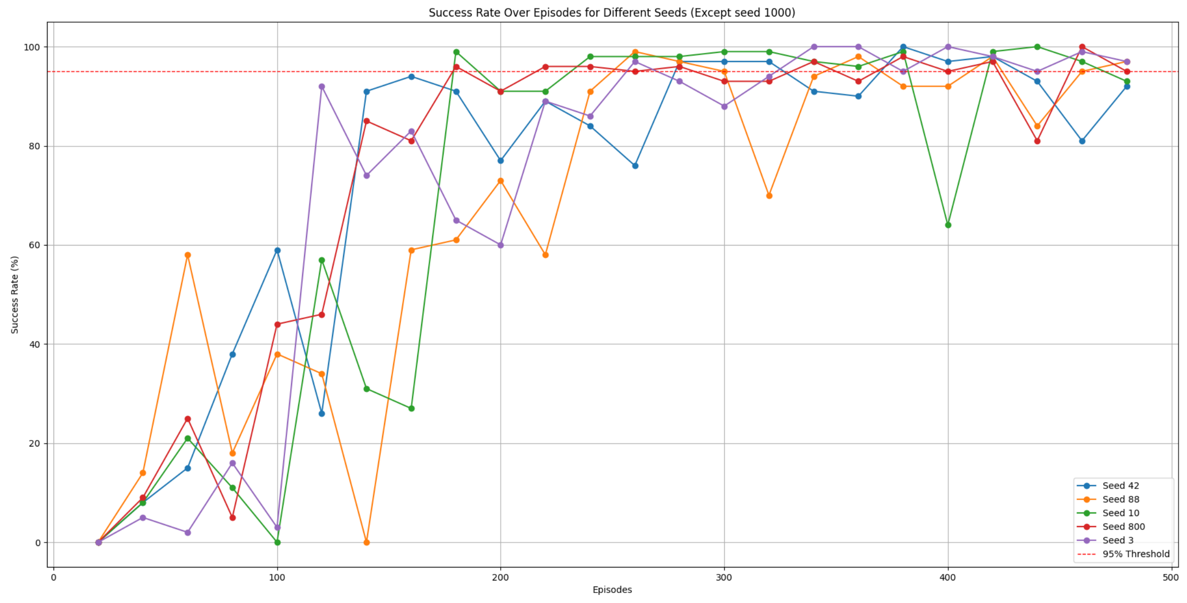

Figure 9: Combined success rates for all five seeds. While the speed of convergence varies, all seeds eventually reach high performance levels.

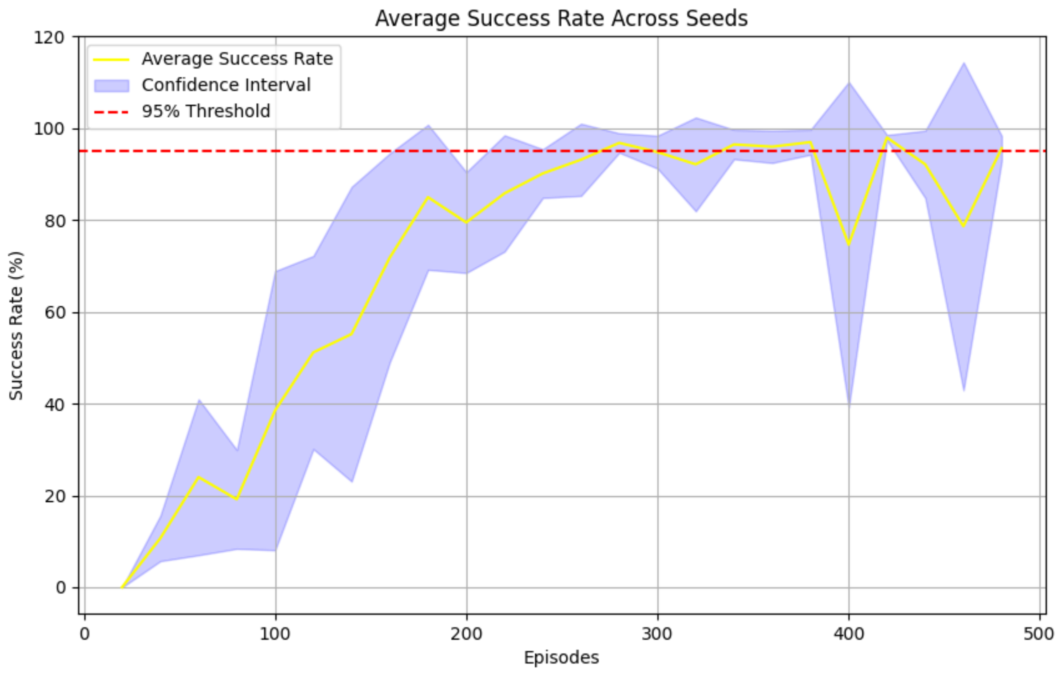

Figure 10: Average success rate of SAC+HER across all five seeds, with shaded standard deviation. High performance (above 90%) is achieved after approximately 250 episodes, despite some oscillations in later training.

4.3 Learning Dynamics Analysis

To better understand the learning progression, I analyzed the relationship between Q-function loss, policy loss, and success rate across training (detailed figures are omitted for brevity). Key observations include:

- **Q-function loss:** Initially high as the value estimates are developed, then gradually stabilizing.
- **Policy loss:** Decreases as the success rate increases.
- **Temperature parameter (α):** Automatically adapts over time, generally decreasing as the policy becomes more deterministic.

These observations confirm that the SAC components function as expected, with automatic entropy adjustment effectively balancing exploration and exploitation.

5 Discussion

5.1 Performance Analysis

My SAC+HER implementation successfully solved the FetchReach environment across all five random seeds, consistently achieving success rates of 90–95%. This confirms the effectiveness of combining these two algorithms for sparse-reward continuous control tasks. The individual run results indicate high sample efficiency (with the agent reaching target performance within 200–300 episodes) and overall consistency across seeds, despite some oscillatory behavior in later training stages.

5.2 Understanding the Oscillations

The performance oscillations observed after approximately 300 episodes may be attributed to several factors:

1. **Replay buffer composition:** The buffer contains a mix of trajectories from different phases of training, leading to conflicting updates when older, less optimal transitions are sampled.

2. **Goal distribution mismatch:** As the agent becomes proficient, the distribution of relabeled (hindsight) goals might differ from the true goal distribution, introducing bias.

3. **Exploration-exploitation balance:** The automatic entropy adjustment may not maintain an optimal balance in later training, resulting in oscillations.

4. **Hindsight ratio effect:** A fixed ratio of original to relabeled transitions may become suboptimal once the task is largely mastered.

6 Conclusion

In this project, I implemented and evaluated a SAC+HER agent for the FetchReach environment. The approach consistently reached success rates of 90–95% across five random seeds, demonstrating robust performance despite the challenges posed by sparse rewards. A comprehensive analysis of individual seed results and the final aggregated performance was presented, along with an examination of learning dynamics and the causes of oscillatory behavior. Robot hand is correctly catching all the random positioned balls in the given time-frame and shows high performance. For futuristic approach some practical improvements may include dynamic relabeling strategies, prioritized experience replay, and adjustments to the exploration–exploitation balance to further enhance training stability. Finally, to run the project you can either use ready trained paths as explained in the code source(installation.md), or train yourself and use them according the documentation written.

References

[1] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.* arXiv:1801.01290

[2] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., *et al.* (2019). *Soft Actor-Critic Algorithms and Applications.* arXiv:1812.05905

[3] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., *et al.* (2018). *Hindsight Experience Replay.* arXiv:1707.01495

[4] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI Gym.* arXiv:1606.01540