Soft Actor-Critic Algorithms and Applications

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P. and Levine, S.

CS8803
Student Presenter: Jeonghwan Kim

Soft Actor-Critic Algorithms and Applications

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., **Ha, S., Tan, J.**, Kumar, V., Zhu, H., Gupta, A., Abbeel, P. and Levine, S.

CS8803
Student Presenter: Jeonghwan Kim

Soft Actor-Critic Algorithms and Applications

Tuomas Haarnoja[*†‡] **Aurick Zhou**[*†] **Kristian Hartikainen**[*†] **George Tucker**[‡]

Sehoon Ha[‡] **Jie Tan**[‡] **Vikash Kumar**[‡] **Henry Zhu**[†] **Abhishek Gupta**[†]

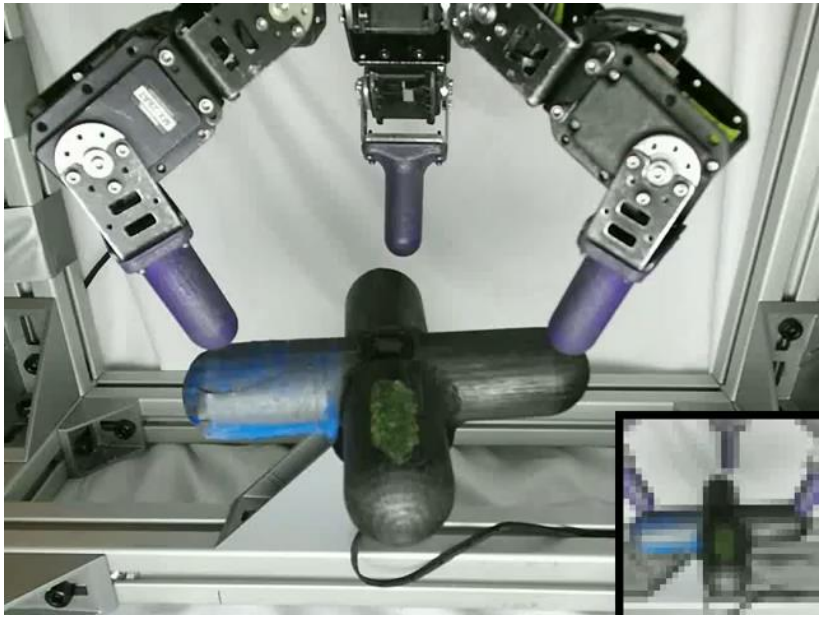
Pieter Abbeel[†] **Sergey Levine**[†‡]

Abstract

Model-free deep reinforcement learning (RL) algorithms have been successfully applied to a range of challenging sequential decision making and control tasks. However, these methods typically suffer from two major challenges: high sample complexity and brittleness to hyperparameters. Both of these challenges limit the applicability of such methods to real-world domains. In this paper, we describe Soft Actor-Critic (SAC), our recently introduced off-policy actor-critic algorithm based on the maximum entropy RL framework. In this framework, the actor aims to simultaneously maximize expected return and entropy; that is, to succeed at the task while acting as randomly as possible. We extend SAC to incorporate a number of modifications that accelerate training and improve stability with respect to the hyperparameters, including a constrained formulation that automatically tunes the temperature hyperparameter. We systematically evaluate SAC on a range of benchmark tasks, as well as challenging real-world tasks such as locomotion for a quadrupedal robot and robotic manipulation with a dexterous hand. With these improvements, SAC achieves state-of-the-art performance, outperforming prior on-policy and off-policy methods in sample-efficiency and asymptotic performance. Furthermore, we demonstrate that, in contrast to other off-policy algorithms, our approach is very stable, achieving similar performance across different random seeds. These results suggest that SAC is a promising candidate for learning in real-world robotics tasks.

1 Introduction

Model-free deep reinforcement learning (RL) algorithms have been applied in a range of challenging domains, from games (Mnih et al., 2013; Silver et al., 2016) to robotic control (Gu et al., 2017; Haarnoja et al., 2018b). The combination of RL and high-capacity function approximators such as neural networks holds the promise of automating a wide range of decision making and control tasks, but widespread adoption of these methods in real-world domains has been hampered by two major



Fast and Stable Learning!

Soft Actor-Critic

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

Tuomas Haarnoja [1] Aurick Zhou [1] Pieter Abbeel [1] Sergey Levine [1]

Abstract

Model-free deep reinforcement learning (RL) algorithms have been demonstrated on a range of challenging decision making and control tasks. However, these methods typically suffer from two major challenges: very high sample complexity and brittle convergence properties, which necessitate meticulous hyperparameter tuning. Both of these challenges severely limit the applicability of such methods to complex, real-world domains. In this paper, we propose soft actor-critic, an off-policy actor-critic deep RL algorithm based on the maximum entropy reinforcement learning framework. In this framework, the actor aims to maximize expected reward while also maximizing entropy. That is, to succeed at the task while acting as randomly as possible. Prior deep RL methods based on this framework have been formulated as Q-learning methods. By combining off-policy updates with a stable stochastic actor-critic formulation, our method achieves state-of-the-art performance on a range of continuous control benchmark tasks, outperforming prior on-policy and off-policy methods. Furthermore, we demonstrate that, in contrast to other off-policy algorithms, our approach is very stable, achieving very similar performance across different random seeds.

1. Introduction

Model-free deep reinforcement learning (RL) algorithms have been applied in a range of challenging domains, from of these methods in real-world domains has been hampered by two major challenges. First, model-free deep RL methods are notoriously expensive in terms of their sample complexity. Even relatively simple tasks can require millions of steps of data collection, and complex behaviors with high-dimensional observations might need substantially more. Second, these methods are often brittle with respect to their hyperparameters: learning rates, exploration constants, and other settings must be set carefully for different problem settings to achieve good results. Both of these challenges severely limit the applicability of model-free deep RL to real-world tasks.

One cause for the poor sample efficiency of deep RL methods is on-policy learning: some of the most commonly used deep RL algorithms, such as TRPO (Schulman et al., 2015), PPO (Schulman et al., 2017b) or A3C (Mnih et al., 2016), require new samples to be collected for each gradient step. This quickly becomes extravagantly expensive, as the number of gradient steps and samples per step needed to learn an effective policy increases with task complexity. Off-policy algorithms aim to reuse past experience. This is not directly feasible with conventional policy gradient formulations, but is relatively straightforward for Q-learning based methods (Mnih et al., 2015). Unfortunately, the combination of off-policy learning and high-dimensional, nonlinear function approximation with neural networks presents a major challenge for stability and convergence (Bhatnagar et al., 2009). This challenge is further exacerbated in continuous state and action spaces, where a separate actor network is often used to perform the maximization in Q-learning. A commonly used algorithm in such settings, deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015), provides for sample-efficient learning but is notoriously challenging

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

Actor Critic

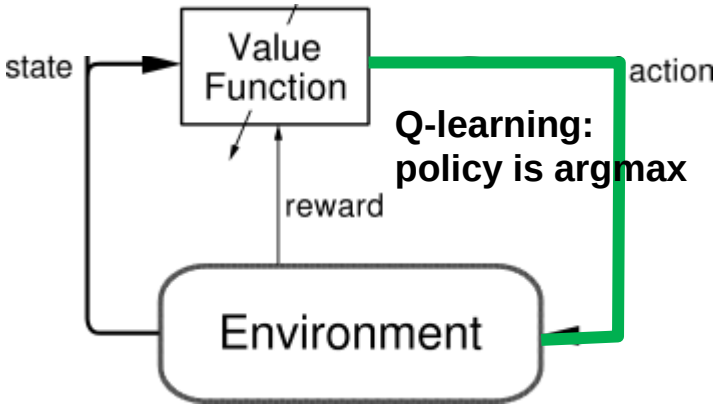
- Actor = Policy $\pi(\cdot, s_t)$
- Critic = Value Function $Q(s_t, a_t)$

Value Function based RL(DQN):

- Policy is represented implicitly
- a = argmax Q(s, a)
- Sample Efficient

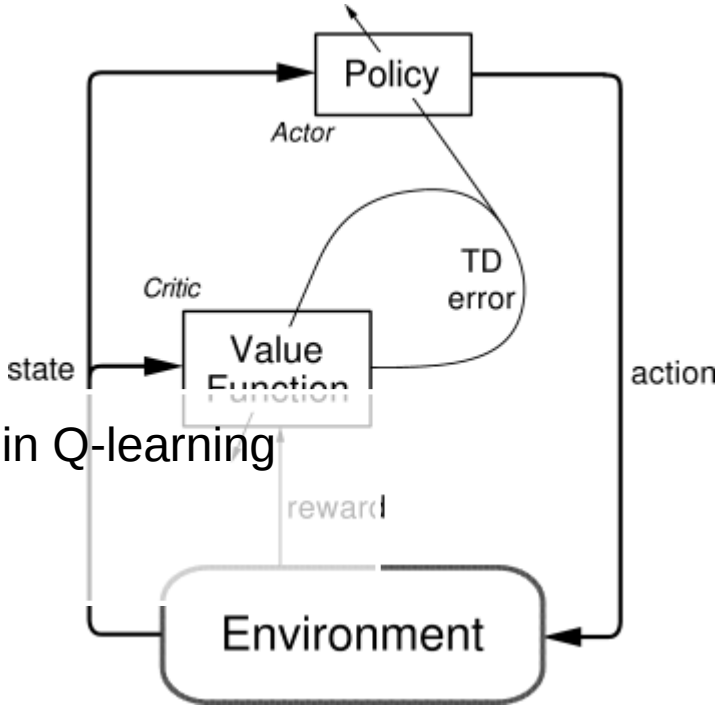
Policy based RL(next week):

- Directly Optimize Policy
- Good for Continuous Domain
- Stable Learning



Actor Critic

- Take best of both worlds
- Nice Continuous performance
- Sample Efficiency



- Update **Value Function** by TD error as we did in Q-learning
- Update **Policy** based on the Value Function

Not optimizing Policy directly
=> We can use **Off-Policy** samples!

Soft Actor-Critic

Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

?

Off-Policy vs On-Policy

- Q: How many of them are off-policy algorithms?

DQN, Rainbow, DDPG, SAC

- A: 4

Update Agent based on samples that are **not obtained from current policy**
Ex) Replay Buffer => Faster Training

- Examples of on-policy methods?

Many of the Policy Gradient Methods (TRPO, PPO, etc.)

On-Policy: PPO

Only use Samples Generated by current policy

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
8: **end for**

Off-Policy: DDPG

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

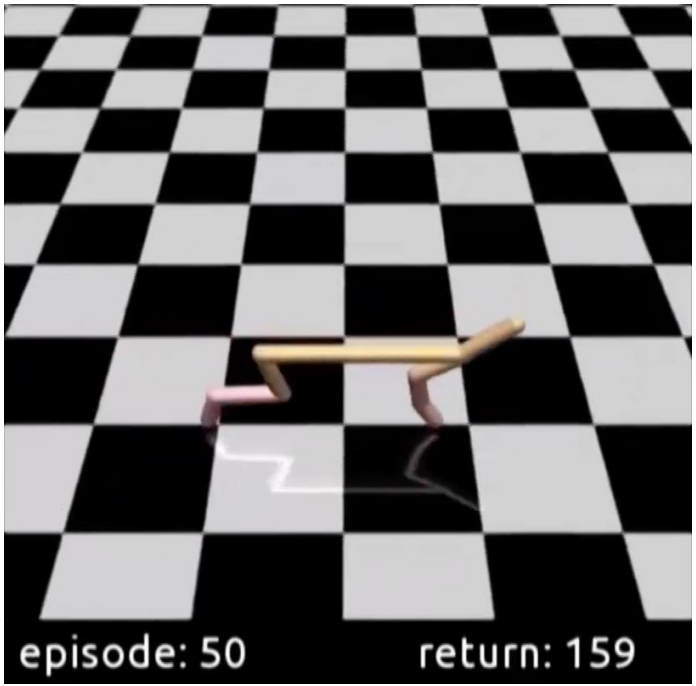
Use Samples from Previous Policies

Deep Deterministic Policy Gradient (DDPG)

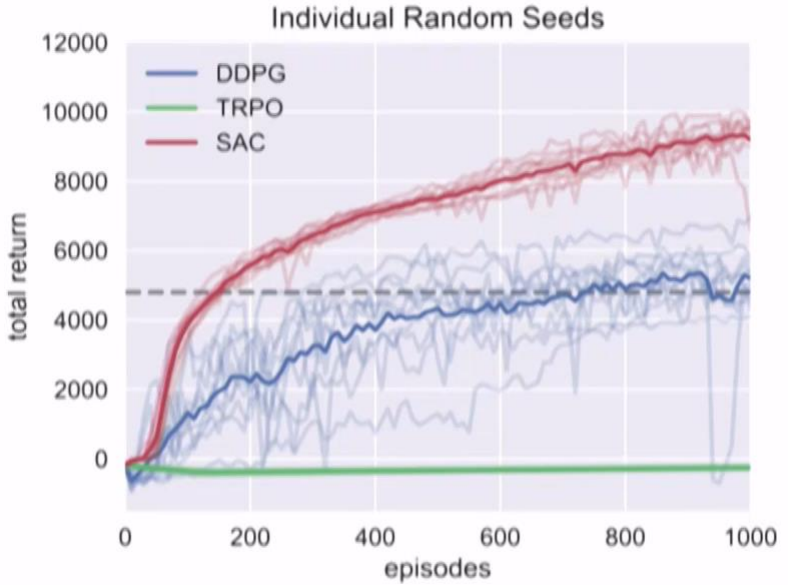
Off-line, Actor Critic

- Sensitive to random seeds
- Brittle w.r.t. hyperparameter settings
  - learning rate, exploration constant, etc.

Off-Policy



HalfCheeta



Brittle to Hyperparameter → Soft : less tuning

Soft Actor-Critic

Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

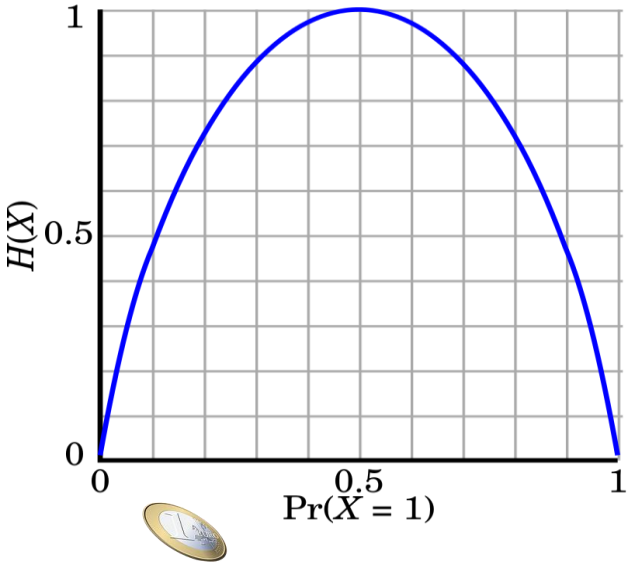
Same as DDPG !

Maximum Entropy RL

- Entropy? Level of “Uncertainty”, “Surprise”

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log \mathrm{P}(x_i)$$

$$\mathrm{H}(\mathrm{P}) = \mathop{\mathbb{E}}_{x \sim \mathrm{P}}[-\log \mathrm{P}(x)]$$



- Coin Tossing example.

Q. Compute the Entropy of 2 cases:

1. P[Head] = 1, P[Tail]=0
2. P[Head]=0.5, P[Tail] = 0.5

- Entropy of a Deterministic Policy = 0

Maximum Entropy RL

- Soft Actor Critic: Optimize policy based on **maximum entropy**

$$\pi^* = \arg\max_{\pi} \mathop{\mathrm{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\Big(R(s_t,a_t,s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right)\Big)\right],$$

Added Entropy Term

knob or temperature

- Balance
  - Reward
  - Exploring Uncertainty

- Original Policy when $\alpha = 0$

Maximum Entropy RL

- Soft Actor Critic: Optimize policy based on **maximum entropy**

$$\pi^* = \arg\max_{\pi} \mathop{\mathrm{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\Big(R(s_t,a_t,s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right)\Big)\right],$$

- Modify Value Functions to achieve Maximum Entropy policy

$$V^{\pi}(s) = \mathop{\mathrm{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\Big(R(s_t,a_t,s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right)\Big)\,\middle|\, s_0 = s\right]$$

$$\begin{aligned} Q^{\pi}(s,a) &= \mathop{\mathrm{E}}_{\substack{s'\sim P\\ a'\sim\pi}}\left[R(s,a,s') + \gamma\left(Q^{\pi}(s',a') + \alpha H\left(\pi(\cdot|s')\right)\right)\right] \\ &= \mathop{\mathrm{E}}_{s'\sim P}\left[R(s,a,s') + \gamma V^{\pi}(s')\right]. \end{aligned}$$

Soft Policy Iteration ⟹ Soft Actor-Critic

1. Soft policy evaluation:

Fix policy, apply soft Bellman backup until converges:

$$Q(s,a) \leftarrow r(s,a) + \mathbb{E}_{s' \sim p_s,\, a' \sim \pi}[Q(s',a') - \log \pi(a'|s')]$$

This converges to Q^{π}

2. Soft policy improvement:

Update the policy through information projection:

$$\pi_{new} = \underset{\pi}{\operatorname{argmin}}\, \mathrm{D}_{\mathrm{KL}}\left(\pi'(\cdot|s) \| \frac{1}{Z} \exp \mathrm{Q}^{\pi_{old}}(s,\cdot) \right)$$

For new policy, we have $Q^{\pi^{new}} \geq Q^{\pi^{old}}$.

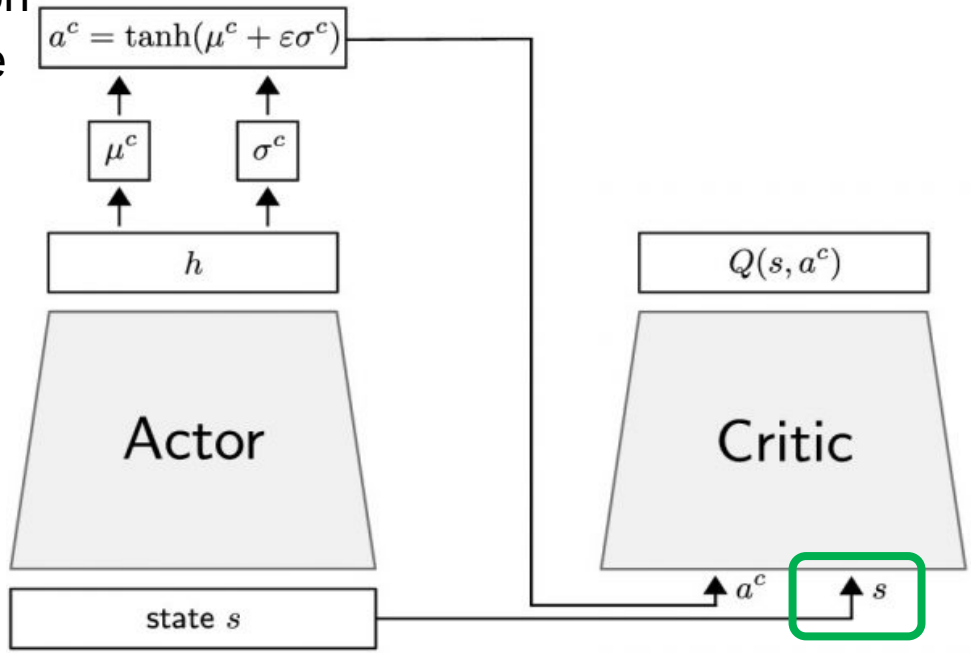
1. Take one stochastic gradient step to minimize Bellman residual

2. Take one stochastic gradient step to minimize KL divergence

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

Stochastic Actor

- **Policy Network** outputs **mean** and **variance** of (squashed) Gaussian
- Use **mean** of gaussian for evaluation
- **Critic** accepts both action and state

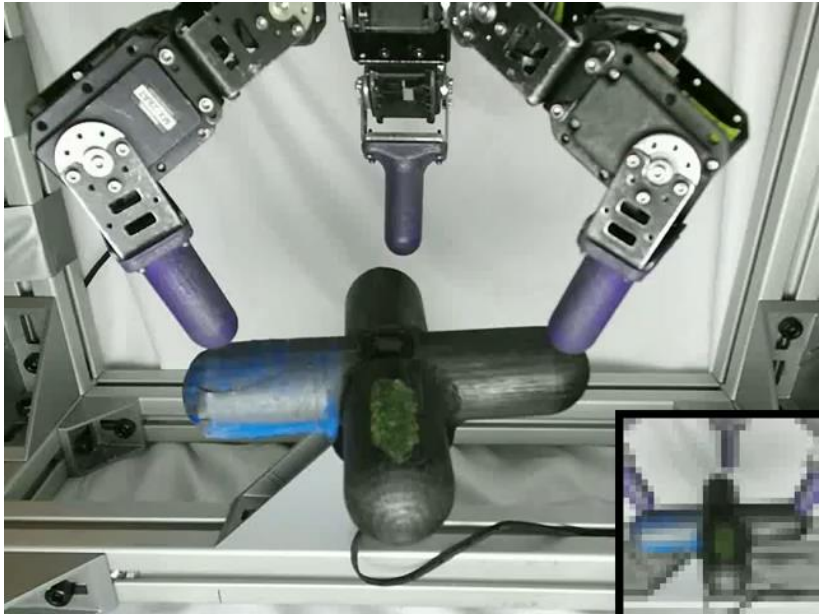


Algorithm 1 Soft Actor-Critic

Input: θ_1, θ_2, ϕ ▷ Initial parameters
$\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$ ▷ Initialize target network weights
$\mathcal{D} \leftarrow \emptyset$ ▷ Initialize an empty replay pool
for each iteration **do**
  for each environment step **do** — Sample Gen.
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$ ▷ Sample action from the policy
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ ▷ Sample transition from the environment
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$ ▷ Store the transition in the replay pool
  end for
  for each gradient step **do** — Network Update
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update policy weights
    $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$ ▷ Adjust temperature
    $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target network weights
  end for
end for
Output: θ_1, θ_2, ϕ ▷ Optimized parameters

Trained on flat terrain

Policy robust to difficult terrains



Q&A

CS8803
Student Presenter: Jeonghwan Kim