Actor Free critic Update for Off-policy and Offline learning

Abstract

This work investigates Actor-Free critic Updates (AFU), a novel reinforcement learning algorithm that decouples critic updates from the actor through value-advantage decomposition, comparing its performance against traditional actor-critic methods in both off-policy learning with randomly sampled data and offline-to-online transition scenarios. Our experiments across multiple continuous control environments demonstrate that while AFU does not consistently outperform traditional methods in pure off-policy learning with random data, it exhibits superior stability during the critical transition from offline to online learning, suggesting significant potential for applications where sample efficiency and smooth deployment from pretraining are essential.

Introduction

Reinforcement learning algorithms for continuous control have seen remarkable advances with actor-critic architectures becoming the standard approach. These methods combine policy gradient techniques with value-based learning to enable sample-efficient optimization in continuous action spaces. However, traditional actor-critic methods like Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic (SAC) face important limitations in their ability to learn from arbitrary data distributions and other offline methods like Implicit Q-Learning (IQL) and Calibrated Q-Learning (Cal-QL) show limitations to transition smoothly from offline to online learning.

In this work, we investigate Actor-Free critic Updates, a recently proposed algorithm that fundamentally redesigns the actor-critic architecture. Unlike traditional approaches where critic updates depend on actions sampled by the actor, AFU maintains critic updates completely independent from the actor through value and advantage decomposition with conditional gradient scaling. This structural departure theoretically enables more effective learning from arbitrary data distributions and potentially smoother transitions between offline and online learning phases.

We conducted comprehensive experiments to test two main hypotheses: first, that AFU can learn more effectively from randomly sampled state-action pairs than traditional algorithms due to its actor-independent critic updates; second, that AFU exhibits greater stability during the transition from offline to online learning due to its approach to solving the max-Q problem. Our experiments spanned multiple continuous control environments of varying complexity, from simple tasks like CartPole to more complex scenarios like Lunar Lander, providing a thorough evaluation of these hypotheses.

Related Work

Reinforcement learning algorithms for continuous control domains have evolved significantly in recent years. The development of actor-critic architectures has proven particularly effective, with algorithms like DDPG [4] and SAC [5] establishing themselves as standard approaches. These methods combine the advantages of policy gradient methods with value-based learning, enabling sample-efficient learning in continuous action spaces.

DDPG employs a deterministic actor that maximizes a learned Q-function, operating entirely off-policy to improve sample efficiency. However, DDPG presents stability issues, demonstrating sensitivity to hyperparameter choices and exploration strategies. SAC extends this framework by incorporating entropy maximization, encouraging exploration while learning a stochastic policy. The entropy term provides additional stability, allowing SAC to achieve both better sample efficiency and final performance compared to DDPG in many domains.

The challenge of addressing the distribution shift when transitioning from offline to online reinforcement learning has received increasing attention. Offline reinforcement learning methods train agents using previously collected datasets without environment interaction, eliminating exploration costs but introducing optimization difficulties. IQL [1] approaches this problem by completely avoiding direct querying of the learned Q-function with unseen actions during training, using expectile regression to estimate the maximum value of Q-functions. Similarly, CAL-QL [2] applies a constraint to the Conservative Q-learning (CQL) framework to reduce overestimation bias during offline learning while enabling efficient online fine-tuning. Both methods aim to mitigate performance degradation observed when agents trained offline begin interacting with environments directly.

The ability to learn truly off-policy—from data generated by arbitrary or random policies—represents another research direction. Traditional actor-critic methods are categorized as off-policy but often struggle when presented with data significantly different from their current policy distribution. This limitation arises because their critic updates depend on actions sampled by the actor, creating an implicit coupling that restricts genuine off-policy learning. Some algorithms have attempted to address this issue through several methods but the fundamental actor-critic interdependence remains.

AFU [3] introduces a structural departure from previous approaches by maintaining critic updates that remain entirely independent from the actor. Unlike other algorithms derived from Q-learning for continuous control, AFU aims to solve the maximization problem inherent in Q-learning through a mechanism based on value and advantage decomposition, employing conditional gradient scaling. This approach potentially enables more effective learning from arbitrary data distributions without requiring explicit constraints during the critic learning phase.

Preliminaries

We consider a discounted infinite horizon Markov Decision Problem (MDP) defined as a tuple $< S, A, T, R, \gamma >$, where S represents the state space, A denotes a continuous action space, T is a stochastic transition function, $R : S \times A \to \mathbb{R}$ is a reward function, and $0 \leq \gamma < 1$ is a discount factor. When an agent performs an action $a \in A$ in state $s \in S$, it transitions to a new state s' according to the transition probability $T(s'|s, a)$ and receives a reward $r = R(s, a)$. We denote transitions as tuples (s, a, r, s').

The goal in reinforcement learning is to find a policy $\pi : S \to A$ that maximizes the expected sum of discounted rewards. The optimal Q-function Q^* is defined as:

$$Q^*(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a, \pi^*\right],$$

where the policy used from $t = 1$ onwards is π^*, which selects actions optimally in every state. The optimal value function V^* satisfies $V^*(s) = \max_{a \in A}(Q^*(s, a))$.

In deep reinforcement learning, we approximate the value functions using neural networks. We denote by V_φ a function approximator for the value function, and by Q_ψ a function approximator for the Q-function (the critic), where φ, and ψ are the parameter vectors of neural networks.

On-policy and off-policy learning represent two distinct approaches in reinforcement learning. In on-policy learning, the agent learns from data collected using its current policy. In contrast, off-policy learning allows the agent to learn from data collected using any policy, including random exploration or previously stored experiences. This distinction is crucial as truly off-policy algorithms can potentially learn more efficiently by reusing diverse experiences.

Offline reinforcement learning takes the off-policy concept further by learning entirely from a fixed dataset of previously collected transitions without any environment interaction during training. When transitioning from offline to online learning (where the agent begins to interact with the environment), algorithms often suffer from distribution shift problems as the learned policy encounters states and actions not represented in the offline dataset.

Actor-critic methods combine policy-based and value-based approaches. The actor (policy network) determines which actions to take, while the critic (value network) evaluates these actions. In traditional actor-critic architectures like SAC and DDPG, the critic's updates depend on actions sampled by the actor, creating an interdependence between the two components. The AFU algorithm represents a departure from this approach by updating the critic independently of the actor.

The temporal difference (TD) learning used in many algorithms aims to minimize the following loss function for the critic:

$$L_Q(\psi) = \mathbb{E}_{(s,a,r,s')\sim B}\Big[\Big(Q_\psi(s,a) - r - \gamma V_\varphi(s')\Big)^2\Big],$$

where B represents a mini-batch of transitions sampled from an experience replay buffer.

Methods

We implemented several state-of-the-art deep reinforcement learning algorithms using PyTorch and the BBRL (BlackBoard Reinforcement Learning) framework. BBRL provides a modular architecture where agents interact through a shared workspace, facilitating the implementation of complex algorithms with clear separation of components. Each algorithm implementation follows the same structure, comprising neural network architectures, training procedures, and evaluation methods. Our hyperparameters are given in Appendix A page 15. More details about these algorithms are given in Appendix C page 16.

(DDPG) Deep Deterministic Policy Gradient combines the deterministic policy gradient algorithm with deep neural networks. It consists of two primary networks. The actor network implements a deterministic policy $\pi(s)$ that maps states to specific actions. The critic network evaluates the quality of state-action pairs by approximating the Q-function $Q(s,a)$. For stable learning, DDPG employs a target network for the critic, which is updated using soft updates: $\theta^{\text{target}} \leftarrow \tau\theta + (1-\tau)\theta^{\text{target}}$, where τ is a small value controlling the update rate. DDPG also uses a replay buffer to store and randomly sample transitions, breaking the correlation between consecutive samples and stabilizing learning.

(SAC) Soft Actor-Critic extends the actor-critic architecture by incorporating entropy maximization, encouraging exploration through stochastic policies. Our SAC implementation includes a policy network, a Q-network and a V-network. Unlike DDPG's deterministic policy, SAC's policy is stochastic, outputting a Gaussian distribution over actions. The network produces both the mean $\mu(s)$ and log standard deviation $\log(\sigma(s))$ of this distribution. Actions are sampled using the reparameterization trick and squashed through a tanh function to bound them. SAC employs two Q-networks to mitigate overestimation bias, a common issue in Q-learning. Both networks approximate $Q(s,a)$ and the minimum of their predictions is used for updates. The value network estimates the state value $V(s)$, which represents the expected future

return starting from state s when following the policy. SAC optimizes the expected return plus the entropy of the policy using a temperature parameter that determines the relative importance of entropy versus reward. This parameter is automatically adjusted during training to achieve a target entropy.

(AFU) The Actor-Free critic Updates algorithm represents a significant departure from traditional actor-critic methods. While algorithms like DDPG and SAC update their critic using actions generated by the actor, AFU decouples these components, enabling critic updates that are truly independent of the actor. AFU's architecture consists of two value networks, two advantage networks, a Q-network and a policy network. The advantage networks estimates the advantage function $A(s, a)$, which represents how much better taking action a in state s is compared to the average action.

It uses six neural networks: a main critic network with parameters ψ, twin value networks with parameters φ_1 and φ_2, their target versions with parameters $\varphi_{1'}$ and $\varphi_{2'}$, twin advantage networks with parameters ξ_1 and ξ_2, and a policy network with parameters θ. The temperature parameter α controls exploration, with target entropy set to $-|\mathcal{A}|$ where $\mathcal{A}$ is the action space. The gradient reduction parameter $\rho \in [0, 1]$ controls value function learning dynamics. Experiences (s, a, r, s') are stored in a replay buffer $\mathcal{B}$ of capacity N, from which mini-batches of size B are sampled. Networks are updated using learning rates l_Q, l_V, l_π, and l_α for critic, value/advantage networks, policy, and temperature respectively, with target networks soft-updated at rate τ.

The first is a Q-network, which does a forward pass on a state-action pair and estimates the expected reward of an agent taking action a in state s. To update its weights, we compute the following loss:

$$L_Q(\psi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{B}} \left[\left(Q_\psi(s, a) - r - \gamma \min_{i\in\{1,2\}} V_{\varphi_{i'}}(s') \right)^2 \right].$$

The second is a combined value and advantage loss for each network pair, which updates both value and advantage networks based on the relation $Q(s, a) = V(s) + A(s, a)$. To update their weights, we compute:

$$L_{\text{VA}}(\varphi_i, \xi_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{B}} \left[Z\left(\Upsilon_i^a(s) - r - \gamma \min_{i\in\{1,2\}} V_{\varphi_{i'}}(s'), A_{\xi_i}(s, a) \right) \right],$$

where:

$$Z(x, y) = \begin{cases} (x + y)^2 & \text{if } x \leq 0 \\ x^2 + y^2 & \text{otherwise.} \end{cases}$$

$$\Upsilon_i^a(s) = (1 - \rho.I_i(s, a))V_{\varphi_i}(s) + \rho.I_i(s, a).V_{\varphi_i}^{\text{nograd}}(s)$$

$$I_i(s, a) = \begin{cases} 1 & \text{if } V_{\varphi_i}(s) + A_{\xi_i}(s, a) < Q_\psi(s, a) \\ 0 & \text{otherwise.} \end{cases}.$$

The last is a policy network that parameterizes a Gaussian distribution over actions. It outputs mean actions and learns log standard deviations as parameters. Using the reparameterization trick and tanh squashing for bounded actions, the policy is optimized by minimizing:

$$L_\pi(\theta) = \mathbb{E}_{s\sim\mathcal{B}} \left[\alpha \log(\pi_\theta(a_s \mid s)) - Q_\psi(s, a_s) \right].$$

The key innovation in AFU lies in how it computes the value and advantage functions. In reinforcement learning with continuous action spaces, the critic update requires estimating $V(s') = \max_{a'} Q(s', a')$ for the target value calculation. This creates the max-Q problem. Instead of relying on the actor to estimate the maximum value of $Q(s, a)$ over actions, AFU directly decomposes $Q(s, a)$ into $V(s) + A(s, a)$. This decomposition, combined with a conditional gradient scaling mechanism, allows AFU to solve the maximization problem in Q-learning without depending on the actor. This mechanism allows AFU to maintain stable learning by adaptively adjusting the gradient flow depending on whether the current estimate falls short of the target. This technical approach to solving the max-Q

problem was a key motivation for our study, as we expected this independence would provide advantages in off-policy learning scenarios.

(IQL) Implicit Q-Learning approaches offline reinforcement learning by avoiding direct querying of the Q-function with unseen actions during training. This algorithm uses expectile regression to estimate the maximum value of Q-functions without requiring access to the optimal action. IQL operates on three main components: a value network, two Q-networks to reduce overestimation bias, and a policy network. The value function is trained using expectile regression at a specific quantile τ, enabling it to approximate the maximum of the Q-function distribution. The policy is then extracted through advantage-weighted regression, allowing it to favor actions with higher advantages while remaining within the data distribution. This approach effectively addresses the distribution shift problem in offline RL by never evaluating actions outside the training dataset, making it well-suited for purely offline learning scenarios but potentially limiting its adaptability during transitions to online learning.

(Cal-QL) Calibrated Q-Learning builds upon conservative Q-learning approaches for offline reinforcement learning while enabling smoother transition to online learning. CAL-QL employs a Q-network, a target network, and a policy network similar to SAC's stochastic policy. What distinguishes CAL-QL is its use of reference values precomputed from the dataset using Monte Carlo returns, which serve as a calibration mechanism. The learning objective combines a standard temporal difference error with a conservative regularization term that explicitly penalizes out-of-distribution actions. This calibration helps prevent both underestimation and overestimation of Q-values, addressing a key challenge in offline-to-online transition where traditional conservative approaches tend to be overly pessimistic. CAL-QL's balanced approach allows it to maintain sufficient conservatism during offline learning while facilitating effective exploration when transitioning to online learning.

(Off-policy Reinforcement Learning) Off-policy reinforcement learning decouples the behavior policy used for collecting experiences from the target policy being optimized. This separation enables learning about one policy while following another, distinguishing it from on-policy methods.

The core challenge is solving the Bellman equation through samples generated by a policy different from the one being optimized. This involves addressing the max-Q problem—correctly estimating the maximum Q-value for each state—which drives the bootstrapping process in temporal difference methods. Practical off-policy algorithms must approximate this solution under distribution mismatch conditions.

A maze environment illustrates these concepts. In true off-policy learning with uniform random sampling, we completely decouple the behavior policy from both the target policy and temporal structure. Rather than sequential exploration, this approach randomly teleports to any cell, takes an arbitrary action, observes the result, and repeats—without following connected trajectories.

The critic network updates through two distinct pathways with actor-critic architectures. The first pathway comes through actor-generated samples: in traditional actor-critic methods, the behavior policy $\beta(s)$ generates actions that determine what state-action pairs the critic evaluates. This creates a closed loop where the critic's learning depends on the quality of the actor's exploration. The second pathway comes directly from environment feedback: when rewards are immediate and dense, the critic can learn value estimations by directly associating observed state-action pairs with their outcomes, even if those actions were selected randomly. This second pathway explains why algorithms can sometimes learn from completely random data despite the theoretical limitation that « an

on-policy algorithm can only converge if the data is on-policy. »

However, direct updates become ineffective in environments with sparse rewards or those requiring long action sequences. If reaching a goal in a 100×100 maze requires a specific sequence of movements, the probability of randomly sampling this sequence becomes vanishingly small. Even when rare rewards are encountered, propagating their value across long temporal horizons presents challenges for function approximators.

In a maze with sparse rewards, value information flows through the state space via backward chaining. When only a single cell contains the reward, Q-values remain uninformative until that reward state is discovered. Once found, values propagate backward—first updating states adjacent to the goal, then states two steps away, and so on. States far from the reward receive meaningful updates only after the entire chain of intermediate states has been properly valued.

For true off-policy learning with uniform random sampling, direct reward observations become extremely rare in sparse environments. Furthermore, even after observing a reward, propagating this information remains challenging. State-action pairs require already-valued subsequent states to update meaningfully. This backward chaining process becomes highly inefficient with random sampling, requiring thousands of samples for states far from rewards.

This explains why pure off-policy learning with uniform random sampling faces limitations in environments with sparse rewards or long action sequences. The direct feedback pathway becomes almost non-existent in these settings, forcing the critic to rely entirely on bootstrapping from estimated values of subsequent states—a process that becomes exponentially more difficult as the dependency chain lengthens.

Algorithms like AFU address these challenges through their unique approach to the max-Q problem. By decoupling value estimation from the actor's policy and directly approximating the maximum Q-value for each state, AFU breaks the strong dependency between actor and critic. This advantage function decomposition allows AFU to build accurate value estimates even when the behavior policy $\beta(s)$ differs significantly from the target policy $\pi(s)$, as it doesn't rely exclusively on the actor's action selections to estimate state values. This property makes AFU particularly suited for off-policy learning scenarios where sample efficiency and value propagation through state spaces with sparse rewards are critical concerns.

(Offline Reinforcement Learning) Offline reinforcement learning involves training an agent using a fixed dataset of transitions without any direct environment interaction. This dataset often contains samples from expert or near-expert policies, enabling the agent to develop foundational value estimations and behaviors. When properly trained, the agent can then transition to online learning, where it interacts directly with the environment to refine and optimize its policy further.

A significant challenge in this transition is the distribution shift problem. When an agent trained on a fixed dataset begins interacting with the environment, it encounters states and actions not represented in the original dataset. Traditional approaches address this problem by adopting conservative policies that underestimate Q-values for out-of-distribution actions, preventing the agent from selecting potentially harmful actions.

However, this conservatism can lead to performance degradation during the transition to online learning. For algorithms that severely underestimate Q-values, actions encountered during online exploration may appear disproportionately valuable compared to those in the dataset. This can trigger a cycle where the agent repeatedly selects suboptimal actions it perceives as valuable, leading to significant performance drops. This problem is particularly severe when the offline dataset comes from an expert policy,

as most out-of-distribution actions are likely suboptimal.

Offline reinforcement learning algorithms often employ conservatism to prevent overestimation of out-of-distribution actions. While this conservatism is beneficial during offline learning, it can hinder exploration during subsequent online learning. Algorithms like IQL and CAL-QL address this through different mechanisms: IQL uses expectile regression to estimate maximum values without querying unseen actions, while CAL-QL explicitly constrains Q-values for out-of-distribution actions.

Unlike conservative methods, AFU addresses the distribution shift problem through its advantage function decomposition approach. By not artificially constraining Q-values for unseen actions, but instead directly estimating maximum Q-values through its value-advantage decomposition, AFU theoretically should maintain more stable performance during the transition from offline to online learning, avoiding the sharp performance degradation often observed with other algorithms.

Experimental Setup

Environments. For our experiments, we utilized the *Gymnasium* framework, a widely adopted benchmark for reinforcement learning research. *Gymnasium* provides standardized environments, allowing for reproducible evaluation of algorithmic performance. We primarily focused on continuous control tasks including Pendulum, CartPole (continuous version), LunarLander (continuous version) and MountainCar (continuous version). These tasks represent increasing levels of complexity, reward sparsity, and control difficulty. Each environment poses specific challenges that test the robustness and generalization capabilities of learning algorithms, particularly under random or suboptimal data distributions.

To facilitate our research on off-policy learning properties, we extended these environments with additional functionality. Each environment was modified to support direct state manipulation through the implementation of a `_set_state()` method. This modification enables control over the system state, allowing us to sample uniformly from the state-action space during our off-policy learning experiments. It should be noted that such direct state manipulation represents a research tool rather than a practical capability in real-world scenarios, where complete state control is rarely possible.

CartPole is a classic control problem involving a pole hinged on a moving cart. The objective is to apply continuous forces to the cart to keep the pole upright for as long as possible. In the continuous version, the action space consists of a single scalar representing the applied force (as opposed to the binary push-left or push-right actions in the discrete variant). Although low-dimensional and typically easy to solve, CartPole is useful for early validation and ablation studies. The dynamics are symmetric and forgiving, yet it reveals instability in algorithms that fail to maintain robustness under noisy or off-distribution data. It also exposes potential degradation in performance when policies begin overfitting to non-representative transitions in the buffer.

MountainCar is a goal-directed task in which an underpowered car must learn to climb a steep hill. The car starts in a valley and must build momentum by first moving away from the goal. The reward is sparse and delayed, with no gradient signal until the car reaches the target. This setup makes learning highly dependent on proper exploration. Although simple in dimensionality, the task's all-or-nothing nature can lead to multimodal reward distributions as movements are penalized unless the hill is climbed. It exposes instability in deterministic policies and brittle actor-critic interactions.

In Pendulum, the agent must swing and stabilize a pendulum in the upright position by applying torque. The continuous reward landscape can produce deceptive gradients and multiple attractor basins. Policies often settle in one

of three modes: complete failure, partial stabilization with oscillations, or full swing-up and stabilization. This makes exploration difficult and exposes the inability of some off-policy algorithms to escape poor local optima originating from initial initialization or transition between reward modes without sufficient support in the data.

Lunar Lander simulates the controlled descent of a lunar module onto a target landing pad. The agent must apply precise thrust to land softly within bounds. LunarLander is highly sensitive to precise control, and its reward structure is sparse with steep penalties. Successful episodes are rare in random exploration, making this environment extremely challenging for purely off-policy training. Agents may learn to avoid extreme penalties without acquiring successful landing strategies, leading to misleading plateaus in learning curves.

Off-policy learning experimental setup. Our first experiment aims to evaluate whether AFU can truly learn from randomly generated data, a capability that would distinguish it from traditional off-policy algorithms like DDPG and SAC. We want to test if AFU can learn effectively from completely random samples with no correlation to any learning policy.

Traditional actor-critic algorithms like SAC and DDPG depend on the actor to generate actions for the critic's updates, creating an implicit coupling. In contrast, AFU computes critic updates independently of the actor, potentially enabling true off-policy learning from arbitrary data distributions.

To test this hypothesis, we designed a protocol where instead of collecting experience through environment interaction with the current policy, we randomly sample states from the observation space and actions from the action space. This sampling is uniform, representing the extreme case of off-policy data with no correlation to any learning policy.

For each environment, we ran 5 independent trials to ensure statistical significance. We implemented this experiment by replacing the standard experience collection process with a uniform random sampling procedure. For each environment, we randomly generated state-action pairs, executed them in the environment to observe rewards and next states, and added these transitions to the replay buffer. The algorithms then trained on these randomly collected transitions for 200,000 time steps without using their own policies to generate training data at any point.

At fixed intervals (every 100 steps), we paused training to evaluate the current policy by running 10 evaluation episodes and recording the average returns. This allowed us to track performance progression throughout the training process and compare the learning capabilities of different algorithms.

This experiment directly tests each algorithm's ability to learn from data that is completely independent of its own policy decisions. If AFU can converge under these conditions while traditional algorithms struggle, it would validate its advantage in truly off-policy learning.

Offline to online transition experimental setup. Our second experiment investigates the stability of algorithms during the transition from offline to online learning. This transition presents a significant challenge due to the distribution shift between the fixed offline dataset and the data generated by the current policy during online learning.

We hypothesize that AFU may exhibit superior stability during this transition due to its ability to estimate maximum Q-values without relying on the actor. This capability potentially reduces the need for explicit conservatism, allowing AFU to adapt more smoothly when transitioning to online learning.

To test this hypothesis, we designed a two-phase experimental protocol. First, we generated datasets by collecting transitions during standard training runs in both Pendulum and Lunar

Lander environments. We specifically captured trajectories during performance evaluation episodes, ensuring a mixture of near-optimal trajectories and diverse actions. This provided a balanced dataset representing different stages of policy learning.

For each environment, we ran 5 independent trials to ensure statistical significance. Each trial consisted of two phases: 200,000 time steps of offline training using the pre-collected datasets, followed by 200,000 time steps of online interaction with the environment. During offline training, the algorithms learned exclusively from the fixed dataset without environment interaction. Once the offline phase was complete, we switched to online learning where the agents began directly interacting with the environment while continuing to update their policies.

We implemented a consistent evaluation protocol throughout both phases. At fixed intervals (every 100 steps), we paused training to evaluate the current policy by running 10 evaluation episodes and recording the average returns. This allowed us to track performance progression throughout both phases and analyze the stability of the transition.

The primary evaluation metric was the area between the learning curve and the maximum stable performance level following the transition point. This quantifies how quickly and stably an algorithm recovers during the online phase, with lower values indicating better performance. We also compared the interquartile means and ranges across the 5 trials to assess both performance and consistency.

Algorithms like IQL and CAL-QL, which employ different conservative mechanisms for offline learning, were included as baselines. IQL uses expectile regression to estimate maximum values without querying unseen actions, while CAL-QL explicitly constrains Q-values for out-of-distribution actions. These were compared against both AFU and standard SAC to evaluate the effectiveness of different approaches to the offline-to-online transition challenge.

Results

Our experimental investigation examines two aspects of deep reinforcement learning algorithms: their ability to learn from random, non-sequential data and their stability when transitioning from offline to online learning. These questions address capabilities needed for practical applications - learning from arbitrary data distributions and adapting when deployed after offline pretraining. Through evaluation across environments of varying complexity, we analyze how AFU's advantage decomposition approach compares to traditional actor-critic methods like SAC and DDPG, as well as to offline RL algorithms like IQL and CAL-QL. Additional visualizations are given in Appendix D page 18.

Off-policy Learning. In this section, we present and analyze the results of our off-policy learning experiments across various continuous control environments. These experiments test whether algorithms can learn from purely random data with no correlation to any policy.

We first examine the results on three environments of increasing complexity: CartPole, MountainCar, and Pendulum. In the CartPole environment (Fig. 1), all three algorithms demonstrated successful convergence when trained on randomly sampled state-action pairs. This outcome contradicts our initial hypothesis, as we expected traditional actor-critic methods like SAC and DDPG to struggle with entirely random data.

An interesting observation in the CartPole results is a temporary performance drop around 50,000 training steps for both AFU and SAC. This pattern may be attributed to their entropy-maximizing approach, which encourages exploration. After this exploratory phase, both algorithms recover and converge to robust solutions.

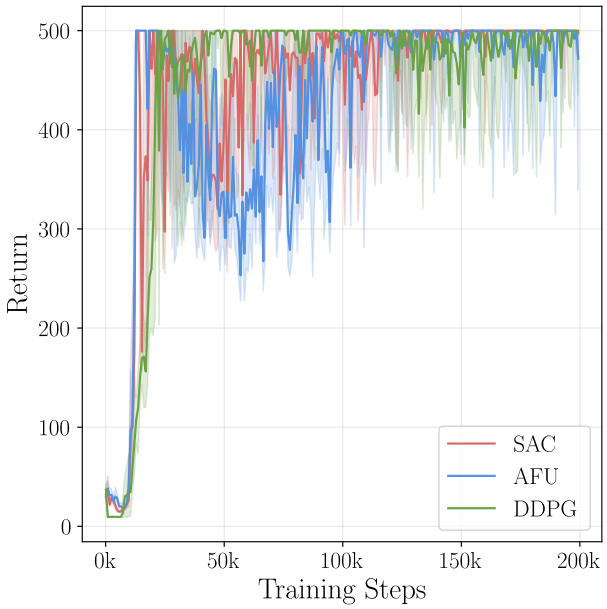

Fig. 1. – Evolution of training performance for SAC, AFU, and DDPG algorithms in the CartPole environment under off-policy learning with uniformly sampled state-action pairs. The curves represent the interquartile means of returns over 200,000 training steps, with shaded regions showing interquartile ranges across 5 independent trials. Performance evaluations were conducted every 100 training steps by deploying the current policy for 10 complete episodes.

The MountainCar environment presents a more challenging task due to its deceptive reward structure, where energy expenditure necessary to climb the hill is penalized unless the flag is reached. Fig. 2 shows that all three algorithms successfully converged under random sampling. This highlights how uniform sampling across the state-action space can overcome exploration challenges posed by deceptive gradients. The uniform sampling enables discovery of the rare rewarding state-action sequences that might be difficult to encounter through standard exploration.

The Pendulum environment provides particularly revealing insights. In off-policy training with random sampling (Fig. 3), AFU converged rapidly alongside SAC and DDPG. This is notable because in standard on-policy training, AFU displayed significant performance variance, successfully learning strategies for favorable initial conditions but struggling with cases where the pendulum starts at the bottom position with zero momentum.

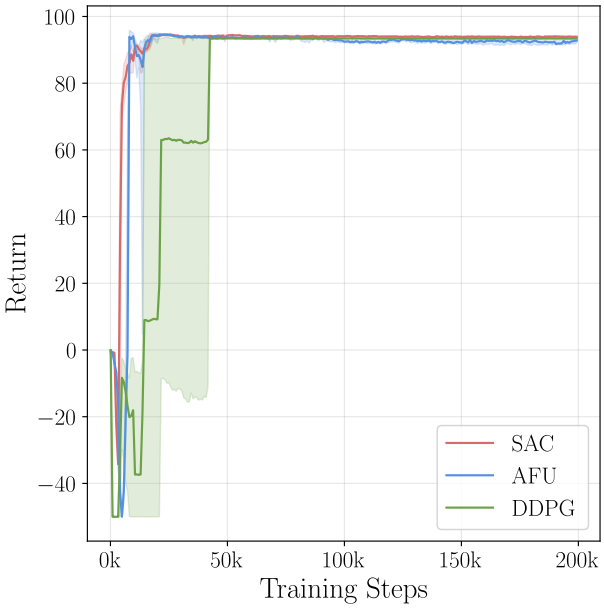

Fig. 2. – Evolution of training performance for SAC, AFU, and DDPG algorithms in the Mountain Car environment under off-policy learning with uniformly sampled state-action pairs. The curves represent the interquartile means of returns over 200,000 training steps, with shaded regions showing interquartile ranges across 5 independent trials. Performance evaluations were conducted every 100 training steps by deploying the current policy for 10 complete episodes.

The value function visualizations for Pendulum (Fig. 4, Fig. 24, and Fig. 25) provide deeper insights into the learned policies. All three algorithms under off-policy training developed similar spiral-structured value functions, characteristic of optimal pendulum control. This contrasts with AFU's on-policy visualization (Fig. 5), which shows a notably different value estimation at the critical bottom position, incorrectly assigning high value to states where the pendulum is at the bottom with zero momentum.

In environments with dense, immediate rewards (like CartPole, MountainCar, and Pendulum), direct feedback from the environment appears

sufficient to drive learning even with random actions. The critic can gradually build meaningful representations by associating state-action pairs with observed rewards and subsequent states.

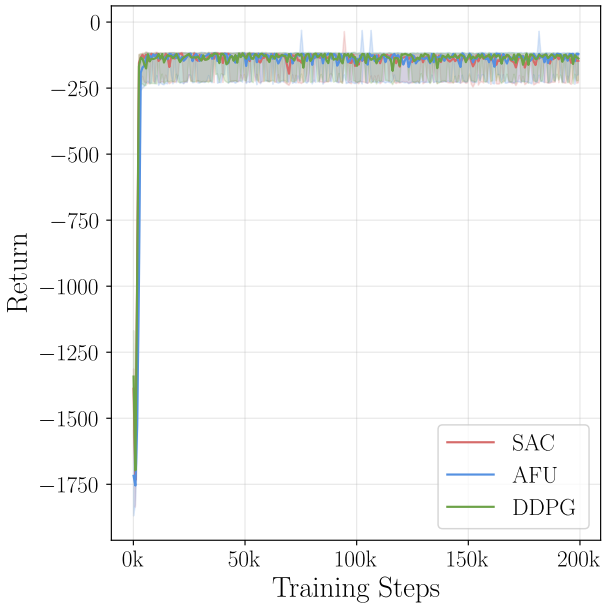

Fig. 3. – Evolution of training performance for SAC, AFU, and DDPG algorithms in the Pendulum environment under off-policy learning with uniformly sampled state-action pairs. The curves represent the interquartile means of returns over 200,000 training steps, with shaded regions showing interquartile ranges across 5 independent trials. Performance evaluations were conducted every 100 training steps by deploying the current policy for 10 complete episodes.

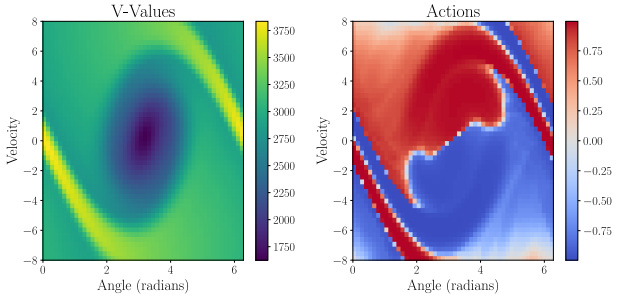

Fig. 4. – Visualization of AFU algorithm's learned value function (left) and action policy (right) in the Pendulum environment under off-policy learning.

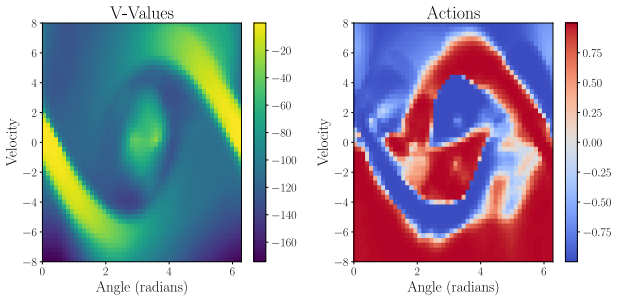

Fig. 5. – Visualization of AFU algorithm's learned value function (left) and action policy (right) in the Pendulum environment under on-policy learning.

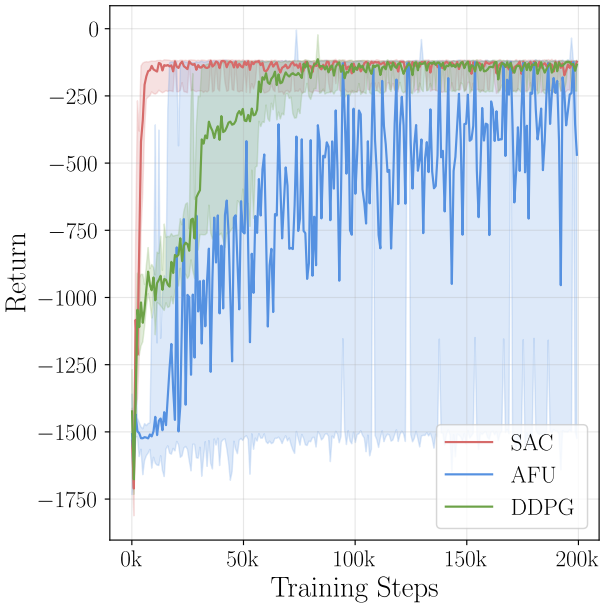

Fig. 6. – Evolution of training performance for SAC, AFU, and DDPG algorithms in the Pendulum environment under on-policy learning with policy-generated samples. The curves represent the interquartile means of returns over 200,000 training steps, with shaded regions showing interquartile ranges across 5 independent trials. Performance evaluations were conducted every 100 training steps by deploying the current policy for 10 complete episodes.

Uniform random sampling also completely removes the exploration problem from the learning process. Rather than relying on an actor to discover high-value regions of the state-action space, random sampling guarantees coverage of the entire space. This explains why algorithms converged faster and more consistently in off-

policy training for simpler environments, and why even problematic cases for AFU in on-policy training (like Pendulum's bottom position) were successfully learned in off-policy mode.

The LunarLander environment represents a significantly more complex control task requiring precise sequential actions to achieve successful landing. As shown in Fig. 7, none of the algorithms demonstrated meaningful learning progress under random sampling. This contrasts with on-policy results, where AFU and SAC successfully learned optimal policies.

The failure of all algorithms in LunarLander indicates fundamental limitations to learning from purely random data when tasks require specific sequential decision-making. This limitation affects all tested approaches regardless of their specific mechanism.

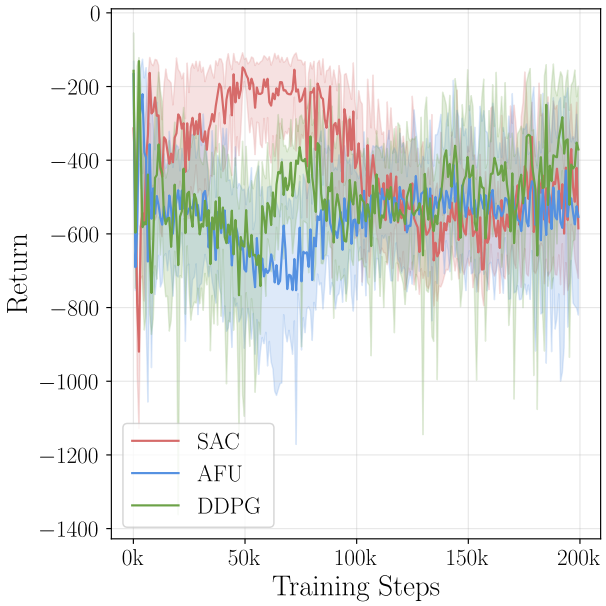

Fig. 7. – Evolution of training performance for SAC, AFU, and DDPG algorithms in the Lunar Lander environment under off-policy learning with uniformly sampled state-action pairs. The curves represent the interquartile means of returns over 200,000 training steps, with shaded regions showing interquartile ranges across 5 independent trials. Performance evaluations were conducted every 100 training steps by deploying the current policy for 10 complete episodes.

We conducted an additional experiment using pre-trained critic values from successful on-policy training to initialize the critics for off-policy learning in LunarLander. Even with this advantage, none of the algorithms showed convergence. We also experimented with larger neural networks to test if increased capacity would help maintain and propagate rare reward signals, but these attempts did not yield successful convergence.

Our findings suggest that for simple environments with dense reward signals, all tested methods can learn effectively from random data. For complex environments with sparse rewards dependent on action sequences, all current approaches struggle. The ability to learn from random samples appears more closely tied to the reward structure, state-action space dimensionality, and temporal dependency of the environment than to the specific algorithm design.

Offline to online transition. Our second experimental investigation focused on the transition from offline to online reinforcement learning. This scenario is particularly relevant in robotics applications, where pretraining an agent on a dataset before deployment can significantly improve sample efficiency and reduce costly or potentially dangerous interactions with the real environment.

The Pendulum environment yielded unexpected results (Fig. 8). All algorithms achieved relatively good performance during offline learning, with SAC nearly converging to optimal policy. This surprising success of SAC in the offline phase aligns with our earlier observation that SAC can learn effectively from off-policy data in this environment.

IQL maintained stable performance during the transition, without the expected decline. This stability may be attributed to the simplicity of the Pendulum environment, where the network learned during offline training generalized well to the online phase. CAL-QL, being a conservative algorithm, showed limited progress

during offline learning, consistent with its pessimistic approach. AFU demonstrated incomplete convergence during offline learning, which corresponds to its difficulties in on-policy learning for this environment, particularly in estimating values when the pendulum is in the bottom position.

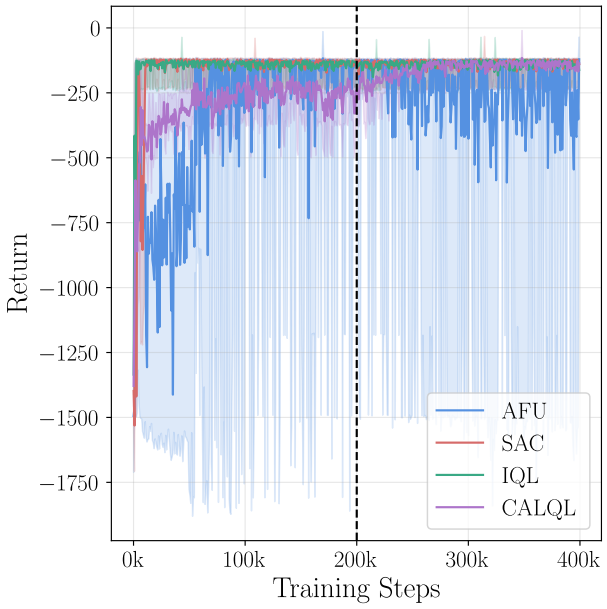

Fig. 8. – Evolution of training performance for AFU, SAC, IQL, and CAL-QL algorithms in the Pendulum environment during offline-to-online transition learning. The curves represent interquartile means of returns over 400,000 training steps, with shaded regions showing interquartile ranges across 5 independent trials. The vertical dashed line at 200,000 steps indicates the transition point from offline learning (using a pre-collected dataset) to online learning (with direct environment interaction).

The visualization of value functions (Figures Fig. 28, Fig. 29, Fig. 30, Fig. 31) confirms these observations, revealing similar value estimation patterns to those observed in on-policy learning.

The Lunar Lander environment provided more insightful results (Fig. 9). SAC nearly converged to a compromise « hovering » policy during offline learning, then rapidly improved toward the optimal landing policy after transitioning to online learning.

IQL, as a pessimistic algorithm, never progressed beyond the compromise policy throughout both phases. Its conservatism prevented exploration of potentially risky but necessary landing strategies, keeping it within the boundaries of the safer hovering policy. As expected from a conservative approach, its performance declined when transitioning to online learning.

CAL-QL similarly remained at the compromise policy level. Its calibration mechanism prevented the severe underestimation that leads to performance collapse, but its conservative nature still constrained exploration, resulting in negative interquartile mean rewards even after online learning, indicating unsuccessful landing attempts on average.

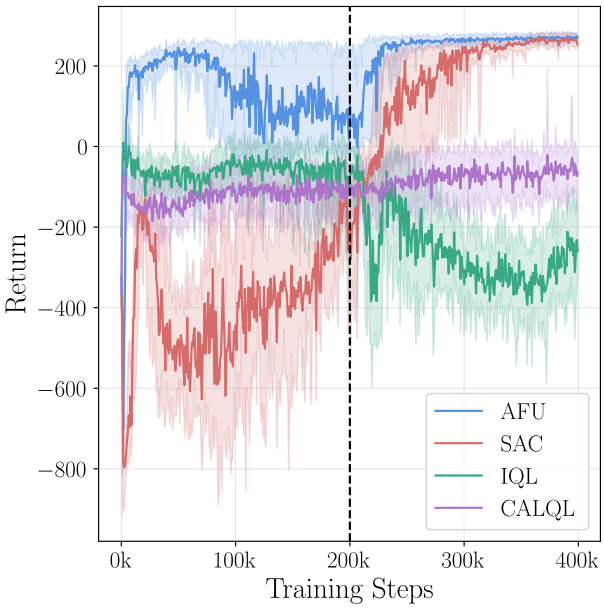

Fig. 9. – Evolution of training performance for AFU, SAC, IQL, and CAL-QL algorithms in the Lunar Lander environment during offline-to-online transition learning. The curves represent the interquartile means of returns over 400,000 training steps, with shaded regions showing interquartile ranges across 5 independent trials. The vertical dashed line at 200,000 steps indicates the transition point from offline learning (using a pre-collected dataset) to online learning (with direct environment interaction).

AFU exhibited particularly interesting behavior for our hypothesis. It achieved high performance early in offline learning, followed by a slight decline, while maintaining predominantly successful landing policies as indicated by positive returns. When transitioning to online learning, AFU immediately resumed improvement, rapidly converging to the optimal policy. The interquartile range suggests consistent successful landings across trials, confirming effective learning. This pattern aligns with the ideal scenario for robotics applications, where offline learning establishes fundamental competencies that are then refined through online interaction.

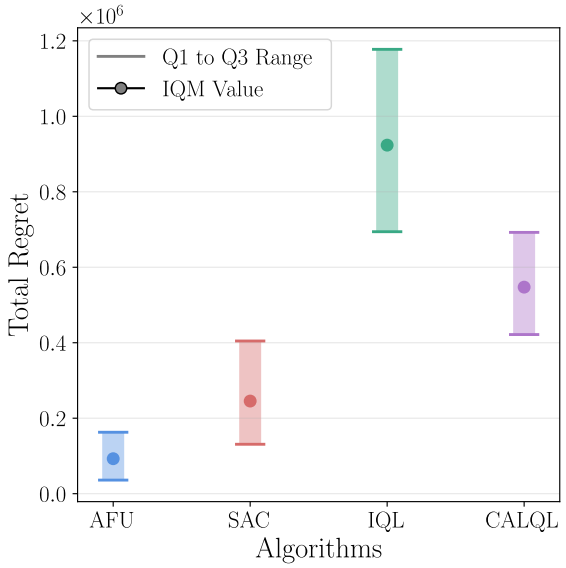

Fig. 10. – Comparison of total regret for different algorithms during offline-to-online transition in the Lunar Lander environment. Lower values indicate better performance, with vertical bars showing the Q1 to Q3 range and dark markers representing the interquartile mean (IQM) value across 5 independent trials.

To quantitatively assess performance during transition, we measured the area under the curve representing the difference between each algorithm's performance and the optimal value (Fig. 10). This total regret metric shows that AFU significantly outperformed other algorithms, with the smallest gap between its performance and the optimal policy. This confirms our hypothesis that AFU's approach to solving the max-Q problem provides advantages during the offline-to-online transition.

Conclusion

Our investigation into AFU's capabilities across different learning scenarios has yielded several important insights regarding our initial hypotheses.

Regarding our first hypothesis about AFU's ability to learn from randomly sampled data, the results were more nuanced than anticipated. In simpler environments (CartPole, MountainCar, and Pendulum), all algorithms, including traditional methods like SAC and DDPG, successfully learned from random data despite theoretical limitations in their critic update mechanisms. This success can be attributed to direct environment feedback enabling critic updates without relying on actor-proposed actions, particularly in environments with dense, immediate rewards. However, in the more complex Lunar Lander environment, which requires coordinated action sequences, no algorithm achieved meaningful learning from random sampling, including AFU. This highlights fundamental limitations in learning from purely random data when tasks require sequential decision-making, regardless of the specific max-Q estimation approach.

Our second hypothesis regarding AFU's stability during offline-to-online transitions was supported by our experiments, particularly in the Lunar Lander environment. AFU demonstrated superior performance during this transition, quickly adapting to online learning without the performance degradation observed in more conservative approaches like IQL and CAL-QL. The quantitative analysis of total regret confirmed AFU's advantage, showing significantly smaller deviations from optimal performance compared to other algorithms. This validates our theoretical understanding that AFU's approach to solving the max-Q problem, which does not artificially constrain Q-values for unseen actions,

provides tangible benefits in realistic learning scenarios.

Overall, our work demonstrates that while the independence between critic and actor updates in AFU does not necessarily translate to superior performance in all off-policy learning scenarios, it provides significant advantages during the critical transition from offline to online learning. This finding has practical implications for applications where sample efficiency is paramount, such as robotics, where pretraining on datasets before deployment can reduce costly or potentially dangerous interactions with the environment. The results suggest that AFU's unique architecture makes it particularly well-suited for such applications.

Appendix A

Hyperparameters.

The following hyperparameters were used for the experiments:

- Hidden size : 256
- Learning rate: 3×10^{-4}
- Discount factor (γ): 0.99
- Batch size: 256
- Replay buffer size: 200 000
- Target network update rate (τ): 0.01
- Gradient reduction (for AFU only) : 0.5
- Noise standard deviation (for DDPG only) : 0.1
- Expectile regression (τ') (for IQL only): 0.9
- Inverse temperature (β) (for IQL only): 3.0
- Conservative factor (α) (for Cal-QL only): 5
- Random repetitions (n) (for Cal-QL only): 4

Reproducibility.

The code for all algorithms and experiments is available at https://github.com/paulchambaz/afu-study. The repository includes detailed instructions for setting up the environment and running the experiments. The code is licensed under the GPLv3 license and freely available.

Appendix B

Implementation Notes on CAL-QL. In our review of the CAL-QL implementation presented in the original paper, we identified two minor inconsistencies in the provided pseudocode that we corrected in our implementation:

1. In the target Q-value calculation, the original code shows:

```
target_qval =
target_critic(batch['observations'],
next_pi_actions)
```

However, this should use the next observations according to the Bellman equation:

```
target_qval =
target_critic(batch['next_observations'],
next_pi_actions)
```

2. In the random actions Q-value calculation, the original code indicates:

```
q_rand_is = critic(batch['observations'],
random_actions) - random_pi
```

For consistency with the log-probabilities used elsewhere in the algorithm, this should instead use the logarithm of the uniform density:

```
q_rand_is = critic(batch['observations'],
random_actions) - log(random_pi)
```

These corrections align with both the mathematical principles of the algorithm and the authors' actual implementation in their source code repository. While these inconsistencies are minor and likely typographical in nature, they are worth noting for those aiming to implement CAL-QL correctly.

Implementation Notes on IQL. In our review of the Implicit Q-Learning implementation presented in the original paper, we identified a minor inconsistency in the provided formula that we corrected in our implementation:

The policy loss function in Equation (7) of the paper shows:

$$L_\pi(\phi) = \mathbb{E}_{(s,a)\sim D}$$

$$\left[\exp\big(\beta\big(Q_\theta(s,a) - V_\psi(s)\big)\big)\log \pi_\phi(a \mid s)\right].$$

However, since this is a loss function that should be minimized during optimization, the correct formula should include a negative sign:

$$L_\pi(\phi) = -\mathbb{E}_{(s,a)\sim D}$$
$$\left[\exp\big(\beta\big(Q_\theta(s,a) - V_\psi(s)\big)\big)\log \pi_\phi(a \mid s)\right].$$

This correction aligns with the mathematical principles of advantage-weighted regression, where the objective is to maximize the likelihood of actions with high advantages. While this inconsistency is minor and likely typographical in nature, it's worth noting for those aiming to implement IQL correctly.

Appendix C

Quick description of DDPG. DDPG optimizes a continuous control policy using deterministic actions and off-policy learning. It uses two neural networks with parameters θ and φ: a critic (Q-network) and an actor (policy network). The critic $Q_\theta(s,a)$ estimates expected returns for state-action pairs, while the actor $\mu_\varphi(s)$ maps states directly to deterministic actions. Target networks with parameters θ' and φ' stabilize learning through soft updates with rate τ. Exploration during training uses Gaussian noise $\mathcal{N}(0,\sigma^2)$ with standard deviation σ. Experiences (s,a,r,s') are stored in a replay buffer $\mathcal{B}$ of size N, from which we sample mini-batches of size B for updates.

The first is a Q-network, which does a forward pass on a state-action pair and estimates the expected reward that the agent gets after doing a given action in a given state. To update its weights, we compute the following loss:

$$L_Q(\theta) = \mathbb{E}_{(s,a,r,s')\sim B}$$
$$\left[\Big(Q_\theta(s,a) - \big(r + \gamma Q_{\theta'}\big(s', \mu_{\varphi'}(s')\big)\big)\Big)^2\right].$$

The second is a policy network (Actor), which outputs deterministic actions for any given state. Unlike SAC's stochastic policy, it directly maps states to optimal actions without probability distributions. Using a tanh activation to bound actions within $[-1, 1]$, the policy is optimized by maximizing the expected Q-value:

$$L_\mu(\varphi) = -\mathbb{E}_{s\sim B}\Big[Q_\theta\big(s, \mu_\varphi(s)\big)\Big].$$

Quick description of SAC. SAC implements an actor-critic method with entropy regularization for improved exploration. It uses five neural networks: twin critics with parameters θ_1 and θ_2 to reduce overestimation bias, a value network with parameters ψ, a target value network with parameters ψ', and a policy network with parameters φ. The temperature parameter α controls exploration, with target entropy set to $-|\mathcal{A}|$ where $\mathcal{A}$ is the action space. Experience tuples (s,a,r,s') are stored in a replay buffer $\mathcal{B}$ of capacity N, from which mini-batches of size B are sampled. Networks are updated using learning rates l_Q, l_V, l_π, and l_α for critics, value network, policy, and temperature respectively, with target networks soft-updated at rate τ.

The first is a V-network, which does a forward pass on a given state and estimates the expected reward an agent can gain in the state. To update its weights, we compute the following loss:

$$L_V(\psi) = \mathbb{E}_{(s,a,r,s')\sim B}$$
$$\left[\Big(V_\psi(s) - \big(\min Q_{\theta_i}(s,a_s) - \alpha\log\big(\pi_\varphi(a_s \mid s)\big)\big)\Big)^2\right],$$

where $a_s \sim \pi_\varphi(.|s)$ is an action sampled from the policy.

The second is a Q-network, which does a forward pass on a state-action pair and estimates the expected reward that the agent gets after doing a given action in a given state. To update its weights, we compute the following loss:

$$L_Q(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim B}$$
$$\left[\Big(Q_{\theta_i}(s,a) - r - \gamma V_{\psi^{\text{target}}}(s')\Big)^2\right].$$

The last is a policy network, which parameterizes a Gaussian distribution over actions. It outputs mean actions and learns log standard deviations as parameters. Using the reparameterization trick and tanh squashing for bounded actions, the policy is optimized by minimizing:

$$L_\pi(\phi) = \mathbb{E}\Big[\alpha \log\big(\pi_\phi(a_s \mid s)\big) - Q_{\theta_i}(s, a_s)\Big].$$

It is worth noting that we implemented the August 2018 version of SAC, which includes the value function network. More recent implementations often omit this network and directly use the Q-networks, resulting in a simplified architecture while maintaining similar performance characteristics.

Quick description of IQL. Implicit Q-Learning is an offline reinforcement learning algorithm that decouples value learning from policy improvement. It uses five neural networks: twin critics with parameters θ_1 and θ_2, their target versions with parameters $\theta_{1'}$ and $\theta_{2'}$, a value network with parameters ψ, and a policy network with parameters φ. The expectile regression parameter $\tau' \in [0, 1]$ controls the conservatism in value estimation, while the temperature parameter $\beta > 0$ determines how aggressively to exploit advantages. Experience tuples (s, a, r, s') are stored in a replay buffer $\mathcal{B}$ of capacity N, from which mini-batches of size B are sampled. Networks are updated using learning rates l_Q, l_V, l_π, and l_α for critics, value network, policy, and temperature respectively, with target networks soft-updated at rate τ.

The first is a value network, which estimates the expected return from a state without considering actions. To update its weights, we compute the asymmetric L_2 loss:

$$L_V(\psi) = \mathbb{E}_{(s,a)\sim\mathcal{B}}\Big[L_2^{\tau'}\big(Q_\theta(s, a) - V_\psi(s)\big)\Big],$$

where $L_2^{\tau'}(u) = |\tau' - \mathbb{1}(u < 0)|\ .u^2$ is the asymmetric L_2 loss with parameter τ'.

The second is a Q-network, which estimates expected returns for state-action pairs. To update its weights, we compute the standard TD loss:

$$L_Q(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{B}}\Big[\big(Q_{\theta_i}(s, a) - r - \gamma V_\psi(s')\big)^2\Big].$$

The last is a policy network that parameterizes a Gaussian distribution over actions. It outputs mean actions and learns log standard deviations as parameters. Using the reparameterization trick and tanh squashing for bounded actions, the policy is optimized by maximizing likelihood weighted by advantages:

$$L_\pi(\varphi) = \mathbb{E}_{(s,a)\sim\mathcal{B}}$$
$$\Big[\exp\big(\beta.\min\big(0, Q_{\theta'}(s, a) - V_\psi(s)\big)\big).\log \pi_\varphi(a|s)\Big],$$

where β controls the temperature and we clip advantages to be non-positive to avoid overoptimism.

Quick description of CAL-QL. Calibrated Q-learning extends offline RL with a conservative regularization approach. It uses three neural networks: a critic (Q-network) with parameters ψ, its target version with parameters ψ', and a policy network with parameters φ. The temperature parameter $\alpha > 0$ controls the conservative regularization strength, while the sampling parameter n determines how many random and policy-sampled actions are used for regularization. The algorithm incorporates Monte Carlo returns from a dataset as reference values $V_\mu(s)$ to mitigate overestimation. Experience tuples (s, a, r, s') are stored in a replay buffer $\mathcal{B}$ of capacity N, from which mini-batches of size B are sampled. Networks are updated using learning rates l_Q, l_pi, and l_α for critic, policy, and temperature respectively, with target networks soft-updated at rate τ.

The first is a Q-network loss, which combines a standard TD loss with a conservative regularization term:

$$L_Q(\psi) = L_Q^{\text{TD}}(\psi) + \alpha.L_Q^{\text{CON}}(\psi).$$

The TD loss follows the standard form:

$$L_Q^{\text{TD}}(\psi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{B}}$$
$$\Big[\big(Q_\psi(s, a) - r - \gamma Q_{\psi'}(s', a_{s'})\big)^2\Big].$$

The conservative regularization term penalizes Q-value overestimation:

$$L_Q^{\text{CON}}(\psi) = \mathbb{E}_{s\sim\mathcal{B}} \Bigg[\log\Bigg(\sum_{i=1}^{n} \exp\big(Q_\psi(s, a_R^i) - \log\big(0.5^{|\mathcal{A}|}\big)\big) + \sum_{i=1}^{n} \exp\Big(\max\Big(V_\mu(s), Q_\psi(s, a_s^i) - \log \pi_\varphi(a_s^i|s)\Big)\Big)\Bigg) - Q_\psi(s, a)\Bigg],$$

where a_R^i are random actions sampled uniformly and $V_\mu(s)$ are reference values from Monte Carlo returns.

The policy network is optimized through the standard SAC policy loss:

$$L_\pi(\varphi) = \mathbb{E}_{s\sim\mathcal{B}}\Big[\alpha \log\big(\pi_\varphi(a_s \mid s)\big) - Q_\psi(s, a_s)\Big].$$

Appendix D

Off-policy learning experimental results.

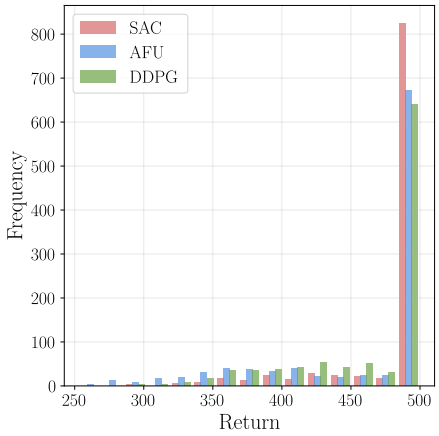

Fig. 11. – Distribution of returns for SAC, AFU, and DDPG algorithms in the CartPole environment under off-policy learning with uniformly sampled state-action pairs. The histogram shows the frequency of return values achieved during the final 1% of training episodes across 5 independent trials, representing the converged performance of each algorithm.

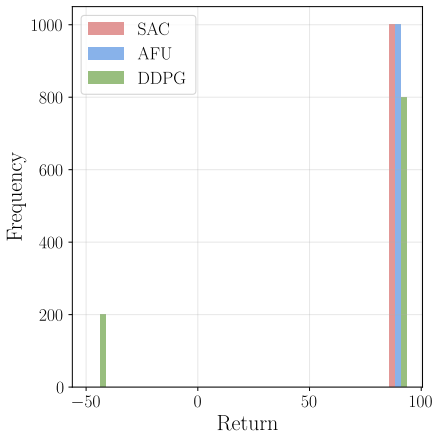

Fig. 12. – Distribution of returns for SAC, AFU, and DDPG algorithms in the Mountain Car environment under off-policy learning with uniformly sampled state-action pairs. The histogram shows the frequency of return values achieved during the final 1% of training episodes across 5 independent trials, representing the converged performance of each algorithm.

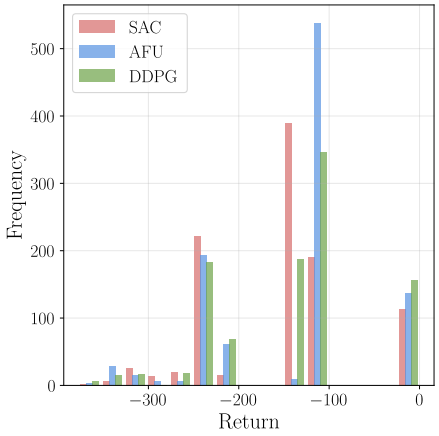

Fig. 13. – Distribution of returns for SAC, AFU, and DDPG algorithms in the Pendulum environment under off-policy learning with uniformly sampled state-action pairs. The histogram shows the frequency of return values achieved during the final 1% of training episodes across 5 independent trials, representing the converged performance of each algorithm.

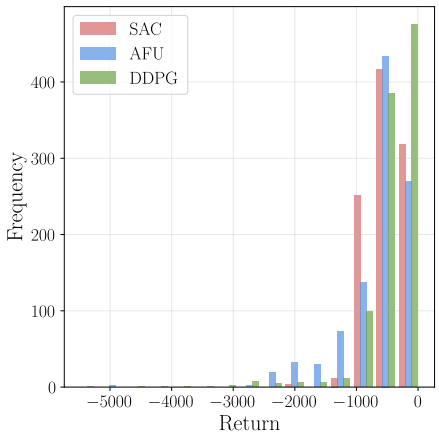

Fig. 14. – Distribution of returns for SAC, AFU, and DDPG algorithms in the Lunar Lander environment under off-policy learning with uniformly sampled state-action pairs. The histogram shows the frequency of return values achieved during the final 1% of training episodes across 5 independent trials, representing the converged performance of each algorithm.

On-policy learning experimental results.

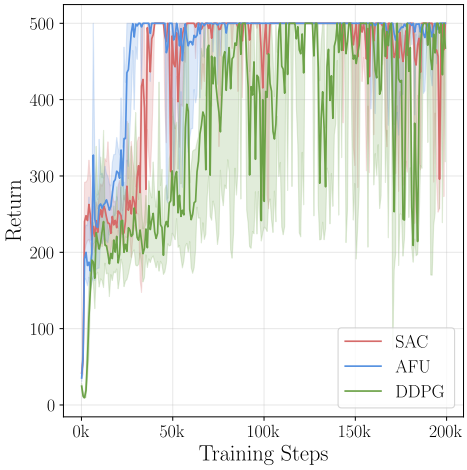

Fig. 15. – Evolution of training performance for SAC, AFU, and DDPG algorithms in the CartPole environment under on-policy learning with policy-generated samples. The curves represent the interquartile means of returns over 200,000 training steps, with shaded regions showing interquartile ranges across 5 independent trials. Performance evaluations were conducted every 100 training steps by deploying the current policy for 10 complete episodes.

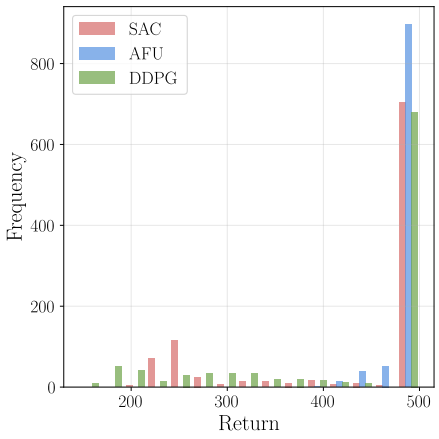

Fig. 16. – Distribution of returns for SAC, AFU, and DDPG algorithms in the CartPole environment under on-policy learning with policy-generated samples. The histogram shows the frequency of return values achieved during the final 1% of training episodes across 5 independent trials, representing the converged performance of each algorithm.

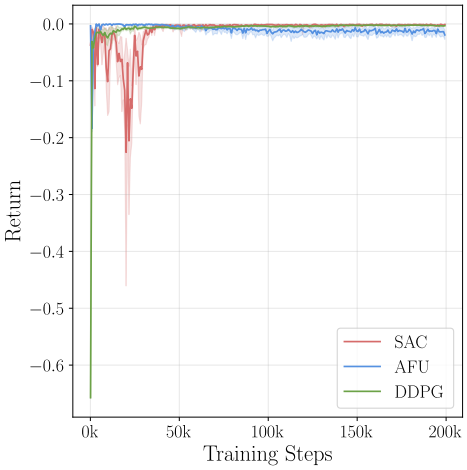

Fig. 17. – Evolution of training performance for SAC, AFU, and DDPG algorithms in the Mountain Car environment under on-policy learning with policy-generated samples. The curves represent the interquartile means of returns over 200,000 training steps, with shaded regions showing interquartile ranges across 5 independent trials. Performance evaluations were conducted every 100 training steps by deploying the current policy for 10 complete episodes.

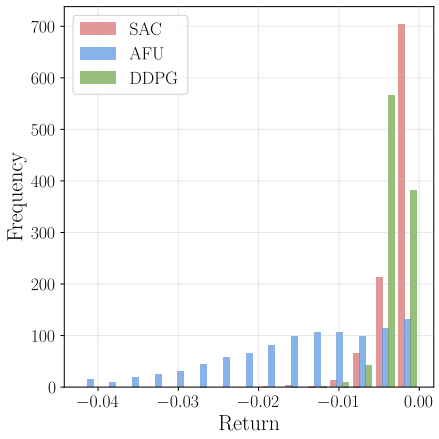

Fig. 18. – Distribution of returns for SAC, AFU, and DDPG algorithms in the Mountain Car environment under on-policy learning with policy-generated samples. The histogram shows the frequency of return values achieved during the final 1% of training episodes across 5 independent trials, representing the converged performance of each algorithm.

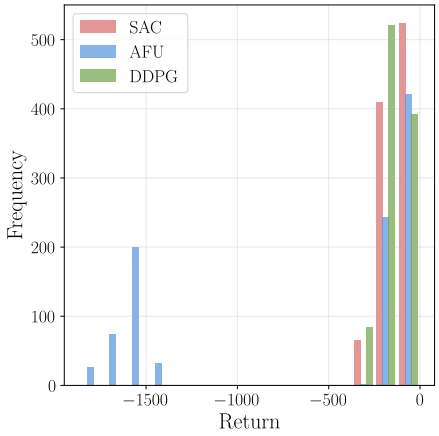

Fig. 19. – Distribution of returns for SAC, AFU, and DDPG algorithms in the Pendulum environment under on-policy learning with policy-generated samples. The histogram shows the frequency of return values achieved during the final 1% of training episodes across 5 independent trials, representing the converged performance of each algorithm.

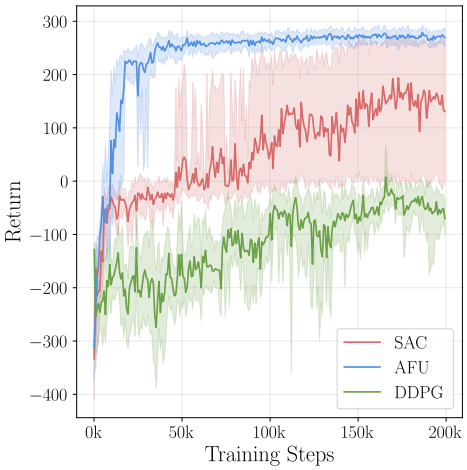

Fig. 20. – Evolution of training performance for SAC, AFU, and DDPG algorithms in the Lunar Lander environment under on-policy learning with policy-generated samples. The curves represent the interquartile means of returns over 200,000 training steps, with shaded regions showing interquartile ranges across 5 independent trials. Performance evaluations were conducted every 100 training steps by deploying the current policy for 10 complete episodes.

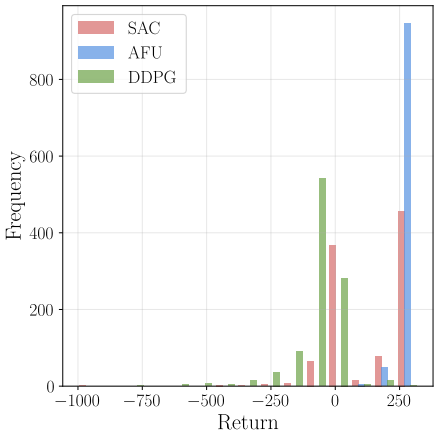

Fig. 21. – Distribution of returns for SAC, AFU, and DDPG algorithms in the Lunar Lander environment under on-policy learning with policy-generated samples. The histogram shows the frequency of return values achieved during the final 1% of training episodes across 5 independent trials, representing the converged performance of each algorithm.

Offline online transition learning experimental results.

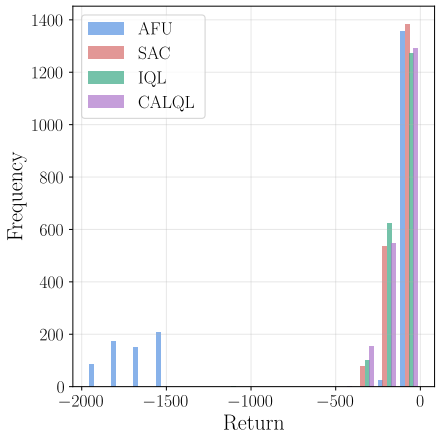

Fig. 22. – Distribution of returns for AFU, SAC, IQL, and CAL-QL algorithms in the Pendulum environment after completing the offline-to-online transition learning process. The histogram shows the frequency of return values achieved during the final 1% of training episodes across 5 independent trials, representing the converged performance of each algorithm after online fine-tuning.

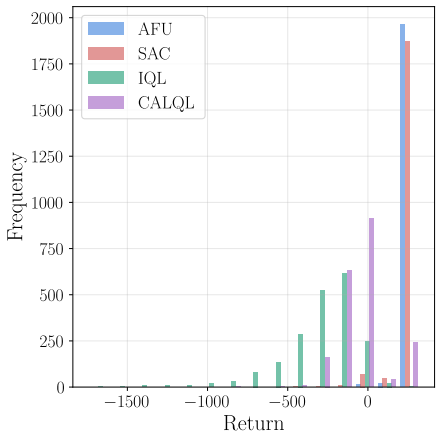

Fig. 23. – Distribution of returns for AFU, SAC, IQL, and CAL-QL algorithms in the Lunar Lander environment after completing the offline-to-online transition learning process. The histogram shows the frequency of return values achieved during the final 1% of training episodes across 5 independent trials, representing the converged performance of each algorithm after online fine-tuning.

Visualization of pendulum learned value function and action policy.

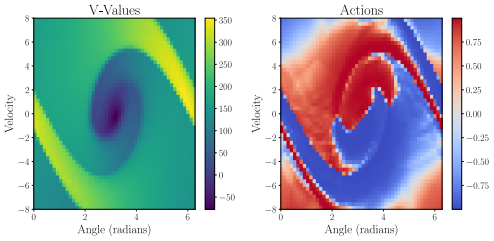

Fig. 24. – Visualization of SAC algorithm's learned value function (left) and action policy (right) in the Pendulum environment under off-policy learning.

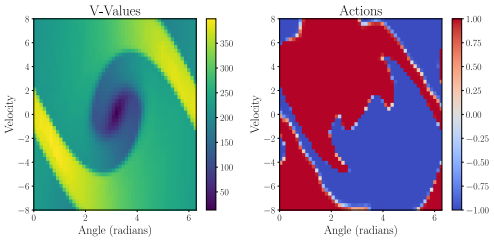

Fig. 25. – Visualization of DDPG algorithm's learned value function (left) and action policy (right) in the Pendulum environment under off-policy learning.

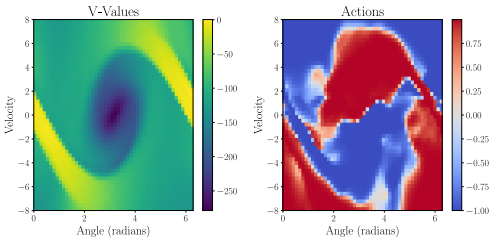

Fig. 26. – Visualization of SAC algorithm's learned value function (left) and action policy (right) in the Pendulum environment under on-policy learning.

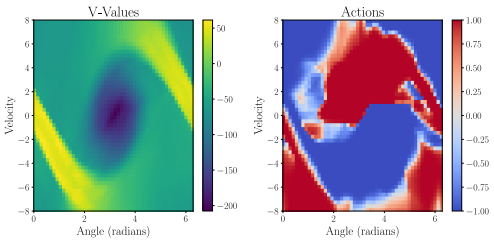

Fig. 27. – Visualization of DDPG algorithm's learned value function (left) and action policy (right) in the Pendulum environment under on-policy learning.

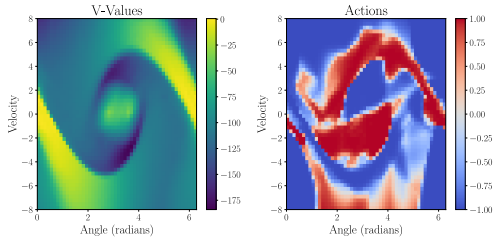

Fig. 28. – Visualization of AFU algorithm's learned value function (left) and action policy (right) in the Pendulum environment after offline-to-online transition.

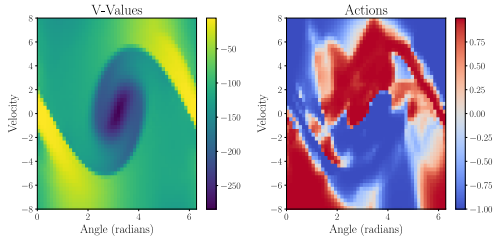

Fig. 29. – Visualization of SAC algorithm's learned value function (left) and action policy (right) in the Pendulum environment after offline-to-online transition.

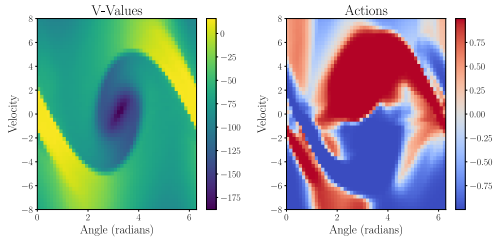

Fig. 30. – Visualization of IQL algorithm's learned value function (left) and action policy (right) in the Pendulum environment after offline-to-online transition.

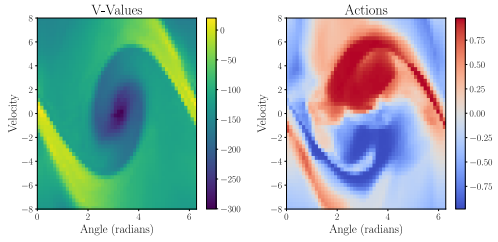

Fig. 31. – Visualization of CAL-QL algorithm's learned value function (left) and action policy (right) in the Pendulum environment after offline-to-online transition.

Bibliographie

[1] Ilya Kostrikov, Ashvin Nair, et Sergey Levine. 2021. Offline Reinforcement Learning with Implicit Q-Learning. Consulté à l'adresse https://arxiv.org/abs/2110.06169

[2] Mitsuhiko Nakamoto, Yuexiang Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, et Sergey Levine. 2024. Cal-QL: Calibrated Offline RL Pre-Training for Efficient Online Fine-Tuning. Consulté à l'adresse https://arxiv.org/abs/2303.05479

[3] Nicolas Perrin-Gilbert. 2024. AFU: Actor-Free critic Updates in off-policy RL for continuous control. Consulté à l'adresse https://arxiv.org/abs/2404.16159

[4] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, et Daan Wierstra. 2019. Continuous control with deep reinforcement learning. Consulté à l'adresse https://arxiv.org/abs/1509.02971

[5] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, et Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. Consulté à l'adresse https://arxiv.org/abs/1801.01290