Actor Free critic Update for Off-policy and Offline learning

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

Introduction

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, CALQL [2] applies a constraint to the conservative Q-learning 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 arbitra-

ry data distributions without requiring explicit constraints during the critic learning phase.

Preleminaries

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 approximators 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:

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

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

Methods

Environment. 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).

To facilitate our research on off-policy learning properties, we extended these environments with additional functionality. Each envi-

ronment was modified to support direct state manipulation through the implementation of a `_set_state()` method. This modification enables precise 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.

Algorithms. 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.

(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 target networks for both actor and critic, which are 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 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 the advantage function $A(s, a)$, which represents how much better taking action a in state s is compared to the average action.

The key innovation in AFU lies in how it computes the value and advantage functions. 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.

(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. CALQL employs a Q-network, a target network, and a policy network similar to SAC's stochastic policy. What distinguishes CALQL 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. CALQL's balanced approach allows it to maintain sufficient conservatism during offline learning while facilitating effective exploration when transitioning to online learning.

(Off-Policy Learning and the Role of the Critic)

In reinforcement learning (RL), the term **off-policy** refers to the capacity of an algorithm to learn an optimal policy independently of the data-generating policy (i.e., the **behavior policy**). This property is crucial for sample efficiency and enables the reuse of past experiences, a foundation of modern deep RL through experience replay buffers.

Q-values, denoted $\hat{}\{Q\}\text{pi}(s, a)$, estimate the expected return of taking action a in state s and then following policy pi thereafter. They are updated iteratively based on observed transitions (s, a, r, s'), following temporal difference learning principles. Standard update rules take the general form:

hat{Q}π(s_t, a_t) leftarrow hat{Q}π(s_t, a_t) + alpha left[r(s, a) + F^π(s_{t+1}) - hat{Q}π(s_t, a_t)right],

where $\hat{}\{F\}\text{pi}\left(s_{\{t+1\}}\right)$ is a bootstrapped estimate of future value. In fully off-policy algorithms like Q-learning, this term is defined as:

F^π(s*{t+1}) = max_a hat*{Q}π(s{t+1}, a),

which explicitly decouples the update from the behavior policy, allowing learning from arbitrary past actions. This contrasts with on-policy methods like SARSA, or pseudo-off-policy actor-critic algorithms like DDPG, TD3, or even SAC, where:

F^π(s*{t+1}) = hat*{Q}π(s{t+1}, π(s_{t+1})),

or in SAC's case, a softened expectation over the policy distribution plus entropy. While SAC and its variants are labeled off-policy due to their use of replay buffers and arbitrary data, the above form of update demonstrates a tight coupling between policy improvement and critic estimation—more akin to SARSA than true Q-learning. This makes them vulnerable to local optima due to circular dependencies: the policy needs a good critic to improve, and the critic needs an improving policy to estimate values accurately.

(AFU: True Off-Policy Critic Improvement)

AFU (Action Fitting Update) explicitly breaks this cycle. It seeks to perform Q-value maximi-

zation over the action space **without** relying on policy-driven gradients. Rather than relying on a policy pi to select the next action in s', AFU aims to directly optimize $Q^pi(s', a)$ via a standalone maximization mechanism. This design allows the critic to improve independently of the actor, drawing from the true spirit of Q-learning.

This fundamental difference enables AFU to maintain effective learning from purely random policies, as shown in our experiments. In simple environments such as Cartpole or MountainCar, AFU achieves competitive or superior performance without requiring improvements from the actor policy. That is, the Q-function can be improved significantly just through off-policy updates on replayed transitions—even when the sampled actions come from a random or poorly performing behavior policy.

This capability highlights a key insight of our research: in simple environments, the critic alone—via direct maximization of Q-values—can drive convergence without needing actor updates. This sheds light on the power of decoupled critic learning and calls into question the need for tightly intertwined actor-critic loops in low-complexity settings.

In more complex environments, however, exploration and policy improvement become critical. Sparse rewards, state-dependent transition dynamics, and multimodal reward distributions (as seen in Pendulum and Lunar Lander) challenge purely off-policy critics due to poor data coverage in important regions of the state-action space. Nevertheless, the fully off-policy design of AFU provides a principled and robust baseline from which more advanced exploration strategies can be layered.

(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. The hypothesis stems from the theoretical properties of AFU's critic update mechanism, which, unlike SARSA-based algorithms, does not rely on the actor's improvement to approximate the max-Q operation.

Traditional actor-critic algorithms like SAC and DDPG are structurally closer to SARSA than to Q-learning. Despite being categorized as off-policy, they depend on the actor to generate actions for the critic's updates, creating an implicit coupling. In contrast, AFU's value and advantage decomposition, combined with conditional gradient scaling, allows it to compute critic updates independently of the actor, potentially enabling true off-policy learning.

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. If AFU can converge under these conditions while traditional algorithms struggle, it would validate its theoretical advantage in truly off-policy learning.

First Results.

We conducted experiments to evaluate the performance of AFU in an off-policy settings. Our results are showned in Appendix D.

(Cartpole)

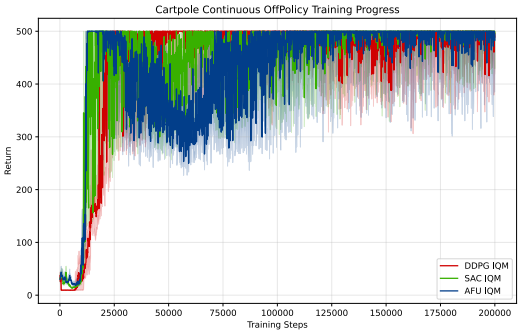

Fig. 1. – Performance of DDPG, SAC, and AFU in the Cartpole environment under off-policy training conditions. All algorithms converge to a similar performance level, but AFU looses performance after 50k steps before converging again.

Fig. 1 shows the training progress of DDPG, SAC and AFU in the Cartpole environment. The histogram of the last rewards of DDPG, SAC and AFU in the Cartpole environment are shown in Fig. 5, Fig. 6 and Fig. 7 respectively. As expected, all three algorithms (DDPG, SAC, AFU) converge in this low-dimensional, low-variance problem Fig. 1. However, an interesting phenomenon occurs around 50k steps, where both AFU and, to a lesser extent SAC, exhibit a sudden drop in performance. One possible explanation is that after converging to a nearly optimal policy, both methods begin incorporating spurious transitions from the replay buffer generated by the random behavior policy. These out-of-distribution samples could destabilize the learned Q-values, disrupting the optimal policy.

(MountainCar)

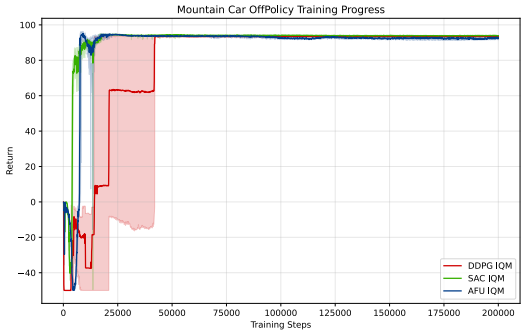

Fig. 2. – Performance of DDPG, SAC, and AFU in the MountainCar environment under off-policy training conditions. All algorithms converge to a similar performance level, DDPG is a little slower than SAC and AFU.

All algorithms eventually learn to solve the task, but the reward distributions diverge widely Fig. 2. AFU exhibits a clear bimodal distribution Fig. 10, it means that it follows two different policies. SAC shows behavior consistent with a normal-like distribution centered around suboptimal returns Fig. 9. DDPG exhibits extreme variance Fig. 8, indicating a « hit-or-miss » policy: it either successfully reaches the goal or fails completely, likely due to deterministic policy collapses conditioned by specific initial states.

(Pendulum)

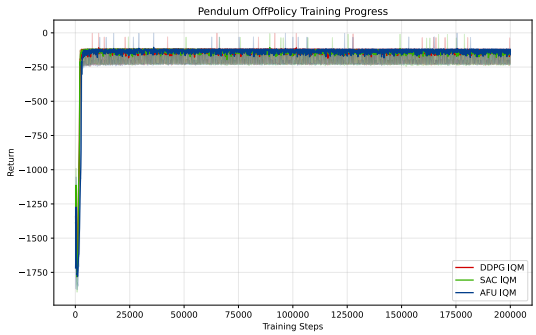

Fig. 3. – Performance of DDPG, SAC, and AFU in the Pendulum environment under off-policy training conditions. The three algorithms converge to a similar suboptimal performance level.

All algorithms converge toward moderately suboptimal policies Fig. 3, but with a distinctive

trimodal distribution of returns shared across algorithms Fig. 11, Fig. 12, Fig. 13. One plausible interpretation is that this trimodality reflects inherent metastable policies in the Pendulum environment: (1) complete failure to stabilize, (2) marginal stabilization with persistent oscillations, and (3) near-optimal stabilization with almost no oscillations. Random policy sampling appears insufficient to facilitate transitions from one mode of behavior to another, trapping learners in local basins of attraction.

(LunarLander Continuous)

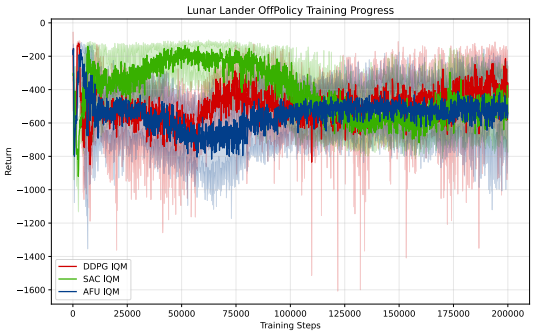

Fig. 4. – Performance of DDPG, SAC, and AFU in the LunarLander environment under off-policy training conditions. None of the algorithms converge to a stable performance level.

None of the considered algorithms exhibit true convergence under random off-policy data Fig. 4. Reward distributions remain highly skewed, with a mean trending toward zero but with extreme variance (most rewards are between −1000 and 0) Fig. 14, Fig. 15, Fig. 16. The agents appear to learn to avoid catastrophic crashes, yet none manage to acquire the precise control strategy required for a successful landing. This failure can be attributed to the extreme sparsity of successful landing experiences in a random dataset—a classic sparse reward challenge.

From these results, we draw several key observations:

– The ability to learn from a purely random policy rapidly decreases with task complexity and reward sparsity.
– The reward distributions themselves reveal the underlying topological structure of policy space unique to each environment and algorithm. For example, the trimodal reward distribution in Pendulum suggests a jagged landscape with distinct metastable solutions.
– Random policies rarely explore « critical » states—such as successful landings in LunarLander or periodic stabilization in Pendulum —which prevents the critic from accurately modeling such transitions.

Taken together, these findings suggest that while AFU's actor-free critics can leverage random data in simpler environments, task complexity (such as multimodal stability regions or sparse reward regimes) requires either more structured exploration or additional constraints to overcome fundamental data sparsity.

(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.

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 CALQL address this through different mechanisms: IQL uses expectile regression to estimate maximum values without querying unseen actions, while CALQL explicitly constrains Q-values for out-of-distribution actions.

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 generated datasets from policies at different stages of training, capturing trajectories at fixed intervals during policy learning. We then trained algorithms offline on these datasets before transitioning to online learning. The primary evaluation metric is the area between the learning curve and the maximum stable performance level following the transition point, which quantifies how quickly and stably an algorithm recovers during the online phase.

(First Results)

We also evaluated AFU in an Offline-to-Online setting, where the agent first learns from a fixed dataset and then transitions to online learning. We used the same datasets as in the previous experiments, but this time we trained AFU for 1 million steps in an offline setting before switching to online training.

Conclusion

Appendix A

Experimental Details.

The custom environment wrappers used in this study include the following methods:

- `_set_state`: Allows resetting the environment to a specific state.
- `get_obs`: Retrieves the current observation.
- `get_observation_space`: Returns the observation space of the environment.
- `get_action_space`: Returns the action space of the environment.

These wrappers were implemented to ensure compatibility with the experiments.

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: 200000
- 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 CALQL 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 CALQL 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

Implementation detail 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:

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

Implementation detail 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:

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

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:

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

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:

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

Implementation detail of AFU. Actor Free critic Updates implements an actor-critic architecture with a novel advantage decomposition approach. 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, $l_p i$, 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:

$$Z(x,y) = \begin{cases}(x+y)^2 & \text{if } x \le 0\\ x^2+y^2 & \text{otherwise}\end{cases}$$
$$\Upsilon_i^a(s) = (1-\rho.I_i(s,a))V_{\varphi_i}(s) + \rho.I_i(s,a).V_{\varphi_i}^{\text{nograd}}(s)$$
$$I_i(s,a) = \begin{cases}1 & \text{if } V_{\varphi_i}(s) + A_{\xi_i}(s,a) < Q_\psi(s,a)\\ 0 & \text{otherwise}\end{cases}$$

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:

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

Implementation detail 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:

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

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:

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

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:

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

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

Implementation detail 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:

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

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. The following figures show the performance of DDPG, SAC and AFU in an off-policy learning setting. The results are averaged over 5 runs with dif-

ferent random seeds. The shaded area represents the standard deviation.

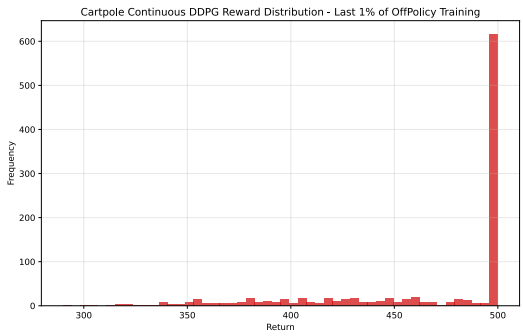

Fig. 5. – DDPG last rewards in the Cartpole environment under off-policy training conditions.

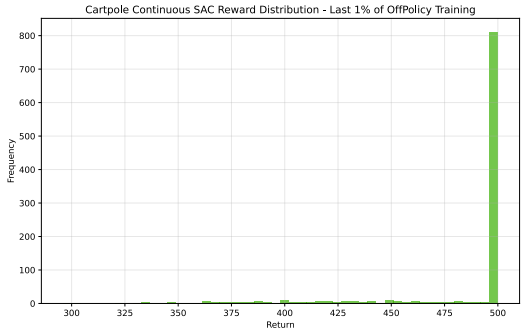

Fig. 6. – SAC last rewards in the Cartpole environment under off-policy training conditions.

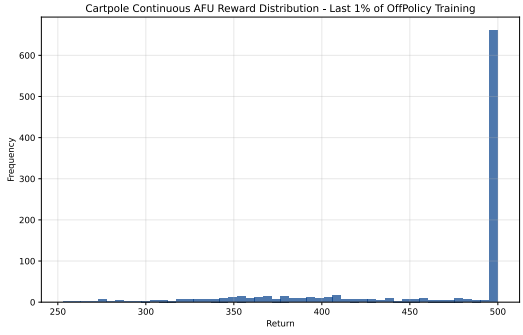

Fig. 7. – AFU last rewards in the Cartpole environment under off-policy training conditions.

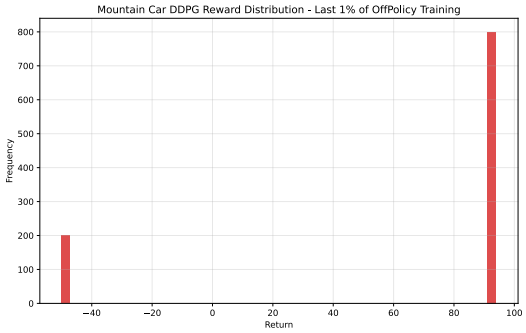

Fig. 8. – DDPG last rewards in the MountainCar environment under off-policy training conditions. There is a very good reward as well as a very bad one, indicating that DDPG is either able to reach the goal or not at all.

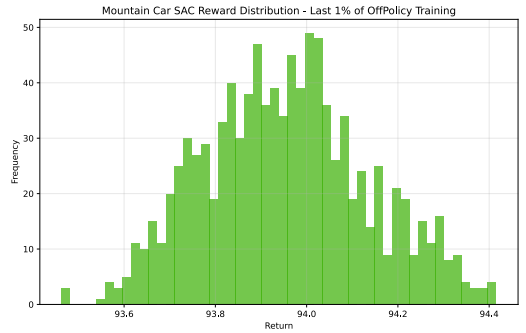

Fig. 9. – SAC last rewards in the MountainCar environment under off-policy training conditions. The distribution is normal-like around 94, indicating that SAC does not reach the optimal goal but is able to get a good reward.

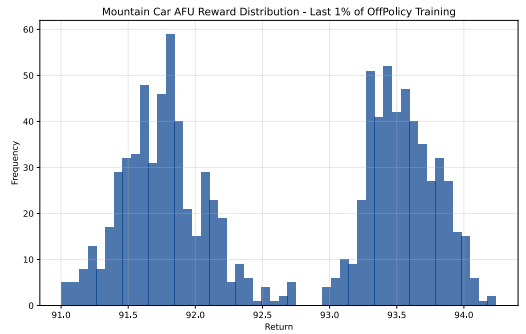

Fig. 10. – AFU last rewards in the MountainCar environment under off-policy training conditions. The distribution is bimodal, indicating that AFU is following two policies.

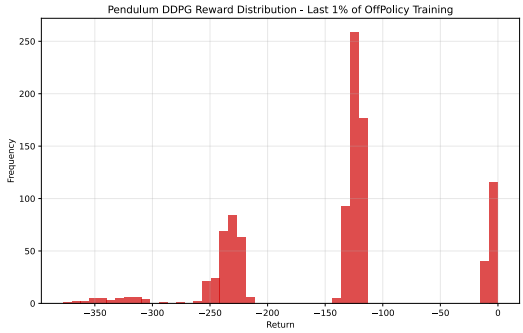

Fig. 11. – DDPG last rewards in the Pendulum environment under off-policy training conditions. The distribution is trimodal, indicating that DDPG is following three policies: a very bad, a medium and a very good one.

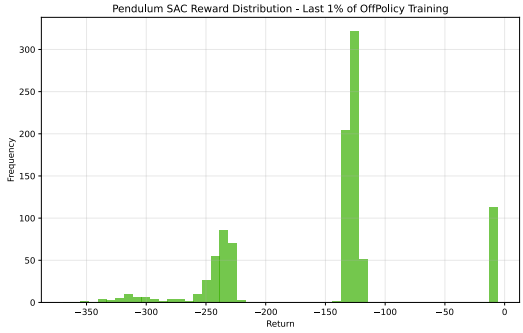

Fig. 12. – SAC last rewards in the Pendulum environment under off-policy training conditions. The distribution is trimodal, indicating that SAC is following three policies: a very bad, a medium and a very good one.

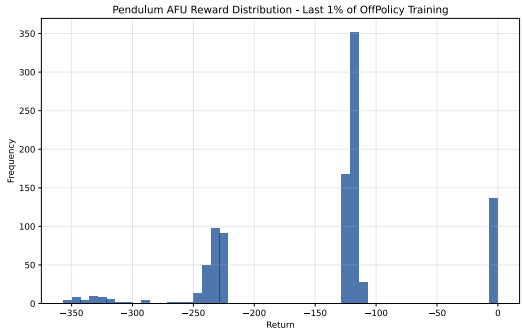

Fig. 13. – AFU last rewards in the Pendulum environment under off-policy training conditions. The distribution is trimodal, indicating that AFU is following three policies: a very bad, a medium and a very good one.

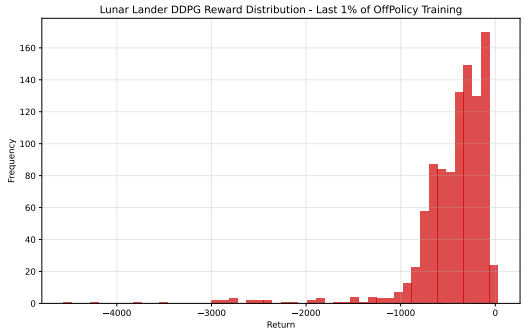

Fig. 14. – DDPG last rewards in the LunarLander environment under off-policy training conditions. The distribution approaches values between −1000 and 0, suggesting that the algorithm doesn't learn a catastrophic policy but isn't able to land the rocket either.

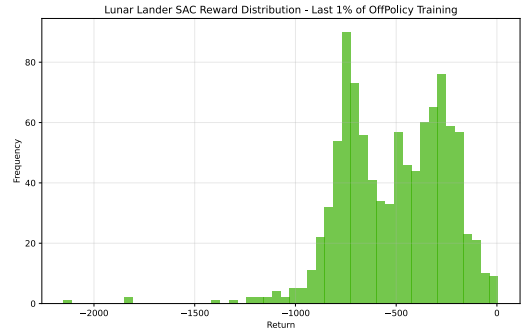

Fig. 15. – SAC last rewards in the LunarLander environment under off-policy training conditions. The distribution approaches values between −1000 and 0, suggesting that the algorithm doesn't learn a catastrophic policy but isn't able to land the rocket either.

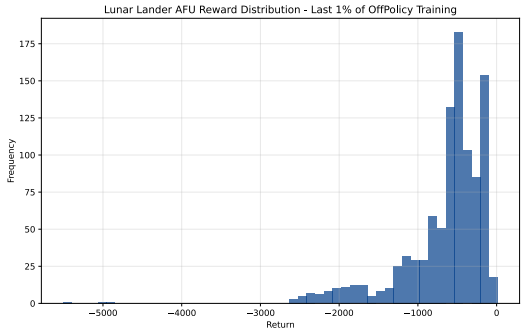

Fig. 16. – AFU last rewards in the LunarLander environment under off-policy training conditions. The distribution approaches values between −1000 and 0, suggesting that the algorithm doesn't learn a catastrophic policy but isn't able to land the rocket either.

Appendix E

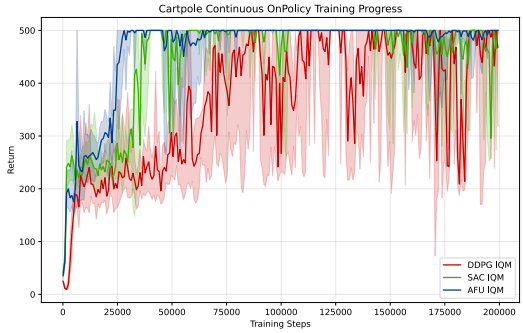

Fig. 17. – Every algorithms converges to a similar performance level in the Cartpole environment under on-policy training conditions with AFU converging faster than SAC and SAC converging faster than DDPG. The good performances are expected since the environment is very simple. We observe a drop in performance at the end of the training, indicating that SAC and DDPG are not able to learn a stable policy.

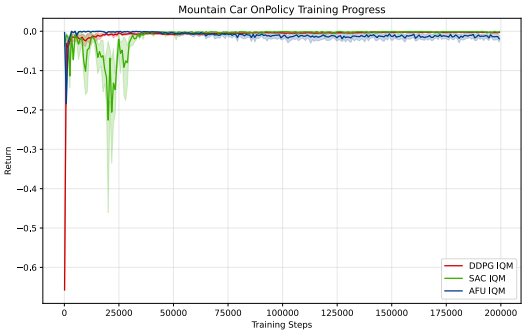

Fig. 18. – Every algorithms converges to a similar performance level in the MountainCar environment under on-policy training conditions but AFU is a bit suboptimal. The good performances are expected since the environment is very simple.

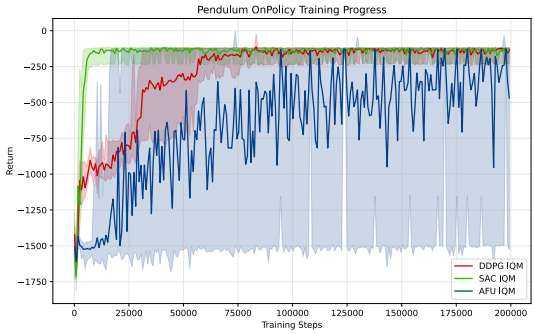

Fig. 19. – SAC is very good in the Pendulum environment under on-policy training conditions while AFU is very inconsistent. This can be explained by the nature of the environment–it is very sensible to small variations so each action is critical–thus the action must be chosen wisely.

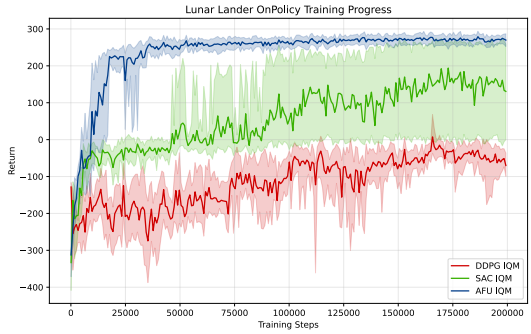

Fig. 20. – AFU shows that it is way better than DDPG and SAC in the Lunar Lander environment under on-policy training conditions. AFU is adapted to complex environments with sparse rewards and is able to learn a stable policy. DDPG and SAC are not able to learn a stable policy. It is interesting to compare the performance of AFU in the Lunar Lander environment with the performance of AFU in Pendulum. Lunar Lander gives more informative rewards (altitude, position, orientation) and need more strategic planning as the margin error is more important and the main goal is to land correctly so with less time constraint.

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