AFU: Actor-Free critic Updates in off-policy RL for continuous control

Nicolas Perrin-Gilbert
Sorbonne Université, CNRS, Institut des Systèmes Intelligents et de Robotique, ISIR,
F-75005 Paris, France
nicolas.perrin-gilbert@cnrs.fr

Abstract

This paper presents AFU, an off-policy deep RL algorithm addressing in a new way the challenging "max-Q problem" in Q-learning for continuous action spaces, with a solution based on regression and conditional gradient scaling. AFU has an actor but its critic updates are entirely independent from it. As a consequence, the actor can be chosen freely. In the initial version, AFU-alpha, we employ the same stochastic actor as in Soft Actor-Critic (SAC), but we then study a simple failure mode of SAC and show how AFU can be modified to make actor updates less likely to become trapped in local optima, resulting in a second version of the algorithm, AFU-beta. Experimental results demonstrate the sample efficiency of both versions of AFU, marking it as the first model-free off-policy algorithm competitive with state-of-the-art actor-critic methods while departing from the actor-critic perspective.

1 Introduction

Q-learning [31] stands as a fundamental algorithm in the realm of model-free RL. As mentioned in [32], "it provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains". It is centered around the Bellman optimality equation and leverages dynamic programming to compute or approximate a function known as the optimal Q-function Q^*. The integration of deep neural networks to approximate Q-functions and the efficient computation of gradient-based updates has led to the successful development of the Deep Q-Network (DQN) algorithm [25], catalyzing important advancements in reinforcement learning. However, Q-learning requires computing the maximum of the Q-function over the action space, which can be difficult if it is continuous and multi-dimensional. To circumvent this "max-Q problem", Q-learning can be combined with an actor-critic perspective, which involves coupling policy gradient with Q-learning updates to estimate the action-value function. Although this type of coupling primarily aims to evaluate the actor with a Q-learning critic, its byproduct is that the actor is trained to generate actions that maximize the Q-function, thereby solving the max-Q problem indirectly. This approach gave rise to DDPG [21], a seminal actor-critic algorithm benefiting from the off-policy nature of Q-learning and consequently from a high sample-efficiency. Yet, in DDPG and its derivatives like TD3 [7], the actor may become trapped in local optima [22]. Other approaches have attempted to face the max-Q problem head-on, like CAQL [28] or Implicit Q-Learning (IQL, [18]), but the former does not scale well to high-dimensional action spaces, and requires adaptations such as constraining the action range for complex problems, whereas in the latter, the expectile loss becomes unbalanced when trying to produce estimates that are very close to the true maxima, which has so far restricted the application of IQL and similar methods to offline RL.

Overall, in continous state and action spaces, the most successful modern off-policy deep reinforcement learning algorithms are actor-critic algorithms that tend to fail in batch settings, when a large part of the training data is uncorrelated to the distribution under the current policy [8]. In this sense,

17th European Workshop on Reinforcement Learning (ewrl 2024).

they are not *truly* off-policy. On the other end, truly off-policy algorithms adapted from Q-learning [31], such as Implicit Q-Learning [18], are adapted to offline RL but do not perform well in online RL. This motivates the quest for a truly off-policy algorithm that is well suited for online RL. In this paper, we make a step in this direction by proposing a novel way to solve the max-Q problem, using regression and conditional gradient scaling (see Section 4), resulting in a new algorithm that adapts Q-learning to continuous action spaces. The algorithm still has an actor to select actions and produce episodes, but unlike state-of-the-art model-free off-policy algorithms, most of which are derived from TD3 or SAC [11], its critic updates are entirely independent from the actor. We call the algorithm AFU for "Actor-Free Updates". In its first version, AFU-alpha (see Sections 5 and 6), we use a stochastic actor and train it like the actor in SAC. We then study in Section 7 a simple failure mode of SAC (and AFU-alpha), and show that the value function trained by regression in AFU can help improve the actor update and make it less prone to local optima, resulting in a new version of the algorithm, AFU-beta (see Section 8), which does not fail in the same way. Our experiments show that AFU-alpha and AFU-beta are competitive in sample-efficiency with TD3 and SAC without being more computationally expensive. To the best of our knowledge, AFU is the first model-free off-policy RL algorithm that is competitive with the state-of-the-art and truly departs from the actor-critic perspective.

2 Related Work

In domains where high sample efficiency is crucial, such as robotics, the off-policy nature of RL algorithms becomes paramount. This allows training on samples obtained from different policies or older versions of the current policy, facilitating faster learning and compatibility with various exploration strategies. One way to obtain an off-policy algorithm is to adapt Q-learning, but as previously mentioned, in continuous action spaces, direct approaches attempting to solve the max-Q problem of Q-learning have faced limitations. Besides CAQL, which formulates the max-Q problem as a mixed-integer program, and IQL, which treats Q-functions as state-dependent random variables and relies on expectile regression to estimate their maxima, we can cite NAF [10] and ICNN [2], which impose action-convex Q-functions making the max-Q problem tractable, QT-Opt [16], which uses a stochastic optimizer to tackle non-convex max-Q problems, or approaches based on a discretization of the action space, such as SMC-learning [20] and SDQN [23]. However, these methods often struggle with complex, high-dimensional continuous control tasks, either due to a lack of expressiveness or prohibitive computational costs. Close to IQL, $\mathcal{X}$-QL [9] is an offline RL algorithm relying on an objective directly estimating the optimal soft-value function in the maximum entropy RL setting without needing to sample from a policy. A variant of $\mathcal{X}$-QL [9] works in the online setting, but in this case critic updates depend on actions sampled by the actor for the Bellman backup. A unique off-policy algorithm, AWR [27], employs regression to train a value function and a policy but falls short of the sample efficiency achieved by state-of-the-art off-policy algorithms. Presently, the most successful approaches in model-free off-policy RL for continuous control are actor-critic algorithms with interwoven actor and critic updates. The first off-policy actor-critic algorithm was introduced in [6], and the most recent ones are typically based on TD3, an improvement of DDPG, or on SAC, which relies on an entropy maximization framework that led to various off-policy algorithms by creating connections between policy gradients and Q-learning updates (see [26]). Among the algorithms improving upon TD3 and SAC, we can mention TQC [19], a distributional approach to control the overestimation bias, REDQ [4] or AQE [33] which employ critic ensembles, DroQ [14] which uses dropout and layer normalization in the critic networks, and BAC [15] which merges Q-function updates from SAC and IQL. While these ideas could be incorporated into our proposed algorithm AFU, we leave this for future work and focus on comparing AFU to SAC and TD3. In contrast to methods building upon SAC and TD3, AFU is structurally distinct because the critic updates remain unaffected by the actor. Notably, the critic is never trained with out-of-distribution (OOD) actions, yet AFU achieves a level of sample efficiency competitive with SAC and TD3. One might object that OOD actions can be beneficial in the online setting, because they favor exploration. As pointed out in [9], OOD actions in Bellman backups introduce over-optimism, but online learning allows agents to correct over-optimism by collecting additional data. Yet, by achieving results comparable in sample-efficiency to TD3 and SAC without OOD actions, we show that the benefit of Bellman backups with OOD actions in the online setting is in fact not so obvious. If OOD actions can introduce an over-optimism that then needs to be corrected, it may be preferable to design online learning methods that do not yield over-optimism in the first

place, and use other strategies to favor exploration. Furthermore, unlike other direct adaptations of Q-learning to continuous control, AFU does not fail on the most complex tasks. On the contrary, the challenging MuJoCo task Humanoid is one of the environments in which AFU performs the best comparatively to SAC and TD3.

3 Preliminaries

We consider a discounted infinite horizon Markov Decision Problem (MDP) $< S, A, T, R, \gamma >$, where S is a state space, A a continuous action space, T a stochastic transition function, $R : S \times A \to \mathbb{R}$ a reward function, and $0 \leq \gamma < 1$ a discount factor. We denote by s' (resp. s_{t+1}) a state obtained after performing an action a (resp. a_t) in state s (resp. s_t). Transitions are tuples (s, a, r, s') with $r = R(s, a)$. The optimal Q-function Q^* is defined by: $Q^*(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a, \pi^*\right]$, where the policy used from $t = 1$ onwards is π^*, which selects actions optimally in every state. The optimal value function V^* verifies $V^*(s) = \max_{a \in A}(Q^*(s, a))$. Let V_{φ_1} and V_{φ_2} denote two function approximators for the value function, and Q_ψ a function approximator for the Q-function (the critic). We use feed forward neural networks for all the function approximators. For the value function, we also consider target networks (see [24]), i.e. parameter vectors $\varphi_1^{\text{target}}$ and $\varphi_2^{\text{target}}$ updated with the rule $\varphi_i^{\text{target}} \leftarrow \tau\varphi_i + (1 - \tau)\varphi_i^{\text{target}}$ for some target smoothing coefficient $0 < \tau < 1$. We wish to train the critic on mini-batches B of transitions (s, a, r, s') taken from an experience replay buffer, with the following loss derived from the clipped Double Q-learning loss of TD3 [7]:

$$L_Q(\psi) = \underset{(s,a,r,s') \in B}{\text{Mean}} \left[\left(Q_\psi(s, a) - r - \gamma \min_{i \in \{1,2\}} V_{\varphi_i^{\text{target}}}(s') \right)^2 \right] \tag{1}$$

The use of two function approximators V_{φ_1} and V_{φ_2} aims at avoiding the overestimation bias that can make Q-learning based approaches diverge (see [13]). In practice, transitions can be terminal, which requires a simple modification of the loss ignored here for the sake of clarity (γ is set to 0 for terminal transitions). Provided that $V_{\varphi_1}(s)$ and $V_{\varphi_2}(s)$ return good estimates of the maximum of $Q_\psi(s, \cdot)$, a maximization usually referred to as the max-Q problem (see [28]), Equation (1) amounts to the mean squared Bellman error that drives Q_ψ toward Q^* [3].

4 A new way to solve the max-Q problem

The main remaining problem is: how to efficiently train V_{φ_1} and V_{φ_2}? For the learning to be successful, V_{φ_1} and V_{φ_2} should both converge to precise solutions of the max-Q problem, and the convergence should be fast, because if changes in Q_ψ are not tracked promptly, errors such as overestimation of Q-values could lead to failures.

4.1 Method

We introduce two new function approximators A_{ξ_1} and A_{ξ_2}, for the optimal advantage function defined by $A^*(s, a) = Q^*(s, a) - V^*(s)$. For any state-action pair (s, a), $A^*(s, a) \leq 0$. Preliminarily, we assume that outputs of A_{ξ_i} can only be non-positive. Assuming that Q_ψ is fixed, training V_{φ_i} and A_{ξ_i} can be done by minimizing the following regression loss on mini-batches B:

$$l_{V,A}(\varphi_i, \xi_i) = \underset{(s,a,\_,\_) \in B}{\text{Mean}} \left[\left(V_{\varphi_i}(s) + A_{\xi_i}(s, a) - Q_\psi(s, a) \right)^2 \right]. \tag{2}$$

This loss causes the values $V_{\varphi_i}(s)$ to become upper bounds of $Q_\psi(s, \cdot)$, but not tight ones. A natural next step would be to add a regularization term penalizing large outputs of V_{φ_i}, which results in an approach very similar to methods based on regression with asymmetric losses such as IQL, SQL, EQL and $\mathcal{X}$-QL [9]. The issue is that the resulting convergence is either slow (for very small regularization coefficients) or significantly biased (for larger coefficients). For some problems, finding the right coefficient is possible, but in general standard regularization does not lead to satisfactory results in the context of online RL. We propose a different approach based on conditional gradient rescaling, noticing that when $V_{\varphi_i}(s) + A_{\xi_i}(s, a)$ is greater than its target $Q_\psi(s, a)$, by gradient descent both $V_{\varphi_i}(s)$ and $A_{\xi_i}(s, a)$ would decrease by the same amount, and conversely when $V_{\varphi_i}(s) + A_{\xi_i}(s, a)$ is smaller than the target, values would both increase by the same amount. Without regularization, all

upper bounds of $Q_\psi(s,\cdot)$ are equally good values for $V_{\varphi_i}(s)$, but we can asymmetrically modulate the gradients to put a "downward pressure" on $V_{\varphi_i}(s)$ and make it progressively decrease as $A_{\xi_i}(s,a)$ progressively increases. To this end, we apply only a fraction of the gradient descent update on φ_i when $V_{\varphi_i}(s)$ would increase. It can be done by defining, for $0 < \varrho < 1$:

$$\Upsilon_i^a(s) = (1 - \varrho I_i^{s,a})\, V_{\varphi_i}(s) + \varrho I_i^{s,a} V_{\varphi_i^{\text{no\_grad}}}(s),$$

where $\varphi_i^{\text{no\_grad}}$ is a copy of the parameters φ_i, and $I_i^{s,a} = \begin{cases} 1, & \text{if } V_{\varphi_i}(s) + A_{\xi_i}(s,a) < Q_\psi(s,a). \\ 0, & \text{otherwise.} \end{cases}$

Replacing $V_{\varphi_i}(s)$ by $\Upsilon_i^a(s)$ in (2) yields a new version of the loss:

$$\Lambda_{V,A}(\varphi_i, \xi_i) = \underset{(s,a,\_,\_)\in B}{\text{Mean}} \left[\left(\Upsilon_i^a(s) + A_{\xi_i}(s,a) - Q_\psi(s,a) \right)^2 \right]. \tag{3}$$

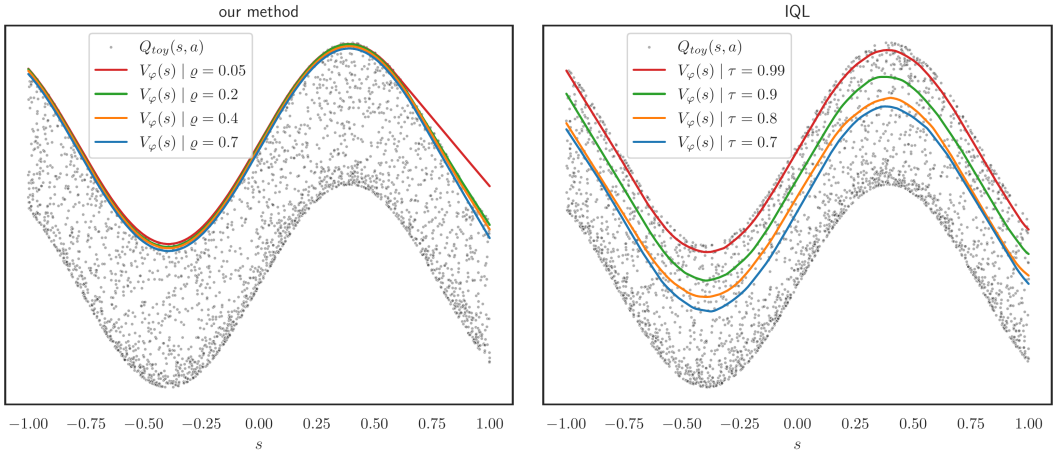
Remark: the proposed method based on conditional gradient rescaling is similar to an adaptive regularization scheme in which the weight of the regularization is proportional to the absolute value of the error, see Appendix B.

So far, we have assumed that $A_{\xi_i}(s,a)$, typically the output of a neural network, can only be non-positive. But imposing a strict constraint on the sign of $A_{\xi_i}(s,a)$ could potentially lead to jittering gradients, so we instead restrict its sign in a soft way (see Appendix C), resulting in this loss:

$$\Lambda'_{V,A}(\varphi_i, \xi_i) = \underset{(s,a,\_,\_)\in B}{\text{Mean}} \left[Z\Big(\Upsilon_i^a(s) - Q_\psi(s,a), A_{\xi_i}(s,a) \Big) \right], \tag{4}$$

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

4.2 Experiments



(a) $V_\varphi(s)$ is trained jointly with $A_\xi(s,a)$ by iterating gradient descent steps on the loss $\Lambda'_{V,A}(\varphi,\xi)$ described by Equation (4). $\varrho \in [0.2, 0.7]$ results in precise approximations of $s \mapsto \max_{a\in A}(Q_\psi(s,a))$.

(b) Results of the training with the loss from IQL [18] for 4 different values of the hyperparameter τ. Values used in actual (offline) RL experiments are not greater than 0.9.

Figure 1: $Q_{toy}(s,a) = \sin(4s) + 0.7\cos(4a)$ for $(s,a) \in [-1,1]^2$. Our method and IQL both train $V_\varphi(s)$ to approximate $s \mapsto \max_{a\in A}(Q_{toy}(s,a))$, i.e. solve a max-Q problem. Trainings are done with 3000 gradient descent steps on batches of 256 uniformly randomly drawn values of (s,a).

We empirically compare our method to 3 baselines on a toy problem. We define the function $Q_{toy}(s,a) = \sin(4s) + 0.7\cos(4a)$ for $s \in [-1,1]$ and $a \in [-1,1]$. We use a single feedforward neural network for V (V_φ) and a single feedforward neural network for A (A_ξ). Both networks have two hidden layers of size 256 and ReLU activations in the hidden layers. Our method trains both V_φ and A_ξ, while the 3 baselines IQL, SQL and EQL directly train V_φ. All 3 baselines have been successfully applied to offline reinforcement learning. IQL is Implicit Q-Learning [18], and SQL and

EQL are respectively Sparse Q-Learning and Exponential Q-Learning, both introduced in [34]. For a fixed s, IQL treats $Q_{toy}(s,a)$ as a random variable (the randomness being determined by the action) and uses an expectile regression loss to train $V_\varphi(s)$ to estimate a state conditional upper expectile of this random variable. The expectile is determined by the parameter $0 < \tau < 1$, and the closer τ is to 1, the closer $V_\varphi(s)$ gets to $\max_{a \in A}(Q_{toy}(s,a))$. However, if τ is very close to 1 (e.g. $\tau = 0.99$), the loss becomes unbalanced, with elements weighted hundreds of times more than others, which results in instabilities in the context of RL, so in practice the values used in [18] are not greater than 0.9. Figure 1 compares our method to IQL. We observe that, with our method, although $\varrho = 0.05$ leads to overestimations, a wide range of parameter values (from $\varrho = 0.2$ to $\varrho = 0.7$) yield precise results, while with IQL a precise result is only obtained with a hyperparameter value inapplicable to RL ($\tau = 0.99$). In Appendix D, the more complete Figure 6 shows a comparison to IQL, SQL and EQL. The same observation can be made for all 3 baselines: results with hyperparameters that are suitable to RL are significantly less precise than the ones obtained with our method.

5 Actor-free critic updates and actor training

First, we remove the dependency to Q_ψ in loss (4) by replacing $Q_\psi(s,a)$ by the targets used to train it in (1). We obtain the following loss (for $i \in \{1,2\}$):

$$L_{V,A}(\varphi_i, \xi_i) = \underset{(s,a,r,s') \in B}{\text{Mean}} \left[Z\Big(\Upsilon_i^a(s) - r - \gamma \min_{i \in \{1,2\}} V_{\varphi_i^{\text{target}}}(s'), A_{\xi_i}(s,a) \Big) \right]. \tag{5}$$

With the losses $L_Q(\psi)$ (1) and $L_{V,A}(\varphi_i, \xi_i)$ (5), we can train Q_ψ, V_{φ_i} and A_{ξ_i} without needing an actor. Compared to methods derived from DDPG (like TD3), solving directly the max-Q problem has an advantage over first using an actor to solve the argmax-Q problem, i.e. to approximate $\text{argmax}_{a \in A}(Q_\psi(s,a))$. The reason is that continuous changes in Q_ψ result in continuous changes of its state conditioned maxima, while it can result in discontinuous changes of its state conditioned argmax. So, in an off-policy setting, if the exploration policy discovers better results with very different actions, the maximum of $Q_\psi(s,a)$ can be tracked smoothly, while the tracking of the argmax can be much more difficult, with the potential arising of deceptive value landscapes in which the actor can get stuck (see [22]). This theoretical advantage, as well as the actor-free Q_ψ, V_{φ_i} and A_{ξ_i} updates are all important aspects of our approach. However, since we are interested in online reinforcement learning, we still need an actor to select actions and produce episodes. To train this actor, if we would use the same gradient ascent over $a \mapsto Q_\psi(s,a)$ as in DDPG, our global method would be prone to the same failure modes as DDPG, and most of the advantages of the max-Q based training of V_{φ_i} would be lost. One thing we can notice is that, since we do not need the actor to return $\text{argmax}_{a \in A}(Q_\psi(s,a))$, we also do not need the actor to be deterministic. To benefit from a better exploration, we opt for a stochastic actor and follow the approach proposed in SAC [11] with automatic tuning of the temperature parameter α. It relies on two losses $L_\pi(\theta)$ and $L_{\text{temp}}(\alpha)$, and on a target entropy $\bar{\mathcal{H}}$ (see Appendix E):

$$L_\pi(\theta) = \underset{\substack{(s,\_,\_,\_) \in B \\ a_s \sim \pi_\theta(\cdot|s)}}{\text{Mean}} \Big[\alpha \log(\pi_\theta(a_s|s)) - Q_\psi(s,a_s) \Big]. \tag{6}$$

$$L_{\text{temp}}(\alpha) = \underset{\substack{(s,\_,\_,\_) \in B \\ a_s \sim \pi_\theta(\cdot|s)}}{\text{Mean}} \Big[-\alpha \log(\pi_\theta(a_s|s)) - \alpha \bar{\mathcal{H}} \Big]. \tag{7}$$

6 AFU-alpha

We combine the losses L_Q (1), $L_{V,A}$ (5), L_π (6), and L_{temp} (7) to devise a new off-policy reinforcement learning algorithm. It has a critic (Q_ψ) and an actor (π_θ), but the critic updates are derived from our novel adaptation of Q-learning to continuous action spaces (obtained with our new method to solve the max-Q problem), therefore they are independent from the actor. Hence the name AFU for the algorithm, for "Actor-Free Updates". We specifically call it AFU-alpha to contrast it with AFU-beta introduced in Section 8. AFU-alpha, described in Algorithm 1, alternates between environments steps that gather experience in a replay buffer and gradient steps that draw batches from the replay buffer to compute loss gradients and update all parameters of the function approximators. In our implementation, an iteration consists of a single environment step followed by a single gradient step.

Algorithm 1 AFU-alpha and AFU-beta

Set $0 < \varrho < 1$, $0 < \tau < 1$, $\bar{\mathcal{H}}$, and learning rates η_Q, $\eta_{V,A}$, η_π, η_{temp}.
Initialize empty replay buffer $\mathfrak{R}_b$, and params ψ, $\varphi_1 = \varphi_1^{\text{target}}$, $\varphi_2 = \varphi_2^{\text{target}}$, ξ_1, ξ_2, α, θ.
for each iteration **do**
  for each environment step **do**
    Sample action $a \sim \pi_\theta(\cdot|s)$.
    Perform environment step $s, a \to s'$, compute $r = R(s,a)$, and insert (s,a,r,s') in $\mathfrak{R}_b$.
  end for
  for each gradient step **do**
    Draw batch of transitions B from $\mathfrak{R}_b$ and compute loss gradients on that batch.
    $\psi \leftarrow \psi - \eta_Q \nabla_\psi L_Q(\psi)$
    $\varphi_{i\in\{1,2\}} \leftarrow \varphi_i - \eta_{V,A} \nabla_{\varphi_i} L_{V,A}(\varphi_i, \xi_i)$
    $\xi_{i\in\{1,2\}} \leftarrow \xi_i - \eta_{V,A} \nabla_{\xi_i} L_{V,A}(\varphi_i, \xi_i)$
    $\varphi^{\text{target}}_{i\in\{1,2\}} \leftarrow \tau\varphi_i + (1-\tau)\varphi_i^{\text{target}}$
    $\zeta \leftarrow \zeta - \eta_\pi \nabla_\zeta L_\mu(\zeta)$
    $\theta \leftarrow \theta - \eta_\pi \nabla_\theta L_\pi(\theta)$
    $\theta \leftarrow \theta - \eta_\pi \nabla_\theta^{\text{MODIF}} L_\pi(\theta)$
    $\alpha \leftarrow \alpha - \eta_{\text{temp}} \nabla_\alpha L_{\text{temp}}(\alpha)$
  end for
end for

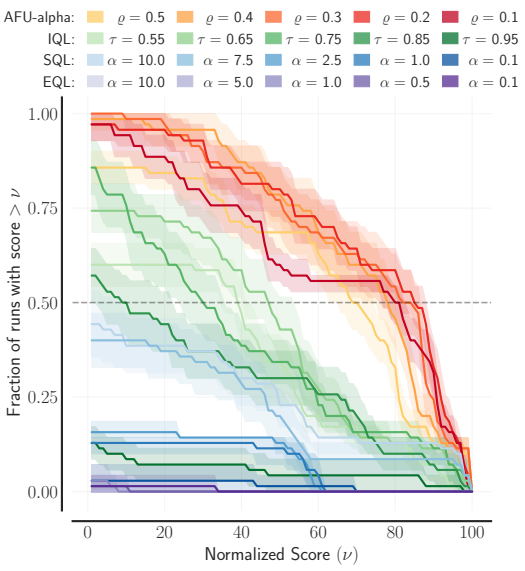
Experiments We test AFU-alpha on a classical benchmark of 7 MuJoCo [29] tasks from the Gymnasium library [30]. We compare it to SAC and TD3, and to variants of AFU-alpha in which the loss $L_{V,A}$ aiming at solving the max-Q problem is replaced by the corresponding loss taken from IQL, SQL or EQL. The results are shown in Figure 2. For both SAC and AFU-alpha, we use the same heuristic for the definition of $\bar{\mathcal{H}}$: we set it to $-d$, where d is the dimension of the action space. Updates in AFU-alpha, SAC and TD3 use the same value of τ and same learning rates. For each algorithm, for each value of the hyperparameter (ϱ for AFU-alpha, τ for IQL, α for SQL and EQL), and for each of the 7 MuJoCo tasks, we perform 10 runs initialized with different random seeds, and evaluate the performance of the policy every 10,000 steps on 10 rollouts. The first 10,000 steps of each run use uniformly drawn random actions (and no gradient steps). Learning curves are smoothed with a moving average window of size 10. The raw score of a run is the last average return, i.e. the average return over the last 10 evaluations. For each task, we linearly rescale the scores based on two reference points: (1) the maximum evaluation seen across all algorithms and all runs corresponds to a score of 100, and (2) the mean episode return across all algorithms and runs corresponds to a score of 0. Following the recommendations of [1], we compute with the *rliable* library the performance profiles for each algorithm across the 7 tasks: Ant-v4, HalfCheetah-v4, Hopper-v4, Humanoid-v4, InvertedDoublePendulum-v4, Reacher-v4 and Walker2d-v4. The length of the runs is 1 million steps for InvertedDoublePendulum and Reacher, 3 million steps for Ant, Hopper, Humanoid and Walker2d, and 5 million steps for HalfCheetah.

In Figure 2a, we see that our proposed method for the max-Q problem yields significantly better results than the IQL, SQL and EQL baselines. The best results are obtained with $\varrho \in \{0.2, 0.3\}$. Figure 2b shows that AFU-alpha is competitive with SAC and TD3.

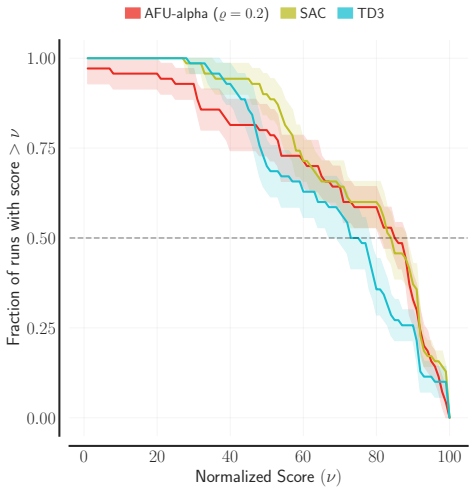
7 A simple failure mode of SAC

With a deterministic actor trained by stochastic gradient ascent over the Q-function landscape, DDPG, TD3 and similarly structured deterministic actor-critic algorithms can easily get stuck in local optima (see [22]). With a stochastic actor and updates based on the Kullback-Leibler (KL) divergence between output distributions and target distributions of the form $\exp\big(\frac{1}{\alpha}Q(s,\cdot)\big)/z(s)$, algorithms like SAC are less prone to deadlocks. For instance, in areas where the gradient of the Q-function is close to zero, exploiting the KL loss results in an increase of the variance of the action distribution, which eventually helps find larger gradients and escape from the flat region. Yet, the policy networks used in practice mostly output unimodal action distributions[1], and with this restriction even the KL

[1]This is starting to change, thanks to the influence of recent methods such as diffusion policies (see [5, 12]), but such expressive and multimodal stochastic policies are still more cumbersome than unimodal policies.



(a) AFU-alpha works best with $\varrho \in \{0.2, 0.3\}$. Using the IQL, SQL and EQL baselines to solve the max-Q problem results in a clear performance deterioration.



(b) AFU-alpha is competitive with SAC and TD3 on a benchmark of 7 diverse tasks, with action space dimensions ranging from 1 (InvertedDoublePendulum) to 17 (Humanoid), and observation space dimensions ranging from 11 to 376 (Humanoid).

Figure 2: Experimental evaluation of AFU-alpha on a benchmark of 7 MuJoCo tasks.

loss generates undesirable local optima. We illustrate this with a trivial environment which we call SFM (for "SAC Failure Mode"). It consists of a single state s_0, and unidimensional actions in $[-1, 1]$. The reward of an action is given by the function:

$$R_{SFM}(s_0, a) = \begin{cases} 5 - 100(a - 0.1)^2, & \text{if } a \geq -0.6, \\ 0, & \text{otherwise.} \end{cases}$$

All transitions are terminal, so all episodes stop after one step. The optimal policy selects $a = 0.1$ and yields a return of 5. We train SAC on SMF with the same hyperparameters as in our other experiments. We start by performing 1000 steps with random actions, which helps the critic Q_{SAC} quickly converge toward the optimal Q-function, Q^*, which is simply equal to R_{SFM}. Figure 3 shows Q_{SAC} after 20,000 steps. Although Q_{SAC} converges toward a very precise approximation of Q^*, the actor policy converges toward a suboptimal solution, as shown in Figure 4a. If we just modify SAC by locking the mode of the policy distribution at 0, we can see in Figure 4b that the actor loss becomes much smaller, even after convergence of the actor entropy, which indicates that the policy of the default SAC algorithm gets stuck in a local optimum. There are two phases in the failure mode: at the beginning, when the entropy is relatively large, the asymmetry of R_{SFM} makes the actor shift toward -1. As seen in Figure 3, Q_{SAC} approximates the discontinuity in R_{SFM} with a steep slope, and when the policy distribution becomes concentrated on the left of this slope, it acts as a barrier that traps the actor. Later in the training, when the entropy becomes smaller and converges to the target entropy (-1), it would be much preferable for the mode of the policy to converge back toward 0.1, but the steep slope results in a deceptive gradient in the KL loss that prevents it from happening, and SAC remains stuck in the local optimum.

8 AFU-beta

With the same actor loss as SAC, AFU-alpha fails similarly on SFM. We propose to improve the actor loss to make it less likely to get stuck in local optima. The first idea is to train by regression an estimate of where the mode of the actor should be. If the learning progresses well, $Q_\psi(s, a) > \min_{i \in \{1,2\}} \left(V_{\varphi_i}(s)\right)$ should only be possibly true in the vicinity of the argmax ($\text{argmax}_{a \in A} \left(Q(s, a)\right)$), so we use actions a with a Q-value greater than $\min_{i \in \{1,2\}} \left(V_{\varphi_i}(s)\right)$ as targets. To find such actions we use both actions in the mini-batches and actions resampled with the actor on those mini-batches.

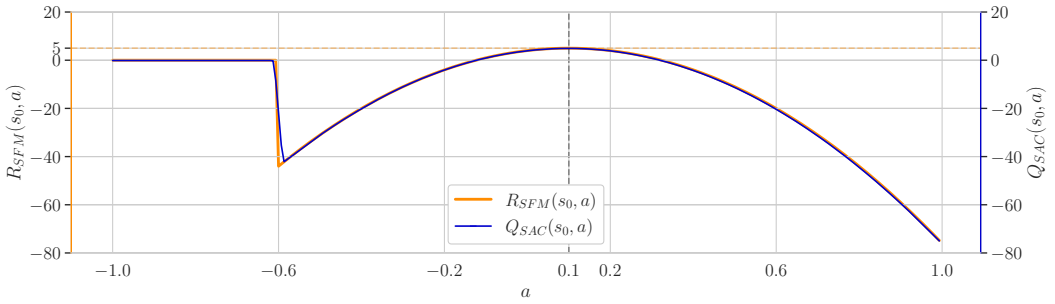
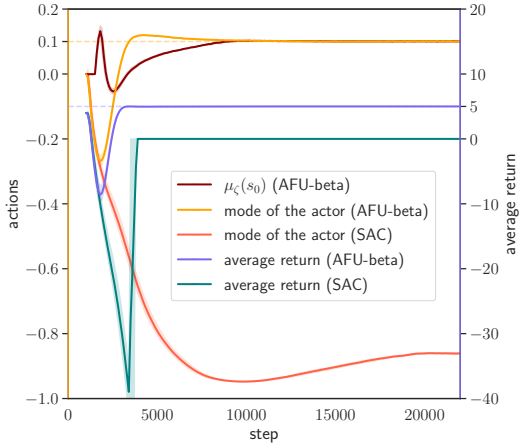
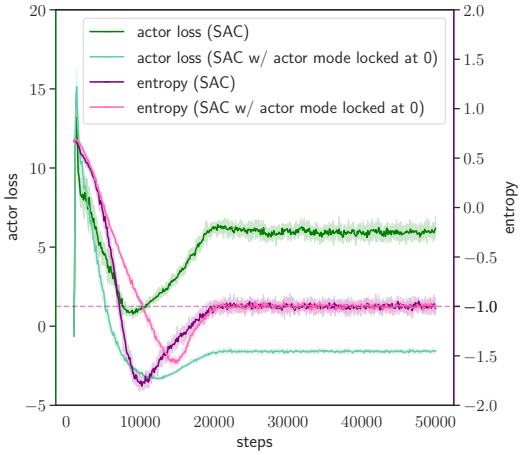


Figure 3: In orange: the reward function R_{SFM} of the SFM environment. Since all transitions are terminal, R_{SFM} coincides with the optimal Q-function. In blue: the critic (Q_{SAC}) obtained after a training of 20,000 steps with SAC [11].



(a) AFU-beta converges to the optimal solution, while SAC converges to a suboptimal solution. How to read the figure: the y-axis on the left (action) applies to the first three curves ($\mu_\zeta(s)$ and the modes of the actor for AFU-beta and SAC), while the y-axis on the right (average return) applies for the two other curves (average returns for AFU-beta and SAC).



(b) Evolution of the actor loss and entropy for SAC and SAC with the mode of its actor locked at 0. The y-axis on the left (actor loss) applies to the first two curves (actor losses for both versions of SAC), and the y-axis on the right applies to the two other curves (entropies for both versions of SAC).

Figure 4: Trainings of SAC and AFU-beta in the SFM environment. Plots show results averaged over 10 runs with different random seeds, and shaded areas range from the 25th to the 75th percentile.

Let us consider a mini-batch B of transitions (s, a, r, s'), and actions a_s resampled with the actor. We denote by $\mathcal{M}(B)$ the set of state-action pairs $(s, a_\bullet)$ such that $a_\bullet = a$ or $a_\bullet = a_s$ and $Q_\psi(s, a_\bullet) > \min_{i\in\{1,2\}}\big(V_{\varphi_i}(s)\big)$. We introduce a new deterministic function approximator $\mu_\zeta : S \to A$ with parameters ζ and train it with the following loss:

$$L_\mu(\zeta) = \underset{(s,a_\bullet)\in\mathcal{M}(B)}{\text{Mean}} \left[\Big(\mu_\zeta(s) - a_\bullet\Big)^2\right]. \tag{8}$$

In our implementation, most of the parameters between ζ and θ are shared: we simply modify the output dimension of π_θ to make it also return $\mu_\zeta(s)$. It does not change the approach in any way, but when computing the gradient of the loss (8), one must carefully ignore the influence of the parameters ζ on resampled actions a_s.

Let us reconsider the actor loss from Equation (6). It balances two terms, the first one ($\alpha \log(\pi_\theta(a_s|s))$) that maximizes the entropy, and the second one ($-Q_\psi(s, a_s)$) that encourages π_θ to output actions maximizing $Q_\psi(s, \cdot)$. In the gradient $\nabla_\theta L_\pi(\theta)$, which can be expressed by making explicit the relationship between sampled actions a_s and the input noise (see [11]), the second term results in small modifications of θ that attempt to change the actions a_s in the direction of

$\nabla_a Q_\psi(s,a)$, where a is evaluated in a_s, and which we write by abuse of notation $\nabla_{a_s} Q_\psi(s,a_s)$. If $\nabla_{a_s} Q_\psi(s,a_s)$ points away from the global optimum, it can contribute to the creation of a local minimum in the actor loss. We want to edit $\nabla_{a_s} Q_\psi(s,a_s)$ in order to avoid deceptive gradients. To do so, we compute the dot product between $\nabla_{a_s} Q_\psi(s,a_s)$ and $\mu_\zeta(s) - a_s$, which is an estimate of a direction toward $\text{argmax}_{a \in A}\big(Q(s,a)\big)$. If the dot product is positive or zero, the gradient does not point away from $\mu_\zeta(s)$, so we can keep it unchanged. However, if $\nabla_{a_s} Q_\psi(s,a_s) \cdot (\mu_\zeta(s) - a_s) < 0$, then we project $\nabla_{a_s} Q_\psi(s,a_s)$ onto $(\mu_\zeta(s) - a_s)^\perp$ to anneal the dot product. We do it only if we estimate that a_s is not already in the vicinity of the argmax, i.e. if $Q_\psi(s,a_s) < \min_{i\in\{1,2\}}\big(V_{\varphi_i}(s)\big)$. We introduce the following operator:

$$\mathcal{G}^{s,a_s}(v) = \begin{cases} \text{proj}_{(\mu_\zeta(s)-a_s)^\perp}\big(v\big), & \text{if } v \cdot (\mu_\zeta(s) - a_s) < 0 \text{ and } Q_\psi(s,a_s) < \min_{i\in\{1,2\}}\big(V_{\varphi_i}(s)\big). \\ v, & \text{otherwise.} \end{cases}$$

When computing the gradient $\nabla_\theta L_\pi(\theta)$, we replace the terms $\nabla_{a_s} Q_\psi(s,a_s)$ (resulting from the chain rule) by $\mathcal{G}^{s,a_s}\big(\nabla_{a_s} Q_\psi(s,a_s)\big)$. It leads to a modified gradient which we denote by $\nabla_\theta^{\text{MODIF}} L_\pi(\theta)$.

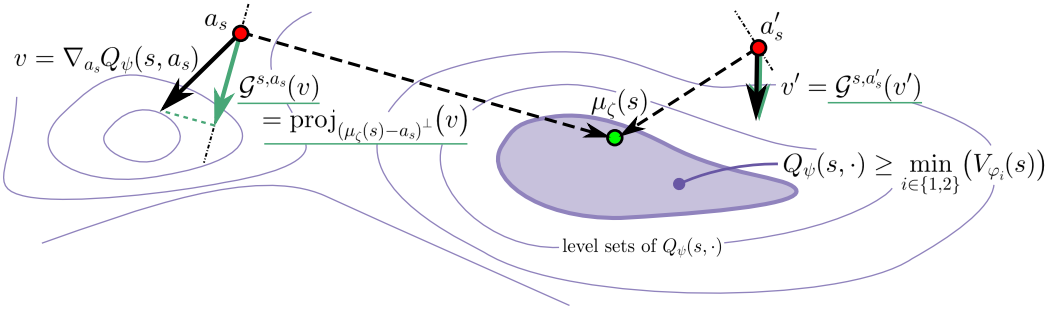


Figure 5: The gradient v at a_s (on the left) points away from $\mu_\zeta(s)$, which determines the direction toward the vicinity of the argmax of $Q_\psi(s,\cdot)$, so we modify v to get $\mathcal{G}^{s,a_s}\big(v\big)$ by projecting it on the hyperplane orthogonal to $\mu_\zeta(s) - a_s$. The gradient v' at a'_s (on the right) points in the direction (half-space) of $\mu_\zeta(s)$, so we do not modify it, and $\mathcal{G}^{s,a'_s}\big(v'\big) = v'$.

This process is illustrated in Figure 5. It can be understood as an artificial modification of the landscape of $Q_\psi(s,\cdot)$ so that, outside the region defined by $Q_\psi(s,\cdot) \geq \min_{i\in\{1,2\}}\big(V_{\varphi_i}(s)\big)$, its gradient never points away from $\mu_\zeta(s)$. $\mu_\zeta(s)$ has the advantage of being trained by regression, and its training includes actions coming directly from the replay buffer, not only ones resampled by the actor. It means that, in a very off-policy setting, if a new peak of $Q_\psi(s,a)$ appears far from the actions currently likely to be sampled, the update of $\mu_\zeta(s)$ can occur first and then guide the update of π_θ by removing all deceptive gradients that would need to be crossed to reach the new peak. More generally, the use of μ_ζ prevents the actor from being trapped in local optima, as long as the training of the critic is doing well. Since training the critic is independent from the actor, we believe that our proposed approach goes one step further in the development of sound foundations for a purely off-policy reinforcement learning algorithm performing well in continuous action spaces.

We call AFU-beta the updated algorithm. It works like AFU-alpha, with the additional training of μ_ζ and the replacement of $\nabla_\theta L_\pi(\theta)$ by $\nabla_\theta^{\text{MODIF}} L_\pi(\theta)$, as described in Algorithm 1. Figure 4a shows that, in the SFM environment, unlike SAC, AFU-beta quickly converges to the optimal solution.

We evaluate AFU-beta on the MuJoCo benchmark in the same way as AFU-alpha and show results in Figure 7 (Appendix F). Again, AFU-beta is competitive with SAC and TD3. The differences between AFU-beta and AFU-alpha are not very significant on the MuJoCo benchmark, possibly because issues with local optima are rarely encountered in these environments. We leave for future work the search for meaningful and complex environments in which AFU-beta has a notable advantage over AFU-alpha.

9 Conclusion

We presented AFU, an off-policy RL algorithm with critic updates independent from the actor. At its core is a novel way to solve the continuous action Q-function maximization (max-Q) problem using

regression and conditional gradient scaling, which we believe could have applications outside the field of reinforcement learning.

The first version of AFU (AFU-alpha) has a stochastic actor trained as in SAC [11]. We provide a simple example of failure mode for SAC, and show how the value function trained in AFU can help improve the actor loss and make it less prone to local optima, resulting in a second version of AFU (AFU-beta) which does not exhibit the same failure mode as SAC.

Our experimental results on a classical benchmark show that both versions of AFU are competitive with SAC and TD3, two state-of-the-art off-policy model-free RL algorithms. As far as we know, AFU is the first off-policy RL algorithm that is competitive in sample-efficiency with the state-of-the-art and truly departs from the actor-critic perspective. We believe that it could open up new avenues for off-policy RL algorithms applied to continuous control problems.

References

[1] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34:29304–29320, 2021.

[2] Brandon Amos, Lei Xu, and J Zico Kolter. Input convex neural networks. In *International Conference on Machine Learning*, pages 146–155. PMLR, 2017.

[3] Leemon C. Baird. *Reinforcement learning through gradient descent*. PhD thesis, Carnegie Mellon University Pittsburgh, PA, USA, 1999.

[4] Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized Ensembled Double Q-learning: Learning fast without a model. *arXiv preprint arXiv:2101.05982*, 2021.

[5] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[6] Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

[7] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[8] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, volume 97, pages 2052–2062. PMLR, 2019.

[9] Divyansh Garg, Joey Hejna, Matthieu Geist, and Stefano Ermon. Extreme q-learning: Maxent RL without entropy. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[10] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep Q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838. PMLR, 2016.

[11] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[12] Philippe Hansen-Estruch, Ilya Kostrikov, Michael Janner, Jakub Grudzien Kuba, and Sergey Levine. IDQL: implicit q-learning as an actor-critic method with diffusion policies. *CoRR*, abs/2304.10573, 2023.

[13] Hado Hasselt. Double Q-learning. *Advances in Neural Information Processing Systems*, vol. 23, 2010.

[14] Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. Dropout Q-functions for doubly efficient reinforcement learning. *arXiv preprint arXiv:2110.02034*, 2021.

[15] Tianying Ji, Yu Luo, Fuchun Sun, Xianyuan Zhan, Jianwei Zhang, and Huazhe Xu. Seizing serendipity: Exploiting the value of past success in off-policy actor-critic. *arXiv preprint arXiv:2306.02865*, 2023.

[16] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. QT-Opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with Implicit Q-learning. In *The 10th International Conference on Learning Representations (ICLR)*, 2022.

[19] Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In *International Conference on Machine Learning*, pages 5556–5566. PMLR, 2020.

[20] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. *Advances in Neural Information Processing Systems*, 20, 2007.

[21] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[22] Guillaume Matheron, Nicolas Perrin, and Olivier Sigaud. Understanding failures of deterministic actor-critic with continuous action spaces and sparse rewards. In *International Conference on Artificial Neural Networks*, pages 308–320. Springer, 2020.

[23] Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep RL. *arXiv preprint arXiv:1705.05035*, 2017.

[24] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937. PMLR, 2016.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[26] Brendan O'Donoghue, Remi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. Combining policy gradient and Q-learning. *arXiv preprint arXiv:1611.01626*, 2016.

[27] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-Weighted Regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[28] Moonkyung Ryu, Yinlam Chow, Ross Anderson, Christian Tjandraatmadja, and Craig Boutilier. CAQL: Continuous Action Q-Learning. In *8th International Conference on Learning Representations (ICLR)*, 2020.

[29] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[30] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.

[31] Christopher J.C.H. Watkins. Learning from delayed rewards. 1989.

[32] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[33] Yanqiu Wu, Xinyue Chen, Che Wang, Yiming Zhang, and Keith W Ross. Aggressive Q-learning with ensembles: Achieving both high sample efficiency and high asymptotic performance. *arXiv preprint arXiv:2111.09159*, 2021.

[34] Haoran Xu, Li Jiang, Jianxiong Li, Zhuoran Yang, Zhaoran Wang, Wai Kin Victor Chan, and Xianyuan Zhan. Offline RL with no OOD actions: In-sample learning via Implicit Value Regularization. In *The 11th International Conference on Learning Representations (ICLR)*, 2023.

A Hyperparameters

The following hyperparameters were used in all our experiments.

We did not do any reward scaling.

AFU, SAC & TD3 Hyperparameters

optimizer	Adam [17]
actor learning rate	$3 \cdot 10^{-4}$
critic learning rate	$3 \cdot 10^{-4}$
temperature learning rate **(only AFU & SAC)**	$3 \cdot 10^{-4}$
discount (γ)	0.99
replay buffer size	10^6
initial steps with random actions	10^4
number of hidden layers (all networks)	2
number of hidden units per layer	256
number of samples per mini-batch	256
nonlinearity	ReLU
target smoothing coefficient (τ)	0.01
target update interval	1
policy update interval **(only TD3)**	2
exploration noise standard deviation **(only TD3)**	0.2
noise clipping **(only TD3)**	0.5
target entropy **(only AFU & SAC)**	$-d$ ($d =$ action space dimension)
initial temperature **(only AFU & SAC)**	1
max. actor log std (before tanh) **(only AFU & SAC)**	2
min. actor log std (before tanh) **(only AFU & SAC)**	-10

B Conditional gradient rescaling seen as adaptive regularization

Let us denote by $e(s,a)$ the error:

$$e(s,a) = V_{\varphi_i}(s) + A_{\xi_i}(s,a) - Q_\psi(s,a),$$

and let us assume that this error is negative (otherwise our method simply applies a standard gradient descent step). We denote by $e'(s,a)$ the following term:

$$e'(s,a) = (1-\varrho)V_{\varphi_i}(s) + \varrho V_{\varphi_i^{\text{no\_grad}}}(s) + A_{\xi_i}(s,a) - Q_\psi(s,a),$$

which is equal to $e(s,a)$ in value but has different gradients.

Our method applies a gradient descent step on $e'(s,a)^2$ for both φ_i and ξ_i. For ξ_i, the gradient is the same as for $e(s,a)^2$ and $e'(s,a)^2$. For φ_i, the gradients are:

$$\nabla_{\varphi_i} e(s,a)^2 = 2e(s,a)\nabla_{\varphi_i} V_{\varphi_i}(s),$$
$$\nabla_{\varphi_i} e'(s,a)^2 = (1-\varrho)2e(s,a)\nabla_{\varphi_i} V_{\varphi_i}(s).$$

Let us define $e^{\text{no\_grad}}(s,a)$: a "frozen" version of $e(s,a)$ leading to no gradients at all (i.e. relying on copies of both φ_i and ξ_i). Our method is equivalent to the application of a gradient step (for both φ_i and ξ_i) to the following term:

$$e(s,a)^2 - 2\varrho e^{\text{no\_grad}} V_{\varphi_i}(s) = e(s,a)^2 + 2\varrho |e^{\text{no\_grad}}(s,a)| V_{\varphi_i}(s),$$

where we see the squared error and a simple regularization term that penalizes large values of $V_{\varphi_i}(s)$ (thus putting a "downward pressure" on $V_{\varphi_i}(s)$). As a result, the proposed conditional gradient rescaling method can be understood as an adaptive regularization scheme in which the regularization weight is proportional to the absolute value of the error. It means that the convergence of $V_{\varphi_i}(s)$ toward $\max_{a \in A}(Q_\psi(s,a))$ is not theoretically guaranteed: if the regression quickly converges to an exact solution, the gradients vanish and $V_{\varphi_i}(s)$ can remain strictly greater than the true maximum. However, if a non negligible error remains, the adaptive regularization is effective and $V_{\varphi_i}(s)$ progressively decreases toward an approximation of the true maximum, which is what we observe in practice.

C Constraining the sign of $A_{\xi_i}(s,a)$ in a soft way

We let A_{ξ_i} possibly return positive outputs, but we modify the regression loss to have only non-positive *targets* for $A_{\xi_i}(s,a)$. In Equation (3), we call $Q_\psi(s,a) - \Upsilon_i^a(s)$ the *target* of $A_{\xi_i}(s,a)$, as it is the value of $A_{\xi_i}(s,a)$ minimizing $\Big(\Upsilon_i^a(s) + A_{\xi_i}(s,a) - Q_\psi(s,a)\Big)^2$. If $Q_\psi(s,a) - \Upsilon_i^a(s) > 0$, the best non-positive *target* for $A_{\xi_i}(s,a)$ is 0, in which case the *target* for $\Upsilon_i^a(s) - Q_\psi(s,a)$ should also be 0. In this situation, we replace $\Big(\Upsilon_i^a(s) + A_{\xi_i}(s,a) - Q_\psi(s,a)\Big)^2$ by $\Big(\Upsilon_i^a(s) - Q_\psi(s,a)\Big)^2 + \Big(A_{\xi_i}(s,a)\Big)^2$. To do so, we introduce Z:

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

The loss of Equation 3 is updated as follows:

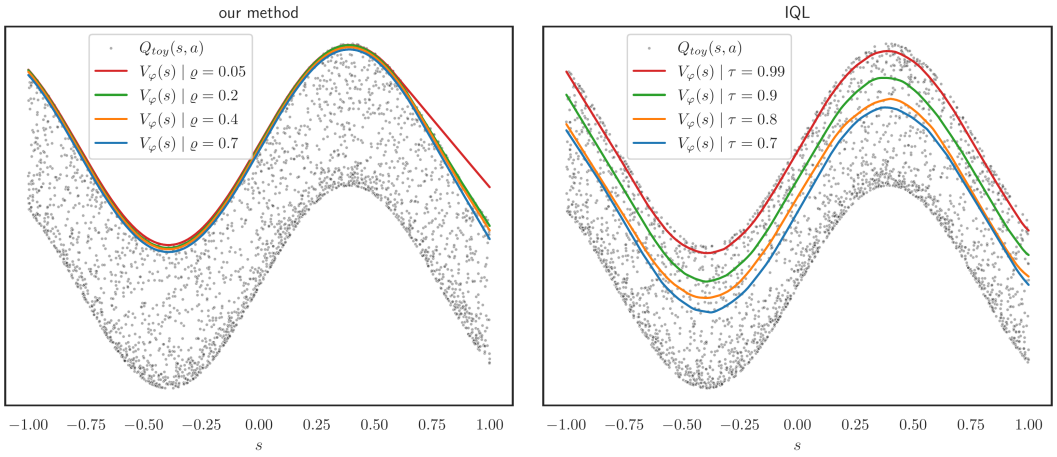
$$\Lambda'_{V,A}(\varphi_i, \xi_i) = \underset{(s,a,\_,\_)\in B}{\text{Mean}} \Big[Z\Big(\Upsilon_i^a(s) - Q_\psi(s,a), A_{\xi_i}(s,a)\Big)\Big].$$

D Experiments on a toy max-Q problem

We empirically compare our method to 3 baselines (IQL, SQL and EQL) on a toy problem. We define the function $Q_{toy}(s,a) = \sin(4s) + 0.7\cos(4a)$ for $s \in [-1,1]$ and $a \in [-1,1]$. We use a single feedforward neural network for V (V_φ) and a single feedforward neural network for A (A_ξ). Both networks have two hidden layers of size 256 and ReLU activations in the hidden layers. Our method trains both V_φ and A_ξ, while the 3 baselines IQL, SQL and EQL directly train V_φ. All 3 baselines have been successfully applied to offline reinforcement learning.

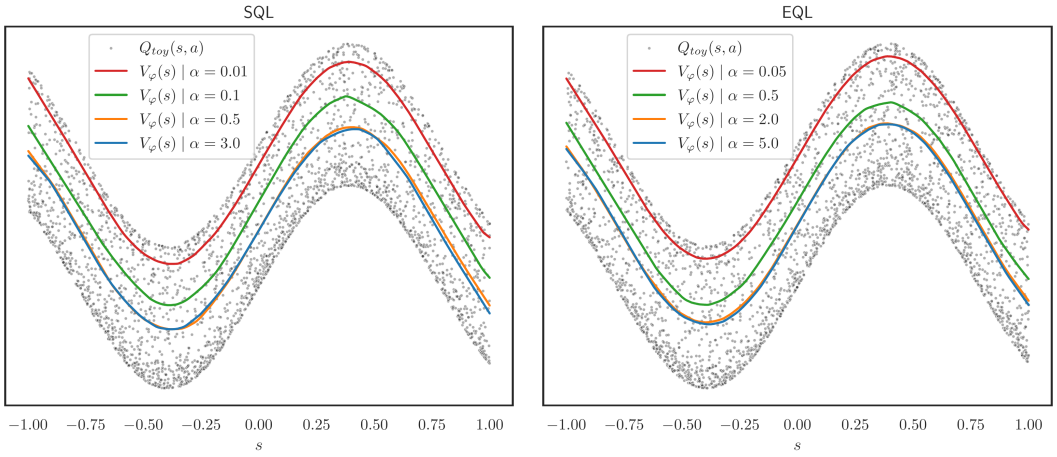
SQL and EQL are derived in [34] from a general method called Implicit Value Regularization. It relies on a behavior-regularized MDP with a term that penalizes policies diverging from the underlying behavior policy of the training dataset. Various f-divergences can be used to measure the difference between the policy and the behavior policy, resulting in distinct algorithms, including SQL and EQL which are special cases. They have distinct losses for the training of $V_\varphi(s)$, both depending on a parameter α, and in both cases, for $\alpha \to 0$, $V_\varphi(s)$ is trained to approximate the maximum operator over in-support values, i.e. $\max_{a\in A}(Q_{toy}(s,a))$. However, similarly to IQL, very small values of α result in unbalanced losses, so in practice the values leading to the best results on the benchmarks tested in [34] are $\alpha = 0.1$, $\alpha = 0.5$, $\alpha = 1$ and $\alpha = 3$ for SQL, and $\alpha = 0.5$, $\alpha = 2.0$ and $\alpha = 5$ for EQL.

For IQL, SQL and EQL, since unbalanced losses are not an issue on this simple toy problem, we include parameters leading to a better resolution of the max-Q problem, but that are not representative of the parameters working well in actual offline RL experiments. We observe that, with our method, although $\varrho = 0.05$ leads to overestimations, for a wide range of parameter values (from $\varrho = 0.2$ to $\varrho = 0.7$), we obtain more accurate results than with all the other baselines, even when considering parameter values that are inapplicable to offline RL. Besides, with our proposed approach, the different values of ϱ that perform well do not result in unbalanced losses.



(a) $V_\varphi(s)$ is trained jointly with $A_\xi(s,a)$ by iterating gradient descent steps on the loss $\Lambda'_{V,A}(\varphi,\xi)$ described by Equation (4). $\varrho \in [0.2, 0.7]$ results in precise approximations of $s \mapsto \max_{a\in A}(Q_\psi(s,a))$.

(b) Results of the training with the loss from IQL [18] for 4 different values of the hyperparameter τ. Values used in actual (offline) RL experiments are not greater than 0.9.



(c) Results of the training with the loss from SQL [34] for 4 different values of the hyperparameter α. Values used in actual (offline) RL experiments are not smaller than 0.1.

(d) Results of the training with the loss from EQL [34] for 4 different values of the hyperparameter α. Values used in actual (offline) RL experiments are not smaller than 0.5.

Figure 6: $Q_{toy}(s,a) = \sin(4s) + 0.7\cos(4a)$ for $s \in [-1,1]$ and $a \in [-1,1]$. We compare our method to IQL, SQL and EQL which all train $V_\varphi(s)$ to approximate the function $s \mapsto \max_{a\in A}(Q_{toy}(s,a))$, i.e. solve the max-Q problem. All trainings are done with 3000 gradient descent steps. At each step, a loss is computed on a batch composed of 256 uniformly randomly drawn values of s and a.

E SAC-like actor training

Let π_θ denote the actor. We follow a common implementation in which its backbone is a feedforward neural network returning action distributions as state-dependent Gaussians with diagonal covariance matrices. Since actions are usually constrained between -1 and 1, we apply a $\tanh$ transformation to its outputs. Given a state s, the resulting probability density function is $\pi_\theta(\cdot|s)$. The actor π_θ can transform input noise vectors sampled from a fixed distribution into action samples. Again, we train π_θ on mini-batches of transitions. We use the actor to resample an action a_s for each state s of a mini-batch B. The actor loss $L_\pi(\theta)$ is based on the average Kullback-Leibler divergence between the

actor's output distributions and targeted Boltzmann policy distributions. It is defined as follows:

$$L_\pi(\theta) = \underset{\substack{(s,\_,\_,\_)\in B \\ a_s\sim\pi_\theta(\cdot|s)}}{\text{Mean}}\Big[\alpha\log(\pi_\theta(a_s|s)) - Q_\psi(s,a_s)\Big],$$

where α is a temperature parameter. As in SAC, we adjust this temperature via gradient descent on a loss aiming at keeping the average entropy of action distributions close to a target entropy $\bar{\mathcal{H}}$:

$$L_{\text{temp}}(\alpha) = \underset{\substack{(s,\_,\_,\_)\in B \\ a_s\sim\pi_\theta(\cdot|s)}}{\text{Mean}}\Big[-\alpha\log(\pi_\theta(a_s|s)) - \alpha\bar{\mathcal{H}}\Big].$$

F Experimental results for AFU-beta

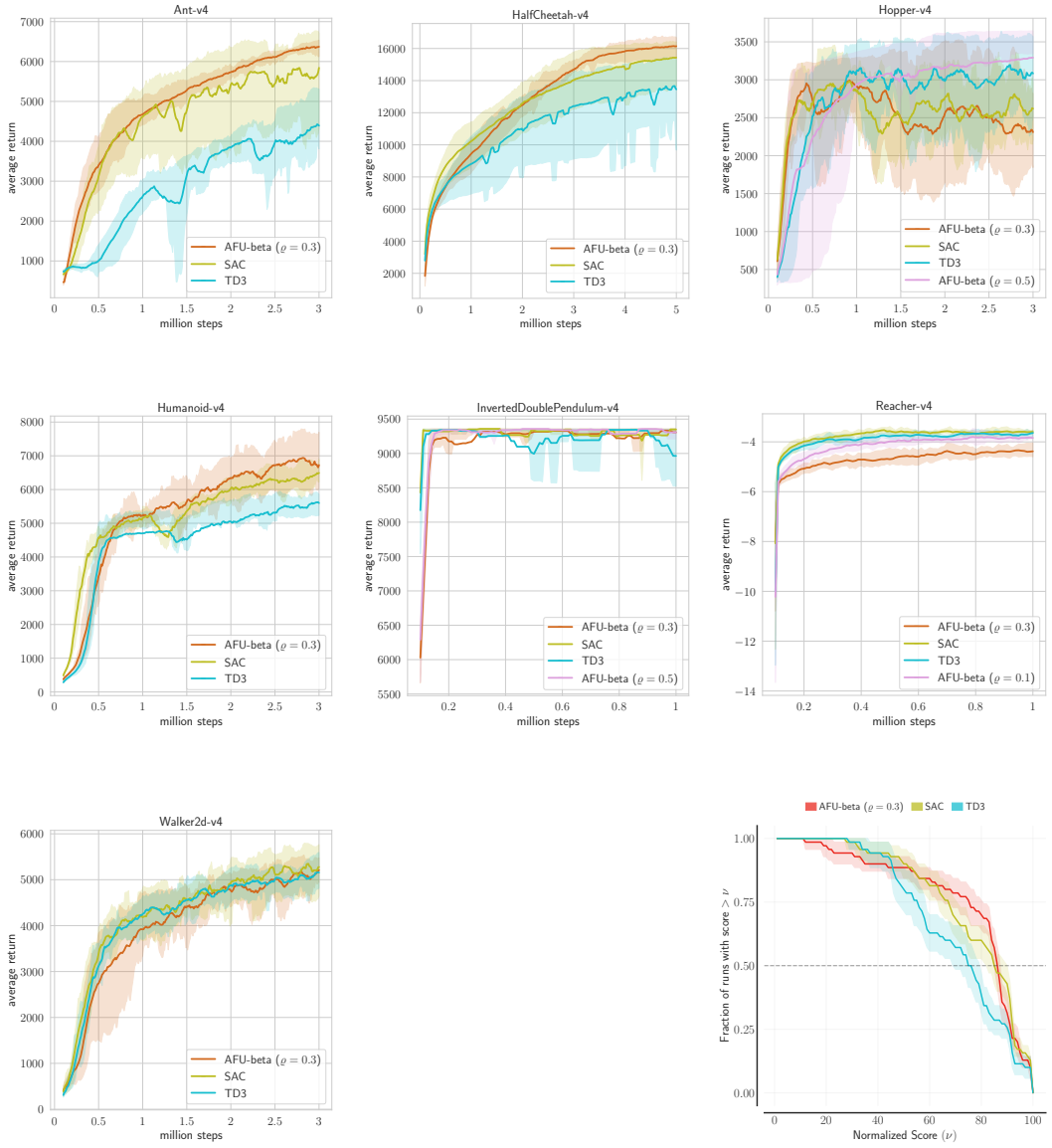


Figure 7: Experimental evaluation of AFU-beta for $\varrho = 0.3$ on 7 MuJoCo tasks. We show results with other values of ϱ (among $\{0.1, 0.2, 0.4, 0.5\}$) for tasks in which one of the other values performed significantly better than 0.3. Results are averaged over 10 runs with different random seeds, and the shaded areas range from the 25th to the 75th percentile. The performance profile plot at the bottom right summarizes results and shows that AFU-beta is competitive with SAC and TD3.

G Learning curves

The plots below show learning curves for AFU-alpha and AFU-beta for all the values of the hyperparameter ϱ (in $\{0.1, 0.2, 0.3, 0.4, 0.5\}$).

All learning curves are averaged over 10 runs with different random seeds, and the shaded areas range from the 25th to the 75th percentile. For each run, evaluations are done over 10 rollouts every 10,000 steps, and each run is smoothed with a moving average window of size 10. The first 10,000 steps are always done without gradient steps and with uniformly randomly drawn actions.

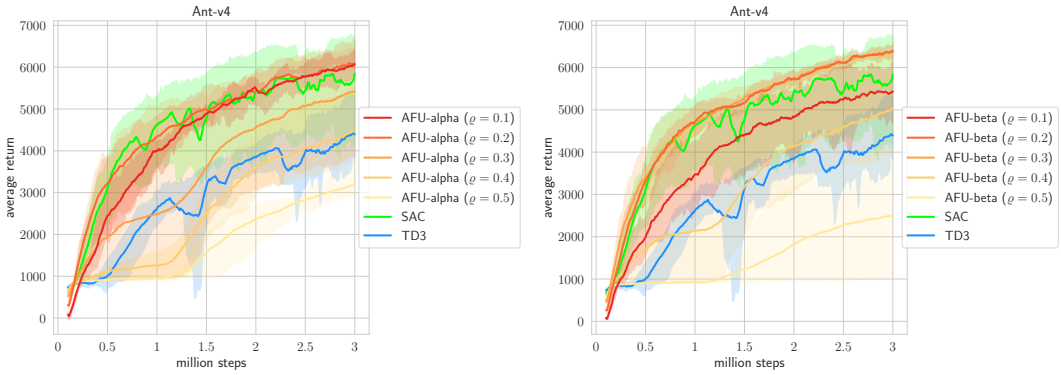


Figure 8: Left: AFU-alpha on Ant-v4. Right: AFU-beta on Ant-v4.

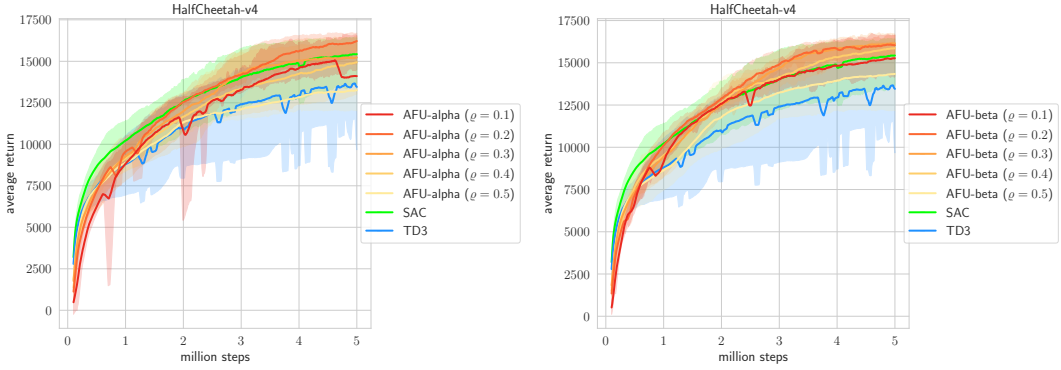


Figure 9: Left: AFU-alpha on HalfCheetah-v4. Right: AFU-beta on HalfCheetah-v4.

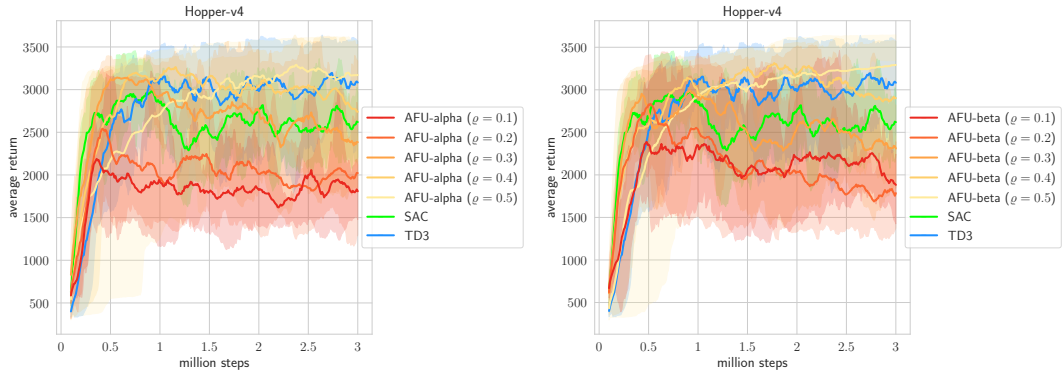


Figure 10: Left: AFU-alpha on Hopper-v4. Right: AFU-beta on Hopper-v4.

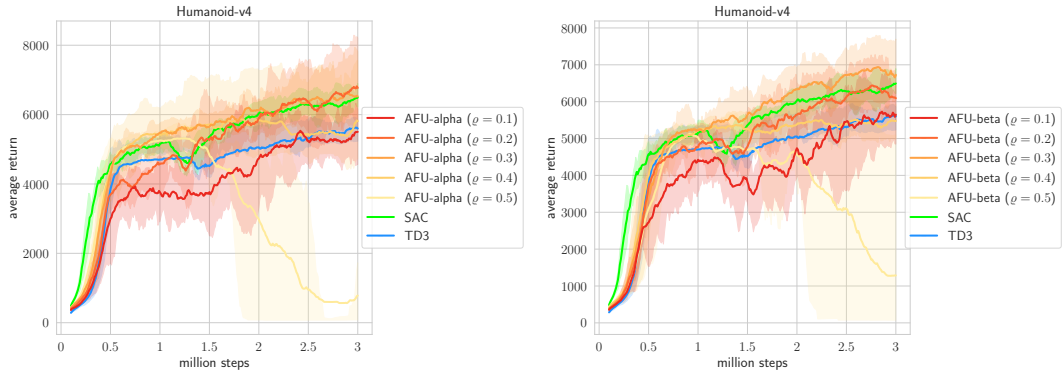


Figure 11: Left: AFU-alpha on Humanoid-v4. Right: AFU-beta on Humanoid-v4.

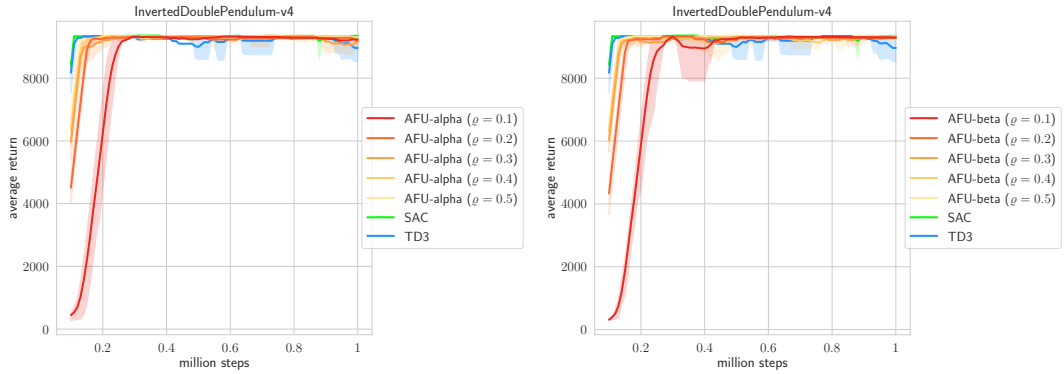


Figure 12: Left: AFU-alpha on InvertedDoublePendulum-v4. Right: AFU-beta on InvertedDoublePendulum-v4.

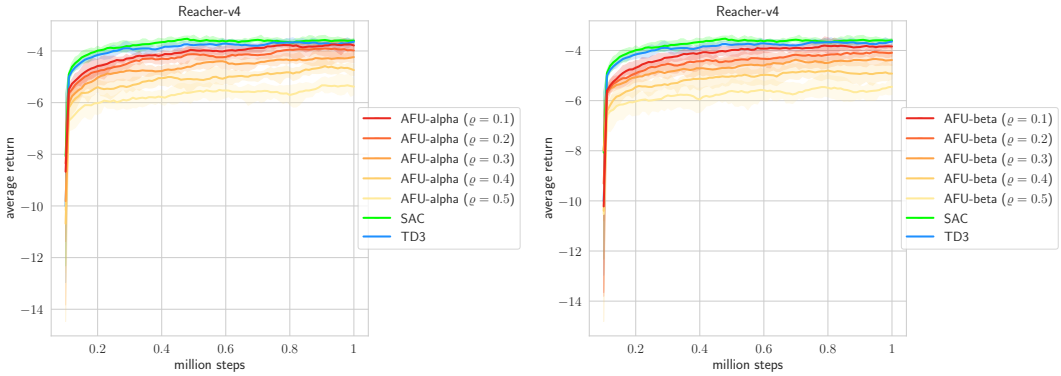


Figure 13: Left: AFU-alpha on Reacher-v4. Right: AFU-beta on Reacher-v4.

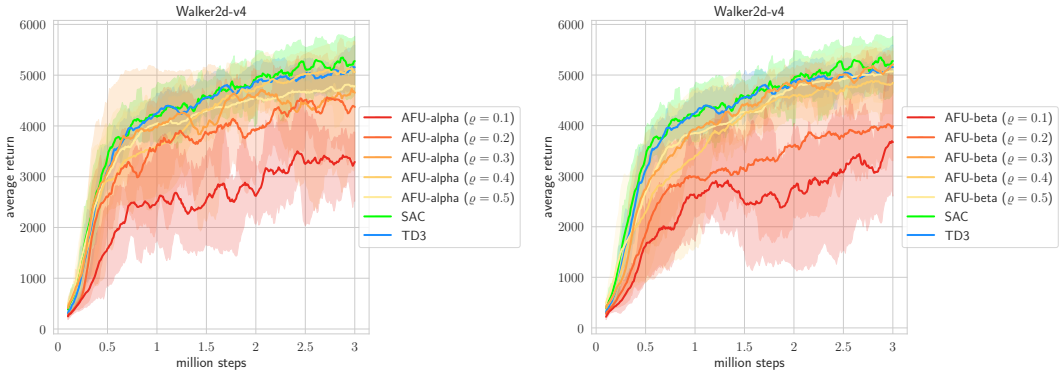


Figure 14: Left: AFU-alpha on Walker2d-v4. Right: AFU-beta on Walker2d-v4.