Adaptive Q-Network: On-the-fly Target Selection for Deep Reinforcement Learning

Théo Vincent[1,2,*] **Fabian Wahren**[1,2] **Jan Peters**[1,2,3,4]

Boris Belousov[1] **Carlo D'Eramo**[2,3,5]

[1]DFKI GmbH, SAIROL [2] Department of Computer Science, TU Darmstadt
[3] Hessian.ai [4] Centre for Cognitive Science, TU Darmstadt
[5]Center for Artificial Intelligence and Data Science, University of Würzburg

Abstract

Deep Reinforcement Learning (RL) is well known for being highly sensitive to hyperparameters, requiring practitioners substantial efforts to optimize them for the problem at hand. In recent years, the field of automated Reinforcement Learning (AutoRL) has grown in popularity by trying to address this issue. However, these approaches typically hinge on additional samples to select well-performing hyperparameters, hindering sample-efficiency and practicality in RL. Furthermore, most AutoRL methods are heavily based on already existing AutoML methods, which were originally developed neglecting the additional challenges inherent to RL due to its non-stationarities. In this work, we propose a new approach for AutoRL, called Adaptive Q-Network (AdaQN), that is tailored to RL to take into account the non-stationarity of the optimization procedure without requiring additional samples. AdaQN learns several Q-functions, each one trained with different hyperparameters, which are updated online using the Q-function with the smallest approximation error as a shared target. Our selection scheme simultaneously handles different hyperparameters while coping with the non-stationarity induced by the RL optimization procedure and being orthogonal to any critic-based RL algorithm. We demonstrate that AdaQN is theoretically sound and empirically validate it in MuJoCo control problems, showing benefits in sample-efficiency, overall performance, training stability, and robustness to stochasticity.

1 Introduction

Deep Reinforcement Learning (RL) has proven effective at solving complex sequential decision problems in various domains (Mnih et al., 2015; Haarnoja et al., 2018; Silver et al., 2017). Despite their success in many fields, deep RL algorithms suffer from brittle behavior with respect to their hyperparameters (Mahmood et al., 2018; Henderson et al., 2018). For this reason, the field of automated Reinforcement Learning (AutoRL) (Parker-Holder et al., 2022) has gained popularity in recent years. AutoRL methods aim to optimize the hyperparameter selection so that RL practitioners can avoid time-consuming hyperparameter searches. AutoRL methods also seek to minimize the number of required samples to achieve reasonable performances so that RL algorithms can be used in applications where a limited number of interactions with the environment is available. AutoRL is still in its early stage of development, and most existing methods adapt techniques that have been shown to be effective for automated Machine Learning (AutoML) (Hutter et al., 2019; Falkner et al., 2018). However, RL brings additional challenges that have been overlooked till now, despite notoriously requiring special care (Igl et al., 2021). For example, due to the highly non-stationary nature of RL, there is no static selection of hyperparameters that works optimally for a given problem and

*Correspondance to: theo.vincent@dfki.de

Preprint. Under review.

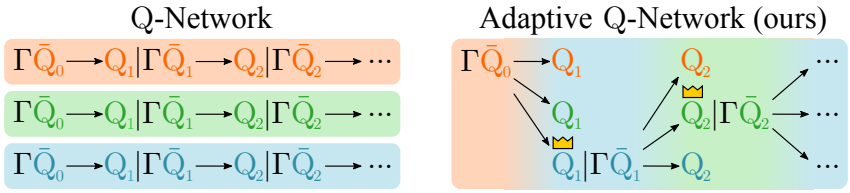

Figure 1: **Left:** Each line represents a training of *Q-Network* (QN) with different hyperparameters. **Right:** At the i^{th} target update, *Adaptive Q-Network* (AdaQN) selects the network Q_i (highlighted with a crown) that is the closest to the previous target $\Gamma\bar{Q}_{i-1}$, where Γ is the Bellman operator.

algorithm (Mohan et al., 2023). This has led to the development of several techniques for adapting the training process online to prevent issues like local minima (Nikishin et al., 2022; Sokar et al., 2023), lack of exploration (Klink et al., 2020), and catastrophic forgetting (Kirkpatrick et al., 2017).

In this work, we introduce a novel approach for AutoRL to improve the effectiveness of learning algorithms by coping with the non-stationarities of the RL optimization procedure. Our investigation stems from the intuition that the effectiveness of each hyperparameter selection changes dynamically after each training update. Building upon this observation, we propose to *adaptively* select the best-performing hyperparameters configuration to carry out Bellman updates. To highlight this idea, we call our approach *Adaptive Q-Network* (AdaQN). In practice, AdaQN uses an ensemble of Q-functions, trained with different hyperparameters, and selects the one with the smallest approximation error (Schaul et al., 2015; D'Eramo & Chalvatzaki, 2022) to be used as a shared target for each Bellman update. By doing this, our method considers several sets of hyperparameters at once each time the stationarity of the optimization problem breaks, i.e., at each target update (Figure 1). In the following, we provide theoretical motivation for our selection mechanism of AdaQN. Then, we empirically validate our approach in MuJoCo control problems, showing that AdaQN has several advantages over carrying out individual runs with static hyperparameters. Importantly, AdaQN performs better or on par with individual runs while using the same amount of samples. More interestingly, by trading off different hyperparameters configurations dynamically, AdaQN not only can match the performance of the best hyperparameter selection at no cost of additional samples, but can also reach superior overall performance than any individual run.

2 Preliminaries

We consider discounted Markov decision processes (MDPs) defined as $M = \langle S, A, P, R, \gamma \rangle$, where S and A are measurable state and action spaces, $P : S \times A \to \Delta(S)$[2] is a transition kernel, $R : S \times A \to \mathbb{R}$ is a reward function, and $\gamma \in [0, 1)$ is a discount factor (Puterman, 1990). A policy is a function $\pi : S \to \Delta(A)$, inducing an action-value function $Q^\pi(s, a) \triangleq \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a\right]$ that gives the expected discounted cumulative return executing action a in state s, following policy π thereafter. The objective is to find an optimal policy $\pi^* = \operatorname{argmax}_\pi V^\pi(\,)$, where $V^\pi(\,) = \mathbb{E}_{a \sim \pi(\,\cdot\,)}[Q^\pi(\,, a)]$. Approximate value iteration (AVI) and approximate policy iteration (API) are two paradigms to tackle this problem (Sutton & Barto, 1998). While AVI aims at estimating the optimal action-value function Q^*, i.e., the action-value function of the optimal policy, API is a two-step procedure that alternates between Approximate policy evaluation (APE), a method to evaluate the action-value function Q^π of the current policy π, and policy improvement, which updates the current policy by taking greedy actions on Q^π.

In this work, we focus on AVI and APE. Both aim to solve a Bellman equation, whose solution is Q^* for AVI and Q^π for APE. Those solutions are the fixed point of a Bellman operator Γ, where Γ is the optimal Bellman operator Γ^* for AVI, and the Bellman operator Γ^π for APE. For a Q-function Q, a state s and an action a, $\Gamma^* Q(s, a) = r + \gamma \mathbb{E}_{s' \sim \mathcal{P}(s,a)}[\max_{a'} Q(s', a')]$ and $\Gamma^\pi Q(s, a) = r + \gamma \mathbb{E}_{s' \sim \mathcal{P}(s,a), a' \sim \pi(s')}[Q(s', a')]$. Γ^* and Γ^π are γ-contraction mapping in the infinite norm. Because of this, these two methods repeatedly apply their associated Bellman operator, starting from a random Q-function. The fixed point theorem ensures that each iteration is closer to the fixed point of the respective Bellman equation. Hence, the more Bellman iterations are performed, the closer the iterated Q-function will be to the desired action-value function.

[2] $\Delta(\mathcal{X})$ denotes the set of probability measures over a set $\mathcal{X}$.

In model-free RL, Γ^* and Γ^π are approximated by their empirical version that we note $\hat{\Gamma}$ without distinguishing between AVI and APE since the nature of $\hat{\Gamma}$ can be understood from the context. We denote Θ, the space of Q-functions parameters. In common Q-Network (QN) approaches, given a *fixed* vector of parameters $\bar{\theta}$ 2 Θ, another vector of parameters θ is learned to minimize

$$L_{\text{QN}}(\theta j\bar{\theta}, s, a, r, s') = (\hat{\Gamma}_{r,s^0} Q_{\bar{\theta}}(s, a) \quad Q_\theta(s, a))^2 \tag{1}$$

for a sample (s, a, r, s'). $Q_{\bar{\theta}}$ is usually called the target Q-function because it is used to compute the target $\hat{\Gamma} Q_{\bar{\theta}}$ while Q_θ is called the online Q-function. We refer to this step as the *projection step* since the Bellman iteration $\Gamma Q_{\bar{\theta}}$ is projected back into the space of Q-functions that can be represented by our function approximations. The target parameters $\bar{\theta}$ are regularly updated to the online parameters θ so that the following Bellman iterations are learned. This step is called the *target update*. We note $\bar{\theta}_i$ the target parameters after the i^{th} target update. Importantly, when the projection step is carried out for non-linear function approximation, convergence to the fixed point is no longer guaranteed. Nevertheless, well-established theoretical results in AVI help relate the approximation error $k\Gamma^* Q_{\bar{\theta}_{i \ 1}} \quad Q_{\bar{\theta}_i} k$ at each target update i to the performance loss $kQ^* \quad Q^{\pi_N} k$, i.e., the distance between the last Q-function learned during the training and the optimal one Q^*.

Theorem 2.1. ***(Theorem 3.4 (Farahmand, 2011)).*** *Let N 2 N^*, and ρ, ν two probability measures on S A. For any sequence $(\bar{\theta}_i)_{i=0}^N$ 2 Θ^{N+1} where R_γ depends on the reward function and the discount factor, we have*

$$kQ^* \quad Q^{\pi_N} k_{1,\rho} \quad C_{N,\gamma,R_\gamma} + \inf_{r \in [0,1]} F(r; N, \rho, \gamma) \left(\sum_{i=1}^N \alpha_i^{2r} k\Gamma^* Q_{\bar{\theta}_{i \ 1}} \quad Q_{\bar{\theta}_i} k_{2,\nu}^2 \right)^{\frac{1}{2}} \tag{2}$$

where C_{N,γ,R_γ}, $F(r; N, \rho, \gamma)$, and $(\alpha_i)_{i=0}^N$ do not depend on $(\bar{\theta}_i)_{i=0}^N$. π_N is a greedy policy computed from Q_{θ_N}.

For a fixed number of Bellman iterations N, we obtain, by applying Theorem 2.1 to the sequence of parameters learned during the training $(\bar{\theta}_i)_{i=0}^N$, that the performance loss is upper bounded by a term that decreases when each approximation error decreases, i.e., each projection step improves.

Automated reinforcement learning Methods for AVI or APE (Mnih et al., 2015; Dabney et al., 2018; Haarnoja et al., 2018; Bhatt et al., 2024) are highly dependent on the chosen hyperparameters (Henderson et al., 2018; Andrychowicz et al., 2020; Engstrom et al., 2019). AutoRL addresses this issue by automatizing the search for effective hyperparameters. To mitigate the burden of non-stationarity imposed by the RL setting, some approaches consider adapting the hyperparameters during the training to better adapt to situations where specific hyperparameters are only optimal for a certain period of time during the training. Considering Theorem 2.1 w.r.t. a sequence of parameters $(\bar{\theta}_i)_{i=0}^N$, we observe that the approximation errors $(k\Gamma^* Q_{\bar{\theta}_{i \ 1}} \quad Q_{\bar{\theta}_i} k_{2,\nu})_{i=1}^N$ are the only terms of the approximation error bound on which an RL algorithm has influence. Based on this observation, we propose a new method for AutoRL that leverages Theorem 2.1 to minimize each approximation error over multiple sets of hyperparameters, hence minimizing the resulting performance loss.

3 Adaptive temporal-difference target selection

We propose a new method belonging to the AutoRL family, called Adaptive Q-Network (AdaQN), which considers K pairs of online networks trained with different sets of hyperparameters. Each online network is trained w.r.t. a shared target chosen from the set of online networks at each target update. More precisely, we consider K online vectors of parameters $(\theta^k)_{k=1}^K$ in the loss and we note $\bar{\theta}$, the parameters of the shared target network. Then, each online network parameterized by θ^k is trained to minimize its distance to $\Gamma Q_{\bar{\theta}}$. For any sample (s, a, r, s'), this results in

$$L_{\text{AdaQN}}((\theta^k)_{k=1}^K j\bar{\theta}, s, a, r, s') = \sum_{k=1}^K L_{\text{QN}}(\theta^k j\bar{\theta}, s, a, r, s'). \tag{3}$$

where the target $\bar{\theta}$ is selected at each target update according to

$$\bar{\theta} \quad \underset{\theta^k, k \in \{1,\ldots,K\}}{\text{argmin}} \sum_{(s,a,r,s^0) \in \mathcal{D}} L_{\text{QN}}(\theta^k j\bar{\theta}, s, a, r, s'), \tag{4}$$

where D is the replay buffer, thus utilizing the network that minimizes the TD-error at the current training step to compute the target shared across the online Q-networks. In the following, we provide the theoretical justification for this choice. Key to our approach is that each vector of parameters θ^k can be trained with a different optimizer, learning rate, architecture, activation function, or any other hyperparameter that only affects its training.

3.1 Theoretical motivation

We denote $\bar{\theta}_i$ the shared target Q-function after the i^{th} target update. The above-proposed selection strategy in Equation (4) addresses the goal of minimizing the sum of approximation errors $\sum_{i=1}^{N} \|\Gamma Q_{\bar{\theta}_{i-1}} - Q_{\bar{\theta}_i}\|_{2,\nu}^2$ in Equation (2), by cleverly selecting a target Q-function from a set of diverse target Q-functions. Then, the resulting bound on the performance loss would be smaller than the bound for any individual trial on a fixed set of hyperparameters. This means that, within a single training, AdaDQN would match or outperform every individual training performed from a grid search by taking all combinations of hyperparameters individually. Given $\bar{\theta}_{i-1}$, an optimal choice for $\bar{\theta}_i$ is the index of the closest online Q-function from the target $\Gamma Q_{\bar{\theta}_{i-1}}$:

$$\bar{\theta}_i = \underset{\theta^k, k \in \{1,\dots,K\}}{\operatorname{argmin}} \|\Gamma Q_{\bar{\theta}_{i-1}} - Q_{\theta^k}\|_{2,\nu}^2, \tag{5}$$

where ν is the distribution of state-action pairs in the replay buffer. This way, the sum of approximation errors $\sum_{i=1}^{N} \|\Gamma Q_{\bar{\theta}_{i-1}} - Q_{\bar{\theta}_i}\|_{2,\nu}^2$ would be the minimal achievable sum of approximation errors with the considered set of hyperparameters since at each target update i, the algorithm would select the Q-function that minimizes the i^{th} approximation error $\|\Gamma Q_{\bar{\theta}_{i-1}} - Q_{\bar{\theta}_i}\|_{2,\nu}$. Note that Equation (5) contains the true Bellman operator Γ which is not known. For this reason, our proposed selection mechanism in Equation (4) uses the empirical Bellman operator $\hat{\Gamma}$. Theorem 3.1 shows under which condition the selected index in Equation (4) is the same as the one in Equation (5). The proof, inspired by the bias-variance trade-off in supervised learning, can be found in Appendix A.

Theorem 3.1. *Let $(\theta^k)_{k=1}^K \in \Theta^K$ and $\bar{\theta} \in \Theta$ be vectors of parameters representing $K+1$ Q-functions. Let $D = \{(s,a,r,s')\}$ be a set of samples. Let ν be the distribution represented by the state-action pairs present in D. If, for every state-action pair in D, the empirical Bellman operator is an unbiased estimate of the Bellman operator, then we have*

$$\underset{k \in \{1,\dots,K\}}{\operatorname{argmin}} \sum_{(s,a,r,s') \in \mathcal{D}} L_{QN}(\theta^k, \bar{\theta}, s, a, r, s') = \underset{k \in \{1,\dots,K\}}{\operatorname{argmin}} \|\Gamma Q_{\bar{\theta}} - Q_{\theta^k}\|_{2,\nu}^2.$$

We stress that the selection strategy presented in Equation (5) is a *sufficient* condition for AdaQN to reach a sum of approximation errors that is smaller than any sum of approximation errors that an individual trial of hyperparameters would achieve. This means that a suboptimal sequence of target Q-functions can also lead to better performance than every single trial of a grid search.

3.2 Algorithmic implementation

Multiple algorithms can be derived from our formulation. Algorithm 1 shows an adaptive version of Deep Q-Network (DQN, Mnih et al. (2015)) that we call Adaptive Deep Q-Network (AdaDQN). Similarly, Adaptive Soft Actor-Critic (AdaSAC), presented in Algorithm 2, is an adaptive version of Soft Actor-Critic (SAC, Haarnoja et al. (2018)). In SAC, the target network is updated using Polyak averaging (Lillicrap et al., 2015). Therefore, we consider K target networks $(\bar{\theta}^k)_{k=1}^K$. Each target network $\bar{\theta}^k$ is updated from its respective online network θ^k, as shown in Line 11 of Algorithm 2. However, each online network is trained w.r.t. a *shared* target chosen from a single target that comes from the set of K target networks. Similarly to the strategy presented in Equation (4), the shared target network is chosen as

$$\bar{\theta} \leftarrow \bar{\theta}^{\psi} \text{ where } \psi = \underset{k \in \{1,\dots,K\}}{\operatorname{argmin}} \sum_{(s,a,r,s') \in \mathcal{D}} L_{\text{QN}}(\theta^k, \bar{\theta}, s, a, r, s'). \tag{6}$$

Having multiple online networks enables us to choose which network to use to sample actions in a value-based setting (see Line 3 of Algorithm 1) and which network is selected to train the actor in an actor-critic setting (see Line 13 of Algorithm 2). This choice is related to the behavioral policy,

Algorithm 1 Adaptive Deep Q-Network (AdaDQN). Modifications to DQN are marked in purple.

Initialize K online parameters $(\theta^k)_{k=1}^K$, and an empty replay buffer $\mathcal{D}$. Set the target parameters $\bar{\theta}$ θ^0 and the cumulative losses $L_k = 0$, for $k = 1, \ldots, K$. Set $\psi = 0$ the index to be selected for computing the target.
repeat
  Set ψ^b $\mathcal{U}\{1, \ldots, K\}$ w.p. ϵ_b and $\psi^b = \psi$ w.p. 1 ϵ_b.
  Take action a_t ϵ-greedy$(Q_{\theta^{\psi^b}}(s_t, \ ))$; Observe reward r_t, next state s_{t+1}.
  Update $\mathcal{D}$ $\mathcal{D} \bigcup \{(s_t, a_t, r_t, s_{t+1})\}$.
  every G **steps**
    Sample a mini-batch $\mathcal{B} = \{(s, a, r, s')\}$ from $\mathcal{D}$.
    Compute the *shared* target y $r + \gamma \max_{a'} Q_{\bar{\theta}}(s', a')$.
    for $k = 1, ..., K$ **do**
      Compute the loss w.r.t θ^k, $\mathcal{L}_{\text{QN}}^k = \sum_{(s,a,r,s') \in \mathcal{B}} (y \quad Q_{\theta^k}(s, a))^2$.
      Update L_k $L_k + \mathcal{L}_{\text{QN}}^k$.
      Update θ^k using its *specific* optimizer and learning rate from $\nabla_{\theta^k} \mathcal{L}_{\text{QN}}^k$.
  every T **steps**
    Update ψ $\text{argmin}_k L_k$; L_k 0, for $k \in \{1, \ldots, K\}$.
    Update the target network with $\bar{\theta}$ θ^ψ.

thus, we choose the same strategy in both settings. Intuitively, the optimal choice would be to pick the best-performing network; however, this information is not available. More importantly, only exploring using the best-performing network could lead the other online networks to learn passively, see Section 4.1. This would make the performances of the other networks drop, as identified in Ostrovski et al. (2021), making them useless for the rest of the training. Inspired by ϵ-greedy policies commonly used in RL for exploration, we choose to select a random network with probability ϵ_b and to select the online network corresponding to the selected target network as a proxy for the best-performing network with probability 1 ϵ_b. We use a linear decaying schedule for ϵ_b.

4 Experiments

We evaluate our method on lunar lander (Brockman et al., 2016) for AdaDQN, in Section 4.1. We also show experiments on the MuJoCo benchmark (Todorov et al., 2012) for AdaSAC, in Section 4.2. AdaQN automatizes the selection of hyperparameters in a very different way compared to the existing approaches in the literature (see Section 5), which makes the comparison to any autoRL method impractical. Therefore, we propose to compare AdaQN to an exhaustive grid search where both approaches have access to the same sets of hyperparameters. We report the performances of *every* single set of hyperparameters given as input to AdaQN. This allows us to see how AdaQN performs compared to the best static set of hyperparameters. Additionally, this work focuses on sample efficiency, this is why we report the performances of the agent by counting every environment interaction used for the meta-optimization process as advocated by Franke et al. (2021). For reporting the grid search results, this translates to multiplying the number of interactions of the best achieved performance by the number of individual trials. We also report the average performance over the individual runs as the performance that a random search would obtain in expectation. The remaining hyperparameters are kept unchanged. Table 1 and 2 of Appendix B reference all the hyperparameters used for the experiments. The code is based on the Stable Baselines implementation (Raffin et al., 2021). It is available in the supplementary material and will be made open source upon acceptance.

Performance metric. As recommended by Agarwal et al. (2021), we plot the interquartile mean (IQM) along with shaded regions showing pointwise 95% percentile stratified bootstrap confidence intervals. IQM is a trade-off between the mean and the median where the tail of the score distribution is removed on both sides to consider only 50% of the runs. We argue that the final score is not enough to properly compare RL algorithms since methods that show higher initial performances are better suited for real-world experiments compared to methods that only show higher performances later during training. This is why we also analyze the performances with the Area Under the Curve (AUC) that computes the integral of the IQM along the training. We also report the worst-performing seed to

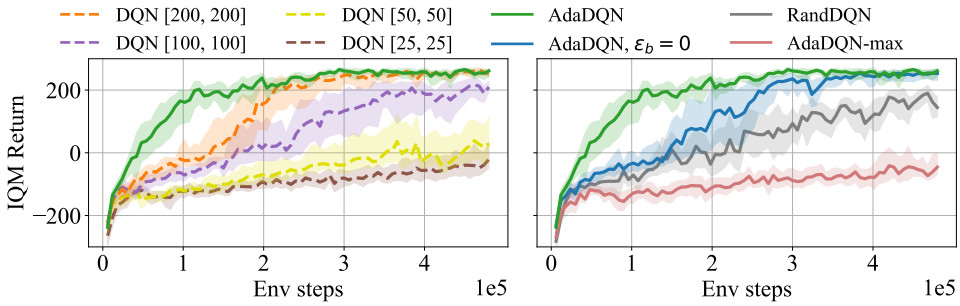

Figure 2: On-the-fly architecture selection on **Lunar Lander**. All architectures contain two hidden layers. The number of neurons in each layer is indicated in the legend. **Left:** AdaDQN yields a better AUC than every DQN run. **Right:** Ablation on the behavioral policy and on the strategy to select the target network used to compute the target. Each version of AdaDQN uses the 4 presented architectures. The strategy presented in Equation (4) outperforms the other considered strategies.

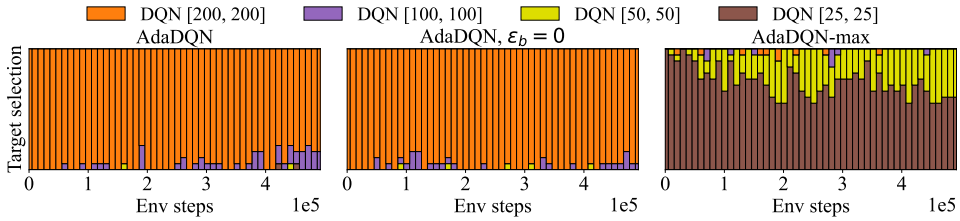

Figure 3: Bar plots of the distribution of networks selected as target networks across all seeds. **Left:** In most cases, AdaDQN selects the best individual architecture. **Middle:** AdaDQN with $\epsilon_b = 0$ also focuses on the best individual architecture. The fact that it underperforms compared to AdaDQN is coming from the behavioral policy. **Right:** AdaDQN-max is a version of AdaDQN where the minimum operator is replaced by the maximum operator for selecting the following target network.

analyze the robustness of AdaQN w.r.t. stochasticity. The IQM is computed over 20 seeds for the Lunar Lander experiments and over 9 seeds and 6 environments for the MuJoCo experiments.

4.1 A proof of concept

We consider 4 different architectures for AdaDQN and compare their performances to the individual runs in Figure 2 (left). The 4 architectures are composed of two hidden layers, with the number of neurons forming each layer indicated in the legend. Interestingly, AdaDQN performs better than the best individual architecture, meaning that by selecting different architectures during the training, AdaDQN better copes with the non-stationarities of the optimization procedure. Figure 3 shows the distribution of the selected target (see Line 14 of Algorithm 1) along the training across all seeds. Importantly, the selected target is not always computed from the same target network. For example, the network DQN $[100, 100]$ is sometimes selected instead of DQN $[200, 200]$. This result is even more interesting because, intuitively, AdaDQN should always select the biggest network (DQN $[200, 200]$) since it greatly outperforms the other networks.
To better understand this, Figure 2 (right) shows the performances for AdaDQN along with a version of AdaQN where ϵ_b is equal to 0 during the training (AdaDQN $\epsilon_b = 0$). This variant of AdaQN performs similarly to the best individual run but underperforms compared to AdaDQN. Despite selecting the target networks in a similar way as AdaDQN (see Figure 3 (middle)), AdaDQN $\epsilon_b = 0$ leaves only a few online transitions to the other networks. As explained in Section 3.2, this shows the benefit of allowing each online Q-function to interact with the environment ($\epsilon_b > 0$) such that they do not learn passively. Additionally, we show a version where the target network is selected randomly from the set of online networks, calling this variant RandDQN. This version is similar to the way the target is computed in REDQ (Chen et al., 2020). While sampling uniformly from similar agents yields better performance, as shown in Chen et al. (2020), this strategy suffers when one agent is not performing well. In our case, DQN $[25, 25]$ harms the overall performance. Finally, taking the maximum instead of the minimum to select the target (AdaDQN-max) performs as badly as the worst

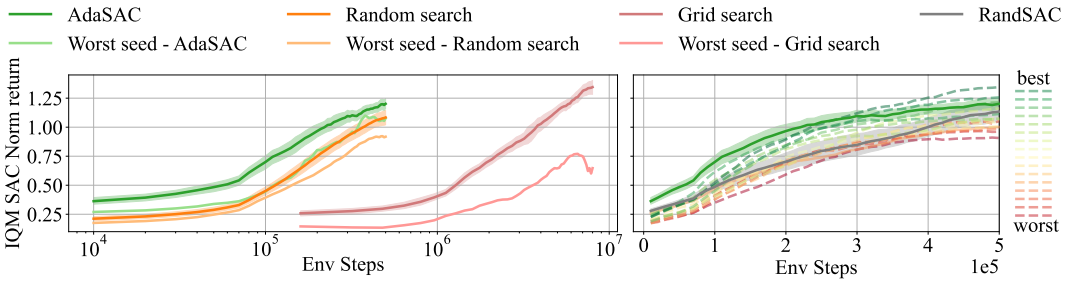

Figure 4: On-the-fly hyperparameter selection on **MuJoCo**. The 16 sets of hyperparameters are the elements of the cartesian product between the learning rates $f0.0005, 0.001g$, the optimizers fAdam, RMSProp$g$, the critic's architectures $f[256, 256], [512, 512]g$ and the activation functions fReLU, Sigmoid$g$. **Left:** AdaDQN is more sample-efficient than grid search. **Right:** AdaDQN yields a better AUC than every DQN run while having a greater final score than 13 out of 16 DQN runs. The color shading of the dashed lines is used to indicate their ranking for the AUC metric.

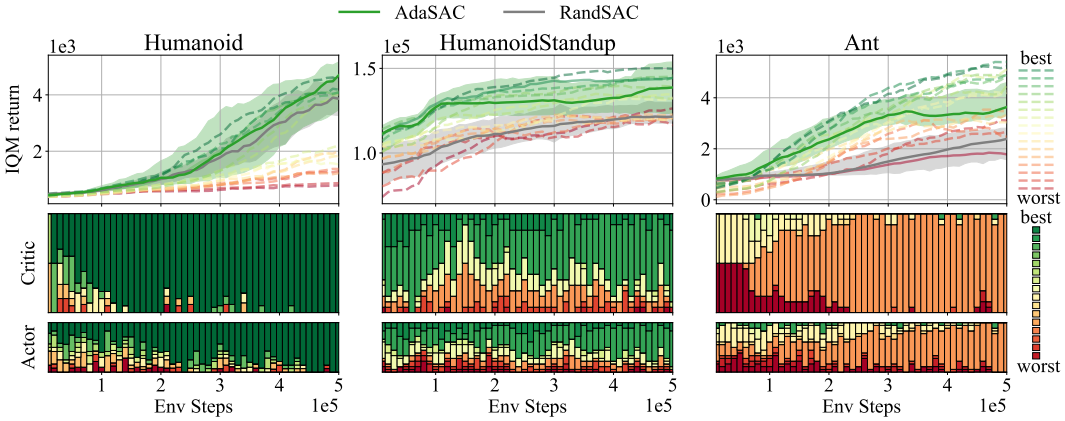

Figure 5: **Top:** Per environment return of AdaSAC and RandSAC when the 16 sets of hyperparameters, described in Figure 4, are given as input. **Bottom:** Bar plot presenting the distribution of networks selected for the critic's loss and the actor's loss along the training. AdaSAC outperforms RandSAC and most individual run by designing non-trivial hyperparameter schedules.

available agent (DQN $[25, 25]$). Figure 3 (bottom) shows that the worst-performing agent is almost always selected, in line with the theoretical analysis developed in Section 3.1.

4.2 Continuous control: MuJoCo environments

We evaluate AdaSAC for a wider choice of hyperparameters on 6 MuJoCo environments. We consider selecting from 16 sets of hyperparameters. Those sets form the Cartesian product between the learning rates $f0.0005, 0.001g$, the optimizers fAdam, RMSProp$g$, the critic's architectures $f[256, 256], [512, 512]g$ and the activation functions fReLU, Sigmoid$g$. This fixes K to $2^4 = 16$. The values of the hyperparameters were chosen to be representative of common choices made when tuning by hand a SAC agent. In Figure 4 (left), we compare AdaSAC with a grid search performed on the 16 set of hyperparameters. AdaSAC is an order of magnitude more sample efficient than grid search. Notably, AdaSAC's worst-performing seed performs on par with the random search approach while greatly outperforming the worst-performing seed of the grid search. In Figure 4 (right), we show AdaSAC's performance along with the individual performance of each set of hyperparameters. AdaSAC yields the highest AUC metric while outperforming 13 out of the 16 individual runs in terms of final performance. Interestingly, AdaSAC reaches the performance of vanilla SAC in less than half of the samples. We also show the performance of RandSAC, for which a random target network is selected to compute the target instead of following the strategy presented in Equation (4). For clarity, the labels of the 16 individual runs are not shown, they are available in Figure 7 of Appendix D.1.

We now analyze the proposed strategy for selecting the networks during the training of AdaSAC. The top row of Figure 5 shows the scores obtained for 3 MuJoCo environments. In the bottom row,

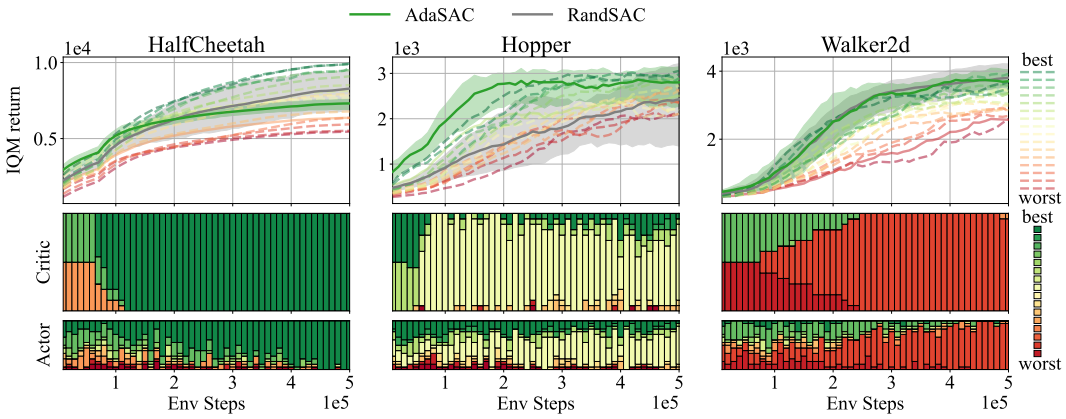

Figure 6: **Top:** Per environment return of AdaSAC and RandSAC when the 16 sets of hyperparameters, described in Figure 4, are given as input. **Bottom:** Bar plot presenting the distribution of networks selected for the critic's loss and the actor's loss along the training.

we plot the distribution across the seeds of the target networks selected to compute the critic's loss and the actor's loss (see Line 12 and 13 of Algorithm 2). Figure 6 shows a similar plot for the 3 remaining MuJoCo environments. Overall, the selected networks are changing during the training, which indicates that the loss is not always minimized by the same set of hyperparameters. This supports the idea of selecting the target network based on Equation (4). Furthermore, the selected networks are not the same across the different environments. This shows that AdaSAC designs non-trivial hyperparameter schedules that cannot be handcrafted. On Humanoid, selecting the network corresponding to the best-performing agent leads to similar performances to the best-performing agents when trained individually. On HumanoidStandup, the selection strategy avoids selecting the worst-performing agents. This is why AdaSAC outperforms RandSAC, which blindly selects agents. A similar scenario happens on Ant. HalfCheetah is the only environment where RandSAC slightly outperforms AdaSAC. AdaSAC still yields a better final performance than 6 of the individual runs. Finally, on Hopper and Walker2d, AdaSAC outperforms every individual run for the AUC metric by mainly selecting hyperparameters that are *not* performing well when trained individually. The ability of AdaSAC to adapt the hyperparameters at each target update allows the proposed algorithm to better fit the targets, which yields better performances. The distribution of the selected online networks in the actor's loss, shown at the bottom of Figures 5 and 6, are chosen uniformly at the beginning since ϵ_b starts with a high value. At the end of the training, the distribution of the selected online networks is similar to the distribution of selected target networks in the critic's loss since ϵ_b is low at the end of the training.

Ablations. We evaluate AdaSAC against a grid search in 4 different scenarios to show that it is robust to variations in the hyperparameters. All the figures are shown in Appendices D.2, D.3, D.4, and D.5. We first give 4 architectures of different capacities as input. Then, we run AdaSAC with 3 different learning rates, and we also run another study where AdaSAC is given 3 different optimizers. In each case, AdaSAC matches the best-performing individual runs while successfully ignoring the worst-performing sets of hyperparameters. Finally, we evaluate AdaSAC with 3 different activation functions given as input. This time, AdaSAC outperforms 2 out of the 3 individual runs. Interestingly, RandSAC seems to suffer more in that setting by only outperforming the worst-performing individual run. This further demonstrates that the strategy presented in Equation (4) is effective.

5 Related work

We follow the clustering of AutoRL methods presented in Parker-Holder et al. (2022) to position our work in the AutoRL landscape. Contrary to AdaQN, many approaches consider optimizing the hyperparameters through multiple trials. A classic approach is to cover the search space with a grid search or a random search (Hutter et al., 2019; Bergstra & Bengio, 2012). Some other methods use Bayesian optimization to guide the search in the space of hyperparameters (Chen et al., 2018; Falkner et al., 2018; Nguyen et al., 2020; Shala et al., 2022), leveraging the information collected from individual trials. Those methods consider the problem of finding the best set of hyperparameters but do not consider changing the hyperparameters during a single trial, which would be more appropriate

for handling the non-stationarities inherent to the RL problem. Evolutionary approaches have been developed for that purpose (Stanley et al., 2009; Awad et al., 2021; Jaderberg et al., 2019), where the best elements of a population of agents undergo genetic modifications during training. As an example, in SEARL (Franke et al., 2021), a population of agents is first evaluated in the environment. Then, the best-performing agents are selected, and a mutation process is performed to form the next generation of agents. Finally, the samples collected during evaluation are taken from a shared replay buffer to train each individual agent, and the loop repeats itself. Even if selecting agents based on their performance in the environment seems reasonable, we argue that it hinders sample-efficiency since exploration in the space of hyperparameters might lead to poorly performing agents. Moreover, if poorly performing agents are evaluated, low-quality samples will then be stored in the replay buffer and used later for training, which could lead to long-term negative effects. This is why we propose to base the selection mechanism on the approximation error, which does not require additional interaction with the environment. Other methods have considered evaluating the Q-functions offline (Tang & Wiens, 2021). Most approaches consider the Bellman error $k\Gamma Q_i \quad Q_i k$ instead of the approximation error $k\Gamma Q_{i-1} \quad Q_i k$, where Q_i is the Q-function obtained after i Bellman iterations (Farahmand & Szepesvári, 2011; Zitovsky et al., 2023). However, this approach considers the empirical estimate of the Bellman error which is a biased estimate of the true Bellman error (Baird, 1995). Lee et al. (2022) propose to rely on the approximation error to choose between two different classes of functions in an algorithm called ModBE. AdaQN differs from ModBE since ModBE does not adapt the hyperparameters at each Bellman iteration but instead performs several training steps before comparing two classes of functions. We argue that the approximation error is a natural metric in our specific setting, since choosing the best-performing Q-function or the Q-function minimizing its Bellman error would not guarantee that the bound presented in Theorem 2.1 is minimized at the end of the training or, in other words, that the future Bellman iterations would lead to well-performing agents.

Similarly to AdaQN, Meta-Gradient methods are optimizing the hyperparameters in a single training (Finn et al., 2017; Zahavy et al., 2020; Flennerhag et al., 2021). Our approach differs from Meta-Gradient methods because we do not require the hyperparameters to be differentiable. This is why we can consider different optimizers or different activation functions. Finally, the cluster of methods under the name "Blackbox Online Tuning" is closer to our approach. However, most methods in this cluster focus on the behavioral policy (Schaul et al., 2019; Badia et al., 2020) or build the hyperparameters as functions of the state of the environment (Sutton & Singh, 1994; White & White, 2016). For example, Riquelme et al. (2019) develop a method called adaptive TD that selects between the TD update or the Monte-Carlo update depending on whether the TD update belongs to a confidence interval computed from several Monte-Carlo estimates.

6 Discussion and conclusion

We have presented Adaptive Q-Network (AdaQN), a new method that selects the hyperparameters during learning by training diverse Q-networks w.r.t. a shared target, selected as the target network corresponding to the most accurate online network. We demonstrated that AdaQN is theoretically sound and we devised its algorithmic implementation. By adaptively selecting from a set of hyperparameters, AdaQN achieves strong performance against individual runs in terms of sample efficiency, final performance, and robustness.

We can identify some limitations of our work and suggest ways to tackle them. Our work focuses on optimizing the agent's hyperparameters; thus, the environment's hyperparameters, such as the reward or the discount factors, cannot be optimized with AdaQN. Nevertheless, an off-the-shelf AutoRL algorithm can be used in combination with AdaQN to optimize those hyperparameters. To go beyond the fixed set of hyperparameters, one could consider extending our work to a population-based approach in a similar way as Franke et al. (2021) do it but, this time, a new population of online networks could be generated starting from the agents having the lowest losses w.r.t. a shared target. Moreover, since AdaQN considers multiple Q-functions in the loss, the training time and memory requirements increase. We provide an extensive study in Appendix C to show that this increase remains reasonable compared to the gain in performance and sample-efficiency. We stress that, in theory, the computations can be parallelized so that the adaptive version of an algorithm requires the same amount of time as its original algorithm. The additional "for loop" in Line 9 of Algorithm 1 and the one in Line 7 of Algorithm 2 can be run in parallel as long as enough parallel processing capacity is available, as it is common in modern GPUs.

7 Acknowledgments

This work was funded by the German Federal Ministry of Education and Research (BMBF) (Project: 01IS22078). This work was also funded by Hessian.ai through the project 'The Third Wave of Artificial Intelligence – 3AI' by the Ministry for Science and Arts of the state of Hessen and by the grant "Einrichtung eines Labors des Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) an der Technischen Universität Darmstadt".

References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, 2021.

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., et al. What matters for on-policy deep actor-critic methods? a large-scale study. In *International Conference on Learning Representations*, 2020.

Awad, N., Mallik, N., and Hutter, F. Dehb: Evolutionary hyberband for scalable, robust and efficient hyperparameter optimization. In *International Joint Conference on Artificial Intelligence*, 2021.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning, ICML, 13-18 July, Virtual Event*, 2020.

Baird, L. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning*, 1995.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. In *Journal of Machine Learning Research*, 2012.

Bhatt, A., Palenicek, D., Belousov, B., Argus, M., Amiranashvili, A., Brox, T., and Peters, J. Crossq: Batch normalization in deep reinforcement learning for greater sample efficiency and simplicity. In *International Conference on Learning Representations*, 2024.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. In *arXiv preprint arXiv:1606.01540*, 2016.

Chen, X., Wang, C., Zhou, Z., and Ross, K. W. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2020.

Chen, Y., Huang, A., Wang, Z., Antonoglou, I., Schrittwieser, J., Silver, D., and de Freitas, N. Bayesian optimization in alphago. In *arXiv preprint arXiv:1812.06855*, 2018.

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. Implicit quantile networks for distributional reinforcement learning. In *International Conference on Machine Learning*, 2018.

D'Eramo, C. and Chalvatzaki, G. Prioritized sampling with intrinsic motivation in multi-task reinforcement learning. In *International Joint Conference on Neural Networks*, 2022.

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. Implementation matters in deep rl: A case study on ppo and trpo. In *International Conference on Learning Representations*, 2019.

Falkner, S., Klein, A., and Hutter, F. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, 2018.

Farahmand, A.-m. Regularization in reinforcement learning. 2011.

Farahmand, A.-m. and Szepesvári, C. Model selection in reinforcement learning. In *Machine learning*, 2011.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

Flennerhag, S., Schroecker, Y., Zahavy, T., van Hasselt, H., Silver, D., and Singh, S. Bootstrapped meta-learning. In *International Conference on Learning Representations*, 2021.

Franke, J. K., Koehler, G., Biedenkapp, A., and Hutter, F. Sample-efficient automated deep reinforcement learning. In *International Conference on Learning Representations*, 2021.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In *Association for the Advancement of Artificial Intelligence*, 2018.

Hutter, F., Kotthoff, L., and Vanschoren, J. (eds.). *Automated Machine Learning - Methods, Systems, Challenges*. Springer, 2019.

Igl, M., Farquhar, G., Luketina, J., Böhmer, J., and Whiteson, S. Transient non-stationarity and generalisation in deep reinforcement learning. In *International Conference on Learning Representations*, 2021.

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castañeda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J. Z., Silver, D., Hassabis, D., Kavukcuoglu, K., and Graepel, T. Human-level performance in 3d multiplayer games with population-based reinforcement learning. In *Science*, 2019.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. In *National Academy of sciences*, 2017.

Klink, P., D'Eramo, C., Peters, J. R., and Pajarinen, J. Self-paced deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

Lee, J. N., Tucker, G., Nachum, O., Dai, B., and Brunskill, E. Oracle inequalities for model selection in offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2022.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In *arXiv preprint arXiv:1509.02971*, 2015.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., and Bergstra, J. Benchmarking reinforcement learning algorithms on real-world robots. In *Conference on Robot Learning*, 2018.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. In *Nature*, 2015.

Mohan, A., Benjamins, C., Wienecke, K., Dockhorn, A., and Lindauer, M. Autorl hyperparameter landscapes. In *International Conference on Automated Machine Learning*, 2023.

Nguyen, V., Schulze, S., and Osborne, M. A. Bayesian optimization for iterative learning. In *Advances in Neural Information Processing Systems*, 2020.

Nikishin, E., Schwarzer, M., D'Oro, P., Bacon, P.-L., and Courville, A. The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning*, 2022.

Ostrovski, G., Castro, P. S., and Dabney, W. The difficulty of passive learning in deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2021.

Parker-Holder, J., Rajan, R., Song, X., Biedenkapp, A., Miao, Y., Eimer, T., Zhang, B., Nguyen, V., Calandra, R., Faust, A., et al. Automated reinforcement learning (autorl): A survey and open problems. In *Journal of Artificial Intelligence Research*, 2022.

Puterman, M. L. Markov decision processes. *Handbooks in Operations Research and Management Science*, 1990.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. In *Journal of Machine Learning Research*, 2021.

Riquelme, C., Penedones, H., Vincent, D., Maennel, H., Gelly, S., Mann, T. A., Barreto, A., and Neu, G. Adaptive temporal-difference learning for policy evaluation with per-state uncertainty estimates. In *Advances in Neural Information Processing Systems*, 2019.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. In *arXiv preprint arXiv:1511.05952*, 2015.

Schaul, T., Borsa, D., Ding, D., Szepesvari, D., Ostrovski, G., Dabney, W., and Osindero, S. Adapting behaviour for learning progress. In *arXiv preprint arXiv:1912.06910*, 2019.

Shala, G., Biedenkapp, A., Hutter, F., and Grabocka, J. Gray-box gaussian processes for automated reinforcement learning. In *International Conference on Learning Representations*, 2022.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. In *Nature*, 2017.

Sokar, G., Agarwal, R., Castro, P. S., and Evci, U. The dormant neuron phenomenon in deep reinforcement learning. In *International Conference on Machine Learning*, 2023.

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. A hypercube-based encoding for evolving large-scale neural networks. In *Artificial Life*, 2009.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT Press, 1998.

Sutton, R. S. and Singh, S. P. On step-size and bias in temporal-difference learning. In *Yale Workshop on Adaptive and Learning Systems*, 1994.

Tang, S. and Wiens, J. Model selection for offline reinforcement learning: Practical considerations for healthcare settings. In *Machine Learning for Healthcare Conference*, 2021.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, 2012.

White, M. and White, A. A greedy approach to adapting the trace parameter for temporal difference learning. In *International Conference on Autonomous Agents and Multiagent Systems*, 2016.

Zahavy, T., Xu, Z., Veeriah, V., Hessel, M., Oh, J., van Hasselt, H., Silver, D., and Singh, S. A self-tuning actor-critic algorithm. In *Advances in Neural Information Processing Systems*, 2020.

Zitovsky, J. P., De Marchi, D., Agarwal, R., and Kosorok, M. R. Revisiting bellman errors for offline model selection. In *International Conference on Machine Learning*, 2023.

A Proofs

Theorem A.1. *Let $(\theta^k)_{k=1}^K \in \Theta^K$ and $\bar{\theta} \in \Theta$ be vectors of parameters representing $K+1$ Q-functions. Let $D = \{(s,a,r,s')\}$ be a set of samples. Let ν be the distribution represented by the state-action pairs present in D. If, for every state-action pair in D, the empirical Bellman operator is an unbiased estimate of the Bellman operator, then we have*

$$\operatorname*{argmin}_{k\in\{1,\dots,K\}} \sum_{(s,a,r,s')\in\mathcal{D}} L_{QN}(\theta^k|\bar{\theta}, s, a, r, s') = \operatorname*{argmin}_{k\in\{1,\dots,K\}} \|\Gamma Q_{\bar{\theta}} - Q_{\theta^k}\|_{2,\nu}^2.$$

Proof. Let $(\theta^k)_{k=1}^K \in \Theta^K$ and $\bar{\theta} \in \Theta$ be vectors of parameters representing $K+1$ Q-functions. Let $D = \{(s,a,r,s')\}$ be a set of samples. Let ν be the distribution of the state-action pairs present in D. We assume that for every state-action pair in D, the empirical Bellman operator is an unbiased estimate of the Bellman operator. For every state-action pair (s,a) in D, we define the set $D_{s,a} = \{(r,s') | (s,a,r,s') \in D\}$. We assume that for any $\bar{\theta} \in \Theta$, $\mathbb{E}_{(r,s')\sim\mathcal{D}_{s,a}}[\hat{\Gamma}_{r,s'}Q_{\bar{\theta}}(s,a)] = \Gamma Q_{\bar{\theta}}(s,a)$. Additionally, we note M the cardinality of D, $M_{s,a}$ the cardinality of $D_{s,a}$ and $\mathring{D}$ the set of unique state-action pairs in D. We take $k \in \{1,\dots,K\}$ and write

$$\begin{aligned}
\sum_{(s,a,r,s')\in\mathcal{D}} L_{\text{QN}}(\theta^k|\bar{\theta}, s, a, r, s') &= \sum_{(s,a,r,s')\in\mathcal{D}} \left(\hat{\Gamma}_{r,s'}Q_{\bar{\theta}}(s,a) - Q_{\theta^k}(s,a)\right)^2 \\
&= \sum_{(s,a,r,s')\in\mathcal{D}} \left(\hat{\Gamma}_{r,s'}Q_{\bar{\theta}}(s,a) - \Gamma Q_{\bar{\theta}}(s,a) + \Gamma Q_{\bar{\theta}}(s,a) - Q_{\theta^k}(s,a)\right)^2 \\
&= \sum_{(s,a,r,s')\in\mathcal{D}} \left(\hat{\Gamma}_{r,s'}Q_{\bar{\theta}}(s,a) - \Gamma Q_{\bar{\theta}}(s,a)\right)^2 \\
&+ \sum_{(s,a,r,s')\in\mathcal{D}} \left(\Gamma Q_{\bar{\theta}}(s,a) - Q_{\bar{\theta}^k}(s,a)\right)^2 + 2 \sum_{(s,a,r,s')\in\mathcal{D}} \left(\hat{\Gamma}_{r,s'}Q_{\bar{\theta}}(s,a) - \Gamma Q_{\bar{\theta}}(s,a)\right)\left(\Gamma Q_{\bar{\theta}}(s,a) - Q_{\theta^k}(s,a)\right).
\end{aligned}$$

The second last equation is obtained by introducing the term $\Gamma Q_{\bar{\theta}}(s,a)$ and removing it. The last equation is obtained by developing the previous squared term. Now, we study each of the three terms:

- $\sum_{(s,a,r,s')\in\mathcal{D}} \left(\hat{\Gamma}_{r,s'}Q_{\bar{\theta}}(s,a) - \Gamma Q_{\bar{\theta}}(s,a)\right)^2$ is independent of θ^k
- $\sum_{(s,a,r,s')\in\mathcal{D}} \left(\Gamma Q_{\bar{\theta}}(s,a) - Q_{\theta^k}(s,a)\right)^2$ equal to $M \|\Gamma Q_{\bar{\theta}} - Q_{\theta^k}\|_{2,\nu}^2$ by definition of ν.
- And finally,

  $$\sum_{(s,a,r,s')\in\mathcal{D}} \left(\hat{\Gamma}_{r,s'}Q_{\bar{\theta}}(s,a) - \Gamma Q_{\bar{\theta}}(s,a)\right)\left(\Gamma Q_{\bar{\theta}}(s,a) - Q_{\theta^k}(s,a)\right)$$

  $$= \sum_{(s,a)\in\mathring{\mathcal{D}}} \left[\sum_{(r,s')\in\mathcal{D}_{s,a}} \left(\hat{\Gamma}_{r,s'}Q_{\bar{\theta}}(s,a) - \Gamma Q_{\bar{\theta}}(s,a)\right)\left(\Gamma Q_{\bar{\theta}}(s,a) - Q_{\theta^k}(s,a)\right)\right] = 0$$

  since, for every $(s,a) \in \mathring{D}$,

  $$\sum_{(r,s')\in\mathcal{D}_{s,a}} \left(\hat{\Gamma}_{r,s'}Q_{\bar{\theta}}(s,a) - \Gamma Q_{\bar{\theta}}(s,a)\right) = M_{s,a}\mathbb{E}_{(r,s')\sim\mathcal{D}_{s,a}}[\hat{\Gamma}_{r,s'}Q_{\bar{\theta}}(s,a) - \Gamma Q_{\bar{\theta}}(s,a)] = 0,$$

  the last equality holds from the assumption.

Thus, we have

$$\sum_{(s,a,r,s')\in\mathcal{D}} L_{\text{QN}}(\theta^k|\bar{\theta}, s, a, r, s') = \text{constant w.r.t } \theta^k + M \|\Gamma Q_{\bar{\theta}} - Q_{\theta^k}\|_{2,\nu}^2$$

This is why

$$\operatorname*{argmin}_{k\in\{1,\dots,K\}} \sum_{(s,a,r,s')\in\mathcal{D}} L_{\text{QN}}(\theta^k|\bar{\theta}, s, a, r, s') = \operatorname*{argmin}_{k\in\{1,\dots,K\}} \|\Gamma Q_{\bar{\theta}} - Q_{\theta^k}\|_{2,\nu}^2.$$

□

B Algorithms and Hyperparameters

Algorithm 2 Adaptive Soft Actor-Critic (AdaSAC). Modifications to SAC are marked in purple.

Initialize the policy parameters ϕ, K online parameters $(\theta^k)_{k=1}^K$, and an empty replay buffer $\mathcal{D}$.
For $k = 1, \ldots, K$, set the target parameters $\bar{\theta}^k \quad \theta^k$ and the cumulative losses $L_k = 0$. Set $\psi_1 = 0$ and $\psi_2 = 1$ the indices to be selected for computing the target.
repeat
  Take action $a_t \quad \pi_\phi(\cdot|s_t)$; Observe reward r_t, next state s_{t+1}; $\mathcal{D} \quad \mathcal{D} \bigcup \{(s_t, a_t, r_t, s_{t+1})\}$.
  for UTD updates **do**
    Sample a mini-batch $\mathcal{B} = \{(s, a, r, s')\}$ from $\mathcal{D}$.
    Compute the *shared* target

$$y \quad r + \gamma \left(\min_{k \in \{\psi_1, \psi_2\}} Q_{\bar{\theta}^k}(s', a') \quad \alpha \log \pi_\phi(a'|s') \right), \text{where } a' \quad \pi_\phi(\cdot|s').$$

    for $k = 1, ..., K$ **do**
      Compute the loss w.r.t θ^k, $L_{\text{QN}}^k = \sum_{(s,a,r,s') \in \mathcal{B}} (y \quad Q_{\theta^k}(s, a))^2$.
      Update $L_k \quad (1 \quad \tau) L_k + \tau L_{\text{QN}}^k$.
      Update θ^k using its *specific* optimizer and learning rate from $\nabla_{\theta^k} L_{\text{QN}}^k$.
      Update the target networks with $\bar{\theta}^k \quad \tau \theta^k + (1 \quad \tau) \bar{\theta}^k$.
    Set ψ_1 and ψ_2 to be the indexes of the 2 lowest values of L.
  Set $(\psi_1^b, \psi_2^b) \quad \text{Distinct}\{1, \ldots, K\}$ w.p ϵ_b and $(\psi_1^b, \psi_2^b) = (\psi_1, \psi_2)$ w.p $1 \quad \epsilon_b$.
  Update ϕ with gradient ascent using the loss

$$\min_{k \in \{\psi_1^b, \psi_2^b\}} Q_{\theta^k}(s, a) \quad \alpha \log \pi_\phi(a|s), \quad a \quad \pi_\phi(\cdot|s)$$

Table 1: Summary of all hyperparameters used for the Lunar Lander experiments.

Shared across all algorithms	
Discount factor γ	0.99
Initial replay buffer size	1000
Replay buffer size	10^4
Batch Size	32
Target update frequency T	200
Training frequency G	1
Activation function	ReLU
Learning rate	3 10^{-4}
Optimizer	Adam
Starting ϵ	1
Ending ϵ	0.01
Linear decay duration of ϵ	1000
AdaDQN	
Starting ϵ_b	1
Ending ϵ_b	0.01
Linear decay duration of ϵ_b	Until end

Table 2: Summary of all hyperparameters used for the MuJoCo experiments.

Shared across all algorithms	
Discount factor γ	0.99
Initial replay buffer size	5000
Replay buffer size	10^6
Batch Size	256
Target update rate τ	0.005
UTD	1
Policy delay	1
Actor's architecture	$256, 256$
Actor's activation function	ReLU
Actor's learning rate	0.001
Actor's optimizer	Adam
Vanilla SAC	
Critic's architecture	$256, 256$
Critic's activation function	ReLU
Critic's learning rate	0.001
Critic's optimizer	Adam
AdaSAC	
Starting ϵ_b	1
Ending ϵ_b	0.01
Linear decay duration of ϵ_b	Until end

C Training time and memory requirements

Table 3: Final performances, memory requirements, and training time of AdaSAC compared to the average individual run. Computations are made on an NVIDIA GeForce RTX 4090 Ti.

Sets of *hyperparameters* given as input to AdaSAC	Additional GPU vRAM usage	Additional training time
16 sets of *hyperparameters* described in Figure 4	300 Mb	352%
4 *architectures* described in Figure 8	$<$ 100 Mb	72%
3 *learning rates* described in Figure 10	$<$ 100 Mb	43%
3 *optimizers* described in Figure 12	$<$ 100 Mb	43%
3 *activation functions* described in Figure 14	$<$ 100 Mb	43%

D Additional Experiments

D.1 On-The-Fly Hyperparameter Selection on MuJoCo

Table 4: Number of individual runs outperformed by AdaSAC and RandSAC for the Area Under the Curve (AUC) and Final Performance (FP) metrics, when the 16 sets of hyperparameters, described in Figure 4, are given as input. AdaSAC outperforms every individual run for the AUC metric. Only 3 individual runs reach a higher final performance than AdaSAC's policy. Unlike AdaSAC, RandSAC randomly selects a target at each target update, which leads to poorer performance than AdaSAC on both metrics.

		Hopper	Ant	HalfCheetah	Walker2d	Humanoid	HumanoidStandup	All
AdaSAC	AUC	**16**	8	7	**16**	12	11	**16**
	FP	11	7	6	15	**16**	11	13
RandSAC	AUC	6	1	8	**16**	12	4	9
	FP	7	1	10	15	11	5	12

- SAC learning rate: 0.001, optimizer: Adam, architecture: 512, 512, activation fn: ReLU
- SAC learning rate: 0.0005, optimizer: Adam, architecture: 256, 256, activation fn: ReLU
- SAC learning rate: 0.001, optimizer: Adam, architecture: 512, 512, activation fn: Sigmoid
- SAC learning rate: 0.0005, optimizer: Adam, architecture: 512, 512, activation fn: ReLU
- SAC learning rate: 0.001, optimizer: Adam, architecture: 256, 256, activation fn: Sigmoid
- SAC learning rate: 0.0005, optimizer: Adam, architecture: 512, 512, activation fn: Sigmoid
- SAC learning rate: 0.0005, optimizer: RMSProp, architecture: 256, 256, activation fn: ReLU
- SAC learning rate: 0.0005, optimizer: RMSProp, architecture: 512, 512, activation fn: ReLU
- SAC learning rate: 0.001, optimizer: RMSProp, architecture: 512, 512, activation fn: Sigmoid
- SAC learning rate: 0.0005, optimizer: Adam, architecture: 256, 256, activation fn: Sigmoid
- SAC learning rate: 0.001, optimizer: RMSProp, architecture: 256, 256, activation fn: ReLU
- vanilla SAC learning rate: 0.001, optimizer: Adam, architecture: 256, 256, activation fn: ReLU
- SAC learning rate: 0.001, optimizer: RMSProp, architecture: 256, 256, activation fn: Sigmoid
- SAC learning rate: 0.001, optimizer: RMSProp, architecture: 512, 512, activation fn: ReLU
- SAC learning rate: 0.0005, optimizer: RMSProp, architecture: 512, 512, activation fn: Sigmoid
- SAC learning rate: 0.0005, optimizer: RMSProp, architecture: 256, 256, activation fn: Sigmoid

Figure 7: Legend of Figure 4 showing the ranking by AUC of the 16 considered sets of hyperparameters described. Importantly, the ranking changes for each MuJoCo environment.

D.2 On-The-Fly Architecture Selection on MuJoCo

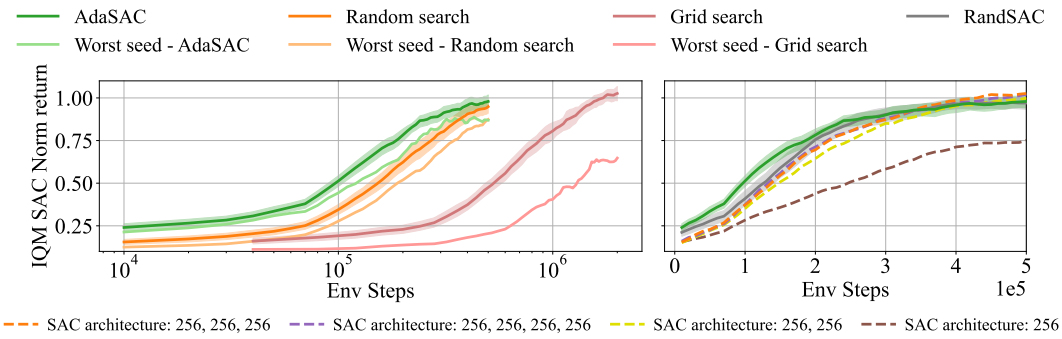

Figure 8: On-the-fly architecture selection on **MuJoCo**. All architectures contain hidden layers with 256 neurons. The architectures are indicated in the legend. **Left:** AdaSAC is more sample-efficient than grid search. **Right:** AdaSAC yields similar performances as the best-performing architectures. RandSAC and AdaSAC are on par.

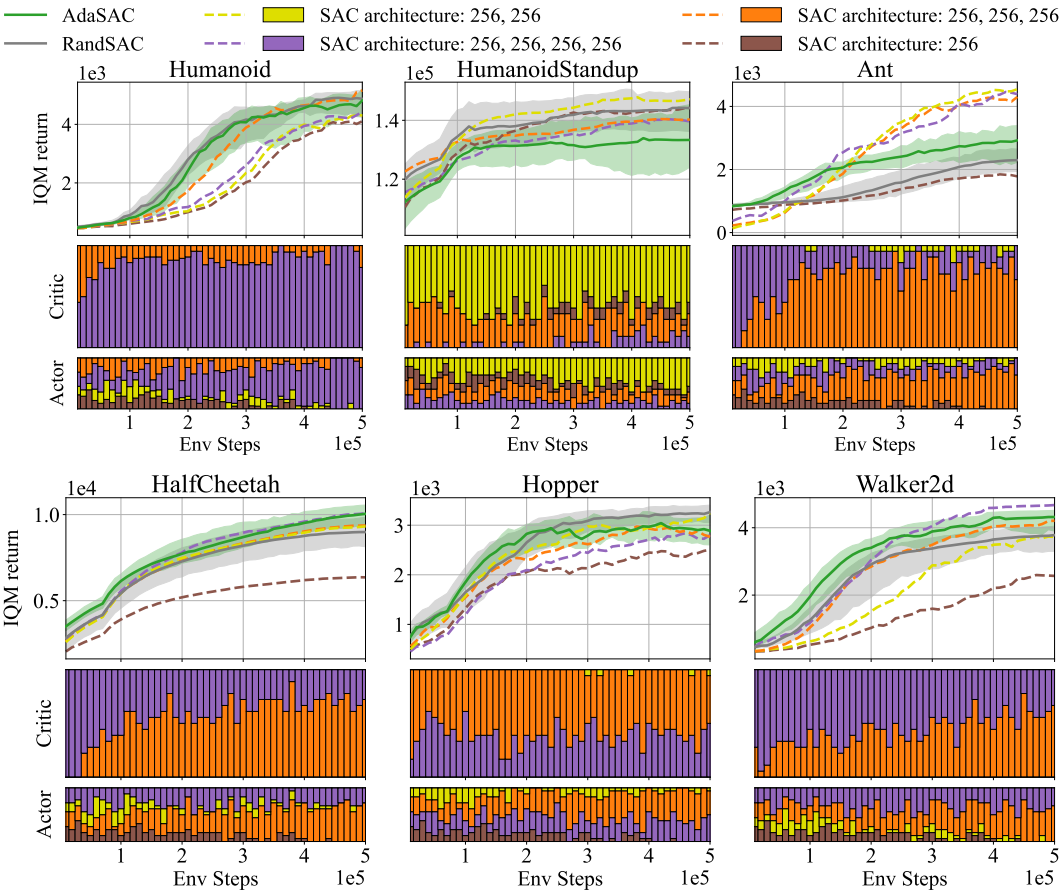

Figure 9: **Top:** Per environment return of AdaSAC and RandSAC when 4 different architectures are given as input. **Bottom:** Bar plot presenting the distribution of networks selected for the critic's loss and the actor's loss along the training. AdaSAC effectively ignores the worst-performing architecture (256) while actively selecting the best-performing architecture. Interestingly, in HumanoidStandup, despite not performing well, AdaSAC selects the architecture $256, 256$ which is the best-performing-architecture.

D.3 On-The-Fly Learning Rate Selection on MuJoCo

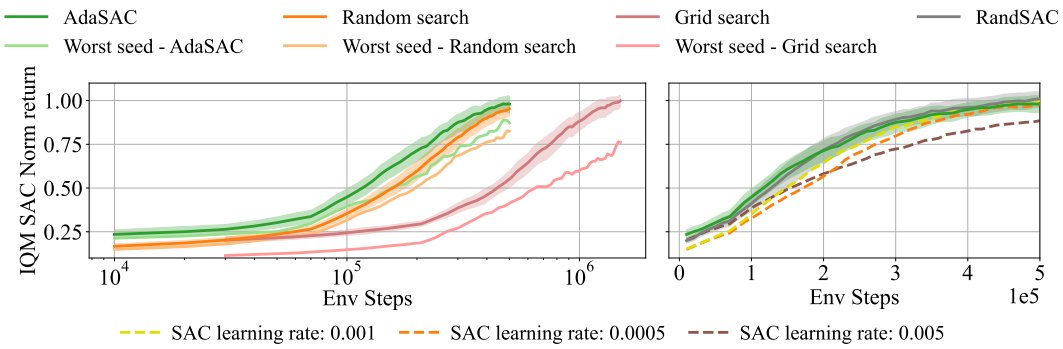

Figure 10: On-the-fly learning rate selection on **MuJoCo**. 3 different learning rates are given to AdaSAC and RandSAC as input. **Left:** AdaSAC is more sample-efficient than grid search. **Right:** AdaSAC yields performances slightly above the best-performing learning rate. RandSAC and AdaSAC are on par.

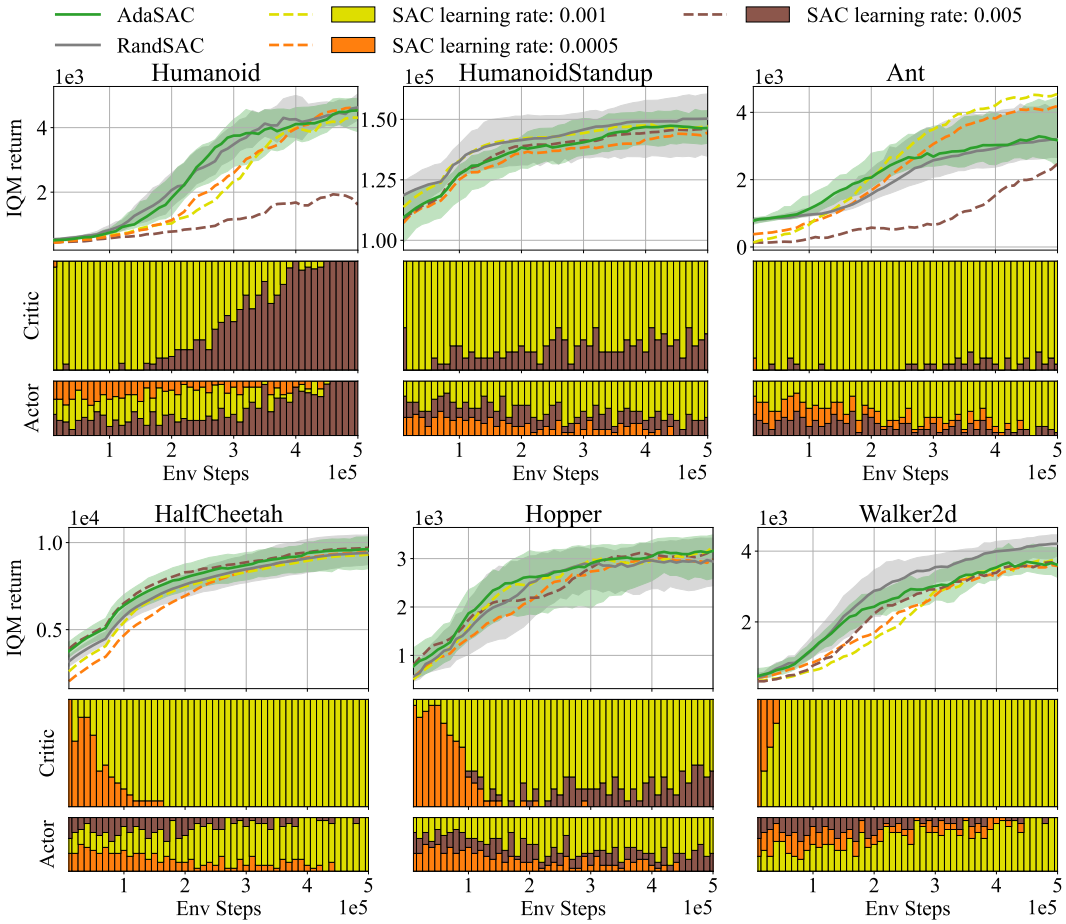

Figure 11: **Top:** Per environment return of AdaSAC and RandSAC when 3 different learning rates are given as input. **Bottom:** Bar plot presenting the distribution of networks selected for the critic's loss and the actor's loss along the training. AdaSAC creates a custom learning rate schedule for each environment and each seed. Interestingly, this schedule seems to choose a low learning rate at the beginning of the training before increasing it later during the training.

D.4 On-The-Fly Optimizer Selection on MuJoCo

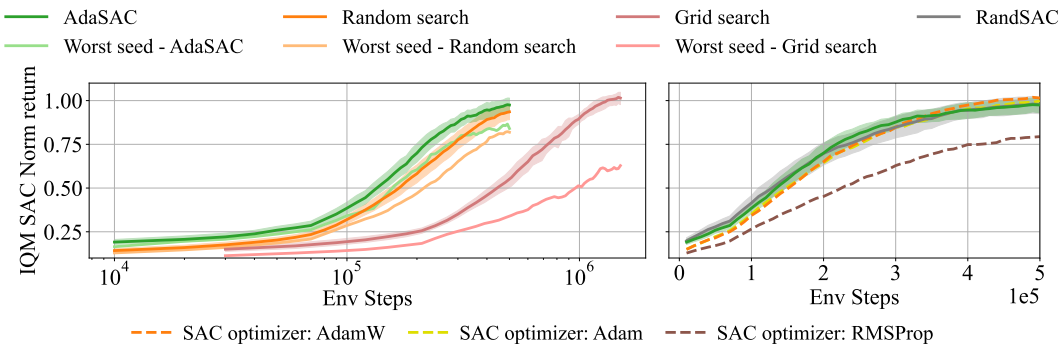

Figure 12: On-the-fly optimizer selection on **MuJoCo**. 3 different optimizers are given to AdaSAC and RandSAC as input. **Left:** AdaSAC is more sample-efficient than grid search. **Right:** AdaSAC yields similar performances as the best-performing optimizer (AdamW, Loshchilov & Hutter (2018)). RandSAC and AdaSAC are on par.

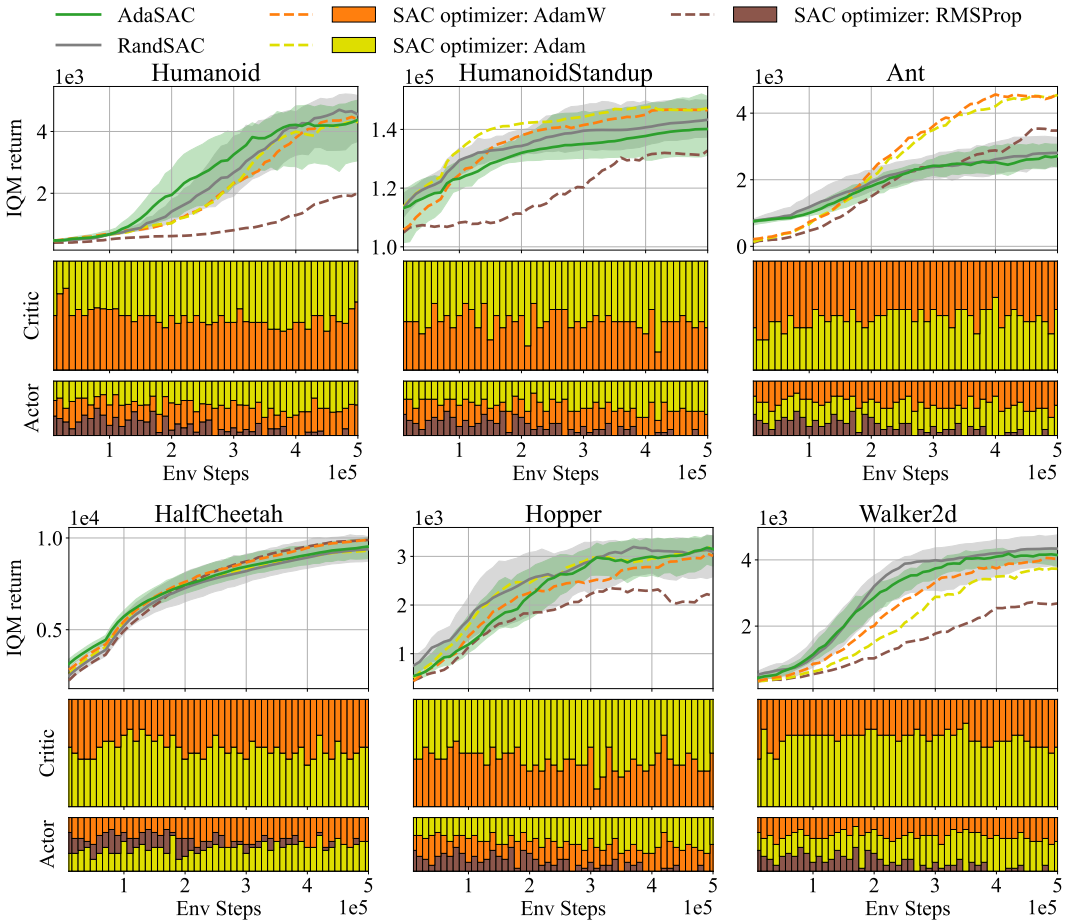

Figure 13: **Top:** Per environment return of AdaSAC and RandSAC when 3 different optimizers are given as input. **Bottom:** Bar plot presenting the distribution of networks selected for the critic's loss and the actor's loss along the training. AdaSAC effectively ignores the RMSProp, which performs poorly in most environments.

D.5 On-The-Fly Activation Function Selection on MuJoCo

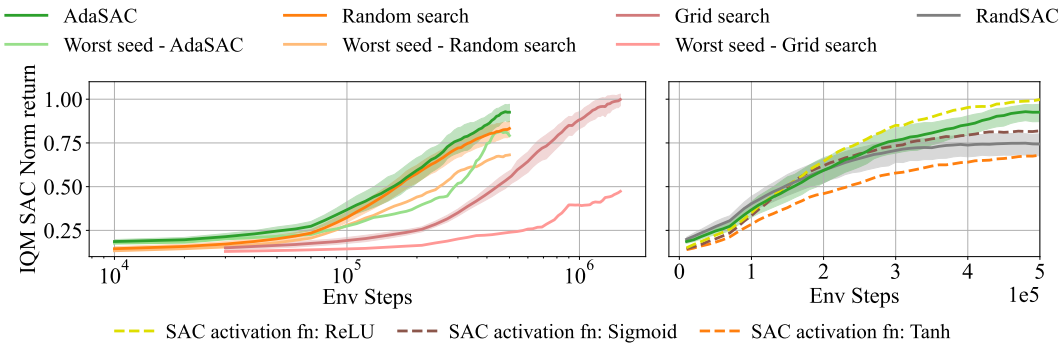

Figure 14: On-the-fly activation function selection on **MuJoCo**. 3 different activation functions are given to AdaSAC and RandSAC as input. **Left:** AdaSAC is more sample-efficient than grid search. **Right:** AdaSAC performs slightly below the best-performing activation function. In this setting, RandSAC suffers from the fact that a network with Tanh activation functions is selected as the target network one-third of the time.

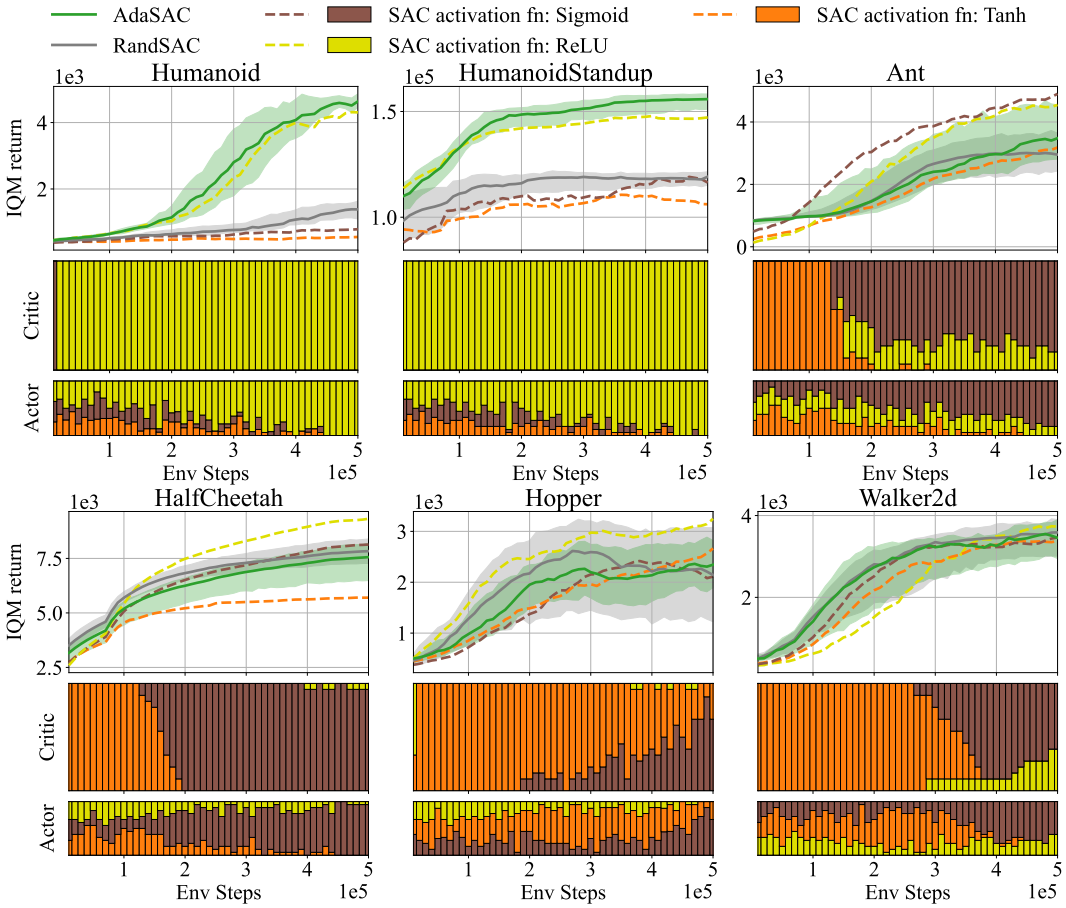

Figure 15: **Top:** Per environment return of AdaSAC and RandSAC when 3 different activation functions are given as input. **Bottom:** Bar plot presenting the distribution of networks selected for the critic's loss and the actor's loss along the training. Remarkably, AdaSAC selects the Sigmoid activation function in environments where this activation function seems beneficial (Ant, HalfCheetah, and Walker2d), while it does not select the Sigmoid activation function in environments where the individual run performs poorly (Humanoid and HumanoidStandup).