TAAC: Temporally Abstract Actor-Critic for Continuous Control

Anonymous Author(s)
Affiliation
Address
email

Abstract

We propose temporally abstract actor-critic (TAAC), an off-policy RL algorithm that incorporates closed-loop temporal abstraction into the actor-critic framework in a simple manner. TAAC adds a second-stage binary policy to choose between the previous action and a new action output by an actor. Crucially, its act-or-repeat decision hinges on the actually sampled action instead of the expected behavior of the actor. This post-acting switching scheme let the overall policy make more informed decisions. TAAC has two important features: persistent exploration and a new compare-through Q operator for multi-step TD backup. We demonstrate TAAC's advantages over several strong baselines across 5 different categories of 14 continuous control tasks. Code will be made public.

1 Introduction

Deep reinforcement learning (RL) has achieved great success in various continuous action domains such as locomotion and manipulation (Schulman et al., 2015; Lillicrap et al., 2016; Duan et al., 2016; Schulman et al., 2017; Fujimoto et al., 2018; Haarnoja et al., 2018). Despite promising empirical results, these widely applicable continuous RL algorithms execute a newly computed action at every step of the finest time scale of a problem. With no decision making at higher levels, they attempt to solve the challenging credit assignment problem over a long horizon. As a result, considerable sample efficiency improvements have yet to be made by them in complex task structures (Riedmiller et al., 2018; Li et al., 2020; Lee et al., 2020b) and extremely sparse reward settings (Andrychowicz et al., 2017; Plappert et al., 2018; Zhang et al., 2021).

On the other hand, it is argued that temporal abstraction (Parr and Russell, 1998; Dietterich, 1998; Sutton et al., 1999; Precup, 2000) is one of the crucial keys to solving control problems with complex structures. Larger steps are taken at higher levels of abstraction while lower-level actions only need to focus on solving isolated subtasks (Dayan and Hinton, 1993; Vezhnevets et al., 2017; Bacon et al., 2017). However, most hierarchical RL methods are task specific or nontrivial to adapt. For example, the options framework (Sutton et al., 1999; Precup, 2000; Bacon et al., 2017) requires pre-defining an option space, while the feudal RL framework Vezhnevets et al. (2017); Nachum et al. (2018); Zhang et al. (2021) requires tuning the hyperparameters of dimensionality and domain range of the goal space. In practice, their final performance usually hinges on these choices.

Perhaps the simplest form of an option or sub-policy would be just repeating an action for a certain number of steps, a straightforward idea that has been explored before (Lakshminarayanan et al., 2017; Sharma et al., 2017; Dabney et al., 2021; Metelli et al., 2020; Lee et al., 2020a). This line of works can be regarded as a middle ground between "flat" and hierarchical RL. Nevertheless, they assume a fixed candidate set of action durations, and repeat actions in an open-loop manner. Open-loop control forces an agent to commit to the same action over a pre-determined duration with no early

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

termination. It weakens the agent's ability of handling emergency situations and correcting wrong durations predicted earlier.

In this paper, our goal is to keep the simplicity of "flat" RL while modeling lightweight temporal abstraction. We propose **t**emporally **a**bstract **a**ctor-**c**ritic (TAAC), an off-policy RL algorithm that incorporates closed-loop action repetition into an actor-critic framework. Generally, we add a second stage that chooses between a candidate action output by an actor and the action from the previous step (Figure 1). Crucially, its act-or-repeat decision hinges on the actually sampled individual action instead of the expected behavior of the actor. This post-acting switching scheme let the overall policy make more informed decisions. Although this two-stage formulation seems straightforward, the key question we ask is: instead of training it with a generic actor-critic algorithm, *how can we exploit its special structure to try to maximize the efficiency of policy evaluation and improvement, so that the overall algorithm yields strong sample efficiency and final performance*? This will be the primary focus and contribution of the paper.

For policy evaluation, we propose a new compare-through Q operator for multi-step TD backup. For policy improvement, we compute the actor gradient by multiplying a scaling factor to a $\frac{\partial Q}{\partial a}$ term similarly found in DDPG (Lillicrap et al., 2016) and SAC (Haarnoja et al., 2018), where the scaling factor is the optimal probability of choosing the actor's action in the second stage (Eq. 7). TAAC is much easier to train compared to sophisticated hierarchical methods, and it has two important features compared to "flat" RL algorithms. First, TAAC encourages persistent exploration because the second stage can repeat the same action with a non-zero probability. Second, TAAC natively supports multi-step TD backup with a compare-through Q operator. We evaluate TAAC on 5 categories of 14 continuous control tasks, covering simple control, locomotion, terrain walking (Brockman et al., 2016), manipulation (Plappert et al., 2018), and self-driving (Dosovitskiy et al., 2017). Averaged over these tasks, TAAC largely outperforms 6 strong baselines.

2 Related work

Many works have been proposed to handle a hybrid continuous-discrete action space (Xiong et al., 2018; Fan et al., 2019; Delalleau et al., 2020). However, they mainly focus on environment-defined hybrid actions instead of temporal abstraction. In contrast, TAAC defines a binary switching action as a *latent* variable and does not directly compute its Q value, as it is not from the environment. Sharma et al. (2017) also learns a hybrid action space and does treat the discrete action as a latent variable of continuous action repeating steps, but the introduced temporal abstraction is open-loop. One recent work close to TAAC is PIC (Chen et al., 2021) which also learns to repeat the last action to address the action oscillation issue within consecutive steps. However, PIC was proposed for discrete control and its extension to continuous control is unclear yet. Also, PIC predicts whether to repeat the last action *independent of* a newly sampled action, which requires its switching policy to make a decision regarding the core policy's *expected* behavior. In an application section, H-MPO (Neunert et al., 2020) explored how continuous control can benefit from a meta binary action that modifies the overall system behavior. Again, like PIC their binary decision is made *in parallel with* a newly sampled action. Different from PIC and H-MPO, TAAC only decides "act-or-repeat" *after* comparing the previous action with a newly sampled action. Moreover, TAAC employs a new compare-through Q operator, and is trained by a much simpler actor gradient by absorbing the closed-form solution of the binary policy into the continuous action objective to avoid parameterizing a discrete policy.

3 Preliminaries

We consider the RL problem as policy search in a Markov Decision Process (MDP). Let $s \in \mathbb{R}^M$ denote a state, where a continuous action $a \in \mathbb{R}^N$ is taken. Let $\pi(a|s)$ be the action policy, and $\mathcal{P}(s_{t+1}|s_t, a_t)$ the probability of the environment transitioning to s_{t+1} after an action a_t is taken at s_t. Upon reaching s_{t+1}, the agent receives a scalar reward $r(s_t, a_t, s_{t+1})$. Assuming an initial state $s_0 \sim \mathcal{P}(s_0)$, the RL objective is to find a policy π^* that maximizes the expected discounted return: $\mathbb{E}_{\pi,\mathcal{P}}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1})\right]$, where $\gamma \in (0, 1)$ is a discount factor. We further define $Q^\pi(s_t, a_t) = \mathbb{E}_\pi[\sum_{t'=t}^{\infty} \gamma^{t'-t} r(s_{t'}, a_{t'}, s_{t'+1})]$ as the discounted return starting from s_t given that a_t is taken, and $V^\pi(s_t) = \mathbb{E}_{a_t \sim \pi} Q^\pi(s_t, a_t)$ as the state value of s_t. We denote the marginal state visitation distribution by $\rho^\pi(s)$, then the RL objective can also be written as $\mathbb{E}_{s \sim \rho^\pi, a \sim \pi} Q^\pi(s, a)$.

However, computing $\rho^\pi(s)$ is usually intractable. As a result, in an off-policy actor-critic setting with π and Q parameterized by ϕ and θ, the objective is sometimes modified (Lillicrap et al., 2016; Haarnoja et al., 2018) as

$$\max_\phi \mathop{\mathbb{E}}_{s\sim\mathcal{D}} V_\theta^{\pi_\phi}(s) = \max_\phi \mathop{\mathbb{E}}_{s\sim\mathcal{D}, a\sim\pi_\phi} Q_\theta(s,a). \tag{1}$$

This assumes that 1) s is sampled from a replay buffer instead of the current policy, and 2) the dependency of the critic Q on the policy π_ϕ is dropped when computing the gradient of ϕ. Meanwhile, θ is learned separately via policy evaluation with typical TD backup.

4 Temporally abstract actor-critic

To enable temporal abstraction, we motivate a decomposition of the agent's action decision into two stages (Figure 1): 1) sampling a new candidate action $\hat{a} \sim \pi_\phi(\cdot|s, a^-)$ conditioned on the action a^- at the previous time step, and 2) choosing between a^- and $\hat{a}$ as the actual output at the current step. The overall TAAC algorithm is summarized in Algorithm 1 Appendix A.

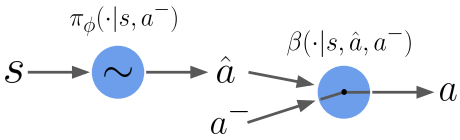

Figure 1: TAAC's two-stage policy during inference. In the first stage, an action policy π_ϕ samples a candidate action $\hat{a}$. In the second stage, a binary switching policy β chooses between this candidate and the previous action a^-.

4.1 Two-stage policy

Formally, let $\beta(b|s, \hat{a}, a^-)$ be the binary switching policy, where $b = 0/1$ means choosing $a^-/\hat{a}$. For simplicity, in the following we will denote $\beta_b = \beta(b|s, \hat{a}, a^-)$ (always assuming its dependency on s, $\hat{a}$, and a^-). Then our two-stage policy π^{ta} for temporal abstraction is defined as

$$\pi^{\text{ta}}(a|s, a^-) \triangleq \int_{\hat{a}} \pi_\phi(\hat{a}|s, a^-) \left[\beta_0 \delta(a - a^-) + \beta_1 \delta(a - \hat{a})\right] \mathrm{d}\hat{a}, \tag{2}$$

which can be shown to be a proper probability distribution of a. This two-stage policy repeats previous actions through a binary policy β, a decision maker that compares a^- and $\hat{a}$ side by side given the current state s. Repeatedly favoring $b = 0$ results in temporal abstraction of executing the same action for multiple steps. Moreover, this control is closed-loop, as it does not commit to a pre-determined time window; instead it can stop repetition whenever necessary. As a special case, when $\beta_1 = 1$, π^{ta} reduces to π_ϕ; when $\beta_0 = 1$, $\pi^{\text{ta}}(a|s, a^-) = \delta(a - a^-)$.

4.2 Policy evaluation with the compare-through operator

The typical one-step TD learning objective for a policy π is

$$\min_\theta \mathop{\mathbb{E}}_{(s,a,s')\sim\mathcal{D}} \left[Q_\theta(s,a) - \mathcal{B}^\pi Q_{\bar{\theta}}(s,a)\right]^2, \text{ with } \mathcal{B}^\pi Q_{\bar{\theta}}(s,a) = r(s,a,s') + \gamma V_{\bar{\theta}}^\pi(s'), \tag{3}$$

where $\mathcal{B}^\pi$ is the Bellman operator, and $\bar{\theta}$ slowly tracks θ to stabilize the learning (Mnih et al., 2015). For multi-step bootstrapping, importance correction is usually needed, for example, Retrace (Munos et al., 2016) relies on a function of $\frac{\pi(a|s)}{\mu(a|s)}$ (μ is a behavior policy) to correct for the off-policyness of a trajectory. Unfortunately, our π^{ta} makes probability density computation challenging because of the marginalization over $\hat{a}$. Thus importance correction methods including Retrace cannot be applied to our case. To address this issue, below we propose a new multi-step Q operator, called *compare-through*. Then we explain how π^{ta} can exploit this operator for efficient policy evaluation.

For a learning policy π, given a trajectory $(s_0, a_0, s_1, a_1, \ldots, s_N, a_N)$ from a behavior policy μ, let $\tilde{a}_0 = a_0$ and let $\tilde{a}_n$ denote the action sampled from π at states $s_n (n \geq 1)$. We define a point estimate of the compare-through operator $\mathcal{T}^\pi$ as

$$\mathcal{T}^\pi Q_{\bar{\theta}}(s_0, a_0) \approx \sum_{t=0}^{n-1} \gamma^t r(s_t, a_t, s_{t+1}) + \gamma^n Q_{\bar{\theta}}(s_n, \tilde{a}_n), n = \min\left(\{n | \tilde{a}_n \neq a_n\} \cup \{N\}\right). \tag{4}$$

Intuitively, starting from $n = 1$, we set $n \leftarrow n + 1$ if $\tilde{a}_n = a_n$. We repeat this comparison until $\tilde{a}_n \neq a_n$ or $n = N$, and use $Q_{\bar{\theta}}(s_n, \tilde{a}_n)$ as the bootstrapping Q value. A formal definition of $\mathcal{T}^\pi$ is described by Eq. 17, and its relation to Retrace (Munos et al., 2016) is shown in Appendix K.

Theorem 1 (Policy evaluation convergence) *The compare-through operator $\mathcal{T}^\pi$, whose expectation form defined by Eq. 17 and point estimate defined by Eq. 4, has a unique fixed point Q^π.*

For the detailed proof we refer the reader to Appendix K.

Clearly, any discrete policy could exploit the compare-through operator since there can be a non-zero chance of two discrete actions being compared equal. A typical stochastic continuous policy such as Gaussian used by SAC (Haarnoja et al., 2018) always has $\tilde{a}_n \neq a_n$ for $n \geq 1$ w.r.t. any behavior policy μ. In this case $\mathcal{T}^\pi$ is no more than just a Bellman operator $\mathcal{B}^\pi$. However, if a continuous policy is specially structured to be "action-reproducible", it will enjoy the privilege of using s_n for $n > 1$ as the target state. Our two-stage π^{ta} is such a case, where each action a_n ($\tilde{a}_n$) is accompanied by a repeating choice b_n ($\tilde{b}_n$). Starting from $n = 1$, if $\tilde{b}_n = b_n = 0$ (both repeated), then we know that $\tilde{a}_n = a_n = a_0$[1]. See Figure 2 for an illustration. Thus for multi-step TD learning we use $\mathcal{T}^{\pi^{\text{ta}}}$ to replace $\mathcal{B}^\pi$ in Eq. 3.

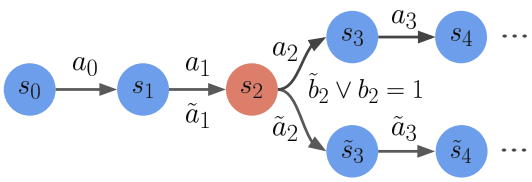

Figure 2: An illustration of the compare-through operator by exploiting action repetition of π^{ta}. The upper branch is the trajectory sampled by a rollout policy; the lower one is sampled by the current policy during training. We have $\tilde{a}_1 = a_1 = a_0$ due to $b_1 = \tilde{b}_1 = 0$. The two trajectories diverge at s_2 because either $b_2 = 1$ or $\tilde{b}_2 = 1$. For bootstrapping, we use s_2 as the target state in this example.

Remark The compare-through operator $\mathcal{T}^\pi$ is certainly not meant to replace multi-step TD learning with importance correction (Munos et al., 2016) in a general scenario, as it is only effective for "action-reproducible" policies. Also its bootstrapping step is usually hard-limited (by checking $\tilde{a}_n \neq a_n$) to be smaller than the maximal bootstrapping step N.

4.3 Policy improvement with a closed-form β^*

It can be shown (Appendix B) that the parameterized state value of π^{ta} is

$$V_\theta^{\pi^{\text{ta}}}(s|a^-) = \mathop{\mathbb{E}}_{\hat{a}\sim\pi_\phi, b\sim\beta} \left[(1-b)Q_\theta(s,a^-) + bQ_\theta(s,\hat{a})\right], \tag{5}$$

Intuitively, the actual Q value of each $a \sim \pi^{\text{ta}}$ is an interpolation by β between the Q values of a^- and $\hat{a}$. Our policy improvement objective is then $\max_{\phi,\beta} \mathbb{E}_{(s,a^-)\sim\mathcal{D}} V_\theta^{\pi^{\text{ta}}}(s|a^-)$. Note that each time we sample the previous action a^- along with a state s from the replay buffer. To encourage exploration, following prior works (Mnih et al., 2016; Riedmiller et al., 2018) we augment this objective with a joint entropy $\mathcal{H}_{s,a^-} = \mathbb{E}_{\pi_\phi(\hat{a}|s,a^-)\beta_b}\left[\log \pi_\phi(\hat{a}|s,a^-) + \log \beta_b\right]$. Thus the final policy improvement objective is

$$\max_{\phi,\beta} \mathop{\mathbb{E}}_{(s,a^-)\sim\mathcal{D}} \left[V_\theta^{\pi^{\text{ta}}}(s|a^-) + \alpha\mathcal{H}_{s,a^-}\right] = \max_{\phi,\beta} \mathop{\mathbb{E}}_{\substack{(s,a^-)\sim\mathcal{D}\\ \hat{a}\sim\pi_\phi, b\sim\beta}} \left[\begin{array}{c} (1-b)Q_\theta(s,a^-) + bQ_\theta(s,\hat{a}) \\ -\alpha\left(\log\beta_b + \log\pi_\phi(\hat{a}|s,a^-)\right) \end{array}\right], \tag{6}$$

where α is a temperature parameter. Given any $(s, a^-, \hat{a})$, we can derive a closed-form solution of the (non-parametric) β policy for the innermost expectation $b \sim \beta$ as

$$\beta_1^* \triangleq \frac{\exp\left(\frac{Q_\theta(s,\hat{a})}{\alpha}\right)}{\exp\left(\frac{Q_\theta(s,\hat{a})}{\alpha}\right) + \exp\left(\frac{Q_\theta(s,a^-)}{\alpha}\right)}.$$

Then applying the re-parameterization trick $\hat{a} = f_\phi(\epsilon, s, a^-), \epsilon \sim \mathcal{N}(0, I)$, one can show that the estimated actor gradient is

$$\Delta\phi \triangleq \left(\beta_1^* \frac{\partial Q_\theta(s,\hat{a})}{\partial\hat{a}} - \alpha\frac{\partial \log\pi_\phi(\hat{a}|s,a^-)}{\partial\hat{a}}\right)\frac{\partial f_\phi}{\partial\phi} - \alpha\frac{\partial\log\pi_\phi(\hat{a}|s,a^-)}{\partial\phi}. \tag{7}$$

[1] There is an assumption that the behavior policy repeats the previous action with a non-zero probability at every state. This can be satisfied if we add noise to its β or enforce β's entropy to be positive.

	SAC (Haarnoja et al., 2018)	SAC-Ntd	SAC-Nrep	SAC-Krep (Sharma et al., 2017)	SAC-EZ (Dabney et al., 2021)	SAC-Hybrid (Neunert et al., 2020)	TAAC-1td	TAAC
Persistent exploration	✗	✗	✓	✓	✓	✓	✓	✓
Multi-step TD	✗	✓	✓	✓	✗	✓	✗	✓
Closed-loop repetition	✗	✗	✗	✗	✗	✓	✓	✓
Learnable duration	✗	✗	✗	✓	✗	✓	✓	✓

Table 1: A summary of the 8 major comparison methods in our experiments.

This gradient has a very similar form with SAC's (Haarnoja et al., 2018), except that here $\frac{\partial Q}{\partial \hat{a}}$ has a scaling factor β_1^*. We refer the reader to a full derivation of the actor gradient in Appendix C.

Remark on β^* This closed-form solution is only possible after $\hat{a}$ is sampled. It's essentially comparing the Q values between a^- and $\hat{a}$. This side-by-side comparison is absent in previous closed-loop repeating methods like PIC (Chen et al., 2021) and H-MPO (Neunert et al., 2020).

Remark on multi-step actor gradient According to Figure 1, the newly generated action $\hat{a}_t$ at the current step t, if repeated as future $a_{t+1}^-, \ldots, a_{t'}^-$ $(t' > t)$, will also influence the maximization of future V values: $V_\theta^{\pi^{\text{ta}}}(s_{t+1}|a_{t+1}^-), \ldots, V_\theta^{\pi^{\text{ta}}}(s_{t'}|a_{t'}^-)$. As a result, in principle $\hat{a}_t$ has a multi-step gradient. To exactly compute this multi-step gradient, a fresh rollout of the current π^{ta} via interacting with the environment is necessary. For reasons detailed in Appendix D, we end up truncating this full gradient to the first step, by letting Eq. 6 sample (s, a^-) pairs at non-consecutive time steps from the replay buffer. This one-step truncation is also (implicitly) adopted by H-MPO (Neunert et al., 2020) for their action repetition application case. Interestingly, the truncation results in a simple implementation of actor gradient of TAAC similar to SAC's.

Automatically tuned temperatures Given two entropy targets $\mathcal{H}'$ and $\mathcal{H}''$, we learn temperatures α' and α'' from the objective

$$\min_{\log(\alpha'),\log(\alpha'')} \mathop{\mathbb{E}}_{(s,a^-)\sim\mathcal{D},\hat{a}\sim\pi_\phi,b\sim\beta^*} \Big[\log(\alpha')(-\log\beta_b^* - \mathcal{H}') + \log(\alpha'')(-\log\pi_\phi(\hat{a}|s,a^-) - \mathcal{H}'')\Big]. \tag{8}$$

by adjusting $\log(\alpha)$ instead of α as in SAC (Haarnoja et al., 2018). We adjust temperatures for π_ϕ and β separately, to enable a finer control of the two policies and their entropy terms, similar to the separate policy constraints (Abdolmaleki et al., 2018; Neunert et al., 2020). Appendix E shows how several equations slightly change if two temperatures are used (α is replaced by α' and α'').

5 Experiments

5.1 Tasks

To test if the proposed algorithm generalizes, we perform experiments over a wide range of continuous control tasks under different scenarios:

a) **SimpleControl**: Three control tasks (Brockman et al., 2016) with small action and observation spaces: *MountainCarContinuous*, *LunarLanderContinuous*, and *InvertedDoublePendulum* ;
b) **Locomotion**: Four locomotion tasks (Brockman et al., 2016) that feature complex physics and action spaces: *Hopper*, *Ant*, *Walker2d*, and *HalfCheetah*;
c) **Terrain**: Two locomotion task that require adapting to randomly generated terrains: *BipedalWalker* and *BipedalWalkerHardcore*;
d) **Manipulation**: Four Fetch (Plappert et al., 2018) tasks with sparse rewards and hard exploration (reward given only upon success): *FetchReach*, *FetchPush*, *FetchSlide*, and *FetchPickAndPlace*;
e) **Driving**: One CARLA autonomous-driving task (Dosovitskiy et al., 2017) that have complex high-dimensional multi-modal sensor inputs (camera, radar, IMU, collision, GPS, *etc.*): *Town01*. The goal is to reach a destination starting from a randomly spawned location in a large realistic town, while avoiding collisions and red light violations.

Among the 5 categories, (d) and (e) might benefit greatly from temporal abstraction because of hard exploration or the problem structure (*e.g.*, driving naturally involves repeated actions). Categories (a)-(c) appear unrelated with temporal abstraction, but we test if seemingly unrelated tasks can also benefit from it. By comparing TAAC against different methods across vastly different tasks, we hope to demonstrate its generality, because adaptation to this kind of task variety requires few assumptions about the task structure and inputs/outputs. For more task details, we refer the reader to Appendix H.

5.2 Comparison methods

While there exist many off-policy hierarchical RL methods that model temporal abstraction, for example Nachum et al. (2018); Riedmiller et al. (2018); Levy et al. (2019); Li et al. (2020); Zhang et al. (2021), we did not find them readily scalable to our entire list of tasks (especially to high dimensional input space like CARLA), without considerable efforts of adaptation. Thus our primary focus is to compare TAAC with baselines of different formulations of action repetition: vanilla SAC (Haarnoja et al., 2018), SAC-Nrep, SAC-Krep (Sharma et al., 2017), SAC-EZ (Dabney et al., 2021), and SAC-Hybrid (Neunert et al., 2020). Although originally some baselines have their own RL algorithm backbones, in this experiment we choose SAC as the common backbone because: 1) SAC is state-of-the-art among open-sourced actor-critic algorithms, 2) a common backbone facilitates reusing the same set of core hyperparameters for a fairer comparison, and 3) by reducing experimental variables, it gives us a better focus on the design choices of action repetition instead of other orthogonal algorithmic components.

Let N be a parameter controlling the maximal number of action repeating steps. SAC-Nrep simply repeats every action N times. SAC-Krep, inspired by FiGAR (Sharma et al., 2017), upgrades an action a to a pair of (a, K), indicating that the agent will repeat action a for the next K steps ($1 \leq K \leq N$) without being interrupted until finishing. To implement SAC-Krep, following Delalleau et al. (2020) we extended the original SAC algorithm to support a mixture of continuous and discrete actions (Appendix F). Note that SAC-Krep's open-loop control is in contrast to TAAC's closed-loop control. SAC-EZ incorporates the temporally extended ϵ-greedy exploration (Dabney et al., 2021) into SAC. During rollout, if the agent decides to explore, then the action is uniformly sampled and the duration for repeating that action is sampled from a truncated zeta distribution $zeta(n) \propto n^{-\mu}, 1 \leq n \leq N$. This fixed duration model encourages persistent exploration depending on the value of the hyperparameter μ. The training step of SAC-EZ is the same with SAC. SAC-Hybrid shares a similar flavor of H-MPO (Neunert et al., 2020) for closed-loop action repetition. It defines a hybrid policy to output the continuous action and binary switching action in parallel, assuming their conditional independence given the state. This independence between hybrid actions and how the Q values are computed in SAC-Hybrid are the biggest differences with TAAC. We also apply Retrace (Munos et al., 2016) to its policy evaluation with N-step TD as done by H-MPO. We refer the reader to the algorithm details of SAC-Hybrid in Appendix G.

In order to analyze the benefit of persistent exploration independent of that of multi-step TD learning, we also compare two additional methods. SAC-Ntd is a variant of SAC where a Q value is bootstrapped by an N-step value target with Retrace (Munos et al., 2016) to correct for off-policyness. For ablating TAAC, we evaluate TAAC-1td that employs a typical Bellman operator $\mathcal{B}^{\pi^{\text{ta}}}$ for one-step bootstrapping. Thus we have 8 methods in total for comparison in each task. See Table 1 for a summary and Appendix I for method details.

In our experiments, we set the repeating hyperparameter N to 3 on **SimpleControl**, **Locomotion** and **Manipulation**, and to 5 on **Terrain** and **Driving**. The consideration here is mainly for open-loop methods like SAC-Nrep and SAC-Krep because they will yield poor performance with large values of N. TAAC is not sensitive to N for policy evaluation, as shown in Section 5.5.

5.3 Evaluation protocol

To measure the final model quality, we define *score* as the episodic return $\sum_{t=0}^{T} r(s_t, a_t, s_{t+1})$ of evaluating (taking $\arg\max$ of a policy) a method for a task episode, where T is a pre-defined time limit or when the episode terminates early. To facilitate score aggregation across tasks, we adopt the metric of *normalized score* (*n-score*). For each task, we obtain the score (averaged over 100 episodes) of a random policy and denote it by Z_0. We also evaluate the best method on that task and obtain its average score Z_1. Given a score Z, its normalized value is calculated as $\frac{Z-Z_0}{Z_1-Z_0}$. With this definition, the n-score of each task category (a)-(e) can be computed as the averaged n-score across tasks within that category. Additionally, to measure training convergence speed, we approximate *n-AUC* (area under the n-score curve normalized by x value range) by averaging n-scores on a n-score curve throughout the training. A higher n-AUC value indicates a faster convergence speed.

On every task, each method is trained for the same number of environment frames. Crucially, for fair comparisons we also make each method train 1) for the same number of gradient steps, 2) with the same mini-batch size and learning rate, 3) using roughly the same number of weights, and 4) with a

		SAC	SAC-Ntd	SAC-Nrep	SAC-Krep	SAC-EZ	SAC-Hybrid	TAAC-1td	TAAC
Final n-score (model quality)	**SimpleControl**	0.64[0.03]	0.54[0.07]	0.84[0.01]	0.55[0.03]	0.65[0.03]	0.88[0.14]	0.83[0.13]	**0.99**[0.04]
	Locomotion	0.89[0.06]	0.88[0.11]	0.70[0.08]	0.82[0.07]	**0.92**[0.07]	0.63[0.08]	0.88[0.04]	0.90[0.05]
	Terrain	0.35[0.10]	0.48[0.02]	0.54[0.03]	0.48[0.03]	0.45[0.15]	0.79[0.03]	0.96[0.02]	**1.00**[0.03]
	Manipulation	0.60[0.12]	0.75[0.17]	0.91[0.07]	0.98[0.05]	0.68[0.07]	0.49[0.07]	**0.99**[0.02]	0.99[0.01]
	Driving	0.88[0.05]	0.84[0.04]	0.92[0.03]	0.95[0.04]	0.71[0.03]	**1.00**[0.04]	0.92[0.02]	0.97[0.04]
	All	0.68[0.08]	0.71[0.10]	0.78[0.05]	0.77[0.05]	0.71[0.07]	0.69[0.08]	0.92[0.05]	**0.96**[0.03]
n-AUC (convergence speed)	**SimpleControl**	0.45[0.01]	0.41[0.01]	0.51[0.02]	0.28[0.03]	0.45[0.02]	0.62[0.08]	0.60[0.10]	**0.72**[0.04]
	Locomotion	0.69[0.03]	0.64[0.07]	0.55[0.05]	0.59[0.04]	0.64[0.05]	0.50[0.06]	0.72[0.02]	**0.74**[0.05]
	Terrain	0.17[0.02]	0.19[0.02]	0.38[0.01]	0.23[0.01]	0.21[0.03]	0.50[0.01]	0.50[0.02]	**0.59**[0.02]
	Manipulation	0.41[0.04]	0.50[0.10]	0.69[0.06]	0.71[0.08]	0.49[0.06]	0.38[0.02]	0.71[0.04]	**0.77**[0.02]
	Driving	0.38[0.02]	0.41[0.04]	0.52[0.02]	0.51[0.03]	0.32[0.02]	0.61[0.02]	0.60[0.03]	**0.65**[0.02]
	All	0.46[0.03]	0.47[0.06]	0.55[0.04]	0.50[0.05]	0.47[0.04]	0.50[0.04]	0.65[0.04]	**0.72**[0.03]

Table 2: The comparison results. Error margins inside brackets are computed by averaging the standard deviations (across random seeds) of individual tasks in a category. The last two shaded columns are our methods.

common set of hyperparameters (tuned with vanilla SAC) for the SAC backbone . More details of the experimental settings are described in Appendix I.

5.4 Results

The n-AUC and final n-score values are shown in Table 2. First of all, we conclude that the tasks are diverse enough to reduce the evaluation variance. The averaged standard deviations are small for most methods. Thus the comparison results are not coincidences and are likely to generalize to other scenarios. Overall, TAAC largely outperforms the 6 baselines regarding both final n-score ($0.96[0.03]$ *vs.* second-best $0.78[0.05]$) and n-AUC ($0.72[0.03]$ *vs.* second-best $0.55[0.04]$), with relatively small standard deviations. Note that the n-AUC gap naturally tends to be smaller than the final n-score gap because the former reflects a convergence trend throughout the training. Moreover, TAAC achieved top performance of n-AUC on each individual task category. Although some baselines achieved best final n-scores by slightly dominating TAAC, their performance is not consistent over all task categories. More observations can be made below.

- *Persistent exploration and the compare-through operator are both crucial.* Even with one-step TD, TAAC-1td's performance ($0.92[0.05]$ and $0.65[0.04]$) already outperforms the baselines. This shows that persistent exploration alone helps much. Furthermore, TAAC is generally better than TAAC-1td ($0.96[0.03]$ *vs.* $0.92[0.05]$ and $0.72[0.03]$ *vs.* $0.65[0.04]$). This shows that efficient policy evaluation by the compare-through operator is also a key component of TAAC.
- *A proper formulation of closed-loop action repetition is important.* The idea of closed-loop action repetition alone is not a magic ingredient, as SAC-Hybrid only has moderate performance among the baselines. Notably, it performs worst on **Locomotion** and **Manipulation**. Our analysis revealed that its "act-or-repeat" policy tends to repeat with very high probabilities on *Ant*, *FetchPickAndPlace*, and *FetchPush* even towards the end of training, resulting in very poor n-scores. All these three tasks feature hard exploration[2]. This result suggests that a good formulation of the idea is crucial. The two-stage decision $\pi(\hat{a}|s, a^-)\pi(b|s, a^-, \hat{a})$ of TAAC is clearly more informed than the parallel decisions $\pi(\hat{a}|s, a^-)\pi(b|s, a^-)$ of SAC-Hybrid. Furthermore, even with a latent "act-or-repeat" action, TAAC manages to maintain the complexity of the Q function $Q(s, a)$ as in the original control problem, while SAC-Hybrid has a more complex form $Q((s, a^-), (\hat{a}, b))$ (Appendix G).
- *Naive action repetition works well.* Interestingly, in this particular experiment, SAC-Nrep is the overall top-performing baseline due to its relatively balanced results on all task categories. However, when it fails, the performance could be very bad (**Locomotion** and **Terrain**). While its sample efficiency is good, it has a *difficulty of approaching the optimal control* (final mean n-scores ≤ 0.92 in individual task categories) due to its lack of flexibility in the action repeating duration. This suggests that action repetition greatly helps, but a fixed duration is difficult to pick.
- *Limitation: action repetition hardly benefits tasks with densely shaped rewards and frame skipping.* We notice that TAAC is no better than SAC and SAC-EZ on **Locomotion** regarding the final n-score, although it has slight advantages of n-AUC. The locomotion tasks have densely shaped rewards to guide policy search. Thus action repetition hardly helps locomotion exploration, especially when the 4 tasks already have built-in frameskips (4 frames for *Hopper* and *Walker2d*; 5 frames for *Ant* and *HalfCheetah*). We believe that more sophisticated temporal abstraction (*e.g.*, skills) is needed to improve the performance in this case.

[2] Although *Ant* has densely shaped rewards, a policy lacking exploration could easily gets stuck in a local minimum where the ant stands stationarily to maximize the alive bonus rewards.

	Final n-score (model quality)						n-AUC (convergence speed)					
	S	**L**	**T**	**M**	**D**	**All**	**S**	**L**	**T**	**M**	**D**	**All**
TAAC-Ntd ($N = 10$)	0.88[0.16]	0.73[0.18]	0.79[0.06]	0.82[0.07]	0.72[0.01]	0.79[0.12]	0.56[0.11]	0.54[0.12]	0.53[0.03]	0.62[0.04]	0.38[0.06]	0.55[0.08]
TAAC ($N = 10$)	1.00[0.02]	0.85[0.11]	0.97[0.04]	0.94[0.02]	0.94[0.03]	0.93[0.05]	0.73[0.01]	0.67[0.06]	0.59[0.02]	0.73[0.03]	0.60[0.00]	0.68[0.03]
SAC-Hybrid	0.88[0.14]	0.63[0.08]	0.79[0.03]	0.49[0.07]	1.00[0.04]	0.69[0.08]	0.62[0.08]	0.50[0.06]	0.50[0.01]	0.38[0.02]	0.61[0.02]	0.50[0.04]
SAC-Hybrid-RepThr	0.89[0.12]	0.65[0.08]	0.75[0.03]	0.68[0.17]	0.90[0.04]	0.74[0.11]	0.58[0.08]	0.48[0.06]	0.47[0.01]	0.47[0.09]	0.49[0.01]	0.50[0.06]

Table 3: Off-policyness experiments results. Error margins inside brackets are computed by averaging the standard deviations (across random seeds) of individual tasks in a category. **S** - **SimpleControl**; **L** - **Locomotion**; **T** - **Terrain**; **M** - **Manipulation**; **D** - **Driving**.

5.5 Off-policyness experiments

To verify that our compare-through operator is not affected by off-policyness in policy evaluation, we compare TAAC to a variant TAAC-Ntd which always bootstraps a Q value with an N-step target, regardless of β's outputs and without importance correction[3]. We choose a large trajectory length $N = 10$ to amplify the effect of off-policyness. Table 3 shows that N-step TD without importance correction significantly degrades the performance ($0.79[0.12]$ *vs.* $0.93[0.05]$ and $0.55[0.08]$ *vs.* $0.68[0.03]$). In contrast, the compare-through operator well addresses this issue for TAAC.

Furthermore, we implement a variant of SAC-Hybrid by replacing the Retrace operator with our compare-through operator, to see if the result difference between TAAC and SAC-Hybrid is mainly due to different ways of handling off-policyness. Table 3 shows that SAC-Hybrid-RepThr performs similarly to SAC-Hybrid ($0.74[0.11]$ *vs.* $0.69[0.08]$ and $0.50[0.06]$ *vs.* $0.50[0.04]$), suggesting that it is indeed the formulation of SAC-Hybrid that creates its performance gap with TAAC.

5.6 Policy behavior visualization and analysis

In this section, we mainly answer two questions:

1) How is the exploration behavior of TAAC compared to that of SAC, a "flat" RL algorithm?
2) What is the action repetition behavior of TAAC in a trained control task?

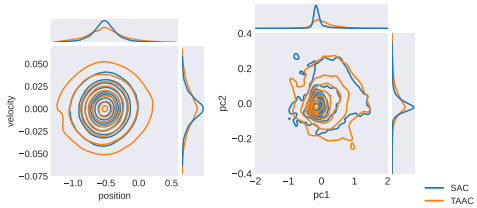

Figure 3: KDE plots of 100K state vectors visited by TAAC and SAC with their randomized policies. The side plots on top and right represent 1D marginal density functions. Left: *MountainCarContinuous*: x is the car's position and y is the car's velocity; Right: *BipedalWalker*: xy represent the top-2 principal components of the walker's hull.

5.6.1 Exploration behavior

TAAC introduces persistent exploration along previous actions, having a better chance of escaping the local neighborhood around a state when acting randomly. Thus TAAC's exploration should yield a better state space coverage than SAC's does, assuming other identical conditions. To verify this, we visualize the state space coverage by SAC and TAAC during their initial exploration phases.

Specifically, we select two tasks *MountainCarContinuous* and *BipedalWalker* for this purpose. For either TAAC or SAC, we play a random version of its policy on either task for 50K environment frames, to simulate the initial exploration phase where the model parameters have not yet been altered by training. During this period, we record all the 50K state vectors for analysis. For *MountainCarContinuous*, each state vector is 2D, representing the car's "x-position" and "x-velocity" on the 1D track. For *BipedalWalker*, each state vector is 24D, where the first 4D sub-vector indicates the statistics of the walker's hull: "angle", "angular velocity", "x-velocity", and "y-velocity". For visualization, we first extract this 4D sub-vector and form a combined dataset of 100K vectors from both TAAC and SAC. Then we apply PCA (Jolliffe, 1986) to this dataset and project each 4D vector down to 2D. After this, we are able to draw KDE (kernel density estimate) plots for both *MountainCarContinuous* and *BipedalWalker* in Figure 3. We see that on both tasks, a random policy of TAAC is able to cover more diverse states compared to SAC. This suggests that in general, TAAC is better at exploration compared to a "flat" RL method, thanks to its ability of persistent exploration.

5.6.2 Action repetition behavior

In theory, to achieve optimal continuous control, the best policy should always sample a new action at every step and avoid action repetition at all. Thus one might assume that TAAC's second-stage policy

[3]We cannot apply Retrace to TAAC-Ntd since the probability density of π^{ta} is computationally intractable.

Tasks	*MCC*	*LLC*	*IDP*	*HOP*	*ANT*	*WAL*	*HC*	*BW*	*BWH*	*FR*	*FP*	*FS*	*FPP*	*TOW*
Action repetition percentage	89%	74%	26%	37%	13%	26%	1%	25%	39%	9%	55%	54%	49%	55%

Table 4: Action repetition percentages by evaluating a trained TAAC model for 100 episodes on each of the 14 tasks. Refer to Section 5.1 for the full task names.

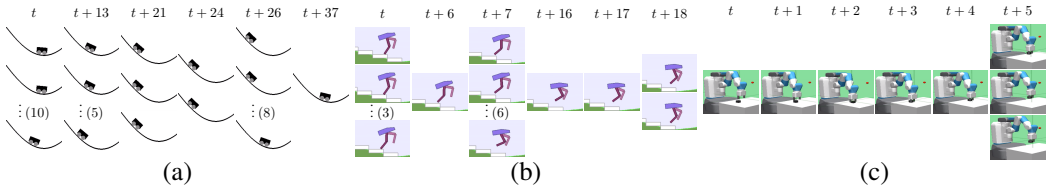

Figure 4: Example frames (cropped) from top-performing evaluation episodes of TAAC. Each column contains consecutive frame(s) generated by the same action, where "$\vdots\,(n)$" denotes n similar frames omitted due to space limit. (a): In *MountainCarContinuous* the car first backs up to build gravitational potential and rushes down; (b): In *BipedalWalkerHardcore* the bipedal walker jumps one step down; (c): In *FetchPickAndPlace* the robot arm approaches the object and lifts it to a goal location.

shifts from frequently repeating actions in the beginning of training, to not repeating actions at all towards the end of training, in order to optimize the environment reward. However, some examples in Figure 4 suggest that a significant repetition frequency still exists in TAAC's top-performing evaluation episodes. Note that because we take $\arg\max$ of the policy during evaluation, this suggests that for those repeating steps, we have $\beta_0^* > \beta_1^*$ by the trained model. In fact, if we evaluate 100 episodes for each of the three tasks in Figure 4 and compute the action repetition percentage (repeating steps divided by total steps), the percentages are surprisingly 89%, 39%, and 49%, even though the agents are doing extremely well! A complete list of action repetition percentages for all 14 tasks can be found in Table 4. Generally, TAAC is able to adjust the repetition frequency according to tasks, resulting in a variety of frequencies across the tasks. More importantly, if we inspect the repeated actions, they are often not even close to action boundaries $\{a_{min}, a_{max}\}$, ruling out the possibility of the policy being forced to repeat due to action clipping/squashing.

We believe that there are two reasons for this surprising observation. First, with many factors such as function approximation, noise in the estimated gradient, and stochasticity of the environment dynamics, it is hardly possible for a model to reach the theoretical upper bound of an RL objective. Thus for a new action and the previous action, if their estimated Q values are very similar, then TAAC might choose to repeat with a certain chance. For example in Figure 4 (c), while the robot arm is lifting the object towards the goal in the air, it can just repeat the same lifting action for 3 steps (at $t+5$), without losing the optimal return (up to some estimation error). Similar things happen to the bipedal walker (b) when it is jumping in the air (9 steps repeated at $t+7$), and to the mountain car (a) when it is rushing down the hill (11 steps repeated at $t+26$). Second, as a function approximator, the policy network π_ϕ have a limited capacity, and it is not able to represent the optimal policy at every state in a continuous space. For non-critical states that can be handled by repeating previous actions, *TAAC might learn to offload the decision making of π_ϕ onto β, and save π_ϕ's representational power for critical states*. For example, the robot arm in Figure 4 (c) invokes a fine-grained control by π_ϕ when it's grasping the object from the table, while later does not invoke π_ϕ for lifting.

6 Conclusion

We have proposed TAAC, an off-policy RL algorithm that is a middle ground between one-level RL and hierarchical RL. TAAC incorporates closed-loop temporal abstraction into actor-critic by adding a second-stage policy that chooses between the previous action and a new action output by an actor. TAAC yielded strong empirical results on a variety of continuous control tasks, standing out from prior works that also model action repetition. The evaluation and visualization revealed the success factors of TAAC: persistent exploration and a compare-through Q operator for multi-step TD backup. We believe that our work has provided valuable insights into modeling temporal abstraction and action hierarchies for solving complex RL tasks in the future.

Societal impact This paper proposes a general algorithm to improve the efficiency of RL for continuous control. The algorithm can be applied to many robotics scenarios in the real world for different purposes, to make robots learn motor skills faster than before. As with other RL algorithms, how to address the potentially unsafe behaviors and risks of the deployed algorithm caused to the surroundings in real-world scenarios remains an open problem and requires much consideration.

References

Abdolmaleki, A., Springenberg, J. T., Degrave, J., Bohez, S., Tassa, Y., Belov, D., Heess, N., and Riedmiller, M. A. (2018). Relative entropy regularized policy iteration. *arXiv*.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. In *NeurIPS*.

Bacon, P.-L., Harb, J., and Precup, D. (2017). The option-critic architecture. In *AAAI*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *CoRR*, abs/1606.01540.

Chen, C., Tang, H., Hao, J., Liu, W., and Meng, Z. (2021). Addressing action oscillations through learning policy inertia. In *AAAI*.

Dabney, W., Ostrovski, G., and Barreto, A. (2021). Temporally-extended epsilon-greedy exploration. In *ICLR*.

Dayan, P. and Hinton, G. E. (1993). Feudal reinforcement learning. In *NeurIPS*, pages 271–278.

Delalleau, O., Peter, M., Alonso, E., and Logut, A. (2020). Discrete and continuous action representation for practical RL in video games. In *AAAI workshop on reinforcement learning in games*.

Dietterich, T. G. (1998). The maxq method for hierarchical reinforcement learning. In *ICML*, pages 118–126.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). CARLA: An open urban driving simulator. In *CoRL*, pages 1–16.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *ICML*.

Fan, Z., Su, R., Zhang, W., and Yu, Y. (2019). Hybrid actor-critic reinforcement learning in parameterized action space. In *IJCAI*.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *ICML*, pages 1587–1596.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic algorithms and applications. *arXiv*, abs/1812.05905.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*, pages 770–778.

Jolliffe, I. T. (1986). Principal component analysis and factor analysis. In *Principal Component Analysis*. Springer New York.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR*.

Lakshminarayanan, A. S., Sharma, S., and Ravindran, B. (2017). Dynamic action repetition for deep reinforcement learning. In *AAAI*.

Lee, J., Lee, B.-J., and Kim, K.-E. (2020a). Reinforcement learning for control with multiple frequencies. In *NeurIPS*.

Lee, Y., Yang, J., and Lim, J. J. (2020b). Learning to coordinate manipulation skills via skill behavior diversification. In *ICLR*.

Levy, A., Jr., R. P., and Saenko, K. (2019). Learning multi-level hierarchies with hindsight. In *ICLR*.

Li, A. C., Florensa, C., Clavera, I., and Abbeel, P. (2020). Sub-policy adaptation for hierarchical reinforcement learning. In *ICLR*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *ICLR*.

Metelli, A. M., Mazzolini, F., Bisi, L., Sabbioni, L., and Restelli, M. (2020). Control frequency adaptation via action persistence in batch reinforcement learning. In *ICML*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *ICML*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. (2016). Safe and efficient off-policy reinforcement learning. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *NeurIPS*.

Nachum, O., Gu, S., Lee, H., and Levine, S. (2018). Data-efficient hierarchical reinforcement learning. In *NeurIPS*.

Neunert, M., Abdolmaleki, A., Wulfmeier, M., Lampe, T., Springenberg, J., Hafner, R., Romano, F., Buchli, J., Heess, N., and Riedmiller, M. (2020). Continuous-discrete reinforcement learning for hybrid control in robotics. In *CoRL*.

Parr, R. and Russell, S. (1998). Reinforcement learning with hierarchies of machines. In *NeurIPS*, pages 1043–1049.

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., Kumar, V., and Zaremba, W. (2018). Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv*.

Precup, D. (2000). *Temporal Abstraction in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst.

Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., van de Wiele, T., Mnih, V., Heess, N., and Springenberg, J. T. (2018). Learning by playing solving sparse reward tasks from scratch. In *ICML*.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015). Trust region policy optimization. In *ICML*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. In *arXiv*.

Sharma, S., Srinivas, A., and Ravindran, B. (2017). Learning to repeat: Fine grained action repetition for deep reinforcement learning. In *ICLR*.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181 – 211.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. (2017). Feudal networks for hierarchical reinforcement learning. In *ICML*, page 3540–3549.

Xiong, J., Wang, Q., Yang, Z., Sun, P., Han, L., Zheng, Y., Fu, H., Zhang, T., Liu, J., and Liu, H. (2018). Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *arXiv*.

Zhang, J., Yu, H., and Xu, W. (2021). Hierarchical reinforcement learning by discovering intrinsic options. In *ICLR*.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] In the experiment results section
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In the conclusion section.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] Appendix
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code will reveal author identities. We will release the code and instructions if the paper is accepted.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Appendix
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We reported standard deviations in the experiment.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Appendix H.3
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See the tasks section for the simulators we use.
   (b) Did you mention the license of the assets? [No] Licenses have been included in the original simulator papers/urls/websites.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No] Our new asset only contains our source code. We will release it if the paper is accepted, for the reason stated above.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] The data/simulators are all virtual.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] The data/simulators are all virtual.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]