MITIGATING SUBOPTIMALITY OF DETERMINISTIC POLICY GRADIENTS IN COMPLEX Q-FUNCTIONS

Ayush Jain[1]*, **Norio Kosaka**[2], **Xinhu Li**[1], **Kyung-Min Kim**[3], **Erdem Bıyık**[1], **Joseph J Lim**[4]
[1]University of Southern California [2] Line Yahoo Corp [3]NAVER [4]KAIST

ABSTRACT

In reinforcement learning, off-policy actor-critic approaches like DDPG and TD3 are based on the deterministic policy gradient. Herein, the Q-function is trained from off-policy environment data and the actor (policy) is trained to maximize the Q-function via gradient ascent. We observe that in complex tasks like dexterous manipulation and restricted locomotion, the Q-value is a complex function of action, having several local optima or discontinuities. This poses a challenge for gradient ascent to traverse and makes the actor prone to get stuck at local optima. To address this, we introduce a new actor architecture that combines two simple insights: (i) use multiple actors and evaluate the Q-value maximizing action, and (ii) learn surrogates to the Q-function that are simpler to optimize with gradient-based methods. We evaluate tasks such as restricted locomotion, dexterous manipulation, and large discrete-action space recommender systems and show that our actor finds optimal actions more frequently and outperforms alternate actor architectures.

1 INTRODUCTION

In sequential decision-making, the goal is to build an optimal agent that maximizes the expected cumulative returns (Sondik, 1971; Littman, 1996). Value-based reinforcement learning (RL) approaches learn each action's expected returns with a Q-function and maximize it (Sutton & Barto, 1998). However, in continuous action spaces, evaluating the Q-value of every possible action is impractical. This necessitates an actor to globally maximize the Q-function and efficiently navigate the vast action space (Grondman et al., 2012). Yet, this is particularly challenging in tasks such as restricted locomotion, with a *non-convex* Q-function landscape with many local optima (Figure 2).

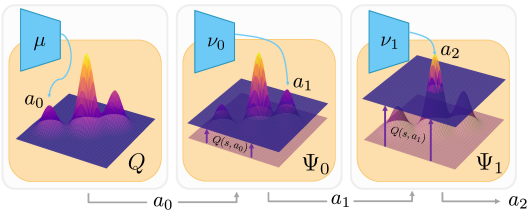


Figure 1: An actor μ trained with gradient ascent on a challenging Q-landscape gets stuck in local optima. Our approach learns a sequence of surrogates Ψ_i of the Q-function that successively prune out the Q-landscape below the current best Q-values, resulting in fewer local optima. Thus, the actors ν_i trained to ascend on these surrogates produce actions with a more optimal Q-value.

Can we build an actor architecture to find closer to optimal actions in such complex Q-landscapes? Prior methods perform a search over the action space with evolutionary algorithms like CEM (De Boer et al., 2005; Kalashnikov et al., 2018; Shao et al., 2022), but this requires numerous costly re-evaluations of the Q-function. To avoid this, deterministic policy gradient (DPG) algorithms (Silver et al., 2014), such as DDPG (Lillicrap et al., 2015) and TD3 (Fujimoto et al., 2018) train a parameterized actor to output actions with the objective of maximizing the Q-function *locally*.

A significant challenge arises in environments where the Q-function has many local optima, as shown in Figure 2. An actor trained via gradient ascent may converge to a local optimum with a much lower Q-value than the global maximum. This leads to *suboptimal* decisions during deployment and *sample-inefficient* training, as the agent fails to explore high-reward trajectories (Kakade, 2003).

To improve actors' ability to identify optimal actions in complex, non-convex Q-function landscapes, we propose the Successive Actors for Value Optimization (SAVO) algorithm. SAVO leverages two

*Correspondence to: ayushj@usc.edu.

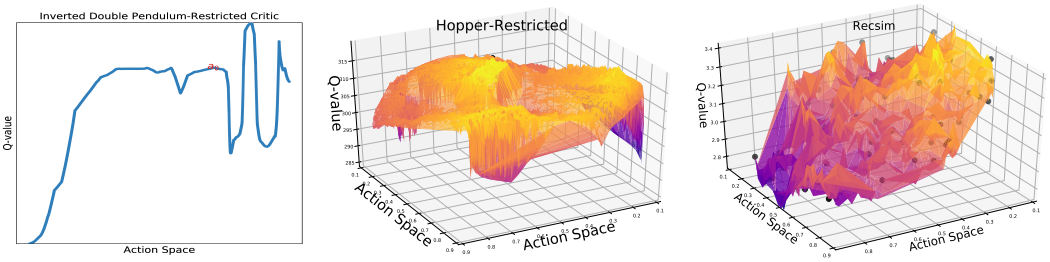


Figure 2: **Complex Q-landscapes.** We plot Q-value versus action a for some state. In control of Inverted-Double-Pendulum-Restricted (left) and Hopper-Restricted (middle), certain action ranges are unsafe, resulting in various locally optimal action peaks. In a large discrete-action recommendation system (right), there are local peaks at actions representing real items (black dots).

key insights: (1) combining multiple policies using an $\arg\max$ on their Q-values to construct a superior policy (§4.1), and (2) simplifying the Q-landscape by excluding lower Q-value regions based on high-performing actions, inspired by tabu search (Glover, 1990), thereby reducing local optima and facilitating gradient-ascent (§4.2). By iteratively applying these strategies through a sequence of simplified Q-landscapes and corresponding actors, SAVO progressively finds more optimal actions.

We evaluate SAVO in complex Q-landscapes such as (i) *continuous* control in dexterous manipulation (Rajeswaran et al., 2017) and restricted locomotion (Todorov et al., 2012), and (ii) *discrete* decision-making in the large action spaces of simulated (Ie et al., 2019) and real-data recommender systems (Harper & Konstan, 2015), and gridworld mining expedition (Chevalier-Boisvert et al., 2018). We use the reframing of large discrete action RL to continuous action RL following Van Hasselt & Wiering (2009) and Dulac-Arnold et al. (2015), where a policy acts in continuous actions, such as the feature space of recommender items (Figure 2), and the nearest discrete action is executed.

Our key contribution is SAVO, an actor architecture to find better optimal actions in complex non-convex Q-landscapes (§4). In experiments, we visualize how SAVO's successively learned Q-landscapes have fewer local optima (§6.2), making it more likely to find better action optima with gradient ascent. This enables SAVO to outperform alternative actor architectures, such as sampling more action candidates (Dulac-Arnold et al., 2015) and learning an ensemble of actors (Osband et al., 2016) (§6.1) across continuous and discrete action RL.

2 Related Work

Q-learning (Watkins & Dayan, 1992; Tesauro et al., 1995) is a fundamental value-based RL algorithm that iteratively updates Q-values to make optimal decisions. Deep Q-learning (Mnih et al., 2015) has been applied to tasks with manageable discrete action spaces, such as Atari (Mnih et al., 2013; Espeholt et al., 2018; Hessel et al., 2018), traffic control (Abdoos et al., 2011), and small-scale recommender systems (Chen et al., 2019). However, scaling Q-learning to continuous or large discrete action spaces requires specialized techniques to efficiently maximize the Q-function.

Analytical Q-optimization. Analytical optimization of certain Q-functions, such as wire fitting algorithm (Baird & Klopf, 1993) and normalized advantage functions (Gu et al., 2016; Wang et al., 2019), allows closed-form action maximization without an actor. Likewise, Amos et al. (2017) assume that the Q-function is convex in actions and use a convex solver for action selection. In contrast, the Q-functions considered in this paper are inherently non-convex in action space, making such an assumption invalid. Generally, analytical Q-functions lack the expressiveness of deep Q-networks (Hornik et al., 1989), making them unsuitable to model complex tasks like in Figure 2.

Evolutionary Algorithms for Q-optimization. Evolutionary algorithms like simulated annealing (Kirkpatrick et al., 1983), genetic algorithms (Srinivas & Patnaik, 1994), tabu search (Glover, 1990), and the cross-entropy method (CEM) (De Boer et al., 2005) are employed in RL for global optimization (Hu et al., 2007). Approaches such as QT-Opt (Kalashnikov et al., 2018; Lee et al., 2023; Kalashnikov et al., 2021) utilize CEM for action search, while hybrid actor-critic methods like CEM-RL (Pourchot & Sigaud, 2018), GRAC (Shao et al., 2022), and Cross-Entropy Guided Policies (Simmons-Edler et al., 2019) combine evolutionary techniques with gradient descent. Despite their effectiveness, CEM-based methods require numerous Q-function evaluations and struggle with

high-dimensional actions (Yan et al., 2019). In contrast, SAVO achieves superior performance with only a few (e.g., three) Q-evaluations, as demonstrated in experiments (§6).

Actor-Critic Methods with Gradient Ascent. Actor-critic methods can be on-policy (Williams, 1992; Schulman et al., 2015; 2017) primarily guided by the policy gradient of expected returns, or off-policy (Silver et al., 2014; Lillicrap et al., 2015; Fujimoto et al., 2018; Chen et al., 2020) primarily guided by the Bellman error on the critic. Deterministic Policy Gradient (DPG) (Silver et al., 2014) and its extensions like DDPG Lillicrap et al. (2015), TD3 (Fujimoto et al., 2018) and REDQ (Chen et al., 2020) optimize actors by following the critic's gradient. Soft Actor-Critic (SAC) (Haarnoja et al., 2018) extends DPG to stochastic actors. However, these methods can get trapped in local optima within the Q-function landscape. SAVO addresses this limitation by enhancing gradient-based actor training. This issue also affects stochastic actors, where a local optimum means an *action distribution* (instead of a single action) that fails to minimize the KL divergence from the Q-function density fully, and is a potential area for future research.

Sampling-Augmented Actor-Critic. Sampling multiple actions and evaluating their Q-values is a common strategy to find optimal actions. Greedy actor-critic (Neumann et al., 2018) samples high-entropy actions and trains the actor towards the best Q-valued action, yet remains susceptible to local optima. In large discrete action spaces, methods like Wolpertinger (Dulac-Arnold et al., 2015) use k-nearest neighbors to propose actions, requiring extensive Q-evaluations on up to 10% of total actions. In contrast, SAVO efficiently generates high-quality action proposals through successive actor improvements without being confined to local neighborhoods.

Ensemble-Augmented Actor-Critic. Ensembles of policies enhance exploration by providing diverse action proposals through varied initializations (Osband et al., 2016; Chen & Peng, 2019; Song et al., 2023; Zheng12 et al., 2018; Huang et al., 2017). The best action is selected based on Q-value evaluations. Unlike ensemble methods, SAVO systematically eliminates local optima, offering a more reliable optimization process for complex tasks (§6).

3 Problem Formulation

Our work tackles the effective optimization of the Q-value landscape in off-policy actor-critic methods for continuous and large-discrete action RL. We model a task as a Markov Decision Process (MDP), defined by a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma\}$ of states, actions, transition probabilities, reward function, and a discount factor. The action space $\mathcal{A} \subseteq \mathbb{R}^D$ is a D-dimensional *continuous* vector space. At every step t in the episode, the agent receives a state observation $s_t \in \mathcal{S}$ from the environment and acts with $a_t \in \mathcal{A}$. Then, it receives the new state after transition s_{t+1} and a reward r_t. The objective of the agent is to learn a policy $\pi(a \mid s)$ that maximizes the expected discounted reward, $\max_\pi \mathbb{E}_\pi \left[\sum_t \gamma^t r_t\right]$.

3.1 Deterministic Policy Gradients (DPG)

DPG (Silver et al., 2014) is an off-policy actor-critic algorithm that trains a deterministic actor μ_ϕ to maximize the Q-function. This happens via two steps of generalized policy iteration, GPI (Sutton & Barto, 1998): policy evaluation estimates the Q-function (Bellman, 1966) and policy improvement greedily maximizes the Q-function. To approximate the $\arg\max$ over continuous actions in Eq. (2), DPG proposes the policy gradient to update the actor locally in the direction of increasing Q-value,

$$Q^\mu(s,a) = r(s,a) + \gamma \mathbb{E}_{s'}\left[Q^\mu(s', \mu(s'))\right], \tag{1}$$

$$\mu(s) = \arg\max_a Q^\mu(s,a), \tag{2}$$

$$\nabla_\phi J(\phi) = \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_a Q^\mu(s,a)\big|_{a=\mu(s)} \nabla_\phi \mu_\phi(s)\right]. \tag{3}$$

DDPG (Lillicrap et al., 2015) and TD3 (Fujimoto et al., 2018) made DPG compatible with deep networks via techniques like experience replay and target networks to address non-stationarity of online RL, twin critics to mitigate overestimation bias, target policy smoothing to prevent exploitation of errors in the Q-function, and delayed policy updates so critic is reliable to provide actor gradients.

3.2 The Challenge of an Actor Maximizing a Complex Q-landscape

DPG-based algorithms train the actor following the chain rule in Eq. (3). Specifically, its first term, $\nabla_a Q^\mu(s,a)$ involves gradient ascent in Q-versus-a landscape. This Q-landscape is often highly non-convex (Figures 2, 3) and non-stationary because of its own training. This makes the actor's

output $\mu(s)$ get stuck at suboptimal Q-values, thus leading to insufficient policy improvement in Eq. (2). We can define the suboptimality of the μ w.r.t. Q^μ at state s as

$$\Delta(Q^\mu, \mu, s) = \arg\max_a Q^\mu(s, a) - Q^\mu(s, \mu(s)) \geq 0. \tag{4}$$

Suboptimality in actors is a crucial problem because it leads to (i) **poor sample efficiency** by slowing down GPI, and (ii) **poor inference performance** even with an optimal Q-function, Q^* as seen in Figure 3 where a TD3 actor gets stuck at a locally optimum action a_0 in the final Q-function.

This challenge fundamentally differs from the well-studied field of non-convex optimization, where non-convexity arises in the *loss function w.r.t. the model parameters* (Goodfellow, 2016). In those cases, stochastic gradient-based optimization methods like SGD and Adam (Kingma & Ba, 2014) are effective at finding acceptable local minima due to the smoothness and high dimensionality of the parameter space, which often allows for escape from poor local optima (Choromanska et al., 2015). Moreover, overparameterization in deep networks can lead to loss landscapes with numerous good minima (Neyshabur et al., 2017).

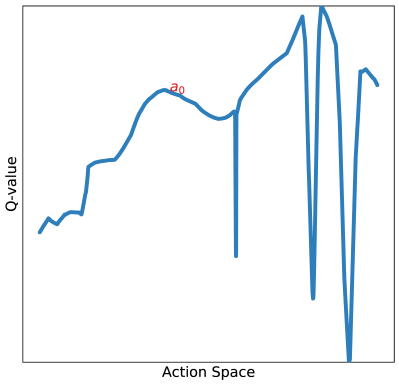


Figure 3: Non-convex Q-landscape in Inverted-Pendulum-Restricted leads to a suboptimally converged actor.

In contrast, our challenge involves non-convexity in the *Q-function w.r.t. the action space*. The actor's task is to find, for every state s, the action a that maximizes $Q^\mu(s, a)$. Since the Q-function can be highly non-convex and multimodal in a, the gradient ascent step $\nabla_a Q^\mu(s, a)$ used in Eq. (3) may lead the actor to converge to suboptimal local maxima in action space. Unlike parameter space optimization, the actor cannot rely on high dimensionality or overparameterization to smooth out the optimization landscape in action space because the Q-landscape is determined by the task's reward. Furthermore, the non-stationarity of the Q-function during training compounds this challenge. These properties make our non-convex challenge unique, requiring a specialized actor to navigate the complex Q-landscape.

Tasks with several local optima in the Q-function include restricted inverted pendulum shown in Figure 3, where certain regions of the action space are invalid or unsafe, leading to a rugged Q-landscape (Florence et al., 2022). Dexterous manipulation tasks exhibit discontinuous behaviors like inserting a precise peg in place with a small region of high-valued actions (Rajeswaran et al., 2017) and surgical robotics have a high variance in Q-values of nearby motions (Barnoy et al., 2021).

3.2.1 Large Discrete Action RL Reframed as Continuous Action RL

We discuss another practical domain where non-convex Q-functions are present. In large discrete action tasks like recommender systems (Zhao et al., 2018; Zou et al., 2019; Wu et al., 2017), a common approach (Van Hasselt & Wiering, 2009; Dulac-Arnold et al., 2015) is to use continuous representations of actions as a medium of decision-making. Given a set of actions, $\mathcal{I} = \{\mathscr{I}_1, \ldots, \mathscr{I}_N\}$, a predefined module $\mathcal{R} : \mathcal{I} \to \mathcal{A}$ assigns each $\mathscr{I} \in \mathcal{I}$ to its representation $\mathcal{R}(\mathscr{I})$, e.g., text embedding of a given movie (Zhou et al., 2010). A continuous action policy $\pi(a \mid s)$ is learned in the action representation space, with each $a \in \mathcal{A}$ converted to a discrete action $\mathscr{I} \in \mathcal{I}$ via nearest neighbor,

$$f_{\text{NN}}(a) = \arg\min_{\mathscr{I}_i \in \mathcal{I}} \|\mathcal{R}(\mathscr{I}_i) - a\|_2.$$

Importantly, the nearest neighbor operation creates a challenging piece-wise continuous Q-function with suboptima at various discrete points as shown in Figure 2 (Jain et al., 2021; 2020).

4 Approach: Successive Actors for Value Optimization (SAVO)

Our objective is to design an actor architecture that efficiently discovers better actions in complex, non-convex Q-function landscapes. We focus on gradient-based actors and introduce two key ideas:

1. **Maximizing Over Multiple Policies:** By combining policies using an $\arg\max$ over their Q-values, we can construct a policy that performs at least as well as any individual policy (§4.1).
2. **Simplifying the Q-Landscape:** Drawing inspiration from tabu search (Glover, 1990), we propose using actions with good Q-values to eliminate or "tabu" the Q-function regions with lower Q-values, thereby reducing local optima and facilitating gradient-based optimization (§4.2).

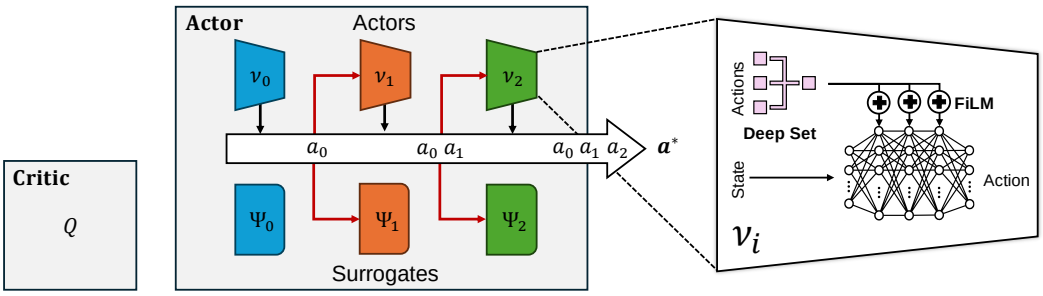


Figure 4: **SAVO Architecture.** (left) Q-network is unchanged. (center) Instead of a single actor, we learn a sequence of actors and surrogate networks connected via action predictions. (right) Conditioning on previous actions is done with the help of a deep-set summarizer and FiLM modulation.

4.1 Maximizer Actor over Action Proposals

We first show how additional actors can improve DPG's policy improvement step. Given a policy μ being trained with DPG over Q, consider k additional arbitrary policies $\nu_1, \ldots, \nu_k$, where $\nu_i : \mathcal{S} \to \mathcal{A}$ and let $\nu_0 = \mu$. We define a maximizer actor μ_M for $a_i = \nu_i(s)$ for $i = 0, 1, \ldots, k$,

$$\mu_M(s) := \underset{a \in \{a_0, a_1, \ldots, a_k\}}{\arg\max} Q(s, a), \tag{5}$$

Here, μ_M is shown to be a better maximizer of $Q(s, a)$ in Eq. (2) than $\mu\ \forall s$:

$$Q(s, \mu_M(s)) = \max_{a_i} Q(s, a_i) \geq Q(s, a_0) = Q(s, \mu(s)).$$

Therefore, by policy improvement theorem (Sutton & Barto, 1998), $V^{\mu_M}(s) \geq V^{\mu}(s)$, proving that μ_M is better than a single μ for a given Q. Appendix A proves the following theorem by showing that policy evaluation and improvement with μ_M converge.

Theorem 4.1 (Convergence of Policy Iteration with Maximizer Actor)**.** *A modified policy iteration algorithm where $\nu_0 = \mu$ is the current policy learned with DPG and* maximizer actor μ_M *defined in Eq. (5), converges in the tabular setting to the optimal policy.*

This algorithm is valid for arbitrary $\nu_1, \ldots \nu_k$. We experiment with ν's obtained by **sampling** from a Gaussian centered at μ or **ensembling** on μ to get diverse actions. However, in high-dimensionality, *randomness* around μ is not sufficient to get action proposals to significantly improve μ.

4.2 Successive Surrogates to Reduce Local Optima

To train additional policies ν_i that can improve upon μ_M, we introduce *surrogate* Q-functions with fewer local optima, inspired by the principles of tabu search (Glover, 1990), which is an optimization technique that uses memory structures to avoid revisiting previously explored inferior solutions, thereby enhancing the search for optimal solutions. Similarly, our surrogate functions act as memory mechanisms that "tabu" certain regions of the Q-function landscape deemed suboptimal based on previously identified good actions. Given a known action $a^\dagger$, we define a surrogate function that elevates the Q-values of all inferior actions to $Q(s, a^\dagger)$, which serves as a constant threshold:

$$\Psi(s, a; a^\dagger) = \max\{Q(s, a), Q(s, a^\dagger)\}. \tag{6}$$

Extending this idea, we define a sequence of surrogate functions using the actions from previous policies. Let $a_{<i} = \{a_0, a_1, \ldots, a_{i-1}\}$. The i-th surrogate function is:

$$\Psi_i(s, a; a_{<i}) = \max\left\{Q(s, a), \max_{j<i} Q(s, a_j)\right\}. \tag{7}$$

Theorem 4.2. *For a state $s \in \mathcal{S}$ and surrogates Ψ_i defined as above, the number of local optima decreases with each successive surrogate:*

$$N_{opt}(Q(s, \cdot)) \geq N_{opt}(\Psi_1(s, \cdot; a_0)) \geq \cdots \geq N_{opt}(\Psi_k(s, \cdot; a_{<k})),$$

where $N_{opt}(f)$ denotes the number of local optima of function f over $\mathcal{A}$.

Proof Sketch. As $\Psi_i \to \Psi_{i+1}$, the anchor Q-value in Eq. (7) weakly increases, $\max_{j<i} Q(s, a_j) \leq \max_{j<(i+1)} Q(s, a_j)$, thus, eliminating more local minima below it (proof in Appendix B.1). □

4.3 Successive Actors for Surrogate Optimization

To effectively reduce local optima using the surrogates $\Psi_1, \ldots, \Psi_k$, we design the policies ν_i to optimize their respective surrogates $\Psi_i(s, a; a_{<i})$. Each ν_i focuses on regions where $Q(s, a) \geq \max_{j<i} Q(s, a_j)$, allowing it to find better optima than previous policies. The actor ν_i is conditioned on previous actions $\{a_0, \ldots, a_{i-1}\}$, summarized using deep sets (Zaheer et al., 2017) (Figure 4). The maximizer actor μ_M (Eq. (5)) selects the best action among all proposals.

We train each actor ν_i using gradient ascent on its surrogate Ψ_i, similar to DPG:

$$\nabla_{\phi_i} J(\phi_i) = \mathbb{E}_{s\sim\rho^{\mu_M}} \left[\nabla_a \Psi_i(s, a; a_{<i})\big|_a \nabla_{\phi_i} \nu_i(s; a_{<i}) \right]. \tag{8}$$

4.4 Approximate Surrogate Functions

The surrogates Ψ_i have zero gradients when $Q(s, a) < \tau$, where $\tau = \max_{j<i} Q(s, a_j)$,

$$\nabla_a \Psi_i(s, a; a_{<i}) = \begin{cases} \nabla_a Q^{\mu_M}(s, a) & \text{if } Q(s, a) \geq \tau, \\ 0 & \text{if } Q(s, a) < \tau. \end{cases}$$

This means the policy gradient only updates ν_i when $Q(s, a) \geq \tau$, which may slow down learning. To address this issue, we ease the gradient flow by learning a smooth lossy approximation $\hat{\Psi}_i$ of Ψ_i.

We approximate each surrogate Ψ_i with a neural network $\hat{\Psi}_i$. This approach leverages the universal approximation theorem (Hornik et al., 1989; Cybenko, 1989) and benefits from empirical evidence that deep networks can effectively learn non-smooth functions (Imaizumi & Fukumizu, 2019). The smooth surrogate $\hat{\Psi}_i$ enables continuous gradient propagation, which is essential for optimizing the actors ν_i. We train $\hat{\Psi}_i$ to approach Ψ_i by minimizing the mean squared error at two critical points:

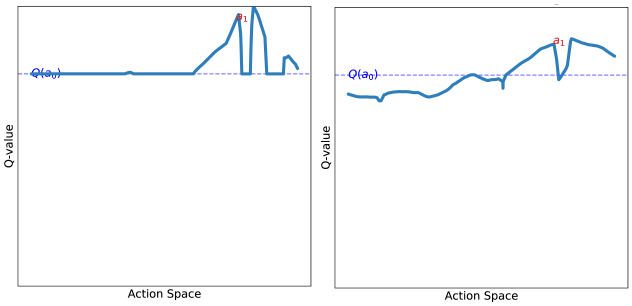


Figure 5: In restricted inverted pendulum, given an anchor $Q(a_0)$ value, Ψ (left) has some zero-gradient surfaces which $\hat{\Psi}$ (right) *approximately* follows while allowing non-zero gradients towards high Q-values to flow into its actor ν.

1. $\tilde{\mu}_M(s)$ is the action selected by the current maximizer actor μ_M, having a high Q-value,
2. $\nu_i(s; a_{<i})$ is the action proposed by the i-th actor conditioned on previous actions $a_{<i}$,

$$\mathcal{L}_{\text{approx}} = \mathbb{E}_{s\sim\rho^{\mu_M}} \left[\sum_{a\in\{\tilde{\mu}_M(s), \nu_i(s; a_{<i})\}} \left\| \hat{\Psi}_i(s, a; a_{<i}) - \Psi_i(s, a; a_{<i}) \right\|_2^2 \right]. \tag{9}$$

This design ensures $\hat{\Psi}_i$ is updated on high Q-value actions and thus the landscape is biased towards those values. This makes the gradient flow trend in the direction of high Q-values. So, even when a_i from ν_i falls in a region of zero gradients for Ψ_i, in $\hat{\Psi}_i$ would provide policy gradient in a higher Q-value direction, if it exists. Figure 5 shows Ψ_1 and $\hat{\Psi}_1$ in restricted inverted pendulum task.

4.5 SAVO-TD3 Algorithm and Design Choices

While the SAVO architecture (Figure 4) can be integrated with any off-policy actor-critic algorithm, we choose to implement it with TD3 (Fujimoto et al., 2018) due to its compatibility with continuous and large-discrete action RL (Dulac-Arnold et al., 2015). Using the SAVO actor in TD3 enhances its ability to find better actions in complex Q-function landscapes. Algorithm 1 depicts SAVO (highlighted) applied to TD3. We discuss design choices in SAVO and validate them in §6.

1. **Removing policy smoothing**: We eliminate TD3's policy smoothing, which adds noise to the target action $\tilde{a}$ during critic updates. In non-convex landscapes, nearby actions may have significantly different Q-values and noise addition might obscure important variations.

2. **Exploration in Additional Actors**: Added actors ν_i explore the surrogate landscapes for high-reward regions by adding OU (Lillicrap et al., 2015) or Gaussian (Fujimoto et al., 2018) noise to their actions.

3. **Twin Critics for Surrogates**: To prevent overestimation bias in surrogates $\hat{\Psi}_i$, we use twin critics to compute the target of each surrogate, mirroring TD3.

4. **Conditioning on Previous Actions**: Actors ν_i and surrogates $\hat{\Psi}_i$ are conditioned on preceding actions via FiLM layers (Perez et al., 2018) as in Figure 4.

5. **Discrete Action Space Tasks**: We apply 1-nearest-neighbor f_{NN} before Q-value evaluation to ensure the Q-function is only queried at in-distribution actions.

SAVO-TD3 employs SAVO actor to systematically reduce the local optima in its base algorithm TD3. We empirically validate the proposed design improvements.

Algorithm 1 SAVO-TD3

Initialize $Q, Q_2, \mu, \quad \hat{\Psi}_1, \ldots, \hat{\Psi}_k, \quad \nu_1, \ldots, \nu_k$
Initialize target networks $Q' \leftarrow Q$, $Q'_2 \leftarrow Q_{twin}$
Initialize replace buffer $\mathcal{B}$.
for timestep $t = 1$ to T **do**
  Select Action:
  Evaluate $a_0 = \mu(s), a_i = \nu_i(s; a_{<i})$
  Add perturbations with OU Noise $\hat{a}_i = a_i + \epsilon_i$
  Evaluate $\mu_M(s) = \arg\max_{a \in \{\hat{a}_0, \ldots, \hat{a}_k\}} Q^{\mu}(s, a)$
  Exploration action $a = \tilde{\mu}_M(s) = \mu_M(s) + \epsilon$
  Observe reward r and new state s'
  Store $(s, a, \{\hat{a}_i\}_{i=0}^{K}, r, s')$ in $\mathcal{B}$
  Update:
  Sample N transitions $(s, a, \{\hat{a}_i\}_{i=0}^{K}, r, s')$ from $\mathcal{B}$
  Compute target action $\tilde{a} = \mu_M(s')$
  Update $Q, Q_2 \leftarrow r + \gamma \min\{Q'(s', \tilde{a}), Q'_2(s', \tilde{a})\}$

  Update $\hat{\Psi}_i$ with Eq. 9 $\forall i = 1, \ldots k$
  Update actor μ with Eq. 3
  Update actor ν_i with Eq. 8 $\forall i = 1, \ldots k$
end for

5 Environments

We evaluate SAVO on discrete and continuous action space environments with challenging Q-value landscapes. More environment details are presented in Appendix C and Figure 13.

Locomotion in Mujoco. We evaluate on MuJoCo (Todorov et al., 2012) environments of Hopper-v4, Walker2D-v4, Inverted Pendulum-v4, and Inverted Double Pendulum-v4.

Locomotion in Restricted Mujoco. We create a restricted locomotion suite of the same environments as in MuJoCo. A *hard* Q-landscape is realized via high-dimensional discontinuities that restrict the action space. Concretely, a set of predefined hyper-spheres (as shown in Figure 6) in the action space are sampled and set to be valid actions, while the other invalid actions have a null effect if selected. The complete details can be found in Appendix C.3.1.

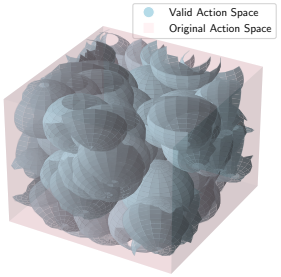
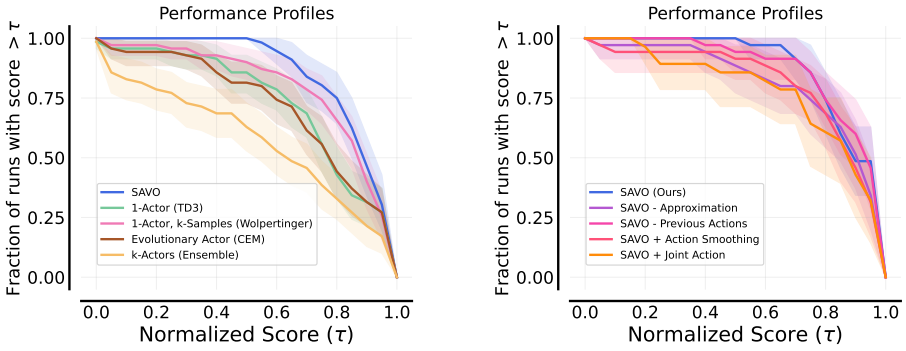


Figure 6: Hopper's 3D visualization of Action Space.

Adroit Dexterous Manipulation. Rajeswaran et al. (2017) propose manipulation tasks with a dexterous multi-fingered hand. *Door*: In this task, a robotic hand is required to open a door with a latch. The challenge lies in the precise manipulation needed to unlatch and swing open the door using the fingers. *Hammer*: the robotic hand must use a hammer to drive a nail into a board. This task tests the hand's ability to grasp the hammer correctly and apply force accurately to achieve the goal. *Pen*: This task involves the robotic hand manipulating a pen to reach a specific goal position and rotation. The objective is to control the pen's orientation and position using fingers, which demands fine motor skills and coordination.

Mining Expedition in Grid World. We develop a 2D Mining grid world environment (Chevalier-Boisvert et al., 2018) where the agent (Figure 13) navigates a 2D maze to reach the goal, removing mines with correct pick-axe tools to reach the goal in the shortest path. The action space includes navigation and tool-choice actions, with a procedurally-defined action representation space. The Q-landscape is non-convex because of the diverse effects of nearby action representations.

Simulated and Real-Data Recommender Systems. RecSim (Ie et al., 2019) simulates sequential user interactions in a recommender system with a large discrete action space. The agent must recommend the most relevant item from a set of 10,000 items based on user preference information. The action representations are simulated item characteristic vectors in simulated and movie review embeddings in the real-data task based on MovieLens (Harper & Konstan, 2015) for items.



(a) SAVO versus baseline actor architectures.

(b) SAVO versus ablations of SAVO

Figure 7: Aggregate performance profiles using normalized scores over 7 tasks and 10 seeds each.

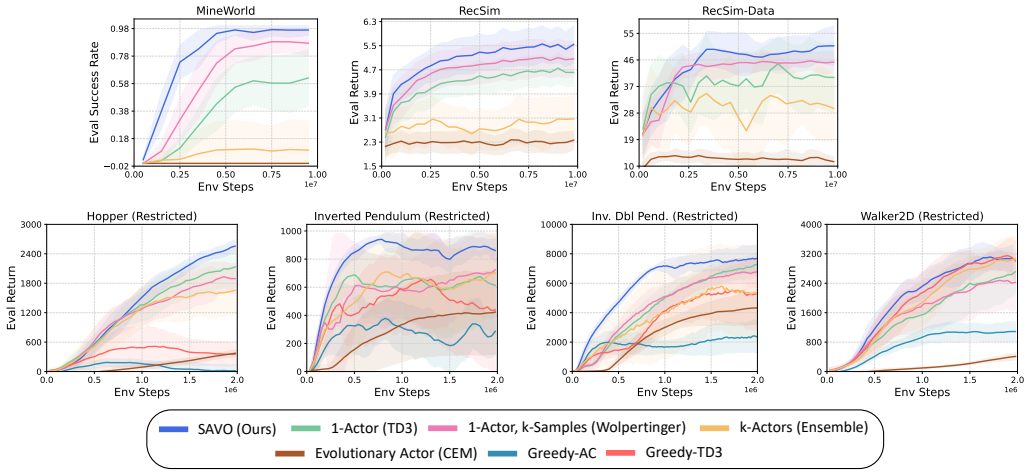


Figure 8: SAVO against baselines on discrete and continuous tasks. Results averaged over 10 seeds.

6 Experiments

6.1 Effectiveness of SAVO in challenging Q-landscapes

We compare SAVO against the following baseline actor architectures, where $k = 3$:

- **1-Actor (TD3)**: Conventional single actor architecture which is susceptible to local optima.
- **1-Actor, k samples (Wolpertinger)**: Gaussian sampling centered on actor's output. For discrete actions, we select k-NN discrete actions around the continuous action (Dulac-Arnold et al., 2015).
- **k-Actors (Ensemble)**: Best Q-value among diverse actions from ensemble (Osband et al., 2016).
- **Evolutionary actor (CEM)**: Iterative CEM search over the action space (Kalashnikov et al., 2018).
- **Greedy-AC**: Greedy Actor Critic (Neumann et al., 2018) trains a high-entropy proposal policy and primary actor trained from best proposals with gradient updates.
- **Greedy TD3**: Our version of Greedy-AC with TD3 exploration and update improvements.
- **SAVO**: Our method with k successive actors and surrogate Q-landscapes.

We ablate the crucial components and design decisions in SAVO:

- **SAVO - Approximation**: removes the approximate surrogates (§4.4), using Ψ_i instead of $\hat{\Psi}_i$.
- **SAVO - Previous Actions**: removes conditioning on $a_{<i}$ in SAVO's actors and surrogates.
- **SAVO + Action Smoothing**: TD3's policy smoothing (Fujimoto et al., 2018) compute Q-targets.
- **SAVO + Joint Action**: trains an actor with a joint action space of $3 \times D$. The k action samples are obtained by splitting the joint action into D dimensions. Validates successive nature of SAVO.

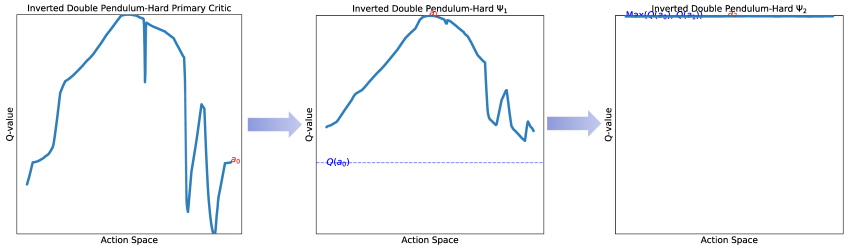
Aggregate performance. We utilize performance profiles (Agarwal et al., 2021) to aggregate results across different environments in Figure 7a (evaluation mechanism detailed in Appendix G.1). SAVO

consistently outperforms baseline actor architectures like single-actor (TD3) and sampling-augmented actor (Wolpertinger), showing wide robustness across challenging Q-landscapes. In Figure 7b, SAVO outperforms its ablations, validating each proposed component and design decision.

Per-environment results. In discrete action tasks, the Q-value landscape is only well-defined at exact action representations and nearby discrete actions might have very different values (§3.2.1). This makes the Q-value landscape uneven, with multiple peaks and valleys (Figure 2). For example, actions in Mining Expedition involve both navigation and tool-selection which are quite different, while RecSim and RecSim-Data have many diverse items to choose from. Methods like Wolpertinger that sample many actions a local neighborhood perform better than TD3 which considers a single action (Figure 8). However, SAVO achieves the best performance by directly simplifying the non-convex Q-landscape. In restricted locomotion, there are several good actions that are far apart. SAVO actors can search and explore widely to optimize the Q-landscape better than only nearby sampled actions. Figure 16 ablates SAVO in all 7 environments and shows that the most critical features are its successive nature, removing policy smoothing, and approximate surrogates.

6.2 Q-Landscape Analysis: Do successive surrogates reduce local optima?

Figure 9 visualizes surrogate landscapes in Inverted-Pendulum-Restricted for a given state s. Successive pruning and approximation smooth the Q-landscapes, reducing local optima. A single actor gets stuck at a local optimum a_0 (left), but surrogate $\hat{\Psi}_1$ uses a_0 as an anchor to find a better optimum a_1. The maximizer policy finally selects the highest Q-valued action among a_0, a_1, a_2. Figure 24 extends this visualization to Inverted-Double-Pendulum-Restricted. Figure 23 shows how one actor is sufficient in the *convex* Q-landscape of unrestricted Inverted-Pendulum-v4. Figures 25, 26 show how Hopper-v4 Q-landscape provides a path to global optimum, while Hopper-Restricted is non-convex.



(a) $Q(s, a_0)$ (b) $\hat{\Psi}_1(s, a_1; a_0)$ (c) $\hat{\Psi}_2(s, a_2; \{a_0, a_1\})$

Figure 9: Each successive surrogate learns a Q-landscape with fewer local optima and thus is easier to optimize by its actor. SAVO helps a single actor escape the local optimum a_0 in Inverted Pendulum.

6.3 Challenging Dexterous Manipulation (Adroit)

In Adroit dexterous manipulation tasks (Door, Pen, Hammer) (Rajeswaran et al., 2017), SAVO improves the sample efficiency of TD3 (Figure 10). The non-convexity in Q-landscape likely arises from nearby actions having high variance outcomes like grasping, missing, dropping, or no impact.

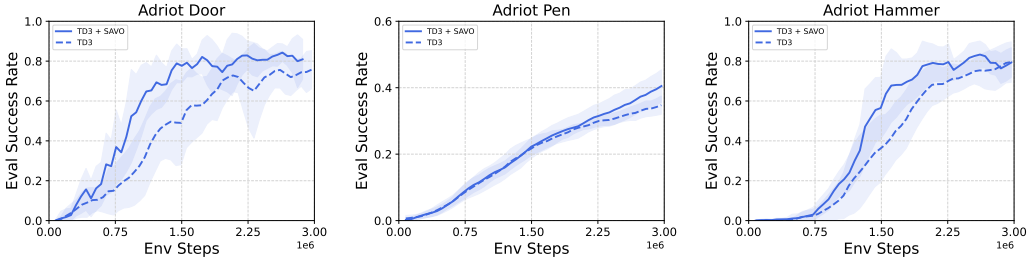
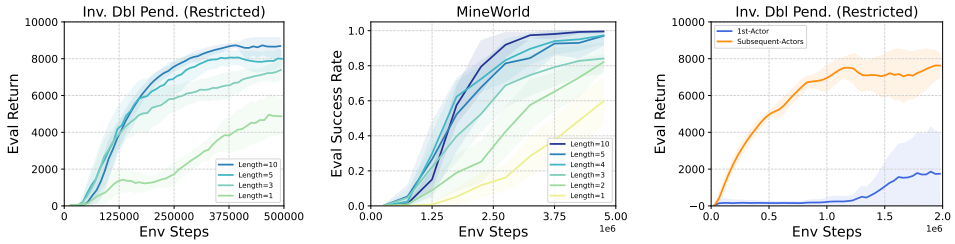


Figure 10: SAVO improves the sample-efficiency of TD3 on Adroit dexterous manipulation tasks.

6.4 Quantitative Analysis: The Effect of Successive Actors and Surrogates

We investigate the effect of increasing the number of successive actor-surrogates in SAVO in Pendulum (Figure 11a) and MineWorld (Figure 11b). Additional actor-surrogates significantly help to reduce severe local optima initially. However, the improvement saturates as the suboptimality gap reduces.



(a) Length Analysis Pendulum (b) Length analysis Mine World (c) Return Improvement

Figure 11: (L) More successive actor-surrogates are better, (R) SAVO v/s single-actor on inference.

Next, we show that successive actors are needed because a single actor can get stuck in local optima even with an optimal Q-function. In Figure 11c, we consider a SAVO agent trained to optimality with 3 actors. When we remove the additional actors, the remaining single-actor agent resembles TD3 trained to maximize an "optimal" Q-function. However, the significant performance gap indicates that the single actor could not find optimal actions for the given Q-function.

6.5 Does RL with Resets address the issue of Q-function Optimization?

Primacy bias (Nikishin et al., 2022; Kim et al., 2024) occurs when an agent is trapped in suboptimal behaviors from early training. To mitigate this, methods like resetting parameters and re-learning from the replay buffer aim to reduce reliance on initial samples. We run TD3 in MineEnv with either a full-reset or last-layer reset every 200k, 500k, or 1 million iterations. None of these versions outperformed the original TD3 algorithm without resets. This is because resetting does not help an actor to navigate the Q-landscape better and can even cause an otherwise optimal actor to get stuck in a suboptimal solution during retraining. In contrast, the SAVO actor architecture specifically addresses the non-convex Q-landscapes, being a more robust method to finding closer to optimal actions.

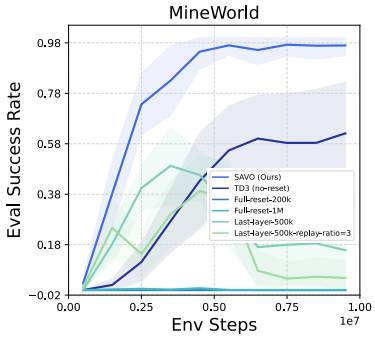


Figure 12: Reset (primacy bias) does not improve Q-optimization.

6.6 Further experiments to validate SAVO

- **Unrestricted locomotion.** Figure 15 shows that both SAVO and baselines achieve optimal performance in simple Q-landscapes, confirming effective hyperparameter tuning (§G.4, §G.3) and indicating that the baselines underperform due to the complexity introduced in Q-landscapes.
- **SAVO orthogonal to SAC.** Figure 17 shows that SAVO+TD3 > SAC > TD3, indicating that SAC's stochastic policy does not address non-convexity, but can itself suffer from local optima (Figure 18)
- **Design Choices.** Figure 20 shows that LSTM, DeepSet, and Transformers are all valid choices as summarizers of preceding actions $a_{<i}$ in SAVO. Figure 21 shows that FiLM conditioning on $a_{<i}$ especially helps for discrete action space tasks but has a smaller effect in continuous action space. In Figure 22a, we find Ornstein-Uhlenbeck (OU) noise and Gaussian noise to be largely equivalent.
- **Massive Discrete Actions.** SAVO outperforms in RecSim with 100k and 500k actions (Figure 19).

7 Limitations and Conclusion

Method	GPU Mem.	Return	Time
TD3	619MB	1107.795	0.062s
SAVO k=3	640MB	2927.149	0.088s
SAVO k=5	681MB	3517.319	0.122s

Table 1: Compute v/s Performance Gain (Mujoco)

Introducing more actors in SAVO has negligible influence on GPU memory, but leads to longer inference time (Table 1). However, even for 3 actor-surrogates, SAVO achieves significant improvements in all our experiments. Further, for tasks with a simple convex Q-landscape, a single actor does not get stuck in local optima, making the gain from SAVO negligible. In conclusion, we improve Q-landscape optimization in deterministic policy gradient RL with Successive Actors for Value Optimization (SAVO) in both continuous and large discrete action spaces. We demonstrate with quantitative and qualitative analyses how the improved optimization of Q-landscape with SAVO leads to better sample efficiency and performance.

ACKNOWLEDGEMENTS

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant (No.RS-2019-II190075, Artificial Intelligence Graduate School Program, KAIST) and National Research Foundation of Korea (NRF) grant (NRF-2021H1D3A2A03103683, Brain Pool Research Program), funded by the Korea government (MSIT). Ayush Jain was supported partly as intern at Naver AI Lab during the initiation of the project. We appreciate the fruitful discussions with members of CLVR Lab at KAIST and LiraLab at USC.

BIBLIOGRAPHY

Monireh Abdoos, Nasser Mozayani, and Ana LC Bazzan. Traffic light control in non-stationary environments based on multi agent q-learning. In *2011 14th International IEEE conference on intelligent transportation systems (ITSC)*, pp. 1580–1585. IEEE, 2011.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

Brandon Amos, Lei Xu, and J Zico Kolter. Input convex neural networks. In *International conference on machine learning*, pp. 146–155. PMLR, 2017.

Leemon C Baird and A Harry Klopf. Reinforcement learning with high-dimensional continuous actions. *Wright Laboratory, Wright-Patterson Air Force Base, Tech. Rep. WL-TR-93-1147*, 15, 1993.

Yotam Barnoy, Molly O'Brien, Will Wang, and Gregory Hager. Robotic surgery with lean reinforcement learning. *arXiv preprint arXiv:2105.01006*, 2021.

Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

Lukas Biewald. Experiment tracking with weights and biases. *Software available from wandb. com*, 2:233, 2020.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Gang Chen and Yiming Peng. Off-policy actor-critic in an ensemble: Achieving maximum general entropy and effective environment exploration in deep reinforcement learning. *arXiv preprint arXiv:1902.05551*, 2019.

Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. Generative adversarial user model for reinforcement learning based recommendation system. In *International Conference on Machine Learning*, pp. 1052–1061. PMLR, 2019.

Xinyue Chen, Che Wang, Zijian Zhou, and Keith W Ross. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2020.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

Anna Choromanska, MIkael Henaff, Michael Mathieu, Gerard Ben Arous, and Yann LeCun. The Loss Surfaces of Multilayer Networks. In Guy Lebanon and S. V. N. Vishwanathan (eds.), *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pp. 192–204, San Diego, California, USA, 09–12 May 2015. PMLR. URL https://proceedings.mlr.press/v38/choromanska15.html.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pp. 158–168. PMLR, 2022.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Fred Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, 1990.

Ian Goodfellow. Deep learning, 2016.

Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, part C (applications and reviews)*, 42(6):1291–1307, 2012.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International conference on machine learning*, pp. 2829–2838. PMLR, 2016.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Jiaqiao Hu, Michael C Fu, and Steven I Marcus. A model reference adaptive search method for global optimization. *Operations research*, 55(3):549–568, 2007.

Zhewei Huang, Shuchang Zhou, BoEr Zhuang, and Xinyu Zhou. Learning to run with actor-critic ensemble. *arXiv preprint arXiv:1712.08987*, 2017.

Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging, 2015.

Eugene Ie, Chih-wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. Recsim: A configurable simulation platform for recommender systems. *arXiv preprint arXiv:1909.04847*, 2019.

Masaaki Imaizumi and Kenji Fukumizu. Deep neural networks learn non-smooth functions effectively. In *The 22nd international conference on artificial intelligence and statistics*, pp. 869–878. PMLR, 2019.

Ayush Jain, Andrew Szot, and Joseph Lim. Generalization to new actions in reinforcement learning. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4661–4672. PMLR, 13–18 Jul 2020. URL http://proceedings.mlr.press/v119/jain20b.html.

Ayush Jain, Norio Kosaka, Kyung-Min Kim, and Joseph J Lim. Know your action set: Learning action relations for reinforcement learning. In *International Conference on Learning Representations*, 2021.

Sham Machandranath Kakade. *On the sample complexity of reinforcement learning*. University of London, University College London (United Kingdom), 2003.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pp. 651–673. PMLR, 2018.

Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*, 2021.

Woojun Kim, Yongjae Shin, Jongeui Park, and Youngchul Sung. Sample-efficient and safe deep reinforcement learning via reset deep ensemble agents. *Advances in Neural Information Processing Systems*, 36, 2024.

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

Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

Kuang-Huei Lee, Ted Xiao, Adrian Li, Paul Wohlhart, Ian Fischer, and Yao Lu. Pi-qt-opt: Predictive information improves multi-task robotic reinforcement learning at scale. In *Conference on Robot Learning*, pp. 1696–1707. PMLR, 2023.

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

Michael Lederman Littman. *Algorithms for sequential decision-making*. Brown University, 1996.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

Samuel Neumann, Sungsu Lim, Ajin Joseph, Yangchen Pan, Adam White, and Martha White. Greedy actor-critic: A new conditional cross-entropy method for policy improvement. *arXiv preprint arXiv:1810.09103*, 2018.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. *Advances in neural information processing systems*, 30, 2017.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International conference on machine learning*, pp. 16828–16847. PMLR, 2022.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29, 2016.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Aloïs Pourchot and Olivier Sigaud. Cem-rl: Combining evolutionary and gradient-based methods for policy search. *arXiv preprint arXiv:1810.01222*, 2018.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Lin Shao, Yifan You, Mengyuan Yan, Shenli Yuan, Qingyun Sun, and Jeannette Bohg. Grac: Self-guided and self-regularized actor-critic. In *Conference on Robot Learning*, pp. 267–276. PMLR, 2022.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pp. 387–395. Pmlr, 2014.

Riley Simmons-Edler, Ben Eisner, Eric Mitchell, Sebastian Seung, and Daniel Lee. Q-learning for continuous actions with cross-entropy guided policies. *arXiv preprint arXiv:1903.10605*, 2019.

Edward Jay Sondik. *The optimal control of partially observable Markov processes*. Stanford University, 1971.

Yanjie Song, Ponnuthurai Nagaratnam Suganthan, Witold Pedrycz, Junwei Ou, Yongming He, Yingwu Chen, and Yutong Wu. Ensemble reinforcement learning: A survey. *Applied Soft Computing*, pp. 110975, 2023.

Mandavilli Srinivas and Lalit M Patnaik. Genetic algorithms: A survey. *computer*, 27(6):17–26, 1994.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

Gerald Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38 (3):58–68, 1995.

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

Hado Van Hasselt and Marco A Wiering. Using continuous action spaces to solve discrete problems. In *2009 International Joint Conference on Neural Networks*, pp. 1149–1156. IEEE, 2009.

Pin Wang, Hanhan Li, and Ching-Yao Chan. Quadratic q-network for learning continuous control for autonomous vehicles. *arXiv preprint arXiv:1912.00074*, 2019.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Qingyun Wu, Hongning Wang, Liangjie Hong, and Yue Shi. Returning is believing: Optimizing long-term user engagement in recommender systems. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 1927–1936, 2017.

Mengyuan Yan, Adrian Li, Mrinal Kalakrishnan, and Peter Pastor. Learning probabilistic multi-modal actor models for vision-based robotic grasping. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 4804–4810. IEEE, 2019.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. Deep reinforcement learning for page-wise recommendations. *Proceedings of the 12th ACM Conference on Recommender Systems*, Sep 2018. doi: 10.1145/3240323.3240374. URL http://dx.doi.org/10.1145/3240323.3240374.

Zhuobin Zheng12, Chun Yuan, Zhihui Lin12, and Yangyang Cheng12. Self-adaptive double bootstrapped ddpg. In *International Joint Conference on Artificial Intelligence*, 2018.

Tao Zhou, Zoltán Kuscsik, Jian-Guo Liu, Matúš Medo, Joseph Rushton Wakeling, and Yi-Cheng Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.

Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. Reinforcement learning to optimize long-term user engagement in recommender systems. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2810–2818, 2019.

APPENDIX

Table of Contents

A Proof of Convergence of Maximizer Actor in Tabular Settings

Theorem A.1 (Convergence of Policy Iteration with Maximizer Actor)**.** *In a finite Markov Decision Process (MDP) with finite state space $\mathcal{S}$, consider a modified policy iteration algorithm where, at each iteration n, we have a set of $k+1$ policies $\{\nu_0, \nu_1, \ldots, \nu_k\}$, with $\nu_0 = \mu_n$ being the policy at the current iteration learned with DPG. We define the* maximizer actor μ_M *as:*

$$\mu_M(s) = \arg \max_{a \in \{\nu_0(s), \nu_1(s), \ldots, \nu_k(s)\}} Q^{\mu_n}(s, a), \tag{10}$$

where $Q^{\mu_n}(s, a)$ is the action-value function for policy μ_n. Then, the modified policy iteration algorithm using the maximizer actor is guaranteed to converge to a final policy μ_N.

Proof. To prove convergence, we will show that the sequence of policies μ_n yields monotonically non-decreasing value functions that converge to a stable value function V^N.

Policy Evaluation Converges

Thus, iteratively applying $\mathcal{T}^\pi$ starting from any initial Q_0 converges to the unique fixed point Q^π.

Given the current policy μ_n, the policy evaluation computes the action-value function Q^{μ_n}, satisfying:

$$Q^{\mu_n}(s, a) = R(s, a) + \gamma \sum_{s'} P(s, a, s') V^{\mu_n}(s'),$$

where $V^{\mu_n}(s') = Q^{\mu_n}(s', \mu_n(s'))$.

In the tabular setting, the Bellman operator $\mathcal{T}^{\mu_n}$ defined by

$$[\mathcal{T}^{\mu_n} Q](s, a) = R(s, a) + \gamma \sum_{s'} P(s, a, s') Q(s', \mu_n(s'))$$

is a contraction mapping with respect to the max norm $\|\cdot\|_\infty$ with contraction factor γ,

$$\|\mathcal{T}^{\mu_n} Q - \mathcal{T}^{\mu_n} Q'\|_\infty \leq \gamma \|Q - Q'\|_\infty.$$

Therefore, iteratively applying $\mathcal{T}^{\mu_n}$ converges to the unique fixed point Q^{μ_n}.

Policy Improvement with DPG and Maximizer Actor

Step 1: DPG Update

We define $\tilde{\mu}_n$ as the DPG policy that locally updates μ_n towards maximizing the expected return based on Q^{μ_n}. For each state s, we perform a gradient ascent step using the Deep Policy Gradient (DPG) method to obtain an improved policy $\tilde{\mu}_n$:

$$\tilde{\mu}_n(s) \leftarrow \mu_n(s) + \alpha \nabla_a Q^{\mu_n}(s, a)\big|_{a=\mu_n(s)},$$

where $\alpha > 0$ is a suitable step size.

This DPG gradient step leads to local policy improvement following over μ_n (Silver et al., 2014):

$$V^{\tilde{\mu}_n}(s) \geq V^{\mu_n}(s), \quad \forall s \in \mathcal{S}.$$

(b) *Maximizer Actor*

Given additional policies $\nu_1, \ldots, \nu_k$, define the maximizer actor μ_{n+1} as:

$$\mu_{n+1}(s) = \arg \max_{a \in \{\tilde{\mu}_n(s), \nu_1(s), \ldots, \nu_k(s)\}} Q^{\mu_n}(s, a).$$

Since $\mu_{n+1}(s)$ selects the action maximizing $Q^{\mu_n}(s, a)$ among candidates, we have:

$$Q^{\mu_n}(s, \mu_{n+1}(s)) = \max_{a \in \{\tilde{\mu}_n(s), \nu_1(s), \ldots, \nu_k(s)\}} Q^{\mu_n}(s, a) \geq Q^{\mu_n}(s, \tilde{\mu}_n(s)) \geq V^{\mu_n}(s).$$

By the Policy Improvement Theorem, since $Q^{\mu_n}(s, \mu_{n+1}(s)) \geq V^{\mu_n}(s)$ for all s, it follows that:

$$V^{\mu_{n+1}}(s) \geq V^{\mu_n}(s), \quad \forall s \in \mathcal{S}.$$

Thus, the sequence $\{V^{\mu_n}\}$ is monotonically non-decreasing.

CONVERGENCE OF POLICY ITERATION

Since the sequence $\{V^{\mu_n}\}$ is monotonically non-decreasing and bounded above by V^*, it converges to some $V^\infty \leq V^*$. Given the finite number of possible policies, the sequence $\{\mu_n\}$ must eventually repeat a policy. Suppose that at some iteration N, the policy repeats, i.e., $\mu_{N+1} = \mu_N$.

At this point, since the policy hasn't changed, we have:

$$\mu_N(s) = \arg \max_{a \in \{\tilde{\mu}_N(s), \nu_1(s), \ldots, \nu_k(s)\}} Q^{\mu_N}(s, a), \quad \forall s \in \mathcal{S}.$$

Since $\tilde{\mu}_N(s)$ is obtained by performing a DPG update on $\mu_N(s)$, and we have that $\mu_N(s)$ maximizes $Q^{\mu_N}(s, a)$ among $\{\tilde{\mu}_N(s), \nu_1(s), \ldots, \nu_k(s)\}$, it must be that:

$$Q^{\mu_N}(s, \mu_N(s)) \geq Q^{\mu_N}(s, a), \quad \forall a \in \{\tilde{\mu}_N(s), \nu_1(s), \ldots, \nu_k(s)\}.$$

Moreover, since $\tilde{\mu}_N(s)$ is obtained via gradient ascent from $\mu_N(s)$, and yet does not yield a higher Q-value, it implies that:

$$\nabla_a Q^{\mu_N}(s, a)\big|_{a=\mu_N(s)} = 0.$$

This suggests that $\mu_N(s)$ is a local maximum of $Q^{\mu_N}(s, a)$. This shows that this modification to the policy iteration algorithm of DPG is guaranteed to converge.

Since the set $\{\tilde{\mu}_N(s), \nu_1(s), \ldots, \nu_k(s)\}$ includes more actions from $\mathcal{A}$, $\mu_N(s)$ is the action that better maximizes $Q^{\mu_N}(s, a)$ than $\tilde{\mu}_N$. Therefore, μ_N is a greedier policy with respect to Q^{μ_N} than $\tilde{\mu}_N$.

□

B PROOF OF REDUCING NUMBER OF LOCAL OPTIMA IN SUCCESSIVE SURROGATES

Theorem B.1. *Consider a state $s \in \mathcal{S}$, the function Q as defined in Eq. 1, and the surrogate functions Ψ_i as defined in Eq. 7. Let $N_{opt}(f)$ denote the number of local optima (assumed countable) of a function $f : \mathcal{A} \to \mathbb{R}$, where $\mathcal{A}$ is the action space. Then,*

$$N_{opt}(Q(s, a)) \geq N_{opt}(\Psi_0(s, a; \{a_0\})) \geq N_{opt}(\Psi_1(s, a; \{a_0, a_1\})) \geq \cdots \geq N_{opt}(\Psi_k(s, a; \{a_0, \ldots, a_k\})).$$

Proof. For each $i \geq 0$, define the surrogate function Ψ_i recursively:

$$\Psi_i(s, a; \{a_0, \ldots, a_i\}) = \max\{Q(s, a), \tau_i\}, \tag{11}$$

where

$$\tau_i = \max_{0 \leq j \leq i} Q(s, a_j).$$

Note that τ_i is non-decreasing with respect to i, i.e., $\tau_{i+1} \geq \tau_i$.

Our goal is to show that for each $i \geq 0$,

$$N_{\text{opt}}(\Psi_i(s, a; \{a_0, \ldots, a_i\})) \geq N_{\text{opt}}(\Psi_{i+1}(s, a; \{a_0, \ldots, a_{i+1}\})).$$

We proceed by considering how the set of local optima changes from Ψ_i to Ψ_{i+1}.

Consider any local optimum a' of Ψ_i. There are two cases:

Case 1: $Q(s, a') > \tau_i$

In this case, $\Psi_i(s, a') = Q(s, a')$ and Ψ_i coincides with Q in a neighborhood of a'. Since a' is a local optimum of Ψ_i, it is also a local optimum of Q. Because $\tau_{i+1} \geq \tau_i$, there are two subcases:

Subcase 1a: $Q(s, a') > \tau_{i+1}$

Here, $\Psi_{i+1}(s, a') = Q(s, a')$ and, in a neighborhood of a', Ψ_{i+1} coincides with Q. Thus, a' remains a local optimum of Ψ_{i+1}.

Subcase 1b: $Q(s, a') \leq \tau_{i+1}$

Since $Q(s, a') > \tau_i$ and $\tau_{i+1} \geq \tau_i$, this implies $\tau_{i+1} > \tau_i$ and $Q(s, a') = \tau_{i+1}$. Then,

$$\Psi_{i+1}(s, a') = \tau_{i+1},$$

and in a neighborhood around a', $\Psi_{i+1}(s, a) \geq \tau_{i+1}$. Thus, a' is not a local optimum of Ψ_{i+1} because there is no neighborhood where $\Psi_{i+1}(s, a) < \Psi_{i+1}(s, a')$.

Case 2: $Q(s, a') \leq \tau_i$

In this case, $\Psi_i(s, a') = \tau_i$, and Ψ_i is constant at τ_i in a neighborhood of a'. Thus, a' may be considered a local optimum in Ψ_i if the function does not exceed τ_i nearby. When moving to Ψ_{i+1}, since $\tau_{i+1} \geq \tau_i$, we have:

$$\Psi_{i+1}(s, a') = \tau_{i+1} \geq \tau_i.$$

In the neighborhood of a', Ψ_{i+1} remains at least τ_{i+1}, so a' is not a local optimum in Ψ_{i+1} unless $Q(s, a) < \tau_{i+1}$ in a neighborhood around a'.

However, since $\Psi_{i+1}(s, a) \geq \tau_{i+1}$ for all a, the function does not decrease below $\Psi_{i+1}(s, a')$ in any neighborhood of a'. Therefore, a' is not a local optimum of Ψ_{i+1}.

Conclusion: From the above cases, we observe that:

- Any local optimum a' of Ψ_i where $Q(s, a') > \tau_{i+1}$ remains a local optimum in Ψ_{i+1}.
- Any local optimum a' of Ψ_i where $Q(s, a') \leq \tau_{i+1}$ does not remain a local optimum in Ψ_{i+1}.

Since Ψ_{i+1} does not introduce new local optima (because $\Psi_{i+1}(s, a) \geq \Psi_i(s, a)$ for all a and coincides with Q only where $Q(s, a) > \tau_{i+1}$), the number of local optima does not increase from Ψ_i to Ψ_{i+1}.

Base Case: For $i = 0$, we have:

$$\Psi_0(s, a; \{a_0\}) = \max\{Q(s, a), Q(s, a_0)\}.$$

If we consider $Q(s, a_0)$ to be less than the minimum value of $Q(s, a)$ (which can be arranged by choosing a_0 appropriately or by defining τ_0 to be less than $\inf_a Q(s, a)$), then $\Psi_0(s, a) = Q(s, a)$, and the base case holds trivially.

Inductive Step: Assuming that

$$N_{\text{opt}}(\Psi_i(s, a; \{a_0, \ldots, a_i\})) \leq N_{\text{opt}}(\Psi_i(s, a; \{a_0, \ldots, a_i\})),$$

we have shown that

$$N_{\text{opt}}(\Psi_{i+1}(s, a; \{a_0, \ldots, a_{i+1}\})) \leq N_{\text{opt}}(\Psi_i(s, a; \{a_0, \ldots, a_i\})).$$

By induction, it follows that:

$$N_{\text{opt}}(Q(s, a)) \geq N_{\text{opt}}(\Psi_0(s, a; \{a_0\})) \geq N_{\text{opt}}(\Psi_1(s, a; \{a_0, a_1\})) \geq \cdots \geq N_{\text{opt}}(\Psi_k(s, a; \{a_0, \ldots, a_k\})).$$

□

C Environment Details

C.1 MiningEnv

The grid world environment, introduced in §5, requires an agent to reach a goal by navigating a 2D maze as soon as possible while breaking the mines blocking the way.

State: The state space is an 8+K dimensional vector, where K equals to *mine-category-size*. This vector consists of 4 independent pieces of information: Agent Position, Agent Direction, Surrounding Path, and Front Cell Type.

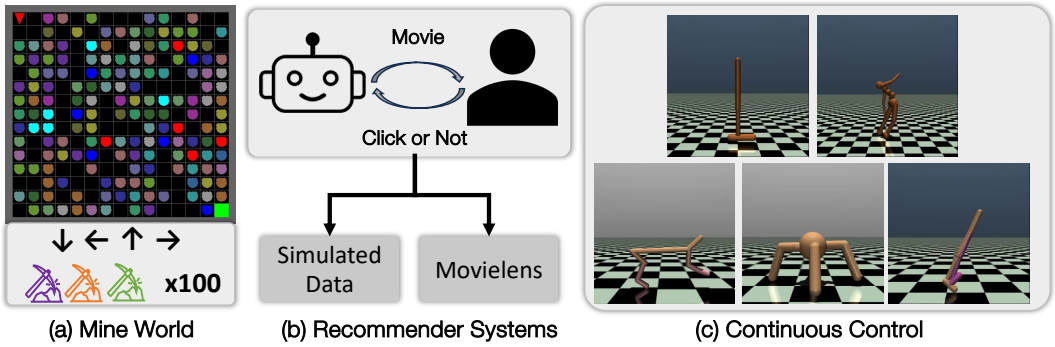


Figure 13: **Benchmark Environments** involve discrete action space tasks like Mine World and recommender systems (simulated and MovieLens-Data) and restricted locomotion tasks.

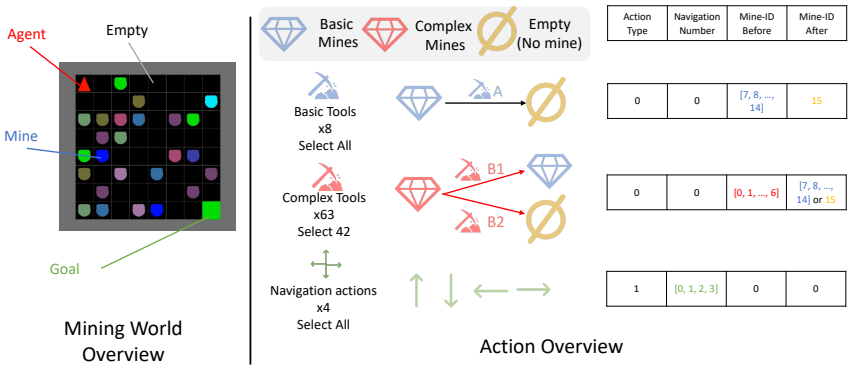


Figure 14: **Mining Expedition**. The red agent must reach the green goal by navigating the grid and using one or more pick-axes to clear each mine blocking the path.

1. Agent Position: A 2D vector representing the agent's x and y coordinates.
2. Agent Direction: One dimension representing directions (0: right, 1: down, 2: left, 3: up).
3. Surrounding Path: A 4-dimensional vector indicating if the adjacent cells are empty or a goal (1: empty/goal, 0: otherwise).
4. Front Cell Type: A $(K+1)$-dimensional one-hot vector with first K dimensions representing the type of mine and the last dimension representing if the cell is empty (zero) or goal (one).

Finally, we will normalize each dimension to [0, 1] with each dimension's minimum/maximum value.

Termination: An episode terminates when the agent reaches the goal or after 100 timesteps. Upon reset, the grid layout changes while keeping the agent's start and goal positions fixed.

Actions: Actions include navigation (up, down, left, right) and pick-axe categories. Navigation changes the agent's direction and attempts to move forward. The agent cannot step into a mine but will change direction when trying to step onto a mine or the border of the grid. The pick-axe tool actions (50 types) have a predefined one-to-one mapping of how they interact with the mines, which means they can be successfully applied to only one kind of mine, and either transform that kind of mine into another type of mine or directly break it.

Reward: The agent's reward comprises a goal-reaching reward, a distance-based step reward, and rewards for successful tool use or mine-breaking. The goal reward is discounted by steps taken over the episode, encouraging shorter paths to reach the goal.

$$
\begin{aligned}
R(s,a) \; = \mathbb{1}_{Goal} \cdot R_{\text{Goal}} \left(1 - \lambda_{\text{Goal}} \frac{N_{\text{current steps}}}{N_{\text{max steps}}}\right) + \\
R_{\text{Step}} \left(D_{\text{distance before}} - D_{\text{distance after}}\right) + \\
\mathbb{1}_{correct\ tool\ applied} \cdot R_{\text{Tool}} + \\
\mathbb{1}_{successfully\ break\ mine} \cdot R_{\text{Bonus}}
\end{aligned}
\tag{12}
$$

where $R_{\text{Goal}} = 10,\ R_{\text{Step}} = 0.1,\ R_{\text{Tool}} = 0.1,\ R_{\text{Bonus}} = 0.1,\ \lambda_{\text{Goal}} = 0.9,\ N_{\text{max steps}} = 100$

Action Representations Actions are represented as a 4D vector with normalized values [0, 1] as described in Figure 14. Dimensions represent skill category (navigation or pick-axe), movement direction (right, down, left, up), mine type where this action can be applied, and the outcome of applying the tool to the mine, respectively.

C.2 RecSim

In the simulated recommendation system (RecSys) environment, the agent selects an item from a large set that aligns with the user's interests. Users are modeled with dynamically changing preferences that evolve based on their interactions (clicks). The agent's objective is to infer these evolving preferences from user clicks and recommend the most relevant items to maximize the total number of clicks.

State: The user's interest is represented by an embedding vector $e_u \in \mathbb{R}^n$, where n is the number of item categories. This embedding evolves over time as the user interacts with different items. When a user clicks on an item with embedding $e_i \in \mathbb{R}^n$, the user interest embedding e_u is updated as follows:

$$e_u \leftarrow e_u + \Delta e_u, \quad \text{with probability } \frac{e_u^\top e_i + 1}{2}$$
$$e_u \leftarrow e_u - \Delta e_u, \quad \text{with probability } \frac{1 - e_u^\top e_i}{2},$$

where Δe_u represents an adjustment that depends on the alignment between e_u and e_i. This update mechanism adjusts the user's preference towards the clicked item, reinforcing the connection between the current action on future recommendations.

Action: The action set consists of all items that can be recommended, and the agent must select the item most relevant to the user's long-term preferences over the episode.

Reward: The reward is based on user feedback: either a click (reward = 1) or skip (reward = 0). The user model computes a score for each item using the dot product of the user and item embeddings:

$$\text{score}_{item} = \langle e_u, e_i \rangle$$

The click probability is computed with a softmax over the item score and a predefined skip score:

$$p_{item} = \frac{e^{\text{score}_{item}}}{e^{\text{score}_{item}} + e^{\text{score}_{skip}}}, \quad p_{skip} = 1 - p_{item}$$

The user then stochastically chooses to click or skip based on this distribution.

Action Representations: Following Jain et al. (2021), items are represented as continuous vectors sampled from a Gaussian Mixture Model (GMM), with centers representing item categories.

C.3 Continuous Control

MuJoCo (Todorov et al., 2012) is a physics engine that provides a suite of standard reinforcement learning tasks with continuous action spaces, commonly used for benchmarking continuous control algorithms. We briefly describe some of these tasks below:

Hopper: The agent controls a one-legged robot that must learn to hop forward while maintaining balance. The objective is to maximize forward velocity without falling.

Walker2d: The agent controls a two-legged bipedal robot that must learn to walk forward efficiently while maintaining balance. The goal is to achieve stable locomotion at high speeds.

HalfCheetah: The agent controls a planar, cheetah-like robot with multiple joints in a 2D environment. The task requires learning a coordinated gait to propel the robot forward as quickly as possible.

Ant: The agent controls a four-legged, ant-like robot with multiple degrees of freedom. The challenge is to learn to walk and navigate efficiently while maximizing forward progress.

C.3.1 Restricted Locomotion in Mujoco

The restricted locomotion Mujoco tasks are introduced to demonstrate how common DPG-based approaches get stuck in local optima when the Q-landscape is complex and non-convex. This setting

limits the range of actions the agent can perform in each dimension, simulating realistic scenarios such as wear and tear of hardware. For example, action space may be affected as visualized in Figure 6. A mixture-of-hypersphere action space is used to simulate such asymmetric restrictions, which affect the range of torques for the Hopper and Walker joints, as well as the forces applied to the inverted pendulum and double pendulum. The hyperspheres are sampled randomly, and their size and radius are carefully tuned to ensure that the action space has enough valid actions to solve the task.

Definition of restriction.

- **Restricted Hopper & Walker**
  Invalid action vectors are replaced with 0 by changing the environment's step function code:

```
def step(action):
    ...
    if check_valid(action):
        self.do_simulation(action)
    else:
        self.do_simulation(np.zeros_like(action))
    ...
```

  The Hopper action space is 3-dimensional, with torque applied to $[\text{thigh}, \text{leg}, \text{foot}]$, while the Walker action space is 6-dimensional, with torque applied to $[\text{right thigh}, \text{right leg}, \text{right foot}, \text{left thigh}, \text{left leg}, \text{left foot}]$. The physical implication of restricted locomotion is that zero torques are exerted for the Δt duration between two actions, i.e., no torques are applied for 0.008 seconds. This effectively slows down the agent's current velocities and angular velocities due to friction whenever the agent selects an invalid action.

- **Inverted Pendulum & Inverted Double Pendulum**
  Invalid action vectors are replaced with -1 by changing the environment's step function code:

```
def step(action):
...
    if not check_valid(action):
        action[:] = -1.
    self.do_simulation(action)
...
```

  The action space is 1-dimensional, with force applied on the cart. The implication is that the cart is pushed in the left direction for 0.02 (default) seconds. Note that the action vectors are not zeroed because a 0-action is often the optimal action, particularly when the agent starts upright. This would make the optimal policy trivially be *learning to select invalid actions*.

D Additional Results

D.1 Experiments on Continuous Control (Unrestricted MuJoCo)

In standard MuJoCo tasks, the Q-landscape is likely easier to optimize compared to MuJoCo-Restricted tasks. In Figure 15, we find that baseline models consistently perform well in all standard tasks, unlike in MuJoCo-Restricted tasks. Thus, we can infer the following:

1. Baselines have sufficient capacity, are well-tuned, and can navigate simple Q-landscapes optimally.
2. SAVO performs on par with other methods in MuJoCo tasks where the Q-landscape is easier to optimize, showing that SAVO is a robust, widely applicable actor architecture.
3. Baselines performing well in unrestricted locomotion but suboptimally in restricted locomotion delineates the cause of suboptimality to be the complexity of the underlying Q-landscapes, such as those shown in Figure 2. SAVO is closer to optimal in both settings because it can navigate both simple and complex Q-functions better than alternate actor architectures.

D.2 Per-Environment Ablation Results

Figure 16 shows the per-environment performance of SAVO ablations, compiled into aggregate performance profiles in Figure 7b. The **SAVO - Approximation** variant underperforms significantly

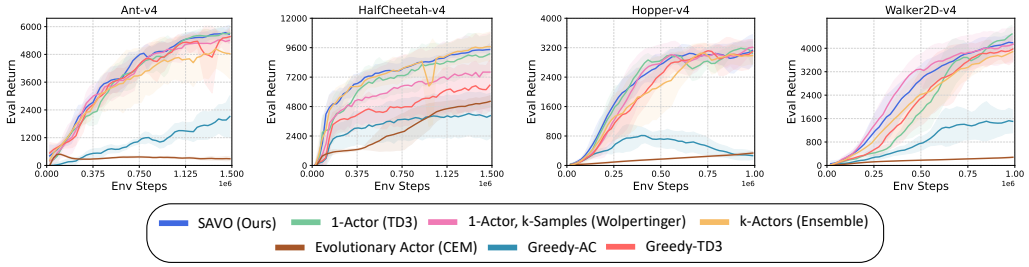


Figure 15: **Unrestricted Locomotion** (§D.1). SAVO and most baselines perform optimally in standard MuJoCo continuous control tasks, where the Q-landscape is easy to navigate.

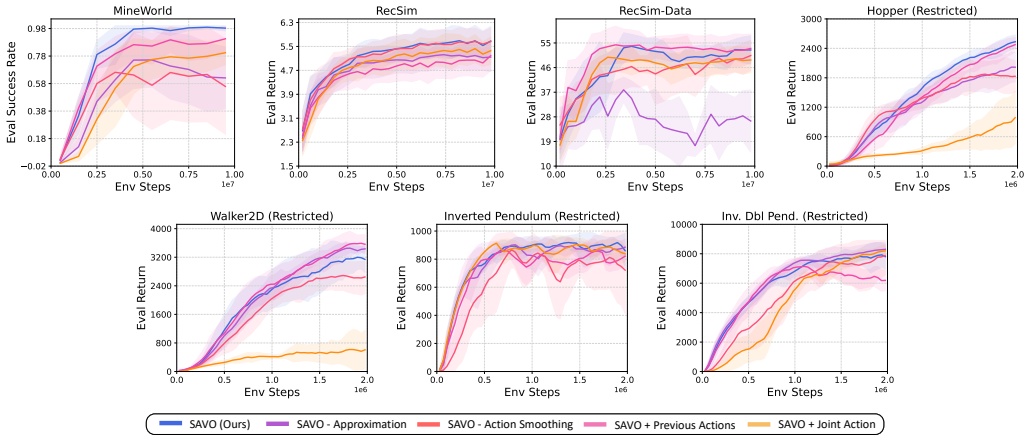


Figure 16: **Ablations of SAVO variations** (§D.2) shows the importance of (i) the approximation of surrogates, (ii) removing TD3's action smoothing, (iii) conditioning on preceding actions in the successive actor and surrogate networks, and (iv) individual actors that separate the action candidate prediction instead of a joint high-dimensional learning task.

in discrete action space tasks, where traversing between local optima is complex due to nearby actions having diverse Q-values (see the right panel of Figure 2). Similarly, adding TD3's target action smoothing to SAVO results in inaccurate learned Q-values when several differently valued actions exist near the target action, as in the complex landscapes of all tasks considered.

Removing information about preceding actions does not significantly degrade SAVO's performance since preceding actions' Q-values are indirectly incorporated into the surrogates' training objective (see Eq. (9)), except for MineWorld where this information helps improve efficiency.

The **SAVO + Joint** ablation learns a single actor that outputs a joint action composed of k constituents, aiming to cover the action space so that multiple coordinated actions can better maximize the Q-function compared to a single action. However, this increases the complexity of the architecture and only works in low-dimensional tasks like Inverted-Pendulum and Inverted-Double-Pendulum. SAVO simplifies action candidate generation by using several successive actors with specialized objectives, enabling easier training without exploding the action space.

D.3 SAC IS ORTHOGONAL TO SAVO

We compare Soft Actor-Critic (SAC), TD3, and TD3 + SAVO across various Mujoco-Restricted tasks. Figure 17 shows that SAC sometimes outperforms and sometimes underperforms TD3. Therefore, SAC's stochastic policy does not address the challenge of non-convexity in the Q-function. In contrast, SAVO+TD3 consistently outperforms TD3 and SAC, demonstrating the effectiveness of SAVO in complex Q-landscapes. While SAC can be better than TD3 in certain environments, its algorithmic modifications are orthogonal to the architectural improvements due to the SAVO actor.

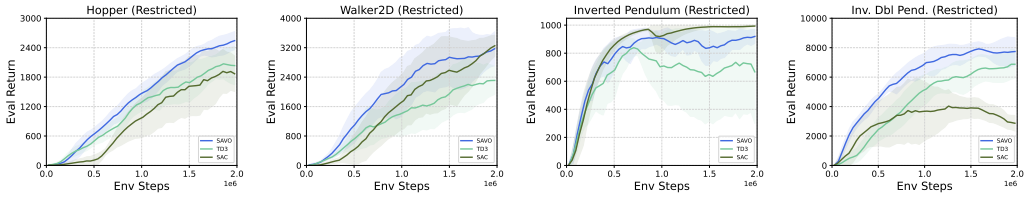


Figure 17: **SAC is orthogonal to the effect of SAVO** (§D.3). SAC is a different algorithm than TD3, whereas SAVO is a plug-in actor architecture for TD3. Thus, tasks where SAC outperforms TD3 differ from tasks where SAVO outperforms TD3. Also, TD3 outperforms SAC in Restricted Hopper and Inverted-Double-Pendulum. However, SAVO+TD3 guarantees improvement over TD3.

In the Restricted Inverted Double Pendulum task (Figure 18), SAC underperforms even TD3. Analogous to TD3, the suboptimality is due to the non-convexity in the soft Q-function landscape, where small changes in nearby actions can lead to significantly different environment returns. We combine SAC with SAVO's successive actors and surrogates to better maximize the soft Q-function, naively considering action candidates for μ_M as the mean action of the stochastic actors. We observe that this preliminary version of SAC + SAVO shows significant improvements over SAC in complex Q-landscapes. In future work, we aim to formalize a SAVO-like objective that effectively enables SAC's stochastic actors to navigate the non-convexity of its soft Q-function.

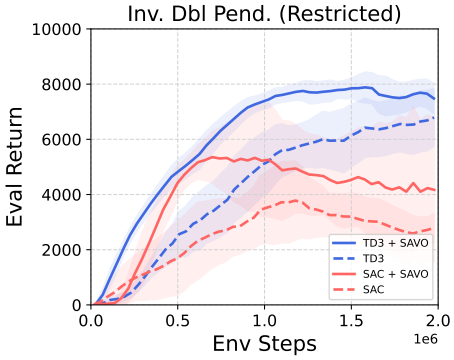


Figure 18: **SAC is suboptimal in complex Q-landscape** (§D.3) of Restricted Inverted Double Pendulum, but SAVO helps.

D.4 Increasing Size of Discrete Action Space in RecSim

We test the robustness of our method to more challenging Q-value landscapes in Figure 19, especially in discrete action space tasks with massive action spaces. In RecSim, we increase the number of actual discrete actions from $10,000$ to $100,000$ and $500,000$. The experiments show that SAVO outperforms the best-performing baseline of TD3 + Sampling (Wolpertinger) and the best-performing ablation of SAVO + Joint Action. This shows that SAVO maintains robust performance even as the action space size increases and the Q-function landscape becomes more intricate. In contrast, the baselines experienced performance deterioration as action sizes grew larger.

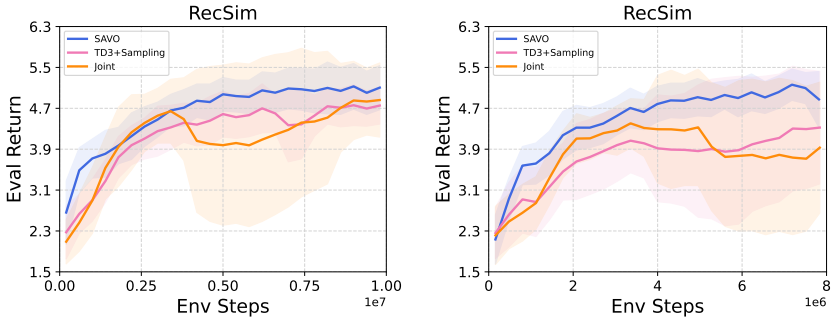


Figure 19: **Increasing RecSim action set size** (§D.4). (Left) $100,000$ items, (Right) $500,000$ items (6 seeds) maintains the performance trends of SAVO and the best-performing baseline (TD3 + Sampling) and the best-performing ablation (SAVO with Joint-Action).

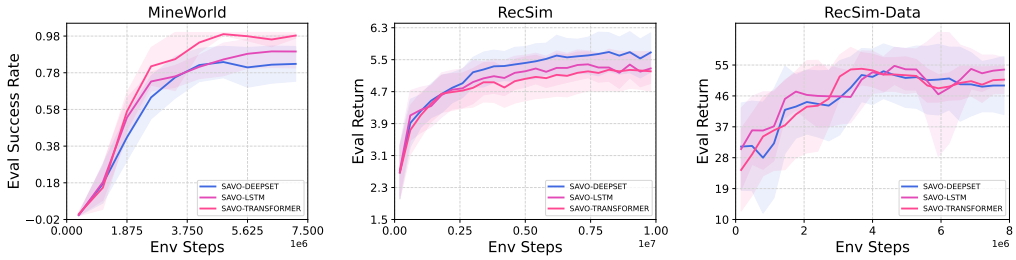


Figure 20: **Action summarizer comparison** (§E.1). The effect is not significant The results are averaged over 5 random seeds, and the seed variance is shown with shading.

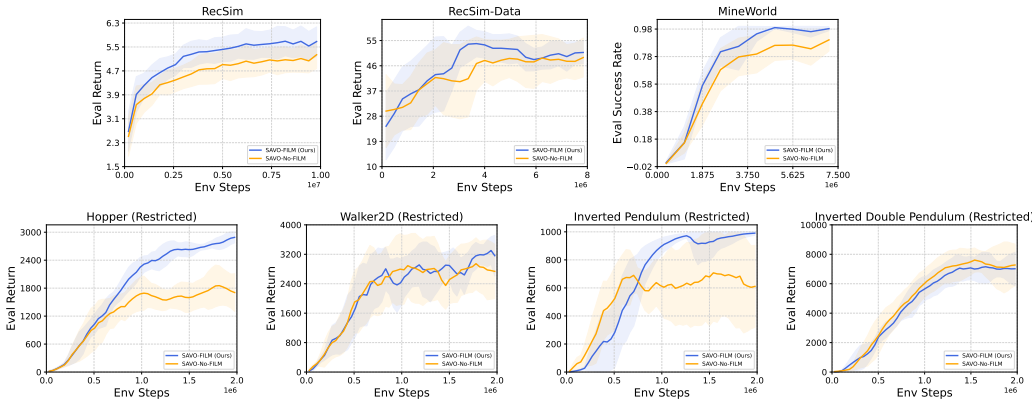


Figure 21: **FiLM to condition on preceding actions** (§E.2). FiLM ensures layerwise dependence on the preceding actions for acting in actors ν_i and for predicting value in surrogates $\hat{\Psi}_i$, which generally results in better performance across tasks.

E Validating SAVO Design Choices

E.1 Design Choices: Action summarizers

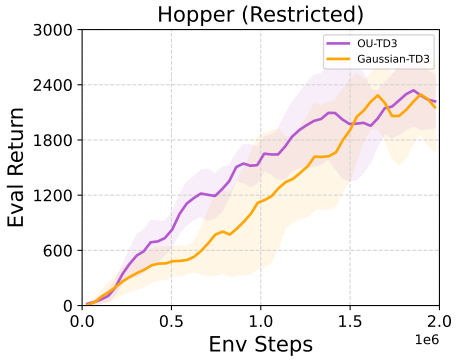
Three key architectures were considered for the design of the action summarizer: DeepSets, LSTM, and Transformer models, represented by SAVO, SAVO-LSTM, and SAVO-Transformer in Figure 20, respectively. In general, the effect of the action summarizer is not significant, and we choose DeepSet for its simplicity for most experiments.

E.2 Conditioning on Previous Actions: FiLM vs. MLP

We examined two approaches for conditioning on the previous action list summary: Feature-wise Linear Modulation (FiLM) and concatenation with input, represented by the FiLM and non-film variants in Figure 21. Across tasks, FiLM outperformed the non-FiLM version, showing the effectiveness of layerwise conditioning in leveraging prior action information for action selection and surrogate value prediction. This shows that the successive actors are appropriately utilizing the actions from the preceding actors to tailor their search for optimal actions, and the successive surrogates can better evaluate Q-values, knowing where they could be thresholded by the loss function.

E.3 Exploration Noise comparison: OUNoise vs Gaussian

We compare Ornstein-Uhlenbeck (OU) noise with Gaussian noise across our environments and find that OU noise was generally better, with the difference being minimal. We chose to use OU for our experiments and a comparison on Hopper-Restricted is shown in Figure 22a. We note that TD3 (Fujimoto et al., 2018) also suggests no significant difference between OU and Gaussian noise and favored Gaussian for simplicity. All our baselines use the same exploration backbone, and we confirm



(a) Noise types

Figure 22: **OU versus Gaussian Noise** (§E.3). We do not see a significant difference due to this choice, and select OU noise due to better overall performance in experiments.

that using OU noise is consistent with the available state-of-the-art results with TD3 + Gaussian noise on common environments like Ant, HalfCheetah, Hopper, and Walker2D.

F Network Architectures

F.1 Successive Actors

The entire actor is built as a successive architecture (see Figure 4), where each successive actor receives two pieces of information: the current state and the action list generated by preceding actors. Each action is concatenated with the state to contextualize it and then summarized using a list-summarizer, described in §F.3. This list summary is concatenated with the state again and passed into an MLP with ReLU (3 layers for *MuJoCo* tasks and 4 layers for *MineWorld* and *RecSim*) as described in Table 2. This MLP generates one action for each successive actor, which is subsequently used as an input action to the succeeding action lists. For discrete action space tasks, this generated action is processed with a 1-NN to find the nearest exact discrete action. Finally, the actions generated by each individual successive actor are accumulated, and the maximizer actor μ_M step from Eq. (5) selects the highest-valued action according to the *Critic* Q-network, described in §F.2.

F.2 Successive Surrogates

As Figure 4 illustrates, there is a surrogate network for each actor in the successive actor-architecture. Each successive critic receives three pieces of information: the current state, the action list generated by preceding actors, and the action generated by the actor corresponding to the current surrogate. Each action is concatenated with the state to contextualize it and then summarized using a list-summarizer, described in §F.3. This list summary is concatenated with the state and the current action, and passed into a 2-layer MLP with ReLU (See Table 2). This MLP generates the surrogate value $\hat{\Psi}_i(s, a; a_{<i})$ and is used as an objective to ascend over by its corresponding actor ν_i.

F.3 List Summarizers

To extract meaningful information from the list of candidate actions, we employed several list summarization methods following Jain et al. (2021). These methods are described below:

Bi-LSTM: The action representations of the preceding actors' actions are first passed through a two-layer multilayer perceptron (MLP) with ReLU activation functions. The output of this MLP is then processed by a two-layer bidirectional LSTM network (Huang et al., 2015). The resulting output is fed into another two-layer MLP to create an action set summary, which serves as an input for the actor-network (§F.1) and the surrogate network (§F.2).

DeepSet: The action representations of the preceding actors' actions are initially processed by a two-layer MLP with ReLU activations. The outputs are then aggregated using mean pooling over all candidate actions to compress the information into a fixed-size summary. This summary is passed through another two-layer MLP with ReLU activation to produce the action set summary, which serves as an input for the actor-network (§F.1) and the surrogate network (§F.2).

Transformer: Similar to Bi-LSTM, the action representations of the preceding actors' actions are first processed by a two-layer MLP with ReLU activations. The outputs are then input into a Transformer network with self-attention and feed-forward layers to summarize the information. The resulting summary is used as part of the input to the actor-network (§F.1) and the surrogate network (§F.2).

F.4 Feature-wise Linear Modulation (FiLM)

Feature-wise Linear Modulation (Perez et al., 2018) is a technique used in neural networks to condition intermediate feature representations based on external information, enhancing the network's adaptability and performance across various tasks. FiLM modulates the features of a layer by applying learned, feature-wise affine transformations. Specifically, given a set of features $\mathbf{F}$, FiLM applies a scaling and shifting operation,

$$\text{FiLM}(\mathbf{F}) = \gamma \odot \mathbf{F} + \beta,$$

where γ and β are modulation parameters learned from another source (e.g., a separate network or input), and $\odot$ denotes element-wise multiplication. This approach allows the network to selectively emphasize or de-emphasize aspects of the input data, effectively capturing complex and context-specific relationships. FiLM has been successfully applied in tasks such as visual question answering and image captioning, where conditioning visual features on textual input is essential. We apply FiLM while conditioning the actor and surrogate networks on the summary of preceding actions.

G Experiment and Evaluation Setup

G.1 Aggregated Results: Performance Profiles

To rigorously validate the aggregate efficacy of our approach, we adopt the robust evaluation methodology proposed by Agarwal et al. (2021). By incorporating their suggested performance profiles, we conduct a comprehensive comparison between our method and baseline approaches, providing a thorough understanding of the statistical uncertainties inherent in our results. Figure 7a shows the performance profiles across all tasks. The x-axis represents normalized scores, calculated using *min-max scaling* based on the initial performance of untrained agents aggregated across random seeds (i.e., *Min*) and the final performance from Figure 8 (i.e., *Max*). The results show that our method consistently outperforms the baselines across various random seeds and environments. Our performance curve remains at the top as the x-axis progresses, while the baseline curves decline earlier. This highlights the reliability of SAVO over different environments and 10 seeds.

G.2 Implementation Details

We used PyTorch (Paszke et al., 2019) for our implementation, and the experiments were primarily conducted on workstations with either NVIDIA GeForce RTX 2080 Ti, P40, or V32 GPUs on. Each experiment seed takes about 4-6 hours for Mine World, 12-72 hours for Mujoco, and 6-72 hours for RecSim, to converge. We use the Weights & Biases tool (Biewald, 2020) for plotting and logging experiments. All the environments were interfaced using OpenAI Gym wrappers (Brockman et al., 2016). We use the Adam optimizer (Kingma & Ba, 2014) throughout for training.

G.3 Common Hyperparameter Tuning

To ensure fairness across all baselines and our methods, we performed hyperparameter tuning over parameters that are common across methods:

- **Learning Rates of Actor and Critic**: *(Actor)* We searched over learning rates $\{0.01, 0.001, 0.0001, 0.0003\}$ and found that 0.0003 was the most stable for the actor's learning across all tasks. *(Critic)* Similar to the actor, we searched over the same set of learning rates and found the same value of 0.0003 was the most stable for the critic's learning across all tasks.

- **Network Sizes of Actor and Critic**: For each task, we searched over simple 3 or 4 MLP layers to determine the network size that performed best but did not observe major differences. *(Critic)* To ensure a fair comparison, we used the same network size for the critic (Q-network) and surrogates across all methods within each task. *(Actor)* Similar to the critic, we used the same network size for the various actors in all the baselines and successive actors in SAVO within a particular task.

G.4 HYPERPARAMETERS

The environment and RL algorithm hyperparameters are described in Table 2.

Hyperparameter	Mine World	MuJoCo & Adroit	RecSim
Environment			
Total Timesteps	10^7	3×10^6	10^7
Number of epochs	5,000	8,000	10,000
# Envs in Parallel	20	10	16
Episode Horizon	100	1000	20
Number of Actions	104	N/A	10000
True Action Dim	4	5	30
Extra Action Dim	5	N/A	15
RL Training			
Batch size	256	256	256
Buffer size	5×10^5	5×10^5	10^6
Actor: LR	3×10^{-4}	3×10^{-4}	3×10^{-4}
Actor: ϵ_{start}	1	1	1
Actor: ϵ_{end}	0.01	0.01	0.01
Actor: ϵ decay steps	5×10^6	5×10^5	10^7
Actor: ϵ in Eval	0	0	0
Actor: MLP Layers	128_64_64_32	256_256	64_32_32_16
Critic: LR	3×10^{-4}	3×10^{-4}	3×10^{-4}
Critic: γ	0.99	0.99	0.99
Critic: MLP Layers	128_128	256_256	64_32
# updates per epoch	20	50	20
List Length	3	3	3
Type of List Encoder	DeepSet	DeepSet	DeepSet
List Encoder LR	3×10^{-4}	3×10^{-4}	3×10^{-4}

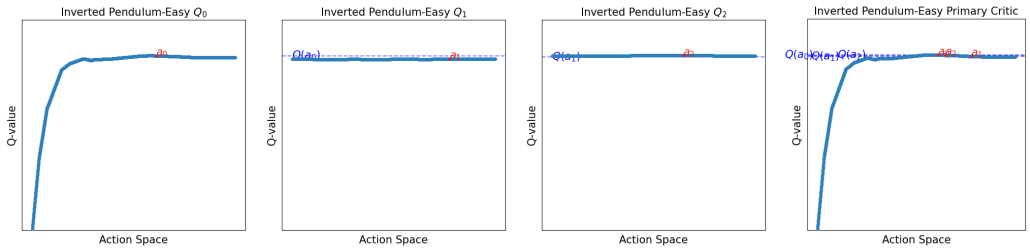
Table 2: Environment/Policy-specific Hyperparameters

H Q-VALUE LANDSCAPE VISUALIZATIONS

H.1 1-DIMENSIONAL ACTION SPACE ENVIRONMENTS

We analyzed the Q-value landscapes in Mujoco environments to show how successive critics help actors find better actions by reducing local optima. Figure 23 illustrates a typically smooth and easy-to-optimize Q-value landscape in unrestricted Inverted-Pendulum. Figure 24 illustrates that in restricted locomotion, the Q-value landscape (leftmost and rightmost figures) is uneven with many local optima. However, the Q-value landscapes learned by successive surrogates $\hat{\Psi}_i$ become successively smoother by removing local peaks below the Q-values of previously selected actions. This helps actors find closer to optimal actions than with a single critic.

Finally, when we plot the actions a_0, a_1, a_2 selected by the learned successive actors on the original Q-landscape (rightmost figure), we see they often achieve higher Q-values than a_0, the action a single actor has learned. Thus, the maximizer actor μ_M often finds closer to optimal actions than a single actor, resulting in better performance as shown in the return comparison between μ_M and single actor (Figure 11c) and the performance against baselines (Figure 8).



(a) $Q_0(s, a_0)$ (b) $Q_1(s, a_1|a_0)$ (c) $Q_2(s, a_2|\{a_0, a_1\})$ (d) $Q(s, a_i) \forall i = 0, 1, 2$

Figure 23: Successive surrogate landscapes and the Q-landscape of Inverted Pendulum-v4.

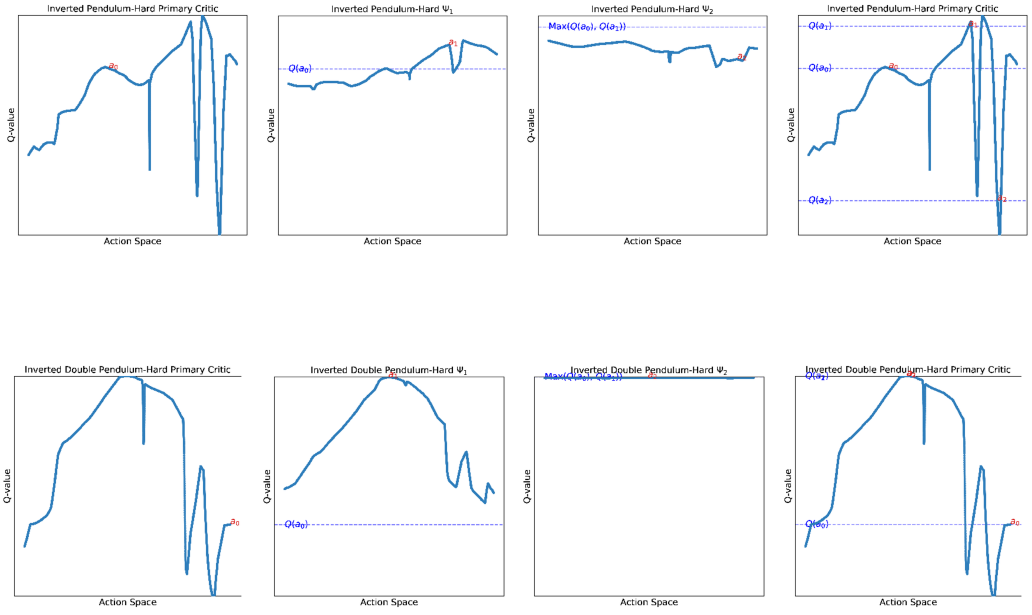


Figure 24: Successive surrogate landscapes and Q landscape for Restricted Inverted-Pendulum and Restricted Inverted-Double-Pendulum environments.

H.2 High-Dimensional Action Space Environments

Figure 25 visualize Q-value landscapes for a TD3 agent in Hopper-v4. We project actions from the 3D action space of Hopper-v4 onto a 2D plane using Uniform Manifold Approximation and Projection (UMAP) and sample 10,000 actions evenly to ensure thorough coverage. The Q-values are plotted using `trisurf`, which may introduce some artificial roughness but offers more reliable visuals than grid-surface plotting. Despite limitations of dimensionality reduction — such as distortion of distances — the Q-landscape for Hopper-v4 reveals a large globally optimal region (shown in yellow), offering a clear gradient path that prevents the gradient-based actor from getting stuck in local optima.

In contrast, Hopper-Restricted (Figure 26) has more complex Q-landscapes due to valid actions being restricted in one of the hyperspheres shown in Figure 6. Consequently, these Q-landscapes appear to have more locally optimal regions than Hopper-v4. This creates many peaks where gradient-based actors might get trapped, degrading the resultant agent performance.

The curse of dimensionality limits conclusive analyses on higher dimensional environments like Walker2D-v4 (6D) and Ant-v4 (8D) because projecting to 2D causes significant information loss, making it difficult to assess convexity in their Q-landscapes.

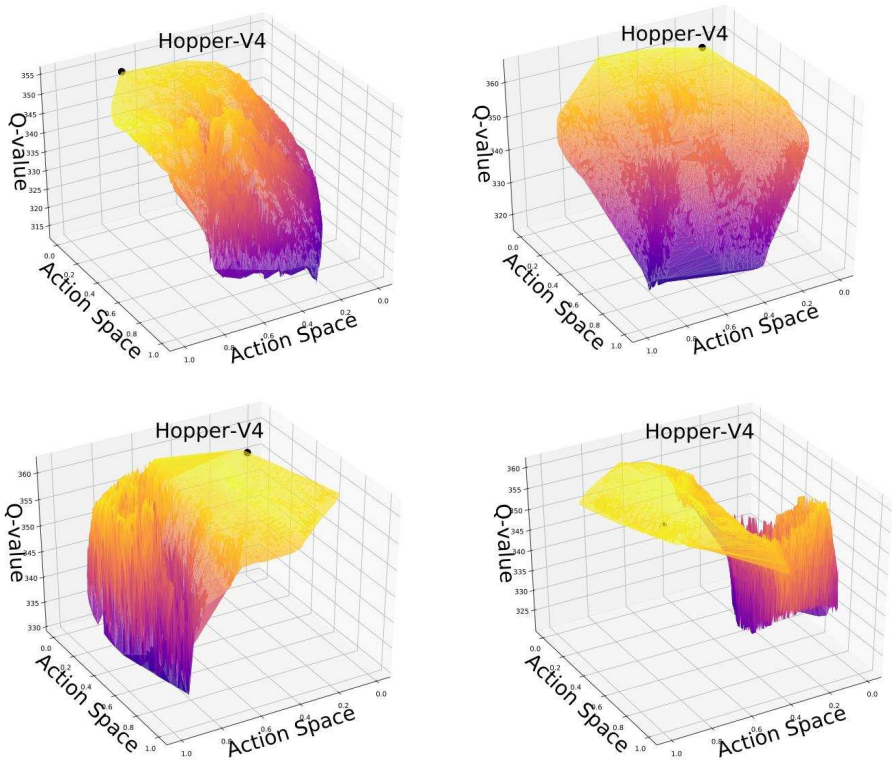


Figure 25: Hopper-v4: Q landscape visualization at different states show a path to optimum.

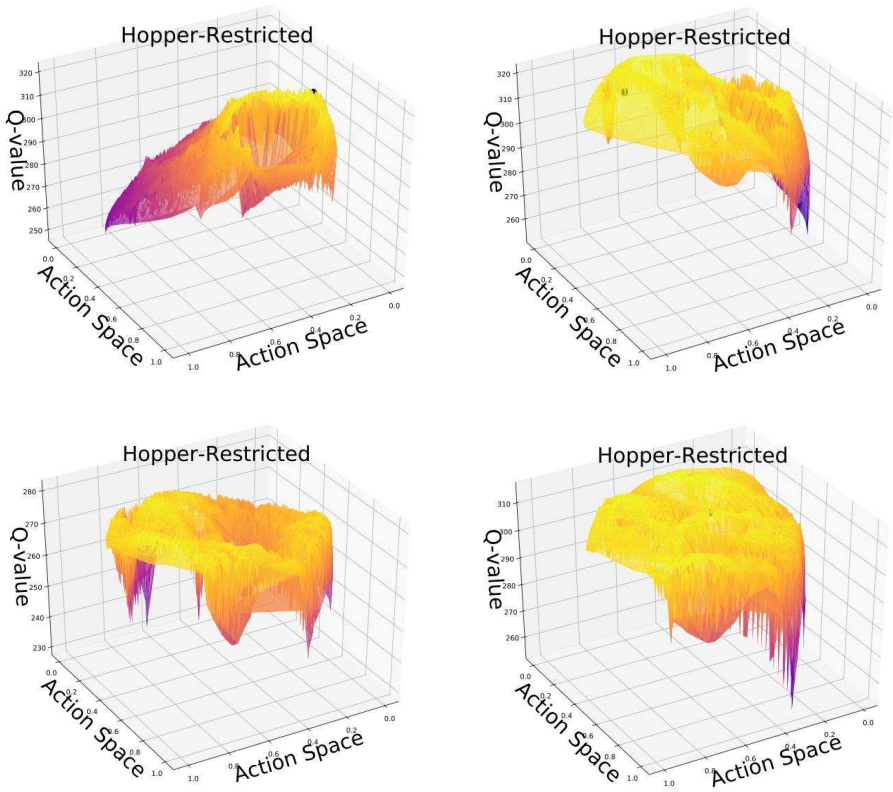


Figure 26: Hopper-restricted: Q landscape visualization at different states show several local optima.