BACHELOR'S THESIS / ENGINEERING 2024

Entropy-Regularized Deep Reinforcement Learning from a Linear Programming Perspective

Nicolás Vila Alarcón

upf. Universitat Pompeu Fabra Barcelona

Entropy-Regularized Deep Reinforcement Learning from a Linear Programming Perspective

TREBALL DE FI DE GRAU DE
Nicolás Vila Alarcón

Gergely Neu

Degree in Mathematical Engineering on Data Science

Academic Year 2023 - 2024

Acknowledgement

I would like to express my gratitude to my supervisor, Prof. Gergely Neu, for his consistent guidance and invaluable feedback throughout this thesis. I'm also immensely thankful for the constant encouragement and support from my family and friends. Lastly, I am grateful to the researchers whose work has provided the foundation and inspiration for this thesis.

Abstract

In recent years, Reinforcement Learning (RL) has gained significant attention due to its applications in real-world problems, particularly leveraging Deep Neural Networks (DNNs). While state of the art models like Proximal Policy Optimization (PPO), Twin Delayed DDPG (TD3), or Soft Actor-Critic (SAC) employ the squared Bellman error (SBE) in some way or other to address the Bellman Optimality Equations, the limitations of SBE are well-documented.

To address these issues, alternative algorithms rooted in the Linear Program (LP) reformulation of Markov Decision Processes (MDPs), such as REPS and Q-REPS, have shown to be promising. Yet, there exists a notable gap in research regarding the adaptation and application of these LP-based approaches within a large scale DNN learning setting.

We propose an adaptation of LP-based algorithms with the use of DNNs and perform an empirical study to understand the impact of different design choices in their practical implementation that may directly or indirectly affect their performance. We train over 30,000 agents in several discrete classical control environments of different complexity and provide insights and practical recommendations. Furthermore, we evaluate their performance compared to other well-known RL algorithms.

Resum

En els darrers anys, l'aprenentatge de reforç (RL) ha guanyat una atenció important a causa de les seves aplicacions en problemes del món real, especialment aprofitant les xarxes neuronals profundes (DNN). Si bé els models d'última generació com 'Proximal Policy Optimization' (PPO), el 'Twin Delayed DDPG ' (TD3) o el 'Soft Actor Critic' (SAC) utilitzen l'error quadrat de Bellman (SBE) d'una manera o altra per abordar les equacions d'optimitat de Bellman, les limitacions de l'SBE estan ben documentades.

Per abordar aquests problemes, els algorismes alternatius arrelats a la reformulació del programa lineal (LP) dels processos de decisió de Markov (MDP), com ara REPS i Q-REPS, han demostrat ser prometedors. No obstant això, hi ha un buit notable entre la teoria i la investigació sobre l'adaptació i l'aplicació d'aquests enfocaments basats en LP en un entorn d'aprenentatge amb DNN a gran escala.

Proposem una adaptació d'algorismes basats en LP amb l'ús de DNN i realitzem un estudi empíric per entendre l'impacte de diferents opcions de disseny en la seva implementació pràctica que poden afectar directament o indirectament el seu rendiment. Entrenem més de 30.000 agents en diversos entorns de control clàssics discrets de diferent complexitat i proporcionem idees i recomanacions pràctiques. A més, avaluem el seu rendiment en comparació amb altres algorismes de RL coneguts.

Resumen

En los últimos años, el aprendizaje por refuerzo (Reinforcement Learning, RL) ha recibido una gran atención debido a sus aplicaciones en problemas del mundo real, en particular aprovechando las redes neuronales profundas (Deep Neural Networks, DNN). Mientras que los modelos más avanzados, como la 'Proximal Policy Optimization' (PPO), 'Twin Delayed DDPG ' o Soft Actor-Critic (SAC) emplean el error cuadrático de Bellman (SBE) de una forma u otra para abordar las Ecuaciones de Optimalidad de Bellman, las limitaciones del SBE están bien documentadas.

Para resolver estos problemas, los algoritmos alternativos basados en la reformulación del programa lineal (LP) de los procesos de decisión de Markov (MDP), como REPS y Q-REPS, han demostrado ser prometedores. Sin embargo, existe una brecha entre la teoría y la investigación relativa a la adaptación y aplicación de estos enfoques basados en LP dentro de un entorno de aprendizaje mediante DNNs a gran escala.

Proponemos una adaptación de los algoritmos basados en LP con el uso de DNNs y realizamos un estudio empírico para comprender el impacto de las diferentes opciones de diseño en su implementación práctica que pueden afectar directa o indirectamente a su rendimiento. Entrenamos a más de 30.000 agentes en varios entornos de control clásico discreto de diferente complejidad y proporcionamos ideas y recomendaciones prácticas. Además, evaluamos su rendimiento en comparación con otros algoritmos de RL bien conocidos.

Contents

List of Tables

Introduction

Over the years, the field of Reinforcement Learning (RL) has evolved significantly, especially with the rise of Deep Reinforcement Learning (DRL). In particular, this field has gained popularity due to its success in various challenging tasks surpassing human capabilities [Silver et al., 2017]. These achievements highlight the practical applicability of DRL algorithms in real-world scenarios. However, there remains a significant amount of work and goals to accomplish, such as developing algorithms that learn faster and find better policies, gaining a deeper understanding of current algorithms, and investigating practical design choices in empirical settings. In many cases, these practical design choices are crucial for enhancing performance [Engstrom et al., 2020].

Regularization has become key in enhancing the performance of RL algorithms. Many state-of-the-art RL algorithms, incorporate some form of regularization to improve their performance. Recently, exploring Linear Programming as a framework for developing RL algorithms, exemplified by REPS [Peters et al., 2010] and Q-REPS [Bas-Serrano et al., 2021], has introduced a new path for research and algorithm development. Based on the initial results, this approach appears to be promising.

In this work, we aim to investigate the applicability of regularized LP-based RL algorithms. Our objectives include developing a practical implementation of Q-REPS using deep neural networks (DNNs), and understanding the impact of various practical design choices on the algorithm's final performance through a large-scale empirical study. Additionally, we will explore other LP-based approaches beyond Q-REPS.

Objectives

The primary objectives of this work are as follows:

- Reducing the gap between LP-based algorithms and practical implementations in large-scale settings by investigating the performance of Q-REPS when utilized with Deep Neural Networks.

- Comparing the effectiveness of the proposed methods against existing algorithms in various tasks outlined in the literature and determining optimal design choices for LP algorithms employing DNNs.
- Identifying alternative LP-based approaches to Q-REPS.

Structure of the thesis

In chapter 1, we discuss Markov Decision Processes (MDPs) as well as dynmamic programming for solving them, and explain how Deep Reinforcement Learning (DRL) algorithms can be derived from them. In chapter 2, we introduce the canonical formulation for constructing regularized algorithms in a dynamic programming setting. In chapter 3, we present the derivation for the linear programming framework in MDPs, along with the motivation behind this approach and its advantages from an optimization perspective. Additionally, we introduce regularization within this LP framework. In chapter 4, we present a series of experiments related to two of the presented LP-based algorithms as well as their corresponding results comparing different algorithm design choices. We also compare such algorithms to other state-of-the-art methods. We conclude with a discussion and considerations for future work.

Chapter 1

MARKOV DECISION PROCESSES AND DYNAMIC PROGRAMMING

Reinforcement learning studies how an agent acts within an environment. More specifically, we are interested in studying how this agent can learn how to act - learn a behavior- in order to maximize some notoion of long term reward. Essentially, this leads to a sequential decision-making framework, whose dynamics can be explained from Markoc Decision Processes (MDPs). The several definitions and methods described in this introductory section can be found in [Sutton and Barto, 2018].

1.1 Markov Decision Processes (MDPs)

More specifically, these dynamics are usually described by an infinite-horizon discounted Markov Decision Process (MDP) [Agarwal et al., 2021], which is essentially saying that the decision making process can potentially be infinite, but that recent rewards will have bigger weights than future rewards. The dynamics of these MDPs are defined as $M = (S, A, P, r, \gamma)$ described by:

- A state space $\mathcal{S}$
- An action space $\mathcal{A}$
- A transition function P with $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ where $\Delta(S)$ is the space of probability distributions over $\mathcal{S}$. We will use $P(s'|s, a)$ to denote the probability of ending in state s' after taking the action a in state s

- A reward function $r : \mathcal{S} \times \mathcal{A} \to [0, 1]$. We will denote $r(s, a)$ as the reward associated with taking action a in state s. We will consider r to be deterministic.
- A discount factor $\gamma \in [0, 1)$ that defines the horizon of the problem.

Definitions: Policies and Values

Policies. In a given MDP $M = (S, A, P, r, \gamma, \mu)$, the agent acts within an environment according to the following process: initially, the agent finds itself at some state, sampled from the initial state distribution $s_0 \sim \nu_0$; at each time step t, the agent chooses an action $a_t \in \mathcal{A}$, obtains the immediate reward $r_t = r(s_t, a_t)$, and observes the next state s_{t+1} from $s_{t+1} \sim P(\cdot|s_t, a_t)$. The interaction record at time t, $\tau_t = (s_0, a_0, r_0, s_1, \ldots, s_t, a_t, r_t)$. is called trajectory.

In general, we consider a policy to be a decision making tactic, that can change over time given the observed trajectories; precisely, a policy is a mapping from a trajectory to an action, i.e. $\pi : H \to \Delta(A)$ where H is the set of all possible trajectories (of all lengths) and $\Delta(A)$ is the space of probability distributions over $\mathcal{A}$. A stationary policy $\pi : S \to \Delta(A)$ specifies a decision-making strategy in which the agent chooses actions based only on the current state, i.e. $a_t \sim \pi(\cdot|s_t)$. A stationary deterministic policy is of the form $\pi : S \to A$.

Note that because of the Markov Property, the state s_{t+1}, depends only on (s_t, a_t). This is why optimal policies can be stationary .

Values For a fixed policy and a starting state $s_0 = s$, we define the value function $V^\pi : S \to \mathbb{R}$ as the discounted sum of future rewards, considering the initial state.

$$V^\pi(s) = \mathbb{E}_{h\sim\pi, s_0=s}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|\pi, s_0 = s\right]. \tag{1.1}$$

where expectation is with respect to the randomness of the trajectory, that is, the randomness in state transitions and the stochasticity of π. Here, since $r(s, a)$ is bounded between 0 and 1, we have $0 \leq V^\pi(s) \leq \frac{1}{1-\gamma}$.

Similarly, the state-action value (or Q-value) function $Q^\pi : S \times A \to \mathbb{R}$ is defined as

$$Q^\pi(s, a) = \mathbb{E}_{h\sim\pi, s_0=s, a_0=a}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|\pi, s_0 = s, a_0 = a\right]. \tag{1.2}$$

and $Q^\pi(s, a)$ is also bounded by $\frac{1}{1-\gamma}$. The intuition is now that we can estimate the value of an initial state, conditioned on starting with action a.

The objective

Therefore, we can introduce the general objective of MDPs, which is to find the policy that maximizes the expected return, starting from any possible state. The more specific objective for reinforcement learning, in general, is to achieve this same objective without prior knowledge of the MDP parameters themselves.

$$\max_{\pi} \mathbb{E}_{h\sim\pi, s_0=s}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|\pi, s_0 = s\right] = \max_{\pi} V^{\pi}(s) \tag{1.3}$$

where s_0 is drawn from an initial state distribution ν_0.

Bellman Optimality Equations

The Bellman Optimality equations are a set of fixed point equations based on a recursion in the Q-function. Firstly, for a stationary policy, we define the value function and the state-action function as in 1.4 and 1.5 respectively. We then say that Q satisfies the Bellman Optimality Equations if equation 1.6 holds for all states and actions.

$$V^{\pi}(s) = Q(s, \pi(\cdot|s)) \tag{1.4}$$

$$Q^{\pi}(s, a) = r(s, a) + \gamma \mathbb{E}_{a\sim\pi(\cdot|s,a), s'\sim P(\cdot|s,a)} V^{\pi}(s') \tag{1.5}$$

$$Q(s, a) = r(s, a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)}\left[\max_{a'\in A} Q(s', a')\right] \tag{1.6}$$

In words, this is saying that the value of taking action a in the current state, is equal to its immediate reward plus the expected future value. If this recursion holds, we can be sure to have found an optimal solution. In other words, $Q = Q^*$ if and only if it satisfies the Bellman Optimality Equations. Moreover, the deterministic optimal policy is defined as

$$\pi(s) = \arg\max_{a\in A} Q^*(s, a)$$

Similarly, we can define the value function to be:

$$V_Q = \max_{a\in A} Q(s, a)$$

In this manner, we obtain the Bellman Optimality Operator 1.7. The solution to the Bellman Optimality equations will be then a fixed point of the operator $\mathcal{T}$. That is, Q will be optimal when equation 1.8 holds.

$$\mathcal{T}Q := r + \gamma P V_Q \tag{1.7}$$

$$\mathcal{T}Q = Q \tag{1.8}$$

Value Iteration

Under the terms defined in the previous sections, we can develop a simple practical algorithm based on dynamic programming. More specifically, we can find a policy by iteratively applying the bellman operator, starting from some initial Q_0:

$$Q_{k+1}(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a'} Q_k(s',a') \right] \forall (s,a) \in \mathcal{S} \times \mathcal{A} \tag{1.9}$$

$$\pi_{k+1}(a|s) = \begin{cases} 1 & \text{if } a = \arg\max Q_k(s,\cdot) \\ 0 & \text{otherwise} \end{cases} \tag{1.10}$$

We can observe that at each iteration, we incur in an error that we expect to minimize in future iterations (1.11).

$$\left(r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a'} Q_k(s',a') \right] - Q_{k+1}(s,a) \right) \tag{1.11}$$

1.1.1 Towards Deep RL

With the dynamic programming method defined previously, we can craft a practical algorithm converging to the optimal action value function. The exact solution would require us to know the transition function P, which we usually have no access to. Usually, the best we can do is to sample from the environment and estimate it. Suppose we have availability of samples from an environment $\mathcal{B} = (r_n, s_n, a_n, s'_n)$. Another issue is that in reality, problems usually have very large action-state spaces, and it is unfeasable to represent in an exact manner the Q-function. Therefore, we would need some sort of function approximation to represent Q. The most popular way to do so is using NNs. Therefore, by defining a Q function with a neural network, Q_θ we could learn its representation by using the value iteration objective as the objective function to minimize, which is widely known as the squared bellman error (SBE). [1]:

$$\min_\theta \mathbb{E}_{\mathcal{B}} \left[\left(r_i + \gamma \max_{a'} Q_{\bar{\theta}}(s'_i, a') - Q_\theta(s_i, a_i) \right)^2 \right] \tag{1.12}$$

This objective function is commonly used in practical deep reinforcement learning (RL) algorithms and is the standard approach for finding the optimal Q_θ. A notable algorithm employing this approach is the Deep Q-Network (DQN) algorithm, which was groundbreaking in demonstrating that it could solve most of the Atari benchmarks [Mnih et al., 2013].

[1] $Q_{\bar{\theta}}$ is the state-action function of a previous iteration.

Algorithm 1 Deep Q-Network (DQN)

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q_θ with random weights θ
for episode $= 1, M$ **do**
  Initialize state s_1
  for timestep $= 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(s_t, a; \theta)$
    Execute action a_t, observe reward r_t and new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Sample random minibatch of transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{D}$
    Set target $y_i = \begin{cases} r_i & \text{if episode terminates at step } i+1 \\ r_i + \gamma \max_{a'} Q_{\bar{\theta}}(s_{i+1}, a') & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_i - Q_\theta(s_i, a_i))^2$ with respect to θ
    Every C steps, reset $\bar{\theta} = \theta$
  end for
end for

Chapter 2

REGULARIZED DYNAMIC PROGRAMMING

Regularization has become a widely used technique to improve performance of machine learning algorithms. Its application extends to reinforcement learning, where it has been mainly used to improve algorithm performance by improving exploration, and increasing robustness. We will compare regularized and non-regularized RL objectives. Next, we'll examine different regularization functions and discuss the main differences and benefits of this new objective. Finally, we will observe how new algorithms are derived from this regularized setting. In the RL framework, regularization can be viewed as an additional term within our objective function 2.1, usually acting upon the set of optimal policies. This term imposes constraints on the solution, relative to the regularization function, and in principle should help in the search for an optimal policy.

In this chapter, we adopt the formulation from [Geist et al., 2019] to define various regularized dynamic programming procedures. We find this formulation helpful for providing a clear presentation of different regularization strategies applicable within this dynamic programming framework.

$$\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) - \alpha \Omega(\pi) \right] \tag{2.1}$$

Here, $\Omega(\pi)$ is a convex function, acting on the policy.

2.1 Redefining Value Iteration

In the previous chapter we saw how in value iteration we defined the following dynamic programming schema:

$$Q_{k+1}(s,a) = r(s,a) + \gamma P\left[\max_{a'} Q_k(s',a')\right] \forall (s,a) \in \mathcal{S} \times \mathcal{A}$$

$$\pi_{k+1}(s) = \arg\max_{a \in \mathcal{A}} Q_k(\cdot, a)$$

More specifically, the value iteration procedure can be equivalently reformulated as:

$$\pi_{k+1}(s) = \arg\max_{a \in \mathcal{A}} Q_k(\cdot, a) \leftrightarrow \pi_{k+1} = \arg\max_{\pi} \langle \pi, Q_k \rangle$$

$$Q_{k+1} = r + \gamma P \langle \pi_{k+1}, Q_k \rangle$$

where $\langle \pi, Q_k \rangle = \sum_{a \in \mathcal{A}} \pi(a|s) Q_k(s,a)$ and $Pv = \sum_{s'} P(s'|s,a) v(s')$

Under this value-iteration schema, regularization could be added to either the policy evaluation or policy improvement steps, as well as both steps. We will consider two well-known regularization functions: entropy and Kullback-Leibler (KL) regularization.

2.2 Entropy Regularization

Entropy is understood as a measure of how far the policy is of being just a random policy. When we include this term in our policy update, we are encouraging the policy to be more adaptable to changes. In other words, it allows the policy to explore better the different set of states -it commits less to observed high reward states-. Firstly, we define entropy as $\mathcal{H}(\pi) = -\langle \pi, ln\pi \rangle$. Then, we define the dynamic prpgramming schema as following.

$$\pi_{k+1} = \arg\max_{\pi} \langle \pi, Q_k \rangle + \alpha \mathcal{H}(\pi)$$

$$Q_{k+1} = r + \gamma P(\langle \pi_{k+1}, Q_k \rangle + \alpha \mathcal{H}(\pi_{k+1})$$

When the regularization function is the negative entropy $-\mathcal{H}(\pi)$, a known convex function, the optimal solution can be found by the Legendre-Fenchel transform 2.2, which yields the analytical solution 2.4.

$$\Omega^*(Q) = \max_{\pi} \{\langle \pi, Q \rangle - \Omega(\pi)\}, \nabla\Omega^*(Q) = \arg\max_{\pi} \{\langle \pi, Q \rangle - \Omega(\pi)\} \quad (2.2)$$

$$\tau \ln \left[\sum_a \exp(\frac{Q_k}{\tau}) \right] = \max_\pi (\pi, q_k) + \tau H(\pi) \tag{2.3}$$

$$\pi_{k+1} \propto e^{(Q_k/\tau)} \tag{2.4}$$

From this schema, we can define algorithms such as soft-dqn from the SBE objective 2.6 [Liu et al., 2020]. The policy can also be learnt, which would be necessary for continuous actions settings. This is exactly how SAC [Haarnoja et al., 2018] works. The actor (policy) is learned through 2.7 and the critic (q function) is learnt through 2.5, which is matematically equivalent to 2.6

$$\min_\theta \mathbb{E}_\mathcal{B} \left[\left(r_i + \gamma \max_{a'} Q_{\bar{\theta}}(s_i', a') - \tau \ln \pi_{k+1}(a|s) - Q_\theta(s_i, a_i) \right)^2 \right] \tag{2.5}$$

$$\min_\theta \mathbb{E}_\mathcal{B} \left[\left(r_i + \gamma\tau \ln \left[\sum_a \exp(\frac{Q_{\bar{\theta}}}{\tau}) \right] - Q_\theta(s_i, a_i) \right)^2 \right] \tag{2.6}$$

$$J(w) = \hat{\mathbb{E}}_N \left[\mathbb{E}_{a \sim \pi_\psi(\cdot|s_n)} \left[Q_\theta(s_n, a) - \alpha(\ln \pi_\psi(a|s_n)) \right] \right] \tag{2.7}$$

2.3 KL Divergence Regularization

Kulback-Leibler divergence is a measure of how different a distribution is from another one. Adding a KL regularization term to the iterative schema, essentially controls that the next policy is not too far from the previous policy or from a fixed policy, depending on how we define the regularization. We define the KL divergence as: $D_{KL}(\pi_1||\pi_2) = \langle \pi_1, \ln \pi_1 - \ln \pi_2 \rangle$ And the schema turns into:

$$\pi_{k+1} = \arg\max_\pi \langle \pi, Q_k \rangle - \lambda D_{KL}(\pi||\pi_k)$$

$$Q_{k+1} = r + \gamma P(\langle \pi_{k+1}, Q_k \rangle - \lambda D_{KL}(\pi_{k+1}||\pi_k))$$

Using KL-divergence as regularization, we find the analytical solution again by the Legendre-Fenchel transform 2.9. In this case, it is necessary to learn an actor in any case given that we can't store all past Q_k. The policy network π_ψ is lernt through the objective 2.9, which is used by TRPO [Schulman et al., 2017a] to the extent that in their work, instead of regularizing the actor, they constrain it.

$$\pi_{k+1} \propto \pi_k e^{(Q_k/\tau)} = \pi_0 \exp \left(\sum_{j=0}^{k} (Q_j/\tau) \right) \tag{2.8}$$

$$J(w) = \hat{\mathbb{E}}_N \left[\mathbb{E}_{a \sim \pi_\psi(\cdot|s_n)} \left[Q_\theta(s_n, a) - \alpha(\ln \pi_\psi(a|s_n) - \ln \pi_{\psi_{old}}(a|s_n)) \right] \right] \quad (2.9)$$

Note that both regularization functions could be combined as well, and with the tools mentioned in this chapter, one could derive a wide range of regularized algorithms by choosing different regularization functions, and combining them in the policy evaluation and policy improvement steps.

$$\pi_{k+1} = \arg\max_{\pi} \langle \pi, Q_k \rangle + \alpha \mathcal{H}(\pi) - \lambda D_{KL}(\pi || \pi_k)$$

$$Q_{k+1} = r + \gamma P(\langle \pi_{k+1}, Q_k \rangle + \alpha \mathcal{H}(\pi_{k+1}) - \lambda D_{KL}(\pi_{k+1} || \pi_k))$$

Chapter 3

MDPS FROM A LINEAR PROGRAMMING PERSPECTIVE

Even though the squared bellman error (SBE) used in the previous chapter is very popular and its use has been widely extended as the hegemonic approach to developing new RL algorithms, it has a considerable number of shortcomings from an optimization point of view. The Bellman optimality equations are a set of fixed-point equations that establish a non-linear system of equations, which is generally hard to deal with. These equations are non-convex, non-smooth, have unbounded gradients, are impossible to evaluate precisely, and provide biased estimates. To address these disadvantages, several heuristic tricks are often used in deep RL settings to stabilize training and ensure the convergence of the algorithm.

In this chapter, our aim is to explore a different approach to RL that tries to overcome the shortcomings of existing algorithms using the SBE. To do so, we will rely on a reformulation of the objective function, rewriting it as a linear program [Puterman, 2014].

3.1 Reformulation of the MDP objective

We start from our initial goal, which is to maximize the expected discounted sum of rewards.

$$\begin{aligned} R_\gamma^\pi &= \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] \\ &= \sum_{t=0}^{\infty} \mathbb{E}_\pi \left[\gamma^t r(s_t, a_t) \right] \\ &= \sum_{t=0}^{\infty} \sum_{s,a} \gamma^t P_\pi(s_t = s, a_t = a) r(s, a) \\ &= \frac{1}{1-\gamma} \sum_{s,a} (1-\gamma) \sum_{t=0}^{\infty} \gamma^t P_\pi(s_t = s, a_t = a) r(s, a) \\ &= \frac{1}{1-\gamma} \sum_{s,a} \mu_\pi(s, a) r(s, a) \end{aligned}$$

where we define $P_p i$ explicitly as the probability measure induced by executing policy π in the MDP and $\mu_\pi(s, a) = \sum_{t=0}^{\infty} \gamma^t P_\pi(s_t, a_t)$ as the discounted occupancy measure of the policy π. The occupancy measure can be understood as a joint probability distribution over the space of states and actions according to a policy π. In this way, we see that our objective, the sum of discounted rewards, is a linear function on the occupancy measure $\mu_\pi(s, a)$

$$(1-\gamma) R_\gamma^\pi = \langle \mu_\pi, r \rangle \tag{3.1}$$

For any policy, the occupancy measure must strictly satisfy the following condition, meaning that an occupancy measure is valid if and only if the following condition is true [Manne, 1960].

$$\sum_b \mu(s', b) = \gamma \sum_{s,a} P(s'|s, a) \mu(s, a) + (1-\gamma) \nu_0(s') \quad \forall s' \tag{3.2}$$

We can therefore rewrite our optimization objective that corresponds to the following Linear Program:

$$\begin{aligned} \underset{\mu \in \mathbb{R}_+^{S \times A}}{\text{maximize}} \quad & \langle \mu, r \rangle \\ \text{subject to} \quad & \sum_b \mu(s', b) = \gamma \sum_{s,a} P(s'|s, a) \mu(s, a) + (1-\gamma) \nu_0. \end{aligned}$$

Any solution μ^* to this LP will maximize the total discounted reward, and the induced policy will also be optimal as the actions taken by $\pi_{\mu^*}(a|s)$ will yield maximal reward. The intuition is that we will find a policy that eventually ends up choosing actions that will lead to states with high rewards. Therefore, the maximum stationary state action probability will be the one that that visits states associated with high rewards very often. We can define the policy from any valid occupancy measure as : $\pi_\mu(a|s) = \frac{\mu(s,a)}{\sum_{a'} \mu(s,a')}$

This reparametrization is very useful because it allows rewriting our goal as a single optimization function with a system of linear equations, avoiding fixed point equations and recursions. Nevertheless, this LP formulation as it is, has a set of shortcomings as well. Firstly, it is unrealistic in real-world problems since this LP has way too many variables and constraints to be solved by a regular LP solver. Additionally, in reality we can't hope to have access to the transition matrix P, and we will need to work with sample transitions. In conclusion, we will need to introduce some modifications in order to make this LP formulation work in a practical scenario.

3.2 Regularized Linear Programming

The regularized MDP formulation through LP lenses is simple in the sense that we just add a regularization function to the LP in 3.1. Two prominent regularized RL algorithms derived from this objective are REPS [Peters et al., 2010] and Q-REPS [Bas-Serrano et al., 2021]. Both include a linear relaxation of some of the constraints in this LP in order to make it practical in real world scenarios. In this work we mainly focus on Q-REPS given that it is a posterior work that while being heavily inspired by REPS, improves some of its limitations. As we will see, this approach is a strong alternative for regularized dynamic programming.

$$\begin{aligned} \underset{\mu \in \mathbb{R}_+^{S\times A}}{\text{maximize}} \quad & \langle \mu, r \rangle - \alpha\Omega(\mu) \\ \text{subject to} \quad & \sum_b \mu(s', b) = \gamma \sum_{s,a} P(s'|s,a)\mu(s,a) + (1-\gamma)\nu_0, \end{aligned}$$

3.3 Practical algorithms

3.3.1 Q-REPS

In this section, we introduce the algorithm that is the subject of our main experiments. Q-REPS is derived from the regularized LP formulation of MDPs. More specifically, it uses both the (negative) entropy and KL-divergence as regularization functions. The authors introduce a relaxation on the original LP, adding another set of primal variables $d \in \mathbb{R}^{S\times A}$ and a feature map $\phi : \mathbb{R}^{X\times A} \to \mathbb{R}^m$ where d can be seen as a mirror image of μ. Their objective LP is the following.

$$\begin{aligned} \underset{\mu\in\mathbb{R}_+^{S\times A}}{\text{maximize}} \quad & \langle \mu, r\rangle - \eta D_{\text{KL}}(\mu||\mu_0) - \alpha H(d||d_0) \\ \text{subject to} \quad & \sum_b d(s',b) = \gamma \sum_{s,a} P(s'|s,a)\mu(s,a) + (1-\gamma)\nu_0, \\ & \sum_{s,a} d(s,a)\phi(s,a) = \sum_{s,a} \mu(s,a)\phi(s,a) \end{aligned} \tag{3.3}$$

The solutions to this problem as shown in [Bas-Serrano et al., 2021] are found by Lagrangian duality. Precisely, the Q-function and the value function naturally emerge as Lagrangian multipliers, where $\Delta_\theta(s,a)$ represents the Bellman error, akin to the soft-DQN Bellman error in (2.6), and $Q_\theta(s,a) = \langle \theta, \phi(s,a)\rangle$ denotes the Q-function parameterized by θ. The optimal θ^* is found by optimizing the convex function in (3.9). This approach introduces a new objective function, which the authors define as the Logistic Bellman Error (LBE), dependent solely on θ - the Q-function - see (3.8).

$$\Delta_\theta(s,a) = r(s,a) + \gamma P V_\theta - Q_\theta(s,a) \tag{3.4}$$

$$V_\theta(s) = \alpha \log\left(\sum_a \pi_0(a|s) e^{Q_\theta(s,a)/\alpha}\right) \tag{3.5}$$

$$\mu^*(x,a) \propto \mu_0(x,a) e^{\Delta^*(x,a)/\eta} \tag{3.6}$$

$$\pi_d^*(a|x) = \pi_0(a|x) e^{(Q^*(x,a) - V^*(x))/\alpha} \tag{3.7}$$

$$\mathcal{G}(\theta) = \eta \log\left(\sum_{s,a} \mu_0(s,a) e^{\Delta_\theta(s,a)/\eta}\right) + (1-\gamma)\langle \nu_0, V_\theta\rangle \tag{3.8}$$

In an empirical setting, the Empirical Logistic Bellman Error (ELBE) is defined in (3.9), where we now use $\hat{\Delta}_\theta(s,a,s')$ as our empirical Bellman error, as given in (3.10). The policy is determined using an approximate policy iteration procedure,

with updates defined in (3.11), which shares a similar solution to the updates in (2.8).

$$\hat{\mathcal{G}}(\theta) = \eta \log\left(\frac{1}{N}\sum_{n=1}^{N} e^{\hat{\Delta}_\theta(s,a,s')/\eta}\right) + (1-\gamma)\langle \nu_0, V_\theta\rangle \tag{3.9}$$

$$\hat{\Delta}_\theta(s,a,s') = r(s,a) + \gamma V_\theta(s') - Q_\theta(s,a) \tag{3.10}$$

$$\pi_{k+1}(a|s) \propto \pi_0(a|s) e^{\frac{1}{\alpha}\sum_{j=0}^{k} Q_{\theta,j}(s,a) - V_{\theta,j}(s)} \tag{3.11}$$

In an empirical setting, the objective in (3.9) is biased due to the conditional expectation taken over s' within the exponent. It is shown that this bias can be controlled by the regularization coefficient (see Theorem 2 of [Bas-Serrano et al., 2021]). Because of this, an alternative practical algorithmic procedure defined in the paper, with unbiased gradients, is based on a reparametrization of (3.9), defined by the following objective:

$$S_k(\theta, z) = \sum_n z(n)\left(\hat{\Delta}_\theta(s_n, a_n, s'_n) - \eta \ln(N z(n))\right) + (1-\gamma)(\nu_0, V_\theta) \tag{3.12}$$

It can be shown that indeed: $\min_\theta \mathcal{G}(\theta) = \min_\theta \max_z S_k(\theta, z)$ (see Proposition 2 of [Bas-Serrano et al., 2021]). To construct an unbiased stochastic gradient for θ we sample and index n from the distribution $z_{k,t}$ and let $(s, a, s') = (s_{k,n}, a_{k,n}, s'_{k,n})$. We then sample a state $\bar{s} \sim \nu_0$ and two actions, $a' \sim \pi_\theta(\cdot|s')$ and $\bar{a} \sim \pi_\theta(\cdot|\hat{s})$ (see Proposition 3 of [Bas-Serrano et al., 2021]). Finally let

$$\hat{g}_{k,t} = \gamma\phi(s', a') - \phi(s, a) + (1-\gamma)\phi(\bar{s}, \bar{a}) \tag{3.13}$$

In this setting, we perform stochastic updates on θ, and all entries in the z distribution are updated using an exponentiated gradient descent step to ensure that the parameters do not become negative. With both of the defined objectives in this section we describe the following two algorithms to solve the proposed LP:

Algorithm 2 ELBE Q-REPS

Initialize Q_θ with random parameters θ;
Initialize π_0 with uniform distribution across all actions;
for $k = 0, 1, 2, \ldots, K-1$ **do**
  Run π and collect N sample transitions $\{(s_n, a_n, r_n, s'_n)\}_{n=1}^N$
  for $t = 1, 2, \ldots, T$ **do**
    $\theta_{k,t+1} \leftarrow \theta_{k,t} - \rho \nabla_\theta \hat{\mathcal{G}}(\theta)$
  end for
  $\theta_k = \frac{1}{T}\sum_t \theta_{k,t}$
  $\pi_{k+1}(a|s) \propto \pi_0(a|s) e^{\frac{1}{\alpha}\sum_{j=0}^{k}(Q_{\theta_j}(s,a) - V_{\theta_j}(s))}$
end for

Algorithm 3 MinMax-Q-REPS

Initialize Q_θ with random parameters θ
Initialize π_0 with uniform distribution across all actions
for $k = 0, 1, 2, \ldots, K-1$ **do**
  Run π and collect N transitions $\{(s_n, a_n, r_n, s'_n)\}_{n=1}^{N}$
  Initialize $z_{k,0} \sim \mathcal{U}(N)$
  for $t = 1, 2, \ldots, T$ **do**
    $\theta_{k,t+1} \leftarrow \theta_{k,t} - \rho \hat{g}_{k,t}$
    $h_{k,t}(n) = \hat{\Delta}_\theta(s_n, a_n, s'_n) - \eta \ln(N z(n))$
    $z_{k,t+1}(n) \propto z_{k,t}(n) e^{(\beta h_{k,t}(n))}$
  end for
  $\theta_k = \frac{1}{T} \sum_t \theta_{k,t}$
  $\pi_{k+1}(a|s) \propto \pi_0(a|s) e^{\frac{1}{\alpha} \sum_{j=0}^{k} (Q_{\theta_j}(s,a) - V_{\theta_j}(s))}$
end for

3.3.2 Primal-Dual Approximate Policy Iteration

In this section, we introduce a novel algorithm based on the dual program of the linear programming (LP) model presented in the previous section. While Q-REPS is rooted on the primal form of the LP, which focuses in obtaining an occupancy measure μ from which an optimal policy can be derived, our approach leverages the dual of this LP. Specifically, we aim to develop a policy evaluation subroutine that can effectively evaluate any given policy. Our proposed algorithm can be viewed as an approximate policy iteration algorithm, using the LP formulation to construct a policy evaluation subroutine. To evaluate the policy, we solve an LP that characterizes the action-value function of the policy in question. This approach is not explicitly detailed in the existing literature. For the policy iteration subroutine, we employ a softmax policy update, a method widely adopted in previous works (see [Kakade, 2001], [Agarwal et al., 2020], [Abbasi-Yadkori et al., 2019]).

Policy Evaluation Subroutine: We fix a policy π, and define the operator $\mathcal{M}$ as $\mathcal{M}_\pi Q = \sum_a \pi(a|s) Q_\pi(s,a)$. We then define the following LP, coming from the dual of 3.1, and solve it by lagrangian duality (3.14)

$$\begin{aligned} \min_{V \in \mathbb{R}^S} \quad & (1-\gamma)\langle \nu_0, V_\pi \rangle \\ \text{subject to} \quad & Q = r_\pi + \gamma P V_\pi, \quad \pi \in \mathcal{S} \times \mathcal{A}, \forall s \in S, \forall a \in \mathcal{A} \\ & V_\pi = \mathcal{M}_\pi Q, \quad \forall s \in \mathcal{S} \end{aligned}$$

$$\mathcal{L}(Q, \mu) = \langle \mu, r_\pi + \gamma P \mathcal{M}_\pi Q - Q \rangle + (1-\gamma)\langle \nu_0, V \rangle \tag{3.14}$$

With the parametrization of the action-value function as Q_θ, our problem becomes the unconstrained optimization objective:

$$\max_{\mu \in R^+} \min_{\theta} \mathcal{L}(\mu, \theta) \tag{3.15}$$

$$\mathcal{L}(\mu, \theta) = \mathbb{E}_{\substack{(s,a)\sim\mu \\ s' \sim P(\cdot|s,a)}} \left[\mu(s,a) \left(r_\pi(s,a) + \gamma P \mathcal{M}_\pi Q_\theta(s') - Q_\theta(s,a) \right) + (1-\gamma)\langle \nu_0, \mathcal{M}_\pi Q_\theta \rangle \right]$$

Importance Sampling Importance sampling (IS) is a technique used for estimating the expected value of $f(x)$ where x has a data distribution p. However, Instead of sampling from p, we calculate the result from sampling q.

$$\mathbb{E}_p[f(x)] = \mathbb{E}_q \left(\frac{f(X)p(X)}{q(X)} \right)$$

In this case, IS is used to correct the bias when the joint state action distribution under which the agent is acting (behavior policy μ) is different from the distribution that the agent is learning about (target μ).

$$\hat{\mathbb{E}}_{\mu}[f(x)] \approx \frac{1}{n}\sum_{i=1}^{n}\frac{f(X_i)\mu(X_i)}{\mu_{old}(X_i)}, \quad X_i \sim \mu_{old}. \tag{3.16}$$

When addressing a practical approach to solving the objective, we will need to work with sample transitions which will have been sampled from an old occupancy measure μ_{old} instead of the target μ. This can introduce bias into our objective. To mitigate this bias, we will use importance sampling in order to work with sample transitions $(s_n, a_n, s'_n)_{n=1}^{N}$ in a practical setting.

$$\mu_{k+1}, \theta_{k+1} = \max_{\mu \in R^+} \min_{\theta} \frac{1}{N}\sum_{n=1}^{N}\left[\frac{\mu(s_n,a_n)}{\mu_k(s_n,a_n)}\left(r_\pi(s_n,a_n) + \gamma\mathcal{M}_\pi Q_\theta(s'_n) - Q_\theta(s_n,a_n)\right)\right] + (1-\gamma)\langle\nu_0, \mathcal{M}_\pi Q_\theta\rangle \tag{3.17}$$

However, instead of estimating μ, in large scale settings it is easier to consider the positive ratio $z(n) = \frac{\mu(s_n,a_n)}{\mu_k(s_n,a_n)}$, where z is just a vector with n entries, instead of storing a full occupancy measure of dimensions $\mathcal{S} \times \mathcal{A}$ at each step. Define also $\delta_{\pi,\theta}(s_n, a_n, s'_n) = r_\pi(s_n, a_n) + \gamma\mathcal{M}_\pi Q_\theta(s'_n) - Q_\theta(s_n, a_n)$. Our optimization problem can be formulated as:

$$z_{k+1}, \theta_{k+1} = \max_{z \in R^+} \min_{\theta} \frac{1}{N}\langle z, \delta_{\pi,\theta}\rangle + (1-\gamma)\langle\nu_0, \mathcal{M}_\pi Q_\theta\rangle \tag{3.18}$$

Note that here, z is not bounded, and in the optimization process it could become really large. With the objective to avoid such behavior we add an unnormalized entropy regularization to z where η is merely a regularization coefficient.

$$\mathcal{H}(z) = \sum_{n} z(n)\ln z(n) - z(n)$$

$$\hat{J}(\theta, z) = \frac{1}{N}\sum_{n} z(n)\delta_{\pi,\theta}(n) - \eta\mathcal{H}(z) + (1-\gamma)\langle\nu_0, \mathcal{M}_\pi Q_\theta\rangle \tag{3.19}$$

As we explained in the beginning, this optimization process is used as a policy evaluation subroutine (3.20), and after each evaluation step, we update the fixed policy 3.21, where τ serves as a temperature or learning rate coefficient for controlling policy updates. This process is reflected in Algorithm 4, which bears similarities to Q-REPS.

$$z_{k+1}, \theta_{k+1} = \max_{z \in R^+} \min_{\theta} J(\theta, z) \tag{3.20}$$

$$\pi_{k+1} = \pi_k \text{ softmax } (\tau Q_{\theta_{k+1}}) \tag{3.21}$$

The gradients for θ are constructed as in MinMax Q-REPS 3.13, and we update all entries of vector z following an exponentiated gradient descent step. There are two main differences between this approach and Q-REPS. The first is that here, z is not a probability distribution and it isn't constrained to be in the simplex space, therefore we don't need to renormalize after each step. The second difference resides in how the value function is computed.

Algorithm 4 Primal-Dual Approximate Policy Iteration

Initialize π_0, Q_θ;
for $k = 0, 1, 2, \ldots, K-1$ **do**
  Run π_k and collect sample transitions $\{(s_n, a_n, r_n, s'_n)\}_{n=1}^N$
  Initialize $z_{k,0} = \mathbb{1}_N$
  primal-dual policy evaluation:
  for $t = 1, 2, \ldots, T$ **do**
    $\theta_{k,t+1} \leftarrow \theta_{k,t} - \rho \hat{g}_{k,t}$
    $h_{k,t}(n) = \hat{\delta}_\theta(s_n, a_n, s'_n) - \eta \ln(z(n))$
    $z_{k,t+1}(n) = z_{k,t}(n) e^{(\beta h_{k,t+1}(n))}$
  end for
  primal-dual policy iteration: $\pi_{k+1} \propto \pi_k$ softmax $(\tau Q_{\theta_{k+1}}(s, a))$
end for

Chapter 4

IMPLEMENTATION OF LP-BASED RL ALGORITHMS

In this chapter, we present key adaptations and experiments conducted on LP-based reinforcement learning (RL) algorithms in a large-scale setting. Our code [1] relies on the single-file code implementations provided by [Huang et al., 2021] and the environemnts provided by the Gymnasium library [Brockman et al., 2016].

We found that previous implementations of Q-REPS [2] [3] were challenging to reproduce. While the original work doesn't rely on neural networks, other implementations have, but failed to offer a clear analysis of the different algorithmic components. To address these issues, we provide a single-file, reproducible implementation of Q-REPS. Our implementation includes detailed code-level optimizations and comprehensive documentation of each component of the algorithm. This approach ensures that researchers and practitioners can easily replicate our results and gain a deeper understanding of the different components of Q-REPS.

In the subsequent sections, we will detail the specific adaptations made to the algorithms for large-scale settings, the experimental setup, and the results obtained. These contributions aim to improve reproducibility and the practical application of LP-based RL algorithms.

[1] https://github.com/niicovila/Entropy-RL-Linear-Programming
[2] https://github.com/sebascuri/qreps
[3] https://github.com/sebimarkgraf/q_reps

4.1 Large-scale Q-REPS Experiments

Data collection and optimization loop

We consider an on-policy reinforcement learning setting for discrete environments and chose to parameterize the Q-function and the policy using deep neural networks (DNNs). Additionally, we explored the parameterization of the sampler in the saddle point optimization procedure defined in Equation 3.12. We conducted experiments comparing both proposed algorithms for Q-REPS and examined common design choices shared by these algorithms.

We initialize a vectorized environment that runs E (typically independent) environments (`num_envs`) sequentially or in parallel using multiple processes [Mnih et al., 2016]. The transitions produce a batch of E observations and require a batch of E actions to step all the environments. Each environment is run until we gather N steps accross all environments. We then run `update_epochs` epochs of stochastic gradient based optimization. The updates are performed in minibatches, a common approach in other deep RL methods. We create n minibatches of size $\frac{N}{m}$ each, and within each optimization epoch, we separate the full batch into individual transitions, and assign each one randomly to minibatches. This approach leads to higher diversity of data within the minibatch.

We conducted a wide range of experiments regarding Q-REPS. Our main interest is studying the main differences between optimizing Equation 3.9 directly or solving the optimization problem via the saddle point procedure in 3.12 within a DNN setting. We performed practical experiments regarding advantage estimation, use of different policy losses, regularization coefficients, network architecture, and other important hyperparameters to understand which design choices might be beneficial for this particular algorithm.

Policy Parameterization

We parameterize the policy because the update defined in 3.11 would require storing all past Q-functions, which is impractical. Note that this update is very similar to the one defined in Equation 2.8. The Q-REPS policy update is indeed a KL-regularized update. Therefore, we propose the policy objective in Equation 4.1, inspired by the solutions in 2.9 (see section 4.1 in [Geist et al., 2019])to learn a parametrized actor π_ψ.

$$J(\psi) = \hat{\mathbb{E}}_N \left[\mathbb{E}_{a \sim \pi_\psi(\cdot|s_n)} \left[Q_\theta(s_n, a) - V_\theta(s_n) - \alpha(\ln \pi_\psi(a|s_n) - \ln \pi_{\psi_{\text{old}}}(a|s_n)) \right] \right] \tag{4.1}$$

The other considered approach is minimizing the weighted negative log likelihood, as proposed in [Sebastian Markgraf, 2022]. This negative log-likelihood

learns a parametrized policy whose actions distribution across states should converge to the close form solution that we obtain in (3.11).

$$WNLL(\psi) = -\hat{\mathbb{E}}_N \left[\mathbb{E}_{a \sim \pi_\psi(\cdot|s_n)} \left[\ln \pi_\psi(a_n|s_n) e^{(Q_\theta(s_n,a_n) - V_\theta(s_n))/\alpha} \right] \right] \tag{4.2}$$

Both of the proposed policy objectives are differentiable with respect to the policy parameters, and we can use stochastic gradient algorithms to update them. The two proposed methods to solve Q-REPS within a DNN setting are summarized in Algorithms 5 and 6

Algorithm 5 Q-REPS ELBE

Initialize critic network Q_θ and actor network π_ψ with random parameters θ, ψ
for $k = 1$ to K **do**
  Run π_ψ and store a batch of N transition tuples $\{(s_n, a_n, r_n, s'_n)\}_{n=1}^N$
  for $t = 1$ to T **do**
    Shuffle transitions and create M minibatches
    for $m = 1$ to M **do**
      Sample a new mini-batch of transitions (s_m, a_m, r_m, s'_m)
      $\theta \leftarrow \arg\min_\theta \hat{\mathcal{G}}(\theta)$
      $\psi \leftarrow \arg\min_\psi$ Objective Function Eq. 4.1 or 4.2
    end for
  end for
end for

Algorithm 6 Q-REPS MinMax

Initialize critic network Q_θ and actor network π_ψ with random parameters θ, ψ
for $k = 1$ to K **do**
  Run π_ψ and store a batch of N transition tuples $\{(s_n, a_n, r_n, s'_n)\}_{n=1}^{N}$
  Shuffle transitions and create M minibatches
  for $m = 1$ to M **do**
    if P=True **then**
      Initialize network z_ϕ
    else
      Initialize $z_{k,0}(n) \sim \mathcal{U}(N//M)$
    end if
    for $t = 1$ to T **do**
      Sample a new mini-batch of transitions (s_m, a_m, r_m, s'_m)
      $\theta \leftarrow \arg\min_\theta S_k(z, \theta)$
      if P=True **then**
        $\phi \leftarrow \arg\min_\phi -S_k(z_\phi, \theta)$
      else
        $z_{k,t+1} \propto z_{k,t} e^{\beta(\hat{\Delta}_\theta(s_n, a_n, s'_n) - \alpha \ln(N z(n)))}$
      end if
      $\psi \leftarrow \arg\min_\psi$ Objective Function Eq. 4.1 or 4.2
    end for
  end for
end for

Experiments

In this section, we present the experiments conducted with Q-REPS. We use CartPole-v1 [Barto et al., 1983], LunarLander-v2 [Brockman et al., 2016], and Acrobot-v1 [Sutton, 1995] as benchmark environments to evaluate and compare the performance of Q-REPS against other state-of-the-art algorithms. Additionally, we conduct an analysis of various design choices within our algorithm to determine which configurations yield the best performance. The objective is to understand how these design choices influence the overall performance of the algorithm. In addition to that, we aimed to compare best practices between the proposed methods (ELBE and MinMax) and to understand the fundamental design differences between these methods.

Experimental Design: As highlighted by [Andrychowicz et al., 2020], hyperparameter choices can exhibit significant interactions, necessitating their concurrent tuning to avoid misleading results from varying a single hyperparameter while keeping others fixed. To address this, we structured our experiments by grouping hyperparameter choices into thematic groups where interactions were anticipated. For each hyperparameter configuration, we trained three models with different random seeds, each for 100,000 environment steps. We then performed two analyses for each configuration, complementing each other.

For each potential value of a hyperparameter, we plot the 95th percentile of rewards for the selected design choice as well as the distribution of parameter values among the top-performing models for each game in the experiment (top 5% configurations). This is informative because over-representation of certain values in high-performing models might highlight their significance in achieving good performance.

We identified six thematic groups for our experiments. Detailed experimental designs and plots for each group are provided in Appendices C-H. For each group, we also included learning rates and regularization coefficients, considering their potential interactions with all other design choices.

Training Setup (based on the results in Appendix C and F)

We examine several fundamental parameters of the optimization process: the number of sample transitions needed to learn an effective policy, the number of minibatches in which to break each batch, the number of update epochs per step, various optimizers, and the number of parallel environments used.

Results and Interpretation: For ELBE, iteration size does not appear to be a critical factor and should be tuned according to each environment, although MinMax

demonstrates a slight preference for smaller iteration sizes (fig. C.2). Regarding minibatch size, again, there is no optimal choice across all environments; however, ELBE in CartPole-v1 benefits from smaller minibatch sizes (the inverse to `num_minibatches`), whereas MinMax performs better with larger ones (fig. C.3). This might be caused because the MinMax alternative learns a distribution over batches of data within each step, and it might perform better when it has a larger batch to learn from. A higher number of update epochs consistently improves ELBE's performance, whereas MinMax doesn't display a clear uprising trend related to this choice(fig. C.4). Both RMSprop and Adam are effective optimizers for both the policy and the q-function in both ELBE and MinMax.(fig. C.8, C.9). Lastly, implementing a linear decay in the learning rate can enhance performance but is not essential (fig. C.13).

Policy Losses (based on the results in Appendix D)

We compare two different policy losses applicable to the Q-REPS algorithm: KL loss and WNLL loss. These losses are defined in detail in the previous section. The goal of this study is to understand how each loss impacts the performance of the Q-REPS algorithm and the learning process of the policy.

Results and Interpretation: Our experimental results indicate that the KL loss generally exhibits superior performance for ELBE optimization across most benchmarks, with the exception of Acrobot-v1. Conversely, the WNLL loss proves more effective for the MinMax algorithm.

We interpret this as a result of the different behaviors that both algorithms exhibit during their training phases. For MinMax, the algorithm learns a sampler distribution based on the transition samples it has encountered. We believe that MinMax has a higher optimistic bias, and the estimates for actions and states that have not been observed during training are more penalized than those in ELBE. This higher penalization might be mitigated by optimizing directly with ELBE, which could reduce the negative impact of unseen states on the estimates, leading to a more stable and less overly pessimistic update for these states. This likely results in more balanced estimates across both seen and unseen states during training. Since the WNLL loss only leverages observed actions to optimize the policy, it aligns more effectively with the MinMax algorithm's approach of focusing more accurately on observed samples. That might be the reason why a WNLL loss outperforms the KL loss in this specific context (D.2).

Regularization Coefficients (based on the results in Appendix E)

We investigate the effectiveness of various regularization coefficients for the variables α and η. This study examines the impact of using identical coefficients for both variables as well as experimenting with different values. The goal is to identify an optimal regularization strategy.

Results and Interpretation: **Results and Interpretation**: Tuning the regularization coefficients is key for improving performance in each environment, especially in ELBE since it controls the bias introduced by the loss function, and should be customized to the specific conditions of each setting. In most cases, η can be set equal to α since they typically have similar magnitudes. However, tuning both parameters separately, if feasible, is advisable to find a better combination (fig. E.3 & E.2).

Advantage Estimation (based on the results in Appendix F)

We compare two commonly used advantage estimators. Additionally, we test advantadge normalization, vary the number of parallel environments used and total iterations, as this changes the length of the experience fragments collected in each step. Additionally we explore the use of a periodically updated target Q network as a method for stabilizing training. This approach is known to be crucial for stabilizing training in DQN [Mnih et al., 2013]. Finally, we also included update epochs in our experiment to investigate if any of the advantadge estimators learns a policy faster than the other.

- **N-step returns** ([Sutton and Barto, 2018]) with $n = 1$, which is used in the theoretical algorithm, defined as:

$$G_t^{(n)} = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n V(s_{t+n}) \approx Q^\pi(s_t, a_t)$$

  and

$$\hat{\Delta}_\theta(s, a, s') = G_t^{(n)} - Q_\theta(s, a)$$

- **Generalized Advantage Estimation** ([Schulman et al., 2018]) defined as:

$$\hat{V}_t^{\text{GAE}} = (1 - \lambda) \sum_{N \geq 0} \lambda^{N-1} \hat{V}_t^N \approx Q^\pi(s_t, a_t)$$

  and

$$\hat{\Delta}_\theta(s, a, s') = \hat{V}_t^{\text{GAE}} - Q_\theta(s, a)$$

In the case of GAE, the estimation of advantages is impractical within each minibatch since it requires the use of future states. Therefore, we precomputed all advantages at the beginning of each iteration using full trajectories. In such case, we corrected the KL loss policy objective with importance sampling weights.

Results and Interpretation: In general, using GAE improves performance in both ELBE and MinMax (fig. F.4). When using GAE, the exact expected KL loss cannot be computed since GAE calculates the advantages of the observed samples. In this scenario, WNLL policy loss performs better in both ELBE and MinMax (fig. F.6). Interestingly, GAE reduces training time by requiring fewer update epochs for achieving good performance. Conversely, without GAE, the KL loss policy is more effective but demands more epochs in the optimization loop, leading to longer training times (fig. F.11). This flexibility suggests that in data-scarce situations, using KL loss without GAE can maximize data efficiency by performing more epochs on the same minibatch. If a generative model is available, GAE with WNLL loss is preferable for reducing training times. Using a target network can stabilize training in some cases, but it is generally not advised as it does not provide a performance improvement (fig. F.7). Normalizing advantages is advisable when using 1-step advantages but not with GAE (fig. F.3). GAE proves beneficial in both ELBE and MinMax optimizations while reducing training times. However, in our experiments, we have used a 1-step estimator with the KL policy objective, given that it will learn a policy with fewer sample transitions observed. The number of parallel environments and iteration size should be tuned to each specific scenario, particularly in environments with fixed step terminations.

Network Architecture (based on the results in Appendix G)

We investigate the impact of different neural network architectures for the policy and Q-function. Considerations include network structure and size, weight initialization, and activation functions, as well as the standard deviation of the last layer of the networks.

Results and Interpretation: We do not find architectural details to be crucial for achieving good performance of our models. While proper tuning can yield slightly higher performance, it does not lead to significant improvements. Optimization improvements are often environment-specific, indicating the need for tuning based on each environment. Overall, MinMax performed worse but seemed less sensitive to architectural changes.

Both ELBE and MinMax policy networks can safely range from 32 to 64 units in all cases (fig. G.2) and maintain a depth of about 2-4 layers (fig. G.3). The

critic performs better with a larger hidden size, and the exact value can be tuned for a specific environment by choosing between 64 and 256 units, generally using no more than 4 hidden layers (fig. G.4, G.5). The policy network benefits from greater depth and less width compared to the critic. Both ELU and Tanh activation functions are effective for the policy (fig. G.6), while Tanh is recommended for the Q-function (fig. G.7). Orthogonal with gain initialization is preferred for MinMax, while He uniform initialization is usually better for ELBE (fig. G.8). Scaling actions at the last layer of the policy network might be beneficial in some environments, but no significant improvement is observed across all environments (fig. G.9). We recommend not scaling the last layer of the Q-function (fig. G.10). When using a parameterized sampler with MinMax, Tanh activations are recommended. Other specifications of the sampler network seem to be environment-dependent (fig. G.11, G.12). Under these experiments, a parameterized sampler might be beneficial compared to a non-parameterized sampler, but such choice did not substantially improve MinMax performance (fig. G.14). This may be because architecture-level optimizations are more beneficial for ELBE, perhaps adding more stability, whereas the MinMax approach did not benefit much from tuning the architecture. It appears that the performance of this approach is improved by other parameters than network architecture. Sampler learning rate can be safely set to around 0.001 for both parameterized and non-parameterized samplers (fig. G.19).

Timestep Handling (based on the results in Appendix H)

The main parameter regarding timestep handling is the discount factor γ. This factor is important in determining the relative importance of future rewards compared to immediate rewards. We also test choices for number of environments and total iterations.

Interpretation: We observe that it is not crucial to find a specific value for γ to achieve good performance. Setting it to $\gamma = 0.99$, as we did in all our experiments by default, is sufficient to ensure good overall performance. For ELBE, fewer parallel environments are generally used to improve performance, while MinMax seems to benefit from a larger number of environments. Note that with more parallel environments, we typically construct a batch of more independent samples. This is particularly beneficial for the MinMax sampler learner, which learns a sample distribution across the minibatch. Acrobot-v1 works best with a large number of parallel environments and iterations, indicating that it performs better with more independent data.

Critic Averaging (based on the results in Appendix C-H)

We examine the effect of averaging critics after each optimization step to assess whether this improves the stability and performance of the training process.

Results and Interpretation: Critic averaging generally stabilizes training and contributes more positively to MinMax's performance than to ELBE's, particularly when using KL loss. This might be because it averages the errors in the q-function estimates for unseen states and ensures that updates made for previous minibatches are considered equally in the new update (fig. D.10 & D.11). However, it may occasionally hurt ELBE's performance (fig. C.10). Note that ELBE updates the critic on all minibatches within each epoch, whereas MinMax updates only one minibatch per loop, which might inherently stabilize training in ELBE.

Sampler Parametrization (based on the results in Appendix C-H)

We investigate the impact of parametrizing the sampler distribution with a neural network on training stability. This study aims to determine whether this approach contributes to more stable training dynamics.

Interpretation: We have observed that a parameterized sampler can stabilize the training of the policy in some environments, but it is not a crucial element to ensure performance (fig. C.12 & D.4). In the specific case of CartPole, using WNLL loss benefits from a parameterized sampler, whereas using KL loss does not (fig. D.5). In the network architecture experiments, we observed that although MinMax did not achieve as good performance as ELBE, using a parameterized sampler was beneficial. This suggests that a well-tuned sampler can aid in stabilizing training (fig. G.14). When using a parameterized sampler, it is generally better not to average critics (fig. C.11).

4.1.1 Discussion

In our experiments, we explored various aspects of the Q-REPS algorithm's optimization process across different environments. We consistently observed that ELBE generally outperformed MinMax across all tested environments. While tuning other hyperparameters such as network architecture or implementing a learning rate linear decay contributed to performance and stability in specific cases, they did not consistently lead to substantial performance gains across all environments. Our findings underscore the importance of environment-specific tuning of key parameters. Policy loss comparisons indicated that KL loss is generally superior for ELBE, while WNLL loss is more effective for MinMax due to their inherent different training behaviors. Notably, when using GAE, WNLL showed better performance in both cases. Overall, regularization coefficients, learning

rates, update epochs, and advantage estimators demonstrated significant impacts on algorithm performance. To achieve optimal performance in Q-REPS algorithm implementations, careful tuning of these parameters is essential.

Performance Comparison

We compared Q-REPS with other state-of-the-art algorithms using their implementations from [Huang et al., 2021]. The training curves show the average episodic return across three different random seeds. Our observations indicate that, when appropriately tuned for each environment, Q-REPS either outperforms or performs comparably to the other algorithms. This demonstrates Q-REPS' effectiveness in solving MDPs, even with neural network approximations. We believe this represents a promising initial result for LP-based Deep RL algorithms, demonstrating their potential to achieve competitive performance on several environments. Detailed hyperparameter settings for each environment are provided in Appendix B.

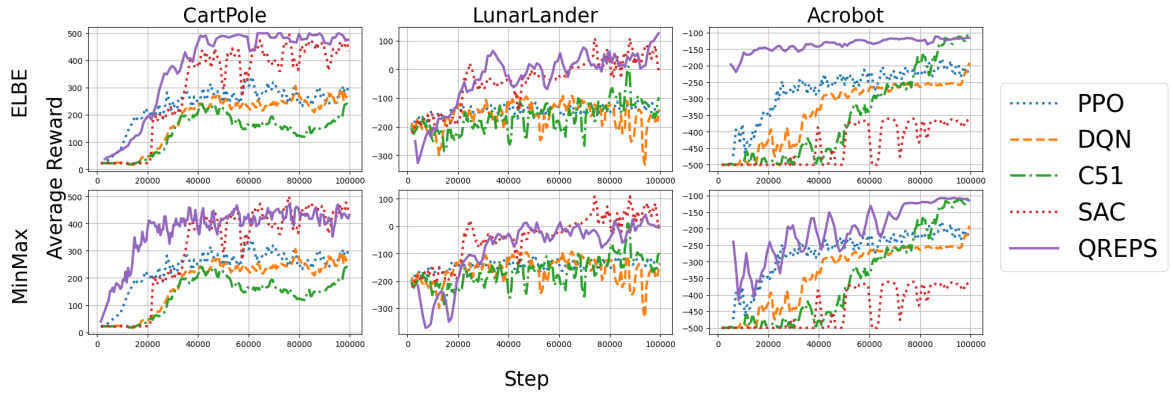


Figure 4.1: Training curves.

4.2 Primal-Dual Approximate Policy Iteration Experiments

Data collection and optimization loop

We consider an on-policy reinforcement learning setting, focusing initially on a tabular framework. Since this is a new algorithm, even though very similar to Q-REPS, our priority is to implement and validate it in smaller, more manageable environment before scaling it up. By starting with a tabular setting, we can compare it with Q-REPS or other alternatives with ease. We collect a batch of N transition samples, independent from each other. We then run several steps of stochastic gradient descent to learn both z and the q-function with the corresponding batch of transition samples. Finally, we reset z and repeat this procedure for the next step.

Tabular primal-dual approximate policy-iteration

We work with the FrozenLake-v1 environment to directly implement Algorithm 7. To compute the gradient of Equation 3.19 in a tabular setting, we calculate the partial derivatives of $J(Q, z)$ with respect to $Q(s, a)$:

$$\frac{\partial J(Q,z)}{\partial Q(s,a)} = \sum_n z(n) \frac{\partial \hat{\delta}_\pi(s_n, a_n, s'_n)}{\partial Q(s,a)} + \sum_{s,y,a} (1-\gamma)\nu_0(s) \frac{\partial \mathcal{M}_\pi Q(s)}{\partial Q(y,a)}. \tag{4.3}$$

We have:

$$\frac{\partial \mathcal{M}_\pi Q(s)}{\partial Q(y,a)} = \mathbb{I}_{\{s=y\}} \sum_a \pi(a|s) = \mathbb{I}_{\{s=y\}} \tag{4.4}$$

$$\frac{\partial \hat{\delta}_\pi(s_n, a_n, s'_n)}{\partial Q(s,a)} = \gamma \sum_{y,a} \frac{\partial \mathcal{M}_\pi Q(s'_n)}{\partial Q(y,a)} - 1 \tag{4.5}$$

To compute an unbiased gradient of Q from a batch of sample transitions (s_n, a_n, s'_n), we perform the following procedure: sample s_0 from the initial state distribution ν_0 and $a_0 \sim \pi(\cdot|s_0)$. Then sample an action $a' \sim \pi(\cdot|s'_n)$. We construct the stochastic gradient for the batch by sampling $n \sim z$, and use the corresponding sample transition for the update:

$$\hat{g}_Q(s,a) = \begin{cases} -1 & \text{if } (s,a) = (s_n, a_n) \\ \gamma & \text{if } (s,a) = (s'_n, a') \\ 1-\gamma & \text{if } (s,a) = (s_0, a_0) \end{cases} \tag{4.6}$$

Algorithm 7 Tabular Primal-Dual API Algorithm

Initialize π_0 and Q_0;
for $t = 0, 1, 2, \ldots, T-1$ **do**
  Run π_t and collect sample transitions $\{(s_n, a_n, r_n, s'_n)\}_{n=1}^{N}$
  Initialize $z_{k,0} = \mathbb{1}_N$
  for $k = 0, 1, 2, \ldots, K$ **do**
    $V_{k+1,t} = \mathcal{M} Q_{k,t}$
    $\delta_{k+1,t}(n) = r_\pi(s_n, a_n) + \gamma V_{k+1,t}(s'_n) - Q_{k,t}(s_n, a_n)$
    $Q(s,a) \leftarrow Q(s,a) - \rho \hat{g}_Q(s,a)$
    $z_{k+1,t}(n) = z_{k,t}(n) e^{\beta'(\delta_{k+1,t}(n) - \lambda \ln(z))}$
  end for
  $Q_{t+1} = \frac{1}{K} \sum_k Q_{k,t}$
  $\pi_{t+1} \propto \pi_t$ softmax $(\alpha Q_{t+1}(s,a))$
end for

Benchmarking

We evaluate the performance of our proposed primal-dual API (PD-API) algorithm in comparison to the canonical primal-dual approach (Algorithm 8) and MinMax Q-REPS in a tabular context. The canonical primal-dual approach works with a value function and a tabular occupancy measure μ, while both MinMax Q-REPS and our proposed algorithm employ a sampler variable z and collect a batch of N samples, necessitating multiple updates at each step. Conversely, the canonical approach performs a single update on μ and the value function using only one sample per step.

Our implementations of the PD-API and Q-REPS are nearly identical, with the only difference being the normalization of the z distribution and the corresponding computation of the value function in Q-REPS. Thus, comparing Q-REPS to PD-API is similar to comparing the performance of the different proposed value functions within each algorithm. We refer to the following optimization procedure as the canonical primal-dual approach:

$$\begin{aligned} \underset{\mu \in \mathbb{R}_+^{S \times A}}{\text{maximize}} \quad & \langle \mu, r \rangle \\ \text{subject to} \quad & E^T \mu = (1-\gamma)\nu_0 + \gamma P^T \mu \end{aligned}$$

$$\mathcal{L}(v, \mu) = \langle \mu, r \rangle + \langle v, (1-\gamma)\nu_0 + \gamma P^\intercal \mu - E^\intercal \mu \rangle.$$

$$\nabla_\mu \mathcal{L}(v, \mu) = r + \gamma P v - E v$$

$$\nabla_v \mathcal{L}(v, \mu)_x = \mathbb{E}[\mathbb{I}_{\{S_0 = s\}} + \gamma \mathbb{I}_{\{S' = s\}} - \mathbb{I}_{\{S = s\}}]$$

Algorithm 8 Canonical Primal Dual

Initialize π, μ, V, $\mu' = \mu$,
for $t = 1$ to T **do**
  Sample a random pair of state and action (s_t, a_t)
  $\mu_{t+1} \leftarrow \mu_t e^{\beta \nabla_\mu \mathcal{L}(v,\mu)}$
  Sample a state from the marginal $s \sim \mu(s)$, and an action $a \sim \mu(s, \cdot)$
  $V_{t+1} \leftarrow V_t - \rho \nabla_v \mathcal{L}(v, \mu)_s$
  $\mu' = \mu' + \mu_{t+1}$
  $\pi_t(a|s) = \frac{\mu'(s,a)}{\sum_{a'} \mu'(s,a')}$
end for
return π_T

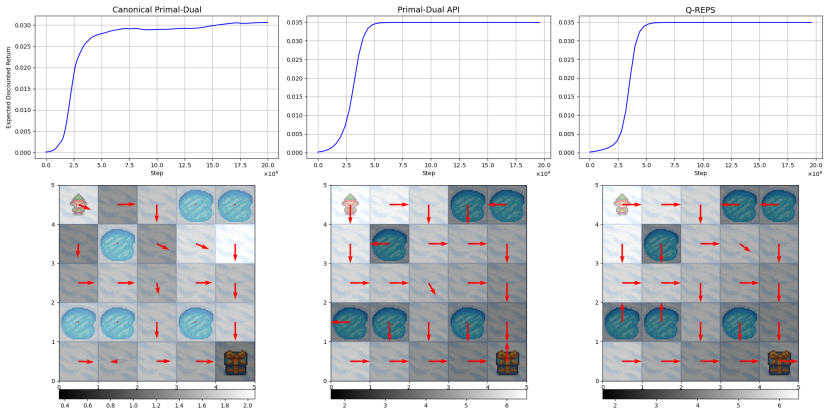
4.2.1 Discussion

We examine the policies, value functions, and convergence rates of Q-REPS and the PD-API algorithm across 5x5 and 8x8 grid environments using 200,000 sample transitions. Both Q-REPS and PD-API perform well in both tabular and large-scale settings, contrasting with the canonical primal-dual method, which lacks scalability due to its dependency on the occupancy measure μ. In tabular scenarios, the canonical primal-dual method generally demonstrates faster convergence, likely because of its direct access to μ, which facilitates optimal policy construction. However, the final optimal policies found by Q-REPS and PD-API seem to be slightly better. Despite this advantage, Q-REPS and PD-API perform comparably well without direct access to μ, with Q-REPS exhibiting slightly faster convergence, even though we have observed that PD-API usually has more stable training, probably because the value function is not computed via a logsumexp. Q-REPS and the PD-API present promising alternatives due to their broader applicability and adaptability to larger environments, overcoming the limitations of the canonical primal-dual approach.

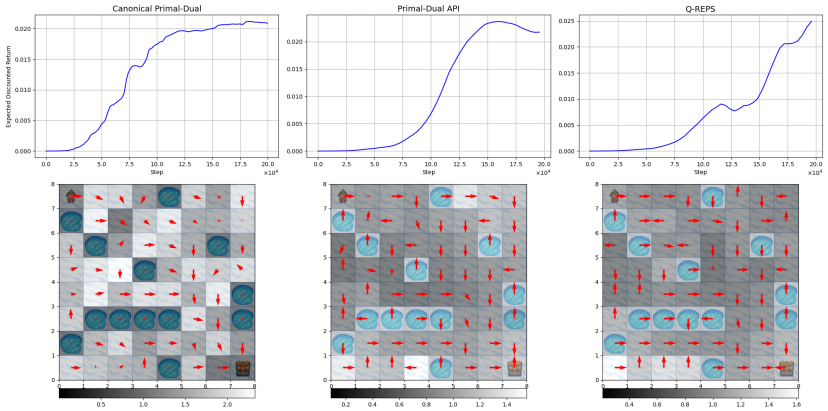
Both of the compared algorithms process batches of N samples, achieving convergence with fewer policy updates compared to the canonical primal-dual method for the same number of sample transitions. Despite having fewer updates, their convergence rates remain similar. These two sampler-based algorithms share many components, resulting in similar solutions in most cases (fig. 4.2).

In less complex grids with straightforward paths and fewer obstacles, PD-API often matches Q-REPS in performance of the generated policies policies and value function estimation. Conversely, Q-REPS shows significant advantages in more complicated settings with complex layouts, where we have found PD-API to struggle with in some cases (fig. I.29). This better generalization is likely

due to Q-REPS's entropy-regularized value function, which might be facilitating a more effective exploration of the state space, preventing over-commitment on observed samples. Therefore, Q-REPS would be a preferred choice in large scale settings, for its superior exploration and convergence capabilities across different environment complexites.



(a) Results on 5x5 grid



(b) Results on 8x8 grid

Figure 4.2: Policies and training curves from different tabular algorithms. Each cell color represents the value assigned to the corresponding state by the algorithm, as indicated by the color bar. Hyperparameters used can be found in Appendix I

We conducted further analysis on the optimal properties of the sampler-based algorithms (all results detailed in Appendix I) within a 5x5 grid, by running a large scale empirical study of different algorithm design choices. This included considerations such as whether to sample from the initial state distribution $s_0 \sim \nu_0$ or from the state distribution associated with the value function $s_0 \sim \frac{V(s)}{\sum_{s'} V(s')}$. We also explored various approaches to initializing the Q-function: setting it to zero $Q_0(s,a) = 0$, distributing it uniformly $Q_0(s,a) = \frac{1}{|S||A|}$, or setting it to its maximal value $Q_0(s,a) = \frac{1}{1-\gamma}$ to potentially enhance exploration, especially when sampling from the state distribution associated to the value function. Additionally, we explored different configurations for batch size, learning rates, and update epochs.

Our observations indicate that larger batch sizes and more frequent updates consistently yield better performance in both Q-REPS and PD-API (fig. I.3 & I.6), albeit with longer training times, as anticipated. There was no discernible performance difference observed between sampling from ν_0 and $s_0 \sim \frac{V(s)}{\sum_{s'} V(s')}$ (fig. I.21,). Initializing Q_0 either with a uniform distribution or to zero generally produced favorable outcomes (fig. I.18).

In Q-REPS, larger updates appear to enhance training efficacy, while the PD-API approach benefited from smaller policy updates. Moreover, policy updates - controlled by the temperature parameter- were three times larger in Q-REPS compared to the PD-API variant (I.15).We also noted that the regularization coefficient applied in Q-REPS tends to be smaller than that used in PD-API, significantly influencing stability (I.12).

Chapter 5

CONCLUSIONS

We have shown that LP-based algorithms have practical applications and can be effectively implemented on a large scale with DNNs, much like other RL algorithms. Our results indicate that the performance of these methods can be comparable to the current state of the art. There is potential for further improvements to enhance numerical stability. LP offers a promising alternative for developing large-scale RL algorithms. The flexibility of this approach allows for various modifications to the regularization functions, value functions, and other components, enabling the creation of diverse algorithms. We hope this work lays the groundwork for implementing these algorithms with DNNs, with future research aimed at improving their performance and stability.

The dual algorithm we proposed can be viewed as a modification of Q-REPS, where the Q-value function is not regularized. While this may improve numerical stability, it does not explore as effectively as Q-REPS, especially in more complex settings.

There is significant potential for further research both within the general LP framework and specifically within the Q-REPS approach presented here. Future work could involve tuning these algorithms for more complex environments, such as Atari games, and analyzing their performance. Continuous environments could also be explored using this framework by introducing Gumbel regression [Garg et al., 2023] to directly find the entropy-regularized value function, instead of computing the closed-form solution of the logsumexp as in Q-REPS. This would allow for the actor and value function to be parameterized for continuous actions, making the algorithm suitable for such scenarios. Further analysis and rigorous development of this idea would be necessary to understand whether this would be feasible or not.

In conclusion, our research highlights the potential of LP-based algorithms in large-scale reinforcement learning (RL) applications. We have studied the al-

gorithmic development and improvement of Q-REPS. While our proposed modifications are promising, significant work remains to enhance their stability and performance capabilities and to adapt them to more complex and continuous environments. Future efforts could focus on rigorous experimentation and theoretical advancements of these LP algorithms in continuous control. We hope this will inspire further development of LP-based reinforcement learning algorithms for larger settings, providing alternatives to the current methods used in RL algorithm development.

Bibliography

[Abbasi-Yadkori et al., 2019] Abbasi-Yadkori, Y., Bartlett, P., Bhatia, K., Lazic, N., Szepesvari, C., and Weisz, G. (2019). POLITEX: Regret bounds for policy iteration using expert prediction. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3692–3702. PMLR.

[Agarwal et al., 2021] Agarwal, A., Jiang, N., Kakade, S. M., and Sun, W. (2021). *Reinforcement Learning: Theory and Algorithms*.

[Agarwal et al., 2020] Agarwal, A., Kakade, S. M., Lee, J. D., and Mahajan, G. (2020). On the theory of policy gradient methods: Optimality, approximation, and distribution shift.

[Andrychowicz et al., 2020] Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., Gelly, S., and Bachem, O. (2020). What matters in on-policy reinforcement learning? a large-scale empirical study.

[Barto et al., 1983] Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846.

[Bas-Serrano et al., 2021] Bas-Serrano, J., Curi, S., Krause, A., and Neu, G. (2021). Logistic q-learning.

[Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

[Engstrom et al., 2020] Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. (2020). Implementation matters in deep rl: A case study on ppo and trpo. In *International Conference on Learning Representations*.

[Garg et al., 2023] Garg, D., Hejna, J., Geist, M., and Ermon, S. (2023). Extreme q-learning: Maxent rl without entropy.

[Geist et al., 2017] Geist, M., Piot, B., and Pietquin, O. (2017). Is the bellman residual a bad proxy?

[Geist et al., 2019] Geist, M., Scherrer, B., and Pietquin, O. (2019). A theory of regularized markov decision processes.

[Haarnoja et al., 2018] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

[Henderson et al., 2018] Henderson, P., Romoff, J., and Pineau, J. (2018). Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods.

[Huang et al., 2021] Huang, S., Dossa, R. F. J., Ye, C., and Braga, J. (2021). Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms.

[Kakade, 2001] Kakade, S. M. (2001). A natural policy gradient. In Dietterich, T., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press.

[Kegl, 2023] Kegl, B. (2023). A systematic study comparing hyperparameter optimization engines on tabular data.

[Kingma and Ba, 2017] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

[Liaw et al., 2018] Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., and Stoica, I. (2018). Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*.

[Liu et al., 2020] Liu, J., Liu, S., and Gu, X. (2020). Soft q network.

[Manne, 1960] Manne, A. S. (1960). Linear programming and sequential decisions.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

[Neu et al., 2017] Neu, G., Jonsson, A., and Gómez, V. (2017). A unified view of entropy-regularized markov decision processes.

[Peters et al., 2010] Peters, J., Mulling, K., and Altun, Y. (2010). Relative entropy policy search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 24(1):1607–1612.

[Puterman, 2014] Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[Ruder, 2017] Ruder, S. (2017). An overview of gradient descent optimization algorithms.

[Schulman et al., 2017a] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2017a). Trust region policy optimization.

[Schulman et al., 2018] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2018). High-dimensional continuous control using generalized advantage estimation.

[Schulman et al., 2017b] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b). Proximal policy optimization algorithms.

[Sebastian Markgraf, 2022] Sebastian Markgraf, Philipp Becker, O. C. M. H. (2022). Validating logistic q-learning.

[Silver et al., 2017] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm.

[Sutton, 1995] Sutton, R. S. (1995). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In Touretzky, D., Mozer, M., and Hasselmo, M., editors, *Advances in Neural Information Processing Systems*, volume 8. MIT Press.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Appendix A

DEFAULT SETTINGS

Table A.1: Default settings used in experiments I.

Choice	Name	Default value
C1	num_envs (E)	16
C2	iteration_size (N)	1024
C3	gamma (γ)	0.99
C4	update_epochs (T)	50
C5	num_minibatches (n)	16
C6	policy_lr (τ)	3e-4
C7	q_lr (ρ)	3e-4
C8	beta (β)	3e-4
C9	alpha (α)	2.0
C10	eta (η)	2.0
C11	policy_activation	Tanh
C12	policy_hidden_size	128
C13	policy_num_hidden_layers	2
C14	actor_last_layer_std	0.01
C15	q_activation	Tanh
C16	q_hidden_size	128
C17	q_num_hidden_layers	4
C18	q_last_layer_std	1.0
C19	sampler_activation	Tanh
C20	sampler_hidden_size	128
C21	sampler_num_hidden_layers	2
C22	q_optimizer (O)	Adam
C23	actor_optimizer (O')	Adam
C24	sampler_optimizer (SO)	Adam

Table A.2: Default settings used in experiments II.

Choice	Name	Default value
C25	`eps` (ϵ)	1e-8
C26	`MinMax_optimization`	False
C27	`layer_init`	He uniform
C28	`anneal_lr`	False
C29	`parametrized_sampler` (P)	False
C30	`policy_loss` (PL)	KL
C31	`average_critics` (AC)	True
C32	`normalize_delta` (ND)	False
C33	`advantadge_estimator` (AE)	1-step

Appendix B

HYPERPARAMETERS USED FOR THE EXPERIMENTS

While for PPO, C51 and SAC, hyperparameters were kept as the default parameters in cleanrl's implementation, and they are implementations that already include code level optimizations.

The parameters used for Q-REPS in the ELBE and in the MinMax procedures are defined in the tables below. Parameters that were left as default (Appendix A) are omitted in this tables.

Table B.1: Parameters for QREPS in ELBE experiments.

Hyperparameter	**CartPole-v1**	**LunarLander-v2**	**Acrobot-v1**
α	32	2	12
η	0.5	2	12
policy_lr (τ)	0.0003	0.003	0.001
q_lr (ρ)	0.0003	0.0003	0.001
anneal_lr (AL)	✓	✓	✓
average_critics (AC)	✓	✓	✓
policy_loss	KL	KL	WNLL

Table B.2: Parameters for QREPS in MinMax experiments. (MinMax=True)

Hyperparameter	**CartPole-v1**	**LunarLander-v2**	**Acrobot-v1**
iteration_size (N)	512	1024	1024
num_minibatches (n)	4	16	16
update_epochs (T)	300	50	50
α	4	1	12
η	4	1	12
β	0.003	0.003	3e-5
policy_lr (τ)	3e-5	3e-5	3e-4
q_lr (ρ)	0.00025	0.001	3e-3
q_optimizer (O)	RMSprop	Adam	Adam
policy_optimizer (O')	SGD	Adam	Adam
anneal_lr (AL)	✓	✗	✓
average_critics (AC)	✓	✓	✓
policy_loss	KL	KL	KL
parametrize_sampler (P)	✗	✓	✓
layer_init	-	Xavier normal	Xavier normal
policy_hidden_size	-	64	256
policy_num_hidden_layers	-	2	2
policy_activation	-	ReLU	-
actor_last_layer_std	-	-	0.01
q_num_hidden_layers	-	1	8
q_hidden_size	-	64	128
q_last_layer_std	-	0.01	20.1
sampler_hidden_size	-	256	256
sampler_num_hidden_layers	-	4	8
sampler_activation	-	Tanh	ReLU

Appendix C

TRAINING SETUP

C.1 Design

For each of the 3 environments, we sampled 2000 choice configurations where we sampled the following choices independently and uniformly from the following ranges:

- `iteration_size`: {256, 512, 1024, 2048}
- `update_epochs`: {10, 25, 50, 100, 150}
- `num_minibatches`: {4, 8, 16, 32, 64}
- `q_lr`: {3e-05, 0.0001, 0.00025, 0.0003, 0.001}
- `policy_lr`: {3e-05, 0.0001, 0.00025, 0.0003, 0.001}
- α: {0.5, 1, 2, 4, 8, 12, 32, 64, 100}
- `MinMax_optimization`: {True, False}
  - For the case `MinMax`: True,
    - `beta` (sampler learning rate): {3e-05, 0.0001, 0.0003, 0.001}
    - `parametrized_sampler`: {True, False}
- `average_critics`: {True, False}
- `policy_optimizer`: {SGD, Adam, RMsprop}
- `q_optimizer`: {SGD, Adam, RMsprop}
- `anneal_lr`: {True, False}

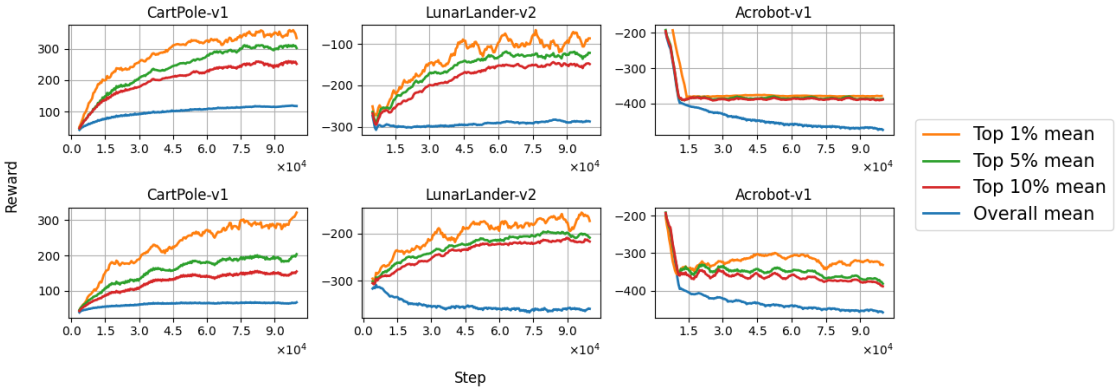


Figure C.1: Training curves. ELBE procedure on the first row, MinMax on the second row.

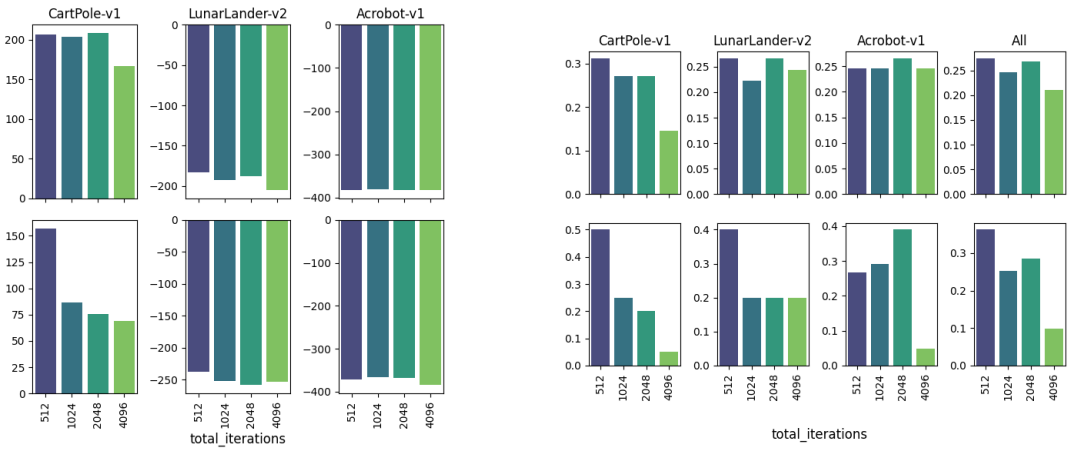


Figure C.2: Analysis of choice `iteration_size`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

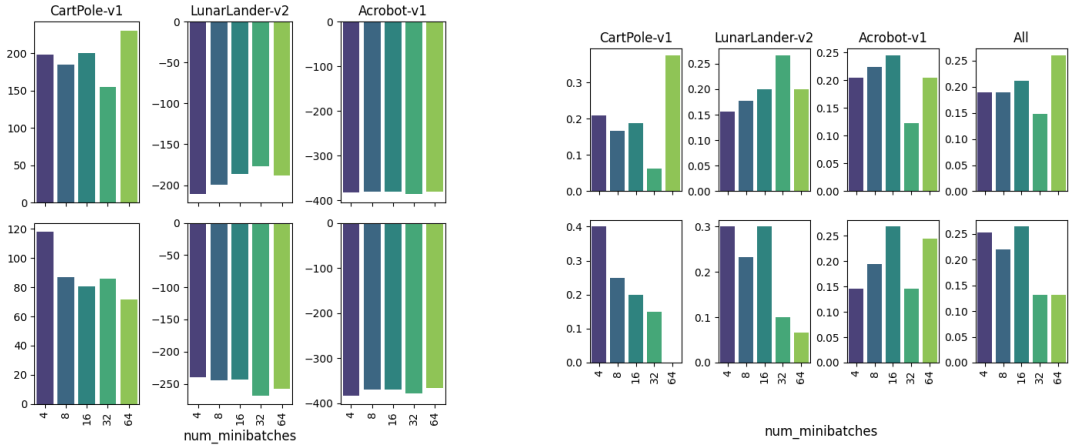


Figure C.3: Analysis of choice `num_minibatches`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

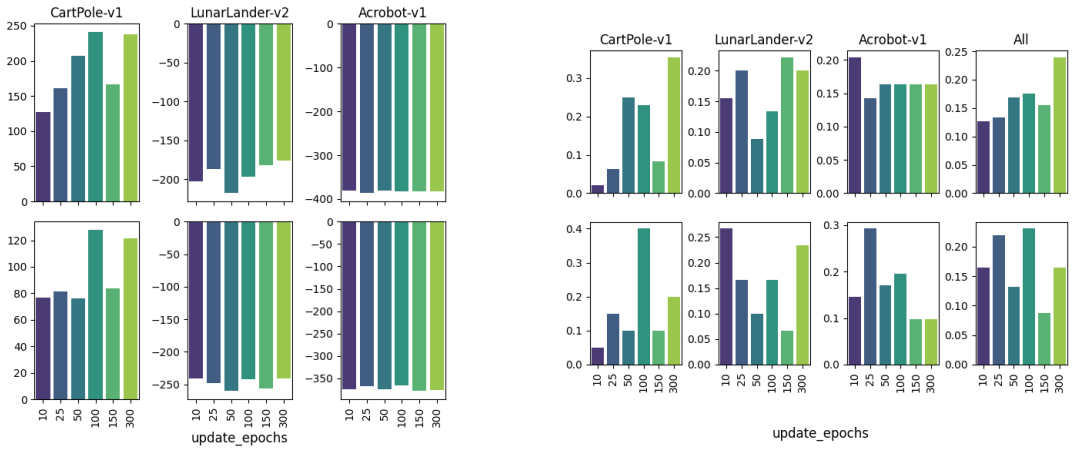


Figure C.4: Analysis of choice `update_epochs`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

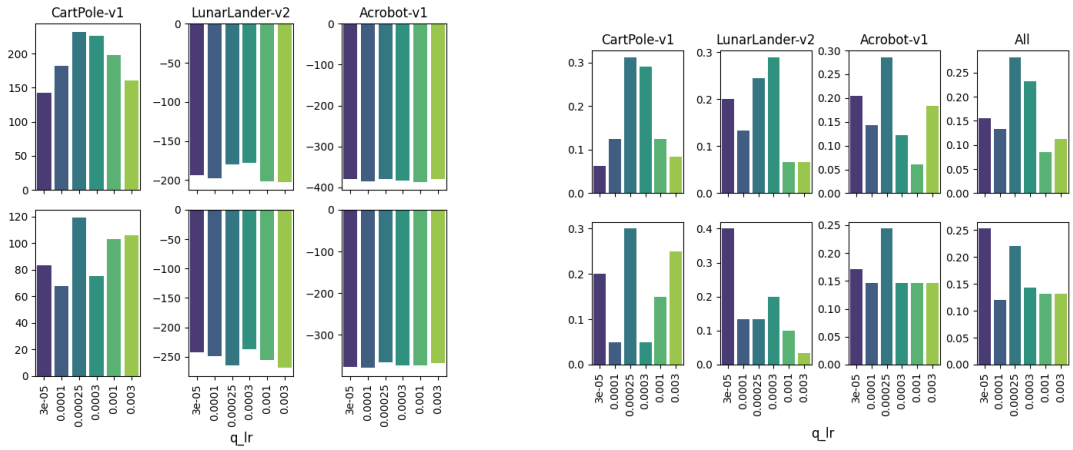


Figure C.5: Analysis of choice `q_lr`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

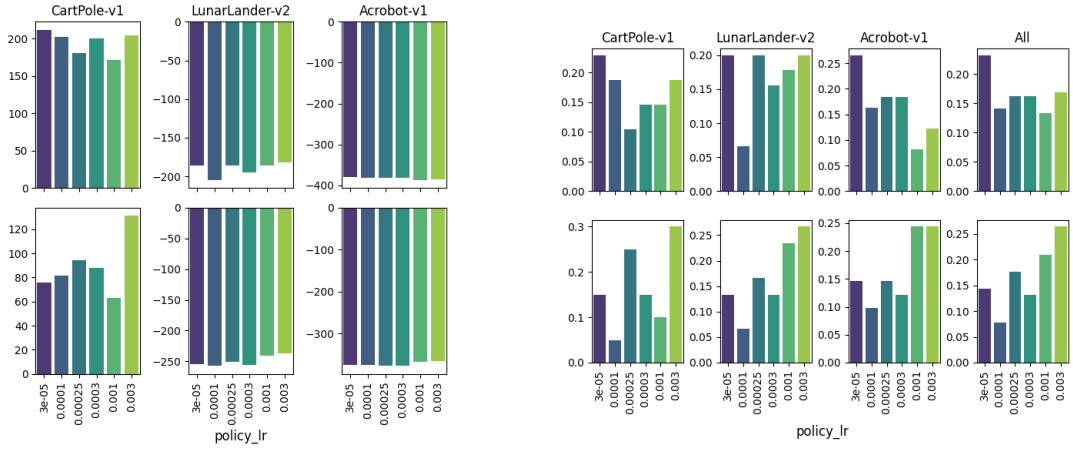


Figure C.6: Analysis of choice `policy_lr`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

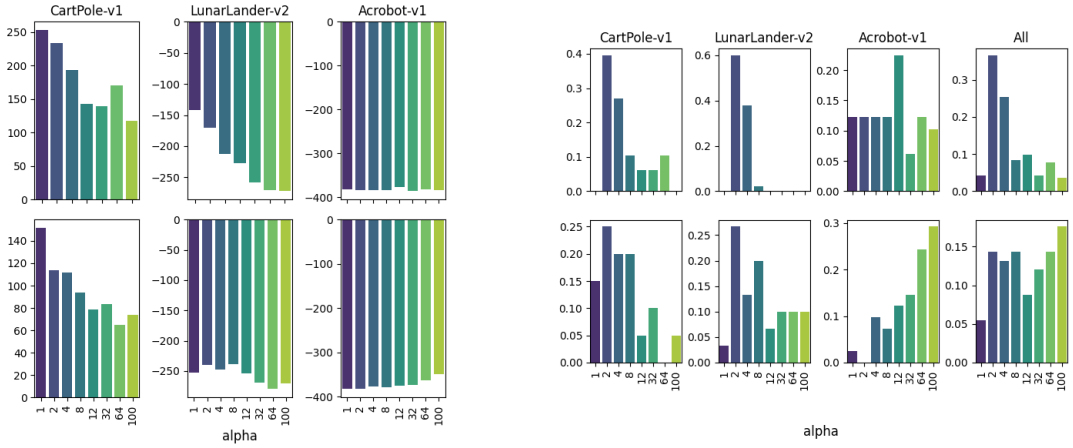


Figure C.7: Analysis of choice `alpha`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

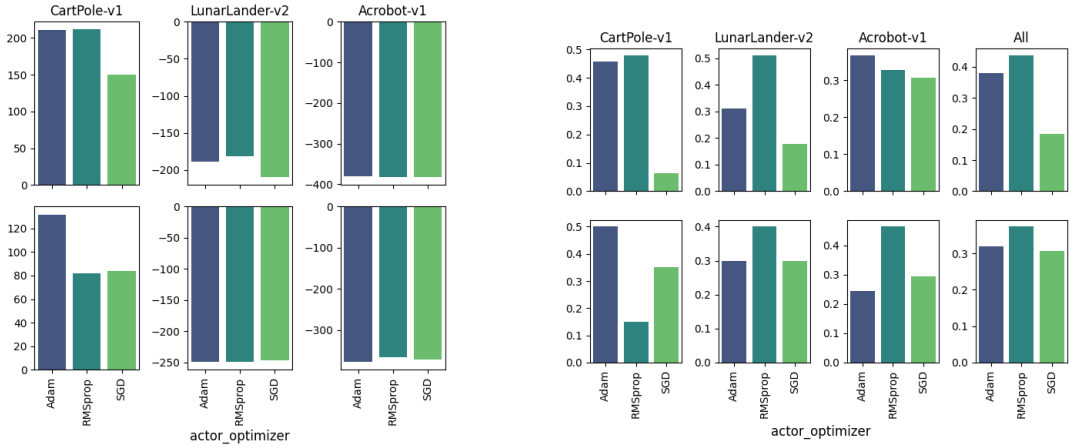


Figure C.8: Analysis of choice `actor_optimizer`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

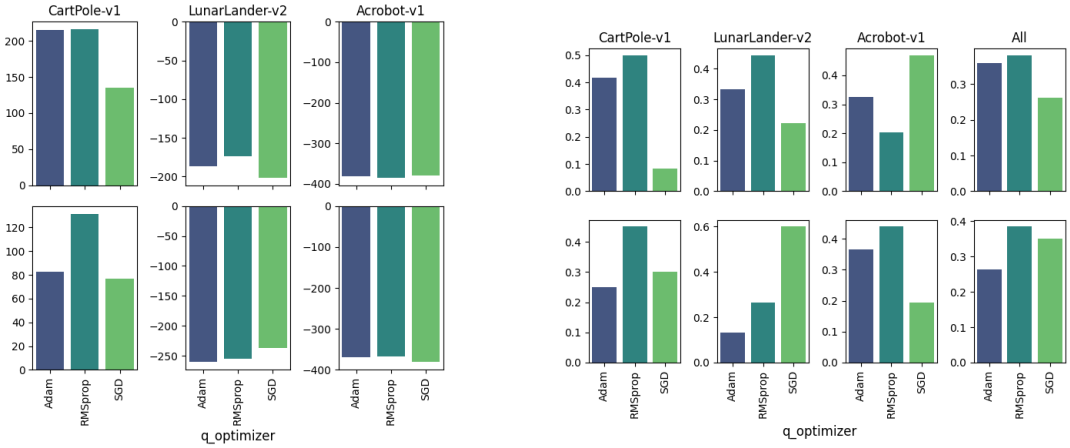


Figure C.9: Analysis of choice `q_optimizer`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

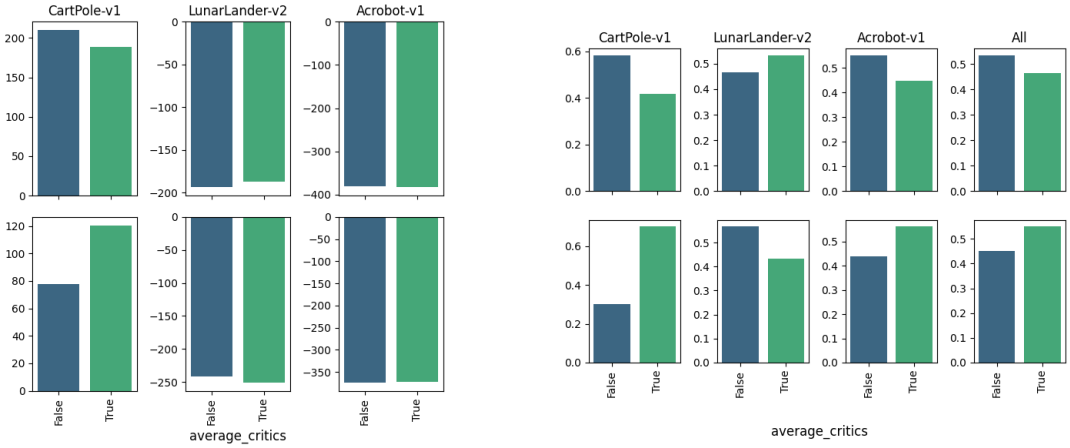


Figure C.10: Analysis of choice `average_critics`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

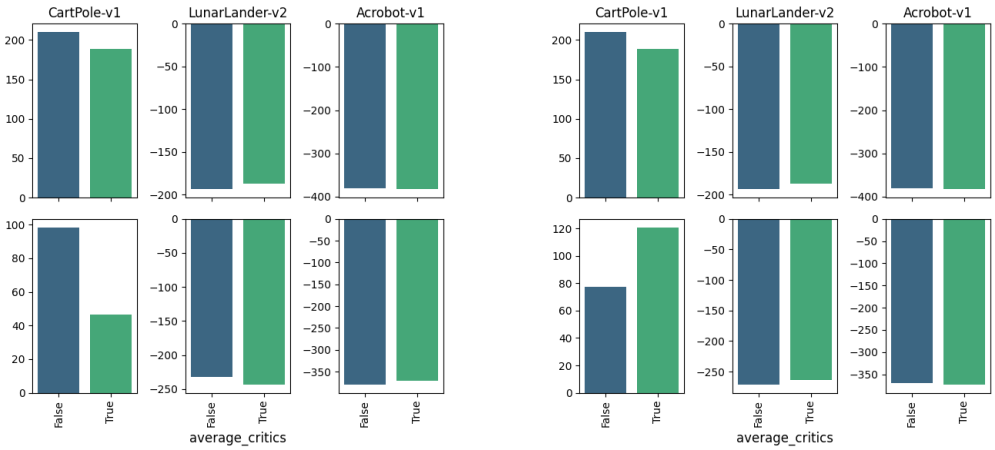


Figure C.11: Analysis of choice `average_critics`: 95th percentile of performance scores conditioned on `parametrized_sampler`=True (left), 95th percentile of performance scores conditioned on `parametrized_sampler`=False (right). ELBE procedure on the first row, MinMax on the second row.

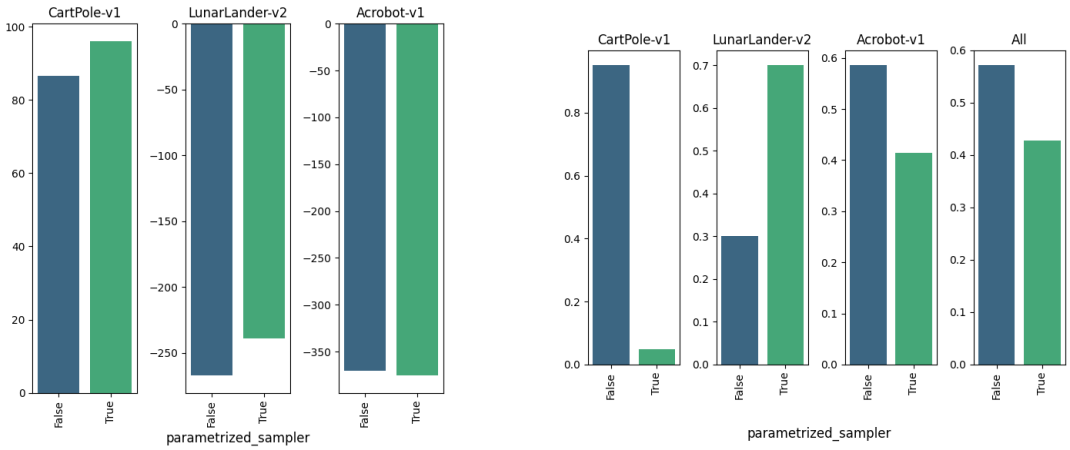


Figure C.12: Analysis of choice `parametrized_sampler`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

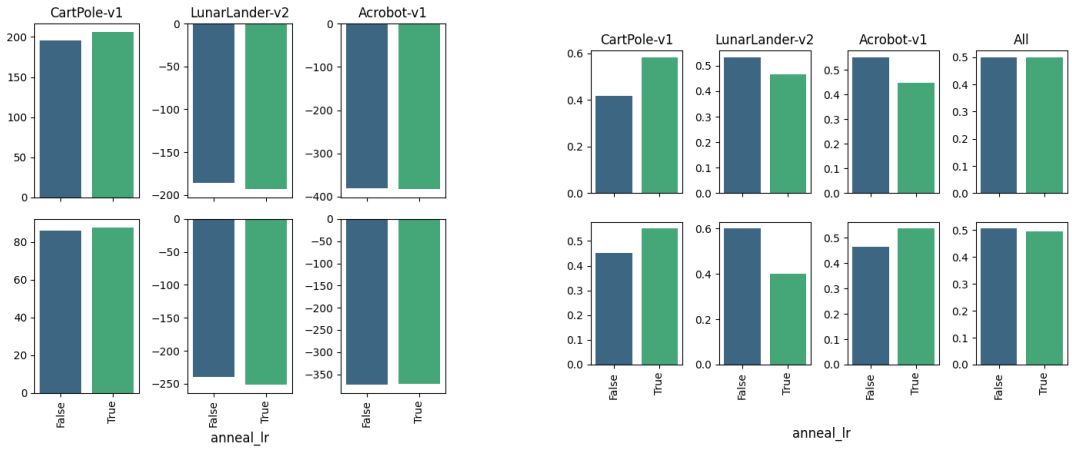


Figure C.13: Analysis of choice `anneal_lr`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

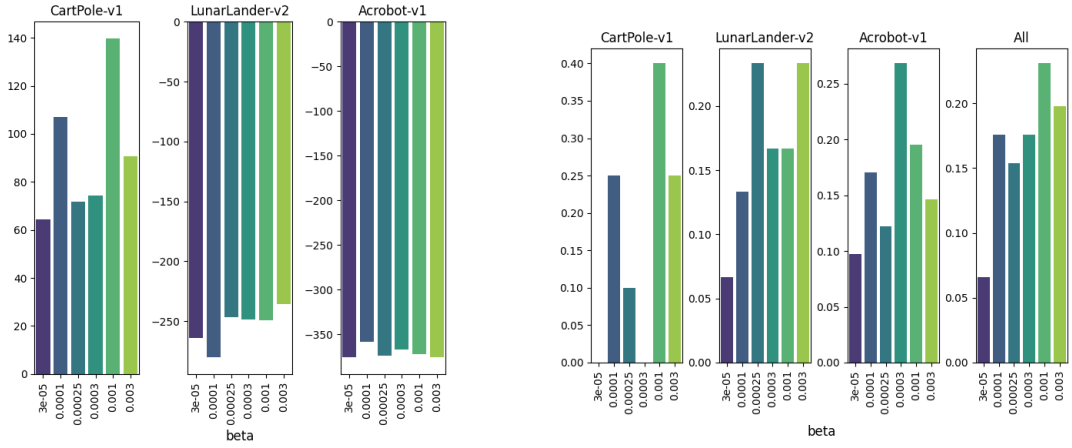


Figure C.14: Analysis of choice `beta`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

Appendix D

POLICY LOSSES

D.1 Design

For each of the 3 environments, we sampled 2000 choice configurations where we sampled the following choices independently and uniformly from the following ranges:

- `policy_loss`: {KL, WNLL}
- `policy_lr`: {3e-05, 0.0001, 0.00025, 0.0003, 0.001}
- `q_lr`: {3e-05, 0.0001, 0.00025, 0.0003, 0.001}
- α: {0.5, 1, 2, 4, 8, 12, 32, 64, 100}
- `MinMax_optimization`: {True, False}
  - For the case `MinMax`: True,
    - `beta` (sampler learning rate): {3e-05, 0.0001, 0.0003, 0.001}
    - `parametrized_sampler`: {True, False}
- `average_critics`: {True, False}
- `anneal_lr`: {True, False}

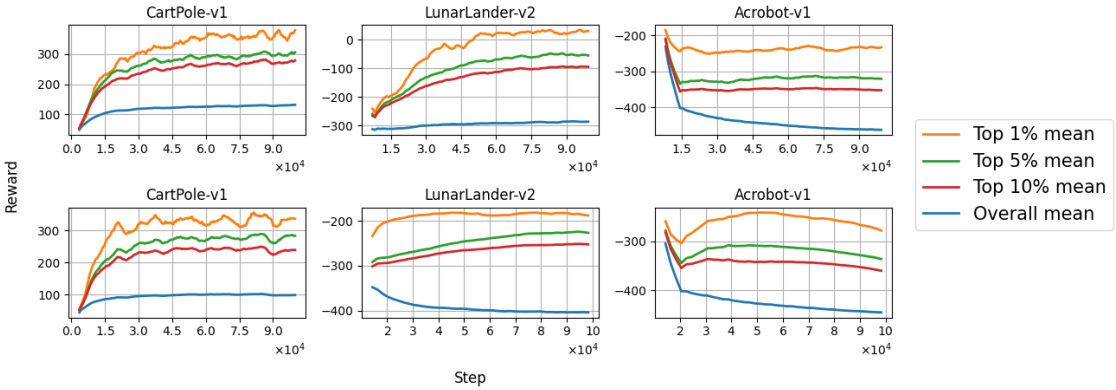


Figure D.1: Training curves. ELBE procedure on the top plots, MinMax on the bottom.

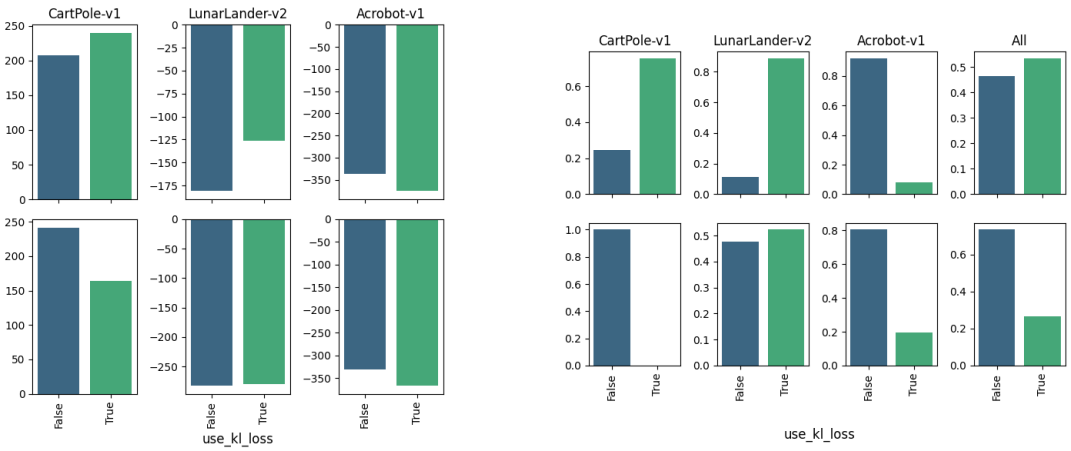


Figure D.2: Analysis of choice `policy_loss`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

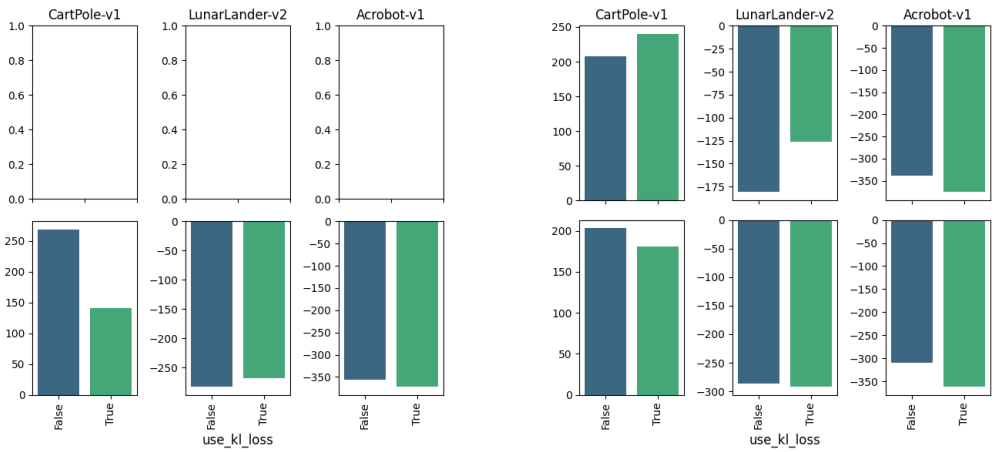


Figure D.3: Analysis of choice `policy_loss`: 95th percentile of performance scores conditioned on `parametrized_sampler`=True (left), 95th percentile of performance scores conditioned on `parametrized_sampler`=False (right). ELBE procedure on the first row, MinMax on the second row.

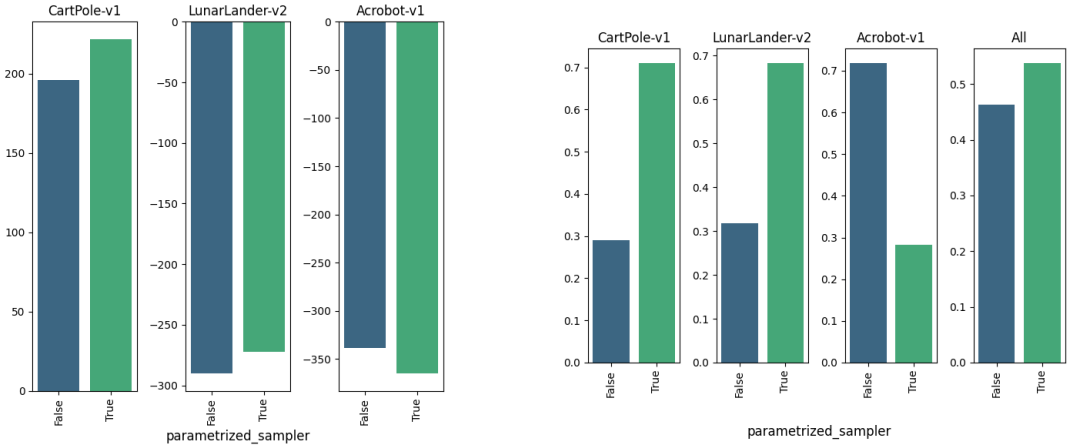


Figure D.4: Analysis of choice `parametrized_sampler`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

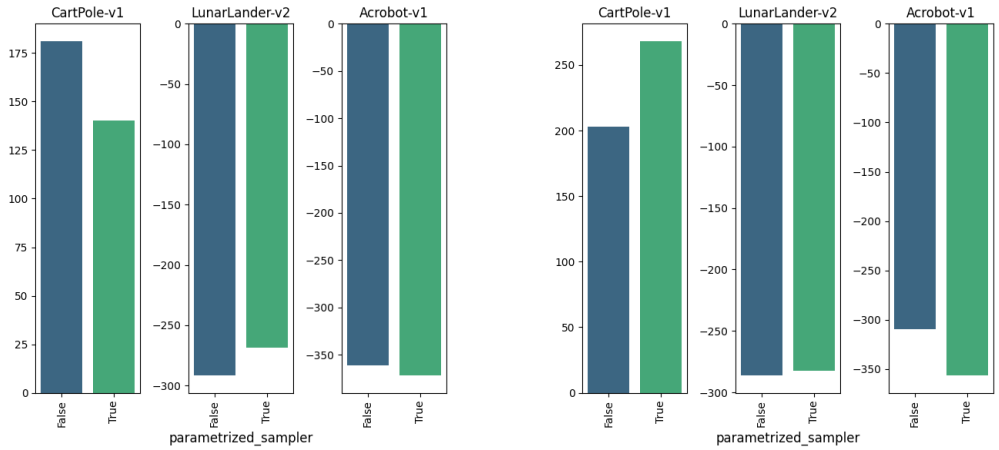


Figure D.5: Analysis of choice `parametrized_sampler`: 95th percentile of performance scores conditioned on `policy_loss`=KL (left), 95th percentile of performance scores conditioned on `policy_loss`=WNLL (right). ELBE procedure on the first row, MinMax on the second row.

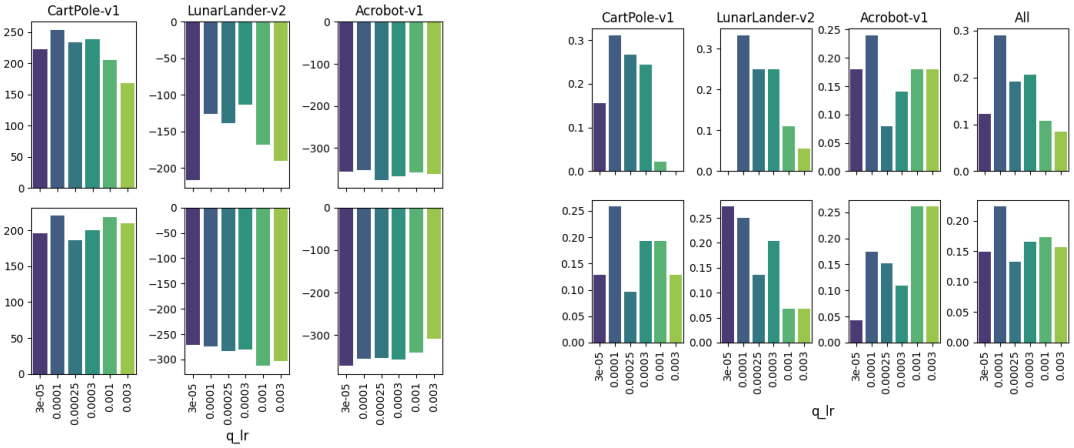


Figure D.6: Analysis of choice `q_lr`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

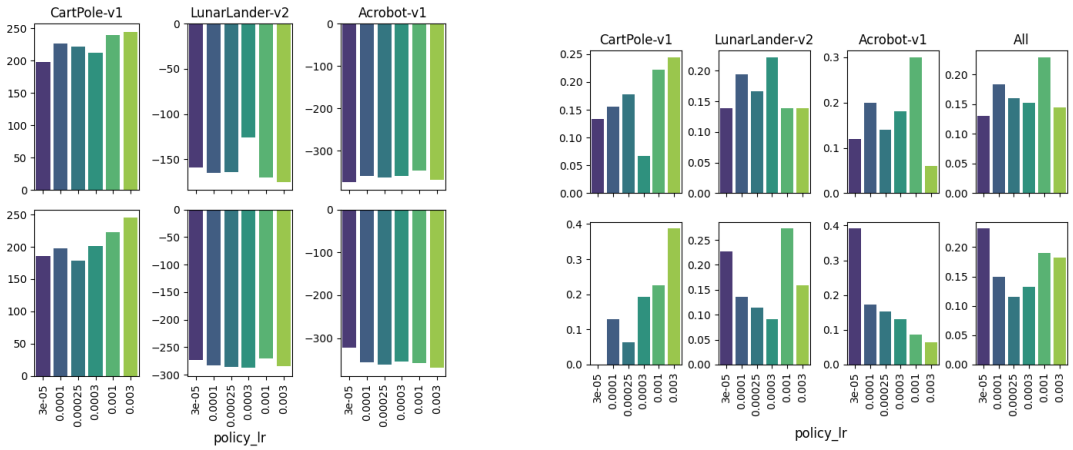


Figure D.7: Analysis of choice `policy_lr`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

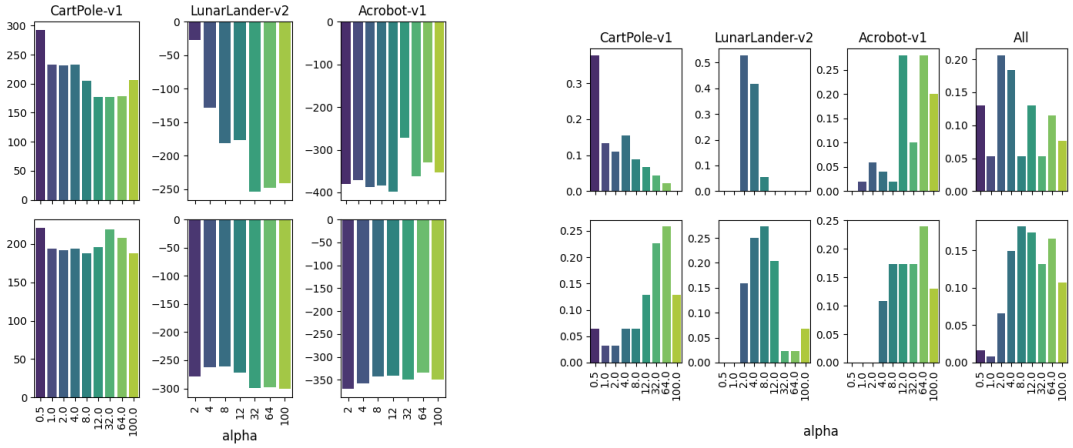


Figure D.8: Analysis of choice `alpha`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

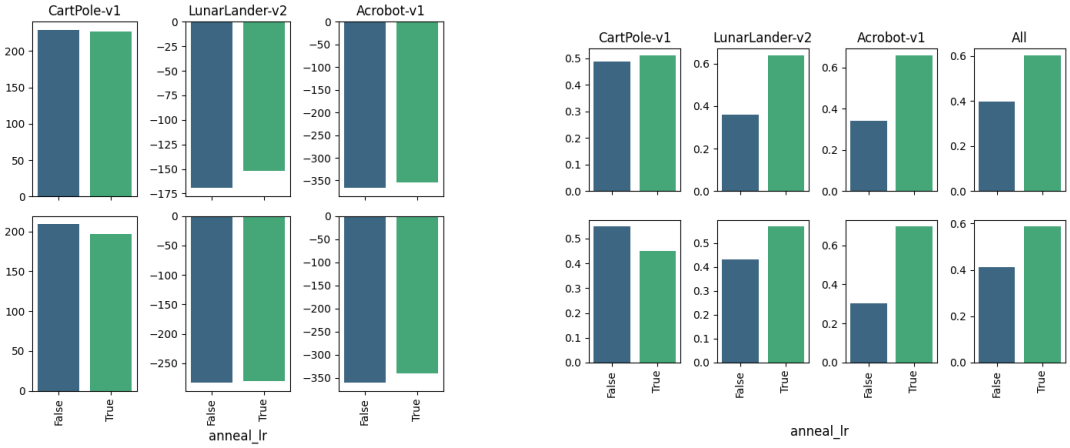


Figure D.9: Analysis of choice `anneal_lr`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

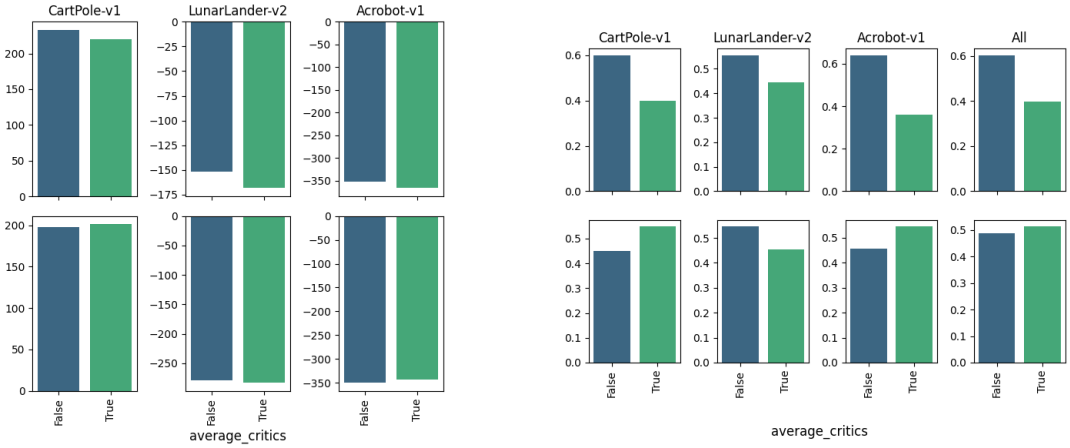


Figure D.10: Analysis of choice `average_critics`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

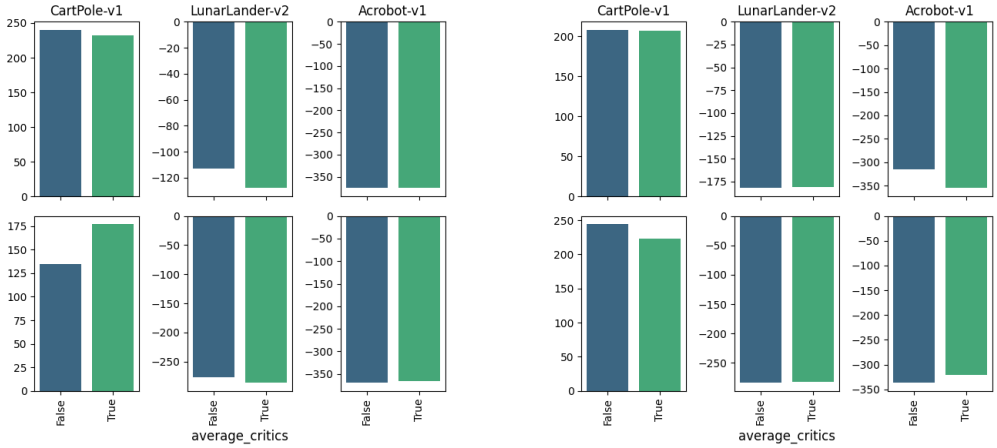


Figure D.11: Analysis of choice `average_critics`: 95th percentile of performance scores conditioned on `policy_loss`=KL (left), 95th percentile of performance scores conditioned on `policy_loss`=WNLL (right). ELBE procedure on the first row, MinMax on the second row.

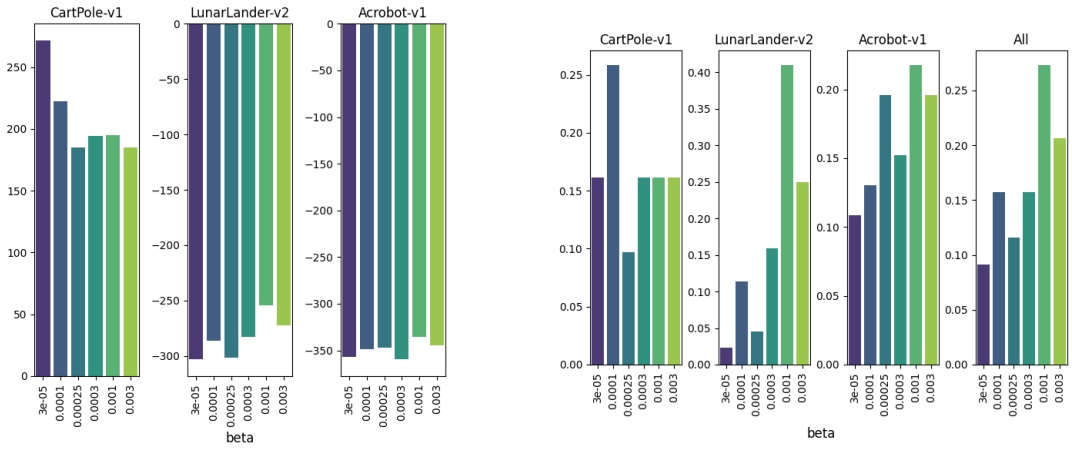


Figure D.12: Analysis of choice `beta`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

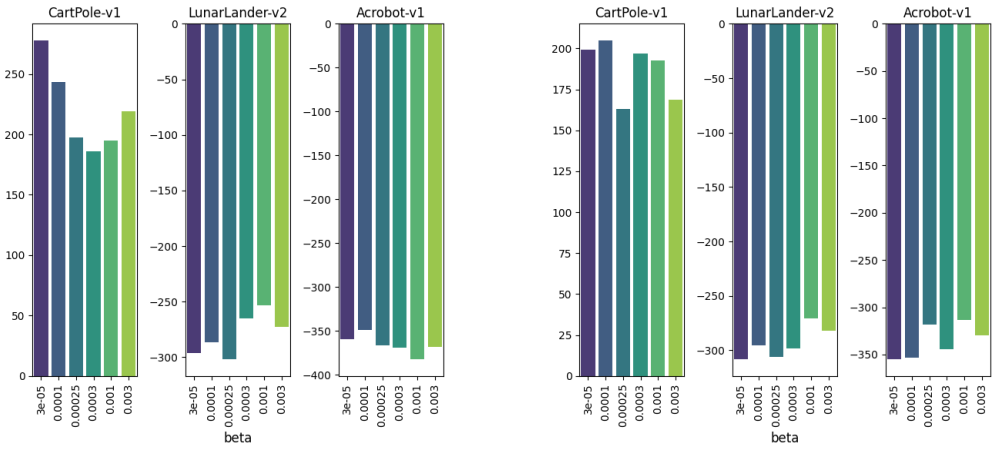


Figure D.13: Analysis of choice `beta`: 95th percentile of performance scores conditioned on `parametrized_sampler`=True (left), 95th percentile of performance scores conditioned on `parametrized_sampler`=False (right). ELBE procedure on the first row, MinMax on the second row.

Appendix E

REGULARIZATION

E.1 Design

For each of the 3 environments, we sampled 2000 choice configurations where we sampled the following choices independently and uniformly from the following ranges:

- α: {0.5, 1, 2, 4, 8, 12, 32, 64, 100}
- η: {0.5, 1, 2, 4, 8, 12, 32, 64, 100}
- `policy_lr`: {3e-05, 0.0001, 0.00025, 0.0003, 0.001}
- `q_lr`: {3e-05, 0.0001, 0.00025, 0.0003, 0.001}
- `MinMax_optimization`: {True, False}
  - For the case `MinMax`: True,
    - `beta` (sampler learning rate): {3e-05, 0.0001, 0.0003, 0.001}
    - `parametrized_sampler`: {True, False}
- `average_critics`: {True, False}
- `anneal_lr`: {True, False}

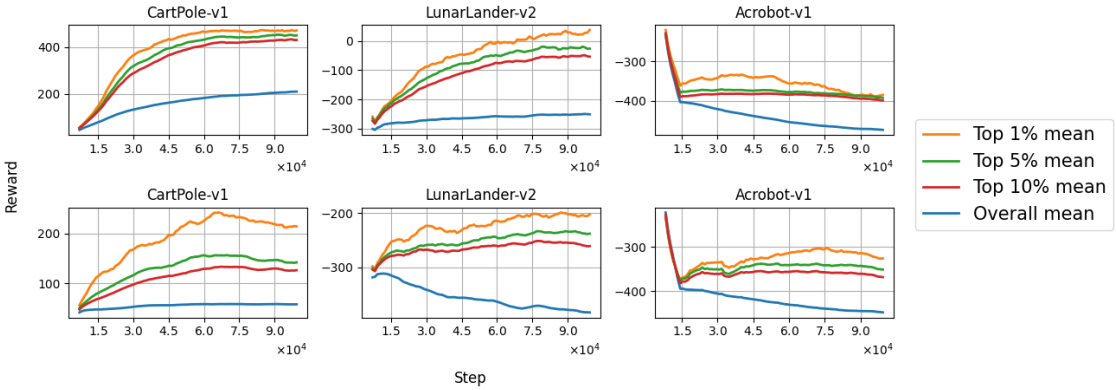


Figure E.1: Training curves. ELBE procedure on the top plots, MinMax on the bottom.

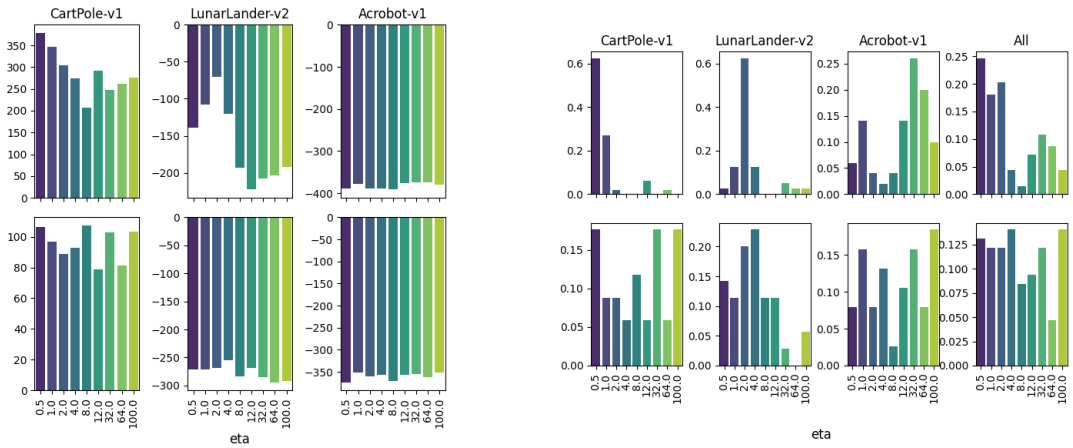


Figure E.2: Analysis of choice `eta`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

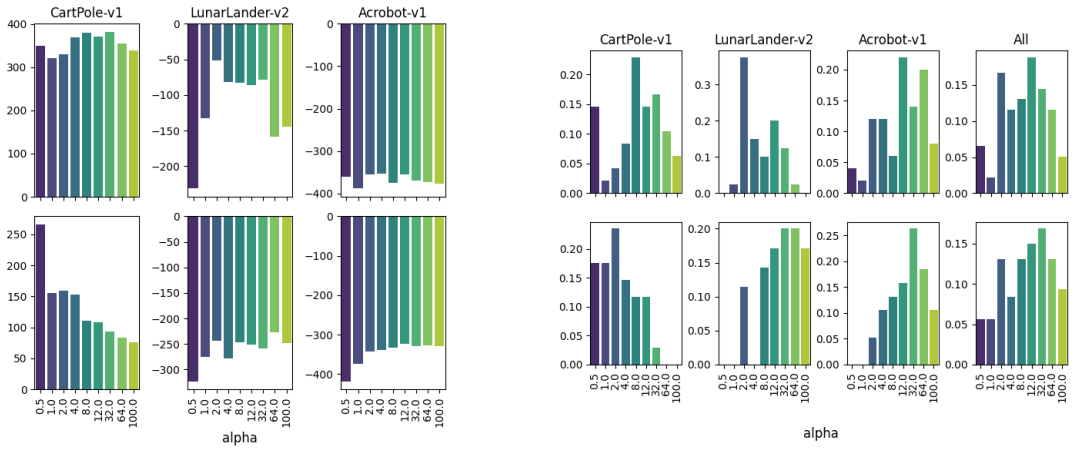


Figure E.3: Analysis of choice `alpha`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

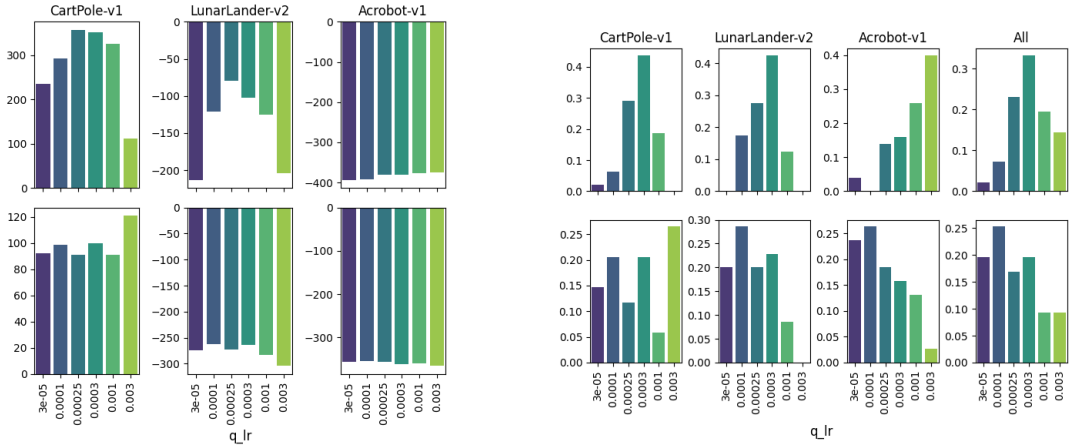


Figure E.4: Analysis of choice `q_lr`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

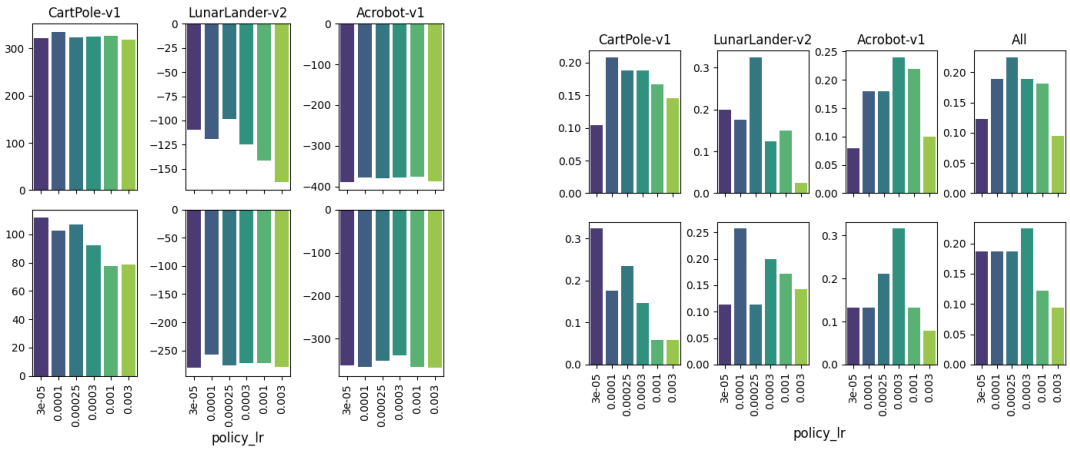


Figure E.5: Analysis of choice `policy_lr`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

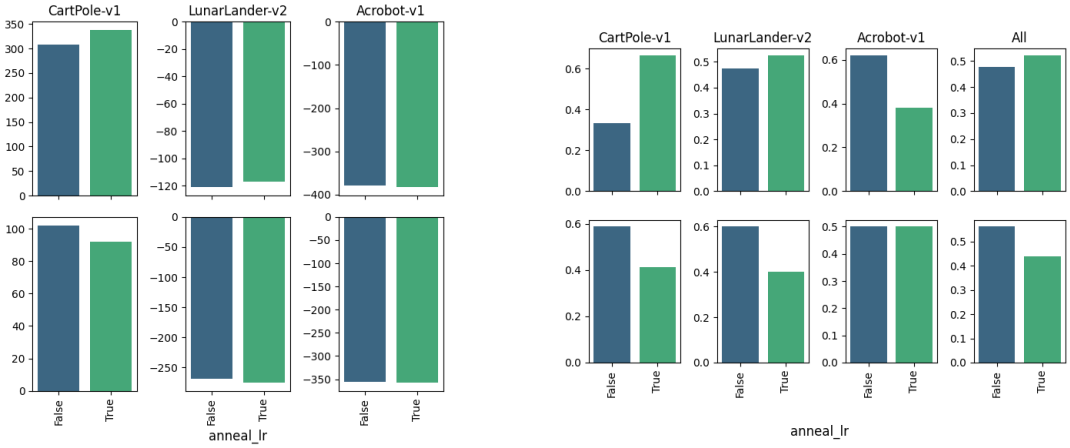


Figure E.6: Analysis of choice `anneal_lr`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

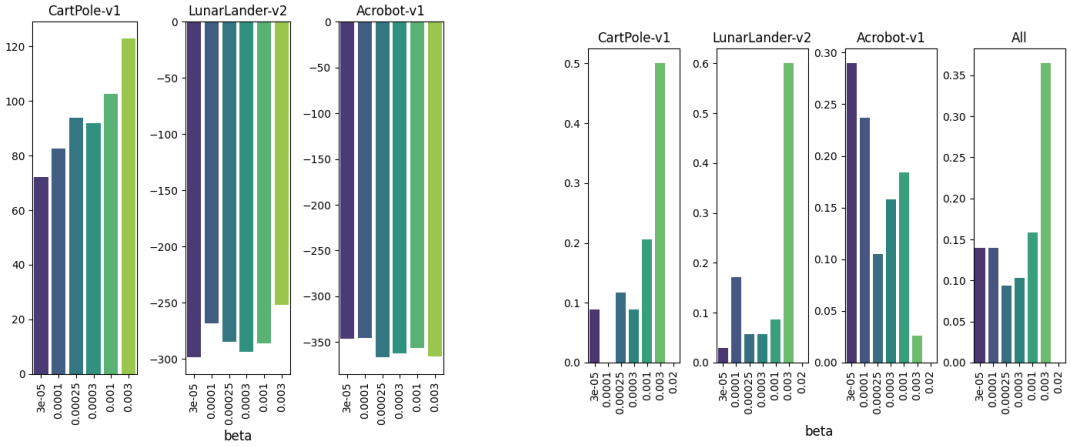


Figure E.7: Analysis of choice `beta`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

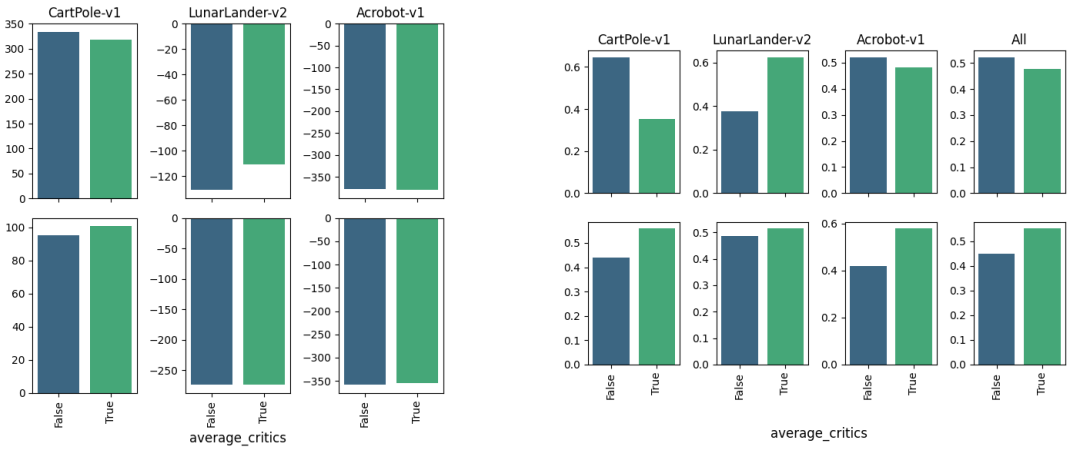


Figure E.8: Analysis of choice `average_critics`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

Appendix F

ADVANTADGE ESTIMATION

F.1 Design

For each of the 3 environments, we sampled 3000 choice configurations where we sampled the following choices independently and uniformly from the following ranges:

- `delta_normalization`: {False, True}
- `advantage_estimator`: {GAE, 1-step}
- `iteration_size`: {256, 512, 1024, 2048}
- `policy_loss`: {KL, WNLL}
- `num_envs`: {4, 8, 16, 32, 64, 128, 256}
- `update_epochs`: {1, 10, 20, 30, 40, 50, 100, 150}
- `target_network`: {True, False}
- `target_network_frequency`: {2, 4, 8, 16, 32, 64}
- `MinMax_optimization`: {False, True}
  - For the case `MinMax_optimization` = True:
    - `beta` (sampler learning rate): {3e-05, 0.0001, 0.0003, 0.001}
    - `parametrized_sampler`: {False, True}
- `policy_lr`: {3e-05, 0.0001, 0.00025, 0.0003, 0.001}
- `q_lr`: {3e-05, 0.0001, 0.00025, 0.0003, 0.001}
- α: {0.5, 1, 2, 4, 8, 12, 32, 64, 100}

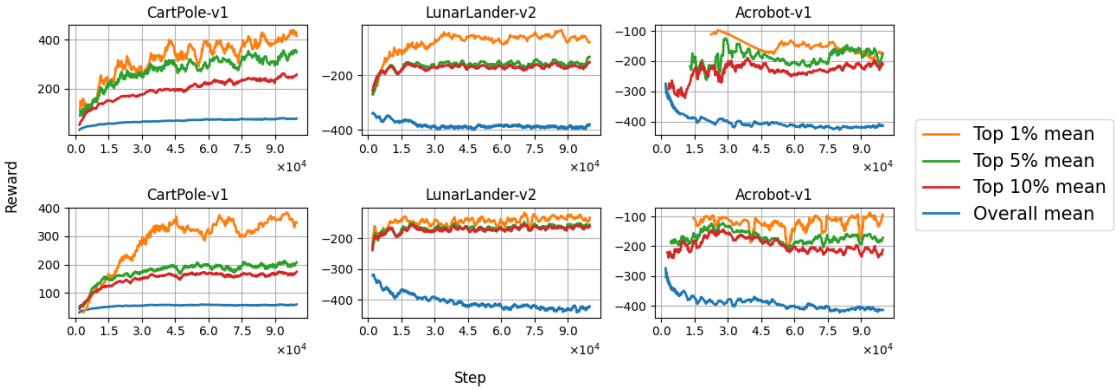


Figure F.1: Training curves. ELBE procedure on the top plots, MinMax on the bottom.

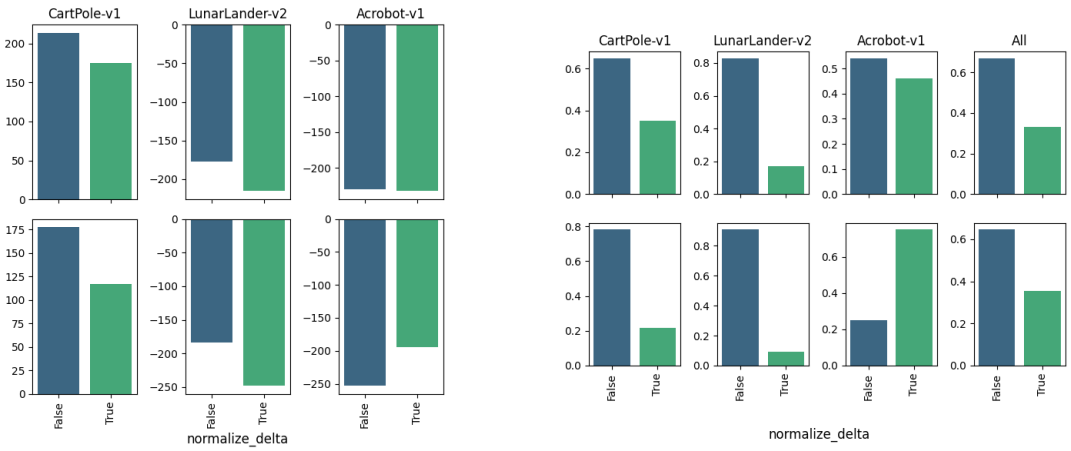


Figure F.2: Analysis of choice `normalize_delta`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

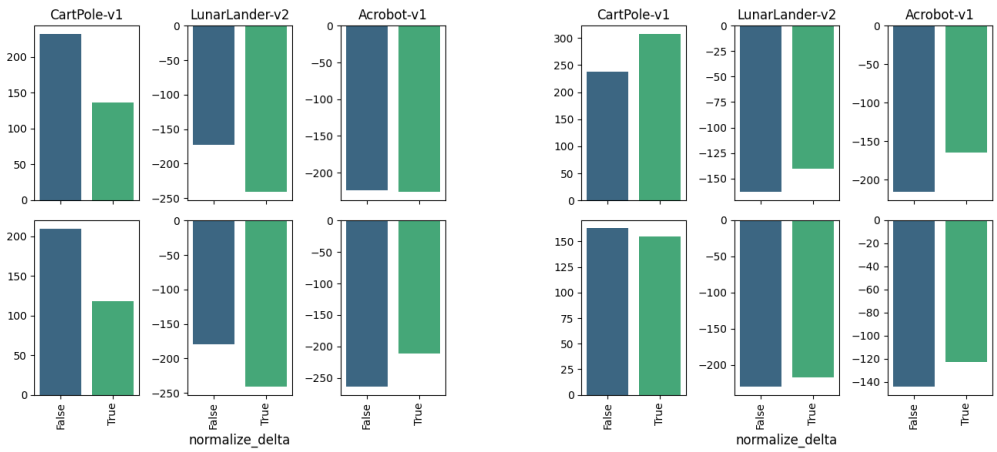


Figure F.3: Analysis of choice `normalize_delta`: 95th percentile of performance scores conditioned on GAE=True (left), 95th percentile of performance scores conditioned on GAE=False (right). ELBE procedure on the first row, MinMax on the second row.

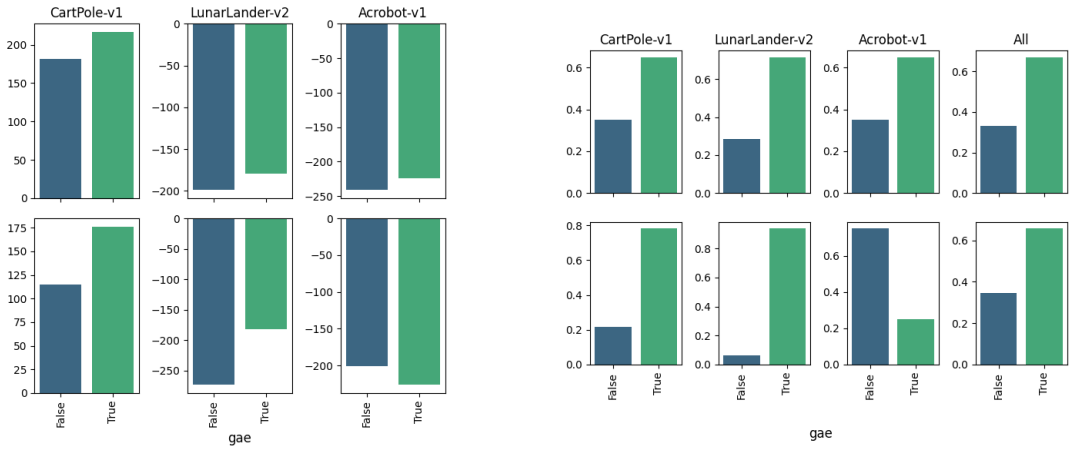


Figure F.4: Analysis of choice `advantadge_estimator`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

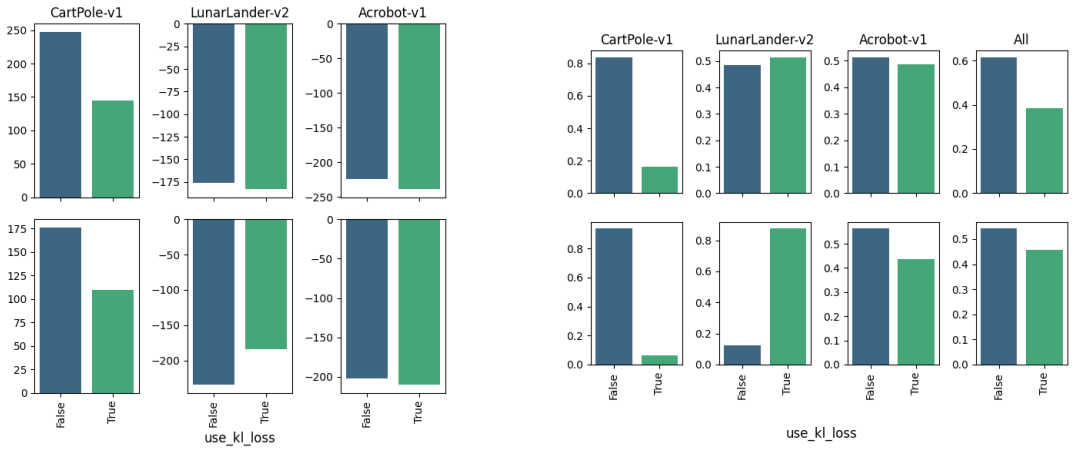


Figure F.5: Analysis of choice `policy_loss`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

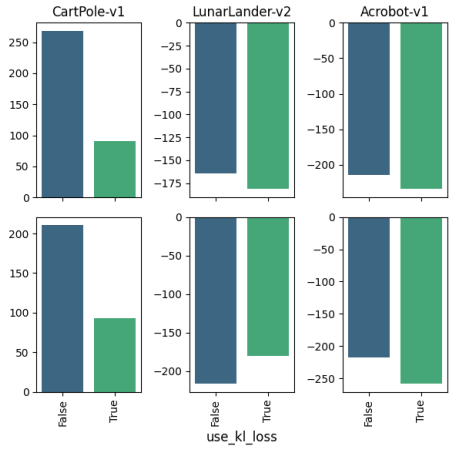


Figure F.6: Analysis of choice `policy_loss`: 95th percentile of performance scores conditioned on GAE=True

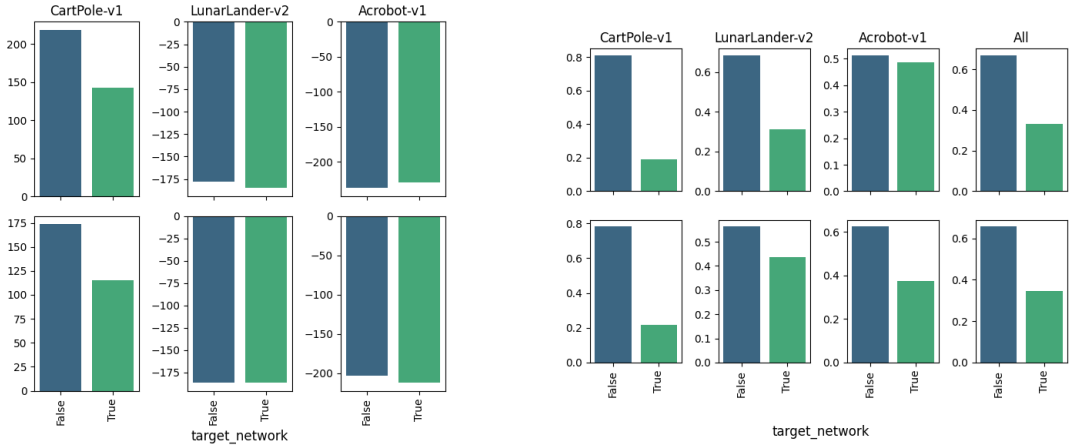


Figure F.7: Analysis of choice `target_network`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

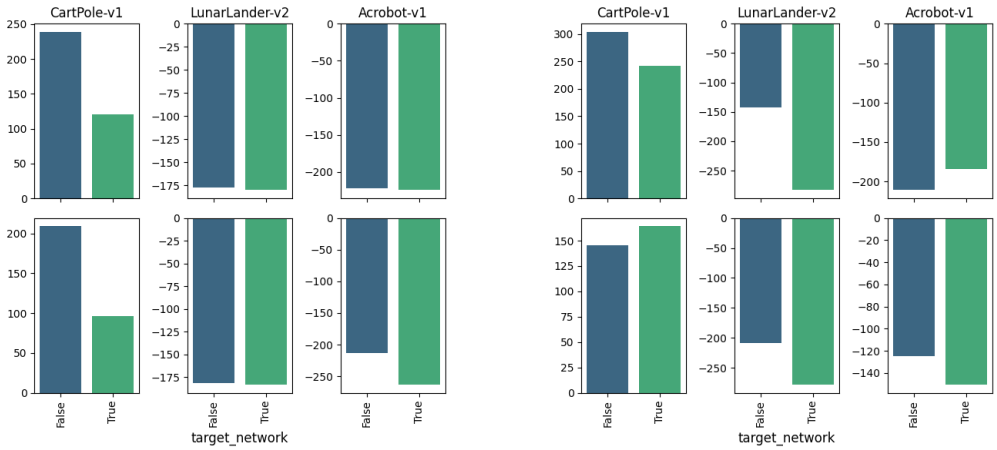


Figure F.8: Analysis of choice `target_network`: 95th percentile of performance scores conditioned on GAE=True (left), 95th percentile of performance scores conditioned on GAE=False (right). ELBE procedure on the first row, MinMax on the second row.

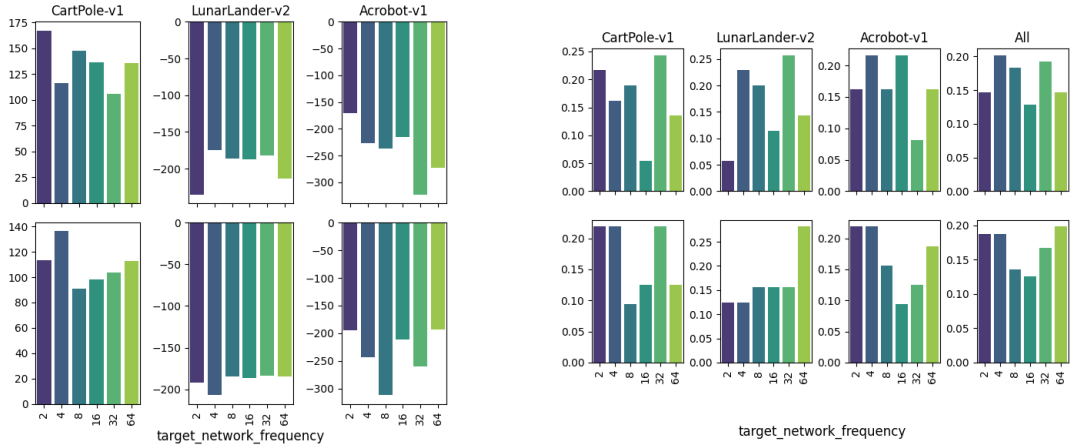


Figure F.9: Analysis of choice `target_network_frequency`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

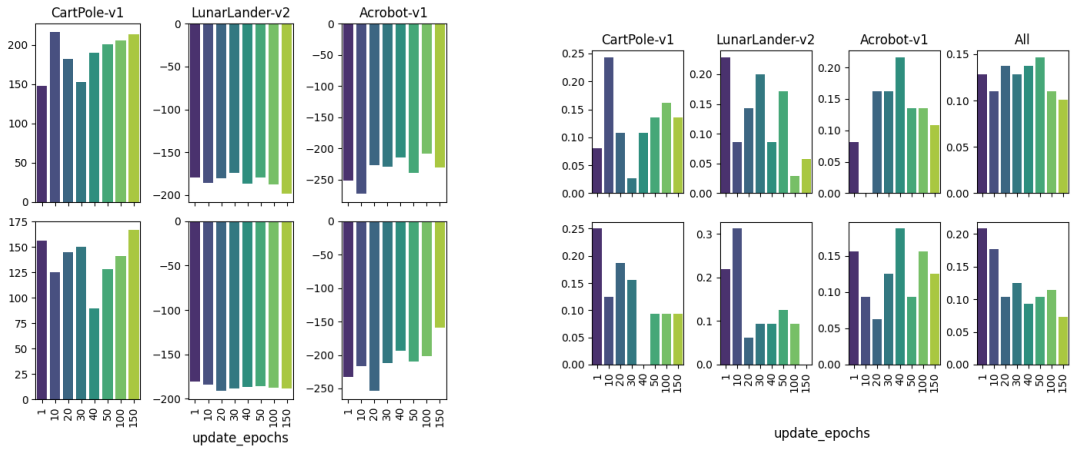


Figure F.10: Analysis of choice `update_epochs`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

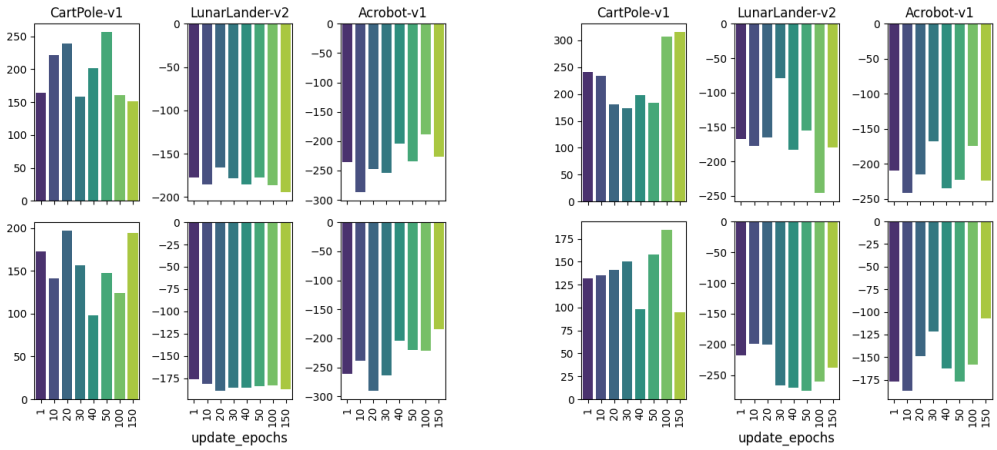


Figure F.11: Analysis of choice `update_epochs`: 95th percentile of performance scores conditioned on GAE=True (left), 95th percentile of performance scores conditioned on GAE=False (right). ELBE procedure on the first row, MinMax on the second row.

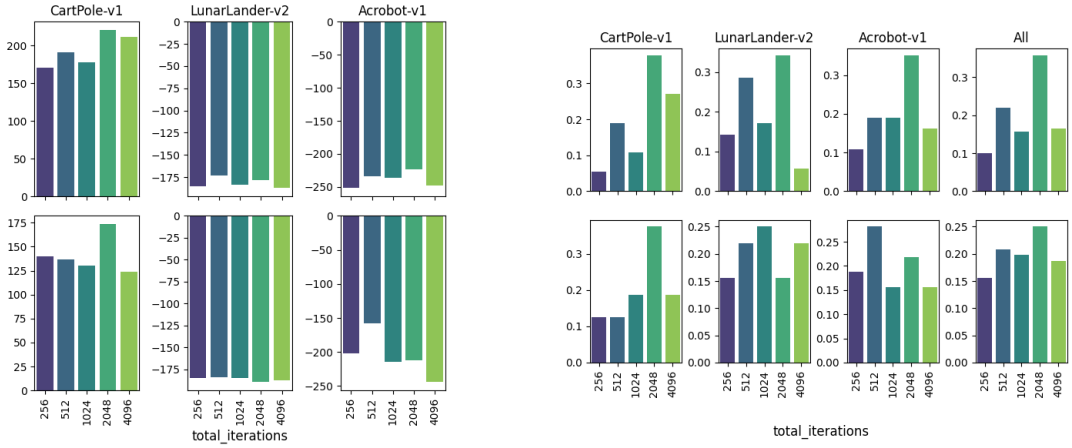


Figure F.12: Analysis of choice `total_iterations`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

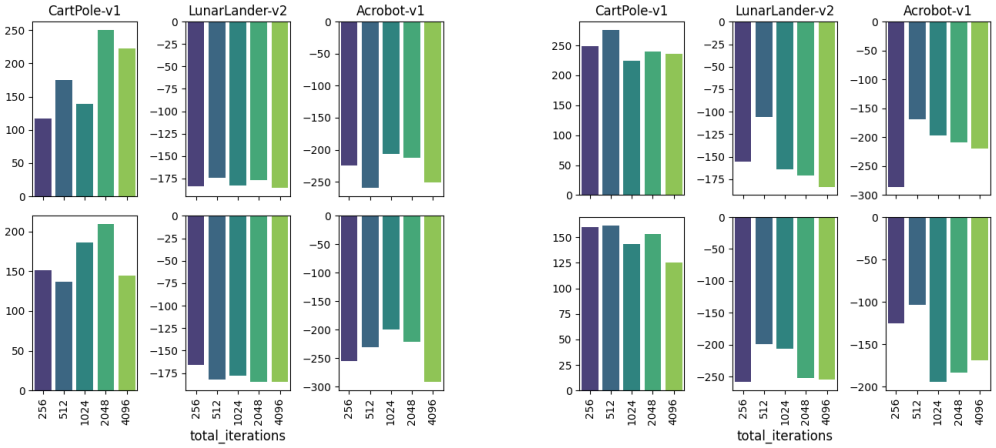


Figure F.13: Analysis of choice `total_iterations`: 95th percentile of performance scores conditioned on GAE=True (left), 95th percentile of performance scores conditioned on GAE=False (right). ELBE procedure on the first row, MinMax on the second row.

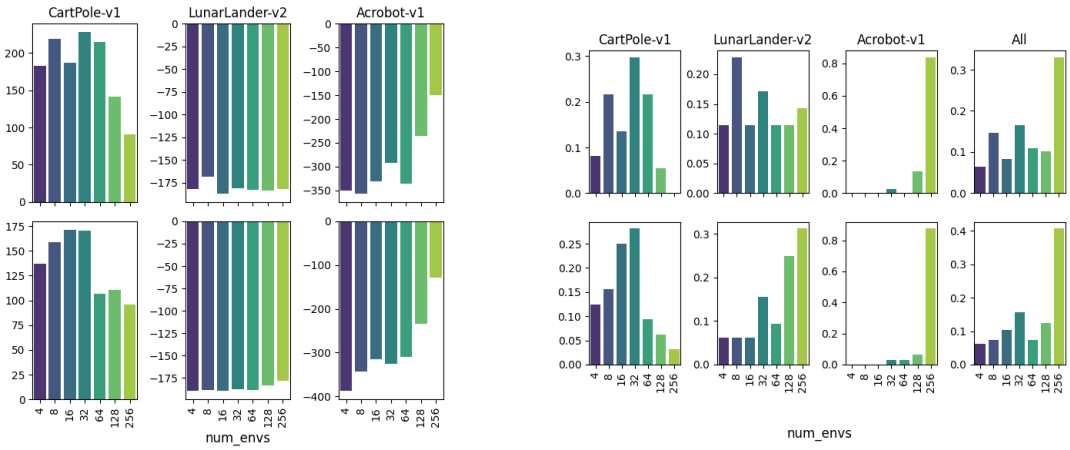


Figure F.14: Analysis of choice `num_envs`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

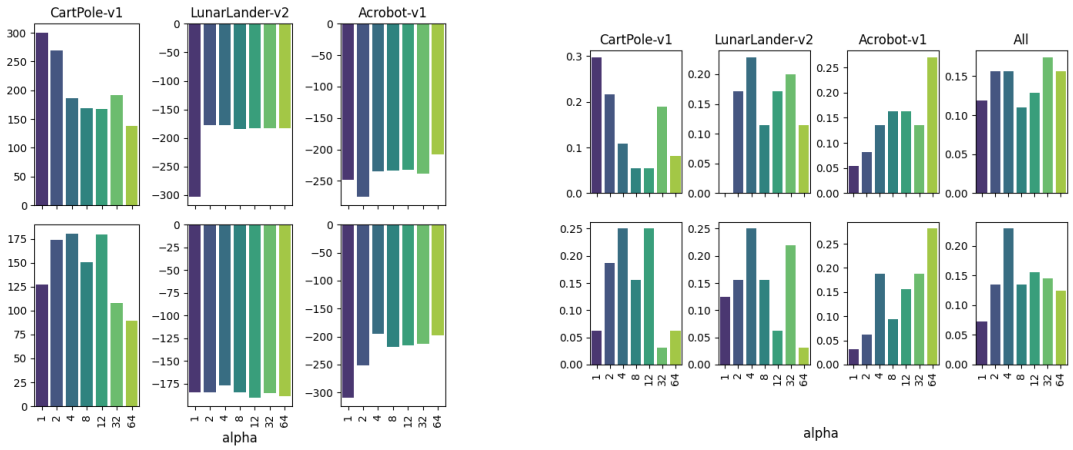


Figure F.15: Analysis of choice `alpha`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

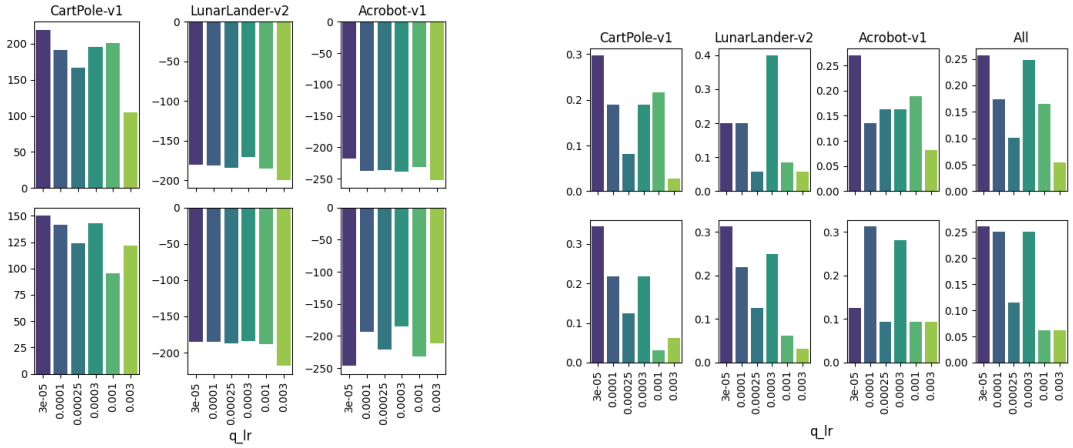


Figure F.16: Analysis of choice `q_lr`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

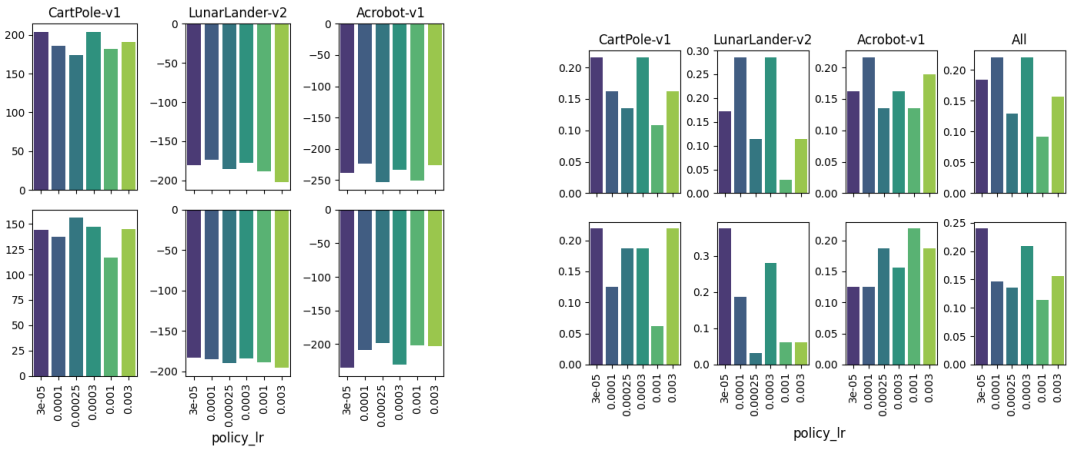


Figure F.17: Analysis of choice `policy_lr`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

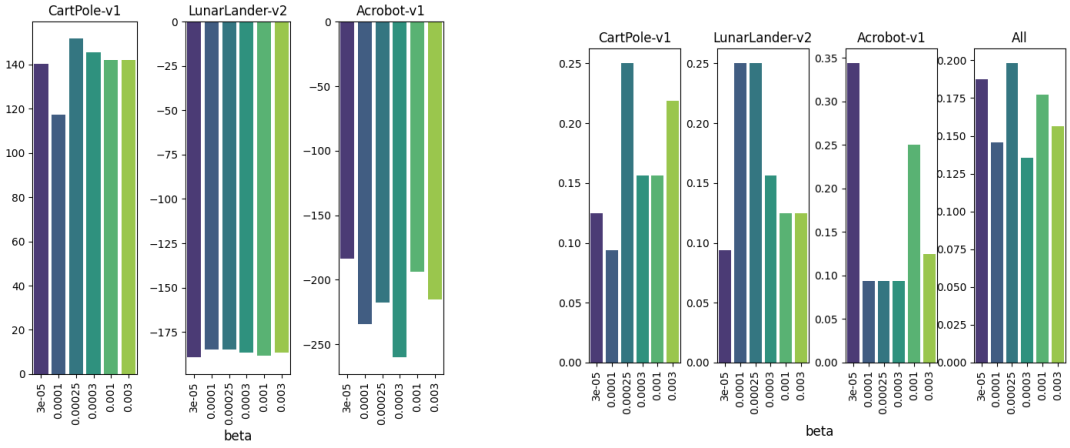


Figure F.18: Analysis of choice `beta`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

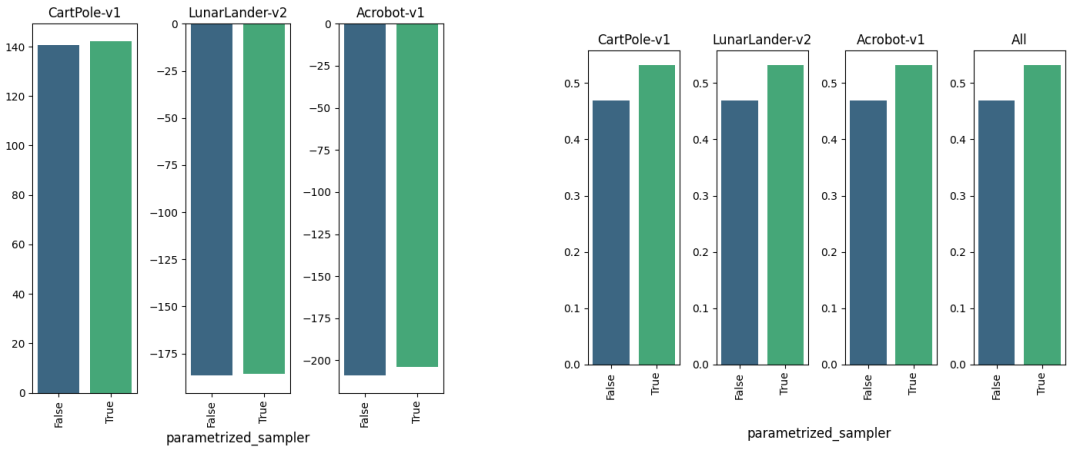


Figure F.19: Analysis of choice `parametrized_sampler`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

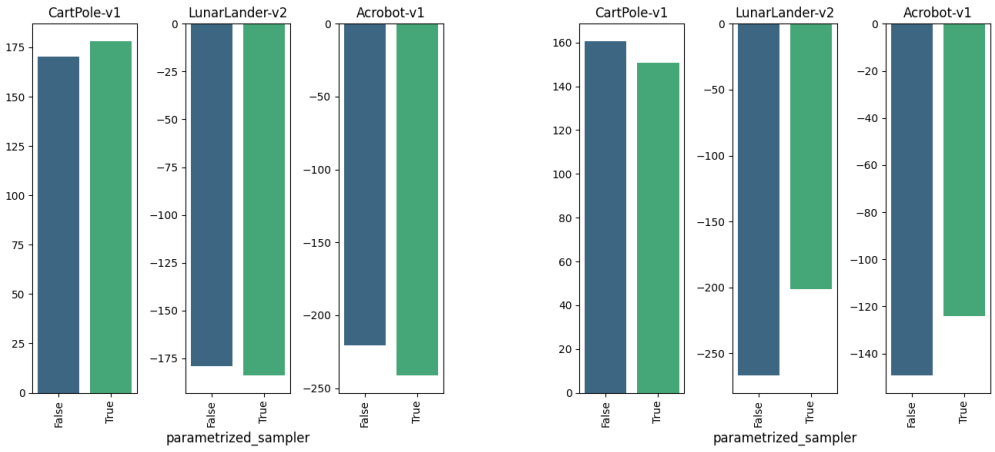


Figure F.20: Analysis of choice `total_iterations`: 95th percentile of performance scores conditioned on GAE=True (left), 95th percentile of performance scores conditioned on GAE=False (right). ELBE procedure on the first row, MinMax on the second row.

Appendix G

NETWORK ARCHITECTURE

G.1 Design

For each of the 3 environments, we sampled 3000 choice configurations where we sampled the following choices independently and uniformly from the following ranges:

- `q_last_layer_std` : {0.001, 0.01, 0.1, 1.0}
- `actor_last_layer_std`: {0.001, 0.01, 0.1, 1.0}
- `layer_init` : {Default, Orthogonal(gain=1.41), Orthogonal, Xavier Normal, Xavier uniform, He normal, He uniform}
- `policy_hidden_size` (C49): {16, 32, 64, 128, 256, 512}
- `policy_activation`: {ELU, Leaky ReLU, ReLU, Sigmoid, Tanh}
- `policy_num_hidden_layers`: {1, 2, 4, 8}
- `q_hidden_size` : {16, 32, 64, 128, 256, 512}
- `q_num_hidden_layers`: {1, 2, 4, 8}
- `q_activation` : {ELU, Leaky ReLU, ReLU, Sigmoid, Tanh}
- `policy_lr`: {3e-05, 0.0001, 0.00025, 0.0003, 0.001}
- `q_lr`: {3e-05, 0.0001, 0.00025, 0.0003, 0.001}
- `MinMax_optimization`: {True, False}
  - For the case `MinMax`: True,

    * `beta` (sampler learning rate): {3e-05, 0.0001, 0.0003, 0.001}
    * `parametrized_sampler`: {True, False}
        - For the case `Parametrized_sampler`: True,
        - `sampler_hidden_size` : {16, 32, 64, 128, 256, 512}
        - `sampler_activation`: {ELU, Leaky ReLU, ReLU, Sigmoid, Tanh}
        - `sampler_num_hidden_layers`: {1, 2, 4, 8}

- `anneal_lr`: {True, False}
- α: {0.5, 1, 2, 4, 8, 12, 32, 64, 100}

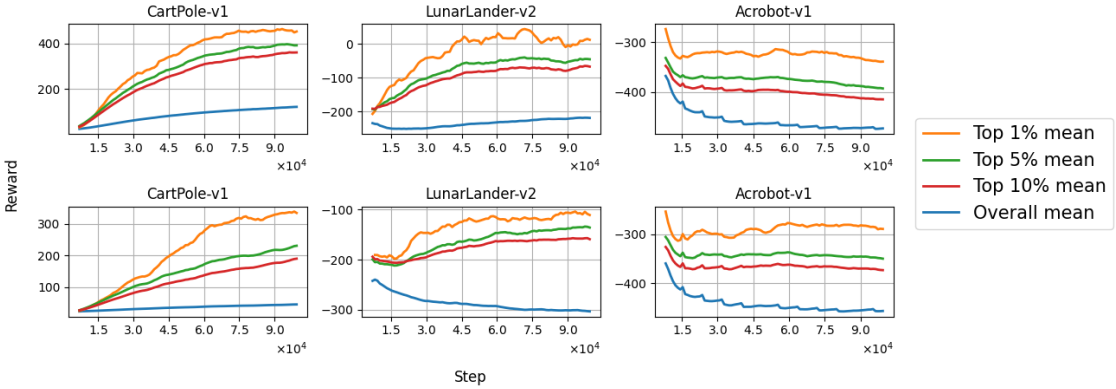


Figure G.1: Training curves. ELBE procedure on the top plots, MinMax on the bottom.

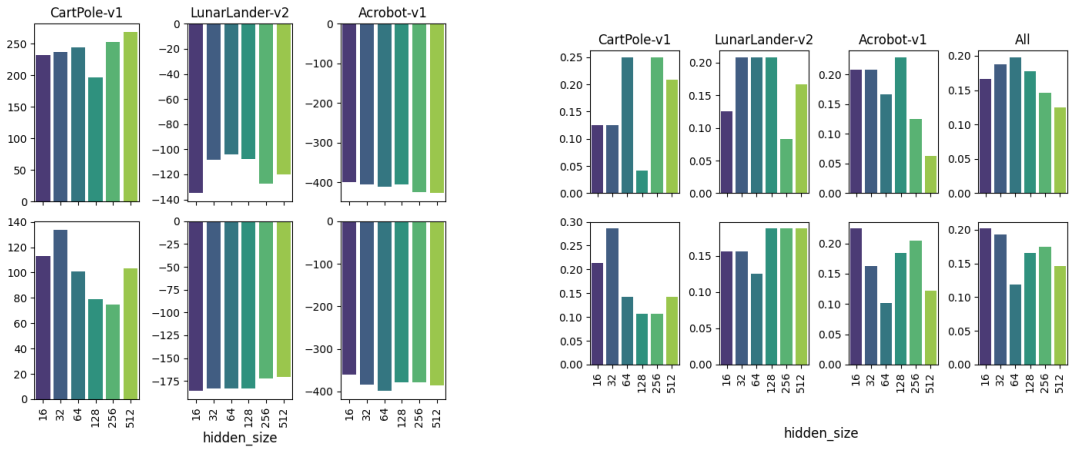


Figure G.2: Analysis of choice `policy_hidden_size`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

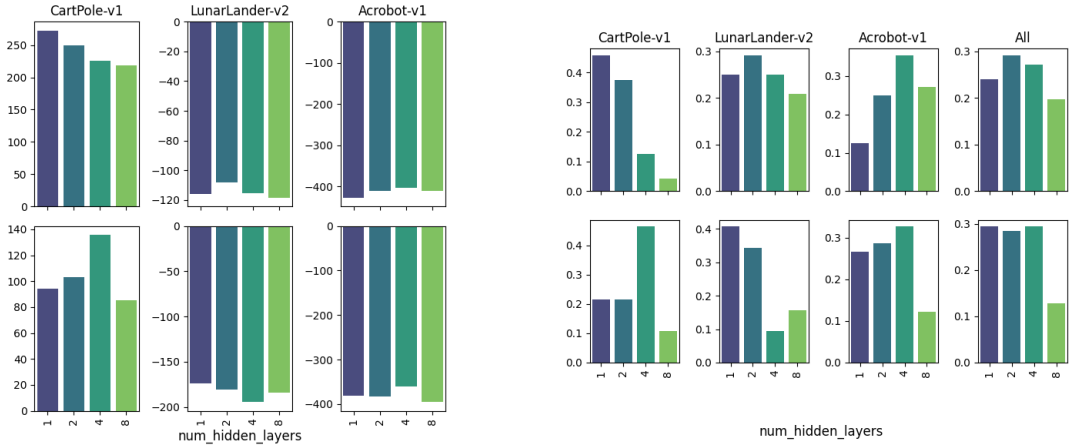


Figure G.3: Analysis of choice `policy_num_hidden_layers`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

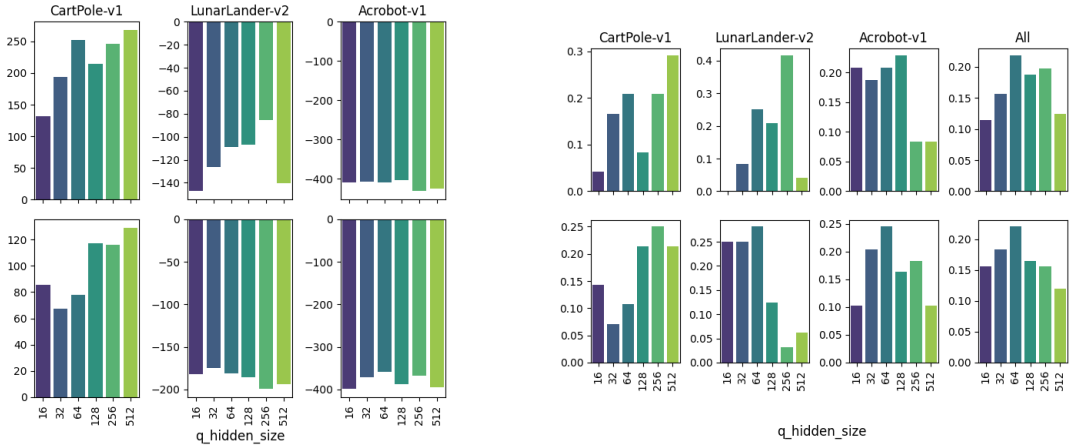


Figure G.4: Analysis of choice `q_hidden_size`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

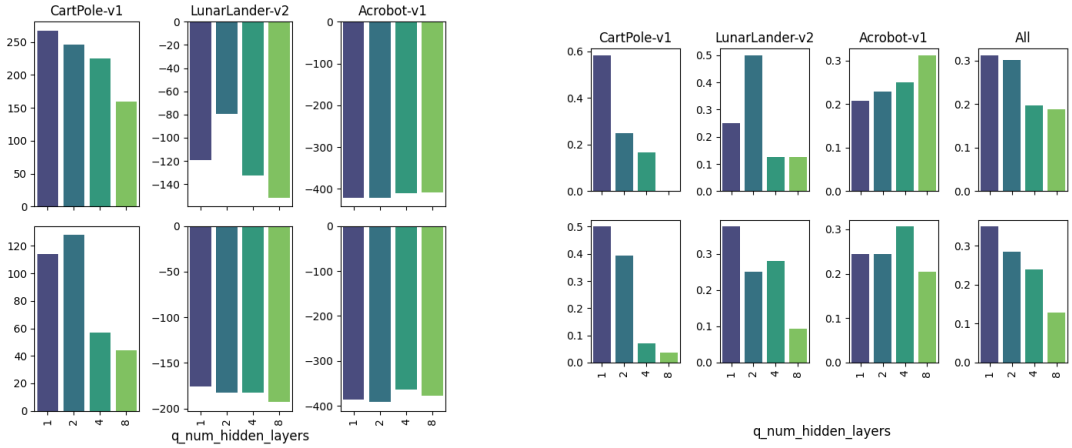


Figure G.5: Analysis of choice `q_num_hidden_layers`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

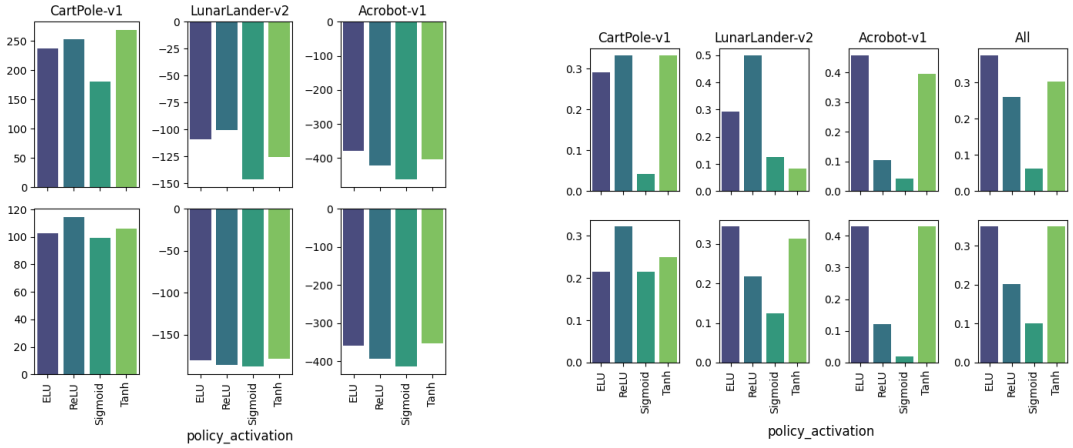


Figure G.6: Analysis of choice `policy_activation`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

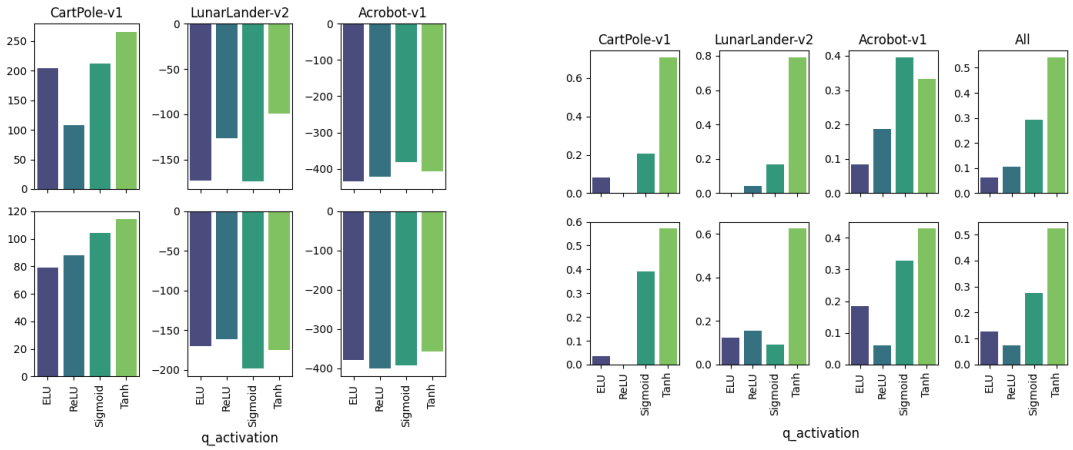


Figure G.7: Analysis of choice `q_activation`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

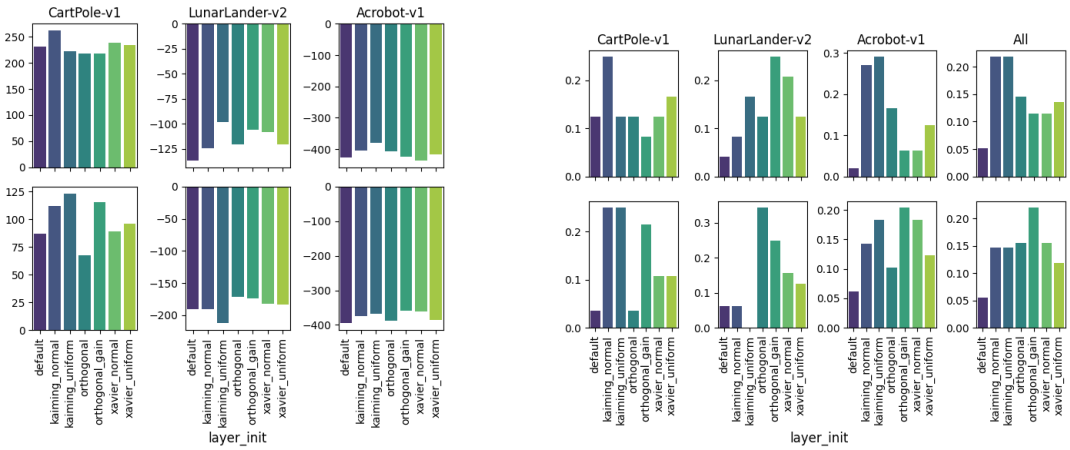


Figure G.8: Analysis of choice `layer_initialization`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

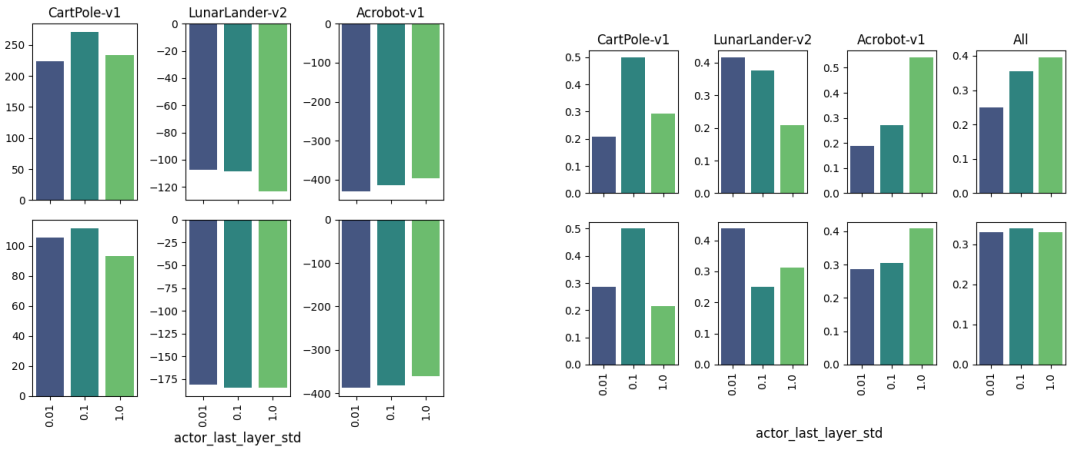


Figure G.9: Analysis of choice `policy_last_layer_std`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

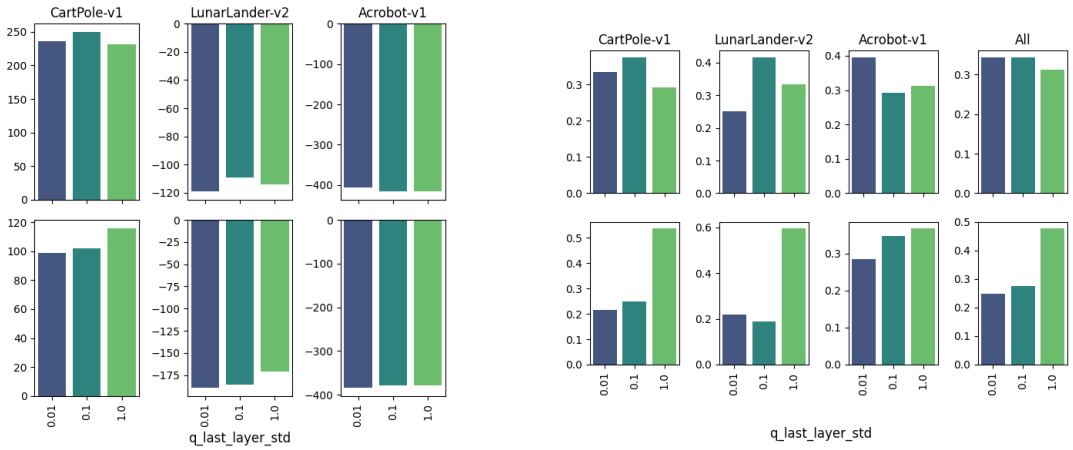


Figure G.10: Analysis of choice `q_last_layer_std`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

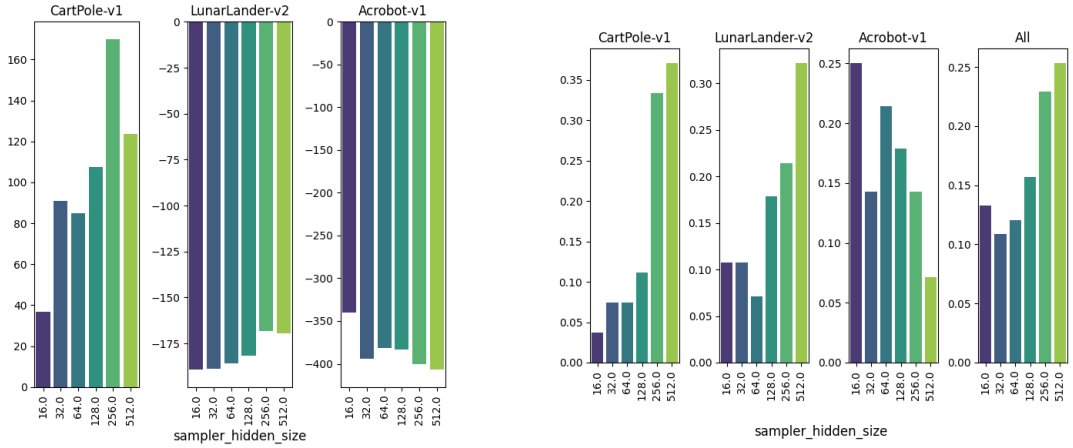


Figure G.11: Analysis of choice `sampler_hidden_size`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

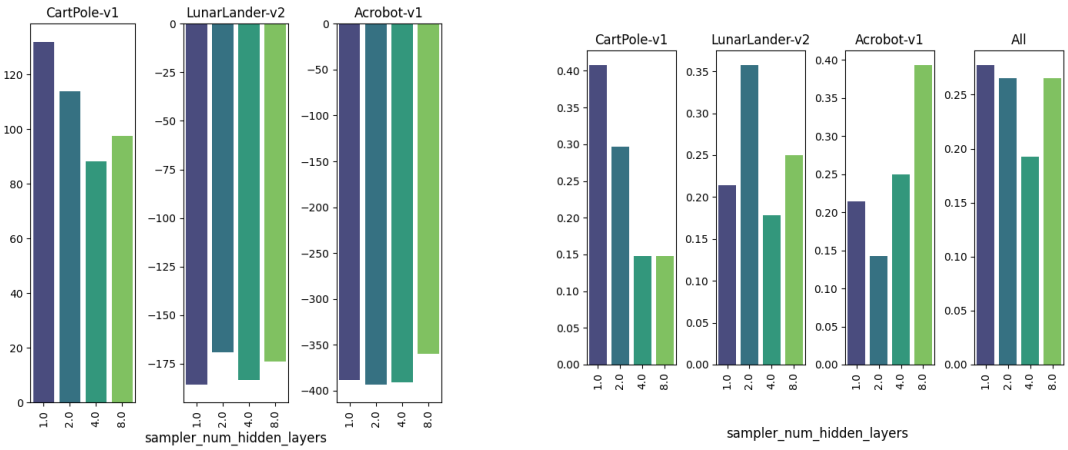


Figure G.12: Analysis of choice `sampler_num_hidden_layers`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

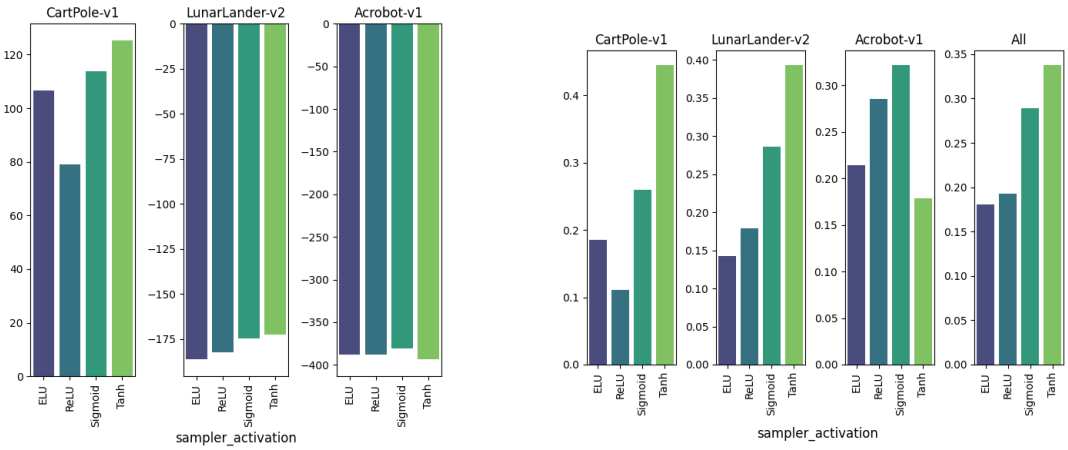


Figure G.13: Analysis of choice `sampler_activation`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

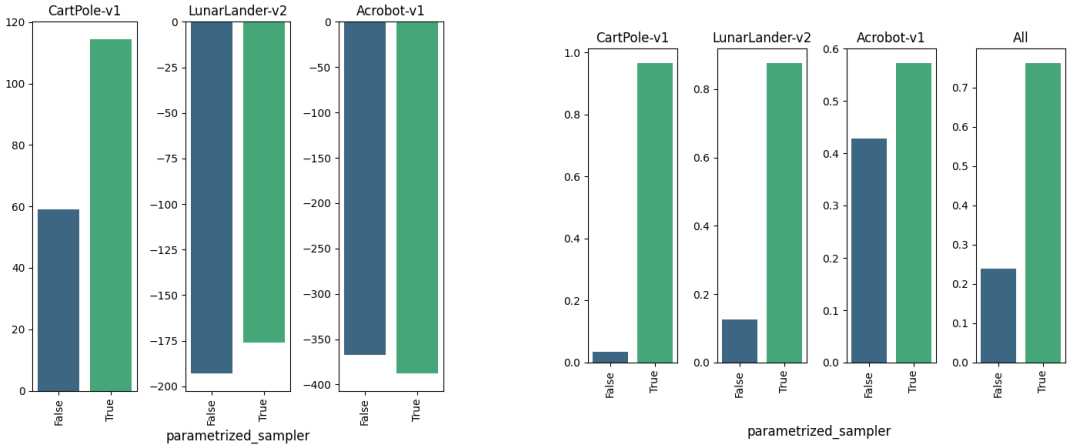


Figure G.14: Analysis of choice `parametrized_sampler`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

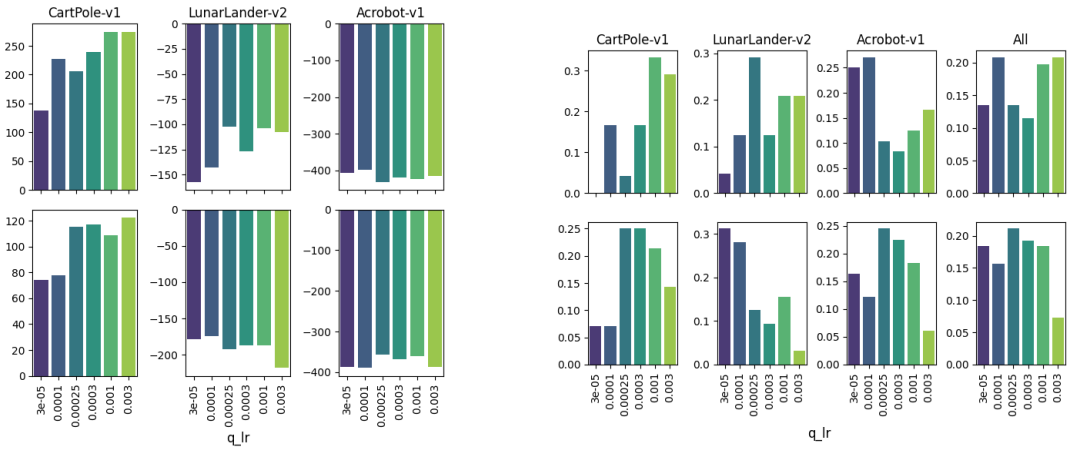


Figure G.15: Analysis of choice `q_lr`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

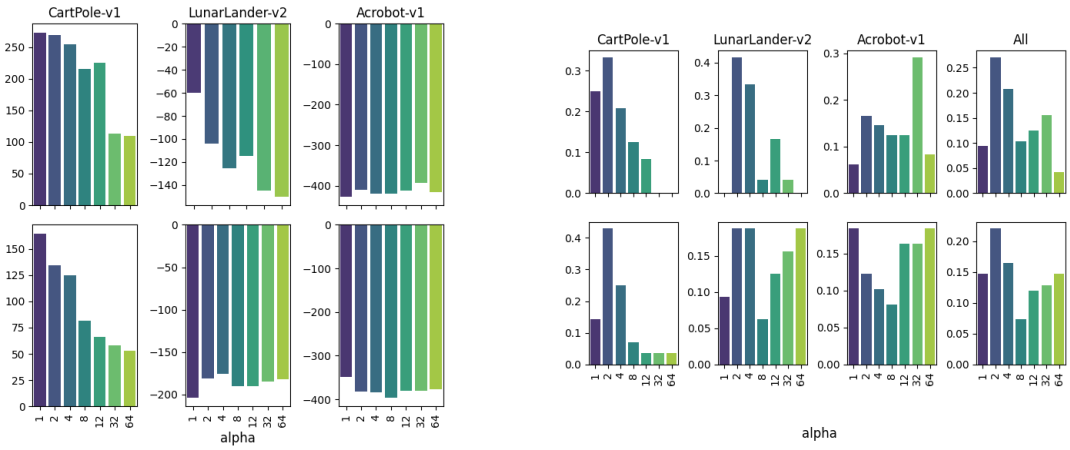


Figure G.16: Analysis of choice `alpha`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

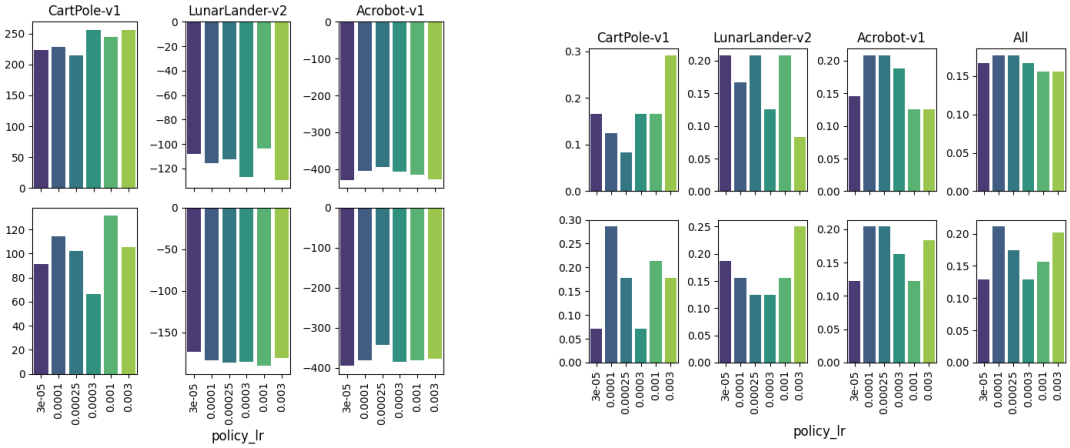


Figure G.17: Analysis of choice `policy_lr`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

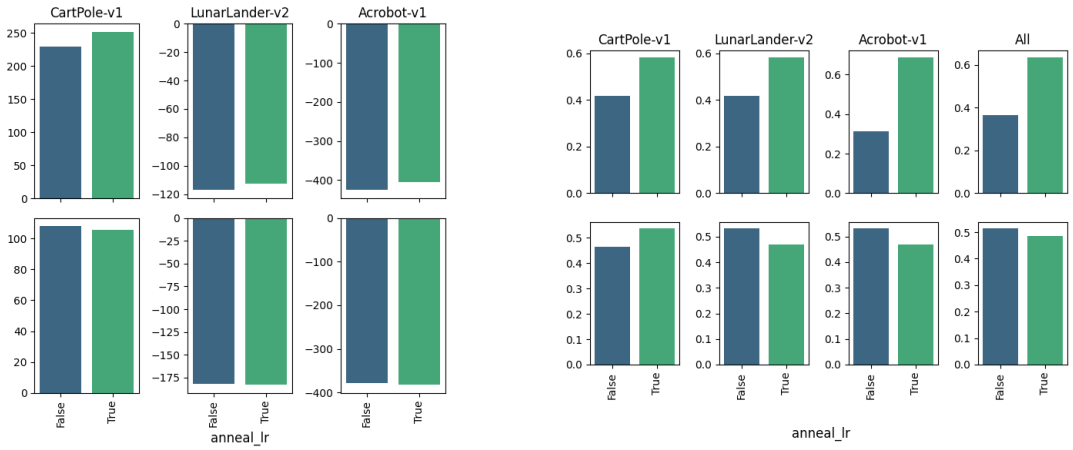


Figure G.18: Analysis of choice `anneal_lr`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

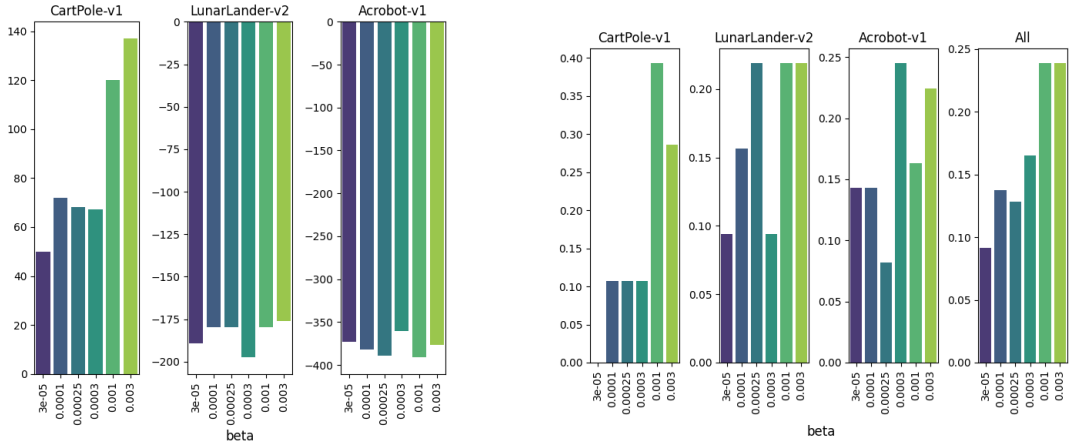


Figure G.19: Analysis of choice `beta`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

Appendix H

TIME

H.1 Design

For each of the 3 environments, we sampled 2000 choice configurations where we sampled the following choices independently and uniformly from the following ranges:

- `gamma`: {0.95, 0.97, 0.99, 0.999}
- `policy_lr`: {3e-05, 0.0001, 0.00025, 0.0003, 0.001}
- `q_lr`: {3e-05, 0.0001, 0.00025, 0.0003, 0.001}
- `MinMax_optimization`: {True, False}
  - For the case `MinMax_optimization`: True,
    - `beta` (sampler learning rate): {3e-05, 0.0001, 0.0003, 0.001}
    - `parametrized_sampler`: {True, False}
- α: {0.5, 1, 2, 4, 8, 12, 32, 64, 100}

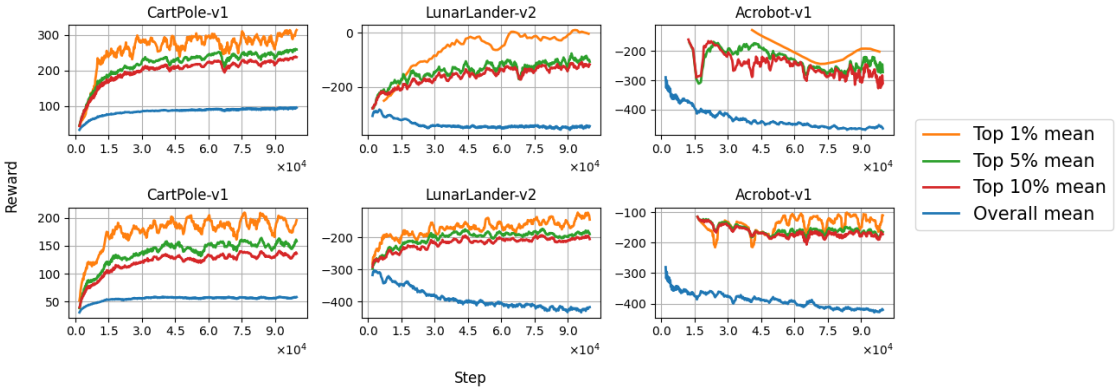


Figure H.1: Training curves. ELBE procedure on the top plots, MinMax on the bottom.

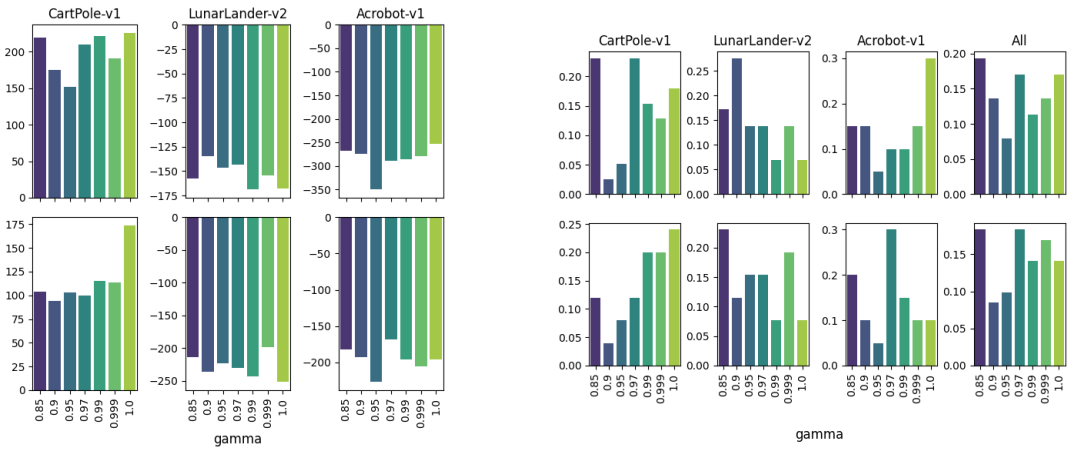


Figure H.2: Analysis of choice `gamma`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

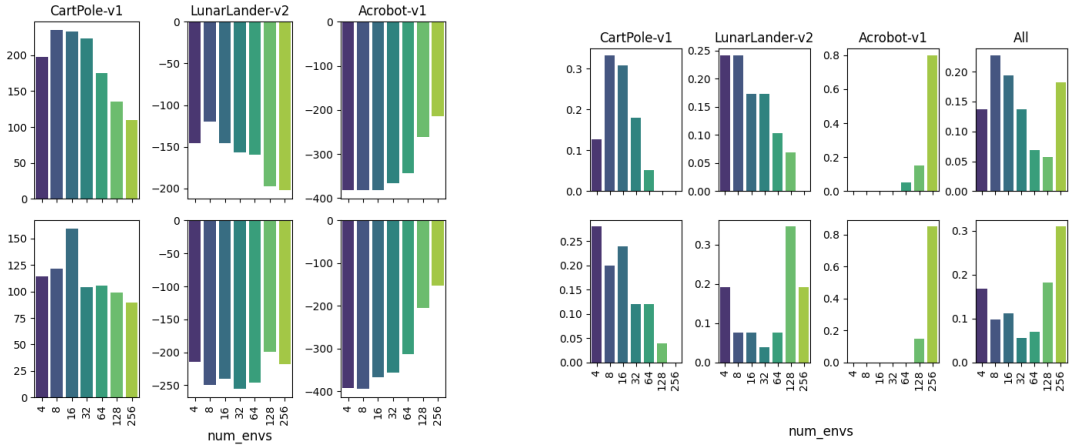


Figure H.3: Analysis of choice `num_envs`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

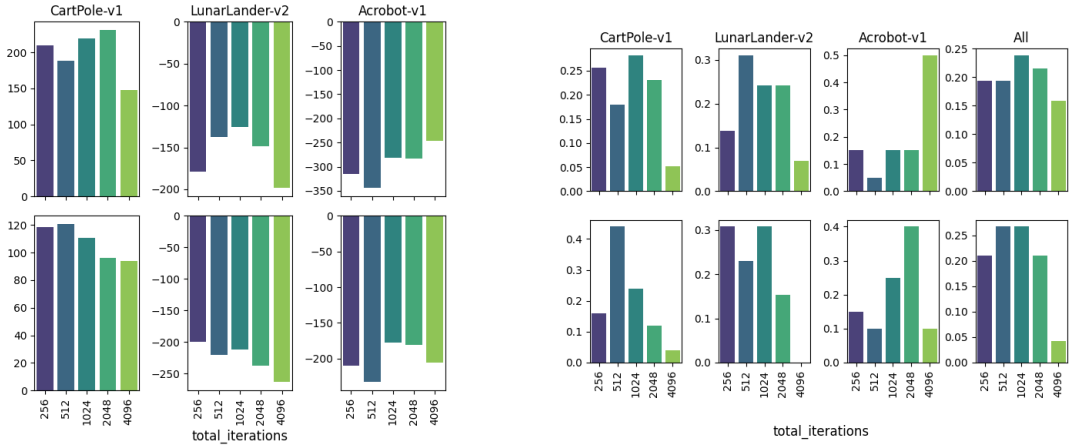


Figure H.4: Analysis of choice `total_iterations`: 95th percentile of performance scores conditioned on choice (left), distribution of choices within the top 5% configurations (right). ELBE procedure on the first row, MinMax on the second row.

Appendix I

Q-REPS AND PD-API IN A TABULAR SETTING

We study the impact of the different design choices explained in the main text for the tabular setting. We sampled 1000 choice configurations for PD-API and qreps independently, where we sampled the following choices independently and uniformly from the following ranges:

I.1 Design

- `update_epochs` (K) : {5, 8, 10, 15, 20, 25, 30, 50}
- `reg_coefficient`: {0.1, 0.5, 1.0, 2.0, 2.5, 4.0, 8.0, 10.0}
- `policy_temp` : {0.003, 0.005, 0.01, 0.03, 0.04, 0.05, 0.1, 0.2, 0.4}
- `z_lr`: {0.0001, 0.0005, 0.001, 0.002, 0.003, 0.005, 0.01, 0.03, 0.05}
- `q_lr`: {0.0001, 0.0005, 0.001, 0.002, 0.003, 0.005, 0.01, 0.03, 0.05}
- `sampling_strategy`: {random, value_dist}
- `q_init`: {zero, unform_dist, max}

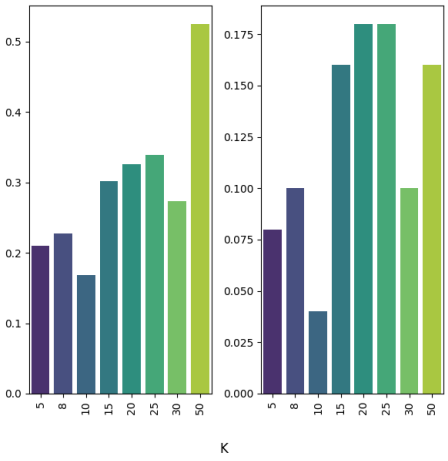


Figure I.1: PD-API

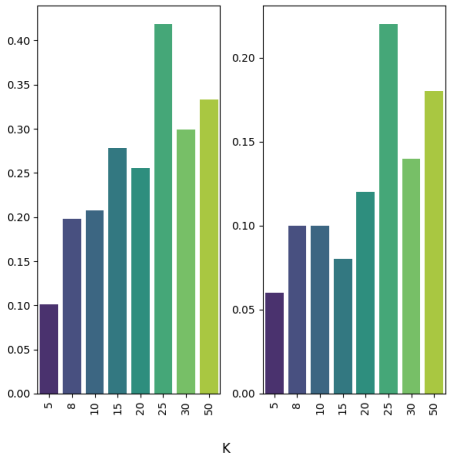


Figure I.2: Q-reps

Figure I.3: Analysis of choice `update_epochs`: 95th percentile of performance scores (left), distribution of choices within the top 5% configurations (right).

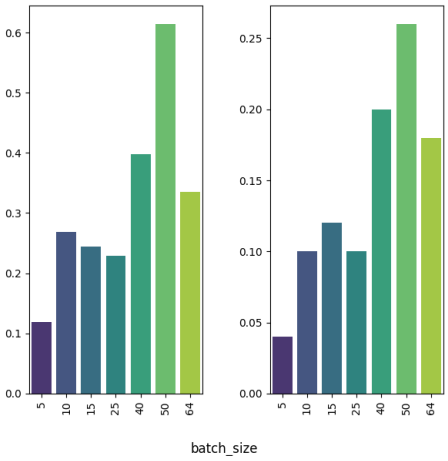


Figure I.4: PD-API

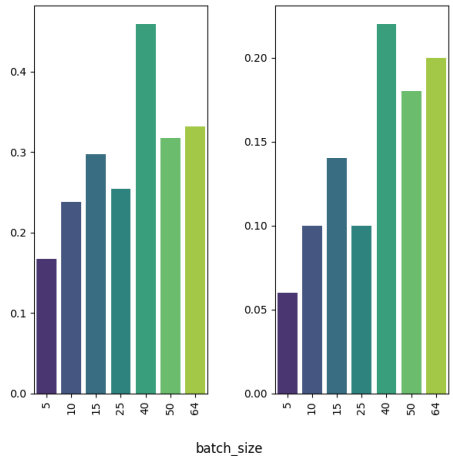


Figure I.5: Q-reps

Figure I.6: Analysis of choice `batch_size`: 95th percentile of performance scores (left), distribution of choices within the top 5% configurations (right).

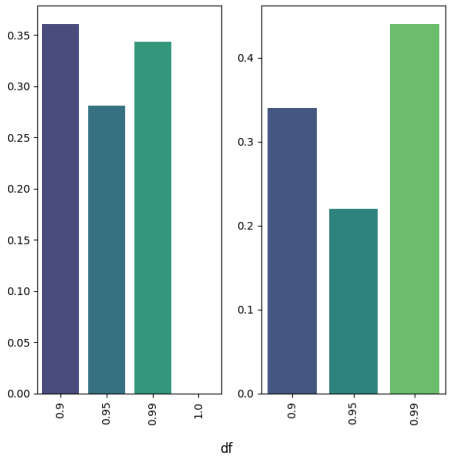


Figure I.7: PD-API

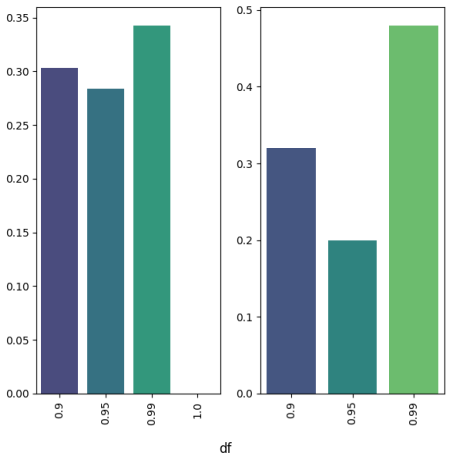


Figure I.8: Q-reps

Figure I.9: Analysis of choice γ: 95th percentile of performance scores (left), distribution of choices within the top 5% configurations (right).

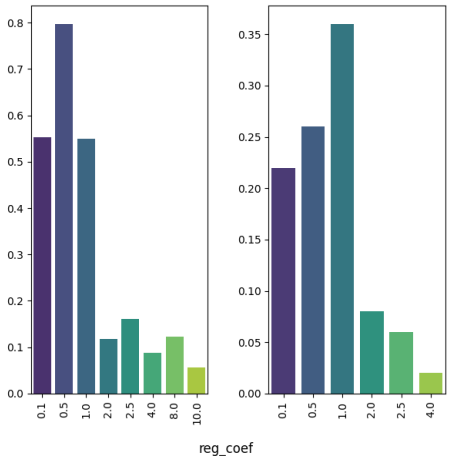


Figure I.10: PD-API

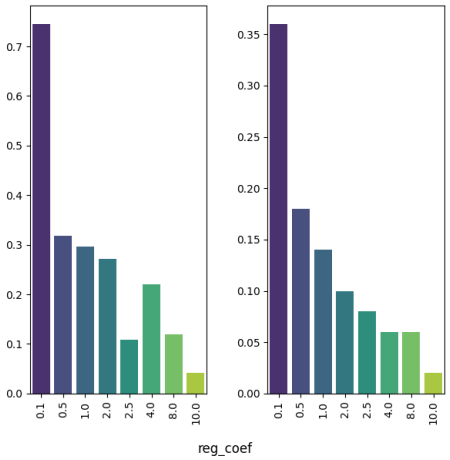


Figure I.11: Q-reps

Figure I.12: Analysis of choice `reg_coef`: 95th percentile of performance scores (left), distribution of choices within the top 5% configurations (right).

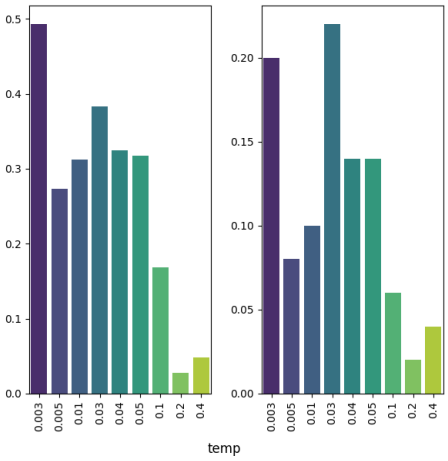


Figure I.13: PD-API

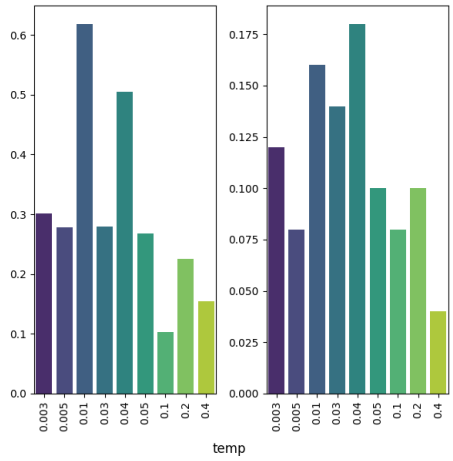


Figure I.14: Q-reps

Figure I.15: Analysis of choice `policy_temp`: 95th percentile of performance scores (left), distribution of choices within the top 5% configurations (right).

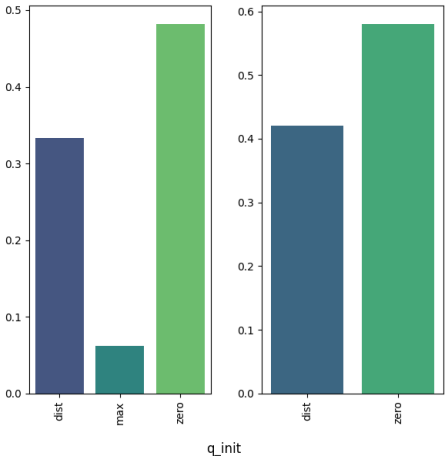


Figure I.16: PD-API

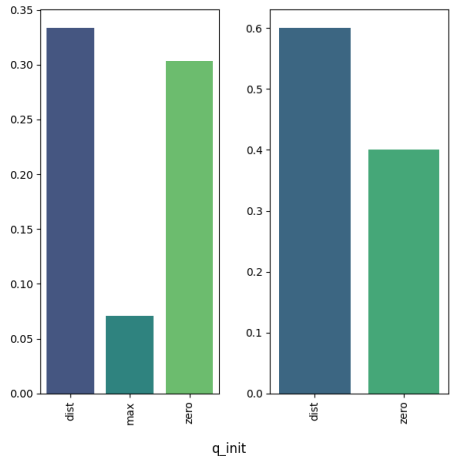


Figure I.17: Q-reps

Figure I.18: Analysis of choice `q_init`: 95th percentile of performance scores (left), distribution of choices within the top 5% configurations (right).

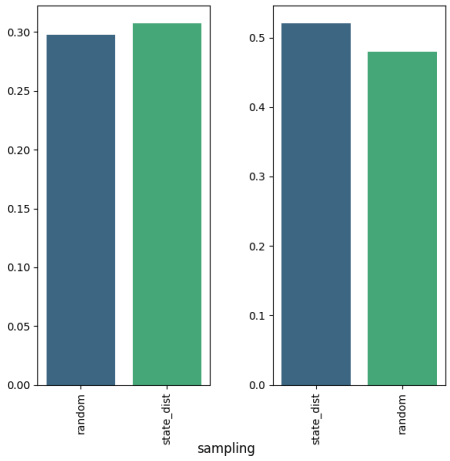


Figure I.19: PD-API

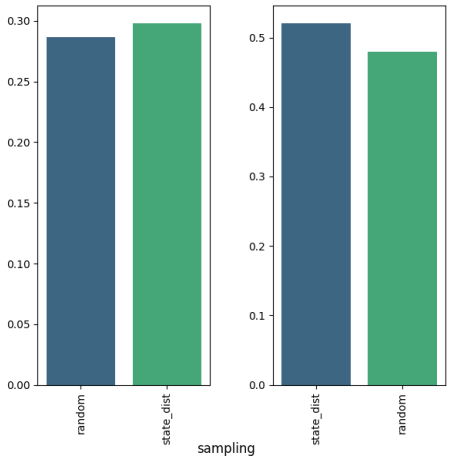


Figure I.20: Q-reps

Figure I.21: Analysis of choice `sampling`: 95th percentile of performance scores (left), distribution of choices within the top 5% configurations (right).

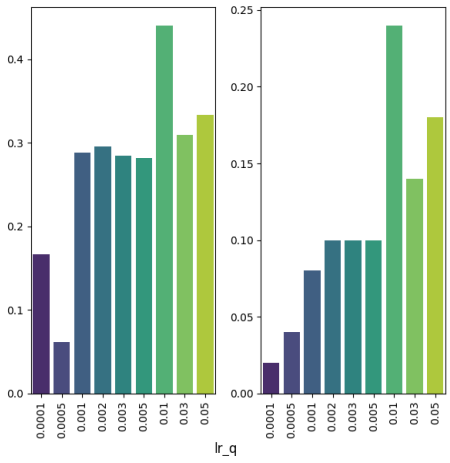


Figure I.22: PD-API

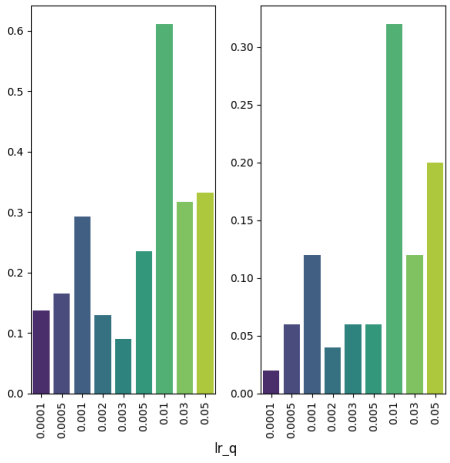


Figure I.23: Q-reps

Figure I.24: Analysis of choice `lr_q`: 95th percentile of performance scores (left), distribution of choices within the top 5% configurations (right).

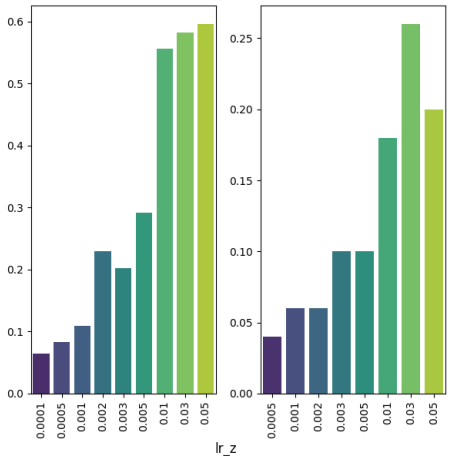


Figure I.25: PD-API

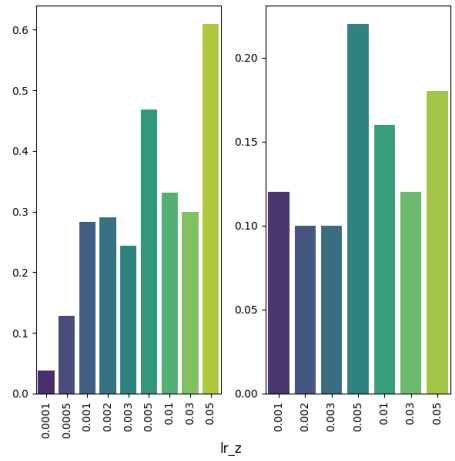


Figure I.26: Q-reps

Figure I.27: Analysis of choice `lr_z`: 95th percentile of performance scores (left), distribution of choices within the top 5% configurations (right).

I.2 Parameters used in the tabular setting

Table I.1: Parameters for Tabular experiments.

Hyperparameter	**Canonical**	**PD-API**	**Q-REPS**
`update_epochs`	1	30	30
`reg_coefficient`	-	0.1	0.1
`policy_temp`	-	0.03	0.03
`batch_size`	1	40	40
`z_lr`	-	0.01	0.01
`q_lr`	-	0.003	0.003
`mu_lr`	0.001	-	-
`v_lr`	0.0031	-	-
`sampling_strategy`	value	value	value
`q_init`	-	uniform	zero
γ	0.95	0.95	0.95

I.3 Experiments on different grid configurations

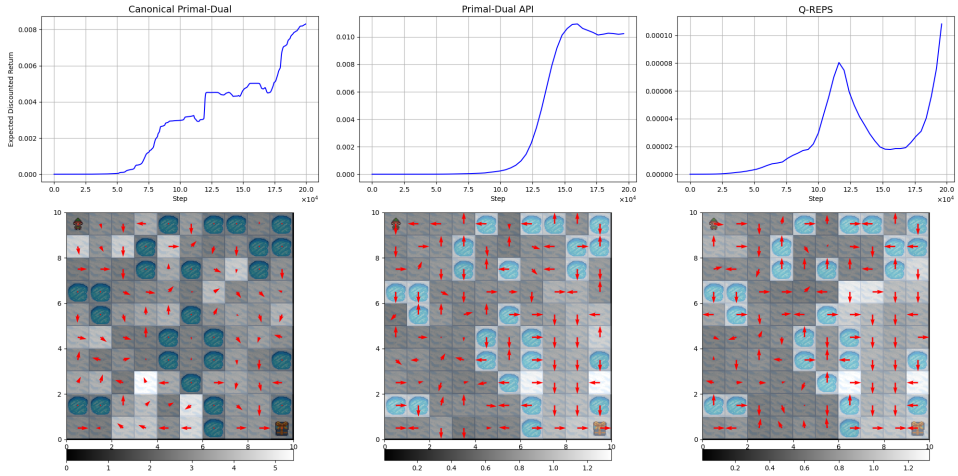


Figure I.28: Testing algorithm scalability to a 10x10 grid without changing hyperparameters.

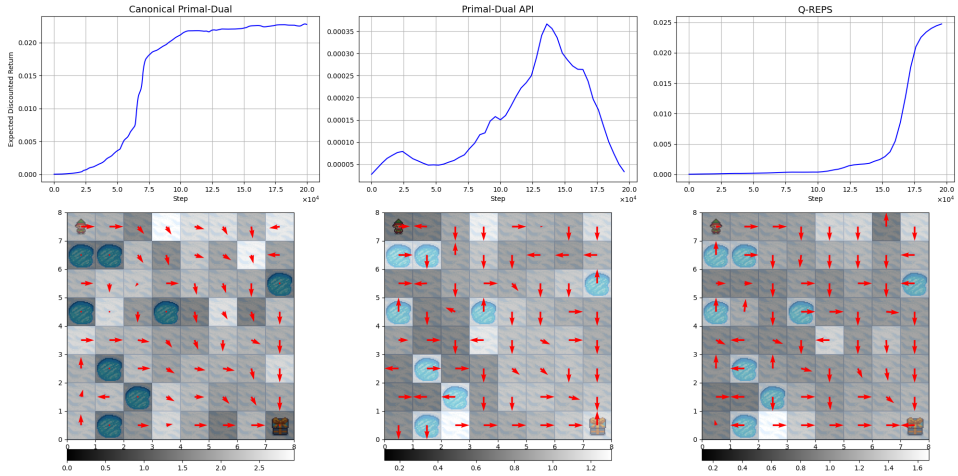


Figure I.29: Example where PD-API fails to converge.