Differentiable Trust Region Projection Layers

Differentiable Trust Region Projection Layers

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von
M.Sc. Fabian Otto
aus Fürth

Tübingen
2024

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	02.07.2024
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Gerhard Neumann
2. Berichterstatter:	Prof. Dr. Georg Martius

For my wife

Abstract

Deep reinforcement learning and especially policy gradient methods have achieved remarkable success in various domains. However, challenges remain for policy gradient-based methods, characterized by issues such as premature exploitation and the difficulty of selecting appropriate step sizes. Mitigating these challenges requires nuanced approaches, and one effective strategy is to impose trust regions in the form of Kullback-Leibler divergence constraints on policy updates. Well-known methods such as Trust Region Policy Optimization and Proximal Policy Optimization adopt this approach, but they often rely on heuristic-based algorithms, exhibit implementation-dependent behavior, or lack scalability. In response to these limitations, this thesis introduces a novel algorithm based on differentiable trust region projection layers. This method offers a comprehensive and mathematically principled approach, ensuring efficiency, stability, and consistency for deep policy gradient methods. Importantly, the proposed algorithm delivers comparable or superior results to existing methods while remaining agnostic to specific implementation choices and enforcing the trust regions exactly per state. Moreover, it facilitates stable learning in high-dimensional and complex action spaces, making it particularly suitable for learning in the trajectory space through movement primitives from classical robotics. This integration combines the advantages of classical robotics, such as generating smooth and energy-efficient trajectories as well as adapting to sparse and non-Markovian rewards, with the scalability of deep reinforcement learning methods. Additionally, we extend this method from the on-policy setting to the off-policy setting and also eliminate the need for an explicit state-action-value function while preserving learning stability. This innovation streamlines the learning process and enhances exploration and exploitation efficiency for off-policy learning, especially in higher-dimensional action spaces. All proposed algorithms are validated through extensive experiments on a variety of simulated tasks, including locomotion and manipulation.

Kurzfassung

Tiefe Reinforcement Learning- und insbesondere Policy Gradient-Methoden haben in verschiedenen Bereichen bemerkenswerte Erfolge erzielt. Dennoch bleiben Herausforderungen für Policy Gradient-basierte Methoden bestehen, die durch Probleme wie vorzeitige Exploitation und die Schwierigkeit, geeignete Schrittgrößen zu wählen, gekennzeichnet sind. Eine effektive Strategie ist die Definition von Trust Regions in Form von Kullback-Leibler Divergenzbeschränkungen für Policy Updates. Bekannte Verfahren wie Trust Region Policy Optimization und Proximal Policy Optimization verfolgen diesen Ansatz, sind aber oft auf heuristische Algorithmen angewiesen, zeigen implementierungsabhängiges Verhalten oder sind nicht skalierbar. Um diese Einschränkungen zu überwinden, wird in dieser Arbeit ein neuartiger Algorithmus vorgestellt, der auf differenzierbaren Projektionsschichten für Trust Regions basiert. Diese Methode bietet einen umfassenden und mathematisch prinzipiellen Ansatz, der Effizienz, Stabilität und Konsistenz für tiefe Policy Gradient-Methoden gewährleistet. Wichtig ist, dass der vorgeschlagene Algorithmus vergleichbare oder bessere Ergebnisse als existierende Methoden liefert, während er unabhängig von spezifischen Implementierungsentscheidungen bleibt und die Trust Regions genau pro State durchsetzt. Darüber hinaus ermöglicht er stabiles Lernen in hochdimensionalen und komplexen Aktionsräumen, was ihn besonders geeignet macht für das Lernen im Trajektorienraum durch Bewegungsprimitive aus der klassischen Robotik. Diese Integration kombiniert die Vorteile der klassischen Robotik, wie die Erzeugung von flüssigen und energieeffizienten Trajektorien und die Anpassung an begrenzte und nicht-markovsche Belohnungen, mit der Skalierbarkeit tiefer Reinforcement Learning-Methoden. Darüber hinaus erweitern wir diese Methode von der On-Policy-Umgebung auf die Off-Policy-Umgebung und eliminieren die Notwendigkeit einer expliziten State-Action-Value-Funktion unter Beibehaltung der Lernstabilität. Diese Neuerung vereinfacht den Lernprozess und verbessert die Effizienz des Exploration-Exploitation Trade-offs beim Off-Policy Lernen, insbesondere in höherdimensionalen Aktionsräumen. Alle vorgeschlagenen Algorithmen werden durch umfangreiche Experimente mit einer Vielzahl von simulierten Herausforderungen, einschließlich Fortbewegung und Manipulation, validiert.

Acknowledgments

I want to express my sincere gratitude to my advisor, Gerhard Neumann, whose unwavering support has been a cornerstone throughout the challenging phases of this doctoral journey. His constructive feedback, consistent encouragement, and dedicated investment of time in my research have proven indispensable to the successful completion of this dissertation. In moments of self-doubt, he stood by my side, offering not only guidance but also a belief in my abilities. His unwavering faith in my potential has been a driving force, and I am deeply and forever grateful for that.

A special note of gratitude goes to the Bosch Center for Artificial Intelligence for their generous financial support, which played a pivotal role in facilitating crucial experiments, conference attendance, and access to essential resources that contributed significantly to the successful completion of this research. In particular, I would like to express my special gratitude to my two supervisors at the Bosch Center for Artificial Intelligence, Vien Anh Ngo and Hanna Ziesche. They have always been like a second academic supervisor to me, and their guidance, valuable advice, and substantive contributions to my work have been indispensable throughout this academic endeavor.

The camaraderie of our research group has been a source of inspiration. To my colleagues, thank you for the late-night discussions, the collaborative problem solving, and the shared joy of small victories. Your friendship has made this academic journey both enjoyable and memorable. Special appreciation goes to Philipp Becker and Onur Celik, my closest collaborators, and to dedicated colleagues including Hongyi Zhou, Rudolf Lioutikov, and Ge Li, whose contributions have greatly enriched the depth and quality of my research. I would also like to express my gratitude to a broader network of individuals with whom I have had the privilege of exchanging ideas, validating thoughts, and engaging in insightful discussions. I specifically thank David Adrian, Hadi Beik-Mohammadi, Felix Berkenkamp, Matthias Bürger, Fabian Duffhauß, Niklas Freymuth, Ning Gao, Maximilian Hüttenrauch, Andras Kupcsik, Jan Peters, Leonel Rozo, Aleksandar Taranovic, and Jan Wöhlke for their meaningful input and support throughout this research journey.

My deepest gratitude extends to my family, with heartfelt acknowledgment to my wife, Alina Otto-Krieger. Her unwavering support and understanding have been the bedrock of my academic journey. During the most challenging moments, her enduring encouragement and belief in my capabilities not only kept me motivated but also transformed obstacles into opportunities for growth. Her sacrifices and understanding were instrumental in navigating the demands of this pursuit. I am profoundly thankful for her enduring love, strength, and the countless ways she enriched this journey.

Contents

Chapter 1

Introduction

1.1 Problem Setting

Deep *reinforcement learning* (RL) applications have achieved notable success in several domains, including games (Mnih et al., 2015; Silver et al., 2017), robotics (Levine et al., 2015), and control (Duan et al., 2016). In particular, in the area of policy search, *policy gradient* (PG) methods have undergone significant developments (Peters and Schaal, 2008). However, the use of vanilla PG methods presents many challenges, especially in dealing with the exploration-exploitation tradeoff. Conceptually, PG methods aim to shift the policy distribution towards areas of the problem that yield the most positive feedback based on the given task. Nevertheless, a natural consequence is that the agent tends to explore minimally and exploit a locally optimal solution, seeking a consistent stream of almost equally well-performing samples (see Figure 1.1 left). This premature exploitation often hinders the agent from reaching its full potential. Therefore, a crucial aspect of achieving a well-performing agent lies in facilitating early exploration and progressively enhancing exploitation over time.
Additionally, another related significant concern in PG methods can be demonstrated when examining the PG of a simple Gaussian policy

$$\nabla_\theta \log \pi_\theta(a) = \frac{(a-\theta)}{\sigma^2} \quad \text{for} \quad \pi_\theta(a) = \mathcal{N}(a|\theta, \sigma^2).$$

The magnitude of the gradient varies considerably, inversely proportional to the variance of the Gaussian distribution. Consequently, selecting an appropriate step size becomes challenging, as it must potentially be adapted throughout the training process. Excessive step-sizes may lead the policy to leap into „unexplored“ regions where performance can be arbitrarily good or bad (see Figure 1.1 left), while overly small step-sizes increase the required number of samples, potentially preventing the policy from finding a satisfactory solution (see Figure 1.1 center).
To address these limitations, a common practice involves imposing constraints on the allowable changes between successive policy distributions. This is frequently achieved through a *Kullback-Leibler* (KL) divergence constraint between successive policies to regulate the size of policy updates. This approach enables a dynamic adjustment of the

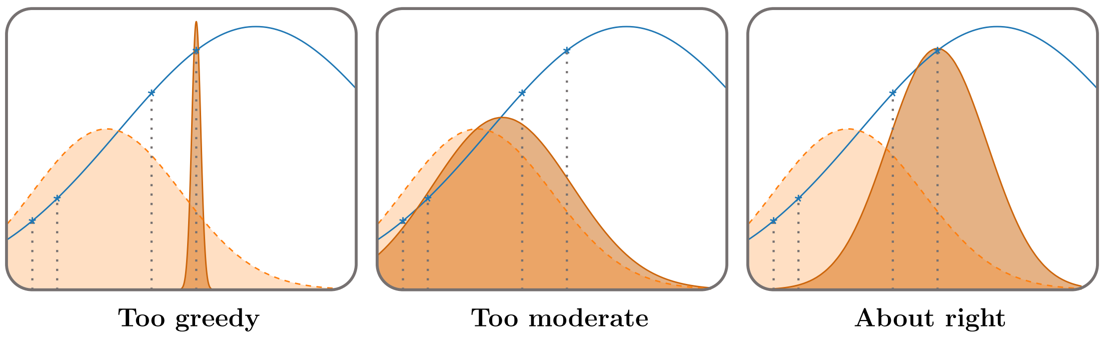


Figure 1.1: Controlling the exploration-exploitation behavior in vanilla PG methods can be challenging. These methods inherently aim to reduce entropy and locally optimize performance to generate similarly performing samples (left). However, this often leads to premature convergence. Conversely, decreasing the step size may result in inadequate learning speed, impacting sample efficiency (center). Furthermore, depending on the policy parametrization, the scale of the gradient can be influenced by the current policy distribution. Balancing exploration and selecting an optimal step size becomes a challenging task (right). Trust regions now offer a solution by constraining the extent to which the policy distribution can change, thereby improving the optimization process.

constraint based on the policy distribution. While smaller changes in the early stages of training are less critical and fall within the constraint limits, the same magnitude of change becomes more significant later on when entropy or exploration is reduced. Two prominently known methods in this realm are *trust region policy optimization* (TRPO) (Schulman et al., 2015a) and *proximal policy optimization* (PPO) (Schulman et al., 2017). Notably, PPO has been applied to challenging tasks, including competitive multiplayer games in OpenAI Five (Berner et al., 2019), emergent tool use (Baker et al., 2020), and locomotion (Heess et al., 2017). The success of PPO likely stems from its speed and simplicity; however, other approaches, like TRPO, are more principled but struggle to scale to the complexities of the aforementioned problems. Despite PPO's success, its algorithmic approach is heuristic-based and highly reliant on implementation details (Engstrom et al., 2020). Moreover, even other principled approaches, such as TRPO, often approximate the trust region and fail to impose constraints for individual states.

We can now further improve and refine these RL agents by selecting better-suited representations or by incorporating modifications that directly support behavior generation and exploration. This may involve narrowing down the search space of viable options, thereby reducing complexity, or increasing robustness to ensure more reliable gradients. Additionally, we can go one step further. Many prominent RL methods (Haarnoja et al., 2018; Schulman et al., 2015a, 2017; Fujimoto et al., 2018; Abdolmaleki et al., 2018a) learn policies that directly predict raw actions, which are subsequently executed in the environment. However, depending on the specific control signal, this approach can yield significantly different performances (Schneider et al., 2023). In classical robotics, it is more common to represent policies at the trajectory level, encompassing the entire motion (Schaal et al., 2005; Schaal, 2006; Ijspeert et al., 2013; Paraschos et al., 2013). This allows these methods to optimize and explore the entire motion to accomplish a task, rather than seeking a local best action for the current state of the system, as is common in deep RL. In this context, it becomes easier to consider additional aspects of the movement, such as energy efficiency and overall smoothness, or even to work with environments that provide minimal or less structured feedback. However, classical robotics approaches lack the generalization and scalability inherent in modern deep RL methods. While previous attempts (Bahl et al., 2020) aimed to combine the benefits of both deep RL and classical trajectory-based methods, they only leveraged some of the capabilities. In this thesis, our focus is on creating more principled and efficient trust region methods to improve upon existing approaches, aiming for superior exploration, learning stability, and overall performance. Additionally, we investigate how well-known classical robotics approaches can be integrated into the domain of deep RL, aiming to synthesize the strengths of both paradigms.

1.2 Overview and Contributions

This thesis consolidates the findings of four previously published papers and organizes them in the chronological order of their publication, which is also their natural structure. Each main chapter discusses a different major project. The first main chapter introduces a novel trust region deep RL method based on a more rigorous mathematical framework. In the second main chapter, we explore the integration of these improved trust regions with more conventional robotics methods to combine the advantages of both fields. Finally, in the last main chapter, we present a method for implementing the proposed trust-region technique in an off-policy setting to improve sampling efficiency. The main contributions of this research are outlined below:

- In Chapter 3 we propose a new algorithm - *trust region projection layer* (TRPL). While trust region methods are a popular tool in RL as they yield robust policy updates in continuous and discrete action spaces, enforcing such trust regions in deep RL is difficult. Hence, many approaches, such as TRPO (Schulman et al., 2015a) and PPO (Schulman et al., 2017), are based on approximations. Due to those approximations, they violate the constraints or fail to find the optimal solution within the trust region. Moreover, they are difficult to implement, often lack sufficient exploration, and have been shown to depend on seemingly unrelated implementation choices (Engstrom et al., 2020; Henderson et al., 2018; Andrychowicz et al., 2020). Therefore, we propose differentiable neural network layers to enforce trust regions for deep Gaussian policies via closed-form projections. Unlike existing methods, those layers formalize trust regions for each state individually and can complement existing reinforcement learning algorithms. We derive trust region projections based on the KL divergence, the *Wasserstein L2-distance* (W2), and the Frobenius norm for Gaussian distributions. We empirically demonstrate that those projection layers achieve similar or better results than existing methods while being almost agnostic to specific implementation choices. This has previously been published in Otto et al. (2021).

- In Chapter 4, we use TRPL and combine it with advances from standard robotics, the new method is called *movement primitive-based planning policy* (MP3). By integrating *movement primitives* (MPs) into the deep RL framework, MP3 enables the generation of smooth trajectories throughout the whole learning process while effectively learning from sparse and non-Markovian rewards. Additionally, MP3 maintains the capability to adapt to changes in the environment during execution. Although many early successes in robot RL have been achieved by combining RL with MPs, these approaches are often limited to learning single stroke-based motions, lacking the ability to adapt to task variations or adjust motions during execution. Moreover, the high dimensionality of the MP parameters has so far hampered the effective use of deep RL methods. We introduce an episode-based RL method for the non-linear adaptation of MP parameters to different task variations and extend the approach to incorporate replanning strategies. This allows adaptation of the MP parameters throughout motion execution, addressing the lack of online motion adaptation in stochastic domains requiring feedback. We further make use of TRPL and show that exact trust regions are key for successful RL in high-dimensional action domains as for MPs. We compare our approach against state-of-the-art deep RL and RL with MPs methods and investigate different reward formulations - dense, sparse, and non-Markovian rewards. While step-based algorithms only work well for dense rewards, our approach also performs favorably on sparse and non-Markovian rewards. Additionally, the replanning strategies improve performance in domains requiring replanning and lower sample complexity. This work has previously been published in Otto et al. (2022) and has been extended in Otto et al. (2023) together with Hongyi Zhou, who contributed the replanning and task adaptation experiments for box pushing and table tennis tasks, along with additional ablation studies.

- In Chapter 5, we investigate the need for an explicit state-action-value function representation, which is typically used in existing off-policy RL algorithms and becomes problematic in high-dimensional action spaces. Existing algorithms often encounter challenges where they struggle with the curse of dimensionality, as maintaining a state-action-value function in such spaces becomes data-inefficient. We propose a novel off-policy trust region optimization approach, called Vlearn, that eliminates the requirement for an explicit state-action-value function. Instead, we demonstrate how to efficiently leverage just a state-value function as the critic, thus overcoming several limitations of existing methods. By doing so, Vlearn addresses the computational challenges posed by high-dimensional action spaces. Furthermore, Vlearn introduces an efficient approach to address the challenges associated with pure state-value function learning in the off-policy setting. This approach not only simplifies the implementation of off-policy PG algorithms but also leads to consistent and robust performance across various benchmark tasks. Specifically, by removing the need for a state-action-value function, Vlearn simplifies the learning process and allows for more efficient exploration and exploitation in complex environments. This has previously been published in Otto et al. (2024)

Chapter 2

Background and Related Work

The concept of *reinforcement learning* (RL) consists of two main components – the agent and the environment. The environment represents the task or world with which the agent interacts – this can be a specific task, ranging from specific tasks like video games or control problems to intricate systems such as power grids or data centers. The agent aims to learn how to translate (partial) observations of the environment into actions in order to maximize a reward signal. These actions affect not only the reward received but also the subsequent observations. The reward signal can be interpreted as feedback about the value of the current state of the world. This idea is illustrated in Figure 2.1.
The above setup distinguishes RL from supervised and unsupervised learning. Unlike in supervised learning, it is often not feasible to acquire examples of desired agent behavior that apply to all potential situations in which the agent is expected to operate. Especially for unseen situations, the agent must learn from its own experience based on interactions with the environment. Unsupervised learning is more closely related, but attempts to discover latent structure in the data. While discovering structure in the collected experience can be useful for RL as well, it does not address the RL problem of maximizing a reward. This chapter introduces the RL setting in more detail and takes a closer look at some of the most popular RL algorithms, as well as a particular class of algorithms called trust region methods.

2.1 Reinforcement Learning

This chapter is meant to give a small overview of the required background in RL and is loosely based on the structure and content in Sutton and Barto (2018). Generally, in RL we consider the problem of a policy search in an *Markov Decision Process* (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho_0, \gamma)$.

State Space $\mathcal{S}$ The state $s \in \mathcal{S}$ received from the environment is the complete description of the world, *i.e.*, there exists no latent information that is not part of the state. Hence, this is also called the *fully observed* case. This includes, for example, the positions and velocities of a robot's joints, as well as the locations, sizes, and weights of objects with which the robot interacts. When the agent receives only partial information

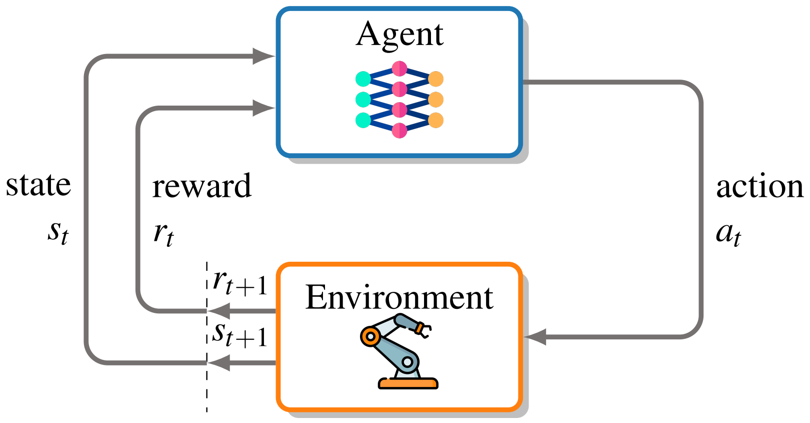


Figure 2.1: The diagram illustrates the interactions between an agent and its environment in a MDP. In an MDP, the agent in a given state takes actions that lead to a transition to a new state based on a stochastic process. The associated reward is then received by the agent. The key principle is the Markov property, where the future state and reward depend only on the current state and action. These interactions form the basis of the RL framework, guiding the agent's decision-making process as it seeks to maximize cumulative rewards over time. Adapted from Sutton and Barto (2018).

about the environment, this is typically referred to as observation $o \in \mathcal{O}$. This is relevant, for example, when learning from image-based observations, where velocity and direction of motion are not directly observable from a single image. Formally, *partially observed Markov decision processs* (POMDPs) capture this specific scenario where information may have been omitted or transformed from the full state s. For the purposes of this paper, however, we consider only the fully observed case, *i.e.* the MDP setting, and refer the reader to Kaelbling et al. (1998) for an in-depth look at POMDPs.

Action Space $\mathcal{A}$ The action space $\mathcal{A}$ encompasses the entire collection of permissible actions a available to the agent within a given environment. The action space is either discrete, such as in most video games or chess, or continuous, such as in most robot control problems. While deep RL uses neural networks as function approximators, this tends to help mostly with generalization for different types of state spaces. Depending on the type of action space, only some algorithms are directly applicable and most others would require (significant) changes to work on the respective other action space. For example, the popular method *deep Q-network* (DQN) (Mnih et al., 2015), which works well for discrete action spaces, would require discretizing the action space for continuous control problems and, thereby limiting the expressiveness the agent can achieve. For this work, however, we will focus primarily on continuous action spaces, since our primary application area is robotics, which almost always requires continuous control signals.

Policy π The policy is the strategy that selects actions based on the current state. Formally, it is a mapping $\pi : \mathcal{S} \rightarrow \mathcal{A}$ from states $s \in \mathcal{S}$ to the probabilities of selecting the possible actions $a \in \mathcal{A}$, hence it is often denoted as $\pi(a|s)$. While RL algorithms often use stochastic policies to facilitate exploration in the environment, the policy can also be deterministic, such as in *deterministic policy gradient* (DPG) (Silver et al., 2014).
In its simplest form, a policy can be thought of as a lookup table, but in deep RL the policy is most often represented by a neural network with a set of learnable parameters, which we denote as θ. In the latter case, the policy is formally written as $\pi_\theta(a|s)$ or $\pi(a|s;\theta)$. The structure of the network, or more specifically the probability distribution of the actions, depends on the action space. The two most common deep policy types are categorical policies for discrete action spaces and (diagonal) Gaussian policies for continuous action spaces. On a general note, the chosen distributions should offer a straightforward and ideally differentiable approach to sampling actions and calculating the log-likelihoods of actions, $\log \pi_\theta(a|s)$, as these characteristics are relevant for most RL algorithms.
The central task of RL algorithms is now to determine how to update the policy to maximize the rewards received.

Trajectory τ We can now bring states, actions, and policies together. In RL, we typically try to optimize the behavior of some policy over trajectories $\tau = (s_0, a_0, s_1, a_1, ...)$. Trajectories are a sequence of states and actions in the environment, where the first state s_0 is sampled from the start-state distribution $\rho_0 : \mathcal{S} \rightarrow [0,1]$. For each MDP, there exist transition probabilities $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$, which define the dynamics of the MDP and can be either deterministic or stochastic. They describe the probability of transitioning to state $s' \in \mathcal{S}$ given the current state $s \in \mathcal{S}$ and action, $a \in \mathcal{A}$

$$p(s'|s,a) = Pr\left\{s_{t+1} = s'|s_t = s, a_t = a\right\}, \tag{2.1}$$

that means

$$\int_{s' \in \mathcal{S}} p(s'|s,a)\, ds' = 1, \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}. \tag{2.2}$$

This equation reveals one important property of MDPs. The next state depends solely on the previous state and action, *i.e.*, the previous state must contain all prior information about the agent-environment interaction that are relevant to future changes. This is known as the Markov property.
By utilizing our environment dynamics and policy, we can now compute the probability of following a specific trajectory τ

$$p(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t), \tag{2.3}$$

where T is the length of the trajectory.

Reward $\mathcal{R}$ As mentioned earlier, our main goal is to maximize a reward signal $R_t \in \mathcal{R}$. Consequently, the reward returned by the environment is one of the most, if not the most, important parts in RL. Given the current state and the action chosen by the agent, we can evaluate the reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

$$r(s,a) = \mathbb{E}\left[R_t | s_t = s, a_t = a\right].$$

At times, the reward function is also expressed solely as a function of the current state

$$r(s) = \mathbb{E}\left[R_t | s_t = s\right],$$

or it may depend on the current state, action, and the next state

$$r(s,a,s') = \mathbb{E}\left[R_t | s_t = s, a_t = a, s_{t+1} = s'\right].$$

Throughout this paper, however, we will mostly use $r(s,a)$ or just r_t.
The goal of the agent is now to maximize the cumulative reward over a trajectory, also denoted as return G_t. One straightforward approach to achieving this is through the finite-horizon undiscounted return

$$G_t = \sum_{k=t}^{T} r_k. \tag{2.4}$$

In this setting, a terminal state occurs at time step T, indicating the end of the task, whether it be due to success or failure. This allows for a natural division of the agent-environment interaction into subsequences known as episodes. Tasks that follow this episodic structure are referred to as episodic tasks, such as the task of grasping an object. If there is no clear time boundary, there is no direct episode, and the task may potentially continue indefinitely ($T = \infty$) and could result in an infinitely large reward. To address this, we introduce the notion of discounting, accounting for the temporal distance at which rewards are acquired. Distant rewards contribute significantly less than immediate rewards. This leads to the formulation of the infinite-horizon discounted return, characterized by a discount factor $\gamma \in [0,1)$

$$G_t = \sum_{k=t}^{\infty} \gamma^{k-t} r_k. \tag{2.5}$$

Yet, in practice, it is common to impose an artificial time limit to allow for better comparison.

Value Estimates An important concept that is used by many RL algorithms is that of state-value $V(s)$ or state-action-value $Q(s,a)$ functions. They allow us to estimate the value of a state (and of performing a given action) by specifying the excepted future return. For the state-value function, we typically consider starting in state $s \in \mathcal{S}$ and then following a policy π

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[G_t | s_t = s\right] = \mathbb{E}_{\pi}\left[\sum_{k=t}^{\infty} \gamma^{k-t} R_k \middle| s_t = s\right], \quad \forall s \in \mathcal{S}. \tag{2.6}$$

Equivalently, the state-action-value function can be written as

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[G_t | s_t = s, a_t = a\right] = \mathbb{E}_{\pi}\left[\sum_{k=t}^{\infty} \gamma^{k-t} R_k \middle| s_t = s, a_t = a\right], \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}. \tag{2.7}$$

The main difference to the state-value function is that we first take action $a \in \mathcal{A}$ in state s and then follow the policy afterward. This action is not necessarily the one that would be chosen by the policy but can be any action.

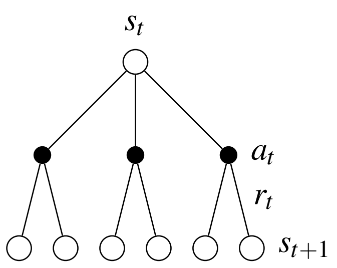
When learning state-value and state-action-value functions, it is often beneficial to use their recursive formulations. For the state-action-value function, we have

$$\begin{aligned}
Q^{\pi}(s,a) &= \mathbb{E}_{\pi}\left[G_t | s_t = s, a_t = a\right] \\
&= \mathbb{E}_{\pi}\left[R_t + \gamma G_{t+1} | s_t = s, a_t = a\right] \\
&= \int_{s'} p(s'|s,a)\left[r(s,a) + \gamma \mathbb{E}_{\pi}\left[G_{t+1} | s_{t+1} = s'\right]\right] ds' \\
&= \int_{s'} p(s'|s,a)\left[r(s,a) + \gamma \int_{a'} \pi(a'|s') \mathbb{E}_{\pi}\left[G_{t+1} | s_{t+1} = s', a_{t+1} = a'\right] da'\right] ds' \\
&= \int_{s'} p(s'|s,a)\left[r(s,a) + \gamma \mathbb{E}_{\pi}\left[Q^{\pi}(s',a') | s_{t+1} = s'\right]\right] ds' \\
&= \mathbb{E}_{s' \sim \mathcal{P}}\left[r(s,a) + \gamma \mathbb{E}_{\pi}\left[Q^{\pi}(s',a') | s_{t+1} = s'\right] | s_t = s, a_t = a\right]. &(2.8) \\
&= \mathbb{E}_{s' \sim \mathcal{P}}\left[r(s,a) + \gamma V^{\pi}(s') | s_t = s, a_t = a\right]. &(2.9)
\end{aligned}$$

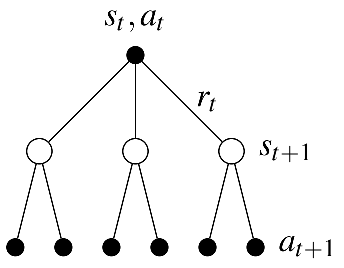
The recursive formulations in Equations (2.8) and (2.9) are also known as Bellman equations (Bellman, 1957). Furthermore, in Equation (2.9) we can see an important connection between the state-value and state-action-value functions. It is often used to estimate the state-action value while learning only an explicit model of the state-value function. Similarly, we can formulate the Bellman equation for the state-value function

$$\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}\left[G_t | s_t = s\right] \\
&= \int_{a} \pi(a|s) \int_{s'} p(s'|s,a)\left[r(s,a) + \gamma \mathbb{E}_{\pi}\left[G_{t+1} | s_{t+1} = s'\right]\right] ds'\, da \\
&= \mathbb{E}_{\pi}\left[r(s,a) + \gamma V^{\pi}(s') \mid s_t = s\right]. &(2.10)
\end{aligned}$$

Effectively, the Bellman equation illustrates the correlation between the value of a given state (and action) and the values of its potential successor states, which can be thought



(a) State-value function $V^{\pi}(s)$ backup.



(b) State-action-value function $Q^{\pi}(s,a)$ backup.

Figure 2.2: Each open circle in the diagrams represents a state, while each solid circle corresponds to a state-action pair. Starting from the root node, the agent can select actions based on its policy for a given state s for the state-value function or has to take the given action a for the state-action-value function. Based on the environment dynamics, this leads to various subsequent states s_{t+1} accompanied by a corresponding reward r. The Bellman equation (Equations (2.8) and (2.9)) now combines all possible paths, weighting them according to their probability. Based on Sutton and Barto (2018).

of as looking ahead, as shown in Figure 2.2.
As mentioned in the policy section, our goal is to find a policy that maximizes the cumulative reward. If this policy is better than or equal to all other policies, it is called the optimal policy π^*. While there can be more than one such policy, they all share the same optimal state-value and state-action-value function

$$V^*(s) = \max_{\pi} \mathbb{E}_{\pi}\left[G_t | s_t = s\right] = \max_{\pi} V^{\pi}(s)$$
$$Q^*(s,a) = \max_{\pi} \mathbb{E}_{\pi}\left[G_t | s_t = s, a_t = a\right] = \max_{\pi} Q^{\pi}(s,a).$$

Furthermore, this special case gives rise to the Bellman optimality equation based on Equations (2.8) and (2.10)

$$V^*(s) = \max_{a} \mathbb{E}_{s' \sim \mathcal{P}}\left[r(s,a) + \gamma V^*(s')\right], \tag{2.11}$$

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{P}}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a')\right]. \tag{2.12}$$

Note that we are not following a specified policy here, but rather choosing the best action. This further means that any policy that acts greedily based on the optimal state-value function qualifies as an optimal policy.
Lastly, for some RL methods, it is not necessary to provide a specific state-value estimate, but much rather the relative advantage an action has over actions. This concept is represented by the advantage function

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s). \tag{2.13}$$

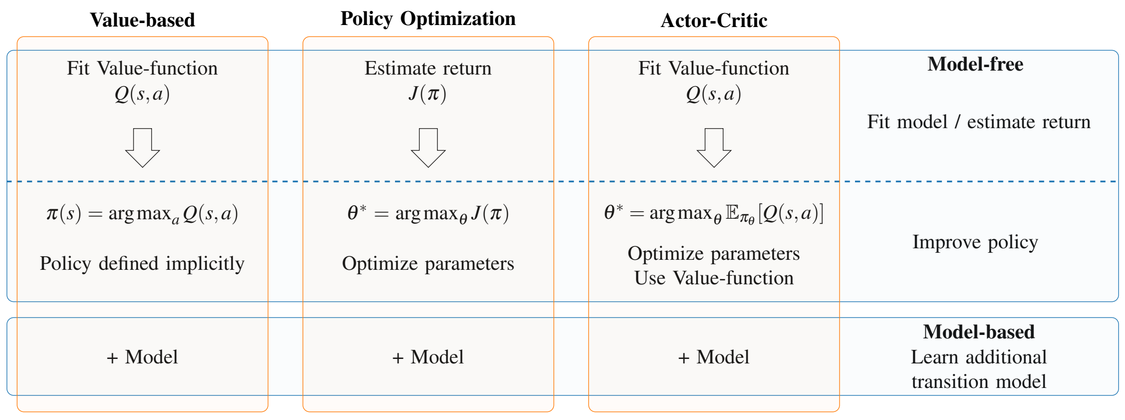

Value-based
Policy Optimization
Actor-Critic
Fit Value-function $Q(s,a)$
Estimate return $J(\pi)$
Fit Value-function $Q(s,a)$
Model-free
Fit model / estimate return
$\pi(s) = \arg\max_a Q(s,a)$
Policy defined implicitly
$\theta^* = \arg\max_\theta J(\pi)$
Optimize parameters
$\theta^* = \arg\max_\theta \mathbb{E}_{\pi_\theta}[Q(s,a)]$
Optimize parameters
Use Value-function
Improve policy
+ Model
+ Model
+ Model
Model-based
Learn additional transition model

Figure 2.3: Taxonomy of the RL algorithm landscape. We generally classify RL algorithms into two main categories: model-based and model-free methods. Model-based methods utilize a world model during training, while model-free methods do not have access to such a model. Furthermore, algorithms can be categorized based on how they learn the policy. Value-based methods primarily learn a policy implicitly through a state-action-value function, whereas policy-based methods aim to directly optimize the objective outlined in Equation (2.14). Actor-critic methods represent an intersection of both approaches, incorporating elements of both value-based and policy-based methods.

It describes how much better or worse a particular action a is compared to the average performance for that state s under the policy π. In practice, *policy gradient* (PG) methods (Sutton et al., 1999a) are a class of algorithms that rely heavily on this concept.
Similar to the policy, in deep RL the state-value and the state-action-value functions are typically represented by a neural network. Hence, we write $Q_\phi^\pi(s,a)$ or $V_\phi^\pi(s)$, where ϕ are the learnable parameters.

Objective $J(\pi)$ The general RL objective of finding a policy that maximizes the expected return can now be formalized. This can be achieved by combining the probability of a trajectory (Equation (2.3)) with its reward (Equations (2.4) and (2.5)) to obtain the expected return

$$J(\pi) = \int_\tau p(\tau|\pi) G_0 \, d\tau = \mathbb{E}_\pi [G_0] \tag{2.14}$$

and the corresponding optimal policy

$$\pi^* = \arg\max_\pi J(\pi).$$

2.1.1 Learning the Policy

The main goal of any RL algorithm is now to optimize the above objective and find the optimal policy. As shown in Figure 2.3, we typically consider two major groups of RL algorithms – *model-free* and *model-based*. The main difference between the two is whether the agent has access to or learns a model of the environment dynamics $p(s_{t+1}|s_t, a_t)$. By using this model, the agent can plan and explore different action choices without interacting with the real environment, which can be costly. One of the most straightforward ways to achieve this is by leveraging dynamic programming (Bellman, 1966). However, dynamic programming is typically avoided in the context of RL due to its high computational cost and the need for a (near) perfect dynamics model. Nevertheless, the basic principles of dynamic programming often inspire the design of more practical methods within RL that aim to reduce computational cost and dependence on precise models. For instance, in board games, where the dynamics are known, model-based RL has been highly successful (Silver et al., 2018, 2017), but in many other applications, the ground truth dynamics are still difficult to obtain. In such cases, it is necessary to learn a model from interactions with the environment, which may lead to overfitting the agent to this specific learned model. Consequently, this can result in suboptimal performance when applied to the actual target application. In contrast, model-free approaches eliminate the need for a dynamics model and instead learn directly from interactions with the environment. Because of their independence from explicit models, we will focus on model-free approaches in this work.

Value-based Methods Another dimension in classifying RL methods is based on how they learn the policy (refer to Figure 2.3). *Value-based* methods do not make use of an explicit policy parametrization π_θ, but instead just learn a state-action-value function $Q(s,a)$. This way, the current best policy is implicitly defined as $\pi(s) = \arg\max_a Q(s,a)$. For exploration, a common strategy in value-based methods is ε-greedy. This strategy selects a random action with probability ε and otherwise chooses the best action according to the state-action-value function

$$\pi_{\text{greedy}}(a|s) = \begin{cases} \varepsilon/|\mathcal{A}| + 1 - \varepsilon & \text{if } a = \arg\max_{a'} Q(s,a') \\ \varepsilon/|\mathcal{A}| & \text{otherwise} \end{cases}, \tag{2.15}$$

where $|\mathcal{A}|$ is the number of possible actions in the environment. This is also the method of choice in one of the most commonly used value-based methods – Q-learning (Watkins and Dayan, 1992).

Training the state-action value function usually involves temporal difference learning (Sutton, 1988; Watkins and Dayan, 1992), with updates grounded in the Bellman equation (see Equation (2.8))

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1}) - Q(s_t,a_t) \right], \tag{2.16}$$

where α is the learning rate. This directly approximates the optimal state-action-value function $Q^*(s,a)$ from Equation (2.12) and can guarantee convergence in the tabular case.

This learning approach, known as *temporal-difference* (TD)-learning (Sutton, 1988), can be applied to state-value or state-action-value functions, with Q-learning representing a specific variant focused on the latter. TD-learning involves taking a one-step estimate of the value function $r_t + \gamma V(s_{t+1})$ and computing the resulting TD error

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t). \tag{2.17}$$

This error quantifies the disparity between the current value estimate and a one-step estimate of the returns, and it can be further extended to n-step or λ-returns (Sutton and Barto, 2018). This is the same principle as in the Q-learning update step above. Unlike dynamic programming, Equation (2.16) directly shows that Q-learning is inherently model-free, as it does not necessitate knowledge of the environment dynamics. Simultaneously, improving the Q-function estimate directly results in maximizing the general RL objective from Equation (2.14).
Furthermore, Q-learning is classified as an *off-policy* algorithm, as it improves a policy distinct from the one employed to generate the data. The latter is referred to as the behavior policy. In standard Q-learning, the ε-greedy policy serves as the behavior policy, exploring and generating data for the update. The policy targeted for improvement is defined by the Q-function $Q(s,a)$ where the action a_{t+1} for state s_{t+1} is chosen based on the highest Q-value, as shown in Equation (2.16),
Alternatively, we can select the action for state s_{t+1} based on the same policy, we use for exploration

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\Big[\big(r_t + \gamma Q(s_{t+1}, \pi_{\text{greedy}}(a_{t+1}|s_{t+1})\big) - Q(s_t,a_t)\Big]. \tag{2.18}$$

This method is known as SARSA (Rummery and Niranjan, 1994) and is considered to be an *on-policy* approach since it uses the same policy for the exploration and the update. However, Q-learning-based approaches are typically favored due to their enhanced sample efficiency, achieved through mechanisms like replay buffers (Lin, 1992). For on-policy learning, policy optimization methods, which we discuss next, are a more common choice.

Policy Optimization Unlike value-based methods, policy optimization methods represent a policy π_θ directly with parameters θ and optimize these parameters. This eliminates the need for an explicit representation of a value function in the action selection process. While we will see later that a value function can still aid in learning the policy parameters, it is not mandatory. For this reason, policy optimization is considered a more direct approach for solving RL problems compared to value-based methods. This is especially beneficial when learning a policy is simpler than learning the corresponding value function for a given task. Another strength of policy optimization methods lies in their ability to learn optimal stochastic policies, a feature crucial, *e.g.*, for imperfect information games. Additionally, they exhibit better theoretical properties, as the action probabilities change smoothly based on the parameters. In contrast, small changes in state-action values for value-based methods can suddenly favor a different action, leading to significant shifts in the behavior of the ε-greedy policy.

To optimize the policy, gradient ascent is typically employed on the objective $J(\pi_\theta)$ from Equation (2.14), hence why these methods are also known as PG methods

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\pi_\theta).$$

This requires the assumption that the policy is differentiable with respect to its parameters. Moreover, while Monte Carlo estimates can be used to approximate the expectation of $J(\pi_\theta)$, this is not directly feasible for its gradients. However, the likelihood-ratio trick $\nabla_\theta p(\tau|\pi) = p(\tau|\pi)\nabla_\theta \log p(\tau|\pi)$ enables the rewriting of the gradient as an expectation

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathbb{E}_\pi \left[G_0\right] \\
&= \int_\tau \nabla_\theta p(\tau|\pi) G_0 \, d\tau \\
&= \int_\tau p(\tau|\pi) \nabla_\theta \log p(\tau|\pi) G_0 \, d\tau \\
&= \mathbb{E}_\pi \left[\nabla_\theta \log p(\tau|\pi) G_0\right].
\end{aligned}$$

This expectation can now be approximated by computing a Monte Carlo estimate of the gradient of the trajectory probabilities $p(\tau|\pi)$ and the returns. Yet, Equation (2.3) reveals that this requires knowledge of the prior state distribution and the environment dynamics, which is not accessible in the model-free setting. Nonetheless, the gradient can be simplified by extending the trajectory probability

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \mathbb{E}_\pi \left[\left(\nabla_\theta \log \rho_0(s_0) + \sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) + \sum_{t=0}^{\infty} \nabla_\theta \log p(s_{t+1}|s_t, a_t) \right) G_0 \right] \\
&= \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t \right] \\
&= \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) Q^\pi(s_t, a_t) \right]
\end{aligned}$$

In the second step, we eliminate the dependency on the prior state distribution and the environment dynamics since they do not rely on the policy parameters and consequently, their gradient is always 0. Moreover, we can replace the full trajectory returns G_0 starting at the beginning of the episode $t = 0$ with the returns of the current time step G_t. This simply follows from the idea that past rewards cannot be altered by selecting a different action in the future. Alternatively, in the last step, we can replace the rewards-to-come G_t with an estimate of the state-action-value function $Q(s_t, a_t)$, which is equivalent. This

simplification now allows the gradient to be approximated using real trajectory samples to estimate the returns G_t. The general approach we showed here is also known as REINFORCE (Williams, 1992).
From a practical standpoint, direct use of REINFORCE is typically undesirable due to its high variance. To mitigate this, subtracting a baseline $b(s_t)$ from the returns or state-action-value function can be used to reduce the variance

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(G_t - b(s_t) \right) \right].$$

The baseline can take various forms, such as a function, constant, or even a random variable, but it must be independent of the action a to avoid introducing bias. Ensuring the baseline's independence from the chosen action maintains the unbiased nature of the estimate since we just subtract zero

$$\mathbb{E}_\pi \left[\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t) \right] = b(s_t) \nabla_\theta \mathbb{E}_\pi \left[\log \pi_\theta(a_t|s_t) \right] = b(s_t) \nabla_\theta 1 = 0$$

Besides the average or expected reward, a straightforward choice for this baseline is the expected return, *i.e.*, the state-value function $V^\pi(s_t)$ from Equation (2.10). Following the definition in Equation (2.13), we can further replace the Q-function and V-function with the advantage function

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(Q^\pi(s_t, a_t) - V^\pi(s_t) \right) \right] \\ &= \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) A^\pi(s_t, a_t) \right] \end{aligned} \tag{2.19}$$

Since REINFORCE is a Monte Carlo method, computing an estimate of the advantage function can also be done using Monte Carlo, for instance, with n-step returns

$$\hat{A}^\pi(s_t, a_t) = \sum_{k=t}^{t+n-1} \gamma^{k-t} r_k + \gamma^n V^\pi(s_{t+n}) - V^\pi(s_t).$$

Here, choosing an optimal value for n can be challenging, as a small n yields lower variance but potentially high bias, whereas a large n has higher variance but lower bias. To avoid committing to a specific n, an average of all n-step returns can be computed, allowing for a direct trade-off between bias and variance. This commonly used approach to estimate the advantage function is known as *generalized advantage estimation* (GAE) (Schulman et al., 2015b):

$$\hat{A}^\pi_{\text{GAE}}(s_t, a_t) = \sum_{k=t}^{\infty} (\gamma\lambda)^{k-t} \big(r_k + \gamma V^\pi(s_{k+1}) - V^\pi(s_k) \big),$$

where $\lambda \in [0,1]$ is a hyperparameter. Setting $\lambda = 0$ corresponds to one-step advantages (similar to the standard advantage), and for $\lambda \to 1$, GAE includes more steps in the computation, introducing bias but potentially reducing variance.

A major drawback associated with on-policy methods is their inability to effectively use past trajectories, resulting in significant sampling inefficiency. To overcome this limitation and exploit the value of past trajectories, a common approach is to incorporate importance sampling

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) A^\pi(s_t,a_t)\right] \\ &= \mathbb{E}_{\pi_{\text{old}}}\left[\sum_{t=0}^{\infty} \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \nabla_\theta \log \pi_\theta(a_t|s_t) A^\pi(s_t,a_t)\right], \end{aligned} \tag{2.20}$$

where π_{old} represents a potentially distinct data-generating behavior policy, akin to scenarios encountered in off-policy or value-based methods. This integration of importance sampling enables a more nuanced and efficient exploration of the learning space by incorporating information from prior trajectories. This mitigates the sample inefficiency associated with on-policy methods, improving the overall learning process.

Actor-Critic Actor-critic methods serve as a hybrid approach, combining aspects of both value-based and policy optimization methodologies. In this paradigm, the actor guides action selection and thus directs exploration, while the critic provides valuable feedback to enhance the learning process. In contrast to relying on a baseline estimate and Monte Carlo samples, we consider actor-critic methods to directly learn a parametrized state-action value function $Q^\pi(s,a)$ alongside the policy. This circumvents introducing variance but introduces potential bias due to a potentially imperfect critic estimate. Furthermore, the differentiable Q-function facilitates direct policy updates, especially in the deep learning setting, which we will show in the next section. Additionally, actor-critic methods commonly operate in off-policy scenarios, incorporating experience replay or other replay buffer mechanisms (Lin, 1992). This enables algorithms to learn from past trajectories, improving their sample efficiency. However, it should be noted, that the distinction between actor-critic methods and direct policy optimization is not universally agreed upon. For instance, REINFORCE with a state-value function baseline can be considered part of the actor-critic framework, often termed ”advantage actor-critic” (see Equation (2.19)). This holds especially true for more modern RL approaches, which will be discussed in the subsequent section. In this work, our focus lies predominantly on actor-critic algorithms that adhere to the aforementioned properties, often referred to as ”Q-actor-critic” approaches. These methods directly learn a

Q-function (here without a baseline) and update the policy as

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) Q_\phi^\pi(s_t, a_t) \right] \tag{2.21}$$

or *TD actor-critic* approaches, which approximate the advantage with the 1-step TD error (see Equation (2.17))

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(r_t + \gamma V_\phi^\pi(s_{t+1}) - V_\phi^\pi(s_t) \right) \right],$$

where ϕ are the learnable critic parameters. In summary, actor-critic methods offer a flexible and powerful framework for RL by harnessing the strengths of both value-based and policy optimization approaches.

2.1.2 Deep Reinforcement Learning Algorithm Landscape

Conventional RL methods demonstrate effectiveness in specific scenarios, yet their shortcomings become evident in environments characterized by complex state spaces, intricate dynamics, or the need for abstract feature extraction. In response to these challenges, deep RL emerges as a solution, leveraging the representational power of neural networks to capture intricate patterns and nuances within the environment. With the foundation of RL methods established in the previous section, we delve into a more detailed exploration of contemporary RL approaches, particularly in the field of deep RL.
First, we revisit standard Q-learning (see Equation (2.16)), for which DQN (Mnih et al., 2013) directly transfers its idea to the deep learning setting. The optimization objective becomes

$$J(Q_\phi) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}} \left[\left(r_t + \gamma \max_{a_{t+1}} Q_{\bar{\phi}}(s_{t+1}, a_{t+1}) - Q_\phi(s,a) \right)^2 \right]. \tag{2.22}$$

The transition from the tabular case in standard Q-learning (see Equation (2.16)) to DQN introduces new challenges, such as the adoption of mean squared error instead of direct Q-function updates. Moreover, the nonlinear function approximation adds complexity, with the above equation involving two significant changes.
To increase the sample efficiency in RL, modern off-policy algorithms, including DQN, leverage a replay buffer (Lin, 1992). Past transitions (s_t, a_t, r_t, s_{t+1}), generated by the exploration or behavior policy, are stored in the replay memory $\mathcal{D}$ and are subsequently sampled for each Q-function update. While uniform sampling from the replay buffer is the most common approach, more advanced techniques, such as prioritized experience replay (Schaul et al., 2016), have also been proposed. Random sampling from the replay buffer helps reduce variance by mitigating the auto-correlation present in the orig-

inal online learning setting, where consecutive samples are used. Additionally, in online learning, where the current Q-function parameters determine the next samples, experience replay helps average the behavior distribution over multiple previous states. This leads to smoother learning and prevents oscillations or divergence in the parameters. The second major improvement is the use of target networks. Instead of optimizing towards the one-step estimate using the current Q-function with parameters ϕ, DQN employs a second (target) network with parameters $\bar{\phi}$ to estimate the Q-value of the next state, as shown in Equation (2.22). This target network is either periodically updated by copying the current Q-function parameters or through polyak averaging $\bar{\phi} \leftarrow \tau\phi + (1-\tau)\bar{\phi}$. This strategy prevents substantial changes in the optimization target, *i.e.* the one-step estimate, thereby averting short-term oscillations and contributing to the overall stability of the training process. Both of these changes align the problem more closely with a supervised learning problem.

While the original DQN achieved considerable success and attention, subsequent iterations aimed to address several of its shortcomings. Some focused on mitigating overestimation bias (Van Hasselt et al., 2016), while others explored the separation of state values and state-dependent action advantages (Wang et al., 2016). Leveraging distributional critics (Bellemare et al., 2017) and combining multiple previous improvements (Hessel et al., 2018) further refined the landscape. Yet, while it is possible to directly transfer Q-learning to continuous action spaces (Kalashnikov et al., 2018), it is computationally expensive as each action selection requires solving an optimization problem. Although discretization is an option (Seyde et al., 2022), it is inherently limiting when no appropriate discretization is available. For continuous problems, modern off-policy actor-critic methods have established themselves as a superior option. They optimize a policy network that generates continuous action values based on the Q-function estimator (Degris et al., 2012; Zhang et al., 2019). This approach accommodates both deterministic (Lillicrap et al., 2015; Fujimoto et al., 2018) and probabilistic policy distributions (Haarnoja et al., 2018; Abdolmaleki et al., 2018b).

In particular, *soft actor critic* (SAC) (Haarnoja et al., 2018) stands out among these methods, aiming to optimize its policy with the objective

$$J(\pi_\theta) = \mathbb{E}_{s_t \sim \mathcal{D}, a_t \sim \pi_\theta}\left[\alpha \log \pi_\theta\left(a_t | s_t\right) - Q_\phi(s_t, a_t)\right], \tag{2.23}$$

where α is either a fixed hyperparameter or automatically adjusted during training. In general, this objective is very much related to the one used for the PG of standard actor-critic methods (see Equation (2.21)), however, it additionally adds a weighted entropy term $\mathbb{E}_{s_t \sim \mathcal{D}, a_t \sim \pi_\theta}\left[\alpha \log \pi_\theta\left(a_t | s_t\right)\right]$. While this objective can be optimized using the likelihood ratio gradient (Williams, 1992), this does not fully leverage all available information. For SAC and related methods, instead of effectively using the Q-function as a weight for the gradient, we can directly exploit the differentiability of the Q-function resulting in a lower variance. To do this, we need to apply the reparametrization trick

(Kingma and Welling, 2014) to Equation (2.23), which yields

$$J(\pi_\theta) = \mathbb{E}_{s_t \sim \mathcal{D}, \varepsilon_t \sim \mathcal{N}} \left[\alpha \log \pi_\theta \left(f_\theta(\varepsilon_t; s_t) | s_t \right) - Q_\phi(s_t, f_\theta(\varepsilon_t; s_t) | s_t)) \right],$$

where the policy is implicitly defined by the transformation $a_t = f_\theta(\varepsilon_t; s_t)$ with the noise vector ε_t, which is typically sampled from a spherical Gaussian. This approach can be seen as an extension of methods used for deterministic policies (Lillicrap et al., 2015; Fujimoto et al., 2018) to tractable stochastic policies. Building upon Equation (2.22), the entropy is incorporated into the Q-function network objective, yielding the training formulation

$$J(Q_\phi) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\phi(s, a) - \left(r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p} \left[V_{\bar{\phi}}(s_{t+1}) \right] \right) \right)^2 \right],$$

where

$$V_{\bar{\phi}}(s_{t+1}) = \mathbb{E}_{a \sim \pi_\theta} \left[Q_{\bar{\phi}}(s_{t+1}, a_{t+1}) - \alpha \log \pi_\theta(a_{t+1} | s_{t+1}) \right].$$

Unlike DQN, which relies on an indirect policy via the Q-function, SAC features a parameterized policy and therefore selects its actions a_{t+1} based on that policy.
Nevertheless, similar to the challenges encountered in training a state-action-value function estimator in DQN, the continuous setting presents similar difficulties. Here, some strategies directly transfer from the discrete to the continuous settings (Fujimoto et al., 2018). Additionally, common techniques like ensembles can be employed to address some of these challenges and improve overall performance (Chen et al., 2021).

2.2 Trust Region Methods

In the on-policy setting, trust region methods (Schulman et al., 2017, 2015a; Akrour et al., 2019) have proven effective in stabilizing the PG. These methods constrain the magnitude of policy updates, reducing the probability of extreme behavioral shifts. While Kakade and Langford (2002) initially explored mixing policies, today's approaches predominantly use *Kullback-Leibler* (KL) trust regions to bound the updates (Schulman et al., 2015a, 2017; Akrour et al., 2019). This concept of KL trust regions originates in natural PG approaches (Peters et al., 2005; Kakade, 2001; Bagnell and Schneider, 2003), which may even provide analytical solutions to the problem. In contrast to earlier works, Peters et al. (2010) introduces a maximal step size constraint instead of a small fixed step size for the trust region. They provide a solution based on the dual of the now constraint optimization problem, which has also been transferred to the model-based setting (Abdolmaleki et al., 2015). Still, these approaches are not straightforward to extend to highly nonlinear policies, such as neural networks. In an attempt to transfer those ideas to deep learning, *trust region policy optimization* (TRPO) (Schulman et al., 2015a) approximates the KL constraint, along with a backtracking line search to enforce a hard KL constraint.

TRPO starts from formulating a constraint objective based on the KL divergence between the probability distributions of the policy, namely the trust region, which was also used by Peters et al. (2010). This KL divergence constraint is incorporated into the standard PG objective from Equation (2.20) as follows

$$J(\pi_\theta) = \mathbb{E}_{\pi_{\text{old}}}\left[\sum_{t=0}^{\infty} \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A^\pi(s_t,a_t)\right] \quad \text{s.t.} \quad \mathbb{E}_{\pi_{\text{old}}}\left[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot|s_t)),\pi_\theta(\cdot|s_t))\right] \leq \varepsilon,$$

where ε is a hyperparameter dictating the size of the trust region. While the ideal scenario involves enforcing the constraint for every possible state, *i.e.* constraining the maximum change of the KL, practically achieving this across the entire state space proves challenging due to its size. As a consequence, TRPO adopts the expected KL divergence as a heuristic approximation. As previously mentioned, natural PG can theoretically solve the above trust region problem in closed form, which is however expensive to compute exactly. Therefore, it is common to simplify the objective by using Taylor approximations, which is also what TRPO does

$$\theta_{k+1} = \arg\max_\theta g^T(\theta - \theta_k) \quad \text{s.t.} \quad \frac{1}{2}(\theta-\theta_k)^T H(\theta-\theta_k) \leq \varepsilon,$$

where g is the PG, H is the Hessian or Fisher Information Matrix, and $\theta_k = \theta_{\text{old}}$. This approximation can then be analytically solved using Lagrangians

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g,$$

where $\alpha \in (0,1)$ is the backtracking coefficient. Theoretically, $\alpha = 1$ allows us to update the policy according to the natural PG (Kakade, 2001), however, this is only the solution to an approximation. Consequently, a backtracking line search, *i.e.* verifying the constraint satisfaction for decreasing values of α, can correct the update size accordingly to avoid violating the trust region. Lastly, since computing and storing the inverse Hessian H^{-1} is expensive and ultimately only the vector product $H^{-1}g$ is required to compute the gradient step, TRPO employs conjugate gradients to solve $Hx = g$ for $x = H^{-1}g$ to approximate this product.

While the original TRPO paper provides proof for the method's convergence guarantees, these assurances do not extend to scenarios involving non-linear function approximation. Furthermore, the final algorithm faces scalability issues with larger networks, primarily due to its reliance on second-order optimization methods. To address these concerns, Schulman et al. (2017) introduced *proximal policy optimization* (PPO), which adopts a different approach by avoiding directly enforcing the KL trust region. Instead, PPO clips the probability ratio in the importance sampling objective

$$J(\pi_\theta) = \mathbb{E}_{\pi_{\theta_k}}\left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a),\ \operatorname{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\varepsilon, 1+\varepsilon\right)A^{\pi_{\theta_k}}(s,a)\right)\right].$$

This modification enables the utilization of efficient first-order optimization methods while ensuring robust training. Intuitively, PPO restrains the impact of actions that become significantly more or less likely, lowering the benefit to the new policy of deviating excessively from the old policy. In this context, the parameter ε in PPO serves a role similar to the trust region constraint in TRPO. However, recent studies (Engstrom et al., 2020; Andrychowicz et al., 2020) showed that implementation choices are essential for achieving state-of-the-art results with PPO. Notably, *Code-level optimizations*, such as reward scaling, as well as value function, observation, reward, and gradient clipping, have been shown to compensate for the removal of core components of the algorithm, such as the clipping of the probability ratio. Additionally, it is important to acknowledge that PPO heavily relies on its exploration behavior and may encounter challenges getting trapped in local optima (Wang et al., 2019).

Besides, these widely known methods, various approaches to this constraint optimization problem have emerged in the literature. Tangkaratt et al. (2018) proposed an alternative by employing a closed-form solution based on the method of Lagrangian multipliers. However, there are potential limitations to the representational power of this method because it requires a quadratic parametrization of the Q-function. Another avenue explored by Pajarinen et al. (2019) extends earlier approaches that utilized compatible value function approximations (Kakade, 2001; Peters et al., 2003, 2005) to neural networks to enforce trust regions. In the reinforcement learning as inference paradigm (Levine, 2018), Abdolmaleki et al. (2018a) introduced *maximum aposteriori policy optimization* (MPO), which exhibits a close connection to KL trust region approaches and has been further extended to an on-policy version using similar optimization schemes and constraints Song et al. (2020). Unlike previous approaches, *projected approximate policy iteration* (PAPI) (Akrour et al., 2019) proposes a projection-based solution to implement KL trust regions. Instead of directly solving the constraint problem, PAPI projects an intermediate policy that already satisfies the trust region constraint onto the constraint bounds, thereby maximizing the size of the update step. However, it is worth noting that PAPI relies on other trust region methods, such as TRPO and PPO, to generate this intermediary policy and cannot operate as a standalone solution. Additionally, the projection is performed after the policy optimization, potentially leading to suboptimal policies during training. Considering the computational complexity, both TRPO and PAPI simplify the constraint by using the expected KL divergence, while a constraint per state would be more desirable.

In the off-policy setting, trust region methods have also received some attention. Past efforts have focused on training the value function with off-policy data (Gu et al., 2016) or extended TRPO (Schulman et al., 2015a) to the off-policy setting (Nachum et al., 2018; Meng et al., 2022). Wang et al. (2017) employ a more standard approach by combining off-policy trust regions with additional advancements, such as Retrace (Munos et al., 2016) and truncated importance sampling. Peng et al. (2019) take it one step further and propose a novel perspective by segregating policy and value-function learning into distinct supervised learning steps. Yet, in most cases, standard off-policy trust region methods lag behind modern actor-critic approaches (Haarnoja et al., 2018; Fujimoto et al., 2018) and cannot achieve competitive performance. Notably, MPO (Abdolmaleki et al., 2018b) stands out as an off-policy trust region method, albeit in a non-classical sense. Its formulation, based on the *expectation maximization* (EM) algorithm, provides more flexibility, with trust regions resembling the optimization of a parametric E-step without an explicit M-step.

While in the on-policy setting trust region methods often simplify the learning process by estimating the less complex state-value function, in the off-policy settings, the majority of methods still rely on estimating state-action-value functions to efficiently learn from replay buffer data. Approaches that exclusively utilize state-value functions must resort to importance sampling to address the distributional discrepancy between target and behavior policies. This typically involves reweighting and truncating the importance of the Bellman targets in n-step returns (Espeholt et al., 2018; Luo et al., 2020), a technique that has also proven beneficial for learning state-action-value functions (Munos et al., 2016). Nevertheless, this form of off-policy correction is computationally expensive as it necessitates storing and processing full or partial trajectories. Moreover, the truncation technique in importance weight calculations is known to introduce significant bias. Consequently, implementing trajectory-based target estimators could exacerbate the already pronounced issue of bias propagation. On the other hand, prior work has demonstrated that using importance sampling on the targets is suboptimal, favoring the importance weighting of the entire Bellman error, similar to the PG approach (Mahmood et al., 2014; Dann et al., 2014).

In addition to the KL divergence as a trust region measure, Pacchiano et al. (2020) employ the Wasserstein distance to constrain the agent behavior. For bandits, Richemond and Maginnis (2017) also propose an algorithm with Wasserstein-based trust regions, while Song and Zhao (2020) focus on solving the trust region problem for distributional policies, employing both KL and Wasserstein-based trust regions for discrete action spaces.

Lastly, instead of solely controlling the step size using the trust region, Abdolmaleki et al. (2015) introduced the idea of explicitly managing the decrease in entropy during the optimization process. This concept was later extended to deep RL by Pajarinen et al. (2019) and Akrour et al. (2019). They employ either an exponential or linear decay of the minimum entropy bound during policy optimization to control the exploration process and escape local optima.

2.3 Policy Parameterization

Until now, the discussed concepts and methods could be classified as *step-based RL* (SRL), where the policy interacts directly with the environment through raw actions based on the current state. These predicted actions typically translate into specific positions, velocities, or torques. They represent, for example, the specific control signals that will be applied to the robot's actuators, influencing its movements and behavior. For learning such policies, PG methods (Sutton et al., 1999b; Schulman et al., 2015a, 2017) are often employed due to their ease of implementation and capability to yield high-quality policies. Typically combined with some form of trust region to stabilize the on-policy update due to the large variance of the gradients. In particular, PPO has shown robust and efficient performance even on large-scale problems such as OpenAI Five (Berner et al., 2019) and GPT-4 (Achiam et al., 2023), but as mentioned above, it depends heavily on implementation details (Engstrom et al., 2020) and uses ad hoc heuristics that might not work well for complex exploration problems. In contrast, off-policy actor-critic methods like SAC (Haarnoja et al., 2018) and *twin-delayed deep deterministic policy gradient* (TD3) (Fujimoto et al., 2018) offer higher sample efficiency at the cost of increased computational complexity. Additionally, they may introduce a higher bias in the policy update due to their reliance on the critic network to evaluate the actions.

Instead of this step-based view, we can also consider RL from a trajectory perspective, where the primary goal is to directly optimize complete trajectory behaviors rather than individual actions per state in isolation. Within the framework of contextual episode-based policy search (Deisenroth et al., 2013; Daniel et al., 2012), RL is treated as a black-box optimization problem. The goal is to maximize the expected return $R(\boldsymbol{w},\boldsymbol{c})$ by optimizing a contextual search distribution $\pi(\boldsymbol{w}|\boldsymbol{c})$ over the controller parameters $\boldsymbol{w}$. Here, the context vector $\boldsymbol{c}$ characterizes the given task, such as the specified goal or object locations. The controller is typically represented by a *movement primitive* (MP) (Paraschos et al., 2013; Schaal, 2006; Ijspeert et al., 2013; Li et al., 2022) or other trajectory generators. Similar to SRL from above, we can formulate the objective as

$$\arg\max_{\pi(\boldsymbol{w}|\boldsymbol{c})} \mathbb{E}_{p(\boldsymbol{c})}\left[\mathbb{E}_{\pi(\boldsymbol{w}|\boldsymbol{c})}[R(\boldsymbol{w},\boldsymbol{c})]\right],$$

where $p(\boldsymbol{c})$ denotes the context distribution associated with the task. The return function $R(\boldsymbol{w},\boldsymbol{c})$ imposes no structural assumptions and can be any non-Markovian function of the resulting trajectory, given the black-box nature of the problem.

Now, the policy does not parameterize raw actions for the environment; instead, it parametrizes a controller or a MP, which is a widely used tool for motion representation and generation in robotics. MPs are used as building blocks for movements, allowing for the modulation of motion behavior and the creation of more complex movements through combination or concatenation. With their concise parameterization and flexibility, MPs have become a popular choice in imitation learning (Maeda et al., 2014; Gomez-Gonzalez et al., 2016; Pahič et al., 2020; Li et al., 2022; Rozo and Dave, 2022) and RL (Celik et al., 2022; Li et al., 2017). In this work, our focus is on trajectory-based movement representations, following the trajectory-oriented approach in previous RL literature (Schaal, 2006; Ijspeert et al., 2013; Paraschos et al., 2013; Li et al., 2022). These representations, given a parameter vector, generate desired trajectories for the agent to follow. The goal of this type of policy parametrization is to optimize the weight vector or its distribution to improve the resulting movements. While there are several different variations of MPs, we will provide an overview of the three that we consider the most important and will use in this work.

Probabilistic Movement Primitives Utilizing a linear basis function model, *probabilistic movement primitives* (ProMPs) (Paraschos et al., 2013) generate a trajectory $\boldsymbol{\lambda}$ as

$$y(t) = \boldsymbol{\Phi}^{\intercal}(t)\boldsymbol{w}, \quad \boldsymbol{\lambda} = [y_t]_{t=0:T} = \boldsymbol{\Phi}_{0:T}^{\intercal}\boldsymbol{w},$$

where $\boldsymbol{w}$ is the time-independent weight vector provided by the policy, $y(t)$ represents the trajectory position at time step t and $\boldsymbol{\Phi}$ encompasses pre-defined time-dependent basis functions, such as normalized *radial basis functionss* (RBFs). Due to the simplicity of the linear basis function representation, ProMPs allow for fast trajectory computation and enable modeling the trajectory's statistics from the weight vector's distribution. These statistics often include temporal correlations within trajectories and motion correlations across different *degree of freedom* (DoF). However, a notable limitation of ProMPs lies in their lack of smoothness during trajectory replanning and concatenation. This implies that when selecting a new weight vector $\boldsymbol{w}$ during execution, there is no guarantee that the trajectory will initiate from the desired initial conditions at the current time step. This absence of smoothness and the inability to adapt the trajectory's starting state present practical challenges in applying ProMPs, especially in scenarios where the weight vector requires dynamic updates throughout motion execution due to unforeseen changes in the environment.

Dynamic Movement Primitives In contrast to ProMPs, *dynamic movement primitives* (DMPs) (Schaal, 2006; Ijspeert et al., 2013) shape trajectories through the integration of a dynamic system, offering smooth replanning for both position and velocity (Brandherm et al., 2019; Ginesi et al., 2019; Lee et al., 2020). However, this enhanced smoothness comes at a computational expense, as DMPs necessitate online numerical integration to compute a trajectory. Incorporating DMPs into neural networks introduces coupling between the forward and backward passes of the networks with this numerical integration process (Bahl et al., 2020; Pahič et al., 2018, 2020), resulting in complex and slow models.

Probabilistic Dynamic Movement Primitives In an effort to combine the strengths and address the weaknesses of ProMPs and DMPs, Li et al. (2022) recently introduced *probabilistic dynamic movement primitives* (ProDMPs). This innovation involves solving the underlying *ordinary differential equation* (ODE) of the DMP, effectively replacing the computationally expensive online numerical integration with position and velocity basis functions. These basis functions can be computed offline and shared across all trajectories, leading to a representation of trajectory position and velocity akin to that of ProMPs, as expressed in the following

$$y(t) = c_1 y_1(t) + c_2 y_2(t) + \boldsymbol{\Phi}(t)^\intercal \boldsymbol{w}_g, \quad \dot{y}(t) = c_1 \dot{y}_1(t) + c_2 \dot{y}_2(t) + \dot{\boldsymbol{\Phi}}(t)^\intercal \boldsymbol{w}_g,$$

where y_1 and y_2 denote the two linearly independent complementary functions of the governing ODE of the DMP, with corresponding derivatives with respect to time $\dot{y}_1$ and $\dot{y}_2$. The coefficients c_1 and c_2, shared by both position and velocity representations, are determined by solving an initial condition problem of the ODE. The position and velocity basis functions, denoted by $\boldsymbol{\Phi}$ and $\dot{\boldsymbol{\Phi}}$ respectively, can be computed once offline and are later used as constant functions. The vector $\boldsymbol{w}_g$ in both equations concatenates the DMP's original weights $\boldsymbol{w}$ and goal attractor g into one vector. To simplify our notation, we will refer to all learned parameters of MPs collectively as $\boldsymbol{w}$ from this point forward. A concise derivation of these equations is presented in Appendix B.1. For a more in-depth understanding, readers are directed to the original paper (Li et al., 2022).
Theoretically, both DMP and ProDMP can be used in domains requiring online replanning. In this work, we specifically use the ProDMPs model as our trajectory generator because it ensures smooth replanning with low computational cost.

Learning Movement Primitive Policies Most *episode-based RL* (ERL) algorithms focus on the non-contextual setting and only learn a single weight $\boldsymbol{w}$. They use different optimization techniques, such as PGs (Sehnke et al., 2010), natural gradients (Wierstra et al., 2014), stochastic search strategies (Hansen and Ostermeier, 2001; Mannor et al., 2003; Abdolmaleki et al., 2019), or trust-region optimization techniques (Abdolmaleki et al., 2015; Daniel et al., 2012; Tangkaratt et al., 2017). Early methods that incorporate context adaptation (Tangkaratt et al., 2017; Abdolmaleki et al., 2019) only consider a linear mapping from context to parameter space, imposing a major constraint on their performance.
A related line of work comes from the domain of evolutionary strategies (Mania et al., 2018; Salimans et al., 2017; Chrabaszcz et al., 2018). They propose full gradient-free black-box approaches as an alternative to gradient- and step-based methods for finding optimal neural network parameters. These approaches treat learning the neural network policy parameters (several thousand parameters) as the black-box optimization problem as opposed to learning MP parameters (20-50 parameters). While these methods demonstrate competitiveness for black-box optimization of neural networks, they typically overlook contextual setups requiring distinct parameters for different contexts. In

contextual scenarios, where the performance of a rollout depends on both the parameter vector and the context (*e.g.* the goal), these methods introduce additional noise during evaluation, given their lack of context awareness. For example, a well-parametrized neural network may perform suboptimally when evaluated for a challenging context, while a poorly-parametrized network may perform well for a simple context. Such approaches, however, remain oblivious to context and are not competitive for complex contextual scenarios. In contrast, MP-based approaches do not optimize at the level of a global neural network control policy, which would involve thousands of parameters. Instead, they focus on local control parameters of MPs or similar controllers, typically ranging from 10 to 50 dimensions. This localized optimization strategy proves advantageous, especially in scenarios where context-specific adaptation is crucial.

Reinforcement Learning with Movement Primitives. While the majority of research in *RL with MPs* (MPRL) concentrates on learning a single MP parameter vector tailored to one specific task configuration (Abdolmaleki et al., 2015; Kober and Peters, 2008; Stulp and Sigaud, 2012a,b), some methods permit linear adaptation of the MP's parameter vector to the context (Daniel et al., 2012; Kupcsik et al., 2017; Celik et al., 2022). Additionally, a few RL approaches integrate non-linear policies with predefined action primitives, such as pushing or grasping motions (Dalal et al., 2021; Zenkri et al., 2022). An example that directly incorporates MPs and deep networks within a SRL context is the *neural dynamic policies* (NDP) (Bahl et al., 2020). NDP attempts to embed the structure of DMPs into deep policies by reparameterizing action spaces through second-order differential equations. This can be seen as an intersection between step-based and episode-based methods by learning sub-trajectories via DMPs spanning multiple time steps. While this approach facilitates effective replanning, the primary focus of exploration occurs at the action level rather than at the trajectory level, similar to standard step-based approaches. This neglects the main benefit of using MPs in an RL context. Moreover, using DMPs necessitates several numerical integration steps that also need to be differentiated, making it computationally expensive.

Chapter 3

A Differentiable Trust Region Projection Layer

This chapter serves as the cornerstone of the thesis, introducing a novel trust region method, which was previously published in Otto et al. (2021). The method, characterized by its mathematical rigor and stability, surpasses existing approaches. It provides better control over the exploration-exploitation trade-off inherent in *policy gradient* (PG) methods, demonstrating particular efficacy in higher-dimensional action spaces. The robustness and versatility of this novel method lay a solid foundation for subsequent chapters, where its strengths will be further harnessed and leveraged to address complex challenges in the field.

3.1 Introduction

Deep *reinforcement learning* (RL) has shown considerable advances in recent years with prominent application areas such as games (Mnih et al., 2015; Silver et al., 2017), robotics (Levine et al., 2015), and control (Duan et al., 2016). In policy search, PG methods have been highly successful and have gained, among others, great popularity (Peters and Schaal, 2008). However, often it is difficult to tune learning rates for vanilla PG methods because they tend to reduce the entropy of the policy too quickly. This results in a lack of exploration and, as a consequence, in premature or slow convergence. A common practice to mitigate these limitations is to impose a constraint on the allowed change between two successive policies. Kakade and Langford (2002) provided a theoretical justification for this in the approximate policy iteration setting. Two of the arguably most favored policy search algorithms, *trust region policy optimization* (TRPO) (Schulman et al., 2015a) and *proximal policy optimization* (PPO) (Schulman et al., 2017), follow this idea using the *Kullback-Leibler* (KL) between successive policies as a constraint.

We propose closed-form projections for Gaussian policies, realized as differentiable neural network layers. These layers constrain the change in successive policies by projecting the updated policy onto trust regions. First, this approach is more stable concerning what Engstrom et al. (2020) refer to as *code-level optimizations* than other approaches. Second, it comes with the benefit of imposing constraints for individual states, allowing for

the possibility of state-dependent trust regions. This allows us to constrain the state-wise maximum change of successive policies. In this, we differ from previous works, which constrain only the expected change and thus cannot rely on exact guarantees of monotonic improvement. Furthermore, we propose three different similarity measures, the KL divergence, the Wasserstein L2 distance, and the Frobenius norm, to base our trust region approach on. The last layer of the projected policy is now the trust region layer, which relies on the old policy as input. This would result in an ever-growing stack of policies, rendering this approach infeasible. To circumvent this issue, we introduce a penalty term into the RL objective to ensure the input and output of the projection stay close together. While this still results in an approximation of the trust region update, we show that the trust regions are properly enforced. We also extend our approach to allow for a controlled evolution of the entropy of the policy, which has been shown to increase the performance in difficult exploration problems (Pajarinen et al., 2019; Akrour et al., 2019).
We compare and discuss the effect of the different similarity measures as well as the entropy control on the optimization process. Additionally, we benchmark our algorithm against existing methods and demonstrate that we achieve similar or better performance.

3.2 Preliminaries and Problem Statement

We consider the general problem of a policy search in a *Markov Decision Process* (MDP) as defined in Section 2.1. To find the optimal policy, we have seen that traditional PG methods often make use of the likelihood ratio gradient and an importance sampling estimator. Moreover, instead of directly optimizing the returns, it is more effective to optimize the advantage function as this results in an unbiased estimator of the gradient with less variance

$$\max_{\theta} \hat{J}(\pi_\theta, \pi_{\theta_{\text{old}}}) = \max_{\theta} \mathbb{E}_{(s,a)\sim\pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\theta_{\text{old}}}}(s,a) \right], \tag{3.1}$$

where $A^{\pi}(s,a) = \mathbb{E}[R^{\gamma}|s_0 = s, a_0 = a; \pi] - \mathbb{E}[R^{\gamma}|s_0 = s; \pi]$ describes the advantage function, and the expectation is w.r.t $\pi_{\theta_{\text{old}}}$, *i.e.* $s' \sim p(\cdot|s,a), a \sim \pi_{\theta_{\text{old}}}(\cdot|s), s_0 \sim \rho(s_0), s \sim \rho_{\pi_{\theta_{\text{old}}}}$ where $\rho_{\pi_{\theta_{\text{old}}}}$ is the stationary state distribution of policy $\pi_{\theta_{\text{old}}}$. The advantage function is commonly estimated by *generalized advantage estimation* (GAE) (Schulman et al., 2015b). Trust region methods use additional constraints for the given objective. Using a constraint on the maximum KL over the states has been shown to guarantee monotonic improvement of the policy (Schulman et al., 2015a). However, since all current approaches do not use a maximum KL constraint but an expected KL constraint, the guarantee of monotonic improvement does not hold exactly either. We are not aware of such results for the *Wasserstein L2-distance* (W2) or the Frobenius norm.
For our projections we assume Gaussian policies $\pi_{\theta_{\text{old}}}(a_t|s_t) = \mathcal{N}(a_t|\mu_{\text{old}}(s_t), \Sigma_{\text{old}}(s_t))$

and $\pi_\theta(a_t|s_t) = \mathcal{N}(a_t|\mu(s_t), \Sigma(s_t))$ represent the old as well as the current policy, respectively. We explore three trust regions on top of Equation (3.1) that employ different similarity measures between old and new distributions, more specifically the frequently used reverse KL divergence, the W2, and the Frobenius norm.

Reverse KL Divergence The KL divergence between two Gaussian distributions with means μ_1 and μ_2 and covariances Σ_1 and Σ_2 can generally be written as

$$\mathrm{KL}(\{\mu_1, \Sigma_1\} \parallel \{\mu_2, \Sigma_2\}) = \frac{1}{2}\left[(\mu_2 - \mu_1)^T \Sigma_2^{-1}(\mu_2 - \mu_1) + \log\frac{|\Sigma_2|}{|\Sigma_1|} + \mathrm{tr}\{\Sigma_2^{-1}\Sigma_1\} - d\right],$$

where d is the dimensionality of μ_1, μ_2. The KL uses the Mahalanobis distance to measure the similarity between the two mean vectors. The difference in the covariances is measured by the difference in shape, *i.e.*, the difference in scale, given by the log ratio of the determinants, plus the difference in rotation, given by the trace term. Given the KL is non-symmetric, it is not a distance, yet still a frequently used divergence between distributions. We will use the more common reverse KL for our trust region, where the first argument is the new policy and the second is the old policy.

Wasserstein Distance The Wasserstein distance is a distance measure based on an optimal transport formulation, for more details see Villani (2008). The W2 for two Gaussian distributions can generally be written as

$$\mathcal{W}_2\left(\{\mu_1, \Sigma_1\}, \{\mu_2, \Sigma_2\}\right) = |\mu_1 - \mu_2|^2 + \mathrm{tr}\left(\Sigma_1 + \Sigma_2 - 2\left(\Sigma_2^{1/2}\Sigma_1\Sigma_2^{1/2}\right)^{1/2}\right).$$

A key difference to the KL divergence is that the Wasserstein distance is a symmetric distance measure, *i.e.*, $\mathcal{W}_2(q, p) = \mathcal{W}_2(p, q)$. Our experiments also revealed that it is beneficial to measure the W2 distance in a metric space defined by the covariance of the old policy distribution, denoted here as Σ_2, as the distance measure is then more sensitive to the data-generating distribution. The W2 distance in this metric space reads

$$\begin{aligned}\mathcal{W}_{2,\Sigma_2}\left(\{\mu_1, \Sigma_1\}, \{\mu_2, \Sigma_2\}\right) = &(\mu_2 - \mu_1)^T \Sigma_2^{-1}(\mu_2 - \mu_1) \\ &+ \mathrm{tr}\left(\Sigma_2^{-1}\Sigma_1 + \mathbb{I} - 2\Sigma_2^{-1}\left(\Sigma_2^{1/2}\Sigma_1\Sigma_2^{1/2}\right)^{1/2}\right).\end{aligned}$$

Frobenius Norm The Frobenius norm is a matrix norm and can directly be applied to the difference of the covariance matrices of the Gaussian distributions. To measure the distance of the mean vectors, we will, similar to the KL divergence, employ the Mahalanobis distance as this empirically leads to improved performance in comparison to just taking the squared distance. Hence, we will denote the following metric as the

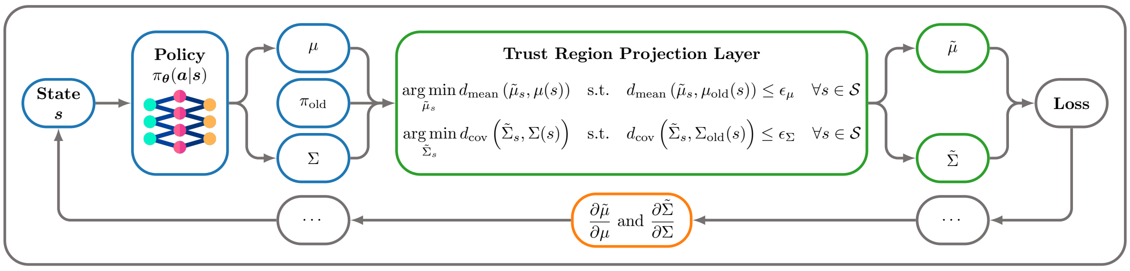

State s
Policy $\pi_\theta(a|s)$
μ
π_{old}
Σ
Trust Region Projection Layer
$\arg\min_{\tilde{\mu}_s} d_{\text{mean}}(\tilde{\mu}_s, \mu(s))$ s.t. $d_{\text{mean}}(\tilde{\mu}_s, \mu_{\text{old}}(s)) \leq \epsilon_\mu \quad \forall s \in \mathcal{S}$
$\arg\min_{\tilde{\Sigma}_s} d_{\text{cov}}(\tilde{\Sigma}_s, \Sigma(s))$ s.t. $d_{\text{cov}}(\tilde{\Sigma}_s, \Sigma_{\text{old}}(s)) \leq \epsilon_\Sigma \quad \forall s \in \mathcal{S}$
$\tilde{\mu}$
$\tilde{\Sigma}$
Loss
$\cdots$
$\frac{\partial\tilde{\mu}}{\partial\mu}$ and $\frac{\partial\tilde{\Sigma}}{\partial\Sigma}$
$\cdots$

Figure 3.1: Overview of the proposed TRPL. A Gaussian policy π_θ is predicting one set of parameters of the distribution μ and Σ per state. The parameters for each state are then projected individually to satisfy the trust region in case the bounds are violated for any of the states with respect to the reference policy π_{old}. This is achieved by solving the two optimization problems per state, which can be done in (partially) closed form and fully differentiable. The resulting projected parameters $\tilde{\mu}$ and $\tilde{\Sigma}$ can then be used for further computations, such as sampling or loss calculation. Since everything is differentiable, the gradient can then be backpropagated to the neural network policy. As a similarity measure for the trust region, we use the reverse KL divergence, the W2, and the Frobenius norm.

Frobenius norm between two Gaussian distributions

$$F(\{\mu_1,\Sigma_1\},\{\mu_2,\Sigma_2\}) = (\mu_2-\mu_1)^T\Sigma_2^{-1}(\mu_2-\mu_1) + \operatorname{tr}\left((\Sigma_2-\Sigma_1)^T(\Sigma_2-\Sigma_1)\right).$$

The Frobenius norm also constitutes a symmetric distance measure.

3.3 Differentiable Trust-Region Layers for Gaussian Policies

We present projections based on the three similarity measures, *i.e.*, Frobenius norm, W2, and KL divergence. These projections realize state-wise trust regions and can be directly integrated into the optimization process as differentiable neural network layers, hence we call our approach *trust region projection layer* (TRPL). Additionally, we extend TRPL to include an entropy constraint to gain control over the evolution of the policy entropy during optimization. The trust regions are defined by a distance or divergence $d(\pi(\cdot|s), \pi_{\text{old}}(\cdot|s))$ between probability distributions. Complementing Equation (3.1) with the trust region constraint leads to

$$\max_\theta \hat{J}(\pi_\theta, \pi_{\theta_{\text{old}}}) \quad \text{s.t.} \quad d(\pi_{\theta_{\text{old}}}(\cdot|s)), \pi_\theta(\cdot|s)) \leq \varepsilon \quad \forall s \in \mathcal{S}. \tag{3.2}$$

While, in principle, we want to enforce the constraint for every possible state, in practice, we can only enforce them for states sampled from rollouts of the current policy.

To solve the problem in Equation (3.2), a standard neural network will output the parameters μ, Σ of a Gaussian distribution π_θ, ignoring the trust region bounds. These parameters are provided to the trust region layers, together with the mean and covariance of the old policy and a parameter specifying the size of the trust region ε. The new policy is then given by the output of the trust region layer. Since the old policy distribution is fixed, all distances or divergences used in this paper can be decomposed into a mean and a covariance dependent part. This enables us to use separate trust regions as well as bounds for mean and covariance, allowing for more flexibility in the algorithm. The trust region layers aim to project π_θ into the trust region by finding parameters $\tilde{\mu}$ and $\tilde{\Sigma}$ that are closest to the original parameters μ and Σ while satisfying the trust region constraints. The projection is based on the same distance or divergence which was used to define the respective trust region. Formally, this corresponds to the following optimization problems for each s

$$\arg\min_{\tilde{\mu}_s} d_{\text{mean}}\left(\tilde{\mu}_s, \mu(s)\right), \quad \text{s.t.} \quad d_{\text{mean}}\left(\tilde{\mu}_s, \mu_{\text{old}}(s)\right) \leq \varepsilon_\mu, \quad \text{and} \tag{3.3}$$

$$\arg\min_{\tilde{\Sigma}_s} d_{\text{cov}}\left(\tilde{\Sigma}_s, \Sigma(s)\right), \quad \text{s.t.} \quad d_{\text{cov}}\left(\tilde{\Sigma}_s, \Sigma_{\text{old}}(s)\right) \leq \varepsilon_\Sigma, \tag{3.4}$$

where $\tilde{\mu}_s$ and $\tilde{\Sigma}_s$ are the optimization variables for state s. Here, d_{mean} is the mean dependent part and d_{cov} is the covariance dependent part of the employed distance or divergence. For brevity of notation, we will neglect all dependencies on the state in the following. We denote the projected policy as $\tilde{\pi}(a|s) = \mathcal{N}(a|\tilde{\mu}, \tilde{\Sigma})$. An overview of the resulting approach can be found in Figure 3.1.

3.3.1 Projection of the Mean

For all three trust region objectives, we make use of the same distance measure for the mean, the Mahalanobis distance. Thus, the optimization problem for the mean is

$$\arg\min_{\tilde{\mu}} (\mu - \tilde{\mu})^{\mathrm{T}} \Sigma_{\text{old}}^{-1} (\mu - \tilde{\mu}) \quad \text{s.t.} \quad (\mu_{\text{old}} - \tilde{\mu})^{\mathrm{T}} \Sigma_{\text{old}}^{-1} (\mu_{\text{old}} - \tilde{\mu}) \leq \varepsilon_\mu. \tag{3.5}$$

By making use of the method of Lagrangian multipliers (see Appendix A.2.2), we can formulate the dual and solve it for the projected mean $\tilde{\mu}$ as

$$\tilde{\mu} = \frac{\mu + \omega \mu_{\text{old}}}{1 + \omega} \quad \text{with} \quad \omega = \sqrt{\frac{(\mu_{\text{old}} - \mu)^{\mathrm{T}} \Sigma_{\text{old}}^{-1} (\mu_{\text{old}} - \mu)}{\varepsilon_\mu}} - 1. \tag{3.6}$$

This equation can directly be used as mean for the Gaussian policy, while it easily allows computing gradients. Note, that for the mean part of the KL we would need to use the Σ^{-1} instead of Σ_{old}^{-1} in the objective of Equation (3.5). Yet, this objective still results in a valid trust region problem which is much easier to optimize.

3.3.2 Projection of the Covariance

Frobenius Projection. The Frobenius projection formalizes the trust region for the covariance with the squared Frobenius norm of the matrix difference, which yields

$$\arg\min_{\tilde{\Sigma}} \operatorname{tr}\left((\Sigma-\tilde{\Sigma})^T(\Sigma-\tilde{\Sigma})\right), \quad \text{s.t.} \quad \operatorname{tr}\left((\Sigma_{\text{old}}-\tilde{\Sigma})^T(\Sigma_{\text{old}}-\tilde{\Sigma})\right) \leq \varepsilon_\Sigma.$$

We again use the method of Lagrangian multipliers (see Appendix A.2.3) and get the covariance $\tilde{\Sigma}$ as

$$\tilde{\Sigma} = \frac{\Sigma + \eta\Sigma_{\text{old}}}{1+\eta} \quad \text{with} \quad \eta = \sqrt{\frac{\operatorname{tr}\left((\Sigma_{\text{old}}-\Sigma)^T(\Sigma_{\text{old}}-\Sigma)\right)}{\varepsilon_\Sigma}} - 1, \tag{3.7}$$

where η is the corresponding Lagrangian multiplier.

Wasserstein Projection. Deriving the Wasserstein projection follows the same procedure. We obtain the following optimization problem

$$\begin{aligned} &\arg\min_{\tilde{\Sigma}} \operatorname{tr}\left(\Sigma_{\text{old}}^{-1}\Sigma + \Sigma_{\text{old}}^{-1}\tilde{\Sigma} - 2\Sigma_{\text{old}}^{-1}\left(\Sigma^{1/2}\tilde{\Sigma}\Sigma^{1/2}\right)^{1/2}\right), \\ &\text{s.t.} \quad \operatorname{tr}\left(\mathbb{I} + \Sigma_{\text{old}}^{-1}\tilde{\Sigma} - 2\Sigma_{\text{old}}^{-1}\left(\Sigma_{\text{old}}^{1/2}\tilde{\Sigma}\Sigma_{\text{old}}^{1/2}\right)^{1/2}\right) \leq \varepsilon_\Sigma, \end{aligned} \tag{3.8}$$

where $\mathbb{I}$ is the identity matrix. A closed form solution to this optimization problem can be found by using the methods outlined in Takatsu (2011). However, we found the resulting solution for the projected covariance matrices to be numerically unstable. Therefore, we made the simplifying assumption that both the current Σ and the old covariance Σ_{old} commute with $\tilde{\Sigma}$. Under the common premise of diagonal covariances, this commutativity assumption always holds. For the more general case of arbitrary covariance matrices, we would need to ensure the matrices are sufficiently close together, which is effectively ensured by Equation (3.8). Again, we introduce Lagrange multipliers and solve the dual problem to obtain the optimal primal and dual variables (see Appendix A.2.4). Note, however, that here we chose the *square root* of the covariance matrix[1] as primal variable. The corresponding projection for the square root covariance $\tilde{\Sigma}^{1/2}$ is then

$$\tilde{\Sigma}^{1/2} = \frac{\Sigma^{1/2} + \eta\Sigma_{\text{old}}^{1/2}}{1+\eta} \quad \text{with} \quad \eta = \sqrt{\frac{\operatorname{tr}\left(\mathbb{I} + \Sigma_{\text{old}}^{-1}\Sigma - 2\Sigma_{\text{old}}^{-1/2}\Sigma^{1/2}\right)}{\varepsilon_\Sigma}} - 1, \tag{3.9}$$

[1]We assume the true matrix square root $\Sigma = \Sigma^{1/2}\Sigma^{1/2}$ and not a Cholesky factor $\Sigma = LL^{\mathrm{T}}$ since it naturally appears in the expressions for the projected covariance from the original Wasserstein formulation.

where η is the corresponding Lagrangian multiplier. We see the same pattern emerging as for the Frobenius projection. The chosen similarity measure reappears in the expression for the Lagrangian multiplier and the primal variables are weighted averages of the corresponding parameters of the old and the predicted Gaussian.

KL Projection. Identically to the previous two projections, we reformulate Equation (3.4) as

$$\arg\min_{\tilde{\Sigma}} \operatorname{tr}\left(\Sigma^{-1}\tilde{\Sigma}\right) + \log\frac{|\Sigma|}{|\tilde{\Sigma}|}, \quad \text{s.t.} \quad \operatorname{tr}\left(\Sigma_{\text{old}}^{-1}\tilde{\Sigma}\right) - d + \log\frac{|\Sigma_{\text{old}}|}{|\tilde{\Sigma}|} \leq \varepsilon_{\Sigma}, \tag{3.10}$$

where d is the dimensionality of the action space. It is impossible to acquire a fully closed form solution for this problem. However, following Abdolmaleki et al. (2015), we can obtain the projected precision $\tilde{\Lambda} = \tilde{\Sigma}^{-1}$ by interpolation between the precision matrices of the old policy π_{old} and the current policy π

$$\tilde{\Lambda} = \frac{\eta^* \Lambda_{\text{old}} + \Lambda}{\eta^* + 1}, \quad \eta^* = \arg\min_{\eta} g(\eta), \text{ s.t. } \eta \geq 0, \tag{3.11}$$

where η is the corresponding Lagrangian multiplier and $g(\eta)$ the dual function. While this dual cannot be solved in closed form, an efficient solution exists using a standard numerical optimizer, such as BFGS, since it is a 1-D convex optimization. Regardless, we want a differentiable projection and thus also need to backpropagate the gradients through numerical optimization. To this end, we follow Amos and Kolter (2017) and compute those gradients by taking the differentials of the KKT conditions of the dual. We refer to Appendix A.2.5 for more details and derivations.

Entropy Control. Previous works (Akrour et al., 2019; Abdolmaleki et al., 2015) have shown the benefits of introducing an entropy constraint $\mathcal{H}(\pi_\theta) \geq \beta$ in addition to the trust region constraints. Such a constraint allows for more control over the exploration behavior of the policy. In order to endow our algorithm with this improved exploration behavior, we make use of the results from Akrour et al. (2019) and scale the standard deviation of the Gaussian distribution with a scalar factor $\exp\{(\beta - \mathcal{H}(\pi_\theta))/d\}$, which can also be individually computed per state.

3.3.3 Analysis of the Projections

It is instructive to compare the three projections. The covariance update is an interpolation for all three projections, but the quantities that are interpolated differ. For the Frobenius projection we directly interpolate between the old and current covariances (Equation (3.7)), for the W2 projection between their respective matrix square roots (Equation (3.9)), and for the KL projection between their inverses (Equation (3.11)).

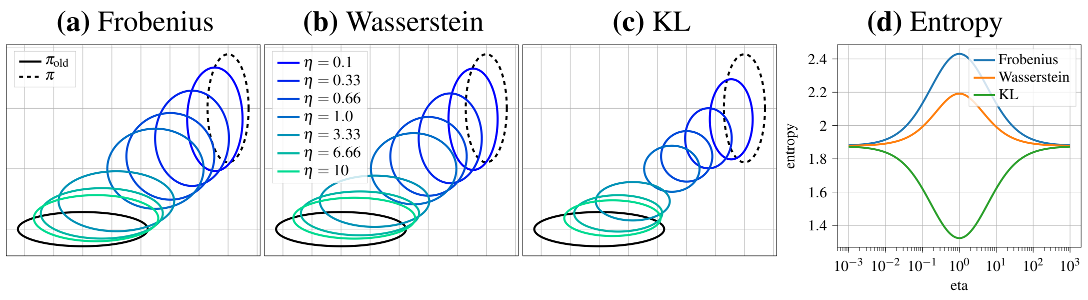


Figure 3.2: **(a)**, **(b)**, and **(c)**: Interpolated covariances for the different projections for various values of η. For Frobenius and Wasserstein the intermediate distributions have a larger entropy, while for the KL projection, the intermediate entropy is smaller. **(d)**: Entropy of the interpolated distributions. In this example π and π_{old} have the same entropy. It can be seen that the entropy increases for the Frobenius and Wasserstein projections when transitioning between the distributions, while it decreases for the KL. A more general statement regarding this can be found in Theorem 1.

In other words, each projection suggests which parametrization to use for the covariance matrix. The different interpolations also have an interesting effect on the entropy of the resulting covariances, which can be observed in Figure 3.2. Further, we can prove the following theorem about the entropy of the projected distributions

Theorem 1 *Let π_θ and $\pi_{\theta_{old}}$ be Gaussian and $\eta \geq 0$, then for the entropy of the projected distribution $\mathcal{H}(\tilde{\pi})$ it holds that $\mathcal{H}(\tilde{\pi}) \geq minimum(\mathcal{H}(\pi_\theta), \mathcal{H}(\pi_{\theta_{old}}))$ for the Frobenius (Equation* (3.7)*) and the Wasserstein projection (Equation* (3.9)*), as well as, $\mathcal{H}(\tilde{\pi}) \leq maximum(\mathcal{H}(\pi_\theta), \mathcal{H}(\pi_{\theta_{old}}))$ for the KL projection (Equation* (3.11)*).*

The proof is based on the multiplicative version of the Brunn-Minkowski inequality and can be found in Appendix A.2.1. Intuitively, this means that the Frobenius and Wasserstein projections act more *aggressively*, *i.e.*, they tend to yield a higher entropy, while the KL projection acts more *conservatively*, *i.e.*, it tends to yield a lower entropy. This could also explain why many KL based trust region methods lose entropy too quickly and converge prematurely. By introducing an explicit entropy control, those effects can be mitigated.

3.3.4 Successive Policy Updates

The above projections can directly be implemented for training the current policy. Note, however, that at each epoch i the policy π_i predicted by the network before the projection layer does not respect the constraints and thus relies on calling this layer. The policy of the projection layer $\tilde{\pi}_i$ not only depends on the parameters of π_i but also on the old policy network $\pi_{i,\text{old}} = \tilde{\pi}_{i-1}$. This would result in an ever-growing stack of policy networks becoming increasingly costly to evaluate. In other words, $\tilde{\pi}_i$ is computed using all stored networks of $\pi_i, \pi_{i-1}, \dots, \pi_0$. We now discuss the parametrization of $\tilde{\pi}$ via amortized optimization.

We need to encode the information of the projection layer into the parameters θ of the next policy, *i.e.* $\tilde{\pi}(a|s;\theta) = p \circ \pi_\theta(a|s)$ is a composition function in which p denotes the projection layer. The output of π_θ is (μ, Σ), and p computes $(\tilde{\mu}, \tilde{\Sigma})$ according to Equations (3.6), (3.7), (3.9) and (3.11). Formally, we aim to find a set of parameters $\theta^* = \arg\min_\theta \mathbb{E}_{\boldsymbol{s}\sim\rho_{\pi_{\text{old}}}}\left[d\left(\tilde{\pi}(\cdot|\boldsymbol{s}), \pi_\theta(\cdot|\boldsymbol{s})\right)\right]$, where $\rho_{\pi_{\text{old}}}$ is the stationary state distribution of the old policy and d is the similarity measure used for the projection, such that we minimize the expected distance or divergence between the projection and the current policy prediction.

The most intuitive way to solve this problem is to use the existing samples for additional regression steps after the policy optimization. Still, this adds a computational overhead. Therefore, we propose to concurrently optimize both objectives during training by penalizing the main objective, *i.e.*,

$$\arg\min_\theta \mathbb{E}_{(s,a)\sim\pi_{\theta_{\text{old}}}}\left[\frac{\tilde{\pi}(a|s;\theta)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\text{old}}}(s,a)\right] - \alpha \mathbb{E}_{\boldsymbol{s}\sim p_{\pi_{\text{old}}}}\left[d\left(\tilde{\pi}(\cdot|\boldsymbol{s};\theta), \pi_\theta(\cdot|\boldsymbol{s})\right)\right]. \quad (3.12)$$

Note that the importance sampling ratio is computed based on a Gaussian distribution generated by the trust region layer and not directly from the network output. Furthermore, the gradient for the regression penalty does not flow through the projection, it is solely acting as a supervised learning signal. As appropriate similarity measures d for the penalty, we resort to the measures used in each projection. For a detailed algorithmic view see Appendix A.1.

Several authors (Dalal et al., 2018; Chow et al., 2019; Yang et al., 2020) used projections as network layers to enforce limitations in the action or state space given environmental restrictions, such as robotic joint limits.

3.4 Experiments

Mujoco Benchmarks We evaluate the performance of TRPL regarding sample complexity and final reward in comparison to *projected approximate policy iteration* (PAPI) (Akrour et al., 2019) and PPO (Schulman et al., 2017) on the OpenAI gym benchmark suite (Brockman et al., 2016). We explicitly did not include TRPO in the evaluation, as Engstrom et al. (2020) showed that it can achieve similar performance to PPO. For our experiments, the PAPI projection and its conservative PPO version are executed in the setting sent to us by the authors. The hyperparameters for all three projections and PPO have been selected with Optuna (Akiba et al., 2019). See Appendix A.4 for a full listing of all hyperparameters. We use a shared set of hyperparameters for all environments except for the Humanoid, which we optimized separately. Next to the standard PPO implementation with all code-level optimizations, we further evaluate PPO-M, which only

Table 3.1: Mean return with 95% confidence interval of 20 epochs after completing 20% of the total training and for the last 20 epochs. We trained 40 different seeds for each experiment and computed five evaluation rollouts per epoch. The projections with (-E) and without entropy control are considered separately, therefore, each column may have up to two best runs (bold).

	Hopper-v2		Walker2d-v2		Halfcheetah-v2		Ant-v2		Humanoid-v2	
	20%	final	20%	final	20%	final	20%	final	20%	final
FROB	1646 ± 19	**2578 ± 17**	2142 ± 23	3443 ± 19	2525 ± 11	3552 ± 14	1265 ± 11	3035 ± 26	2176 ± 65	5202 ± 23
W2	1586 ± 31	2490 ± 19	2284 ± 20	3390 ± 17	**2586 ± 9**	3692 ± 15	1362 ± 12	3086 ± 28	2502 ± 87	5057 ± 25
KL	1584 ± 20	2476 ± 10	2071 ± 36	**3583 ± 14**	2369 ± 9	**4255 ± 17**	1460 ± 26	**3335 ± 21**	**2923 ± 53**	**5510 ± 27**
PAPI	1378 ± 20	**2549 ± 12**	1663 ± 21	3232 ± 20	1875 ± 5	2380 ± 6	645 ± 5	3198 ± 17	1824 ± 74	5367 ± 22
PPO-M	1030 ± 23	2321 ± 19	1994 ± 18	2771 ± 40	1922 ± 15	3272 ± 18	1494 ± 9	2783 ± 32	604 ± 10	5172 ± 23
PPO	**1881 ± 30**	2515 ± 15	**2490 ± 33**	**3447 ± 17**	2048 ± 8	2880 ± 8	**1657 ± 10**	2852 ± 25	1723 ± 67	4969 ± 18
FROB-E	1587 ± 30	2478 ± 11	2037 ± 21	3370 ± 27	**2762 ± 10**	4568 ± 14	730 ± 9	**3475 ± 26**	3662 ± 44	5807 ± 18
W2-E	1518 ± 36	2437 ± 13	2174 ± 19	3303 ± 25	2266 ± 8	4213 ± 13	855 ± 27	3361 ± 30	3658 ± 56	**5844 ± 8**
KL-E	1502 ± 18	2497 ± 16	2215 ± 31	3171 ± 28	2611 ± 12	**4584 ± 18**	955 ± 16	3437 ± 21	**3801 ± 42**	5430 ± 16

leverages the core PPO algorithm. TRPL and PPO-M solely use the observation normalization, network architecture, and initialization from the original PPO implementation. All algorithms parametrize the covariance as a non-contextual diagonal matrix. We refer to the Frobenius projection as *FROB*, the Wasserstein projection as *W2*, and the KL projection as *KL*.

Table 3.1 gives an overview of the final performance and convergence speed on the Mujoco benchmarks, Figure A.2 in the appendix displays the full learning curves. After each epoch, we evaluate five episodes without applying exploration noise to obtain the return values. Note that we initially do not include the entropy projection to provide a fair comparison to PPO. The results show that TRPL is able to perform similarly or better than PPO and PAPI across all tasks. While the performance on the Hopper-v2 is comparable, the projections significantly outperform all baselines on the HalfCheetah-v2. The KL projection even demonstrates the best performance on the remaining three environments. Besides that, the experiments present a relatively balanced performance between projections, PPO, and PAPI. The differences are more apparent when comparing the projections to PPO-M, which uses the same implementation details as our projections. The asymptotic performance of PPO-M is on par for the Humanoid-v2, but it convergences much slower and is noticeably weaker on the remaining tasks. Consequently, the approximate trust region of PPO alone is not sufficient for good performance, only paired with certain implementation choices. Still, the original PPO cannot fully replace a mathematically sound trust region as ours, although it does not exhibit a strong performance difference. For this, Figure 3.3 visualizes the mean KL divergence at the end of each epoch for all methods. Although neither W2 nor Frobenius projection use the KL, we leverage it here as a standardizing measure to compare the change in the policy distributions. All projections are characterized by an almost constant change, whereas for PPO-M the changes are highly inconsistent. The code-level optimizations of PPO can mitigate this to some extent but cannot properly enforce the desired constant change in the policy distribution.

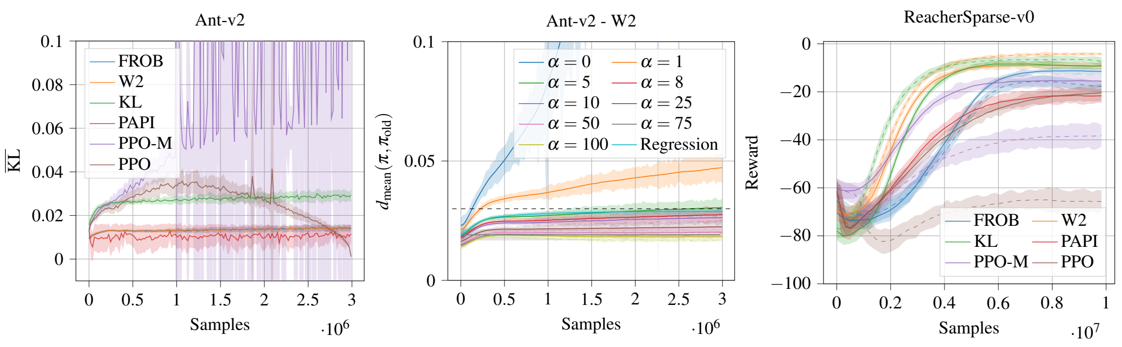


Figure 3.3: (Left): Mean KL divergence for Ant-v2 as a standardizing measure to compare the policy changes among all methods. (Center): Mahalanobis distance between the mean values of the unprojected and old policy when using different α in comparison to a full regression. The mean bound for the W2 projection is set to 0.03 (dotted black line). (Right): Mean cumulative reward with 95% confidence interval based on 40 seeds for the semi-sparse 5-link Reacher task. For each method, besides PAPI, we train policies with (dashed) and without (solid) contextual covariances.

In particular, we have found that primarily the learning rate decay contributes to the relatively good behavior of PPO. Albeit, PAPI provides a similar principled trust region projection as we do, it still has some inconsistency by approaching the bound iteratively.

Entropy Control To demonstrate the effect of combining TRPL with entropy control, as described in Section 3.3.2, we evaluate all Mujoco tasks again for this extended setting. The target entropy in each iteration i is computed by exponentially decaying the initial entropy $\mathcal{H}_0$ to κ with temperature τ as $\kappa + (\mathcal{H}_0 - \kappa)\tau^{\frac{10i}{N}}$, where N is the total number of training steps. The bottom of Table 3.1 shows the results for our projections with entropy control. Especially on the more complex tasks with more exploration, all three projections significantly benefit from the entropy control. Their asymptotic performance for the HalfCheetah-v2, Ant-v2, and Humanoid-v2 increases and yields a much faster convergence in the latter. For the other Mujoco tasks the performance remains largely constant since the complexity of these tasks is insufficient to benefit from an explicit entropy control, as also noted by Pajarinen et al. (2019) and Abdolmaleki et al. (2015).

Contextual Covariances. To emphasize the advantage of *state-wise* trust regions, we consider the case of policies with state-dependent covariances. Existing methods, such as PPO and TRPO, are rarely used in this setting. In addition, PAPI cannot project the covariance in the contextual case. Further, Andrychowicz et al. (2020) demonstrated that for the standard Mujoco benchmarks, contextual covariances are not beneficial in an on-policy setting. Therefore, we choose to evaluate a task motivated by optimal control which benefits from a contextual covariance. We extend the Mujoco *Reacher-v2* to a 5-link planar robot, the distance penalty to the target is only provided in the last time step, $t = 200$, and the observation space also contains the current time step t. This semi-sparse reward specification imposes a significantly harder exploration problem as the agent is only provided with feedback at the last time step. We again tuned all hyperparameters using Optuna (Akiba et al., 2019) and did not include the entropy projection. All feasible approaches are compared with and without contextual covariances, the results therefore are presented in Figure 3.3 (right). All three projections significantly outperform the baseline methods with the non-contextual covariance. Additionally, both the W2 and KL projections improve their results in the contextual case. In contrast, all baselines decrease in performance and are not able to leverage the advantage of contextual information. This poor performance mainly originates from incorrect exploitation. PPO reduces the covariance too quickly, whereas PAPI reduces it too slowly, leading to suboptimal performance for both. The Frobenius projection, however, does not benefit from contextual covariances either, since numerical instabilities arise from too small covariance values close to convergence. Those issues can be mitigated using a smaller covariance bound, but they cannot be entirely avoided. The KL projection, while yielding the best results throughout all experiments, relies on numerical optimization. Generally, this is computationally expensive, however, by leveraging an efficient C++ implementation this problem can be negated (see Appendix A.2.5). As a bonus, the KL projection has all properties of existing KL-based trust region methods that have monotonic improvement guarantees. Nevertheless, for quick benchmarks, the W2 is preferred, given it is slightly less prone to hyperparameter choices and does not require a dedicated custom implementation.

Trust Region Regression Loss. Lastly, we investigate the main approximation of our approach, the trust region regression loss (Equation (3.12)). In the following ablation, we evaluate how different choices of the regression weight α affect constraint satisfaction. Figure 3.3 (center) shows the Mahalanobis distance between the unprojected and the old policy means for different α values. In addition, for one run we choose $\alpha = 0$ and execute the trust region regression separately after each epoch for several iterations. One key observation is that decreasing the penalty up to a certain threshold leads to larger changes in the policy and pushes the mean closer to its maximum bound. Intuitively, this can be explained by the construction of the bound. As the penalty is added only to the loss when the bound is violated, larger changes in the policy are punished while smaller steps do not directly affect the loss negatively. By selecting a larger α, this behavior is reinforced. Furthermore, we can see that some smaller values of α yield a behavior that is similar to the full regression setting. Consequently, it is justified to use a computationally simpler penalty instead of performing a full regression after each epoch.

3.5 Discussion and Future Work

In this work, we proposed differentiable projection layers to enforce trust region constraints for Gaussian policies in deep RL. While being more stable than existing methods, they also offer the benefit of imposing the constraints on a state level. Unlike previous approaches that only constrain the expected change between successive policies and for whom monotonic improvement guarantees thus only hold approximately, TRPL can constrain the maximum change. Our results illustrate that trust regions are an effective tool in policy search for a wide range of different similarity measures. Apart from the commonly used reverse KL, we also leverage the Wasserstein distance and Frobenius norm. We demonstrated the subtle but important differences between those three different types of trust regions and showed our benchmark performance is on par or better than existing methods that use more code-level optimizations. For future work, we plan to continue our research with more exploration-heavy environments, in particular with contextual covariances. Additionally, more sophisticated heuristics or learning methods could be used to adapt the trust region bounds for better performance. Lastly, we are interested in using our trust region layers for other deep RL approaches, such as actor-critic methods.

Chapter 4

MP3: Movement Primitive-Based (Re-)Planning Policy

This chapter builds upon *trust region projection layer* (TRPL) from the previous chapter, integrating it with insights from classical robotics. By doing so, we shift our focus from the conventional action space to the trajectory space, which has significant advantages. Learning a deep policy in trajectory space not only allows to view and explore the problem from a more holistic view but also harnesses the scalability and generalization capabilities of deep *reinforcement learning* (RL) as well as ensures smoothness, energy efficiency, and compatibility with rewards beyond the typical dense rewards prevalent in deep RL. The results of this work, initially published in Otto et al. (2022), were expanded upon in Otto et al. (2023). The extension, conducted in collaboration with Hongyi Zhou, introduces trajectory replanning, enabling more versatile and reactive behavior. Hongyi Zhou's contributions include the replanning and task adaptation experiments for box pushing and table tennis, along with additional ablation studies.

4.1 Introduction

Movement primitives (MPs) are a powerful and versatile method for representing robot trajectories with a concise set of parameters. This makes MPs an easy-to-use and efficient tool for RL tasks. By directly exploring the space of desired trajectories, MPs simplify the exploration process and produce smooth, "robot-friendly" motions. As a result, *RL with MPs* (MPRL) has been responsible for many of the early successes in robot RL, with notable applications in *table tennis* (Mülling et al., 2013; Gomez-Gonzalez et al., 2016), *ball in a cup* (Kormushev et al., 2010), and *pancake flipping* (Kormushev et al., 2013). However, these algorithms were previously limited to simple setups and could only learn single stroke-based open-loop motions. Consequently, these motions were difficult to adapt to task variations or during motion execution. With increasing computation power, the field of deep RL rose. These methods can learn complex closed-loop sensorimotor policies, which is the reason why this research field dominated recent years. In this paper, we extend the work from Otto et al. (2021, 2022) to address the shortcomings of previous RL with MPs approaches and propose a new method that integrates MPs into a deep RL pipeline. Our method allows for non-linear adaptation and replanning during the execution of the MP, while still maintaining the beneficial exploration properties of the MP framework in the context of RL.

Traditional deep RL methods use a step-based policy, where at each time step the policy explores the atomic action space. During interaction with the environment, the agent collects state, action, and reward data points at each time step, which are used to update the policy. Although using every atomic action generates a vast amount of data points for the policy update, it also complicates exploration due to the typical random walk behavior and introduces a lot of noise in the policy evaluation process (see Figure 4.2). Therefore, these methods often rely on informative reward signals throughout the interaction sequence, making them less effective in sparse or temporally sparse settings where feedback from the environment is delayed. Moreover, step-based exploration can result in slower convergence and jerky, potentially dangerous behavior for the agent.

In contrast, MPRL is typically based on *episode-based RL* (ERL) (Deisenroth et al., 2013; Abdolmaleki et al., 2015; Daniel et al., 2012; Otto et al., 2022). ERL methods learn to parameterize a desired trajectory used for a controller based on a task description known as the context, which remains fixed throughout the entire episode. For example, in a table tennis scenario, the context is given by the target position where the robot has to return the ball. These methods explore the trajectory space, meaning that a parameter is sampled given the context only once at the beginning of the episode and executed without resampling. This exploration strategy results in time-correlated exploration, smooth behaviors, and improved performance in sparse or non-Markovian reward settings (Otto et al., 2022). Yet, only one data point is generated per executed trajectory, as these algorithms collect only one context-parameter pair per episode. This data collection procedure limits sample efficiency. In our recent work (Otto et al., 2022), we integrated ERL with MPs into a deep *policy gradient* (PG) algorithm that is based on TRPL (Otto et al., 2021). While this algorithm can non-linearly adapt the parameters of the MP to the given context and achieve high-quality policies for complex robotic tasks, it is inherently constrained to generating open-loop trajectories that cannot be adapted or adjusted during execution.

This work is an extension of Otto et al. (2022), where we also add learning non-linear replanning policies instead of just the initial adaptation of the MP to the context, combining the benefits of ERL with MPs and *step-based RL* (SRL) methods. We still explore the trajectory space, yet, the agent is now able to change the desired trajectory during an episode, enabling it to adapt its behavior to unpredictable changes in the environment. In the original paper (Otto et al., 2022), *probabilistic movement primitives* (ProMPs) (Paraschos et al., 2013) were used as the MP representation. However, ProMPs are unable to generate smooth trajectories if the MP parameters are changed during motion execution. As a result, ProMPs are not suitable for learning replanning policies. In contrast, in this paper, we employ the recently introduced *probabilistic dynamic movement primitives* (ProDMPs) (Li et al., 2022) to address the limitations of commonly used MPs (Schaal et al., 2005; Schaal, 2006; Paraschos et al., 2013). ProDMP constitute a reformulation of the popular *dynamic movement primitive* (DMP) (Schaal, 2006; Ijspeert et al., 2013) framework which is better suited to be integrated into a neural network architecture as it does not require expensive numerical integration. Additionally, ProDMP can use any initial state as the initial condition, allowing for the generation of smooth desired trajectories even when applying replanning. Our policies are parameterized with neural networks and efficiently trained using TRPL (Otto et al., 2022), which has demonstrated significantly improved stability and quality of the learned policy in comparison to other PG methods, such as *proximal policy optimization* (PPO) (Schulman et al., 2017). We demonstrate the effectiveness of our method by presenting various complex simulated robotic tasks, such as robot table tennis, beer-pong, a complex box-pushing task, and large-scale manipulation tasks on Meta-World (Yu et al., 2019). We compare our approach to state-of-the-art SRL and ERL methods and illustrate improved performance in sophisticated, sparse reward settings and settings that require replanning.

4.2 Deep Reinforcement Learning with Movement Primitives

In this work, we present a framework to effectively combine MPs with deep RL methods. This framework consists of three major components (see Figure 4.1):

- One RL policy which takes the environment observation as input and outputs an MP weight vector that is used for multiple time steps.
- One MP model which uses the weight vector as input to generate a desired trajectory.
- One low-level controller that converts the desired trajectory into raw actions and interacts with the environment.

This approach is simple but highly versatile. Theoretically, any policy search algorithm applicable to continuous action spaces can be used here. However, it is worth noting that the dimensionality of the weight space of the MP is usually larger than the raw action space. Therefore, the policy search algorithm used must be able to explore high-dimensional spaces efficiently. Furthermore, MPs can be replaced by any parameterized trajectory generator, as long as the desired trajectory can be uniquely determined by a weight vector. We specifically chose MPs in this work because they are capable of generating smooth trajectories and allowing effective replanning. In addition, the planning horizon (length of the generated trajectory before a new weight vector is chosen) can vary from a single step to the entire episode. Two special cases correspond to two common RL paradigms: (i) When the planning horizon is equal to one, our framework is similar to an SRL algorithm (although with a higher dimensional action space). (ii) When it is equal to the episode length, the framework corresponds to an ERL algorithm. We refer

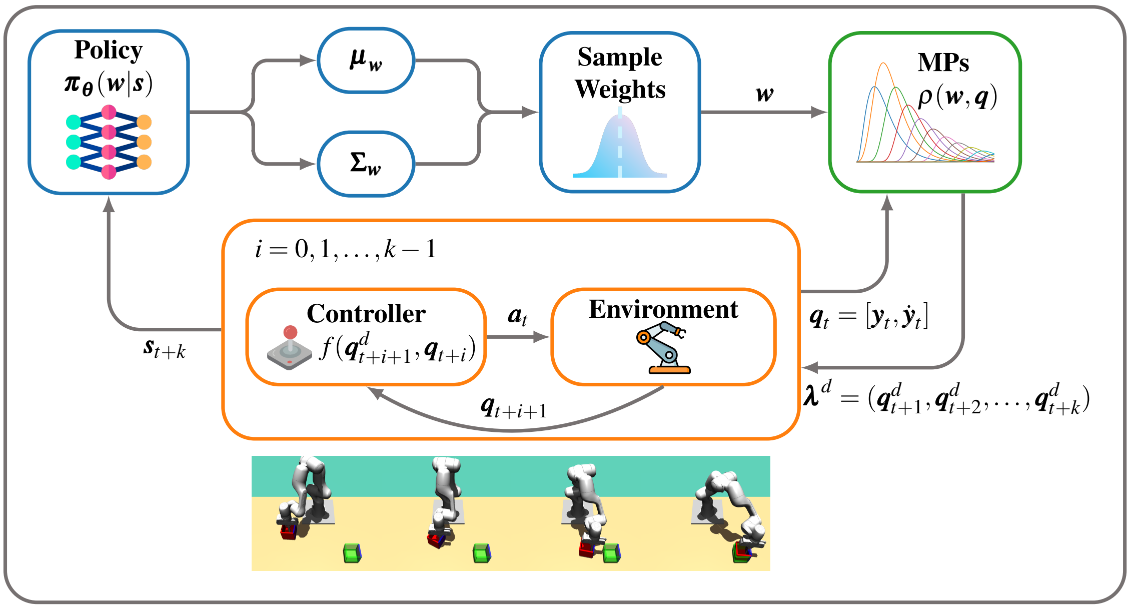


Figure 4.1: This figure provides an overview of the proposed framework that combines deep RL with MPs. Instead of generating a raw action directly, the policy generates a set of weights that parameterizes an MP. The MP predicts a desired trajectory given the weights and initial conditions, which is then converted to raw actions using a tracking controller.

to these cases as *movement primitive-based planning policy* (MP3) and *MP3-Black Box* (MP3-BB), respectively.

4.2.1 Reinforcement Learning Objective with Movement Primitives

While traditional SRL methods rely on single raw actions $\boldsymbol{a}_t \in \mathcal{A}$ per time step, we train a policy to select a weights vector $\boldsymbol{w}_t \in \mathcal{W}$ in MP's parameter space $\mathcal{W}$. The weights vector is then translated to a desired trajectory of the proprioceptive states $\boldsymbol{\lambda}^d = (\boldsymbol{q}^d_{t+1}, \boldsymbol{q}^d_{t+2}, \ldots, \boldsymbol{q}^d_{t+k})$, where $q^d_t = [\boldsymbol{y}^d_t, \dot{\boldsymbol{y}}^d_t]$ consists of desired position $\boldsymbol{y}^d_t$ and desired velocity $\dot{\boldsymbol{y}}^d_t$ at time step t, and k denotes the planning horizon. Given the desired trajectory and the measured proprioceptive state, a tracking controller $f(q^d_t, q_t)$ decides the action at each step, resulting in a trajectory in the raw action space $(\boldsymbol{a}_{t+1}, \boldsymbol{a}_{t+2}, \ldots, \boldsymbol{a}_{t+k}) \in \mathcal{A}$. In contrast to the step-wise sample $(\boldsymbol{s}_t, \boldsymbol{a}_t, R_t)$ used in SRL, we use temporarily-abstracted samples of the form $(\boldsymbol{s}_t, \boldsymbol{w}_t, R^k_t)$. The reward $R^k_t = R_{t:t+k-1}$ of each trajectory segment is defined as the cumulative reward over all the segment's time steps t to $t+k-1$

$$R^k_t(\boldsymbol{s}_t, \boldsymbol{w}_t) = \sum_{i=0}^{k-1} \gamma^i r(\boldsymbol{s}_{t+i}, \boldsymbol{a}_{t+i}), \tag{4.1}$$

where a_t and s_t are the executed actions and observed states following the desired trajectory and tracked by the controller. While our approach supports different k for each segment, we only consider planning segments with equal length in this work. We can compute the episode return by taking the cumulative discounted sum of the segment rewards. Using the notation from above, we can express this as

$$G_t^k = \sum_{i=0}^{\lceil T/k-1 \rceil} \gamma^{ik} R_{t+ki}^k(\boldsymbol{w}_{t+ki}, \boldsymbol{s}_{t+ki}), \tag{4.2}$$

where $\gamma \in (0,1]$ is the discount factor. It is worth noting that there are two special cases to consider. In the black-box setting, in other words, when the MP parameters are chosen only at the beginning of the episode, then $k = T$ and the segment reward equals the episode return

$$R_0^{T-1} = \sum_{t=0}^{T-1} \gamma^t r(a_t, s_t). \tag{4.3}$$

The second special case is step-based RL. That is, we choose a new parameter vector at every time step, *i.e.* $k = 1$. In this case, segment reward is equivalent to step reward

$$R_t^1 = r(a_t, s_t). \tag{4.4}$$

This gives the insight that we can alter between SRL and ERL by choosing different planning horizons k.

4.2.2 Policy-gradients for MP weight-selection policies

With these rewards, we can now also define matching value and advantage functions

$$V^\pi(\boldsymbol{s}) = \mathbb{E}\left[G_t^k | \boldsymbol{s}_t = \boldsymbol{s}; \pi_\theta\right] \qquad A^\pi(\boldsymbol{s}, \boldsymbol{w}) = \mathbb{E}\left[G_t^k | \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{w}_t = \boldsymbol{w}; \pi_\theta\right] - V^\pi(\boldsymbol{s}). \tag{4.5}$$

Following the step-based PG (Williams, 1992; Schulman et al., 2015a), we optimize the advantage function using the likelihood ratio gradient and an importance sampling estimator. The resulting objective

$$\hat{J}(\pi_{\boldsymbol{\theta}}, \pi_{\boldsymbol{\theta}_{\text{old}}}) = \mathbb{E}_{(\boldsymbol{s},\boldsymbol{w})\sim p(\boldsymbol{s}),\pi_{\boldsymbol{\theta}_{\text{old}}}}\left[\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s})}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{w}|\boldsymbol{s})} A^{\pi_{\boldsymbol{\theta}_{\text{old}}}}(\boldsymbol{s}, \boldsymbol{w})\right], \tag{4.6}$$

is maximized with respect to $\boldsymbol{\theta}$, with $\pi_{\boldsymbol{\theta}_{\text{old}}}$ being the old behavior policy used for sampling. We can further make use of a learned state-value function $V_\phi(\boldsymbol{s}) \approx V^\pi(\boldsymbol{s})$ for the advantage estimator, which is approximated by optimizing

$$\arg\min_{\phi} \mathbb{E}_{(\boldsymbol{s},\boldsymbol{w})\sim p(\boldsymbol{s}),\pi_{\boldsymbol{\theta}_{\text{old}}}} \left[\left(V_\phi(\boldsymbol{s}) - G_t^k \right)^2 \right]. \tag{4.7}$$

This formulation also enables the use of advantage estimation methods, such as general advantage estimation (Schulman et al., 2016). During the update of the policy, neither the MP $\rho(\boldsymbol{w},\boldsymbol{q})$ nor the controller $f(\boldsymbol{q}^d_{t+1},\boldsymbol{q}_t)$ are needed, *i.e.*, our approach would work with any form of parametrizable controller.

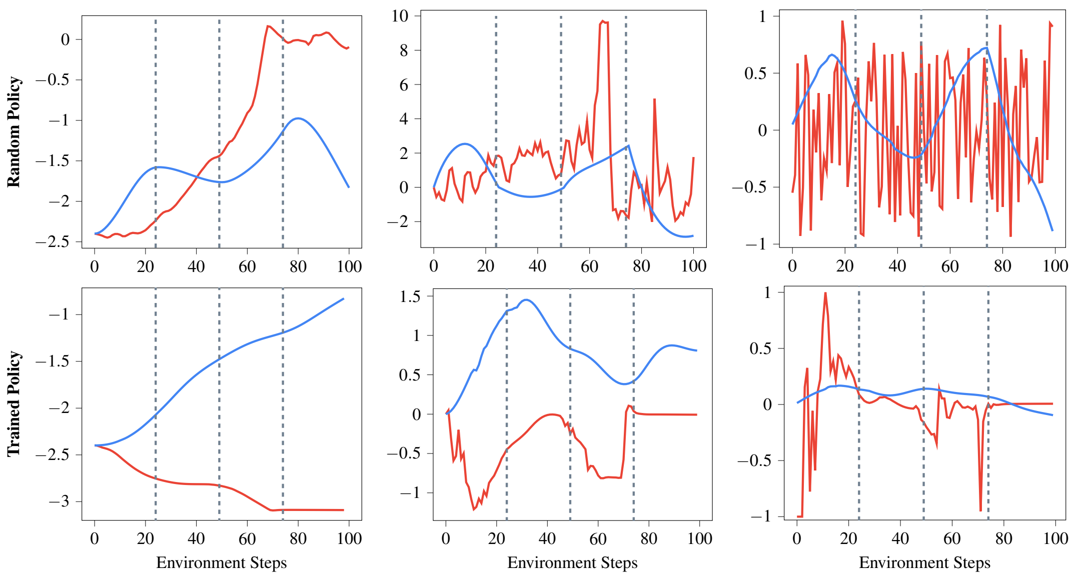
4.2.3 Choice of the Deep Reinforcement Learning Algorithm

Most deep RL methods can theoretically be used to train policies in the weight space of MPs. Yet, training in this space requires learning policies with a higher degree of precision compared to the step-based case as the selected action is active for more time steps with no opportunity for correction during the trajectory segment. To address this issue, we chose TRPL as it has been shown to be more stable and accurate than other RL methods (Otto et al., 2021). TRPL implements exact trust regions for policy updates and enforces them per state, while most other deep RL methods (Schulman et al., 2015a, 2017; Akrour et al., 2019) only provide approximate trust region updates that are enforced for the average policy change across all states.

4.2.4 Choice of the Planning Horizon

Our method harnesses the merits of two common RL paradigms: *step-based RL* (SRL) and *episode-based RL* (ERL). The agent's behavior can seamlessly switch between the two paradigms according to the planning horizon k. As discussed previously, there are two special cases when selecting the planning horizon.

Black-Box Setting The first special case arises when the planning horizon is equal to the episode length, that is, $k = T$. In this case, the agent generates only one desired trajectory for the entire episode, similar to an open-loop motion planner. Since this setting treats RL as a black-box optimization problem, we refer to it as MP3-BB in the following discussion. MP3-BB does not assume the existence of a step reward, but instead evaluates the performance of the entire trajectory as a whole. The black-box nature facilitates dealing with sparse and non-Markovian rewards, leading to a more intuitive reward design. Another advantage is state abstraction. The MP3-BB agent does not need any intermediate state information about the task execution that varies during the execution process (such as joints position/velocity, etc.), but focuses only on the critical information that defines the essence of the tasks (for example goals, obstacles, etc.), this is also referred to as context. However, MP3-BB presents some challenges. Although



(a) Position (rad) (b) Velocity (rad/s) (c) Torque (Nm)

Figure 4.2: This figure presents a comparison between step-level exploration (red) and the proposed trajectory-level exploration using MPs (blue). We generated rollouts in box-pushing environments where the raw action space is joint torques. For the MP setting, we used ProDMP and performed replanning every 25 steps. The trajectories of the third joint are plotted. In the top row, we randomly sampled rollouts from untrained policies to demonstrate the initial exploration. In the bottom row, we evaluated two policies trained with PPO (red) and MP3 (blue) respectively, using 40M environment interactions. The results show that MP3 enables smooth trajectory generation at both exploration and evaluation processes.

the reduction of the observation dimension to the context space results in a modest impact on sample efficiency, the high cost of a single sample makes it less sample-efficient. Moreover, its black-box nature limits its applicability in dynamic environments, where the agent must adapt to environmental changes during execution.

Step-Based Setting At the opposite end of the planning horizon spectrum is the case where $k = 1$. Here, the agent only executes the desired trajectory for one step, after which it generates a new plan, repeating this loop throughout the episode similar to the SRL setting. However, using MPs as a trajectory generator instead of raw actions has two essential differences. Firstly, the use of MPs guarantees second-order smoothness (position and velocity), resulting in a more consistent and smooth behavior during exploration and evaluation (see Figure 4.2). Secondly, the number of MP basis functions is a hyperparameter that allows for scaling the dimension of the action space from the size of raw actions to arbitrary dimensions, enriching the model's design choices but also increasing the complexity. Although increasing the dimensionality of the action space may seem to increase exploration difficulty, it can still benefit from the smooth trajectories generated by MPs. Nonetheless, we did not observe significant improvements using this setting over the standard SRL setting, and our discussion only aims to highlight the flexibility of the proposed method.

Re-planning with MPs The more general case falls somewhere between the two extremes of SRL and ERL. In this approach, the agent generates a new desired trajectory after executing the current trajectory for a predefined number of steps ($1 < k < T$). This method leverages the strengths of both SRL and ERL while addressing some of their shortcomings:

1. In SRL, stochastic raw action selection often results in jerky random walk behavior that does not fully explore the trajectory space of the agent. In contrast, our approach explores the weight space of MPs, leveraging the MP's smoothness guarantees for more consistent and effective exploration (see Figure 4.2).

2. Stochastic raw action selection in SRL also results in noisy returns (see Figure 4.2), translating to high variance in the PG estimation. The smaller number of exploration steps in the re-planning setting yields less variance in gradient estimation, leading to a more stable policy update.

3. Trajectory-level exploration encapsulates the temporal abstraction within each trajectory segment, reducing the number of decisions to make for each episode and improving the agent's ability to handle the sparsity in the reward function.

4. The policies trained by SRL often struggled to generate smooth trajectories (see Figure 4.2) due to the lack of continuity between consecutive steps. In contrast, policies trained with our approach can generate smooth trajectories in both the exploration and evaluation processes.

5. The ERL agent only makes decisions at the beginning of each episode and treats each episode as a black box, thus can deal with the temporal sparsity and non-Markovian property in rewards. However, the black-box perspective also limits their ability to address observation noises and dynamics in the environment, as a result, ERL agents cannot adapt their plan according to the changes in the environment during execution, which makes them less flexible and robust compared to SRL. Our approach addresses this shortcoming by incorporating periodic re-planning during online execution.

Many SRL algorithms use a similar design called *frame-skipping*, which can help with the partial observability of some Atari games (Braylan et al., 2015). However, frame-skipping just repeats the same action for the "skipped" frames, limiting the trajectory's expressive capacity. In contrast, planning with MP can "skip" more frames without compromising expressiveness.

4.2.5 Adapting to Different Reward Settings

The proposed algorithm offers a significant advantage in the flexibility to adapt to different reward signal designs. By considering the information available at each step, the rewards can be classified into three distinct categories.

Dense Rewards Dense rewards provide task-related feedback signals at each time step. For example, in a reacher task where the objective is to reach a desired point with the end-effector, a common task-related feedback could be the distance between the end-effector and the desired point. Well-shaped, dense rewards are crucial for the success of most SRL algorithms. However, designing efficient dense rewards can already be challenging for tasks where evaluating the quality of the action step-wise is difficult, such as in beer pong and table tennis.

Sparse Rewards Sparse rewards provide task-related feedback only when specified conditions are satisfied. These conditions can be either temporal-related (for instance, providing reward only at episode end) or task-related metrics (for instance, providing reward only when the end-effector is close enough to the desired point). Our approach is less affected by the temporal sparse setting because we use highly temporal-abstracted samples in policy updates. In contrast to dense reward signals, sparse rewards are usually more intuitive to design and more suitable for the task where task completion at a specific time point (for instance, the episode end) is desired. Take the aforementioned reacher task as an example, dense reward based on distance error at every step implicitly encourages the agent to reach the desired point as fast as possible, leading to policies with large acceleration and overshooting. This is undesirable if we only want to reach the target at the end. Temporal sparse rewards address this issue by rewarding the agent based on the final state.

Non-Markovian Rewards Rewards that provide task-related feedback without adhering to the Markovian condition, *i.e.*, the reward signal is not fully determined by the current state-action pair but also incorporates the past states and actions, are called non-Markovian rewards. The non-Markovian property exists widely in RL tasks. For instance, in playing table tennis, once the ball is hit, the agent's actions no longer influence the trajectory of the ball. Therefore, the reward is not conditioned solely on the action at that time step, but also on the actions preceding the hit. This setting is extremely challenging for SRL algorithms as their policy updates rely on the Markovian assumption. Our approach with the black-box setting can leverage non-Markovian rewards effectively by treating the entire episode trajectory as a single sample.

4.3 Experiments

For our evaluation, we begin by demonstrating the effectiveness of our method in handling sparse rewards, improving precision, and energy efficiency without replanning, that is, in the black-box setting with $k = T$. We investigate challenging control problems that are typically difficult to solve in the standard step-based setting. Next, we conduct a large-scale study on all 50 Meta-World tasks (Yu et al., 2019) to showcase our competitive performance on various robot manipulation tasks that come with highly shaped dense rewards with and without replanning. Finally, we evaluate our method with replanning for several tasks with changing goals or high uncertainties and perform a thorough ablation study. We compare our methods, which we call MP3 and MP3-BB for the replanning and black-box cases respectively, against several other step-based methods, including PPO (Schulman et al., 2017), TRPL (Otto et al., 2021), *soft actor critic* (SAC) (Haarnoja et al., 2018), and *neural dynamic policies* (NDP) (Bahl et al., 2020), as well as a deep *evolution strategies* (ES) (Salimans et al., 2017), the linear adaption method *contextual model-based relative entropy stochastic search* (CMORE) with ProMPs (Tangkaratt et al., 2017) as well as MP3-PPO and MP3-BB-PPO, which are equivalent to MP3 and MP3-BB but trained with PPO instead of TRPL. For the ERL methods (MP3-BB and MP3-BB-PPO), we leverage ProMPs for motion generation and for the replanning versions (MP3 and MP3-PPO), we use ProDMPs.
It is worth noting that the authors of NDP report their performance in terms of the used samples rather than environment interactions (the original work only uses every fifth interaction). However, we believe that reporting the total number of environment interactions leads to a fairer comparison and also helps to explain the relatively poor performance of NDP in our experiments. For both MP3-BB and MP3-BB-PPO, we provide only the context information $\boldsymbol{c}$ instead of leveraging the full state observation $\boldsymbol{s}$. The context information $\boldsymbol{c}$ is a subset of the observation space that is randomly initialized after each reset and includes the stochastic elements, such as the goal or object positions. Unless otherwise specified, we measure the trajectory segment performance $R_{t:t+k}(\boldsymbol{w}, \boldsymbol{s})$ as the cumulative trajectory return. We evaluate our method on 20 different seeds and compute ten evaluation runs after each iteration. To report our results, we use the *interquartile mean* (IQM) with a 95% stratified bootstrap confidence interval and performance profiles where feasible (Agarwal et al., 2021). For a detailed description of the hyperparameters used in the evaluation, please refer to Appendix B.4.

4.3.1 Black-Box Reinforcement Learning

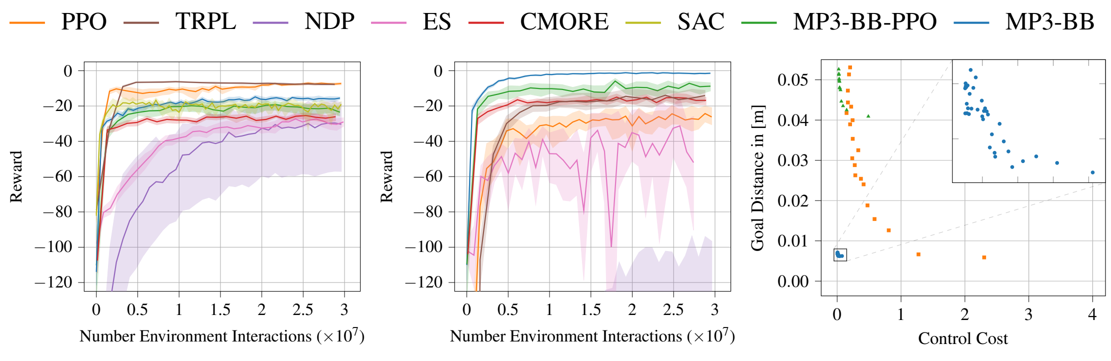
Performance As a preliminary task, we extend the reacher from OpenAI gym (Brockman et al., 2016) by using five actuated joints and by restricting the context space, *i.e.* the location of the goal, to $y \geq 0$. This results in an increased control complexity but slightly decreased task complexity. For a detailed environment description, please see Appendix B.2.1. We investigate two types of rewards: a dense reward equivalent to the original reacher, and a sparse reward that provides only the distance to the goal in the last episode time step. We study the sparse reward setting as it is better suited to generate energy-efficient motions. Yet, it is also more difficult to learn. While MP3-BB and MP3-BB-PPO can solve the task for both rewards, NDP and ES fail in both cases (Figures 4.3a and 4.3b). PPO and TRPL achieve a slightly better asymptotic performance than MP3-BB in the dense setting, but are unable to consistently reach the goal for the sparse reward signal. Although SAC achieves a comparable performance to MP3-BB in the dense setting, it cannot leverage the sparse reward (see Appendix B.3). CMORE performs reasonably well, however, it is only able to cover part of context space due to its linear adaption strategy.

To demonstrate the learning capabilities of all algorithms in handling sparse rewards within a more complex scenario, we conduct evaluations on a box-pushing task. The goal is to move a box to a given goal location and orientation using a simulated Franka Emika Panda. The goal location and orientation are randomized at the beginning of each episode. For a detailed environment description, please see Appendix B.2.2. Similar to the reacher task, we consider a dense reward signal and a temporal sparse reward signal. The dense reward is based on the position and orientation error for each time step, while the temporal sparse reward depends only on the distance errors at the last time step. We observed that the temporal sparse reward used in the original work leads to policies that pass through the target location at episode end. To address this issue, we increased the control cost penalty and introduced a joint velocity penalty at the episode end. We re-tuned baselines that exhibited competitive performance in the original setting to accommodate the new reward setup.

While MP3 and MP3-BB achieve the highest precision and sample efficiency in dense reward setting (Figure 4.4a), all SRL algorithms, excluding SAC, yielded acceptable performance at the end. The performance of SAC was adversely affected due to the penalties associated with constraint violation and control cost. In the sparse reward setting(Figure 4.4b), all the algorithms experience a certain degree of performance decline, while SRL algorithms encounter substantial performance degradation, the MP-based algorithms are capable of maintaining reasonable performance.

Although dense rewards may perform well in certain tasks (*e.g.* reacher and box-pushing), there are two main reasons to consider sparse rewards. First, sparse rewards are usually easier to design because we only need to consider the state at the last time step. Another reason is that dense rewards force the agent to reach the goal as fast as possible, which typically yields energy-inefficient motions. In contrast, sparse rewards penalize the goal distance only in the final time step, while accounting for energy cost in each time step.

Energy Efficiency To illustrate the trade-off between precision and energy efficiency, we analyzed the final behaviors of both reward setups with different action penalty factors in the reward function. For each of these factors, we computed the average precision and energy consumption. Our results, shown in Figures 4.3c and 4.4c, demonstrate

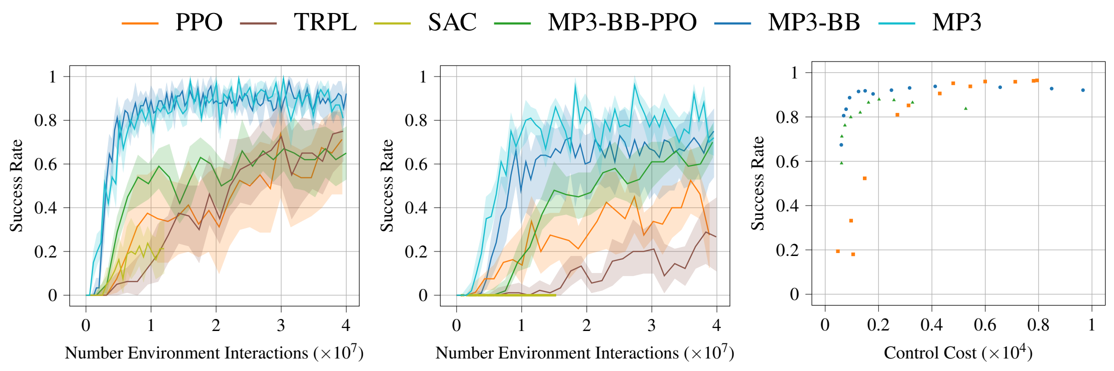


(a) 5D Reacher - Dense (b) 5D Reacher - Sparse (c) 5D Reacher - Energy

Figure 4.3: The Figures 4.3a and 4.3b show the learning curve for the 5D reacher task with dense and sparse reward signals. Although TRPL and PPO achieve the best performance in the dense reward setting, both of them are struggling under the sparse reward setting. In contrast, MP3-BB achieves the best performance in the sparse reward setting. Figure 4.3c shows the trade-off between the energy efficiency (sum of the squared control cost) and the task precision (distance to the goal at the last step). We average over 100 evaluation runs and all seeds and choose action penalty factors in the intervals $(0, 100]$. MP3-BB with sparse reward achieves the highest precision with much lower energy consumption compared to the PPO trained with the dense reward.

that decreasing the action penalty factor leads to higher task precision for all methods. However, for the dense reward, a high task precision comes with the cost of high energy penalties, whereas the sparse reward generates behavior of similar precision with one (box pushing) or even two (reacher) orders of magnitude less energy consumption. When analyzing the final behaviors, agents trained with the dense reward quickly move to the target and stay there, while the agents with the sparse reward reach the target only slightly before the specified end of the episode, resulting in a much slower, smoother, and more energy-efficient motion.

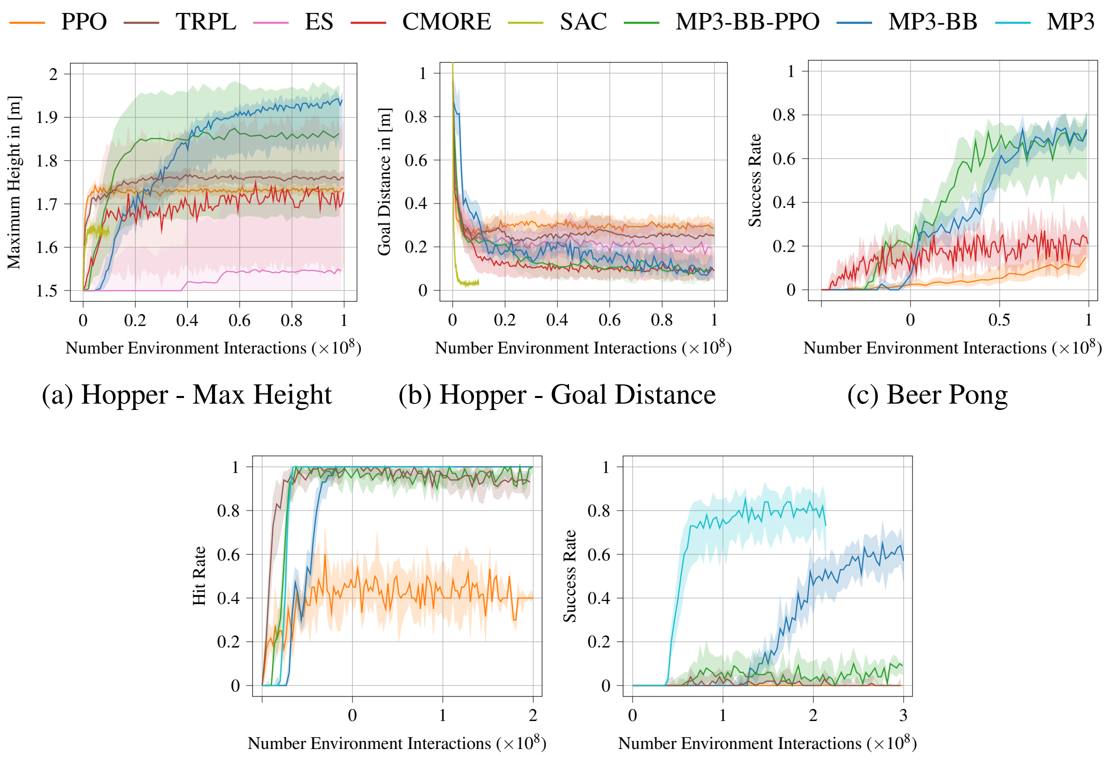
Dealing with non-Markovian Rewards To assess the effectiveness of our method in complex reward settings, we test it with non-Markovian rewards, which are particularly useful for robot learning tasks that require the agent to use feedback from the full trajectory history. We first use a modified version of the OpenAI Gym hopper (Brockman et al., 2016), which aims to jump as high as possible and land at a target location (see Appendix B.2.3). The non-Markovian reward measures the highest point of the jump and the shortest distance to the target during the episode. We compare our approach with two other methods, CMORE and ES, which also use the non-Markovian reward. In addition, we train three different algorithms, PPO, SAC, and TRPL, using a Markovian version that provides height and target distance at each time step. For this setting, we performed extensive reward shaping to optimize for maximum height and minimum target distance.



(a) Box Push - Dense (b) Box Push - Sparse (c) Box Push - Energy

Figure 4.4: Figure 4.4a shows the learning curve for the box-pushing task with a dense reward signal, while Figure 4.4b with a sparse reward signal. For the dense reward all the methods except SAC achieve remarkable performance, while MP3 and MP3-BB achieve higher sample efficiency, this is due to the efficient exploration in parameters space. All methods suffer performance decrease with sparse reward signal Figure 4.4b, MP3 and MP3-BB are less influenced as they leverage highly temporal abstracted samples. MP3-BB achieve similar sample efficiency with MP3 because it can take advantage of more compact observations (context observation). Figure 4.4c shows the tradeoff between energy efficiency(sum of squared action) and task precision (success rate). Similar to the energy plot in the 5D reacher task (Figure 4.3c), MP3-BB shows better energy-efficient behavior.

Overall, our results show that MP3-BB achieves higher jump heights with smaller target distances compared to most other methods (Figures 4.5a and 4.5b). While MP3-BB-PPO and CMORE can match the target distance, SAC can even exceed it, but none of them can reliably learn a good jump height. This can also be seen in the agent's behavior. MP3-BB charges energy and then jumps only once, whereas the step-based methods try to maximize height in each time step, resulting in multiple jumps in one episode. This illustrates the need for non-Markovian rewards to describe certain behaviors.
To further strengthen the ability of MP3-BB in solving tasks with non-Markovian rewards, we conduct experiments in a Beer pong environment(Celik et al., 2022). In this task, the goal is to throw a ball into a cup at various locations on a table. The return depends on the entire trajectory of the ball, which can be calculated by using information such as table contacts or the minimum distance to the cup (a detailed description of the task can be found at Appendix B.2.4). However, directly training PPO on such a reward as well as designing a Markovian version of the reward is both challenging in this case. To address this issue and make PPO a stronger baseline, we simplified the task for PPO by fixing the ball release time and considering the time between the ball release and the end of the ball trajectory as the last time step, allowing PPO to compute the reward similarly to the non-Markovian setting. This kind of simplification is unnecessary for ERL



(a) Hopper - Max Height (b) Hopper - Goal Distance (c) Beer Pong

(d) Table Tennis - Hit Rate (e) Table Tennis - Success Rate

Figure 4.5: The Figures 4.5a and 4.5b show the maximum jumping height of the hopper's center of mass and the target distance, respectively. With the non-Markovian reward, the hopper can jump approximately 20cm higher with increased goal precision. Figure 4.5c shows the beer pong task, where PPO struggles to throw the ball into the cup, even with a fixed optimal release point, while MP3-BB can consistently succeed in the task with dynamic release points. The success and hit rate of the table tennis task are shown in Figures 4.5d and 4.5e, respectively. The episode is considered a success when the agent hits the ball and successfully returns the ball near the goal position. The results demonstrate that MP3-BB consistently hits the ball and returns it in most cases, and MP3 is even able to improve that performance further.

algorithms. We use MP3-BB and CMORE with the non-Markovian reward and learn the ball release time as an additional controller parameter. See results at Figure 4.5c. We see that both MP3-BB and MP3-BB-PPO are able to throw the ball into the cup, while PPO struggles. Even CMORE can throw the ball reliably, but only for a subset of the context space. Interestingly, we again observe that MP3-BB-PPO has a larger confidence interval compared to MP3-BB, indicating that it is not always able to solve the task consistently. This behavior is similar to the jumping task, where we also observed a larger confidence interval for MP3-BB-PPO.
Finally, we trained agents in a simulated table tennis environment (Celik et al., 2022).

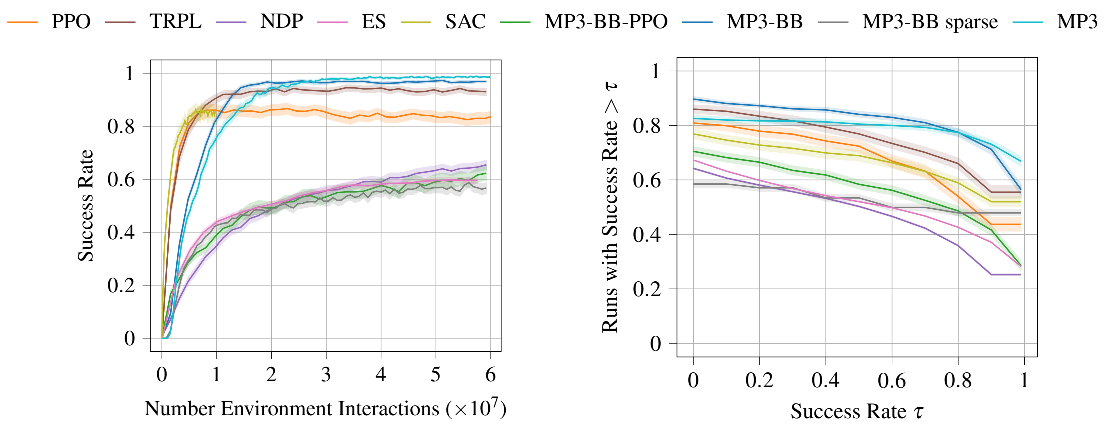
The context is four-dimensional, given by features of the incoming ball trajectory and the desired location for returning the ball (a detailed environment description is provided at Appendix B.2.5). The episode return depends on several factors, such as the minimum distance between the racket trajectory and ball trajectory, whether the racket hits the ball and the distance error between the ball's landing position and the target. It is worth noting that the agent's action cannot influence the return after hitting the ball. Thus, the return after hitting is not conditioned on the action at the current step but on the previous actions, which poses a significant challenge for SRL algorithms. For step-based methods, we consider the time after hitting the ball as one time step, akin to the beer pong task. For the MP3 and MP3-BB approaches, we also learn the start time of the trajectory and the speed of the desired trajectory (which is a parameter of the MP). Both parameters help to learn the precise timing required to play table tennis. The results (Figures 4.5d and 4.5e) show that MP3-BB succeeds in hitting the ball and returning it within the vicinity of the goal in more than 60% of the cases. MP3-BB-PPO and TRPL can hit the ball but fail to return it accurately, PPO even fails to hit the ball consistently.

In summary, MP3-BB provides an effective solution for handling non-Markovian reward structures, which are often more natural and easier to define than engineered dense rewards. By leveraging these reward structures, the method can facilitate the learning of more sophisticated and efficient behaviors, leading to improved overall performance.

4.3.2 Large Scale Robot Manipulation

We also showcase our ability to learn high quality policies on the Meta-World benchmark suite (Yu et al., 2019). To verify our algorithms can adapt to task variations and solve tasks consistently, we use a more rigorous evaluation protocol compared to the one used by Yu et al. (2019). In contrast to using a fixed context for each episode, we randomly generate new contexts with each reset. Additionally, instead of considering any instance of success during the episode as a task solved, we consider a task successfully solved only if the last time step successfully solves the task. The last time step success metric rules out cases where the task is only momentarily solved, but subsequently disrupted by random agent motion. We train individual policies for each environment but use the same hyperparameters. Our results (Figure 4.6a) show that PPO and TRPL achieve the best sample complexity, but MP3-BB performs competitively in terms of asymptotic performance and even outperforms PPO slightly in terms of asymptotic performance. Although the gap between PPO and MP3-BB in the aggregated view is relatively small, the corresponding performance profiles (Figure 4.6b) reveal that MP3-BB performs better above the 80% threshold. This means that MP3-BB finds more consistent solutions than PPO with higher precision and solves these tasks without failures. SAC performs similar to PPO, whereas NDP, ES, and MP3-BB-PPO are not achieving a competitive performance.

We conducted an additional ablation study in which we trained MP3-BB using sparse rewards, meaning only the final step reward of each episode was used. We denote this



(a) MetaWorld - Sample Efficiency (b) MetaWorld - Performance Profile

Figure 4.6: This plot shows the success rate (Figure 4.6a) for all 50 Meta-World tasks, and the corresponding performance profile (Figure 4.6b), which is the fraction of runs that perform above the threshold given by the x-axis. Despite having lower sample efficiency, MP3-BB achieves a higher asymptotic policy quality than PPO. Finally, MP3 can further improve the asymptotic performance while being slightly less sample efficient. According to the performance profile (Figure 4.6b), MP3-BB can solve more tasks (fewer runs with zero success rate) while MP3 can solve the tractable tasks more consistently (higher fraction of runs with $\geq 80\%$ success rate).

variant as MP3-BB sparse. While the *interquartile mean* (IQM) score is lower, it is still able to complete 50% of the tasks with a 100% success rate. This is a better performance than what we observed with PPO trained with dense rewards. Furthermore, the slope of the performance profile is quite small, indicating that nearly all the tasks that can be solved by the agent are solved with a high degree of accuracy.

4.3.3 Replanning with Movement Primitives

We evaluate our approach in the online replanning case by decreasing the planning horizon, such that $1 \leq k < T$, positioning it between SRL ($k = 1$) and ERL ($k = T$). This approach, which we refer to as MP3, offers two significant benefits. Firstly, it leads to a more precise policy due to the closed-loop nature of the method. Secondly, it enables the handling of environmental dynamics through online replanning. However, there are two reasons why it is advantageous to consider it as a complementary approach rather than a complete replacement for MP3-BB. First, this design choice may limit the ability to address non-Markovian rewards, although we have also observed good results in this setting. Second, using replanning necessitates incorporating the internal proprioceptive state into the observation space, whereas the black-box setting can leverage a more concise observation space known as the context space.

Quality of the Learned Policy We evaluated the performance of MP3 agents by conducting experiments in three challenging environments: the Meta-World benchmark suite (Yu et al., 2019) for large-scale robot manipulation, box-pushing with dense and sparse rewards, as well as table tennis with non-Markovian reward.

We kept the same planning horizon (20% of the max episode length) in all 50 Meta-World environments. While the step-based methods achieved the best sample efficiency for Meta-World (Figure 4.7a), MP3 slightly outperforms the black-box approach and reaches the best asymptotic IQM score. The performance profile (Figure 4.6b) further indicates that although the number of unsolvable tasks remains constant, the quality of the solved tasks improves.

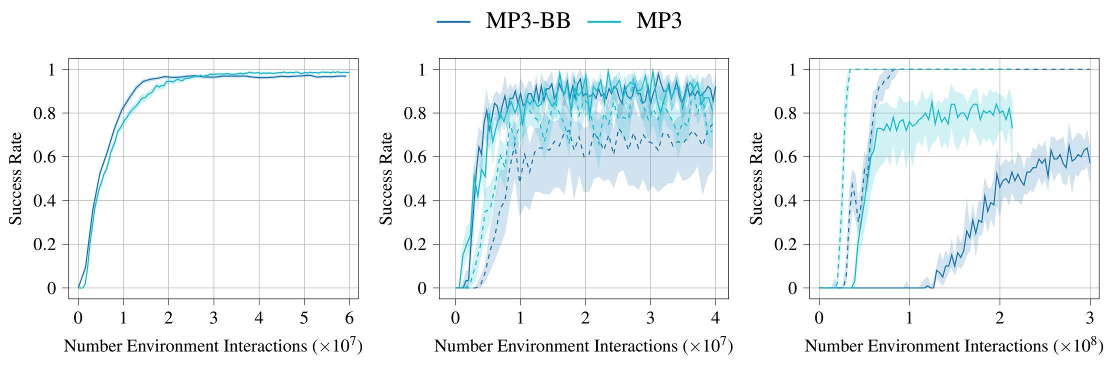
In all the box-pushing experiments (Figure 4.7b), we kept the planning horizon to 25% of the max episode length. MP3 achieved the same performance with MP3-BB regarding success rate and sample efficiency in dense reward setting. In sparse reward setting, MP3 exhibited better precision and sample efficiency compared to MP3-BB. This is primarily due to the use of different MPs representations (ProDMPs for MP3 and ProMPs for MP3-BB). ProDMPs-based policies tend to generate trajectories with lower episode energy, which helps the agent to focus on the main target (move the box to the target) instead of regularized by the control penalty. We conducted an ablation in Figure 4.10d to verify this assumption.

While non-Markovian rewards cannot be used as freely as in the MP3-BB case, it is still possible to leverage them as long as the non-Markovian behavior is limited to one trajectory segment. For table tennis, we choose planning horizon k such that we ensure the last trajectory segment starts before the racket hits the ball. In this setting, MP3 learns table tennis skills with a better success rate and much fewer samples than MP3-BB (Figure 4.7c).

In conclusion, we observed that the use of replanning yields better asymptotic performance in all cases while it can harm slightly the sample efficiency (observed in Meta-World experiments). We attribute this to the higher dimensional state space that must be considered in the replanning case compared to the black-box case.

Dealing with Uncertainties in the Environments To demonstrate the robustness of MP3 in handling unforeseen events in the environment, we modified the box pushing and the table tennis tasks to include uncertainties that require incorporating feedback throughout the execution of the episode.

In the box-pushing experiments, we randomly switch to a new target position and orientation during execution after 20% of the max episode length. We compared the performance of our method against step-based PPO and TRPL in the dense reward setting. The results in Figure 4.8a show that MP3 achieved the best performance under this setting. To investigate the reasons behind this performance gap, we conducted a qualitative analysis of the trained policies with all three methods. We observed that the policies generated by PPO and TRPL accelerate much faster and keep a high velocity from the beginning of

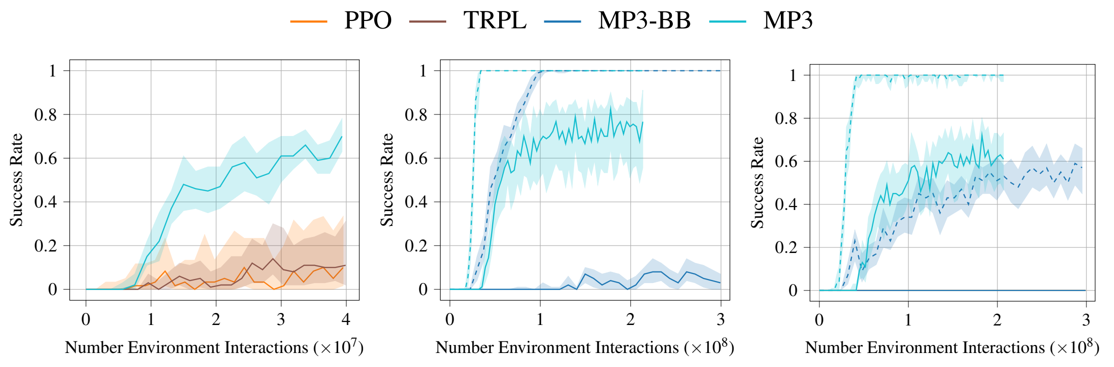


(a) MetaWorld - 50 Tasks (b) Box Pushing (c) Table Tennis

Figure 4.7: Figure 4.7a presents a comparison of the learning curves between MP3 and MP3-BB across Meta-World's 50 tasks. Due to its closed-loop nature, MP3 achieves higher asymptotic performance with a marginal compromise in sample efficiency. This is attributed to the advantage of MP3-BB in utilizing a more compact observation space. Figure 4.7b shows the performance of both methods in the box-pushing task, considering both dense (solid) and sparse (dashed) reward settings. While both methods achieve similar sample efficiency and asymptotic performance in dense reward settings, MP3 reaches a higher success rate in the sparse reward setting. Figure 4.7c demonstrates the hit rate (dashed) and success rate (solid) in the robot table tennis task. Both methods can consistently hit the ball, but MP3 outperforms MP3-BB by returning the ball with higher precision and requiring significantly fewer samples to converge.

the episode, which makes adapting to the new targets more difficult with the presence of control cost penalty and limited episode length. On the other hand, the acceleration and velocity of MP3 agents are regularized by the ProDMP's representation, leading to uniform motion, thus yielding an amenable success rate in this challenging goal-switching setting.

For the table tennis environment, we test two kinds of uncertainties. We compare MP3 only with the MP3-BB as the SRL algorithms have shown to be incapable of solving the table tennis task even in a static environment. Firstly, we modify the desired landing position of the ball, similar to the goal change for the box-pushing task. Specifically, we initialize the desired landing position at a random location on the left side of the table. After half of the maximum episode steps, there is a 50% chance that the target landing position will change to a new random position on the right side of the table. Our results in Figure 4.8b suggest that the MP3 agent is able to adapt its behavior and return the ball to the new target point with high precision. In contrast, the MP3-BB agent, which only receives the initial observation containing the initial target position, can only hit the ball but cannot solve this task. While we would expect the MP3-BB agent to solve at least those cases where the goal is not altered, we found that the conflicting reward feedback hinders the agent from learning high-quality policies. Secondly, we add wind to the



(a) Box Push - Switch (b) Table Tennis - Switch (c) Table Tennis - Wind

Figure 4.8: This figure displays the success rate of perturbed tasks with and without replanning. The success rate of box pushing with goal switching is shown at Figure 4.8a, as well as the success rate of table tennis with goal switching Figure 4.8b and with wind Figure 4.8c. In the box pushing tasks Figure 4.8a, the solid lines represent learning curves of dense reward, while the dashed lines are learning curves from temporal-sparse reward. While step-based algorithms with dense reward already struggle in the easier 20% setting, MP3 trained with sparse reward can solve both settings with a remarkable success rate. In table tennis tasks Figures 4.8b and 4.8c, the solid line represents the success rate, and the dashed line the hit rate. MP3 can effectively handle changes in tasks and environmental perturbations. In contrast, the MP3-BB fails in these cases, as it only relies on a single observation at the beginning of the episode, which lacks critical information about the environment and task dynamics.

environment by applying a random force to the ball, which is unknown to the agent and constant for an entire episode. However, the agent can still infer the underlying applied force according to the velocity of the ball, but only after observing the ball for a certain number of time steps. Due to the wind, the MP3-BB agent is not able to hit the ball consistently, while the MP3 agent slightly drops in performance but can still achieve reasonably good results (Figure 4.8c).

4.3.4 Ablation Studies

We conduct ablation studies to evaluate each component's influence on the proposed method that aim to answer the following questions:

Q1 What is the impact of varying the number of bases and the length of the replanning horizon on the performance of MP3?

Q2 How does the performance of the non dynamic-based ProMP with replanning compare to MP3?

Q3 Can the policy be effectively learned in the parameter space without incorporating proper trust regions?

Q4 How does the performance of dynamic-based ProDMPs compare to non dynamic-based ProMPs in MP3-BB setting?

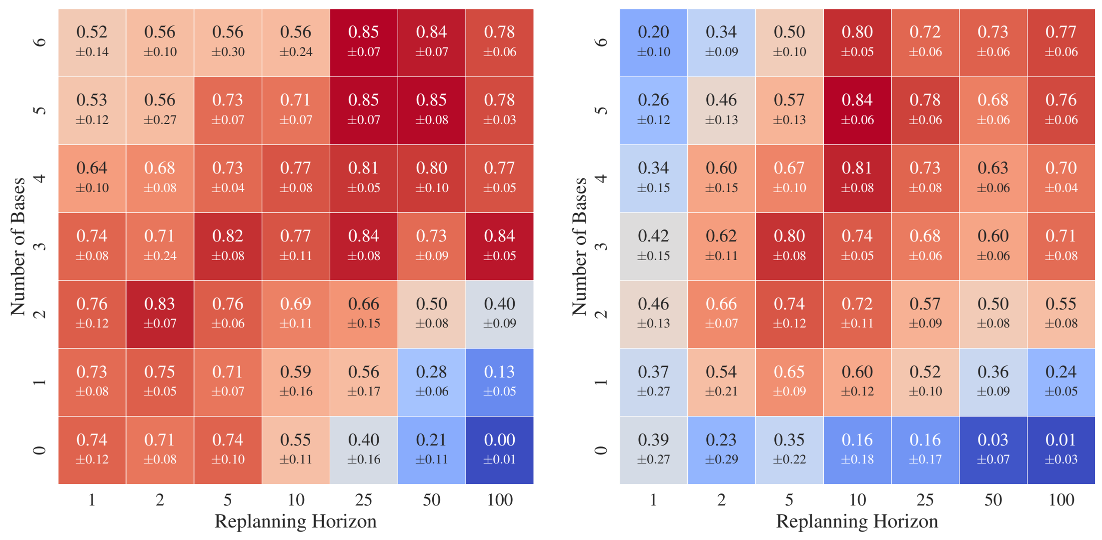
Firstly, we study the correlations between the number of bases of MPs and the length of the planning horizon (replanning steps) in Figure 4.9. We train agents in box-pushing environments with both dense and sparse rewards, using different combinations of planning horizons $k \in \{1,2,5,10,25,50,100\}$ and the number of bases $N \in \{0,1,2,3,4,5,6\}$. A value of 0 for the number of bases indicates that the agent only uses the goal basis of the ProDMPs, leading to the same action space dimension as SRL algorithms. When the planning horizon is equal to 1, the learning objective of replanning reduces to a SRL objective. In this case, the only difference between replanning and SRL is that the agent explores the parameters space of the MP, which usually has a higher dimensionality ((N + 1) $\times$ DoF) compared to the action space that a step-based agent explores. Another special case is when the replanning horizon equals the episode length ($k = 100 = T$), which corresponds to the MP3-BB setting.

Q1 can be answered according to results in Figure 4.9. First, planning with a longer horizon requires a greater number of bases to achieve optimal performance. A longer planning horizon means less chance for the agent to adapt trajectories by adjusting the weights, limiting the ability to generate complex trajectories. This limitation, in turn, reduces performance in tasks that require fine manipulation, such as box pushing. Second, longer planning horizons contribute to improved performance in the (temporal) sparse reward setting. This is attributed to the usage of high temporal abstracted samples in the policy updates. However, it does not necessarily mean MP3-BB will always perform better in the sparse reward setting, as the black-box setting lacks the ability to correct its behavior due to the absence of the feedback signals from inter-execution observations.

For **Q2**, we compare replanning with dynamic-based (ProDMPs) and non dynamic-based MPs (ProMPs) in Figures 4.10a and 4.10b. The results demonstrate that the policy with dynamic-based MPs yields a policy with a higher success rate and lower control cost. This is largely due to the fact that non-dynamic MPs can result in abrupt transitions between different planning segments, leading to discontinuities in the motion.

To address **Q3**, we evaluate policy search algorithms without trust regions in the replanning setting and present the results in Figure 4.10c. In both dense and sparse reward settings of box pushing, MP3 outperforms MP3-PPO in terms of sample efficiency and success rate. The need for a more stable optimization and the higher dimensional nature of learning in parameter spaces could account for this observed improvement.

Finally, to answer **Q4**, we compare the performance between the black-box agent with ProDMPs and with ProMPs. The results in Figure 4.10d show that in dense and sparse reward settings, both algorithms' sample efficiency and success rate are similar. In the sparse reward setting, MP3-BB with ProMPs shows slightly higher asymptotic performance. This difference is due to the different shapes of bases, and we believe the minor



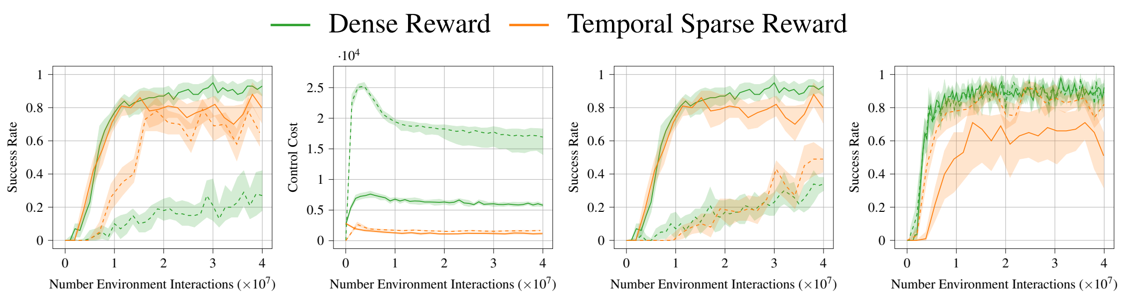
(a) Median Success Rate for Dense Reward (b) Median Success Rate for Sparse Reward

Figure 4.9: This figure shows the median success rate and standard deviation for different configurations of box-pushing environments using dense (Figure 4.9a) and sparse reward (Figure 4.9b), with varying numbers of bases N and replanning horizons k. When $N = 0$, the weight bases are disabled, and only the goal basis of the ProDMP is used, with the action space dimension equal to that of SRL. $k = 1$ and $k = 100 = T$ correspond to SRL with MPs and MP3-BB, respectively. We evaluate each combination using ten random seeds and 20 contexts per seed. In general, the results suggest that (i) longer planning horizons require more bases for optimal performance, (ii) long planning horizons can help improve performance in sparse reward settings.

performance gap can be mitigated by selecting MP's parameters that minimize the differences in bases. The overall results suggest that the types of MP make no significant difference in the black-box setting.

4.4 Conclusion and Limitations

Our work presents a new approach for combining SRL and ERL by integrating recent advancements in trust-region-based policy search (Otto et al., 2021) and MPs (Li et al., 2022). Unlike the commonly used step-based exploration in SRL, our method incorporates consistent and effective exploration at the trajectory level. This approach is a promising way to handle tasks with sparse and non-Markovian rewards, enabling a more intuitive reward design. Furthermore, our method showed competitive performance against state-of-the-art SRL algorithms in large-scale robot manipulation tasks, as confirmed by thorough empirical evaluations.



(a) MP3: Success (b) MP3: Control Cost (c) MP3: PPO vs TRPL (d) BB: ProMP vs ProDMP

Figure 4.10: This figure presents an ablation study for MP3 and MP3-BB with different MPs and learning algorithms. The figures Figures 4.10a and 4.10b show the success rate and episode control cost for the box-pushing task, respectively. Here, we compare MP3 with ProDMPs (solid) to MP3 with ProMPs (dashed) to show the need for ProDMPs when replanning. Figure 4.10c demonstrates the need for using TRPL in MP3 (solid). MP3-PPO with ProDMPs using replanning (dashed) on box pushing tasks is not able to achieve the same performance. Lastly, the figure Figure 4.10d shows that MP3-BB with ProDMPs (dashed) is performing similarly to the MP3-BB with ProMPs (solid) in dense reward setting, and slightly better in the sparse reward setting.

Although our proposed method shows promise, there remain some limitations that require addressing in future work. Firstly, our current approach only considers fixed-length planning horizons and relies solely on time-based replanning triggers. Yet, in real-world applications, it may be necessary to incorporate event-based replanning triggers, such as detecting unforeseen obstacles or changing targets. Therefore, we will investigate how to integrate event-based triggers into our method in the future. Secondly, our method, and ERL approaches in general, typically require more interaction time than SRL in dense reward settings. This is mainly due to the encapsulation of temporal-correlated information in highly abstracted samples. To address this issue, we will leverage the information within each planning segment to improve efficiency. Lastly, our evaluation of the Meta-World benchmark suite identified that most failure cases occur in tasks requiring sub-goal achievement and skill sequencing. Therefore, we will investigate how to achieve long-horizon planning by incorporating sub-goals into our framework in future work.

Chapter 5

Vlearn: Off-Policy Learning with Efficient State-Value Function Estimation

This chapter extends the applicability of the *trust region projection layer* (TRPL) beyond the on-policy framework to the off-policy setting, enhancing sample efficiency. Additionally, it introduces a novel approach to leverage purely V-function critics, particularly advantageous in higher-dimensional action spaces, as it circumvents the complexities associated with learning the Q-function. The key innovation of Vlearn lies in eliminating the necessity for a Q-function, streamlining the learning process, and facilitating more efficient exploration and exploitation in intricate environments. These findings were disseminated in a prior publication, as detailed in Otto et al. (2024).

5.1 Introduction

Reinforcement learning (RL) has emerged as a powerful paradigm for training intelligent agents through interaction with their environment (Mnih et al., 2013; Silver et al., 2016). Within RL, there exist two primary approaches for model-free learning: on-policy and off-policy methods. On-policy methods rely on newly generated (quasi-)online samples in each iteration (Schulman et al., 2017; Otto et al., 2021), whereas off-policy methods leverage a replay buffer populated by a behavior policy (Abdolmaleki et al., 2018b; Haarnoja et al., 2018; Fujimoto et al., 2018). Although on-policy methods can compensate for stale/off-policy data to some extent via importance sampling (Espeholt et al., 2018), they are still not able to fully exploit it.
To harness the full potential of off-policy data, off-policy methods traditionally focus on learning state-action-value functions (Q-functions) as critics (Degris et al., 2012). The Q-function's dependency on the state as well as the action enables it to update only those actions that have been observed in the transitions generated by the behavior policy. However, the complexity associated with learning Q-functions, especially in high-dimensional action spaces, is often undesirable, and alternatives similar to the on-policy setting based on state-value functions (V-functions), would be preferable.

In this work, we introduce Vlearn, a novel approach to off-policy *policy gradient* (PG) learning that exclusively leverages V-functions. While existing methods, such as V-trace (Espeholt et al., 2018), have tried to increase the amount of stale data on-policy methods can exploit, we found them to struggle in a full off-policy setting. In particular, they only aim to reweigh the Bellman targets. Yet, previous work (Mahmood et al., 2014) has already shown that importance sampling for the full Bellman error is preferable to such reweighing of the Bellman targets. Our method optimizes an upper bound of the original Bellman error, which can be derived using Jensen's inequality. This bound effectively shifts the importance weights from the Bellman targets to the optimization objective itself, which simplifies V-function updates and increases the stability of learning a V-function from off-policy data, a hallmark of previous approaches. In addition to this advancement, we further enhance the stability of policy learning by combining it with the trust region update introduced by Otto et al. (2021), forming an efficient off-policy trust region method. In our experiments, we demonstrate the benefits of this approach, especially in environments with high-dimensional action spaces, such as the notoriously difficult dog locomotion tasks of *DeepMind control* (DMC), which currently cannot be solved by most standard off-policy actor-critic methods.

5.2 Efficient State-Value Function Learning from Off-Policy Data

Most popular off-policy actor-critic methods (Haarnoja et al., 2018; Fujimoto et al., 2018; Abdolmaleki et al., 2018b) now aim to find a policy that maximizes the cumulative discounted reward by making use of a learnable state-action value estimate $Q^{\pi}_{\theta}(s,a) = \mathbb{E}_{\pi}[G_t|s_t = s, a_t = a]$. To train this estimator, they rely on a dataset $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})_{t=1\ldots N}\}$ and a behavioral policy $\pi_b(\cdot|s)$ responsible for generating this dataset. Typically, $\mathcal{D}$ takes the form of a replay buffer (Lin, 1992), with the corresponding behavior policy π_b being a mixture of the historical policies used to populate the buffer. Training the state-action value function then usually involves temporal difference learning (Sutton, 1988; Watkins and Dayan, 1992), with updates grounded in the Bellman equation (Bellman, 1957). The objective commonly optimized is

$$L_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\Big[\big(Q^{\pi}_{\theta}(s_t,a_t) - \big(r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim\mathcal{D},a_{t+1}\sim\pi(\cdot|s_{t+1})}Q_{\bar{\theta}}(s_{t+1},a_{t+1})\big)\big)^2\Big], \tag{5.1}$$

where $Q_{\bar{\theta}}(s,a)$ is a frozen target network. Furthermore, to mitigate overestimation bias, most methods employ two state-action value functions (Fujimoto et al., 2018). To maintain the stability of this approach, the target network's weights are updated at each time step as $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$.

Instead of using the state-action function $Q^{\pi}_{\theta}(s_t,a_t)$, an alternative approach is to work solely with the state-value function $V^{\pi}_{\theta}(s_t)$. This estimator can be trained by minimizing the following loss function using importance sampling

$$L_{\text{base}}(\theta) = \mathbb{E}_{s_t \sim \mathcal{D}}\left[\left(V_\theta^\pi(s_t) - \mathbb{E}_{(a_t, s_{t+1}) \sim \mathcal{D}}\left[\frac{\pi(a_t|s_t)}{\pi_b(a_t|s_t)}\big(r(s_t, a_t) + \gamma V_{\bar{\theta}}(s_{t+1})\big)\right]\right)^2\right]. \tag{5.2}$$

Directly optimizing this can be difficult as approximating the inner expectation with just one Monte Carlo sample yields a high variance. Achieving a more reliable estimate necessitates multiple samples, implying either multiple action executions per state or occurrences of the same state in the replay buffer, an unrealistic assumption for most RL problems. An essential distinction from the Q-function based objective in Equation (5.1) lies in the introduction of the importance weight $\pi(a_{t,j}|s_t)/\pi_b(a_{t,j}|s_t)$, which accounts for the difference between the behavior distribution $\pi_b(\cdot|s)$ and the current policy $\pi(\cdot|s)$. Unlike the Q-function, which updates its estimate solely for the chosen action, the V-function lacks a dependency on the action and implicitly updates its estimate for all actions. As a result, we need to consider the difference between the current policy and the policy that collected the data.

The objective in Equation (5.2) bears similarity to the 1-step V-trace estimate (Espeholt et al., 2018). It can be seen as a naive version of V-trace because a large difference between the target and behavior policies may result in importance weights approaching either zero or infinity, consequently impacting the Bellman optimization target for the state-value function $V_\theta^\pi(s_t)$. While truncated importance weights can prevent excessively large values for the Bellman target, the importance ratios and target may still approach values close to zero. V-trace reformulates this objective and effectively interpolates between the Bellman target and the current (target) value function in the one-step case $n = 1$

$$L_{\text{V-trace}}(\theta) = \mathbb{E}_{s_t \sim \mathcal{D}}\left[\Big(V_\theta^\pi(s_t) - \Big((1-\rho_t)V_{\bar{\theta}}(s_t) + \rho_t\big(r_t + \gamma V_{\bar{\theta}}(s_{t+1})\big)\Big)\Big)^2\right], \tag{5.3}$$

where $\rho_t = \min(\pi(a_t|s_t)/\pi_b(a_t|s_t), \varepsilon_\rho)$ are the truncated importance weights with a user-specified upper bound (typically $\varepsilon_\rho = 1$). This objective can then also be extended for off-policy corrections of n-step returns (Espeholt et al., 2018). Yet, this formulation has two main drawbacks. Since it is impractical or even impossible to implement multiple actions for the same state, V-trace approximates the inner expectation with only one action sample, resulting in potentially undesirable high variance estimates. Moreover, while the V-trace reformulation avoids optimizing the value estimate toward zero for small importance ratios, the interpolation now optimizes it increasingly toward the current (target) value function. This interpolation also leads to a shift of the optimum, which means for samples with small importance ratios the value function is barely making any learning progress and maintains its current estimate. It is important to note that only the optimum changes; the scale of the loss function, and thus the scale of the gradient, remains the same for all importance ratios. In the original work, this issue is less pronounced, as V-trace provides off-policy corrections for asynchronous distributed workers. In this setup, the policy distributions are expected to stay relatively close, mitigating the impact of any potential divergence. However, in a full off-policy setting, the samples in the replay buffer and the current policy diverge significantly faster and more widely. The size of the replay buffer, learning speed, or entropy of different policies can all affect the importance ratio. This behavior for various importance ratios is illustrated in Figure 5.1.

5.2.1 VLearn: Off-Policy State-Value Function Learning

A significant challenge with V-trace lies in its use of importance sampling exclusively for the target computations, which can additionally become increasingly costly to compute with larger n-step returns. Moreover, it has already been shown for the linear case that applying the importance weights to just the Bellman targets is inferior to applying them to the full loss (Mahmood et al., 2014; Dann et al., 2014). However, contradicting these previous findings, the former approach remains the main method in the deep learning setting (Munos et al., 2016; Espeholt et al., 2018). To address the above-mentioned issues, we propose a more effective approach to training the value function by following Mahmood et al. (2014); Dann et al. (2014) and shifting the importance ratio to the full loss function by extending their results to the general function approximation case.

Theorem 2 *Consider the following loss, minimized with respect to the V-function's parameters θ*

$$L_{Vlearn}(\theta) = \mathbb{E}_{(s_t,a_t,s_{t+1})\sim\mathcal{D}}\left[\frac{\pi(a_t|s_t)}{\pi_b(a_t|s_t)}\Big(V_\theta^\pi(s_t) - (r_t + \gamma V_{\bar{\theta}}(s_{t+1}))\Big)^2\right]. \tag{5.4}$$

This objective serves as an upper bound to the importance-weighted Bellman loss (Equation (5.2)*). Furthermore, this upper bound is consistent; that is, a value function minimizing Equation* (5.4) *also minimizes Equation* (5.2)*.*

The first statement's proof relies on Jensen's Inequality, while the proof of the second statement is an extension of the work from Neumann and Peters (2008). The complete proofs are provided in Appendices C.1.1 and C.1.2, respectively.
This upper bound closely resembles the standard state-action-value loss functions used in other off-policy methods but is based on the action-independent state-value function. It introduces importance weights to account for the mismatch between behavior and policy distribution. Importantly, evaluating Equation (5.4) becomes straightforward using the provided replay buffer $\mathcal{D}$. When we use samples from the joint state-action distribution, dealing with only one action per state becomes more manageable. This helps to reduce variance compared to the original and V-trace objectives. Additionally, another issue with using importance sampling is that its estimator can have a large variance when the target and behavior policies are very different from each other. Evaluating the variance

of the importance sampling estimator is generally intractable (Munos et al., 2016; Koller and Friedman, 2009). Therefore, we consider the stateless *Markov Decision Process* (MDP), the multi-armed bandit problem, as a simplified scenario. The critic's empirical losses for the base, Vlearn, and V-trace objective (Equations (5.2) to (5.4)) can then be simplified as

$$
\begin{aligned}
L_{\text{base}} &= \left(V - \frac{1}{N} \sum_{a \sim \pi_b} \rho_a r(a) \right)^2 \\
L_{\text{V-trace}} &= \frac{1}{N} \sum_{a \sim \pi_b} \Big(V - \big((1 - \rho_a) V + \rho_a r(a) \big) \Big)^2 \\
L_{\text{Vlearn}} &= \frac{1}{N} \sum_{a \sim \pi_b} \rho_a \Big(V - r(a) \Big)^2,
\end{aligned}
$$

where we denote weights $\rho_a = \pi(a)/\pi_b(a)$. Taking the derivatives of the above losses with respect to the estimator V and solving the derivative of zero for V yields the following three estimators

$$
\hat{V}_{\text{base}} = \frac{\sum_{a \sim \pi_b} \rho_a r(a)}{N} \qquad \hat{V}_{\text{V-trace}} = \frac{\sum_{a \sim \pi_b} \rho_a^2 r(a)}{\sum_{a \sim \pi_b} \rho_a^2} \qquad \hat{V}_{\text{Vlearn}} = \frac{\sum_{a \sim \pi_b} \rho_a r(a)}{\sum_{a \sim \pi_b} \rho_a}.
$$

Through this simplified scenario, we show an intuition that the proposed weighted loss of Vlearn yields the self-normalized importance weighting estimator, while V-trace results in a squared self-normalized estimator and the base objective yields a standard importance weighting estimator. For the self-normalized estimator, it is well known that it can be more robust in case of extreme importance weights, and often lower than the standard importance weighting estimator in general machine learning (Koller and Friedman, 2009) and in RL settings (Swaminathan and Joachims, 2015; Futoma et al., 2020). However, the squared estimator from V-trace is expected to result in larger variances.

Furthermore, unlike V-trace, each sample optimizes towards the Bellman target but has an impact on the total loss per step depending on its importance weight, as shown in Figure 5.1. The smaller importance weights mainly scale the gradient without causing a shift in optimizing to a different optimum, such as the current (target) value function. This approach makes learning the state-value function in an off-policy setting more stable and efficient.

An essential consideration is defining the behavior policy π_b in an off-policy setting, where π_b is a mixture of all past policies that contributed to the replay buffer. Computationally, storing and evaluating this mixture policy would be expensive. However, since we rely on importance sampling, which only requires access to the (log-)probabilities of each action, we can easily extend the replay buffer with this single entry at minimal additional cost. In addition, this (log-)probability can then be used directly for importance sampling in the policy update step.

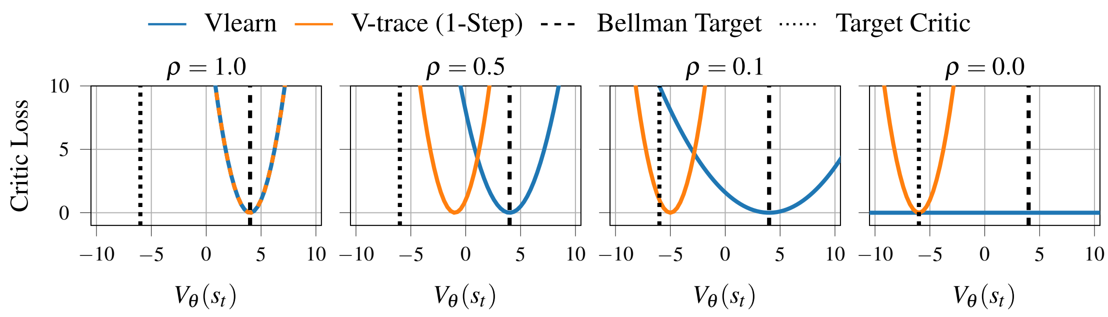


Figure 5.1: To provide intuition on the differences between Vlearn and V-trace we consider the following example for different importance ratios ρ. Assume that for a state s_t, the Bellman target is $r(s,a) + \gamma V_{\bar{\theta}}(s_{t+1}) = 4$, the target critic predicts $V_{\bar{\theta}}(s_t) = -6$ and we plot the loss for potential values of $V_\theta(s)$ for Vlearn and V-trace. For on-policy samples ($\rho = 1.0$) both losses are the same. However, for samples that are more and more off-policy ($\rho \to 0$) we see how V-trace increasingly relies on the target critic, shifting the optimal value towards it. Vlearn on the other hand simply reduces the scale of the loss and thus the importance of the sample. This makes Vlearn more robust to errors in the target critic.

5.2.2 Off-Policy Policy Learning with VLearn

To find the optimal policy, conventional PG techniques frequently use the gradient of the likelihood ratio and importance sampling estimator to optimize an estimate of the Q-function. In particular, a more effective approach then involves optimizing the advantage function, denoted as $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$. Adding the V-function as a baseline yields an unbiased gradient estimator with reduced variance. The optimization can then be formulated as follows

$$\max_\phi \hat{J}(\pi_\phi, \pi_b) = \max_\phi \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[\frac{\pi_\phi(a|s)}{\pi_b(a|s)} A^\pi(s,a) \right]. \tag{5.5}$$

In the on-policy case, the advantage values are typically estimated via Monte Carlo approaches, such as general advantage estimation (Schulman et al., 2016). Yet, in our setting, we cannot compute a good Monte Carlo estimate and cannot rely on the Q-function estimate (Degris et al., 2012). Further, the direct use of the V-function to improve the policy is not feasible. However, in conjunction with the replay buffer, a strategy akin to the standard PG becomes viable. This approach allows us to employ an off-policy estimate of the V-function, offering a substantial boost in sample efficiency. The advantage estimate is computed using the one-step return of our off-policy evaluated value function $A^\pi(s_t,a_t) = r_t + \gamma V^\pi_\theta(s_{t+1}) - V^\pi_\theta(s_t)$. The above objective can then be optimized by any PG algorithm, *e.g.*, by *proximal policy optimization* (PPO) (Schulman et al., 2017) or via the TRPL (Otto et al., 2021).
In this work, we use TRPL (Otto et al., 2021) as it has been shown to stabilize learning even for complex and high-dimensional action spaces (Otto et al., 2022, 2023). Unlike

Algorithm 1

```
1:  Initialize policy $\phi$, critics $\theta_1, \theta_2$, target critics $\bar{\theta}_1, \bar{\theta}_2$
2:  Initialize replay buffer $\mathcal{D}$, truncation level $\varepsilon_\rho$, trust region bounds $\varepsilon_\mu$, $\varepsilon_\Sigma$
3:  Initialize $\pi_{\text{old}} \leftarrow \pi_\phi$
4:  for $i = 0, 1, \dots, N$ do                                                      ▷ epoch
5:      for $j = 0, 1, \dots, M$ do
6:          Collect sample $(s, a, r, s', \pi_\phi(a|s))$ and add to $\mathcal{D}$
7:          Sample batch $\mathcal{B} = \{(s, a, r, s', \pi_b(a|s))\}_{t=1 \dots K}$ from $\mathcal{D}$
8:          Get current policy $\pi_\phi(a|s)$ for all $s$ in $\mathcal{B}$
9:          Compute projected policy $\tilde{\pi}_\phi = \text{TRPL}(\pi_\phi, \pi_{\text{old}}, \varepsilon_\mu, \varepsilon_\Sigma)$ for all $s$ in $\mathcal{B}$
10:         Update all critic networks with gradient
                $\nabla_{\theta_i} \frac{1}{K} \sum_{\mathcal{B}} \min\left(\frac{\tilde{\pi}_\phi(a|s)}{\pi_b(a|s)}, \varepsilon_\rho\right) \left(V^\pi_{\theta_i}(s) - \left(r + \gamma V_{\bar{\theta}_i}(s')\right)\right)^2$ for $i \in \{1, 2\}$
11:         if update policy then
12:             Compute advantage estimates $\hat{A} = r + \gamma \min_{i=1,2} V^\pi_{\theta_i}(s') - \min_{i=1,2} V^\pi_{\theta_i}(s)$
13:             Update policy with trust region loss
                    $\nabla_\phi \left[\frac{1}{K} \sum_{\mathcal{B}} \left[\min\left(\frac{\tilde{\pi}_\phi(a|s)}{\pi_b(a|s)}, \varepsilon_\rho\right) \hat{A}\right] - \alpha \mathrm{d}(\pi_\phi, \tilde{\pi})\right]$
14:         $\bar{\theta}_i \leftarrow \tau \theta_i + (1 - \tau) \bar{\theta}_i$ for $i \in \{1, 2\}$
15:     $\pi_{\text{old}} \leftarrow \pi_\phi$
```

PPO, it provides a mathematically sound and scalable approach to enforce trust regions exactly per state. Moreover, TRPL allows us to use the constraint policy also during the value function update, which now requires importance sampling. PPO has only the clipped objective for the policy update, and using a clipped value function has been shown to potentially degrade performance (Engstrom et al., 2020).

TRPL efficiently enforces a trust region for each input state of the policy using differentiable convex optimization layers (Agrawal et al., 2019), providing more stability and control during training and at the same time reducing the dependency on implementation choices (Engstrom et al., 2020). Intuitively, the layer ensures the predicted Gaussian distribution from the policy network always satisfies the trust region constraint. This way, the objective from Equation (5.5) can directly be optimized as the trust region always holds. The layer receives the network's initial prediction for the mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ of a Gaussian distribution, and projects them into the trust region when either exceeds their respective bounds. This is done individually for each state provided to the network. The resulting Gaussian policy distribution, characterized by the projected parameters $\tilde{\boldsymbol{\mu}}$

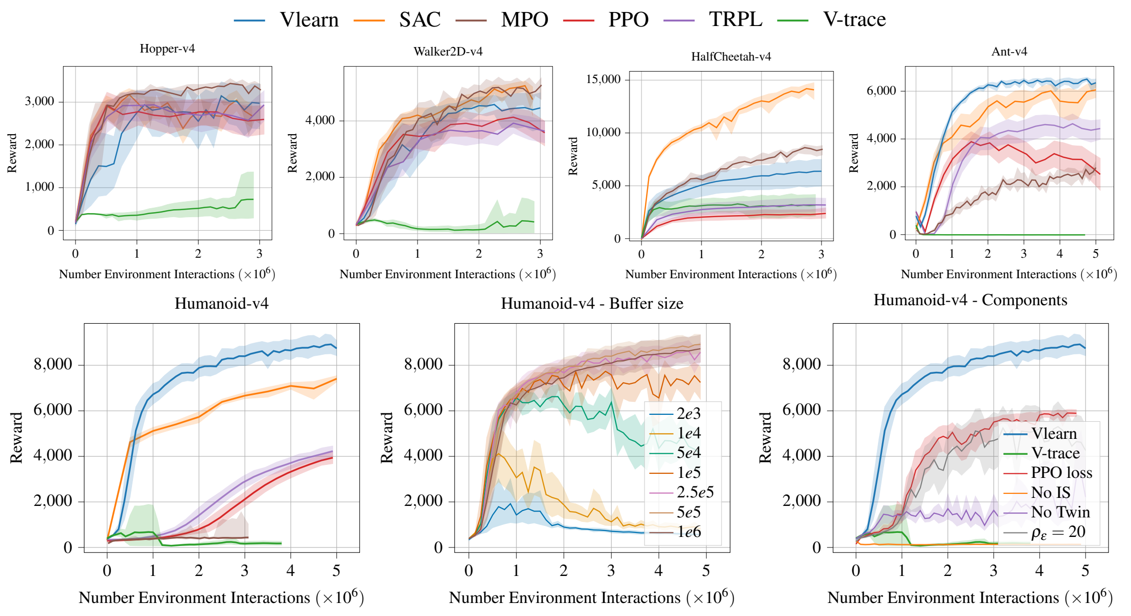


Figure 5.2: Shown are the mean over 10 seeds and 95% bootstrapped confidence intervals for the Gymnasium tasks. While *soft actor critic* (SAC) performs well on the lower dimensional tasks, Vlearn achieves better asymptotic performance for the higher dimensional problems - Ant-v4 and Humanoid-v4. In comparison to V-trace, our method learns significantly more stable while also achieving an overall better performance. The figure in the bottom center shows an ablation study that compares the effect the replay buffer size has on the policy's performance. For smaller replay buffers, learning becomes unstable or does not converge, while larger sizes tend to lead to similar final performances. The figure in the bottom right shows an ablation study for variations of Vlearn.

and $\tilde{\mathbf{\Sigma}}$, is then used for subsequent computations, *e.g.* sampling and loss computation. Formally, as part of the forward pass, the layer solves the following two optimization problems for each state $\boldsymbol{s}$

$$\begin{aligned}\arg\min_{\tilde{\boldsymbol{\mu}}_s} d_{\text{mean}}\left(\tilde{\boldsymbol{\mu}}_s, \boldsymbol{\mu}(s)\right), \text{s.t.}\, d_{\text{mean}}\left(\tilde{\boldsymbol{\mu}}_s, \boldsymbol{\mu}_{\text{old}}(s)\right) \le \varepsilon_{\boldsymbol{\mu}} \\ \arg\min_{\tilde{\mathbf{\Sigma}}_s} d_{\text{cov}}\left(\tilde{\mathbf{\Sigma}}_s, \mathbf{\Sigma}(\boldsymbol{s})\right), \text{s.t.}\, d_{\text{cov}}\left(\tilde{\mathbf{\Sigma}}_s, \mathbf{\Sigma}_{\text{old}}(\boldsymbol{s})\right) \le \varepsilon_{\Sigma},\end{aligned}$$

where $\tilde{\boldsymbol{\mu}}_s$ and $\tilde{\mathbf{\Sigma}}_s$ are the optimization variables for input state $\boldsymbol{s}$. The trust region bounds ε_μ, ε_Σ are for the mean and covariance of the Gaussian distribution, respectively. For the dissimilarities between means d_{mean} and covariances d_{cov}, we use the decomposed KL-divergence. How to receive the gradients for the backward pass via implicit differentiation is described in Appendix A.2.5.

As the original V-trace algorithm is primarily used for off-policy corrections in distributed on-policy learning methods, it typically does not incorporate a target network. In contrast, it is common practice for off-policy methods (Haarnoja et al., 2018; Fujimoto et al., 2018; Abdolmaleki et al., 2018b) to incorporate a target network. This practice is also followed by V-trace's predecessor, Retrace (Munos et al., 2016), and we include it for VLearn as well.

5.2.3 Behavior Policy vs Old Policy

For Vlearn, we keep track of three different policies: The current policy π_ϕ, which is optimized, the old policy π_{old}, which is used as a reference for the trust region, and the behavioral policy π_b, which is needed for off-policy correction using importance sampling. Similar to most on-policy trust region methods, we maintain π_{old} as a copy of the main policy network from the previous iteration. Choosing the behavioral policy for the trust region would be detrimental to performance because it would slow down the policy network update or, in the worst case, force the policy back to a much older and worse policy distribution. Conversely, we cannot use the copied old policy as a behavior policy because the actual behavior policy π_b can be arbitrarily far away, especially for older samples. Generally, the behavior policy is a mixture of all past policies that have been used to populate the replay buffer. However, storing and/or evaluating this would be expensive, so we assume that each sample belongs to only one mixture component, specifically the policy that originally created the action. Since we rely on importance sampling for the off-policy correction, we can simply store the (log-)probability for each action as part of the replay buffer to represent π_b during training.
To stabilize our method further, we make use of improvements of other actor-critic and PG methods, including twin critics and delayed policy updates (Fujimoto et al., 2018), tanh squashing (Haarnoja et al., 2018), as well as advantage normalization (Schulman et al., 2017; Otto et al., 2021). While the overestimation bias is not directly a problem when using just state-value functions, we found them to be beneficial in practice. We assume the twin network can be seen as a small ensemble that provides a form of regularization. Given there is no significant drawback and the widespread adoption of the twin network approach in other baseline methods, we have chosen to maintain its use in our case as well. Additionally, similar to prior works (Munos et al., 2016; Espeholt et al., 2018) we aim to reduce the variance by replacing the standard importance sampling ratio with truncated importance sampling $\min(\pi(a_t|s_t)/\pi_b(a_t|s_t), \varepsilon_\rho)$. An overview of our full approach is shown in Algorithm 1.

5.3 Experiments

For our experiments, we evaluate Vlearn on a variety of different continuous control tasks from Gymnasium (Towers et al., 2023) and DMC (Tunyasuvunakool et al., 2020). As baselines, we trained the standard off-policy methods SAC (Haarnoja et al., 2018) and

maximum aposteriori policy optimization (MPO) (Abdolmaleki et al., 2018b) as well as PPO (Schulman et al., 2017) and the on-policy version of TRPL (Otto et al., 2021). In addition, we compare our method to 1-step V-trace (Espeholt et al., 2018) by replacing our lower bound objective from Equation (5.4) with the objective in Equation (5.3). This comparison aims to highlight the difference in the placement of the importance ratios. To ensure a fair assessment, we want to eliminate any external factors that could influence the results and thus do not use n-step returns. While the n-step V-trace can be computed recursively, our approach could leverage the product of the n-step importance weights, preserving consistency with V-trace. All other components of the method remain the same as in Vlearn.
The evaluation protocol follows the approach described in Agarwal et al. (2021). All methods are evaluated for 10 different seeds each, and their performance is aggregated using mean values and 95% bootstrapped confidence intervals for individual tasks. We maintain uniformity in the architecture of the policy and critic networks for all methods, incorporating layer normalization (Ba et al., 2016) before the first hidden layer. Hyperparameters are kept constant and only adjusted as appropriate for the higher dimensional dog tasks. Detailed hyperparameter information for all methods can be found in Appendix C.2.

5.3.1 Gymnasium

The results for the Gymnasium tasks are shown in Figure 5.2. Vlearn shows good performance on lower dimensional tasks, achieving asymptotic performance comparable to SAC, except for HalfCheetah-v4. While Vlearn generally outperforms on-policy methods on the HalfCheetah-v4, it exhibits a slower convergence rate compared to SAC and seems to reach a lower local optimum, similar to the on-policy methods. In general, HalfCheetah-v4 appears to be an outlier, challenging the learning capabilities of all trust-region methods. Even MPO, employing a related KL regularization concept, does not match the performance of SAC. Moreover, the environment itself has shown extreme behavior in the past (Zhang et al., 2021). However, for the other lower dimensional tasks Vlearn and MPO perform comparably to the other baseline methods.
For the two higher dimensional tasks, Ant-v4 and Humanoid-v4, Vlearn outperforms all other baselines. Across both environments, Vlearn demonstrates better convergence speed and superior asymptotic performance. Of particular note is the remarkable performance increase for Humanoid-v4, where Vlearn achieves a 25% increase over SAC. While the improvement is more subtle in the case of Ant-v4 with its 8-dimensional action space, the advantage of Vlearn becomes apparent in the much larger 17-dimensional action space of Humanoid-v4. In this scenario, focusing solely on learning a state-value function proves to be a less complex and more effective approach than attempting to learn the full state-action value function. Neither for the Ant-v4 nor for the Humanoid-v4 can MPO achieve competitive performance.
In direct comparisons between Vlearn and its V-trace counterpart, Vlearn consistently

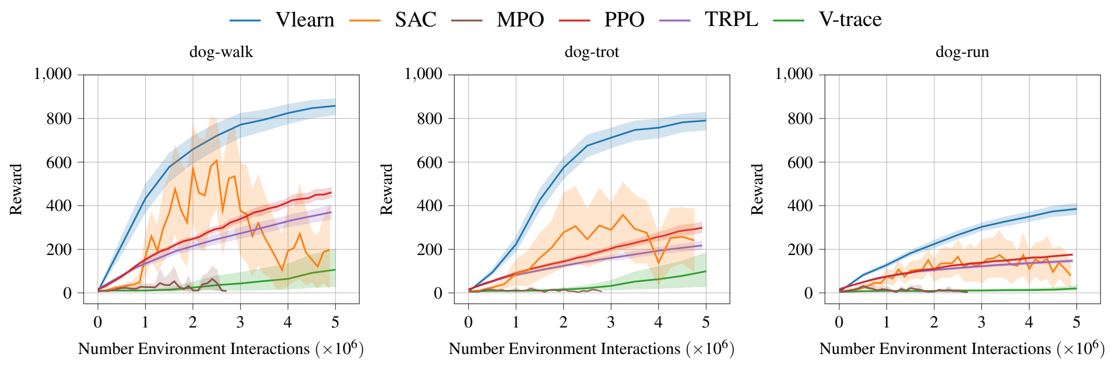


Figure 5.3: Performance on the 38-dimensional DMC dog tasks. Shown are the mean over 10 seeds and 95% bootstrapped confidence intervals. While SAC and MPO struggle to learn a consistent policy, Vlearn excels across all three tasks. On-policy methods show modest improvements and V-trace even struggles to make any meaningful progress.

outperforms V-trace across all tasks. In particular, the V-trace version hardly learns anything within a fixed number of environment interactions. Our investigations revealed that while V-trace is eventually able to learn, it experiences a significant drop in performance after an extended training period. Since the only difference between the two experiments lies in the V-trace objective, this is most likely the source of this problem. The key distinction between our objective and V-trace lies in how they handle situations where the importance ratio approaches zero and the position of the expectations. In our objective, we assign these samples a weight close to zero, minimizing their influence on the gradient. Conversely, V-trace attempts to bring these samples closer to the target network, potentially leading to performance degradation. Additionally, estimating the joint expectation over states and actions is typically more stable. This observation aligns with findings in Mahmood et al. (2014); Dann et al. (2014), where it was demonstrated that importance sampling exclusively for the Bellman targets can lead to inferior performance. While they evaluated this only for the linear case, we found similar results for general nonlinear function approximation.

5.3.2 DeepMind Control Suite

In addition to the Gymnasium tasks, we conducted experiments using the DMC dog environments. The dog tasks pose the most challenging problems in our evaluation, with a 38-dimensional action space modeling a realistic pharaoh dog. Consistent with the Gymnasium setting, Vlearn exhibits superior performance on these high-dimensional dog locomotion tasks. Although we found that SAC benefits from layer normalization for these tasks, it struggles to learn well-performing policies for all three movement types. While SAC improves for “easier” motions, its convergence remains highly unstable, and its final performance often falls below that of on-policy methods. MPO has similar problems as with the higher dimensional Gymnasium tasks and is unable to solve the tasks. In contrast, Vlearn learns reliably for all three distinct dog movement types.

Comparing our V-function learning approach to the V-trace estimator, the disparities are even more pronounced for the DMC dog tasks. For these high-dimensional problems, the V-trace estimator fails to learn and is among the worst baselines, even falling behind on-policy methods. Generally, we observe a performance decline across the board, even in the case of lower-dimensional problems.

5.3.3 Ablation Studies

In our ablation study, we investigate the effect of replay buffer size as well as the individual components of our method on the learning process. With varying replay buffer sizes of $\{2e3, 1e4, 5e4, 1e5, 2.5e5, 5e5, 1e6\}$, for 10 seeds each, we trained multiple agents on the Gymnasium Humanoid-v4. Note that while the original V-trace (Espeholt et al., 2018) relies on a relatively large replay buffer, the massively parallel computation used there implies that policy distribution differences are not as pronounced as in standard off-policy methods like SAC. As shown in Figure 5.2 (bottom right), significant improvement occurs when transitioning from updating the policy and critical setting near an on-policy setting, using the smallest replay buffer size, to using a medium buffer. In particular, moving from a small buffer to a medium buffer yields significant benefits, while there is little difference between buffer sizes at the upper end of the range. In our experiments, we found that a replay buffer size of $5e5$ consistently produced optimal performance across all tasks (see Figures 5.2 and 5.3).

For the second ablation study, we trained multiple variations of Vlearn for Humanoid-v4 to investigate the effects of the individual components (Figure 5.2). First, as a naive baseline, we removed the importance sampling (No IS) and assumed all samples in the replay buffer were from the current policy. As expected, not having the correction from importance sampling does not allow the agent to learn at all. When replacing the TRPL policy loss with the clipped PPO loss (PPO loss), our results suggest that the heuristic trust region provided by PPO is insufficient for the off-policy case where stabilizing learning is even more important. While PPO can achieve a decent performance on the task, its asymptotic performance lags behind the agent using TRPL. For the importance weight truncation, we followed previous works (Espeholt et al., 2018; Munos et al., 2016) by selecting $\varepsilon_\rho = 1$ to reduce variance and potentially avoid exploding gradients. Similar to the PPO loss, we see that for a larger truncation level ($\varepsilon_\rho = 20$) learning becomes unstable. Lastly, we trained an agent without the twin critic networks (No Twin), which performs significantly worse. We assume the twin network can be seen as a small ensemble that provides a form of regularization.

5.4 Conclusion and Limitations

In this work, we have demonstrated an efficient methodology for learning a V-function from off-policy data, leveraging an upper bound objective to subsequently update the policy network. Our approach not only ensures computational efficiency but also showcases enhanced stability and performance compared to existing methods. Integrating this idea with the trust regions from TRPL yields an effective off-policy method, especially well-suited for scenarios involving high-dimensional action spaces.
Although our method excels in handling high-dimensional action spaces, it may still require a substantial amount of data to achieve optimal performance. Hence, sample efficiency still remains a challenge that needs to be addressed further. Additionally, in specific environments, such as HalfCheetah-v4, Vlearn did not achieve a competitive performance compared to existing approaches. Improving the method's reliability, particularly in lower-dimensional scenarios, represents an ongoing challenge. For future work, we are looking to combine this work with other advances in RL, such as distributional critics (Bellemare et al., 2017) or ensembles (Chen et al., 2021). Furthermore, we are looking to extend our approach to the realm of offline RL, which offers the opportunity to leverage pre-collected data efficiently, opening doors to real-world applications and minimizing the need for extensive data collection.

Chapter 6

Discussion and Future Work

In this work, we have presented improvements to trust region methods and incorporated insights from classical robotics, resulting in the development of more adept agents. Our work addresses critical challenges in deep RL through three primary contributions. First, the introduced TRPL algorithm introduces differentiable neural network layers to enforce trust regions for deep Gaussian policies, presenting a robust alternative to established methods like TRPO and PPO. These layers, designed to formalize trust regions for individual states, exhibit comparable or superior results while being implementation-agnostic. Second, MP3 incorporates MPs into deep RL, thereby enabling trajectory-based exploration and optimization. This integration not only fosters the generation of smooth and energy efficient behavior but also facilitates adaptation to task variations during execution. Leveraging TRPL, MP3 surpasses existing methods in both deep RL and RL with MPs across diverse reward formulations, addressing challenges in various domains. Lastly, Vlearn eliminates the explicit need for a state-action-value function in off-policy RL, overcoming data inefficiency in high-dimensional action spaces. This approach streamlines the learning process, enhances exploration and exploitation efficiency, and consistently performs well in diverse benchmark tasks. Collectively, these contributions advance trust region methods by offering robust and effective solutions to critical problems and showcasing their applicability across domains.

Nevertheless, despite the precision of methods like TRPL and TRPO in providing exact trust regions, and the natural scalability of KL-based trust regions with the policy entropy, all methods struggle with the adaptability of the trust region itself. Notably, in simpler or flat regions of the problem space, the trust region may inadvertently hinder performance, as larger steps beyond the predefined trust region could be safely taken in these areas. This prompts the question of whether it is feasible to leverage or learn a schedule or model that can dynamically adjust the trust region boundary based on the context or state of the problem at hand. Enabling the policy to make more substantial progress in specific areas could significantly enhance sample complexity, a concept already explored in trust region methods within standard policy search (Arenz et al., 2020; Hüttenrauch and Neumann, 2022), but still relatively unexplored for deep RL.

Furthermore, the application of RL to physical systems remains challenging (Dulac-Arnold et al., 2020). While direct training on robotic hardware is feasible (Haarnoja

et al., 2018), obtaining a sufficient number of samples at scale for more intricate problems remains difficult. Simulation provides an alternative, where sample complexity is less relevant, and agents trained in simulation can be transferred to the real world (Zhao et al., 2020). However, this approach introduces new challenges due to the disparity between simulation and real-world hardware. In addition to sample complexity, safety is another critical consideration when learning for physical systems. Extending trust regions to incorporate safety considerations, coupled with the idea of adaptive bounds, could be beneficial. Decreasing bounds in areas where safety violations are unlikely or impossible and increasing them in high-risk states of the system could enhance overall safety. Notably, MP3 already offers a mechanism for safety assessment, given that the (partial) trajectory is known before execution begins; extending this to generate only safe trajectories is a possible future enhancement. Furthermore, trust region layers could be directly employed in conjunction with guidance policies that encode safety, prior knowledge, or (human) preferences to constrain the search space.
Lastly, in their current form, all the proposed methods in this work are tailored for continuous control, a highly relevant area in RL research with numerous applications in real-world tasks, particularly in robotics. In the past, discrete action spaces were more often seen for video games (Bellemare et al., 2013) or board games (Silver et al., 2018). However, discrete action spaces have gained increased relevance, especially with the recent prominence of large language models and human preference alignment using RL from human feedback (Christiano et al., 2017). While PPO is currently widely used (Achiam et al., 2023; Touvron et al., 2023), it has several issues as highlighted earlier. Transferring more principled approaches like TRPL or Vlearn to RL from human feedback could offer improvements in robustness and control over the alignment process. Despite the possibility of also achieving alignment through direct preference optimization (Rafailov et al., 2023), there is currently no clear consensus on the preferred method. Consequently, there is a clear need for more specifically tailored and principled approaches in the area of RL from human feedback for (multimodal) large language models.

Bibliography

Abdolmaleki, A., Lioutikov, R., Peters, J. R., Lau, N., Pualo Reis, L., and Neumann, G. (2015). Model-based relative entropy stochastic search. *Advances in Neural Information Processing Systems*, 28.

Abdolmaleki, A., Simões, D., Lau, N., Reis, L. P., and Neumann, G. (2019). Contextual direct policy search. *Journal of Intelligent & Robotic Systems*, 96(2):141–157.

Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., and Riedmiller, M. (2018a). Maximum a posteriori policy optimisation. In *International Conference on Learning Representations*.

Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., and Riedmiller, M. (2018b). Maximum a posteriori policy optimisation. In *International conference on Learning Representations*.

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

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

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. (2019). Differentiable convex optimization layers. *Advances in neural information processing systems*, 32.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2623–2631.

Akrour, R., Pajarinen, J., Peters, J., and Neumann, G. (2019). Projections for approximate policy iteration algorithms. In *Proceedings of Machine Learning Research*, pages 181–190.

Amos, B. and Kolter, J. Z. (2017). OptNet: Differentiable Optimization as a Layer in Neural Networks. In *34th International Conference on Machine Learning*, pages 179–191.

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. In *arXiv preprint*.

Arenz, O., Zhong, M., and Neumann, G. (2018). Efficient Gradient-Free Variational Inference using Policy Search. In *Proceedings of Machine Learning Research*, pages 234–243.

Arenz, O., Zhong, M., and Neumann, G. (2020). Trust-region variational inference with gaussian mixture models. *The Journal of Machine Learning Research*, 21(1):6534–6593.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bagnell, J. A. and Schneider, J. (2003). Covariant policy search. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, page 1019–1024, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Bahl, S., Mukadam, M., Gupta, A., and Pathak, D. (2020). Neural dynamic policies for end-to-end sensorimotor learning. *Advances in Neural Information Processing Systems*.

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. (2020). Emergent tool use from multi-agent autocurricula. In *International Conference on Learning Representations*.

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. PMLR.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Bellman, R. (1957). A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684.

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

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J. W., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Brandherm, F., Peters, J., Neumann, G., and Akrour, R. (2019). Learning replanning policies with direct policy search. *IEEE Robotics and Automation Letters*, 4(2):2196–2203.

Braylan, A., Hollenbeck, M., Meyerson, E., and Miikkulainen, R. (2015). Frame skip is a powerful parameter for learning to play atari. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. In *arXiv preprint*.

Celik, O., Zhou, D., Li, G., Becker, P., and Neumann, G. (2022). Specializing versatile skill libraries using local mixture of experts. In *Conference on Robot Learning*, pages 1423–1433. PMLR.

Chen, X., Wang, C., Zhou, Z., and Ross, K. (2021). Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations (ICLR)*.

Chow, Y., Nachum, O., Faust, A., Ghavamzadeh, M., and Duenez-Guzman, E. (2019). Lyapunov-based Safe Policy Optimization for Continuous Control. In *ICML Workshop RL4RealLife Submission*.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. (2018). Back to basics: Benchmarking canonical evolution strategies for playing atari. In *IJCAI*.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30.

Dalal, G., Dvijotham, K., Vecerik, M., Hester, T., Paduraru, C., and Tassa, Y. (2018). Safe Exploration in Continuous Action Spaces. In *arXiv preprint*.

Dalal, M., Pathak, D., and Salakhutdinov, R. R. (2021). Accelerating robotic reinforcement learning via parameterized action primitives. *Advances in Neural Information Processing Systems*, 34:21847–21859.

Daniel, C., Neumann, G., and Peters, J. (2012). Hierarchical relative entropy policy search. In *Artificial Intelligence and Statistics*, pages 273–281. PMLR.

Dann, C., Neumann, G., Peters, J., et al. (2014). Policy evaluation with temporal differences: A survey and comparison. *Journal of Machine Learning Research*, 15:809–883.

Degris, T., White, M., and Sutton, R. S. (2012). Off-policy actor-critic. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 179–186, Madison, WI, USA. Omnipress.

Deisenroth, M. P., Neumann, G., Peters, J., et al. (2013). A survey on policy search for robotics. *Foundations and trends in Robotics*, 2(1-2):388–403.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking Deep Reinforcement Learning for Continuous Control. *33rd International Conference on Machine Learning*, pages 2001–2014.

Dulac-Arnold, G., Levine, N., Mankowitz, D. J., Li, J., Paduraru, C., Gowal, S., and Hester, T. (2020). An empirical investigation of the challenges of real-world reinforcement learning. *arXiv preprint arXiv:2003.11881*.

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. (2020). Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO. In *International Conference on Learning Representations*.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR.

Futoma, J., Hughes, M. C., and Doshi-Velez, F. (2020). Popcorn: Partially observed prediction constrained reinforcement learning. *arXiv preprint arXiv:2001.04032*.

Ginesi, M., Meli, D., Nakawala, H., Roberti, A., and Fiorini, P. (2019). A knowledge-based framework for task automation in surgery. In *2019 19th International Conference on Advanced Robotics (ICAR)*, pages 37–42. IEEE.

Gomez-Gonzalez, S., Neumann, G., Schölkopf, B., and Peters, J. (2016). Using probabilistic movement primitives for striking movements. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 502–508. IEEE.

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., and Levine, S. (2016). Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*.

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

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195.

Heess, N., Tb, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S., et al. (2017). Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Hüttenrauch, M. and Neumann, G. (2022). Regret-aware black-box optimization with natural gradients, trust-regions and entropy control. *arXiv preprint arXiv:2206.06090*.

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., and Schaal, S. (2013). Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors. *Neural Computation*, 25(2).

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134.

Kakade, S. (2001). A natural policy gradient. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, page 1531–1538, Cambridge, MA, USA. MIT Press.

Kakade, S. M. and Langford, J. C. (2002). Approximately Optimal Approximate Reinforcement Learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 267–274.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. (2018). Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. *International Conference on Learning Representations*.

Kober, J. and Peters, J. (2008). Policy search for motor primitives in robotics. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc.

Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Kormushev, P., Calinon, S., and Caldwell, D. G. (2010). Robot motor skill coordination with em-based reinforcement learning. In *2010 IEEE/RSJ international conference on intelligent robots and systems*, pages 3232–3237. IEEE.

Kormushev, P., Calinon, S., and Caldwell, D. G. (2013). Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148.

Kupcsik, A., Deisenroth, M. P., Peters, J., Poha, L. A., Vadakkepata, P., and Neumann, G. (2017). Model-based contextual policy search for data-efficient generalization of robot skills. *Artificial Intelligence*, 247:415–439. Impact Factor: 3.333.

Lee, H., Seo, H., and Kim, H.-G. (2020). Trajectory optimization and replanning framework for a micro air vehicle in cluttered environments. *Ieee Access*, 8:135406–135415.

Levine, S. (2018). Reinforcement learning and control as probabilistic inference: Tutorial and review. *CoRR*, abs/1805.00909.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2015). End-to-End Training of Deep Visuomotor Policies. In *The Journal of Machine Learning Research*, volume 17, pages 1334–1373.

Li, G., Jin, Z., Volpp, M., Otto, F., Lioutikov, R., and Neumann, G. (2022). Prodmps: A unified perspective on dynamic and probabilistic movement primitives. *arXiv preprint arXiv:2210.01531*.

Li, Z., Zhao, T., Chen, F., Hu, Y., Su, C.-Y., and Fukuda, T. (2017). Reinforcement learning of manipulation and grasping using dynamical movement primitives for a humanoidlike mobile manipulator. *IEEE/ASME Transactions on Mechatronics*, 23(1):121–131.

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

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8:293–321.

Luo, M., Yao, J., Liaw, R., Liang, E., and Stoica, I. (2020). Impact: Importance weighted asynchronous architectures with clipped target networks. In *International Conference on Learning Representations (ICLR)*.

Maeda, G., Ewerton, M., Lioutikov, R., Amor, H. B., Peters, J., and Neumann, G. (2014). Learning interaction for collaborative tasks with probabilistic movement primitives. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 527–534. IEEE.

Mahmood, A. R., Van Hasselt, H. P., and Sutton, R. S. (2014). Weighted importance sampling for off-policy learning with linear function approximation. *Advances in neural information processing systems*, 27.

Mania, H., Guy, A., and Recht, B. (2018). Simple random search of static linear policies is competitive for reinforcement learning. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Mannor, S., Rubinstein, R. Y., and Gat, Y. (2003). The cross entropy method for fast policy search. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 512–519.

Meng, W., Zheng, Q., Shi, Y., and Pan, G. (2022). An off-policy trust region policy optimization method with monotonic improvement guarantee for deep reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 33(5):2223–2235.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

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

Mülling, K., Kober, J., Kroemer, O., and Peters, J. (2013). Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. (2016). Safe and efficient off-policy reinforcement learning. *Advances in neural information processing systems*, 29.

Nachum, O., Norouzi, M., Xu, K., and Schuurmans, D. (2018). Trust-pcl: An off-policy trust region method for continuous control. In *International Conference on Learning Representations (ICLR)*.

Neumann, G. and Peters, J. (2008). Fitted q-iteration by advantage weighted regression. *Advances in neural information processing systems*, 21.

Otto, F., Becker, P., Anh Vien, N., Ziesche, H. C., and Neumann, G. (2021). Differentiable trust region layers for deep reinforcement learning. In *International Conference on Learning Representations*.

Otto, F., Becker, P., Ngo, V. A., and Neumann, G. (2024). Vlearn: Off-policy learning with efficient state-value function estimation. under review.

Otto, F., Celik, O., Zhou, H., Ziesche, H., Ngo, V. A., and Neumann, G. (2022). Deep black-box reinforcement learning with movement primitives. In *Conference on Robot Learning*, pages 1244–1265. PMLR.

Otto, F., Zhou, H., Celik, O., Li, G., Lioutikov, R., and Neumann, G. (2023). MP3: Movement primitive-based (re-)planning policy. under review.

Pacchiano, A., Parker-Holder, J., Tang, Y., Choromanska, A., Choromanski, K., and Jordan, M. I. (2020). Learning to Score Behaviors for Guided Policy Optimization. In *Proceedings of the International Conference on Machine Learning*.

Pahič, R., Gams, A., Ude, A., and Morimoto, J. (2018). Deep encoder-decoder networks for mapping raw images to dynamic movement primitives. In *2018 IEEE International Conference on Robotics and Automation*.

Pahič, R., Ridge, B., Gams, A., Morimoto, J., and Ude, A. (2020). Training of deep neural networks for the generation of dynamic movement primitives. *Neural Networks*.

Pajarinen, J., Thai, H. L., Akrour, R., Peters, J., and Neumann, G. (2019). Compatible Natural Gradient Policy Search. *Machine Learning*, 108(8-9):1443–1466.

Paraschos, A., Daniel, C., Peters, J. R., and Neumann, G. (2013). Probabilistic movement primitives. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Peng, X. B., Kumar, A., Zhang, G., and Levine, S. (2019). Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*.

Peters, J., Muelling, K., and Altun, Y. (2010). Relative entropy policy search. In *Proceedings of the Twenty-Fourth National Conference on Artificial Intelligence (AAAI), Physically Grounded AI Track*.

Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697.

Peters, J., Vijayakumar, S., and Schaal, S. (2003). Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20.

Peters, J., Vijayakumar, S., and Schaal, S. (2005). Natural actor-critic. In *Machine Learning: ECML 2005: 16th European Conference on Machine Learning*, pages 280–291. Springer.

Petersen, K. B. and Pedersen, M. S. (2012). The matrix cookbook. Version 20121115.

Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D., and Finn, C. (2023). Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*.

Richemond, P. H. and Maginnis, B. (2017). On Wasserstein Reinforcement Learning and the Fokker-Planck equation. In *arXiv preprint*.

Rozo, L. and Dave, V. (2022). Orientation probabilistic movement primitives on riemannian manifolds. In *Conference on Robot Learning*, pages 373–383. PMLR.

Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

Schaal, S. (2006). *Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics*, pages 261–280. Springer Tokyo, Tokyo.

Schaal, S., Peters, J., Nakanishi, J., and Ijspeert, A. (2005). Learning movement primitives. In *Robotics research. the eleventh international symposium*, pages 561–572. Springer.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay. In *International Conference on Learning Representations*.

Schneider, J., Schumacher, P., Häufle, D., Schölkopf, B., and Büchler, D. (2023). Investigating the impact of action representations in policy gradient algorithms. In *Workshop on effective Representations, Abstractions, and Priors for Robot Learning (RAP4Robots) @ ICRA 2023*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust Region Policy Optimization. In *Proceedings of Machine Learning Research*, pages 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *International Conference on Learning Representations*.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. In *arXiv preprint*.

Sehnke, F., Osendorfer, C., Rückstiess, T., Graves, A., Peters, J., and Schmidhuber, J. (2010). Parameter-exploring policy gradients. *Neural Networks*, 21(4):551–559.

Seyde, T., Werner, P., Schwarting, W., Gilitschenski, I., Riedmiller, M., Rus, D., and Wulfmeier, M. (2022). Solving continuous control via q-learning. *International Conference on Learning Representations*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

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. In *arXiv preprint*, pages 1–19.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr.

Song, H. F., Abdolmaleki, A., Springenberg, J. T., Clark, A., Soyer, H., Rae, J. W., Noury, S., Ahuja, A., Liu, S., Tirumala, D., Heess, N., Belov, D., Riedmiller, M., and Botvinick, M. M. (2020). V-mpo: On-policy maximum a posteriori policy optimization for discrete and continuous control. In *International Conference on Learning Representations*.

Song, J. and Zhao, C. (2020). Optimistic Distributionally Robust Policy Optimization. In *arXiv preprint*, pages 15872–15882.

Stulp, F. and Sigaud, O. (2012a). Path integral policy improvement with covariance matrix adaptation. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 1547–1554, Madison, WI, USA. Omnipress.

Stulp, F. and Sigaud, O. (2012b). Policy improvement methods: Between black-box optimization and episodic reinforcement learning. *Journées Francophones Planification, Décision, et Apprentissage pour la conduite de systèmes*.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999a). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999b). Policy gradient methods for reinforcement learning with function approximation. In Solla, S., Leen, T., and Müller, K., editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press.

Swaminathan, A. and Joachims, T. (2015). The self-normalized estimator for counterfactual learning. *advances in neural information processing systems*, 28.

Takatsu, A. (2011). Wasserstein geometry of Gaussian measures. *Osaka Journal of Mathematics*, 48:1005–1026.

Tangkaratt, V., Abdolmaleki, A., and Sugiyama, M. (2018). Guide actor-critic for continuous control. In *Proceedings of the International Conference on Learning Representations*.

Tangkaratt, V., van Hoof, H., Parisi, S., Neumann, G., Peters, J., and Sugiyama, M. (2017). Policy search with high-dimensional context variables. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G. d., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Shen, A. T. J., and Younis, O. G. (2023). Gymnasium.

Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., Heess, N., and Tassa, Y. (2020). dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Villani, C. (2008). *Optimal transport: old and new*, volume 338. Springer Science & Business Media.

Wang, Y., He, H., Tan, X., and Gan, Y. (2019). Trust region-guided proximal policy optimization. In *Advances in Neural Information Processing Systems 32*, pages 626–636.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2017). Sample efficient actor-critic with experience replay. In *International Conference on Learning Representations (ICLR)*.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8:279–292.

Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J., and Schmidhuber, J. (2014). Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.

Yang, T.-Y., Rosca, J., Narasimhan, K., and Ramadge, P. J. (2020). Projection-Based Constrained Policy Optimization. In *International Conference on Learning Representations*.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. (2019). Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL).*

Zenkri, O., Vien, N. A., and Neumann, G. (2022). Hierarchical policy learning for mechanical search. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 1954–1960. IEEE.

Zhang, B., Rajan, R., Pineda, L., Lambert, N., Biedenkapp, A., Chua, K., Hutter, F., and Calandra, R. (2021). On the importance of hyperparameter optimization for model-based reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pages 4015–4023. PMLR.

Zhang, S., Boehmer, W., and Whiteson, S. (2019). Generalized off-policy actor-critic. *Advances in neural information processing systems*, 32.

Zhao, W., Queralta, J. P., and Westerlund, T. (2020). Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 737–744. IEEE.

Appendix A

A Differentiable Trust Region Projection Layer

A.1 Algorithm

Here we provide the algorithmic view of the proposed TRPL in Algorithm 2. The trust region projections themselves do not require approximations, the old policy update in the last step is the only point where we introduce an approximation. This update would normally require additional supervised regression steps that minimize the distance between the network output and the projection. However, by leveraging the regression penalty during policy optimization this optimization step can be omitted. Both approaches yield a policy, which is independent of the old policy distribution, *i.e.* it can act without the projection while maintaining the trust region. However, the penalty does not require additional computation and the policy can directly generate new trajectories, equivalently to other trust region methods, such as PPO.
The layer itself in Algorithm 3 acts as the final layer after predicting a Gaussian distribution. It projects this predicted Gaussian onto the trust region in case it is violating the specified bounds. As output, it generates a projected mean and covariance that satisfy the respective trust region bound. The entropy control in the last step can be disabled.

Algorithm 2

```
1: Initialize policy $\theta_{0,0}$
2: for $i = 0, 1, \dots, N$ do                                              ▷ epoch
3:     Collect set of trajectories $\mathcal{D}_i = \{\tau_k\}$ with $\pi(\theta_{i,0})$
4:     Compute advantage estimates $\hat{A}_t$ with GAE
5:     for $j = 0, 1, \dots, M$ do
6:         Use $\pi(\theta_{i,j})$ to predict Gaussian action distributions $\mathcal{N}(\mu_{i,j}, \Sigma_{i,j})$ for $\mathcal{D}_i$
7:         $\tilde{\pi} = \text{TrustRegionLayer}(\mu_{i,j}, \Sigma_{i,j}, \mu_{i,0}, \Sigma_{i,0})$
8:         Update policy with Adam using the following policy gradient:
           $$\theta_{i,j+1} \leftarrow \text{Adam}\left(\nabla_\theta \left[\mathbb{E}_{\pi(\theta_{i,0})}\left[\frac{\tilde{\pi}(a|s;\theta)}{\pi(a|s;\theta_{i,0})}\hat{A}_t\right] - \alpha \mathbb{E}_{\boldsymbol{s}\sim\pi(\theta_{i,0})}\left[d\left(\tilde{\pi}(\cdot|\boldsymbol{s};\theta), \pi(\cdot|\boldsymbol{s};\theta)\right)\right]\right]\Bigg|_{\theta=\theta_{i,j}}\right)$$
9:     Successive policy update: $\theta_{i+1,0} \leftarrow \theta_{i,M}$
```

Algorithm 3

```
   Initialize bounds $\varepsilon_\mu, \varepsilon_\Sigma$, temperature $\tau$ as well as target $\kappa$ and initial entropy $\mathcal{H}_0$.
1: procedure TrustRegionLayer($\mu, \Sigma, \mu_{\text{old}}, \Sigma_{\text{old}}$)
2:     if $d_{\text{mean}}(\mu, \mu_{\text{old}}) > \varepsilon_\mu$ then
3:         Compute $\tilde{\mu}$ with Equation (3.6)
4:     else
5:         $\tilde{\mu} = \mu$
6:     if $d_{\text{cov}}(\Sigma, \Sigma_{\text{old}}) > \varepsilon_\Sigma$ then
7:         Compute $\tilde{\Sigma}$ with Equations (3.7), (3.9) and (3.11)
8:     else
9:         $\tilde{\Sigma} = \Sigma$
10:    $\beta = \kappa + (\mathcal{H}_0 - \kappa)\tau^{\frac{10i}{N}}$   ▷ (Optional) entropy control as described in Section 3.3.2
11:    if $\mathcal{H}(\Sigma) < \beta$ then
12:        $c = \exp\{(\beta - \mathcal{H}(\Sigma))/\dim(a)\}$
13:        $\tilde{\Sigma} = c\tilde{\Sigma}$
14:    return $\tilde{\mu}, \tilde{\Sigma}$
```

A.2 Derivations

A.2.1 Proof of Theorem 1

This section provides a proof for Theorem 1. We mainly used the multiplicative version of the Brunn-Minkowski inequality

$$\log(\alpha|\Sigma_1| + \beta|\Sigma_2|) \geq \log(|\Sigma_1|)^{\alpha} \log(|\Sigma_2|)^{\beta}$$

where Σ_1, Σ_2 are p.s.d, α, β are positive, and $\alpha + \beta = 1$.

Frobenius Projection

$$\begin{aligned}
\mathrm{H}(\tilde{\pi}) &= 0.5 \log |2\pi e \tilde{\Sigma}| \\
&= 0.5 \log \left| 2\pi e \left(\frac{1}{\eta+1}\Sigma + \frac{\eta}{\eta+1}\Sigma_{\text{old}} \right) \right| \\
&\geq 0.5 \log \left| (2\pi e \Sigma)^{\frac{1}{\eta+1}} \det (2\pi e \Sigma_{\text{old}})^{\frac{\eta}{\eta+1}} \right| \\
&= \frac{1}{\eta+1} 0.5 \log |2\pi e \Sigma| + \frac{\eta}{\eta+1} 0.5 \log |2\pi e \Sigma_{\text{old}}| \\
&= \frac{1}{\eta+1} \mathrm{H}(\pi_\theta) + \frac{\eta}{\eta+1} \mathrm{H}(\pi_{\theta_{\text{old}}}) \geq \text{minimum} \left(\mathrm{H}(\pi_\theta), \mathrm{H}(\pi_{\theta_{\text{old}}}) \right)
\end{aligned}$$

Wasserstein Projection Let k denote the dimensionality of the distributions under consideration.

$$\begin{aligned}
\mathrm{H}(\tilde{\pi}) &= 0.5 \log (2\pi e)^k |\tilde{\Sigma}| \\
&= 0.5 \log (2\pi e)^k \left| \left(\frac{1}{\eta+1}\Sigma^{0.5} + \frac{\eta}{\eta+1}\Sigma_{\text{old}}^{0.5} \right) \right|^2 \\
&= 0.5 \log (2\pi e)^k + \log \left| \frac{1}{\eta+1}\Sigma^{0.5} + \frac{\eta}{\eta+1}\Sigma_{\text{old}}^{0.5} \right| + \log \left| \frac{1}{\eta+1}\Sigma^{0.5} + \frac{\eta}{\eta+1}\Sigma_{\text{old}}^{0.5} \right| \\
&\geq 0.5 \log (2\pi e)^k + \log \left| \Sigma^{0.5} \right|^{\frac{1}{\eta+1}} \left| \Sigma_{\text{old}}^{0.5} \right|^{\frac{\eta}{\eta+1}} + \log \left| \Sigma^{0.5} \right|^{\frac{1}{\eta+1}} \left| \Sigma_{\text{old}}^{0.5} \right|^{\frac{\eta}{\eta+1}} \\
&= 0.5 \log (2\pi e)^k + \log |\Sigma|^{\frac{1}{\eta+1}} |\Sigma_{\text{old}}|^{\frac{\eta}{\eta+1}} \\
&= 0.5 \log \left(| (2\pi e \Sigma|^{\frac{1}{\eta+1}} |2\pi e \Sigma_{\text{old}}|^{\frac{\eta}{\eta+1}} \right) \\
&= \frac{1}{\eta+1} 0.5 \log |2\pi e \Sigma| + \frac{\eta}{\eta+1} 0.5 \log |2\pi e \Sigma_{\text{old}}| \\
&= \frac{1}{\eta+1} \mathrm{H}(\pi_\theta) + \frac{\eta}{\eta+1} \mathrm{H}(\pi_{\theta_{\text{old}}}) \geq \text{minimum} \left(\mathrm{H}(\pi), \mathrm{H}(\pi_{\text{old}}) \right)
\end{aligned}$$

KL Projection

$$
\begin{aligned}
\mathrm{H}(\tilde{\pi}) &= 0.5\log|2\pi e\tilde{\Sigma}| \\
&= 0.5\log\left|\frac{1}{\eta+1}(2\pi e\Sigma)^{-1}+\frac{\eta}{\eta+1}(2\pi e\Sigma_{\mathrm{old}})^{-1}\right|^{-1} \\
&= -0.5\log\left|\frac{1}{\eta+1}(2\pi e\Sigma)^{-1}+\frac{\eta}{\eta+1}(2\pi e\Sigma_{\mathrm{old}})^{-1}\right| \\
&\leq -0.5\log\left(\left|(2\pi e\Sigma)^{-1}\right|^{\frac{1}{\eta+1}}\left|(2\pi e\Sigma_{\mathrm{old}})^{-1}\right|^{\frac{\eta}{\eta+1}}\right) \\
&= 0.5\log\left(|2\pi e\Sigma|^{\frac{1}{\eta+1}}|2\pi e\Sigma_{\mathrm{old}}|^{\frac{\eta}{\eta+1}}\right) \quad \text{(use the fact that: } \det(A^{-1})=1/\det(A)) \\
&= \frac{1}{\eta+1}0.5\log|2\pi e\Sigma|+\frac{\eta}{\eta+1}0.5\log|2\pi e\Sigma_{\mathrm{old}}| \\
&= \frac{1}{\eta+1}\mathrm{H}(\pi_\theta)+\frac{\eta}{\eta+1}\mathrm{H}(\pi_{\theta_{\mathrm{old}}}) \leq \mathrm{maximum}\left(\mathrm{H}(\pi_\theta),\mathrm{H}(\pi_{\mathrm{old}})\right)
\end{aligned}
$$

A.2.2 Mean Projection

First, we consider only the mean objective

$$
\begin{aligned}
\min_{\tilde{\mu}} \quad & (\mu-\tilde{\mu})^{\mathrm{T}}\Sigma_{\mathrm{old}}^{-1}(\mu-\tilde{\mu}) \\
\text{s.t.} \quad & (\mu_{\mathrm{old}}-\tilde{\mu})^{\mathrm{T}}\Sigma_{\mathrm{old}}^{-1}(\mu_{\mathrm{old}}-\tilde{\mu}) \leq \varepsilon_\mu,
\end{aligned}
$$

which give us the following dual

$$
\mathcal{L}(\tilde{\mu},\omega) = (\mu-\tilde{\mu})^{\mathrm{T}}\Sigma_{\mathrm{old}}^{-1}(\mu-\tilde{\mu})+\omega\left((\mu_{\mathrm{old}}-\tilde{\mu})^{\mathrm{T}}\Sigma_{\mathrm{old}}^{-1}(\mu_{\mathrm{old}}-\tilde{\mu})-\varepsilon_\mu\right). \tag{A.1}
$$

Differentiating w.r.t. $\tilde{\mu}$ yields

$$
\frac{\partial\mathcal{L}(\tilde{\mu},\omega)}{\partial\tilde{\mu}} = 2\Sigma_{\mathrm{old}}^{-1}(\tilde{\mu}-\mu)-2\omega\Sigma_{\mathrm{old}}^{-1}(\tilde{\mu}-\mu_{\mathrm{old}}).
$$

Setting the derivative to 0 and solving for $\tilde{\mu}$ gives

$$
\tilde{\mu}^* = \frac{\mu+\omega\mu_{old}}{1+\omega}.
$$

Inserting the optimal mean $\tilde{\mu}^*$ in Equation (A.1) results in

$$\begin{aligned}
L(\omega) &= \left(\frac{\mu+\omega\mu_{old}}{1+\omega}-\mu\right)^T \Sigma_{\text{old}}^{-1}\left(\frac{\mu+\omega\mu_{old}}{1+\omega}-\mu\right)+ \\
&+\omega\left(\left(\frac{\mu+\omega\mu_{old}}{1+\omega}-\mu_{old}\right)^T \Sigma_{\text{old}}^{-1}\left(\frac{\mu+\omega\mu_{old}}{1+\omega}-\mu_{old}\right)-\varepsilon_\mu\right) \\
&= \frac{\omega^2\left(\mu-\mu_{old}\right)^T\Sigma_{\text{old}}^{-1}\left(\mu-\mu_{old}\right)}{(1+\omega)^2}+\frac{\omega\left(\mu-\mu_{old}\right)^T\Sigma_{\text{old}}^{-1}\left(\mu-\mu_{old}\right)}{(1+\omega)^2}-\omega\varepsilon_\mu.
\end{aligned}$$

Thus, differentiating w.r.t ω yields

$$\frac{\partial\mathcal{L}(\omega)}{\partial\omega}=\frac{\left(\mu-\mu_{old}\right)^T\Sigma_{\text{old}}^{-1}\left(\mu-\mu_{old}\right)}{(1+\omega)^2}-\varepsilon_\mu.$$

Now solving $\frac{\partial\mathcal{L}(\omega)}{\partial\omega}\overset{!}{=}0$ for ω, we arrive at

$$\omega^*=\sqrt{\frac{\left(\mu-\mu_{old}\right)^T\Sigma_{\text{old}}^{-1}\left(\mu-\mu_{old}\right)}{\varepsilon_\mu}}-1.$$

A.2.3 Frobenius Covariance Projection

We consider the following objective for the covariance part

$$\begin{aligned}
\min_{\tilde{\Sigma}} \quad & \|\tilde{\Sigma}-\Sigma\|_F^2 \\
\text{s.t.} \quad & \|\tilde{\Sigma}-\Sigma_{\text{old}}\|_F^2\leq\varepsilon_\Sigma
\end{aligned}$$

with the corresponding Lagrangian

$$\mathcal{L}(\tilde{\Sigma},\eta)=\|\tilde{\Sigma}-\Sigma\|_F^2+\eta\left(\|\tilde{\Sigma}-\Sigma_{\text{old}}\|_F^2-\varepsilon_\Sigma\right). \tag{A.2}$$

Differentiating w.r.t. $\tilde{\Sigma}$ yields

$$\frac{\partial\mathcal{L}(\tilde{\Sigma},\eta)}{\partial\tilde{\Sigma}}=2\left(\left(\tilde{\Sigma}-\Sigma\right)+\eta\left(\Sigma_{\text{old}}-\Sigma\right)\right).$$

We can again solve for $\tilde{\Sigma}$ by setting the derivative to 0, i.e.,

$$\tilde{\Sigma}^*=\frac{\Sigma+\eta\Sigma_{\text{old}}}{1+\eta}.$$

Inserting $\tilde{\Sigma}^*$ into Equation (A.2) yields the dual function

$$g(\eta) = \|\frac{\Sigma + \eta \Sigma_{\text{old}}}{1+\eta} - \Sigma\|_F^2 + \eta \left(\|\frac{\Sigma + \eta \Sigma_{\text{old}}}{1+\eta} - \Sigma_{\text{old}}\|_F^2 - \varepsilon_\Sigma \right).$$

Differentiating w.r.t. η results in

$$\frac{\partial \mathcal{L}(\eta)}{\partial \eta} = \frac{\|\Sigma - \Sigma_{\text{old}}\|_F^2}{(1+\eta)^2} - \varepsilon_\Sigma.$$

Hence, $\frac{\partial \mathcal{L}(\eta)}{\partial \eta} \overset{!}{=} 0$ yields

$$\eta^* = \frac{\|\Sigma - \Sigma_{\text{old}}\|_F}{\sqrt{\varepsilon_\Sigma}} - 1.$$

A.2.4 Wasserstein Covariance Projection

As described in the main text, the Gaussian distributions to have been rescaled by Σ_{old}^{-1} to measure the distance in the metric space that is defined by the variance of the data. For notational simplicity, we show the derivation of the covariance projection only for the unscaled scenario. The scaled version can be obtained by a simple redefinition of the covariance matrices. For our covariance projection we are interested in solving the following optimization problem

$$\begin{aligned} \min_{\tilde{\Sigma}} \quad & \operatorname{tr}\left(\tilde{\Sigma} + \Sigma - 2\left(\Sigma^{1/2} \tilde{\Sigma} \Sigma^{1/2} \right)^{1/2} \right) \\ \text{s.t.} \quad & \operatorname{tr}\left(\tilde{\Sigma} + \Sigma_{\text{old}} - 2\left(\Sigma_{\text{old}}^{1/2} \tilde{\Sigma} \Sigma_{\text{old}}^{1/2} \right)^{1/2} \right) \leq \varepsilon_\Sigma, \end{aligned}$$

which leads to the following Lagrangian function

$$\begin{aligned} \mathcal{L}(\tilde{\Sigma}, \eta) = & \operatorname{tr}\left(\tilde{\Sigma} + \Sigma - 2\left(\Sigma^{1/2} \tilde{\Sigma} \Sigma^{1/2} \right)^{1/2} \right) \\ & + \eta \left(\operatorname{tr}\left(\tilde{\Sigma} + \Sigma_{\text{old}} - 2\left(\Sigma_{\text{old}}^{1/2} \tilde{\Sigma} \Sigma_{\text{old}}^{1/2} \right)^{1/2} \right) - \varepsilon_\Sigma \right). \end{aligned} \tag{A.3}$$

Assuming that $\tilde{\Sigma}$ commutes with Σ as well as Σ_{old}, Equation (A.3) simplifies to

$$\begin{aligned} \mathcal{L}(\tilde{\Sigma}, \eta) &= \operatorname{tr}\left(\tilde{\Sigma} + \Sigma - 2\tilde{\Sigma}^{1/2} \Sigma^{1/2} \right) + \eta \left(\operatorname{tr}\left(\tilde{\Sigma} + \Sigma_{\text{old}} - 2\tilde{\Sigma}^{1/2} \Sigma_{\text{old}}^{1/2} \right) - \varepsilon \right) \\ &= \operatorname{tr}\left(S^2 + \Sigma - 2S\Sigma^{1/2} \right) + \eta \left(\operatorname{tr}\left(S^2 + \Sigma_{\text{old}} - 2S\Sigma_{\text{old}}^{1/2} \right) - \varepsilon \right), \end{aligned} \tag{A.4}$$

where S is the unique positive semi-definite root of the positive semi-definite matrix $\tilde{\Sigma}$, i.e. $S = \tilde{\Sigma}^{1/2}$. Instead of optimizing the objective w.r.t $\tilde{\Sigma}$, we optimize w.r.t S in order, which greatly simplifies the calculation. That is, we solve

$$\frac{\partial \mathcal{L}(S,\eta)}{\partial S} = (1+\eta)2S - 2\left(\Sigma^{1/2} + \eta\Sigma_{\text{old}}^{1/2}\right) \stackrel{!}{=} 0$$

for S, which leads us to

$$S^* = \frac{\Sigma^{1/2} + \eta\Sigma_{\text{old}}^{1/2}}{1+\eta}, \tilde{\Sigma}^* \qquad = \frac{\Sigma + \eta^2\Sigma_{\text{old}} + 2\eta\Sigma^{1/2}\Sigma_{\text{old}}^{1/2}}{(1+\eta)^2}.$$

Inserting this into Equation (A.4) yields the dual function

$$g(\eta) = \frac{\eta\left(\text{tr}\left(\Sigma + \Sigma_{\text{old}} - 2\Sigma^{1/2}\Sigma_{\text{old}}^{1/2}\right)\right)}{1+\eta} - \eta\varepsilon_\Sigma$$

The derivative of the dual w.r.t. η is given by

$$\frac{\partial \mathcal{L}(\eta)}{\partial \eta} = \frac{\text{tr}\left(\tilde{\Sigma} + \Sigma_{\text{old}} - 2\tilde{\Sigma}^{1/2}\Sigma_{\text{old}}^{1/2}\right)}{(1+\lambda)^2} - \varepsilon_\Sigma.$$

Now solving $\frac{\partial \mathcal{L}(\eta)}{\partial \eta} \stackrel{!}{=} 0$ for η, we arrive at

$$\eta^* = \sqrt{\frac{\text{tr}\left(\tilde{\Sigma} + \Sigma_{\text{old}} - 2\tilde{\Sigma}^{1/2}\Sigma_{\text{old}}^{1/2}\right)}{\varepsilon_\Sigma}} - 1$$

A.2.5 KL-Divergence Projection

We derive the KL-Divergence projection in its general form, i.e., simultaneous projection of mean and covariance under an additional entropy constraint

$$\tilde{\pi}^* = \arg\min_{\tilde{\pi}} \text{KL}\left(\tilde{\pi}||\pi_\theta\right) \quad \text{s.t.} \quad \text{KL}\left(\tilde{\pi}||\pi_{\theta_{\text{old}}}\right) \leq \varepsilon, \quad \text{H}\left(\tilde{\pi}\right) \geq \beta.$$

Instead of working with this minimization problem, we consider the equivalent maximization problem

$$\tilde{\pi}^* = \arg\max_{\tilde{\pi}} -\text{KL}\left(\tilde{\pi}||\pi_\theta\right) \quad \text{s.t.} \quad \text{KL}\left(\tilde{\pi}||\pi_{\theta_{\text{old}}}\right) \leq \varepsilon, \quad \text{H}\left(\tilde{\pi}\right) \geq \beta, \tag{A.5}$$

which is similar to the one considered in *Model Based Relative Entropy Stochastic Search* (MORE) (Abdolmaleki et al., 2015), with a few distinctions. To see those distinctions let η and ω denote the Lagrangian multipliers corresponding to the KL and entropy constraint respectively and consider the Lagrangian corresponding to the optimization problem in Equation (A.5)

$$\begin{aligned}\mathcal{L} &= -\mathrm{KL}(\tilde{\pi}||\pi_\theta) + \eta\left(\varepsilon - \mathrm{KL}(\tilde{\pi}||\pi_{\theta_{\text{old}}})\right) + \omega\left(\mathrm{H}(\tilde{\pi}) - \beta\right) \\ &= \mathbb{E}_{\tilde{\pi}}\left[\log \pi_\theta\right] + \eta\left(\varepsilon - \mathrm{KL}(\tilde{\pi}||\pi_{\theta_{\text{old}}})\right) + (\omega+1)\mathrm{H}(\tilde{\pi}) - \omega\beta.\end{aligned}$$

Opposed to Abdolmaleki et al. (2015) we are not working with an unknown reward but using the log density of the target distribution π instead. Thus we do not need to fit a surrogate and can directly read off the parameters of the squared reward. They are given by the natural parameters of π, i.e, $\Lambda = \Sigma^{-1}$ and $q = \Sigma^{-1}\mu$. Additionally, we need to add a constant 1 to ω to account for the additional entropy term in the original objective, similar to (Arenz et al., 2018).

Following the derivations from Abdolmaleki et al. (2015) and Arenz et al. (2018) we can obtain a closed form solution for the natural parameters of $\tilde{\pi}$, given the Lagrangian multipliers η and ω

$$\tilde{\Lambda} = \frac{\eta\Lambda_{\text{old}} + \Lambda}{\eta + 1 + \omega} \quad \text{and} \quad \tilde{q} = \frac{\eta q_{\text{old}} + q}{\eta + 1 + \omega}. \tag{A.6}$$

To obtain the optimal Lagrangian multipliers we can solve the following convex dual

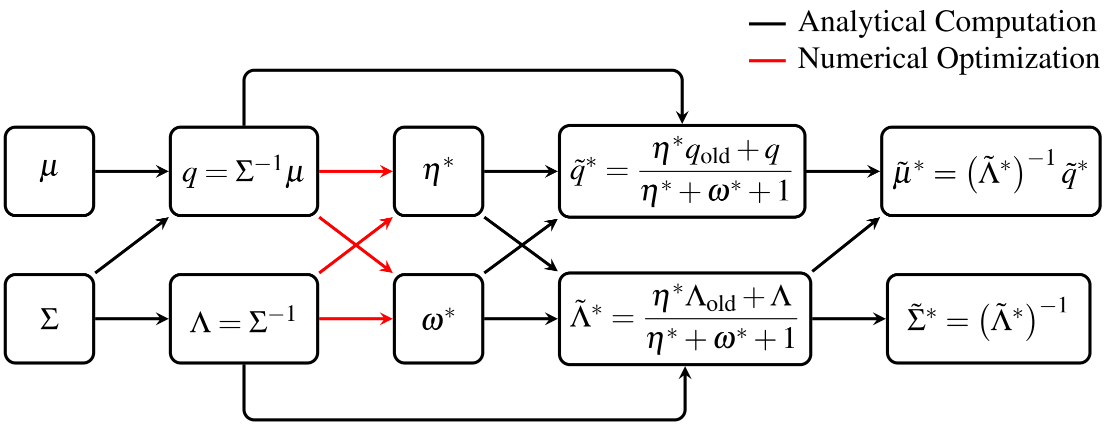

Analytical Computation
Numerical Optimization
μ
Σ
$q = \Sigma^{-1}\mu$
$\Lambda = \Sigma^{-1}$
η^*
ω^*
$\tilde{q}^* = \frac{\eta^* q_{\text{old}} + q}{\eta^* + \omega^* + 1}$
$\tilde{\Lambda}^* = \frac{\eta^*\Lambda_{\text{old}} + \Lambda}{\eta^* + \omega^* + 1}$
$\tilde{\mu}^* = (\tilde{\Lambda}^*)^{-1}\tilde{q}^*$
$\tilde{\Sigma}^* = (\tilde{\Lambda}^*)^{-1}$

Figure A.1: Compute graph of the KL projection layer. The layer first computes the natural parameters of π from the mean and covariance. Then it numerically optimizes the dual to obtain the optimal Lagrangian multipliers which are used to get the optimal natural parameters. Ultimately, the optimal mean and covariance are computed from the optimal natural parameters. We omit the dependency on constants, i.e., the bound ε and β as well as the parameters of π_{old} for clarity of the visualization.

function using gradient descent

$$\begin{aligned}g(\eta,\omega) =& \eta\varepsilon - \omega\beta + \eta\left(-\frac{1}{2}q_{\text{old}}^T\Lambda_{\text{old}}^{-1}q_{\text{old}} + \frac{1}{2}\log\det\left(\Lambda\right) - \frac{k}{2}\log(2\pi)\right)\\ &+ (\eta+1+\omega)\left(\frac{1}{2}\tilde{q}^T\tilde{\Lambda}^{-1}q - \frac{1}{2}\log\det\left(\tilde{\Lambda}\right) + \frac{k}{2}\log(2\pi)\right) + \text{const},\\ &\frac{\partial g(\eta,\omega)}{\partial\eta} = \varepsilon - \text{KL}(\tilde{\pi}||\pi_{\theta_{\text{old}}}) \quad \text{and} \quad \frac{\partial g(\eta,\omega)}{\partial\omega} = H(\tilde{\pi}) - \beta.\end{aligned}$$

Given the optimal Lagrangian multipliers, η^* and ω^* we obtain the parameters of the optimal distribution $\tilde{\pi}^*$ using Equation (A.6).

Forward Pass For the forward pass we compute the natural parameters of π, solve the optimization problem and compute mean and covariance of $\tilde{\pi}^*$ from the optimal natural parameters. The corresponding compute graph is given in Figure A.1.

Backward Pass Given the computational graph in Figure A.1 gradients can be propagated back through the layer using standard back-propagation. All gradients for the analytical computations (black arrows in Figure A.1) are straight forward and can be found in (Petersen and Pedersen, 2012). For the gradients of the numerical optimization of the dual (red arrows in Figure A.1) we follow Amos and Kolter (2017) and differentiate the KKT conditions around the optimal Lagrangian multipliers computed during the forward pass. The KKT Conditions of the dual are given by

$$\begin{aligned}&\nabla g(\eta^*,\omega^*) + m^T\nabla\begin{pmatrix}-\eta^*\\-\omega^*\end{pmatrix} = \begin{pmatrix}\varepsilon - \text{KL}\left(\tilde{\pi}^*||\pi_{\theta_{\text{old}}}\right) - m_1\\ \text{H}(\tilde{\pi}^*) - \beta - m_2\end{pmatrix} = 0, \quad \text{(Stationarity)}\\ &m_1(-\eta^*) = 0 \quad \text{and} \quad m_2(-\omega^*) = 0 \quad \text{(Complementary Slackness)}\end{aligned}$$

here $m = (m_1, m_2)^T$ denotes the Lagrangian multipliers for the box constraints of the dual (η and ω need to be non-negative). Taking the differentials of those conditions yields the equation system

$$\begin{pmatrix} -\frac{\partial \mathrm{KL}\left(\tilde{\pi}^*||\pi_{\theta_{\mathrm{old}}}\right)}{\partial \eta^*} & -\frac{\partial \mathrm{KL}\left(\tilde{\pi}^*||\pi_{\theta_{\mathrm{old}}}\right)}{\partial \omega^*} & -1 & 0 \\ \frac{\partial \mathrm{H}(\tilde{\pi}^*)}{\partial \eta^*} & \frac{\partial \mathrm{H}(\tilde{\pi}^*)}{\partial \omega^*} & 0 & -1 \\ -m_1 & 0 & -\eta^* & 0 \\ 0 & -m_2 & 0 & -\omega^* \end{pmatrix} \begin{pmatrix} d\eta \\ d\omega \\ dm_1 \\ dm_2 \end{pmatrix}$$
$$= \begin{pmatrix} \frac{\partial \mathrm{KL}\left(\tilde{\pi}^*||\pi_{\theta_{\mathrm{old}}}\right)}{\partial q} dq + \frac{\partial \mathrm{KL}\left(\tilde{\pi}^*||\pi_{\theta_{\mathrm{old}}}\right)}{\partial \Lambda} d\Lambda \\ -\frac{\partial \mathrm{H}(\tilde{\pi}^*)}{\partial q} dq - \frac{\partial \mathrm{H}(\tilde{\pi}^*)}{\partial \Lambda} d\Lambda \\ 0 \\ 0 \end{pmatrix}$$

which is (analytically) solved to obtain the desired partial derivatives

$$\frac{\partial \eta}{\partial q}, \frac{\partial \eta}{\partial \Lambda}, \frac{\partial \omega}{\partial q}, \text{ and } \frac{\partial \omega}{\partial \Lambda}.$$

Implementation We implemented the entire layer using C++, Armadillo, and OpenMP for parallelization. The implementation stores all necessary quantities for the backward pass, so numerical optimization is only necessary during the forward pass. Before we do numerical optimization, we verify whether it is necessary. If the target distribution $\tilde{\pi}$ is within the trust region, we can immediately set $\tilde{\pi}^* = \pi_\theta$, *i.e.*, the forward and backward passes become the identity mapping. This check yield significant speed-ups, especially in early iterations, if the target is still close to the old distribution. If the projection is necessary we use the L-BFGS to optimize the 2D convex dual, which is still fast. For example, for a 17-dimensional action space and a batch size of 512, such as in the Humanoid-v2 experiments, the layer takes roughly 170ms for the forward pass and 3.5ms for the backward pass if the all 512 Gaussians are actually projected[1]. If none of the Gaussians needs to be projected its less than 1ms for forward and backward pass.

Simplifications If only diagonal covariances are considered the implementation simplifies significantly, as computationally heavy operations (matrix inversions and cholesky decompositions) simplify to pointwise operations (divisions and square roots). If only the covariance part of the KL is projected, we set $\mu_{\mathrm{old}} = \mu = \tilde{\mu}^*$ and $d\mu = 0$ which is again a simplification for both the derivations and implementation. If an entropy equality constraint, instead of an inequality constraint, it is sufficient to remove the $\omega > 0$ constraint in the dual optimization.

[1]On a 8 Core Intel Core i7-9700K CPU @ 3.60GHz

A.3 Additional Results

Figure A.2 shows the training curves for all Mujoco environments with a 95% confidence interval. Besides the projections we also show the performance for PAPI and PPO. In Figure A.3 the projections also leverage the Entropy control based on the results from from Akrour et al. (2019).

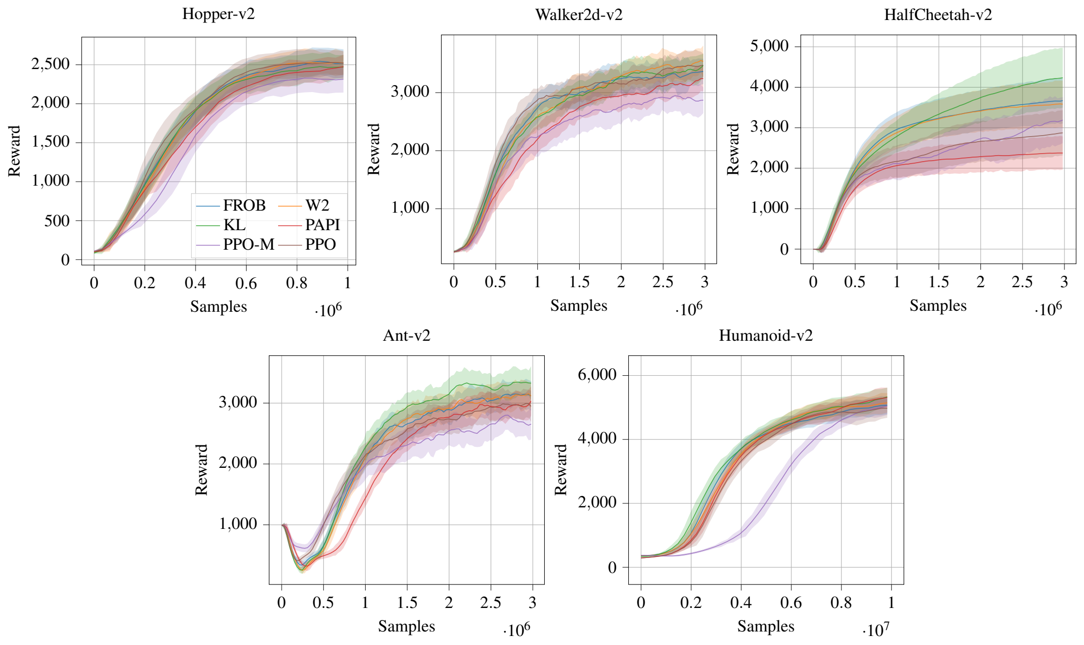


Figure A.2: Training curves for the projection layer as well as PPO and PAPI on the test environment. We trained 40 agents with different seeds for each environment using five evaluation episodes for every data point. The plot shows the total mean reward with 95% confidence interval.

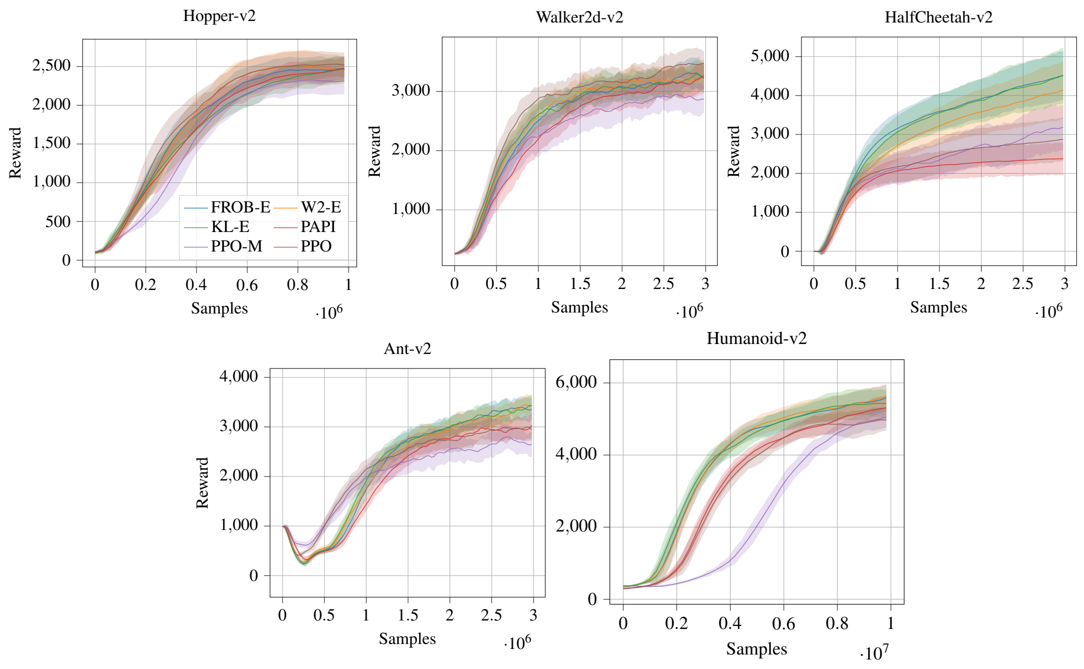


Figure A.3: Training curves for the projection layer with entropy control (-E) as well as PPO and PAPI on the test environment. We trained 40 agents with different seeds for each environment using five evaluation episodes for every data point. The plot shows the total mean reward with 95% confidence interval.

A.4 Hyperparameters

Tables A.1 and A.2 show the hyperparameters used for the experiments in Table 3.1. Target entropy, temperature, and entropy equality are only required when the entropy projection is included in the layer, otherwise those values are ignored.

Table A.1: Hyperparameters for all three projections as well as PAPI, PPO. and PPO-M on the Mujoco benchmarks from Table 3.1

	Frobenius	W2	KL	PAPI	PPO	PPO-M
rollouts	2048					
GAE λ	0.95					
discount factor	0.99					
$\varepsilon_\mu/\varepsilon$	0.03			0.015	n.a.	
ε_Σ	0.001			n.a.		
target entropy	0			n.a.		
temperature	0.5			n.a.		
entropy equality	False			False	n.a.	
optimizer	adam					
epochs vf	10					
epochs	20			10		
lr	5e-5			3e-4		
lr vf	4.5e-4			2.5e-4		
minibatch size	32			64		
trust region loss weight α	8.0			n.a.		
entropy loss penalty	0					
normalized observations	True					
normalized rewards	False			True		False
observation clip	n.a.			10		n.a.
reward clip	n.a.			10		n.a.
vf clip	n.a.			0.2		n.a.
importance ratio clip	n.a.			0.2		
contextual std	False					
hidden layers	[64, 64]					
hidden layers vf	[64, 64]					
hidden activation	tanh					

Table A.2: Hyperparameters for all three projection as well as PAPI and PPO on the Humanoid-v2 from Table 3.1.

	Frobenius	W2	KL	PAPI	PPO	PPO-M
rollouts	16384					
GAE λ	0.95					
discount factor	0.99					
$\varepsilon_\mu/\varepsilon$	0.05			0.015	n.a.	
ε_Σ	1e-4	0.01	1e-4	n.a.		
target entropy	0			n.a.		
temperature	0.2			n.a.		
entropy equality	True			False	n.a.	
optimizer	adam					
epochs vf	10					
epochs	10					
lr	1e-4			1e-4		
lr vf	4.5e-4			1e-4		
minibatch size	256			512		
entropy loss penalty	0					
trust region loss weight α	8.0			n.a.		
normalized observations	True					
normalized rewards	False			True		False
observation clip	n.a.			10		n.a.
reward clip	n.a.			10		n.a.
vf clip	n.a.			0.2		n.a.
importance ratio clip	n.a.			0.2		
contextual std	False					
hidden layers	[64, 64]					
hidden layers vf	[64, 64]					
hidden activation	tanh					

Table A.3: Hyperparameters for all three projection as well as PAPI and PPO on our ReacherSparse-v0 task from Figure 3.3. The second value for ε_Σ of the KL projection is the bound when using a contextual covariance.

	Frobenius	W2	KL	PAPI	PPO	PPO-M
rollouts	16384	16384	16384	16384	16384	16384
GAE λ	0.95	0.95	0.95	0.95	0.95	0.95
discount factor	0.99	0.99	0.99	0.99	0.99	0.99
$\varepsilon_\mu/\varepsilon$	0.03	0.03	0.03	0.03	n.a.	n.a.
ε_Σ	5e-5	1e-3	5e-5/1e-3	n.a.	n.a.	n.a.
target entropy	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
temperature	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
entropy equality	n.a.	n.a.	n.a.	False	n.a.	n.a.
optimizer	adam	adam	adam	adam	adam	adam
epochs vf	10	10	10	10	10	10
epochs	20	20	20	20	20	20
lr	3e-4	3e-4	3e-4	1e-4	1e-4	1e-4
lr vf	4.5e-4	4.5e-4	4.5e-4	1e-4	1e-4	1e-4
minibatch size	256	256	256	512	512	512
entropy loss penalty	0	0	0	0	0	0
trust region penalty α	8.0	8.0	8.0	n.a.	n.a.	n.a.
normalized observations	True	True	True	True	True	True
normalized rewards	False	False	False	True	True	False
observation clip	n.a.	n.a.	n.a.	10	10	n.a.
reward clip	n.a.	n.a.	n.a.	10	10	n.a.
vf clip	n.a.	n.a.	n.a.	0.2	0.2	n.a.
importance ratio clip	n.a.	n.a.	n.a.	0.2	0.2	0.2
contextual std	False	False	False	False	False	False
hidden layers	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64, 64]
hidden layers vf	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64, 64]
hidden activation	tanh	tanh	tanh	tanh	tanh	tanh

Appendix B

MP3: Movement Primitive-Based (Re-)Planning Policy

B.1 Derivations of Probabilistic Dynamic Movement Primitives

In this section, we will briefly present the main derivations of ProDMPs. We start with the fundamental aspects of DMPs and then derive ProDMPs from the analytical solution of the DMPs' ODE. Finally, we present the solution to a initial value problem of the ODE, which allows us to perform smooth replanning during trajectory execution in a computationally efficient manner. For the sake of simplicity, we introduce the approach by means of a 1-DoF dynamical system. For higher DoF systems, we refer to the original paper by Li et al. (2022).

Dynamic Movement Primitives Schaal (2006); Ijspeert et al. (2013) model a single movement execution as a trajectory $\boldsymbol{\lambda} = [y_t]_{t=0:T}$ using a second-order linear dynamical system with a non-linear forcing function f,

$$\tau^2\ddot{y} = \alpha(\beta(g-y) - \tau\dot{y}) + f(x), \quad f(x) = x\frac{\sum \varphi_i(x)w_i}{\sum \varphi_i(x)} = x\boldsymbol{\varphi}_x^\intercal \boldsymbol{w}, \tag{B.1}$$

where $y = y(t)$, $\dot{y} = \mathrm{d}y/\mathrm{d}t$, $\ddot{y} = \mathrm{d}^2y/\mathrm{d}t^2$ represent the position, velocity, and acceleration of the system at time step t, respectively. α and β are spring-damper constants, g is a goal attractor, and τ is a time constant that can be used to adjust the execution speed of the resulting trajectory. To achieve goal convergence, DMPs define the forcing function based on an exponentially decaying phase variable $x(t) = \exp(-\alpha_x/\tau\, t)$, where $\varphi_i(x)$ represents the (unnormalized) basis functions. The shape of the trajectory as it converges to the goal is controlled by the weights $w_i \in \boldsymbol{w}$, $i = 1...N$. The trajectory of the motion $\boldsymbol{\lambda}$ is obtained by integrating the system numerically from the starting time to the target time point. However, this process is often computationally expensive.

Solving the Dynamic Movement Primitives' Underlying ODE. Li et al. recognize that the governing equation of DMPs, given in Equation (B.1), has an analytical solution, as it is a second-order linear non-homogeneous ODE with constant coefficients. To better convey this method, the ODE and its homogeneous counterpart can be rewritten in a standard form as:

$$\textbf{Non-homo. ODE:} \quad \ddot{y} + \frac{\alpha}{\tau}\dot{y} + \frac{\alpha\beta}{\tau^2}y = \frac{f(x)}{\tau^2} + \frac{\alpha\beta}{\tau^2}g \equiv F(x,g), \tag{B.2}$$

$$\textbf{Homo. ODE:} \quad \ddot{y} + \frac{\alpha}{\tau}\dot{y} + \frac{\alpha\beta}{\tau^2}y = 0. \tag{B.3}$$

With appropriate configuration of the spring-damper coefficients, *i.e.*, $\beta = \alpha/4$ (Schaal (2006); Ijspeert et al. (2013)), the system is critically damped and the motion generated by the DMPs will settle to the target position smoothly and efficiently. The analytical solution of Equation (B.2) in this case takes the form

$$y = c_1y_1 + c_2y_2 - y_1\int\frac{y_2F}{Y}\mathrm{d}t + y_2\int\frac{y_1F}{Y}\mathrm{d}t, \quad Y = y_1\dot{y}_2 - \dot{y}_1y_2, \tag{B.4}$$

$$y_1 = y_1(t) = \exp\left(-\frac{\alpha}{2\tau}t\right), \qquad y_2 = y_2(t) = t\exp\left(-\frac{\alpha}{2\tau}t\right), \tag{B.5}$$

where y_1 and y_2 are the complementary functions of the homogeneous function in Equation (B.3) and $\dot{y}_1$, $\dot{y}_2$ their corresponding derivatives w.r.t. time. By utilizing the fundamental of calculus, which states that $\int h(t)\mathrm{d}t = \int_0^t h(t')\mathrm{d}t' + c$, where $c \in \mathbb{R}$ is a constant, the two indefinite integrals in Equation (B.4) can be transformed into two definite integrals. During this transformation, the learnable parameters $\boldsymbol{w}$ and g which control the shape of the trajectory, can be extracted from the resulting definite integrals. Finally, the trajectory position and velocity can be expressed in a compact matrix form as

$$y = c_1y_1 + c_2y_2 + \begin{bmatrix} y_2\boldsymbol{p_2} - y_1\boldsymbol{p_1} & y_2q_2 - y_1q_1 \end{bmatrix}\begin{bmatrix}\boldsymbol{w}\\ g\end{bmatrix} \tag{B.6}$$

$$\dot{y} = c_1\dot{y}_1 + c_2\dot{y}_2 + \begin{bmatrix} \dot{y}_2\boldsymbol{p_2} - \dot{y}_1\boldsymbol{p_1} & \dot{y}_2q_2 - \dot{y}_1q_1 \end{bmatrix}\begin{bmatrix}\boldsymbol{w}\\ g\end{bmatrix}, \tag{B.7}$$

where $\boldsymbol{p}_1$, $\boldsymbol{p}_1$, q_1, q_2 represent the elements used to formulate the definite integrals in the matrix form, as

$$\boldsymbol{p}_1(t) = \frac{1}{\tau^2}\int_0^t t'\exp\left(\frac{\alpha}{2\tau}t'\right)x(t')\boldsymbol{\varphi}_x^\intercal\mathrm{d}t', \quad \boldsymbol{p}_2(t) = \frac{1}{\tau^2}\int_0^t \exp\left(\frac{\alpha}{2\tau}t'\right)x(t')\boldsymbol{\varphi}_x^\intercal\mathrm{d}t', \tag{B.8}$$

$$q_1(t) = \left(\frac{\alpha}{2\tau}t - 1\right)\exp\left(\frac{\alpha}{2\tau}t\right) + 1, \quad q_2(t) = \frac{\alpha}{2\tau}\left[\exp\left(\frac{\alpha}{2\tau}t\right) - 1\right]. \tag{B.9}$$

It is worth noting that, despite the closed form solution for q_1 and q_2, $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ cannot be obtained analytically because of the complex nature of the $\boldsymbol{\varphi}_x$. As a result, they must be computed numerically. However, the extraction of the learnable parameters $\boldsymbol{w}$ and g from the integrals in Equations (B.6) and (B.7)) enables the sharing of the remaining integrals among all trajectories to be generated. In other words, these integrals can be pre-computed offline and used as constant functions during online trajectory computation, which significantly simplifies the trajectory generation procedure and speeds it up. These remaining integrals are referred to as the position basis $\boldsymbol{\Phi}(t)$ and velocity basis $\dot{\boldsymbol{\Phi}}(t)$, and the ProDMPs represent the position and velocity in a similar manner of ProMPs as:

$$y(t) = c_1 y_1(t) + c_2 y_2(t) + \boldsymbol{\Phi}(t)^\intercal \boldsymbol{w}_g, \quad \dot{y}(t) = c_1 \dot{y}_1(t) + c_2 \dot{y}_2(t) + \dot{\boldsymbol{\Phi}}(t)^\intercal \boldsymbol{w}_g, \tag{B.10}$$

where $\boldsymbol{w}_g$ is a concatenation vector containing $\boldsymbol{w}$ and g.

Solve the initial value problem. To compute the coefficients c_1 and c_2, a solution to the initial value problem represented by Equation (B.10) must be found. Li et al. suggest using the current robot state, which consists of the robot's position and velocity $(y_b, \dot{y}_b)$ at the replanning time step t_b, as the natural condition for ensuring a smooth transition between the previous and newly generated trajectory. We denote the values of the complementary functions and their derivatives at time t_b as $y_{1_b}, y_{2_b}, \dot{y}_{1_b} \dot{y}_{2_b}$, and the values of the position and velocity basis functions as $\boldsymbol{\Phi}_b, \dot{\boldsymbol{\Phi}}_b$. By substituting these values into Equation (B.10), c_1 and c_2 can be calculated as:

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \frac{\dot{y}_{2_b} y_b - y_{2_b} \dot{y}_b}{y_{1_b} \dot{y}_{2_b} - y_{2_b} \dot{y}_{1_b}} + \frac{y_{2_b} \dot{\boldsymbol{\Phi}}_b^\intercal - \dot{y}_{2_b} \boldsymbol{\Phi}_b^\intercal}{y_{1_b} \dot{y}_{2_b} - y_{2_b} \dot{y}_{1_b}} \boldsymbol{w}_g \\ \frac{y_{1_b} \dot{y}_b - \dot{y}_{1_b} y_b}{y_{1_b} \dot{y}_{2_b} - y_{2_b} \dot{y}_{1_b}} + \frac{\dot{y}_{1_b} \boldsymbol{\Phi}_b^\intercal - y_{1_b} \dot{\boldsymbol{\Phi}}_b^\intercal}{y_{1_b} \dot{y}_{2_b} - y_{2_b} \dot{y}_{1_b}} \boldsymbol{w}_g \end{bmatrix}. \tag{B.11}$$

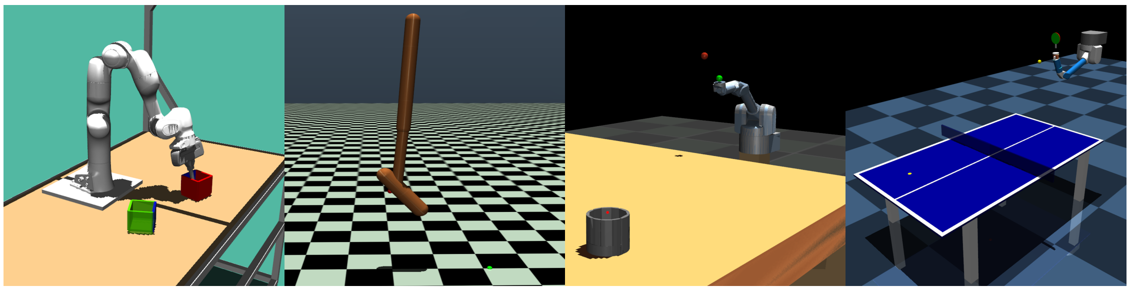


Figure B.1: Visualization of the four control tasks box pushing, hopper jumping, beer pong, and table tennis.

B.2 Environment Details

B.2.1 Reacher5d

For the Reacher task we modify the original OpenAI gym Reacher-v2 by adding three additional joints, resulting in a total of five joints. The task goal is still to minimize the distance between the goal point $\mathbf{p}_{goal}$ and the end-effector $\mathbf{p}$. We, however, only sample the goal point for $y \geq 0$, *i.e.* in the first two quadrants, to slightly reduce task complexity while maintaining the increased control complexity. The observation space remains unchanged, unless for the sparse reward where we additionally add the current step value to make learning possible for step-based methods. The context space only contains the coordinates of the goal position. The action space is the 5d equivalent to the original version.
For the reward the original setting leverages the goal distance

$$R_{\text{goal}} = \|\mathbf{p} - \mathbf{p}_{goal}\|_2$$

and the action cost

$$\tau_t = \sum_i^K (a_t^i)^2,$$

Dense Reward. The dense reward in the 5d setting, hence, stays the same and the agent receives in each time step t

$$R_{\text{tot}} = -\tau_t - R_{\text{goal}}$$

Sparse Reward. The sparse reward only returns the task reward in the last time step T and additionally adds a velocity penalty $R_{\text{vel}} = \sum_i^K (\dot{q}_T^i)^2$, where $\dot{\mathbf{q}}$ are the joint velocities, to avoid overshooting

$$R_{\text{tot}} = \begin{cases} -\tau_t & t < T, \\ -\tau_t - 200R_{\text{goal}} - 10R_{\text{vel}} & t = T. \end{cases}$$

B.2.2 Box Pushing

The goal of the box-pushing task is to move a box to a specified goal location and orientation using the seven DoF Franka Emika Panda. Hence, the context space for this task is the goal position $x \in [0.3, 0.6]$, $y \in [-0.45, 0.45]$ and the goal orientation $\theta \in [0, 2\pi]$. In addition to the contexts, the observation space for the step-based algorithms contains the positions and velocities of the joint angles, as well as position and orientation quaternions for the actual box and the target. For the action space we use the torques per joint and additionally add gravity compensation in each time step, that does not have to be learned. The task is considered successfully solved if the position distance ≤ 0.05m and the orientation error ≤ 0.5rad. For the total reward we consider different sub-rewards. First, the distance to the goal

$$R_{\text{goal}} = ||\mathbf{p} - \mathbf{p}_{goal}||,$$

where $\mathbf{p}$ is the box position and $\mathbf{p}_{goal}$ the goal position itself. Second, the rotation distance

$$R_{\text{rotation}} = \frac{1}{\pi} \arccos |\mathbf{r} \cdot \mathbf{r}_{goal}|,$$

where $\mathbf{r}$ and $\mathbf{r}_{goal}$ are the box orientation and goal orientation in quaternion, respectively. Third, an incentive to keep the rod within the box

$$R_{\text{rod}} = \text{clip}(||\mathbf{p} - \mathbf{h}_{pos}||, 0.05, 10)$$

where $\mathbf{h}_{pos}$ is the position of the rod tip. Fourth, a similar incentive that encourages to maintain the rod in a desired rotation

$$R_{\text{rod\_rotation}} = \text{clip}(\frac{2}{\pi} \arccos |\mathbf{h}_{rot} \cdot \mathbf{h}_0|, 0.25, 2),$$

where $\mathbf{h}_{rot}$ and $\mathbf{h}_0 = (0.0, 1.0, 0.0, 0.0)$ are the current and desired rod orientation in quaternion, respectively. And lastly, we utilize the following error

$$\text{err}(\mathbf{q}, \dot{\mathbf{q}}) = \sum_{i \in \{i | |q_i| > |q_i^b|\}} (|q_i| - |q_i^b|) + \sum_{j \in \{j | |\dot{q}_j| > |\dot{q}_j^b|\}} (|\dot{q}_j| - |\dot{q}_j^b|).$$

Here, $\mathbf{q}$, $\dot{\mathbf{q}}$, $\mathbf{q}^b$, and $\dot{\mathbf{q}}^b$ are the robot joint's position and velocity as well as their respective bounds. Additionally, we consider an action cost in each time step t

$$\tau_t = \sum_i^K (a_t^i)^2,$$

where $K = 7$ is the number of DoF. Similar to the aforementioned reacher task, we consider both dense and sparse reward setups.

Dense Reward. The dense reward provides information about the goal and rotation distance in each time step t on top of the utility rewards

$$R_{\text{tot}} = -R_{\text{rod}} - R_{\text{rod\_rotation}} - 5e^{-4}\tau_t - \text{err}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - 3.5R_{\text{goal}} - 2R_{\text{rotation}}.$$

Temporal Sparse Reward. The time-dependent sparse reward is similar to the dense reward, but only returns the goal and rotation distance in the last time step T

$$R_{\text{tot}} = \begin{cases} -R_{\text{rod}} - R_{\text{rod\_rotation}} - 0.02\tau_t - \text{err}(\mathbf{q}, \dot{\mathbf{q}}), & t < T, \\ -R_{\text{rod}} - R_{\text{rod\_rotation}} - 0.02\tau_t - \text{err}(\mathbf{q}, \dot{\mathbf{q}}) - 350R_{\text{goal}} - 200R_{\text{rotation}}, & t = T. \end{cases}$$

Goal Switching. To demonstrate the ability of our algorithm to handle the changing goal. We randomly switch to a new target at 20% of the max episode length. To ensure that the new target is solvable within the given episode length, we sample its position near to the old target position. The new target position, denoted as $[x_{new}, y_{new}]$, is computed as follows:

$$[x_{new}, y_{new}] = [x_{old}, y_{old}] + [\Delta x, \Delta y],$$

where $\Delta x, \Delta y$ are randomly sampled within the range $[-0.25, 0.2]$. Additionally, the new target orientation is determined by uniformly sampling a value from the range of $[0, 2\pi]$.

B.2.3 Hopper Jump

In the Hopper jump task the agent has to learn to jump as high as possible and land on a certain goal position at the same time. We consider five basis functions per joint resulting in an 15 dimensional weight space. The context is four-dimensional consisting of the initial joint angles $\theta \in [-0.5, 0]$, $\gamma \in [-0.2, 0]$, $\phi \in [0, 0.785]$ and the goal landing position $x \in [0.3, 1.35]$. The full observation space extends the original observation space from the OpenAI gym Hopper by adding the x-value of the goal position and the x-y-z difference between the goal point and the reference point at the Hopper's foot. The action space is the same as for the original Hopper task. We consider a non-Markovian reward function for the episode-based algorithms and a step-based reward for PPO, which we have extensively designed to obtain the highest possible jump.

Non-Markovian Reward. In each time-step t we provide an action cost

$$\tau_t = 10^{-3}\sum_{i}^{K}(a_t^i)^2,$$

where $K = 3$ is the number of DoF. In the last time-step T of the episode we provide a reward which contains information about the whole episode as

$$\begin{aligned}
R_{height} &= 10h_{max}, \\
R_{gdist} &= ||p_{foot,T} - p_{goal}||_2, \\
R_{cdist} &= ||p_{foot,contact} - p_{goal}||_2, \\
R_{healthy} &= \begin{cases} 2 & \text{if } z_T \in [0.5, \infty] \text{and } \theta, \gamma, \phi \in [-\infty, \infty] \\ 0 & \text{else} \end{cases},
\end{aligned}$$

where h_{max} is the maximum jump height in z-direction of the center of mass reached during the whole episode, $p_{foot,t}$ is the x-y-z position of the foot's heel at time step t, $p_{foot,contact}$ is the foot's heel position when having a contact with the ground after the first jump, p_{goal} is the goal landing position of the heel. $R_{healthy}$ is a slightly modified reward of the healthy reward defined in the original hopper task. The hopper is considered as 'healthy' if the z position of the center of mass is within the range $[0.5m, \infty]$. This encourages the hopper to stand at the end of the episode. Note that all states need to be within the range $[-100, 100]$ for $R_{healthy}$. Since this is defined in the hopper task from OpenAI already, we haven't mentioned it here. The total reward at the end of an episode is given as

$$R_{tot} = -\sum_{t=0}^{T} \tau_t + R_{height} + R_{gdist} + R_{cdist} + R_{healthy}.$$

Step-Based Reward. We consider a step-based alternative reward such that PPO is also able to learn a meaningful behavior on this task. We have tuned the reward such that we can obtain the best performance. The observation space is the same as in the original hopper task from OpenAI extended with the goal landing position and the current distance of the foot's heel and the goal landing postion. We again consider the action cost in each time-step t

$$\tau_t = 10^{-3} \sum_{i}^{K} (a_t^i)^2,$$

and additionally consider the rewards

$$\begin{aligned}
R_{height,t} &= 3h_t \\
R_{gdist,t} &= 3||p_{foot,t} - p_{goal}||_2 \\
R_{healthy,t} &= \begin{cases} 1 & \text{if } z_t \in [0.5, \infty] \text{and } \theta, \gamma, \phi \in [-\infty, \infty] \\ 0 & \text{else} \end{cases},
\end{aligned}$$

where these rewards are now returned to the agent in each time-step t, resulting in the reward per time-step

$$r_t(s_t, a_t) = -\tau_t + R_{height,t} + R_{gdist,t} + R_{healthy,t}.$$

B.2.4 Beer Pong

In the Beer Pong task the $K = 7$ DoF robot has to throw a ball into a cup on a big table. The context is defined by the cup's two dimensional position on the table which lies in the range $x \in [-1.42, 1.42]$, $y \in [-4.05, -1.25]$. For the step-based algorithms we consider cosine and sine of the robot's joint angles, the angle velocities, the ball's distance to the bottom of the cup, the ball's distance to the top of the cup, the cup position and the current time step. The action space for the step-based algorithms is defined as the torques for each joint, the parameter space for the episode-based methods is 15 dimensional which consists of the two weights for the basis functions per joint and the duration of the throwing trajectory, *i.e.* the ball release time.
We generally consider action penalties

$$\tau_t = \frac{1}{K}\sum_i^K (a_t^i)^2,$$

consisting of the sum of squared torques per joint. For $t < T$ we consider the reward

$$r_t(s_t, a_t) = -\alpha_t \tau_t,$$

with $\alpha_t = 10^{-2}$. For $t = T$ we consider the non-Markovian reward

$$R_{task} = \begin{cases} -4 - min(||p_{c,top} - p_{b,1:T}||_2^2) - 0.5||p_{c,bottom} - p_{b,T}||_2^2 \cdots & \\ \cdots - 2||p_{c,bottom} - p_{b,k}||_2^2 - \alpha_T \tau, & \text{if cond. 1} \\ -4 - min(||p_{c,top} - p_{b,1:T}||_2^2) - 0.5||p_{c,bottom} - p_{b,T}||_2^2 - \alpha_T \tau, & \text{if cond. 2} \\ -2 - min(||p_{c,top} - p_{b,1:T}||_2^2) - 0.5||p_{c,bottom} - p_{b,T}||_2^2 - \alpha_T \tau, & \text{if cond. 3} \\ -||p_{c,bottom} - p_{b,T}||_2^2 - \alpha_T \tau, & \text{if cond. 4} \end{cases}$$

$$R_{task} = \begin{cases} -4 - min(||p_{c,top} - p_{b,1:T}||_2^2) - 0.5||p_{c,bottom} - p_{b,T}||_2^2 \cdots & \\ \cdots - 2||p_{c,bottom} - p_{b,k}||_2^2 - \alpha_T \tau, & \text{if cond. 1} \\ -4 - min(||p_{c,top} - p_{b,1:T}||_2^2) - 0.5||p_{c,bottom} - p_{b,T}||_2^2 - \alpha_T \tau, & \text{if cond. 2} \\ -2 - min(||p_{c,top} - p_{b,1:T}||_2^2) - 0.5||p_{c,bottom} - p_{b,T}||_2^2 - \alpha_T \tau, & \text{if cond. 3} \\ -||p_{c,bottom} - p_{b,T}||_2^2 - \alpha_T \tau, & \text{if cond. 4} \end{cases},$$

where $p_{c,top}$ is the position of the top edge of the cup, $p_{c,bottom}$ is the ground position of the cup, $p_{b,t}$ is the position of the ball at time point t, and τ is the squared mean torque over all joints during one rollout and $\alpha_T = 10^{-4}$. The different conditions are:

- cond. 1: The ball had a contact with the ground before having a contact with the table.
- cond. 2: The ball is not in the cup and had no table contact
- cond. 3: The ball is not in the cup and had table contact
- cond. 4: The ball is in the cup.

Note that $p_{b,k}$ is the ball's and the ground's contact position and is only given, if the ball had a contact with the ground first.
At time step $t = T$ we also give information whether the agent's chosen ball release time B was reasonable

$$R_{release} = \begin{cases} -30 - 10(B - B_{min})^2, & \text{if } B < B_{min} \\ -30 - 10(B - B_{max})^2, & \text{if } B < B_{max} \end{cases},$$

where we define $B_{min} = 0.1s$ and $B_{max} = 1s$, such that the agent is encouraged to throw the ball within the time range $[B_{min}, B_{max}]$.
The total return over the whole episode is therefore given as

$$R_{tot} = \sum_{t=1}^{T-1} r_t(s_t, a_t) + R_{task} + R_{release}$$

A throw is considered as successful if the ball is in the cup at the end of an episode.

B.2.5 Table Tennis

We consider table tennis for the entire table, *i.e.* incoming balls are anywhere on the side of the robot and goal locations anywhere on the opponents side. The goal is to use the 7 DoFs robotic arm to hit the incoming ball based on its landing position and return it as close as possible to the specified goal location. As context space we consider the initial ball position $x \in [-1, -0.2]$, $y \in [-0.65, 0.65]$ and the goal position $x \in [-1.2, -0.2]$, $y \in [-0.6, 0.6]$. The full observation space again contains the positions and velocities of the joints on top of the above context information. The torques of the joints make up the action space. For this experiment, we do not use any gravity compensation and allow in the episode-based setting to learn the start time t_0 and the trajectory duration T. The task is considered successful if the returned ball lands on the opponent's side of the table and within ≤ 0.2m to the goal location. The max episode length of the table tennis environment is 350 steps. However, to accelerate the simulation, the episode will end immediately if any of the following terminated conditions are met:

- terminated cond. 1: A contact between the ball and the floor is detected,

- terminated cond. 2: The agent has hit the ball and then a contact between the ball and the table is detected.

The reward signal in the table tennis environment is defined as

$$r_{task} = \begin{cases} 0, & \text{if cond. 1} \\ 0.2 - 0.2\tanh\left(\min ||\mathbf{p}_r - \mathbf{p}_b||^2\right), & \text{if cond. 2} \\ 3 - 2\tanh\left(\min ||\mathbf{p}_r - \mathbf{p}_b||^2\right) - \tanh\left(||\mathbf{p}_l - \mathbf{p}_{goal}||^2\right), & \text{if cond. 3} \\ 6 - 2\tanh\left(\min ||\mathbf{p}_r - \mathbf{p}_b||^2\right) - 4\tanh\left(||\mathbf{p}_l - \mathbf{p}_{goal}||^2\right), & \text{if cond. 4} \\ 7 - 2\tanh\left(\min ||\mathbf{p}_r - \mathbf{p}_b||^2\right) - 4\tanh\left(||\mathbf{p}_l - \mathbf{p}_{goal}||^2\right), & \text{if cond. 5} \end{cases}$$

where $\mathbf{p}_r$ is the position of racket center, $\mathbf{p}_b$ is the position of the ball, $\mathbf{p}_l$ is the ball landing position, $\mathbf{p}_{goal}$ is the target position. The different conditions are

- cond. 1: the end of episode is not reached,
- cond. 2: the end of episode is reached,
- cond. 3: cond.2 is satisfied and robot did hit the ball,
- cond. 4: cond.3 is satisfied and the returned ball landed on the table,
- cond. 5: cond.4 is satisfied and the landing position is at the opponent's side.

The episode ends when any of the following conditions are met

- the maximum horizon length is reached
- ball did land on the floor without hitting
- ball did land on the floor or table after hitting

For MP3-BB-PPO and MP3-BB, the whole desired trajectory is obtained ahead of environment interaction, making use of this property we can collect some samples without physical simulation. The reward function based on this desired trajectory is defined as

$$r_{traj} = -\sum_{(i,j)} |\tau^d_{ij}| - |q^b_j|, \quad (i,j) \in \{(i,j) \mid |\tau^d_{ij}| > |q^b_j|\}$$

where τ^d is the desired trajectory, i is the time index, j is the joint index, q^b is the joint position upperbound. The desired trajectory is considered as invalid if $r_{traj} < 0$, an invalid trajectory will not be executed by robot. The overall reward for the black box approaches is defined as

$$r = \begin{cases} r_{traj}, & r_{traj} < 0 \\ r_{task}, & \text{otherwise} \end{cases}$$

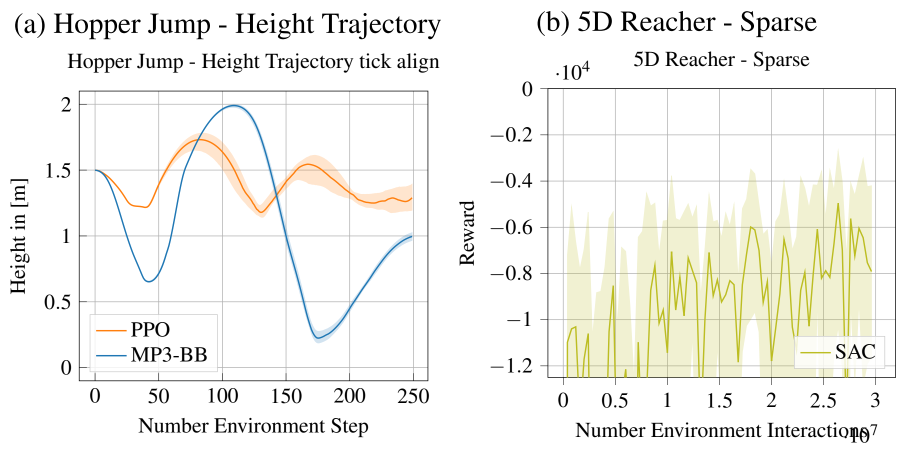


Figure B.2: a The improved performance on the Hopper Jump task is also demonstrated on the jumping profile for a fixed context. While MP3-BB jumps once as high as possible, PPO constantly tries to maximize the height at each time step which leads to several jumps throughout the episode and consequently to a lower maximum height. b Learning curve of SAC for the sparse reward of the 5D Reacher task.

Goal Switching. To evaluate the capability of our approach in handling goal changes in the presence of non-Markovian reward, we designed a goal-switching task based on the table tennis environment. Given that the episode lengths are not fixed in this environment, we fixed the target changing time at the 99-th step after the episode begins. To simplify the task and make it easier to visualize, we restricted the range of the randomly sampled initial target to the left half of the table, specifically $y \in [-0.65, 0], x \in [-1.2, 0.2]$. At the 99-th step, there is 50% of chance that the goal is switched to another random position from the right side of the table, namely $y \in [0, 0.65], x \in [-1.2, 0.2]$.
Wind as External Perturbation. To further investigate the performance of our approach in handling environmental perturbations, we introduced artificial wind to the environment. At the beginning of each episode, we randomly sample a value $f \in [-0.1, 0.1]$ to represent the constant wind force. This force was then applied as an external force to the ball at each simulation step. It's important to note that, in this specific task, we also augmented the observation space of the agent to include the velocity of the ball. By incorporating this information, the agent was able to infer the underlying "wind speed" and adjust its behavior accordingly. Since this information is not directly observable at the beginning of the episode, episode-based policies, struggled to solve the task.

B.3 Additional Evaluations

We provide some additional results for the height trajectory and the performance of the sparse 5D Reacher in Figure B.2.

B.4 Hyperparameters

For all methods, where applicable, we optimized the learning rate, sample size, batch size, number of layers, and the number of epochs. For all MP based methods, we additionally optimized the number of basis functions. Moreover, we found that NDP requires tuning of the scale of the predicted DMP weights, which was hard-coded to 100 in the original code base. However, this value only worked for the Meta-World tasks, but not for the other tasks, hence we adjusted it to allow for a fair comparison. The population size of ES is always half the number of samples because two function evaluations are used per parameter vector.

Table B.1: Hyperparameters for the 5D Reacher experiments.

	PPO	NDP	TRPL	SAC	CMORE	ES	MP3-BB-PPO	MP3-BB
number samples	16000	16000	16000	1000	120	200	64	64
GAE λ	0.95	0.95	0.95	0.95	n.a.	n.a.	n.a.	n.a.
discount factor	0.99	0.99	0.99	0.99	n.a.	n.a.	n.a.	n.a.
$\varepsilon_\mu/\varepsilon$	n.a.	n.a.	0.005	n.a.	0.1	n.a.	n.a.	0.05
ε_Σ	n.a.	n.a.	0.0005	n.a.	n.a.	n.a.	n.a.	0.0005
optimizer	adam	adam	adam	adam	n.a.	adam	adam	adam
epochs	10	10	20	1000	n.a.	n.a.	100	100
learning rate	3e-4	3e-4	5e-5	3e-4	n.a.	1e-2	3e-4	3e-4
use critic	True	True	True	True	False	False	False	False
epochs critic	10	10	10	1000	n.a.	n.a.	n.a.	n.a.
learning rate critic (and alpha)	3e-4	3e-4	3e-4	3e-4	n.a.	n.a.	n.a.	n.a.
number minibatches	32	32	64	n.a.	n.a.	n.a.	1	1
batch size	n.a.	n.a.	n.a.	256	n.a.	n.a.	n.a.	n.a.
buffer size	n.a.	n.a.	n.a.	1e6	n.a.	n.a.	n.a.	n.a.
learning starts	0	0	0	10000	0	0	0	0
polyak_weight	n.a.	n.a.	n.a.	5e-3	n.a.	n.a.	n.a.	n.a.
trust region loss weight	n.a.	n.a.	10	n.a.	n.a.	n.a.	n.a.	10
normalized observations	True	True	True	False	False	False	False	False
normalized rewards	True	True	False	False	False	False	False	False
observation clip	10.0	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	0.2	n.a.	n.a.	n.a.	n.a.	0.2	n.a.
importance ratio clip	0.2	0.2	n.a.	n.a.	n.a.	n.a.	0.2	n.a.
hidden layers	[32, 32]	[32, 32]	[32, 32]	[128,128]	n.a.	[32, 32]	[32, 32]	[32, 32]
hidden layers critic	[32, 32]	[32, 32]	[32, 32]	[128,128]	n.a.	n.a.	n.a.	n.a.
hidden activation	tanh	tanh	tanh	relu	n.a.	tanh	tanh	tanh
initial std	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
number basis functions	n.a.	5	n.a.	n.a.	5	n.a.	5	5
number zero basis	n.a.	n.a.	n.a.	n.a.	1	n.a.	1	1
weight scale	n.a.	20	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.

Table B.2: Hyperparameters for the box pushing experiments.

	PPO	NDP	TRPL	SAC	ES	MP3	MP3-BB-PPO	MP3-BB
number samples	16000	16000	16000	1000	250	160	160	40
GAE λ	0.95	0.95	0.95	0.95	n.a.	n.a.	n.a.	n.a.
discount factor	1.0	0.99	1.0	0.99	n.a.	1.0	n.a.	n.a.
ε_μ	n.a.	n.a.	0.005	n.a.	n.a.	0.05	n.a.	0.05
ε_Σ	n.a.	n.a.	0.00005	n.a.	n.a.	0.0005	n.a.	0.0005
optimizer	adam	adam	adam	adam	adam	adam	adam	adam
epochs	10	10	20	1000	n.a.	20	100	20
learning rate	1e-4	1e-4	5e-5	1e-4	1e-2	3e-4	1e-4	3e-4
use critic	True	True	True	True	False	True	True	True
epochs critic	10	10	10	1000	n.a.	10	100	10
learning rate critic (and alpha)	1e-4	1e-4	2e-4	1e-4	n.a.	3e-4	1e-4	3e-4
number minibatches	40	32	40	n.a.	n.a.	1	1	1
batch size	n.a.	n.a.	n.a.	256	n.a.	n.a.	n.a.	n.a.
buffer size	n.a.	n.a.	n.a.	1e6	n.a.	n.a.	n.a.	n.a.
learning starts	0	0	0	10000	0	0	0	0
polyak_weight	n.a.	n.a.	n.a.	5e-3	n.a.	n.a.	n.a.	n.a.
trust region loss weight	n.a.	n.a.	10	n.a.	n.a.	10	n.a.	10
normalized observations	True	True	True	False	False	False	False	False
normalized rewards	True	True	False	False	False	False	False	False
observation clip	10.0	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	0.2	n.a.	n.a.	n.a.	n.a.	0.2	n.a.
importance ratio clip	0.2	0.2	n.a.	n.a.	n.a.	n.a.	0.2	n.a.
hidden layers	[256, 256]	[256, 256]	[256, 256]	[256, 256]	[256, 256]	[128, 128]	[128, 128]	[128, 128]
hidden layers critic	[256, 256]	[256, 256]	[256, 256]	[256, 256]	n.a.	[32, 32]	[32, 32]	[32, 32]
hidden activation	tanh	tanh	tanh	relu	tanh	relu	tanh	relu
initial std	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
MP type	n.a.	DMP	n.a.	n.a.	n.a.	ProDMP	ProMP	ProMP
number basis functions	n.a.	5	n.a.	n.a.	n.a.	4	5	5
number zero basis	n.a.	n.a.	n.a.	n.a.	n.a.	0	1	1
k	n.a.	5	n.a.	n.a.	n.a.	25	100	100
weight scale	n.a.	10	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.

Table B.3: Hyperparameters for the Meta-World experiments.

	PPO	NDP	TRPL	SAC	ES	MP3	MP3-BB-PPO	MP3-BB
number samples	16000	16000	16000	1000	200	64	16	16
GAE λ	0.95	0.95	0.95	0.95	n.a.	1	n.a.	n.a.
discount factor	0.99	0.99	0.99	0.99	n.a.	1	n.a.	n.a.
ε_μ	n.a.	n.a.	0.005	n.a.	n.a.	0.075	n.a.	0.005
ε_Σ	n.a.	n.a.	0.0005	n.a.	n.a.	0.0005	n.a.	0.0005
optimizer	adam	adam	adam	adam	adam	adam	adam	adam
epochs	10	10	20	1000	n.a.	10	100	100
learning rate	3e-4	3e-4	5e-5	3e-4	1e-2	5e-5	3e-4	3e-4
use critic	True	True	True	True	False	True	False	False
epochs critic	10	10	10	1000	n.a.	10	n.a.	n.a.
learning rate critic (and alpha)	3e-4	3e-4	3e-4	3e-4	n.a.	3e-4	n.a.	n.a.
number minibatches	32	32	64	n.a.	n.a.	32	1	1
batch size	n.a.	n.a.	n.a.	256	n.a.	n.a.	n.a.	n.a.
buffer size	n.a.	n.a.	n.a.	1e6	n.a.	n.a.	n.a.	n.a.
learning starts	0	0	0	10000	0	0	0	0
polyak_weight	n.a.	n.a.	n.a.	5e-3	n.a.	n.a.	n.a.	n.a.
trust region loss weight	n.a.	n.a.	10	n.a.	n.a.	10	n.a.	10
normalized observations	True	True	True	False	False	True	False	False
normalized rewards	True	True	False	False	False	False	False	False
observation clip	10.0	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	0.2	n.a.	n.a.	n.a.	n.a.	0.2	n.a.
importance ratio clip	0.2	0.2	n.a.	n.a.	n.a.	n.a.	0.2	n.a.
hidden layers	[128, 128]	[128, 128]	[128, 128]	[256, 256]	[128, 128]	[256, 256]	[32, 32]	[32, 32]
hidden layers critic	[128, 128]	[128, 128]	[128, 128]	[256, 256]	n.a.	[256, 256]	n.a.	n.a.
hidden activation	tanh	tanh	tanh	relu	tanh	tanh	tanh	relu
initial std	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
MP type	n.a.	DMP	n.a.	n.a.	n.a.	ProDMP	ProMP	ProMP
number basis functions	n.a.	5	n.a.	n.a.	n.a.	3	5	5
number zero basis	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	1	1
k	n.a.	n.a.	n.a.	n.a.	n.a.	100	n.a.	n.a.
weight scale	n.a.	100	n.a.	n.a.	n.a.	10	10	10

Table B.4: Hyperparameters for the Hopper Jump experiments.

	PPO	TRPL	SAC	CMORE	ES	MP3-BB-PPO	MP3-BB
number samples	16384	16384	1000	60	200	64	64
GAE λ	0.95	0.95	0.95	n.a.	n.a.	n.a.	n.a.
discount factor	0.99	0.99	0.99	n.a.	n.a.	n.a.	n.a.
$\varepsilon_\mu/\varepsilon$	n.a.	0.005	n.a.	0.1	n.a.	n.a.	0.005
ε_Σ	n.a.	0.00005	n.a.	n.a.	n.a.	n.a.	0.0005
optimizer	adam	adam	adam	n.a.	adam	adam	adam
epochs	10	20	1000	n.a.	n.a.	100	100
learning rate	3e-4	5e-5	1e-4	n.a.	0.01	1e-4	5e-5
use critic	True	True	True	False	False	False	False
epochs critic	10	10	1000	n.a.	n.a.	n.a.	n.a.
learning rate critic (and alpha)	3e-4	3e-4	1e-4	n.a.	n.a.	n.a.	n.a.
number minibatches	32	64	n.a.	n.a.	n.a.	1	1
batch size	n.a.	n.a.	256	n.a.	n.a.	n.a.	n.a.
buffer size	n.a.	n.a.	1e6	n.a.	n.a.	n.a.	n.a.
learning starts	0	0	10000	0	0	0	0
polyak_weight	n.a.	n.a.	5e-3	n.a.	n.a.	n.a.	n.a.
trust region loss weight	n.a.	10	n.a.	n.a.	n.a.	n.a.	25
normalized observations	True	True	False	False	False	False	False
normalized rewards	True	False	False	False	False	False	False
observation clip	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	n.a.	n.a.	n.a.	n.a.	0.2	n.a.
importance ratio clip	0.2	n.a.	n.a.	n.a.	n.a.	0.2	n.a.
hidden layers	[128, 128]	[128, 128]	[128, 128]	n.a	[128, 128]	[32, 32]	[32, 32]
hidden layers critic	[128, 128]	[128, 128]	[128, 128]	n.a	n.a	n.a	n.a
hidden activation	tanh	tanh	relu	n.a.	tanh	tanh	tanh
initial std	1.0	1.0	1.0	1.0	1.0	1.0	1.0
number basis functions	n.a.	n.a.	n.a.	5	n.a.	5	5
number zero basis	n.a.	n.a.	n.a.	1	n.a.	1	1

Table B.5: Hyperparameters for the Beer Pong experiments.

	PPO	CMORE	MP3-BB-PPO	MP3-BB
number samples	16384	60	160	160
GAE λ	0.95	n.a.	n.a.	n.a.
discount factor	0.99	n.a.	n.a.	n.a.
$\varepsilon_\mu/\varepsilon$	n.a.	0.1	n.a.	0.005
ε_Σ	n.a.	n.a.	n.a.	0.0005
optimizer	adam	n.a.	adam	adam
epochs	10	n.a.	100	100
learning rate	3e-4	n.a.	1e-4	5e-5
use critic	True	False	False	False
epochs critic	10	n.a.	n.a.	n.a.
learning rate critic (and alpha)	3e-4	n.a.	n.a.	n.a.
number minibatches	32	n.a.	1	1
trust region loss weight	n.a.	n.a.	n.a.	25
normalized observations	True	False	False	False
normalized rewards	True	False	False	False
observation clip	10.0	n.a.	n.a.	n.a.
reward clip	10.0	n.a.	n.a.	n.a.
critic clip	0.2	n.a.	0.2	n.a.
importance ratio clip	0.2	n.a.	0.2	n.a.
hidden layers	[128, 128]	n.a.	[32, 32]	[32, 32]
hidden layers critic	[128, 128]	n.a.	n.a.	n.a.
hidden activation	tanh	n.a.	tanh	tanh
initial std	1.0	1.0	1.0	1.0
number basis functions	n.a.	2	2	2
number zero basis	n.a.	2	2	2

Table B.6: Hyperparameters for the Table Tennis experiments.

	PPO	TRPL	MP3	MP3-BB-PPO	MP3-BB
number samples	16000	16000	360	200	200
GAE λ	0.95	0.95	n.a.	n.a.	n.a.
discount factor	0.99	0.99	1.0	n.a.	n.a.
ε_μ	n.a.	0.0005	0.005	n.a.	0.0005
ε_Σ	n.a.	0.00005	0.0005	n.a.	0.00005
optimizer	adam	adam	adam	adam	adam
epochs	10	20	20	100	100
learning rate	1e-4	5e-5	2e-4	1e-4	3e-4
use critic	True	True	True	True	True
epochs critic	10	10	10	100	100
learning rate critic (and alpha)	1e-4	1e-4	2e-4	1e-4	3e-4
number minibatches	40	40	1	1	1
trust region loss weight	n.a.	10.0	10	n.a.	25
normalized observations	True	True	False	False	False
normalized rewards	True	False	False	False	False
observation clip	10.0	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	n.a.	n.a.	0.2	n.a.
importance ratio clip	0.2	n.a.	n.a.	0.2	n.a.
hidden layers	[256, 256]	[256, 256]	[256]	[256]	[256]
hidden layers critic	[256, 256]	[256, 256]	[256]	[256]	[256]
hidden activation	tanh	tanh	relu	tanh	relu
initial std	1.0	1.0	1.0	1.0	1.0
MP type	n.a.	n.a.	ProDMP	ProMP	ProMP
number basis functions	n.a.	n.a.	3	3	3
number zero basis	n.a.	n.a.	0	1	1
k	n.a.	n.a.	50	n.a.	n.a.
weight scale	n.a.	n.a.	n.a.	n.a.	n.a.

Appendix C

Vlearn: Off-Policy Learning with Efficient State-Value Function Estimation

C.1 Derivations

We now show the proof of our new objective function. We start with a standard loss function for learning a V-value function for policy π as defined in Equation (5.2). Then we derive its upper bound shown in Equation (5.4).

C.1.1 Upper Bound Objective

$$
\begin{aligned}
L^*(\theta) &= \mathbb{E}_{s\sim d^\pi(s)}\left[\left(V^\pi_\theta(s) - \mathbb{E}_{a\sim\pi(\cdot|s)}\left(r(s,a)+\gamma V_{\bar\theta}(s')\right)\right)^2\right] \\
&= \mathbb{E}_{s\sim d^\pi(s)}\left[\left(V^\pi_\theta(s) - \mathbb{E}_{a\sim\pi_b(\cdot|s)}\left[\frac{\pi(a|s)}{\pi_b(a|s)}\left(r(s,a)+\gamma V_{\bar\theta}(s')\right)\right]\right)^2\right] \\
&= \mathbb{E}_{s\sim d^\pi(s)}\Bigg[\Bigg(\mathbb{E}_{a\sim\pi_b(\cdot|s)}\left[\frac{\pi(a|s)}{\pi_b(a|s)}V^\pi_\theta(s)\right] \\
&\qquad - \mathbb{E}_{a\sim\pi_b(\cdot|s)}\left[\frac{\pi(a|s)}{\pi_b(a|s)}\left(r(s,a)+\gamma V_{\bar\theta}(s')\right)\right]\Bigg)^2\Bigg] \\
&(V^\pi_\theta \text{ does not depend on a}) \\
&= \mathbb{E}_{s\sim d^\pi(s)}\left[\left(\mathbb{E}_{a\sim\pi_b(\cdot|s)}\left[\frac{\pi(a|s)}{\pi_b(a|s)}\left(V^\pi_\theta(s)-\left(r(s,a)+\gamma V_{\bar\theta}(s')\right)\right)\right]\right)^2\right] \\
&= \mathbb{E}_{s\sim d^\pi(s)}\left[\left(\int \underbrace{\pi_b(a|s)\frac{\pi(a|s)}{\pi_b(a|s)}}_{\text{weight terms t}}\underbrace{\left(V^\pi_\theta(s)-\left(r(s,a)+\gamma V_{\bar\theta}(s')\right)\right)}_{x}da\right)^2\right] \\
&= \mathbb{E}_{s\sim d^\pi(s)}\left[f\left(\int txda\right)\right],
\end{aligned}
$$

where we denote the convex function $f(x)=x^2$. While the weight terms are in $[0,1]$ and normalized, $\int tda = \int \pi_b(a|s)\frac{\pi(a|s)}{\pi_b(a|s)}da = \int \pi(a|s)da = 1$, the Jensen's inequality is applied as

$$\begin{aligned}
\mathbb{E}_{s\sim d^{\pi}(s)}\left[f\left(\int txda\right)\right] &\leq \mathbb{E}_{s\sim d^{\pi}(s)}\left[\int tf(x)\,da\right] \\
&= \mathbb{E}_{s\sim d^{\pi}(s)}\left[\mathbb{E}_{a\sim\pi_b(\cdot|s)}\left[\frac{\pi(a|s)}{\pi_b(a|s)}\left(V_\theta^\pi(s)-\left(r(s,a)+\gamma V_{\bar\theta}(s')\right)\right)^2\right]\right] \\
&\text{(Jensen's inequality)} \\
&= \mathbb{E}_{s\sim d^{\pi}(s),a\sim\pi_b(\cdot|s)}\left[\frac{\pi(a|s)}{\pi_b(a|s)}\left(V_\theta^\pi(s)-\left(r(s,a)+\gamma V_{\bar\theta}(s')\right)\right)^2\right] \\
&= L(\theta),
\end{aligned}$$

where $d^{\pi}(s)$ is the stationary distribution induced by policy π. Therefore, one can estimate $L(\theta)$ using Monte Carlo samples from the joint state-action distribution

$$L(\theta)=\sum_t\sum_j\frac{\pi(a_{t,j}|s_t)}{\pi_b(a_{t,j}|s_t)}\left(V_\theta^\pi(s_t)-\left(r_{t,j}+\gamma V_{\bar\theta}(s_{t+1,j})\right)\right)^2.$$

C.1.2 Consistency of Upper Bound Objective

We follow a similar derivation as in Neumann and Peters (2008) (for the case of state-action value function $Q(s,a)$) to prove the consistency between $L^*(\theta)$ and $L(\theta)$, *i.e.* the solution for minimizing $L(\theta)$ is the same for the original objective $L^*(\theta)$. For simplicity, we denote $\bar V = r(s,a)+\gamma V_{\bar\theta}(s')$.

$$\begin{aligned}
L^*(\theta) &= \mathbb{E}_{s\sim d^{\pi}(s)}\left[\left(V_\theta^\pi(s)-\mathbb{E}_{a\sim\pi(\cdot|s)}[\bar V]\right)^2\right] \\
&= \mathbb{E}_{s\sim d^{\pi}(s)}\left[V_\theta^\pi(s)^2-2V_\theta^\pi(s)\mathbb{E}_{a\sim\pi(\cdot|s)}[\bar V]+\mathbb{E}_{a\sim\pi(\cdot|s)}[\bar V]^2\right] \\
L(\theta) &= \mathbb{E}_{s\sim d^{\pi}(s)}\left[\mathbb{E}_{a\sim\pi_b(\cdot|s)}\left[\frac{\pi(a|s)}{\pi_b(a|s)}(V_\theta^\pi(s)-\bar V)^2\right]\right] \\
&= \mathbb{E}_{s\sim d^{\pi}(s)}\left[\mathbb{E}_{a\sim\pi_b(\cdot|s)}\left[\frac{\pi(a|s)}{\pi_b(a|s)}\left(V_\theta^\pi(s)^2-2V_\theta^\pi(s)\bar V+\bar V^2\right)\right]\right] \\
&= \mathbb{E}_{s\sim d^{\pi}(s)}\left[V_\theta^\pi(s)^2-2V_\theta^\pi(s)\mathbb{E}_{a\sim\pi_b(\cdot|s)}\left[\frac{\pi(a|s)}{\pi_b(a|s)}\bar V\right]+\mathbb{E}_{a\sim\pi_b(\cdot|s)}\left[\frac{\pi(a|s)}{\pi_b(a|s)}\bar V^2\right]\right] \\
&(V_\theta^\pi \text{ does not depend on a}) \\
&= \mathbb{E}_{s\sim d^{\pi}(s)}\left[V_\theta^\pi(s)^2-2V_\theta^\pi(s)\mathbb{E}_{a\sim\pi(\cdot|s)}[\bar V]+\mathbb{E}_{a\sim\pi(\cdot|s)}\left[\bar V^2\right]\right]
\end{aligned}$$

The above results show that $L^{*}(\theta)$ and $L(\theta)$ are identical except for a constant term added, which remains independent of V_{θ}^{π}.

C.2 Hyperparameters

Table C.1: Hyperparameters for the Gymnasium (Towers et al., 2023) experiments in Figure 5.2. The larger sample size for the the two on-policy methods is for the Humanoid-v4 experiments.

	PPO	TRPL	Vlearn/V-trace	SAC	MPO
number samples	2048/16384	2048/16384	1	1	1
GAE λ	0.95	0.95	n.a.	n.a.	n.a.
discount factor	0.99	0.99	0.99	0.99	0.99
ε_μ	n.a.	0.05	0.1	n.a.	1e-3
ε_Σ	n.a.	0.0005	0.0005	n.a.	2e-6
optimizer	adam	adam	adam	adam	adam
updates per epoch	10	20	1000	1000	1000
learning rate	3e-4	5e-5	5e-4	3e-4	3e-4
use critic	True	True	True	True	True
epochs critic	10	10	n.a.	n.a.	n.a.
learning rate critic (and alpha)	3e-4	3e-4	5e-4	3e-4	3e-4
learning rate dual	n.a.	n.a.	n.a.	n.a.	1e-2
number minibatches	32	64	n.a.	n.a.	n.a.
batch size	n.a.	n.a.	64	256	256
buffer size	n.a.	n.a.	5e5	1e6	1e6
learning starts	0	0	0	10000	10000
policy update interval	n.a.	n.a.	2	1	1
polyak_weight	n.a.	n.a.	5e-3	5e-3	5e-3
trust region loss weight	n.a.	10	10	n.a.	n.a.
num action samples	n.a.	n.a.	n.a.	1	20
normalized observations	True	True	True	False	False
normalized rewards	True	False	False	False	False
observation clip	10.0	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	n.a.	n.a.	n.a.	n.a.
importance ratio clip	0.2	n.a.	n.a.	n.a.	n.a.
hidden layers	[64, 64]	[64, 64]	[256, 256]	[256,256]	[256,256]
hidden layers critic	[64, 64]	[64, 64]	[256, 256]	[256,256]	[256,256]
hidden activation	tanh	tanh	relu	relu	relu
initial std	1.0	1.0	1.0	1.0	1.0

Table C.2: Hyperparameters for the dog tasks from DeepMind Control (Tunyasuvunakool et al., 2020) Figure 5.3.

	PPO	TRPL	Vlearn/V-trace	SAC	MPO
number samples	16384	16384	1	1	1
GAE λ	0.95	0.95	n.a.	n.a.	n.a.
discount factor	0.99	0.99	0.99	0.99	0.99
ε_μ	n.a.	0.05	0.1	n.a.	1e-3
ε_Σ	n.a.	0.0005	0.0005	n.a.	1e-6
optimizer	adam	adam	adam	adam	adam
updates per epoch	10	20	1000	1000	1000
learning rate	3e-4	5e-5	1e-4	3e-4	3e-4
use critic	True	True	True	True	True
epochs critic	10	10	n.a.	n.a.	n.a.
learning rate critic (and alpha)	3e-4	3e-4	1e-4	3e-4	3e-4
number minibatches	32	64	n.a.	n.a.	1e-2
batch size	n.a.	n.a.	64	256	n.a.
buffer size	n.a.	n.a.	5e5	1e6	256
learning starts	0	0	0	10000	1e6
policy update interval	n.a.	n.a.	2	1	10000
polyak_weight	n.a.	n.a.	5e-3	5e-3	1
trust region loss weight	n.a.	10	10	n.a.	5e-3
normalized observations	True	True	True	False	False
normalized rewards	True	False	False	False	False
observation clip	10.0	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	n.a.	n.a.	n.a.	n.a.
importance ratio clip	0.2	n.a.	n.a.	n.a.	n.a.
hidden layers	[64, 64]	[64, 64]	[512, 512]	[512,512]	[512,512]
hidden layers critic	[64, 64]	[64, 64]	[512, 512]	[512,512]	[512,512]
hidden activation	tanh	tanh	relu	relu	relu
initial std	1.0	1.0	1.0	1.0	1.0