Reinforcement Learning with Sparse and Multiple Rewards

Reinforcement Learning mit Mehreren und Seltenen Rewards
Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
genehmigte Dissertation von Simone Parisi aus Palermo
Januar 2020 — Darmstadt — D 17

TECHNISCHE UNIVERSITÄT DARMSTADT

Reinforcement Learning with Sparse and Multiple Rewards
Reinforcement Learning mit Mehreren und Seltenen Rewards

Genehmigte Dissertation von Simone Parisi aus Palermo

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Prof. Dr. Joschka Boedecker

Tag der Einreichung: 05.11.2019
Tag der Prüfung: 19.12.2019

Darmstadt — D 17

Erklärung zur Dissertation

Hiermit versichere ich, die vorliegende Dissertation ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.

Darmstadt, den January 6, 2020

(Simone Parisi)

Thesis Statement

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form. In the submitted thesis the written copies and the electronic version are identical in content.

Darmstadt, January 6, 2020

(Simone Parisi)

Abstract

Over the course of the last decade, the framework of reinforcement learning has developed into a promising tool for learning a large variety of task. The idea of reinforcement learning is, at its core, very simple yet effective. The learning agent is left to explore the world by performing *actions* based on its observations of the *state* of the world, and in turn receives a feedback, called *reward*, assessing the quality of its behavior. However, learning soon becomes challenging and even impractical as the complexity of the environment and of the task increase. In particular, learning without any pre-defined behavior (1) in the presence of rarely emitted or sparse rewards, (2) maintaining stability even with limited data, and (3) with possibly multiple conflicting objectives are some of the most prominent issues that the agent has to face. Consider the simple problem of a robot that needs to learn a parameterized controller in order to reach a certain point based solely on the raw sensory observation, e.g., internal reading of joints position and camera images of the surrounding environment, and on the binary reward "success" / "failure". Without any prior knowledge of the world's dynamics, or any hint on how to behave, the robot will start acting randomly. Such exploration strategy will be (1) very unlikely to bring the robot closer to the goal, and thus to experience the "success" feedback, and (2) likely generate useless trajectories and, subsequently, learning will be unstable. Furthermore, (3) there are many different ways the robot can reach the goal. For instance, the robot can quickly accelerate and then suddenly stop at the desired point, or it can slowly and smoothly navigate to the goal. These behaviors are clearly opposite, but the binary feedback does not provide any hint on which is more desirable.

It should be clear that even simple problems such as a reaching task can turn non-trivial for reinforcement learning. One possible solution is to pre-engineer the task, e.g., hand-crafting the initial exploration behavior with imitation learning, shaping the reward based on the distance from the goal, or adding auxiliary rewards based on speed and safety. Following this solution, in recent years a lot of effort has been directed towards scaling reinforcement learning to solve complex real-world problems, such as robotic tasks with many degrees of freedom, videogames, and board games like Chess, Go, and Shogi. These advances, however, were possible largely thanks to experts prior knowledge and engineering, such as pre-initialized parameterized agent behaviors and reward shaping, and often required a prohibitive amount of data. This large amount of required prior knowledge and pre-structuring is arguably in stark contrast to the goal of developing autonomous learning.

In this thesis we will present methods to increase the autonomy of reinforcement learning algorithms, i.e., learning without expert pre-engineering, by addressing the issues discussed above. The key points of our research address (1) techniques to deal with multiple conflicting reward functions, (2) methods to enhance exploration in the presence of sparse rewards, and (3) techniques to enable more stable and safer learning. Progress in each of these aspects will lift reinforcement learning to a higher level of autonomy.

First, we will address the presence of conflicting objective from a multi-objective optimization perspective. In this scenario, the standard concept of optimality is replaced by Pareto optimality, a concept for representing compromises among the objectives. Subsequently, the goal is to find the Pareto frontier, a set of solutions representing different compromises among the objectives. Despite recent advances in multi-objective optimization, achieving an accurate representation of the Pareto frontier is still an important challenge. Common practical approaches rely on experts to manually set priority or thresholds on the objectives. These methods require prior knowledge and are not able to learn the whole Pareto frontier but just a portion of it, possibly missing interesting solutions. On the contrary, we propose a manifold-based method which learn a continuous approximation of the frontier without the need of any prior knowledge.

We will then consider learning in the presence of sparse rewards and present novel exploration strategies. Classical exploration techniques in reinforcement learning mostly revolve around the immediate reward, that is, how to choose an action to balance between exploitation and exploration for the current state. These methods, however, perform extremely poorly if only sparse rewards are provided. State-of-the-art exploration strategies, thus, rely either on local exploration along the current solution together with sensible initialization, or on handcrafted strategies based on heuristics. These approaches, however, either require prior knowledge or have poor guarantees of convergence, and often falls in local optima. On the contrary, we propose an approach that plans exploration actions far into the future based on what we call *long-term visitation value*. Intuitively, this value assesses the number of unvisited states that the agent can visit in the future by performing that action.

Finally, we address the problem of stabilizing learning when little data is available. Even assuming efficient exploration strategies, dense rewards, and the presence of only one objective, reinforcement learning can exhibit unstable behavior. Interestingly, the most successful algorithms, namely actor-critic methods, are also the most sensible to this issue. These methods typically separate the problem of learning the value of a given state from the problem of learning the optimal action to execute in such a state. The former is fullfilled by the so-called *critic*, while the latter by the so-called *actor*. In this scenario, the instability is due the interplay between these two components, especially when nonlinear approximators, such as neural networks, are employed. To avoid such issues, we propose to regularize the learning objective of the actor by penalizing the error of the critic. This improves stability by avoiding large steps in the actor update whenever the critic is highly inaccurate.

Altogether, the individual contributions of this thesis allow reinforcement learning to rely less on expert pre-engineering. The proposed methods can be applied to a large variety of common algorithms, and are evaluated on a wide array of tasks. Results on both standard and novel benchmarks confirm their effectiveness.

Zusammenfassung

Im Laufe des letzten Jahrzehnts hat sich *Reinforcement Learning* zu einem vielversprechenden Instrument für das Erlernen einer Vielzahl von Aufgaben entwickelt. Die Idee des *Reinforcement Learning* ist in ihrer Essenz sehr einfach, jedoch zugleich effektiv. Der lernende Agenten erkundet die Welt durch seine Aktionen, welche auf Beobachtungen des Umgebungszustands basieren. Dabei erhält er Belohnungen, welche die Zielgerichtetheit des Verhaltens bewerten. Jedoch wird das Lernen mit zuzunehmender Komplexität der Umgebung beziehungsweise Aufgabe schnell anspruchsvoll oder sogar unmöglich. Insbesondere zählen das Lernen ohne vorgegebenes Verhalten, (1) das Fehlen von informativen Belohnungen, (2) das Erhalten der Stabilität trotz begrenzter Daten, und (3) sich möglicherweise widersprechenden Ziele zu den bekanntesten Problemen die der Agent lösen muss.

Betrachten wir die Aufgabe dass ein Roboter einen parametrisierten Regler lernen soll welcher es ihm ermöglicht nur mithilfe von rohen Sensormessungen, wie beispielsweise Messungen der Gelenkwinkel des Roboters oder Kamerabilder der Umgebung, sowie den binären Belohnungen “Erfolg” / “Misserfolg” einen Punkt im Raum zu erreichen. Ohne vorheriges Wissen über die Dynamik der Umgebung oder einen Hinweis wie er sich zu verhalten hat, wird der Roboter mit zufälligen Aktionen beginnen. Eine solche Erkundungsstrategie wird den Roboter aller Wahrscheinlichkeit nach nicht näher an das Ziel bringen beziehungsweise dazu führen dass er die Rückmeldung “Erfolg” erhält. Darüber hinaus hat die rein zufällige Exploration die Tendenz unsichere Trajektorien zu generieren, wodurch das Lernen instabil wird. Zudem gibt es viele verschiedene Wege auf denen der Roboter das Ziel erreichen kann. Zum Beispiel kann der Roboter schnell beschleunigen und dann plötzlich am Zielpunkt stoppen, oder es kann langsam und gleichmäßig zum Ziel fahren. Ein binäres Feedback allein ist nicht ausreichend um festzulegen welche diese offensichtlich gegensätzlichen Verhaltensweisen erwünscht ist.

Selbst einfache Aufgaben wie das erreichen eines Punkts im Raum können im Kontext des *Reinforcement Learning* zu einer Herausforderung werden. Eine mögliche Lösung ist das Problem vorzubearbeiten wie zum Beispiel vordefiniertes Explorationsverhalten mittels *Imitation Learning*, das Belohnungssignal proportional zu dem Abstand vom Ziel zu formulieren, oder eine zusätzliche Belohnung für Geschwindigkeit und Sicherheit einzuführen. Diesem Ansatz folgend wurde in den letzten Jahren ein großer Aufwand betrieben um *Reinforcement Learning* skalierbar zu machen, wodurch die Lösung von komplexen Problemen in der realen Welt ermöglicht wird. Beispiele hierfür sind Roboter-basierte Probleme mit vielen Freiheitsgraden oder Spiele wie Schach, Go und Shogi. Diese Fortschritte wurden jedoch weitgehend durch Vorwissen in Form von bereits initialisierten Verhalten oder einem bewusst hilfreich definiertem Belohnungssignal ermöglicht und benötigen zudem oft eine unerschwinglich große Menge an Daten. Das benötigte Vorwissen sowie das Vorstrukturieren der Aufgabe stellen stehen jedoch im starken Kontrast zu dem übergeordneten Ziel des selbständigen Lernens.

In dieser Arbeit werden wir Methoden präsentieren welche die Selbständigkeit von *Reinforcement Learning* Algorithmen verbessern indem wir die oben beschriebenen Probleme adressieren, das heißt Lernen ohne Vorbearbeitung durch Experten. Die zentralen Punkte unserer Forschung sind Methoden um (1) mit mehreren konkurrierenden Belohnungssignalen umzugehen, (2) die

Erkundung der Umgehung in Abwesenheit von informativen Belohnungssignalen zu verbessern, und (3) ein stabileres und sichereres Lernen zu ermöglichen. Ein Fortschritt in jeder der drei genannten Aspekte wird die Lernmethoden des *Reinforcement Learning* auf eine höhere Ebene der Selbständigkeit heben.

Als erstes werden wir uns dem Problem der konkurrierenden Ziele aus Sicht eines darauf ausgelegten Optimierungsverfahrens widmen. In diesem Szenario wird das Standardkonzept der Optimalität durch die sogenannte Pareto-Optimalität, ein Konzept welches Kompromisse zwischen verschiedenen Zielen darstellt, ersetzt. Es wird danach gestrebt die sogenannte Pareto-Grenze, eine Menge an Lösungen welche Kompromisse zwischen den einzelnen Zielen repräsentiert, zu finden. Trotz kürzlicher Fortschritte im Bereich der Mehrzieloptimierung, ist die genaue Erfassung der Pareto-Grenze immer noch eine Herausforderung. In der Praxis verlassen sich die meisten Ansätze darauf dass Experten die Priorität order Schwellenwerte für die Erfüllung der individuellen Ziele manuell festlegen. Diese Ansätze benötigen fachspezifisches Wissen und sind nur in der Lage einen Ausschnitt der Pareto-Grenze zu lernen, wodurch sie aller Voraussicht nach interessante Lösungen übersehen. Im Gegensatz dazu, stellen wir einen auf Mannigfaltigkeiten basierenden Ansatz vor, der eine kontinuierliche Approximation der Pareto-Grenze lernt ohne dabei Wissen vorauszusetzen.

Im Anschluss befassen wir uns mit neuen Explorationstrategien sowie dem Lernen mit seltenen Belohnungen. Typische Explorationstrategien im Bereich des *Reinforcement Learning* fokussieren sich auf die instantane Belohnung im gegenwärtigen Zeitschritt, was gleichbedeutend mit der Frage nach einer Balance zwischen Erkundung aus Ausschöpfung des bisherigen Kenntnisstands ist. Diese Methoden versagen jedoch im Fall von seltenen Belohnungen beziehungsweise einem uninformativen Belohnungssignal. Zum gegenwärtigen Stand der Forschung verlassen sich Explorationstrategien daher entweder auf eine lokale Erkundung entlang des derzeitigen Verhaltensmusters mit sinnvoller Initialisierung, oder auf problemspezifische Heuristiken. Diese Vorgehensweisen setzen jedoch entweder Wissen oder haben schlechte Konvergenzgarantien und finden oft nur lokale Optima. Dem entgegen, schlagen wir eine Methode vor welche die Explorationsverhalten basierend auf dem sogenannten *long-term visitation value* weit in die Zukunft plant. Der *long-term Visitation Value* schätzt die Zahl der noch nicht entdeckten Zustände ab welche der Agent durch ausführen der gegebenen Aktion besuchen kann.

Abschließend adressieren wir das Problem der Stabilisierung des Lernvorgangs für den Fall dass nur wenige Daten zur Verfügung stehen. Selbst unter der Annahme von effizienten Explorationstrategien, eines informativen Belohnungssignals sowie eines einzigen wohldefinierten Ziels können *Reinforcement Learning* Algorithmen zu instabilem Verhalten führen. Interessanterweise sind die erfolgreichsten Algorithmen, besonders jene mit sogenannter *Actor-Critic* Architektur, auch diejenigen welche am empfindlichsten für das genannte Problem sind. Diese Architektur separiert das Lernen des Werts eines Zustands vom Lernen der optimalen Aktion gegeben eines Zustands. Ersteres geschieht durch den *Critic*, wohingegen der *Actor* für die zweite Aufgabe zuständig ist. In dieser Zusammensetzung entsteht die Instabilität durch das Wechselspiel der beiden Komponenten, insbesondere wenn dabei nichtlineare Funktionsapproximatoren wie beispielsweise künstliche neuronale Netzwerke verwendet werden. Um solche Probleme zu vermeiden, frühen wir eine Regularisierung des Lernziels für den *Actor* ein welches den Schätzfehler des *Critic* bestraft. Dies führt zu einer Stabilisierung des Lernvorgangs durch die Vermeidung von großen Veränderungen des *Actor* wenn der *Critic* sehr ungenau ist.

Insgesamt erlauben die einzelnen Beiträge dieser Arbeit den *Reinforcement Learning* Algorithmen sich weniger auf Expertenwissen beziehungsweise die Vorbearbeitung des Problems zu verlassen. Die vorgestellten Methoden sind auf eine große Vielfalt von Algorithmen anwendbar und wurden auf einem weiten Spektrum an Problemen evaluiert. Ihre Effektivität wird durch die dargelegten Ergebnisse auf bekannten sowie neuartigen Leistungsvergleichen bestätigt.

Acknowledgments

I would like to thank my advisor Jan Peters for his advice and support throughout my PhD. He and my other co-authors – Hany, Alex, Matteo, Chris, Herke, Voot, Emti, and Joni – played a big role during my research and helped me developing the insights that lead to this thesis through discussions and feedback. Similarly, I would like to thank my colleagues Riad, Samuele, Boris, Marco, Carlo, Davide, Fabio and all other teammates for being part of a constructive and pleasant research environment with lots of discussions, suggestions, and constructive feedback. I would also like to extend my thanks to the staff at the Intelligent Autonomous Systems group, who provided essential support and helped pass bureaucratic hurdles. Thanks, Veronika, Sabine, Marion and Nanette!

I am grateful for Professor Joschka Boedecker, as expert on the topics in this thesis, to agree to be the external committee member. Your input is greatly appreciated. I would like to thank also the other members of my thesis committee, Professors Oskar von Stryk, Marc Fischlin, and Felix Wolf, without whose time investment the thesis defense would not have been possible. Their feedback, as well as the criticism and suggestions from the anonymous reviewers of my publications, have helped to improve the quality of this thesis.

My warmest thanks go to my family, for their unflinching support and continued encouragement. I thank my parents for teaching me the value of hard and honest work, and my sister for bringing happiness and making me smile everyday.

Finally, big thanks to all my friends who have been close to me during my PhD years. I would not have been able to write this thesis without your support.

Contents

Figures and Tables

List of Figures

List of Tables

Notation

The following tables give an overview of the notation and list most of the symbols used throughout the thesis. Symbols that only pertain to a specific section are defined where they are used.

Notation	**Description**
x	scalar
$\{x^{[i]}\}_{i=1\dots N}$	set of N elements x_1 through x_N
$\boldsymbol{x} = [x_1, x_2, \dots, x_N]$	vector of N elements
x_i	i-th element of the vector $\boldsymbol{x}$
$\boldsymbol{x}_{1:N}$	series of N vectors $\boldsymbol{x}_1$ through $\boldsymbol{x}_N$
$\boldsymbol{X}$	matrix
$\boldsymbol{X}\top$	transpose of a matrix
$p(x)$	probability density
$\mathbb{E}_{p(x)}[f(x)]$	expectation of $f(x)$ over $p(x)$
$\nabla_x f(x)$	partial derivative of $f(x)$ w.r.t. x

Symbols	**Description**
I	identity matrix
s	state
a	action
c	context
r	reward
$\mathcal{R}$	reward function
$\mathcal{P}$	transition function
γ	discount factor
π	policy
μ_π	steady-state distribution under policy π
J	expected return of policy π
ϱ	high-level policy
$\mathcal{J}$	expected return of high-level policy ϱ
θ	policy parameters
$\boldsymbol{\omega}$	high-level policy parameters
ω	critic parameters
ϕ	features
$\mathcal{F}$	Pareto frontier
$\mathcal{S}$	Pareto optimality indicator function

1 Introduction

Over the course of the last decade, the framework of reinforcement learning has developed into a promising tool for learning a large variety of task. A lot of effort has been directed towards scaling reinforcement learning to solve high-dimensional problems, such as robotic tasks with many degrees of freedom [Kober and Peters, 2012, Kormushev et al., 2013, Deisenroth et al., 2013, Parisi et al., 2015], videogames [Mnih et al., 2015, Vinyals et al., 2019], and board games likes Chess, Go and Shogi [Silver et al., 2017b]. Prominent algorithms include policy search [Peters and Schaal, 2008a, Kalakrishnan et al., 2012, Theodorou et al., 2010, Levine and Koltun, 2013], trajectory optimization [Kolter et al., 2008, Tassa et al., 2012] and actor-critic methods [Lillicrap et al., 2016, Schulman et al., 2015, 2016]. These advances, however, have been possible largely thanks to experts prior knowledge and engineering, such as pre-structured parameterized agent behavior policies and reward shaping. In particular, the design of task-specific exploration strategies and auxiliary rewards is frequently needed, especially if the task to be solved includes different conflicting objectives, or if the reward is sparse, i.e., it is rarely emitted. Furthermore, a prohibitive amount of data is often required, and without it the learning could be slower or more prone to unstable behavior. For instance, with little data it is often necessary to reduce the learning rate or to use larger batches, which may not always be possible due to limited computational resources. This large amount of required prior-knowledge, engineering, and pre-structuring, as well as data inefficiency and instability, are arguably in stark contrast to the goal of developing autonomous learning.

Motivated by the ambition of lifting the autonomy of artificial agents, we devoted our research to make reinforcement learning less dependent on human pre-engineering, by addressing the issues discussed above. The key points of our research address (1) techniques to deal with multiple conflicting reward functions, (2) methods to enhance exploration in the presence of sparse rewards, and (3) techniques to enable more stable and safer agent updates.

In the following sections of this chapter, we introduce these three topics which constitute the main body of the thesis. We start by motivating our work with a real-world robotic case study. The task, namely the robot tetherball game, highlights the benefits of using reinforcement learning rather than relying on human programming. At the same time, it underlines the need of classical reinforcement learning for human expertise in order to perform efficiently. Subsequently, we contrast the state-of-the-art to the desired properties of fully autonomous reinforcement learning in the context of multi-objective optimization, exploration strategies with sparse rewards, and algorithm stability. For each topic, we introduce the thesis contributions and the methods which will be then discussed deeply in Chapters 2, 3, and 4.

Altogether, the individual contributions of this thesis allow reinforcement learning to rely less on expert pre-engineering. The proposed methods can be applied to a large variety of common algorithms, and are evaluated on a wide array of tasks. Results on both standard and novel benchmarks confirm their effectiveness.

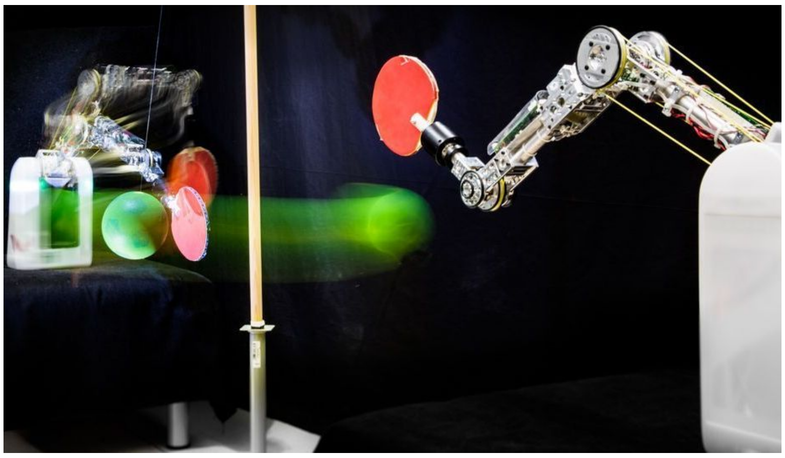


Figure 1.1: In robot tetherball, a pole with a ball hanging from the top is placed between two robotic arms. The goal of the robots is to repeatedly hit the ball in one direction without giving the opponent the chance to unwind it.

1.1 A Real-World Case Study: The Robot Tetherball Game

In the robot tetherball game, shown in Figure 1.1, two robots – the players – are set up facing each other and a pole is centered between them. A ball is attached to the top of the pole using a rope. The goal of the robots is to hit the ball such that the rope winds around the pole as often as possible in one direction, and without giving the opponent the chance to unwind it. In [Parisi et al., 2015], we used this platform to conduct a systematic evaluation of learned skills against manually programmed solutions, and to study the limitation of current reinforcement learning algorithms. More specifically, we aimed to answer the following questions. (1) Who would win between a player trained with reinforcement learning and one manually programmed? (2) How much engineering does reinforcement learning need to be efficiently applied to a real robotic task? (3) What are the limitations of current reinforcement learning algorithms?

To answer these questions, we manually programmed one player and trained the second independently. The former, i.e., the *hand-crafted player*, is aware of its dynamics and uses a mechanical model of tetherball to predict the ball trajectory. It computes an interception point and plans a stroke movement to hit the ball. The latter, i.e., *learned player*, instead, knows nothing about its dynamics and the environment, and is trained solely on sampled trajectories. We formulated the game as a contextual episodic Markov Decision Process [Deisenroth et al., 2013] and used Dynamic Motor Primitives [Ijspeert et al., 2002] to represent the player trajectories. This allowed us to employ reinforcement learning algorithms to train the trajectory generation of the second player. For a complete description of both players, we refer to Appendix A.

As evaluation, we let the real robots play full matches against each other and compare them according to the ball hit rate and the overall score. A match is started by the referee throwing the ball randomly to one of the robots and ends when none of the robots is able to hit the ball anymore. The score of a match is determined by how often the ball winds around the pole for each player, i.e., by the number of misses of the opponent. The two robots played a total of 25 games and results are reported in Table 1.1. The hand-crafted player won six of the 25 games, while the learned player won the remaining 19 games. Throughout the games, the hand-crafted player scored eight times, while the learned player scored 38 times. The better performance of the learned player is due to its ability to learn to compensate for errors arising in the model

Table 1.1: Tetherball match results for the real robots. The learned player defeats the hand-crafted one and achieves better overall hit rate, averaged over 25 matches.

Player	Hit rate	Matches won	Total score
Hand-crafted	71%	6/25	8
Learned	**85%**	**19/25**	**38**

of both the robot and the environment. The first error is due to the highly non-linear forward dynamics caused by springs in the joints. Furthermore, lacking a full inverse dynamics model, the hand-crafted player is also sensitive to errors in the tracking of the planned trajectory. The second error depends on the inaccurate tracking of the ball, for which we used a kinect.

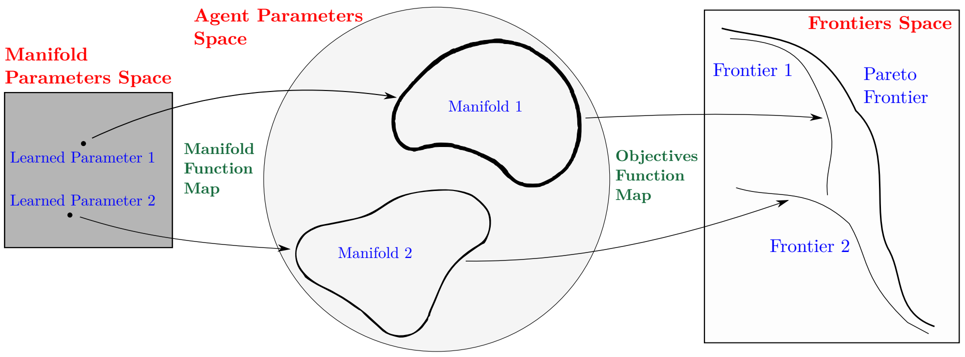
From a more practical perspective, mathematically describing tetherball requires a complex mechanical model [Abdulsamad et al., 2014]. Even after disregarding effects like friction, air drag and changes in string tension, the underlying equations are still highly nonlinear. Deriving the model, thus, is time consuming and requires a large amount of prior knowledge. The learned player, instead, did not need any prior knowledge of this kind, therefore making reinforcement learning more appealing for solving complex tasks.

However, the learned player is not truly autonomous, as it needed other kind of prior knowledge. First, its learning components required the tuning of open parameters. This process, despite being far less time intensive than building a mathematical model for a given task, still requires expertise. Second, bootstrapping the learning by demonstrating initial trajectories with imitation learning was essential to make the learning process feasible. Imitation learning, in fact, provided an initial exploration which helped avoiding dangerous trajectories and ensuring stable updates. Third, the reward function was designed with four components and shaping auxiliary rewards was of critical importance. The first component is the intuitive binary reward "success" / "failure" for hitting / missing the ball. The second is the minimum distance between the player paddle and the ball during a trajectory. This component is needed due to the sparsity of the first one. The third and the fourth are penalties for high-acceleration trajectories and for collisions with the base the robot is standing on. These components are crucial for safety, which can be considered as a separate conflicting objective.

This investigation shows that reinforcement learning is a powerful tool, allowing to solve real tasks even with no knowledge about the environment and its dynamics. However, it is far from being truly autonomous, as it still relies on expertise and pre-engingeering of its components. More specifically, we identified three main challenges, namely learning (1) with additional conflicting objectives, (2) with sparse rewards, and (3) preventing unstable behaviors. In the next sections, we discuss these three challenges from a more general perspective.

1.2 Manifold-Based Reinforcement Learning for Conflicting Objectives

Many real-world control applications, from economics to robotics, are characterized by the presence of multiple conflicting objectives. For example, in the robot tetherball game alongside hitting the ball we want to ensure safety by avoiding high-acceleration trajectories. This would teach the player low-speed smooth trajectories. However, at the same time we want to maximize the chances of winning, thus hitting the ball at high speed would make it harder for the opponent to hit back. In between these two opposite strategies, there are many compromise solutions. The



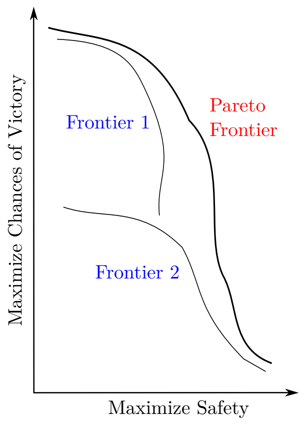


Figure 1.2: Generic scheme of manifold learning for multi-objective optimization. First, we define a parameterized function (top-left), e.g., a Gaussian distribution, to produce different agent manifolds (top-middle). Each agent is characterized by its parameter vector, e.g., the robot PD controller gains, and is evaluated through the objective functions. Then, the resulting objective vectors map into an approximate frontier (top-right). The goal is to learn the manifold parameters such that the Pareto frontier is generated. For this, we need to assess the quality of approximate frontiers. That is, considering for instance the two frontiers on the left, we need to answer the following question: is "frontier 1" better than "frontier 2" with respect to Pareto optimality?

set of these compromises is called *Pareto frontier* and to learn it is the aim of multi-objective optimization. Providing the Pareto frontier is, in fact, often beneficial. First, it encapsulates all the trade-offs among the objectives and gives better insight into the problem, thus helping the a posteriori selection of the most favorable solution. Second, the agent trained by manually tuning a priori the reward function may not meet our expected behavior. For instance, due to the different magnitude of the rewards, small changes in the the preference vector over the rewards may produce largely different solutions.

Despite recent advances in multi-objective optimization, achieving an accurate representation of the Pareto frontier is still an important challenge. Given the vector of rewards – one for each objective – classical approaches *scalarize* it, i.e., transform it into a scalar value, for instance by weighted sum. Subsequently, several optimization procedures are repeat on varying the scalarization, in the hope of finding diverse solutions. These approaches are not only highly inefficient, but depending on the scalarization used either lack guarantees of Pareto-optimality or cannot identify some portion of the Pareto frontier.

To address these limitations, we propose a manifold-based approach [Parisi et al., 2017a]. The idea, shown in Figure 1.2, is to optimize the parameters of a function defining a manifold in the agent parameters space, so that the corresponding image in the objectives space corresponds to the true Pareto frontier. Subsequently, the learning of the manifold is carried by episodic reinforcement learning, which enables importance sampling for improved sample efficiency. Beside the episodic formulation of multi-objective problems and the integration of importance sampling, the contributions of this work include the proposal of two novel indicator functions to asses the quality of approximate Pareto frontiers with respect to Pareto optimality, and the successful application to several bechmark problems, including the simulated robot tetherball and a many-objective problem (i.e., with more than three objectives).

1.3 Long-Term Visitation Value for Exploration with Sparse Rewards

The performance of reinforcement learning strongly depends on the quality of the reward signal. In particular, frequently emitted or “dense” rewards are desirable, as they provide feedback for any action performed by the agent. Instead, infrequent or “sparse” rewards yield little help. Since improving the agent relies on getting feedback via rewards, the agent cannot be improved until a reward is obtained. In situations where this occurs very rarely, the agent can barely learn. Therefore, many algorithms rely on well-shaped reward functions to guide the agent towards good solutions. For example, in the robot tetherall game, the simple binary reward “success” / “failure” for hitting / missing the ball was not sufficient to learn the task, and auxiliary hand-crafter rewards had to be designed, such as a penalty proportional to the quadratic distance between the paddle and the ball, and a penalty on collision. From the perspective of autonomous learning this reward engineering is unacceptable. First, it is easy to misspecify the reward function and cause unexpected behavior, e.g., excessively penalizing for collisions may prevent the robot from moving at all. Second, reward engineering requires expertise, and the resulting reward function may not transfer to different tasks or systems.

Current state-of-the-art exploration techniques for environment with sparse reward mostly use bootstrapping [Osband et al., 2019], or revolve around heuristics to supply an auxiliary “intrinsic” reward even when the environment does not provide one [Pathak et al., 2017, Houthooft et al., 2016, Bellemare et al., 2016]. However, these methods add the auxiliary reward *after* the action choice which makes exploration sample inefficient for long horizon decision making. Furthermore, they are sensitive to so-called “distractor” rewards. These are low-value easily-reachable rewards that attract the agent to poor local optima. Consider the environment depicted in Figure 1.3. The agent navigates in a grid by moving “left”, “right”, “up”, or “down”. A treasure of value 2 lies at the end of the blue corridor, while two treasures of value 1 serve as distractors next to the corridor entrance. The corridor is filled with puddles, each giving a penalty of -0.01. The exploration ends when the agent collects any of the treasures. In this environment, only three states give positive reward, and two of them are poor local optima corresponding to the distractors. Furthermore, the puddles discourage the agent from going through the corridor and, thus, from reaching the highest reward.

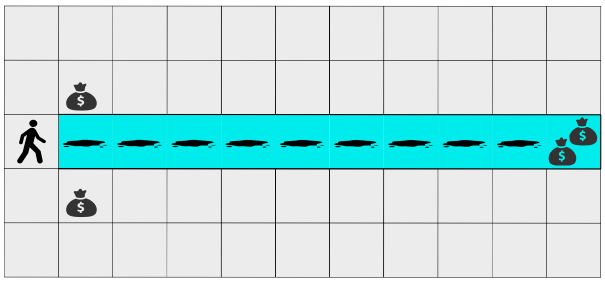


Figure 1.3: The deep gridworld environment.

In this thesis, we show that exploration strategies based on heuristic or bootstrapping perform poorly, not always finding the highest reward. In contrast to these approaches, we propose a novel principled method to perform long-term exploration, i.e., exploration that prefers actions allowing the agent to visit states in the future which have not been visited before. To this aim, we present the *long-term state-action visitation value*. Intuitively, this value expresses how much exploration we can expect on (discount weighted) average in future states if we choose a candidate action in the current state. Evaluated on a wide array of domains, our approach outpeforms state-of-the-art methods.

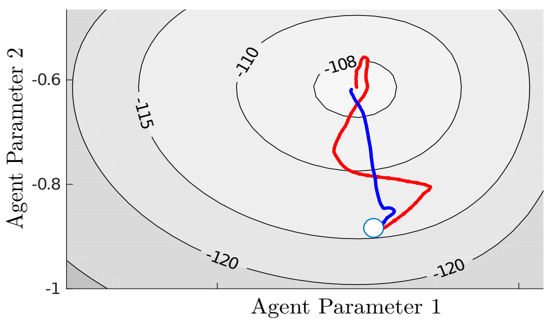


Figure 1.4: Example of unstable vs stable learning. Countour lines denote the magnitude of the reward against the actor parameters. The blue and red paths denote how these parameters were updated over time using reinforcement learning and just few samples per update. The white dot denotes the initial parameters.

Even assuming efficient exploration strategies, dense rewards, and the presence of only one objective, reinforcement learning can exhibit unstable behavior, especially when little data is available. Figure1.4 shows how the agent updates can be unstable even when the reward is quadratic. The red line denotes the update path followed by classical reinforcement learning algorithm *deterministic policy gradient*. Updates such as these are undesirable because take "detours" resulting in a loss of performance and, in the case of real systems, possibly in unsafe behaviors. Instead, update paths such as the the blue one are more appealing, as they straightforwardly learn the best agent parameters.

Interestingly, the most successful algorithms in reinforcement learning literature, namely actor-critic methods, are also the most sensible to instability and usually require expert hand-tuning. These methods separate the problem of learning the value of a given state from the problem of learning the optimal action to execute in such a state. The component in charge of learning the state value is called *critic*, while learning the optimal actions is fullfilled by the so-called *actor*. One of the reasons behind these methods instability is the interplay between the actor and critic during learning, e.g., a wrong step taken by one of them might adversely affect the other and can destabilize the learning [Dai et al., 2018]. Nonetheless, most of the existing methods have focused on stabilizing either the actor or the critic separately. For example, some recent works improve the stability of the critic by using a slowly-changing critic [Lillicrap et al., 2016, Mnih et al., 2015, Hessel et al., 2018], a low-variance critic [Munos et al., 2016, Gruslys et al., 2018], or two separate critics to reduce their bias [van Hasselt, 2010, Fujimoto et al., 2018]. Others have proposed to stabilize the actor instead, e.g., by constraining its update using entropy or the Kullback-Leibler divergence [Peters et al., 2010, Schulman et al., 2015, Akrour et al., 2016, Achiam et al., 2017, Nachum et al., 2018, Haarnoja et al., 2018].

In contrast to these approaches that focus on stabilizing either the actor or the critic, we focus on stabilizing the *interaction* between them. We propose to regularize the learning objective of the actor by penalizing the error of the critic. This improves stability by avoiding large steps in the actor update whenever the critic is highly inaccurate. The resulting method can be used as a plug-and-play method to improve stability of any actor-critic method together with other critic-stabilizing methods. For example, its application to deterministic policy gradient resulted in the blue path of Figure 1.4. Through evaluations on benchmark tasks, we show that our method is complementary to existing actor-critic methods, improving not only their stability but also their final performance and data efficiency.

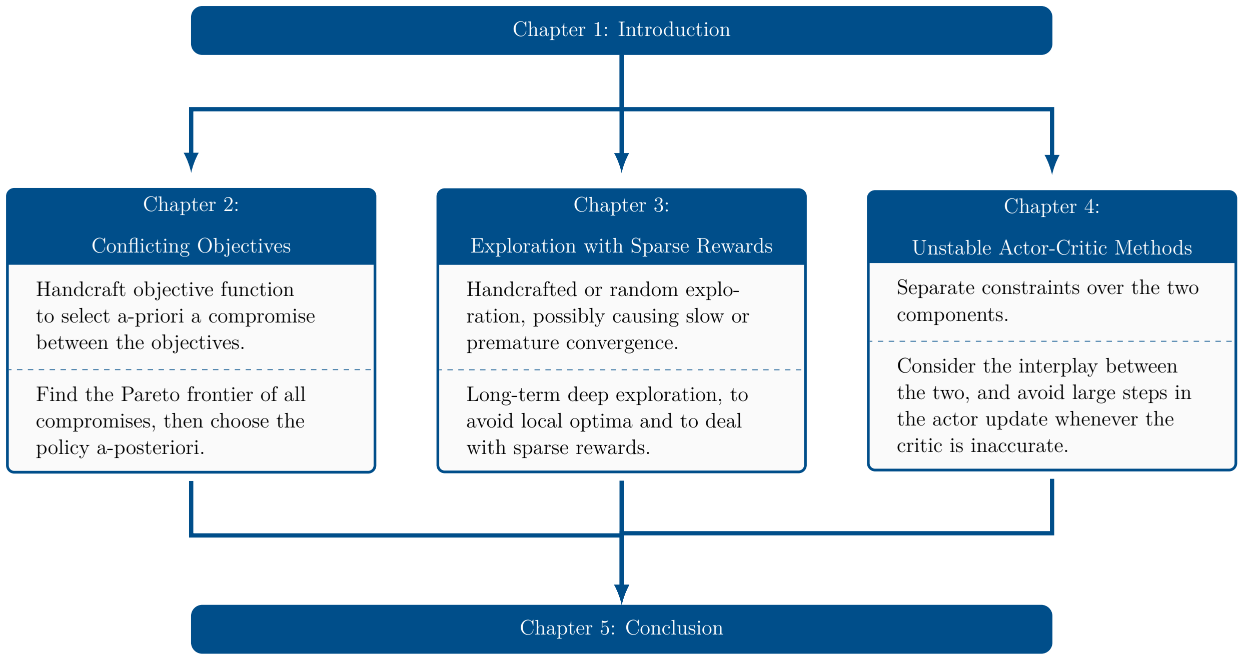

Chapter 1: Introduction
Chapter 2:
Conflicting Objectives
Handcraft objective function to select a-priori a compromise between the objectives.
Find the Pareto frontier of all compromises, then choose the policy a-posteriori.
Chapter 3:
Exploration with Sparse Rewards
Handcrafted or random exploration, possibly causing slow or premature convergence.
Long-term deep exploration, to avoid local optima and to deal with sparse rewards.
Chapter 4:
Unstable Actor-Critic Methods
Separate constraints over the two components.
Consider the interplay between the two, and avoid large steps in the actor update whenever the critic is inaccurate.
Chapter 5: Conclusion

Figure 1.5: Outline of the thesis. Chapters 2-4 can be read independently of the other chapters,and the notation needed for each of them is always introduced. In the outline, each box summarizes a challenge (title), common methods to address it (top half), and our proposed method (bottom half).

1.5 Contributions and Structure of the Thesis

In the next chapters, we addresses the topics discussed in Sections 1.2-1.4. The outline of the thesis is summarized in Figure 1.5, and the contributions in respect of each topic are theoretical, algorithmic, and empirical.

In Chapter 2 we consider learning in the presence of conflicting objectives, introducing the framework of multi-objective optimization and extending it to reinforcement learning.

- **Theoretical contribution:** Formalization of desirable properties for approximate frontiers, and of novel indicator fuctions to assess their quality.
- **Algorithmic contribution:** Manifold-based algorithm capable of learning an approximation of the Pareto frontier in a single run and to take advantage of importance sampling.
- **Empirical contribution:** Evaluation on three benchmark tasks against state-of-the-art, including the simulated robot tetherball and a many-objecive domain.

In Chapter 3 we then address the problem of learning in the presence of sparse rewards and distractor rewards.

- **Theoretical contribution:** Novel value functions for long-term exploration.
- **Algorithmic contribution:** Off-policy actor-critic algorithm capable of learning with sparse reward and in the presence of distractor rewards.
- **Empirical contribution:** Evaluation on several benchmark domains against state-of-the-art. Some domains are novel as well.

Next, **in Chapter 4** we discuss actor-critic methods and their instability. We introduce the components of these methods, identify the most prominent causes for unstable behaviors and present an approach to stabilize them.

- **Theoretical contribution:** Formalization of TD(λ)-regularization for actor-critic methods, showing relationship with existing reinforcement learning methods.
- **Algorithmic contribution:** Modification of common actor-critic methods using TD-regularization.
- **Empirical contribution:** Evaluation on several benchmark continuous tasks against state-of-the-art.

In the final chapter of this thesis, we summarize our findings and discuss future avenues of research. Some of these research ideas are directly linked to extending the methods presented in this thesis, while others have a more general scope.

2 Multi-objective Reinforcement Learning

Many real-world applications are characterized by multiple conflicting objectives. In such problems optimality is replaced by Pareto optimality and the goal is to find the Pareto frontier, a set of solutions representing different compromises among the objectives. Despite recent advances in multi-objective optimization, achieving an accurate representation of the Pareto frontier is still an important challenge. Building on recent advances in reinforcement learning (RL) and multi-objective policy search, we present two novel manifold-based algorithms to solve multi-objective Markov decision processes. These algorithms combine episodic exploration strategies and importance sampling to efficiently learn a manifold in the policy parameter space such that its image in the objective space accurately approximates the Pareto frontier. We show that episode-based approaches and importance sampling can lead to significantly better results in the context of multi-objective reinforcement learning (MORL). Evaluated on three multi-objective problems, our algorithms outperform state-of-the-art methods both in terms of quality of the learned Pareto frontier and sample efficiency.
Source code can be found at `https://github.com/sparisi/mips/tree/loopless/Algs/MORL`

2.1 Introduction

Many real-world problems are characterized by the presence of multiple conflicting objectives, such as economic systems [Shelton, 2001], medical treatment [Lizotte et al., 2012], control of robots [Nojima et al., 2003, Ahmadzadeh et al., 2014], water reservoirs [Castelletti et al., 2013] and elevators [Crites and Barto, 1998]. These applications can be modeled as MORL problems, where the standard notion of optimality is replaced by *Pareto optimality*, a concept for representing compromises among the objectives. Despite the increasing interest in multi-objective problems and recent advances in RL, MORL is still a relatively young field of research.

MORL approaches can be classified in two main categories [Vamplew et al., 2011] based on the number of policies they learn: single policy and multiple policy. While the majority of MORL approaches belong to the former category, in this chapter we focus on the latter and aim to learn a set of policies representing the best compromises among the objectives, namely the *Pareto frontier*. Providing an *accurate* and *uniform* representation of the complete Pareto frontier is often beneficial. It encapsulates all the trade-offs among the objectives and gives better insight into the problem, thus helping the a posteriori selection of the most favorable solution.

Following the same line of thoughts of RL, initially MORL researchers have focused on the development of value function-based approaches, where the attention was posed on the recovery of the optimal value function (for more details, we refer to the survey in [Roijers et al., 2013]). Recently[1], policy search approaches have also been extended to multi-objective problems [Parisi et al., 2014, Pirotta et al., 2015]. However, the majority of MORL approaches perform exploration in the action space [Sutton et al., 1999]. This strategy, commonly known as *step-based*, requires a different exploration noise at each time step and many studies [Deisenroth et al., 2013, Rückstiess et al., 2010] have shown that it is subject to several limitations, primarily due to the high

[1] The first seminal work dates back to 2001 [Shelton, 2001].

variance in the policy update. Furthermore, common algorithms involve the solution of several (independent) single-objective problems in order to approximate the Pareto frontier [Parisi et al., 2014, Gabor et al., 1998, Athan and Papalambros, 1996, Van Moffaert et al., 2012]. This approach implies an inefficient use of the samples, as each optimization is usually carried out on-policy, and most of MORL state-of-the-art approaches are inapplicable to large problems, especially in the presence of several objectives.

In this chapter, we address these limitations and present the first manifold-based episodic algorithms in MORL literature. First, these algorithms follow an *episodic* exploration strategy (also known as *parameter-based* or *black-box*) in order to reduce the variance during the policy update. Second, they perform a *manifold-based* policy search and directly learn a manifold in the policy parameter space to generate infinitely many Pareto-optimal solutions in a single run. By employing *Pareto-optimal* indicator functions, the algorithms are guaranteed to accurately and uniformly approximate the Pareto frontier. Finally, we show how to incorporate *importance sampling* in order to further reduce the sample complexity and to extend these algorithms to the *off-policy* paradigm. To the best of our knowledge, our algorithms are the first ones to tackle all these issues at once.

The remainder of the chapter is organized as follows. In Section 2.2, we introduce the multi-objective problem and discuss related work in MORL literature. Section 2.3 includes the main contributions of this chapter: an episodic manifold-based reformulation of the multi-objective problem, two policy search algorithms and two Pareto-optimal indicator functions to solve it, and an extension to importance sampling for reusing past samples. Section 2.4 provides a thorough empirical evaluation of the proposed algorithms on three problems, namely a water reservoir control task, a linear-quadratic regulator and a simulated robot tetherball game. Finally, in Section 2.5 we discuss the results of this study and propose possible avenues of investigation for future research.

2.2 Preliminaries

In this section, we provide the mathematical framework and the terminology as used in this chapter. Moreover, we present a categorization of the multi-objective approaches presented in MORL literature and we briefly discuss their advantages and drawbacks.

2.2.1 Problem Statement and Notation

Multi-objective Markov decision processes (MOMDPs) are an extension of MDPs in which several pairs of reward functions and discount factors are defined, one for each objective. Formally, a MOMDP is described by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}\left(s'|s,a\right), \boldsymbol{\mathcal{R}}\left(s,a\right), \boldsymbol{\gamma}, \mu \rangle$: $\mathcal{S} \subseteq \mathbb{R}^{d_\text{S}}$ is a continuous state space, $\mathcal{A} \subseteq \mathbb{R}^{d_\text{A}}$ is a continuous action space, $\mathcal{P}\left(s'|s,a\right)$ is a Markovian transition model and $\mathcal{P}\left(s'|s,a\right)\left(s'|s,a\right)$ defines the transition density between state s and s' under action a, $\boldsymbol{\mathcal{R}}\left(s,a\right) = \left[\mathcal{R}_1(s,a) \dots \mathcal{R}_{d_\text{R}}(s,a)\right]^\mathsf{T}$ and $\boldsymbol{\gamma} = \left[\gamma_1 \dots \gamma_{d_\text{R}}\right]^\mathsf{T}$ are vectors of reward functions $\mathcal{R}_i(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and discount factors $\gamma_i \in [0,1)$, respectively, and μ is the initial state distribution.

The *policy* followed by the agent is described by a conditional distribution $\pi(a|s)$ specifying the probability of taking action a in state s. In MOMDPs, a policy π is associated to d_R expected returns $\boldsymbol{J}^\pi = \left[J_1^\pi, \dots, J_{d_\text{R}}^\pi\right] \in \mathcal{F}$, where $\mathcal{F} \subseteq \mathbb{R}^{d_\text{R}}$ is the policy performance space. Using the trajectory-based definition, the i-th expected return is

$$J_i^\pi = \mathbb{E}_{\boldsymbol{\tau} \sim p(\cdot|\pi)}[R_i(\boldsymbol{\tau})] \,,$$

where $\boldsymbol{\tau} = \{s_t, a_t\}_{t=1}^{T_{\boldsymbol{\tau}}} \in \boldsymbol{T}$ is a trajectory (episode) of length $T_{\boldsymbol{\tau}}$ (possibly infinite) drawn from the distribution $p(\boldsymbol{\tau}|\pi)$, with return $R_i(\boldsymbol{\tau}) = \sum_{t=1}^{T_{\boldsymbol{\tau}}} \gamma_i^{t-1} \mathcal{R}_i(s_t, a_t)$. Since it is not common to have multiple discount factors (the problem becomes NP-complete [Feinberg, 2000]), we consider a unique value γ for all the objectives.

Unlike in single-objective MDPs, in MOMDPs a single policy dominating all others usually does not exist. When conflicting objectives are considered, no policy can simultaneously maximize all of them. For this reason, in multi-objective optimization a different dominance concept based on Pareto optimality is used. A policy π *strongly dominates* a policy π' (denoted by $\pi \succ \pi'$) if it outperforms π' on all objectives, i.e.,

$$\pi \succ \pi' \iff \forall i \in \{1 \dots d_\text{R}\}, J_i^\pi > J_i^{\pi'} .$$

Similarly, policy π *weakly dominates* policy π' (which is denoted by $\pi \succeq \pi'$) if it is not worse on all objectives, i.e.,

$$\forall i \in \{1, \dots, d_\text{R}\}, J_i^\pi \geq J_i^{\pi'} \ \wedge \ \exists i \in \{1, \dots, d_\text{R}\}, J_i^\pi = J_i^{\pi'} .$$

If there is no policy π' such that $\pi' \succ \pi$, then the policy π is *Pareto-optimal.* We can also speak of *locally Pareto-optimal* policies, for which the definition is the same as above, except that we restrict the dominance to a neighborhood of π.

Our goal is to determine the set of all Pareto-optimal policies $\Pi^* = \left\{\pi \mid \nexists \pi',\, \pi' \succ \pi\right\}$, which maps to the so-called *Pareto frontier* $\mathcal{F} = \left\{\boldsymbol{J}^{\pi^*} \mid \pi^* \in \Pi^*\right\}$[2]. More specifically, in this chapter we consider *parametric* policies $\pi \in \Theta^\theta \equiv \{\pi_\theta \mid \theta \in \theta \subseteq \mathbb{R}^{d_\theta}\}$, where θ is the *policy parameters space*. For simplicity, we use θ in place of π_θ to denote the dependence on the current policy, e.g., $\boldsymbol{J}(\theta)$ instead of $\boldsymbol{J}^{\pi_\theta}$.

2.2.2 Related Work

MORL approaches can be divided into two categories based on the number of policies they learn [Vamplew et al., 2011]. *Single-policy* methods aim to find the best policy satisfying a preference among the objectives. The majority of MORL approaches belong to this category and differ for the way in which preferences are expressed. They are easy to implement, but require a priori decision about the type of the solution and suffer from instability, as small changes on the preferences may result in significant variation in the solution [Vamplew et al., 2011]. The most straightforward and common single-policy approach is the scalarization where a function is applied to the reward vector in order to produce a scalar signal. Usually, a linear combination (weighted sum) of the rewards is performed and the weights are used to express the preferences over multiple objective [Castelletti et al., 2002, Natarajan and Tadepalli, 2005,

[2] As done in [Harada et al., 2006], we suppose that locally Pareto-optimal solutions that are not Pareto-optimal do not exist.

Van Moffaert et al., 2012]. Less common is the use of non linear mapping [Tesauro et al., 2007]. Although scalarization approaches are simple and intuitive, they may fail in obtaining MOO desiderata, e.g., a uniform distribution of the weights may not produce accurate and evenly distributed points on the Pareto frontier [Das and Dennis, 1997]. On the other hand, several issues of the scalarization are alleviated in RL due to the fact that the Pareto frontier is convex when stochastic policies are considered [Vamplew et al., 2009, Roijers et al., 2013]. For example, the convex hull of stochastic policies, each one being optimal w.r.t. a different linear scalarization, represents a viable approximation of the Pareto frontier[3]. Different single-policy approaches are based on thresholds and lexicographic ordering [Gabor et al., 1998] or different kinds of preferences over the objective space [Mannor and Shimkin, 2001, 2004].

Multiple-policy approaches, on the contrary, aim at learning multiple policies in order to approximate the Pareto frontier. Building the exact frontier is generally impractical in real-world problems, thus, the goal is to build an approximation of the frontier containing solutions that are accurate, evenly distributed and have a range similar to the true frontier [Zitzler et al., 2003]. Whenever possible, multiple-policy methods are preferred, as they permit a posteriori selection of the solution and encapsulate all the trade-offs among the multiple objectives. In addition, a graphical representation of the frontier can give better insights into the relationships among the objectives that can be useful for understanding the problem and the choice of the solution. However, all these benefits come at a higher computational cost, that can prevent learning in online scenarios. The most common approach to approximate the Pareto frontier is to perform multiple runs of a single-policy algorithm by varying the preferences among the objectives [Castelletti et al., 2002, Van Moffaert et al., 2012]. It is a simple approach but suffers from the disadvantages of the single-policy method used. Besides this, few other examples of multiple-policy algorithms can be found in literature. Barrett and Narayanan [Barrett and Narayanan, 2008] proposed an algorithm that learns all the deterministic policies that define the convex hull of the Pareto frontier in a single learning process. Recent studies have focused on the extension of fitted Q-iteration to the multi-objective scenario. While Lizotte et al. [Lizotte et al., 2010, 2012] have focused on a linear approximation of the value function, Castelletti et al. [Castelletti et al., 2012] proposed an algorithm to learn the control policy for all the linear combination of preferences among the objectives in a single run. Finally, Wang and Sebag [Wang and Sebag, 2013] proposed a Monte-Carlo Tree Search algorithm able to learn solutions lying in the concave region of the frontier.

Nevertheless, classic approaches discussed above exploit only deterministic policies resulting in scattered Pareto frontiers, while stochastic policies give a continuous range of compromises among objectives [Roijers et al., 2013, Parisi et al., 2014]. Shelton [Shelton, 2001, Section 4.2.1] was the pioneer both for the use of stochastic mixture policies and policy search in MORL, proposing a gradient-based algorithm to learn mixtures of Pareto-optimal policies. To the best of our knowledge, only the studies in [Parisi et al., 2014, Pirotta et al., 2015] followed the work of Shelton in combining policy search and multiple policy concepts. The former presented two MORL algorithms, called *Radial* (RA) and *Pareto-Following* (PFA) that, starting from an initial policy, perform gradient-based policy search procedures aimed to find a set of non-dominated policies. These algorithms, however, rely on several optimization procedures and are therefore sample inefficient. Differently, the algorithm provided by [Pirotta et al., 2015] learns a function defining a manifold in the policy parameters space. At each step the function is

[3] In episodic tasks, we can even exploit deterministic optimal policies by constructing mixture policies, i.e., policies stochastically choosing between deterministic policies at the beginning of each episode.

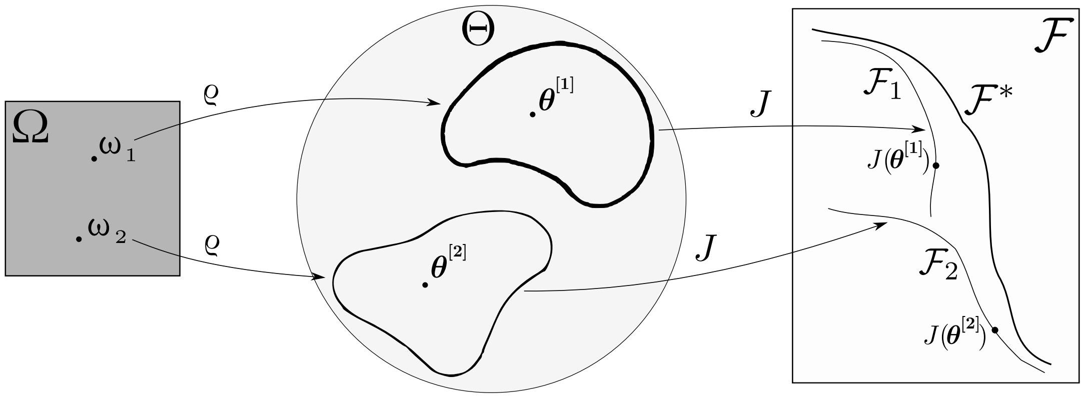


Figure 2.1: Transformation map from the high-level distribution ϱ to approximate frontiers in the objective space $\mathcal{F}$. A high-level parameter vector ω_i maps to a manifold in the policy parameter space θ. Subsequently, the manifold maps to an approximate frontier $\mathcal{F}_i$, with each vector $\theta^{[j]}$ mapping to a return vector $\boldsymbol{J}(\theta^{[j]})$.

optimized performing a single gradient ascent w.r.t. a indicator function that assesses the Pareto optimality of the manifold. Although interesting and promising, this approach is subject to many limitations. First, by following a gradient-based optimization, the indicator function must be differentiable. Therefore, common indicator functions in MORL, e.g., the hypervolume [Vamplew et al., 2011], cannot be employed. Second, the definition of the manifold parametrization is a non-trivial task and might require deep knowledge about the MOMDP.

Furthermore, all these approaches perform exploration in the action space. As already discussed, this strategy is subject to several limitations [Deisenroth et al., 2013, Rückstiess et al., 2010]. First, it causes a large variance in the parameter update estimate due to the per-step randomization. Second, in many real world applications (e.g., robotics) random exploration in every time step might be dangerous and can lead to uncontrolled or undesired behaviors of the agent.

2.3 Manifold-based Episodic Policy Search

In order to overcome limitations of state-of-the-art approaches, we present two novel algorithms that combine manifold-based policy search approach with episodic exploration strategies. By employing an episodic approach, our algorithms are more effective and efficient than step-based ones, since they reduce the variance during the policy update [Zhao et al., 2012]. Furthermore, by following a manifold-based approach, they are able to efficiently generate infinitely many Pareto-optimal solutions in a single run, without the need of several optimization procedures. Note that the most related approach in MOO is an evolutionary algorithm defined in [Igel et al., 2007].

2.3.1 The Episodic Multi-objective Reinforcement Learning Problem

In single-objective *episodic RL*, exploration is performed directly in the parameter space. The policy parameters θ are sampled at the beginning of each episode from a (high-level) *parametric* distribution $\varrho : \boldsymbol{\omega} \to \theta$. Instead of directly finding the policy parameters maximizing $J(\theta)$, methods following this approach aim to solve the problem defined by

$$\max_{\omega \in \Omega} \mathcal{J}(\omega) \equiv \max_{\omega \in \Omega} \int_\theta \varrho(\theta|\omega) J(\theta) \mathrm{d}\theta. \tag{2.1}$$

The above problem can be solved using several techniques, including gradient-based [Wierstra et al., 2014] or distribution-matching methods [Deisenroth et al., 2013]. However, these techniques cannot be directly applied to MORL problems since the function $\boldsymbol{J}(\theta)$ is a vector. As a consequence, the max operator in Equation (2.1) is no longer well defined. Nonetheless, we can extend the definition of Pareto optimality by applying a *indicator function* $\mathcal{S} : \mathcal{F} \to \mathbb{R}$ (also called metric or indicator in MORL literature) that assesses the Pareto optimality of a return vector $\boldsymbol{J}(\theta)$. Assuming that the manifold in the policy parameter space mapping to the Pareto frontier $\mathcal{F}^*$ can be approximated by the parametric distribution $\varrho(\theta|\omega)$, the episode-based problem we aim to solve is given by

$$\max_{\omega \in \Omega} \mathcal{J}_{\mathcal{S}}(\omega) \equiv \max_{\omega \in \Omega} \int_\theta \varrho(\theta|\omega) \mathcal{S}\left(\boldsymbol{J}(\theta)\right) \mathrm{d}\theta. \tag{2.2}$$

Figure 2.1 shows a graphical representation of the mapping between high-level distribution and the policy performance space. The above problem can be solved by any episodic RL approach and there is no constraint on the indicator function $\mathcal{S}$ since it does not depend on the optimization variable ω.

2.3.2 Learning the Manifold by Policy Search

The algorithms we present to solve Problem (2.2), namely *Multi-Objective eREPS* (MO-eREPS) and *Multi-Objective Natural Evolution Strategy* (MO-NES), are novel adaptations of state-of-the-art policy search algorithms. The algorithms have been chosen according to their recent successes in RL literature and their sample-efficiency.

MO-eREPS is an extension of Relative Entropy Policy Search [Peters et al., 2010], recently successfully applied on complex real-world tasks [Parisi et al., 2015]. The algorithm aims to solve Problem (2.2) while keeping a sufficient level of statistical information w.r.t. a reference distribution $\bar{\varrho}$. The level of statistical information is measured using the Kullback-Leibler (KL) divergence and the resulting constraint can be formalized as

$$\begin{aligned} \max_{\omega \in \Omega} \quad & \int_\theta \varrho(\theta|\omega) \mathcal{S}\left(\boldsymbol{J}(\theta)\right) \mathrm{d}\theta, \\ \text{s.t.} \quad & \mathrm{KL}(\varrho(\cdot|\omega)|\bar{\varrho}) \leq \epsilon, \quad \epsilon > 0. \end{aligned}$$

This constrained optimization problem can be solved in closed form by the method of Lagrangian multipliers and the solution is given by

$$\varrho \propto \bar{\varrho} \exp\left(\frac{\mathcal{S}\left(\boldsymbol{J}(\theta)\right)}{\eta}\right),$$

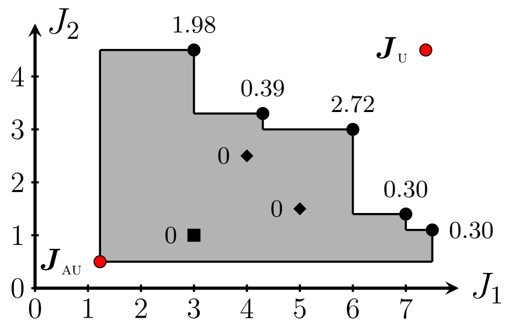


Figure 2.2: Example of hypervolume and reference points for a 2-objective problem. The utopia point $\boldsymbol{J}_{\text{U}}$ optimizes both objectives at the same time. The antiutopia $\boldsymbol{J}_{\text{AU}}$ represents an arbitrary low quality solution and it is used as reference for computing the hypervolume (gray area). The contribution of each solution to the hypervolume growth is denoted by node labels (the higher, the better).

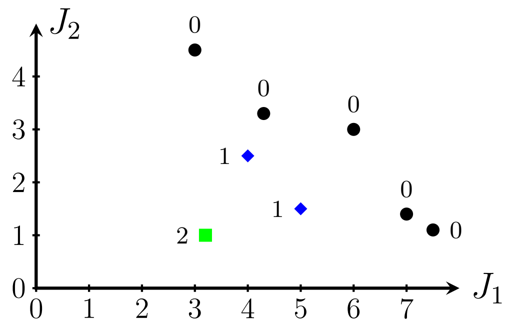


Figure 2.3: Example of non-dominance sorting. Node labels denote the ranking (the lower, the better). Non-dominated solutions (black circles) get the highest ranking of zero. After removing them, a new sub-frontier is identified (blue diamonds) and its solutions are given a dominance count of one. Finally, the last solution (green square) gets the lowest rank of two.

where η is the Lagrangian multiplier. The distribution $\varrho(\theta|\omega)$ is subsequently obtained by a weighted maximum likelihood estimate of samples $\theta^{[i]}$ and weights $\delta^{[i]} = \exp(\mathcal{S}(\boldsymbol{J}(\theta^{[i]}))/\eta)$. The resulting algorithm implements an iterative schema where at each iteration the optimization is solved w.r.t. the distribution recovered at the previous iteration (i.e., $\bar{\varrho}_k = \varrho_{k-1}(\cdot|\omega)$).

MO-NES is the multi-objective counterpart of Natural Evolution Strategy [Wierstra et al., 2014]. It exploits natural gradient ascent for solving Problem (2.2), i.e., it updates the high-level distribution by

$$\omega_{k+1} = \omega_k + \alpha\widetilde{\nabla}_\omega\mathcal{J}(\omega)|_{\omega=\omega_k},$$

where $\widetilde{\nabla}_\omega\mathcal{J}(\omega) = \boldsymbol{F}_\omega^{-1}\nabla_\omega\mathcal{J}(\omega)$ is the natural gradient and $\boldsymbol{F}_\omega$ is the Fisher Information Matrix (FIM) [Amari and Douglas, 1998]

$$\begin{aligned}\nabla_\omega\mathcal{J}(\omega) &= \int_\theta \varrho(\theta|\omega)\nabla_\omega \ln\varrho(\theta|\omega)\mathcal{S}\left(\boldsymbol{J}(\theta)\right)\mathrm{d}\theta,\\ \boldsymbol{F}_\omega &= \int_\theta \varrho(\theta|\omega)\nabla_\omega \ln\varrho(\theta|\omega)\nabla_\omega \ln\varrho(\theta|\omega)^{\mathsf{T}}\mathrm{d}\theta.\end{aligned}$$

Both terms can be approximated using samples $\{\theta^{[i]}, \mathcal{S}(\boldsymbol{J}(\theta^{[i]}))\}_{i=1\dots N}$. However, for some classes of distributions ϱ, the FIM can be computed exactly [Sun et al., 2009], making the algorithm particularly sample efficient.

It is worth noting that the most similar approach to MO-NES defined in MOO literature exploits covariance matrix adaptation evolution strategy (CMA-ES) to perform the optimization [Igel et al., 2007]. However, natural gradient has proved to be more effective and efficient in many real-world problems, overcoming the issues of CMA-ES approach. For a complete comparison of these techniques we refer the reader to [Wierstra et al., 2014].

2.3.3 Indicator Functions for Pareto Optimality

The choice of the indicator function is crucial as it has to encourage the learning of a distribution that generates policies around the true manifold rather than on a local region of it. We call this property *consistency*. Intuitively, a consistent indicator function must be maximized by the true Pareto frontier and must induce a partial ordering over the frontiers, i.e., if the solutions of a manifold $\mathcal{F}_2$ are all dominated by the ones of a manifold $\mathcal{F}_1$, then the indicator function score associate to $\mathcal{F}_1$ must be better than $\mathcal{F}_2$ one. Formally, let $\mathcal{F}$ be the set of all $(d_{\text{R}}-1)$-dimensional manifolds associated to a MOMDP with d_{R} objectives, $\theta_k \in \theta$ be the manifold in the policy parameters space mapping to $\mathcal{F}_k \in \mathcal{F}$ and $\omega_k \in \Omega$ be the high level distribution parameters mapping to θ_k. Let $\mathcal{F}^*$ be the real Pareto frontier and $\mathcal{J}_{\mathcal{S}}(\omega)$ be the manifold performance measure defined in Problem 2.2. An indicator function $\mathcal{S}$ is *consistent* if

$$\forall \omega_k \neq \omega_h, \ \ \mathcal{J}_{\mathcal{S}}(\omega_h) > \mathcal{J}_{\mathcal{S}}(\omega_k) \iff \mathcal{F}_h \equiv \mathcal{F}^* \qquad \text{and} \qquad \text{(a)}$$

$$\forall \theta_h, \theta_k, \ \ \forall \theta_i \in \theta_k, \ \ \exists \theta_j \in \theta_h, \ \ \pi_{\theta_j} \succeq \pi_{\theta_i} \implies \mathcal{J}_{\mathcal{S}}(\omega_h) > \mathcal{J}_{\mathcal{S}}(\omega_k). \qquad \text{(b)}$$

Several Pareto optimality indicator functions have been provided in literature, especially in the field of genetic algorithms. Here, we present and discuss the consistency of two indicator functions built on hypervolume [Beume et al., 2007] and non-dominance [Deb et al., 2000]. The former, denoted by $\mathcal{S}_{\text{HV}}(\boldsymbol{J}(\theta))$, ranks a solution according to its contribution to the *hypervolume* of the approximate frontier (the higher, the better). As shown in Figure 2.2, the hypervolume HV of a frontier is defined as the volume of the portion of the objective space dominated by the frontier w.r.t. a reference point [Vamplew et al., 2011]. Formally, it can be defined as the Lebesgue measure (i.e., the volume) of the union of the hypercuboids in the objective space [Coello et al., 2007]

$$\text{HV}_{\text{R}}(\mathcal{D}) = \text{LEBESGUE}\left(\bigcup_{\boldsymbol{J}(\theta) \in \text{NONDOM}(\mathcal{D})} \left\{ \boldsymbol{J}(\theta^{[i]}) \,\middle|\, \boldsymbol{J}(\theta) \prec \boldsymbol{J}(\theta^{[i]}) \prec \text{R}, \ i = 1 \ldots N \right\} \right),$$

where $\mathcal{D}$ is a dataset of points $\mathcal{D} = \{\boldsymbol{J}(\theta^{[i]})\}_{i=1\ldots N}$, R is a reference point and $\text{NONDOM}(\mathcal{D})$ is the set of all non-dominated points in $\mathcal{D}$. The hypervolume-based indicator of a sample $\boldsymbol{J}(\theta^{[j]}) \in \mathcal{D}$ is defined by

$$\mathcal{S}_{\text{HV}}\left(\boldsymbol{J}(\theta^{[j]})\right) = \text{HV}_{\text{R}}(\mathcal{D}) - \text{HV}_{\text{R}}\left(\mathcal{D} \setminus \boldsymbol{J}(\theta^{[j]})\right) - \Upsilon\left(\boldsymbol{J}(\theta^{[j]})\right),$$

where $\Upsilon(\boldsymbol{J}(\theta^{[j]})) \geq 0$ is a penalization that is positive when $\boldsymbol{J}(\theta^{[j]})$ is dominated in $\mathcal{D}$ and zero otherwise. This penalization is necessary in order to obtain a consistent indicator function. It has been shown that the Pareto frontier achieves the highest hypervolume and that adding a non-dominated point to a set of points results in the growth of the hypervolume of such a set [Zitzler et al., 2007]. However, as the hypervolume contribution of dominated solution is zero, it is possible to add infinite dominated solutions and still achieve the same performance $J_{\mathcal{S}}$. This behavior might bias the learning in favor of broad distributions ϱ that generate as many solutions as possible — including the true Pareto frontier — without truly converging to one that generates *only* Pareto-optimal solutions. By penalizing dominated solutions, this issue is solved.

The second indicator function ranks a solution according to its non-dominance count $\textsc{nd}$ (the lower, the better). The non-dominance count is computed iteratively as follows. First, the sub-frontier $\widetilde{\mathcal{F}}_0 = \textsc{nondom}(\mathcal{D})$ is identified. Solutions belonging to it are given a dominance count of zero and are filtered out from $\mathcal{D}$. Subsequently, the new sub-frontier is identified, i.e., $\widetilde{\mathcal{F}}_1 = \textsc{nondom}(\mathcal{D} \setminus \widetilde{\mathcal{F}}_0)$, and its solutions are given a dominance count of one. The procedure ends when all sub-frontiers are identified and all solutions are assigned a dominance count.

$$\begin{aligned}
\widetilde{\mathcal{F}}_0 &= \textsc{nondom}\left(\mathcal{D}\right), \\
\widetilde{\mathcal{F}}_i &= \textsc{nondom}\left(\mathcal{D} \setminus \bigcup_{j=1}^{i-1} \widetilde{\mathcal{F}}_j\right), \\
\textsc{nd}\left(\boldsymbol{J}(\theta^{[i]})\right) &= j, \qquad \boldsymbol{J}(\theta^{[i]}) \in \widetilde{\mathcal{F}}_j.
\end{aligned}$$

An example is shown in Figure 2.3. Solutions with the same dominance count are additionally ranked according to a *crowding distance* $\textsc{cd}$, in order to achieve spread frontiers. The non-dominance-based indicator function of a sample $\boldsymbol{J}(\theta^{[j]}) \in \mathcal{D}$ is defined by

$$\begin{aligned}
\mathcal{S}_{\text{ND}}\left(\boldsymbol{J}(\theta^{[j]})\right) &= -\textsc{nd}\left(\boldsymbol{J}(\theta^{[j]})\right) + \textsc{cd}\left(\boldsymbol{J}(\theta^{[j]})\right), \\
\textsc{cd}\left(\boldsymbol{J}(\theta^{[j]})\right) &\propto \sum_{\boldsymbol{J}(\theta^{[i]}) \in \widetilde{\mathcal{F}}^{[j]}} \textsc{dist}\left(\boldsymbol{J}_{\text{N}}(\theta^{[j]}), \boldsymbol{J}_{\text{N}}(\theta^{[i]})\right), \\
\boldsymbol{J}_{\text{N}}(\theta^{[j]}) &= \frac{\boldsymbol{J}(\theta^{[j]}) - \min\{\widetilde{\mathcal{F}}^{[j]}\}}{\max\{\widetilde{\mathcal{F}}^{[j]}\} - \min\{\widetilde{\mathcal{F}}^{[j]}\}}.
\end{aligned}$$

where $\textsc{dist}$ is the Euclidean distance operator and $\widetilde{\mathcal{F}}^{[j]}$ is the sub-frontier where a solution $\boldsymbol{J}(\theta^{[j]})$ belongs to. Solutions are normalized (denoted by $\boldsymbol{J}_{\text{N}}$) as the magnitude of the objectives can introduce bias. Additionally, the crowding distance is normalized to sum to unity in order to ensure that $\textsc{cd}(\boldsymbol{J}(\theta^{[j]})) \in [0, 1]$. However, this indicator function is not consistent, as observable from a simple counter-example: assume 2-objective frontiers, where $\mathcal{F}_1 = \{(2,1),(1,2)\}$ dominates $\mathcal{F}_2 = \{(1,0),(0,1)\}$. All solutions have the same dominance count of 1 and the same crowding distance of 0.5 and, therefore, the same performance measure $J_{\mathcal{S}}$, thus violating condition (b). Nonetheless, given its large usage in the evolutionary algorithms field, in the evaluation section we investigate this indicator function as well.

2.3.4 Sample Reuse by Importance Sampling

Pareto-optimal policies can show similar behaviors and visit similar areas of the state-action space. Thus, reusing past samples is crucial for increasing the sample efficiency of a MORL algorithm and its applicability to large problems, especially in real-world tasks where sample parsimony is an important feature (e.g., in robotics). Furthermore, in real-world problems collecting samples of optimal policies might be dangerous (e.g., in the presence of stochastic environments or stochastic policies). Therefore, the possibility of using an off-policy paradigm may be decisive in the choice of the learning algorithm. However, it is not straightforward to reuse past samples in common multiple-policy MORL approaches. When several optimization procedures are performed, it can be questioned if samples collected from parallel runs should

Algorithm 1: Episodic Multi-objective Policy Search with Sample Reuse

```
1  Initialize $\varrho(\theta|\omega), M, k \leftarrow 1$
2  Repeat until terminal condition is reached
3      Collect $i = 1 \dots n_k$ samples
4          Draw policy parameters $\theta^{[k,i]} \sim \varrho(\cdot|\omega_k)$
5          Evaluate policy parameters $\boldsymbol{J}(\theta^{[k,i]}) \leftarrow \mathbb{E}_{\boldsymbol{\tau}\sim p(\cdot|\theta^{[k,i]})}[\boldsymbol{R}(\boldsymbol{\tau})]$
6      Reuse past samples and compose dataset $\mathcal{D}_\theta \leftarrow \left\{\theta^{[m,i]}\right\}_{i=1\dots n_k,\, m=k-M\dots k}$
7      Scalarize returns and compose dataset $\mathcal{D}_\mathcal{S} \leftarrow \left\{\mathcal{S}\left(\boldsymbol{J}(\theta^{[m,i]})\right)\right\}_{i=1\dots n_k,\, m=k-M\dots k}$
8      Compute importance sampling weights $w^{[m,i]} \leftarrow \dfrac{\varrho(\theta^{[m,i]} \mid \omega_k)}{\sum_{j=k-M}^{k} \alpha_j \varrho(\theta^{[m,i]} \mid \omega_j)}$
9      Update distribution $\varrho$ by MO-eREPS or MO-NES
10     $k \leftarrow k + 1$
```

be reused. If not, sample redundancy persists, as procedures with similar preferences over the objectives may collect similar samples. On the other hand, reusing samples from procedures with different preferences might not help the learning at all. On the contrary, following a manifold-based approach, our algorithms perform a single optimization procedure and are not affected by this issue. In this section, we show how to extend MO-eREPS and MO-NES to the *off-policy* paradigm by incorporating *importance sampling (IS)* [Owen and Zhou, 2000].

IS is a technique for estimating the expectation $\mathbb{E}_p[f(x)]$ w.r.t. a distribution p by using samples drawn from another distribution g. Several unbiased IS estimators have been presented in the literature [Owen and Zhou, 2000]. In this chapter, we focus on multiple IS, where N samples are observed from M distributions $\{g_m\}_{m=1}^M$, each one providing n_m samples such that $\sum_{m=1}^M n_m = N$. Let $x_{mi} \sim g_m$, $i = 1 \dots n_m$, $m = 1 \dots M$, and $w_m(x)$ be a partition of unity $0 \le w_m(x) \le \sum_{m=1}^M w_m(x) = 1$. Under mild assumptions on the distributions[4], an unbiased multiple IS estimator is

$$\mathbb{E}_p[f(x)] \approx \sum_{m=1}^{M} \frac{1}{n_m} \sum_{i=1}^{n_m} w_m(x_{mi}) \frac{f(x_{mi})p(x_{mi})}{g_m(x_{mi})}.$$

This estimator generalizes stratified sampling but can also be reduced to the case of mixture IS when the mixture distributions are independent. Among the several heuristics for w_m defined in literature, we exploit the *balance heuristic* [Owen and Zhou, 2000]

$$w_m(x) = \frac{n_m g_m(x)}{\sum_{j=1}^M n_j g_j(x)}.$$

In RL, IS has already been successfully applied for the estimation of the expected return both in step-based [Precup et al., 2000, Degris et al., 2012] and episodic settings [Daniel et al., 2012,

[4] Assume that $g_m(x) > 0$ where $w_m(x)f(x)p(x) \neq 0$.

Zhao et al., 2013]. In our case, we want to solve Problem 2.2 having access to N samples $\theta^{[m,i]}$ drawn from multiple distributions $\varrho(\cdot|\omega_m)$. The multiple IS estimate of the integral is

$$\mathbb{E}_{\varrho(\cdot|\omega)}[\mathcal{S}\left(\boldsymbol{J}(\theta)\right)] \approx \frac{1}{N}\sum_{m=1}^{M}\sum_{i=1}^{n_m}\mathcal{S}\left(\boldsymbol{J}(\theta^{[m,i]})\right)\underbrace{\frac{\varrho(\theta^{[m,i]} \mid \omega)}{\sum_{j=1}^{M}\alpha_j\varrho(\theta^{[m,i]} \mid \omega_j)}}_{w^{[m,i]}}, \tag{2.3}$$

where $\alpha_j = n_j/N$ and $\mathcal{S}$ is some indicator function as described in Section 2.3.3. This estimator is equivalent to mixture sampling with mixture responsibilities α_m and independent distributions. Algorithm 1 summarizes the complete algorithmic procedure. Note that the proposed IS extension can be directly used even in *single-objective episodic algorithms*, where $\mathcal{S}\left(\boldsymbol{J}(\theta)\right)$ is the scalar expected return $J(\theta)$.

2.4 Evaluation

The proposed algorithms, indicator functions and sample reuse were evaluated on three domains and compared against some state-of-the-art MORL algorithms. The quality of the approximate frontiers is evaluated by its hypervolume. For its computation, we consider the normalized frontier, i.e., each point $\boldsymbol{J}(\theta^{[i]})$ is normalized in the interval $[0,1]^{d_{\text{R}}}$ by

$$\boldsymbol{J}_{\text{N}}(\theta^{[i]}) = \frac{\boldsymbol{J}(\theta^{[i]}) - \boldsymbol{J}_{\text{AU}}}{\boldsymbol{J}_{\text{U}} - \boldsymbol{J}_{\text{AU}}},$$

where $\boldsymbol{J}_{\text{U}}$ and $\boldsymbol{J}_{\text{AU}}$ are the utopia and anti-utopia points, respectively. As shown in Figure 2.2, the former represents an ideal solution that simultaneously maximizes all the objectives, the latter an undesirable solution. For 2-objective problems, the hypervolume is exactly computed. For problems involving more objectives, given its high computational complexity, the hypervolume is approximated with a Monte-Carlo estimate as the percentage of points dominated by the frontier in the cube defined by the utopia and antiutopia points. For the estimation, one million points were used. Furthermore, we compare the sample complexity of each algorithm, meant as the total number of episodes collected during learning before convergence, and the number of Pareto-solutions returned. To this aim, the algorithms are executed for a fixed amount of iterations and are evaluated at each iteration. Tables report the number of iterations after which the hypervolume trend is constant.

First, a water reservoir control task is chosen to evaluate the two proposed indicator functions and the effects of IS on MO-eREPS and MO-NES. This task has already been used in literature [Castelletti et al., 2013, Parisi et al., 2014, Pirotta et al., 2015] and, although not highly complex, it is helpful to assess the indicator functions performance and to show how the proposed IS can effectively reduce the sample complexity. The algorithms are then compared with state-of-the-art competitors, namely a weighted sum approach with episodic REPS (WS-eREPS, it consists of episodic REPS to solve several single-objective optimization on varying the weights that linearly combine the immediate rewards), S-Metric Selection Evolutionary Multi-objective Algorithm (SMS-EMOA) [Beume et al., 2007], Pareto-Following (PFA) and Radial (RA) Algorithms [Parisi et al., 2014] and Pareto Manifold Gradient Algorithm (PMGA) [Pirotta et al., 2015]. Afterwards, the algorithms are evaluated in the presence of many objectives on a linear-quadratic regulator and on the more complex task of tetherball robot hitting game.

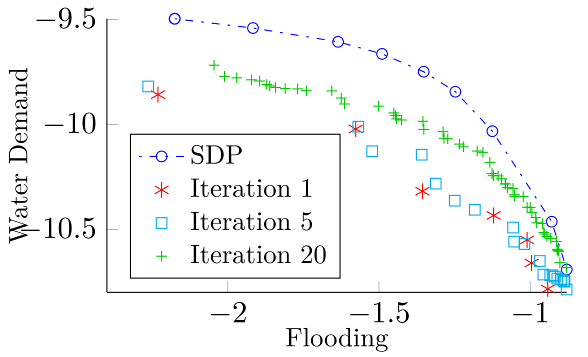


Figure 2.4: MO-NES iterations on the 2-objective water reservoir problem using $\mathcal{S}_{\text{HV}}$. At each iteration, the algorithm generates more non-dominated solutions and moves the approximate frontier closer to the reference one.

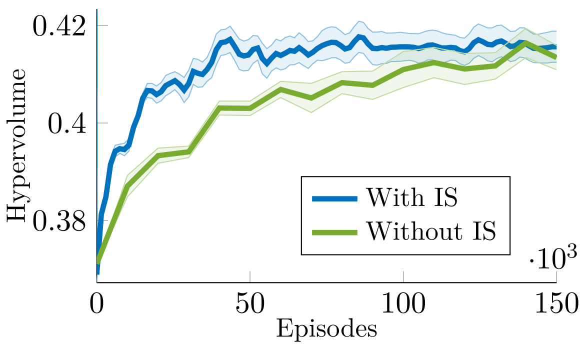


Figure 2.5: Hypervolume trend on the 2-objective water reservoir problem. Shaded area denotes half of standard deviation (results are averaged over ten trials). Using IS, MO-eREPS attains a higher hypervolume using less episodes.

We recall that, by learning a manifold in the policy parameters space, MO-eREPS and MO-NES can generate an infinite number of solutions. However, for the evaluation, a finite number of solutions was drawn and dominated ones were filtered out and, therefore, their frontier are discretized. For the computation of the hypervolume-based indicator function, a constant penalty of $\Upsilon = 0.1$ was applied to dominated solutions. For each case study, domains are first presented and then results (averaged over ten trials) are reported and discussed. For the details of the algorithms setup, e.g., the learning rates, we refer the reader to the Appendix.

2.4.1 Water Reservoir Control

In this task, originally presented by Castelletti et al. [Castelletti et al., 2012], an agent has to control the amount of water to be released from a reservoir while pursuing three conflicting objectives, i.e., preventing flooding along the lake shores and satisfying both water and electricity demands. Below, we present results related to both the 2-objective scenario (in which only flooding and water demand are considered) and the 3-objective one. The environment is stochastic, as the initial state is drawn from a discrete set and a random inflow determines the transition function $\mathcal{P}\left(s'|s,a\right)$, i.e.,

$$s' = s + \xi - \max(\underline{a}, \min(\overline{a}, a)),$$

where $s \in \mathbb{R}$ represents the water volume stored in the reservoir, $a \in \mathbb{R}$ is the amount of water released by the agent and $\xi \sim \mathcal{N}(40, 100)$ is the stochastic water inflow. The constrains $\underline{a}$ and $\overline{a}$ are the minimum and the maximum releases associated to storage s defined by the relations $\overline{a} = s$ and $\underline{a} = \max(s - 100, 0)$. The reward functions are

$$\begin{aligned}
\mathcal{R}_1(s,a,s') &= -\max(h' - \overline{h}, 0),\\
\mathcal{R}_2(s,a,s') &= -\max(\overline{\rho} - \rho, 0),\\
\mathcal{R}_3(s,a,s') &= -\max(\overline{e} - e', 0),
\end{aligned}$$

Table 2.1: Comparison of proposed indicator functions $\mathcal{S}$ on the 2-objective water reservoir problem (margins denote standard deviation over ten trials). The hypervolume-based one attains the best results, both in terms of quality of the frontier and sample efficiency.

	$\mathcal{S}$	Hypervolume	#Episodes (10^3)
MO-eREPS	ND	0.3972 ± 0.0165	414 ± 110
	HV	$\mathbf{0.4124 \pm 0.0098}$	$\mathbf{133 \pm 32}$
MO-NES	ND	0.4048 ± 0.0110	278 ± 67
	HV	$\mathbf{0.4114 \pm 0.0048}$	$\mathbf{82 \pm 12}$

Table 2.2: Effects of using importance sampling on the 2-objective water reservoir problem. IS successfully improves the algorithms performance and substantially reduces the samples required for learning.

	IS	Hypervolume	#Episodes (10^3)
MO-eREPS	✓	$\mathbf{0.4179 \pm 0.0124}$	$\mathbf{42 \pm 6}$
	✗	0.4124 ± 0.0098	133 ± 32
MO-NES	✓	$\mathbf{0.4199 \pm 0.0117}$	$\mathbf{45 \pm 4}$
	✗	0.4114 ± 0.0048	82 ± 12

where $h' = s'/S$ is the reservoir level, $S = 1$ is the reservoir surface, $\overline{h} = 50$ is the flooding threshold, $\rho = \max(\underline{a}, \min(\overline{a}, a))$ is the release from the reservoir, $\overline{\rho} = 50$ is the water demand, $\overline{e} = 4.36$ is the electricity demand and e' is the electricity production

$$e' = \psi \, g \, \eta \, \gamma_{H_2O} \, \rho \, h',$$

where $\psi = 10^{-6}/3.6$ is a dimensional conversion coefficient, $g = 9.81$ the gravitational acceleration, $\eta = 1$ the turbine efficiency and $\gamma_{H_2O} = 1,000$ the water density. $\mathcal{R}_1$ denotes the negative of the cost due to the flooding excess level, $\mathcal{R}_2$ is the negative of the deficit in water supply and $\mathcal{R}_3$ is the negative of the deficit in hydro-power production. The discount factor is set to 1 for all the objectives and the initial state is drawn from a finite set. As the problem is continuous we exploit a Gaussian policy

$$\pi_\theta(a|s) = \mathcal{N}\left(\mu + \nu(s)^\mathsf{T}\kappa, \sigma^2\right),$$
$$\nu_i(s) = \exp\left(-\frac{||s - c_i||_2^2}{b_i}\right),$$

where $\theta = \{\mu, \kappa, \sigma\}$ and $\nu : \mathcal{S} \to \mathbb{R}^{d_\theta}$ are radial basis functions with centers c_i and bandwidths b_i. We used four basis functions uniformly placed in the interval $[-20, 190]$ with bandwidths b_i of 60, for a total of six parameters to learn, i.e., $|\theta| = 6$. The sampling distribution is also Gaussian

$$\varrho(\theta|\omega) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Lambda}^\mathsf{T}\boldsymbol{\Lambda}),$$

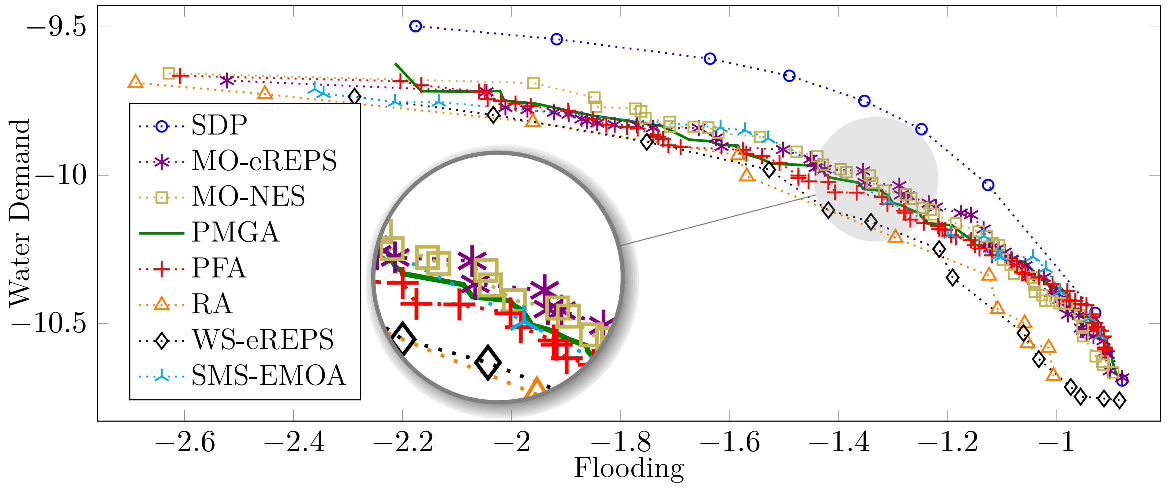


Figure 2.6: Visual comparison for the 2-objective water reservoir problem. The frontiers returned by MO-eREPS and MO-NES are comparable to the ones of state-of-the-art algorithms. In this example, WS-eREPS and RA attains the worst results, as their solutions are dominated by competitors' frontiers.

Table 2.3: Numerical comparison for 2-objective water reservoir (margins denote standard deviation over ten trials). The SDP reference frontier has a hypervolume of 0.4955. PMGA attains the best results, followed by MO-NES and MO-eREPS.

Algorithm	Hypervolume	#Episodes (10^3)	#Solutions
MO-eREPS	0.4179 ± 0.0124	42 ± 6	∞
MO-NES	0.4199 ± 0.0117	45 ± 4	∞
PMGA	$\mathbf{0.4263 \pm 0.0069}$	$\mathbf{16 \pm 1}$	∞
PFA	0.4132 ± 0.0083	28 ± 5	51 ± 11
RA	0.3300 ± 0.0034	59 ± 3	16 ± 3
WS-eREPS	0.3713 ± 0.0062	37 ± 2	17 ± 4
SMS-EMOA	0.3994 ± 0.0151	150 ± 35	14 ± 2

where $\mathbf{\Lambda}$ is an upper triangular matrix. The high-level parameters to be learned are $\omega = \{\boldsymbol{\mu}, \mathbf{\Lambda}\}$, with $|\omega| = 27$. For this distribution, the FIM can be computed in closed form [Sun et al., 2009]. Being episode-based, WS-eREPS performs its searches over the same distribution.

For learning, given the stochasticity of the policy and of the environment, MO-eREPS and MO-NES collect 100 episodes of 100 steps for estimating the quality of a sample $\theta^{[i]}$. For the evaluation, each solution is evaluated over 1,000 episodes of 100 steps. As reference frontier, solutions returned by Stochastic Dynamic Programming (SDP) have been used [Castelletti et al., 2012].

Evaluation of Proposed Indicator Functions

Our first concern is to evaluate the indicator functions $\mathcal{S}$. To this aim, we compare the frontier returned by MO-eREPS and MO-NES without IS using both the hypervolume-based (HV) and

Table 2.4: Numerical comparison for 3-objective water reservoir. The SDP reference frontier has a hypervolume of 0.7192. As we increase the complexity of the problem, PMGA is outperformed by both MO-NES and MO-eREPS in terms of hypervolume. Furthermore, its sample complexity (although still the best along with MO-NES) increases substantially compared to the 2-objective case.

Algorithm	Hypervolume	#Episodes (10^3)	#Solutions
MO-eREPS	0.6763 ± 0.0066	72 ± 32	∞
MO-NES	$\mathbf{0.6779 \pm 0.0021}$	$\mathbf{62 \pm 12}$	∞
PMGA	0.6701 ± 0.0036	$\mathbf{62 \pm 8}$	∞
PFA	0.6521 ± 0.0029	343 ± 13	595 ± 32
RA	0.6510 ± 0.0047	626 ± 36	137 ± 25
WS-eREPS	0.6139 ± 0.0003	187 ± 9	86 ± 10
SMS-EMOA	0.6534 ± 0.0007	507 ± 57	355 ± 14

the non-dominance-based (ND) indicator function on the 2-objective case. At each iteration, the algorithms collect 50 new samples to perform a policy update. An example of learning process with $\mathcal{S}_{\text{HV}}$ is shown in Figure 2.4. Table 2.1 shows no significant differences in hypervolume between the two indicator functions. However, the hypervolume-based indicator function attains greater sample efficiency and lower hypervolume variance. This behavior is not unexpected, considering that $\mathcal{S}_{\text{ND}}$ is not consistent, unlike $\mathcal{S}_{\text{HV}}$. Therefore, for the remainder of the evaluation we use the hypervolume-based indicator function.

Evaluation of Importance Sampling

The next setup aims to evaluate IS. IS-aided algorithms collect only ten samples at each iteration and reuse samples collected by the last four policies for a total of 50 samples per policy update. Table 2.2 shows numerical results, while Figure 2.5 shows the hypervolume trend for MO-eREPS. From the results, we can assert that IS effectively increases the algorithms performance in terms of sample efficiency without affecting the quality of the approximate frontier.

Comparison of State-of-the-art Methods

Finally, we compare MO-eREPS and MO-NES with state-of-the-art algorithms in MORL on both the 2- and 3-objective scenario. For the latter, IS is performed similarly as described above, with the difference that 50 new samples are collected at each iteration, for a total of 250 samples exploited for a policy update.

Figure 2.6 shows the frontiers returned by all the algorithms for the 2-objective case, while Tables 2.3 and 2.4 shows numerical results for both scenarios. For the 2-objective case, PMGA performs slightly better than MO-eREPS and MO-NES in terms of hypervolume, but its sample efficiency is substantially lower. However, for the 3-objective scenario, MO-REPS and MO-NES attain the best results. Their hypervolume is the highest and MO-NES sample complexity is on par with PMGA. Furthermore, MO-eREPS and MO-NES sample complexity scale better with the number of objectives than PMGA. For the formers, in fact, the number of collected samples

Table 2.5: Numerical results for the LQR (margins denote standard deviation over ten trials). Only MO-NES and MO-eREPS were able to scale well to a higher number of objectives, returning frontiers with the highest hypervolume without consuming the fixed samples budget.

Algorithm	Hypervolume	#Episodes (10^3)	#Solutions
MO-eREPS	0.3511 ± 0.0043	620 ± 75	∞
MO-NES	$\mathbf{0.3585 \pm 0.0057}$	$\mathbf{540 \pm 90}$	∞
PMGA	0.3391 ± 0.0044	$1,000$	∞
PFA	0.1687 ± 0.0033	$1,000$	$3,581 \pm 298$
RA	0.2778 ± 0.0029	$1,000$	$1,069 \pm 73$
WS-eREPS	0.2517 ± 0.0063	$1,000$	$3,089 \pm 156$
SMS-EMOA	0.3023 ± 0.0050	$1,000$	$1,713 \pm 149$

increased by a factor of $\sim$1.5 (e.g., from $45,000$ to $62,000$), while for the latter of $\sim$3.8 (from $16,000$ to $62,000$). This behavior might be due to both the use of episodic exploration and IS. We recall that extending IS to competing algorithms is not straightforward. PFA, RA and WS-eREPS perform several policy searches, while PMGA and SMS-EMOA do not rely on any sampling distribution and IS, as proposed in this chapter, cannot be applied.

The qualities shown by MO-eREPS and MO-NES, sample efficiency above all, suggest that they might be particularly suited for many-objectives problems. We empirically investigate this aspect in the next domain.

2.4.2 Linear-Quadratic Regulator

The next evaluation focuses on the performance of the algorithms in the presence of several objectives, i.e., in *many-objectives* problems. To this aim, we solve a linear-quadratic regulator (LQR) problem with five objectives, a particularly interesting case of study as the objective functions $J_i(\theta)$ can be expressed in closed form. The single-objective LQR problem is defined by the following dynamics [Peters and Schaal, 2008b]

$$s' = As + Ba,$$
$$\mathcal{R}(s,a)(s,a) = -s^{\mathsf{T}}Qs - a^{\mathsf{T}}Ra,$$

where s and a are d_{s}-dimensional column vectors, $A, B, Q, R \in \mathbb{R}^{d_{\mathrm{s}} \times d_{\mathrm{s}}}$, Q is a symmetric semidefinite matrix and R is a symmetric positive definite matrix. Dynamics are not coupled, i.e., A and B are identity matrices. The low-level policy is Gaussian $\pi(s,a) = \mathcal{N}(\boldsymbol{K}s, \boldsymbol{I})$, where $K \in \mathbb{R}^{d_{\mathrm{s}} \times d_{\mathrm{s}}}$ is diagonal and $\boldsymbol{I}$ is the identity matrix. The policy parameters are $\theta = \{K_{ii}\}_{i=1\ldots5}$.

The LQR can be easily extended to account for multiple conflicting objectives [Parisi et al., 2014]. The i-th objective represents the problem of minimizing both the distance from the origin w.r.t. the i-th axis and the cost of the action over the other axes, i.e.,

$$\mathcal{R}_i(s,a) = -s_i^2 - \sum_{j \neq i} a_j^2.$$

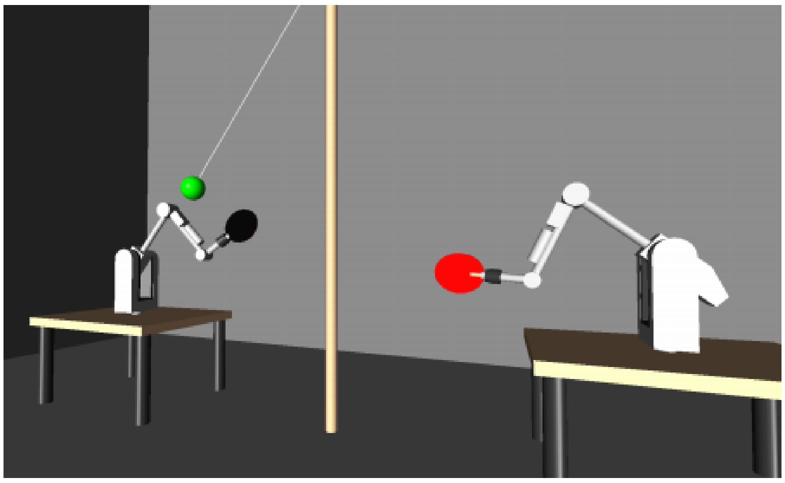


Figure 2.7: In the tetherball robot game one player has to hit a ball hanging from a pole without giving the opponent the chance to unwind it.

Since the maximization of the i-th objective requires to have null action on the other axes, objectives are conflicting. As this reward formulation violates the positiveness of matrix R_i, we change it by adding a sufficiently small ξ-perturbation

$$\mathcal{R}_i(s,a) = -(1-\xi)\left(s_i^2 + \sum_{i \neq j} a_j^2\right) - \xi\left(\sum_{j \neq i} s_j^2 + a_i^2\right).$$

In our experiments we set $\gamma = 0.9, \xi = 0.1$ and the initial state to $s_0 = [10, 10, 10, 10, 10]^\mathsf{T}$.

The high number of objectives substantially increases the complexity of the problem compared to the water reservoir control task. Therefore, we give each algorithm a learning budget of one million samples and end the learning when the budget is consumed or when the hypervolume trend becomes constant. Given the stochasticity of the policy π, during learning solutions are evaluated over 150 episodes of 50 steps. For the evaluation, the closed form $\boldsymbol{J}(\theta)$ is used. For episodic algorithms, a Gaussian sampling distribution is used, i.e., $\varrho(\theta|\omega) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Lambda}^\mathsf{T}\boldsymbol{\Lambda})$ ($|\omega| = 20$, $\boldsymbol{\Lambda}$ is an upper-triangular matrix). At each iteration MO-eREPS and MO-NES collect 200 samples and reuse the previous 800.

As shown in Table 2.5, MO-NES and MO-eREPS attain the best results, outperforming state-of-the-art competitors. In particular, they converged without consuming the whole samples budget, proving once more to be sample efficient. PMGA achieves the third-highest hypervolume, while the remaining algorithms perform substantially worse. The reason is that PFA, RA and WS-eREPS implement inefficient approaches, requiring to solve several independent policy searches. For the same reason, even if PFA and WS-eREPS return more solutions than SMS-EMOA, their hypervolume is lower as they consume the samples budget before completing the optimization procedures.

2.4.3 Simulated Robot Tetherball

The last case study is a robot tetherball hitting game [Parisi et al., 2015], an episodic RL domain shown in Figure 2.7. This problem differs from the previous ones as the number of policy

Table 2.6: Numerical results for the tetherball hitting game (margins denote standard deviation over ten trials). MO-eREPS hypervolume is by far the highest and its sample complexity is remarkably low, especially compared to RA, PFA and WS-eREPS.

Algorithm	Hypervolume	#Episodes (10^3)	#Solutions
MO-eREPS	$\mathbf{0.7983 \pm 0.0146}$	1.6 ± 0.3	∞
MO-NES	0.5879 ± 0.0125	$\mathbf{1.3 \pm 0.4}$	∞
RA	0.6001 ± 0.097	9.8 ± 3	25 ± 5
PFA	0.6302 ± 0.0697	18 ± 2.2	22 ± 9
WS-eREPS	0.6869 ± 0.0192	10 ± 4	26 ± 3
SMS-EMOA	0.6205 ± 0.0799	2.7 ± 0.8	17 ± 6

parameters θ is substantially higher. In the original work, the robot has only one objective — hitting the ball to score — and the ball trajectory was determined by different opponent strokes. Here, we simplify the task by fixing the opponent stroke and we consider two conflicting goals. The first requires the robot to produce safe smooth movements by minimizing the jerk of the trajectory (i.e., avoiding jumps in the joints acceleration). The second considers the strength of the agent stroke and rewards the robot for producing fast (but potentially harmful) movements by maximizing the speed of the ball after a hit. Formally

$$\begin{aligned}
R_1(\boldsymbol{\tau}) &= r_{\text{SCORE}} + r_{\text{DISTANCE}} + r_{\text{JERK}}, \\
R_2(\boldsymbol{\tau}) &= r_{\text{SCORE}} + r_{\text{DISTANCE}} + r_{\text{SPEED}}, \\
r_{\text{SCORE}} &= \begin{cases} 0 & \text{if the agent hits the ball back to the opponent} \\ -10 & \text{otherwise} \end{cases}, \\
r_{\text{DISTANCE}} &= \lambda_1 \left(\exp(-f_{\text{DISTANCE}}(\boldsymbol{\tau})) - 1\right), \\
r_{\text{SPEED}} &= \lambda_2 \left(\exp(-f_{\text{SPEED}}(\boldsymbol{\tau})) - 1\right), \\
r_{\text{JERK}} &= \lambda_3 \left(\exp(-f_{\text{JERK}}(\boldsymbol{\tau})) - 1\right),
\end{aligned}$$

where f_{DISTANCE} is the minimum distance between the ball and the paddle during the episode, f_{JERK} the total jerk along the trajectory and f_{SPEED} the velocity of the ball after being hit. The scale factors λ_i are to transform costs into rewards and to scale the objectives magnitude.

Actions a are the accelerations applied to the joints at each time step. The low-level policy $\pi(s, a)$ are Dynamic Motor Primitives (DMPs) by Ijspeert et al. [Ijspeert et al., 2002], one for each joint. DMPs offer a compact representation of the acceleration profile by a second order dynamical system, i.e., $a = f(\nu(s), \theta)$, where f is a non-linear forcing function and $\nu(s)$ are basis functions. A single vector θ encodes an entire trajectory and by learning the parameters θ the robot performs strokes with different shape and speed. For our experiments, we used five radial basis functions, resulting in a total of 30 parameters θ. As the experiments are performed in simulation, the environment is deterministic and each sample $\theta^{[i]}$ is evaluated over one single trajectory.

The high-level sampling distribution for MO-eREPS is a Gaussian mixture model with eight components, i.e., $\varrho(\theta|\boldsymbol{\omega}) = \sum_{j=1}^{8} p_j \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$, with $\boldsymbol{\omega} = \{p_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\}_{j=1\dots8}$ and $|\boldsymbol{\omega}| = 7,448$.

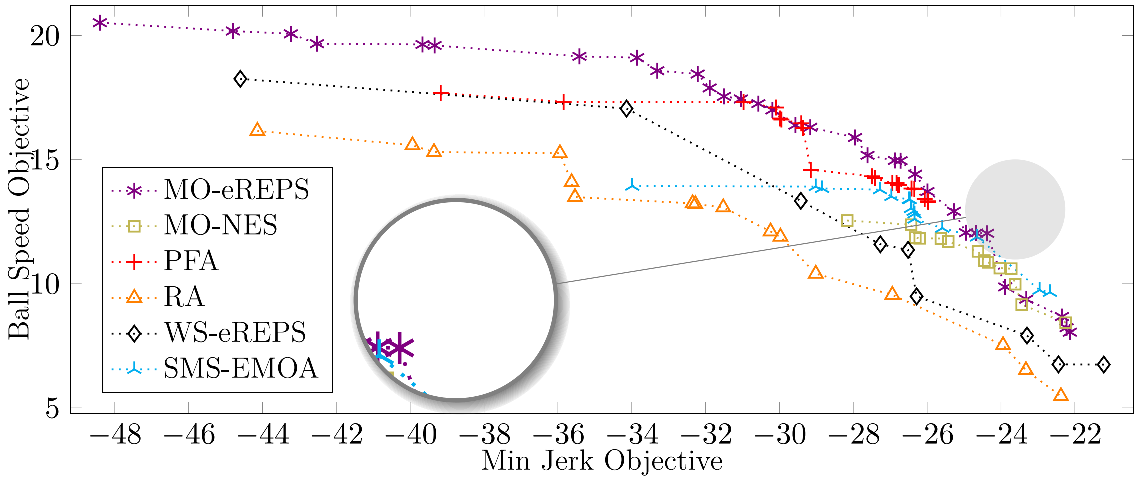


Figure 2.8: Different approximations of the Pareto frontier for the tetherball hitting game. MO-eREPS outperforms all other algorithms, returning a uniform and spread frontier with the highest hypervolume. As in the water reservoir domain, RA and WS-eREPS solutions are dominated by competitors frontiers.

MO-NES uses the same Gaussian of the previous experiments ($|\omega| = 495$), as the computation of the FIM for the mixture model requires a high number of samples. For the initialization of both distributions, eight different trajectories have been sampled in simulation: for MO-eREPS, each one has been used to initialize a component of the mixture model, while for MO-NES their mean and covariance has been used to initialize the single Gaussian. For updating the distribution, at each iteration both algorithms collect 20 new samples and reuse the last 180. As DMPs perform exploration in the policy parameters space rather than in the action space, PFA, RA, WS-eREPS and SMS-EMOA learned policies are high-level distributions as well. Since these algorithms perform several optimization procedures and naturally learn many distributions, they exploit the same single Gaussian as MO-NES, as a mixture model would be redundant. On the contrary, being purely step-based, PMGA is not applicable to this domain.

Table 2.6 and Figure 2.8 show numerical and graphical results. MO-eREPS attains the best results, with the highest hypervolume and the second-lowest sample complexity. However, MO-NES does not perform as well as previous experiments, although it still attains the lowest sample complexity. The reason of such behavior lies in the different sampling distribution used. Exploiting a mixture model, MO-eREPS is able to approximate more accurately the true manifold in the policy parameters space. At the same time, having less parameters to learn, MO-NES converges earlier than MO-eREPS. We stress once more the effectiveness of the hypervolume indicator function $\mathcal{S}_{\text{HV}}$ in driving the manifold to generate only Pareto-optimal solutions. As shown in Figure 2.9, the high-level distribution initially produces only medium and low quality policies, with solutions far from the Pareto frontier. After learning with MO-eREPS and $\mathcal{S}_{\text{HV}}$, it indeed generates only high quality policies, with solutions in the proximity of the Pareto frontier. It is worth noting that only $\sim 15\%$ of these solutions are non-dominated, due to the stochasticity of the sampling distribution. However, all dominated solutions are very close to the approximate frontier. If we slightly relax Pareto-optimality condition and consider a neighborhood of the approximate frontier (e.g., with a 5% tolerance on the objectives), then more than $\sim 60\%$ solutions are non-dominated.

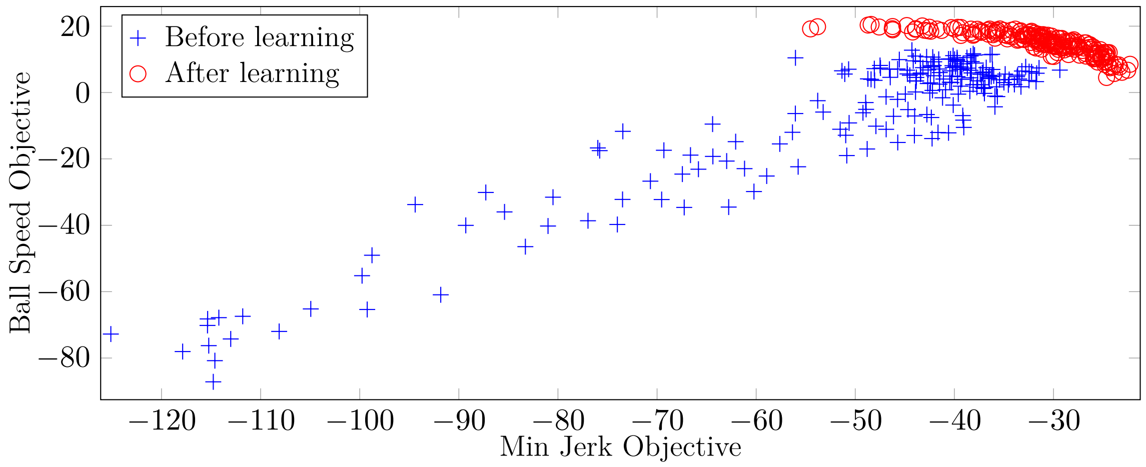


Figure 2.9: Sample solutions drawn by the high-level distribution learned by MO-eREPS. Before learning, the manifold generates random policies, including extremely poor ones. After learning, it generates only solutions in the proximity of the Pareto frontier, denoting once again the effectiveness of the hypervolume-based indicator function.

Concerning the other algorithms, we underline the very high sample complexity of PFA, RA and WS-eREPS and the high hypervolume variance of PFA and SMS-EMOA. The former is due to the algorithms inefficient approach, as multiple optimization procedures are performed without reusing past samples. The latter is caused by SMS-EMOA intrinsic stochastic nature (e.g., mutation and crossover) and PFA high sensibility to the learning parameters (e.g., learning rates).

2.5 Conclusion

In this chapter, we presented two novel manifold-based MORL algorithms that combine episodic approaches and importance sampling to solve MOMDPs. Unlike the majority of state-of-the-art approaches, our algorithms perform a manifold-based policy search and directly learn a manifold in the policy parameter space to generate infinitely many Pareto-optimal solutions in a single run. We also proposed and evaluated an off-policy extension by including importance sampling in the learning process, in order to further reduce the sample complexity, and two Pareto optimality indicator functions to assess the quality of an approximate frontier.

Evaluated on several domains, our algorithms outperformed state-of-the-art competitors both in terms of quality of the learned Pareto frontier and sample efficiency. In particular, they proved to perform well in the presence of many objectives and high-dimensional parameter spaces. Furthermore, since they do not require any pre-parameterization of the manifold and can exploit any Pareto optimality indicator function, our algorithms provide a versatile approach for solving MOMDPs.

In some domains MO-eREPS attained the best results, while in other experiments we have seen that MO-NES performs better. We experienced that MO-NES effectiveness comes from the ability of computing the Fisher information matrix in closed-form (at least for Gaussians). However, the closed-form is known only for few distribution families and therefore MO-NES might not be suitable for many problems.

These properties (above all the sample efficiency) open the way to the use of the proposed algorithms on real-world applications. It would be interesting to evaluate the performance of our algorithms on high-dimensional robotic domains. However, such problems often require to learn the optimal policy for different contexts. We therefore believe that the integration of contextual learning with multi-objective optimization is a relevant topic in MORL both from a theoretical and a practical perspective.

3 Long-Term Visitation Value for Deep Exploration in Sparse Reward Reinforcement Learning

Reinforcement learning with sparse rewards is still an open challenge. Classic methods rely on getting feedback via extrinsic rewards to train the agent, and in situations where this occurs very rarely the agent learns slowly or cannot learn at all. Similarly, if the agent receives also rewards that create suboptimal modes of the objective function, it will likely prematurely stop exploring. More recent methods add auxiliary intrinsic rewards to encourage exploration. However, auxiliary rewards lead to a non-stationary target for the Q-function. In this paper, we present a novel approach that (1) plans exploration actions far into the future by using a long-term visitation count, and (2) decouples exploration and exploitation by learning a separate function assessing the exploration value of the actions. Contrary to existing methods which use models of reward and dynamics, our approach is off-policy and model-free. We further propose new tabular environments for benchmarking exploration in reinforcement learning. Empirical results on classic and novel benchmarks show that the proposed approach outperforms existing methods in environments with sparse rewards, especially in the presence of rewards that create suboptimal modes of the objective function. Results also suggest that our approach scales gracefully with the size of the environment.
Source code is available at `https://github.com/sparisi/visit-value-explore`

3.1 Introduction

Reinforcement learning (RL) is a process where an agent learns how to behave in an environment by trial and error. The agent performs actions and, in turn, the environment may provide a *reward*, i.e., a feedback assessing the quality of the action. The goal of RL is then to learn a *policy*, i.e., a function producing a sequence of actions yielding the maximum cumulative reward. Despite its simplicity, RL achieved impressive results, such as learning to play Atari videogames from pixels [Mnih et al., 2013, Schulman et al., 2017], or beating world-class champions at Chess, Go and Shogi [Silver et al., 2017b]. However, a high-quality reward signal is typically necessary to learn the optimal policy, and without it RL algorithms may perform poorly even in small environments [Osband et al., 2019].

Characteristics of high-quality rewards. One important factor that defines the quality of the reward signal is the *frequency* at which rewards are emitted. Frequently emitted rewards are called “dense”, in contrast to infrequent emissions which are called “sparse”. Since improving the policy relies on getting feedback via rewards, the policy cannot be improved until a reward is obtained. In situations where this occurs very rarely, the agent learns slowly or cannot learn at all. Furthermore, reinforcement signals should encourage the agent to find the best actions to solve the given task. However, the environment may also provide *distracting* rewards, i.e., rewards that create suboptimal modes of the objective function. In this case, the agent should

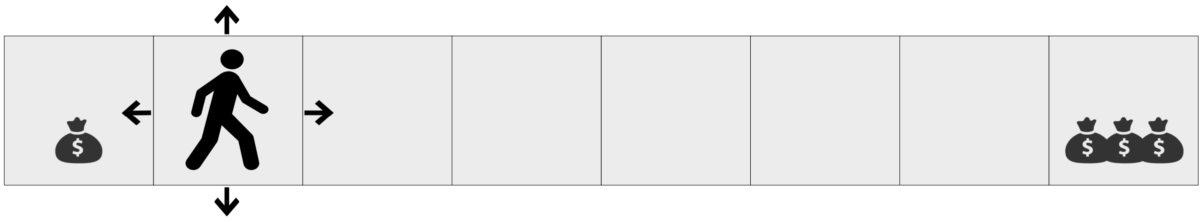


Figure 3.1: A simple problem where naive exploration performs poorly. The agent starts in the second leftmost cell and is rewarded only for finding a treasure. It can move up / down / left / right, and if it tries to move out of the chain or if it finds a treasure its position resets. Selecting the action randomly at each step takes on average 4^N attempts to find the rightmost and most valuable reward, where N is the number of cells to the right of the agent. As no reward is given in any intermediate cell, ϵ-greedy exploration is likely to converge to the local optimum represented by the leftmost reward if ϵ decays too quickly.

be able to extensively explore the environment without prematurely converge to locally optimal solutions caused by distracting rewards.

Reward engineering. In the presence of sparse distracting rewards, efficiently exploring the environment to learn the optimal policy is challenging. Therefore, many RL algorithms rely on well-shaped reward functions such as quadratic costs. For example, in a collect-and-deliver task, we could design an additional reward function that rewards proximity to the items to be collected. These well-shaped functions help to guide the agent towards good solutions and avoid bad local optima. However, from the perspective of autonomous learning, this so-called reward engineering is unacceptable for three reasons. First, the commonly utilized reward functions heavily restrict the solution space and may prevent the agent from learning optimal behavior (especially if no solution to the task is known). Second, it is easy to misspecify the reward function and cause unexpected behavior. For instance, by getting excessively high rewards for proximity to the items the agent may never learn to deliver them. Third, reward engineering requires manual tuning of the reward function and the manual addition of constraints. This process requires expertise and the resulting reward function and constraints may not transfer to different tasks or environments.

In this paper, we address the problem of reinforcement learning with sparse rewards without relying on reward engineering, or requiring prior knowledge about the environment or the task to solve. In particular, we highlight the problem of learning in the presence of rewards that create suboptimal modes of the objective function, which we call "distractors".

The exploration-exploitation dilemma. Learning when rewards are sparse and possibly distracting challenges classic RL algorithms. Initially, the agent has no knowledge about the environment and needs to explore it in search for feedback. As it gathers rewards, the agent develops a better understanding of the environment and can estimate which actions are "good", i.e., yield positive feedback, and which are not. Then, at any point, it needs to decide whether it should explore the environment in search for better rewards by executing new actions, or exploit its current knowledge and execute known good actions. This "exploration-exploitation dilemma" is frequently addressed naively by dithering [Mnih et al., 2013, Lillicrap et al., 2016]. In continuous action spaces Gaussian noise is added to the action, while in discrete spaces actions are chosen ϵ-greedily, i.e., optimally with probability $1-\epsilon$ and randomly with probability ϵ. Typically, the initial noise and ϵ are large and then decay over time. These approaches work in environments where random sequences of actions are likely to cause positive rewards. However, if rewards are sparse or are given only for specific sequences of actions, it is very unlikely that the agent will receive any feedback. In this case, the worst-case sample complexity for learning the optimal policy with dithering exploration is exponential in the number of states and actions [Kakade, 2003, Szepesvari, 2010, Osband et al., 2016b]. Furthermore, if distractor rewards are easily reachable, the agent may prematurely converge to local optima. An example of this problem is depicted in Figure 3.1.

Deep exploration. Well-performing exploration strategies should be able to solve long-horizon problems with sparse and possibly distracting rewards in large state-action spaces while remaining computationally tractable. This can be achieved if the agent takes several coherent actions to explore unknown states instead of just locally choosing the most promising actions. These behaviors are usually referred as "deep exploration" and "myopic exploration", respectively [Osband et al., 2016a]. RL literature has a long history of algorithms for deep exploration, even though the term "deep" became popular only in recent years. The first algorithms to guarantee full exploration of tabular environments date back to Kearns and Singh [2002] and Brafman and Tennenholtz [2002]. These algorithms learn a model of the environment, i.e., the transition and reward functions, and plan actions accordingly to a state-action visitation count in order to favor less visited states. Since then, other model-based [Jaksch et al., 2010, Hester and Stone, 2013] as well as model-free algorithms [Strehl et al., 2006, Bellemare et al., 2016, Jin et al., 2018, Dong et al., 2019] have been proposed. However, despite their strong guarantees, these algorithms either require to learn the complete model of the environment, rely on optimistic initialization, or have impractically high sample complexity.

In continuous environments, intrinsic motivation and bootstrapping are the most prominent approaches. The former defines an additional *intrinsic* reward added to the environment *extrinsic* reward. If the extrinsic reward is sparse, the intrinsic reward can fill the gaps between the sparse signals, possibly giving the agent quality feedback at every timestep. This approach can be used in conjunction with any RL algorithm by just providing the modified reward signal. However, the quality of exploration strongly depends on the intrinsic reward, which may not scale gracefully with the extrinsic reward and therefore needs hand-tuning. Furthermore, combining exploration and exploitation by summing intrinsic and extrinsic reward can result in undesired behavior due to the non-stationarity of the augmented reward function. Moreover, convergence is harder to guarantee [Kolter and Ng, 2009], and in general, there is no agreement on the definition of the best intrinsic reward [Houthooft et al., 2016, Pathak et al., 2017]. Bootstrapping, instead, is used to estimate the actions value posterior distribution over the environment, which then drives exploration. Because actions are selected with regard to the level of uncertainty associated with their value estimates, bootstrapping incentivizes experimentation with actions of highly uncertain value and, thus, induces exploration [Osband et al., 2019]. However, these methods rely on approximated posteriors and usually lack guarantees of convergence, unless either the environment model is learned or a time-dependent policy is used [Osband et al., 2019].

Deep exploration via long-term visitation value. *In this paper, we present a novel approach that (1) plans exploration actions far into the future by using a long-term visitation count, and (2) decouples exploration and exploitation by learning a separate function assessing the exploration value of the actions.* Contrary to existing methods which use models of reward and dynamics, our approach is off-policy and model-free.

We further comprehensively benchmark our approach against existing algorithms on several environments, stressing the challenges of learning with sparse and distracting rewards. Empirical results show that the proposed approach outperforms existing algorithms, and suggest that it scales gracefully with the size of the environment.

3.2 Preliminaries

We start with a description of the RL framework, providing the reader with the notation used in the remainder of the paper. Subsequently, we review the most prominent exploration strategies in literature, discuss their shortcomings, and identify open challenges.

3.2.1 Reinforcement Learning and Markov Decision Processes

We consider RL in an environment governed by a Markov Decision Process (MDP). An MDP is described by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mu_1 \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}\left(s'|s,a\right)$ defines a Markovian transition probability density between the current s and the next state s' under action a, $\mathcal{R}\left(s,a\right)$ is the reward function, and μ_1 is initial distribution for state s_1. The state and action spaces could be finite, i.e., $|\mathcal{S}| = d_{\text{S}}$ and $|\mathcal{A}| = d_{\text{A}}$, or continuous, i.e., $\mathcal{S} \subseteq \mathbb{R}^{d_{\text{S}}}$ and $\mathcal{A} \subseteq \mathbb{R}^{d_{\text{A}}}$. The former case is often called *tabular* MDP, as a table can be used to store and query all state-action pairs if the size of the state-action space is not too large. If the model of the environment is known, these MDPs can be solved exactly with dynamic programming. The latter, instead, requires function approximation and algorithms typically guarantee convergence to local optima. In the remainder of the paper, we will consider only tabular MDPs, and will consider only model-free RL, i.e., the model of the environment is neither known nor learned.

Given an MDP, the goal of RL is to learn to act. Formally, we want to find a *policy* $\pi(a|s)$ to take an appropriate action a when the agent is in state s. By following such a policy starting at initial state s_1, we obtain a sequence of states, actions and rewards $(s_t, a_t, r_t)_{t=1\ldots H}$, where $r_t = \mathcal{R}(s_t, a_t)$ is the reward at time t, and H is the total timesteps also called *horizon*, which can possibly be infinite. We refer to such sequences as *trajectories* or *episodes*.

Our goal is to find a policy that maximizes the expected return

$$Q^*(s,a) := \max_{\pi} \; \mathbb{E}_{\mu_\pi(s)\pi(a|s)}[Q^\pi(s,a)]\,, \tag{3.1}$$

$$\text{where} \quad Q^\pi(s_t,a_t) := \mathbb{E}_{\prod_{i=t}^{H}\pi(a_{i+1}|s_{i+1})\mathcal{P}(s_{i+1}|s_i,a_i)}\left[\sum_{i=t}^{H} \gamma^{i-t} r_i\right], \tag{3.2}$$

where $\mu_\pi(s)$ is the (discounted) state distribution under π, i.e., the probability of visiting state s under π, and $\gamma \in [0,1)$ is the discount factor which assigns weights to rewards at different timesteps. $Q^\pi(s,a)$ is the action-state value function (or Q-function) which is the expected return obtained by executing a in state s and then following π afterwards. In tabular MDPs, the Q-function can be stored in a table, commonly referred as Q-table, and in stationary infinite-horizon MDPs the greedy policy $\pi^*(a|s) = \arg\max_a Q^*(s,a)$ is always optimal [Puterman, 1994]. However, the Q-function of the current policy is initially unknown and has to be learned.

Q-learning. For tabular MDPs, Q-learning by Watkins and Dayan [1992] was one of the most important breakthroughs in RL, and it is still at the core of recent successfull algorithms. At each training step i, the agent chooses an action according to a *behavior* policy $\beta(a|s)$, i.e., $a_t \sim \beta(\cdot|s_t)$, and then updates the Q-table according to

$$Q^\pi_{i+1}(s_t, a_t) = Q^\pi_i(s_t, a_t) + \eta\delta(s_t, a_t, s_{t+1}) \tag{3.3}$$

$$\delta(s_t, a_t, s_{t+1}) = \begin{cases} r_t + \gamma \max_a Q^\pi_i(s_{t+1}, a) - Q^\pi_i(s_t, a_t) & \text{if } s_t \text{ is non-terminal} \\ r_t - Q^\pi_i(s_t, a_t) & \text{otherwise} \end{cases} \tag{3.4}$$

where η is the learning rate and $\delta(s, a, s')$ is the *temporal difference (TD) error*. The blue term is often referred to as *TD target*. This approach is called *off-policy* because the policy $\beta(s, a)$ used to explore the environment is different from the target greedy policy $\pi(a|s)$.

The behavior policy and the "exploration-exploitation dilemma". As discussed in Section 4.1, an important challenge in designing the behavior policy is to account the exploration-exploitation dilemma. *Exploration* is a long-term endeavour where the agent tries to maximize the possibility of finding better rewards in states not yet visited. On the contrary, *exploitation* maximizes the expected rewards according to the current knowledge of the environment. Typically, in continuous action spaces Gaussian noise is added to the action, while in discrete spaces actions are chosen ϵ-greedily, i.e., optimally with probability $1-\epsilon$ and randomly with probability ϵ. In this case, the exploration-exploitation dilemma is simply addressed by having larger noise and ϵ at the beginning, and then decaying them over time. This naive dithering exploration is extremely inefficient in MDPs with sparse rewards, especially if some of them are "distractors", i.e., rewards that create suboptimal modes of the objective function. In the the remainder of this section, we review more sophisticatd approaches addressing exploration with sparse rewards. We first show theoretically grounded ideas for tabular or small MDP settings which are generally computationally intractable for large MDPs. We then give an overview of methods which try to apply similar principles to large domains by making approximations.

3.2.2 Related Work

Since finding high value states is crucial for successful RL, a large body of work has been devoted to efficient exploration in the past years. We begin by discussing an optimal solution to exploration, then continue with confidence-bound-based approaches. Then, we discuss methods based on intrinsic motivation, and describe posterior optimality value-distribution-based methods. We conclude by stating the main differences between our approach and these methods.

Optimal solution to exploration. In this paper, we consider model-free RL, i.e., the agent does not know the MDP dynamics. It is well-known that in model-free RL the optimal solution to exploration can be found by a Bayesian approach. First, assign an initial (possibly uninformative) prior distribution over possible unknown MDPs. Then, at each time step the posterior belief distribution over possible MDPs can be computed using the Bayes' theorem based on the current prior distribution, executed action and made observation. The action that optimally explores is then found by planning over possible future posterior distributions, e.g., using tree search, sufficiently far forward. Unfortunately, optimal exploration is intractable even for very small tasks [Szepesvari, 2010, Strens, 2000, Poupart et al., 2006]. The worst-case computational effort is exponential with respect to the planning horizon in tabular MDPs, and can be even more challenging with continuous state-action spaces.

Optimism. In tabular MDPs, many of the provably efficient algorithms are based on optimism in the face of uncertainty (OFU) [Lai and Robbins, 1985]. In OFU, the agent acts greedily with respect to an optimistic action value estimate composed of the value estimate and a bonus term that is proportional to the uncertainty of the value estimate. After executing an optimistic action, the agent then either experiences a high reward learning that the value of the action was indeed high, or the agent experiences a low reward and learns that the action was not optimal. After visiting a state-action pair, the exploration bonus is reduced. This approach is superior to naive approaches in that it avoids actions where low value and low information gain are possible. Under the assumption that the agent can visit every state-action pair infinitely many times, the overestimation will decrease and almost optimal behavior can be obtained [Kakade, 2003]. Most algorithms are optimal up to a polynomial amount of states, actions, or horizon length. RL literature provides many variations of these algorithms which use bounds with varying efficacy or different simplifying assumptions, such as [Kearns and Singh, 2002, Brafman and Tennenholtz, 2002, Kakade, 2003, Auer and Ortner, 2007, Jaksch et al., 2010, Dann and Brunskill, 2015].

Upper confidence bound. Perhaps the best well-known OFU algorithm is the upper confidence bound (UCB) algorithm [Auer et al., 2002]. UCB chooses actions based on a bound computed from visitation counts: the lower the count, the higher the bonus. Recently, Jin et al. [2018] proved that episodic Q-learning with UCB exploration converges to a regret proportional to the square root of states, actions, and timesteps, and square root of the cube of the episode length. Dong et al. [2019] gave a similar proof for infinite horizon MDPs with sample complexity proportional to the number of states and actions. In our experiments, we compare our approach to Q-learning with UCB exploration as defined by Auer et al. [2002].

Intrinsic motivation. The previously discussed algorithms are often computationally intractable, or their guarantees no longer apply in continuous state-action spaces, or when the state-action spaces are too large. Nevertheless, these provably efficient algorithms provided inspiration for more practical algorithms. Commonly, exploration is conducted by adding a bonus reward, also called auxiliary reward, to interesting states. This is often referred to as intrinsic motivation [Ryan and Deci, 2000]. For example, the bonus can be similar to the UCB [Strehl and Littman, 2008, Bellemare et al., 2016], thus encouraging actions of high uncertainty or of low visitation count.
Other forms of bonus are based on the prediction error of some quantity. For example, the agent may learn a model of the dynamics and try to predict the next state [Stadie et al., 2015, Houthooft et al., 2016, Pathak et al., 2017, Schmidhuber, 1991, 2006]. By giving a bonus proportional to the prediction error, the agent is incentivized to explore unpredictable states. Unfortunately, in the presence of noise, unpredictable states are not necessarily interesting states. Recent work addresses this issue by training an ensemble of models [Pathak et al., 2019] or predicting the outputs of a neural network [Burda et al., 2019].

As we will discuss in Section 3.3.2, our approach has similarities with the use of auxiliary reward. However, the use of auxiliary rewards can be inefficient for long-horizon exploration. Our approach addresses this issue by using a decoupled long-horizon exploration policy and, as the experiments in Section 3.4 show, it outperforms auxiliary-reward-based approaches [Strehl and Littman, 2008].

Thompson sampling. Another principled well-known technique for exploration is Thompson sampling [Thompson, 1933]. Thompson sampling samples actions from a posterior distribution which specifies the probability for each action to be optimal. Similarly to UCB-based methods, Thompson sampling is guaranteed to converge to an optimal policy in multi-armed bandit problems [Kaufmann et al., 2012, Agrawal and Goyal, 2013], and has shown strong empirical

performance [Scott, 2010, Chapelle and Li, 2011]. For a discussion of known shortcomings of Thompson sampling, we refer to [Russo and Van Roy, 2014, Russo et al., 2017, 2018].
Inspired by these successes, recent approaches have tried to follow the principle of Thompson sampling in Q-learning [Osband et al., 2013, 2019, D'Eramo et al., 2019]. These approaches (implicitly) assume that an empirical distribution over Q-functions – an ensemble of randomized Q-functions – is similar to the distribution over action optimalities used in Thompson sampling. Actions are therefore sampled from such ensemble. Since the Q-functions in the ensemble become similar when updated with new samples, there is a conceptual similarity with the action optimality distribution used in Thompson sampling for which variance also decreases with new samples. While there are no general proofs for the randomized Q-function approaches, Osband et al. [2019] proved a bound on the Bayesian regret of an algorithm based on randomized Q-functions in a tabular time-inhomogeneous MDP with a transition kernel drawn from a Dirichlet prior, providing a starting point for more general proofs.
Other exploration methods approximating a distribution over action optimalities include for example the work of Fortunato et al. [2017] and Plappert et al. [2018], who apply noise to the parameters of the model. This may be interpreted as approximately inferring a posterior distribution [Gal and Ghahramani, 2016], although this is not without contention [Osband, 2016]. Osband et al. [2016a] more directly approximates the posterior over Q-functions through bootstrapping [Efron, 1982]. The lack of a proper prior is an issue when rewards are sparse, as it causes the uncertainty of the posterior to vanish quickly. Osband et al. [2019] and Osband et al. [2018] try to address this by enforcing a prior via regularization or by adding randomly initialized but fixed neural network on top of each ensemble member.
Methods based on posterior distributions have yielded high performance in some tasks. However, because of the strong approximations needed to model posterior optimality distributions instead of maintaining visitation counts or explicit uncertainty estimates, these approaches may often have problems in assessing the state uncertainty correctly. In our experiments, the proposed approach outperforms state-of-the-art methods based on approximate posterior distributions, namely the algorithms proposed by Osband et al. [2016a, 2019] and D'Eramo et al. [2019].

Above, we discussed state-of-the-art exploration methods that (1) use confidence bounds or a distribution over action optimalities to take exploration into account in a principled way, (2) modify the reward by adding an intrinsic reward signal to encourage exploration, and (3) approximate a posterior distribution over value functions. The main challenge that pertains to all these methods is that they do not take long-term exploration into account explicitly. *Contrary to this, similarly to how value iteration propagates state values, we propose an approach that uses dynamic programming to explicitly propagate visitation information backwards in time.* This allows our approach to efficiently find the most promising parts of the state space that have not been visited before, and avoid getting stuck in suboptimal regions that at first appear promising.

3.3 Long-Term Visitation Value for Deep Exploration

In this section, we present our novel off-policy method for improving exploration 1with sparse and distracting rewards. We start by motivating our approach with a toy example, showing the limitation of current count-based algorithms and the need for "long-term visitation counts". Subsequently, we formally define the proposed method, and show how it solves the toy example.

3.3.1 Example of the Limitations of Existing Immediate Count Methods

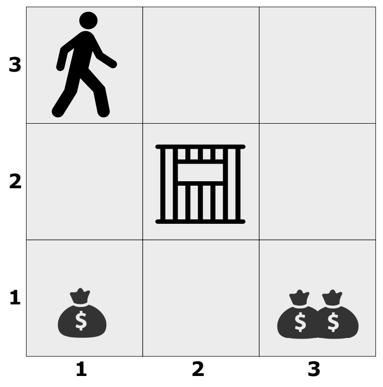


Figure 3.2: Toy problem domain.

Consider the toy problem of an agent moving in a grid, shown in Figure 3.2. Reward is 0 everywhere except in $(1, 1)$ and $(3, 1)$, where it is 1 and 2, respectively. The reward is given for executing an action in the state, i.e., on transitions from reward states. The agent initial position is fixed in $(1, 1)$ and the episode ends when a reward is collected or after five steps. Any action in the prison cells $(2, 2)$ has only an arbitrarily small probability of success. If it fails, the agent stays in $(2, 2)$. In practice, the prison cell almost completely prevents any further exploration. Despite its simplicity, this domain highlights two important challenges for exploration in RL with sparse rewards. First, there is a locally optimal policy collecting the lesser reward, which acts as distractor. Second, the prison state is particularly dangerous as it severely hinders exploration.

Assume that the agent has already explored the environment for four episodes performing the following actions: (1) left, down, left, down, down (reward 1 collected); (2) up, down, right (fail), right (fail), right (fail); (3) right, left, right, down, down (fail); (4) right, down, left (fail), up (fail), right (fail). The resulting state-action visitation count is shown in Figure 3.3a. Given this data, we initialize $Q^\pi(s, a)$ to zero, train it with Q-learning with $\gamma = 0.99$ until convergence, and then derive three greedy policies. The first simply maximizes $Q^\pi(s, a)$. The second maximizes the behavior $Q^\beta(s, a)$ based on UCB1 [Auer et al., 2002], defined as

$$Q^\beta(s, a) = Q^\pi(s, a) + \kappa \sqrt{\frac{2 \log \sum_{a_j} n(s_t, a_j)}{n(s_t, a)}}, \tag{3.5}$$

where $n(s, a)$ is the state-action count, and κ is a scaling coefficient[1]. The third maximizes an augmented Q-function $Q^+(s, a)$ trained by adding the intrinsic reward based on the immediate count [Strehl and Littman, 2008], i.e.,

$$r_t^+ = r_t + \alpha\, n(s_t, a_t)^{-1/2}, \tag{3.6}$$

where α is a scaling coefficient set to $\alpha = 0.1$ as in [Strehl and Littman, 2008, Bellemare et al., 2016]. Figure 3.3a shows the state-action count after the above trajectories, and Figures 3.3b, 3.3c, and 3.3d depict the respective Q-functions (colormap) and policies (arrows). Given the state-action count, consider what the agent has done and knows so far.

a) It has executed all actions at least once in (1,3) and (2,2) (the prison cell),
b) It still did not execute "up" in (1,2),
c) It visited (1,3) once and experienced the reward of 1, and
d) It still did not execute "up" and "right" in (2,1).

[1] In classic UCB, the reward is usually bounded in $[0, 1]$ and no coefficient is needed. This is not usually the case for MDPs, where Q-values can be larger/smaller than 1/0. We thus need to scale the square root with $\kappa = (r_{\max} - r_{\min})/(1 - \gamma)$, which upper-bounds the difference between the largest and smallest possible Q-value.

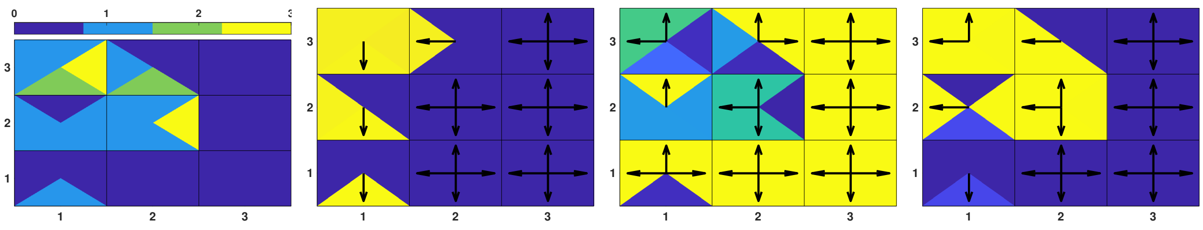

(a) State-action count (b) Greedy (c) UCB1 (d) Greedy (with bonus)

Figure 3.3: Toy example (part 1). Visitation count ((a)) and behavior policies ((b), (c), (d)) for the toy problem below. Cells are divided into four triangles, each one representing the count ((a)) or the behavior Q-value ((b), (c), (d)) for the actions "up / down / left / right" in that cell. Arrows denote the action with the highest Q-value. None of the behavior policies exhibits long-term deep exploration. In (1,3) both UCB1 ((c)) and the augmented Q-table ((d)) myopically select the action with the smallest immediate count. The latter also acts badly in (1,2) and (2,3). There, it selects "left" which has count one, instead of "up" which has count zero.

In terms of exploration, the best decisions would be to select actions not executed yet, such as "up" in (1,2). *But what should the agent do in states where it has already executed all actions at least once? Ideally, it should select actions driving it to unvisited states, or to states where it has not executed all actions yet.* However, none of the policies above exhibits such behavior.

a) The greedy policy (Figure 3.3b), as expected, points to the only reward collected so far.

b) The UCB1 policy (Figure 3.3c) yields Q-values of much larger magnitude. Most of the actions have been executed few times, thus their UCB is larger than their Q-value. The policy correctly selects unexecuted actions in (1,2) and (2,3), but fails (1,3). There it also selects the least executed actions ('up" and "left") which, however, have already been executed once. The agent should know that these actions do not bring it anywhere (the agent hits the environment boundaries and does not move) and should avoid them. However, it selects them because the policy $Q^\beta(s, a)$ is based on the *immediate* visitation count. Only when the count will be equal for all actions, the policy will select a random action. This behavior is clearly myopic and extremely inefficient.

c) The policy learned using the auxiliary bonus (Figure 3.3d) is even worse, as it acts badly not only in (1,3), but also in (1,2) and (2,3). There, it chooses "left", which was already selected once, instead of "up" (or "right in (2,3)), which instead has not been selected yet. The reason of such behavior is that this auxiliary reward requires optimistic initialization [Strehl and Littman, 2008]. The agent, in fact, is positively rewarded for simply executing any action, with value proportional to the visitation count. However, if the Q-function is initialized to zero, the positive feedback will make the agent believe that the action is good. This mechanism encourages the agent to execute the same action again and hinders exploration. This problem is typical of all methods using an auxiliary reward based on visitation count, unless optimistic initialization of the Q-table is used [Strehl and Littman, 2008]. More recent methods use more sophisticated rewards [Jin et al., 2018, Dong et al., 2019]. However, despite their strong theoretical convergence guarantees, for large size problems these methods may require an impractical number of samples and are often outperformed by standard algorithms.

Despite its simplicity, the above toy problem highlights the limitations of current exploration strategies based on the immediate count. More specifically, (1) considering the immediate count may result in shallow exploration, and (2) directly augmenting the reward to train the Q-function, i.e., coupling exploration and exploitation, can have undesired effects due to the non-stationarity of the augmented reward function. To address these limitations, we propose an approach that (1) plans exploration actions far into the future by using a *long-term visitation count*, and (2) decouples exploration and exploitation by learning a separate function assessing the *exploration value* of the actions. Contrary to existing methods which use models of reward and dynamics [Hester and Stone, 2013], which can be hard to come by, our approach is off-policy and model-free.

Visitation value. The key idea of our approach is to train a visitation value function using the temporal difference principle. The function, called W-function and denoted by $W^\beta(s,a)$, approximates the cumulative sum of an intrinsic rewards r^W based on the visitation count, i.e.,

$$W^\beta(s_t,a_t) := \mathbb{E}_{\prod_{i=t}\beta(a_{t+1}|s_{t+1})\mathcal{P}(s_{i+1}|s_i,a_i)}\left[\sum_{i=t}^{H}\gamma_w^{i-t}r_i^W\right], \tag{3.7}$$

where γ_w is the visitation count discount factor. The higher the discount, the more in the future the visitation count will look. In practice, we estimate $W^\beta(s,a)$ using TD learning on the samples produced using the behavior policy β. Therefore, we use the following update

$$W^\beta_{i+1}(s_t,a_t) = W^\beta_i(s_t,a_t) + \eta\delta^W(s_t,a_t,s_{t+1}), \tag{3.8}$$

where η is the learning rate, and $\delta^W(s_t,a_t,s_{t+1})$ is the W-function TD error, which depends on the reward r^W (can be either maximized or minimized). Subsequently, following the well-known upper confidence bound (UCB) algorithm [Lai and Robbins, 1985], the behavior policy is greedy with respect to the sum of the Q-function and an upper confidence bound, i.e.,

$$\beta(a|s) = \arg\max_a\left\{Q^\pi(s,a) + \kappa U_W(s,a)\right\}, \tag{3.9}$$

where $U_W(s,a)$ is an upper confidence bound based on $W^\beta(s,a)$, and κ is the same exploration constant used by UCB1 in Eq. (3.5).

This proposed approach (1) considers long-term visitation count by employing the W-function, which is trained to maximize the cumulative – not the immediate – count. Intuitively, the W-function encourages the agent to explore state-actions which have not been visited before. Given the current state s, in fact, $W^\beta(s,a)$ specifies how much exploration we can expect on (discount-weighted) average in future states if we choose action a. Furthermore, since the Q-function is still trained greedily with respect to the extrinsic reward as in Q-learning, while the W-function is trained greedily on the intrinsic reward and favors less visited states, our approach (2) effectively decouples exploration and exploitation. This allows us to more efficiently prevent underestimation of the visitation count exploration term. Below, we propose two versions of this approach based on two different rewards r^W and upper bounds $U_W(s,a)$.

W-function as long-term UCB. The visitation reward is the UCB of the state-action count at the current time, i.e.,

$$r_t^W = \begin{cases} \sqrt{\frac{2 \log \sum_{a_j} n(s_t, a_j)}{n(s_t, a_t)}} & \text{if } s_t \text{ is non-terminal} \\ \frac{1}{1-\gamma_w} \sqrt{\frac{2 \log \sum_{a_j} n(s_t, a_j)}{n(s_t, a_t)}} & \text{otherwise.} \end{cases} \tag{3.10}$$

Notice how we distinguish between terminal and non-terminal states. As terminal state-action pairs prevent exploration, we need to avoid them after the first visit. Therefore, we assume that for terminal state-action pairs the agent gets the same visitation reward at each time step forever. This W-function represents the discount-weighted average UCB along the trajectory, i.e.,

$$W_{\text{UCB}}^{\beta}(s_t, a_t) = \sum_{i=t}^{H} \gamma_w^{i-t} \sqrt{\frac{2 \log \sum_{a_j} n(s_i, a_j)}{n(s_i, a_i)}}, \tag{3.11}$$

and its TD error is

$$\delta^W(s_t, a_t, s_{t+1}) = \begin{cases} r_t^W + \gamma_w \max_a W_{\text{UCB},i}^{\beta}(s_{t+1}, a) - W_{\text{UCB},i}^{\beta}(s_t, a_t) & \text{if } s_t \text{ is non-terminal} \\ r_t^W - W_{\text{UCB},i}^{\beta}(s_t, a_t) & \text{otherwise.} \end{cases} \tag{3.12}$$

This TD error has two important consequences. First, the target of non-terminal states will be higher than the one of terminal states despite the different visitation reward (Eq. (3.10)), given proper initialization of W_{UCB}^{β}. We will discuss this in more detail later in this section. Second, since we can only update the W-function approximately based on a limited number of samples, it is beneficial to overestimate its target with the max operator. While it is known that in highly stochastic environments the overestimation can degrade the performance of value-based algorithm [van Hasselt, 2010, D'Eramo et al., 2017], it has been shown that underestimation does not perform well when the exploration is challenging [Tateo et al., 2017], and for a wide class of environments, overestimation allows finding the optimal solution due to self-correction [Sutton and Barto, 2018]. Assuming identical visitation over state-action pairs, the W-function becomes the discounted immediate UCB, i.e.,

$$W_{\text{UCB}}^{\beta}(s_t, a_t) = \frac{1}{1-\gamma_w} \sqrt{\frac{2 \log \sum_{a_j} n(s_t, a_j)}{n(s_t, a_t)}}, \tag{3.13}$$

thus, we can use $W_{\text{UCB}}^{\beta}(s, a)$ directly in the behavior policy

$$\beta(a_t|s_t) = \arg\max_a \left\{ Q^{\pi}(s_t, a) + \kappa \underbrace{(1-\gamma_w) W_{\text{UCB}}^{\beta}(s_t, a)}_{U_W^{\text{UCB}}(s,a)} \right\}, \tag{3.14}$$

In Section 3.3.3, we discuss how to initialize $W^{\beta}_{\text{UCB}}(s,a)$. Finally, notice that when $\gamma_w = 0$, Eq. (3.13) is the classic UCB, and Eq. (3.14) is equivalent to the UCB1 policy in Eq. (3.5).

W-function as long-term count. This W-function approximates the total count over a path. Its visitation reward and TD error are

$$r^W_t = \begin{cases} n(s_t, a_t) & \text{if } s_t \text{ is non-terminal} \\ \frac{1}{1-\gamma_w} \max_{(s,a)} n(s,a) & \text{otherwise,} \end{cases} \tag{3.15}$$

$$\delta^W(s_t, a_t, s_{t+1}) = \begin{cases} r^W_t + \gamma_w \min_a W^{\beta}_{\text{N},i}(s_{t+1}, a) - W^{\beta}_{\text{N},i}(s_t, a_t) & \text{if } s_t \text{ is non-terminal} \\ r^W_t - W^{\beta}_{\text{N},i}(s_t, a_t) & \text{otherwise.} \end{cases} \tag{3.16}$$

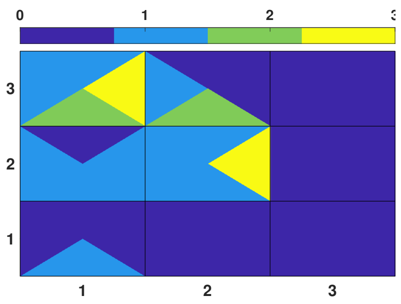
Similarly to the UCB-based reward discussed above, we distinguish between terminal and non-terminal states. Since terminal states prevent any further exploration, we penalize them by using an upper bound of the state-action count as visitation reward. Regarding the TD target, the min operator now replaces the max operator of Eq. (3.12), because to encourage exploration we need to prioritize low-count state-action pairs. This yields an optimistic estimate of the W-function, because the lower the W-function the more optimistic the exploration is. Assuming identical visitation over state-action pairs, the W-function becomes the discounted cumulative count, i.e.,

$$W^{\beta}_{\text{N}}(s_t, a_t) = \frac{1}{1-\gamma_w} n(s_t, a_t). \tag{3.17}$$

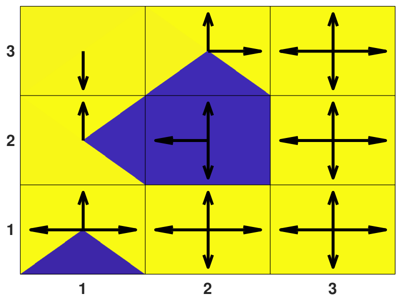
We then compute the pseudocount $\hat{n}(s_t, a_t) = (1-\gamma_w) W^{\beta}_{\text{N}}(s_t, a_t)$ and use it in the UCB policy

$$\beta(a_t|s_t) = \arg\max_a \left\{ Q^{\pi}(s_t, a) + \kappa \underbrace{\sqrt{\frac{2 \log \sum_{a_j} \hat{n}(s_t, a_j)}{\hat{n}(s_t, a)}}}_{U^{\text{N}}_{W}(s,a)} \right\}. \tag{3.18}$$

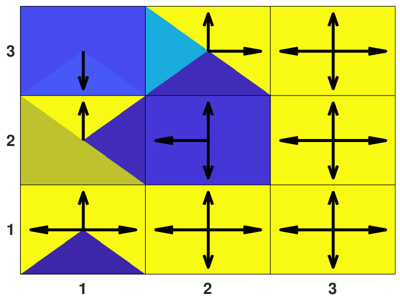
When the counts are zero, $W^{\beta}_{\text{N}}(s,a) = 0$, thus we need to initialize it to zero. We discuss how to avoid numerical problem in the square root in Section 3.3.3. Finally notice that when $\gamma_w = 0$, Eq. (3.18) is equivalent to the UCB1 policy in Eq. (3.5).



(a) State-action count



(b) UCB W-function policy



(c) Count W-function policy

Figure 3.4: Toy example (part 2). Visitation count ((a)) and behavior policies derived from the proposed W-functions ((b), (c)). Due to the low count, the exploration upper bound dominates the greedy Q-value, and the proposed behavior policies successfully achieve long-term exploration. Both policies prioritize unexecuted actions, and avoid terminal and prison states.

Example of Long-Term Exploration with the W-function. Consider again the toy problem presented in Section 3.3.1. Figure 3.4 shows the behavior policies of Eq. (3.14) and Eq. (3.18) with $\gamma_w = 0.99$. The policies are identical and both drive the agent to unvisited states, or to states where it has not executed all actions yet, successfully exhibiting long-term exploration. The only difference is the magnitude of the state-action pair values (the colormap, especially in unvisited states) due to their different initialization.

a) In (1,3), they both select "down" despite "up" and "left" having lower count. The agent, in fact, knows that from (1,2) it can go "down" to (1,1) where it executed only one action. Thus, it knows that there it can try new actions. On the other hand, the only thing it knows about (2,3) is that it leads to the prison state, where the agent already executed all actions. Thus, (1,2) has higher long-term exploration value than (2,3).
b) In (1,2) they both select "up", overturning the greedy action "up" (Figure 3.3b). This is an important aspect, because in order to explore the environment we always need to prioritize unexecuted actions.
c) The same happens in (2,3), where actions with count zero are selected, rather than the greedy ones.
d) None of the policies lead the agent to the prison state, because it has already been visited and all actions have been executed in there.
e) They both do not select "right" in (2,2) because it has been executed one more time than other actions. Nonetheless, the difference in the action value is minimal, as shown by the action colormap (almost uniform).

3.3.3 W-function Initialization and Bounds

In UCB bandits, it is assumed that each arm/action will be executed once before actions are executed twice. In order to enforce this, we need to make sure that the upper confidence bound $U_W(s, a)$ is high enough so that an action cannot be executed twice before all other actions are executed once. Below, we discuss how to initialize the W-functions and to set bounds in order achieve the same behavior. For the sake of simplicity, below we drop subscripts and superscripts, i.e., we write $Q(s, a)$ in place of $Q^\pi(s, a)$, $W(s, a)$ in place of $W^\beta_{\text{UCB}}(s, a)$ in the first part, and $W(s, a)$ in place of $W^\beta_{\text{N}}(s, a)$ in the second part.

W^{β}_{ucb} **initialization.** To compute the initialization value we consider the worst-case scenario, that is, in state s all actions a but $\bar{a}$ have been executed. In this scenario, we want that $W(s,\bar{a})$ is an upper bound of $W(s,a)$. Following Eq. (3.14) and (3.12), we have

$$Q_{\max} + \kappa(1-\gamma_w)W(s,\bar{a}) = Q_{\max} + \kappa(1-\gamma_w)\left(\sqrt{2\log(|\mathcal{A}|-1)} + \gamma_w \max_a W(s',a)\right), \tag{3.19}$$

where the blue term is the TD target for non-terminal states, and $\sqrt{2\log(|\mathcal{A}|-1)}$ is the immediate visitation reward for executing $\bar{a}$ in the worst-case scenario (all other $|\mathcal{A}|-1$ actions have been executed, and $\hat{a}$ has not been executed yet). In this scenario, $W(s,\bar{a})$ is an upper bound of $W(s,a)$ under the assumption of uniform count, i.e.,

$$\forall (s,a) \neq (s,\bar{a}), \quad n(s,a) = \bar{n}(s) \Rightarrow W(s,\bar{a}) \geq W(s,a). \tag{3.20}$$

We can then write Eq. (3.19) as

$$Q_{\max} + \kappa(1-\gamma_w)W(s,\bar{a}) = Q_{\max} + \kappa(1-\gamma_w)\left(\sqrt{2\log(|\mathcal{A}|-1)} + \gamma_w W(s,\bar{a})\right). \tag{3.21}$$

In order to guarantee to select $\bar{a}$, which has not been select yet, we need to initialize $W(s,\bar{a})$ such that the following is satisfied

$$Q_{\min} + \kappa(1-\gamma_w)W(s,\bar{a}) > Q_{\max} + \kappa(1-\gamma_w)\left(\sqrt{2\log(|\mathcal{A}|-1)} + \gamma_w W(s,\bar{a})\right), \tag{3.22}$$

where $Q_{\min}$ denotes the smallest possible Q-value. We get

$$\begin{aligned} W(s,\bar{a}) &> \frac{Q_{\max}-Q_{\min}}{\kappa(1-\gamma_w)} + \sqrt{2\log(|\mathcal{A}|-1)} + \gamma_w W(s,\bar{a}) \\ W(s,\bar{a}) &> \frac{Q_{\max}-Q_{\min}}{\kappa(1-\gamma_w)^2} + \frac{\sqrt{2\log(|\mathcal{A}|-1)}}{(1-\gamma_w)}. \end{aligned} \tag{3.23}$$

Initializing the W-function according to Eq. (3.23) guarantees to select $\bar{a}$.

W^{β}_{n} **upper bound.** Since this W-function represents the discounted cumulative count, it is initialized to zero. However, numerical problems may occur in the square root of Eq. (3.18) because the pseudocount can be smaller than one or even zero. In these cases, the square root of negative numbers and the division by zero are not defined. To address these issues, we add +1 inside the logarithm and replace the square root with an upper bound when $\hat{n}(s,a) = 0$. Similarly to the previous section, the bound needs to be high enough so that an action cannot be executed twice before all other actions are executed once. Following Eq. (3.18), we need to ensure that

$$\begin{aligned} Q_{\min} + \kappa U_W(s,\bar{a}) &> Q_{\max} + \kappa U_W(s,a), \\ U_W(s,\bar{a}) &> \frac{Q_{\max}-Q_{\min}}{\kappa} + U_W(s,a). \end{aligned} \tag{3.24}$$

However, this time assuming uniform count does not guarantee a uniform *pseudocount*, i.e.,

$$\forall (s,a) \neq (s,\bar{a}), \quad n(s,a) = \bar{n}(s) \nRightarrow W(s,\bar{a}) \geq W(s,a). \tag{3.25}$$

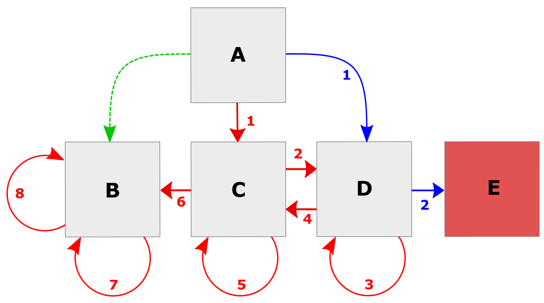


Figure 3.5: [
Chainworld with uniform count.]Chainworld with uniform count.

To illustrate this issue, consider the chain MDP in Figure 3.5. The agent always starts in A and can move "left", "right", and "down". The episode ends after eight steps, or when terminal state E is reached. The goal of this example is to explore all states uniformly. In the first episode, the agent follows the blue trajectory (A, D, E). In the second episode, it follows the red trajectory (A, C, D, D, C, B, B, B). In both trajectories, an action with count zero was always selected first. At the beginning of the third episode, all state-action pairs have been executed once, except for "left" in A, i.e., $n(s,a) = 1 \ \forall (s,a) \neq (A, left), \ n(A, left) = 0$. Thus, we need to enforce the agent to select it. However, the discounted *cumulative* count $\hat{n}(s,a) = (1-\gamma_w)W(s,a)$ is *not* uniform. For example with $\gamma_w = 0.9$, $\hat{n}(A, right) = 3.8$ and $\hat{n}(A, down) = 4.52$.

More in general, the worst-case scenario we need to consider to satisfy Eq. (3.24) is the one where (1) action $\bar{a}$ has not been executed yet, (2) an action $\dot{a}$ has been executed once and has the smallest pseudocount, and (3) all other actions have been executed once and have the highest pseudocount. In this scenario, to select $\bar{a}$ we need to guarantee $U_W(s,\bar{a}) > U_W(s,\dot{a})$. Since we assume that counts are uniform and no action action has been executed twice, the smallest pseudocount is $\hat{n}(s,\dot{a}) = 1-\gamma_w$, while the highest is $\hat{n}(s,a) = 1$. Then, Eq. (3.24) becomes

$$\begin{aligned} U_W(s,\bar{a}) &> \frac{Q_{\max} - Q_{\min}}{\kappa} + U_W(s,\dot{a}) \\ U_W(s,\bar{a}) &> \frac{Q_{\max} - Q_{\min}}{\kappa} + \sqrt{\frac{2\log\left((1-\gamma_w) + |\mathcal{A}| - 2\right)}{1-\gamma_w}}. \end{aligned} \tag{3.26}$$

Replacing the square root in Eq. (3.18) with the above upper bound when $n(s,\bar{a}) = 0$ guarantees to select $\bar{a}$.

Example of Behavior Under Uniform Count. Consider our toy problem again. We distinguish two cases: (1) all state-action pairs have been executed once except for "left" in (1,3), and (2) the count is one everywhere. The Q-table is optimal for visited state-action pairs, thus the greedy policy simply brings the agent to (3,1) where the highest reward lies. Figure 3.6 shows baseline and proposed behavior policies in case (1). In case (2), all policies look like the greedy one (Figure 3.6a).

a) The augmented reward policy (Figure 3.6b) has no interest in trying "left" in (1,3), and its value is the lowest. The agent, in fact, cannot get the visitation bonus without first executing the action. This shows once more that this passive way of exploration is highly inefficient.

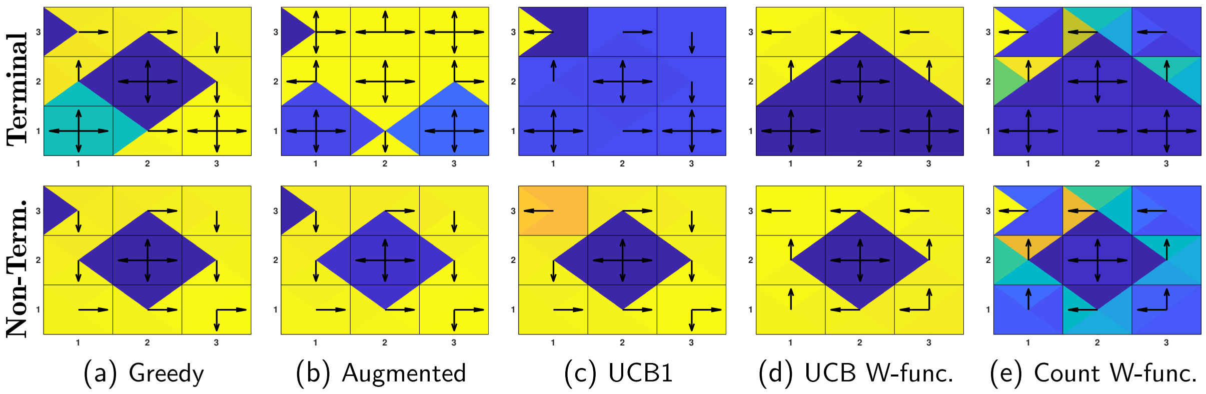


Figure 3.6: Toy example (part 3). Behavior policies under uniform visitation count. In the top row, reward states (1,1) and (3,1) are terminal. In the bottom row, they are not and the MDP has infinite horizon. All state-action pairs been executed once, except for "left" in (1,3). Only the proposed methods ((d), (e)) correctly indentify such state-action pair as the most interesting for exploration, and propagate this information to other states thanks to TD learning. When the action is executed, all policies converge to the greedy one ((a)).

b) UCB1 policy (Figure 3.6c) acts greedily everywhere except in (1,3), where it correctly selects the unexecuted action. However, since UCB1 considers only the *immediate* visitation, it has no way of propagating this information to other state-action pairs.

c) By contrast, for the proposed W-function policies (Figure 3.6d and 3.6e) the unexecuted state-action pair is the most interesting, and the policy brings the agent there from any state. Thanks to the separate W-function, we can propagate the value of unexceted actions to other states, always prioritizing the exploration of new state-action pairs. Under uniform count $n(s,a) = 1 \ \forall(s,a)$, then the agent acts greedily with respect to the Q-function.

3.3.4 Final Remarks

In this section, we discuss the uniform count assumption used to derive the bounds, the differences between the proposed W-functions and intrinsic motivation, the benefit of using the count-based one, and potential issues in stochastic environments.

Uniform count assumption. In Section 3.3.3 we derived an initialization for W^{β}_{UCB} and a bound for W^{β}_{N} to guarantee that an action is not executed twice before all other actions are executed once. For that, we assumed the visitation count to be uniform for all state-action pairs. We acknowledge that this assumption is not true in practice. First, it may be necessary to revisit old states in order to visit new ones. Second, when an episode is over the state is reset and a new episode starts. This is common practice also for infinite horizon MDPs, when usually the environment is reset after some steps. Thus, the agent initial state will naturally have higher visitation count. Nonetheless, in Section 3.4.2, we will empirically show that our approach allows the agent to explore the environment as uniformly as possible.

W-function vs auxiliary rewards. Using an auxiliary reward such as an exploration bonus or an intrinsic motivation signal has similarities with the proposed approach, especially with W^{β}_{UCB}, but the two methods are not equivalent. Consider augmenting the reward with the same visitation reward used to train the W-function, i.e., $r_t^+ = r_t + \alpha\, r_t^W$. Using TD learning, the augmented Q-function used for exploration can be rewritten as[2]

$$Q^+(s_t, a_t) = r_t + \alpha r_t^W + \gamma \max_a Q^+(s_{t+1}, a). \tag{3.27}$$

Then consider the behavior policy derived from W_{UCB}^β in Eq. (3.14), and decompose it as well

$$Q^\pi(s_t, a_t) + \alpha W_{\text{UCB}}^\beta(s_t, a_t) = r_t + \gamma \max_a Q^\pi(s_{t+1}, a) + \alpha r_t^W + \alpha\gamma_w \max_a W_{\text{UCB}}^\beta(s_{t+1}, a), \tag{3.28}$$

where we considered $\alpha = \kappa(1 - \gamma_w)$. The red terms are equivalent, but the blue terms are not because of the max operator, since $\max_x\{f(x) + g(x)\} \neq \max_x\{f(x)\} + \max_x\{g(x)\}$.

This explicit separation between exploitation (Q) and exploration (W) has two important consequences. First, it is easy to implement the proposed behavior policy for any off-policy algorithm, as the exploration term is separated and independent from the Q-function estimates. This implies that we can propagate the exploration value independently from the action-value function estimate. With this choice we can select a discount factor γ_w which differs from the MDP discount γ. By choosing a high exploration discount factor γ_w, we do long-term exploration allowing us to find the optimal policy when exploration is crucial. By choosing a low exploration discount factor γ_w, instead, we perform short-term exploration which may converge faster to a greedy policy. In the evaluation in Section 3.4.2, we show experiments for which the discount factors are the same, as well as experiments where they differ. Furthermore, we investigate the effect of changing the W-function discount while keeping the Q-function discount fixed.
Second, auxiliary rewards are non-stationary, as the visitation count changes at every step. This leads to a non-stationary target for the Q-function. With our approach, by decoupling exploration and exploitation, we have stationary target for the Q-function while moving all the non-stationarity to the W-function.

Benefits of zero-initialization. Because of its initialization (Eq. (3.23)), the proposed W_{UCB}^β gives an optimistic estimate which is needed for efficient exploration. By contrast, W_{N}^β does not require any kind of optimistic initialization. This can be crucial for continuous domains where function approximation is necessary, and for which proper optimistic initialization is not always feasible [Osband et al., 2019]. In the experiments of Section 3.4, we show that the two W-functions attain almost equal results, making W_{N}^β very promising for solving non-tabular MDPs with sparse rewards. However, the evaluation in continuous domain is out of the scope of this paper, and will be pursued in future work.

Terminal states and stochasticity. The visitation rewards for training the W-functions penalize terminal state-action pairs (Eq. (3.10) and (3.15)). Consequently, once visited, the agent will avoid visiting them again. One may think that this would lead to poor exploration in stochastic environments, where the same state-action pair can lead to both terminal and non-terminal states. In this scenario, trying the same action again in the future may yield better exploration depending on the stochastic transition function. However, as we empirically show in Section 3.4.2, stochasticity in the transition function does not harm our approach.

[2] For the sake of simplicity, we consider only non-terminal states.

3.4 Evaluation

In this section, we evaluate the proposed method against state-of-the-art methods for exploration in model-free RL on several domains. The experiments are designed to highlight the challenges of learning with sparse rewards, and are split into two parts. In the first, we present the environments and we address (1) learning when only few states give a reward, (2) learning when a sequence of correct actions is needed to receive a reward, (3) exploring efficiently when few steps are allowed per training episode, (4) avoiding local optima and distracting rewards, and (5) avoiding dangerous states that stop the agent preventing exploration. In the second, we further investigate (6) the visitation count at convergence, (7) the empirical sample complexity, (8) the impact of the visitation value hyperparameter γ_w, (9) the performance in the infinite horizon scenario, and (10) with stochastic transition function.
To ensure reproducibility, the experiments are performed over fixed random seeds. For deterministic MDPs, the seeds are $1, 2, \ldots, 20$. For stochastic MDPs, the seeds are $1, 2, \ldots, 50$. Pseudocode and further details of the hyperparameters are given in Appendix C. Source code is available at `https://github.com/sparisi/visit-value-explore`

Baseline Algorithms. The evaluated algorithms all use Q-learning [Watkins and Dayan, 1992] and share the same hyperparameters, e.g., learning rate and discount factor. In most of the experiments, we use infinite replay memory as presented by Osband et al. [2019]. In Appendix C.6 we also present an evaluation without replay memory on some domains. The algorithms all learn two separate Q-tables, a behavior one for exploration, and a target one for testing the quality of the greedy policy. The difference among the algorithms is how the behavior policy performs exploration. In this regard, we compare against the following exploration strategies: random, ϵ-greedy, bootstrapped, count-based.
For bootstrapping, we evaluate the original version proposed by Osband et al. [2016a], as well as the more recent versions of D'Eramo et al. [2019] and Osband et al. [2019]. These methods all share the core idea, i.e., to use an ensemble of Q-tables, each initialized differently and trained on different samples, to approximate a distribution over Q-values via bootstrapping. The differences are the following. Osband et al. [2016a] select a random Q-table at the beginning of each episodes, and use it until the episode is over. Osband et al. [2019] further regularize the Q-tables using the squared ℓ_2-norm to "prior tables" representing a prior over the Q-function. D'Eramo et al. [2019] instead select the Q-table randomly within the ensemble at each step to approximate Thompson sampling, but do not use any prior.
For count-based algorithms, we compare against UCB1 exploration [Auer et al., 2002] on the immediate state-action count as in Eq. (3.5), and against augmenting the reward with the exploration bonus proposed by Strehl and Littman [2008] as in Eq. (3.6). Notice that Strehl and Littman [2008] apply the exploration bonus to a model-based algorithm, but the same bonus has since then been used by many model-free algorithms [Bellemare et al., 2016, Dong et al., 2019]. More recent methods use more sophisticated bonus to derive strong convergence guarantees [Jin et al., 2018, Dong et al., 2019]. However, for large size problems these methods may require an impractical number of samples and are often outperformed by standard algorithms.

3.4.1 Part 1: Discounted Return and States Discovered

We use the following environments: deep sea [Osband et al., 2018, 2019], taxi driver [Asadi and Littman, 2017], and four novel benchmark gridworlds. All the environments have sparse discounted rewards, and each has peculiar characteristics making it challenging to solve.

Evaluation Criteria. For each environment, we evaluate the algorithms on varying two different conditions: optimistic vs zero initialization of the behavior Q-tables, and short vs long horizon, for a total of four scenarios. For each of them, we report the expected discounted return of $\pi(a|s)$, which is greedy over $Q^\pi(s,a)$, and the number of visited states over training samples. In all environments the initial state is fixed.

Optimistic initialization [Lai and Robbins, 1985] consists of initializing the Q-tables to a large value. After visiting a state-action pair, either the agent experiences a high reward confirming his belief about its quality, or the agent experiences a low reward and learns that the action was not optimal and will not execute it again in that state. By initializing the Q-tables to the upper bound $r_{\max}/(1-\gamma)$ we are guaranteed to explore all the state-action pair even with just an ϵ-greedy policy. However, this requires prior knowledge of the reward bounds, and in the case of continuous states and function approximation the notion of an universally “optimistic prior” does not carry over from the tabular setting [Osband et al., 2019]. Thus, it is important that an algorithm is able to explore and to learn even with a simple zero or random initialization.

The length of the horizon is also an important factor for exploration. If the agent can perform a limited number of actions during an episode, it has to carefully choose them in order to explore as efficiently as possible. Prominent example in this regard are real-world tasks such as videogames, where the player has limited time to fulfill the objective before the game is over, or robotics, where the autonomy of the robot is limited by the hardware. Thus, an algorithm should be able to explore and learn even when the training episode horizon is short. In the “short horizon” scenario, we allow the agent to explore for a number of steps slightly higher than the ones needed to find the highest reward. For example, if the highest reward is eight steps away from the initial position, the “short horizon” is ten steps. The “long horizon” scenario is always twice as long.

Results are first presented in plots showing number of states discovered and discounted return against training steps averaged over 20 seeds. Due to the large number of algorithms, confidence intervals are not reported in plots. At the end of the section, a recap table summarizes the results and reports the success of an algorithm, i.e., the number of seeds the algorithm learned the optimal policy, and the percentage of states visited, both with 95% confidence interval.

Deep Sea

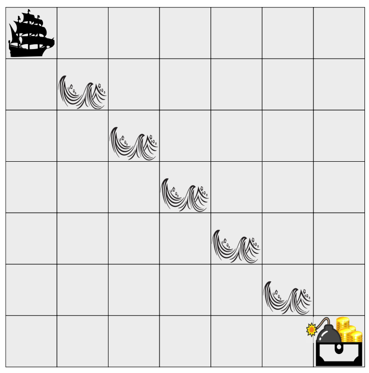


Figure 3.7: The deep sea.

In this environment (Figure 3.7), the agent (the ship) always starts at the top-left corner of a gridworld of size $N \times N$, and has to descend through the grid to open a chest, located in the bottom-right corner. At the beginning of the episode, either a treasure or a bomb appear inside the chest, with a 50-50. The outcome is known to the agent, as the state consists of the ship position and a flag denoting if the bomb or the treaure has appeared. At every step, the ship descends by one step, and must chose to turn left or right. If it hits the left boundary, it will descend without turning. For example, from the starting position $(1,1)$, doing “left” will move the ship to $(2,1)$, while doing “right” will move it to $(2,2)$. The episode ends when the ship reaches the bottom of the grid, and thus the horizon is fixed to N. The reward is 1 if the ship finds the treasure, -1 if it finds the bomb, and $-0.01/N$ if the ship does “right” in any

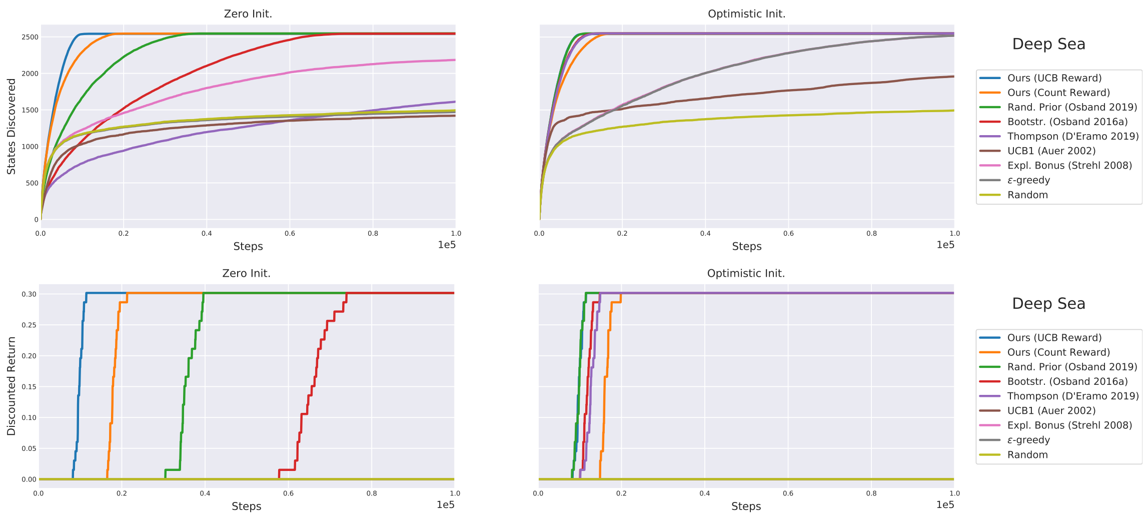


Figure 3.8: Deep sea results. Our method achieves the best results, followed by bootstrapping.

cell along the diagonal (denoted by sea waves). The optimal policy selects always "right" if the treasure has appeared, otherwise "left" at the initial state and then can act randomly.

The challenge of this environment lies in having to select the same action ("right") N times in a row in order to receive either 1 or -1. Doing "left" even only once prevents the ship from reaching the chest. At the same time, the agent is discouraged from doing "right", since doing so along the diagonal yields an immediate penalty. Consequently, an ϵ-greedy policy without optimistic initialization would perform poorly, as the only short-term reward it receives would discourage it from doing "right". A random policy is also highly unlikely to reach the chest. In particular, the probability such a policy reaches it in any given episode is $(1/2)^N$. Thus, the expected number of episodes before observing either the bomb or the treasure is 2^N. Even for a moderate value of $N = 50$, this is an impractical number of episodes [Osband et al., 2019].

Results. Figure 3.8 shows the results on a grid of depth $N = 50$. Since this domain has a fixed horizon, the evaluation does not include the "short vs long horizon" comparison. With zero initialization, only our algorithms and bootstrapping by Osband et al. [2016a] and Osband et al. [2019] discover all states and converge to the optimal policy within steps limit. Both proposed visitation-count-based (blue and orange) outperform the other two, converging twice or more as faster. Bootstrapped exploration with random prior (green) comes second, reproducing the results reported by Osband et al. [2018] and Osband et al. [2019]. Results with optimistic inizialization are similar, with the main difference being that also approximate Thompson sampling by D'Eramo et al. [2019] (purple) converges. Boostrapping by Osband et al. [2018] (green) matches the performance of visitation-count-based with UCB (blue), and the two lines almost overlap. For more details about the percentage of successful trials and the percentage of states discovered by each algorithm with confidence intervals, we refer to Table 3.1.

This evaluation shows that the proposed exploration outperform state-of-the-art boostrapping on domain with fixed horizon and no local optima. In the next sections, we investigate domains with absorbing states, i.e., that can prematurely end the episode, and distractor reward. In Section 3.4.2, we also present an empirical study on sample complexity on varying the deep sea depth N. Furthermore in Section 3.4.2, we show the state visitation count at the end of the learning for the deep sea and other domains, discussing why the proposed visitation-value-based exploration achieves new state-of-the-art results.

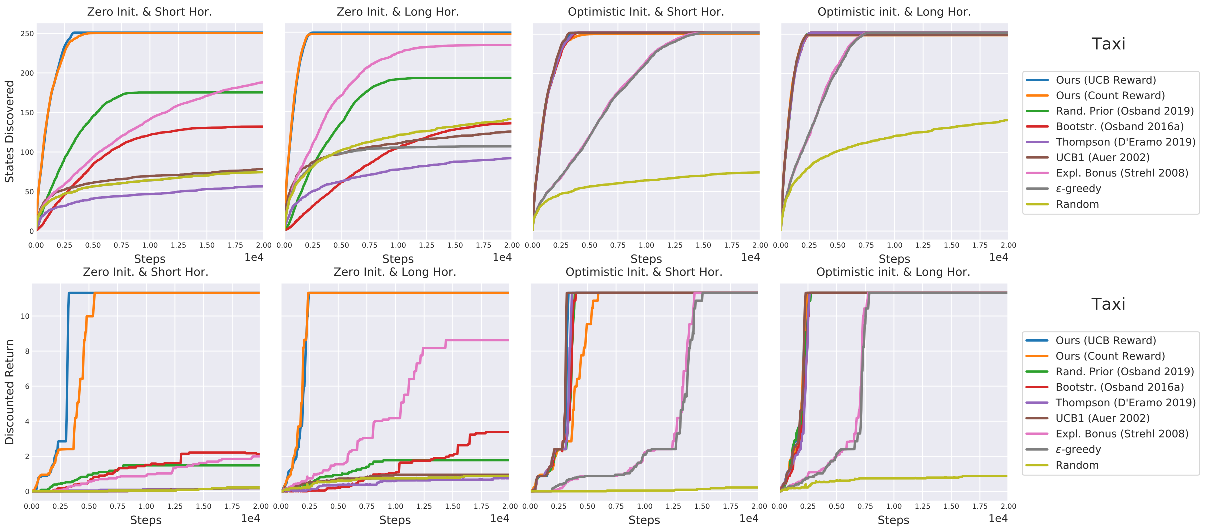


Figure 3.10: Results on the taxi domain averaged over 20 seeds. The proposed algorithms outperform all others, being the only solving the task without optimistic initialization.

Multi-Passenger Taxi Driver

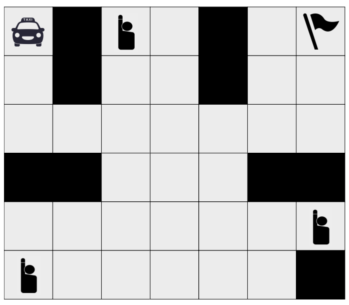


Figure 3.9: The taxi driver.

In this environment (Figure 3.9), the agent (the taxi) starts at the top-left corner of a grid, and has to pick up the passengers and drop them off at the destination (the flag). It can carry multiple passengers at the same time. If one, two, or all three passengers reach the destination, the agent is rewarded with 1, 3, or 15, respectively. Everywhere else the reward is 0. The state consists of the taxi position and three booleans denoting if a passenger has been picked up or not. The agent can turn left, right, up, or down. Black cells cannot be visited. The taxi ends its ride upon visiting the destination, regardless of how many passengers it is carrying. The optimal policy picks up all passengers and drops them off in 29 steps[3].

This domain is challenging for two reasons. First, it admits many locally optimal policies, depending on how many passengers reach the destination. Second, with a short horizon learning to pick up and drop off all passengers is difficult and requires effective exploration.

Results. In this domain, the "short horizon" consists of 33 steps[4], while the "long" of 66. As shown in Figure 3.10, bootstrapped algorithms here perform significantly worse than before. None of them, in fact, learned to pick all three passengers if Q-tables are initialized to zero, and the "long horizon" does not help. In particular, random prior bootstrapping (green) clearly converged prematurely due to the presence of local optima (the algorithm does not discover new states after 750-1,000 steps). The proposed algorithms (blue and orange), instead, perform extremly well, quickly discovering all states and then learning the optimal policy. Other algorithms learned to pick one or two passengers, and only the auxiliary visitation bonus (pink) learned to pick all passengers in some seeds but only with long horizon. With optimistic initialization, most

[3] The agent needs 28 steps to pick all passengers and reach the destination. However, it then needs to execute an additional action to get the reward.

[4] Our algorithms could learn also with a horizon of only 29 steps. However, with zero initialization other algorithms performed extremely poorly. We thus decided to give the agent 33 steps.

of the algorithms match the performance of our proposed ones. Among them, ϵ-greedy (gray) and bonus-based exploration (pink) learn slowly. This is not surprising for the former, while for the latter it may be due to the small bonus coefficient (see Eq. (3.6)) or to the issues discussed in Section 3.3.1. Only random exploration (light green) still cannot learn to pick all passengers. For more details about the percentage of successful trials and the percentage of states discovered by each algorithm with confidence intervals, we refer to Table 3.1.

This evaluation shows that the proposed exploration outperforms existing algorithms in the presence of local optima, and that its performance is not affected by the length of horizon. Next, we investigate what happens when we combine the challenges of the deep sea and the taxi.

Deep Gridworld

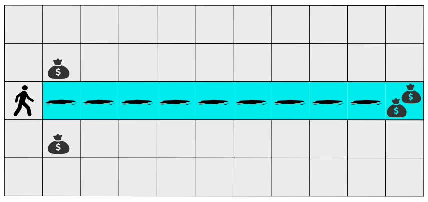


Figure 3.11: The deep gridworld.

The first of the four gridworlds we propose is inspired by the deep sea domain. In this environment (Figure 3.11), the agent navigates in a 5×11 grid by moving "left", "right", "up", or "down". All states can be visited, but the blue corridor can be accessed only from its left side. The agent can exit the corridor anytime by moving either "up" or "down". A treasure of value 2 lies at the end of the corridor, while two treasures of value 1 serve as distractors next to the corridor entrance. The corridor is filled with puddles, each giving a penalty of -0.01. The agent always starts next to the entrance and the episode ends when the agent collects any of the treasures. The optimal policy always does "right".

The deep gridworld combines the challenges of both the deep sea and the taxi domains. Like the former, to discover the highest reward the agent needs to execute the action "right" multiple times in a row, receiving negative immediate rewards due to the puddles. However, doing a different action does not preclude reaching the end of the corridor, because the agent can still go back to the entrance and try again within the same episode. At the same time, the presence of the two distractor treasures results in local optima, as in the taxi driver domain.

Results. Because it is possible to exit the corridor without ending the episode, we set a longer horizon compared to the previous domains, i.e., 55 steps for the "short" scenario and 110 for the "long" one. Results shown in Figure 3.12 confirm previous results. The proposed exploration quickly discoves all states and learn to navigate through the corridor. By contrast, other algorithms, including bootstrapping, get stuck in local optima and learn to collect one of the two lesser treasures. Similarly to the taxi domain, with optimistic initialization all algorithms but random exploration (light green) learn to navigate through the corridor.

This evaluation shows that distractor rewards are challenging for existing algorithms but not for ours. Next, we increase the difficulty by adding more distractors and new special states.

Gridworlds

These environments (Figure 3.16, 3.17, and 3.18) share the same structure. The agent navigates in a grid by moving "left", "right", "up", or "down". Black cells cannot be visited, while any action in the prison has only an arbitrarily small probability of success. If it fails, the agent does not move. In practice, the prison almost completely prevents any further exploration. The reward is 0 everywhere except in treasure or penalty cells, and is given for executing an action

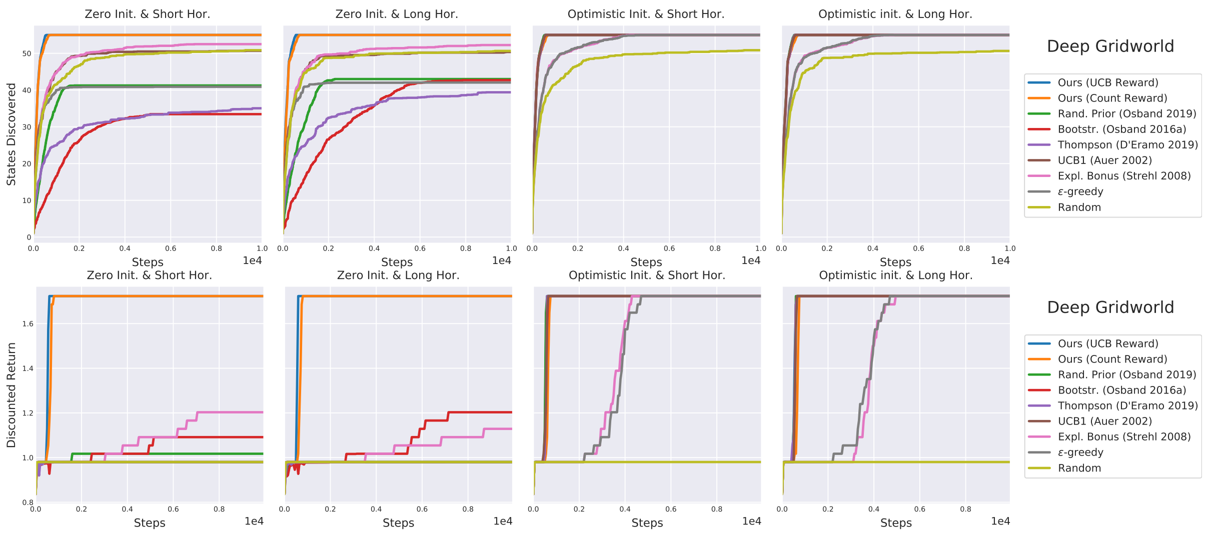


Figure 3.12: Results on the deep gridworld averaged over 20 seeds. Once again, the proposed algorithms are the only solving the task without optimistic initialization.

in the state, i.e., on transitions. Treasure cells (denoted by a money bag or a green number) give a bonus reward of different magnitude and end the episode. Penalty cells (denoted by a red number) give a penalty reward and end the episode. The agent also gets a small penalty of -0.01 at each step. The goal, thus, is to find the biggest treasure using as few steps as possible. The initial position is fixed such that it is far from the biggest treasure and close to the smallest one.

Similarly to the deep gridworld, these domains are designed to highlight the difficulty of learning with many distractors (the smaller treasures) and, thus, of many local optima. The addition of the constant penalty at each step further discourages exploration and makes the distractors more appealing, since ending the episode will stop receiving penalties. Each domain has increasing difficulty and introduces additional challenges, as we explain below.

The "toy" gridworld. We start with a simple 5×5 gridworld with one single reward of value 1, as shown in Figure 3.16. The focus of this domain is learning with sparse reward, without distractors or additional difficulties. The optimal policy takes only nine steps to reach the treasure, and the "short horizon" scenario ends after eleven steps.

The "prison" gridworld. Shown in Figure 3.17, this domain increases the value of the furthest reward to 5, adds two distractors of value 1 and 2, and adds a prison cell. The first distractor is close to the initial position, while the second is close to the goal reward. The prison cell is also close to the goal. The optimal policy navigates around the first distractor, then right below the prison, and finally up to the goal. The prison highlights the importance of seeking states with low long-term visitation count, rather than low immediate count. As discussed in Section 3.3.1, in fact, current count-based algorithms cannot deal with this kind of states efficiently.

The "wall" gridworld. Shown in Figure 3.18, the last gridworld is characterized by increased grid size (50×50) and reward magnitude, and by the wall separating the grid into two areas. The first area, where the agent starts, has six small rewards (both treasures and penalties). The second area has two bigger treasures in the upper-right and bottom-right corners, of value 500 and 10,000, respectively. The optimal policy brings the agent beyond the wall and then to the bottom right corner, where the highest reward lies. The wall significaly increases the difficulty, due to the narrow passage which the agent needs to find in order to visit the second area of the

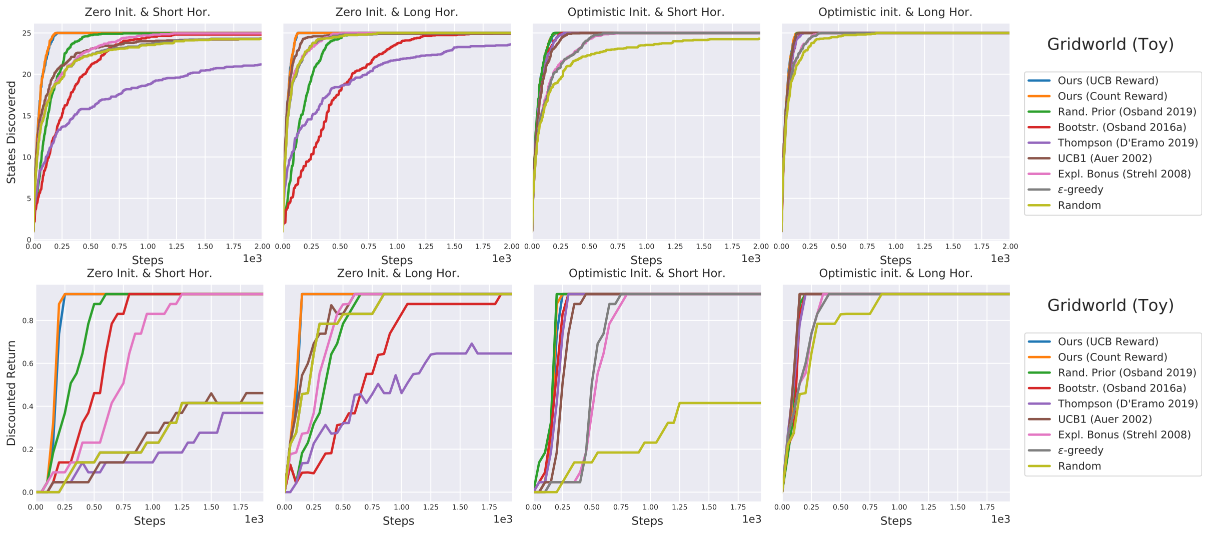


Figure 3.13: Results on the toy gridworld averaged over 20 seeds. Bootstrapped and bonus-based exploration perform well, but cannot match the proposed one (blue and orange line overlap).

grid. Learning is even harder when the horizon is short, as the agent cannot afford to lose time randomly looking for new states. To increase the chance of success of all algorithms, we set the "short horizon" to 330 steps, and 135 are needed to reach the reward of 10,000.

Results. The gridworld environments confirm previous results. Without distractors (toy gridworld, Figure 3.13), bootstrapped algorithms (green and red) perform well, and so does using the auxiliary visitation bonus (pink). Neither, however, match ours (blue and orange, almost overlapping). Increasing the horizon helps the remainder algorithms, including random exploration (light green), except for approximate Thompson sampling (purple). In the next gridworld, however, the distractors and the prison cell substantially harm all algorithms except ours (Figure 3.14). Without optimistic initialization, in fact, existing algorithms cannot find the highest reward even with long horizon, and all converge to local optima. This behavior was expected, given the study of Section 3.3.1. Finally, the wall gridworld results emphasize even more the superiority of our algorithms (Figure 3.15). With zero initialization, in fact, every other algorithm cannot go beyond the wall and find even the reward of 500. The results also stress that using the UCB reward visitation value (blue) over the count reward (orange) performs slightly best.

This evaluation strenghtens the findings of previous experiments. First, it stresses how difficult it is for existing algorithms to learn with distracting rewards and short horizon. Second, it shows that our proposed approach overcomes these challenges. Next, we show how efficiently the algorithms explore in terms of final visitation count and sample complexity.

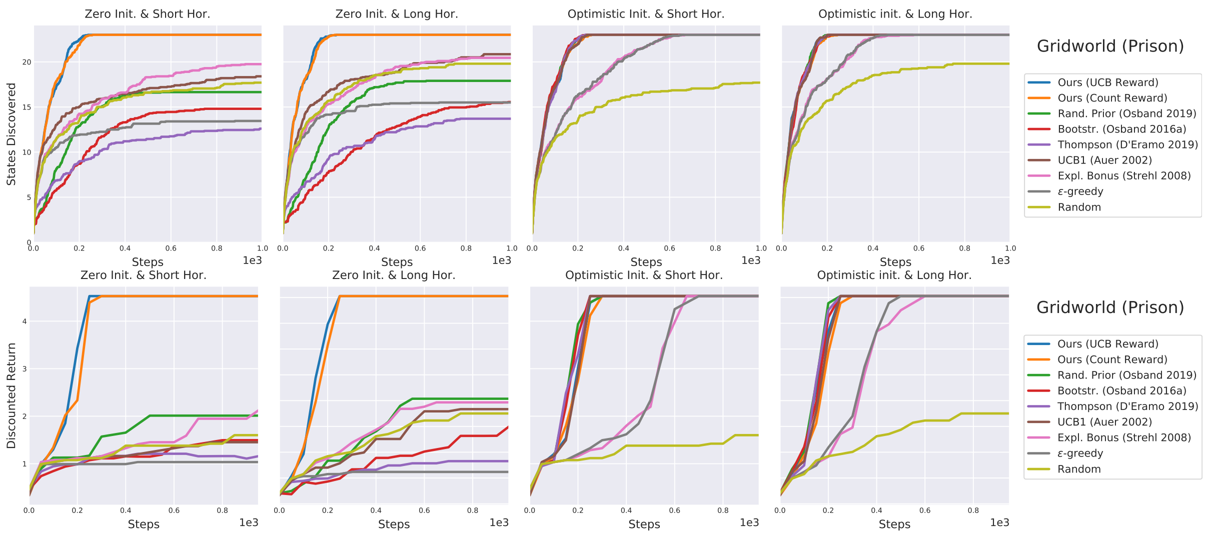


Figure 3.14: Distractors and the "prison" affect the performance of all algorithms but ours.

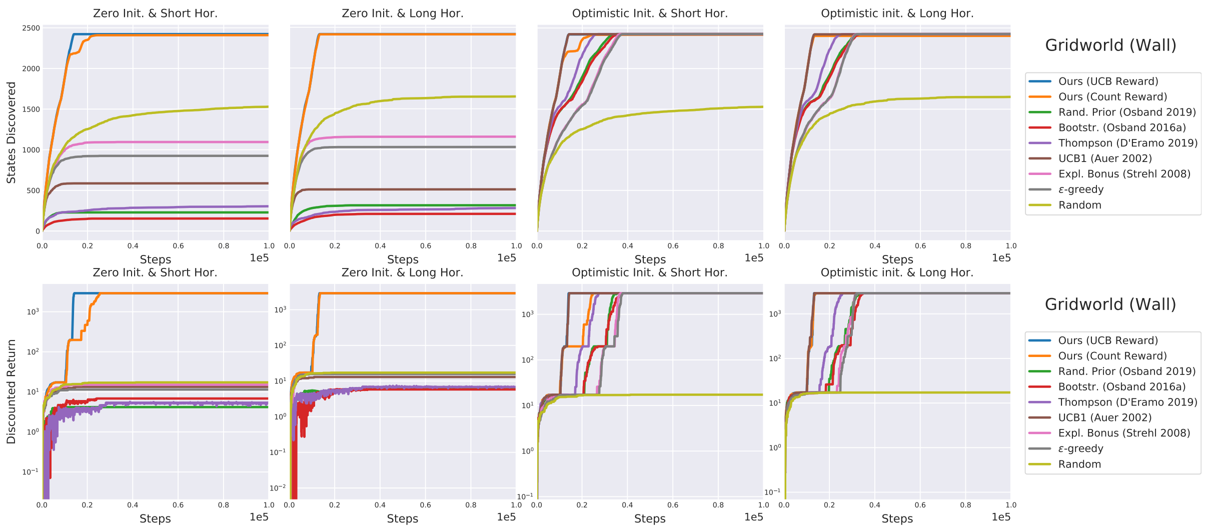


Figure 3.15: The final gridworld emphasizes the better performance of the proposed algorithms, especially the one using UCB reward. Return plot is in log scale.

The Gridworlds Environments

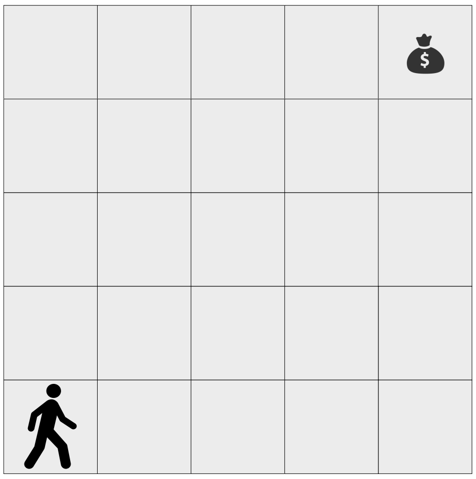


Figure 3.16: The “toy” gridworld.

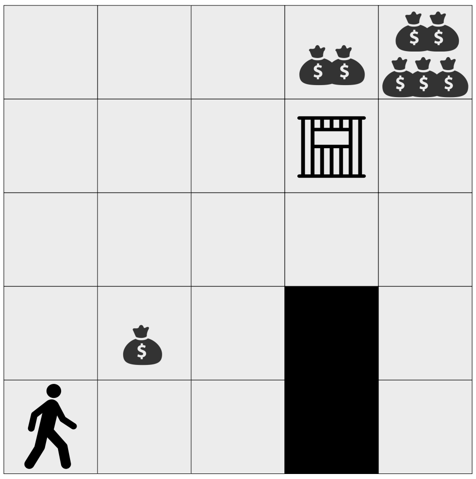


Figure 3.17: The “prison” gridworld.

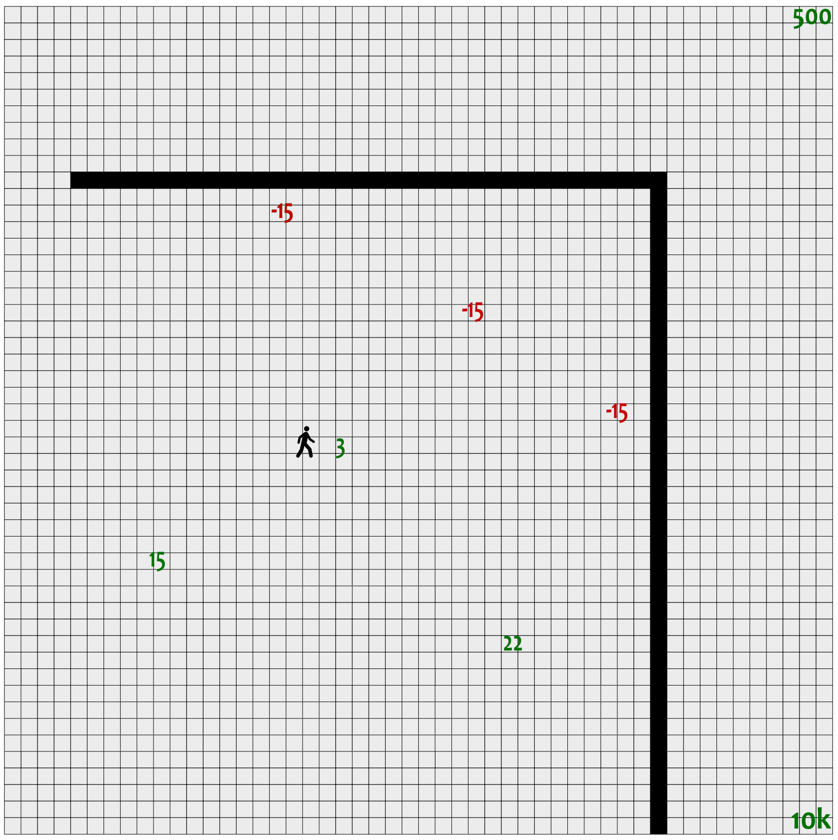


Figure 3.18: The “wall” gridworld.

Table 3.1: Results recap for the "zero initialization short horizon" scenarios. Only the proposed exploration strategy is able to always discover all states and solve the task in all 20 seeds.

	Algorithm	Discovery (%)	Success (%)
Deep Sea	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	99.90 ± 0.01	100 ± 0
	Bootstr. (Osband 2016a)	99.77 ± 0.05	100 ± 0
	Bootstr. (D'Eramo 2019)	63.25 ± 3.31	0 ± 0
	UCB1 (Auer 2002)	55.72 ± 0.34	0 ± 0
	Expl. Bonus (Strehl 2008)	85.65 ± 1.0	0 ± 0
	ϵ-greedy	57.74 ± 1.11	0 ± 0
	Random	58.59 ± 1.35	0 ± 0
Taxi	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	69.60 ± 2.96	13.07 ± 2.96
	Bootstr. (Osband 2016a)	52.44 ± 7.55	18.77 ± 9.24
	Bootstr. (D'Eramo 2019)	22.44 ± 1.81	1.9 ± 1.47
	UCB1 (Auer 2002)	31.17 ± 0.70	1.53 ± 1.37
	Expl. Bonus (Strehl 2008)	74.62 ± 2.24	17.6 ± 2.56
	ϵ-greedy	29.64 ± 0.98	1.92 ± 1.49
	Random	29.56 ± 0.98	1.92 ± 1.49
Deep Gridworld	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	75 ± 9.39	59.03 ± 4.2
	Bootstr. (Osband 2016a)	60.82 ± 11.78	63.35 ± 6.89
	Bootstr. (D'Eramo 2019)	63.73 ± 10.35	56.85 ± 0.06
	UCB1 (Auer 2002)	92.18 ± 0.36	56.88 ± 0
	Expl. Bonus (Strehl 2008)	95.45 ± 1.57	69.81 ± 8.84
	ϵ-greedy	74.36 ± 4.42	56.88 ± 0
	Random	92.45 ± 0.64	56.88 ± 0
Gridworld (Toy)	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	99.8 ± 0.39	100 ± 0
	Bootstr. (Osband 2016a)	99.2 ± 0.72	100 ± 0
	Bootstr. (D'Eramo 2019)	84.8 ± 4.33	40 ± 21.91
	UCB1 (Auer 2002)	97.4 ± 0.99	50 ± 22.49
	Expl. Bonus (Strehl 2008)	99.8 ± 0.39	100 ± 0
	ϵ-greedy	97.4 ± 1.17	45 ± 22.25
	Random	97.2 ± 1.15	45 ± 22.25
Gridworld (Prison)	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	72.39 ± 6.9	44.44 ± 14.69
	Bootstr. (Osband 2016a)	64.35 ± 11.1	33.03 ± 8.54
	Bootstr. (D'Eramo 2019)	54.78 ± 6.69	25.61 ± 8.39
	UCB1 (Auer 2002)	80.78 ± 2.56	32.31 ± 4.11
	Expl. Bonus (Strehl 2008)	85.87 ± 3	46.96 ± 12.35
	ϵ-greedy	58.38 ± 5.27	22.84 ± 2.48
	Random	76.96 ± 3.08	35.36 ± 10.37
Gridworld (Wall)	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	9.44 ± 3.14	0.14 ± 0.06
	Bootstr. (Osband 2016a)	6.35 ± 2.74	0.23 ± 0.08
	Bootstr. (D'Eramo 2019)	12.55 ± 4.45	0.18 ± 0.08
	UCB1 (Auer 2002)	24.7 ± 4.45	0.46 ± 0.03
	Expl. Bonus (Strehl 2008)	45.16 ± 2.29	0.52 ± 0.04
	ϵ-greedy	38.15 ± 1.95	0.39 ± 0.08
	Random	65.28 ± 0.45	0.59 ± 0

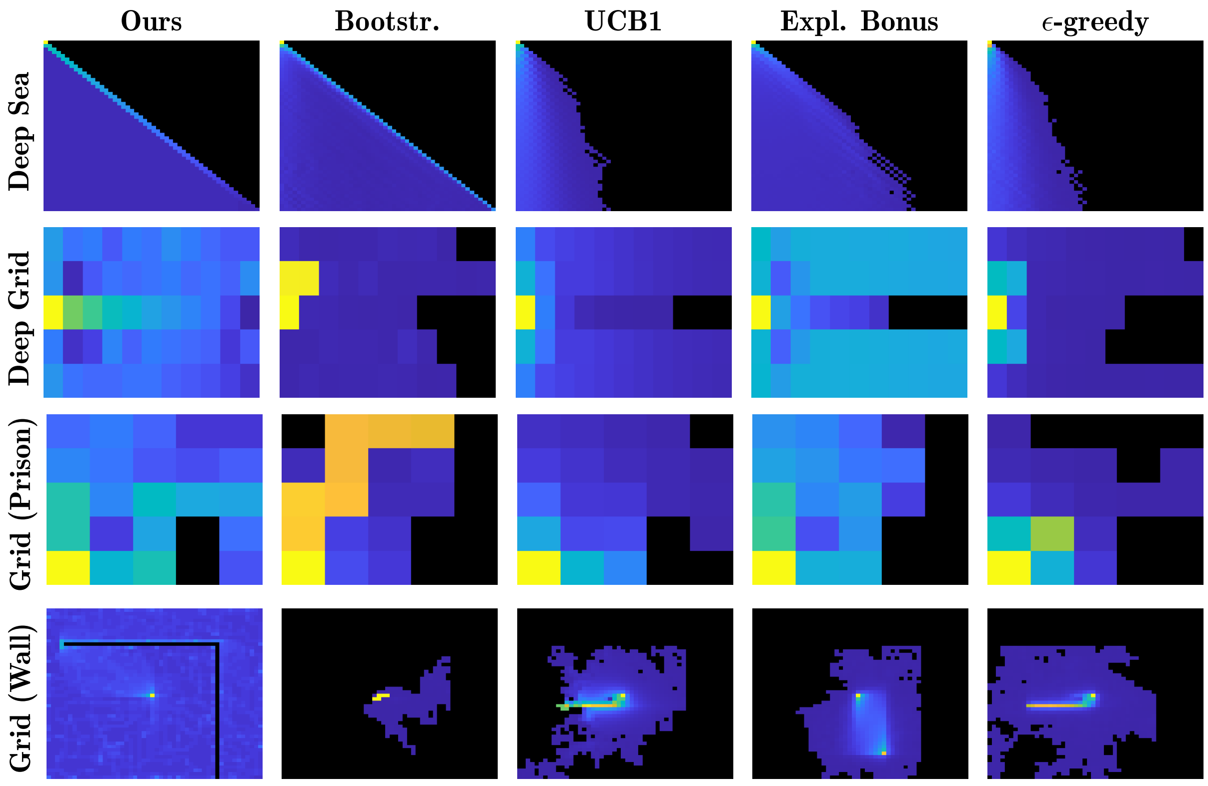


Figure 3.19: Visitation count at the end of the learning. The seed is the same across all images. Initial states naturally have higher count than other states. Recall that the upper portion of the deep sea and some gridworlds cells cannot be visited. Only the proposed methods explore the environment uniformly[5]. Other algorithms myopically focus on distractors.

3.4.2 Part 2: In-Depth Investigation

In this section, we present an in-depth analysis of issues associated with learning with sparse rewards, explaining in more detail why our approach outperformed existing algorithms in the previous evaluation. We start by reporting the visitation count at the end of the learning, showing that our algorithms explore the environment uniformly. We continue with a study on the impact of the visitation discount γ_w, showing how it affects the performance of the W-function. Next, we present the empirical sample complexity analysis of all algorithms on the deep sea domain, showing that our approach scales gracefully with the number of states. Finally, we evaluate the algorithms in the infinite horizon setting and in stochastic MDPs, evaluating the accuracy of the Q-function and the empirical sample complexity. Even in these scenarios, our approach outperforms existing algorithms.

Visitation Count at the End of the Learning

In Section 3.3.3 and 3.3.4, we discussed that the proposed behavior policies guarantee that an action is not executed twice before all other actions are executed once, under the uniform count assumption. We also acknowledged that this assumption cannot hold in practice because of state

[5] Figures show the count of UCB-based W-function. Count-based W-function performed very similarly.

resets, or becuase the agent may need to revisit the same state to explore new ones. For example, in the deep sea the agent needs to traverse diagonal states multiple times to visit every state. Nonetheless, in this section we empirically show that our approach allows the agent to explore the environment as uniformly as possible.

Figure 3.19 reports the state visitation count at the end of learning in the "short-horizon zero-initialization" scenario for some domains. Only the proposed method (first column) uniformly explores all states. Recall, in fact, that episodes reset after some steps or when terminal states are reached. Thus initial states (which are fixed) are naturally visited more often. The count in our method uniformly decreases proportionally to the distance from the initial states. This denotes that at each episode, starting from the same state, the agent followed different paths, exploring new regions of the environment uniformly.
Other algorithms suffer from distractors and local optima. In particular, UCB1 (third column) explores very slowly. The reason is that the agent myopically selects actions based on the *immediate* count, which makes exploration highly inefficient. By selecting the action with the lowest immediate count, the agent does not take into account where the action will lead it, i.e., if future states have already been visited or not. By contrast, our approach achieves deep exploration by taking into account the long-term visitation count of future states.
The performance of bootrstrapping (second column) is peculiar. It immediately converges to the first reward found in the gridworlds, barely exploring afterwards, but performs very well on the deep sea. This hints that bootstrapping may be sensitive to any kind of reward, including negative penalties. In the deep sea, in fact, diagonal states provide the only intermediate feedback beside the final reward/penalty. Coincidentially, traversing the diagonal also leads to the only positive reward. The agent may thus be guided by the reward received on the diagonal, even if it is a negative penalty. This behavior can be explained by recalling that Osband et al. [2019] regularize TD learning with the ℓ_2-distance from a prior Q-table. The agent may therefore be attracted to any state providing some kind of feedback in order to minimize the regularization.

Impact of Visitation Value Discount Factor γ_w

Here, we investigate the effect of the visitation discount factor γ_w on the "toy" gridworld (Figure 3.17). Figure 3.20 shows that the higher the discount, the better. As expected, with $\gamma_w = 0$ the algorithms behave very similarly to UCB1. In this case, in fact, the proposed approach and UCB1 are equivalent. However, the learning curves are not exactly the same because the W-function is updated with TD learning at every step.
The better exploration of our approach is also confirmed by Figure 3.21. The colormaps show the count at the end of the learning, and the higher γ_w, the more uniform the exploration is. With smaller γ_w, in fact, the agent rarely discovers states far from the initial position.

Empirical Sample Complexity on the Deep Sea

Here, we propose the same evaluation presented by Osband et al. [2018] to investigate the empirical sample complexity of the proposed algorithms, and to assess how they scale to large problems. Figure 3.22 plots training steps to learn the optimal policy as function of the environment size N. Only our approach (blue and orange) scales gracefully to large problem sizes. Results suggest an empirical scaling of $\mathcal{O}(N^{2.5})$ for the visitation-value-based with count reward, and even smaller for UCB reward. Bootstrapping attains an empirical complexity of $\mathcal{O}(N^3)$, confirming

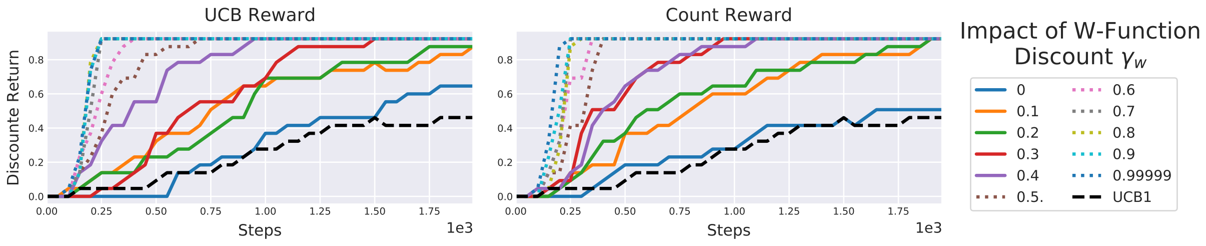


Figure 3.20: Performance of the proposed algorithms on varying of γ_w on the "toy" gridworld (Figure 3.17). The higher γ_w, the more the agent explore, allowing to discover the reward faster.

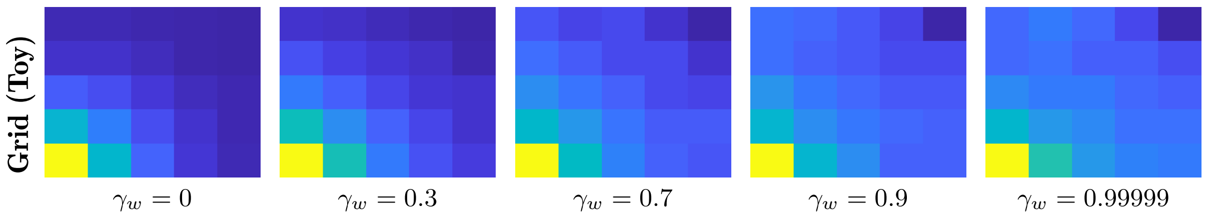


Figure 3.21: Visitation count for W_{N}^{β} at the end of the learning on the same random seed (out of 20). The higher the discount, the more uniform the count is. W_{UCB}^{β} performed very similarly.

the findings of Osband et al. [2019]. However, in many cases ($N = 23, 27, 33, 37, 43, 47, 51, 55$) there was one seed for which bootstrapping either did not learn within the steps limit (green dashed line, empty dots) due to premature convergence, or learned after substantially more steps than the average (red dots, large error bar). Missing algorithms (random, ϵ-greedy, UCB1, approximate Thompson sampling) performed extremely poorly, often not learning within 500,000 steps even for small N, and are thus not reported.

Infinite Horizon Stochastic Chainworld

This evaluation investigates how the algorithms perform in a stochastic MDP with infinite horizon. In particular, we are interested in (1) the mean squared value error (MSVE) at each timestep, (2) the sample complexity when varying the number of states, and (3) how behavior policies explore and if they converge to the greedy policy.

Evaluation Criteria. The MSVE is computed between the true value function of the optimal policy $V^*(s)$ and the learned value function of the current policy $V^{\pi_t}(s)$, i.e.,

$$\mathrm{MSVE} = \frac{1}{N}\sum_{i=1}^{N}\left(V^*(s_i) - V^{\pi_t}(s)\right)^2. \tag{3.29}$$

The value function is computed according to the Bellman expectation equation in matrix form, i.e., $V^\pi = (I - \gamma\mathcal{P}^\pi)^{-1}R^\pi$, where I is the identity matrix, and $\mathcal{P}^\pi$ and R^π are the transition and reward functions induced by the policy, respectively [Sutton and Barto, 2018]. This error indicates how much the learned greedy policy π_t deviates from the optimal policy.
Similarly, the sample complexity is defined as the number of timesteps t such that the non-stationary policy π_t at time t is not ε-optimal for current state s_t, i.e, $V^*(s_t) - V^{\pi_t}(s_t) > \varepsilon$ [Strehl and Littman, 2008, Dong et al., 2019].
For the behavior of exploration policies and their empirical convergence, we show how the visitation count changes over time.

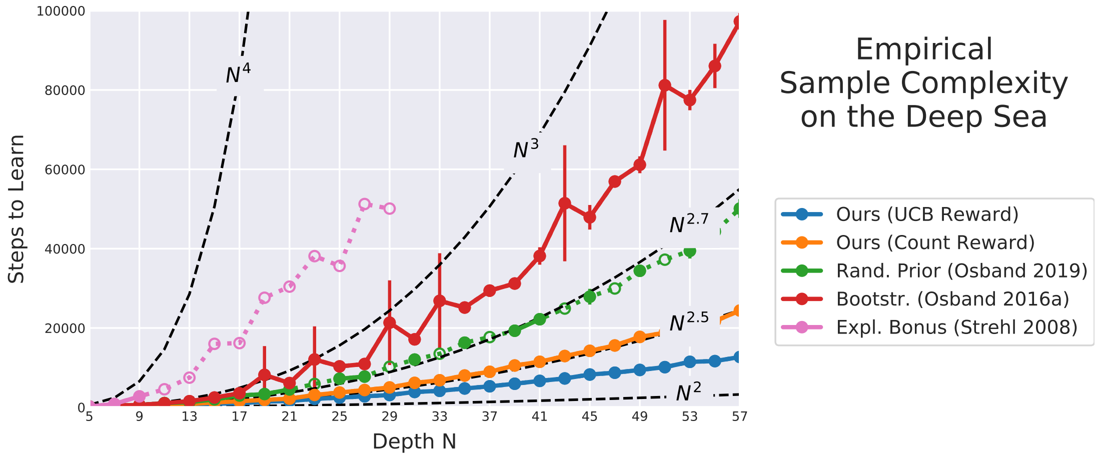


Figure 3.22: Steps to learn on varying the deep sea size $N = 5, 7, \ldots, 59$. Each filled dot denotes the average over ten runs with 500,000 steps limit. Error bars denote 95% confidence interval. In empty dots connected with dashed lines, the algorithm did not learn within steps limit at least once. In this case, the average is over converged runs only. Missing algorithms (random, ϵ-greedy, UCB1, Thompson sampling) performed poorly and are not reported. Only our algorithms learn with confidence interval close to zero, and attain the lowest sample complexity.

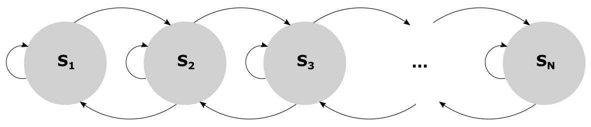


Figure 3.24: The ergodic chainworld.

MDP Characteristics. The MDP, shown in Figure 3.24, is a chainworld with $s = 1, \ldots, N$ states, three actions, and stochastic transition defined as follows. The first action moves the agent forward with probability p, backward otherwise. The second action moves the agent backward with probability p, forward otherwise. The third action keeps the agent in the current state with probability p, and randomly moves it backward or forward otherwise. The initial state is the leftmost state, and no state is terminal. This MDP is ergodic, i.e., all state are transient, positive recurrent and aperiodic for any deterministic policy. In our experiments, we set $p = 0.99$. The reward is 10^{-8} for doing "stay" in the initial state s_1, 1 for doing "stay" in the last state s_N, and 0 everywhere else.

Unlike previous MDPs, this has no terminal state, the horizon is infinite, and the state is never reset. Instead, to simulate infinite horizon, the agent explores the environment for one episode of 200,000 steps. To comply with the classic sample complexity definition [Strehl and Littman, 2008, Dong et al., 2019], we use classic Q-learning without replay memory. We do not compare against bootstrapping algorithms, because they are designed to select a different behavior Q-function at every episode, and this setup has only one episode. Approximate Thompson sampling by D'Eramo et al. [2019], instead, selects the Q-function at every step, and it is included in the comparison (without memory as well). All algorithms use zero Q-function initialization. Due to the stochasticity of the MDP, we increased the number of random seeds to 50.

Results. Figure 3.23 shows how the algorithms explore over time. Only ours (3.23a and 3.23b) and UCB1 (3.23c) explore uniformly, but UCB1 finds the last state (with the reward) later. The auxiliary rewards (3.23d) and approximate Thompson sampling (3.23e) also perform poorly, since the exploration is not uniform and the reward is found only late. ϵ-greedy exploration (3.23f), instead, is soon stuck in the local optimum represented by the small reward in the initial state.

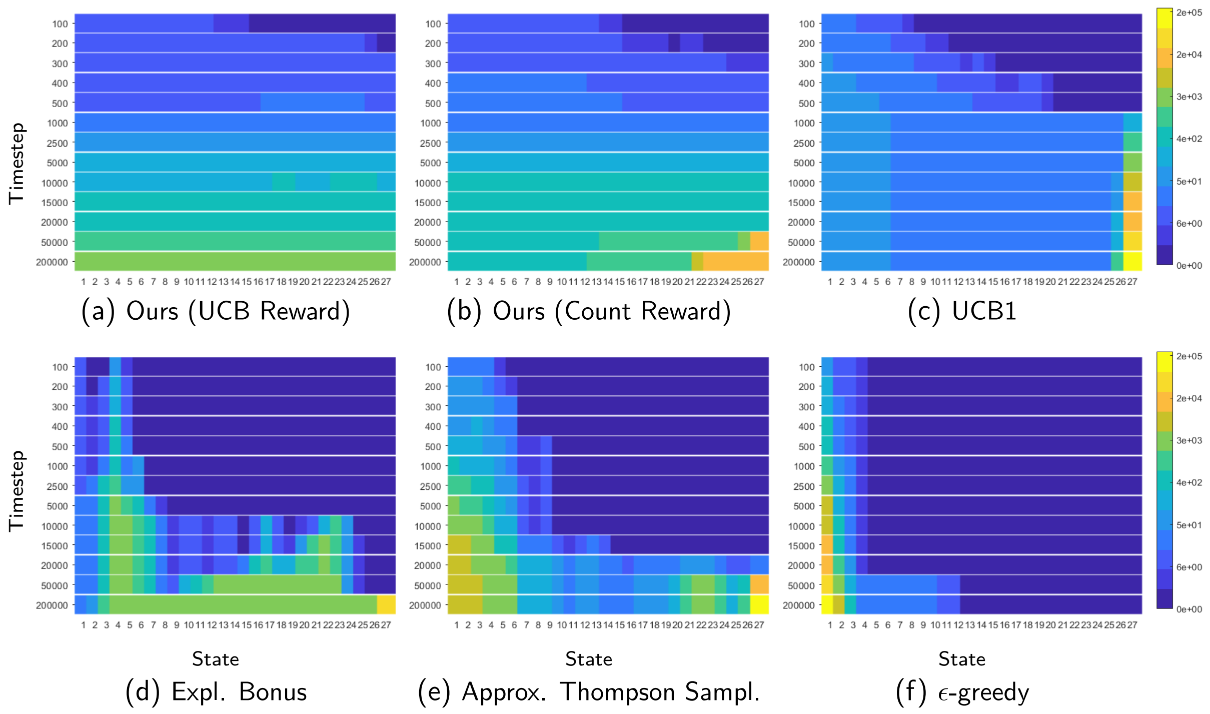


Figure 3.23: Visitation counts for the chainworld with 27 states (x-axis) over time (y-axis). As time passes (top to bottom), the visitation count grows. Only our algorithms and UCB1 explore uniformly, producing a uniform colormap. However, UCB1 finds the last state (with the reward) much later. Once UCB1 finds the reward (Figure 3.23c, step 1,000) the visitation count increses only in the last state. This suggests that when UCB1 finds the reward contained in the last state it effectively stops exploring other states. By contrast, our algorithms keep exploring the environment for longer. This allows the agent to learn the true Q-function for all states, and explains why our algorithms achieve lower sample complexity and MSVE in Figures 3.25 and 3.26.

The visitation count also shows that all exploration policies but ours act greedily once the reward is found. For example, UCB1 finds the reward at step 1,000, and after that its visitation count increses only in the last state (and nearby states, because of the stochastic transition). This suggests that when these algorithms find the reward contained in the last state they effectively stops exploring other states. By contrast, our algorithms explore the environment for longer, yielding a more uniform visitation count. This allows the agent to learn the true Q-function for all states, achieving lower sample complexity and MSVE as shown in Figures 3.25 and 3.26.

Figures 3.25 shows the sample complexity on varying the chain size over three values of ε. In all three cases, our algorithms scale gracefully with the number of states, suggesting a complexity of $\mathcal{O}(N^{2.5})$ and strengthening the findings of Section 3.4.2. By contrast, other algorithms performed poorly. Their sample samplexity has large confidence interval, and as ε decreases the complexity increases to the point that they almost never learn an ε-optimal policy.

These results are confirmed by Figure 3.26 (left plots), where only our algorithms attain low sample complexity even for $\varepsilon = 0$. Our algorithms are also the only ones to learn an almost perfect Q-function, with a MSVE close to zero (right plots). As anticipated, these results are explained by the uniform visitation count in Figure 3.23. By not converging to the greedy policy too quickly after discovering the reward, the agent keeps visiting old states and propagate to them the information about the reward, thus learning the true optimal Q-function for all states.

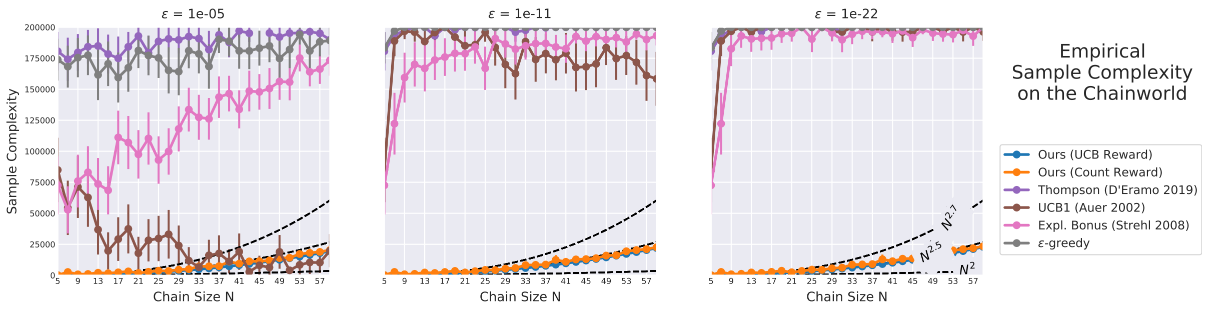


Figure 3.25: Sample complexity on varying the chain size $N = 5, 7, \dots, 59$. Error bars denote 95% confidence interval. Only the proposed algorithms (blue and orange lines almost overlap) show little sensitivity to ε, as their sample complexity only slightly increases as ε decreases. By contrast, other algoritms are highly influenced by ε. Their estimate complexity has large confidence interval, and for smaller ε they almost never learn an ε-optimal policy.

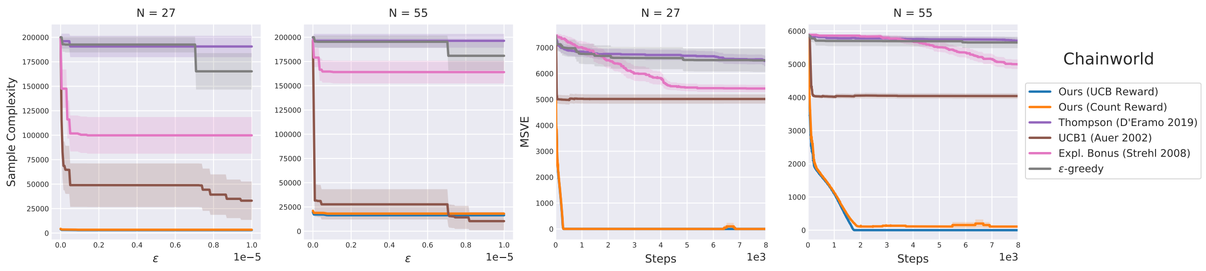


Figure 3.26: Sample complexity against ε, and MSVE over time for 27- and 55-state chainworlds. Shaded areas denote 95% confidence interval. As seen in Figure 3.25, the sample complexity of our approach is barely influenced by ε. Even for $\varepsilon = 0$, our algorithms attain low sample complexity, whereas other algorithms complexity is several orders of magnitude higher. Similarly, only our algorithms learn an almost perfect Q-function, achieving a MSVE close to zero.

Stochastic Gridworlds

As we discussed in Section 3.3.4, the visitation rewards for training the W-functions penalize terminal state-action pairs. One may think that this would lead to poor exploration in stochastic environments, where the same state-action pair can lead to both terminal and non-terminal states. Here, we evaluate all algorithms again on some of the previous environments but this time with stochastic transition, and show that the proposed W-functions still solve the MDPs and outperform existing algorithms. Each transition (s, a) has probability p of succeeding, probability $1 - p/2$ of not moving the agent, and probability $1 - p/2$ of moving the agent to a random adjacent state. In our experiments, we set $p = 0.9$. Due to the stochasticity of the transition function, we increased the number of random seeds from 20 to 50.

Figure 3.27 shows the number of states discovered, and the expected discounted return computed in closed form as $V^\pi(s_0)$, where V^π is defined according to the Bellman expectation equation as $V^\pi = (I - \gamma\mathcal{P}^\pi)^{-1}R^\pi$. Similarly to the deterministic setting (Figures 3.12, 3.13, and 3.14, leftmost plots), our algorithms (blue and orange) outperform all baseline algorithms, being the only ones to solve all three MPDs within steps limit[6]. With the UCB reward (blue),

[6] Notice that the steps limit is higher compared to the deterministic setting. Because of the stochasticity, in fact, we decreased the learning rate from 0.5 to 0.1.

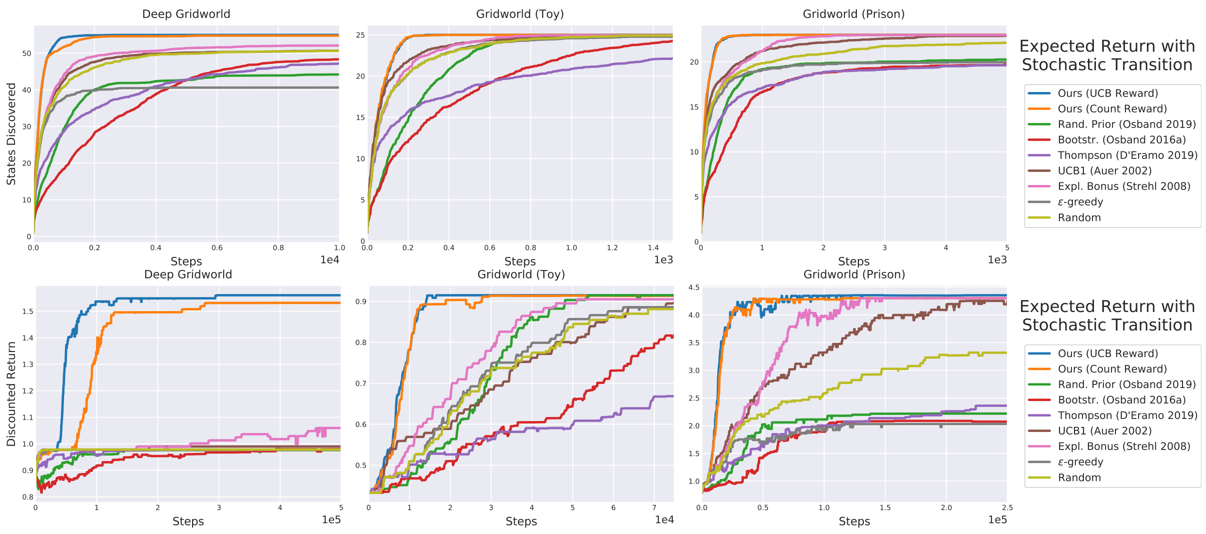


Figure 3.27: Results on three of the previous MDPs, but this time with stochastic transition function. Once again, only our algorithms visit all states and learn the optimal policy within steps limit in all MDPs, and their performance is barely affected by the stochasticity.

our approach always learns the optimal policy. With count reward (orange), our approach almost always learns the optimal policy, and converges to a distractor only two times (out of 50) in the deep gridworld, and once in the "prison" gridworld. However, they both quickly discovered all states, as shown by top plots. The better performance of UCB-based reward is due to the optimistic initialization of the W-function. As seen in Section 3.4.2, Figure 3.23a, this version of the algorithm explores the environment for longer, and converges to the greedy policy much later. Therefore, the agent will visit the high-reward state more often, and it is less likely to learn suboptimal policies.
Bootstrapping with a prior (green) is not affected by the stochasticity, and it performs as in the deterministic setting (it solves the "toy" gridworld but not the other two). Vanilla bootstrapping (red), instead, is heavily affected by the stochasticity, and its performance is substantially worse compared to the deterministic setting (it cannot learn even the "toy" gridworld within steps limit). UCB1 (brown) and bonus-based exploration (pink), instead, benefit from the stochasticity. In the deterministic setting they could not solve these MDPs, while here they solve the two gridworlds (even though much later than ours). This improvement is explained by the larger number of visitated states (top plots), which denotes an overall better exploration. This is not surprising, if we consider that a stochastic transition function naturally helps exploration.

3.5 Conclusion and Future Work

Effective exploration with sparse rewards is an important challenge in RL, especially when sparsity is combined with the presence of "distractors", i.e., rewards that create suboptimal modes of the objective function. Classic algorithms relying on dithering exploration typically perform poorly, often converging to poor local optima or not learning at all. Methods based on immediate counts have strong guarantees but are empirically not sample efficient, while we showed that methods based on intrinsic auxiliary rewards require hand-tuning and are prone to suboptimal behavior. In this paper, we presented a novel approach that (1) plans exploration actions far into the future by using a long-term visitation count, and (2) decouples exploration and exploitation by learning a separate function assessing the exploration value of the actions. Contrary to existing methods which use models of reward and dynamics, our approach is off-policy and model-free. Empirical results showed that the proposed approach outperforms existing methods in environments with sparse and distracting rewards, and suggested that our approach scales gracefully with the size of the environment.

The proposed approach opens several avenues of research. First, in this work we focused on empirical results. In the future, we will investigate the theoretical properties of the proposed approach in more detail. Second, in this work we considered model-free RL. In the future, we will extend the proposed approach and combine it with model-based RL. Third, the experimental evaluation focused on identifying the challenges of learning with sparse and distracting rewards. In the future, we will consider more diverse tasks with continuous states and actions, and extend the proposed exploration to actor-critic methods.

4 TD-Regularized Actor-Critic Methods

Actor-critic methods can achieve incredible performance on difficult reinforcement learning problems, but they are also prone to instability. This is partly due to the interaction between the actor and the critic during learning, e.g., an inaccurate step taken by one of them might adversely affect the other and destabilize the learning. To avoid such issues, we propose to regularize the learning objective of the actor by penalizing the temporal difference (TD) error of the critic. This improves stability by avoiding large steps in the actor update whenever the critic is highly inaccurate. The resulting method, which we call the *TD-regularized* actor-critic method, is a simple plug-and-play approach to improve stability and overall performance of the actor-critic methods. Evaluations on standard benchmarks confirm this.
Source code can be found at `https://github.com/sparisi/td-reg`

4.1 Introduction

Actor-critic methods have achieved incredible results, showing super-human skills in complex real tasks such as playing Atari games and the game of Go [Silver et al., 2017a, Mnih et al., 2016]. Unfortunately, these methods can be extremely difficult to train and, in many cases, exhibit unstable behavior during learning. One of the reasons behind their instability is the interplay between the actor and the critic during learning, e.g., a wrong step taken by one of them might adversely affect the other and can destabilize the learning [Dai et al., 2018]. This behavior is more common when nonlinear approximators, such as neural networks, are employed, but it could also arise even when simple linear functions are used[1]. Figure 4.1 (left) shows such an example where a linear function is used to model the critic but the method fails in three out of ten learning trajectories. Such behavior is only amplified when deep neural networks are used to model the critic.

In this chapter, we focus on developing methods to improve the stability of actor-critic methods. Most of the existing methods have focused on stabilizing either the actor or the critic. For example, some recent works improve the stability of the critic by using a slowly-changing critic [Lillicrap et al., 2016, Mnih et al., 2015, Hessel et al., 2018], a low-variance critic [Munos et al., 2016, Gruslys et al., 2018], or two separate critics to reduce their bias [van Hasselt, 2010, Fujimoto et al., 2018]. Others have proposed to stabilize the actor instead, e.g., by constraining its update using entropy or the Kullback-Leibler (KL) divergence [Peters et al., 2010, Schulman et al., 2015, Akrour et al., 2016, Achiam et al., 2017, Nachum et al., 2018, Haarnoja et al., 2018]. In contrast to these approaches that focus on stabilizing either the actor or the critic, we focus on stabilizing the *interaction* between them.

Our proposal is to stabilize the actor by penalizing its learning objective whenever the critic's estimate of the value function is highly inaccurate. We focus on critic's inaccuracies that are

[1] Convergence is assured when special types of linear functions known as *compatible* functions are used to model the critic [Sutton et al., 1999, Peters and Schaal, 2008a]. Convergence for other types of approximators is assured only for some algorithms and under some assumptions [Baird, 1995, Konda and Tsitsiklis, 2000, Castro et al., 2008].

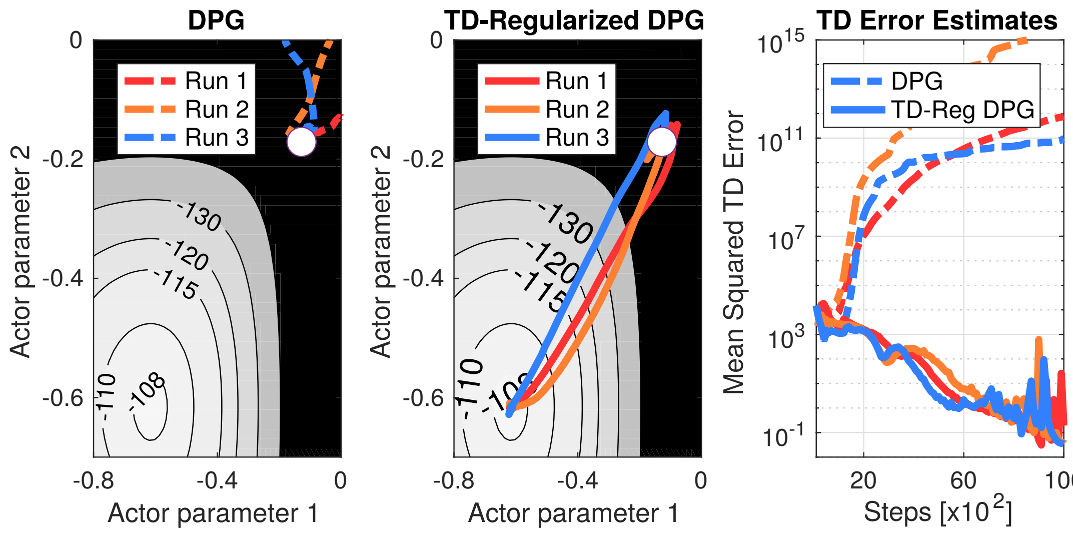


Figure 4.1: Left figure shows three runs that failed to converge out of ten runs for an actor-critic method called deterministic policy gradient (DPG). The contour lines show the true expected return for the two parameters of the actor, the white circle shows the starting parameter vector. For DPG, we approximate the value function by an incompatible linear function (details in Section 4.5.1). None of the three runs make it to the maximum, located at the bottom-left corner. By contrast, as shown in the middle figure, adding the TD-regularization fixes the instability and all the runs converge. The rightmost figure shows the estimated TD error for the two methods. We clearly see that TD-regularization reduces the error over time and improves not only stability and convergence but also the overall performance.

caused by severe violation of the Bellman equation, as well as large temporal difference (TD) error. We penalize for such inaccuracies by adding the critic's TD error as a regularization term in the actor's objective. The actor is updated using the usual gradient update, giving us a simple yet powerful method which we call the *TD-regularized actor-critic method*. Due to this simplicity, our method can be used as a plug-and-play method to improve stability of existing actor-critic methods together with other critic-stabilizing methods. in this chapter, we show its application to stochastic and deterministic actor-critic methods [Sutton et al., 1999, Silver et al., 2014], trust-region policy optimization [Schulman et al., 2015] and proximal policy optimization [Schulman et al., 2017], together with Retrace [Munos et al., 2016] and double-critic methods [van Hasselt, 2010, Fujimoto et al., 2018]. Through evaluations on benchmark tasks, we show that our method is complementary to existing actor-critic methods, improving not only their stability but also their performance and data efficiency.

4.1.1 Related Work

Instability is a well-known issue in actor-critic methods, and many approaches have addressed it. The first set of methods do so by stabilizing the critic. For instance, the so-called target networks have been regularly used in deep RL to improve stability of TD-based critic learning methods [Mnih et al., 2015, Lillicrap et al., 2016, van Hasselt, 2010, Gu et al., 2016b]. These target networks are critics whose parameters are slowly updated and are used to provide stable TD targets that do not change abruptly. Similarly, Fujimoto et al. [2018] proposed to take the minimum value between a pair of critics to limit overestimation, and delay to update the policy parameters so that the per-update error is reduced. Along the same line, Munos et al. [2016] proposed to use truncated importance weighting to compute low-variance TD targets to stabilize critic learning. Instead, to avoid sudden changes in the critic, Schulman et al. [2016] proposed to constrain the learning of the value function such that the average Kullback-Leibler divergence between the previous and the current value function is sufficiently small. These methods can be categorized as methods that improve the stability by stabilizing the critic.

An alternative approach is to stabilize the actor by forcing it to not change abruptly. This is often done by incorporating a Kullback-Leibler divergence constraint to the actor learning objective. This constraint ensures that the actor does not take a large update step, ensuring safe and stable actor learning [Peters et al., 2010, Schulman et al., 2015, Akrour et al., 2016, Achiam et al., 2017, Nachum et al., 2018].

Our approach differs from both these approaches. Instead of stabilizing either the actor or the critic, we focus on stabilizing the interaction between the two. We do so by penalizing the critic error during the learning of the actor. Our approach directly addresses the instability arising due to the interplay between the actor and the critic.

Prokhorov and Wunsch [1997] proposed a method in a spirit similar to our approach where they only update the actor when the critic is sufficiently accurate. This *delayed* update can stabilize the actor, but it might require many more samples to ensure an accurate critic, which could be time consuming and make the method very slow. Our approach does not have this issue. Another recent approach proposed by Dai et al. [2018] uses a dual method to address the instability due to the interplay between the actor and the critic. In their framework the actor and the critic have competing objectives, while ours encourages cooperation between them.

4.2 Actor-Critic Methods and Their Instability

We start with a description of the reinforcement learning (RL) framework, and then review actor-critic methods. Finally, we discuss the sources of instability considered in this chapter for actor-critic methods.

4.2.1 Reinforcement Learning and Policy Search

We consider RL in an environment governed by a Markov Decision Process (MDP). An MDP is described by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mu_1 \rangle$, where $\mathcal{S} \subseteq \mathbb{R}^{d_{\mathrm{S}}}$ is the state space, $\mathcal{A} \subseteq \mathbb{R}^{d_{\mathrm{A}}}$ is the action space, $\mathcal{P}\left(s' | s, a\right)$ defines a Markovian transition probability density between the current s and the next state s' under action a, $\mathcal{R}\left(s, a\right)$ is the reward function, and μ_1 is initial distribution for state s_1. Given such an environment, the goal of RL is to learn to act. Formally, we want to find a *policy* $\pi(a|s)$ to take an appropriate action when the environment is in state s. By following such a policy starting at initial state s_1, we obtain a sequence of states, actions and rewards $(s_t, a_t, r_t)_{t=1\ldots T}$, where $r_t = \mathcal{R}(s_t, a_t)$ is the reward at time t (T is the total timesteps). We refer to such sequences as *trajectories* or *episodes*. Our goal is to find a policy that maximizes the expected return of such trajectories,

$$\max_{\pi} \; \mathbb{E}_{\mu_\pi(s)\pi(a|s)}\left[Q^\pi(s, a)\right], \tag{4.1}$$

$$\text{where} \quad Q^\pi(s_t, a_t) := \mathbb{E}_{\prod_{i=t+1}^{T} \pi(a_i|s_i)\mathcal{P}(s_{i+1}|s_i, a_i)}\left[\sum_{i=t}^{T} \gamma^{i-t} r_{i+1}\right], \tag{4.2}$$

where $\mu_\pi(s)$ is the state distribution under π, i.e., the probability of visiting state s under π, $Q^\pi(s,a)$ is the action-state value function (or Q-function) which is the expected return obtained by executing a in state s and then following π, and, finally, $\gamma \in [0,1)$ is the discount factor which assigns weights to rewards at different timesteps.

One way to solve the optimization problem of Eq. (4.2) is to use policy search [Deisenroth et al., 2013], e.g., we can use a parameterized policy function $\pi(a|s;\theta)$ with the parameter θ and take the following gradients steps

$$\theta_{i+1} = \theta_i + \alpha_\theta \, \nabla_\theta \, \mathbb{E}_{\mu_\pi(s)\pi(a|s;\theta)}[Q^\pi(s,a)]\big|_{\theta=\theta_i}, \tag{4.3}$$

where $\alpha_\theta > 0$ is the stepsize and i is a learning iteration. There are many ways to compute a stochastic estimate of the above gradient, e.g., we can first collect one trajectory starting from $s_1 \sim \mu_1(s)$, compute Monte Carlo estimates $\widehat{Q}^\pi(s_t,a_t)$ of $Q^\pi(s_t,a_t)$, and then compute the gradient using REINFORCE [Williams, 1992] as

$$\nabla_\theta \, \mathbb{E}_{\mu_\pi(s)\pi(a|s;\theta)}[Q^\pi(s,a)] \approx \sum_{t=1}^{T} [\nabla_\theta \log \pi(a_t|s_t;\theta)] \, \widehat{Q}^\pi(s_t,a_t), \tag{4.4}$$
$$\text{where} \quad \widehat{Q}^\pi(s_t,a_t) = \mathcal{R}(s_t,a_t) + \gamma \widehat{Q}^\pi(s_{t+1},a_{t+1}), \quad \forall t = 1,2,\dots,T-1,$$

and $\widehat{Q}^\pi(s_T,a_T) := \mathcal{R}(s_T,a_T) = r_T$. The recursive update to estimate $\widehat{Q}^\pi$ is due to the definition of Q^π shown in Eq. (4.2). The above stochastic gradient algorithm is guaranteed to converge to a locally optimal policy when the stepsize are chosen according to Robbins-Monro conditions [Robbins and Monro, 1985]. However, in practice, using one trajectory might have high variance and the method requires averaging over many trajectories which could be inefficient. Many solutions have been proposed to solve this problem, e.g., baseline substraction methods [Greensmith et al., 2004, Gu et al., 2016a, Wu et al., 2018]. Actor-critic methods are one of the most popular methods for this purpose.

4.2.2 Instability of Actor-Critic Methods

In actor-critic methods, the value function is estimated using a parameterized function approximator, i.e., $Q^\pi(s,a) \approx \widehat{Q}(s,a;\omega)$, where ω are the parameters of the approximator such as a linear model or a neural network. This estimator is called the *critic* and can have much lower variance than traditional Monte Carlo estimators. Critic's estimate are used to optimize the policy $\pi(a|s;\theta)$, also called the *actor*.

Actor-critic methods alternate between updating the parameters of the actor and the critic. Given the critic parameters ω_i at iteration i and its value function estimate $\widehat{Q}(s,a;\omega_i)$, the actor can be updated using a policy search step similar to Eq. (4.3),

$$\theta_{i+1} = \theta_i + \alpha_\theta \;\; \nabla_\theta \, \mathbb{E}_{\mu_\pi(s)\pi(a|s;\theta)}\left[\widehat{Q}(s,a;\omega_i)\right]\Big|_{\theta=\theta_i}. \tag{4.5}$$

The parameters ω are updated next by using a gradient method, e.g., we can minimize the *temporal difference* (TD) error δ_Q using the following update:

$$\omega_{i+1} = \omega_i + \alpha_\omega \, \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta),\mathcal{P}(s'|s,a)} \left[\delta_Q(s,a,s';\theta_{i+1},\omega_i) \nabla_\omega \widehat{Q}(s,a;\omega)|_{\omega=\omega_i} \right],$$
$$\text{where} \quad \delta_Q(s,a,s';\theta,\omega) := \mathcal{R}(s,a) + \gamma \, \mathbb{E}_{\pi(a'|s',\theta)} \left[\widehat{Q}(s',a';\omega) \right] - \widehat{Q}(s,a;\omega). \qquad (4.6)$$

The above update is approximately equivalent to minimizing the mean square of the TD error [Baird, 1995]. The updates (4.5) and (4.6) together constitute a type of actor-critic method. The actor's goal is to optimize the expected return shown in Eq. (4.2), while the critic's goal is to provide an accurate estimate of the value function.

A variety of options are available for the actor and the critic, e.g., stochastic and deterministic actor-critic methods [Sutton et al., 1999, Silver et al., 2014], trust-region policy optimization methods [Schulman et al., 2015], and proximal policy optimization methods [Schulman et al., 2017]. Flexible approximators, such as deep neural networks, can be used for the actor and the critic. Actor-critic methods exhibit lower variance than the policy gradient methods that use Monte Carlo methods to estimate of the Q-function. They are also more sample efficient. Overall, these methods, when tuned well, can perform extremely well and achieved state-of-the-art performances on many difficult RL problems [Mnih et al., 2015, Silver et al., 2017a].

However, one issue with actor-critic methods is that they can be unstable, and may require careful tuning and engineering to work well [Lillicrap et al., 2016, Dai et al., 2018, Henderson et al., 2017]. For example, deep deterministic policy gradient (DDPG) [Lillicrap et al., 2016] requires implementation tricks such as target networks, and it is known to be highly sensitive to its hyperparameters [Henderson et al., 2017]. Furthermore, convergence is guaranteed only when the critic accurately estimates the value function [Sutton et al., 1999], which could be prohibitively expensive. In general, stabilizing actor-critic methods is an active area of research.

One source of instability, among many others, is the interaction between the actor and the critic. The algorithm alternates between the update of the actor and the critic, so inaccuracies in one update might affect the other adversely. For example, the actor relies on the value function estimates provided by the critic. This estimate can have lower variance than the Monte Carlo estimates used in Eq. (4.4). However, Monte Carlo estimates are unbiased, because they maintain the recursive relationship between $\widehat{Q}^\pi(s_t,a_t)$ and $\widehat{Q}^\pi(s_{t+1},a_{t+1})$ which ensures that the expected value of $\widehat{Q}^\pi(s_t,a_t)$ is equal to the true value function (the expectation is taken with respect to the trajectories). When we use function approximators, it is difficult to satisfy such recursive properties of the value function estimates. Due to this reason, critic estimates are often biased. At times, such inaccuracies might push the actor into wrong directions, from which the actor may never recover. In this chapter, we propose a new method to address instability caused by such bad steps.

4.3 TD-Regularized Actor-Critic

As discussed in the previous section, the critic's estimate of the value function $\widehat{Q}(s,a;\omega)$ might be biased, while Monte Carlo estimates can be unbiased. In general, we can ensure the unbiased property of an estimator if it satisfies the Bellman equation. This is because the following recursion ensures that each $\widehat{Q}(s, a; \omega)$ is equal to the true value function in expectation, as shown below

$$\widehat{Q}(s, a; \omega) = \mathcal{R}(s, a) + \gamma\, \mathbb{E}_{\mathcal{P}(s'|s,a),\pi(a'|s';\theta)} \left[\widehat{Q}(s', a'; \omega)\right], \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}. \tag{4.7}$$

If $\widehat{Q}(s', a'; \omega)$ is unbiased, $\widehat{Q}(s, a; \omega)$ will also be unbiased. Therefore, by induction, all estimates are unbiased. Using this property, we modify actor's learning goal (Eq. (4.2)) as the following constrained optimization problem

$$\max_{\theta}\ \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta)} \left[\widehat{Q}(s, a; \omega)\right], \tag{4.8}$$

$$\text{s.t. } \widehat{Q}(s, a; \omega) = \mathcal{R}(s, a) + \gamma\, \mathbb{E}_{\mathcal{P}(s'|s,a),\pi(a'|s';\theta)} \left[\widehat{Q}(s', a'; \omega)\right], \forall s \in \mathcal{S}, \forall a \in \mathcal{A}. \tag{4.9}$$

We refer to this problem as the *Bellman-constrained policy search*. At the optimum, when $\widehat{Q} = Q^\pi$, the constraint is satisfied, therefore the optimal solution of this problem is equal to the original problem of Eq. (4.3). For a suboptimal critic, the constraint is not satisfied and constrains the maximization of the expected return proportionally to the deviation in the Bellman equation. We expect this to prevent a large update in the actor when the critic is highly inaccurate for some state-action pairs. The constrained formulation is attractive but computationally difficult due to the large number of constraints, e.g., for continuous state and action space this number might be infinite. In what follows, we make three modifications to this problem to obtain a practical method.

4.3.1 Modification 1: Unconstrained Formulation Using the Quadratic Penalty Method

Our first modification is to reformulate the constrained problem as an unconstrained one by using the quadratic penalty method [Nocedal and Wright, 2006]. In this method, given an optimization problem with equality constraints

$$\max_{\theta} f(\theta), \quad \text{s.t. } h_j(\theta) = 0, \quad j = 1, 2, \ldots, M \tag{4.10}$$

we optimize the following function

$$\widehat{f}(\theta, \eta) := f(\theta) - \eta \sum_{j=1}^{M} b_j h_j^2(\theta), \tag{4.11}$$

where b_j are the weights of the equality constraints and can be used to trade-off the effect of each constraint, and $\eta > 0$ is the parameter controlling the trade-off between the original objective function and the penalty function. When $\eta = 0$, the constraint does not matter, while when $\eta \to \infty$, the objective function does not matter. Assuming that ω is fixed, we propose to optimize the following quadratic-penalized version of the Bellman-constrained objective for a given η

$$\begin{aligned} L(\theta,\eta) := {} & \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta)} \left[\widehat{Q}(s,a;\omega)\right] \\ & - \eta \iint b(s,a) \left(\mathcal{R}(s,a) + \gamma \mathbb{E}_{\mathcal{P}(s'|s,a),\pi(a'|s';\theta)} \left[\widehat{Q}(s',a';\omega)\right] - \widehat{Q}(s,a;\omega)\right)^2 \mathrm{d}s\mathrm{d}a, \end{aligned} \tag{4.12}$$

where $b(s, a)$ is the weight of the constraint corresponding to the pair (s, a).

4.3.2 Modification 2: Reducing the Number of Constraints

The integration over the entire state-action space is still computationally infeasible. Our second modification is to focus on few constraints by appropriately choosing the weights $b(s, a)$. A natural choice is to use the state distribution $\mu_\pi(s)$ and the current policy $\pi(a|s; \theta)$ to sample the candidate state-action pairs whose constraints we will focus on. In this way, we get the following objective

$$L(\theta,\eta) := \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta)}\left[\widehat{Q}(s,a;\omega)\right] \\ -\eta\mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta)}\left[\left(\mathcal{R}(s,a)+\gamma\mathbb{E}_{\mathcal{P}(s'|s,a),\pi(a'|s';\theta)}\left[\widehat{Q}(s',a';\omega)\right]-\widehat{Q}(s,a;\omega)\right)^2\right]. \tag{4.13}$$

The expectations in the penalty term in Eq. (4.13) can be approximated using the observed state-action pairs. We can also use the same samples for both the original objective and the penalty term. This can be regarded as a *local* approximation where only a subset of the infinitely many constraint are penalized, and the subset is chosen based on its influence to the original objective function.

4.3.3 Modification 3: Approximation Using the TD Error

The final difficulty is that the expectation over $\mathcal{P}(s'|s, a)$ is inside the square function. This gives rise to a well-known issue in RL, called the *double-sampling* issue [Baird, 1995]. In order to compute an unbiased estimate of this squared expectation over $\mathcal{P}(s'|s, a)$, we require two sets of independent samples of s' sampled from $\mathcal{P}(s'|s, a)$. The independence condition means that we need to independently sample many of the next states s' from an identical state s. This requires the ability to reset the environment back to state s after each transition, which is typically impossible for many real-world systems. Notice that the squared expectation over $\pi(a'|s'; \theta)$ is less problematic since we can always internally sample many actions from the policy without actually executing those actions on the environment.

To address this issue, we propose a final modification where we pull the expectation over $\mathcal{P}(s'|s, a)$ outside the square

$$L(\theta,\eta) := \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta)}\left[\widehat{Q}(s,a;\omega)\right] \\ -\eta\mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta),\mathcal{P}(s'|s,a)}\Big[\underbrace{\Big(\mathcal{R}(s,a)+\gamma\mathbb{E}_{\pi(a'|s';\theta)}\left[\widehat{Q}(s',a';\omega)\right]-\widehat{Q}(s,a;\omega)\Big)}_{:=\delta_Q(s,a,s';\theta,\omega)}{}^2\Big]. \tag{4.14}$$

This step replaces the Bellman constraint of a pair (s, a) by the *temporal difference* (TD) error $\delta_Q(s, a, s'; \theta, \omega)$ defined over the tuple (s, a, s'). We can estimate the TD error by using TD(0) or a batch version of it, thereby resolving the double-sampling issue. To further reduce the bias of TD error estimates, we can also rely on TD(λ) (more details in Section 4.4.5). Note that this final modification only approximately satisfies the Bellman constraint.

4.3.4 Final Algorithm: the TD-Regularized Actor-Critic Method

Eq. (4.14) is the final objective we use to update the actor. For the ease of notation, in the rest of the paper, we refer to the two terms in $L(\theta, \eta)$ using the following notation

$$L(\theta, \eta) := J(\theta) - \eta G(\theta), \quad \text{where} \tag{4.15}$$
$$J(\theta) := \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta)}[\widehat{Q}(s, a; \omega)], \tag{4.16}$$
$$G(\theta) := \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta),\mathcal{P}(s'|s,a)}[\delta_Q(s, a, s'; \theta, \omega)^2]. \tag{4.17}$$

We propose to replace the usual policy search step (Eq. (4.5)) in the actor-critic method by a step that optimizes the TD-regularized objective for a given η_i in iteration i,

$$\theta_{i+1} = \theta_i + \alpha_\theta \left(\nabla_\theta \, \mathbb{E}_{\mu_\pi(s)\pi(a|s;\theta)} \left[\widehat{Q}(s, a; \omega_i) \right] \Big|_{\theta=\theta_i} - \eta_i \nabla_\theta G(\theta) \Big|_{\theta=\theta_i} \right). \tag{4.18}$$

The blue term is the extra penalty term involving the TD error, where we allow η_i to change with the iteration. We can alternate between the above update of the actor and the update of the critic, e.g., by using Eq. (4.6) or any other method. We call this method the *TD-regularized* actor-critic. We use the terminology "regularization" instead of "penalization" since it is more common in the RL and machine learning communities.

A simple interpretation of Eq. (4.14) is that the actor is penalized for increasing the squared TD error, implying that the update of the actor favors policies achieving a small TD error. The main objective of the actor is still to maximize the estimated expected returns, but the penalty term helps to avoid bad updates whenever the critic's estimate has a large TD error. Because the TD error is an approximation to the deviation from the Bellman equation, we expect that the proposed method helps in stabilizing learning whenever the critic's estimate incurs a large TD error.

In practice, the choice of penalty parameter is extremely important to enable a good trade-off between maximizing the expected return and avoiding the bias in the critic's estimate. In a typical optimization problem where the constraints are only functions of θ, it is recommended to slowly increase η_i with i. This way, as the optimization progresses, the constraints become more and more important. However, in our case, the constraints also depend on ω which changes with iterations, therefore the constraints also change with i. As long as the overall TD error of the critic decreases as the number of iterations increase, the overall penalty due the constraint will eventually decrease too. Therefore, we do not need to artificially make the constraints more important by increasing η_i. In practice, we found that if the TD error decreases over time, then η_i can, in fact, be decreased with i. In Section 4.5, we use a simple decaying rule $\eta_{i+1} = \kappa\eta_i$ where $0 < \kappa < 1$ is a decay factor.

4.4 Application of TD-Regularization to Existing Actor-Critic Methods

Our TD-regularization method is a general plug-and-play method that can be applied to any actor-critic method that performs policy gradient for actor learning. In this section, we first demonstrate its applications to popular actor-critic methods including DPG [Silver et al., 2014], TRPO [Schulman et al., 2015] and PPO [Schulman et al., 2017]. Subsequently, building upon the TD(λ) error, we present a second regularization that can be used by actor-critic methods doing advantage learning, such as GAE [Schulman et al., 2016]. For all algorithms, we show that our method only slightly increases computation time. The required gradients can be easily computed using automatic differentiation, making it very easy to apply our method to existing actor-critic methods. Empirical comparison on these methods are given in Section 4.5.

4.4.1 TD-Regularized Stochastic Policy Gradient (SPG)

For a stochastic policy $\pi(a|s;\theta)$, the gradient of Eq. (4.14) can be computed using the chain rule and log-likelihood ratio trick [Williams, 1992, Sutton and Barto, 1998]. Specifically, the gradients of TD-regularized stochastic actor-critic are given by

$$\nabla_\theta J(\theta) = \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta)}\left[\nabla_\theta \log \pi(a|s;\theta)\widehat{Q}(s,a;\omega)\right], \tag{4.19}$$

$$\begin{aligned}\nabla_\theta G(\theta) = \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta),\mathcal{P}(s'|s,a)}&\Big[\nabla_\theta \log \pi(a|s;\theta)\delta_Q(s,a,s';\theta,\omega)^2 \\ &+ 2\gamma\mathbb{E}_{\pi(a'|s';\theta)}\left[\nabla_\theta \log \pi(a'|s';\theta)\delta_Q(s,a,s';\theta,\omega)\widehat{Q}(s',a';\omega)\right]\Big].\end{aligned} \tag{4.20}$$

When compared to the standard SPG method, TD-regularized SPG requires only extra computations to compute $\nabla_\theta \log \pi(a'|s';\theta)$ and $\delta_Q(s,a,s';\theta,\omega)$.

4.4.2 TD-Regularized Deterministic Policy Gradient (DPG)

DPG [Silver et al., 2014] is similar to SPG but learns a deterministic policy $a = \pi(s;\theta)$. To allow exploration and collect samples, DPG uses a behavior policy $\beta(a|s)$[2]. Common examples are ϵ-greedy policies or Gaussian policies. Consequently, the state distribution $\mu_\pi(s)$ is replaced by $\mu_\beta(s)$ and the expectation over the policy $\pi(a|s;\theta)$ in the regularization term is replaced by an expectation over a behavior policy $\beta(a|s)$. The TD error does not change, but the expectation over $\pi(a'|s',\theta)$ disappears. TD-regularized DPG components are

$$J_{\text{DPG}}(\theta) := \mathbb{E}_{\mu_\beta(s)}\left[\widehat{Q}(s,\pi(s;\theta);\omega)\right], \tag{4.21}$$

$$G_{\text{DPG}}(\theta) := \mathbb{E}_{\mu_\beta(s),\beta(a|s),\mathcal{P}(s'|s,a)}\left[\delta_Q(s,a,s';\theta,\omega)^2\right], \tag{4.22}$$

$$\delta_Q(s,a,s';\theta,\omega) := \mathcal{R}(s,a) + \gamma\widehat{Q}(s',\pi(s';\theta);\omega) - \widehat{Q}(s,a;\omega). \tag{4.23}$$

Their gradients can be computed by the chain rule and are given by

$$\nabla_\theta J_{\text{DPG}}(\theta) = \mathbb{E}_{\mu_\beta(s)}\left[\nabla_\theta\pi(s;\theta)\nabla_a\widehat{Q}(s,a;\omega)\big|_{a=\pi(s;\theta)}\right], \tag{4.24}$$

$$\nabla_\theta G_{\text{DPG}}(\theta) = 2\gamma\,\mathbb{E}_{\mu_\beta(s),\beta(a|s),\mathcal{P}(s'|s,a)}\left[\delta_Q(s,a,s';\theta,\omega)\nabla_\theta\pi(s';\theta)\nabla_{a'}\widehat{Q}(s',a';\omega)\big|_{a'=\pi(s';\theta)}\right].$$

The gradient of the regularization term requires extra computations to compute $\nabla_\theta\pi(s';\theta)$, $\nabla_{a'}\widehat{Q}(s',a';\omega)$ and $\delta_Q(s,a,s';\theta,\omega)$.

[2] Silver et al. [2014] showed that DPG can be more advantageous than SPG as deterministic policies have lower variance. However, the behavior policy has to be chosen appropriately.

4.4.3 TD-Regularized Trust-Region Policy Optimization (TRPO)

In the previous two examples, the critic estimates the Q-function. In this section, we demonstrate an application to a case where the critic estimates the V-function. The V-function of π is defined as $V^\pi(s) := \mathbb{E}_{\pi(a|s)}[Q^\pi(s,a)]$, and satisfies the following Bellman equation

$$V^\pi(s) = \mathbb{E}_{\pi(a|s)}[\mathcal{R}(s,a)] + \gamma\, \mathbb{E}_{\pi(a|s),\mathcal{P}(s'|s,a)}[V(s')]\,, \qquad \forall s \in \mathcal{S}. \tag{4.25}$$

The TD error for a critic $\widehat{V}(s;\omega)$ with parameters ω is

$$\delta_V(s,s';\omega) := \mathcal{R}(s,a) + \gamma \widehat{V}(s';\omega) - \widehat{V}(s;\omega). \tag{4.26}$$

One difference compared to previous two sections is that $\delta_V(s,s';\omega)$ does not directly contain $\pi(a|s;\theta)$. We will see that this greatly simplifies the update. Nonetheless, the TD error still depends on $\pi(a|s;\theta)$ as it requires to sample the action a to reach the next state s'. Therefore, the TD-regularization can still be applied to stabilize actor-critic methods that use a V-function critic.

In this section, we regularize TRPO [Schulman et al., 2015], which uses a V-function critic and solves the following optimization problem

$$\max_\theta \; L_{\text{TRPO}}(\theta,\eta) := J_{\text{TRPO}}(\theta) - \eta G_{\text{TRPO}}(\theta), \qquad \text{s.t. } \mathbb{E}_{\mu_\pi(s)}[\mathrm{KL}(\pi||\pi_{\text{old}})] \le \epsilon, \tag{4.27}$$

$$\text{where} \quad J_{\text{TRPO}}(\theta) := \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta)}\left[\rho(\theta)\widehat{A}(s,a;\omega)\right], \tag{4.28}$$

$$G_{\text{TRPO}}(\theta) := \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta),\mathcal{P}(s'|s,a)}\left[\rho(\theta)\delta_V(s,s';\omega)^2\right], \tag{4.29}$$

where $\rho(\theta) = \pi(a|s;\theta)/\pi(a|s;\theta_{\text{old}})$ are importance weights. KL is the Kullback-Leibler divergence between the new learned policy $\pi(a|s;\theta)$ and the old one $\pi(a|s;\theta_{\text{old}})$, and helps in ensuring small policy updates. $\widehat{A}(s,a;\omega)$ is an estimate of the advantage function $A^\pi(s,a) := Q^\pi(s,a) - V^\pi(s)$ computed by learning a V-function critic $\widehat{V}(s;\omega)$ and approximating $Q^\pi(s,a)$ either by Monte Carlo estimates or from $\widehat{V}(s;\omega)$ as well. We come back to this in Section 4.4.5. The gradients of $J_{\text{TRPO}}(\theta)$ and $G_{\text{TRPO}}(\theta)$ are

$$\nabla_\theta J_{\text{TRPO}}(\theta) = \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta)}\left[\rho(\theta)\nabla_\theta \log\pi(a|s;\theta)\widehat{A}(s,a;\omega)\right], \tag{4.30}$$

$$\nabla_\theta G_{\text{TRPO}}(\theta) = \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta),\mathcal{P}(s'|s,a)}\left[\rho(\theta)\nabla_\theta \log\pi(a|s;\theta)\delta_V(s,s';\omega)^2\right]. \tag{4.31}$$

The extra computation for TD-regularized TRPO only comes from computing the square of the TD error $\delta_V(s,s';\omega)^2$.

Notice that, due to linearity of expectations, TD-regularized TRPO can be understood as performing the standard TRPO with a TD-regularized advantage function $\widehat{A}_\eta(s,a;\omega) := \widehat{A}(s,a;\omega) - \eta\mathbb{E}_{\mathcal{P}(s'|s,a)}[\delta_V(s,s';\omega)^2]$. This greatly simplifies implementation of our TD-regularization method. In particular, TRPO performs natural gradient ascent to approximately solve the KL constraint optimization problem[3]. By viewing TD-regularized TRPO as TRPO

[3] Natural gradient ascent on a function $f(\theta)$ updates the function parameters θ by $\theta \leftarrow \theta + \alpha_\theta \boldsymbol{F}^{-1}(\theta)\boldsymbol{g}(\theta)$, where $g(\theta)$ is the gradient and $\boldsymbol{F}(\theta)$ is the Fisher information matrix.

with regularized advantage, we can use the same natural gradient procedure for TD-regularized TRPO.

4.4.4 TD-Regularized Proximal Policy Optimization (PPO)

PPO [Schulman et al., 2017] simplifies the optimization problem of TRPO by removing the KL constraint, and instead uses clipped importance weights and a pessimistic bound on the advantage function

$$\max_\theta \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta)}\Big[\min\{\rho(\theta)\widehat{A}(s,a;\omega),\ \rho_\varepsilon(\theta)\widehat{A}(s,a;\omega)\}\Big], \tag{4.32}$$

where the $\rho_\varepsilon(\theta)$ is the importance ratio $\rho(\theta)$ clipped between $[1-\varepsilon, 1+\varepsilon]$ and $0<\varepsilon<1$ represents the update stepsize (the smaller ε, the more conservative the update is). By clipping the importance ratio, we remove the incentive for moving $\rho(\theta)$ outside of the interval $[1-\varepsilon, 1+\varepsilon]$, i.e., for moving the new policy far from the old one. By taking the minimum between the clipped and the unclipped advantage, the final objective is a lower bound (i.e., a pessimistic bound) on the unclipped objective.

Similarly to TRPO, the advantage function $\widehat{A}(s,a;\omega)$ is computed using a V-function critic $\widehat{V}(s;\omega)$, thus we could simply use the regularization in Eq. (4.29). However, the TD-regularization would not benefit from neither importance clipping nor the pessimistic bound, which together provide a way of performing small safe policy updates. For this reason, we propose to modify the TD-regularization as follows

$$\max_\theta L_{\text{PPO}}(\theta,\eta) := J_{\text{PPO}}(\theta) - \eta G_{\text{PPO}}(\theta), \qquad \text{where} \tag{4.33}$$

$$J_{\text{PPO}}(\theta) := \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta)}\Big[\min\{\rho(\theta)\widehat{A}(s,a;\omega),\ \rho_\varepsilon(\theta)\widehat{A}(s,a;\omega)\}\Big], \tag{4.34}$$

$$G_{\text{PPO}}(\theta) := \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta),\mathcal{P}(s'|s,a)}\big[\max\{\rho(\theta)\delta_V(s,s';\omega)^2,\ \rho_\varepsilon(\theta)\delta_V(s,s';\omega)^2\}\big], \tag{4.35}$$

i.e., we apply importance clipping and the pessimistic bound also to the TD-regularization. The gradients of $J_{\text{PPO}}(\theta)$ and $G_{\text{PPO}}(\theta)$ can be computed as

$$\nabla_\theta J_{\text{PPO}}(\theta) = \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta)}[f(\theta)] \qquad \text{where} \tag{4.36}$$

$$f(\theta) := \begin{cases} \rho(\theta)\nabla_\theta \log\pi(a|s;\theta)\widehat{A}(s,a;\omega) & \text{if } \rho(\theta)\widehat{A}(s,a;\omega) < \rho_\varepsilon(\theta)\widehat{A}(s,a;\omega) \\ 0 & \text{otherwise,} \end{cases}$$

$$\nabla_\theta G_{\text{PPO}}(\theta) = \mathbb{E}_{\mu_\pi(s),\pi(a|s;\theta)}[g(\theta)] \qquad \text{where} \tag{4.37}$$

$$g(\theta) := \begin{cases} \rho(\theta)\nabla_\theta \log\pi(a|s;\theta)\delta_V(s,s';\omega)^2 & \text{if } \rho(\theta)\delta_V(s,s';\omega)^2 > \rho_\varepsilon(\theta)\delta_V(s,s';\omega)^2 \\ 0 & \text{otherwise.} \end{cases}$$

4.4.5 GAE-Regularization

In Sections 4.4.3 and 4.4.4, we have discussed how to apply the TD-regularization when a V-function critic is learned. The algorithms discussed, TRPO and PPO, maximize the advantage function $\widehat{A}(s,a;\omega)$ estimated using a V-function critic $\widehat{V}(s,\omega)$. Advantage learning has a long history in RL literature [Baird, 1993] and one of the most used and successful advantage estimator is the generalized advantage estimator (GAE) [Schulman et al., 2016]. In this section, we build a connection between GAE and the well-known TD(λ) method [Sutton and Barto, 1998] to propose a different regularization, which we call the *GAE-regularization*. We show that this regularization is very convenient for algorithms already using GAE, as it does not introduce any computational cost, and has interesting connections with other RL methods.

Let the n-step return $R_{t:t+n}$ be the sum of the first n discounted rewards plus the estimated value of the state reached in n steps, i.e.,

$$R_{t:t+n} := r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n \widehat{V}(s_{t+n};\omega), \quad 0 \leq t \leq T-n, \tag{4.38}$$

$$R_{t:T+1} := \sum_{i=t}^{T} \gamma^{i-t} r_i. \tag{4.39}$$

The full-episode return $R_{t:T+1}$ is a Monte Carlo estimate of the value function. The idea behind TD(λ) is to replace the TD error target $r_t + \gamma \widehat{V}(s_{t+1},\omega)$ with the average of the n-step returns, each weighted by λ^{n-1}, where $\lambda \in [0,1]$ is a decay rate. Each n-step return is also normalized by $1-\lambda$ to ensure that the weights sum to 1. The resulting TD(λ) targets are the so-called λ-returns

$$R_t^\lambda := (1-\lambda) \sum_{i=t}^{T-1} \lambda^{i-t} R_{t:i+1} + \lambda^{T-t} R_{t:T+1}, \tag{4.40}$$

and the corresponding TD(λ) error is

$$\delta_V^\lambda(s_t, s_{t+1};\omega) := R_t^\lambda - \widehat{V}(s_t;\omega). \tag{4.41}$$

From the above equations, we see that if $\lambda = 0$, then the λ-return is the usual TD target, i.e., $R_t^0 = r_t + \gamma \widehat{V}(s_{t+1};\omega)$. If $\lambda = 1$, then $R_t^1 = R_{t:T+1}$ as in Monte Carlo methods. In between are intermediate methods that control the bias-variance trade-off between TD and Monte Carlo estimators by varying λ. As discussed in Section 4.2, in fact, TD estimators are biased, while Monte Carlo are not. The latter, however, have higher variance.

Motivated by the same bias-variance trade-off, we propose to replace δ_V with δ_V^λ in Eq. (4.29) and (4.35), i.e., to perform TD(λ)-regularization. Interestingly, this regularization is equivalent to regularize with the GAE advantage estimator, as shown in the following. Let δ_V be an approximation of the advantage function [Schulman et al., 2016]. Similarly to the λ-return, we can define the n-step advantage estimator

$$\begin{aligned} A_{t:t+n} &:= \delta_V(s_t, s_{t+1};\omega) + \gamma \delta_V(s_{t+1}, s_{t+2};\omega) + \ldots + \gamma^{n-1} \delta_V(s_{t+n-1}, s_{t+n};\omega), \\ &= R_{t:t+n} - \widehat{V}(s_t;\omega), \qquad 0 \leq t \leq T-n, \end{aligned} \tag{4.42}$$

$$A_{t:T+1} := R_{t:T+1} - \widehat{V}(s_t;\omega). \tag{4.43}$$

Following the same approach of TD(λ), GAE advantage estimator uses exponentially weighted averages of n-step advantage estimators

$$\widehat{A}^{\lambda}(s_t, a_t; \omega) := (1-\lambda)\sum_{i=t}^{T-1} \lambda^{i-t} A_{t:i+1} + \lambda^{T-t} A_{t:T+1}, \tag{4.44}$$

From the above equation, we see that GAE estimators are discounted sums of TD errors. Similarly to TD(λ), if $\lambda = 0$ then the advantage estimate is just the TD error estimate, i.e., $\widehat{A}^0(s_t, a_t; \omega) = r_t + \widehat{V}(s_{t+1}; \omega) - \widehat{V}(s_t; \omega)$. If $\lambda = 1$ then the advantage estimate is the difference between the Monte Carlo estimate of the return and the V-function estimate, i.e., $\widehat{A}^1(s_t, a_t; \omega) = R_{t:T+1} - \widehat{V}(s_t; \omega)$. Finally, plugging Eq. (4.42) into Eq. (4.44), we can rewrite the GAE estimator as

$$\begin{aligned}
\widehat{A}^{\lambda}(s_t, a_t; \omega) &:= (1-\lambda)\sum_{i=t}^{T-1} \lambda^{i-t}(R_{t:i+1} - \widehat{V}(s_t; \omega)) + \lambda^{T-t}(R_{t:T+1} - \widehat{V}(s_t; \omega)) \\
&= (1-\lambda)\sum_{i=t}^{T-1} \lambda^{i-t} R_{t:i+1} + \lambda^{T-t} R_{t:T+1} - \widehat{V}(s_t; \omega) \\
&= R_t^{\lambda} - \widehat{V}(s_t; \omega) = \delta_V^{\lambda}(s_t, s_{t+1}; \omega),
\end{aligned} \tag{4.45}$$

i.e., the GAE advantage estimator is equivalent to the TD(λ) error estimator. Therefore, using the TD(λ) error to regularize actor-critic methods is equivalent to regularize with the GAE estimator, yielding the following quadratic penalty

$$G_{\text{GAE}}(\theta) = \mathbb{E}_{\mu_\pi(s), \pi(a|s;\theta)}\left[\widehat{A}^{\lambda}(s, a; \omega)^2\right], \tag{4.46}$$

which we call the *GAE-regularization*. The GAE-regularization is very convenient for methods which already use GAE, such as TRPO and PPO[4], as it does not introduce any computational cost. Furthermore, the decay rate λ allows to tune the bias-variance trade-off between TD and Monte Carlo methods[5]. In Section 4.5 we present an empirical comparison between the TD- and GAE-regularization.

Finally, the GAE-regularization has some interesting interpretations. As shown by Belousov and Peters [2017], minimizing the squared advantage is equivalent to maximizing the average reward with a penalty over the Pearson divergence between the new and old state-action distribution $\mu_\pi(s)\pi(a|s;\theta)$, and a hard constraint to satisfy the stationarity condition $\iint \mu_\pi(s)\pi(a|s;\theta)\mathcal{P}(s'|s,a)\,\mathrm{d}s\mathrm{d}a = \mu_\pi(s'), \forall s'$. The former is to avoid overconfident policy update steps, while the latter is the dual of the Bellman equation (Eq. (4.7)). Recalling that the GAE-regularization approximates the Bellman equation constraint with the TD(λ) error, the two methods are very similar. The difference in the policy update is that the GAE-regularization replaces the stationarity condition with a soft constraint, i.e., the penalty[6].

4 For PPO, we also apply the importance clipping and pessimistic bound proposed in Eq. (4.35).

5 We recall that, since GAE approximates $A^{\pi}(s,a)$ with the TD(λ) error, we are performing the same approximation presented in Section 4.3.3, i.e., we are still approximately satisfying the Bellman constraint.

6 For the critic update, instead, Belousov and Peters [2017] learn the V-function parameters together with the policy rather than separately as in actor-critic methods.

Eq. (4.46) is equivalent to minimizing the variance of the centered GAE estimator, i.e., $\mathbb{E}[(\widehat{A}^\lambda(s,a;\omega) - \mu_{\hat{A}})^2] = \mathrm{Var}[\widehat{A}^\lambda(s,a;\omega)]$. Maximizing the mean of the value function estimator and penalizing its variance is a common approach in risk-averse RL called *mean-variance* optimization [Tamar et al., 2012]. Similarly to our method, this can be interpreted as a way to avoid overconfident policy updates when the variance of the critic is high. By definition, in fact, the expectation of the true advantage function of any policy is zero[7], thus high-variance is a sign of an inaccurate critic.

4.5 Evaluation

We propose three evaluations. First, we study the benefits of the TD-regularization in the 2-dimensional linear-quadratic regulator (LQR). In this domain we can compute the true Q-function, expected return, and TD error in closed form, and we can visualize the policy parameter space. We begin this evaluation by setting the initial penalty coefficient to $\eta_0 = 0.1$ and then decaying it according to $\eta_{i+1} = \kappa\eta_i$ where $\kappa = 0.999$. We then investigate different decaying factors κ and the behavior of our approach in the presence of non-uniform noise. The algorithms tested are DPG and SPG. For DPG, we also compare to the twin delayed version proposed by Fujimoto et al. [2018], which achieved state-of-the-art results.

The second evaluation is performed on the single- and double-pendulum swing-up tasks [Yoshikawa, 1990, Brockman et al., 2016]. Here, we apply the proposed TD- and GAE-regularization to TRPO together with and against Retrace [Munos et al., 2016] and double-critic learning [van Hasselt, 2010], both state-of-the-art techniques to stabilize the learning of the critic.

The third evaluation is performed on OpenAI Gym [Brockman et al., 2016] continuous control benchmark tasks with the MuJoCo physics simulator [Todorov et al., 2012] and compares TRPO and PPO with their TD- and GAE-regularized counterparts. Due to time constraints, we were not able to evaluate Retrace on this tasks as well. Details of the hyperparameter settings are given in Appendix D.3.

For the LQR and the pendulum swing-up tasks, we tested each algorithm over 50 trials with fixed random seeds. At each iteration, we turned the exploration off and evaluated the policy over several episodes. Due to limited computational resources, we tested MuJoCo experiments over five trials with fixed random seeds. For TRPO, the policy was evaluated over 20 episodes without exploration noise. For PPO, we used the same samples collected during learning, i.e., including exploration noise.

4.5.1 2D Linear Quadratic Regulator (LQR)

The LQR problem is defined by the following discrete-time dynamics

$$s' = As + Ba + \mathcal{N}(0, 0.1^2), \qquad a = Ks, \qquad \mathcal{R}(s,a) = -s^\intercal X s - a^\intercal Y a,$$

where $A, B, X, Y \in \mathbb{R}^{d\times d}$, X is a symmetric positive semidefinite matrix, Y is a symmetric positive definite matrix, and $K \in \mathbb{R}^{d\times d}$ is the control matrix. The policy parameters we want to learn are $\theta = \mathrm{vec}(K)$. Although low-dimensional, this problem presents some challenges. First, the policy can easily make the system unstable. The LQR, in fact, is stable only if the matrix

[7] $\mathbb{E}_{\pi(a|s)}[A^\pi(s,a)] = \mathbb{E}_{\pi(a|s)}[Q^\pi(s,a) - V^\pi(s)] = \mathbb{E}_{\pi(a|s)}[Q^\pi(s,a)] - V^\pi(s) = V^\pi(s) - V^\pi(s) = 0.$

$(A + BK)$ has eigenvalues of magnitude smaller than one. Therefore, small stable steps have to be applied when updating the policy parameters, in order to prevent divergence. Second, the reward is unbounded and the expected negative return can be extremely large, especially at the beginning with an initial random policy. As a consequence, with a common zero-initialization of the Q-function, the initial TD error can be very large. Third, states and actions are unbounded and cannot be normalized in [0,1], a common practice in RL.

Furthermore, the LQR is particularly interesting because we can compute in closed form both the expected return and the Q-function, being able to easily assess the quality of the evaluated algorithms. More specifically, the Q-function is quadratic in the state and in the action, i.e.,

$$Q^\pi(s,a) = Q_0 + s^\mathsf{T} Q_{ss} s + a^\mathsf{T} Q_{aa} a + s^\mathsf{T} Q_{sa} a,$$

where $Q_0, Q_{ss}, Q_{aa}, Q_{sa}$ are matrices computed in closed form given the MDP characteristics and the control matrix K. To show that actor-critic algorithms are prone to instability in the presence of function approximation error, we approximate the Q-function linearly in the parameters $\widehat{Q}(s, a; \omega) = \phi(s, a)^\mathsf{T} \omega$, where $\phi(s, a)$ includes linear, quadratic and cubic features.

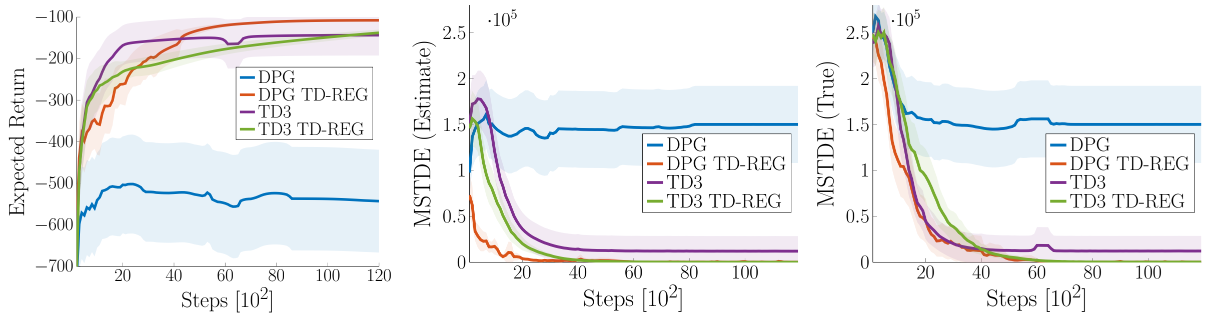
Along with the expected return, we show the trend of two mean squared TD errors (MSTDE): one is estimated using the currently learned $\widehat{Q}(s, a; \omega)$, the other is computed in closed form using the true $Q^\pi(s, a)$ defined above. It should be noticed that $Q^\pi(s, a)$ is not the optimal Q-function (i.e., of the optimal policy), but the true Q-function with respect to the current policy. For details of the hyperparameters and an in-depth analysis, including an evaluation of different Q-function approximators, we refer to Appendix D.1.

Evaluation of DPG and SPG

DPG and TD-regularized DPG (DPG TD-REG) follow the equations presented in Section 4.4.2. The difference is that DPG maximizes only Eq. (4.21), while DPG TD-REG objective includes Eq. (4.22). TD3 is the twin delayed version of DPG presented by Fujimoto et al. [2018], which uses two critics and delays policy updates. TD3 TD-REG is its TD-regularized counterpart. For all algorithms, all gradients are optimized by ADAM [Kingma and Ba, 2014]. After 150 steps, the state is reset and a new trajectory begins.

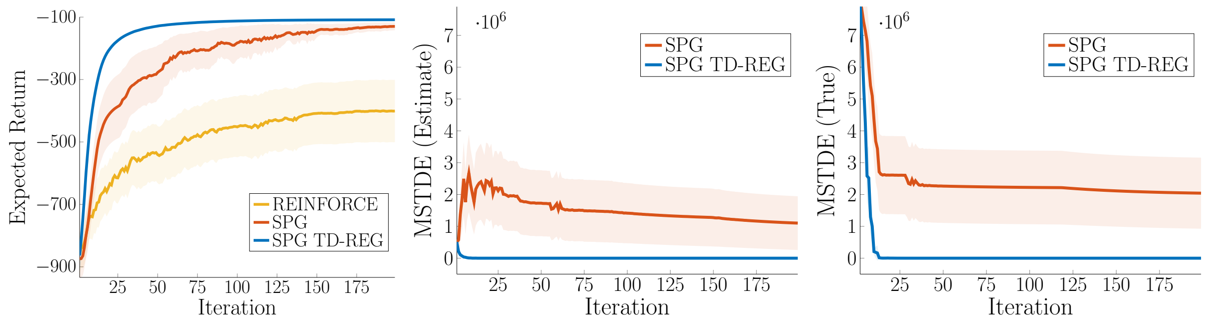
As expected, because the Q-function is approximated with also cubic features, the critic is prone to overfit and the initial TD error is very large. Furthermore, the true TD error (Figure 4.2c) is more than twice the one estimated by the critic (Figure 4.2b), meaning that the critic underestimates the true TD error. Because of the incorrect estimation of the Q-function, vanilla DPG diverges 24 times out of 50. TD3 performs substantially better, but still diverges two times. By contrast, TD-REG algorithms never diverges. Interestingly, only DPG TD-REG always converges to the true critic and to the optimal policy within the time limit, while TD3 TD-REG improves more slowly. Figure 4.2 hints that this "slow learning" behavior may be due to the delayed policy update, as both the estimated and the true TD error are already close to zero by mid-learning. In Appendix D.1 we further investigate this behavior and show that TD3 policy update delay is unnecessary if TD-REG is used. The benefits of TD-REG in the policy space can also be seen in Figure 4.1. Whereas vanilla DPG falls victim to the wrong critic estimates and diverges, DPG TD-REG enables more stable updates.

The strength of the proposed TD-regularization is also confirmed by its application to SPG, as seen in Figure 4.3. Along with SPG and SPG TD-REG, we evaluated REINFORCE [Williams,



(a) (b) (c)

Figure 4.2: DPG comparison on the LQR. Shaded areas denote 95% confidence interval. DPG diverged 24 times out of 50, thus explaining its very large confidence interval. TD3 diverged twice, while TD-regularized algorithms never diverged. Only DPG TD-REG, though, always learned the optimal policy within the time limit.



(a) (b) (c)

Figure 4.3: SPG comparison on the LQR. One iteration corresponds to 150 steps. REINFORCE does not appear in the TD error plots as it does not learn any critic. SPG TD-REG shows an incredibly fast convergence in all runs. SPG-TD, instead, needs much more iterations to learn the optimal policy, as its critic has a much larger TD error. REINFORCE diverged 13 times out of 50, thus explaining its large confidence interval.

1992], which does not learn any critic and just maximizes Monte Carlo estimates of the Q-function, i.e., $\widehat{Q}^\pi(s_t, a_t) = \sum_{i=t}^{T} \gamma^{i-t} r_i$. For all three algorithms, at each iteration samples from only one trajectory of 150 steps are collected and used to compute the gradients, which are then normalized. For the sake of completeness, we also tried to collect more samples per iteration, increasing the number of trajectories from one to five. In this case, all algorithms performed better, but still neither SPG nor REINFORCE matched SPG TD-REG, as they both needed several samples more than SPG TD-REG. More details in Appendix D.1.2.

Analysis of the TD-Regularization Coefficient η

In Section 4.3.4 we have discussed that Eq. (4.14) is the result of solving a constrained optimization problem by penalty function methods. In optimization, we can distinguish two approaches to apply penalty functions [Boyd and Vandenberghe, 2004]. *Exterior* penalty methods start at optimal but infeasible points and iterate to feasibility as $\eta \to \infty$. By contrast, *interior* penalty methods start at feasible but sub-optimal points and iterate to optimality as $\eta \to 0$. In actor-

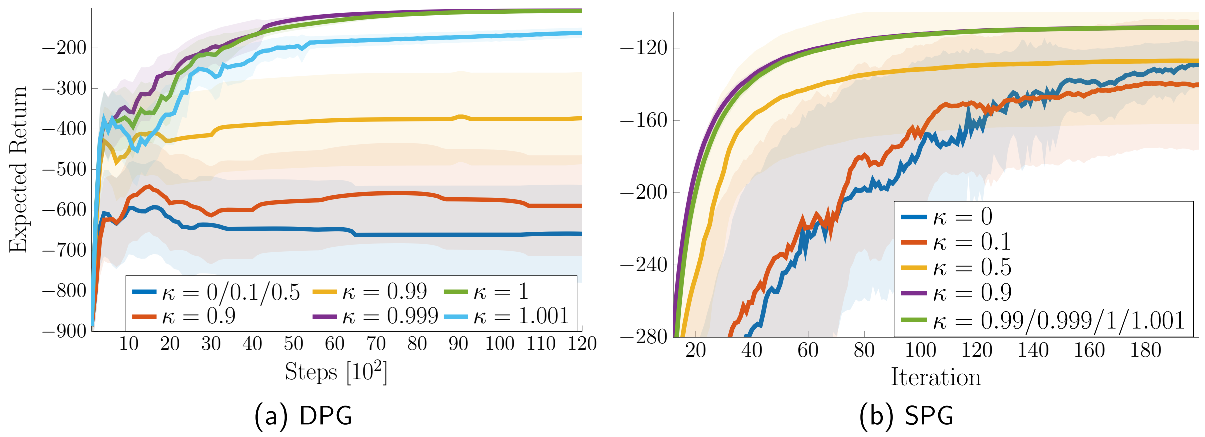


Figure 4.4: Comparison of different values of κ. Shaded areas denote 95% confidence interval. In order to provide enough regularization, κ must be sufficiently large during the whole learning. With small values η vanishes and the TD-regularization is not in effect anymore.

critic, we usually start at infeasible points, as the critic is not learned and the TD error is very large. However, unlike classical constrained optimization, the constraint changes at each iteration, because the critic is updated to minimize the same penalty function. This trend emerged from the results presented in Figures 4.2 and 4.3, showing the change of the mean squared TD error, i.e., the penalty.

In the previous experiments we started with a penalty coefficient $\eta_0 = 0.1$ and decreased it at each policy update according to $\eta_{t+1} = \kappa\eta_t$, with $\kappa = 0.999$. In this section, we provide a comparison of different values of κ, both as decay and growth factor. In all experiments we start again with $\eta_0 = 0.1$ and we test the following κ: 0, 0.1, 0.5, 0.9, 0.99, 0.999, 1, 1.001.

As shown in Figures 4.4a and 4.4b, results are different for DPG TD-REG and SPG TD-REG. In DPG TD-REG, 0.999 and 1 allowed to always converge to the optimal policy. Smaller κ did not provide sufficient help, up to the point where 0.1 and 0.5 did not provide any help at all. However, it is not true that larger κ yield better results, as with 1.001 performance decreases. This is expected, since by increasing η we are also increasing the magnitude of the gradient, which then leads to excessively large and unstable updates.

Results are, however, different for SPG TD-REG. First, 0.99, 0.999, 1, 1.001 all achieve the same performance. The reason is that gradients are normalized, thus the size of the update step cannot be excessively large and $\kappa > 1$ does not harm the learning. Second, 0.9, which was not able to help enough DPG, yields the best results with a slightly faster convergence. The reason is that DPG performs a policy update at each step of a trajectory, while SPG only at the end. Thus, in DPG η, which is updated after a policy update, will decay too quickly if a small κ is used.

Analysis of Non-Uniform Observation Noise

So far, we have considered the case of high TD error due to an overfitting critic and noisy transition function. However, the critic can be inaccurate also because of noisy or partially observable state. Learning in the presence of noise is a long-studied problem in RL literature. To address every aspect of it and to provide a complete analysis of different noises is out of the scope of this chapter. However, given the nature of our approach, it is of particular interest to

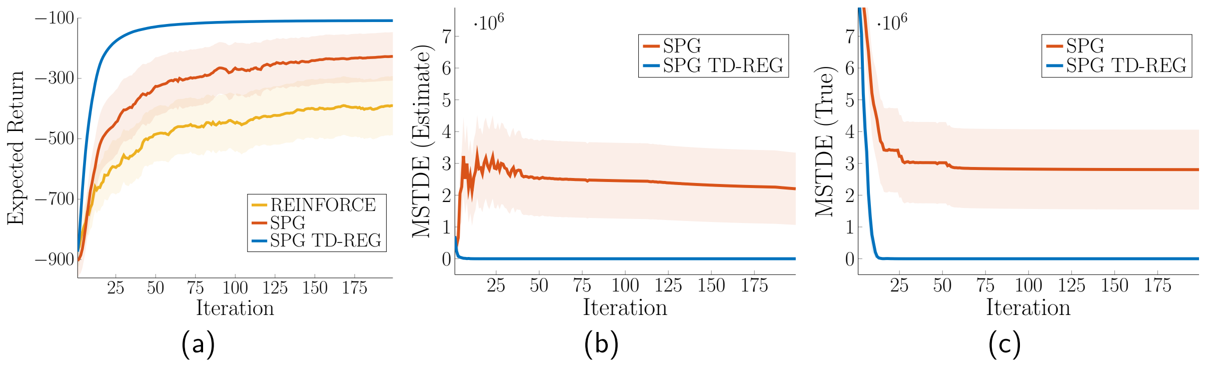


Figure 4.5: SPG comparison on the LQR with non-uniform noise on the state observation. Shaded areas denote 95% confidence interval. The TD error is not shown for REINFORCE as it does not learn any critic. Once again, SPG TD-REG performs the best and is not affected by the noise. Instead of being drawn to low-noise regions (which are far from the goal and correspond to low-reward regions), its actor successfully learns the optimal policy in all trials and its critic achieves a TD error of zero. Un-regularized SPG, which did not diverge in Figure 4.3a, here diverges six times.

analyze the effects of the TD-regularization in the presence of non-uniformly distributed noise in the state space. In fact, since the TD-regularization penalizes for high TD error, the algorithm could be drawn towards low-noise regions of the space in order to avoid high prediction errors. Intuitively, this may not be always a desirable behavior. Thus, in this section we evaluate SPG TD-REG when non-uniform noise is added to the observation of the state

$$s_{\text{OBS}} = s_{\text{TRUE}} + \frac{\mathcal{N}(0, 0.05)}{\texttt{clip}(s_{\text{TRUE}}, 0.1, 200)},$$

where s_{OBS} is the state observed by the actor and the critic, and s_{TRUE} is the true state. The clipping between $[0.1, 200]$ is for numerical stability. The noise is Gaussian and inversely proportional to the state. Since the goal of the LQR is to reach $s_{\text{TRUE}} = 0$, near the goal the noise will be larger and, subsequently, the TD error as well. One may therefore expect that SPG TD-REG would lead the actor towards low-noise regions, i.e., away from the goal. However, as shown in Figure 4.5, SPG TD-REG is the only algorithm learning in all trials and whose TD error goes to zero. By contrast, SPG, which never diverged with exact observations (Figure 4.3a), here diverged six times out of 50 (Figure 4.5a). SPG TD-REG plots, instead, are the same in both Figures. REINFORCE, instead, does not significantly suffer from the noisy observations, since it does not learn any critic.

4.5.2 Pendulum Swing-up Tasks

The pendulum swing-up tasks are common benchmarks in RL. Their goal is to swing-up and stabilize a single- and double-link pendulum from any starting position. The agent observes the current joint position and velocity and acts applying torque on each joint. As the pendulum is underactuated, the agent cannot swing it up in a single step, but needs to gather momentum by making oscillatory movements. Compared to the LQR, these tasks are more challenging –especially the double-pendulum– as both the transition and the value functions are nonlinear.

In this section, we apply the proposed TD- and GAE-regularization to TRPO and compare to Retrace [Munos et al., 2016] and to double-critic learning [van Hasselt, 2010], both state-of-the-art techniques to stabilize the learning of the critic. Similarly to GAE, Retrace replaces the advantage function estimator with the average of n-step advantage estimators, but it additionally employs importance sampling to use off-policy data

$$\widehat{A}^\lambda(s_t, a_t; \omega) := (1-\lambda) \sum_{i=t}^{T-1} \lambda^{i-t} \left(\prod_{j=t}^{T-1} w_j \right) \widehat{A}_{t:i+1} + \lambda^{T-t} w_T \widehat{A}_{t:T+1}, \tag{4.47}$$

where the importance sampling ratio $w_j = \min(1, \pi(a_j|s_j;\theta)/\beta(a_j|s_j))$ is truncated at 1 to prevent the "variance explosion" of the product of importance sampling ratios, and $\beta(a|s)$ is the behavior policy used to collect off-policy data. For example, we can reuse past data collected at the i-th iteration by having $\beta(a|s) = \pi(a|s;\theta_i)$. Double-critic learning, instead, employs two critics to reduce the overestimation bias, similarly to TD3. However, TD3 builds upon DPG and modifies the target policy in the Q-function TD error target, which does not appear in the V-function TD error (compare Eq. (4.6) to Eq. (4.26)). Therefore, we decided to use the double-critic method proposed by van Hasselt [2010]. In this case, at each iteration only one critic is randomly updated and used to train the policy. For each critic update, the TD targets are computed using estimates from the other critic to reduce the overestimation bias.

In total, we tested 12 algorithms, i.e., all combinations of vanilla TRPO (NO-REG), TD-regularization (TD-REG), GAE-regularization (GAE-REG), Retrace (RETR) and double-critic learning (DOUBLE). All results are averaged over 50 trials. However, for the sake of clarity, in the plots we show only the mean of the expected return. Both the actor and the critic are linear function with random Fourier features, as presented in [Rajeswaran et al., 2017]. For the single-pendulum we used 100 features, while for the double-pendulum we used 300 features. In both tasks, we tried to collect as few samples as possible, i.e., 500 for the single-pendulum and 3,000 for the double-pendulum. All algorithms additionally reuse the samples collected in the past four iterations, effectively learning with 2,500 and 15,000 samples, respectively, at each iteration. The advantage is estimated with importance sampling as in Eq. (4.47), but only Retrace uses truncated importance ratios. For the single-pendulum, the starting regularization coefficient is $\eta_0 = 1$. For the double-pendulum, $\eta_0 = 1$ for GAE-REG and $\eta_0 = 0.1$ for TD-REG, as the TD error was larger in the latter task (see Figure 4.6d). In both tasks, it then decays according to $\eta_{t+1} = \kappa\eta_t$ with $\kappa = 0.999$. Finally, both the advantage and the TD error estimates are standardized for the policy update. For more details about the tasks and the hyperparameters, we refer to Appendix D.2.

Figure 4.6 shows the expected return and the mean squared TD error estimated by the critic at each iteration. In both tasks, the combination of GAE-REG and Retrace performs the best. From Figure 4.6a, we can see that GAE-REG is the most important component in the single-pendulum task. First, because only yellow plots converge to the optimal policy. TD-REG also helps, but blue plots did not converge after 500 iterations. Second, because the worst performing versions are the ones without any regularization *but* with Retrace. The reason why Retrace, if used alone, harms the learning can be seen in Figure 4.6c. Here, the estimated TD error of NO-REG + RETR and of NO-REG + DOUBLE + RETR is rather small, but its poor performance in Figure 4.6a hints that the critic is affected by overestimation bias. This is not surprising, considering that Retrace addresses the variance and not the bias of the critic.
Results for the double-pendulum are similar but Retrace performs better. GAE-REG + RETR is still the best performing version, but this time Retrace is the most important component,

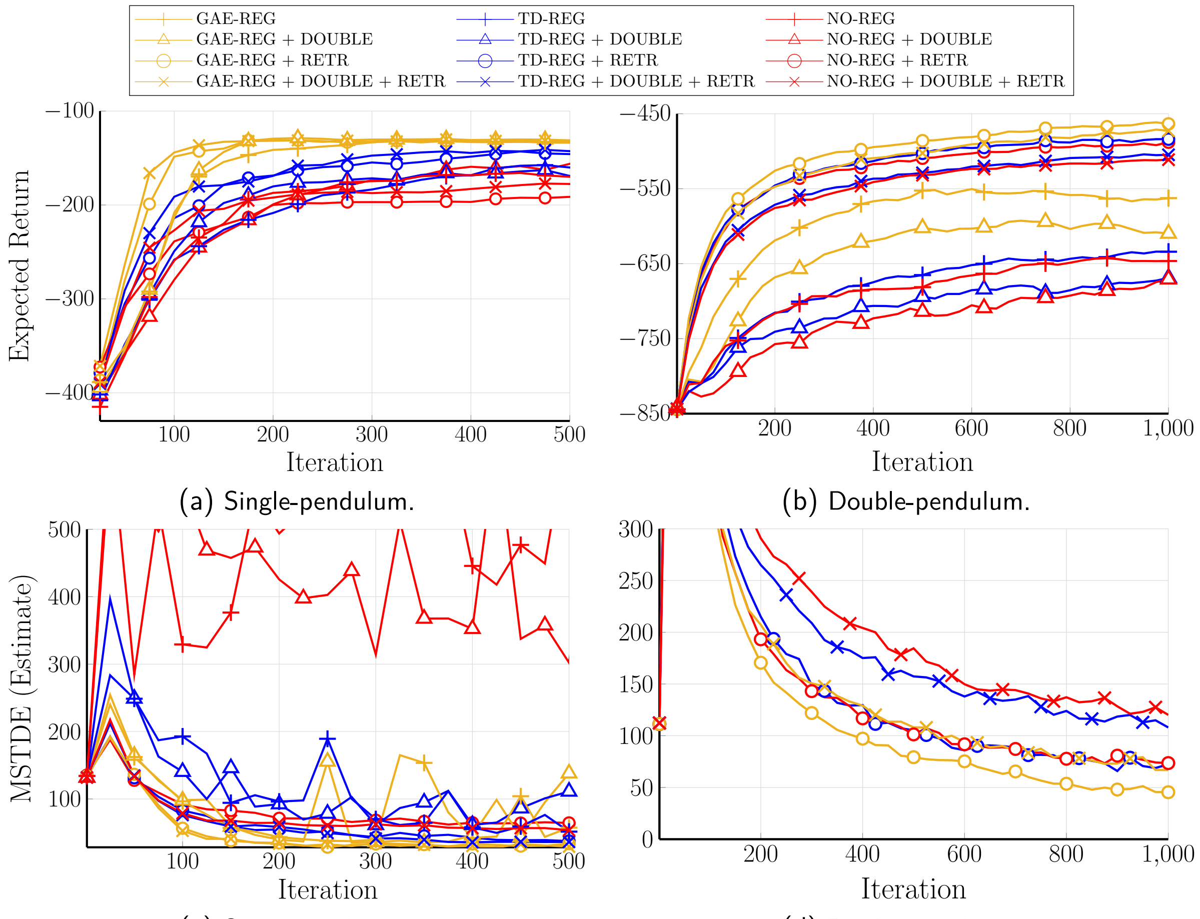


Figure 4.6: TRPO results on the pendulum swing-up tasks. In both tasks, GAE-REG + RETR yields the best results. In the single-pendulum, the biggest help is given by GAE-REG, which is the only version always converging to the optimal policy. In the double-pendulum, Retrace is the most important component, as without it all algorithms performed poorly (their TD error is not shown as too large). In all cases, NO-REG always performed worse than TD-REG and GAE-REG (red plots are always below blue and yellow plots with the same markers).

given that all versions without Retrace performed poorly. We believe that this is due to the larger number of new samples collected per iteration.

From this evaluation, we can conclude that TD- and GAE-regularization are complementary to existing stabilization methods. In particular, the combination of Retrace and GAE-REG yields promising results.

4.5.3 MuJoCo Continuous Control Tasks

We perform continuous control experiments using OpenAI Gym [Brockman et al., 2016] with the MuJoCo physics simulator [Todorov et al., 2012]. For all algorithms, the advantage function is estimated by GAE. For TRPO, we consider a deep RL setting where the actor and the critic are two-layer neural networks with 128 hyperbolic tangent units in each layer. For PPO, the units are 64 in each layer. In both algorithms, both the actor and the critic gradients are optimized by ADAM [Kingma and Ba, 2014]. For the policy update, both the advantage and the TD error estimates are standardized. More details of the hyperparameters are given in Appendix D.3.

From the evaluation on the pendulum tasks, it emerged that the value of the regularization coefficient η strongly depends on the magnitude of the advantage estimator and the TD error. In fact, for TD-REG we had to decrease η_0 from 1 to 0.1 in the double-pendulum task because the TD error was larger. For this reason, for both PPO and TRPO we tested different initial regularization coefficients η_0 and decay factors κ, choosing among all combinations of $\eta_0 = 0.1, 1$ and $\kappa = 0.99, 0.9999$.

Figure 4.7 shows the expected return against training iteration for PPO. On Ant, HalfCheetahm, Walker2d, Humanoid and HumanoidStandup both TD-REG and GAE-REG performed substantially better, especially on Ant-v2, where PPO performed very poorly. On Swimmer and Hopper, TD-REG and GAE-REG also outperformed vanilla PPO, but the improvement was less substantial. On Reacher all algorithms performed the same. This behavior is expected, since Reacher is the easiest of MuJoCo tasks, followed by Swimmer and Hopper. On Ant and Walker, we also notice the "slow start" of TD-regularized algorithms already experienced in the LQR. For the first 1,000 iterations, in fact, vanilla PPO expected return increased faster than PPO TD-REG and PPO GAE-REG, but then it also plateaued earlier.

Results for TRPO (Figure 4.8) are the same except for Humanoid and HumanoidStandup. TD-REG and GAE-REG always outperformed or performed as well as TRPO, and the ranking of the algorithm (first, second and third best performing) is the same of Figure 4.7. On Ant, HalfCheetah and Walker2d, GAE-REG performed better than TD-REG, while on Swimmer GAE-REG surpassed TD-REG. On Hopper, Reacher, Humanoid, and HumanoidStandup, all algorithms performed the same.

It is also interesting to notice that both GAE-REG and TD-REG shared the best performing η_0 and κ. For instance, with PPO on Ant they both performed best with $\eta_0 = 1.0$ and $\kappa = 0.99999$. Only on HalfCheetah (for PPO) and Swimmer (for TRPO) we had to use different η_0 and κ between GAE-REG and TD-REG. On the contrary, the same values do not work for both PPO and TRPO, as for instance with TRPO on Ant the best performing values were $\eta_0 = 0.1$ and $\kappa = 0.99$.

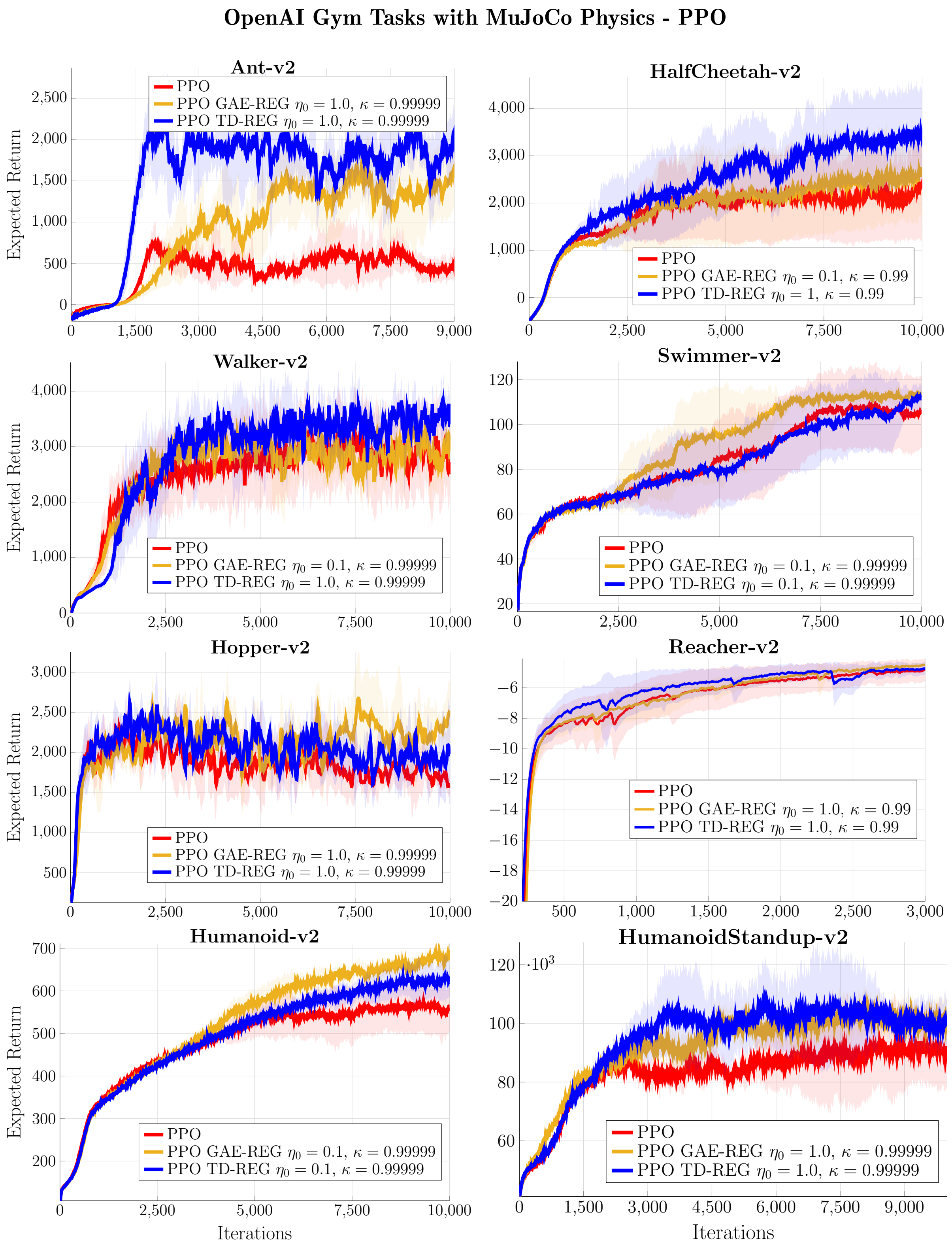


Figure 4.7: Results averaged over five runs, shaded areas denote 95% confidence interval.

OpenAI Gym Tasks with MuJoCo Physics - TRPO

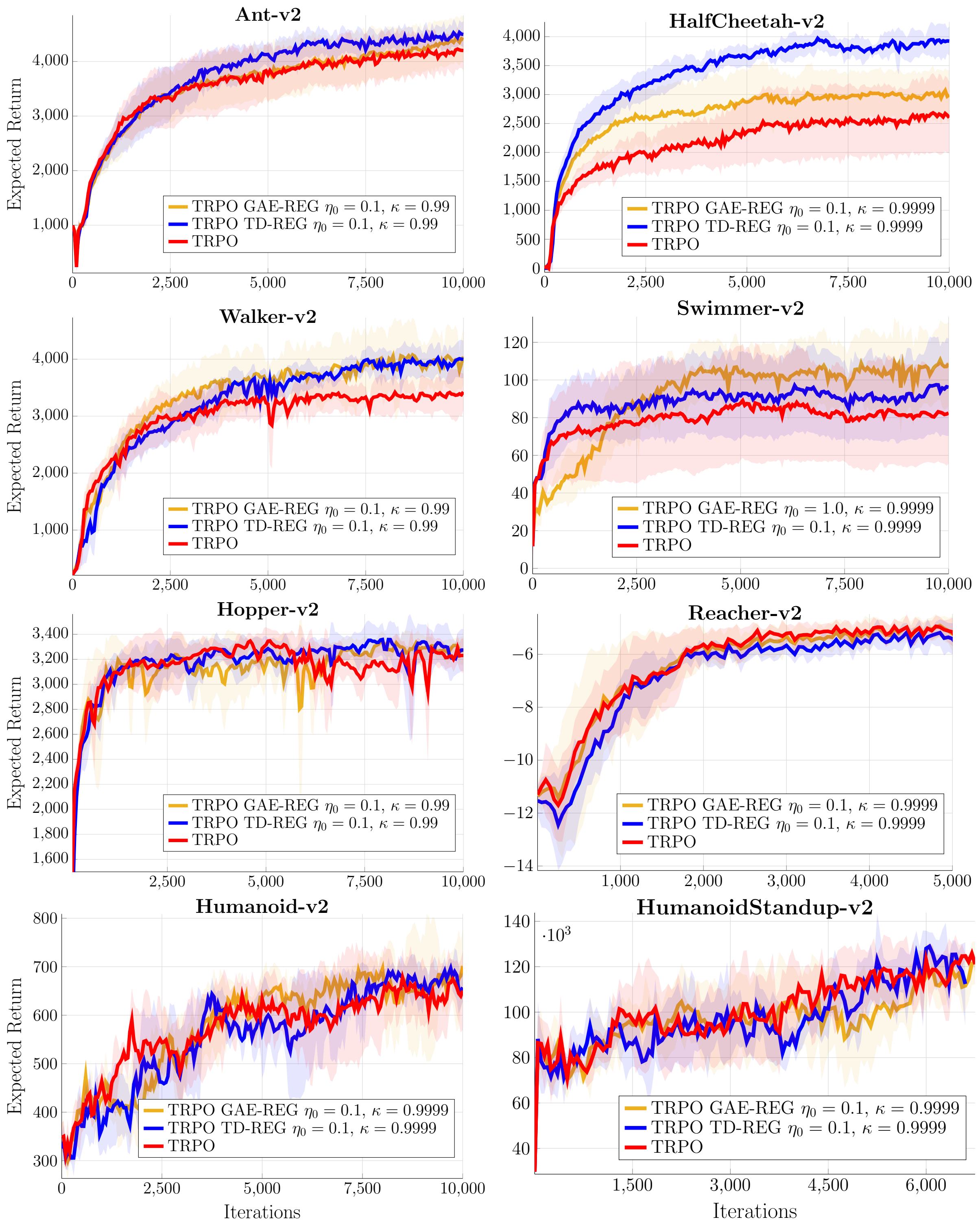


Figure 4.8: Results averaged over five runs, shaded areas denote 95% confidence interval.

From this evaluation it emerged that both TD-REG and GAE-REG can substantially improve the performance of TRPO and PPO. However, as for any other method based on regularization, the tuning of the regularization coefficient is essential for their success.

4.6 Conclusion

Actor-critic methods often suffer from instability. A major cause is the function approximation error in the critic. In this chapter, we addressed the stability issue taking into account the relationship between the critic and the actor. We presented a TD-regularized approach penalizing the actor for breaking the critic Bellman equation, in order to perform policy updates producing small changes in the critic. We presented practical implementations of our approach and combined it together with existing methods stabilizing the critic. Through evaluation on benchmark tasks, we showed that our TD-regularization is complementary to already successful methods, such as Retrace, and allows for more stable updates, resulting in policy updates that are less likely to diverge and improve faster.

Our method opens several avenues of research. In this chapter, we only focused on direct TD methods. In future work, we will consider the Bellman-constrained optimization problem and extend the regularization to residual methods, as they have stronger convergence guarantees even when nonlinear function approximation is used to learn the critic [Baird, 1995]. We will also study equivalent formulations of the constrained problem with stronger guarantees. For instance, the approximation of the integral introduced by the expectation over the Bellman equation constraint could be addressed by using the representation theorem. Furthermore, we will also investigate different techniques to solve the constrained optimization problem. For instance, we could introduce slack variables or use different penalty functions. Another improvement could address techniques for automatically tuning the coefficient η, which is crucial for the success of TD-regularized algorithms, as emerged from the empirical evaluation. Finally, it would be interesting to study the convergence of actor-critic methods with TD-regularization, including cases with tabular and compatible function approximation, where convergence guarantees are available.

5 Conclusion and Future Work

Over the course of the last decade, reinforcement learning has developed into a promising tool for learning a large variety of task. A lot of effort has been directed towards scaling reinforcement learning to solve high-dimensional problems, such as robotic tasks with many degrees of freedom, videogames, and board games. These advances, however, have been possible largely thanks to experts prior knowledge and engineering, such as pre-structured parameterized agent behavior policies and reward shaping. In particular, the design of task-specific exploration strategies and auxiliary rewards is frequently needed, especially if the task to be solved includes different conflicting objectives, or if the reward is sparse, i.e., it is rarely emitted. Furthermore, a prohibitive amount of data is often required, and without it the learning could be slower or more prone to unstable behavior. For instance, with little data it is often necessary to reduce the learning rate or to use larger batches, which may not always be possible due to limited computational resources.

Motivated by the ambition of lifting the autonomy of artificial agents, we devoted our research to make reinforcement learning less dependent on human pre-engineering. To this aim, we considered three open challenges, namely learning (1) with additional conflicting objectives, (2) with sparse rewards, and (3) preventing unstable behaviors. Our contributions in respect of each topic are theoretical, algorithmic, and empirical, and are summarized in Table 5.1.

5.1 Summary of Contributions

In Chapter 2 we considered **learning in the presence of conflicting objectives**, introducing the framework of multi-objective optimization and extending it to reinforcement learning. We discussed that providing the Pareto frontier is favorable over of tuning a scalar reward out of multiple ones. First, because it encapsulates all the trade-offs among the objectives and gives better insight into the problem, thus helping the a posteriori selection of the most desirable solution. Second, because the agent trained by manually tuning a priori the reward function may not meet our expected behavior. In contrast to classical approaches that pre-engineer the task by scalarizing the reward functions, we presented a novel manifold-based method that approximates the Pareto frontier in a single run. We also proposed and evaluated an off-policy extension by including importance sampling in the learning process, in order to further reduce the sample complexity, and formalized two Pareto optimality indicator functions to assess the quality of an approximate frontier. Evaluated on several domains, our algorithms outperformed state-of-the-art competitors both in terms of quality of the learned Pareto frontier and sample efficiency. In particular, the algorithms performed well in the presence of many objectives and high-dimensional parameter spaces. To the best of our knowledge, we were among the first to propose manifold-based approaches for multi-objective reinforcement learning, and to apply it to reinforcement learning many-objective domains.

In Chapter 3 we then addressed the problem of **learning in the presence of sparse rewards and distractor rewards**. Classical exploration techniques in reinforcement learning mostly revolve around the signal received by the immediate reward. However, since improving the agent relies

Table 5.1: Outline of the contributions of the thesis.

Contributions for Learning with Multiple Conflicting Objectives		
Theoretical	**Algorithmic**	**Empirical**
Formalization of desirable properties for approximate frontiers, and of novel indicator fuctions to assess their quality.	Manifold-based algorithm for learning an approximation of the Pareto frontier in a single run and to take advantage of importance sampling.	Evaluation on three benchmark tasks against state-of-the-art, including the simulated robot tetherball and a many-objective domain.

Contributions for Learning with Sparse Reward		
Theoretical	**Algorithmic**	**Empirical**
Novel value functions for long-term exploration, with derivation of optimality bounds.	Off-policy actor-critic algorithm capable of learning with sparse reward and in the presence of distractor rewards.	Evaluation on several benchmark domains, both discrete and continuous, against state-of-the-art. Some domains are novel as well.

Contribution for Actor-Critic Methods Instability		
Theoretical	**Algorithmic**	**Empirical**
TD-regularization for actor-critic methods, showing relationship with existing reinforcement learning methods.	Modification of common actor-critic methods using TD-regularization.	Evaluation on several benchmark continuous tasks against state-of-the-art.

on getting feedback via rewards, the agent cannot be improved until a reward is obtained. In situations where this occurs very rarely, the agent can barely learn. Therefore, many algorithms rely on well-shaped reward functions to guide the agent towards good solutions. From the perspective of autonomous learning this reward engineering is unacceptable. First, it is easy to misspecify the reward function and cause unexpected behavior, e.g., excessively penalizing for collisions may prevent the robot from moving at all. Second, reward engineering requires expertise, and the resulting reward function may not transfer to different tasks or systems. More advanced exploration techniques use bootstrapping or revolve around heuristics to supply an auxiliary "intrinsic" reward even when the environment does not provide one. However, first these methods add the auxiliary reward *after* the action choice which makes exploration sample inefficient for long horizon decision making. Second, they are sensitive to distractor rewards, i.e., low-value easily reachable rewards that correspond to local optima. Contrary to these approaches we proposed long horizon exploration where the action we choose depends on the previous visits to future states, that is, our approach prefers actions which allow the agent to visit states in the future which have not been visited before. We formalized the long-term visitation value function and developed an off-policy actor-critic algorithm for exploration. Our method significantly improves exploration with sparse rewards and distractor rewards, and evaluations on standard and novel benchmarks confirmed this. To the best of our knowledge, we were the first solving efficiently discrete MDPs characterized by both sparse rewards and distractor rewards.

Finally, in Chapter 4 we discussed **actor-critic methods and their instability**. A major cause of such instability is the function approximation error in the critic. Unlike classical approaches which address this by constraining the critic update separately from the actor, have taken into account the relationship between the two. To the best of our knowledge, we were the among the first proposing this kind of approach together with Dai et al. [2018]. We presented a TD-regularized approach penalizing the actor for breaking the critic Bellman equation, in order to perform policy updates producing small changes in the critic. We presented practical implementations of our approach and combined it together with existing methods stabilizing the critic. Through evaluation on benchmark tasks, we showed that our TD-regularization is complementary to already successful methods, such as Retrace, and allows for substantially more stable updates, resulting in policy updates that are less likely to diverge and improve faster.

5.2 Future Work

Our contributions opens several avenues of research. Below, we first present ideas that are directly linked to extending the methods presented in this thesis. Then, we discuss directions of research that have a more general scope.

5.2.1 Extensions to the Proposed Methods

The **multi-objective reinforcement learning** method we proposed has three main characteristics. First, it is based on episodic reinforcement learning. Second, it revolves around Pareto optimality indicator functions. Third, it takes advantage of importance sampling to increase its sample efficiency. Each of these aspects can be extended and improved. The first straightforward extension regards contextual episodic reinforcement learning, used to model many real-world tasks, especially in robotics. Furthermore, multi-objective optimization and multi-task learning are relatively young field of research. In particular, there are few applications of multi-objective methods to multi-task learning. We believe that there is common ground between them, and their combination is a promising direction of research. Second, of the two proposed indicator functions only one is *consistent*, i.e., has Pareto optimality guarantees. However, as it is based on the hypervolume, its computation is demanding. Therefore, our approach – and the multi-objective community in general – would benefit from more computationally efficient yet still Pareto optimal indicator functions. Third, we believe that the sampling efficiency can be further improved. In future work we will adapt the exploration strategy proposed in Chapter 3 to the sampling of the manifold. This would allow to sample manifolds from region of the space that have not been explored yet. Another possibility is to use entropy-regularized mixture models, as proposed by Arenz et al. [2018].

The **exploration strategy** we proposed for sparse reward reinforcement learning is based on the long-term visitation value of state-action pairs. The function approximating this value is trained recursively using its temporal-difference error. Subsequently, at each step our exploration policy computes the visitation value for some candidate actions, and selects the one to execute according to both the Q-function and the visitation value. This strategy proved to be superior to model-free state-of-the-art methods. We believe that model-based reinforcement learning would also benefit from our approach. Model-based methods learn a model of the transition function and, sometimes, of the reward function as well. These models are then used for long-term planning, i.e., the agent selects the action that maximizes the long-term Q-function according to multi-step simulations of the models. One of the shortcomings of these methods is that long-term planning is computationally expensive. In future work, we will combine the proposed visitation value with model-based learning. The idea is to learn a model of the transition function, and use it to predict the next state give action candidates at the current state. Subsequently, the visitation value is applied to the next state rather than on current state-action pairs. This one-step prediction can have a significant impact on the action selection, but does not require computationally demanding multi-step planning as in classical model-based methods.

Finally, to improve the **stability of actor-critic methods** we proposed the TD-regularization loss, based on the relationship between the critic and the actor. This loss penalizes the actor for breaking the critic Bellman equation, in order to perform policy updates producing small changes in the critic. In this thesis, we only focused on direct TD methods. In future work, we will consider the Bellman-constrained optimization problem and extend the regularization to residual methods, as they have stronger convergence guarantees even when nonlinear function approximation is used to learn the critic [Baird, 1995]. We will also study equivalent formulations of the constrained problem with stronger guarantees. For instance, the approximation of the integral introduced by the expectation over the Bellman equation constraint could be addressed by using the representation theorem. Furthermore, we will also investigate different techniques to solve the constrained optimization problem. For instance, we could introduce slack variables or use different penalty functions. Another improvement could address techniques for automatically tuning the regularization coefficient, which is crucial for the success of TD-regularized algorithms, as emerged from the empirical evaluation. Finally, it would be interesting to study the convergence of actor-critic methods with TD-regularization, including cases with tabular and compatible function approximation, where convergence guarantees are available.

5.2.2 General Ideas for Autonomous Reinforcement Learning

The problem of making reinforcement learning fully autonomous is far from solved. Beside the challenges discussed in this thesis, there are many other aspects of rienforcement learning which need to be addressed to achieve truly autonomous agents. Below, we discuss what we think are some of the most relevant and that we will tackle in the future.

Goal-Driven Dimensionality Reduction

Finding state representations on which reinforcement learning can be performed is one of the most tedious process when setting up a learning system. Manually designing features requires expertise and usually these features do not generalize well over different tasks. This problem is prominent in the presence of high-dimensional raw data such as images. For example, consider the robot tetherball game. Truly autonomous agents would learn to play the game based solely on raw observations of the environment, i.e. camera images. However, images provide high-dimensional, redundant and even useless information. Redundant because only tracking the pixels corresponding to the contour of the ball over time is enough to estimate its position and velocity. Useless because everything else on the scene is irrelevant *for the tetherball task*. For instance, the pole and the background should be ignored, as the former is fixed and the latter does not affect the game even if its appearance may change because of light. In the end, as low-dimensional state representations are essential to efficiently solve tasks, we handcrafted features consisting of the ball position, velocity, and pivot point, estimated through a Kalman filter (we refer to Appendix A for more details). Nonetheless, learning from raw input without the need of pre-engineering is mandatory for truly autonomous learning.

To the best of our knowledge, goal-driven dimensionality reduction from raw input is still an open problem in reinforcement learning. On one hand, in fact, applying generic dimensionality reduction, e.g., autoencoders, does not guarantee that the state representation is truly *goal-relevant*, i.e., encoding only information relevant for the task. On the other hand, learning goal-driven representations is challenging because of the circular dependency between the optimal state representation and the optimal policy. To learn goal-relevant representations a large part of the state space needs to be explored using a goal-achieving policy. At the same time, learning a locally or globally optimal policy requires a concise representation.

In the last years, we presented preliminary work to address these limitations. First, in [Parisi et al., 2017b] we combined information-theoretic algorithms with principal component analysis to perform a return-weighted reduction of the state representation. Second, together with Tangkaratt et al. [2017] we proposed a policy search algorithm that learns a quadratic model of the reward function to compute a Gaussian policy analytically. To address dimensionality reduction, we perform nuclear norm regularization on the reward model in a fully supervised manner. Both methods achieved good results on benchmarks, but are limited to linear dimensionality reduction. In the future, we will extend our work to mutual information, a quantity measuring statistical dependency between random variables [Suzuki and Sugiyama, 2013, Niu et al., 2013]. Mutual information is particularly appealing as it can be used to detect non-linear dependencies between two random variables and has already been used for both dimensionality reduction and, very recently, also for reinforcement learning [Kim et al., 2019].

Stochastic Environments and Uncertainty

The majority of reinforcement learning approaches, including the ones presented in this thesis, assume deterministic environments, i.e., where the transition function is deterministic. Most of the benchmarks, for instance, are deterministic simulators. In these environments, learning is easier not only because of the lower variance of gradient estimates, but also because errors in the estimates are due only to errors in the approximators. For instance, in actor-critic methods, errors in the value function estimates would be due only to the bias or the variance in the critic. By contrast, if the environment is stochastic, the error may be due also to the noise in the data. Being able to distinguish when high errors are due to the stochasticity in the environment and when they are due to inner errors in our approximators can make the difference between success and failure. Consider the TD-regularization proposed in this thesis, which penalizes the actor for inaccurate critic estimates. This assumes that high TD error is symptomatic of an inaccurate critic. However, what if the critic error is due to noisy observations or transitions? Then the TD-regularization could lead the actor towards low-noise regions in order to avoid high TD errors, rather than towards regions maximizing the expected return.

In machine learning this problem takes the name of *epistemic vs aleatoric uncertainty*. In reinforcement learning, it is still a relatively young area of research, with researcher addressing it only recently [Sherstan et al., 2018, Pathak et al., 2019]. We believe that the identification of different sources of uncertainty is crucial for the application of reinforcement learning to real-world problem, which are naturally characterized by stochastic dynamics, and we will devote part of our future research to it.

6 List of Publications

Excerpts of the research presented in this thesis have led to the following publications.

6.1 Journal Papers

Simone Parisi, Maximilian Hensel, Jan Peters, and Joni Pajarinen, "*Long-Term Visitation Value for Deep Exploration in Sparse Reward Reinforcement Learning*", arXiv preprint (under review)

Simone Parisi, Voot Tangkaratt, Jan Peters, and Mohammad Emtiyaz Khan, "*TD-Regularized Actor-Critic Methods*", Machine Learning, 108:1467-1501, 2019

Simone Parisi, Matteo Pirotta, and Jan Peters, "*Manifold-based Multi-objective Policy Search with Sample Reuse*", Neurocomputing, Special Issue on Multi-Objective Reinforcement Learning, 263:3-14, 2017

Simone Parisi, Matteo Pirotta, and Marcello Restelli, "*Multi-objective Reinforcement Learning through Continuous Pareto Manifold Approximation*", Journal of Artificial Intelligence Research (JAIR), 57:187-227, 2016

6.2 Conference Papers

Simone Parisi, Simon Ramstedt, and Jan Peters, "*Goal-Driven Dimensionality Reduction for Reinforcement Learning*", Proceedings of the International Conference on Intelligent Robots and Systems (IROS), 2017

Voot Tangkaratt, Herke van Hoof, **Simone Parisi**, Gerhard Neumann, Jan Peters, and Masashi Sugiyama, "*Policy Search with High-Dimensional Context Variables*", Proceedings of the Conference on Artificial Intelligence (AAAI), 2017

Simone Parisi, Alexander Blank, Tobias Viernickel, and Jan Peters, "*Local-utopia policy selection for multi-objective reinforcement learning*", Proceedings of the International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 2016

Simone Parisi, Hany Abdulsamad, Alexandros Paraschos, Christian Daniel, and Jan Peters, "*Reinforcement Learning vs Human Programming in Tetherball Robot Games*", Proceedings of the International Conference on Intelligent Robots and Systems (IROS), 2015

Matteo Pirotta, **Simone Parisi**, and Marcello Restelli, "*Multi-Objective Reinforcement Learning with Continuous Pareto Frontier Approximation*", Proceedings of the Conference on Artificial Intelligence (AAAI), 2015

Simone Parisi, Matteo Pirotta, Nicola Smacchia, Luca Bascetta, and Marcello Restelli, "*Policy gradient approaches for multi-objective sequential decision making*", Proceedings of the International Joint Conference on Neural Networks (IJCNN), 2014

Simone Parisi, Matteo Pirotta, Nicola Smacchia, Luca Bascetta, and Marcello Restelli, "*Policy gradient approaches for multi-objective sequential decision making: A comparison*", Proceedings of the International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 2014

6.3 Workshop Papers

Simone Parisi, Voot Tangkaratt, Jan Peters, and Mohammad Emtiyaz Khan, "*TD-Regularized Actor-Critic Methods*", European Workshop on Reinforcement Learning (EWRL), 2018

Simone Parisi, Voot Tangkaratt, and Jan Peters, "*Regularized Contextual Policy Search via Mutual Information*", Proceedings of the Multi-disciplinary Conference on Reinforcement Learning and Decision Making (RLDM), 2017

7 Curriculum Vitae

7.1 Education

2014 - Present	PhD in Computer Science, Technische Universität Darmstadt, Germany
Advisor	Prof. Jan Peters
Oct - Dec 2017	Research Intern, RIKEN Center for Advanced Intelligence Project, Tokyo, Japan
Advisor	Prof. Masashi Sugiyama, Prof. Emtiyaz Khan
July 2015	Machine Learning Summer School, Max Planck Institute for Intelligent Systems, Tübingen, Germany
2011 - 2014	MSc in Engineering of Computer Systems, Politecnico di Milano, Italy
Thesis Title	Study and Analysis of Policy Gradient Approaches for Multi-objective Decision Problems
Advisor	Prof. Marcello Restelli, Dr. Matteo Pirotta
Final Grade	105 / 110
Aug - Dec 2012	Exchange Student, University of Queensland, Brisbane, Australia
2008 - 2011	BSc in Engineering of Computer Systems. Politecnico di Milano, Italy
Thesis Title	Dino Island: A Turn-based Strategy Videogame
Advisor	Prof. Carlo Ghezzi
Final Grade	99 / 110

7.2 Invited Talks

30 Aug 2019	University of Texas, Learning Agents Research Group, Austin, United States
Host	Peter Stone
28 Aug 2019	Brown University, Dept. of Computer Science, Providence, United States
Host	Michael Littman
26 Aug 2019	Facebook Artificial Intelligence Research, Pittsburgh, United States
Host	Abhinav Gupta
29 May 2019	Max Planck Institute, Dept. of Empirical Inference, Tübingen, Germany
Host	Bernhard Schölkopf
6 May 2019	Delft University of Technology, Dept. of Cognitive Robotics, Netherlands
Host	Jens Kober

3 May 2019	University of Amsterdam, Machine Learning Lab, Amsterdam, Netherlands
Hosts	Herke van Hoof, Max Welling
15 Dec 2017	Advanced Telecommunications Research Institute, Kyoto, Japan
Host	Jun Morimoto
2 Oct 2017	RIKEN Center for Advanced Intelligence Project, Tokyo, Japan
Hosts	Emtiyaz Khan, Masashi Sugiyama

7.3 Teaching Experience

Teaching Assistant

2018 - 2019	Reinforcement Learning, *Technische Universität Darmstadt*
2017	Statistical Machine Learning, *Technische Universität Darmstadt*
2016 - 2017	Robot Learning, *Technische Universität Darmstadt*
2016	Statistical Machine Learning, *Technische Universität Darmstadt*

MSc Thesis Supervision

2019	Kai Cui. *A Study on TD-regularized Actor-critic Methods*
2019	Stefan Hübecker. *Curiosity-driven Reinforcement Learning for Autonomous Driving*
2019	Shuo Zhang. *Integration of Self-imitation and Model-based Learning to Actor-critic Algorithms*

BSc Thesis Supervision

2016	Simon Ramstedt. *Deep Reinforcement Learning with Continuous Actions*
2016	Leon Keller. *Integration of Self-imitation and Model-based Learning to Actor-critic Algorithms*

Project Supervision

2018	Shuo Zhang, Lu Wan. *Enhancing Exploration through Curiosity for Robotics*
2016 - 2017	Simon Ramstedt. *Bayesian Deep Reinforcement Learning: Tools and Methods*
2015 - 2016	Jan-Christoph Klie, Xuelei Li. *Feature Selection for Tetherball Robot Games*
2014 - 2015	Alexander Blank, Tobias Viernickel. *Multi-objective Reinforcement Learning for Tetherball Robot Games*

7.4 Reviewing Experience

Journals

2017	Neurocomputing
2016	Neurocomputing, Special Issue on Multi-objective Reinforcement Learning
2016	Journal of Machine Learning Research (JMLR)
2016	International Journal of Advanced Robotic Systems (IJARS)

Conferences

2019	International Conference on Learning Representations (ICLR)
2018	Conference on Robot Learning (CoRL)
2018	AAAI Conference on Artificial Intelligence (AAAI)
2017	International Conference on Intelligent Robots and Systems (IROS)
2017	International Conference on Robotics and Automation (ICRA)
2017	AAAI Conference on Artificial Intelligence (AAAI)
2016	Robotics: Science and Systems (R:SS)
2016	International Joint Conference on Artificial Intelligence (IJCAI)
2015	International Conference on Intelligent Robots and Systems (IROS)
2015	International Conference on Automation Science and Engineering (CASE)

Workshops

2019	Workshop on Robot Learning, Conference on Neural Information Processing Systems (NIPS)
2018	Workshop on Reinforcement Learning under Partial Observability, Conference on Neural Information Processing Systems (NIPS)
2018	Workshop on Prediction and Generative Modeling in Reinforcement Learning, Conference on Flexible Automation and Intelligent Manufacturing (FAIM)
2018	European Workshop on Reinforcement Learning (EWRL)
2015	European Workshop on Reinforcement Learning (EWRL)

Workshop Proposals

2020	International Conference on Robotics and Automation (ICRA)

Bibliography

H. Abdulsamad, T. Buchholz, T. Croon, and M. El Khoury. Playing tetherball with compliant robots. Technical report, TU Darmstadt, 2014.

J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

S. Agrawal and N. Goyal. Further optimal regret bounds for Thompson sampling. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2013.

S. Ahmadzadeh, P. Kormushev, and D. Caldwell. Multi-objective reinforcement learning for AUV thruster failure recovery. In *Proceedings of the International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 2014.

R. Akrour, A. Abdolmaleki, H. Abdulsamad, and G. Neumann. Model-Free trajectory optimization for reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

S. Amari and S. Douglas. Why natural gradient? In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages 1213–1216 vol.2, 1998.

O. Arenz, M. Zhong, G. Neumann, et al. Efficient gradient-free variational inference using policy search. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

K. Asadi and M. L. Littman. An alternative softmax operator for reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

T. W. Athan and P. Y. Papalambros. A note on weighted criteria methods for compromise solutions in multi-objective optimization. *Engineering Optimization*, 27(2):155–176, 1996.

P. Auer and R. Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 49–56, 2007.

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

L. Baird. Advantage updating. Technical report, Wright-Patterson Air Force Base Ohio: Wright Laboratory, 1993.

L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the International Conference on Machine learning (ICML)*, 1995.

L. Barrett and S. Narayanan. Learning all optimal policies with multiple criteria. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 41–47, 2008.

M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1471–1479, 2016.

B. Belousov and J. Peters. f-Divergence constrained policy improvement, 2017.

N. Beume, B. Naujoks, and M. Emmerich. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669, 2007.

A. M. Bloch. *Nonholonomic mechanics and control*, volume 24. Springer, 2003.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521833787.

R. I. Brafman and M. Tennenholtz. R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 3(Oct):213–231, 2002.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym, 2016.

Y. Burda, H. Edwards, A. Storkey, and O. Klimov. Exploration by random network distillation. In *International Conference on Learning Representations (ICLR)*, 2019.

A. Castelletti, G. Corani, A. Rizzolli, R. Soncini Sessa, and E. Weber. Reinforcement learning in the operational management of a water system. In *IFAC Workshop on Modeling and Control in Environmental Issues*, pages 325–330, 2002.

A. Castelletti, F. Pianosi, and M. Restelli. Tree-based fitted Q-iteration for Multi-Objective markov decision problems. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2012.

A. Castelletti, F. Pianosi, and M. Restelli. A multiobjective reinforcement learning approach to water resources systems operation: Pareto frontier approximation in a single run. *Water Resources Research*, 49(6):3476–3486, 2013.

D. D. Castro, D. Volkinshtein, and R. Meir. Temporal difference based actor critic learning - convergence and neural implementation. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

O. Chapelle and L. Li. An empirical evaluation of Thompson sampling. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2249–2257, 2011.

C. A. C. Coello, G. B. Lamont, and D. A. V. Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Number 2 in Genetic and Evolutionary Computation. Springer-Verlag New York, Inc., New York, USA, September 2007.

R. H. Crites and A. G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33(2-3):235–262, 1998.

B. Dai, A. Shaw, N. He, L. Li, and L. Song. Boosting the actor with dual critic. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

C. Daniel, G. Neumann, and J. Peters. Learning concurrent motor skills in versatile solution spaces. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 3591–3597, 2012.

C. Dann and E. Brunskill. Sample complexity of episodic fixed-horizon reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2818–2826, 2015.

I. Das and J. E. Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. *Structural optimization*, 14 (1):63–69, 1997.

K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proceedings of the International Conference on Parallel Problem Solving From Nature*, 2000.

T. Degris, M. White, and R. S. Sutton. Linear Off-Policy Actor-Critic. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2012.

M. P. Deisenroth, G. Neumann, and J. Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013.

C. D'Eramo, A. Nuara, M. Pirotta, and M. Restelli. Estimating the maximum expected value in continuous reinforcement learning problems. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

C. D'Eramo, A. Cini, and M. Restelli. Exploiting Action-Value uncertainty to drive exploration in reinforcement learning. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2019.

K. Dong, Y. Wang, X. Chen, and L. Wang. Q-learning with UCB exploration is sample efficient for Infinite-Horizon MDP, 2019.

B. Efron. *The Jackknife, the bootstrap and other resampling plans.* SIAM, 1982.

E. A. Feinberg. Constrained discounted markov decision processes and hamiltonian cycles. *Mathematics of Operations Research*, 25(1):130–140, 2000.

M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, et al. Noisy networks for exploration. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in Actor-Critic methods. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

Z. Gabor, Z. Kalmar, and C. Szepesvari. Multi-criteria reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1998.

Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the International Conference on Machine learning (ICML)*, 2016.

E. Greensmith, P. L. Bartlett, and J. Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 5(Nov): 1471–1530, 2004.

A. Gruslys, M. G. Azar, M. G. Bellemare, and R. Munos. The reactor: A fast and sample-efficient Actor-Critic agent for reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

S. Gu, S. Levine, I. Sutskever, and A. Mnih. Muprop: Unbiased backpropagation for stochastic neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016a.

S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep Q-Learning with Model-based acceleration. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016b.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the International Conference on Machine learning (ICML)*, 2018.

K. Harada, J. Sakuma, and S. Kobayashi. Local search for multiobjective function optimization: pareto descent method. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 659–666, 2006.

P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 2017.

M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 2018.

T. Hester and P. Stone. TEXPLORE: Real-time sample-efficient reinforcement learning for robots. *Machine Learning*, 90(3):385–429, 2013.

R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. VIME: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1109–1117, 2016.

C. Igel, N. Hansen, and S. Roth. Covariance matrix adaptation for Multi-objective optimization. *Evolutionary Computation*, 15(1):1–28, 2007.

A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In *Advances in Neural Information Processing Systems (NIPS)*, 2002.

T. Jaksch, R. Ortner, and P. Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 11(Apr):1563–1600, 2010.

C. Jin, Z. Allen-Zhu, S. Bubeck, and M. I. Jordan. Is Q-learning provably efficient? In *Advances in Neural Information Processing Systems (NIPS)*, pages 4863–4873, 2018.

S. Kakade. *On the sample complexity of reinforcement learning.* PhD thesis, University College London, 2003.

M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal. Learning force control policies for compliant robotic manipulation. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2012.

E. Kaufmann, N. Korda, and R. Munos. Thompson sampling: An asymptotically optimal finite-time analysis. In *Proceedings of the International Conference on Algorithmic Learning Theory (ALT)*, 2012.

M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002.

H. Kim, J. Kim, Y. Jeong, S. Levine, and H. O. Song. EMI: Exploration with mutual information. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

J. Kober and J. Peters. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pages 579–610. Springer, 2012.

J. Z. Kolter and A. Y. Ng. Near-Bayesian exploration in polynomial time. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 513–520. ACM, 2009.

J. Z. Kolter, A. Coates, A. Y. Ng, Y. Gu, and C. DuHadway. Space-indexed dynamic programming: learning to follow trajectories. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 488–495, 2008.

V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, 2000.

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

T. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Adv. Appl. Math.*, 6 (1):4–22, Mar 1985.

T. Lens and O. von Stryk. Design and dynamics model of a lightweight series elastic tendon-driven robot arm. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2013.

S. Levine and V. Koltun. Guided policy search. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1–9, 2013.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

D. J. Lizotte, M. H. Bowling, and S. A. Murphy. Efficient reinforcement learning with multiple reward functions for randomized controlled trial analysis. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 695–702, 2010.

D. J. Lizotte, M. Bowling, and S. A. Murphy. Linear Fitted-Q iteration with multiple reward functions. *Journal of Machine Learning Research (JMLR)*, 13:3253–3295, 2012.

S. Mannor and N. Shimkin. The steering approach for Multi-Criteria reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1563–1570, 2001.

S. Mannor and N. Shimkin. A geometric approach to Multi-Criterion reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 5:325–360, 2004.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. In *Proceedings of the NIPS Workshop on Deep Learning*, 2013.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

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

R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare. Safe and efficient Off-Policy reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans. Trust-PCL: An Off-Policy trust region method for continuous control. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

S. Natarajan and P. Tadepalli. Dynamic preferences in multi-criteria reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2005.

G. Niu, W. Jitkrittum, B. Dai, H. Hachiya, and M. Sugiyama. Squared-loss mutual information regularization: A novel Information-theoretic approach to Semi-supervised learning. In *Proceedings of the International Conference on Machine learning (ICML)*, 2013.

J. Nocedal and S. Wright. *Numerical optimization*. Springer series in operations research and financial engineering. Springer, New York, NY, 2. ed. edition, 2006.

Y. Nojima, F. Kojima, and N. Kubota. Local episode-based learning of multi-objective behavior coordination for a mobile robot in dynamic environments. In *Proceedings of the International Conference on Fuzzy Systems*, volume 1, pages 307–312. IEEE, 2003.

I. Osband. Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *Proceedings of the NIPS Workshop on Bayesian Deep Learning*, 2016.

I. Osband, D. Russo, and B. Van Roy. (More) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped DQN. In *Advances In Neural Information Processing Systems (NIPS)*, 2016a.

I. Osband, B. Van Roy, and Z. Wen. Generalization and exploration via randomized value functions. In *Proceedings of the International Conference on Machine learning (ICML)*, 2016b.

I. Osband, J. Aslanides, and A. Cassirer. Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 8617–8629, 2018.

I. Osband, B. V. Roy, D. J. Russo, and Z. Wen. Deep exploration via randomized value functions. *Journal of Machine Learning Research (JMLR)*, 20(124):1–62, 2019.

A. Owen and Y. Zhou. Safe and effective importance sampling. *Journal of the American Statistical Association*, 95(449):135–143, 2000.

S. Parisi, M. Pirotta, N. Smacchia, L. Bascetta, and M. Restelli. Policy gradient approaches for multi-objective sequential decision making. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2014.

S. Parisi, H. Abdulsamad, A. Paraschos, C. Daniel, and J. Peters. Reinforcement learning vs human programming in tetherball robot games. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2015.

S. Parisi, M. Pirotta, and J. Peters. Manifold-based Multi-objective policy search with sample reuse. *Neurocomputing*, 263:3–14, 2017a.

S. Parisi, S. Ramstedt, and J. Peters. Goal-Driven dimensionality reduction for reinforcement learning. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2017b.

D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

D. Pathak, D. Gandhi, and A. Gupta. Self-Supervised exploration via disagreement. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

J. Peters. *Machine Learning of Motor Skills for Robotics.* PhD thesis, University of Southern California, 2007.

J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7):1180–1190, 2008a.

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

J. Peters, K. Muelling, and Y. Altun. Relative entropy policy search. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 2010.

M. Pirotta, S. Parisi, and M. Restelli. Multi-Objective reinforcement learning with continuous pareto frontier approximation. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 2015.

M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz. Parameter space noise for exploration. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete Bayesian reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 697–704, 2006.

D. Precup, R. S. Sutton, and S. P. Singh. Eligibility traces for Off-Policy policy evaluation. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2000.

D. V. Prokhorov and D. C. Wunsch. Adaptive critic designs. *Transactions on Neural Networks*, 8(5):997–1007, 1997.

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994. ISBN 0471619779.

A. Rajeswaran, K. Lowrey, E. V. Todorov, and S. M. Kakade. Towards generalization and simplicity in continuous control. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

H. Robbins and S. Monro. A stochastic approximation method. In *Herbert Robbins Selected Papers*, pages 102–109. Springer, 1985.

D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. A survey of Multi-Objective sequential Decision-Making. *Journal of Artificial Intelligence Research (JAIR)*, 48:67–113, 2013.

T. Rückstiess, F. Sehnke, T. Schaul, D. Wierstra, Y. Sun, and J. Schmidhuber. Exploring parameter space in reinforcement learning. *Paladyn*, 1(1):14–24, 2010.

D. Russo and B. Van Roy. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 39(4):1221–1243, 2014.

D. Russo, D. Tse, and B. Van Roy. Time-sensitive bandit learning and satisficing Thompson sampling, 2017.

D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, Z. Wen, et al. A tutorial on Thompson sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96, 2018.

R. M. Ryan and E. L. Deci. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary Educational Psychology*, 25(1):54–67, 2000.

J. Schmidhuber. A possibility for lmplementing curiosity and boredom in Model-Building neural controllers. In *Proceedings of the International Conference on Simulation of Adaptive Behavior (SAB)*, pages 222–227, 1991.

J. Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18(2):173–187, 2006.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

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

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

S. L. Scott. A modern Bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26(6):639–658, 2010.

C. R. Shelton. *Importance Sampling for Reinforcement Learning with Multiple Objectives.* PhD thesis, Massachusetts Institute of Technology, August 2001.

C. Sherstan, D. R. Ashley, B. Bennett, K. Young, A. White, M. White, and R. S. Sutton. Comparing direct and indirect temporal-difference methods for estimating the variance of the return. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2014.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman1, D. Grewe1, J. Nham, N. Kalchbrenner1, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel1, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587): 484–489, 2017a.

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

D. Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches.* John Wiley & Sons, 2006.

B. C. Stadie, S. Levine, and P. Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. In *Proceedings of the NIPS Workshop on Deep Reinforcement Learning*, 2015.

A. L. Strehl and M. L. Littman. An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences (JCSS)*, 74(8):1309–1331, 2008.

A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman. PAC model-free reinforcement learning. In *Proceedings of the International Conference on Machine learning (ICML)*, pages 881–888, 2006.

M. Strens. A Bayesian framework for reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 943–950, 2000.

Y. Sun, D. Wierstra, T. Schaul, and J. Schmidhuber. Efficient natural evolution strategies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2009.

R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning*, pages 216–224. Elsevier, 1990.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, March 1998.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, 2018.

R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, 1999.

T. Suzuki and M. Sugiyama. Sufficient dimension reduction via squared-loss mutual information estimation. *Neural Computation*, 25(3):725–758, 2013.

C. Szepesvari. *Algorithms for Reinforcement Learning*, volume 4. Morgan & Claypool Publishers, 01 2010.

A. Tamar, D. Di Castro, and S. Mannor. Policy gradients with variance related risk criteria. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2012.

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

Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 4906–4913, 2012.

D. Tateo, C. D'Eramo, A. Nuara, M. Restelli, and A. Bonarini. Exploiting structure and uncertainty of Bellman updates in Markov decision processes. In *Proceedings of the Symposium Series on Computational Intelligence (SSCI)*, 2017.

G. Tesauro, R. Das, H. Chan, J. O. Kephart, D. Levine, F. L. R. III, and C. Lefurgy. Managing power consumption and performance of computing systems using reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

E. Theodorou, J. Buchli, and S. Schaal. Learning policy improvements with path integrals. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 828–835, 2010.

W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2012.

P. Vamplew, R. Dazeley, E. Barker, and A. Kelarev. Constructing stochastic mixture policies for episodic multiobjective reinforcement learning tasks. In *Proceedings of the Australasian Conference on Artificial Intelligence (AI)*, pages 340–349, 2009.

P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, and E. Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, 84(1-2):51–80, 2011.

H. van Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.

K. Van Moffaert, M. M. Drugan, and A. Nowé. Scalarized multi-objective reinforcement learning: Novel design techniques. In *Proceedings of the Symposium Series on Computational Intelligence (SSCI)*, volume 13, pages 94–103, 2012.

O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, , J. Chung, , D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen,

V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. a. Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, pages 1–5, 2019.

W. Wang and M. Sebag. Hypervolume indicator and dominance reward based multi-objective Monte-Carlo tree search. *Machine Learning*, 92(2-3):403–429, 2013.

C. J. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

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

R. J. Williams. Simple statistical Gradient-Following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, May 1992.

C. Wu, A. Rajeswaran, Y. Duan, V. Kumar, A. M. Bayen, S. Kakade, I. Mordatch, and P. Abbeel. Variance reduction for policy gradient with action-dependent factorized baselines. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

T. Yoshikawa. *Foundations of robotics: analysis and control*. Mit Press, 1990.

T. Zhao, H. Hachiya, G. Niu, and M. Sugiyama. Analysis and improvement of policy gradient estimation. *Neural Networks*, 26(0):118 – 129, 2012.

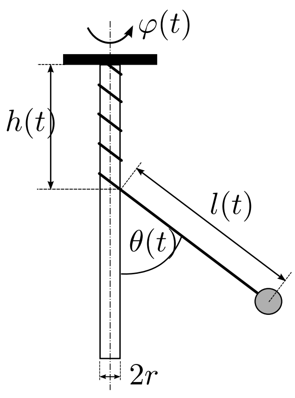
T. Zhao, H. Hachiya, V. Tangkaratt, J. Morimoto, and M. Sugiyama. Efficient sample reuse in policy gradients with parameter-based exploration. *Neural Computation*, 25(6):1512–1547, Jun 2013.

E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, and V. da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *Evolutionary Computation*, 7(2):117–132, 2003.

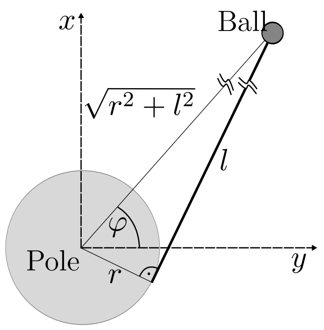
E. Zitzler, D. Brockhoff, and L. Thiele. The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration. In S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, and T. Murata, editors, *Evolutionary Multi-Criterion Optimization*, volume 4403 of *Lecture Notes in Computer Science*, pages 862–876. Springer Berlin Heidelberg, 2007.

A Details of the Robot Tetherball Task

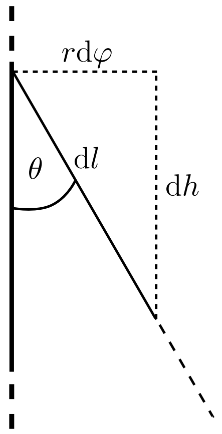
Here, we provide the details about the robot tetherball platform, the learned and the hand-crafted players. We also provide details of the experimental results, highlight the benfits of using reinforcement learning instead of human programming, and discuss the hyperparameters tuning of the learned player.
Regarding the hardware, we use two BioRob robots with six degrees of freedom each [Lens and von Stryk, 2013]. The BioRobs are cable driven lightweight robots, that achieve highly dynamic behavior due to the integration of springs between the motors and the cables which drive the joints. This highly dynamic behavior, however, makes it hard to provide a inverse dynamics model for the robot.



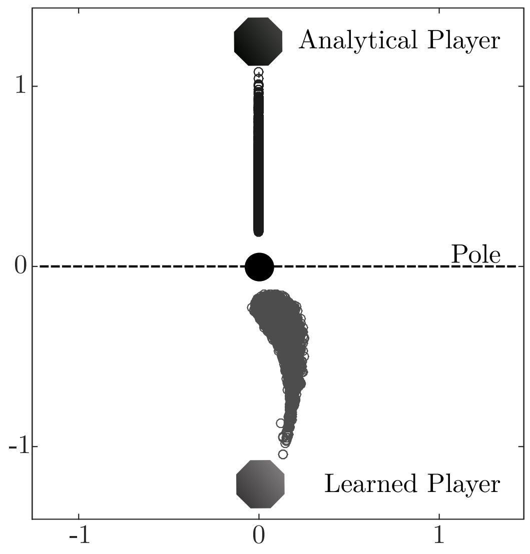
(a) Tetherball Model.



(b) Planar view.



(c) Differential view.



(d) Distribution of hitting points.

Figure A.1: Schematics of the mechanical tetherball model. a) State representation in spherical space with four degrees of freedom. b) Simplification of the $\{x, y\}$ coordinates by ignoring the pole radius r. c) The infinitesimal change of the states for formulating the nonholonomic constraints. d) The distribution of the hitting points for both players. The hand-crafted player hits the ball in the fixed plane defined by the pole and origin of the robot. The learned player does not have any constraint and can freely choose the interception point.

Formally, we denote the state of the environment by $\boldsymbol{x} = \{\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{b}, \dot{\boldsymbol{b}}, h\}$, where $\boldsymbol{q}, \dot{\boldsymbol{q}} \in \mathbb{R}^6$ are the joint positions and velocities, $\boldsymbol{b}, \dot{\boldsymbol{b}} \in \mathbb{R}^3$ are the ball position and velocity in Cartesian coordinates, h is the height of the pivot point along the pole. The control actions $\boldsymbol{u} \in \mathbb{R}^6$ are torques generated by a low-level controller $\boldsymbol{u} = f_{\text{PD}}(\boldsymbol{x}, \boldsymbol{x}_{\text{des}})$. The low-level controller chooses actions $\boldsymbol{u}$ such that the robot follows a desired trajectory $\boldsymbol{\tau}_{\text{des}} = \{\boldsymbol{x}_{\text{des},i}\}_{i=1:H}$, where H is the length of the desired trajectory. While the joint velocities $\dot{\boldsymbol{q}}$ can be controlled directly, the ball positions $\boldsymbol{b}$ and the pivot point h are only controlled indirectly. Given this setup, we formulate both an hand-crafted player and a learned player capable to play robot tetherball against each other.

A.1 The Hand-Crafted Player

In this section, we dissect the tetherball task and discuss how the problem can be described mathematically in order to derive an hand-crafted player. This hand-crafted player uses a mechanical model of tetherball to predict the ball trajectory and the resulting interception point. Subsequently, a stroke movement represented by a spline function is generated and followed by the low-level controller $f_{\text{PD}}(\boldsymbol{x}, \boldsymbol{x}_{\text{des}})$.

A.1.1 Mathematical Trajectory Prediction

Mathematically describing tetherball requires a complex mechanical model [Abdulsamad et al., 2014]. Even after disregarding effects like friction, air drag and changes in string tension, the underlying equations are still highly nonlinear. As shown in Fig. A.1a, we can describe the state of the model with four degrees of freedom when represented in spherical space. Here, $\{\theta, \varphi, l\}$ are the spherical coordinates while h is the vertical translation of the pivot point along the pole. These four coordinates fully determine the Cartesian position of the ball according to the equations

$$x \simeq l \sin\theta \cos\varphi, \quad y \simeq l \sin\theta \sin\varphi, \quad z = l\cos\theta + h. \tag{A.1}$$

As shown in Fig. A.1b, for $\{x, y\}$ coordinates we ignore the pole radius r when compared to the length of the string l. In addition to these algebraic equations, the system is constrained by two nonholonomic, i.e., not integrable, velocity conditions for $\{\dot{l}, \dot{h}\}$, as shown in Fig. A.1c. These constraints reflect the behavior of the string while it (un-)winds around the pole and are given by

$$\dot{l} = \frac{-r\dot{\varphi}}{\sin\theta}, \quad \dot{h} = \frac{r\dot{\varphi}}{\tan\theta}. \tag{A.2}$$

For the purpose of estimating the ball trajectory we need the full equations of motion, which we derive using the principle of Lagrange-D'Alembert [Bloch, 2003] for systems with nonholonomic velocity constraints

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\rho}_i} - \frac{\partial L}{\partial \rho_i} = \sum_{j=1}^{n} \lambda_j a_i^j, \tag{A.3}$$

where ρ_i are the degrees of freedom and $L = T - U$ is the Lagrangian consisting of the kinetic energy $T = mv^2/2$ and potential energy $U = -mgz$. The right side of the equation incorporates the velocity constraints a_i, as shown in Eq. (A.2), and their respective Lagrangian multipliers λ_i

$$\boldsymbol{a}^\mathsf{T}\dot{\boldsymbol{q}} = \begin{bmatrix}0 & r\left[\frac{1}{\tan\theta} - \frac{1}{\sin\theta}\right] & 1 & 1\end{bmatrix}\left[\dot{\theta}, \dot{\varphi}, \dot{l}, \dot{h}\right]^\mathsf{T}. \tag{A.4}$$

By expanding the Lagrange-D'Alembert equation we obtain

$$\begin{aligned}
\frac{d}{dt}\frac{\partial L}{\partial\dot{\theta}} - \frac{\partial L}{\partial\theta} &= 0,\\
\frac{d}{dt}\frac{\partial L}{\partial\dot{\varphi}} - \frac{\partial L}{\partial\varphi} &= \lambda_1\frac{r}{\sin\theta} - \lambda_2\frac{-r}{\tan\theta},\\
\frac{d}{dt}\frac{\partial L}{\partial\dot{l}} - \frac{\partial L}{\partial l} &= \lambda_1,\\
\frac{d}{dt}\frac{\partial L}{\partial\dot{h}} - \frac{\partial L}{\partial h} &= \lambda_2.
\end{aligned} \tag{A.5}$$

The resulting differential equations and constraints are solved for the accelerations $\{\ddot{\theta}, \ddot{\phi}, \ddot{l}, \ddot{h}\}$ and the two Lagrangian multipliers $\{\lambda_1, \lambda_2\}$. Numerically integrating the initial state $\{\theta, \dot{\theta}, \phi, \dot{\phi}, l, h\}$ according to these equations allows us to predict the motion of the ball.
The hand-crafted player uses this model to estimate and continuously update the ball trajectory until interception. This trajectory is transformed into the Cartesian frame of the player and used to choose a hitting point that lies in the plane defined by the pole and origin of the robot (see Fig. A.1d). Once such a point is found on a trajectory with a rotation direction corresponding to the hitting direction of the player, the time and position information are passed on to the lower controller to plan and execute the appropriate joint movement.

A.1.2 Generating Hitting Trajectories

Given the desired interception point $\boldsymbol{x}_{\text{des}}$, the low-level controller starts by solving a constrained inverse kinematics problem

$$\boldsymbol{q}_{\text{des}} = f_{\text{kin}}^{-1}(\boldsymbol{x}_{\text{des}})\text{s.t. } \boldsymbol{q}_{\text{min}} < \boldsymbol{q}_{\text{des}} < \boldsymbol{q}_{\text{max}}. \tag{A.6}$$

Thereupon, it generates a minimum jerk trajectory for a smooth transition to the hitting posture while satisfying the end time and speed constraints. A minimum jerk trajectory is represented by a fifth order polynomial $\boldsymbol{q}_t = \sum_{i=0}^{5} \boldsymbol{c}_i t^i$ whose parameters $\boldsymbol{c}_i$ reflect the start and end conditions of a trajectory. In the end, the joint trajectory is transformed into a series of motor torques $\boldsymbol{u}_i$ by an underlying PD-controller with gravity compensation.

A.2 Learning to Play Tetherball

After introducing the hand-crafted player, based on a mechanical model of tetherball, we show how to formulate the problem of tetherball as a contextual episodic Markov Decision Process (MDP) [Deisenroth et al., 2013] such that we can employ reinforcement learning (RL) algorithms to learn how to play it. We formulate the problem with contexts $\boldsymbol{s} \in \mathcal{S}$, actions $\boldsymbol{\theta} \in \Theta$ and returns $R_{\boldsymbol{\theta},\boldsymbol{s}} \in \mathcal{R}$. The goal of the robot is to learn a context-dependent policy $\pi(\boldsymbol{\theta}|\boldsymbol{s})$ which selects actions according to the context and can, thus, generalize to different scenarios. We choose this distribution to be a Gaussian with linear mean $\pi(\boldsymbol{\theta}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{a} + \boldsymbol{A}\boldsymbol{s}, \boldsymbol{\Sigma})$. We use Dynamic Motor Primitives (DMPs) proposed by Ijspeert et al. [2002] to represent trajectories. DMPs allow us to parametrize trajectories such that one vector of actions $\boldsymbol{\theta}$ defines a desired trajectory to be followed by the robot. In this setting, the context $\boldsymbol{s}$ defines the initial Cartesian position and velocity of the ball as well as the pivot point of the rope at the start of the episode, i.e., $\boldsymbol{s} = \{\boldsymbol{b}_0, \dot{\boldsymbol{b}}_0, h_0\}$. We define each player turn as an episode. A turn starts when a player hits or misses the ball.

Alternatively, the tetherball task could be formulated as infinite horizon problem where a policy $\pi(\boldsymbol{u}|\boldsymbol{x})$ directly selects control actions at every time step. However, formulating real robot learning problems based on time indexed states and actions is difficult for several important reasons. Firstly, a high control and sensory sampling frequency can lead to effects that render the problem Markovian only in a higher order. This means that effects of control actions will not be registered immediately by the sensors, and a history over several time steps has to be kept to make the learning problem feasible. Reducing the frequency too much can lead to jerky behaviors or controllers that do not resolve the desired behavior finely enough to solve the problem at hand. Even worse, it is possible that there is no 'sweet spot' between the two extremes at all. Non-Markovian behavior in the proposed platform is additionally introduced due to the fact that the state of the springs and the extension of the cables cannot be directly observed. Furthermore, solving the optimization problem with a time-dependent policy requires exploration noise for all states. Requiring such noise on all states should be avoided for two reasons. First, it generates non-smooth trajectories which may be not only undesirable but usually harmful to the robot. Second, many step-based RL methods require the ability to sample from the state space *during* the trajectory, i.e., to be able to set the robot and the environment to an arbitrary state, which obviously is physically impossible.

However, while trajectory based learning methods can be easy to use, increase performance and reduce the learning time in many scenarios, they are not suitable for arbitrary problems. If, for example, feedback during the trajectory is crucial, step-based RL methods can be more appropriate.

A.2.1 Compact Skill Representation

As stated above, DMPs offer a compact representation of basic movements. Formally, DMPs are defined as a second order dynamic system that acts like a spring-damper system which is driven by a non-linear forcing function $\boldsymbol{f}(z_t, \boldsymbol{\theta})$

$$\begin{aligned} \ddot{\boldsymbol{q}}_t &= \tau^2 \left(\alpha \left(\beta \left(\boldsymbol{g} - \boldsymbol{q} \right) - \frac{\dot{\boldsymbol{q}}}{\tau} \right) + \boldsymbol{f}(z_t, \boldsymbol{\theta}) \right), \\ \dot{z}_t &= -\tau \alpha_z z_t, \end{aligned} \tag{A.7}$$

where $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$ are the joint positions, velocities and accelerations respectively. The unique goal attractor position and velocity are defined by $\boldsymbol{g}, \dot{\boldsymbol{g}}$. Variable z_t denotes the phase and τ is a temporal scaling factor. Finally, α, β, α_z are fixed parameters. Commonly, a separate DMP is used for each joint of the robot and the phase z_t is shared between all joints in order to synchronize them. The forcing function is linear in its weights $\boldsymbol{\theta}$ but non-linear in the phase z_t, i.e., $\boldsymbol{f}(z_t, \boldsymbol{\theta}) = \boldsymbol{\psi}(z_t)^\mathsf{T}\boldsymbol{\theta}$. Usually, for stroke-based movements, normalized Gaussian basis functions are used. A crucial aspect that affects the accuracy of the resulting trajectory is the number of bases used, as they influence learning speed and expressiveness of the final policy. We evaluate the effects of this choice in the experimental section.

The resulting desired trajectory represented by DMPs is then followed by the same PD-controller with gravity compensation that is used by the hand-crafted player. However, even if the parameters of the PD-controller are not precisely tuned to allow perfect tracking of the input trajectory, the RL agent can learn to produce input trajectories that compensate for the tracking error of the PD-controller, while the hand-crafted player depends on accurate tracking of the desired trajectory $\boldsymbol{\tau}_{\text{des}}$.

A.2.2 Imitation Learning of DMPs

When using DMPs to generate desired trajectories, we can easily bootstrap the learning process by demonstrating an initial trajectory, i.e., by imitation learning. In many cases, bootstrapping by imitation learning is essential to make the learning process feasible, as it not only provides a good initial policy but also helps avoiding sampling trajectories that are dangerous to the robot and its environment.

More formally, given a desired joint trajectory $\boldsymbol{\tau}_{\text{des}} = [\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i, \ddot{\boldsymbol{q}}_i]_{i=1:T}$, a unique goal attractor $\boldsymbol{g}, \dot{\boldsymbol{g}}$, fixed parameters α, β, α_z and a temporal scaling factor τ, we can compute the forcing function $\boldsymbol{f}(z_t, \boldsymbol{\theta})$ for each time step and obtain the parameters $\boldsymbol{\theta}$ by linear regression

$$\begin{aligned}\boldsymbol{f}(z_t, \boldsymbol{\theta}) &= \frac{\ddot{\boldsymbol{q}}_t}{\tau^2} - \alpha\left(\beta(\boldsymbol{g} - \boldsymbol{q}_t) + \frac{\dot{\boldsymbol{g}} - \dot{\boldsymbol{q}}_t}{\tau}\right) \\ \boldsymbol{\theta} &= \left(\boldsymbol{\Psi}^{\mathsf{T}}\boldsymbol{\Psi} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{\Psi}^{\mathsf{T}} \boldsymbol{f}(z_t, \boldsymbol{\theta})\end{aligned} \tag{A.8}$$

where $\boldsymbol{\Psi} = [\boldsymbol{\psi}(z_1), \ldots, \boldsymbol{\psi}(z_T)]$.

In the tetherball games, the movement we want to imitate using DMPs consists of several phases, i.e., the robot starts from its initial resting position, hits the ball and finally goes back to the resting position. While it is already challenging to learn such a movement for fixed initial conditions, we require the robot to generalize to different contexts, i.e., ball positions and velocities to win a match. Thus, the robot needs to adapt shape and execution speed of the DMPs according to new contexts without re-learning the whole task. This generalization is represented by the linear factor in the Gaussian policy. To initialize the policy $\pi(\boldsymbol{\theta}|\boldsymbol{s})$, we use a set of M initial demonstrations paired with different contexts. Given a dataset $\mathcal{D} = \{\boldsymbol{y}_i, \boldsymbol{s}_i\}_{i=1:M}$, we obtain M parameterizations $\boldsymbol{\theta}_i$ using imitation learning and subsequently initialize $\boldsymbol{a}, \boldsymbol{A}, \boldsymbol{\Sigma}$ with linear regression

$$\begin{aligned}\boldsymbol{\Theta} &= [\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_N]^{\mathsf{T}}, \\ \boldsymbol{\Phi} &= \left[[\boldsymbol{s}_1, \cdots, \boldsymbol{s}_N]^{\mathsf{T}}, \mathbf{1}\right], \\ \begin{bmatrix}\boldsymbol{A} \\ \boldsymbol{a}\end{bmatrix} &= (\boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}\boldsymbol{\Theta}, \\ \boldsymbol{\Sigma} &= \operatorname{Cov}(\boldsymbol{\Theta}).\end{aligned} \tag{A.9}$$

When using DMPs we can choose whether to only provide a single demonstration to determine the mean of the initial policy or to provide multiple demonstrations to also determine the covariance of the initial policy. We evaluate the effects of this decision in the experimental section.

A.2.3 Contextual Episodic RL

While imitation learning is a powerful approach to initialize the policy, it is often not sufficient to solve the learning problem. Thus, the robot relies on RL methods to further refine the policy. The goal for the robot, then, is to learn a policy $\pi(\boldsymbol{\theta}|\boldsymbol{s})$ that selects trajectories which hit the ball and wind it around the pole for arbitrary initial ball positions and velocities. More formally, the robot aims to find the policy π that maximizes the reward function

$$\begin{aligned} R^{\pi} &= \int_{\boldsymbol{s}} \mu(\boldsymbol{s}) \int_{\boldsymbol{\theta}} \pi(\boldsymbol{\theta}|\boldsymbol{s}) R_{\boldsymbol{\theta},\boldsymbol{s}} \mathrm{d}\boldsymbol{\theta} \mathrm{d}\boldsymbol{s}, \\ \pi^*(\boldsymbol{\theta}|\boldsymbol{s}) &= \arg\max_{\pi} R^{\pi}, \end{aligned} \tag{A.10}$$

where $\mu(\boldsymbol{s})$ is the distribution over contexts and $R_{\boldsymbol{\theta},\boldsymbol{s}}$ is the expected reward over all possible trajectories executed with actions $\boldsymbol{\theta}$ in context $\boldsymbol{s}$, i.e., $R_{\boldsymbol{\theta},\boldsymbol{s}} = \int_{\boldsymbol{\tau}} p(\boldsymbol{\tau}|\boldsymbol{\theta},\boldsymbol{s}) R(\boldsymbol{\tau},\boldsymbol{s}) \mathrm{d}\boldsymbol{\tau}$. The reward function $R(\boldsymbol{\tau},\boldsymbol{s})$ is used to determine the quality of trajectory $\boldsymbol{\tau}$ executed during an episode with context $\boldsymbol{s}$. For the robot tetherball game, we formulate the reward in terms of four components p_i, i.e., $R(\boldsymbol{\tau},\boldsymbol{s}) = p_1 + p_2 + p_3 + p_4$. The first term $p_1 = k_1(1 - \exp(d^2))$ reflects the minimum distance d between the paddle and the ball during the episode. Component $p_2 = k_2 J$ penalizes the total jerk of all joints during the whole trajectory. The third term p_3 is a penalty occurring when the robot misses the ball or hits it in the wrong direction. Finally, p_4 is an additional penalty given when the trajectory would cause a collision with the base the robot is standing on. The scale factors k_i are to transform costs into rewards and to scale the objectives magnitude.

To solve the problem expressed in Eq. (A.10), we use contextual relative entropy policy search (REPS) [Peters et al., 2010]. The update step for the policy $\pi(\boldsymbol{\theta}|\boldsymbol{s})$ is defined as the solution of the constrained optimization problem that determines the distribution $p(\boldsymbol{s},\boldsymbol{\theta}) = \mu(\boldsymbol{s})\pi(\boldsymbol{\theta},\boldsymbol{s})$ that maximizes the average return R^{π}. At the same time, REPS bounds the relative entropy (also called Kullback-Leibler divergence) between the new distribution p and the current one q to stay close to the observed data to balance exploration and exploitation, i.e., $\int_{\boldsymbol{s}} p(\boldsymbol{s},\boldsymbol{\theta}) \log\left(p(\boldsymbol{s},\boldsymbol{\theta})/q(\boldsymbol{s},\boldsymbol{\theta})\right) \mathrm{d}\boldsymbol{s} \leq \epsilon$. However, the context distribution $p(\boldsymbol{s}) = \int_{\boldsymbol{\theta}} p(\boldsymbol{s},\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}$ cannot be chosen freely, as it is specified by $\mu(\boldsymbol{s})$, i.e., we need to satisfy the constraint $\forall \boldsymbol{s} : p(\boldsymbol{s}) = \mu(\boldsymbol{s})$. For continuous context vectors, we implement this constraint by matching feature averages instead of single probability values, i.e., $\int p(\boldsymbol{s})\boldsymbol{\varphi}(\boldsymbol{s})\mathrm{d}\boldsymbol{s} = \hat{\boldsymbol{\varphi}}$, where $\boldsymbol{\varphi}(\boldsymbol{s})$ is a feature vector describing the context and $\hat{\boldsymbol{\varphi}}$ is the mean observed feature vector. The resulting constrained optimization problem is given by

$$\begin{aligned}
\max_{p} & \iint_{\boldsymbol{s},\boldsymbol{\theta}} p(\boldsymbol{s},\boldsymbol{\theta}) R_{\boldsymbol{\theta},\boldsymbol{s}} \mathrm{d}\boldsymbol{s}\mathrm{d}\boldsymbol{\theta}, \\
\text{s.t.} & \iint_{\boldsymbol{s},\boldsymbol{\theta}} p(\boldsymbol{s},\boldsymbol{\theta}) \log \frac{p(\boldsymbol{s},\boldsymbol{\theta})}{q(\boldsymbol{s},\boldsymbol{\theta})} \mathrm{d}\boldsymbol{s} \leq \epsilon, \\
& \iint_{\boldsymbol{s},\boldsymbol{\theta}} p(\boldsymbol{s},\boldsymbol{\theta}) \boldsymbol{\varphi}(\boldsymbol{s}) \mathrm{d}\boldsymbol{s}\mathrm{d}\boldsymbol{\theta} = \hat{\boldsymbol{\varphi}}, \\
& \iint_{\boldsymbol{s},\boldsymbol{\theta}} p(\boldsymbol{s},\boldsymbol{\theta}) \mathrm{d}\boldsymbol{s}\mathrm{d}\boldsymbol{\theta} = 1. \qquad \text{(A.11)}
\end{aligned}$$

Using the method of Lagrangian multipliers we derive the closed form for the new distribution update step

$$p(\boldsymbol{s},\boldsymbol{\theta}) \propto \exp\left(\frac{R_{\boldsymbol{s},\boldsymbol{\theta}} - V(\boldsymbol{s})}{\eta}\right), \qquad \text{(A.12)}$$

where the value function $V(\boldsymbol{s}) = \boldsymbol{\gamma}^{\mathsf{T}}\boldsymbol{\varphi}(\boldsymbol{s})$ is a context-dependent baseline, while η and $\boldsymbol{\gamma}$ are the Lagrangian parameters, found by optimizing the dual function.
As the relationship between the context-policy parameters pair $\{\boldsymbol{s},\boldsymbol{\theta}\}$ and the corresponding expected reward $R_{\boldsymbol{\theta},\boldsymbol{s}}$ is not known, sample rollouts are used to approximated the integrals in Eq. (A.11). To execute the i-th rollout, we first observe the context $\boldsymbol{s}^{[i]} \sim \mu(\boldsymbol{s})$. Subsequently, we sample the DMPs parameters $\boldsymbol{\theta}^{[i]} \sim \pi(\theta|\boldsymbol{s}^{[i]})$. Finally, we execute the DMPs with parametrization $\boldsymbol{\theta}^{[i]}$ in context $\boldsymbol{s}^{[i]}$ to obtain reward $R^{[i]}_{\theta,\boldsymbol{s}}$. Repeating this process for N rollouts, we obtain the average return R^{π} and the weights $p^{[i]}$ for a maximum likelihood update of the distribution $p(\boldsymbol{s},\boldsymbol{\theta})$

$$\begin{aligned}
p^{[i]} &\propto \exp\left(\frac{R^{[i]}_{\boldsymbol{\theta},\boldsymbol{s}} - V(\boldsymbol{s}^{[i]})}{\eta}\right), \\
R^{\pi} &= \langle R_{\boldsymbol{\theta},\boldsymbol{s}} \rangle. \qquad \text{(A.13)}
\end{aligned}$$

The quality of the learned policy and the convergence speed depend on the number of episodes N and by the basis functions $\boldsymbol{\varphi}$ used. Using many rollouts helps reducing the variance of an update step, but may also increase the number of episodes to converge. In the same way, using many features to approximate $V(\boldsymbol{s})$ can slow down the solution of the constrained optimization problem. We evaluate the effects of these decisions in the experimental section.

A.3 Evaluation

After detailing both the hand-crafted player as well as the RL-based player, we evaluate both and show the results in this section. We start by investigating the effects of the different parameters on the learned player and, subsequently, present the results of comparing the hand-crafted player to the learned player. The evaluations done in simulation are averaged over ten trials, while the evaluations done on the real robot are averaged over three trials. As REPS does not rely on using samples from only the last policy, we base the policy update on samples from the last

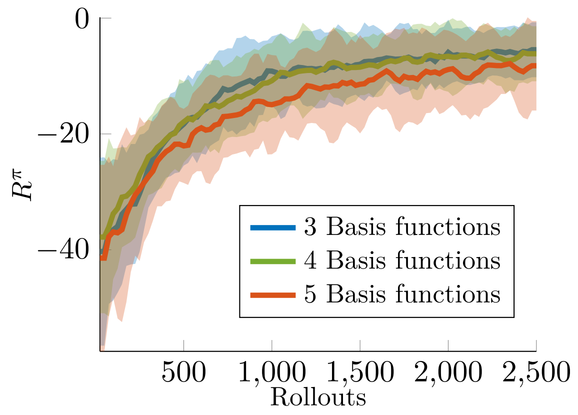


Figure A.2: Results for different number of Gaussian basis functions per DoF used for the DMP. Four bases achieve the best results. Using only three leads to less smooth trajectories, while using five bases increases the complexity of the learning problem without improving the quality of the final policy.

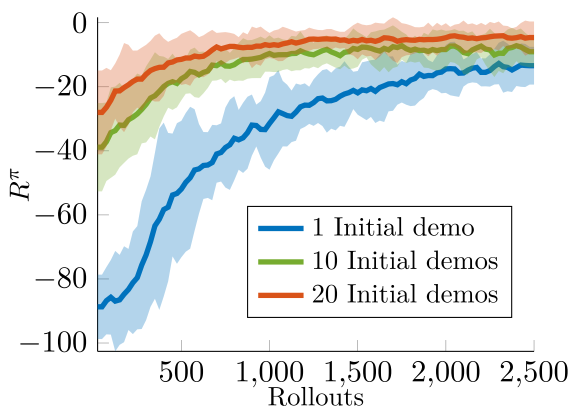


Figure A.3: Comparison over different number of demonstrations to initialize the policy π. Using only a single demonstration, the initial policy performance is considerably worse, as it is too explorative. Providing more demonstrations bootstraps the learning process and leads to better final policies.

20 policies to stabilize the policy update. This stabilization is important to keep the shape of the explorative variance of the policy in high dimensional action spaces. The KL bound for all experiments in Eq. (A.11) is set to $\epsilon = 0.9$. The parameters, α_z, α, β and τ of the DMPs are set to $\alpha_z = 0.8, \alpha = 5, \beta = 1, \tau = 1$. These parameters are chosen to reflect a qualitative trajectory shape that broadly fits the stroke profiles in our task.
For the real robot experiments, a Kinect sensor mounted on the ceiling delivers color and depth images at a rate of 30Hz. As the pivot point of the string cannot be detected by the vision system, an unscented Kalman filter [Simon, 2006] approximates the translation of the pivot point h along the pole.

A.3.1 Evaluation of DMP Basis Functions

An important parameter of DMPs is the number of basis function $\boldsymbol{\psi}(z_t)$ used to parametrize the forcing function $\boldsymbol{f}(z_t, \boldsymbol{\theta})$. A higher number of basis functions allows for more complicated trajectory shapes but also increases the dimensionality of the action space Θ and, thus, makes the learning problem harder. Fig. A.2 shows the comparison of learning with different numbers of Gaussian basis functions $\boldsymbol{\psi}(z_t)$. The plot shows that using three or four bases leads to comparable results, while using five bases per joint is increasing the complexity offsetting the potential benefit of learning more expressive trajectories. While the expected reward achieved with three and four bases is similar, the trajectories generated using four bases are overall smoother. Therefore we use four basis function for all following experiments.

A.3.2 Selection of Initial Demonstration

We also investigate the number of demonstration recorded to initialize the policy $\pi(\boldsymbol{\theta}, \boldsymbol{s})$. Fig. A.3 shows that providing only a single demonstration leads to poor results. The reason

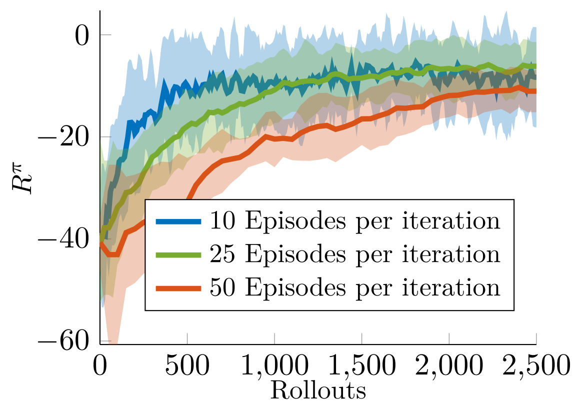


Figure A.4: Evaluation of different number of episodes per iteration. With ten samples the algorithm already performs well. Providing 25 samples slightly improves asymptotic quality but requires more overall samples. With 50 samples the policy does not converge in the limit of 2,500 total evaluations.

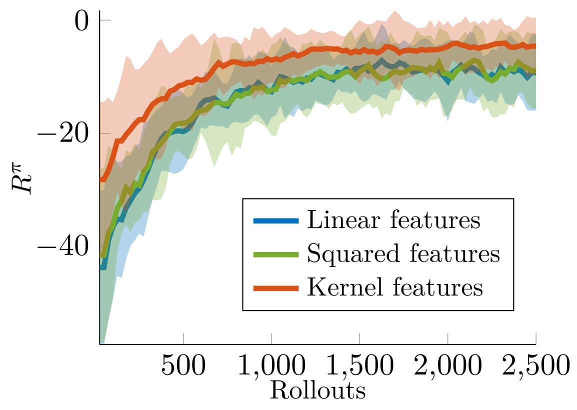


Figure A.5: Comparison over different types of basis functions to approximate the value function. While kernel based features are computationally more intensive and generally require more samples to work well, they also increase performance of the final policy.

is that with only one demonstration the initial covariance $\boldsymbol{\Sigma}$ needs to be set manually and uniformly in all dimensions (in our trials, $\boldsymbol{\Sigma} = 100\boldsymbol{I}$). This decision is not straightforward and can produce policies that are too explorative, significantly increasing the convergence time. Indeed, it can be noticed that after 2,500 episodes, the policy is still improving, but its quality is far below those policies whose initial exploration bounds were determined through additional demonstrations. When using multiple demonstrations, we choose a set of trajectory-context pairs that span the largest context space possible. We also allow low quality demonstrations in which the robot misses the ball. As expected, the more demonstrations are provided, the better the quality of the initial policy is. However, in order to restrict the number of demonstrations to a reasonable value, we decide not to use more than 20 demonstrations.

A.3.3 Evaluation of Value Function Features

Contextual REPS depends on a feature representation $\boldsymbol{\varphi}(\boldsymbol{s})$ of the state $\boldsymbol{s}$ to predict the expected quality $V(\boldsymbol{s})$ of that state. While simpler features like linear or squared features can be more robust and sample efficient, kernel based features promise better performance for complex problems. Fig. A.5 shows that for the problem of robot tetherball kernel based features indeed outperform the simpler alternatives. While linear and squared features can be used to achieve acceptable performance, the greater flexibility of kernel based features offers an edge in due to the non-linearity of the task.

A.3.4 Selection of Number of Rollouts

An important trade-off for any policy search algorithm is the number of evaluations used per policy update. While more samples yield more stable updates, they also slow down the learning process. Fig. A.4 shows that when using only 10 episodes the reward quickly increase but

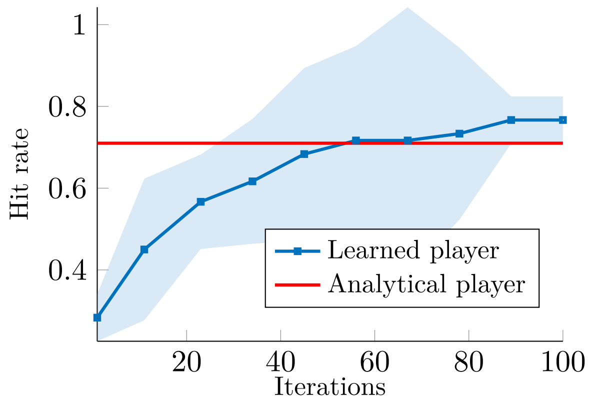


Figure A.6: Comparison of the real robots in the single player setup, averaged over three trials and ten rollouts per iteration. The learned player reaches performance levels similar to the hand-crafted one after about 50 iterations and outperforms it at the end of learning.

Table A.1: Match results for the real robots. The learned player defeats the hand-crafted one and achieves better overall hit rate, averaged over 25 matches.

Player	Hit rate	Matches won	Total score
handHcrafted	71%	6/25	8
Learned	**85%**	**19/25**	**38**

the asymptotic performance is slightly below what can be achieved when using 25 rollouts per iteration. Using 50 episodes slows down the learning process considerably and does not lead to a converged policy after the limit of 2,500 total episodes. The plot for using ten samples per iteration is more noisy as it is averaged over fewer rollouts per data point.

A.3.5 Hand-Crafted vs. Learned Player

According to the results reported, we set the learned player to use 20 initial demonstrations, four Gaussian basis functions for the DMP, kernel based features with three centers for approximating value function, and 25 episodes per iteration for the policy update. To get an initial estimate of their relative quality, we compare the learned player to the hand-crafted one in a single player experiment using the hit rate as measure of evaluation. The comparison is performed on the real system over three trials, every ten iterations and over ten episodes, summing up to 300 rollouts.

As shown in Fig. A.6, the learned player outperforms the hand-crafted system. The advantage of the learned system is due to its ability to learn to compensate the highly non-linear forward dynamics of the robot, as caused by springs in the joints. On the other hand, lacking a full inverse dynamics model, the hand-crafted player fully depends on accurate tracking of the planned trajectory. Therefore, improving the hand-crafted player would require a time-consuming process to explicitly approximate the dynamics of the robots.

A.3.6 Two Player Full Tetherball

As final evaluation, we let the real robots play full matches against each other and compare them according to the score. A match is started by the referee throwing the ball randomly to one of the robots and ends when none of the robots is able to hit the ball anymore. The score of a match is determined by how often the ball winds around the pole for each player, i.e., by the number of misses of the opponent. The two robots played a total of 25 games. The hand-crafted player won six of the 25 games, while the learned player won the remaining 19 games. Throughout the 25 games, the hand-crafted player scored eight times, while the learned player scored 38 times.

B Details of MORL Experiments

Here, we provide the details about the algorithms implementation used in Section 2.4. We devote a section to each domain to describe the settings omitted from the main body of the chapter. However, in order to improve the readability and the comparison of the settings of the domains, we have summarized MO-eREPS and MO-NES common parameters in Table B.1. We recall that ϵ is the bound to the KL divergence used by MO-eREPS, N_{EVAL} is the number of samples drawn from ϱ for the evaluation and ε is the unique parameter of the adaptive step-size algorithm used by MO-NES and described in [Peters, 2007]

$$\alpha = \sqrt{\frac{\varepsilon}{\nabla_{\omega}\mathcal{J}(\omega)^{\mathsf{T}}\boldsymbol{F}_{\omega}^{-1}\nabla_{\omega}\mathcal{J}(\omega)}}, \tag{B.1}$$

where $\boldsymbol{F}_{\omega}$ is the Fisher information matrix.

B.1 Water Reservoir Control

As the water reservoir domain has been fully described in the experimental section, we need only to define some parameters exploited by the algorithms. WS-eREPS scalarizes the objectives by 50 and 500 linearly spaced weights for the 2-objective and the 3-objective case, respectively, and its KL divergence bound is $\epsilon = 1$. RA follows 50 and 500 linearly spaced directions and, along with PFA, exploits the natural gradient [Peters and Schaal, 2008a] and the learning rate described in Equation (B.1) with $\varepsilon = 4$. SMS-EMOA has a maximum population size of 100 and 500, for the 2- and 3-objective cases respectively. Its crossover is uniform and the mutation, which has a chance of 80% to occur, adds a white noise to random chromosomes. At each iteration, the top 10% individuals are kept in the next generation to guarantee that the solution quality will not decrease.

PMGA uses the learning rate described in Equation (B.1) as well, with $\varepsilon = 2$. The algorithm parameterizes the manifold in the policy parameters space by a polynomial $f_{\boldsymbol{\rho}}(\boldsymbol{x})$, where $\boldsymbol{x}$ is the free sampling variable. A first degree polynomial and a second degree polynomial are used for the 2- and 3-objective, respectively. Both parameterizations are forced to pass near the extreme points of the Pareto frontier, computed through single-objective policy search, as described in the original paper [Pirotta et al., 2015]. In both cases, the manifold parameters to be learned by PMGA are six. During learning, one and five parameters $\theta^{[i]}$ are collected from the manifold, for the 2- and 3-objective case, respectively. Since PMGA requires the indicator function $\mathcal{S}$ to be differentiable, we employed the indicator presented in [Pirotta et al., 2015], consisting of a ratio between the distances of a point $\boldsymbol{J}(\theta^{[i]})$ to utopia and anti-utopia points, i.e.,

$$\mathcal{S}\left(\boldsymbol{J}(\theta^{[i]})\right) = \beta_1 \frac{\boldsymbol{J}(\theta^{[i]}) - \boldsymbol{J}_{\text{AU}}}{\boldsymbol{J}(\theta^{[i]}) - \boldsymbol{J}_{\text{U}}} - \beta_2.$$

Table B.1: Algorithms setup details for MO-eREPS and MO-NES.

Domain	2-obj. Water Reservoir	3-obj. Water Reservoir	LQR	Tetherball
$\boldsymbol{J}_{\text{U}}$	$-[0.5, 0.9]$	$-[0.5, 0.9, 0.001]$	$\mathbf{-283}$	$[-20, 20]$
$\boldsymbol{J}_{\text{AU}}$	$-[2.5, 11]$	$-[65, 12, 0.7]$	$\mathbf{-436}$	$[-50, 0]$
ϵ	1 (without IS) ; 2 (with IS)	2	2	1
ε	0.2	0.2	0.1	0.2
N_{EVAL}	500	1,000	10,000	200

We chose $\beta_1 = 1$ and $\beta_2 = 1$. Finally, as approximate frontiers returned by PMGA are continuous, they are discretized by sampling 500 and 1,000 points from the manifold for the 2- and 3-objective case, respectively.

B.2 Linear-Quadratic Regulator

As done in the previous section, we only need to provide the algorithms setup. Both WS-eREPS and RA perform 5,000 optimization procedures. WS-eREPS KL divergence bound is $\epsilon = 1$, while RA and PFA learning rate parameter is $\varepsilon = 5$. SMS-EMOA setting is the same as in the water reservoir domain and its maximum population size is 2,000.
As the LQR is defined only for control actions in the range $[-1, 0]$ and controls outside this range lead to divergence of the system, PMGA parameterizes the manifold by

$$\theta = f_{\boldsymbol{\rho}}(\boldsymbol{x}) = \frac{-1}{\exp(\mathsf{poly}(\boldsymbol{x}, \boldsymbol{\rho}, 2))}, \qquad \boldsymbol{x} \in \mathsf{simplex}([0, 1]^5),$$

where $\mathsf{poly}(\boldsymbol{x}, \boldsymbol{\rho}, 2)$ is the complete degree of variables $\boldsymbol{x}$, coefficients $\boldsymbol{\rho}$ and degree two, for a total of 75 parameters $\boldsymbol{\rho}$ to learn. During learning, 50 samples $\theta^{[i]}$ are collected from the manifold and the same scalarization function $\mathcal{S}$ as in the water reservoir control task is used. As for MO-eREPS and MO-NES, their frontiers are discretized by sampling 10,000 points from the manifold.

B.3 Simulated Robot Tetherball

The tetherball domain has been extensively described in the experimental section. Additional parameters regarding MO-eREPS and MO-NES are reported in Table B.1. WS-eREPS and RA perform 25 optimization procedures. WS-eREPS KL divergence bound is $\epsilon = 2$, as well as RA and PFA learning rate parameter $\varepsilon = 2$. SMS-EMOA crossover, mutation and elitism are the same as above, while its maximum population size is 200.

C Details of Exploration Experiment

Below, we present the pseudocode and the hyperparameters of the algorithms used in Section 3.4. In all of them, we kept two separate Q-tables for the behavior policy $\beta(a|s)$ and the target (greedy) policy $\pi(a|s)$. The former can be either initialized to zero or optimistically, while the latter always to zero. With the optimistic initialization, all entries of $Q(s,a)$ are set to $r_{\max}/(1-\gamma)$. The reason why we do not initialized $Q^\pi(s,a)$ optimistically is that if the agent does not visit all the state-action pairs, and thus never updated the corresponding Q-table entries, it would still had an optimistic belief over some states. The performance of the target policy can therefore be poor until all state-action pairs are visited. For example, in our experiments in the "prison" gridworld, the ϵ-greedy policy was finding some low-value treasures, but during the evaluation the greedy policy was not trying to collect them because it still had an optimistic belief over unvisited empty states.
In most of the pseudocode, we explicitly distinguish between $Q^\beta(s,a)$ and $Q^\pi(s,a)$. If simply $Q(s,a)$ appears, it means that both Q-tables are updated using the same equation.
In all algorithms, we evaluate $\pi(a|s)$, i.e., the greedy policy over $Q^\pi(s,a)$, every 50 training steps. Each evaluation episode has a horizon 10% longer than the training one. The learning rate is $\eta = 0.5$ and the discount factor $\gamma = 0.99$. For MDPs with stochastic transition function (Sections 3.4.2 and 3.4.2) we used $\eta = 0.1$. The ϵ-greedy initially has $\epsilon_0 = 1$, then it decays at step according to $\epsilon_{i+1} = \xi\epsilon_i$, where ξ is chosen such that $\epsilon_{end} = 0.1$ when the learning is over.
Finally we break ties with random selection, i.e., if two or more actions have the same max Q-value the winner is chosen randomly.

C.1 The Generic Q-Learning Scheme

In all experiments we used Q-learning with infinite replay memory. Classic Q-learning by Watkins and Dayan [1992] updates the Q-tables using only the current transition. The so-called *experience replay*, instead, keeps past transitions and use them multiple times for successive updates. This procedure became popular with deep Q-learning [Mnih et al., 2013] and brought substantial improvement to many RL algorithms.
In our experiments, we followed the setup proposed by Osband et al. [2019] and used an infinite memory, i.e., we stored all transitions, since it allowed much faster learning than classic Q-learning. This can also be seen as Dyna-Q [Sutton, 1990] without random memory sampling. In our experiments, the size of the domains does not pose any storage issue. When storage size is an issue, it is possible to fix the memory size and sample random mini-batches as in DQN [Mnih et al., 2013]. More details in Appendix C.6.
Algorithm 2 describes the generic scheme of Q-learning with infinite replay memory. TD learning is used on both $Q^\pi(s,a)$ and $Q^\beta(s,a)$, with the only difference being the initialization ($Q^\pi(s,a)$ is always initialized to zero).

Algorithm 2: Tabular Q-Learning with Replay Memory

```
Initialize $Q^\beta_0(s,a)$, $Q^\pi_0(s,a), i = 0$
While $i < i_{\text{BUDGET}}$ do
    Reset environment to state $s_1$
    For $t = 1 \dots H$ or until $s_t$ is terminal
        Select action according to $a_t \sim \beta(\cdot|s_t)$
        Transition to $s_{t+1} = \mathcal{P}\left(s_{t+1}|s_t,a_t\right)$ and receive reward $r_t = \mathcal{R}\left(s_t,a_t\right)$
        Store tuple $(s_t, a_t, r_t, s_{t+1})$
        For all tuples $(s,a,s',r)$ in the replay memory
            $\delta(s,a,s') = \begin{cases} r_t - Q_i(s,a) & \text{if } s \text{ is terminal} \\ r_t + \gamma \max_a Q_i(s',a) - Q_i(s,a) & \text{otherwise} \end{cases}$
            $Q_{i+1}(s,a) \leftarrow Q_i(s,a) + \eta\delta(s,a,s')$
        Update budget counter: $i \leftarrow i + 1$
```

Depending on $\beta(a|s)$, we have different exploration strategies, i.e.,

$$\beta(a_t|s_t) \sim \text{unif}\{\mathcal{A}\}, \qquad \text{random} \qquad \text{(C.1)}$$

$$\beta(a_t|s_t) \begin{cases} = \arg\max_a \left\{Q^\beta(s_t,a)\right\} & \text{with probability } 1-\epsilon, \\ \sim \text{unif}\{\mathcal{A}\} & \text{with probability } \epsilon, \end{cases} \qquad \epsilon\text{-greedy} \qquad \text{(C.2)}$$

$$\beta(a_t|s_t) = \arg\max_a \left\{ Q^\beta(s_t,a) + \kappa\sqrt{\frac{2\log\sum_{a_j} n(s_t,a_j)}{n(s_t,a)}} \right\}. \qquad \text{UCB1} \qquad \text{(C.3)}$$

Numerical stability and worst-case scenario. For UCB1 [Auer et al., 2002], a visitation count $n(s,a)$ is increased after every transition (see Algorithm 3, line 7). In classic bandit problems, UCB1 is initialized by executing all actions once, i.e., with $n(s,a) = 1$. In MDPs we cannot do that, i.e., we cannot arbitrarily set the agent in any state and execute all actions, thus $n(s,a) = 0$. Following the W-function bound in Section 3.3.3, we always add +1 inside the logarithm, and bound the square root to $(Q_{\max} - Q_{\min})/\kappa + \sqrt{2\log|\mathcal{A}|}$ when $n(s,a) = 0$. This correspond to the case where all actions but $\bar{a}$ has been executed once, and enforces the policy to choose $\bar{a}$. In our experiments, we set $Q_{\max} = r_{\max}/(1-\gamma)$ and $Q_{\min} = 0$.

C.2 Q-Learning with Augmented Reward

This version follows the same scheme of Algorithm 2 with ϵ-greedy exploration. The only difference is that the reward used to train the behavior policy is augmented with the exploration bonus proposed by Strehl and Littman [2008]. In our experiments, $\alpha = 0.1$, as used by Strehl and Littman [2008] and Bellemare et al. [2016].

Algorithm 3: Tabular Q-Learning with Replay Memory and Augmented Reward

```
1  Initialize $Q_0^\beta(s,a)$, $Q_0^\pi(s,a) = 0, n(s,a) = 0, i = 0$
2  While $i < i_{\text{BUDGET}}$ do
3      Reset environment to state $s_1$
4      For $t = 1...H$ or until $s_t$ is terminal
5          Select action according to the $\epsilon$-greedy policy of $Q_i^\beta(s,a)$
6          Transition to $s_{t+1} = \mathcal{P}(s_{t+1}|s_t,a_t)$ and receive reward $r_t = \mathcal{R}(s_t,a_t)$
7          Update visitation count: $n(s_t,a_t) \leftarrow n(s_t,a_t) + 1$
8          Store tuple $(s_t,a_t,r_t,s_{t+1})$
9          For all tuples $(s,a,s',r)$ in the replay memory
10             $\delta^\pi(s,a,s') = \begin{cases} r_t - Q_i^\pi(s,a) & \text{if } s \text{ is terminal} \\ r_t + \gamma \max_a Q_i^\pi(s',a) - Q_i^\pi(s,a) & \text{otherwise} \end{cases}$
11             $Q_{i+1}^\pi(s,a) \leftarrow Q_i^\pi(s,a) + \eta\delta^\pi(s,a,s')$
12             Augment reward: $r_t^+ = r_t + \alpha\, n(s_t,a_t)^{-1/2}$
13             $\delta^\beta(s,a,s') = \begin{cases} r_t^+ - Q_i^\beta(s,a) & \text{if } s \text{ is terminal} \\ r_t^+ + \gamma \max_a Q_i^\beta(s',a) - Q_i^\beta(s,a) & \text{otherwise} \end{cases}$
14             $Q_{i+1}^\beta(s,a) \leftarrow Q_i^\beta(s,a) + \eta\delta^\beta(s,a,s')$
15         Update budget counter: $i \leftarrow i + 1$
```

C.3 Q-Learning with Visitation Value

Algorithms 4 and 5 describe the novel method proposed in this paper. In our experiments $\kappa = r_{\max}/(1-\gamma)$ and $\gamma_w = 0.99$. For the chainworld in Section 3.4.2 we set $\gamma_w = 0.999$. With infinite horizon, in fact, this yielded better results by allowing the agent to explore for longer. For the stochastic gridworld in Section 3.4.2 we set $\gamma_w = 0.9$. As discussed in the results, in fact, a stochastic transition function naturally improves exploration, thus a smaller visitation discount was sufficient and yielded better results.
In Algorithm 4, $W_{\text{UCB}}^\beta(s,a)$ is initialized as Eq. (3.23). In Algorithm 5, line 6, we bound the square root to Eq. (3.26) when $n(s,a) = 0$. In our experiments, with $Q_{\max} = r_{\max}/(1-\gamma)$ and $Q_{\min} = 0$.

Algorithm 4: Tabular Q-Learning with Replay Memory and Visit. Value (UCB)

```
1  Initialize $Q_0^\beta(s,a)$, $Q_0^\pi(s,a) = 0, W^\beta_{\text{UCB},0}(s,a), n(s,a) = 0, i = 0$
2  While $i < i_{\text{BUDGET}}$ do
3      Reset environment to state $s_1$
4      For $t = 1...H$ or until $s_t$ is terminal
5          Select action according to
           $\beta(a_t|s_t) = \arg\max_a \left\{Q_i^\beta(s_t,a) + \kappa(1-\gamma_w)W^\beta_{\text{UCB},i}(s_t,a)\right\}$
6          Transition to $s_{t+1} = \mathcal{P}\left(s_{t+1}|s_t,a_t\right)$ and receive reward $r_t = \mathcal{R}\left(s_t,a_t\right)$
7          Update visitation count: $n(s_t,a_t) \leftarrow n(s_t,a_t) + 1$
8          Store tuple $(s_t,a_t,r_t,s_{t+1})$
9          For all tuples $(s,a,s',r)$ in the replay memory
10             $\delta(s,a,s') = \begin{cases} r_t - Q_i(s,a) & \text{if } s \text{ is terminal} \\ r_t + \gamma \max_a Q_i(s',a) - Q_i(s,a) & \text{otherwise} \end{cases}$
11             $Q_{i+1}(s,a) \leftarrow Q_i(s,a) + \eta\delta(s,a,s')$
12             Compute visitation reward according to Eq. (3.10)
13             $\delta^W(s,a,s') = \begin{cases} r_t^W - W^\beta_{\text{UCB},i}(s,a) & \text{if } s \text{ is terminal} \\ r_t^W + \gamma \max_a W^\beta_{\text{UCB},i}(s',a) - W^\beta_{\text{UCB},i}(s,a) & \text{otherwise} \end{cases}$
14             $W^\beta_{\text{UCB},i+1}(s,a) \leftarrow W^\beta_{\text{UCB},i}(s,a) + \eta\delta^W(s,a,s')$
15         Update budget counter: $i \leftarrow i + 1$
```

Algorithm 5: Tabular Q-Learning with Replay Memory and Visit. Value (Count)

```
1  Initialize $Q_0^\beta(s,a)$, $Q_0^\pi(s,a) = 0, n(s,a) = 0, W^\beta_{\text{N},0}(s,a) = 0, i = 0$
2  While $i < i_{\text{BUDGET}}$ do
3      Reset environment to state $s_1$
4      For $t = 1...H$ or until $s_t$ is terminal
5          Compute pseudocount: $\hat{n}(s_t,a) = (1-\gamma_w)W^\beta_{\text{N},i}(s_t,a)$
6          Select action according to $\beta(a_t|s_t) = \arg\max_a \left\{Q_i^\beta(s_t,a) + \kappa\sqrt{\frac{2\log\sum_{a_j}\hat{n}(s_t,a_j)}{\hat{n}(s_t,a)}}\right\}$
7          Transition to $s_{t+1} = \mathcal{P}\left(s_{t+1}|s_t,a_t\right)$ and receive reward $r_t = \mathcal{R}\left(s_t,a_t\right)$
8          Update visitation count: $n(s_t,a_t) \leftarrow n(s_t,a_t) + 1$
9          Store tuple $(s_t,a_t,r_t,s_{t+1})$
10         For all tuples $(s,a,s',r)$ in the replay memory
11             $\delta(s,a,s') = \begin{cases} r_t - Q_i(s,a) & \text{if } s \text{ is terminal} \\ r_t + \gamma \max_a Q_i(s',a) - Q_i(s,a) & \text{otherwise} \end{cases}$
12             $Q_{i+1}(s,a) \leftarrow Q_i(s,a) + \eta\delta(s,a,s')$
13             Compute visitation reward according to Eq. (3.15)
14             $\delta^W(s,a,s') = \begin{cases} r_t^W - W^\beta_{\text{N},i}(s,a) & \text{if } s \text{ is terminal} \\ r_t^W + \gamma \min_a W^\beta_{\text{N},i}(s',a) - W^\beta_{\text{N},i}(s,a) & \text{otherwise} \end{cases}$
15             $W^\beta_{\text{N},i+1}(s,a) \leftarrow W^\beta_{\text{N},i}(s,a) + \eta\delta^W(s,a,s')$
16         Update budget counter: $i \leftarrow i + 1$
```

Bootstrap algorithms have an ensemble of Q-tables $\{Q^b(s,a)\}_{b=1\dots B}$. The behavior policy is greedy over one Q-table, randomly chosen from the ensemble either at the beginning of the episode or at every step. The behavior Q-tables are initialized randomly. After initializing each $Q^b(s,a)$ either optimistically or to zero, we add random noise drawn from a Gaussian $\mathcal{N}(0,1)$. Then, each one is trained on a random mini-batch from the replay memory. In our experiments, we used batches of size 1,024 and an ensemble of $B = 10$ behavior Q-tables. The target Q-table is still updated as in previous algorithms, i.e., using the full memory.

Algorithm 6: Tabular Bootstrapped Q-Learning with Replay Memory

```
1  Initialize $Q^b_0(s,a)$ for $b = 1\dots B$, $Q^\pi_0(s,a) = 0, i = 0$
2  While $i < i_{\text{BUDGET}}$ do
3      Reset environment to state $s_1$
4      Select random Q-table: $Q^\beta_i(s,a) \sim \text{unif}\{Q^1_i(s,a),\dots,Q^B_i(s,a)\}$
5      For $t = 1...H$ or until $s_t$ is terminal
6          Select action according to the $\epsilon$-greedy policy of $Q^\beta_i(s,a)$
7          Transition to $s_{t+1} = \mathcal{P}(s_{t+1}|s_t,a_t)$ and receive reward $r_t = \mathcal{R}(s_t,a_t)$
8          Store tuple $(s_t,a_t,r_t,s_{t+1})$
9          For all tuples $(s,a,s',r)$ in the replay memory
10             $\delta^\pi(s,a,s') = \begin{cases} r_t - Q^\pi_i(s,a) & \text{if } s \text{ is terminal} \\ r_t + \gamma \max_a Q^\pi_i(s',a) - Q^\pi_i(s,a) & \text{otherwise} \end{cases}$
11             $Q^\pi_{i+1}(s,a) \leftarrow Q^\pi_i(s,a) + \eta\delta^\pi(s,a,s')$
12         For all behavior Q-tables $Q^b_i(s,a)$
13             Sample mini-batch from the replay memory
14             For all tuples $(s,a,s',r)$ in the mini-batch
15                 $\delta^b(s,a,s') = \begin{cases} r_t - Q^b_i(s,a) & \text{if } s \text{ is terminal} \\ r_t + \gamma \max_a Q^b_i(s',a) - Q^b_i(s,a) & \text{otherwise} \end{cases}$
16                 $Q^b_{i+1}(s,a) \leftarrow Q^b_i(s,a) + \eta\delta^b(s,a,s')$
17         Update budget counter: $i \leftarrow i + 1$
```

The above pseudocode defines the generic bootstrap approach proposed by Osband et al. [2016a]. In Section 3.4 we also compared to two slightly different versions. The first uses approximate Thompson sampling and randomly selects the behavior Q-table at every step instead of at the beginning of the episode [D'Eramo et al., 2019]. The second keeps the sampling at the beginning of the episode, but further regularizes the TD error [Osband et al., 2018, 2019]. The regularization is the squared ℓ_2-norm of the distance of the Q-tables from "prior Q-tables" $Q^p(s,a)$, resulting in the following regularized TD update

$$\delta^b_{reg}(s,a,s') = \delta^b(s,a,s') + \nu\left(Q^p(s,a) - Q^b(s,a)\right), \tag{C.4}$$

wehere ν is the regularization coefficient which we set to $\nu = 0.1$ (in the original paper $\nu = 1$, but dividing it by ten worked best in our experiments).

In theory, $Q^p(s,a)$ should be drawn from a distribution. In practice, the same authors fix it at the beginning of the learning, and for the deep sea domain they set it to zero. This configuration also worked best in our experiments.

C.5 Horizons.

All environments in Section 3.4 are infinite horizon MDPs, except for the deep sea which has a finite horizon equal to its depth. However, for practical reason we end the episode after H steps and reset the agent to the initial state. Table C.1 summarizes the training horizon H for each environment, as well as the steps needed for the optimal policy to find the highest reward. Also, recall that an episode can end prematurely if a terminal state (e.g., a reward state) is reached. Finally, notice that the agent receives the reward on state-action transition, i.e., it needs to execute an additional action in the state with a treasure to receive the reward. This is why, for instance, the agent needs 9 steps instead of 8 to be rewarded in the 5×5 gridworlds.

Table C.1: Time horizons H for the tabular MDPs presented in Section 3.4.

	Deep Sea	Taxi	Deep Grid.	Grid. (Toy)	Grid. (Prison)	Grid. (Wall)
Optimal	Depth	29	11	9	9	135
Short H.	Depth	33	55	11	11	330
Long H.	Depth	66	110	22	22	660
Stochastic	-	-	55	15	25	-

C.6 Infinite Memory vs No Memory

Except for the chainworld in Section 3.4.2, the evaluation in Section 3.4 was conducted with Q-learning with infinite replay memory, following the same setup of Osband et al. [2019] as described in Section C.1. Here, we present additional results without replay memory, i.e., using classic Q-learning by Watkins and Dayan [1992] which updates the Q-tables using only the current transition. We show that the proposed method benefits the most from the replay memory, but can learn even without it while other algorithms cannot. We compare ϵ-greedy exploration, our proposed visitation-value-based exploration, and bootstrapped exploration. Nevertheless, the latter needs a replay memory to randomize the Q-tables update, as each Q-table is updated using different random samples. Thus, for bootstrapped Q-learning, we compare the use of finite memory and small batches to infinite memory and large batches..
Algorithm 7 describes the generic scheme of Q-learning without replay memory. The only difference from Algorithm 2 is the absence of the loop over the memory. The same modification is done to Algorithms 5 and 6 to derive the version of our algorithms without infinite memory. Boostrapped Q-learning is the same as Algorithm 4 but with finite memory. The finite memory keeps at most $20H$ steps, where H is the episode horizon, and uses mini-batches of 32 samples instad of 1,024.

Results. Figure C.1 shows the results on two gridworlds. The "toy" one has a single reward, while the "prison" one has also distractors. We refer to Section 3.4.1 for their full description. In the

Algorithm 7: Classic Tabular Q-Learning

```
1  Initialize $Q_0^\beta(s,a)$, $Q_0^\pi(s,a), i = 0$
2  While $i < i_{\text{BUDGET}}$ do
3      Reset environment to state $s_1$
4      For $t = 1 \ldots H$ or until $s_t$ is terminal
5          Select action according to $a_t \sim \beta(\cdot|s_t)$
6          Transition to $s_{t+1} = \mathcal{P}(s_{t+1}|s_t, a_t)$ and receive reward $r_t = \mathcal{R}(s_t, a_t)$
7          Store tuple $(s_t, a_t, r_t, s_{t+1})$
8          $\delta(s_t, a_t, s_{t+1}) = \begin{cases} r_t - Q_i(s_t, a_t) & \text{if } s_t \text{ is terminal} \\ r_t + \gamma \max_a Q_i(s_{t+1}, a) - Q_i(s_t, a_t) & \text{otherwise} \end{cases}$
9          $Q_{i+1}(s_t, a_t) \leftarrow Q_i(s_t, a_t) + \eta\delta(s_t, a_t, s_{t+1})$
10         Update budget counter: $i \leftarrow i + 1$
```

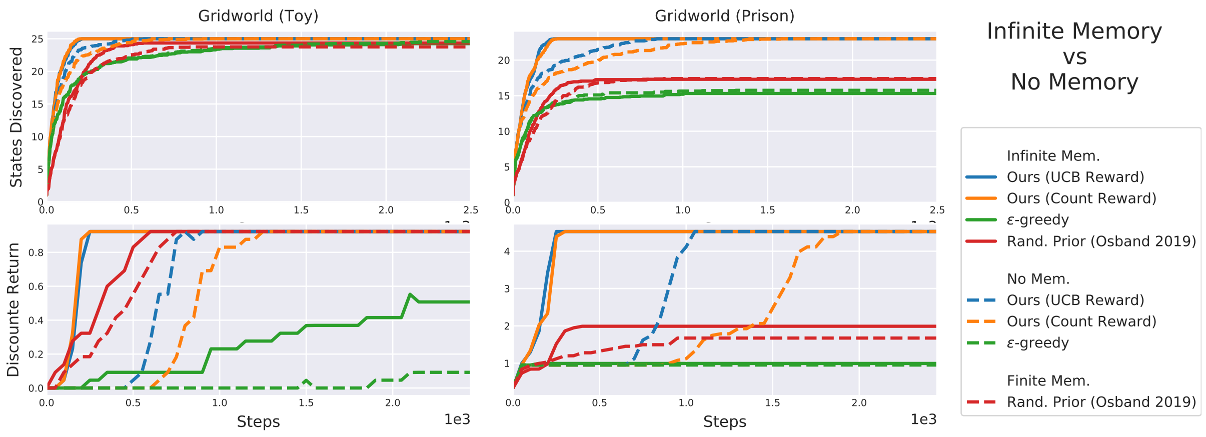


Figure C.1: Comparison of the proposed exploration against ϵ-greedy and bootstrapped exploration with and without replay memory. Each line denotes the average over 20 seeds. Only exploration based on the visitation value always learns in both domains, i.e., both with and withou local optima (distractor rewards), regardless of the memory.

first domain, all algorithms perform well except for ϵ-greedy exploration (green), which missed the reward in some trials. The use of infinite memory helped all algorithms, allowing boostrap and visitation-value-based to learn substantially faster. For example on average, without replay memory our algorithms (blue and orange) converged in 1,000 steps, while with replay memory they took only 250 steps, i.e., $\times 4$ times faster. Bootstrapped Q-learning (red), performance does not change substantially with finite and infinite memory.

However, in the second domain only visitation-value-based exploration always learns, regardless of the memory. As already discussed in Section 3.4.1, bootstrap performs poorly due to distractor rewards, and the infinite memory does not help it. For visitation-value-based exploration, the speed up gained from the infinite memory is even larger than before, due to the higher complexity of the domain. In this case, in fact, with infinite memory learn $\times 4$ and $\times 8$ times faster than without memory (blue and orange, respectively).

This evaluation shows that the proposed method benefits the most from the replay memory, but can learn even without it while other algorithms cannot.

C.7 Recap Tables

Here, we report tables summarizing the results of Section 3.4.1 in terms of discovery (percentage of states discovered during learning) and success (number of times the algorithm learned the optimal policy within steps limit). These are extended version of Table 3.1, which reported results only for the "zero initialization short horizon" scenario.

Table C.2: Deep sea results recap.

	Algorithm	Discovery (%)	Success (%)
Zero Init.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	99.90 ± 0.01	100 ± 0
	Bootstr. (Osband 2016a)	99.77 ± 0.05	100 ± 0
	Bootstr. (D'Eramo 2019)	63.25 ± 3.31	0 ± 0
	UCB1 (Auer 2002)	55.72 ± 0.34	0 ± 0
	Expl. Bonus (Strehl 2008)	85.65 ± 1.0	0 ± 0
	ϵ-greedy	57.74 ± 1.11	0 ± 0
	Random	58.59 ± 1.35	0 ± 0
Opt. Init.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	76.85 ± 0.3	0 ± 0
	Expl. Bonus (Strehl 2008)	98.79 ± 0.21	0 ± 0
	ϵ-greedy	98.79 ± 0.20	0 ± 0
	Random	58.59 ± 1.35	0 ± 0

Table C.3: Taxi results recap.

	Algorithm	Discovery (%)	Success (%)
Zero Init. & Short Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	69.60 ± 2.96	13.07 ± 2.96
	Bootstr. (Osband 2016a)	52.44 ± 7.55	18.77 ± 9.24
	Bootstr. (D'Eramo 2019)	22.44 ± 1.81	1.9 ± 1.47
	UCB1 (Auer 2002)	31.17 ± 0.70	1.53 ± 1.37
	Expl. Bonus (Strehl 2008)	74.62 ± 2.24	17.6 ± 2.56
	ϵ-greedy	29.64 ± 0.98	1.92 ± 1.49
	Random	29.56 ± 0.98	1.92 ± 1.49
Opt. Init. & Short Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	99.94 ± 0.08	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	29.56 ± 0.98	1.92 ± 1.49
Zero Init. & Long Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	76.38 ± 2.31	15.69 ± 2.93
	Bootstr. (Osband 2016a)	53.80 ± 6.63	29.85 ± 12.72
	Bootstr. (D'Eramo 2019)	36.34 ± 3.16	6.54 ± 2.62
	UCB1 (Auer 2002)	49.66 ± 2.45	8.35 ± 1.32
	Expl. Bonus (Strehl 2008)	92.83 ± 2.26	76.18 ± 16.09
	ϵ-greedy	42.26 ± 2.52	7.66 ± 0.03
	Random	55.89 ± 0.98	2.06 ± 1.49
Opt. Init. & Long Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
	Bootstr. (D'Eramo 2019)	95.45 ± 0	100 ± 0
	UCB1 (Auer 2002)	98.74 ± 0.09	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	55.89 ± 0.98	2.06 ± 1.49

Table C.4: Deep gridworld results recap.

	Algorithm	Discovery (%)	Success (%)
Zero Init. & Short Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	75 ± 9.39	59.03 ± 4.2
	Bootstr. (Osband 2016a)	60.82 ± 11.78	63.35 ± 6.89
	Bootstr. (D'Eramo 2019)	63.73 ± 10.35	56.85 ± 0.06
	UCB1 (Auer 2002)	92.18 ± 0.36	56.88 ± 0
	Expl. Bonus (Strehl 2008)	95.45 ± 1.57	69.81 ± 8.84
	ϵ-greedy	74.36 ± 4.42	56.88 ± 0
	Random	92.45 ± 0.64	56.88 ± 0
Opt. Init. & Short Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	92.45 ± 0.64	56.88 ± 0
Zero Init. & Long Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	78.18 ± 8.97	56.88 ± 0
	Bootstr. (Osband 2016a)	77.64 ± 10.02	69.81 ± 8.84
	Bootstr. (D'Eramo 2019)	71.64 ± 10.35	56.85 ± 0.06
	UCB1 (Auer 2002)	95.54 ± 0.46	56.4 ± 3.12
	Expl. Bonus (Strehl 2008)	95 ± 1.66	65.5 ± 7.71
	ϵ-greedy	76.45 ± 4.25	56.88 ± 0
	Random	92.09 ± 0.69	56.88 ± 0
Opt. Init. & Long Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	92.09 ± 0.69	56.88 ± 0

Table C.5: Gridworld (toy) results recap.

	Algorithm	Discovery (%)	Success (%)
Zero Init. & Short Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	99.8 ± 0.39	100 ± 0
	Bootstr. (Osband 2016a)	99.2 ± 0.72	100 ± 0
	Bootstr. (D'Eramo 2019)	84.8 ± 4.33	40 ± 21.91
	UCB1 (Auer 2002)	97.4 ± 0.99	50 ± 22.49
	Expl. Bonus (Strehl 2008)	99.8 ± 0.39	100 ± 0
	ϵ-greedy	97.4 ± 1.17	45 ± 22.25
	Random	97.2 ± 1.15	45 ± 22.25
Opt. Init. & Short Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	97.2 ± 1.15	45 ± 22.25
Zero Init. & Long Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	99.8 ± 0.39	100 ± 0
	Bootstr. (D'Eramo 2019)	94.6 ± 1.9	70 ± 20.49
	UCB1 (Auer 2002)	100 ± 0	98.5 ± 0.22
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	100 ± 0	100 ± 0
Opt. Init. & Long Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	100 ± 0	100 ± 0

Table C.6: Gridworld (Prison) results recap.

	Algorithm	Discovery (%)	Success (%)
Zero Init. & Short Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	72.39 ± 6.9	44.44 ± 14.69
	Bootstr. (Osband 2016a)	64.35 ± 11.1	33.03 ± 8.54
	Bootstr. (D'Eramo 2019)	54.78 ± 6.69	25.61 ± 8.39
	UCB1 (Auer 2002)	80.78 ± 2.56	32.31 ± 4.11
	Expl. Bonus (Strehl 2008)	85.87 ± 3	46.96 ± 12.35
	ϵ-greedy	58.38 ± 5.27	22.84 ± 2.48
	Random	76.96 ± 3.08	35.36 ± 10.37
Opt. Init. & Short Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	76.96 ± 3.08	35.36 ± 10.37
Zero Init. & Long Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	77.83 ± 6.93	56.04 ± 14.79
	Bootstr. (Osband 2016a)	67.61 ± 9.75	44.12 ± 13.02
	Bootstr. (D'Eramo 2019)	59.57 ± 9.92	29.31 ± 4.11
	UCB1 (Auer 2002)	90.17 ± 2.55	51.55 ± 10.67
	Expl. Bonus (Strehl 2008)	88.91 ± 3.1	54.48 ± 11.63
	ϵ-greedy	67.39 ± 5.16	24.69 ± 3.31
	Random	86.09 ± 3.77	49.73 ± 11.5
Opt. Init. & Long Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	86.09 ± 3.77	49.73 ± 11.5

Table C.7: Gridworld (wall) results recap.

	Algorithm	Discovery (%)	Success (%)
Zero Init. & Short Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	9.44 ± 3.14	0.14 ± 0.06
	Bootstr. (Osband 2016a)	6.35 ± 2.74	0.23 ± 0.08
	Bootstr. (D'Eramo 2019)	12.55 ± 4.45	0.18 ± 0.08
	UCB1 (Auer 2002)	24.7 ± 4.45	0.46 ± 0.03
	Expl. Bonus (Strehl 2008)	45.16 ± 2.29	0.52 ± 0.04
	ϵ-greedy	38.15 ± 1.95	0.39 ± 0.08
	Random	65.28 ± 0.45	0.59 ± 0
Opt. Init. & Short Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	65.28 ± 0.45	0.59 ± 0
Zero Init. & Long Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	13.05 ± 3.83	0.2 ± 0.08
	Bootstr. (Osband 2016a)	8.69 ± 3.53	0.2 ± 0.08
	Bootstr. (D'Eramo 2019)	11.64 ± 5.24	0.23 ± 0.08
	UCB1 (Auer 2002)	21.44 ± 3.64	0.44 ± 0.03
	Expl. Bonus (Strehl 2008)	47.75 ± 1.68	0.55 ± 0.03
	ϵ-greedy	42.53 ± 1.89	0.54 ± 0.03
	Random	70.51 ± 0.75	0.59 ± 0
Opt. Init. & Long Hor.	**Ours (UCB Reward)**	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Ours (Count Reward)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
	Rand. Prior (Osband 2019)	100 ± 0	100 ± 0
	Bootstr. (Osband 2016a)	100 ± 0	100 ± 0
	Bootstr. (D'Eramo 2019)	100 ± 0	100 ± 0
	UCB1 (Auer 2002)	100 ± 0	100 ± 0
	Expl. Bonus (Strehl 2008)	100 ± 0	100 ± 0
	ϵ-greedy	100 ± 0	100 ± 0
	Random	70.51 ± 0.75	0.59 ± 0

D Details of TD-REG Experiments

Here, we provide the details about the algorithms implementation used in Section 4.5. We devote a section to each domain to describe the settings and the experiments omitted from the main body of the chapter.

D.1 2D Linear-Quadratic Regulator

The LQR problem is defined by the following discrete-time dynamics

$$s' = As + Ba + \mathcal{N}(0, 0.1^2), \qquad a = Ks, \qquad \mathcal{R}(s,a) = -s^\intercal X s - a^\intercal Y a,$$

where $A, B, X, Y \in \mathbb{R}^{d\times d}$, X is a symmetric positive semidefinite matrix, Y is a symmetric positive definite matrix, and $K \in \mathbb{R}^{d\times d}$ is the control matrix. The policy parameters we want to learn are $\theta = \text{vec}(K)$. For simplicity, dynamics are not coupled, i.e., A and B are identity matrices, and both X and Y are identity matrices as well.
Although it may look simple, the LQR presents some challenges. First, the system can become easily unstable, as the control matrix K has to be such that the matrix $(A+BK)$ has eigenvalues of magnitude smaller than one. Therefore, policy updates cannot be too large, in order to prevent divergence. Second, the reward is unbounded and the expected return can be very large, especially at the beginning with an initial random policy. As a consequence, the initial TD error can be very large as well. Third, states and actions are unbounded and cannot be normalized in [0,1], a common practice in RL.
However, we can compute in closed form both the expected return and the Q-function, being able to easily assess the quality of the evaluated algorithms. More specifically, the Q-function is quadratic in the state and in the action, i.e.,

$$Q^\pi(s,a) = Q_0 + s^\intercal Q_{ss} s + a^\intercal Q_{aa} a + s^\intercal Q_{sa} a,$$

where $Q_0, Q_{ss}, Q_{aa}, Q_{sa}$ are matrices computed in closed form given the MDP characteristics and the control matrix K. It should be noted that the linear terms are all zero.
In the evaluation below, we use a 2-dimensional LQR, resulting in four policy parameters. The Q-function is approximated by $\widehat{Q}(s,a;\theta) = \phi(s,a)^\intercal \omega$, where $\phi(s,a)$ are features. We evaluate two different features: polynomial of second degree (*quadratic features*) and polynomial of third degree (*cubic features*). We know that the true Q-function is quadratic without linear features, therefore quadratic features are sufficient. By contrast, cubic features could overfit. Furthermore, quadratic features result in 15 parameters ω to learn, while the cubic one has 35.
In our experiments, the initial state is uniformly drawn in the interval $[-10, 10]$. The Q-function parameters are initialized uniformly in $[-1, 1]$. The control matrix is initialized as $K = -K_0^\intercal K_0$ to enforce negative semidefiniteness, where K_0 is drawn uniformly in [-0.5, -0.1].
Along with the expected return, we show the trend of two mean squared TD errors (MSTDE) of the critic $\widehat{Q}(s,a;\omega)$: one is estimated using the TD error, the other is computed in closed form

using the true $Q^\pi(s,a)$ defined above. It should be noticed that $Q^\pi(s,a)$ is not the optimal Q-function (i.e., of the optimal policy), but the true Q-function with respect to the current policy. We also show the learning of the diagonal entries of K in the policy parameter space. These parameters, in fact, are the most relevant because the optimal K is diagonal as well, due to the reward and transition functions characteristics ($A = B = X = Y = I$).
All results are averaged over 50 trials. In all trials, the random seed is fixed and the initial parameters are the same (all random). In expected return plots, we bounded the expected return to -10^3 and the MSTDE to $3{\cdot}10^5$ for the sake of clarity, since in the case of unstable policies (i.e., when the matrix $(A + BK)$ has eigenvalues of magnitude greater than or equal to one) the expected return and the TD error are $-\infty$.

D.1.1 DPG Evaluation on the LQR

In this section, we evaluate five versions of deterministic policy gradient [Silver et al., 2014]. In the first three, the learning of the critic happens in the usual actor-critic fashion. The Q-function is learned independently from the policy and a target Q-function of parameters $\bar\omega$, assumed to be independent from the critic, is used to improve stability, i.e.,

$$\delta_Q(s,a,s';\theta,\omega,\bar\omega) = r + \gamma\widehat{Q}(s',\pi(s';\theta);\bar\omega) - \widehat{Q}(s,a;\omega). \tag{D.1}$$

Under this assumption, the critic is updated by following the SARSA gradient

$$\begin{aligned}\nabla_\omega \mathbb{E}_{\mu_\beta(s),\beta(a|s),\mathcal{P}(s'|s,a)}\left[\delta_Q(s,a,s';\theta,\omega,\bar\omega)^2\right] = \\ \mathbb{E}_{\mu_\beta(s),\beta(a|s),\mathcal{P}(s'|s,a)}\left[\delta_Q(s,a,s';\theta,\omega,\bar\omega)\nabla_\omega\widehat{Q}(s,a;\omega)\right],\end{aligned} \tag{D.2}$$

where $\beta(a|s)$ is the behavior policy used to collect samples. In practice, $\bar\omega$ is a copy of ω. We also tried a soft update, i.e., $\bar\omega_{t+1} = \tau_\omega\omega_t + (1-\tau_\omega)\bar\omega_t$, with $\tau_\omega \in (0,1]$, as in DDPG [Lillicrap et al., 2016], the deep version of DPG. However, the performance of the algorithms decreased (TD-regularized DPG still outperformed vanilla DPG). We believe that, since for the LQR we do not approximate the Q-function with a deep network, the soft update just restrains the convergence of the critic.
These three versions of DPG differ in the policy update. The first algorithm (**DPG**) additionally uses a target actor of parameters $\bar\theta$ for computing the Q-function targets, i.e.,

$$\delta_Q(s,a,s';\bar\theta,\omega,\bar\omega) = r + \gamma\widehat{Q}(s',\pi(s';\bar\theta);\bar\omega) - \widehat{Q}(s,a;\omega), \tag{D.3}$$

to improve stability. The policy is updated softly at each iteration, i.e., $\bar\theta_{t+1} = \tau_\omega\theta_t + (1-\tau_\theta)\bar\theta_t$, with $\tau_\theta \in (0,1]$ The second algorithm (**DPG TD-REG**) applies the penalty function $G(\theta)$ presented in this chapter and does not use the target policy, i.e.,

$$\delta_Q(s,a,s';\theta,\omega,\bar\omega) = r + \gamma\widehat{Q}(s',\pi(s';\theta);\bar\omega) - \widehat{Q}(s,a;\omega), \tag{D.4}$$

in order to compute the full derivative with respect to θ for the penalty function (Eq. (4.24)). The third algorithm (**DPG NO-TAR**) is like DPG, but also does not use the target policy. The purpose of this version is to check that the benefits of our approach do not come from the lack of the target actor, but rather from the TD-regularization.

The last two versions are twin delayed DPG (**TD3**) [Fujimoto et al., 2018], which achieved state-of-the-art results, and its TD-regularized counterpart (**TD3 TD-REG**). TD3 proposes three modifications to DPG. First, in order to reduce overestimation bias, there are two critics. Only the first critic is used to update the policy, but the TD target used to update both critics is given by the minimum of their TD target. Second, the policy is not updated at each step, but the update is delayed in order to reduce per-update error. Third, since deterministic policies can overfit to narrow peaks in the value estimate (a learning target using a deterministic policy is highly susceptible to inaccuracies induced by function approximation error) noise is added to the target policy. The resulting TD error is

$$\delta_{Q_i}(s,a,s';\bar{\theta},\omega,\bar{\omega}_1,\bar{\omega}_2) = r + \gamma \min_{j=1,2} \widehat{Q}(s',\pi(s';\bar{\theta})+\xi;\bar{\omega}_j) - \widehat{Q}(s,a;\omega_i), \qquad \text{(D.5)}$$

where the noise $\xi = \texttt{clip}(\mathcal{N}(0,\tilde{\sigma}),-c,c)$ is clipped to keep the target close to the original action. Similarly to DPG TD-REG, **TD3 TD-REG** removes the target policy (but keeps the noise ξ) and adds the TD-regularization to the policy update. Since TD3 updates the policy according to the first critic only, the TD-regularization considers the TD error in Eq. (D.5) with $i = 1$.

Hyperparameters

- Maximum number of steps per trajectory: 150.
- Exploration: Gaussian noise (diagonal covariance matrix) added to the action. The standard deviation σ starts at 5 and decays at each step according to $\sigma_{t+1} = 0.95\sigma_t$.
- Discount factor: $\gamma = 0.99$.
- Steps collected before learning (to initialize the experience replay memory): 100.
- Policy and TD errors evaluated every 100 steps.
- At each step, all data collected (state, action, next state, reward) is stored in the experience replay memory, and a mini-batch of 32 random samples is used for computing the gradients.
- DPG target policy update coefficient: $\tau_\theta = 0.01$ (DPG NO-TAR is like DPG with $\tau_\theta = 1$). With $\tau_\theta = 0.1$ results were worse. With $\tau_\theta = 0.001$ results were almost the same.
- ADAM hyperparameters for the gradient of ω: $\alpha = 0.01$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$. With higher α all algorithms were unstable, because the critic was changing too quickly.
- ADAM hyperparameters for the gradient of θ: $\alpha = 0.0005$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$. Higher α led all algorithms to divergence, because the condition for stability (magnitude of the eigenvalues of $(A + BK)$ smaller than one) was being violated.
- Regularization coefficient: $\eta_0 = 0.1$ and then it decays according to $\eta_{t+1} = 0.999\eta_t$.
- In TD3 original paper, the target policy noise is $\xi \sim \mathcal{N}(0, 0.2)$ and is clipped in [-0.5,0.5]. However, the algorithm was tested on tasks with action bounded in [-1,1]. In the LQR, instead, the action is unbounded, therefore we decided to use $\xi \sim \mathcal{N}(0, 2)$ and to clip it in $[-\sigma_t/2, \sigma_t/2]$, where σ_t is the current exploration noise. We also tried different strategies, but we noticed no remarkable differences. The noise is used only for the policy and critics updates, and it is removed for the evaluation of the TD error for the plots.

- In TD3 and TD3 TD-REG, the second critic parameters are initialized uniformly in $[-1, 1]$, like the first critic.
- In TD3 and TD3 TD-REG, the policy is updated every two steps, as in the original paper.
- The learning of all algorithms ends after $12,000$ steps.

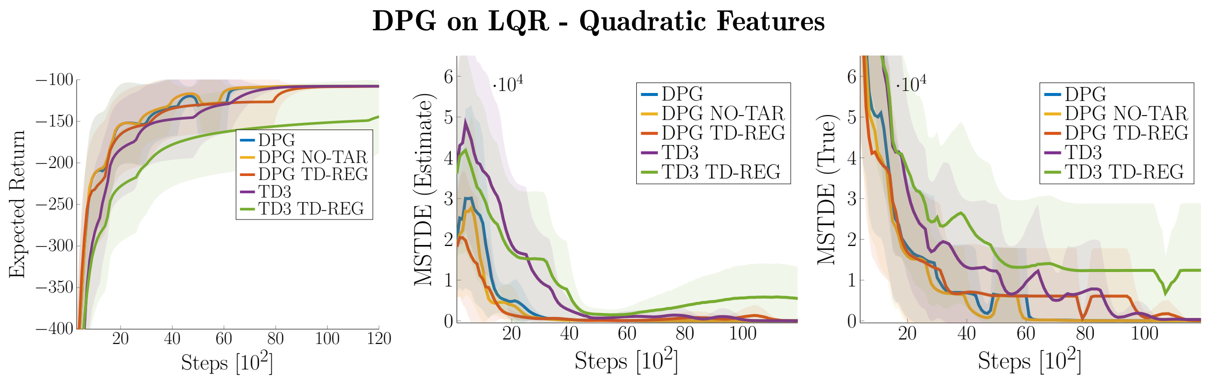


Figure D.1: All algorithms perform similarly (DPG and DPG NO-REG almost overlap), because quadratic features are sufficient to approximate the true Q-function. Only TD3 TD-REG did not converge within the time limit in two runs, but it did not diverge either.

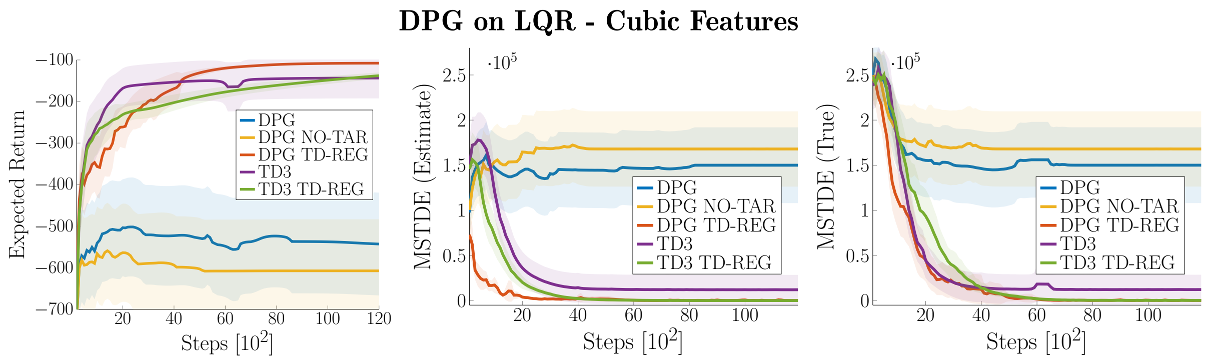


Figure D.2: By contrast, with cubic features the critic is prone to overfit, and DPG and DPG NO-TAR diverged 24 and 28 times, respectively. TD3 performed better, but still diverged two times. TD-regularized algorithms, instead, never diverged, and DPG TD-REG always learned the true Q-function and the optimal policy within the time limit. Similarly to Figure D.1, TD3 TD-REG is the slowest, and in this case it never converged within the time limit, but did not diverge either (see also Figure D.5b).

Results

Figures D.1 and D.5a show the results using quadratic features. Since these features are very unlikely to overfit, all algorithms perform almost the same. However, from Figure D.5a we can clearly see that the TD-regularization keeps the policy parameters on a more straight path towards the optimum, avoiding detours. Both the TD-regularization and TD3 also result in smaller, but safer, update steps. This behavior is reflected by the learning curves in Figure D.1, as DPG and DPG NO-TAR converge slightly faster. However, using both TD3 and TD-REG at the same time can result in excessively slow learning. The green arrows in Figure D.5a are, in

fact, the smallest, and in Figure D.1 TD3 TD-REG did not converge within the time limits in two runs (but it did not diverge either).
Looking at the TD error plots, it is interesting to notice that the estimated TD error is always smaller than the true TD error, meaning that the critic underestimates the TD error. This is a normal behavior, considering the stochasticity of the behavior policy and of the environment. It is also normal that this overestimation bias is less prominent in TD3, thanks to the use of two critics and to delayed policy updates. However, TD3 TD-REG is surprisingly the only algorithm increasing the estimated TD error around mid-learning. We further investigate this behavior at the end of this section.

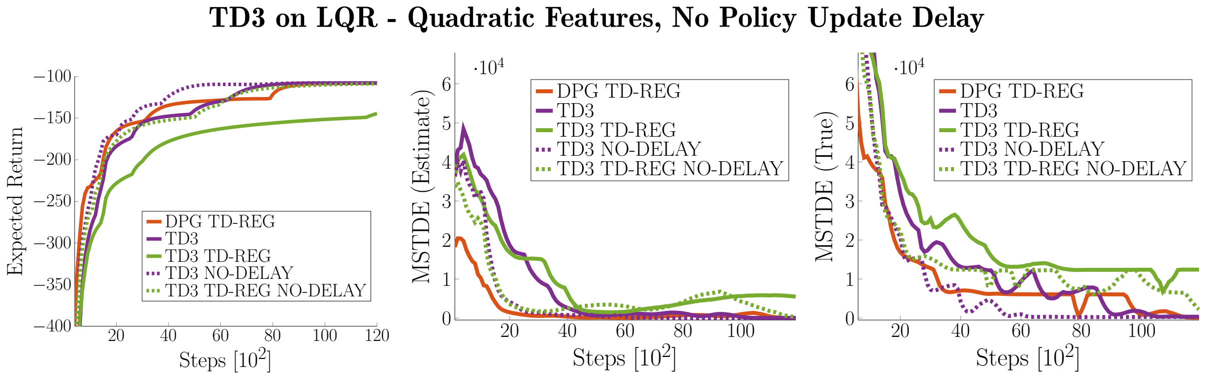


Figure D.3: Without policy update delays, both TD3 and TD3 TD-REG converge faster.

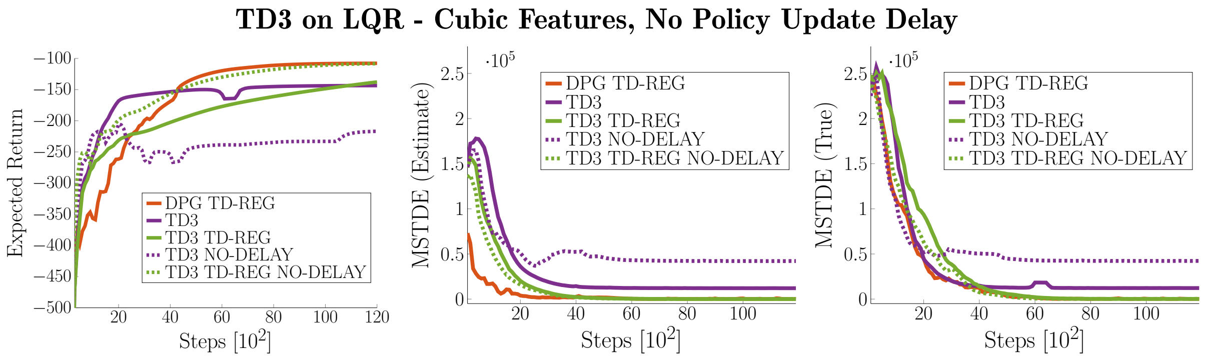
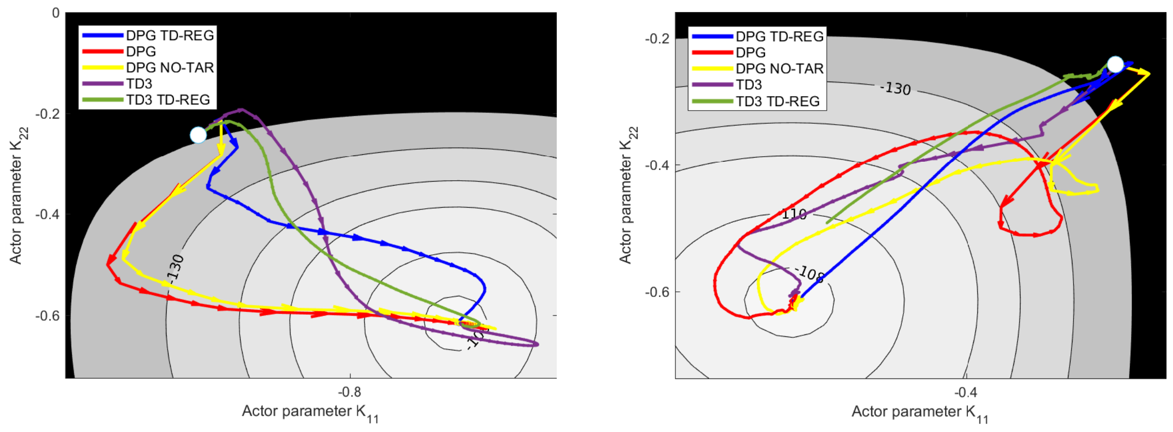


Figure D.4: With cubic features, TD3 NO-DELAY performs worse than TD3, diverging six times instead of two. By contrast, TD3 TD-REG NO-DELAY performs better than TD3 TD-REG and it always converges within the time limit.

Results are substantially different when cubic features are used (Figure D.2). In this case, many features are irrelevant and the model is prone to overfit. As a consequence, the TD error shown in Figure D.2 is much larger than the one shown in Figure D.1, meaning that it is harder for the critic to correctly approximate the true Q-function. The problem is prominent for DPG and DPG NO-TAR, which cannot learn 24 and 28 times, respectively, out of 50 (thus, the very large confidence interval). Similarly to the previous case, the true TD error is underestimated. Initially their critics estimate a TD error of 10^5 but the true one is $2.5{\cdot}10^5$. This large error guides the actors incorrectly, inevitably leading to divergence. Then, after approximately 3,000 steps, the two TD errors match at $1.5{\cdot}10^5$. Among the two algorithms, DPG NO-TAR performs worse, due to the lack of the target policy.

Path of the Learned Policy Parameters θ



(a) Quadratic features. (b) Cubic features.

Figure D.5: Paths followed by the policy parameters in runs in which no algorithm diverged. Each arrow represents 100 update steps. Contour denotes the expected return magnitude. The initial policy parameters are denoted by the white circle. The TD-regularization enables safer and more stable trajectories, leading the policy parameters more straightforwardly to the optimum. Notice that blue and green arrows are initially shorter, denoting smaller update steps. In fact, due to the initial inaccuracy of the critic, the TD-regularization gradient "conflicts" with vanilla gradient at early updates and avoids divergence. However, for TD3 TD-REG combining two critics, delayed policy updates and TD-regularization slows down the learning up to the point that the algorithm never converged within the time limit (Figure D.5b).

TD3 performs substantially better, but still diverges two times. By contrast, TD-REG algorithms never diverges. Figure D.5b shows the benefits of the TD-regularization in the policy space. Initially, when the TD error is large, the policy "moves around" the starting point. Then, as the critic accuracy increases, the policy goes almost straightforwardly to the goal. This "slow start" behavior is also depicted in Figure D.2, where DPG TD-REG expected return initially improves more slowly compared to TD3 and TD3 TD-REG. Finally, we notice once again that the combination of TD3 and TD-REG results in the slowest learning: unlike TD3, TD3 TD-REG never diverged, but it also never converged within the time limit. This behavior is also depicted in Figure D.5b, where green arrows (TD3 TD-REG policy updates) are so small that the algorithm cannot reach the goal in time.

Figure D.2 hints that the "slow learning" behavior of TD3 TD-REG may be due to the delayed policy update, as both the estimated and the true TD error are close to zero by mid-learning. To further investigate this behavior, we performed the same experiments without delayed policy updates for both TD3 and TD3-REG. For the sake of clarity, we report the results on separates plots without DPG and DPG NO-TAR and without shaded areas for confidence interval. In the case of quadratic features (Figure D.3) both TD3 and TD3 TD-REG gain from the removal of the policy update delay. However, in the case of cubic features (Figure D.4), TD3 NO-DELAY performs worse than TD3, as it diverges six times instead of two. By contrast, TD3 TD-REG NO-DELAY performs better than TD3 TD-REG, and the expected return curve shows traits of both TD3 and TD-REG: initially it improves faster, like TD3, and then it always converges to optimality, like DPG TD-REG. We can conclude that delaying policy updates is not necessary when appropriate features are used and overfitting cannot happen. Otherwise, the combination of two critics and TD-regularization yields the best results, with the TD-regularization providing the most benefits.

D.1.2 SPG Evaluation on the LQR

DPG is an on-line algorithm, i.e., it performs a critic/actor update at each time step, using mini-batches of previously collected samples. Instead, stochastic policy gradient (SPG) collects complete trajectories with a stochastic policy before updating the critic/actor. In this section, we evaluate SPG, SPG TD-REG, and REINFORCE [Williams, 1992]. The first two maximize Q-function estimates given by a learned critic. SPG TD-REG additionally applies the TD-regularization presented in the paper. REINFORCE, instead, maximizes Monte Carlo estimates of the Q-function, i.e.,

$$\widehat{Q}^{\pi}(s_t, a_t) = \sum_{i=t}^{T} \gamma^{i-t} r_i. \tag{D.6}$$

The policy is Gaussian, i.e., $\pi(a|s;\theta) = \mathcal{N}(Ks, \Sigma)$, where Σ is a diagonal covariance matrix. K is initialized as for DPG. The diagonal entries of Σ are initialized to five. Six policy parameters are learned, four for K and two for Σ.
For SPG TD-REG, the expectation $\mathbb{E}_{\pi(a'|s';\theta)}\left[\widehat{Q}(s', a'; \omega)\right]$ is approximated with the policy mean Q-value, i.e., $\widehat{Q}(s', Ks'; \omega)$.
For SPG and SPG TD-REG, $\widehat{Q}(s, a; \omega)$ is learned by Matlab `fminunc` optimizer using the samples collected during the current iteration.

Hyperparameters

- Trajectories collected per iteration: 1 or 5.
- Steps per trajectory: 150.
- Discount factor: $\gamma = 0.99$.
- Policy and TD error evaluated at every iteration.
- No separate target policy or target Q-function are used for learning the critic, but we still consider $\nabla_\omega \widehat{Q}(s', a'; \omega) = 0$.
- Policy update learning rate: 0.01. The gradient is normalized if its norm is larger than one.
- The policy update performs only one step of gradient descent on the whole dataset.
- Regularization coefficient: $\eta_0 = 0.1$ and then it decays according to $\eta_{t+1} = 0.999\eta_t$.

Results
Figures D.6 and D.7 show the results when one or five episodes, respectively, are used to collect samples during one learning iteration. REINFORCE performed poorly in the first case, due to the large variance of Monte Carlo estimates of the Q-function. In the second case, it performed better but still converged slowly. SPG performed well except for two runs in Figure D.7, in which it diverged already from the beginning. In both cases, its performance is coupled with the approximator used, with quadratic features yielding more stability. By contrast, SPG TD-REG never failed, regardless of the number of samples and of the function approximator used, and

despite the wrong estimate of the true TD error. Similarly to what happened in DPG, in fact, the critic always underestimates the true TD error (Figures D.6b and D.7b).

Finally, Figure D.8 shows the direction and the magnitude of the gradients. We clearly see that initially the vanilla gradient (which maximizes Q-function estimates, red arrows) points towards the wrong direction, but thanks to the gradient of the TD-regularization (blue arrows) the algorithm does not diverge. As the learning continues, red arrows point towards the optimum and the magnitude of blue arrows decreases, because 1) the critic becomes more accurate and the TD error decreases, and 2) the regularization coefficient η decays. The same behavior was already seen in Figure D.5 for DPG, with the penalty gradient dominating the vanilla gradient at early stages of the learning.

SPG on LQR - One Episode per Iteration

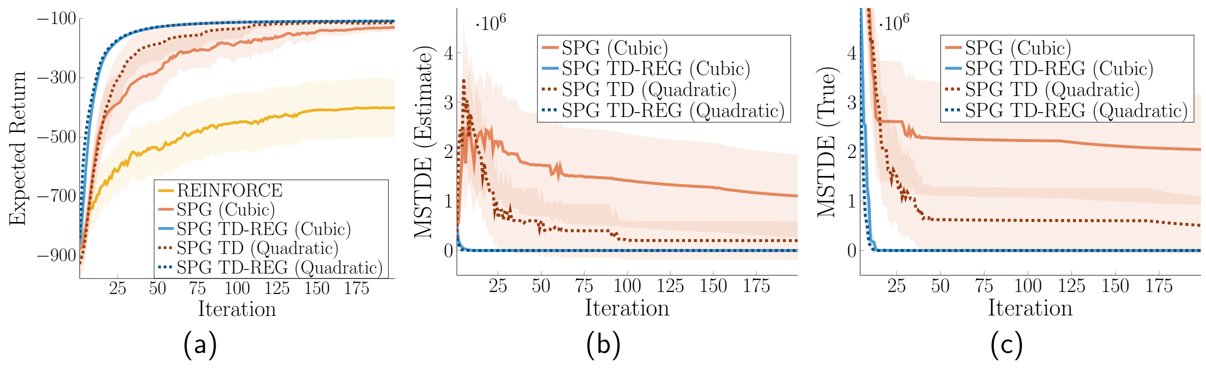


Figure D.6: Results averaged over 50 runs, shaded areas denote 95% confidence interval. Only one episode is collected to update the critic and the policy. SPG TD-REG did not suffer from the lack of samples, and it always learned the optimal critic and policy within few iterations, both with quadratic and cubic features (the two blue plots almost overlap). By contrast, vanilla SPG is much more sensitive to the number of samples. Even if it never diverged, its convergence is clearly slower. REINFORCE, instead, diverged 13 times, due to the high variance of Monte Carlo estimates.

SPG on LQR - Five Episodes per Iteration

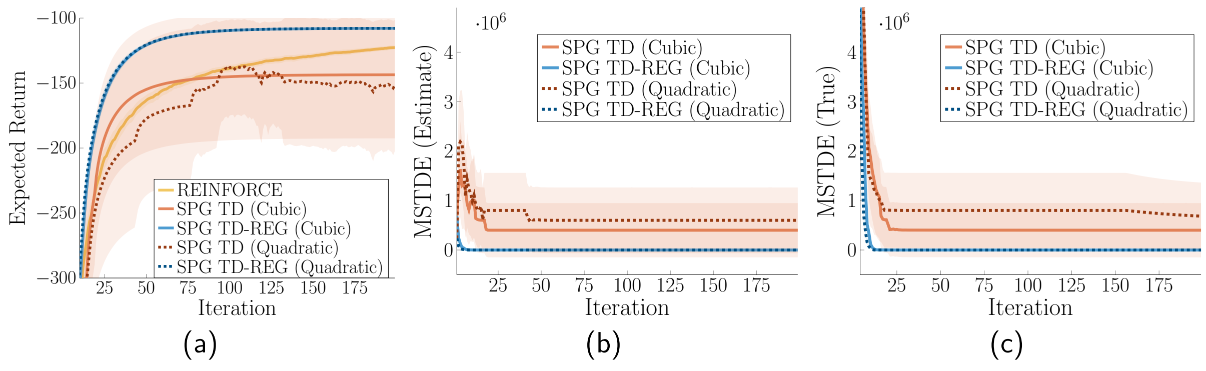


Figure D.7: With more samples per iteration REINFORCE was able to converge, as Monte Carlo estimates have lower variance. However, it still converged more slowly than SPG TD-REG. Surprisingly, vanilla SPG diverged two times, thus explaining its slightly larger confidence interval (in Figure D.7a the y-axis starts at -300 for the sake of clarity). In the other 48 runs, it almost matched SPG TD-REG.

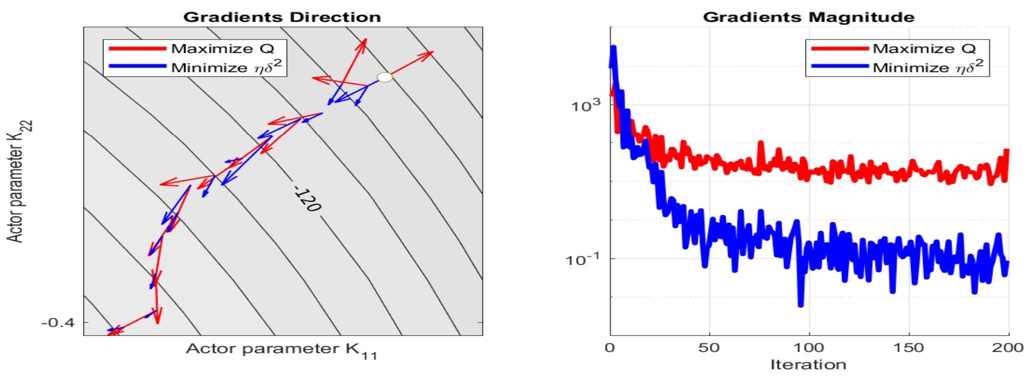


Figure D.8: On the left, beginning of the path followed by SPG TD-REG in the policy parameter space during one trial, when samples from only one episode are collected per iteration. The initial policy parameter vector θ is denoted by the white circle. Each arrow represents one iteration. Red and blue arrows denote the direction of the gradient $\nabla_\theta \mathbb{E}[\widehat{Q}(s,a;\omega)]$ and $\nabla_\theta \mathbb{E}[\eta\delta_Q(s,a,s';\theta,\omega)^2]$, respectively. Contour denotes the expected return magnitude. The magnitude of the arrows has been scaled for better visualization, and the true one is shown on the right. Initially, as the critic estimate is highly inaccurate (see the TD error in Figures D.6b and D.6c), red arrows point to the wrong direction. However, blue arrows help keeping the policy on track.

D.2 Pendulum Swing-up Tasks

Single-pendulum

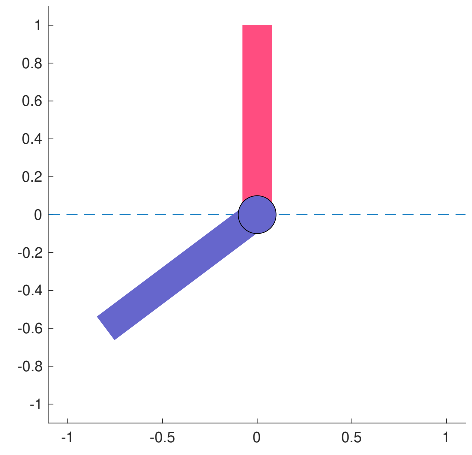
This task (Figure D.9a) follows the equations presented in OpenAI Gym [Brockman et al., 2016]. The state consists of the angle position q and velocity $\dot{q}$ of the link. The former is bounded in $[-\pi, \pi]$ and is initialized uniformly in $[-\pi, \pi]$. The latter is bounded in $[-8, 8]$ and is initialized in $[-1, 1]$. As the pendulum is underactuated, the action is bounded in $[-2, 2]$. The transition function is

$$\begin{aligned}\dot{q}_{t+1} &= \dot{q}_t - \frac{3g}{2l}\sin(q+\pi) + \frac{3}{ml^2}a\delta_t,\\ q_{t+1} &= q_t + \dot{q}_{t+1}\delta_t,\end{aligned}$$

where $g = 10$ is the gravitational acceleration, $m = 1$ is the link mass, $l = 1$ is the link length, and $\delta_t = 0.05$ is the timestep. The goal state is $q = 0$ and the reward function is

$$r_t = -q_t^2 - 0.1\dot{q}_t^2 - 0.001a_t^2.$$

For learning, at each iteration only 10 episodes of 50 steps are collected. For the expected return evaluation, the policy is tested over 1,000 episodes of 150 steps.
The critic approximates the V-function with a linear function $\widehat{V}(s;\omega) = \phi(s)^\mathsf{T}\omega$ where $\phi(s)$ are 100 random Fourier features, as presented in [Rajeswaran et al., 2017]. The bandwidths are computed as suggested by [Rajeswaran et al., 2017] as the average pairwise distances between 10,000 state observation vectors. These states are collected only once before the learning and are shared across all 50 trials. The phase and the offset of the Fourier features are instead random and different for all trials.
The policy is Gaussian, i.e., $\pi(a|s;\theta) = \mathcal{N}(b + K\phi(s), \sigma^2)$. The same Fourier features of the


(a) Single-pendulum.

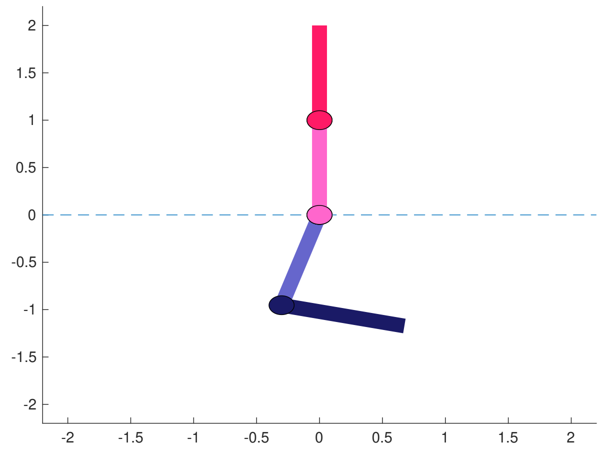

(b) Double-pendulum.

Figure D.9: The two pendulum swing-up tasks, with an example of the current state (in blue) and the goal state (in red).

critic are used for the policy, for a total of 102 policy parameters to learn. K and b are initialized to zero, while σ to four. For the expected return evaluation, the noise of the policy is zeroed. The critic parameters are initialized uniformly in $[-1, 1]$ and are learned with Matlab `fminunc` optimizer such that they minimize the mean squared TD(λ) error (since we use GAE). GAE hyperparameters are $\gamma = 0.99$ and $\lambda = 0.95$.
The policy is learned with TRPO with a KL bound of 0.01. For computing the natural gradient, both the advantage, the TD error and the regularization are standardized, i.e., $\hat{y} \leftarrow (\hat{y} - \mu_{\text{Y}})/\sigma_{\text{Y}}$, where $\hat{y}$ is either the advantage estimator, the TD error estimator, or the regularization (δ^2 or $\widehat{A}^2$, depending if we use TD-REG or GAE-REG), μ_{Y} is the mean of the estimator and σ_{Y} its standard deviation. The conjugate gradient is computed with Matlab `pcg`. Additionally, since TRPO uses a quadratic approximation of the KL divergence, backtracking line search is performed to ensure that the KL bound is satisfied. The starting regularization coefficient is $\eta_0 = 1$ and then decays according to $\eta_{t+1} = \kappa\eta_t$ with $\kappa = 0.999$.

Double-pendulum
This task (Figure D.9b) follows the equations presented by Yoshikawa [1990]. The state consists of the angle position $[q_1, q_2]$ and velocity $[\dot{q}_1, \dot{q}_2]$ of the links. Both angles are bounded in $[-\pi, \pi]$ and initialized uniformly in $[-\pi, \pi]$, while both velocities are bounded in $[-50, 50]$ and initialized in $[-1, 1]$. The agent, however, observes the six-dimensional vector $[\sin(q), \cos(q), \dot{q}]$. As the pendulum is underactuated, the action on each link is bounded in $[-10, 10]$. The transition function is

$$\begin{aligned}
\ddot{q}_{t+1} &= M_t^{-1}(a_t - f_{gt} - f_{ct} - f_{vt}), \\
\dot{q}_{t+1} &= \dot{q}_t - \ddot{q}_{t+1}\delta_t, \\
q_{t+1} &= q_t + \dot{q}_{t+1}\delta_t,
\end{aligned}$$

where $\delta_t = 0.02$ is the timestep, M is the inertia matrix, f_g is gravitational, f_c the Coriolis force, and f_v the frictional force. The entries of M are

$$
\begin{aligned}
M_{11} &= m_1 \left(\frac{l_1}{2}\right)^2 + I_1 + m_2 \left(l_1^2 + \left(\frac{l_2}{2}\right)^2 + 2l_1 \left(\frac{l_2}{2}\right)^2 \cos(q_2)\right) + I_2, \\
M_{12} = M_{21} &= m_2 \left(\left(\frac{l_2}{2}\right)^2 + l_1 \frac{l_2}{2} \cos(q_2)\right) + I_2, \\
M_{22} &= m_2 \left(\frac{l_2}{2}\right)^2 + I_2,
\end{aligned}
$$

where $l_i = 1$ is the length of a link, $m_i = 1$ is the mass of the a link, and $I_i = (1 + 0.0001)/3.0$ is the moment of inertia of a link. Gravitational forces are

$$
\begin{aligned}
f_{g1} &= m_1 g \frac{l_1}{2} \cos(q_1) + m_2 g \left(l_1 \cos(q_1) + \frac{l_2}{2} \cos(q_1 + q_2)\right), \\
f_{g2} &= m_2 g \frac{l_2}{2} \cos(q_1 + q_2),
\end{aligned}
$$

where $g = 9.81$ is the gravitational acceleration. Coriolis forces are

$$
\begin{aligned}
f_{c1} &= -m_2 l_1 \frac{l_2}{2} \sin(q_2)(2\dot{q}_1\dot{q}_2 + \dot{q}_2^2), \\
f_{c2} &= m_2 l_1 \frac{l_2}{2} \sin(q_2)\dot{q}_1^2.
\end{aligned}
$$

Frictional forces are

$$
\begin{aligned}
f_{v1} &= v_1 \dot{q}_1, \\
f_{v2} &= v_1 \dot{q}_2,
\end{aligned}
$$

where $v_i = 2.5$ is the viscous friction coefficient. The goal state is $q = [\pi/2, 0]$ and the reward function is

$$
r_t = -||\pi - \texttt{abs}(\texttt{abs}(q - q_{\text{GOAL}}) - \pi)||_2^2 - 0.001||a||_2^2,
$$

where the first term is the squared distance between the current angles and the goal position $q_{\text{GOAL}} = [\pi/2, 0]$, wrapped in $[-\pi, \pi]$. Note that, compared to the single-pendulum, the frame of reference is rotated of 90°. The first angle q_1 is the angle between the base of the pendulum and the first link. The second angle q_2 is the angle between the two links.
For learning, at each iteration only 6 episodes of 500 steps are collected, because the double-pendulum needs more steps to swing up than the single-pendulum. For the expected return evaluation, the policy is tested over 1,000 episodes of 500 steps. The same actor-critic setup of the single-pendulum is used, except for

- The Fourier features, which are 300,
- The policy, which has a full covariance matrix. Its diagonal entries are initialized to 200 and we learn its Cholesky decomposition, for a total of 605 parameters to learn, and

- The initial regularization coefficient for TD-REG, which is $\eta_0 = 0.1$.

D.3 MuJoCo Continuous Control Tasks

The continuous control experiments are performed using OpenAI Gym [Brockman et al., 2016] with MuJoCo physics simulator. All environments are version 2 (v2). We use the same hyper-parameters and neural network architecture for all tasks. The policy is a Gaussian distribution with a state-independent diagonal covariance matrix. Each algorithm is evaluated over five trials using different random seeds with common seeds for both methods. For TRPO, the policy was evaluated over 20 episodes without exploration noise. For PPO, we used the same samples collected during learning, i.e., including exploration noise.
The actor mean and the critic networks are two-layer neural networks with hyperbolic tangent activation function. Because all tasks actions are bounded in $[-1, 1]$, the actor network output has an additional hyperbolic tangent activation.
At each iteration, we collect trajectories until there are at least 3,000 transition samples in the training batch. The maximum trajectory length is 1,000 steps. Then, we compute the GAE estimator as in Eq. (4.45) and train the critic to minimize the mean squared TD(λ) error (Eq. (4.41)) using ADAM [Kingma and Ba, 2014][1]. The actor is then trained using the same training batch, following the algorithm policy update (PPO, TRPO, and their respective regularized versions). For the policy update, both the advantage, the TD error and the regularization are standardized, i.e., $\hat{y} \leftarrow (\hat{y} - \mu_{\mathrm{Y}})/\sigma_{\mathrm{Y}}$, where $\hat{y}$ is either the advantage estimator, the TD error estimator, or the regularization (δ^2 or $\widehat{A}^2$, depending if we use TD-REG or GAE-REG), μ_{Y} is the mean of the estimator and σ_{Y} its standard deviation.

TRPO Hyperparameters

- Hidden units per layer 128.
- Policy initial standard deviation 1.
- GAE hyperparameters: $\lambda = 0.97$ and $\gamma = 0.995$
- ADAM hyperparameters: 20 epochs, learning rate 0.0003, mini-batch size 128.
- Policy KL divergence bound $\epsilon = 0.01$.
- We also add a damping factor 0.1 to the diagonal entries of the Fisher information matrix for numerical stability
- The maximum conjugate gradient step is set to 10.

PPO Hyperparameters

- Hidden units per layer 64.
- Policy initial standard deviation 2.
- GAE hyperparameters: $\lambda = 0.95$ and $\gamma = 0.99$
- ADAM hyperparameters (for both the actor and the critic): 20 epochs, learning rate 0.0001, mini-batch size 64.
- Policy clipping value $\varepsilon = 0.05$.

[1] TRPO original paper proposes a more sophisticated method to solve the regression problem. However, we empirically observed that batch gradient descent is sufficient for good performance.