POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA
DOCTORAL PROGRAMME IN INFORMATION TECHNOLOGY

EXPLOITING HYPERPARAMETER OPTIMIZATION AND CONTROL FREQUENCY IN REINFORCEMENT LEARNING

Doctoral Dissertation of:
Luca Sabbioni

Supervisor:
Prof. Marcello Restelli

Tutor:
Prof. Nicola Gatti

The Chair of the Doctoral Program:
Prof. Luigi Piroddi

2023 – Cycle XXXV

Abstract

REINFORCEMENT Learning (RL) has driven impressive advances in artificial intelligence in recent years for a wide range of domains, from robotic control to financial trading. However, the performance of current RL methods is strongly dependent on the hyperparameters of the algorithms, which practitioners usually need to tune carefully, and on the environment design, where the control frequency plays a dominant role. The consequent engineering procedures are prone to errors and are time-consuming, especially if they are started from scratch for each task modification. The subject of this dissertation is the development of automatic techniques to enhance the learning capabilities of RL algorithms in a twofold direction.

In the first part, we address the Hyperparameter Optimization (HO) problem, with a particular focus on policy-based techniques for RL: indeed, they rely on strong theoretical guarantees that play a very important role but do not help in the selection of the hyperparameters. To enhance the learning capabilities of this class of algorithms, we frame HO as a Sequential Decision Process and design a solution that allows selecting a dynamic sequence of hyperparameters adaptive to the policy and the context of the environment. Hence, the reward function of the learning process is performance gain, and the action consists in the hyperparameter selection. With this problem definition, it is possible to adopt RL algorithms on a more abstract level to optimize the progress of the whole learning instance.

The second part is devoted to improving RL agents by leveraging the frequency of the agent-environment interaction, which has a deep impact on the control opportunities and the sample complexity of the learning algorithms. We introduce and discuss the concept of action persistence or action repetition: leveraging theoretical results and bounds on the performance loss incurred while employing persistence, we provide algorithmic contributions to detect the most promising frequency. As a conclusive contribution, we employ a new operator that allows for effective use of the experience collected at any time scale to learn a dynamic adaption of the persistence or, in other terms, the best duration of each action.

All contributions are empirically validated through experimental assessments on challenging benchmarks.

Sommario

NEGLI ultimi anni, l'apprendimento tramite rinforzo (Reinforcement Learning, RL) ha ottenuto progressi impressionanti in una vasta gamma di domini, dal controllo robotico al trading finanziario. Tuttavia, le prestazioni dei metodi attuali sono fortemente dipendenti dai loro iperparametri, da regolare con cura, e dalla configurazione dell'ambiente, in cui la frequenza di controllo svolge un ruolo dominante. Le conseguenti procedure di progettazione sono soggette a errori e richiedono molto tempo, soprattutto se avviate da zero per ogni modifica del sistema. Oggetto di questa dissertazione è lo sviluppo di tecniche automatiche per migliorare le capacità di apprendimento degli algoritmi RL in due direzioni.

Nella prima parte, affrontiamo il problema dell'ottimizzazione dei iperparametri, con particolare attenzione alle tecniche di RL tramite gradiente: queste si basano su importanti garanzie teoriche che non aiutano nella selezione dei iperparametri. Per migliorare le capacità di apprendimento di questa classe di algoritmi, definiamo il problema come un processo decisionale sequenziale e progettiamo una soluzione che consente di attuare una sequenza dinamica di iperparametri adattiva alla politica e al contesto dell'ambiente. Di conseguenza, il guadagno dell'istanza di apprendimento è dato dal miglioramento delle performances e l'azione consiste nella scelta degli iperparametri. Con questa definizione, si possono adottare gli algoritmi di RL su un livello più astratto per ottimizzare il progresso dell'intera istanza di apprendimento.

La seconda parte è dedicata al miglioramento degli agenti RL sfruttando la frequenza di interazione agente-ambiente, che ha un impatto profondo sulle potenzialità e sulla complessità degli algoritmi di apprendimento. Introduciamo e discutiamo il concetto di persistenza o ripetizione dell'azione: sfruttando i risultati e i limiti teorici sulle perdite di prestazione durante l'utilizzo della persistenza, forniamo contributi algoritmici per rilevare la frequenza di controllo più promettente. Come contributo conclusivo, utilizziamo un nuovo operatore che consente un efficace utilizzo dell'esperienza raccolta in qualsiasi scala temporale per imparare un adattamento dinamico della persistenza o, in altri termini, della durata migliore di ogni azione.

Tutti i contributi vengono validati empiricamente attraverso valutazioni sperimentali su benchmark complessi.

Contents

List of Tables **X**

List of Figures **XI**

List of Algorithms **XIII**

Glossary **XV**

List of Symbols **XX**

1 Introduction **1**

1.1 Reinforcement Learning: Learning to Act 2
1.2 Motivations and Research Problems 2
1.3 Original Contribution 3
1.4 Structure and Content Outline 5

2 Foundations of Reinforcement Learning **7**

2.1 Introduction 7
2.2 Markov Decision Processes 8
2.2.1 Interaction 9
2.2.2 MDP Extensions 10
2.2.3 Trajectories and Return 11
2.2.4 Policies 13
2.3 Expected Return and Value Functions 15
2.3.1 Bellman Equations 16
2.3.2 Optimality Criteria 17
2.4 Dynamic Programming 19
2.4.1 Value Iteration 20
2.4.2 Policy Iteration 20
2.4.3 Linear Programming Method 21

3 Reinforcement Learning **23**

3.1 Introduction 23
3.2 A Taxonomy for Reinforcement Learning Algorithms 24
3.3 Policy Evaluation 26
3.3.1 Monte Carlo Prediction 27
3.3.2 Temporal Difference Learning 28
3.4 Value-Based Control 30
3.4.1 SARSA 31
3.4.2 Q-learning 32
3.4.3 Approximate Value Iteration 33
3.5 Policy Search 37
3.5.1 Trust-Region Update 43
3.6 Lipschitz MDP 46

I Hyperparameter Optimization through Meta Reinforcement Learning 51

4 Hyperparameter Optimization as a Sequential Decision Problem: Meta Learning the Step Size 55
4.1 Introduction 55
4.2 Motivations 56
4.3 Related Work 59
4.4 Meta-MDP 60
4.5 Context Lipschitz Continuity 64
4.6 Fitted Q-Iteration on Meta-MDP 66
4.7 Experimental Evaluation 68
4.8 Conclusions 71

5 Trust Region Meta Learning for Policy Optimization 73
5.1 Introduction 73
5.2 Motivations 74
5.2.1 Trust Region Policy Optimization. 76
5.2.2 Information Theory 77
5.3 Methodology 79
5.3.1 Optimizing the KL Constraint through Meta-MDPs 79
5.4 Meta-Features Estimation 82
5.4.1 Density and Volumes Estimation 83
5.4.2 KNN-Estimator 84
5.5 Experiments 85
5.6 Conclusions 88

II Dynamic Step and Control Frequency 89

6 Control Frequency Adaptation via Action Persistence 93
6.1 Introduction 93
6.2 Motivations 95
6.3 Related Work 96

6.4 Persisting Actions in MDPs 98
6.4.1 Duality of Action Persistence 98
6.4.2 Persistent Bellman Operators 100
6.5 Bounding the Performance Loss 103
6.6 Persistent Fitted Q-Iteration 106
6.6.1 Theoretical Analysis 107
6.7 Persistence Selection 109
6.8 Experimental Evaluation 110
6.9 Application to Foreign Exchange Trading 112
6.9.1 MDP Model for FX Trading 112
6.10 Conclusions 114

7 All-persistence Bellman Update 117
7.1 Introduction 117
7.2 Motivations 118
7.3 Related Work 119
7.4 All-Persistence Bellman Update 120
7.4.1 Persistence Options 120
7.4.2 All-Persistence Bellman Operator 121
7.5 Persistent Q-learning 125
7.6 Empirical Advantages of Persistence 126
7.7 Persistent Deep Networks 128
7.8 Experimental Evaluation 129
7.9 Conclusions 132

8 Discussion and Conclusions 133
8.1 Hyperparameter Optimization through Meta RL 133
8.1.1 Limitations and Future Works 135
8.2 Dynamic Step and Control Frequency 135
8.2.1 Limitations and Future Works 136
8.3 Final Remarks 137

Appendices 141

A Additional Results of Chapter 4 141
A.1 Proofs 141
A.1.1 Lipschitz Continuity of the Action-Value Function 141
A.1.2 Lipschitz Continuity of the Gradient 143
A.2 Experiment Details 146
A.2.1 Navigation2D Description 146
A.2.2 Minigolf Description 147
A.2.3 CartPole Description 148
A.2.4 Half Cheetah Description 149
A.2.5 Metagrad Implementation 150
A.3 Other Results 151
A.3.1 Meta Cartpole SwingUp 151
A.3.2 Comparison among MetaFQI Iterations. 151

A.3.3 Comparison with Learning Rate schedules: Details 152
A.3.4 Extension of Trajectory Length 153
A.3.5 Experiments with Fixed Contexts 155
A.3.6 Selection of a Single Learning Rate 156

B Additional Results of Chapter 6 **159**
B.1 Proofs 159
B.1.1 Proofs of Section 6.5 159
B.1.2 Proofs of Section 6.6 169
B.1.3 Proofs of Section 6.7 176
B.2 Details on Bounding the Performance Loss (section 6.5) 177
B.2.1 On Using Divergences Other than the Kantorovich 177
B.2.2 Time–Lipschitz Continuity for Dynamical Systems 178
B.3 Details on Experimental Evaluation 179
B.3.1 Experimental Setting 179
B.3.2 Additional Plots 180
B.4 Preliminary Results on Open Questions 180
B.4.1 Improving Exploration with Persistence 180

C Additional Results of Chapter 7 **187**
C.1 Discussion on the Advantages of Dynamic Persistence 187
C.1.1 Details on Persistent Markov Chains and Kemeny Constant . . . 187
C.1.2 Further Exploration Advantages: MountainCar 190
C.1.3 Synchronous Update: Additional Results 192
C.2 Details on Experimental Evaluation 193
C.2.1 Tabular Environments 193
C.2.2 MountainCar 194
C.2.3 Atari Games 194
C.3 Additional Results 195
C.3.1 Tabular Environments 195
C.3.2 PerDQN Additional Results 197
C.3.3 PerQ-learning: computational times 198
C.3.4 Experiments with the same Number of Updates and Comparison with ez-greedy Exploration 199

D Persistor Policy Gradient **203**
D.1 Occupancy Measures 204
D.2 Persistor Gradient Theorem 206
D.3 Persistor Classes 208
D.3.1 Poisson Distribution 208
D.4 Performance Difference 209
D.5 Useful Lemmas 209

List of Figures

2.1 Graphical representation of the agent-environment interaction in a Markov Decision Process. Source: Sutton and Barto (1998) 8

3.1 Illustration of Approximate Value Iteration (AVI, Gordon (1995); Munos (2005)) 34

4.1 Suboptimality of the bandit setting: a counterexample. 63
4.2 metaFQI model performance against NGA with fixed step size 69
4.3 metaFQI model performance comparison against benchmarks 70

5.1 Fine-tuned TRPO vs metaFQI with linear policies 85
5.2 Meta Minigolf: Performances by different Policy Parametrizations . . 86
5.3 Meta Cartpole: Performances across different Policy Parametrizations 87

6.1 Agent-environment interaction with and without action persistence . . 98
6.2 MDP counterexample for persistence performance loss 104
6.3 Extended results for Persistence selection in Cartpole environment. . . 111
6.4 Cumulative return in test for each of the different currency pairs combinations and different persistence values 113
6.5 Portfolio allocation chosen by the three-currency agent 114

7.1 An example of the update order for Algorithm 9. 126
7.2 Normalized Kemeny's constant in tabular environments as function of $K_{\max}$. Bullets represent the minimum. 127
7.3 L_∞ error on 6x6 grid-world between synchronous Q-learning and Per$Q$-learning- 128
7.4 MountainCar PerDQN results. 130
7.5 Results on tabular environments. 130
7.6 PerDQN Atari games results. 131

A.1 Meta Swingup metaFQI model performance 152

A.2 metaFQI model performance among different iterations. For the sake of clarity, only the average values are shown. 152
A.3 metaFQI model performance against an exponentially decreasing (*decaying*) learning rate. 153
A.4 Cartpole metaFQI model performance with extended horizon 154
A.5 HalfCheetah metaFQI model performance with extended horizon . . . 154
A.6 metaFQI model performance obtained by considering or excluding the explicit task parametrization $\boldsymbol{\omega}$. 155
A.7 metaFQI model performance w.r.t. random states. Double Q-learning vs Single Q-learning comparison 156
A.8 metaFQI model performance and tuned NGA on environments with fixed context. 156
A.9 Comparison of performances obtained by a trained agent capable of choosing a fixed initial learning rate. 157

B.1 PFQI results on control problems 183
B.1 PFQI results on robotic locomotion problems 184
B.2 Illustration of (a) PFQI(k) executed in the base MDP $\mathcal{M}$ and (b) the standard FQI executed in the k-persistent MDP $\mathcal{M}_k$. 185
B.3 Performance of the policies learned with FQI on $\mathcal{M}_k$, PFQI(k) on $\mathcal{M}$. 185
B.4 Performance of the policies π_k for $k \in \mathcal{K}$ comparing when they are executed in $\mathcal{M}_k$ and when they are executed in $\mathcal{M}_{(k')^\star}$. 185

C.1 Normalized Kemeny's constant and entropy in Tabular environments as a function of the maximum persistence and horizon H. 189
C.2 Visitation frequencies in Mountain Car environment of different random policies with fixed K persistences. 190
C.3 Visitation frequencies in Mountain Car environment of different random persistence options. 191
C.4 Grid environment representation 192
C.5 k-value function representation for different persistence values k in Gridworld environment 193
C.6 Tabular Gridworld representation 194
C.7 Results of Q-learning, TempoRL, Per$Q$-learning and MSA-Q-learning in different tabular environments and maximum persistences. 196
C.8 PerDQN additional results on MountainCar. 197
C.9 PerDQN additional results on Freeway with maximum persistence 4 and 8 197
C.10 Atari games results for DQN and PerDQN, with $K_{\max} = 4$ and 8 . . . 198
C.11 Computational times required for a complete run of Per$Q$-learning on tabular environments with different Maximum Persistences. 198
C.12 Results of Q-learning, TempoRL, Per$Q$-learning and MSA-Q-learning in different tabular environments and maximum persistences. 199
C.13 Freeway results: PerDQN comparison with DQN with the same global amount of tuples sampled per update as PerDQN. 200
C.14 MountainCar results for DQN, PerDQN and ϵz-greedy DQN 201

List of Tables

5.1 Meta-MDP features list 81

6.1 Results of PFQI in different environments and persistences 110

A.1 Best initial learning rate selected. Evaluation using 20 different random tasks and policies. 153

B.1 Parameters of the experimental setting, used for the PFQI(k) experiments. 180

B.2 Results of PFQI execution of the policy π_k learned with the k–persistent operator in the k'–persistent MDP $\mathcal{M}_{k'}$ in the Cartpole experiment. . . 182

List of Algorithms

1 Q-learning 33
2 Fitted Q-Iteration 35
3 Actor-only Policy Gradient 40
4 Trust Region Policy Optimization (TRPO) 46
5 Meta-MDP Dataset Generation 66
6 Meta-features Dataset Generation 82
7 Persistent Fitted Q-Iteration PFQI(k). 106
8 Heuristic Persistence Selection. 109
9 All Persistence Bellman Update 126
10 Persistent Q-learning (Per$Q$-learning) 127
11 Multiple Replay Buffer Storing 129

Glossary

AI	Artificial Intelligence.
API	Approximate Policy Iteration.
AVI	Approximate Value Iteration.
BE	Bellman Error.
CMDP	Contextual Markov Decision Process.
Conf-MDP	Configurable Markov Decision Process.
Cont-MDP	Continuous-time MDP.
DQN	Deep Q-Network.
DRL	Deep Reinforcement Learning.
FIM	Fisher Information Matrix.
FQI	Fitted Q-Iteration.
FX	Foreign Exchange.
GA	Genetic Algorithm.
HO	Hyperparameter Optimization.
KL	Kullback-Leibler divergence.
KNN	K-Nearest Neighbors.
LC	Lipschitz continuous.
LS	Line Search.
MC	Markov Chain.
MDP	Markov Decision Process.
ML	meta learning.

MLP	multi-layer perceptron.
MSA	Multi-Step Action.
MSE	Mean Squared Error.
NGA	Natural Gradient Ascent.
NPG	Natural Policy Gradient.
PCA	Principal Component Analysis.
PFQI	Persistent Fitted Q-Iteration.
PG	Policy Gradient.
PGT	Policy Gradient Theorem.
PI	Policy Iteration.
PLC	Pointwise Lipschitz continuous.
POMDP	Partially Observable Markov Decision Process.
PPO	Proximal Policy Optimization.
RARL	Risk-Averse Reinforcement Learning.
RL	Reinforcement Learning.
RMSE	Root Mean Squared Error.
SDM	Sequential Decision Making.
Semi-MDP	Semi-Markov Decision Processes.
SGA	Stochastic Gradient Ascent.
SGD	Stochastic Gradient Descent.
TD	Temporal Difference Learning.
TDE	Temporal Difference Error.
TR	Trust Region.
TRPO	Trust Region Policy Optimization.
VI	Value Iteration.

List of Symbols

Sets, Arithmetic and Norms

$\mathbb{N}^+$	Set of positive natural numbers
$[T]$	$\{0, 1, 2, \ldots, T\}$
Δ_Ω	Set of probability measure over the measurable space Ω
$\mathcal{B}(\mathcal{X})$	set of bounded measurable functions over $\mathcal{X}$
$\lvert\mathcal{X}\rvert$	Cardinality of set $\mathcal{X}$
$d_\mathcal{X}$	Distance function in space $\mathcal{X}$
$H(p)$	Entropy of distribution p
$C(p)$	Cross-entropy between distributions p and q
$D_{KL}(p\|\|q)$	KL divergence between distributions p and q
$\mathcal{W}_1(p, q)$	Kantorovich distance between distributions p and q
$\nabla_{\boldsymbol{\theta}}$	Gradient w.r.t. $\boldsymbol{\theta}$
$\nabla_{\boldsymbol{\theta}_i}$	i-th component of $\nabla_{\boldsymbol{\theta}}$
$\\|f\\|_{p,\rho}$	$L_p(\rho)$-norm under probability measure ρ, and $p \in [1, \infty)$, i.e. $\\|f\\|_{p,\rho}^p := \int_\mathcal{X} \rho(dx)\lvert f(x)\rvert^p$
$\\|f\\|_{p,\mathcal{D}}$	Empirical L_p-norm on dataset $\mathcal{D} = \{\S_i\}_i$ and $p \in [1, \infty)$, i.e. $\\|f\\|_{p,\mathcal{D}}^p := \frac{1}{\lvert\mathcal{D}\rvert} \sum_i \lvert f(x_i)\rvert^p$
$\\|f\\|_\infty$	L_∞-norm, i.e. $\\|f\\|_\infty := \sup_{x\in\mathcal{X}} \lvert f(x)\rvert$
$\\|f\\|_L$	Lipschitz semi-norm of function f

Markov Decision Processes

$\mathcal{S}$	State space
$\mathcal{A}$	Action space
P	Transition kernel
R	Reward Model
r	Reward function
γ	Discount factor
μ	Starting state distribution
$\mathcal{M}$	Markov Decision Process
τ	Trajectory
$\mathcal{T}$	Trajectory space
G	Return function
Π^H	Set of history-dependent policies
Π	Set of Markovian stationary policies
ρ_π	trajectory probability distribution under π
p_π	State-transition kernel under π
δ_π^μ	State occupancy distribution
ν_π^μ	State-action occupancy distribution
J^π	Expected return of policy π
V^π	Value function of policy π
Q^π	Action-value function of policy π
A^π	Advantage function of policy π
$V^\star$	Optimal value function
$Q^\star$	Optimal action-value function
$\pi^\star$	Optimal policy
T^π	Bellman Expectation Operator
$T^\star$	Bellman Optimality Operator
δ_t	Temporal difference error on transition t
$\boldsymbol{\theta}$	vector of policy parameters
Θ	Parameter space

Part I

Ω	Context Space
ψ	Task distribution
ρ	Starting policy distribution
$J_{\boldsymbol{\omega}}$	Expected return function in task $\boldsymbol{\omega}$
$\mathfrak{M}$	Meta-MDP
$\widetilde{\gamma}$	Meta-discount factor
$\mathcal{G}$	Meta-return
$\Delta_k^{\mathcal{D}}$	k-NN distance function w.r.t. dataset $\mathcal{D}$
$\widehat{V}_k^{\mathcal{D}}$	k-NN hypersphere volume function w.r.t. dataset $\mathcal{D}$
$\widehat{p}_k^{\mathcal{D}}$	k-NN density estimation w.r.t. dataset $\mathcal{D}$

Part II

f_0	Base control frequency
Δt_0	Base timestep
K_{max}	maximum persistence
$\mathcal{K}$	Persistence space
$\mathcal{M}_k$	k–persistent MDP
$\pi_{k,t}$	k–persistent policy
p_δ	Persistent transition probability kernel
P_k	k–persistent transition model
R_k	k–persistent reward model
T_k^π	k–persistent Bellman expectation operator
$T_k^\star$	k–persistent Bellman optimality operator
$\mathcal{O}^{(k)}$	k–th persistent option space
$\mathcal{O}$	Option space
Ψ	Space of policies over persistence options
$T^{\overline{\kappa}}$	Bellman bootstrap operator
$\mathcal{H}^{\overline{\kappa}}$	All-persistence Bellman operator

CHAPTER *1*

Introduction

Artificial Intelligence (AI) has come a long way since its inception in the 1950s. In the early days of AI, researchers focused on creating programs that could perform specific tasks, such as playing chess or solving mathematical problems. Advances in this field experienced a drastic acceleration in the 1990s, partly thanks to the availability of more powerful computers and the development of new algorithms and techniques. Researchers began to progress significantly on tasks such as image and speech recognition, natural language processing, and decision-making. In the 21st century, AI has made tremendous progress and is now being used in a large variety of fields, including healthcare, finance, and education. A significant impact on the field has been provided by Machine Learning, which opened up many new possibilities for solving complex problems. The main focus of Machine Learning is the development of algorithms and models that can learn from and make predictions on data or, using the words in Dietterich 1990, from one of the pioneers in the field:

> *"Machine learning is the study of methods for constructing and improving software systems by analyzing examples of their desired behavior rather than directly programming them."*
>
> Thomas Dietterich, Machine Learning, Annual review of computer science, 1990.

From this definition of machine learning, we understand the paramount importance of data, from which models can learn and improve their performance over time. Another important definition that represents a motivation for this thesis is from Mitchell 1997:

> *"Machine Learning is the study of computer algorithms that improve automatically through experience."*
>
> Tom Mitchell, Machine Learning, McGraw-Hill, 1997.

This second definition not only confirms once more the significance of experience, which can be represented by data, but also underlines that the learning process needs to be as *autonomous* as possible from human intervention. The learning process should be autonomous to improve, for instance, its robustness and the adaptability of an AI system to task modifications.

1.1 Reinforcement Learning: Learning to Act

From the perspective of taxonomy, Machine Learning can be partitioned into three main paradigms: *Supervised Learning*, *Unsupervised Learning*, and *Reinforcement Learning*. In *Supervised Learning*, a complete data set is provided, meaning that the correct output (or target) is provided for each sample in the training set. The main goal is to learn a mapping between the input data to the corresponding target values. A further dichotomy can be provided in this field, depending on the target space: if the output belongs to a finite set, the problem is denoted as *classification*; instead, in *regression* the goal is to make predictions in the (eventually multi-dimensional) real space. Supervised Learning has been applied to a wide range of problems, including image classification (Lu and Weng, 2007), natural language processing (Chowdhary, 2020), and recommendation systems (Isinkaye et al., 2015): we invite the reader to refer to Bishop and Nasrabadi 2006 for an extensive introduction on the topic.

In *Unsupervised Learning*, the algorithm is not provided with target values. Instead, it must discover the patterns and relationships in the data through techniques such as clustering (Xu and Wunsch, 2005) or anomaly detection (Chandola et al., 2009).

The last category is *reinforcement learning* (RL, Sutton and Barto 1998), where the goal is to learn a policy of actions in a *Sequential Decision Process* by interacting with an environment. In other terms, the RL objective is to train the agent to maximize some reward signal by observing a series of states of the environment and choosing consequent actions. Due to the sequential nature of the agent-environment interaction, RL can be considered the closest approach to artificial intelligence: indeed, the agents need to learn and adapt to the environment via trial and error in a way similar to how humans and animals learn. Therefore, it is unsurprising that the main inspiration for this field comes from the psychology of animal learning and neuroscience. Despite the simplicity of the general framework, RL has achieved many significant successes in various challenging fields in recent years, especially when combined with deep neural networks: huge results were obtained in games, beating, for instance, the world champion in the board game of Go (Silver et al., 2016) or achieving super-human performances in video-games (Mnih et al., 2015), even with multi-agent frameworks (Berner et al., 2019). However, impressive advances are recently being also attained in real-world applications, such as robotic control (Levine et al., 2016), autonomous driving (Shalev-Shwartz et al., 2016) or healthcare (Thapa et al., 2005).

1.2 Motivations and Research Problems

RL researchers have achieved numerous impressive accomplishments in recent years, to a large extent thanks to the application of deep learning techniques. However, the learning process when complex models are adopted is not so straightforward: configuring and setting up RL algorithms efficiently can be challenging since many factors can significantly impact their ability to learn. The *tuning process* regards two different macro-elements:

- the first element to be set up is the *environment*: this process requires a thorough comprehension of the problem at hand and a careful design of the fundamental components, such as the reward function. This can require a lot of thought and attention to detail, as each factor must be carefully defined to accurately model the problem so that the selected algorithms can detect efficient policies. Furthermore, most RL algorithms deal with discrete-time environments where interactions happen at a fixed time scale. Therefore, if the true system evolves in continuous time, there is the need to configure the *control frequency* of the agent, which has a non-negligible effect on the learning capability of the algorithms.
- The second element is *hyperparameter tuning*: RL techniques often have a large number of hyperparameters that need to be tuned to achieve good performance. Finding the optimal values for these hyperparameters can be time-consuming, and it requires a good understanding of the algorithm and the environment in which it is being applied.

Overall, the difficulty of setting up RL environments and algorithms can vary significantly depending on the specific problem being addressed, but in any case, it requires a large amount of handcrafting. This can be considered as one of the major limitations in this research field, especially if we recall that the main goal is to develop *automatically improving* machines, according to Mitchell's definition of AI. This dissertation founds its main motivation in this direction: the goal is to enhance and automatize RL algorithms through a dynamic adaptation of the hyperparameters and the control frequency.

1.3 Original Contribution

After introducing the purpose of our work, we still need to define the specific research directions of our contribution. As a source of inspiration, we reckon with one of the most popular and discussed claims in RL, the *reward hypothesis* (Sutton and Barto, 2018):

> *"That all of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal."*
>
> Richard Sutton and Andrew Barto, Reinforcement learning: An introduction. MIT press, 2018.

This concept can be further summarized in the emblematic sentence *"Reward is enough"* (Silver et al., 2021), and is the source of many discussions and inspirations for many research directions (Vamplew et al., 2022; Abel et al., 2021). In particular, we follow two completely different questions that arise from this sentence:

- (Q1) If any task can be represented as a sequential problem with appropriate reward signals, can *learning* be framed as an RL environment?

- (Q2) The reward signal is essential to train RL agents. But how frequently should it be collected?

These two separate questions drive the contributions presented in this dissertation.

Hyperparameter Optimization through Meta-RL In Part I, we try to answer (Q1) and address the hyperparameter optimization problem for RL. In particular, we frame this problem as a Markov Decision Process, which can be tackled through RL techniques in a meta-learning manner: in this abstract environment, the observed features summarize the current state of the learning process on a set of closely related tasks. The action represents the hyperparameter choice used for the update rule selected. The reward process, instead, is modeled as the gain obtained in terms of expected return. In other terms, we satisfy Sutton and Barto's hypothesis by designing the reward as learning. Thanks to this definition of the problem, it is possible to adopt RL algorithms to learn a meta-policy, able to provide a dynamic selection of hyperparameters, which can be adapted throughout the learning process to the current policy and the task. Besides the general framework, which can be adapted to any learning algorithm (not only RL based), we apply this meta-RL approach to policy-based methods, where the (inner) policies are directly parametrized, and the update rule is based on stochastic gradients. The reason for this choice is related to the fact that, especially for this class of algorithms, the choice of the learning rates is particularly challenging since large stepsizes are riskier but can lead to fast improvements, while smaller steps are safer and, with the effect of slowing down the convergence, can provide monotonic improvement guarantees (Kakade and Langford, 2002; Pirotta et al., 2013b; Schulman et al., 2015). Furthermore, the research regarding learning rate control led to the development of several techniques, including adaptive learning rates schedules (Kingma and Ba, 2014), trust-region updates (Schulman et al., 2015) or approaches aimed at maximizing the immediate gains in terms of performance improvement (Paul et al., 2019). Therefore, we implement the meta-MDP configuration for the dynamic selection of the learning rate for gradient-based update rules and of the trust region constraint for one of the most successful state-of-the-art algorithms: Trust Region Policy Optimization (Schulman et al., 2015).

Control Frequency Adaptation In the second part, we address (Q2) and the problem of setting up the environment in terms of retrieving the best possible control frequency. Indeed, the choice of this parameter has a relevant impact on the ability of RL algorithms to learn a highly performing policy, especially for this class of methods, where models are used to estimate the value of actions affecting the environments in one *time unit*. Trivially, acting with the highest possible frequency leads to the best control opportunities, but the worse signal-to-noise ratio of the rewards collected challenges the learning capabilities of the models, badly affecting their sample complexity. We exploit this trade-off thanks to the introduction of the concept of *action persistence*, which refers to the repetition of a specific action for a predetermined number of time steps. After an analysis of how the persistence affects the performance of the optimal policy, we develop a novel approach to identify the optimal persistence. Successively, we consider dynamically adapting the persistence in an online setting: indeed, in different regions of the state space, the agent may be required to act more frequently to have more active control. Hence, we developed modified versions of the classic Q-learning (Watkins, 1989) and of the state-of-the-art DQN algorithm (Mnih et al., 2015) to make the agent capable of learning the expected gain related not only to the choice of a specific action but also its duration is time.

1.4 Structure and Content Outline

This dissertation is organized as follows. Before diving into our main contributions, we present in the first two chapters the foundations of reinforcement learning. This is not intended to be exhaustive, but only an ancillary introduction of the background knowledge required for the comprehension of our work:

- In Chapter 2 we introduce the fundamental elements of reinforcement learning: at first, we provide the definition of Markov Decision Process and of policy, which are commonly adopted to respectively model the environment (Puterman, 2014) and the agent. The most important tools for RL are then presented in the form of Bellman equations (Bellman et al., 1957), along with exact solution methods, that can be applied under perfect knowledge of the models.
- Chapter 3 is devoted to show an overview of a variety of RL algorithms, essential for the comprehension of the material discussed in the later chapters.

After the introduction, the dissertation is divided in two parts, which represent the original contribution of this thesis. In Part 1 we focus on Hyperparameter Optimization, and to device learning as a Sequential Decision Problem, providing some solution approaches to enhance the learning capabilities of RL algorithms.

- Chapter 4 represents the first original contribution. The main subject is the development of Meta-RL techniques for improving learning and hyperparameter optimization in new tasks. After briefly analyzing the related works in the literature, we devise the problem as a Meta-Markov Decision Process, where the actions correspond to the hyperparameters selection, and the reward is learning. In the first instance, we considered some assumptions related to the smoothness of the environment with respect to the task and obtained interesting bounds on the difference in terms of performances within two different tasks. We then applied the meta-learning concept to the selection of the step size for policy gradient approaches and adopted FQI (Ernst et al., 2005), a value-based RL algorithm, to select the most promising learning rates. The effectiveness of the approach is then evaluated on simulated domains, showing the advantages of the selection of an adaptive step size. The content of this chapter has been presented at the *8th ICML Workshop on Automated Machine Learning* (*AutoML*) in 2021 (Sabbioni et al., 2021) and will be published as a conference paper for *ECML 2023* (Sabbioni et al., 2023).
- In Chapter 5, we analyze the main drawbacks of the previously introduced approach, which shows unfeasible scaling with the dimensionality of the problem. We attempt to solve such limitations through the introduction of a specific set of features motivated by Information Theory. In this way, it is possible for us to build a *policy-agnostic* and *task-agnostic* meta-agent, which is then applied to optimize the hyperparameters of one of the state-of-the-art approach in policy-based RL,

Trust Region Policy Optimization (Schulman et al., 2015). The content of the chapter has been published in (Occorso et al., 2022) at *ECML-PKDD Workshop on Meta-Knowledge Transfer* in 2022.

While in the first part, the aim is to improve learning capabilities for RL algorithms by dynamically optimizing the hyperparameters, in part II the research is focused on the configuration of the control frequency.

- In Chapter 6, we introduce the literature related to the selection of the frame rate and the control frequency in RL, as well as a survey of the methods devoted to the detection of the best frequency. We then introduce the notion of *action persistence*, showing that it can be understood as either a model configuration or a policy parameter. After an analysis of how a fixed persistence affects the performance, we present a novel algorithm, PFQI, with the goal of learning the optimal value function at a given persistence and a heuristic approach to identify the optimal control frequency. The material covered in this chapter has been taken and adapted from (Metelli et al., 2020), published at *ICML 2020*. Among the experimental campaign performed to validate the approach, we furthermore present the results obtained from the application of our approach in the financial setting, in which an artificial agent trades foreign exchange currencies at different time scales. The results of this section are extracted from (Riva et al., 2021), published at *ICAIF 2021*, as part of a wider series of publications in the financial field (Bisi et al., 2020a; Riva et al., 2022).

- Chapter 7 studies the problem of choosing a dynamic control frequency. The action space is therefore enlarged to the space of *persistence options*, in such a way that the agent selects both the action and its duration. We introduce a new Bellman operator, thanks to which, using the information collected from the environment, it is possible to update the knowledge of the whole persistence space. After proving the contraction properties of the new operator, we then extend two well-known algorithms, Q-learning (Watkins and Dayan, 1992) and DQN (Mnih et al., 2015), to operate within the persistence option framework. The content of this chapter, available from (Sabbioni et al., 2022), has been presented at *RLDM 2022* and has been published as a conference paper for *AAAI 2023*.

- In conclusion, in Chapter 8, the overall contribution of this dissertation is summarized, and the main limitations are discussed. Additionally, potential future directions for research in this area are suggested.

Additional results and the proofs that were left out of the main text of the dissertation can be found in the Appendices A, B and C. In Appendix D we present some preliminary results related to a gradient-based approach for persistence learning.

CHAPTER 2

Foundations of Reinforcement Learning

2.1 Introduction

In this chapter we introduce the core components of the Reinforcement Learning (RL) framework: the process of making decisions over time to reach a goal is commonly framed as a Sequential Decision Making (SDM) problem. In this model, the *agent* is the entity making the decisions and interacts with an uncertain *environment*. The agent has the opportunity to collect information related to the environment, and to choose the action accordingly. The environment is then subject to a dynamic *transition*, in part influenced by the decisions of the agent, and the latter one is given a feedback signal, called *reward*.

This framework is very general and can be adopted as a common model across multiple disciplines, including automatic control, neuroscience, operations research, and economics (Hutter, 2004; Sutton, 2022), even though the adopted terminology might be different: for instance, in control theory, the decision maker is addressed as *controller* sending a *control signal* to the environment, while in psychology, the agent is an *organism*, which receives *stimuli* and sends *responses*.

In many real-world scenarios, the environment dynamics and the interaction can be thought of as continuous (Doya, 2000); however, a common restriction adopted in literature formalizes the problem by providing a time discretization into a sequence of decision steps with a fixed control frequency. Of course, a more fine-grained discretization provides a better approximation of a continuous process, but there are many drawbacks in the identification of the best action. The topic of the identification of the best control frequency is discussed in more detail in Chapter 6.

The consequent interaction is formalized as a Markov Decision Process (MDP, Puterman (2014)), providing the ideal theoretical framework for RL. In this Chapter, we will introduce the fundamental elements useful to understand the rest of the disserta-

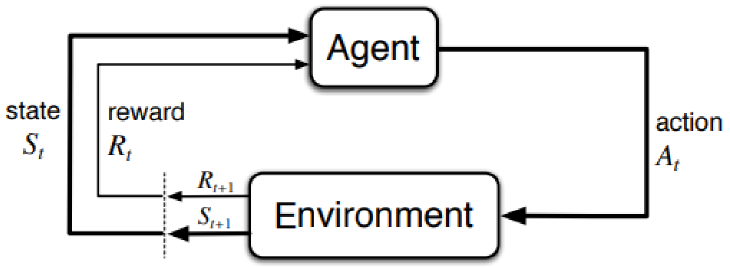


Figure 2.1: *Graphical representation of the agent-environment interaction in a Markov Decision Process. Source: Sutton and Barto (1998)*

tion. For an extended analysis of the most important aspects related to the formulation, we suggest the reader refer to some of the main textbooks on this topic (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 2018; Szepesvári, 2010; Bertsekas, 2005; Agarwal et al., 2019).

Chapter Outline This chapter is organized as follows. In Section 2.2 we provide a formal definition of a Markov Decision Process and an explanation of the agent-environment interaction. In particular, in Subsection 2.2.2 we present a list of possible extensions, some of which will be taken into account in the next chapters. We furthermore define some of the most useful concepts, such as the policy and the trajectories, that are used then in Section 2.3 to introduce some fundamental notions such as of value function and expected return. as well as the Bellman operators, that are paramount to solve an MDP. Finally, in Section 2.4, we exploit the *dynamic programming principle* and present some techniques to find the optimal policy in finite MDPs under perfect knowledge of the environment.

2.2 Markov Decision Processes

As mentioned in the previous Section, a SDM is formally framed thanks to the definition of MDPs. The first introduction of this terminology belongs to Bellman (1954), while the core elements were analyzed in Bellman et al. (1957) and Howard (1960).

Definition 2.1 (Markov Decision Process, Puterman 2014)**.** *A discrete-time, discounted Markov Decision Process (MDP) is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \mu, \gamma \rangle$, where:*

- *$\mathcal{S}$ is denoted as the **state space** and is a measurable set representing all the possible states of the environments.*

- *$\mathcal{A}$, referred to as **action space**, is a (non-empty) measurable set representing all the available actions for the agent. In a more general formulation, the set of available actions is state dependent. In this case, it is possible to consider a collection of sets $\mathcal{A}_s$ for each state $s \in \mathcal{S}$, in the sense that for each state s there is a different subset of available actions belonging to the action space. In the following, we will assume a single action space.*

- $P : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$ *is the* ***transition model*** *or* ***transition kernel****. It provides, for each state-action pair* $(s, a) \in \mathcal{S} \times \mathcal{A}$*, the probability distribution* $P(\cdot|s, a)$ *of reaching the next state over the state space* $\mathcal{S}$*.*
- $R : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathbb{R}}$ *is the* ***reward*** *model, which assigns for each tuple* $(s, a) \in \mathcal{S} \times \mathcal{A}$ *a probability measure* $R(\cdot|s, a)$ *over* $\mathbb{R}$*. The reward model induces the* ***reward function*** r*, which is the expected reward when performing action* $a \in \mathcal{A}$ *in state* $s \in \mathcal{S}$*, i.e.* $r(s, a) := \int_{\mathcal{S}} dr R(r|s, a)$*. In other, more general formulations, the reward model is also considered dependent on the next state* $s' \in \mathcal{S}$ *reached according to the transition model* P*. In this case, the reward function considers also the expectation over* s'*, hence it would be defined as* $r(s, a) = \int_{\mathcal{S}} P(ds'|s, a) \int_{\mathbb{R}} rR(dr|s, a, s')$*.*
- $\mu \in \Delta_{\mathcal{S}}$ *is the starting-state distribution.*
- $\gamma \in [0, 1)$ *is the discount factor. It has the purpose of measuring the current utility of rewards obtained in the future.*

2.2.1 Interaction

The interaction between the agent and the environment happens at each decision step t. Once the agent observes the current state $s_t \in \mathcal{S}$, an *action* $a_t \in \mathcal{A}$ is decided according to its policy; One time step later, the agents receives the reward $r_t \sim R(\cdot|s_t, a_t)$, and the environment transitions to a new state s_{t+1} according to the transition model, i.e. $s_{t+1} \sim P(\cdot|s_t, a_t)$.

A scheme of the agent-environment interaction is depicted in Figure 2.1.

The formalization of the transition kernel in the MDP definition embeds two assumptions:

1. the process is *stationary*, meaning that the transition kernel P is *time-invariant*;
2. the transition satisfies the *Markov property* (Durrett, 2019): once the current state s_t is known (as well as the action a_t), the dynamics of the process is independent of the past. In other terms:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t, a_{t-1}, s_{t-1}, \dots, a_1, s_0). \tag{2.1}$$

A third important assumption usually adopted is the existence of a bound on the support of the reward model:

Assumption 2.1 (Bounded Rewards)**.** *There exists a finite constant* $R_{max} \in \mathbb{R}_{>0}$ *uniformly bounding the support of the random model for each state-action pair, such that* $R : \mathcal{S} \times \mathcal{A} \to \Delta_{[-R_{max}, R_{max}]}$.

As a consequence, the reward function r is uniformly bounded, i.e.

$$\|r(s, a)\|_\infty = \sup_{(s,a)\in\mathcal{S}\times\mathcal{A}} |r(s, a)| \le R_{max}$$

Remark 2.2. *The transition kernel P includes the transition to a new state conditioned on the action chosen. However, there are several cases in which the agent has limited effect on the dynamics of the states. Therefore, it is often possible to partition the features observed in the state into two classes (Chitnis and Lozano-Pérez, 2020): $\mathcal{N}$ is the space of endogenous states, the ones possibly affected by the agent decisions, while $\mathcal{X}$ is the exogenous space. An exogenous state contains the features of the environment that can only be observed by the agent. Hence, $\mathcal{S} = \mathcal{N} \times \mathcal{X}$, and the current state is the combination of the two parts $s_t = (n_t, x_t)$, with $n_t \in \mathcal{N}, x_t \in \mathcal{X}$, and the same partition can be applied to the transition process, split into the two kernels $P_{\mathcal{N}}$ and $P_{\mathcal{X}}$. The exogenous kernel is only dependent on the previous exogenous state, while the endogenous transition is conditioned on the action chosen, and the overall state of the environment s_t:*

$$P(s_{t+1}|s_t, a_t) = P_{\mathcal{N}}(n_{t+1}|s_t, a)P_{\mathcal{X}}(x_{t+1}|x_t).$$

2.2.2 MDP Extensions

As described in the previous sections, MDPs build the general framework to solve sequential decision processes. However, this formulation might be limited in several real-world scenarios and need peculiar generalizations, some of which will be briefly described and taken into account throughout this dissertation.

- Many RL applications are designed under the assumption that the state is *fully observable*, meaning that the agent is capable of observing all the features determining the dynamics of the environment. However, in a large variety of problems, this assumption does not hold, as the agent is only capable of observing a different set of features: this framework is modeled as a Partially Observable Markov Decision Process (POMDP, Åström (1965); Monahan (1982)), defined as a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \Omega, \mathcal{O}, \mu, \gamma \rangle$. At each discrete time step t, the state evolves to s_{t+1} as usual according to the transition kernel P, while the agent is capable of observing a vector of features ω_t belonging to Ω, defined as *observation space*. The observation is given by a (conditioned) probability distribution according to the *observation model* $\mathcal{O}$, such that $o_t \sim \mathcal{O}(\cdot|s_t, a_t)$. We will deal with a peculiar case of POMDP in Chapter 5.

- The evolution towards a new state may be not conditioned on a pre-determined time discretization, but transitions can happen with different *timings*. This is the case of Semi-Markov Decision Processes (Semi-MDP, Howard (1963); Sutton et al. (1999b)), for which the time between consecutive decision steps is variable. A definition can be found in (Lippman, 1973), where a Semi-MDP is described as a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, T, \mu, \gamma \rangle$. The main difference consists in the presence of a model of the transition durations $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Delta_{\mathbb{R}_{>0}}$, such that the time for the system to evolve from state s to s' given action a is a (non-negative) random variable, distributed as $T(\cdot|s, a, s')$. Consequently, the reward is also subject to the time passed between the steps. This concept is usually considered in the context of Hierarchical-RL (Pateria et al., 2021), and we will briefly consider this framework in Chapter 7.

- RL problems are usually formulated as discrete-time problems with a specific time discretization derived from continuous-time processes. A branch of RL literature is focused on the research regarding Continuous-time MDP (Cont-MDP, Bradtke and Duff (1994); Doya (2000); Yildiz et al. (2021)), which are very often less tractable than standard decision processes. The definition of a Cont-MDP considers a continuous state function s_t and a continuous *control input* a_t. A *transition rate* p replaces the transition kernel P. If the process is stochastic, the dynamics also include a diffusion rate σ (Seierstad, 2009). In other terms, the state evolves from time t to T according to the dynamics of p and σ:

$$s_T = \int_t^T p(s_u, a_u) du + \int_t^T \sigma(s_u, a_u) dw_u,$$

where dw is a (possibly multivariate) Brownian Motion. An analogous transformation is applied to the reward process, which is replaced with a continuous-time *reward rate* r_t.

- The dynamic and reward processes of the decision problem can depend on external parameters. For instance, the performance of a race car might be affected by tire wear or the friction of the circuit. This framework can be modeled thanks to the introduction of exogenous variables, called *contexts*. The problem is then devised as a Contextual Markov Decision Process (CMDP, Hallak et al. (2015); Sodhani et al. (2021)), a tuple $\langle \Omega, \mathcal{S}, \mathcal{A}, \mathcal{M} \rangle$ where Ω is called the *context space*. The state and action spaces $\mathcal{S}$ and $\mathcal{A}$ are shared, and $\mathcal{M}$ is a function that maps any context $\omega \in \Omega$ to a standard MDP, such that $\mathcal{M}(\omega) = \langle \mathcal{S}, \mathcal{A}, P_\omega, R_\omega, \gamma_\omega, \mu_\omega \rangle$. In brief, a CMDP includes in a single entity a set of tasks. Each context can be seen as a different *task* with a potentially different solution. One might be interested in training an agent with the goal of generalizing across all the set of possible tasks (Multi-task RL, Sodhani et al. (2021); Calandriello et al. (2014)[1]), or with the purpose of rapidly adapting to the new context with small amounts of experience (Meta-RL, Rakelly et al. (2019)). The definition of a Hidden-Mode MDP in Hadoux et al. (2014) in the context of Non-stationary RL is also closely related to the one provided for a CMDP. Part I in this dissertation deals with CMDPs.

- A variant of the CMDP introduces the possibility of configuring the model parameters that affect the system dynamics. This is the case of Configurable Markov Decision Processes (Conf-MDPs, Metelli et al. (2018); Silva et al. (2019); Ramponi et al. (2021)), where the goal is to jointly train the model and the policy to optimize the return. In Chapter 6, we deal with a particular environment parameter, the *action persistence*.

2.2.3 Trajectories and Return

The interaction of an agent with the MDP generates a sequence of states, actions, and rewards, that can be collected to form a *history*. Starting from the initial state s_0, a history h_t with length $t \in \mathbb{N}$ is hence:

$$h_t = (s_0, a_0, r_0, s_1, a_1, \dots, r_{t-1}, s_t)$$

[1]In the last case, the framework is instead defined as a Multi-Task MDP, although the definition coincides with the CMDP one.

We will define as $\mathcal{H}_t = (\mathcal{S} \times \mathcal{A} \times \mathbb{R})^t \times \mathcal{S}$ the set of all possible histories with length t. The generated episode then continues up to a certain horizon T, the time at which the interaction ends, and the history can also be defined as *trajectory* $\tau = h_T$. The horizon can be either finite ($T < \infty$) or infinite ($T = \infty$), in such a way that the set of all possible trajectories $\mathcal{T}$ degenerates to $(\mathcal{S} \times \mathcal{A} \times \mathbb{R})^{\mathbb{N}}$. In many practical cases, the MDPs are embedded with a fixed horizon T, or with a set $\mathcal{S}_A$ of *absorbing states*[2]. Under these circumstances, the trajectory length can vary, and, without loss of generality, we can define $T(\tau)$ the trajectory length (finite or infinite) associated with each trajectory $\tau \in \mathcal{T}$. At the end of each trajectory $\tau \in \mathcal{T}$ we can observe its *return function* G_γ, defined as the cumulative sum of the rewards encountered, discounted by the discount factor γ:

$$G(\tau) := \sum_{t=0}^{T(\tau)} \gamma^t r_t.$$

Sometimes we will be interested in *sub-trajectories* $\tau_{t_1:t_2}$ of a complete trajectory τ, for $0 \leq t_1 < t_2 \leq T(\tau)$, i.e., the part of history seen starting from time t_1 up to time t_2. In this case, the *partial return* is clearly computed as:

$$G(\tau_{t_1:t_2}) := \sum_{t=0}^{t_2-t_1} \gamma^t r_{t_1+t} = \sum_{t=t_1}^{t_2} \gamma^{t-t_1} r_t. \tag{2.2}$$

In the specific case of $t_2 = T(\tau)$, the sub-trajectory starting from $t_1 = t$ will be more briefly denoted as τ_t, and the related return as $G(\tau_t)$ or $G_t(\tau)$. Moreover, we will often be interested in taking into account *transitions*, i.e., sub-trajectories involving a single step in the environment. These transitions are then considered as tuples (s_t, a_t, r_t, s_{t+1}).

From the definition of return, the significance of the discount factor becomes clear from many points of view: first, there is an economic meaning: future rewards are assigned a lower value with respect to the same value obtained immediately, with a strong connection to econometrics. Higher values of γ are related to a more *far-sighted* view, while a more myopic point of view is associated with a low γ, as the *effective horizon* becomes equivalent to $1/(1-\gamma)$. From another perspective, the presence of the discount factor allows infinite-long trajectories to maintain a well-defined return: this is an important property, as the return function becomes bounded by $R_{max}/(1-\gamma)$, avoiding divergence to infinite. If the length of the trajectories is almost surely finite (either finite-horizon or MDPs for which the probability of entering an absorbing state in a finite time is 1), it is also possible to consider a discount factor equal to 1, making the return function the sum of the rewards collected. The opposite case, $\gamma = 0$, transforms the problem into a *single-step* decision problem. The whole RL framework related to control and planning collapses to a *supervised learning* problem, as the main goal is reduced to provide an approximation of the reward function. As a consequence, it is common to consider γ as a parameter related to the MDP formalization, as it is a component strongly impacting the reward signal. However, sometimes it is considered as a hyperparameter of the training algorithms (Paul et al., 2019). In this dissertation, we

[2]A state $s \in \mathcal{S}$ is defined as *absorbing* or *terminal* if the MDP, once reached s, remains in the same state with no further rewards, i.e., $P(\cdot|s,a) = \delta_s(\cdot), r(s,a) = 0 \forall a \in \mathcal{A}$

will always consider the discount factor as a fixed value embedded with the definition of the MDP.

2.2.4 Policies

The purpose of RL agents is to make decisions based on the observed states of the environment. The mathematical formalization to describe the decision rule of an agent is provided with the definition of the *policy*. In the most general definition, for each decision step t, a policy π_t provides a probability distribution over the Action space $\mathcal{A}$, based on the current history h_t of states, actions and rewards collected:

Definition 2.3 (Policy). *Given a MDP $\mathcal{M}$ with horizon T (either finite or infinite), a history-dependent policy acting on $\mathcal{M}$ is a sequence $\pi = (\pi_t)_{t\in[T]}$, such that, for each t, $\pi_t : \mathcal{H}_t \to \Delta_{\mathcal{A}}$.*

The set of all possible history-dependent policies will be denoted as Π^H. In many scenarios, the current state provides enough information for the agent to make it able to select the best action. Hence, a very important subset $\pi^M \subset \pi^H$ contains all the *Markovian* policies, which depend only on the current state. More formally, we can say that a policy $\pi \in \Pi^H$ is Markovian if there exists a collection $\widetilde{\pi} = (\widetilde{\pi}_t)_{t\in[T]}$ such that, for each t, $\widetilde{\pi}_t : \mathcal{S} \to \Delta_{\mathcal{A}}$ is a.s. equal to π_t, i.e. $\pi_t(\cdot|h_t) = \widetilde{\pi}_t(\cdot|s_t)$ a.s.
Furthermore, as the transition process in a MDP is stationary, we are usually interested in *stationary policies* as well. A markovian policy π is defined as stationary if $\forall t, t' \in [T], s \in \mathcal{S}$, $\pi_t(\cdot|s) = \pi_{t'}(\cdot|s)$. We will usually deal with Markovian, stationary policies, for which we will use a simpler notation $\pi(\cdot|s)$ to denote a specific policy and Π for the related policy set. For the rest of this dissertation, we will adopt the general term "policy" to refer to a Markovian, stationary policy. In Chapter 7, and more in general in the framework of *Option Learning*, we can adopt temporally-extended policies, acting on *macro-actions* or *options* (Precup, 2001; Mann et al., 2015). As a conclusive remark, a policy π is denoted as *deterministic* if the probability distribution over the action space is reduced to a Dirac delta measure. In this case, the policy can be considered as a mapping $\pi : \mathcal{S} \to \mathcal{A}$, and for each state $s \in \mathcal{S}$, the prescribed action is $\pi(s)$.

The common (non-limiting) assumption is that each policy $\pi \in \Pi$ admits a probability density function with respect to the Lebesgue measure, as well as the transition kernel P and the initial distribution μ. Under these assumptions, we are, for instance, allowed to denote with $\pi(a|s)$ to refer to the probability associated with a specific state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$. The same assumption can be applied to all history-dependent policies in Π^H. In this way, it is possible for us to determine the distribution of the trajectories gathered through the interaction of a policy $\pi \in \Pi^H$ with the environment $\mathcal{M}$. Specifically, each trajectory $\tau = (s_0, a_0, r_0, \dots, a_{T-1}, s_T)$, which can be decomposed in the whole set of sub-histories h_t for all $t \in [T]$, is associated to a probability measure $\rho_\pi(\tau)$:

$$\rho_\pi(\tau) = \mu(s_0) \prod_{t=1}^{T} \pi_t(a_t|h_t) P(s_{t+1}|a_t, s_t). \tag{2.3}$$

As a small remark, Equation 2.3 can be simplified in the case of a stationary Markovian policy as follows:

$$\rho_\pi(\tau) = \mu(s_0) \prod_{t=1}^{T} \pi(a_t|s_t) P(s_{t+1}|a_t, s_t). \tag{2.4}$$

Transition Kernels

The combination of an MDP $\mathcal{M}$ and a policy $\pi \in \Pi$ gives rise to a Markov Chain (MC, Norris (1997)) on the state space $\mathcal{S}$, which evolves according to the transition kernel p_π.

$$p_\pi(\cdot|s) = \int_{\mathcal{A}} \pi(\,\mathrm{d}a|s) P(\cdot|s, a) \tag{2.5}$$

Sometimes, we will slightly abuse the notation when dealing with the state-action extension of Equation 2.5, which includes in the process a further application of the policy π, starting from a fixed state-action pair:

$$p_\pi(s', a'|s, a) = \int_{\mathcal{S}} P(\,\mathrm{d}s'|s, a) \int_{\mathcal{A}} \pi(\,\mathrm{d}a'|s'). \tag{2.6}$$

The consequent t-step transition kernel represents the probability distribution on the state space $\mathcal{S}$ for the MC after t steps, starting from s and following the policy π, and is recursively defined as follows:

$$\begin{aligned} p_\pi^1(\cdot|s) &:= p_\pi(\cdot|s); \\ p_\pi^{t+1}(\cdot|s) &:= \int_{\mathcal{S}} p_\pi^t(s'|s) p_\pi(\cdot|s') \,\mathrm{d}s'. \end{aligned} \tag{2.7}$$

In practice, $p_\pi^t(s'|s)$ represents the probability of reaching state $s' \in \mathcal{S}$ in t-step, starting from $s \in \mathcal{S}$ and following π thereafter. The same recursive definition can be applied to the state-action extensions, by following 2.6. These distributions can be combined, keeping into account the discount factor, to define the (discounted) *state-occupancy measure* δ_π^s:

$$\delta_\pi^s(\cdot) := \frac{1-\gamma}{\gamma} \sum_{t=1}^{\infty} \gamma^t p_\pi^t(\cdot|s)$$

This measure represents the (discounted) probability of reaching a state in the future, starting from s_0 and following π. If we want to consider the overall distribution over the state space in any point of the interaction, we need to consider also the initial distribution μ, to retrieve δ_π^μ as follows:

$$\delta_\pi^\mu(\cdot) := (1-\gamma)\mu(\cdot) + \gamma \int_{\mathcal{S}} \mu(s_0) \delta_\pi^{s_0}(\cdot) \,\mathrm{d}s_0 \tag{2.8}$$

Finally, once the distribution over the states is known, the distribution over the state-action pairs can be easily computed from the composition with the policy π:

$$\nu_\pi^\mu(s, a) := \delta_\pi^\mu(s) \pi(a|s) \quad \forall (s, a) \in \mathcal{S} \times \mathcal{A} \tag{2.9}$$

When $\gamma = 1$, δ_π^μ becomes equivalent to the stationary distribution of the MC with transition p_π, in such a way that $\delta_\pi^\mu(s) = \int_{\mathcal{S}} \delta_\pi^\mu(s') p_\pi(s|s') \,\mathrm{d}s'$ or, in operator form, $\delta_\pi^\mu = \delta_\pi^\mu p_\pi$.

2.3 Expected Return and Value Functions

In standard RL, the main objective is to maximize the expected return, where the expectation is with respect to the stochasticity of the environment and/or the policy. The expected return can then be considered the main performance index for the evaluation of a policy, and its improvement.

Formally, the *objective function* is expressed as follows: [3]

$$\max_{\pi \in \Pi} J(\pi) := \mathop{\mathbb{E}}_{\tau \sim \rho_\pi} [G_\gamma(\tau)] = \mathop{\mathbb{E}}_{\substack{s_0 \sim \mu \\ a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim P(s_t, a_t)}} \Big[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\Big] \tag{2.10}$$

One of the key concepts in RL is the *value function* (Sutton and Barto, 1998), which provides a value associated with a state s Thanks to the exploitation of Markov's property of the MDP, this function allows to transform the multi-step optimization problem 2.10 into a fixed-point problem.

Definition 2.4 (Value function)**.** *Let $\mathcal{M}$ be an MDP, π a policy in Π and s any state in $\mathcal{S}$. The value function $V^\pi : \mathcal{S} \to \mathbb{R}$ is defined as the expected return starting from the state s, and following the policy π:*

$$V^\pi(s) := \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim P(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s\right] = \mathop{\mathbb{E}}_{\tau \sim \rho_\pi} [G(\tau) | s_0 = s] . \tag{2.11}$$

In the same fashion, it is also possible to define the *state-action value function*, addressed also as *Q-function* or *action-value function*.

Definition 2.5 (Action-Value function)**.** *Let $\mathcal{M}$ be an MDP, π a policy in Π and (s, a) any state-action pair in $\mathcal{S} \times \mathcal{A}$. The action-value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined as the expected return starting from (s, a), and then following the policy π:*

$$Q^\pi(s, a) := \mathop{\mathbb{E}}_{\substack{s_{t+1} \sim P(\cdot|s_t, a_t) \\ a_{t+1} \sim \pi(\cdot|s_{t+1})}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a\right] . \tag{2.12}$$

While the main purpose of the state value function is the *evaluation* of a policy, the action-value function finds its application in the context of *policy improvement*, since it allows detecting the most profitable actions. In this direction, it may be useful to define also the *advantage function*, which measures the expected return gain obtained by choosing an action a instead of following the policy π in a state s:

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

In conclusion, the expected return $J(\pi)$ or J^π can also be defined in terms of the state-value function by marginalizing over the initial state distribution:

$$J(\pi) = \int_{\mathcal{S}} \mu(\,\mathrm{d}s) V^\pi(s) \tag{2.14}$$

[3]For the sake of simplicity, the summation over the time-steps t in Equation 2.10 and following has ∞ as upper limit without loss of generality.

2.3.1 Bellman Equations

The definitions introduced for the value functions are expressed in terms of expectations of the return on the trajectories produced by the agent-environment interaction. However, we can exploit the Markov structure of the transition process (and the Markovian-stationary policy space Π) to relate the value in subsequent time steps in a *recursive* manner. The relations obtained are usually denoted as *Bellman Equations* (Bellman et al., 1957), and represent one of the cornerstones in RL, perhaps being also the main reason why stationary, Markovian policies are so important in this field.

Proposition 2.6 (Bellman Equations, Bellman 1954). *Let $\mathcal{M}$ be an MDP and $\pi \in \Pi$ be a policy, then:*

$$Q^\pi(s,a) = r(s,a) + \gamma \int_{\mathcal{S}} P(\mathrm{d}s'|s,a) V^\pi(s'), \tag{2.15}$$

$$V^\pi(s) = \int_{\mathcal{A}} \pi(\mathrm{d}a|s) Q^\pi(s,a). \tag{2.16}$$

These equations can also be defined in a more useful way in the form of *operators*.

Definition 2.7 (Bellman Expectation Operators, Bellman et al. 1957). *Let $\mathcal{M}$ be an MDP and $\pi \in \Pi$ be a policy. Consider a bounded, measurable function $f : \mathcal{S} \to \mathbb{R}$ and a state $s \in \mathcal{S}$. The Bellman expectation operator for state value functions T^π is defined as:*

$$(T^\pi f)(s) := \int_{\mathcal{A}} \pi(\mathrm{d}a|s) \left(r(s,a) + \gamma \int_{\mathcal{S}} P(\mathrm{d}s'|s,a) f(s') \right), \tag{2.17}$$

Moreover, considering a bounded, measurable function $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and a state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$, the Bellman expectation operator for state-action value functions T^π is defined as:

$$(T^\pi f)(s,a) := r(s,a) + \gamma \int_{\mathcal{S}} P(\mathrm{d}s'|s,a) \int_{\mathcal{A}} \pi(\mathrm{d}a'|s') f(s',a'), \tag{2.18}$$

Remark 2.8. *Equations 2.15,2.17 and 2.18 make sense if the reward model R depends only on the state-action pair (s,a). In more general formulations, r is dependent also on the next state s' according to the transition model P. Under this framework, the Bellman Equation for the Q-value function becomes:*

$$Q^\pi(s,a) = \int_{\mathcal{S}} P(\mathrm{d}s'|s,a) \big[r(s,a,s') + \gamma V^\pi(s') \big],$$

and the Bellman Expectation Operators need to be modified accordingly.

It is important to remark that the Bellman Operators are *linear* operators. Moreover, they enjoy the following properties:[4]

[4]The same consideration hold for the operators for state-action functions.

- *Monotonicity*: indeed, consider two bounded, measurable functions f and g such that $f(s) \geq g(s)\, \forall s \in \mathcal{S}$. Then:

$$\begin{aligned} T^\pi f(s) &= \int_A \pi(\mathrm{d}a|s)\big(r(s,a) + \gamma \int_{\mathcal{S}} P(\mathrm{d}s'|s,a) f(s')\big) \\ &\geq \int_A \pi(\mathrm{d}a|s)\big(r(s,a) + \gamma \int_{\mathcal{S}} P(\mathrm{d}s'|s,a) g(s')\big) = T^\pi g(s) \end{aligned}$$

- γ*-Contraction in* L_∞ *norm* (Puterman, 2014): indeed, consider two bounded, measurable functions f and g. Then:

$$\begin{aligned} \|T^\pi f - T^\pi g\|_\infty &= \| \int_{\mathcal{A}} \pi(\mathrm{d}a|s) \gamma \int_{\mathcal{S}} P(\mathrm{d}s'|s,a)(f(s') - g(s'))\|_\infty \\ &\leq \gamma \|f - g\|_\infty \int_{\mathcal{A}} \pi(\mathrm{d}a|s) \int_{\mathcal{S}} P(\mathrm{d}s'|s,a) = \gamma \|f-g\|_\infty \end{aligned}$$

As a consequence, the Banach-Caccioppoli fixed-point theorem (Banach, 1922) holds, and T^π admits a unique fixed point: the value function V^π and the action-value function Q^π:

$$\begin{aligned} V^\pi &= T^\pi V^\pi, \\ Q^\pi &= T^\pi Q^\pi. \end{aligned}$$

In Section 3.3, we will iteratively adopt the Bellman Expectation Operators to perform *policy evaluation*. However, in order to improve the expected return of the policy, we will need further tools, as seen in the next section.

2.3.2 Optimality Criteria

As mentioned at the beginning of Section 2.3, the goal is the maximization of the expected return. For practical approaches, it can be easier to define the optimality criteria for value functions; indeed, in the following, we will introduce the *Bellman Optimality Operators*, which can be iteratively applied to improve the policy and to retrieve the optimal one.

Definition 2.9 (Optimal State Value Function and Optimal Action Value function)**.** *Let $\mathcal{M}$ be an MDP. A policy $\pi^\star \in \Pi$ is optimal if:*

$$\begin{aligned} V^{\pi^\star}(s) \geq V^\pi(s) &\quad \forall \pi \in \Pi, \forall s \in \mathcal{S} \\ Q^{\pi^\star}(s,a) \geq Q^\pi(s,a) &\quad \forall \pi \in \Pi, \forall (s,a) \in \mathcal{S} \times \mathcal{A} \end{aligned}$$

The optimal state value function and optimal action-value function are then defined according to the values attained by the optimal policy:

$$\begin{aligned} V^\star(s) := V^{\pi^\star}(s) &\quad \forall \pi \in \Pi, \forall s \in \mathcal{S} \\ Q^\star(s,a) := Q^{\pi^\star}(s,a) &\quad \forall \pi \in \Pi, \forall (s,a) \in \mathcal{S} \times \mathcal{A} \end{aligned}$$

Remark 2.10. *Definition 2.9 can be trivially extended for all history-dependent policies in Π^H. However, the optimal value functions will remain constant. This holds since for every policy $\widetilde{\pi} \in \Pi^H$, there exist a stationary, Markovian policy $\pi \in \Pi$ such that $V^\pi(s) = V^{\widetilde{\pi}}(s)\, \forall s \in \mathcal{S}$ (Puterman 2014, Theorem 5.3.3, part a).*

Remark 2.11. *Definition 2.9 of optimal policy denotes a stronger goal than the maximization of the expected return expressed as the main objective function 2.10: since J^π is the expected value function marginalized over the initial distribution, the optimal policy maximizes the expected return for any initial distribution μ, while the converse does not necessarily hold. In some cases, a policy maximizing the return is denoted as J-optimal.*

In the same fashion as the Bellman Equations, we can define recursive relations also for the optimal value functions:

Proposition 2.12 (Bellman Optimality Equations)**.** *Let $\mathcal{M}$ be an MDP, then:*

$$Q^\star(s,a) = r(s,a) + \gamma \int_{\mathcal{S}} P(\mathrm{d}s'|s,a) V^\star(s'), \tag{2.19}$$

$$V^\star(s) = \sup_{a\in\mathcal{A}} Q^\star(s,a). \tag{2.20}$$

Analogously, $T^\star$ is the operator form related to these Equations:

Definition 2.13 (Bellman Optimality Operators)**.** *Let $\mathcal{M}$ be an MDP. Consider a bounded, measurable function $f : \mathcal{S} \to \mathbb{R}$ and a state $s \in \mathcal{S}$. The Bellman optimality operator for state value functions $T^\star$ is defined as:*

$$(T^\star f)(s) := \sup_{a\in\mathcal{A}} \left\{ r(s,a) + \gamma \int_{\mathcal{S}} P(\mathrm{d}s'|s,a) f(s') \right\} \tag{2.21}$$

Moreover, considering a bounded, measurable function $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and a state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$, the Bellman optimality operator for state-action value functions $T^\star$ is defined as:

$$(T^\star f)(s,a) := r(s,a) + \gamma \int_{\mathcal{S}} P(\mathrm{d}s'|s,a) \max_{a'\in\mathcal{A}} f(s',a'), \tag{2.22}$$

While the Bellman Optimality Equations and Operator enjoy the same generalization as in Remark 2.8, the main difference is the presence of the supremum, for which the operators are no longer linear. On the other side, monotonicity and contraction properties still hold:

- *Monotonicity*: consider two bounded, measurable functions f and g such that $f(s) \geq g(s)\ \forall s \in \mathcal{S}$. Then:

$$\begin{aligned}(T^\star f)(s) &= \sup_{a\in\mathcal{A}} \left\{ r(s,a) + \gamma \int_{\mathcal{S}} P(\mathrm{d}s'|s,a) f(s') \right\} \\ &\geq \sup_{a\in\mathcal{A}} \left\{ r(s,a) + \gamma \int_{\mathcal{S}} P(\mathrm{d}s'|s,a) g(s') \right\} = (T^\star) g(s).\end{aligned}$$

- γ-contraction in L_∞ norm: as before, consider two bounded, measurable functions f and g. Then:

$$\begin{aligned}\|T^\star f - T^\star g\|_\infty &= \left\| \sup_{a\in\mathcal{A}} \gamma \int_{\mathcal{S}} P(\mathrm{d}s'|s,a)(f(s') - g(s')) \right\|_\infty \\ &\leq \gamma \|f-g\|_\infty \sup_{a\in\mathcal{A}} \int_{\mathcal{S}} P(\mathrm{d}s'|s,a) = \gamma\|f-g\|_\infty.\end{aligned}$$

Again, all the hypotheses for the fixed-point theorem hold, hence $T^\star$ admits a unique fixed point: the optimal value function $V^\star$ and the optimal action-value function $Q^\star$:

$$\begin{aligned} V^\star &= T^\star V^\star, \\ Q^\star &= T^\star Q^\star. \end{aligned}$$

A complete analysis and discussion on Bellman operators can be found on (Puterman, 2014; Szepesvári, 2010).

Greedy Policy

In the previous subsection, we defined the optimal policy as the one attaining the optimal (action) value function. Consequently, the knowledge of the optimal action-value function $Q^\star$ is a sufficient condition to retrieve the optimal behavior, which acts greedily to maximize the expected return in each possible state:

Definition 2.14 (Greedy Policy)**.** *Let $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be a bounded, measurable function. A policy $\pi' \in \Pi$ is defined as greedy*[5] *with respect to f if:*

$$\forall s \in \mathcal{S} : \pi'(s) \in \arg\max_{a \in \mathcal{A}} f(s, a). \tag{2.23}$$

Puterman 2014 shows that the greedy policy w.r.t. the optimal action-value function exists and, at least in discrete state spaces, it is an optimal policy:

Theorem 2.15 (Theorem 6.2.7, (Puterman, 2014))**.** *Let $\mathcal{M}$ be an MDP. Let the state space $\mathcal{S}$ be discrete, and suppose that the supremum $V^\star(s) = \sup_{a \in \mathcal{A}} Q^\star(s, a)$ is attained $\forall s \in \mathcal{S}$. Then:*

1. *There exists a deterministic Markovian, stationary optimal policy according to Definition 2.9;*
2. *$\pi^\star \in \Pi$, greedy w.r.t. $Q^\star$ is an optimal policy.*

As a consequence of the theorem, a greedy policy w.r.t. $Q^\star$ allows us to retrieve an optimal policy, which is stationary and it may not be unique. Moreover, optimal stochastic policies may exist. In the case of a continuous state space this theorem does not hold, and an optimal policy may even not exist (Bertsekas and Tsitsiklis, 2008).

2.4 Dynamic Programming

In this section, we consider methods aimed to find the optimal policy in *finite* MDPs, in the case of perfect knowledge of the transition kernel P and of the reward model R. These methods take into account the Bellman Equations defined in the previous sections to exploit the *dynamic programming* principle (Bertsekas and Tsitsiklis, 1996). They consist in iterative approaches that convert Bellman Equation to update rules, leveraging the contraction properties to grant convergence to optimal solutions. The transition or the reward model of the MDP is seldom known, hence dynamic programming methods cannot be usually applied, and we need to resort to Reinforcement Learning algorithms, introduced in the next chapter.

[5] In Puterman 2014 a greedy policy is denoted as *conserving*.

2.4.1 Value Iteration

One of the most straightforward approaches to solve a finite MDP is Value Iteration (VI, Bellman et al. (1957)). The general idea is, starting from an arbitrary value function V_0, to iteratively apply the Bellman Optimality Operator $T^\star$ defined in Equation 2.13; hence, at iteration $k = 0, 1, \dots$, the next value function is simply computed as:

$$V_{k+1} = T^\star V_k. \tag{2.24}$$

In this way, a sequence of value functions is produced, and the algorithms are stopped when

$$\|V_{k+1} - V_k\|_\infty \leq \epsilon \frac{1-\gamma}{2\gamma}. \tag{2.25}$$

At this point, the greedy policy $\overline{\pi}$ related to the last value function V_T is retrieved, as follows:

$$\overline{\pi}(s) \in \arg\max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim P(\cdot|s,a)} [V_T(s')] \right). \tag{2.26}$$

The VI algorithm is then granted to converge to $V^\star$ thanks to the following Theorem:

Theorem 2.16 (Theorem 6.3.1, Puterman 2014)**.** *Let $\mathcal{M}$ be an MDP. Let $V_0 : \mathcal{S} \to \mathbb{R}$ be an arbitrary measurable bounded function, and let $\{V_n\}$ satisfy 2.24. Then, for all $\epsilon > 0$:*

1. *V_n converges to $V^\star$ in L_∞ norm.*
2. *There exists a finite N from which Equation 2.25 holds for all $n \geq N$;*
3. *The stationary greedy policy $\overline{\pi}$ defined in Equation 2.26 is ϵ-optimal, i.e.:*

$$V^{\overline{\pi}}(s) \geq V^\star(s) - \epsilon \;\; \forall s \in \mathcal{S}.$$

4. *$\|V_{n+1} - V^\star\|_\infty < \epsilon/2$ whenever 2.25 holds.*

The fundamental concept leading to the proof of the previous theorem is the contraction property of the operator $T^\star$, thanks to which:

$$\|V_t - V^\star\|_\infty \leq \gamma \|V_{t-1} - V^\star\|_\infty.$$

Consequently, VI has *linear* convergence with rate γ (Theorem 6.3.3, Puterman 2014):

$$\|V_T - V^\star\|_\infty \leq \frac{\gamma^T}{1-\gamma}\|V_1 - V_0\|_\infty \leq \frac{2\gamma^T}{(1-\gamma)^2} R_{max}.$$

2.4.2 Policy Iteration

VI algorithm does not need an explicit notion of policy for the intermediate value functions computed in the process; an alternative approach, instead, starts from an arbitrary policy π_0 and iteratively improves it until $\epsilon-$convergence to the optimal policy. This is the case of Policy Iteration (PI, Howard (1960)), which is decoupled into two alternate steps:

- *Policy Evaluation*: a *prediction* task, aimed at computing the action value function Q^π related to a stationary policy $\pi \in \Pi$;

- *Policy Improvement*: a *control* task, consisting in the extraction of a greedy policy π' from Q^π.

In the *Policy Evaluation* step, the value function Q^π of a given policy $\pi \in \Pi$ is computed by means of the Bellman Expectation Operator (Definition 2.7): at each iteration $k =, 1, \dots$ the next value function is computed as:

$$V_{k+1} = T^\pi V_k. \tag{2.27}$$

In this way, a sequence of value functions is produced, and the algorithm is stopped when $\|V_{k+1} - V_k\| \leq \epsilon(1-\gamma)$, meaning that an $\epsilon-$approximation of the fixed point of the Bellman Equation is found, which is the value function V^π:

$$\|V_k - V_{k-1}\|_\infty \leq \epsilon(1-\gamma) \implies \|V_k - V^\pi\|_\infty \leq \epsilon$$

Another way of computing the value function involves solving a linear system of Bellman Equations extracted from Equations 2.15 and 2.16 in the case of a finite MDP, which can be solved in closed form with a potentially expensive computational time (Sutton and Barto, 2018). In other variants of the algorithm, e.g. *modified policy iteration* (Puterman and Shin, 1978), the repetition of Equation 2.27 is iterated for a fixed amount T since it is not strictly necessary to wait to get the best possible approximation of the value function V^π.
The *policy improvement* step returns instead the policy that maximizes the utility estimated from the last value function obtained, i.e. the greedy policy $\overline{\pi}$ w.r.t. V^π as in Equation 2.26. This procedure allows to obtain a non-decreasing performance for each complete iteration of PI algorithm: indeed, the following result holds:

Theorem 2.17 (Policy Improvement Theorem, Sutton and Barto 2018)**.** *Let $\mathcal{M}$ be an MDP. Let $\pi, \overline{\pi} \in \Pi$ be two stationary, Markovian policies. If it holds that*

$$\int_{\mathcal{A}} \overline{\pi}(\mathrm{d}a|s) Q^\pi(s,a) \geq V^\pi(s) \quad \forall s \in \mathcal{S};$$

then:

$$V^{\overline{\pi}}(s) \geq V^\pi(s) \quad \forall s \in \mathcal{S}.$$

Trivially, the condition is satisfied by construction from a greedy policy, hence the theorem is valid, granting to obtain an overall policy improvement (Theorem 6.4.1, Puterman 2014). Finally, if we find out that the greedy policy $\overline{\pi}$ has the same value function as π, i.e. $\|V^{\overline{\pi}} - V^\pi\|_\infty \leq \epsilon$, it means that the value function is the fixed point of the Bellman Optimal Equation: the value function obtained is hence the optimal value function $V^\star$, and the related policy is the optimal policy $\pi^\star$. Despite an increased computational cost per iteration than VI, due to an explicit computation of the policies, the rate of convergence for PI is *quadratic* (Mansour and Singh, 1999). Moreover, since the number of deterministic policies in a finite MDP is finite (at most $|\mathcal{S}|^{|\mathcal{A}|}$), it can be shown that PI converges to the optimal policy in a finite number of iterations (at most $\mathcal{O}\big(\frac{|\mathcal{A}|}{1-\gamma} \log \frac{1}{1-\gamma}\big)$ (Scherrer, 2013; Ye, 2011)).

2.4.3 Linear Programming Method

Solving a finite MDP under the perfect knowledge of the transition and reward models can be coped with *linear programming*: indeed, the optimality conditions obtained from the Proposition 2.12 can be formulated as a linear problem:

$$\begin{aligned}
\min_{V \in \mathbb{R}^{|\mathcal{S}|}} \quad & \sum_{s \in \mathcal{S}} \rho(s) V(s) \\
s.t. \quad & V(s) \geq r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s') \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A},
\end{aligned}$$

where $\rho \in \Delta_{\mathcal{S}}$ is a distribution over the state space $\mathcal{S}$, such that $\rho(s) > 0 \; \forall s \in \mathcal{S}$. This is a linear problem with $|\mathcal{S}|$ variables, and $|\mathcal{S}||\mathcal{A}|$ constrains, which admits as solution $V^\star$. As many cases in LP problems, *dual* formulation is more tractable (Wang et al., 2007):

$$\begin{aligned}
\max_{\nu \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}} \quad & \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} r(s,a) \nu(s,a) \\
s.t. \quad & \sum_{a \in \mathcal{A}} \nu(s',a) = \rho(s') + \gamma \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \nu(s,a) P(s'|s,a) \quad \forall s' \in \mathcal{S}, \\
& \nu(s,a) \geq 0 \quad \forall s \in \mathcal{S}, \; \forall a \in \mathcal{A}.
\end{aligned}$$

This formulation is usually preferred, as it is an LP problem with $|\mathcal{S}||\mathcal{A}|$ variables, $|\mathcal{S}|$ constraints, and $|\mathcal{S}||\mathcal{A}|$ non-negativity conditions. Its solution $\nu^\star$ represents the occupancy measure $\nu^{\rho}_{\pi^\star}$, as in Equation 2.9, induced by initial distribution ρ and the optimal policy $\pi^\star$. Consequently, it is possible to retrieve the optimal policy from the primal LP problem as the greedy policy w.r.t. $V^\star$, or from the optimal distribution of the dual as:

$$\pi^\star(a|s) = \frac{\nu^\star(s,a)}{\sum_{a' \in \mathcal{A}} \nu^\star(s,a')}.$$

This linear programming formulations can be solved in polynomial time, and their worst-case convergence guarantees are better than the ones achieved by VI and PI. However, it is empirically observed that Dynamic Programming methods usually tend to perform better(Littman, 1996).

CHAPTER 3

Reinforcement Learning

3.1 Introduction

Dynamic Programming methods introduced in Chapter 2 provide useful approaches for solving Sequential Decision Processes under two strong assumptions: full knowledge of the transition and reward models and finite state and action spaces (with extensions to compact spaces (Puterman, 2014)). Unfortunately, in real-world problems, they do not hold, as the state spaces are usually continuous and too large to be solved with such techniques. Moreover, the exact dynamics of the process are seldom known or uncertain. Hence, these problems need to be tackled using RL approaches since they aim to find the optimal policy without prior knowledge of the model. Therefore, it becomes essential for the agent to collect information on the world the agent is interacting with, as *sampling* is necessary to build statistical models to *predict* the value of the agent's behavior, and to *control* the interaction to maximize the returns.

Uncertainty and stochasticity in the process are important, and to make improvements, it is fundamental to *explore* unseen regions of the state and action spaces. Once the available knowledge is rich enough to understand how to perform well, it is possible to *exploit* it to improve the policy. RL algorithms need to deal with a trade-off between the two concepts in the so-called *exploration-exploitation dilemma*.

A further building block for RL algorithms is the concept of *function approximation*: dynamic programming techniques fail in the case of a large (and finite) state or action space because the computational time and the memory required to store information and update the value functions become unfeasible: this problem, known as *the curse of dimensionality* (Szepesvári, 2010; Sutton and Barto, 1998) can only be dealt with function approximation, also needed in the case of continuous spaces.

All these concepts and tasks are tackled with completely different approaches, and this chapter is meant to provide a brief overview of several methodologies, useful for a deeper understanding of the subsequent parts of this dissertation. Of course, providing a complete and exhaustive presentation is impossible, for which we refer the reader to (Sutton and Barto, 1998; Szepesvári, 2010).

Chapter Outline This chapter is organized as follows: in Section 3.2, we review some of the most important dichotomies that categorize the several approaches adopted. The most important classification distinguishes the goal of the algorithm: if the goal is to refine the prediction of the expected returns of a policy, the task is denoted as *policy evaluation*, and the related techniques are discussed in Section 3.3. The remaining algorithms are aimed at *optimizing* the policy: this task can be dealt with by means of an explicit model of the value functions, leading to *value based* approaches (Section 3.4), or thanks to a parametrization of the policy. The techniques in this category are referred to as *policy-search* approaches, or *policy based* and are discussed in Section 3.5. Often, algorithms combine the two approaches, leading to *actor-critic* methods.

3.2 A Taxonomy for Reinforcement Learning Algorithms

RL literature has provided a plethora of different approaches, which can be categorized according to a variety of criteria. In this section, we try to enlist and describe some of the most important dichotomies:

- *Objective*. The first distinction follows the same path traveled by dynamic programming: as mentioned in the previous section, there are two main tasks that RL algorithms aim to solve:

  - *Prediction*, or Evaluation, is the task of estimating the performances of a given policy. These approaches mirror the Policy Evaluation step introduced in section 2.4.2.
  - *Control*, or Optimization, directly follows policy iteration and value iteration methods in dynamic programming, as it consists in solving an MDP by finding the best-performing policy.

  In this manuscript, we will mainly focus on the optimization task, although the estimation of the value function of a policy is a fundamental concept in all RL. The vast majority of control-based approach in RL literature is focused on optimizing the expected return, albeit several branches and research fields are involved with optimizing other objective functions. As a simple example, Risk-Averse Reinforcement Learning (RARL) deals with the optimization of specific risk measures related to the worst possible outcomes of the learned policy (Artzner et al., 1999; Chow et al., 2015; Bisi et al., 2020b).

- *Model Estimation*. In the absence of perfect knowledge of the MDP, sampling is necessary to collect information on the environment. The object learned with this information can differ:

  - the collected samples can be used to create an explicit model of the decision process, upon which dynamic programming techniques can be directly applied. This is the case of *model-based* algorithms (Deisenroth and Rasmussen, 2011; Nagabandi et al., 2018), which lie beyond the purpose of this dissertation. For additional information, the reader can refer to extensive surveys on Model-based algorithms in (Nguyen-Tuong and Peters, 2011; Polydoros and Nalpantidis, 2017).
  - A large portion of RL algorithms do not build an explicit model and the transition or the reward processes but directly try to optimize the value functions and/or the policies. For this reason, they are classified as *Model-free* approaches (Sutton and Barto, 1998; Szepesvári, 2010), which will be extensively adopted in the next chapters.

- *Object Estimator*. Among the model-free approaches, it is possible to distinguish the object which is iteratively optimized:

  - in a similar fashion as in VI described in section 2.4.1, in *Value-based* approaches (Watkins and Dayan, 1992; Scherrer, 2013; Mnih et al., 2015), the goal is to learn the optimal (action-)value function, from which the greedy, optimal policy is extracted. These methods are further discussed in section 3.4.
  - In other cases (Williams, 1992; Baxter and Bartlett, 2001; Schulman et al., 2015; Papini et al., 2017), the optimized object is the policy itself, which is parametrized and updated through gradient ascent. These are denoted as *Policy-Based* methods and will be analyzed later in section 3.5.
  - *Actor-Critic* algorithms (Peters et al., 2005; Lillicrap et al., 2015) combine the two approaches, as the optimization is performed on both the *actor* (the policy) and its related value function estimation (the *critic*).

- *Interaction*: as said, the agent needs to collect some samples from the interaction with the environment to gather useful information. The collected dataset is supposed to be adequate to learn the optimal policy and is adopted throughout the learning process. This is the case of *offline* or *batch* algorithms (Thomas et al., 2015; Ernst et al., 2005). In other cases, the last chosen policies might reach regions in state-action space that were not explored enough in the initial sampling process. Hence *online* approaches allow the agent to learn while interacting with the environment and collecting new samples (Watkins and Dayan, 1992; Schulman et al., 2017).

- *Sampling policies*. The data used to learn is collected by the interaction of a specific policy with the environment, denoted as *behavioral* policy. This policy might differ from the one that is optimized (referred to as *target* policy), as one might choose, for instance, more *exploratory* behavior to gather more useful information. This is the case of *off-policy* RL (Watkins and Dayan, 1992; Silver et al., 2014). Otherwise, the target and behavioral policies coincide, and the method is classified as *on-policy* (Williams, 1992; Schulman et al., 2015).

3.3 Policy Evaluation

In this section, we briefly review some of the main methods for prediction in a model-free scenario, with no claim to be exhaustive but only ancillary for understanding the control methods that will be analyzed later, as they represent the main focus of this dissertation. For a more complete coverage of the topic, we suggest the reader see Sutton and Barto 2018.

The policy evaluation task consists of best approximating the value function V^π related to a given policy π in an unknown MDP. In other terms, the objective can be formulated as the minimization of the Root Mean Squared Error (RMSE) $\mathcal{L}$ between the true value function V^π and its approximation $\widehat{V}$ with respect to the state-space occupancy measure δ_π^μ:

$$\mathcal{L}(\widehat{V}) := \left(\mathop{\mathbb{E}}_{s \sim \delta_\pi^\mu} \left[(V^\pi(s) - \widehat{V}(s))^2 \right] \right)^{1/2} \tag{3.1}$$

In a more general direction, we can consider the L_p norm under a general probability measure ρ, with $p \in [1, \infty)$:

$$\mathcal{L}_{p,\rho}(\widehat{V}) := \|V^\pi - \widehat{V}\|_{p,\rho} = \left(\int_S \rho(\,\mathrm{d}s)(V^\pi(s) - \widehat{V}(s))^p \right)^{1/p},$$

If the state space is *discrete* the value function can be represented as a vector (or a tensor), and the *tabular* representation of the value function for each state is directly minimized. In this setting, the minimization of Equation 3.1 is equivalent to minimizing the L_1 norm of the vector $\boldsymbol{V}^\pi - \widehat{\boldsymbol{V}}$,[1] under the assumption that the induced Markov chain with transition p^π is *irreducible*.
In continuous state spaces, there is no tabular representation: *function approximation* is needed; hence the value functions are approximated through a function space $\mathcal{F}$, which can be either parametric or non-parametric. In the former case, the value function is represented through a set of parameters $\boldsymbol{\omega}$ belonging to a *parameter space* $\Omega \in \mathbb{R}^d$, for some dimension $d > 0$. Given a parametrization, $V^{\boldsymbol{\omega}} : \mathcal{S} \to \mathbb{R}$ represents an approximation of the target value function V^π, and the objective in Equation 3.1 is reformulated, as $\min_{\boldsymbol{\omega} \in \Omega} \mathcal{L}(V^{\boldsymbol{\omega}})$. Often it is required for the optimization that $V^{\boldsymbol{\omega}}$ is differentiable w.r.t. $\boldsymbol{\omega}$.
A direct generalization of the tabular setting previously mentioned can be performed utilizing a linear approximation as:

$$V^{\boldsymbol{\omega}}(s) = \boldsymbol{\omega}^\top \phi(s), \tag{3.2}$$

where $\phi : \mathcal{S} \to \mathbb{R}^d$ is an appropriate feature representation function. *Linear* MDPs with matching features (Cai et al., 2020) are the only framework where the linear approximation is *exact*, i.e., there exists some feature vector such that the value function perfectly matches the target value function V^π. In all the other cases, as with general function approximations, one has to deal with the *bias* introduced with the considered parameter space Ω:

$$Bias(\Omega) := \min_{\boldsymbol{\omega} \in \Omega} \mathcal{L}(V^{\boldsymbol{\omega}})$$

[1]in this case, $\boldsymbol{V}$ denotes the vector representation of $V(s)$ for each state $s \in \mathcal{S}$.

This bias represents the function-class error, i.e., the distance between the target value function V^π and the space of value functions that can be represented, denoted as $\mathcal{V}^\Omega := \{V^{\boldsymbol{\omega}} : \boldsymbol{\omega} \in \Omega\}$. If $V^\pi \in \mathcal{V}^\Omega$, the bias is zero, and the approximation of the target value function is said to be *realizable* in the function-class $\mathcal{V}^\Omega$.
The main approaches to perform Policy Evaluation are *Monte Carlo* approximations or involve *Temporal-Difference* learning.

3.3.1 Monte Carlo Prediction

The term *Monte Carlo* (Robert et al., 1999) is used in statistics to denote any estimation method which relies on averaging repeated random samples. We recall that the value function of a state $s \in \mathcal{S}$ is defined as the return obtained in expectation starting from the same state s and then following the policy π. As a consequence, a straightforward estimator of such a value can be obtained from the average of independent realizations of the process, under the assumption that experience is *episodic*, and an absorbing state is reached almost certainly in finite time. Therefore, we can collect a batch of N trajectories $\{\tau^{(n)}\}_{n=1}^N$ such that the initial state s_0^n in each trajectory coincides with the goal state s, and consider as estimator for such state:

$$\widehat{V}_N^\pi(s) := \frac{1}{N}\sum_{n=1}^N G(\tau^{(n)}) = \frac{1}{N}\sum_{n=1}^N \sum_{t=0}^{T(\tau^{(n)})} \gamma^t r_t. \tag{3.3}$$

Each return $G(\tau^{(n)})$ starting from state s in an unbiased estimator, therefore each average built from i.i.d. trajectories is an unbiased estimator: $\mathbb{E}\left[\widehat{V}_N^\pi(s)\right] = V^\pi(s)$. Moreover, $\widehat{V}_N^\pi(s) \xrightarrow{a.s.} V^\pi(s)$ in the limit for $N \to \infty$ and the variance of the estimate is inversely proportional to N according to the central limit theorem. Sampling a different batch of trajectories starting from each state s is very sample inefficient, and one can consider sub-trajectories transiting in such state s, even if the initial point was different: in this case, we can consider the *first passage time* t_s for a trajectory τ, defined as:

$$t_s := \inf\{t \in T(\tau) : s_t = s\}, \tag{3.4}$$

Consequently, we can still have an unbiased and consistent estimate for $V^\pi(s)$ by considering the average of the partial returns $G(\tau_{t_s}^{(n)})$, related to the sub-trajectories starting from $\{\tau_{t_s}^{(n)}\}$ for each n in the batch: this approach is denoted as *first-visit* Monte Carlo estimator. This estimator, albeit unbiased, is affected by high variance; hence convergence is very slow (Kearns and Singh, 2000). Sometimes, it is more efficient to consider the *every-visit* variant, for which the estimators consider the partial returns for each occurrence of the target state. Since there is the possibility of having more values for the same trajectory, the estimator is biased but still consistent.

To perform the updates in a more computationally efficient manner, e.g., without the need to store the whole sets or partial returns, the estimations can be updated with *incremental updates*.

$$\begin{aligned}\widehat{V}_N^\pi(s) &= \widehat{V}_{N-1}^\pi(s) + \alpha_N(G(\tau^{(N)}) - \widehat{V}_{N-1}^\pi(s)) \\ &= (1-\alpha_N)\widehat{V}_{N-1}^\pi(s) + \alpha_N G(\tau^{(N)}),\end{aligned} \tag{3.5}$$

where α_N is the *learning rate* or *stepsize*, and $G(\tau^{(N)})$ is the latest partial return observed in the N-th trajectory starting from state s. In the case $\alpha_N = 1/N$, the estimation turns out to be equivalent to the empirical mean in equation 3.3, but one can use different learning rate schedules, and the estimators is still granted to be consistent ($\widehat{V}_N^\pi \xrightarrow{a.s.} V^\pi(s)$) if the step-sizes satisfy the *Robbins-Monro* conditions (Robbins and Monro, 1951):

$$\sum_N \alpha_N = \infty, \qquad \sum_N \alpha_N^2 < \infty. \tag{3.6}$$

3.3.2 Temporal Difference Learning

The main alternative to Monte Carlo is Temporal Difference Learning (TD, Sutton and Barto (1998)), which tries to address the problem of high variance by means of truncation in the considered history of future rewards. In this way, the updates are performed in an *online* manner without the need to wait until the end of the episode. For this reason, TD learning can also be applied in the case of non-episodic interactions with the environment.

TD can be thought of as an approximation to the application of the Bellman Equality in Equation 2.16, as the core idea consists in minimizing the discrepancy between the value estimation in a state and the result of the Bellman Operator. This discrepancy can be mathematically framed as the Bellman Error (BE) defined, for an estimator $\widehat{V}$, in the state s, as

$$BE(s) = \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)} \left[r(s,a) + \gamma \mathop{\mathbb{E}}_{s'\sim P(\cdot|s,a)} [\widehat{V}(s')] \right] - \widehat{V}(s). \tag{3.7}$$

The goal of the evaluation problem can then be considered as the minimization of the BE since, by considering the Bellman Equations, in the case $\widehat{V} = V^\pi$ the error is uniformly zero in the whole state space. To minimize the BE, TD evaluates the mismatch in the encountered sub-trajectories: the samples collected, denoted as (1-step) *transitions*, are in the form (s_t, a_t, r_t, s_{t+1}), and the related Temporal Difference Error (TDE) δ_t is computed as:

$$\delta_t := r_t + \gamma\widehat{V}(s_{t+1}) - \widehat{V}(s_t). \tag{3.8}$$

Trivially, the TDE represents a single realization to estimate the BE, as:

$$\mathop{\mathbb{E}}_{\substack{a_t\sim\pi(\cdot|s_t)\\ s_{t+1}\sim P(\cdot|s_t,a_t)}} [\delta_t] = BE(s_t).$$

Starting from a given estimation $\widehat{V}^{(0)}$, the well-known TD(0) method employs an iterative correction according to the following update rule:

$$\widehat{V}^{(t+1)}(s) = \widehat{V}^{(t)}(s) + \alpha_t\delta_t, \tag{3.9}$$

where $\alpha_t > 0$ is the *learning rate*. This update rule can be easily seen as

$$\widehat{V}^{(t+1)}(s_t) = (1-\alpha_t)\widehat{V}^{(t)}(s_t) + \alpha_t \underbrace{\left[r_t + \gamma\widehat{V}^{(t)}(s_{t+1})\right]}_{TD \text{ target}}.$$

It is clear that this approach takes into account previous estimates to perform the online updates: this process is denoted as *bootstrap*. Furthermore, the TDE boils down to make updates in the direction of the TD target, which is the (one-step) estimation of the return $r_t + \widehat{V}^{(t)}(s_{t+1})$. This term, unless $V^{(t)} = V^{\pi}$, is a biased estimate of $V^{\pi}(s_t)$, but accounts for a lower variance than the Monte Carlo approach, because its stochasticity is only dependent on a single transition instead of a whole trajectory. Furthermore, the estimator is still consistent under the Robbins-Monro conditions (equation 3.6). As a final comparison, while TD is usually more efficient than Monte-Carlo, its sensitivity to the initial guess for the value function is larger.

Monte-Carlo and TD learning trade-off: TD(n) and TD(λ). Both Monte-Carlo and TD learning have a similar form of update rule, which involves a different estimator of the value function, said $G^{(t)}$, as

$$\widehat{V}^{(t+1)}(s_t) = (1-\alpha_t)\widehat{V}^{(t)}(s_t) + \alpha_t G^{(t)}, \tag{3.10}$$

where $G^{(t)}$ represents the return of a trajectory from Monte-Carlo value estimation or the TD target in TD(0) learning. To fill the gap between the two approaches, instead of simply considering one-step transitions, one can resort to updating the estimates using longer sequences of states and rewards encountered within a trajectory: the resulting approach, denoted as TD(n), performs updates in the direction of the n-step TD target $G_n^{(t)}$, i.e., the discounted sum of n rewards, and the expected return bootstrapped at the n-th next state. In mathematical terms, the algorithm considers a history $(s_t, a_t, r_t, s_{t+1}, \ldots, r_{t+n-1}, s_{t+n})$, and the update assumes the following form:

$$\widehat{V}^{(t+1)}(s_t) = (1-\alpha_t)\widehat{V}^{(t)}(s_t) + \alpha_t \underbrace{\Big[\sum_{i=0}^{n-1}\gamma^i r_{t+i} + \gamma^n \widehat{V}^{(t)}(s_{t+n})\Big]}_{=:G_n^{(t)}}.$$

A further trade-off between Monte-Carlo and TD involves the exponential average among TD targets with different horizons to obtain TD(λ) algorithm; in this case, a new parameter $\lambda \in [0,1]$ is included, and the resulting estimator takes the following form:

$$G_\lambda^{(t)} := (1-\lambda)\sum_{n=1}^{\infty} G_n^{(t)}.$$

Depending on λ, this approaches ranges from TD(0) for $\lambda = 0$ to Monte-Carlo ($\lambda = 1$) but, as in the latter case, updates can be performed only at the end of an episode. *Eligibility traces* (Singh et al., 2000; Sutton and Barto, 1998) can be adopted to overcome this issue and to weigh the updates properly.

Action Value Prediction In the previous sections, we presented several techniques aimed at learning the best possible approximation of the value function V^{π} of a policy π. The prediction task can be easily extended to learn action-value functions $Q^{\pi}(s,a)$ for all state-action pairs since the same update rule adopted in equation 3.10 can be applied to quality functions, with the purpose of minimizing the mismatch between the current solution $\widehat{Q}^{(t)}$, and an estimator of the expected return starting from each state-action pairs. Trivially, the partial return $G(\tau^{(N)})$ collected in the trajectory $\tau^{(N)}$ starting from the pair (s_t, a_t) is an unbiased estimate of $Q^{\pi}(s_t, a_t)$, hence the Monte-Carlo approach can be directly applied. A small modification is required for TD(0) since the transitions needed to perform an update require the observation of the next chosen action a_{t+1}. Thus, the TD error is modified accordingly:

$$\delta_t = r_t + \gamma \widehat{Q}^{(t)}(s_{t+1}, a_{t+1}) - \widehat{Q}^{(t)}(s_t, a_t). \tag{3.11}$$

Prediction with Function Approximation In the introduction of Policy Evaluation in Section 3.3, an important building block for RL was mentioned: *function approximation*. When state and action spaces are continuous, algorithms must be modified to account for the parametrization $\boldsymbol{\omega}$ of the (action) value functions. Therefore, letting $V^{\boldsymbol{\omega}_t}$ be the approximation of V^{π} at step t, the general update rule in equation 3.10 is modified as follows:[2]

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha_t \big(G^{(t)} - \widehat{V}^{\boldsymbol{\omega}_t}(s_t)\big) \nabla_{\boldsymbol{\omega}} \widehat{V}^{\boldsymbol{\omega}_t}(s_t). \tag{3.12}$$

The Monte-Carlo version of the algorithm is then an instance of Stochastic Gradient Descent (SGD), where the loss function is the Mean Squared Error (MSE) from equation 3.1, while the TD approach follows only a *semi-gradient* update since the TDE is considered as constant instead of dependent on the current parametrization $\boldsymbol{\omega}_t$ in the computation of the gradient. Convergence to a *local minimum* of Monte-Carlo can be proved under Robbins-Monro conditions for the learning rate and regularity assumptions for $V^{\boldsymbol{\omega}}$ (Bottou et al., 1998). *Global minimum* can be provably reached in the case of linear function approximation (3.2) if the approximation is realizable. For what concerns TD learning, due to *bootstrapping*, convergence is guaranteed only for linear approximations with Robbins-Monro step-sizes (Tsitsiklis and Van Roy, 1996).

3.4 Value-Based Control

In this section, we move from the prediction task to control. While *policy-based* algorithms directly optimize a parametric representation of the policy (section 3.5), *Value-based* approaches retrieve an approximation of the optimal policy starting from an estimation of the optimal value function. In this sense, these methods are based upon estimations of the Q-value function to have the possibility to extract its greedy policy. In the following, we introduce some fundamental value-based control algorithms, *SARSA* and *Q-Learning*, which respectively represent the TD generalizations of policy iteration 2.4.2 and value iteration 2.4.1. They focus on learning the optimal quality function $Q^{\star}(s, a)$ for all state-action pairs in a tabular setting. The adopted algorithms must once again deal with function approximation to deal with realistic settings. In section 3.4.3, we introduce some important approaches that will extensively be adopted in the rest of this dissertation.

Notation For ease and clarity, in the following sections, we will drop the notation $\widehat{Q}$ to denote an estimation of the value function in favor of a simple Q term whenever clear from the context.

[2]under the assumption that $V^{\boldsymbol{\omega}}$ is differentiable w.r.t. ω for all $\boldsymbol{\omega} \in \Omega$.

3.4.1 SARSA

One of the most adopted value-based algorithms is *SARSA* (Rummery and Niranjan, 1994). It is the RL analogous of Policy Iteration, as it comprises two main steps: Policy Evaluation and Policy Improvement.

Policy Evaluation In this case, there is no knowledge of the dynamics of the environment; thus Policy Evaluation step is performed by means of TD learning. This algorithm is *on-policy*; hence it performs updates using the experience gathered by following the same policy being learned. Formally, at iteration t an interaction tuple $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ is collected, and the Q value function is modified according to the general TD update rule:

$$Q^{(t+1)}(s_t, a_t) = Q^{(t)}(s_t, a_t) + \alpha_t \delta_t,$$

where δ_t is the TD error defined in Equation 3.11. As with the prediction task, it is possible to consider longer trajectories in the update rule to obtain *SARSA*(n) in the same fashion as with TD(n), or to adopt *eligibility traces* (Van Seijen et al., 2016). A further refinement allows reducing the variance related to the next action a_{t+1} by computing an expectation w.r.t. the current policy. The resulting algorithm, named *Expected SARSA*, considers then the following TD error:

$$\delta_t = r_t + \gamma \mathop{\mathbb{E}}_{a' \sim \pi(\cdot|s_{t+1})} \left[Q^{(t)}(s_{t+1}, a') \right] - Q^{(t)}(s_t, a_t).$$

Policy Improvement As in Policy Iteration, the first step allows to obtain an estimation of the value function of a policy π_t. In Dynamic Programming, the Policy Improvement step involves then the extraction of the greedy-policy w.r.t. $Q^{(t)}$ to retrieve a better-performing policy π_{t+1}. If the information on the environment is only provided through empirical interactions, greedy policies are not the best choice, as they perform exploitation of the limited experience collected, which might be insufficient. More exploration is needed, and literature provided different solutions to balance the exploration-exploitation trade-off.

The most adopted techniques do not account for uncertainty in the estimations to drive the exploratory behavior: for this reason, they are denoted as *undirected* exploration strategies. In the case of a discrete action space, the most adopted solution is the $\epsilon-$greedy policy:

Definition 3.1 (ϵ-Greedy Policy)**.** *Let $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be a bounded, measurable function, and let $\epsilon \in [0, 1]$. A policy $\pi \in \Pi$ is denoted as ϵ-greedy with respect to f if, for all $(s, a) \in \mathcal{S} \times \mathcal{A}$:*

$$\pi(a|s) = \begin{cases} \frac{\epsilon}{|\mathcal{A}|} + 1 - \epsilon & \textit{if } a = \arg\max_{a \in \mathcal{A}} f(s, a), \\ \frac{\epsilon}{|\mathcal{A}|} & \textit{otherwise}. \end{cases} \tag{3.13}$$

In brief, the greedy action is chosen with probability $1 - \epsilon$; otherwise, the action is sampled uniformly in the whole action space. In later sections, we will refer to an ϵ-greedy policy w.r.t. a value function Q as π_Q^ϵ. If $\epsilon = 0$, it reduces to the greedy policy.

The main drawback of ϵ-greedy policies is that they do not leverage the information available to the agent to drive the distribution over the actions. The estimations of the value functions are instead considered in the case of *Boltzmann* policies (or *Softmax*):

Definition 3.2 (Boltzmann Policy). *Let $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be a bounded, measurable function, and let $\tau > 0$. A policy $\pi \in \Pi$ is denoted as Boltzmann with respect to f if, for all $(s, a) \in \mathcal{S} \times \mathcal{A}$:*

$$\pi(a|s) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_{a' \in \mathcal{A}} e^{\frac{Q(s,a')}{\tau}}}. \tag{3.14}$$

The parameter τ, called *temperature*, controls the exploration rate, and the greedy policy is the degenerate case for $\tau \to 0$. Clearly, this technique can be extended for continuous action spaces. Convergence to the optimal policy is granted for *SARSA* if the exploration policies are *GLIE* (Greedy in the Limit with Infinite Exploration), i.e., all state-action pairs are visited an infinite number of times and, in the limit for $t \to \infty$, they tend to the greedy policies:

Theorem 3.3 (Convergence of SARSA, Theorem 1 (Singh et al., 2000)). *Let $\mathcal{M}$ be a discrete MDP, with the Q-values stored in a lookup table. If:*

1. *each action is executed infinitely often in every state, which is visited infinitely often;*
2. *in the limit for $t \to \infty$, the learning policy is a.s. greedy with respect to the Q-value function $Q^{(t)}$;*
3. *The step sizes $\{\alpha_t\}_t$ satisfy the Robbins-Monro conditions;*
4. $\mathbb{V}ar[R(s, a)] < \infty \; \forall (s, a) \in \mathcal{S} \times \mathcal{A}$;[3]

then, $Q^{(t)}$ converges to $Q^\star$, and the learning policy π_t converges to $\pi^\star$.

In the case of ϵ-greedy policies, the GLIE assumption hold if $\epsilon \to 0$, while Boltzmann policies require that $\tau \to 0$.

Albeit the optimal policy is guaranteed to be reached, these exploration methods are far from being efficient. *Provable efficiency* holds instead for *directed* methods, which account for uncertainty in the Q value estimation, and formalize exploration in the form of a bonus function (Kearns and Singh, 2002; Strehl and Littman, 2005, 2008; Jin et al., 2020). Often, these methods are *count-based*, meaning that the number of visitations to each state-action pair is explicitly stored. The effect is to direct the agent to explore actions with uncertain utility.

3.4.2 Q-learning

While SARSA is the TD learning version of Policy Iteration, the corresponding algorithm for Value Iteration is Q-learning (Watkins and Dayan, 1992). Unlike SARSA, it is an *off-policy* approach; hence a *behavioral* policy is responsible for exploration, while the value function learned is related to a different policy, in our case, the optimal one. Q-learning relies upon the iterative application of the empirical version of the Bellman Optimality Operator $T^\star$:

[3]this assumption is trivially satisfied under Assumption 2.1.

Algorithm 1 Q-learning

```
Require: Learning rate sequence $\{\alpha_t\}_t$, exploration coefficient $\epsilon$, number of episodes $N$
Ensure: $Q$-function estimation
 1: Initialize $Q^{(0)}$ arbitrarily
 2: for $episode = 1, \dots, N$ do
 3:     $t \leftarrow 0$
 4:     while $s_t$ is not $terminal$ do
 5:         $a_t \sim \pi^{\epsilon}_{Q^{(t)}}(s_t)$
 6:         Take action $a_t$, observe $s_{t+1}, r_t$
 7:         Compute TDE:
                $\delta_t = r_t + \gamma \max_{a \in \mathcal{A}} \{Q^{(t)}(s_{t+1}, a)\} - Q^{(t)}(s_t, a_t)$
 8:         Update $Q$:
                $Q^{(t+1)}(s_t, a_t) = Q^{(t)}(s_t, a_t) + \alpha_t \delta_t$
 9:         $t \leftarrow t + 1$
10:     end while
11: end for
```

Definition 3.4 (Empirical Bellman Optimality Operators)**.** *Let $\mathcal{M}$ be an MDP. Consider a bounded, measurable function $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and a transition (s, a, r, s'). The Empirical Bellman optimality operator $\widehat{T}^{\star}$ is defined as:*

$$(\widehat{T}^{\star} f)(s, a) := r + \gamma \max_{a' \in \mathcal{A}} f(s', a'). \tag{3.15}$$

In this case, convergence to the optimal policy is guaranteed without the need for GLIE policies: the only assumptions, at least in the bounded reward scenario, are infinite visitation of each state-action pair and Robbins-Monro conditions (Singh et al., 2000). The most adopted behavioral policy is the $\epsilon-$greedy policy: the pseudocode of Q-learning, with random exploration ratio driven by ϵ, is reported in Algorithm 1. In Chapter 7, we will deal with an extension of this algorithm by means of action repetition.

3.4.3 Approximate Value Iteration

As presented in section 3.3, function approximation is needed when the environment is complex or continuous. Therefore, we need to resort to a function space $\mathcal{F} \subseteq \mathscr{B}(\mathcal{S} \times \mathcal{A})$ of measurable functions on $\mathcal{S} \times \mathcal{A}$, and the goal is to find the best approximation of the optimal value function within such space:

$$\widetilde{Q} \in \operatorname*{argmin}_{f \in \mathcal{F}} \left\{ \|f - Q^{\star}\|_{p,\rho} \right\},$$

for some distribution ρ over $\mathcal{S} \times \mathcal{A}$, and $p \geq 1$. The case $p = 2$ reduces to the minimization of the RMSE, while for $p = \infty$ we try to minimize the maximum discrepancy between $\widetilde{Q}$ and $Q^{\star}$ on the whole space $\mathcal{S} \times \mathcal{A}$. The solution $\widetilde{Q}$ attains a loss equal to the bias of the function class and allows to extract a greedy policy $\widetilde{\pi}$ whose performances are close to the optimal policy, as in (Singh and Yee, 1994):

$$V^{\widetilde{\pi}}(s) \geq V^{\star}(s) - \frac{2}{1-\gamma} \|\widetilde{Q} - Q^{\star}\|_{\infty} \quad \forall s \in \mathcal{S}.$$

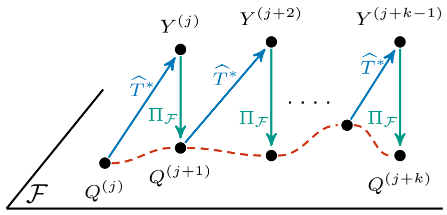


Figure 3.1: *Illustration of AVI. $\widehat{T}^{\star}$ is the empirical Bellman Optimal Operator 3.4. Y are the estimates produced by such operator. The next estimation of the optimal value function is then retrieved through to the projection $\Pi_{\mathcal{F}}$ of the targets Y on the functional space $\mathcal{F}$.*

Function approximation methods can be either applied to extend Policy Iteration or Value Iteration approaches; thus the resulting classes of algorithms are referred to as Approximate Policy Iteration (API, Scherrer (2014)) and AVI. In particular, in the latter family of algorithms, the optimal value function estimations are updated by employing the empirical Bellman Operator $\widehat{T}^{\star}$: this operator is applied on a *batch* of transitions $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_i$, collected following a certain distribution ρ. Eventually, the next estimator of the optimal quality function is retrieved by applying a *projection* (denoted as $\Pi_{\mathcal{F}}$) of the targets onto the function space $\mathcal{F}$. In brief, the update rule can be summarized as $Q^{(t+1)} = \Pi_{\mathcal{F}}\widehat{T}^{\star}Q^{(t)}$. A graphical representation of the update sequence is reported in Figure 3.1. Consequently, AVI methods suffer from two different sources of approximations: the first is the *estimation error* due to the application of the empirical version of the Bellman operator. The second one is generated from the projection procedure, usually performed through least square regression. These two effects generate an overall *approximation error* denoted as ϵ:

$$\epsilon^{(t)} := T^{\star}Q^{(t)} - Q^{(t+1)}.$$

Several contributions in literature (Bertsekas and Tsitsiklis, 1996; Munos, 2005; Antos et al., 2008; Munos and Szepesvári, 2008) provide important theoretical results on convergence properties and error propagation in AVI. In particular, we report a result presented in Farahmand 2011, which studies the propagation of approximation errors to bound the distance between the optimal value function $Q^{\star}$ and the Q-function obtained after T iterations of AVI.

Theorem 3.5 (Error Propagation for AVI. Farahmand 2011, Theorem 3.4)**.** *Let $p \geq 1$. Consider a target distribution $\rho \in \Delta_{\mathcal{S}\times\mathcal{A}}$, a sampling distribution $\nu \in \Delta_{\mathcal{S}\times\mathcal{A}}$. Finally, consider any sequence $(Q^{(t)})_{t=0}^{T} \subset \mathcal{F}$ uniformly bounded by $Q_{\max} \leq \frac{R_{\max}}{1-\gamma}$ generated by performing AVI with a dataset of tuples sampled from ν, and the corresponding sequence of approximation errors $(\epsilon^{(t)})_{t=0}^{T-1}$. Then, for any $r \in [0,1]$, it holds that:*

$$\|Q^{\star} - Q^{\widetilde{\pi}}\|_{p,\rho} \leq \frac{2\gamma}{(1-\gamma)^2}\left[\frac{2}{1-\gamma}\gamma^{\frac{T}{p}}R_{\max} + C_{\mathrm{VI},\rho,\nu}^{\frac{1}{2p}}(T,r)\mathcal{E}^{\frac{1}{2p}}(\epsilon^{(0)},\dots,\epsilon^{(T-1)};r)\right],$$

where the full expressions of $C_{\mathrm{VI},\rho,\nu}(T;r)$ and $\mathcal{E}$ can be found in (Farahmand, 2011).

The term $C_{\mathrm{VI},\rho,\nu}(T;r)$ is a concentrability coefficient taking into account the distribution shift between the sampling distribution ν and the joint distribution obtained from the target ρ and the policies $\{\pi^{(t)}\}_t$. The error term $\mathcal{E}$ instead accumulates the approximation errors and can be further analyzed for specific choices of function spaces.

Algorithm 2 Fitted Q-Iteration

```
Require: $T$ number of iterations, $Q^{(0)}$ initial action-value function,
         $\mathcal{F}$ functional space, $\mathcal{D} = \{(s_i, a_i, s'_i, r_i)\}_{i=1}^{|\mathcal{D}|}$ batch samples
Ensure: greedy policy $\widetilde{\pi}$
1: for $t = 0, \dots, T-1$ do
2:     Build $TS = \{(x_i, y_i)\}_i$:
           $x_i = (s_i, a_i)$
           $y_i = r_i + \gamma \max_{a \in \mathcal{A}} Q^{(t)}(s_i, a)$
3:     perform regression on $TS$ to induce $Q^{(t+1)}$:
           $Q^{(t+1)} \in \operatorname{arginf}_{f \in \mathcal{F}} \|f - y\|_{2,\mathcal{D}}^2$
4: end for
5: Extract greedy policy
           $\widetilde{\pi}(s) \in \arg\max_{a \in \mathcal{A}} Q^{(T)}(s, a), \quad \forall s \in \mathcal{S}.$
```

While a complete analysis of the error propagation of AVI algorithms is in general out of the scope of this dissertation, we will extend and compare the results of Theorem 3.5 in Chapter 6.

A plethora of different algorithms have been developed inside the class of value-based methods with function approximation, especially due to their impressive result in very complex domains (Mnih et al., 2015), or real-world scenarios (Castelletti et al., 2010; Bisi et al., 2020b; Riva et al., 2021). The majority of the literature is devoted to studying discrete action spaces, although several extensions have also been proposed to deal with the continuous setting (Antos et al., 2007; Gu et al., 2016). In the following, we provide further details regarding two practical AVI algorithms: the first is Fitted Q-Iteration (FQI, Ernst et al. (2005)), an offline algorithm adopted multiple times within this dissertation (4 5 and 6), and Deep Q-Network (DQN, Mnih et al. (2015, 2016)), an online method that became very popular for its effectiveness albeit the lack of convergence guarantees. In particular, DQN is parameter-based, as the Q-value function is approximated through a parameter vector $\boldsymbol{\omega}$ in a parameter space Ω. In this case, the general update rule to detect the next Q estimation follows the (semi-)gradient induced by the empirical Bellman optimality operator:

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha_t \left(r + \gamma \max_{a' \in \mathcal{A}} \left\{ Q^{\boldsymbol{\omega}_t}(s, a') \right\} - Q^{\boldsymbol{\omega}_t}(s, a) \right) \nabla_{\boldsymbol{\omega}} Q^{\boldsymbol{\omega}_t}(s, a), \tag{3.16}$$

under the non limiting assumption that the Q-value function is differentiable w.r.t. $\forall \boldsymbol{\omega} \in \Omega$.

FQI Fitted Q-Iteration (Ernst et al., 2005) is an off-policy and offline AVI algorithm. While standard methods adopt least square regression to project the empirical Bellman targets on the selected function space $\mathcal{F}$, the innovative approach of FQI consists of the iterative application of Supervised Learning techniques to provide more refined estimations of the optimal value function. These techniques can be either parametric, as multilayer perceptrons (Riedmiller, 2005), or parameter-free, like the original (Ernst et al., 2005) which adopted a tree-based regression. In our work, we will extensively adopt Extra Trees and Random Forests (Geurts et al., 2006; Breiman, 2001) as regression methods. Being an offline algorithm, FQI considers a dataset $\mathcal{D}$ containing the information collected from experience in the form of transitions. At each iteration of the algorithm, the horizon considered for the Q-value estimation increases by one step: specifically, given $Q^{(t-1)}$, the training set $TS = \{(x_i, y_i)\}_{i=1,\dots,|\mathcal{D}|}$ is built, where each input is equivalent to the state-action pair (i.e., $x_i = (s_i, a_i)$), and the target is the result of the empirical Bellman optimality operator $\widehat{T}^\star$: $y_i = r_i + \gamma \max_{a \in \mathcal{A}} Q^{(t-1)}(s'_i, a)$. In this way, the regression algorithm adopted is trained on TS to learn $Q^{(t)}$, with the goal of minimizing the overall MSE. After T iterations, the greedy policy is extracted to obtain an approximation of the optimal one.

DQN One of the groundbreaking works that brought RL research into the spotlight in recent years is DQN. This algorithm combines the large the powerful approximation capabilities of Deep Learning to address function approximation in RL. Although this Deep Reinforcement Learning (DRL) algorithm does not have robust convergence guarantees, the huge success of DQN is due to its extraordinarily successful application in playing Atari 2600 games from visual input. The most important features, albeit the adoption of convolutional neural networks to deal with image-based state representations, are some tweaks that allow for avoiding divergent learning, a very common and undesired behavior in complex MDPs. The first contribution is the introduction of *target networks*: they are a copy of the network parameters $\boldsymbol{\omega}^-$ that is kept as constant with only a periodic update and used to evaluate the empirical Bellman operator $\widehat{T}^\star$ as follows:

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha_t \left(r + \gamma \max_{a' \in \mathcal{A}} \left\{ Q^{\boldsymbol{\omega}^-}(s, a') \right\} - Q^{\boldsymbol{\omega}_t}(s, a) \right) \nabla_{\boldsymbol{\omega}} Q^{\boldsymbol{\omega}_t}(s, a);$$

in this way, the very frequent changes in the parameters $\boldsymbol{\omega}$ do not show drastic impacts on the TD errors for the next updates, with the result of stabilizing the overall learning process. The second, equally important feature is the adoption of *experience replay buffer*. When the interaction with the environment is performed to collect samples, contiguous transitions within the same trajectories are usually correlated. In the replay buffer, past transitions are stored and, at each iteration, a batch of tuples is randomly sampled; hence the collected data is less correlated. While in the original version of DQN, a uniform distribution is used to extract the tuples in the batch, later implementations adopted *prioritized experience replay* (Schaul et al., 2015), which increases the weight of tuples related to higher TD errors. In Chapter 7 we will introduce a variant of the DQN algorithm to take into account different durations of the actions.

Double Q-learning As the iterative application of the operator $\widehat{T}^\star$ is performed, the sequence of Q-value function approximations can degrade the overall performance of the algorithm. One result regarding error propagation is, for instance, reported in Theorem

3.5. In particular, one of the sources of errors is the overestimation bias due to the computation of the maximum over a noisy Q function. To countermeasure this tendency, (Hasselt, 2010) introduces Double Q-Learning, where the action selection and the value estimation are decoupled through two different value function estimations, $Q_1^{(t)}$ and $Q_2^{(t)}$. In the update step, for each transition (s, a, r, s'), each target for one Q function is computed by considering its greedy action evaluated by the other one:

$$y_1 = r + \gamma Q_1^{(t)}(s', \max_{a\in\ A} Q_2^{(t)}(s', a));$$
$$y_2 = r + \gamma Q_2^{(t)}(s', \max_{a\in\mathcal{A}} Q_1^{(t)}(s', a)).$$

Therefore, the next estimations are induced by performing the projections of the resulting targets over the selected function space. (Fujimoto et al., 2018) suggests instead using the overestimations to build approximate upper bounds to the true value estimate to drive the estimators in the opposite direction. This results in Clipped Double Q-learning, where the targets of the operators are the same for the two estimators:

$$y_1 = y_2 = r + \gamma \max_{a\in\mathcal{A}} \big\{ \min_{i=1,2} Q_i^{(t)}(s', a)\big\}.$$

Another variant is introduced with Dueling DQN (Wang et al., 2016), where the Q-value function is decomposed into the sum of a value function V and of the advantage function A so as to include an inductive bias. The Double Q-learning modification and other refinements were combined and applied on DQN to form *Rainbow* (Hessel et al., 2018), one of the state-of-the-art algorithms for value-based RL.

3.5 Policy Search

In the previous section, we provided some of the most known value-based algorithms, where the main task is the best representation of the optimal value function. When the action space is continuous, the identification of the greedy action can be computationally expensive, even after a fine-grained discretization. One solution consists in adopting *policy optimization* or *policy search* algorithms (Deisenroth et al., 2013): they are designed to search for the optimal solution directly over the space of policies without the need to retrieve the target policy from other learned objects, such as the value function. The vast majority of policy search algorithms belongs to the class of Policy Gradient (PG, Williams (1992); Baxter and Bartlett (2001)) methods, which adopt *parametric policies*: the solution is thus searched within the space Π_Θ of policies that can be represented through a parameter vector $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^m$. For each parameter vector $\boldsymbol{\theta}$, the related policy is denoted as $\pi_{\boldsymbol{\theta}} : \mathcal{S} \to \Delta_{\mathcal{A}}$. An important assumption usually adopted in this framework is that $\pi_{\boldsymbol{\theta}}$ is a stochastic policy and differentiable w.r.t. $\boldsymbol{\theta}$ for all $\boldsymbol{\theta} \in \Theta$.

Notation When parametric policies are considered, we will often abbreviate the dependency on $\pi_{\boldsymbol{\theta}}$ as $\boldsymbol{\theta}$ in function arguments, superscripts, and subscripts. for instance, the state-occupancy measure $\delta^{\mu}_{\pi_{\boldsymbol{\theta}}}$ will be denoted as $\delta^{\mu}_{\boldsymbol{\theta}}$; $Q^{\boldsymbol{\theta}}$ is the short version of $Q^{\pi_{\boldsymbol{\theta}}}$ and $J(\boldsymbol{\theta})$ is the short version of $J(\pi_{\boldsymbol{\theta}})$.

Policy Optimization and Policy Gradient Theorem As said, instead of trying to approximate the optimal value function, Policy Search algorithms directly try to optimize the policy. To do that, a *total order* relation is required among parametrizations. The value function V provides only a *partial order*; hence the optimization is performed on a scalar performance measure, the return $J(\pi)$. Indeed, we can say that a policy π' is better or equivalent than π if $J(\pi') \geq J(\pi)$. Under the assumption that a policy is differentiable w.r.t. the policy parameters $\boldsymbol{\theta}$, then the same property also holds for $J(\boldsymbol{\theta})$. As a straightforward consequence, if we can compute the gradient of the expected return, then we can follow it in a Stochastic Gradient Ascent (SGA) manner to optimize the policy. Fortunately, the Policy Gradient Theorem (PGT) provides a very useful way to compute this gradient term, providing thus a fundamental building block for all policy search methods, for this reason, also denoted as *Policy Gradient* approaches. To provide the reader with the Theorem, we first need to include a useful lemma:

Lemma 3.6 (From Lemma 20 in (Ciosek and Whiteson, 2020))**.** *Let $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$, and let f and g be integrable functions on $\mathcal{S}$ such that, for all $s \in \mathcal{S}$:*

$$f(s) = g(s) + \gamma \int_{\mathcal{S}} p_\pi(s'|s) f(s') \,\mathrm{d}s'.$$

Then:

$$f(s) = g(s) + \frac{\gamma}{1-\gamma} \int_{\mathcal{S}} d_s^\pi(s') g(s') ds'.$$

The importance of this lemma is twofold: firstly, it will be used later to prove the Policy Gradient Theorem 3.8; furthermore, it allows us to completely rethink the expected return in terms of the only reward function and the visitation probability of each state-action pair according to the state-occupancy measure. Indeed, the following corollary holds:

Corollary 3.7. *Let $\mathcal{M}$ be an MDP, and let $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$. Then:*

$$J(\pi) = \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{(s,a)\sim\nu_\pi^\mu} \big[r(s,a)\big]. \tag{3.17}$$

Proof. We start by considering the value function of the policy π

$$\begin{aligned} V^\pi(s) &= \int_{\mathcal{A}} \pi(a|s) Q^\pi(s,a) \,\mathrm{d}a \\ &= \int_{\mathcal{A}} \pi(a|s) r(s,a) \,\mathrm{d}a + \gamma \int_{\mathcal{A}} \pi(a|s) \int_{\mathcal{S}} p(s'|s,a) V^\pi(s') \,\mathrm{d}s' \quad (3.18) \\ &= \int_{\mathcal{S}} \pi(a|s) r(s,a) \,\mathrm{d}a + \frac{\gamma}{1-\gamma} \int_{\mathcal{S}} \delta_s^\pi(s') \int_{\mathcal{A}} \pi(a'|s') R(s',a') \,\mathrm{d}a' \,\mathrm{d}s', \quad (3.19) \end{aligned}$$

where in 3.18 we simply applied the Bellman equation 2.15 and in 3.19 we used the previous Lemma 3.6. Now, we simply recall definition 2.14 for the expected return:

$$\begin{aligned} J(\pi) &= \int_{\mathcal{S}} \mu(s) V^\pi(s) \,\mathrm{d}s \\ &= \int_{\mathcal{S}} \big[\mu(s) + \frac{\gamma}{1-\gamma} \int_{\mathcal{S}} \delta_{s_0}^\pi \mu(s_0) \,\mathrm{d}s_0\big] \int_{\mathcal{A}} \pi(a|s) r(s,a) \,\mathrm{d}a \,\mathrm{d}s \\ &= \frac{1}{1-\gamma} \int_{\mathcal{S}} \delta_\mu^\pi(s) \int_{\mathcal{A}} \pi(a|s) r(s,a) \,\mathrm{d}a \,\mathrm{d}s, \quad (3.20) \end{aligned}$$

where in 3.20 we applied the definition of δ_μ^π in Equation 2.8. $\square$

We can now state the main theorem:

Theorem 3.8 (Policy Gradient Theorem (Sutton et al., 1999a)). *Let $\mathcal{M}$ be an MDP, and let $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$ be differentiable w.r.t. $\boldsymbol{\theta}$. Then:*

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{1-\gamma} \int_{\mathcal{S}} \delta_\mu^{\boldsymbol{\theta}}(s) \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a)\,\mathrm{d}a\,\mathrm{d}s. \tag{3.21}$$

Proof. Starting once again from the Bellman equation 2.15:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} Q^{\boldsymbol{\theta}}(s,a) &= \nabla_{\boldsymbol{\theta}}\big[r(s,a) + \gamma \int_{\mathcal{S}} p(s'|s,a) V^{\boldsymbol{\theta}}(s')\,\mathrm{d}s'\big] \\ &= \gamma \int_{\mathcal{S}} p(s'|s,a) \nabla_{\boldsymbol{\theta}} V^{\boldsymbol{\theta}}(s')\,\mathrm{d}s'\end{aligned}$$

Furthermore:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} V^\pi(s) &= \nabla_{\boldsymbol{\theta}} \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a)\,\mathrm{d}a \\ &= \int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(a|s)\,\mathrm{d}a + \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} Q^{\boldsymbol{\theta}}(s,a)\,\mathrm{d}a \\ &= \int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(a|s)\,\mathrm{d}a + \gamma \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \int_{\mathcal{S}} p(s'|s,a) \nabla_{\boldsymbol{\theta}} V^{\boldsymbol{\theta}}(s')\,\mathrm{d}s'\,\mathrm{d}a \\ &= \int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a)\,\mathrm{d}a + \frac{\gamma}{1-\gamma} \int_{\mathcal{S}} \delta_s^{\boldsymbol{\theta}}(s') \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a'|s') R(s',a')\,\mathrm{d}a'\,\mathrm{d}s',\end{aligned}$$

where in the last equality we applied Lemma 3.6, with $g(s) = \int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a)\,\mathrm{d}a$. Using the same arguments as in Corollary 3.7:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \int_{\mathcal{S}} \mu(s) V^{\boldsymbol{\theta}}(s)\,\mathrm{d}s \\ &= \int_{\mathcal{S}} \left[\mu(s) + \frac{\gamma}{1-\gamma} \int_{\mathcal{S}} \delta_{s_0}^{\boldsymbol{\theta}}(s) \mu(s_0)\,\mathrm{d}s_0\right] \int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a)\,\mathrm{d}a\,\mathrm{d}s \\ &= \frac{1}{1-\gamma} \int_{\mathcal{S}} \delta_\mu^{\boldsymbol{\theta}}(s) \int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a)\,\mathrm{d}a\,\mathrm{d}s \\ &= \frac{1}{1-\gamma} \int_{\mathcal{S}} \delta_\mu^{\boldsymbol{\theta}}(s) \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a)\,\mathrm{d}a\,\mathrm{d}s,\end{aligned}$$

where in the last equation we applied the *log-trick*: $\nabla f = f \nabla \log f$. $\square$

From the statement, a new term is introduced, the *score* function of a policy, i.e., $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\cdot|s)$. In a sense, the score can be understood as the direction in the parameter space Θ that allows maximization of the likelihood that an action is chosen by the policy in a state. The policy gradient can then be understood as the expected value under the state-action occupancy measure of the Q value function multiplied by the policy score. As a final remark, one of the most important generalizations of the Policy Gradient Theorem consists in the inclusion of a *baseline* in the gradient term, i.e., an integrable

Algorithm 3 Actor-only Policy Gradient

```
Require: initial policy parametrization $\boldsymbol{\theta}_0$, stepsize sequence $\{\alpha_t\}_t$, batch size $N$ .
Ensure: policy $\pi_{\boldsymbol{\theta}_T}$
1: for $t = 0, 1, \dots$ do
2:     Collect $N$ trajectories $\{\tau_i\}_{i=1}^N$ such that $\tau_i \sim \rho_{\boldsymbol{\theta}_t}$.
3:     Compute Policy Gradient estimate $\widehat{\nabla} J(\boldsymbol{\theta}_t)$.
4:     Update parameters $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t \widehat{\nabla} J(\boldsymbol{\theta}_t)$.
5: end for
```

function $b : \mathcal{S} \to \mathbb{R}$ that allows reducing the variance of the related estimators. Indeed, we can first consider that:

$$\int_{\mathcal{S}} \delta_\mu^{\boldsymbol{\theta}}(s) \int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) b(s) \, \mathrm{d}a \, \mathrm{d}s = \int_{\mathcal{S}} \delta_\mu^{\boldsymbol{\theta}}(s) b(s) \nabla_{\boldsymbol{\theta}} \underbrace{\int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \, \mathrm{d}a}_{1} \, \mathrm{d}s = \mathbf{0}.$$

As a consequence, the Policy Gradient Theorem also holds in the following form:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{(s,a) \sim \nu_{\boldsymbol{\theta}}} \left[Q^{\boldsymbol{\theta}}(s,a) - b(s) \right]. \tag{3.22}$$

One of the most common baselines adopted (if explicitly computed) is the value function $V^{\boldsymbol{\theta}}$. In this case, the gradient is retrieved by considering the advantage function $A^{\boldsymbol{\theta}}$.

Policy Gradient estimators The policy-gradient-based algorithms that do not compute an explicit estimation of the action-value function $Q^{\boldsymbol{\theta}}$ are denoted as *actor-only*, and they typically resort to an estimator $\widehat{\nabla} J(\boldsymbol{\theta})$ of the true policy gradient. Hence, the general procedure of these techniques, denoted as *vanilla* policy gradient, is the one reported in Algorithm 3(Peters and Schaal, 2006): at each iteration, a batch of N trajectories is collected from the interaction of the current policy $\pi_{\boldsymbol{\theta}_t}$ with the environment, in an *on-policy* fashion. The generated batch is then used to compute an unbiased Monte Carlo estimator of the policy gradient $\widehat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$. Finally, the next parameterization is obtained through a SGA update rule:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t \widehat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t),$$

where α_t is the learning rate. Differently from Value Based approaches, Policy-based methods can only be proven to converge to a local optimum (Bottou et al., 1998; Sutton et al., 1999a) under regularity assumptions on $J(\boldsymbol{\theta})$ and on the stepsizes, which need to satisfy $\lim_{t\to\infty} \alpha_t = 0$ and that $\sum_t \alpha_t = \infty$. However, PG estimators usually have high variance, for which tuning the stepsize can be very difficult: lower learning rates grant safer but slow learning, while high learning rates can lead to instabilities (Papini et al., 2019). The learning rate then requires careful tuning, and different solutions have been provided. Further discussions are provided in Part I.

REINFORCE The first and most known estimator for the policy gradient was introduced in (Williams, 1992). It is actually derived from a different version of the PG

Theorem; indeed, a non trivial consequence of Corollary 3.7 is the following:

$$J(\pi) = \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{(s,a)\sim\nu_\pi^\mu}[r(s,a)] = \mathop{\mathbb{E}}_{\tau\sim\rho_\pi}[G(\tau)],$$

where $G(\tau)$ is the return of the trajectory τ and the distribution over trajectories ρ_π is defined in 2.4. Consequently, the gradient of the expected return w.r.t. the policy parameters can be computed as follows:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \int_{\mathcal{T}} \rho_{\boldsymbol{\theta}}(\tau) G(\tau)\,\mathrm{d}\tau = \int_{\mathcal{T}} \nabla_{\boldsymbol{\theta}} \rho_{\boldsymbol{\theta}}(\tau) G(\tau)\,\mathrm{d}\tau \\ &= \mathop{\mathbb{E}}_{\tau\sim\rho_{\boldsymbol{\theta}}} \big[\nabla_{\boldsymbol{\theta}} \log \rho_{\boldsymbol{\theta}}(\tau) G(\tau)\big] = \mathop{\mathbb{E}}_{\tau\sim\rho_{\boldsymbol{\theta}}} \Big[\sum_{t=0}^{T(\tau)-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) G(\tau)\Big]\end{aligned}$$

REINFORCE algorithm is trivially the Monte-Carlo (unbiased) estimation of this equation through a batch of N trajectories:

$$\widehat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) := \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T(\tau_i)-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t^i|s_t^i) G(\tau_i). \tag{3.23}$$

PGT and G(PO)MDP REINFORCE estimator 3.23 suffers from high variance. To reduce it, one can resort to a state-dependent baseline. Otherwise, it is possible to consider that this estimator contains many redundant terms: indeed, the policy score related to a specific action a_t in a trajectory is multiplied by its overall return $G(\tau)$, even though past rewards are independent of that future action. Trivially, the causality between actions and future rewards can be considered by removing past rewards from the gradient computation. The result is another estimator, simply denoted as Monte-Carlo PGT (Sutton et al., 1999a):

$$\widehat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) := \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T(\tau_i)-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t^i|s_t^i) G_t(\tau_i). \tag{3.24}$$

As the name suggests, this estimator is actually the Monte-Carlo version of Equation 3.21. Its unbiasedness is straightforward, once we recall that the return-to-go $G_t(\tau)$ observed starting from (s_t, a_t) and then following policy $\pi_{\boldsymbol{\theta}}$ is an unbiased estimate of $Q^{\boldsymbol{\theta}}(s_t, a_t)$. Another estimator, developed in (Baxter and Bartlett, 2001), directly exploits the independence of actions and past rewards to refine REINFORCE. The resulting estimator, G(PO)MDP was implemented autonomously from Monte-Carlo PGT, although later (Peters and Schaal, 2008) proved that they are equivalent.

Actor-Critic Algorithms

In the previous sections, we have seen that the variance of the policy gradient estimators can be reduced via baseline functions. In this sense, one can include estimators of the value function $V^{\boldsymbol{\theta}}$. Algorithms that adopt a value function estimation to improve the policy gradient updates are denoted as *Actor-Critic* methods (Konda and Tsitsiklis, 1999); the *actor* is the policy, and the actions are evaluated by the *critic*, which is the value function estimator $V^{\boldsymbol{\omega}}$, that can be learned through any of the prediction methods reviewed in Section 3.3. Thus, instead of taking into account the entire batch of trajectories to provide an estimation of the Q function, we can consider the TD(0) target $G^{(t)}$. Moreover, we can also replace the baseline $b(s)$ from 3.22 with the critic $V^{\boldsymbol{\omega}}(s)$, and no bias is added to the estimator. This choice leads to the following gradient computation:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T(\tau_i)-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t^i | s_t^i) \big(r_t + \gamma V^{\boldsymbol{\omega}}(s_{t+1}) - V^{\boldsymbol{\omega}}(s_t) \big).$$

This approach is known by the name of *Advantage Actor Critic* (A2C, Sutton and Barto (1998)) since the TD target represents an estimation of the advantage function. Another solution consists in directly replacing the Q-value function in the policy gradient theorem with a parametrized estimator $Q^{\boldsymbol{\omega}}$. This choice usually introduces a small bias in the policy update. To avoid this bias, a sufficient condition in the design of the critic is the *compatible critic* hypothesis, i.e.

$$\nabla_{\boldsymbol{\omega}} Q^{\boldsymbol{\omega}}(s,a) = \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A}.$$

Natural Policy Gradient

One of the major drawbacks of the vanilla policy gradient approaches introduced in the previous section is the high sensitivity of the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ w.r.t. the policy parametrization. This means that some parameter updates might have negligible effects, while others might lead to *unsafe* behaviors with undesired performances. One solution to avoid this behavior can be attained by applying a specific preconditioning matrix to the estimated gradient employing the Fisher Information Matrix (FIM, Amari (1998)). The FIM of a state $s \in \mathcal{S}$ is mathematically defined as the covariance of the score, or *likelihood*, of a probability measure: it measures the amount of information related to $\boldsymbol{\theta}$ carried by the sampled action:

$$F_s(\boldsymbol{\theta}) := \mathop{\mathbb{E}}_{a \sim \pi_{\boldsymbol{\theta}}(\cdot|s)} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)^{\top} \right].$$

As described in (Amari, 1998), the FIM is invariant w.r.t. changes in the parameter space Θ. To relate this information measure to the likelihood of the actions taken into account in the computation of the policy gradient, we need to marginalize the Fisher information under the state-occupancy measure:

$$F(\boldsymbol{\theta}) := \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{s \sim \delta_{\mu}^{\boldsymbol{\theta}}} [F_s(\boldsymbol{\theta})].$$

An important variation on the PG approach consists then in the application of the inverse FIM as a preconditioning matrix of the policy gradient:

$$g(\boldsymbol{\theta}) := F(\boldsymbol{\theta})^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}). \tag{3.25}$$

The resulting methodology is denoted as Natural Policy Gradient (NPG, Kakade (2001)) and is indeed covariant with respect to the policy parametrization (Peters et al., 2005). Sometimes, this technique is also denoted as Natural Gradient Ascent (NGA). Another benefit of this new gradient is that the FIM includes information regarding the curvature of the return manifold over the policy space; hence it follows the *steepest ascent direction* and is close to a second-order method: this holds since $F(\boldsymbol{\theta})$ is equivalent to the negative Hessian of the score function:

$$F(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{(s,a)\sim\nu_\mu^{\boldsymbol{\theta}}}[-\nabla^2_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)].$$

Trivially, as PG approaches need estimators for the gradient, the FIM should be estimated as well, but the computation of the inverse might become unfeasible for large policy spaces: a common approach to avoid long computational times is to directly provide an estimate of the natural gradient $g(\boldsymbol{\theta})$ by using the same batch of trajectories considered for the gradient estimator, and through the iterative application of conjugate gradient methods with the application of the Fisher-vector products (Schulman et al., 2015).

3.5.1 Trust-Region Update

Policy Gradient Methods, if Natural Gradient is not taken into account, are first-order methods and do not include any information related to the neighborhood of the current policy. Thus, the choice of proper step sizes is very difficult due to the locality and the variance in the estimations. Furthermore, the new policies obtained through linear updates might induce a completely different occupancy distribution over the state-action space, possibly leading to pejorative results.

An important result in this direction is provided in (Kakade and Langford, 2002), where we can compare the performances of two policies as an expected advantage

Theorem 3.9 (Performance Difference, Kakade and Langford 2002, Lemma 6.1)**.** *Let $\mathcal{M}$ be an MDP, and $\pi, \pi' \in \Pi$ be two policies. Then:*

$$J(\pi') - J(\pi) = \int_{\mathcal{S}} \delta_\mu^{\pi'}(s) \int_{\mathcal{A}} \pi'(a|s) A^\pi(s,a)\, \mathrm{d}a\, \mathrm{d}s. \tag{3.26}$$

Starting with this result, it is possible to derive some lower bounds of the performances on the next policy from the advantage and the returns from the previous one. Starting from API, (Kakade and Langford, 2002) provided the first *monotonic improvement guarantees*, then extended in the policy search framework (Pirotta et al., 2013a, 2015; Papini et al., 2017). These works provide useful rules to select *safe* stepsizes and, therefore, avoid catastrophic degeneration of the performances. In complex environments, the required learning rates are usually very low for safety guarantees to hold, and practitioners in RL seldom adopt these approaches for their slow learning.

One of the alternatives for this issue comes from Convex Optimization: while Line Search (LS) methods first compute the update direction and afterward the stepsize, Trust Region (TR) approaches aim to find the best solution within a certain region of the parameter space. In this sense, they first set the trust region, i.e., the subspace of solutions whose *distance* from the current one is lower than a certain threshold; afterward, they solve the subproblem constrained inside the trust region. In this way, the next updates cannot differ too much from the previous iterations, and we can have reliable estimates of the future performance starting from the data gathered by the current policy. In the following, we will provide further details regarding one of the most known policy search algorithms: Trust Region Policy Optimization (TRPO, Schulman et al. (2015)).

Trust Region Policy Optimization

The core idea of TRPO is directly related to the results presented in (Kakade and Langford, 2002). Indeed, starting from Theorem 3.9, we can notice that, to compute the performance difference between the current policy π and the next candidate π', we need knowledge of the occupancy measure $\delta_\mu^{\pi'}$, making the optimization process very complex. Instead, from the computation of the policy gradient, we can have some knowledge of the distribution δ_μ^π; thus, one can be interested in the optimization of a surrogate objective:

$$L_\pi(\pi') := J(\pi) + \int_{\mathcal{S}} \delta_\mu^\pi(s) \int_{\mathcal{A}} \pi'(a|s) A^\pi(s,a) \,\mathrm{d}a \,\mathrm{d}s. \tag{3.27}$$

This objective is a *local approximation* of $J(\pi')$ to the first order; hence it provides some useful information on a neighborhood of π. Hence, a notion of *distance* in the policy space is needed: in particular, we take into account the Kullback-Leibler divergence (KL) divergence, a common (asymmetric) measure of dissimilarity between two distributions, which will be fully defined in Chapter 5:

$$D_{KL}(\pi(\cdot|s)||\pi'(\cdot|s)) := \int_{\mathcal{A}} \pi(a|s) \log \frac{\pi(a|s)}{\pi'(a|s)} \,\mathrm{d}a.$$

The main result provided in (Schulman et al., 2015) allows relating the distance between two policies and a lower bound on the expected improvement:[4]

Theorem 3.10 (Schulman et al. 2015, Theorem 1)**.** *Let α be the maximum KL divergence, defined as:*

$$\alpha = D_{KL}^{max}(\pi||\pi') := \max_{s \in \mathcal{S}} D_{KL}\big(\pi(\cdot|s)||\pi'(\cdot|s)\big).$$

Then:

$$J(\pi') \geq L_\pi(\pi') - \frac{4\epsilon\gamma}{(1-\gamma)^2}\alpha,$$

where $\epsilon := \max_{(s,a) \in \mathcal{S} \times \mathcal{A}} |A^\pi(s,a)|$.

This equation trivially implies that the expected return $J(\pi')$ can be worse than its surrogate approximation, but the loss is limited by a term directly depending on the maximum KL distance. Consequently, the core idea of TRPO is the following: for each iteration, instead of finding the optimal direction given by the gradient and then following it for a given step-size, we can define a trust region around $\pi_{\boldsymbol{\theta}_t}$ and then

[4]In Theorem 3.10, we provide the formulation under the KL divergence as a distance between policy. In (Schulman et al., 2015), a tighter bound is provided by using the *total variation divergence*.

solving the following constrained optimization subproblem:[5]

$$\begin{aligned} \arg\max_{\theta \in \Theta} \quad & L_{\boldsymbol{\theta}_t}(\boldsymbol{\theta}) \\ s.t. \quad & D_{KL}^{max}(\pi_{\boldsymbol{\theta}_t} \ || \ \pi_{\boldsymbol{\theta}_{t+1}}) \leq \lambda. \end{aligned} \tag{3.28}$$

Here, we have introduced a new hyperparameter, λ, that replaces the stepsize term from PG approaches. Fortunately, this problem admits a solution in closed form, which can represent the entire TRPO in a single update rule:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \underbrace{\sqrt{\frac{2\lambda}{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})^\top F(\boldsymbol{\theta})^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}}}_{=\alpha(\lambda)} F(\boldsymbol{\theta})^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{3.29}$$

This last result represents the main relationship between the Trust Region update in TRPO, and the Natural Policy Gradient presented in the previous section: Equation 3.29 is exactly the NPG update rule, with a stepsize α_t dependent on the trust-region constraint λ and on the second-order included within the FIM $F(\boldsymbol{\theta})$. This is actually because the FIM and the KL divergence are strictly connected, as the latter is the quadratic approximator of the former:

$$D_{KL}\big(\pi_{\boldsymbol{\theta}}(\cdot|s) \ || \ \pi_{\boldsymbol{\theta}'}(\cdot|s)\big) = \frac{1}{2}(\boldsymbol{\theta}' - \boldsymbol{\theta})^\top F_s(\boldsymbol{\theta})(\boldsymbol{\theta}' - \boldsymbol{\theta}) + o\big(\|\boldsymbol{\theta}' - \boldsymbol{\theta}\|^3\big).$$

A further approximation to TRPO is introduced in Proximal Policy Optimization (PPO, Schulman et al. (2017)). It can be considered state of the art along with TRPO for its impressive empirical results. In this algorithm, the KL constraint is removed, in favor of a *clipping factor* added in the surrogate objective, with a similar ratio as the trust region. The main important results of this algorithm are not only related to the experimental improvements on some benchmark domains (Schulman et al., 2017) but also to the lower computational complexity required to perform the iterations since it avoids the estimation of the natural gradient.

TRPO implementation From a theoretical point of view, the results provided in (Schulman et al., 2015) justify the algorithm as a *safe RL* approach due to its monotonic improvement guarantees, in the sense that, once the constraint λ is selected, the subsequent iterations of the update rule in Equation 3.29 are guaranteed to be better as the lower bounds. From the empirical point of view, TRPO is considered a state-of-the-art deep RL algorithm for continuous control due to its successful results in challenging high-dimensional control environments. However, the practical implementation of TRPO, shown in Algorithm 4, is slightly different from the theoretical foundation, arguably leading to question the true guarantees properties of the algorithm, sometimes considered as a heuristics (Papini et al., 2019; Shani et al., 2020). Indeed, later works provided theoretical justifications for the empirical algorithm (Neu et al., 2017) with no monotonic guarantees (Shani et al., 2020). Apart from some approximations introduced to estimate divergences and advantages, the main TRPO implementation requires

[5]As mentioned in Section 3.5, we slightly abuse the notation, and represent the parameters $\boldsymbol{\theta}$ instead of the related policy $\pi_{\boldsymbol{\theta}}$ as the arguments for the various terms. We keep the π term within the KL divergence term to remark that it is not a distance on the parameter space Θ but on the policy space Π.

Algorithm 4 Trust Region Policy Optimization (TRPO)

```
Require: initial policy parameter $\boldsymbol{\theta}_0$, batch size $N$, maximum KL threshold $\lambda$
Ensure: policy $\pi_{\boldsymbol{\theta}_T}$
 1: for $t = 0, \dots, T-1$ do
 2:     Collect $N$ trajectories with $\boldsymbol{\theta}_0$
 3:     Compute Monte-Carlo estimation of $J(\boldsymbol{\theta}_t)$
 4:     Estimate advantage values $A^{\boldsymbol{\theta}_t}(s,a)$ (using any method of advantage estimation)
 5:     Use conjugate method to estimate natural gradient $\boldsymbol{g}_t = \widetilde{\nabla}_{\boldsymbol{\theta}_t} J(\boldsymbol{\theta}_t)$
 6:     Set initial stepsize as $\alpha = \alpha(\lambda)$ from 3.29
 7:     while $\boldsymbol{\theta}_{t+1}$ is not set do
 8:         Set $\widetilde{\boldsymbol{\theta}} = \boldsymbol{\theta}_t + \alpha \boldsymbol{g}_t$
 9:         Compute surrogate objective $L_{\boldsymbol{\theta}_t}(\widetilde{\boldsymbol{\theta}})$
10:         if $D_{KL}^{max}(\pi_{\boldsymbol{\theta}_t} || \pi_{\widetilde{\boldsymbol{\theta}}}) > \lambda$ or $L_{\boldsymbol{\theta}_t}(\widetilde{\boldsymbol{\theta}}) < J(\boldsymbol{\theta}_t)$ then        Double Check
11:             Set $\boldsymbol{\theta}_{t+1} = \widetilde{\boldsymbol{\theta}}$
12:         else
13:             Set $\alpha \leftarrow \alpha/2$        Halving stepsize
14:         end if
15:     end while
16: end for
```

two phases for each iteration: The first phase uses an iterative conjugate gradient optimization to find the search direction, to approximate the *natural gradient* using the Fisher information matrix, as in Kakade and Langford (2002). Afterward, TRPO levers a criteria-based line-search to obtain a feasible step size; starting from a maximum threshold and following the direction found in the previous phase, the stepsize is iteratively halved if the trust region constraint in 3.28 is not satisfied or if the surrogate function does not predict a return improvement w.r.t. the previous expected return. These double checks are needed because the noise introduced in the aforementioned approximations makes the empirical algorithm more unstable, and consequently, the tuning process required to obtain an effective and reliable trust region constraint λ is not trivial, as discussed later in Chapter 5.

3.6 Lipschitz MDP

We conclude the chapter by introducing some concepts of Lipschitz continuity and Lipschitz MDPs. The seminal works in this direction are provided by (Rachelson and Lagoudakis, 2010; Pirotta et al., 2015), to which the reader is invited to refer for a more detailed overview. These notions will be adopted in Chapters 4 and 6 with other ad-hoc assumptions to derive new results in specific settings.

The core idea in (Pirotta et al., 2015) is to provide a set of regularity assumptions on the MDP under which it is possible to provide an upper bound for the stepsize in classic PG algorithms with monotonic convergence guarantees. The main rationale is the same as TRPO, related to the difficulty of tuning the stepsize in a trade-off between convergence issues and slow learning. The introduced Lipschitz assumptions are related to the smoothness of the transition and reward models: whenever similar actions are executed in similar states, the rewards and the next states will be similar. It is possible to prove that, if the model and the policy adopted are Lipschitz-continuous, this property also holds for the expected return and the policy gradient. The smoothness of the gradient can then be adopted to the same extent as in standard optimization (Armijo, 1966) to detect step sizes for which monotonic improvements are guaranteed.

Definition 3.11. *Consider two metric spaces $(\mathcal{X}, d_\mathcal{X})$ and $\mathcal{Y}, d_\mathcal{Y})$.*
A function $f : \mathcal{X} \to \mathcal{Y}$ is called L_f-Lipschitz continuous (in brief, L_f-LC), with $L_f > 0$ denoted as Lipschitz constant, if:

$$d_\mathcal{Y}\big(f(x), f(x')\big) \leq L_f d_\mathcal{X}\big(x, x'\big), \quad \forall x, x' \in \mathcal{X}. \tag{3.30}$$

A function $f : \mathcal{X} \to \mathcal{Y}$ is called Pointwise Lipschitz continuous (PLC) in $x \in \mathcal{X}$ if there exists $L_f(x) < L_f$ such that:

$$d_\mathcal{Y}\big(f(x), f(x')\big) \leq L_f(x) d_\mathcal{X}\big(x, x'\big), \quad \forall x' \in \mathcal{X}.$$

Moreover, we can define the Lipschitz semi-norm of a function f in a function space $\mathcal{F}(\mathcal{X}, \mathcal{Y})$:

$$\|f\|_L \coloneqq \sup_{\substack{x,x' \in \mathcal{X}, \\ x \neq x'}} \left\{ \frac{d_\mathcal{Y}\big(f(x), f(x')\big)}{d_\mathcal{X}\big(x, x'\big)} \right\}. \tag{3.31}$$

When dealing with real functions, the usual metric adopted is the Euclidean distance: $d(x, x') = \|x - x'\|^2$, while for probability distributions, we usually employ the Kantorovich metric (or L_1-Wasserstein): for p and q probability measures:

$$d(p, q) = \mathcal{W}_1(p, q) \coloneqq \sup_{f : \|f\|_L \leq 1} \left| \int_\mathcal{X} f(x)(p - q) \, \mathrm{d}x \right|.$$

We now introduce the main assumptions leading to the definition of *Lipschitz-MDP* and Lipschitz policy (Rachelson and Lagoudakis, 2010; Pirotta et al., 2015). A preliminary, non-limiting assumption is that the state and action spaces $\mathcal{S}$ and $\mathcal{A}$ are complete and separable metric spaces, respectively endowed with distances $d_\mathcal{S}$ and $d_\mathcal{A}$. Moreover, the joint state-action space $\mathcal{S} \times \mathcal{A}$ is endowed with the *taxicab* norm:(Hinderer, 2005)

$$d_{\mathcal{S}\times\mathcal{A}}((s, a), (\overline{s}, \overline{a}))) = d_\mathcal{S}(s, \overline{s}) + d_\mathcal{A}(a, \overline{a}) \quad \forall (s, a), (\overline{s}, \overline{a}) \in \mathcal{S} \times \mathcal{A}.$$

Assumption 3.1 (Lipschitz MDP). *Let $\mathcal{M}$ be an MDP. $\mathcal{M}$ is called $(L_P, L_R)-$LC if the transition model and the reward function are respectively L_P-LC and L_R-LC; i.e., for all $(s, a), (\overline{s}, \overline{a}) \in \mathcal{S} \times \mathcal{A}$:*

$$\begin{aligned} \mathcal{W}_1(P(\cdot|s, a), P(\cdot|\overline{s}, \overline{a})) &\leq L_p \, d_{\mathcal{S}\times\mathcal{A}}((s, a), (\overline{s}, \overline{a})) \\ |r(s, a) - r(\overline{s}, \overline{a})| &\leq L_R \, d_{\mathcal{S}\times\mathcal{A}}((s, a), (\overline{s}, \overline{a})). \end{aligned}$$

The previous definition of Lipschitz return generally refers to the expected reward r; of course, it is possible to generalize the concept to the reward model R.

Assumption 3.2 (Lipschitz Policy). *Let $\pi \in \Pi$ be a Markovian stationary policy. π is called $L_\pi-$LC if, for all $s, \overline{s} \in \mathcal{S}$:*

$$\mathcal{W}_1(\pi(\cdot|, s), \pi(\cdot|, \overline{s})) \leq L_\pi d_\mathcal{S}(s, \overline{s}).$$

These assumptions are enough to prove that also the value functions are Lipschitz continuous (LC):

Lemma 3.12 (Lipschitz value functions, Lemma 1 from Rachelson and Lagoudakis 2010)**.** *Let $\mathcal{M}$ be an $(L_P, L_R)-$LC MDP, and let $\pi \in \Pi$ be L_π-LC. If $\gamma L_P(1+L_\pi) < 1$, then the Q-function Q_π is L_{Q_π}-LC, and the value function V_π is L_{V_π}-LC with respect to the joint state-action space:*

$$L_{Q_\pi} = \frac{L_R}{1-\gamma L_P(1+L_\pi)}; \qquad L_{V_\pi} = L_{Q_\pi}(1-L_\pi).$$

In policy-based algorithms, we usually deal with parametric policies; sometimes, we will need to define the continuity of a policy in terms of its parameters:

Assumption 3.3 (Lipschitz Parametric Policy)**.** *Let Θ be a parameter space with distance d_Θ. Let $\pi_{\boldsymbol{\theta}} \in \Pi$ be a Markovian stationary policy parametrized by $\boldsymbol{\theta} \in \Theta$. π is LC if it satisfies the following conditions:*

1. *State LC:* $\mathcal{W}_1\big(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_{\boldsymbol{\theta}}(\cdot|\overline{s})\big) \leq L_{\pi_{\boldsymbol{\theta}}} d_{\mathcal{S}}(s,\overline{s}) \quad \forall \boldsymbol{\theta} \in \Theta,\ \forall s, \overline{s} \in \mathcal{S}$;
2. *Parametric PLC:* $\mathcal{W}_1\big(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_{\overline{\boldsymbol{\theta}}}(\cdot|s)\big) \leq L_\pi(\boldsymbol{\theta}) d_\Theta(\boldsymbol{\theta}, \overline{\boldsymbol{\theta}}) \quad \forall s \in \mathcal{S},\ \forall \boldsymbol{\theta}, \overline{\boldsymbol{\theta}} \in \Theta.$

From Assumption 3.3, the first condition requires that each parametrized policy chooses similar actions in similar states; moreover, the second condition is related to the smoothness of the parameter space, as similar parametrizations select similar actions in the same state.

Some final assumptions are related to the Lipschitz continuity and boundedness of the gradient of the score function:

Assumption 3.4 (Lipschitz Score Gradient)**.** *Let Θ be a parameter space with distance d_Θ. Let $\pi_{\boldsymbol{\theta}} \in \Pi$ be a Markovian stationary policy parametrized by $\boldsymbol{\theta} \in \Theta$. The gradient of the policy logarithm satisfies the following conditions:*

1. *Uniformly bounded gradient:* $\forall (s,a) \in \mathcal{S} \times \mathcal{A}, \forall \boldsymbol{\theta} \in \Theta, \forall i = 1, \dots, d$
$$|\nabla_{\boldsymbol{\theta}_i} \log \pi_{\boldsymbol{\theta}}(a \mid s)| \leq M^i_{\boldsymbol{\theta}};$$
2. *State-action LC:* $\forall\, (s, \overline{s}, a, \overline{a}) \in \mathcal{S}^2 \times \mathcal{A}^2, \forall \boldsymbol{\theta} \in \Theta, \forall i = 1 \dots, d$
$$|\nabla_{\boldsymbol{\theta}_i} \log \pi_{\boldsymbol{\theta}}(a \mid s) - \nabla_{\boldsymbol{\theta}_i} \log \pi_{\boldsymbol{\theta}}\left(\overline{a} \mid \overline{s}\right)| \leq L^i_{\nabla \log \pi} d_{\mathcal{S}\times\mathcal{A}}\left((s,a), (\overline{s}, \overline{a})\right)\big);$$
3. *Parametric LC:* $\forall \left(\boldsymbol{\theta}, \overline{\boldsymbol{\theta}}\right) \in \Theta, \forall (s,a) \in \mathcal{S} \times \mathcal{A}, \forall i = 1, \dots, d$
$$|\nabla_{\boldsymbol{\theta}_i} \log \pi_{\boldsymbol{\theta}}(a \mid s) - \nabla_{\boldsymbol{\theta}_i} \log \pi_{\overline{\boldsymbol{\theta}}}(a \mid s)| \leq L^i_{\nabla \log \pi}(\boldsymbol{\theta}) d_\Theta\left(\boldsymbol{\theta}, \overline{\boldsymbol{\theta}}\right).$$

Given these assumptions, it is possible to prove the smoothness of both the expected performance and the policy gradient (Pirotta et al., 2015):

Lemma 3.13 (Lipschitz continuity of the return)**.** *Given an $(L_P, L_R)-$LC MDP, the expected return is Lipschitz Continuous w.r.t. the state-occupancy measure:*

$$|J^{\boldsymbol{\theta}} - J^{\overline{\boldsymbol{\theta}}}| \leq \frac{L_R}{1-\gamma} \mathcal{W}_1(\delta^{\boldsymbol{\theta}}_\mu, \delta^{\overline{\boldsymbol{\theta}}}_\mu) \quad \forall \boldsymbol{\theta}, \overline{\boldsymbol{\theta}} \in \Theta.$$

Moreover, if Assumption 3.3 holds with $\gamma L_P(1 + L_{\pi_{\boldsymbol{\theta}}}) < 1$, the performance measure if PLC w.r.t. the policy parameters:

$$|J^{\boldsymbol{\theta}} - J^{\overline{\boldsymbol{\theta}}}| \leq L_J(\boldsymbol{\theta}) d_\Theta(\boldsymbol{\theta}, \overline{\boldsymbol{\theta}}),$$

where

$$L_J(\boldsymbol{\theta}) = \frac{L_R L_\pi(\boldsymbol{\theta})}{(1-\gamma)(1-\gamma L_P(1+L_{\pi_{\boldsymbol{\theta}}}))}. \tag{3.32}$$

This result provides some guarantees that if the policy parametrization does not change too much, the expected performances are similar: if the stepsize is small enough, the result cannot be much worse. In conclusion, adding Assumption 3.4, the Policy Gradient is also LC for a certain Lipschitz constant L_{∇_J}, hence in similar parametrizations, the directions of the updates are close. This result is then adopted to provide some performance improvement guarantees, and to find the maximum constant stepsize $\alpha_t = 1/L_{\nabla_J}$ for which we are guaranteed to obtain a minimum performance improvement. For the purposes of this dissertation, we will not be interested in this particular result. Instead, in Chapter 4 we will generalize the provided Lipschitz assumptions on a family of MDPs to show that in similar environments the expected performance cannot differ too much. Instead, in Chapter 6 we will add an assumption regarding the speed of the evolution of the system, to bound the divergence in the occupancy measures obtained through action repetition.

Part I

Hyperparameter Optimization through Meta Reinforcement Learning

The main goal of Reinforcement Learning (RL, Sutton and Barto 1998) is to build an agent capable of learning a behavior that maximizes the amount of reward collected while interacting with the environment. One of the most successful streams of model-free RL applications adopts policy-based algorithms, which provide solid theoretical groundings and good empirical properties in continuous-action spaces. These techniques make extensive use of *hyper-parameters* to let the user control their behavior, such as the learning speed, the bias-variance trade-off, and many other aspects. Usually, those hyper-parameters are *fine-tuned by hands*, which means that the same algorithm is run multiple times to then select the best model in a validation instance. The choice of those values may change the learning performance of the algorithm drastically, and the same values may not perform equally on very *similar tasks*; this may require performing very careful fine-tuning sessions or starting from scratch the learning procedure when the framework of the problem is modified.

In this part, we deal with the problem of *Hyperparameter Optimization* of policy-based RL algorithms in the case the underlying MDP may be subject to variations in the transition process or the reward model. In chapter 4, after providing a brief survey of the related works, we devise the problem as a Meta-MDP, where the actions are the hyperparameters selected. The consequent framework then allows increasing the number of degrees of freedom of the update rules adopted, with the consequent chance to enhance its learning capabilities by adapting the hyperparameters to the current policy and the task provided. The per-step reward is learning: consequently, the goal is optimizing the overall learning instance, differently from several other hyperparameter optimization techniques, where the focus is on *one-shot learning*. Afterward, the main approach is applied to selecting the learning rate in a SGA manner. In chapter 5, we try to overcome the main limitations of the previous approach by means of a set of *meta-features* directly estimated on the sampled trajectories. The resulting methodology is then *task-agnostic*, i.e., it does not rely on an explicit parametrization of the latent context, and it is *policy-agnostic*, meaning that it can generalize across different policy classes. The resulting algorithm is then tested on the dynamic selection of the trust region constraint for another well-known policy-based algorithm, TRPO.

CHAPTER 4

Hyperparameter Optimization as a Sequential Decision Problem: Meta Learning the Step Size

4.1 Introduction

In the previous chapter, we have reviewed some of the most known RL algorithms: in particular, policy-based algorithms are among the most widely adopted techniques in model-free RL, thanks to their strong theoretical groundings and good properties in continuous action spaces. Unfortunately, these methods require precise and problem-specific hyperparameter tuning to achieve good performance and, as a consequence, they tend to struggle when asked to accomplish a series of heterogeneous tasks. In particular, the selection of the step size has a crucial impact on their ability to learn a highly performing policy, affecting the speed and the stability of the training process and often being the main culprit for poor results.

In this chapter, we introduce the general topic of Hyperparameter Optimization (HO). In the related literature, a solution to the problem is usually searched within a form of a static optimization process, where the configuration of the learning algorithms is set as fixed throughout the whole training process. Instead, we tackle such a problem as a Markov Decision Process, where the selection of each hyperparameter is framed as a policy action, and the reward is provided by the improvement concerning the expected return. In this manner, we adopt a *dynamic* hyperparameter selection with more degrees of freedom. The goal is to train an *adaptive* model with respect to the current solution and the *context* of the problem: in this regard, we take into account families of environments where the overall dynamics depend on external parameters, denoted as contexts or *tasks* (Hallak et al., 2015).

The formulation of the problem we provide, denoted as meta-MDP, can be classified as a meta learning (ML) approach: this term is usually used to imply a raise in the level of abstraction in the problem to be solved, and it is applied with a twofold meaning. Very frequently, the main focus is devoted to *learning how to learn* (Schmidhuber, 1987), i.e., developing models capable of exploiting and improving the learning capabilities of standard algorithms. In this sense, HO operates within this point of view since the goal is to tune the hyperparameter configurations to obtain better and faster learning. Besides, meta-learning is also often referred to denote the generalization capabilities of a learning algorithm with respect to the task it is aimed to solve (Vanschoren, 2018). In this sense, the main idea is to gather information on how an algorithm learns in some training instances, to then *transfer learning* to new tasks, which can be solved more easily without the necessity of starting a learning procedure from scratch. Following this line of thought, we want to exploit the learning capabilities of a RL algorithm on a variable environment, i.e., where the transition process or the reward model can vary with some latent variables: our meta-MDP approach can be adopted to solve any hyperparameter selection problem for RL in contextual processes.

Chapter Outline This chapter is organized as follows: we start with Section 4.2, where we motivate our approach, and we introduce the definition of a CMDP, with some useful examples. Before delving into the main contributions, we briefly review the literature related to HO and Meta-RL (Section 4.3). The definition of the problem as a Meta-MDP is provided in Section 4.4: we discuss the main elements of the model, such as the meta objective function, which is performance learning, and the meta action, consisting in the hyperparameter selection for a policy update. In this framework, we then add an assumption of Lipschitz continuity of the meta-MDPs, in which trajectories sampled from similar contexts are similar. This is a reasonable assumption on contextual processes, where a small change in the settings slightly changes the effects on the dynamics of the environment. Under such conditions, it is possible to derive some guarantees on the Lipschitz continuity of the expected return and its gradient (Section 4.5). This is relevant, as it provides insight into the generalization capabilities of meta-RL approaches, where the performance of policies selected by observing tasks during training can be bounded for test tasks. Finally, in Section 4.6, we adopt the proposed framework to the problem of optimizing the stepsize in policy gradient instances. We attempt to solve the problem by learning a model trained by means of a batch RL algorithm (FQI) to dynamically recommend the most adequate step size for different policies and tasks. The learning procedure is based on a regression through ExtraTrees (Geurts et al., 2006), which shows low sensitivity to the choice of its own hyperparameters. In conclusion, we present an experimental campaign to show the advantages of selecting an adaptive learning rate in heterogeneous environments 4.7 and try to provide some conclusions underlying the benefits and limitations of our approach in Section 4.8. The proofs of all results are available in Appendix A.1.

4.2 Motivations

The main goal of Reinforcement Learning is to build an agent capable of learning the behavior that maximizes the amount of reward collected while interacting with the environment. Among the most successful streams of model-free RL applications, we can find policy-based algorithms 3.5, which provide solid theoretical groundings and good empirical properties in continuous-action spaces. Unfortunately, these methods require precise and problem-specific hyperparameter tuning to achieve good performances, causing them to struggle when applied to a series of heterogeneous tasks. The fundamental parameter to tune is the step size, which has a crucial impact on the ability to learn a highly performing policy, affecting the speed and the stability of the training process and often being the main culprit for poor results. SGD theory provides useful convergence guarantees to local optima under specific assumptions on the regularity of the environment (Bottou et al., 1998; Pirotta et al., 2015) and on the step-size sequence, e.g., Robbins-Monro conditions. However, the typical environments are complex, and the adopted assumptions can hold only with smaller and smaller learning rates, often leading to unfeasible computational times required to reach convergence. Consequently, practitioners usually do not rely on the theoretical properties related to the selection of a specific learning rate but adopt several techniques based only on their empirical results: among the most widely used methods, we can include monotonically decreasing learning rate schedules (linearly or with exponential decay) and adaptive optimizers, such as the very popular Adam optimizer (Kingma and Ba, 2014) and RMSProp (Tieleman and Hinton, 2017)), based on estimates of first and second moments of the gradients. Despite their resulting efficiency in a large variety of environments, they introduce more hyperparameters that still need to be tuned: indeed, the performances of the related algorithms are typically very sensitive with respect to the hyperparameters selected, and there is usually a very small range of effective values (Henderson et al., 2018). From these considerations, the need of HO becomes clear: in Section 4.3 we briefly review the most adopted methods for hyperparameter tuning.

A further complication of this framework comes from the fact that, in real-world scenarios, there may be exogenous variables that can affect the whole dynamics of the processes, which can be denoted as *contexts* or *tasks*. One way to accurately design this kind of framework is by employing Contextual MDPs, introduced in Chapter 2:

Definition 4.1 (Contextual Markov Decision Process Hallak et al. 2015)**.** *A discrete-time Contextual MDP is a tuple $\langle \Omega, \mathcal{S}, \mathcal{A}, \mathcal{M} \rangle$. Ω is called context space. $\mathcal{S}$ and $\mathcal{A}$ are the state space and the action space, and $\mathcal{M}$ is a function mapping any context $\boldsymbol{\omega} \in \Omega$ to an MDP with shared state and action spaces $\mathcal{M}(\boldsymbol{\omega}) = \langle \mathcal{S}, \mathcal{A}, P_{\boldsymbol{\omega}}, R_{\boldsymbol{\omega}}, \mu_{\boldsymbol{\omega}}, \gamma_{\boldsymbol{\omega}} \rangle$.*

In brief, a CMDP includes in a single entity a set of tasks parametrized by a context $\boldsymbol{\omega}$, affecting the dynamics of the environment and the reward process. In the following, we will suppose that the initial distribution μ and the discount factor γ are also fixed among the whole set of tasks. To have a clearer understanding of a CMDP, we provide some examples.

Notation When CMDPs are considered, the performance of a policy π and the related value functions are context-dependent. Consequently, we will include the subscript $\boldsymbol{\omega}$, whenever needed. for instance, the expected return is denoted as $J_{\boldsymbol{\omega}}(\pi)$, or $J^{\pi}_{\boldsymbol{\omega}}$, the contextual version of the action-value function is referred to as $Q^{\pi}_{\boldsymbol{\omega}}$, and the state-occupancy measure is denoted as $\delta^{\mu}_{\pi,\boldsymbol{\omega}}$. We remind the reader that the term ω in the previous chapters was sometimes intended to denote the parameters of value function estimators. In the following, we will only denote with $\boldsymbol{\omega}$ the MDP context.

Example 4.1 (Minigolf)**.** *Suppose the main goal is to learn how to play minigolf. The overall environment is composed of the main field, the hole (goal), the ball, and the club (one can also consider the player as part of the environment). The dynamics are governed by the physical laws that provide the relationship between the input force the player provides through the club and the ball (Penner, 2002). The dynamics are influenced by many factors, such as the friction of the field, the size of the ball, or the length of the club. Furthermore, a high reward is obtained when the ball reaches the goal, which can differ for each hole. Hence, the player must plan his strategy (i.e., the policy) based on the features of the field, the distance from the goal, and the size of the club. A further distinction can be provided if we consider that the player can choose among different clubs; hence the related features can be considered as configurable (Metelli et al., 2018).*

Example 4.2 (CartPole)**.** *Consider the classic CartPole problem introduced in (Barto et al., 1990), where the goal is to balance a pole attached to a cart. The state of the environment is usually designed as a four-dimensional vector: the position and the velocity of the cart, the angle of the pole, and its rotation rate. The action space is* $\{0, 1\}$*, denoting the direction (with a fixed force) the cart is pushed with (left or right). Considering that the pole length and mass can vary, the optimal policy can change, and the problem can become easier or more difficult. Hence we can consider as context a bi-dimensional vector composed of these two features, affecting the transition of the system.*

Example 4.3 (Car race)**.** *Suppose a driver must learn to drive an F1 car and find the best policy within a fixed track. The environment is composed of the car with all its components and the road, and it is governed by the physical laws related to the car and its interaction with the road. Even if the physical structure of the track and the car is the same, the best policy can change with several external factors, such as the friction to the road, its temperature, the weather, or tire consumption. Moreover, the driver must adapt his strategy also concerning several possible vehicle configurations, such as the engine setup and the kind of tires (soft, medium, hard). On the one hand, these can be considered as configurable variables; on the other, each setup is related to a different optimal behavior.*

From the examples, it is clear that the goal in all these cases is not to find the best policy in a fixed context but to be able to generalize to a possibly large set of tasks. In some cases, these variables can be configured, and the objective is reduced to the detection of the best policy-configuration pair $(\pi, \boldsymbol{\omega})$ such that the related expected return is maximized. This particular framework is dealt with Configurable MDPs and is out of the scope of this dissertation (we invite the reader to refer to (Metelli, 2021) for a complete overview of the topic.

When dealing with a set of possible contexts, one can be interested in *transfer learning* (Lazaric et al., 2008; Lazaric, 2012), i.e., storing knowledge from already solved tasks that can be useful to solve new, unseen contexts. Another possibility is to search for a single efficient policy on the whole set of possible tasks: this is the case of *multi-task learning* (Brunskill and Li, 2013; Sodhani et al., 2021). However, a larger number of works in literature is devoted to *Meta-learning*, which is devoted to exploiting the knowledge from already seen tasks, to rapidly adapt the learning algorithms to unseen ones. For example, several works (Finn et al., 2017; Nichol and Schulman, 2018) are focused on finding a suitable initial parametrization of the policy in such a way that a low amount of gradient steps on a new task is enough to provide an efficient agent. However, different contexts might present different challenges in the optimization processes, and while being optimal for some tasks, a set of hyperparameters might be inefficient and unsafe for others. Even though HO and Meta-Learning are often handled as separate fields, they are often interconnected (Franceschi et al., 2017). In the following, we consider the specific problem of learning how to dynamically tune the hyperparameters (in particular, the step size) adaptively with respect to the policy parametrization and the context of a CMDP. As a final remark, we consider that HO solutions are often adapted to optimize any kind of learning process, not only RL-based. However, the high computational complexity usually required by RL methods raises the advantage of learning how to automatically provide the optimal hyperparameters for each task without having to train multiple times the same model with different hyperparameters.

4.3 Related Work

The importance of hyperparameter tuning is widely known in the general Machine Learning field, because it can significantly improve the performance of a model (Henderson et al., 2018; Weerts et al., 2020; Jastrzebski et al., 2020). Therefore, Hyperparameter Optimization (HO) is an important component of Automated Machine Learning (AutoML) with a rich stream of research literature: for an exhaustive introduction and survey on this topic, we invite the reader to check Hutter et al. 2019. Instead, Afshar et al. 2022 and Parker-Holder et al. 2022 provide more specific overviews of algorithm configuration methods for Reinforcement Learning, or *AutoRL*.

The tuning process is usually approached by practitioners as a black-box approach: the most common methods are simple, such as grid search or random search (Bergstra and Bengio, 2012). More advanced methods are obtained by relying on sequential model-based Bayesian optimization (Hutter et al., 2011; Feurer et al., 2015; Snoek et al., 2012), where a probabilistic model is trained to fit the underlying fitness function of the main learning algorithm. In some recent works (Eiben et al., 2007; Sehgal et al., 2019), Genetic Algorithm (GA)s are employed to automatically learn the most performing parameters on RL applications. The main limitation in this kind of approach consists of the need for complete learning instances to evaluate each hyperparameter, which is kept fixed throughout the whole process

A completely different solution consists in training an outer network, typically an RNN (Meier et al., 2018; Ravi and Larochelle, 2017; Andrychowicz et al., 2016; Im et al., 2021) since it is often possible to compute the gradient of the objective function w.r.t. the hyperparameters through implicit differentiation, as shown in Maclaurin et al. (2015) and Lorraine et al. (2020). These methods are often referred to as *bilevel optimization* procedures, where the *outer* loop updates the hyperparameters on a validation set, and the *inner* one is used for training the models with a specific hyperparameter set.

Recent independent papers introduced the formal paradigm of Algorithm Configuration and HO as a Sequential Decision Process (Biedenkapp et al., 2020; Jomaa et al., 2019), albeit many other works developed solutions in this direction, employing RL-based methods (Zhu et al., 2019; Li and Malik, 2016; Xu et al., 2017; Zoph and Le, 2017) or contextual bandits (Li et al., 2017). However, these works are rarely adopted in RL, as they become computationally intractable and sample inefficient. Furthermore, gradient-based methods (Xu et al., 2018) compute the gradient of the return function with respect to the hyperparameters: they rely on a strong assumption that the update function must be differentiable and that the gradient must be computed on the whole chain of training updates. In addition, these approaches are typically online, with limited exploration (as discussed in (Biedenkapp et al., 2020)), or they make use of gradient-based meta-algorithms, where the high level of sensitivity to new meta-hyperparameters makes the problem even more challenging, as the models may be harder to train and may require more data.

When the specific task of learning rate tuning in a policy-gradient framework, Paul et al. (2019) proposed a sample efficient algorithm to learn a hyperparameter schedule employing a Weighted Importance Sampling approach, while Paine et al. (2020) deals with the offline hyperparameter selection for offline RL. In these proposed approaches, HO is meant to optimize the objective function in the next step, similarly to a bandit problem, which favors convergence to local optima.

The concept of rapid adaptation to unseen tasks is usually denoted as meta-learning (Schmidhuber, 1987), and has recently emerged as a fertile and promising research field, especially with regard to gradient-based techniques. One of the cornerstones in this area is MAML (Finn et al., 2017), which learns a model initialization for fast adaptation and has been a starting point for several subsequent works (Nichol and Schulman, 2018; Park and Oliva, 2019). PEARL (Rakelly et al., 2019) decouples the problem of making an inference on the probabilistic context and solving it by conditioning the policy in meta Reinforcement Learning problems. However, all these works heavily rely on choosing (multiple) learning rates.

4.4 Meta-MDP

We now present the concept of meta-MDP, a framework for solving meta-RL tasks that extends the CMDP definition to include the learning model and the policy parameterization. Similar approaches can be found in Garcia and Thomas (2019) and in Li and Malik (2016). In our case, we assume to be able to fully represent the task by the parameterized context itself $\boldsymbol{\omega} \in \Omega$. The collection of tasks can be framed as a set of MDPs $\{\mathcal{M}_{\boldsymbol{\omega}}\}_{\boldsymbol{\omega}\in\Omega}$, such that each task $\mathcal{M}_{\boldsymbol{\omega}}$ can be sampled from a distribution ψ defined on Δ_{Ω}. This set can be seen equivalently as the CMDP provided with Definition 4.1: $\mathscr{M} := \langle \Omega, \mathcal{S}, \mathcal{A}, \mathcal{M} \rangle$, where $\mathcal{M}(\boldsymbol{\omega}) = \mathcal{M}_{\boldsymbol{\omega}}$. Moreover, we need to define a distribution ρ over the policy space Δ_{Θ}, which provides the initial policy parametrization for each learning instance. In this way, the initial state of the meta-environment is determined by the MDP-policy pair $(\mathcal{M}_{\boldsymbol{\omega}}, \pi_{\boldsymbol{\theta}_0})$, where $\boldsymbol{\omega} \sim \psi$, and $\boldsymbol{\theta}_0 \sim \rho$. The hyperparameter selection in a learning instance of a general optimization algorithm can then be designed as a Meta-MDP, for which we provide the formal definition:

Definition 4.2. *Let $\mathscr{M}$ be a CMDP, and consider a parametric space Ω. A* meta-MDP *over $\mathscr{M}$ with update f is defined as a tuple $\mathfrak{M} := \langle \mathcal{X}, \mathcal{H}, \mathcal{L}, (\mathscr{M}, \psi), (\Theta, \rho), f, \widetilde{\gamma} \rangle$,* *where:*

- $\mathcal{X}$ *is called* meta observation space*;*
- $\mathcal{H}$ *is denoted as* learning action space *or* meta action space
- $\mathcal{L}$ *is the* meta reward function*;*
- $(\mathscr{M}, \psi)$ *and* (Θ, ρ) *contain respectively a CMDP* $\mathscr{M}$ *with distribution over tasks* ψ*, and the policy space, with initial distribution* ρ*;*
- f *is the update rule of the learning model chosen;*
- $\widetilde{\gamma}$ *is a discount factor.*

In particular, a meta-MDP attempts to enclose the general elements needed to learn an RL task into a model with properties similar to a classic MDP. A representation of the main features of the current condition of the learning instance is provided by the task-policy pair $(\mathcal{M}_{\boldsymbol{\omega}}, \pi_{\boldsymbol{\theta}})$. Indeed, the meta-state corresponds to the distribution over the trajectories τ induced by the interaction of the policy $\pi_{\boldsymbol{\theta}}$ with the MDP $\mathcal{M}_{\boldsymbol{\omega}}$, which can be referred to also as *inner-task*. Of course, the inclusion of a complete representation of the trajectories is unfeasible: in other words, the meta-state is hidden, and we will encode it using a set of observed meta-features through the meta observation space $\mathcal{X}$, which can be considered as the generalization of the observation space in classic Partially-Observable MDPs (POMDP, Section 2.2.2). Since it is meant to include information regarding the current condition of the learning process, it is (eventually implicitly) dependent on $\boldsymbol{\theta}$ and the context $\boldsymbol{\omega}$.

Interaction The definition of the meta-MDP tries to design the learning process as a sequential decision-making problem, where a step involves the application of the update rule f, and a meta-agent selects the related hyperparameter. The initial state represents the starting setting for the learning instance: it depends on the joint distribution over the policy space and the set $\mathscr{M}$. We consider an *episodic* setting: each (meta) episode consists of a learning instance, and it is based on a different initial policy and a different task. Moreover, we will consider a very important assumption:

Assumption 4.1 (Episodic context)**.** *Once a context* ω *in a CMDP is sampled from* ψ *to determine the initial state, it is fixed throughout a meta-episode.*

Each action $h_k \in \mathcal{H}$ performed on the meta-MDP with policy parametrization $\boldsymbol{\theta}_k$ at the k-th step, represents a specific hyperparameter that regulates the stochastic update rule f, i.e., $\boldsymbol{\theta}_{k+1} = f\left(\boldsymbol{\theta}_k, h_k, \tau_{\boldsymbol{\omega}}(\boldsymbol{\theta}_k)\right)$, where $\tau_{\boldsymbol{\omega}}(\boldsymbol{\theta}_k)$ is the batch of trajectories sampled under the task-policy pair $(\boldsymbol{\omega}_k, \boldsymbol{\theta}_k)$. For the sake of clarity, we will often drop the dependency on τ when considering the application of the update rule. In general, we can consider any update function with a set of tunable hyperparameters; in particular, we can focus on policy gradient approaches, in which the action h determines the step size, and a new batch of trajectories is iteratively sampled to compute $\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta})$, general estimator of the policy gradient $\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ under context $\boldsymbol{\omega}$. We consider a normalized setting, in such a way that the gradient defines only the update direction, and the action determines the distance from the previous policy. Consequently, the update rule takes the following form:

$$f(\boldsymbol{\theta}, h) = \boldsymbol{\theta} + h \frac{\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta})}{\|\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta})\|_2}.$$

In our experimental campaign in Section 4.7, we will include information from the curvature of the return by employing the natural gradient (NPG, Section 3.5).

Unlike a standard MDP, a meta-MDP does not need to include a Markovian transition model that regulates its dynamics: given $x_t \in \mathcal{X}$, the transition to the next meta-state x_{t+1} is, of course, stochastic, but it is induced by the distribution of the trajectories induced by the pair $(\theta_t, \mathcal{M}_\omega)$ and on the update rule f. The initial state hence implicitly depends on ψ and ρ, and the transition to the next state is still Markovian, as it is independent of the previous states observed (once x_t is known). In other terms, conditional to the current parametrization of the policy and the task, the environment can be considered independent of the past policies encountered during the episode.

Learning as Reward As in a standard RL problem, the optimization of a meta-MDP is accomplished by maximizing the collected rewards. The main goal of HO and meta-learning is learning how to learn: as a consequence, we want to consider performance improvement as our main reward. To accelerate the learning over the current MDP $\mathcal{M}_{\boldsymbol{\omega}}$, this function should reflect variations between the returns obtained in different learning steps. To accomplish this, definition 4.2 included the meta reward $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\omega}, h)$ as a function mainly dependent on the current policy parameters $\boldsymbol{\theta}$ and of the meta-action h once the context $\boldsymbol{\omega}$ is fixed:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\omega}, h) := J_{\boldsymbol{\omega}}(f(\boldsymbol{\theta}, h)) - J_{\boldsymbol{\omega}}(\boldsymbol{\theta});$$

where $J_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ and $J_{\boldsymbol{\omega}}(f(\boldsymbol{\theta}, h))$ are respectively the expected returns in the task $\mathcal{M}_{\boldsymbol{\omega}}$ before and after one update step according to the function f, estimated through a batch of sampled trajectories. In the particular case of (normalized) SGA, the function takes the following form:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\omega}, h) = J_{\boldsymbol{\omega}} \left(\boldsymbol{\theta} + h \frac{\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}}{\|\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}\|_2} \right) - J_{\boldsymbol{\omega}}(\boldsymbol{\theta}).$$

A trajectory can be thought of as an entire learning instance of the selected task, where the length represents the number of policy updates. The overall return of a trajectory with length T can be easily defined as in the general case:

$$\mathcal{G} = \sum_{t=0}^{T-1} \widetilde{\gamma}^t \mathcal{L}(\boldsymbol{\theta}_t, \omega, h_t) = \widetilde{\gamma}^{T-1} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_T) + \sum_{t=1}^{T-1} \widetilde{\gamma}^t J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_t) - J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_0).$$

Consequently, the return of a learning instance corresponds to a weighted sum of the returns of the encountered policies. In particular, if the meta discount factor is set to 1, the return reduces to $J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_T) - J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_0)$, i.e., the total performance gain. Similarly, we can define the meta value functions, which will be denoted as $\mathcal{Q}$ and $\mathcal{V}$.

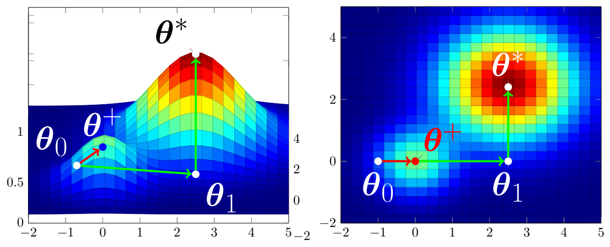


Figure 4.1: *Example of an optimization setting where a Bandit approach would be suboptimal: starting from $\boldsymbol{\theta}_0$, the optimal bandit agent will choose to reach $\boldsymbol{\theta}^+$, a local maximum. An RL agent, however, may plan to make a larger step, up to $\boldsymbol{\theta}_1$, to reach the global optimum $\boldsymbol{\theta}^*$ on the next update. Left Figure: 3D surface plot. Right Figure: objective function heatmap in the parameter space Θ.*

Discount Factor, Contextual Bandit, and Meta-MDP. The choice of the meta-discount factor $\widetilde{\gamma}$ is critical: meta-learning is very often considered as paired with *few-shot learning*, where a short horizon is taken into account for the learning process. $\widetilde{\gamma}$, if lower than 1, explicitly translates into an effective horizon of $\frac{1}{1-\widetilde{\gamma}}$. However, a myopic behavior induced by a low discount factor might lead the meta-agent to prefer actions leading to local optima, while sometimes it might be necessary to take more cautious steps to reach the global optima of the learning process. Setting $\widetilde{\gamma} = 0$, the problem degenerates into a contextual bandit, where the goal is to maximize the immediate reward, in a similar fashion as in Paul et al. (2019). However, it might be inefficient to directly maximize the immediate reward, as an agent might prefer to choose a different hyperparameter to reach the global optimum, which is possibly unreachable in just one step. Figure 4.1 provides an example in this direction, where a bi-dimensional parametrization is considered: starting from the initial parametrization $\boldsymbol{\theta}_0$, the maximization of the immediate return would lead to a local optimum $\boldsymbol{\theta}^+$. We want our agent to be able to plan the updates to maximize the final policy's performance: this is the main reason for the design of HO as a sequential decision-making problem.

Meta-Space Features In this subsection, we deal with the choice of the features observed in the meta-observation x_t. Some properties are generally desirable for its formulation: first of all, it needs to include policy-specific information, as some form of knowledge about the current policy is necessary to adapt the meta-actions to the current setting of the model. Ideally, we can include all current policy parameters $\boldsymbol{\theta}_t$, even if this approach might be difficult for large policy spaces. Finding an informative meta-feature set remains an open problem for future research, as recalled in Section 4.8. Additionally, task-specific features may be informative. The information about the task $\boldsymbol{\omega}$ is used to achieve an *implicit task-identification*, a necessary step to optimize learning in new tasks based on similarities to older ones. Finally, some relevant information could be included in the gradient estimation $\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}$: this vector is implicitly dependent on the stochasticity of the inner MDP $\mathcal{M}_{\boldsymbol{\omega}}$ under policy $\boldsymbol{\theta}_t$ according to the batch of trajectories sampled for its estimation. In our experiments, we will consider the concatenation of all these features $x_t = \langle \boldsymbol{\theta}_t, \widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}, \boldsymbol{\omega} \rangle$. From a more technical point of view, the meta-observation x_t relies on a conditional observation probability $\mathcal{O}(\cdot|\boldsymbol{\theta}_t, \boldsymbol{\omega})$. The transition of the observation process evolves then as follows:

$$\begin{aligned} \boldsymbol{\omega} \sim \psi, \qquad \boldsymbol{\theta}_0 \sim \rho, \qquad & x_0 \sim \mathcal{O}(\cdot|\boldsymbol{\theta}_0, \boldsymbol{\omega}) \\ \boldsymbol{\theta}_t = f(\boldsymbol{\theta}_{t-1}, h_t, \tau_{\boldsymbol{\omega}}(\boldsymbol{\theta}_{t-1})), \qquad & x_t \sim \mathcal{O}(\cdot|\boldsymbol{\theta}_t, \boldsymbol{\omega}) \end{aligned}$$

4.5 Context Lipschitz Continuity

We consider a meta-MDP in which all tasks satisfy the Lipschitz continuity assumption. Under this condition, we can derive a set of bounds on the approximation errors obtained by the meta-agent when acting on unseen tasks. Among others, we obtain that the expected return $J_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ and its gradient are LC w.r.t. the context $\boldsymbol{\omega}$, providing useful theoretical foundations for the meta-RL general framework and inspiring motivation to look for solutions and models capable of generalizing on large task spaces. Let's suppose to be provided with a CMDP $(\Omega, \mathcal{S}, \mathcal{A}, \mathcal{M})$, such that Assumption 3.1 is verified $\forall \boldsymbol{\omega} \in \Omega$, meaning that $\forall \boldsymbol{\omega} \in \Omega$ the MDP $\mathcal{M}_{\boldsymbol{\omega}}$ is $(L_P(\boldsymbol{\omega}) - L_r(\boldsymbol{\omega}))$-LC. Let us also assume that the set of MDPs is LC in the context $\boldsymbol{\omega}$:

Assumption 4.2. *Let $\mathscr{M}$ be a CMDP. $\mathscr{M}$ is called $(L_{\omega_P}, L_{\omega_r})$-Context Lipschitz Continuous (CLC) if, for all $(s,a), (\overline{s}, \overline{a}) \in \mathcal{S} \times \mathcal{A}$, $\forall \boldsymbol{\omega}, \overline{\boldsymbol{\omega}} \in \Omega$:*

$$\begin{aligned} \mathcal{W}_1\left(P_{\boldsymbol{\omega}}(\cdot|s,a), P_{\overline{\boldsymbol{\omega}}}(\cdot|s,a))\right) &\leq L_{\omega_P} d_\Omega(\boldsymbol{\omega}, \overline{\boldsymbol{\omega}}) \\ \left| R_{\boldsymbol{\omega}}(s,a) - R_{\widehat{\boldsymbol{\omega}}}(s,a) \right| &\leq L_{\omega_r} d_\Omega(\boldsymbol{\omega}, \overline{\boldsymbol{\omega}}); \end{aligned}$$

This means that we have some notion of continuity w.r.t. the task: when two MDPs with similar contexts are considered, then their transition and reward processes are similar. These assumptions, along with Assumption 3.2, allow us to make some considerations regarding the inference on the Q-value function:

Theorem 4.1. *Let $\mathscr{M}$ be a $(L_{\omega_P}, L_{\omega_r})$-CLC CMDP for which $\mathcal{M}(\boldsymbol{\omega})$ is $(L_P(\omega), L_R(\omega))$-LC $\forall \boldsymbol{\omega} \in \Omega$. Given a L_π-LC policy π, the action value function $Q^\pi_{\boldsymbol{\omega}}(s,a)$ is L_{ω_Q}-CLC w.r.t. the context $\boldsymbol{\omega}$, i.e.:*

$$\left| Q^\pi_{\boldsymbol{\omega}}(s,a) - Q^\pi_{\overline{\boldsymbol{\omega}}}(s,a) \right| \leq L_{\omega_Q}(\pi) d_\Omega(\boldsymbol{\omega}, \overline{\boldsymbol{\omega}}) \;\; \forall (s,a) \in \mathcal{S} \times \mathcal{A}, \; \forall \boldsymbol{\omega}, \overline{\boldsymbol{\omega}} \in \Omega;$$

where

$$L_{\omega_Q}(\pi) = \frac{L_{\omega_r} + \gamma L_{\omega_p} L_{V_\pi}(\omega)}{1-\gamma}, \qquad L_{V_\pi}(\omega) = \frac{L_r(\omega)(1+L_\pi)}{1-\gamma L_P(\omega)(1+L_\pi)} \tag{4.1}$$

As a consequence, also the return function $J_{\boldsymbol{\omega}}(\pi)$ is context-LC:

$$|J_{\boldsymbol{\omega}}(\pi) - J_{\overline{\boldsymbol{\omega}}}(\pi)| \leq L_{\omega_Q}(\pi) d_\Omega(\boldsymbol{\omega}, \overline{\boldsymbol{\omega}}).$$

In simpler terms, Theorem 4.1 exploits the LC property to derive an upper bound on the distance between the returns in different tasks. This result represents an important guarantee of the generalization capabilities of the approach, as it provides a boundary on the error obtained in testing unseen tasks. This explains the successful behavior of Meta-learning approaches such as MAML (Finn et al., 2017), which typically assume full access to the task distribution ρ. Hence, they can build meta-models capable of quickly adapting to new tasks, with good empirical results for tasks drawn from the same distribution. On the other hand, when test tasks are sampled from a different distribution, the context difference greatly impacts the models learned, and the models can no longer be adequate (Tirinzoni et al., 2018).

Proof for Theorem 4.1 is provided in Appendix A.1, where we also prove that the analytic gradient $\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}^{\boldsymbol{\theta}}$ is CLC w.r.t. the context $\boldsymbol{\omega}$, too. In particular, a boundary on the distance between the gradients of different tasks ensures regularity in the surface of the return function, which is important since the gradient is included in the meta state to capture implicit information about the context space.

As a final consideration, the Lipschitz behavior of the CMDP is inherited by the meta-MDP. The smoothness of this decision problem is transferred from the state-action spaces to the related meta versions. Furthermore, the most important property of a meta-MDP is perhaps more trivial:

Lemma 4.1 (Bounded reward)**.** *Let $\mathscr{M}$ be a CMDP, such that $|R_{\boldsymbol{\omega}}| \leq R_{\max}\ \forall \boldsymbol{\omega} \in \Omega$. Then, a meta-MDP $\mathfrak{M}$ defined over such CMDP has bounded rewards, i.e.*

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\omega}, h) \leq \frac{2}{1-\gamma} R_{max}.$$

The proof of this Lemma is trivial since it relies on the consideration that the expected return $J_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ is bounded by $R_{max}/(1-\gamma)$.

Lemma 4.2 (Lipschitz meta-MDP)**.** *Let $\mathscr{M}$ be a Lipschitz- CMDP, for which Assumptions 3.1 and 3.3 hold. Let the update rule f be L_f-LC, i.e.*

$$\|f(\boldsymbol{\theta}, h) - f(\overline{\boldsymbol{\theta}}, \overline{h})\| \leq L_f d_{\Theta \times \mathcal{H}}((\boldsymbol{\theta}, h), (\overline{\boldsymbol{\theta}}, \overline{h})).$$

Then, the meta-MDP is Lipschitz continuous, i.e., for all $\boldsymbol{\theta}_t, \overline{\boldsymbol{\theta}}_t, h_t, \overline{h_t} \in \Theta^2 \times \mathcal{H}^2$:

$$d_\Theta(\boldsymbol{\theta}_{t+1}, \overline{\boldsymbol{\theta}}_{t+1}) \leq L_f d_{\Theta \times \mathcal{H}}((\boldsymbol{\theta}, h), (\overline{\boldsymbol{\theta}}, \overline{h}));$$
$$|\mathcal{L}(\boldsymbol{\theta}, h) - \mathcal{L}(\overline{\boldsymbol{\theta}}, \overline{h})| \leq (1 + L_f) L_J d_{\Theta \times \mathcal{H}}((\boldsymbol{\theta}, h), (\overline{\boldsymbol{\theta}}, \overline{h})),$$

where L_J is the Lipschitz constant of the return defined in Equation 3.32 (Pirotta et al., 2015), supposed uniform over Θ.

Proof. We report here only the proof of the Lipschitz meta-reward, for which we rely on the smoothness of the update rule, and on the Lipschitz guarantees provided for general MDPs. The proof of the smoothness of the transition is even simpler and not reported.

$$\begin{aligned}
|\mathcal{L}(\boldsymbol{\theta}, h) - \mathcal{L}(\overline{\boldsymbol{\theta}}, \overline{h})| &= |J_{\boldsymbol{\omega}}(f(\boldsymbol{\theta}, h)) - J_{\boldsymbol{\omega}}(\boldsymbol{\theta}) - J_{\boldsymbol{\omega}}(f(\overline{\boldsymbol{\theta}}, \overline{h})) + J_{\boldsymbol{\omega}}(\overline{\boldsymbol{\theta}})| \\
&\leq |J_{\boldsymbol{\omega}}(f(\boldsymbol{\theta}, h)) - J_{\boldsymbol{\omega}}(f(\overline{\boldsymbol{\theta}}, \overline{h}))| + |J_{\boldsymbol{\omega}}(\boldsymbol{\theta}) - J_{\boldsymbol{\omega}}(\overline{\boldsymbol{\theta}})\| \\
&\leq (1 + L_f) L_J d_{\Theta \times \mathcal{H}}((\boldsymbol{\theta}, h), (\overline{\boldsymbol{\theta}}, \overline{h}))
\end{aligned}$$

□

Algorithm 5 Meta-MDP Dataset Generation

```
Require: CMDP $\mathscr{M}$ (with task distribution $\psi$, policy space $\Theta$, initial policy distribution $\rho$),
  number of learning instances $K$, number of learning steps $T$, number of inner trajectories $n$.
Ensure: dataset of transitions $\mathcal{D}$
  Initialize: $\mathcal{D} = \{\}$,
  for $k = 1, \dots, K$ do
    Sample context $\boldsymbol{\omega} \sim \psi(\Omega)$, initial policy $\boldsymbol{\theta}_0 \sim \rho(\Theta)$
    Sample $n$ trajectories in task $\mathcal{M}_{\boldsymbol{\omega}}$ under policy $\pi(\boldsymbol{\theta}_0)$
    Estimate $J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_0)$, $\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_0)$
    for $t = 0, \dots, T-1$ do
      Sample meta-action $h \in \mathcal{H}$
      Update policy
          $$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + h \frac{\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_t)}{\|\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_t)\|}$$
      Sample $n$ trajectories in $(\mathcal{M}_{\boldsymbol{\omega}}, \pi(\boldsymbol{\theta}_t))$
      Estimate $J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_{t+1})$, $\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_{t+1})$
      Set
          $$x = \langle \boldsymbol{\theta}_t, \widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_t), \boldsymbol{\omega} \rangle;$$
          $$x' = \langle \boldsymbol{\theta}_{t+1}, \widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_{t+1}), \boldsymbol{\omega} \rangle;$$
          $$l = J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_{t+1}) - J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_t).$$
      Append $\{(x, h, x', l)\}$ to $\mathcal{D}$
    end for
  end for
```

Remark 4.3. *This result considers only the smoothness w.r.t. the parametrization $\boldsymbol{\theta}$ and the hyperparameter h; We did not include the variations w.r.t. the context $\boldsymbol{\omega}$, since we deal with the assumption of a fixed context, the context has no impact on the transition and reward processes. Of course, these results can be extended to consider also the context $\boldsymbol{\omega}$.*

Lemma 4.2 seems to provide nice properties to the overall learning decision process: when the task and the context are similar, the next updates will not differ too much. However, it relies on the major assumption of smoothness of the update rule. Unfortunately, this holds only for the exact policy gradient, which is proven to be Lipschitz continuous w.r.t. the parametrization (Theorem 3 in Pirotta et al. 2015) and the context (Theorem A.4). However, we cannot rely on such properties when estimators are considered, as the noisy trajectories can lead to very different results.

4.6 Fitted Q-Iteration on Meta-MDP

In this section, we define our approach to learn a dynamic learning rate in the framework of a meta-MDP. As a meta-RL approach, the objectives of our algorithm are to improve the generalization capabilities of PG methods and to remove the need to tune the learning rate for each task manually. Finding an optimal dynamic step size increases the number of degrees of freedom w.r.t. a static learning rate, and it mainly serves two purposes: it maximizes the convergence speed by performing large updates when allowed and improves the overall training stability by selecting low values when the return is close to the optimum or the current region is uncertain. To accomplish these goals, we propose the adoption of the FQI algorithm (Ernst et al., 2005) presented in Section 3.4.3 (Algorithm 2) to the problem of meta-hyperparameter optimization. We will refer to this methodology as *MetaFQI*. As seen, FQI is an off-policy and offline algorithm designed to learn a good approximation of the optimal action-value function by exploiting the Bellman optimality operator. The approach consists in applying Supervised Learning techniques as, in our case, Extra Trees (Geurts et al., 2006), to generalize the estimation of the optimal action-value function $\mathcal{Q}^\star$ over the entire state-action space. The algorithm considers a full dataset $\mathcal{F} = \{(x_t^k, h_t^k, l_t^k, x_{t+1}^k)\}_k$, where each tuple represents an interaction with the meta-MDP: in the $k-$th tuple, x_t^k and x_{t+1}^k are respectively the current and next meta-state, h_t^k the meta-action and l_t^k the meta reward function, as described in Section 4.4. To consider each meta-state x, there is the need to sample n trajectories in the inner MDP to estimate return and gradient. At the iteration N of the algorithm, given the (meta) action-value function $Q^{(N-1)}$ of the previous iteration, the training set $TS_N = \{(i^k, o^k)\}_k$ is built, where each input is equivalent to the state-action pair $i^k = (x_t^k, h_t^k)$, and the target is the result of the Bellman optimal operator (definition 2.13):

$$o^k = l_t^k + \widetilde{\gamma} \max_{h \in \mathcal{H}} Q^{(N-1)}(x_{t+1}^k, h).$$

In this way, the regression algorithm adopted is trained on TS to learn $Q^{(N)}$ with the learning horizon increased by one step.

In general, the dataset is created by following K learning trajectories over the CMDP: at the beginning of each meta-episode, a new context $\boldsymbol{\omega}$ and initial policy $\boldsymbol{\theta}_0$ are respectively sampled from ψ and ρ; then, for each of the T learning steps, the meta action h is randomly sampled to perform the policy update (we consider a uniform distribution over the meta action space $\mathcal{H}$). In this way, the overall dataset is composed of KT tuples. It is also possible to explore the overall task-policy space $\Omega \times \Theta$ through a generative approach: instead of following the learning trajectories, both $\omega, \boldsymbol{\theta}$ and h are sampled every time (equivalent to 1-step long trajectories). We will refer to this last method as "generative" approach, while the former will be referred to as "trajectory" approach. The pseudo-code for the dataset generation process with trajectories is provided in Algorithm 5.

Double Clipped Q Function As mentioned, each FQI iteration approximates the action-value function using the estimates made in the previous step. As the process goes on, the sequence of these compounding approximations can degrade the overall performance of the algorithm. In particular, FQI tends to suffer from overestimation bias, similar to other value-based approaches that rely on taking the maximum of a noisy Q function. To countermeasure this tendency, we adopt a modified version of Clipped Double Q-learning, introduced by Fujimoto et al. (2019) and mentioned in Section 3.4.3, to penalize uncertainties over future states. This approach consists in maintaining two parallel functions $Q_{\{1,2\}}^{(N)}$ for each iteration and choosing the action h maximizing a convex combination of the minimum and the maximum between them:

$$l + \tilde{\gamma} \max_{h \in \mathcal{H}} \left[\lambda \min_{j=1,2} Q_j^{(N)} \left(\overline{x}, h \right) + (1 - \lambda) \max_{j=1,2} Q_j^{(N)} \left(\overline{x}, h \right) \right],$$

with $\lambda > 0.5$. If we set $\lambda = 1$, the update corresponds to Clipped Double Q-learning. The minimum operator penalizes high variance estimates in regions of uncertainty and pushes the policy towards actions that lead to states already seen in the dataset.

The overall procedure introduces external hyperparameters, e.g., the number of decision trees, the minimum number of samples for a split (*min split*), and λ. However, the sensitivity on these parameters is minimal (Geurts et al., 2006), as a different set of hyperparameters does not impact the ability of FQI to converge. This fact helps us in avoiding a conceptual problem related to the selection of a gradient-based meta-algorithm, which usually shows a very high sensitivity w.r.t. the network adopted and the hyperparameters used: trying to optimize the meta hyperparameters introduced would make the whole approach useless since the same problem of hyperparameter optimization would only be set on a more abstract level.

4.7 Experimental Evaluation

In this section, we show an empirical analysis of the performance of our approach in different environments. As we shall see, the meta action allows selecting the best step size and dynamically adapting it to fine-tune the learning procedure. As metaFQI iterations proceed, new estimation errors are gradually introduced, resulting in overfitting of the model (with the *target loss* minimized on the training dataset) and consequently in a degradation of out-of-sample performances over time. This is due to the error propagation w.r.t. the optimal $\mathcal{Q}-$value function in the whole state-action space (and task space, in our case), as in Farahmand et al. (2010). Consequently, the model iterations are evaluated in a validation process, as in the standard model selection procedure, on a set of out-of-sample tasks and policies. From this set, the model obtaining the best mean return, said N is selected. The results of the selected models are shown in Figure 4.2, along with NGA performed with fixed step size, tested on the same 20 trials (i.e., on the same random test tasks and initial policies), and performed with the same batch size for each trial. Our code is based upon OpenAI Gym (Brockman et al., 2016), and Baselines (Dhariwal et al., 2017) toolkits.

Navigation2d: For our first evaluation of the approach, we consider one of the environments presented in Finn et al. (2017), called Navigation2D. This environment consists of a unit square space where an agent aims to reach a random goal in the plane. The task distribution is such that, at each episode, a different goal point is uniformly selected in the unit square. As we can note in the left plots of Figure 4.2, the algorithm is able to select large step sizes with a good starting return gain without suffering from any drop. The algorithm is able to calibrate its action, starting with larger improvements and slowing down once the policy gets good results. In addition, all trajectories reach convergence in fewer steps than any other method.

Minigolf: In our second experiment, inspired by Penner (2002) and Tirinzoni et al. (2019), we consider the scenario of a flat minigolf green, in which the agent has to hit the ball with a putter and place the ball inside the hole in the minimum number of strokes. The CMDP is built by varying the putter length and the friction coefficient.

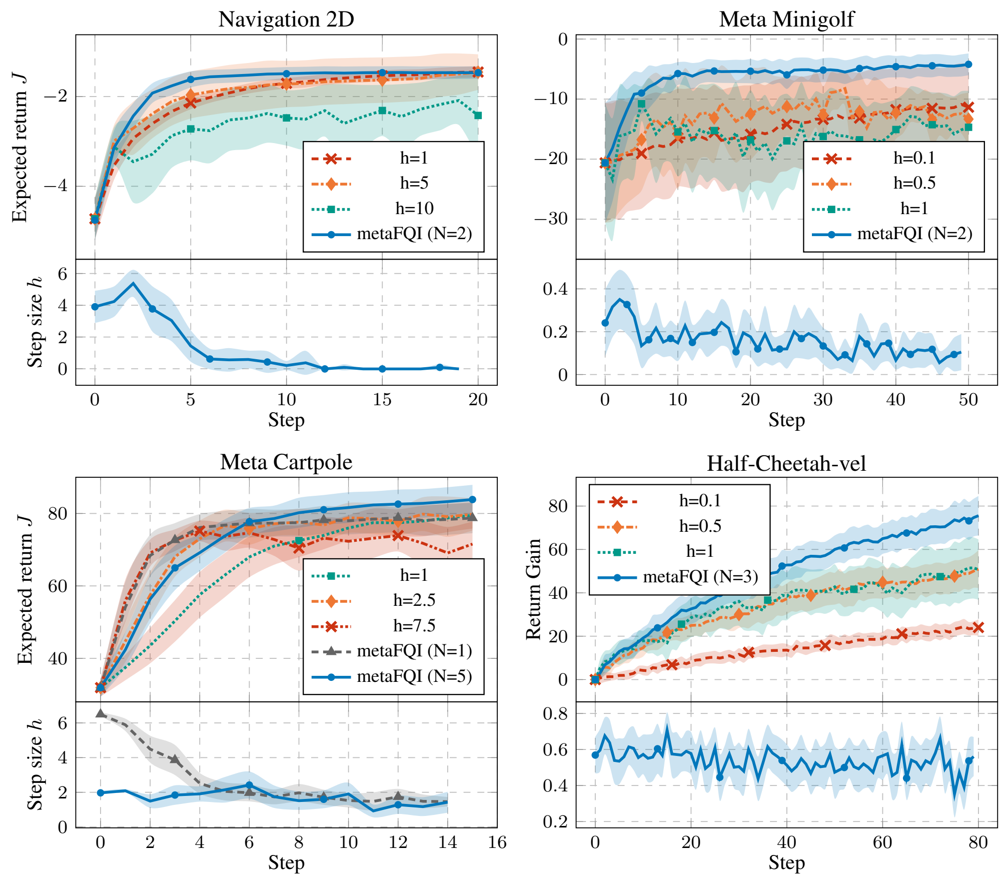


Figure 4.2: *metaFQI model performance against NGA with fixed step size h. The top plots for each environment show the expected returns or the return gain. The bottom plots show the meta-actions chosen through learning iterations. N represents the metaFQI iteration selected. (20 runs/random test contexts, avg $\pm$ 95 % c.i.)*

The environment is Lipschitz w.r.t. the context, but it is the only framework where the reward is non-Lipschitz since, for each step, it can be either 0 if the shot is a success, -1 if the ball does not reach the goal (and the episode continues) or -100 for overshooting. The central plot in Figure 4.2 illustrates the performance of our approach in the same set of random test tasks. We can see that the algorithm is able to consistently reach the optimal values by choosing an adaptive step size. In addition, the convergence to the global optimum is achieved in around 10 meta steps of training, a substantial improvement w.r.t. the choice of a fixed learning rate, which leads (when it converges) to a local minimum, meaning constantly undershooting until the end of the episode.

CartPole: For our third experiment, we examine the CartPole balancing task (Barto et al., 1990), which consists of a pole attached to a cart, where the agent has to move to balance the pole as long as possible. The CMDP is induced by varying the pole mass and length. To be more focused on the very first steps and to better generalize on the overall policy and task space, the training dataset was built considering trajectories with only 15 total updates. To have a fair comparison, the right-hand side plots of Figure 4.2 illustrate an evaluation of the approach in the selected environment, where we have

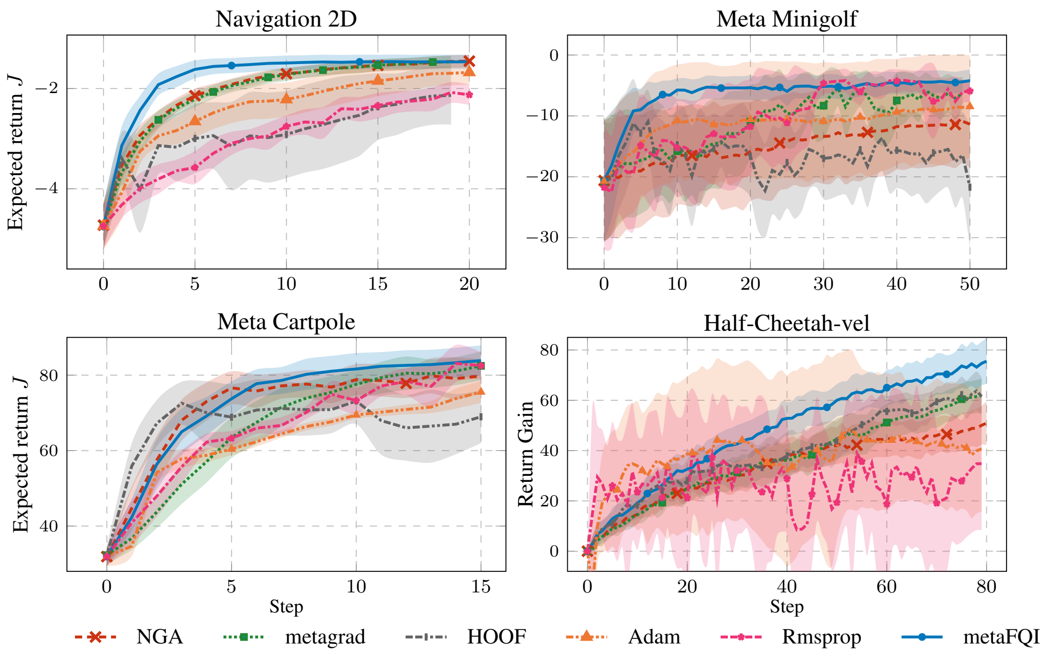


Figure 4.3: *metaFQI model performance comparison against benchmarks (20 runs, 95% c.i.).*

tested the resulting metaFQI model (and NGA with fixed step sizes) performing the same number of total updates as the training trajectories.[1] In Appendix A.2, we provide more results where the models are tested for a longer horizon $T = 60$.

Differently from before, it is possible to see that the best model (solid blue line) is choosing to update the policy with small learning rates: this leads to a lower immediate return gain (high rates have a better learning curve in the first steps) but allows to improve the final performances. This is because the model is planning to maximize the meta action value function with a horizon of $N = 5$ policy updates. Indeed, we also included the results of the first metaFQI iteration, which tries to optimize the immediate reward similarly as a contextual bandit agent. As expected, the agent selects high step sizes for the first iterations, obtaining high immediate rewards only in the first learning steps.

Half-cheetah with goal velocity: As a last environment, we considered the half-cheetah locomotion problem introduced in Finn et al. (2017) with MuJoCo simulator (Todorov et al., 2012), where a planar cheetah has to learn to run with a specific goal velocity. This is the most complex environment among the ones presented since the policy, albeit linear, comprises 108 parameters. From the rightmost plot of Figure 4.2 we can see the performance gain $J(\boldsymbol{\theta}_t) - J(\boldsymbol{\theta}_0)$.[2] The FQI model, trained with NGA trajectories with $T = 80$ total updates, is learning faster than benchmarks, although far from convergence. Interestingly, the fixed learning rates between 0.5 and 1 have almost the same gain return; however, even though the meta actions chosen by the model are almost always within this range, it can still adapt the steps to get higher meta rewards.

[1] Being this environment an alteration of the classic Cartpole, different contexts may reach different convergence returns from the standard policy gradient results.

[2] The expected return changes deeply w.r.t. the task $\boldsymbol{\omega}$, hence the learning curves as in the other plots in Figure 4.2 show very high variance, independently from the robustness of the learning process.

Since all these models are far from convergence, one might wonder if metaFQI can still provide good results for a longer horizon: in Appendix A.3, Figure A.5 we provide more results, where the models are tested for a longer horizon $T = 60$. In brief, if metaFQI models have never seen advanced policies in the training dataset, performance can be negatively affected. To overcome this issue, the learning trajectories sample to produce the batch of tuples should be increased with a sufficiently long horizon.

Comparison with Learning Rate Schedules and Other Benchmarks. In Figure 4.2 we compared our approach with the choice of a fixed step size. Other schedules are often considered. There are, of course, many different gradient descent optimization algorithms for the choice of the learning rate, and among the most widely adopted there are RMSProp and Adam (Kingma and Ba, 2014). The former considers an adaptive learning rate by introducing a momentum term and normalizing the step direction through a moving average of the square gradient. Adam (Adaptive Moment Estimation) also takes advantage of the exponentially decaying average of the second moments of the gradients. We compared our results (*metaFQI*) against tuned implementations of the update rules mentioned (*Adam*, *RMSProp*) and with the best-fixed stepsize (*NGA*). Moreover, we include in the comparison also two other benchmarks for learning rate adaptation: *HOOF, Paul et al. (2019)* and *metagrad, Xu et al. (2018)*, which have been implemented to optimize the stepsize for NGA (more details are included in Appendix A.2). the results are shown in Figure 4.3, in the same settings as the ones provided in Section 4.7. Our approach outperforms the previous methods, showing better learning with, in general, lower variance in the returns obtained. Furthermore, all the considered benchmarks heavily rely on the initial stepsize chosen and on the outer meta-hyperparameters, which deeply affect the learning capabilities.

In conclusion, many other benchmarks and questions are analyzed in Appendix A.3. for instance, we compared the different iterations of metaFQI (Figure A.2) and the results provided by the commonly used exponentially decaying learning rate schedule (Figure A.3).

4.8 Conclusions

In this chapter, we considered the problem of hyperparameter tuning for policy gradient-based algorithms in Contextual Markov Decision Processes, where heterogeneous contexts may require different solutions. In particular, we modeled the general problem through the definition of the meta-MDP, for which any policy-based update rule can be optimized by the choices made by an agent where learning is the reward. We analyzed the case of Lipschitz meta-MDPs, deriving some general guarantees that hold if the reward model and the transition process are smooth with respect to the context. Finally, we implemented the Fitted Q-Iteration algorithm on the meta-MDP where the update rule is the Natural Gradient Ascent, and we used it to choose an adaptive step size through the learning process. The approach has been evaluated in different settings, where we observed good generalization capabilities of the model, which can reach fast convergence speed and robustness without the need for manual hyperparameter tuning.

There are many important challenges to be still addressed for this approach to be effective in real-life applications. First of all, this method can be theoretically extended to any hyperparameter choice. One direct extension of our approach can be applied to the choice of the max Kullback-Leibler divergence constraints in Trust-Region-based approaches (Schulman et al., 2015, 2017), as will be done in Chapter 5. Moreover, the main limitation of our current approach is the same as for many hyperparameter tuning approaches, and it consists of the computational time required to build the training dataset. One possible way to improve the sample efficiency might consist in evaluating the meta-reward by means of importance sampling, similar to Paul et al. (2019). In realistic settings, where deep policies are required, the inclusion of all policy parameters in the meta-state might be inefficient; a solution might consist in learning a compressed representation of the policy through the choice of specific informative meta-features: in this way, our approach would be independent on the policy architecture and scalable for large domains. This last topic, as the application of the meta-MDP approach on a different kind of hyperparameter, is studied in more detail in Chapter 5.

CHAPTER 5

Trust Region Meta Learning for Policy Optimization

5.1 Introduction

In Chapter 4 we introduced the concept of the Meta Markov Decision Process, an abstract formalism to design the problem of hyperparameter tuning as a Sequential Decision Making problem, where the hyperparameters of a specific update rule are seen as the actions of a meta agent. RL aims to train autonomous agents in their interaction with the environment, maximizing a given reward signal. Hence we considered the Fitted Q-Iteration to solve the meta-MDP, where learning is the (meta) reward. In particular, the resulting Meta-Reinforcement Learning algorithm was applied to the step size selection for a gradient-based optimization process. However, the meta-environment was designed to include the observation of the whole policy parametrization, as well as the explicit knowledge of the task: these are strong limitations, as the problem becomes strongly dependent on the neural architecture adopted for the parametrization of the policy, being consequently exposed to the *curse of dimensionality*. In this chapter, we aim to optimize the trust region constraint of the well-known TRPO algorithm (Schulman et al., 2015). Furthermore, the goal is to consider a set of *policy-agnostic* observed features by relying on the information experienced from the interaction with the environment. In this way, we can build a single meta-model capable of predicting the optimal hyperparameter for TRPO at each step, generalizing across different tasks and policy spaces.

Chapter Outline This chapter is organized as follows: Section 5.2 analyses the main motivations behind the choice of a different design of the meta-MDP, applied to overcome the limitations due to the inclusion of the complete policy parametrization provided in Chapter 4. Moreover, we recall some of the main properties of TRPO and the selection of a trust region constraint, which is the main goal of this chapter. The conclusion of the section is devoted to a brief presentation of some information theory concepts, which will be applied later. The main methodology is explained in more detail in Section 5.3, where we devote particular attention to the selection of an informative set of meta-features, which can be estimated through some *nearest neighbors* techniques described later, in Section 5.4. The experimental campaign is then shown in Section 5.5, with some explicative domains.

5.2 Motivations

In Chapter 4, we presented the Meta-MDP approach to frame the hyperparameter optimization problem for RL as an abstract Sequential Decision-Making process. In the design of a possible application of this concept to the optimization of (natural) policy gradient steps, we included in the set of observations for each learning step the following features:

- The current parametrization $\boldsymbol{\theta}_t \in \Theta$, where Θ identifies the *policy class*;
- The explicit parametrization of the context $\boldsymbol{\omega} \in \Omega$;
- The current estimation of the (natural) gradient of the return function w.r.t. the current parameters, $\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_t)$, assessed through a batch of trajectories sampled under the policy-task pair $(\boldsymbol{\theta}_t, \boldsymbol{\omega})$.

Albeit the good results achieved from this approach, there are some assumptions and drawbacks that limit its range of application:

- The context might be unknown, not observable, and non-stationary. for instance, if we think of a locomotion task, a robot might be influenced by the road surface, by weather factors, and by the inner mechanical structure that can slightly change during the transitions. However, the whole composition of such features and their impact on the transition dynamics can be unknown or cannot be measured. However, while this is indeed a limitation, the overall approach can also be considered as *task-agnostic*, as the exclusion of the explicit context parametrization still provides good empirical results, as seen in Section A.2.
- The inclusion of the complete parametrization can be effective only with reduced dimensionality. Real-world RL applications usually involve policies parametrized with multi-layer perceptron (MLP) with several hidden layers and thousands (if not millions) of parameters. Trivially, this meta-feature design is subject to the *curse of dimensionality*, with bad scaling properties concerning the policy space considered.
- Moreover, practitioners usually deal with multiple architectures at once: the number of hidden layers, neurons, and the choice of activation function are often considered as hyperparameters that are subject to a coarse tuning to trade-off the better approximation capabilities of large neural networks, with the related difficulty of training. Very often, taking into account a different policy architecture means

that all the learning processes performed to tune the hyperparameters of the learning algorithm are no longer valid, and the HO procedure must start again from scratch. In our case, including the policy parametrization and the policy gradient in the observation, not only causes dimensionality issues as previously mentioned, but also limits the range of application to a fixed policy class since the consequent meta-MDP becomes architecture-dependent.

- The final limitation is related to the inclusion of the policy gradient, which can be reasonably adopted only when policy-gradient-based update rules are taken into account, and it is usually very noisy, even when the natural gradient is taken into account.

One of the possible solutions that can come to mind to overcome the dimensionality issue involves the trivial application of *dimensionality reduction* techniques, such as Principal Component Analysis (PCA) or autoencoders. However, while these approaches might work in practice, they still provide architecture-dependent features, as different policy classes may require different applications of these methods. A completely different approach can be applied if we consider that the meta-agent is not responsible for finding the best parameters of the problem but for selecting the hyperparameter that provides the best learning improvements for the update rule taken into account. The improvement is due to a better distribution of the trajectories induced by the interaction of the environment (and its context) and the policy, where the term *better* can be related to an improvement of the direct performances or of the information gathered for future updates. for instance, we might think of a policy update leading to a more explorative policy, with slightly worse performances but with the ability to collect more useful information for future performance exploitation. Hence, the real object of interest for the meta-agent is not the perfect knowledge of the task and the policy, but a representation of the *trajectory distribution* $\rho_{\boldsymbol{\theta},\boldsymbol{\omega}}$, which can be defined generalizing Equation 2.4 to include also the context $\boldsymbol{\omega}$:

$$\rho_{\boldsymbol{\theta},\boldsymbol{\omega}}(\tau) := \mu_{\boldsymbol{\omega}}(s_0) \prod_{t=1}^{T} \pi_{\boldsymbol{\theta}}(a_t|s_t) P_{\boldsymbol{\omega}}(s_{t+1}|a_t, s_t).$$

The main goal of this chapter is hence to provide a set of information related to the distribution of trajectories, which is jointly induced by the policy parameters $\boldsymbol{\theta}$ and the context $\boldsymbol{\omega}$, without the explicit knowledge of either. Consequently, the approach provided can be *task-agnostic* and does not need the Assumption 4.1 of a fixed context throughout the learning instances: if the task implicitly changes among the steps, the trajectory distribution observed is modified accordingly. Moreover, it is *policy-agnostic*, or *architecture-agnostic*, meaning that it is not limited to selecting a specific policy class but can include the distribution induced by multiple parametrizations all at once. Considering the trajectory distribution instead of the task-policy parametrization is appealing but has major difficulties, as it cannot be analytically observed. In Section 5.3 we consider a set of *meta-features* that tries to extract some related useful information related to such distribution.

5.2.1 Trust Region Policy Optimization.

Besides selecting a different set of observations, this chapter aims to propose a Meta-RL approach aimed at optimizing the hyperparameters of a different learning algorithm, TRPO (Schulman et al., 2015). As already seen in Section 3.5, at each iteration of the optimization process, TRPO defines a *Trust Region* around the previous solution, then builds a local approximation of the performance measure and searches for the next candidate agent inside the trust region.

TRPO generally provides good empirical performances in many different kinds of tasks but heavily relies on the trust region constraint, as it may have a huge impact on the learning capabilities of the algorithm. Another important aspect of the trust region constraint, and perhaps the biggest difference w.r.t. the selection of the stepsize for SGA updates, is that it is usually *fixed across all the learning steps*: since we will try to adapt the hyperparameter selection throughout the whole learning process automatically, we provide new degrees of freedom, having the possibility of a change at each TRPO learning step.

As already explained in the explanation provided in Section 3.5, the TRPO update rule attempts to solve a trust region subproblem introduced in 3.28, where the goal is to maximize the *surrogate objective function* $L_{\boldsymbol{\theta}_t}(\boldsymbol{\theta})$ (Equation 3.27) among the policies $\pi_{\boldsymbol{\theta}}$ within the trust region of the policy space with radius λ and center in the previous policy $\boldsymbol{\theta}_t$. As seen, this problem admits a closed-form solution, as in 3.29. Here we provide the context-based version of the update rule, where the policy gradient and the FIM simply include the dependency w.r.t. the context $\boldsymbol{\omega}$:

$$\boldsymbol{\theta}_{t+1} = f(\boldsymbol{\theta}_t, \boldsymbol{\omega}, \lambda) = \boldsymbol{\theta}_t + \alpha(\lambda, \boldsymbol{\omega}) F_{\boldsymbol{\omega}}(\boldsymbol{\theta})^{-1} \nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}),$$

where λ is the trust region constraint, or *trust radius*, and where the initial stepsize $\alpha(\lambda, \boldsymbol{\omega})$ is computed as:

$$\alpha(\lambda, \boldsymbol{\omega}) := \sqrt{\frac{2\lambda}{\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta})^\top F_{\boldsymbol{\omega}}(\boldsymbol{\theta})^{-1} \nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta})}}. \tag{5.1}$$

While the theoretical nature of TRPO provides strong monotonic improvement guarantees, the practical algorithm introduces several approximations: as shown in Algorithm 4, the TRPO implementation requires two phases for each iteration: The first phase adopts an iterative conjugate gradient optimization to find the search direction, to approximate the *natural gradient*, i.e., a variation of Equation 3.25 by means of the Fisher information matrix (Kakade, 2001).In this way, it is possible to estimate the update direction and the initial stepsize $\alpha(\lambda)$. During the second phase, it levers a criteria-based line-search to obtain a feasible step size; starting from a maximum threshold following the direction found in the previous phase, the stepsize is halved if the trust region constraint is not satisfied or if the surrogate function does not predict a return improvement.

Furthermore, the hyperparameter λ is kept constant for all iterations, but this may be a major limitation: the trust region concept imposes a limit in which the surrogate function $L_{\boldsymbol{\omega}_t}(\boldsymbol{\omega})$ is trusted to provide a good approximation of the return $J_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ and to improve the real performances; however, different performance surfaces in the parameters space might have a different complexity of approximation: a very smooth $J_{\boldsymbol{\omega}}(\boldsymbol{\theta})$

surface can be easily approximated within a large trust-region, while the approximation of a stiff return manifold cannot be trusted in the same area. At the same time, always imposing a very small trust region makes the algorithm slower, giving up on some advantages of using TRPO instead of a classical PG gradient algorithm.

A fixed-size trust region also has some more subtle consequences on the effects of the algorithm: the double-checking procedure carried out during the second phase at each iteration is needed because of the approximations errors introduced with the estimations, and its accuracy depends on the trust region constraint; the shrinking steps during this second phase are mainly applied to mitigate the bad effects of a too big trust-region. The shrinking process is thus very conservative, slowing down the algorithm. Our goal in this chapter, besides the selection of an informative and dimensionally adequate set of meta-features, is to provide a robust method to find an adaptive selection of the trust region, which can be trusted without the need for a second check: as a consequence, in our implementation of the solution, we modified the *double check* procedure by not checking whether the surrogate function predicts an improvement. The version of Algorithm 4 where the double-check (light blue box) is simplified as $D_{KL}^{max}(\pi_{\boldsymbol{\theta}_t}||\pi_{\widetilde{\boldsymbol{\theta}}}) > \lambda$ will be denoted in the following as xTRPO.

5.2.2 Information Theory

In this section, we provide some basic concepts and mathematical tools of Information Theory, which has become very influential in many engineering and scientific fields. The main source, directly or indirectly used for building this section, is (Shannon, 1948). The basic intuition behind Shannon's information theory relies on the information provided by the likelihood of an event, as *"to know that an unlikely event has occurred is more informative than knowing that a likely event has occurred"* (Goodfellow et al., 2016). Given a density f over the space of events $\mathcal{X}$, Shannon Information aims at quantifying the informativeness (unlikeliness) of an event $x \in \mathcal{X}$ through its likelihood $-\log f(x)$.

Entropy Entropy is a measure that describes the amount of expected information related to the distribution f:

Definition 5.1 (Differential Entropy). *Let $\mathcal{X}$ be a measurable space, and let $f \in \Delta_{\mathcal{X}}$ be any distribution defined over $\mathcal{X}$. The* differential entropy *(or just entropy) $H(f)$ of f is the average information of the events drawn from the distribution:*

$$H(f) = \mathop{\mathbb{E}}_{x\sim f}[h_f] = -\mathop{\mathbb{E}}_{x\sim f}[\log f(x)]$$

The formal interpretation of such measure is related to the amount of information gathered by f, i.e., the minimum average amount of bits needed to encode an event sampled from the distribution f. An interesting property of the entropy is related to *conditional distributions*, as it is possible to rewrite the conditional entropy in a useful way: first of all, we provide the definitions of *joint* and *conditional* entropy:

Definition 5.2 (Joint Entropy and Conditional Entropy). *Let $\mathcal{X}, \mathcal{Y}$ be measurable spaces, and let $f_j \in \Delta_{\mathcal{X}\times\mathcal{Y}}$ be a joint distribution defined over $\mathcal{X} \times \mathcal{Y}$. The* joint entropy $H(f_j)$ *of f_j is defined as:*

$$H(f) = -\mathop{\mathbb{E}}_{(x,y)\sim f_j}[\log f_j(x, y)];$$

Moreover, let f_c the conditional distribution of $y|x$ for any $(x, y) \in \mathcal{X} \times \mathcal{Y}$. The conditional entropy *of $\mathcal{Y}$ given $\mathcal{X}$ under f_c is defined as:*

$$H(f_c) = - \mathop{\mathbb{E}}_{(x,y)\sim f_j} [\log f_c(y|x)].$$

While the joint entropy represents the amount of information needed in expectation to specify the joint outcome of the two values, the conditional entropy instead represents the expected extra information needed to specify the outcome of $\mathcal{Y}$ conditioned to the knowledge of the outcome $x \in \mathcal{X}$. A basic relation that we will use in the following section is the following:

Fact 5.1. *Let $\mathcal{X}, \mathcal{Y}$ be two measurable spaces, with marginal distributions $f_x \in \Delta_{\mathcal{X}}$ and $f_y \in \Delta_{\mathcal{Y}}$. Let f_j and f_c be the joint and conditional distributions, respectively. Then:*

$$H(f_c) = H(f_j) - H(f_x). \tag{5.2}$$

Cross Entropy Cross Entropy is a measure that builds upon both the notions of entropy and information and attempts to measure the amount of information required to encode the difference between two distributions.

Definition 5.3 (Cross Entropy)**.** *Let $\mathcal{X}$ be a measurable space, and let $f_1, f_2 \in \Delta_{\mathcal{X}}$ be two distributions defined over $\mathcal{X}$ such that $f_1 \ll f_2$, i.e., f_1 is absolutely continuous w.r.t. f_2. The* cross entropy *of f_2 from f_1 is defined as*

$$C(f_1, f_2) = \mathop{\mathbb{E}}_{x\sim f_1} [h_{f_2}(x)] = - \mathop{\mathbb{E}}_{x\sim f_1} [\log f_2(x)] \, .$$

The usual interpretation of this measure is the minimum amount of information needed to efficiently encode the events, meaning that the cross-entropy, in a sense, measures the information of the events in $\mathcal{X}$ encoded as if they were sampled from f_2, while they were drawn from f_1. Trivially, if the two distributions are a.e. identical, the amount of information is the same, and the cross entropy is reduced to the entropy, i.e., $C(f_1, f_2) = H(f_1) = H(f_2)$. If the two distributions are different, the cross-entropy increases accordingly.

A similar property as Fact 5.1 holds for the cross conditional entropies:

Fact 5.2. *Let $\mathcal{X}, \mathcal{Y}$ be two measurable spaces, with two marginal distributions $f_x^1, f_x^2 \in \Delta_{\mathcal{X}}$ and $f_y^1, f_y^2 \in \Delta_{\mathcal{Y}}$. Let f_j^1 and f_c^1 be respectively the joint and conditional distributions related to f_x^1 and f_y^1, and analogously for f_j^2 and f_c^2. Then:*

$$C(f_c^1, f_c^2) = C(f_j^1, f_j^2) - C(f_x^2, f_x^2). \tag{5.3}$$

Kullback-Leibler divergence Similarly to cross-entropy, the Kullback-Leibler divergence (KL) is commonly adopted to describe the divergence between two distributions, as it measures the loss of information due to encoding events drawn from a distribution as if they were drawn from a different one:

Definition 5.4 (Kullback-Leibler Divergence)**.** *Let $\mathcal{X}$ be a measurable space, and let $f_1, f_2 \in \Delta_{\mathcal{X}}$ be two distributions defined over $\mathcal{X}$ such that $f_1 \ll f_2$, i.e., f_1 is absolutely continuous w.r.t. f_2. The* cross entropy *of f_2 from f_1 is defined as*

$$D_{KL}(f_1 \;||\; f_2) = \mathop{\mathbb{E}}_{x\sim f_1} \left[\log \frac{f_1(x)}{f_2(x)} \right] .$$

This measure is strictly related to the cross-entropy since:

$$D_{\text{KL}}(f_1 \mid\mid f_2) = C(f_1, f_2) - H(f_1). \tag{5.4}$$

As a consequence of the linear relation w,r,t, entropy, and cross-entropy, for KL divergences, the same properties of conditional distributions hold:

Fact 5.3. *Let $\mathcal{X}, \mathcal{Y}$ be two measurable spaces, with two marginal distributions $f_x^1, f_x^2 \in \Delta_{\mathcal{X}}$ and $f_y^1, f_y^2 \in \Delta_{\mathcal{Y}}$. Let f_j^1 and f_c^1 be respectively the joint and conditional distributions related to f_x^1 and f_y^1, and analogously for f_j^2 and f_c^2. Then:*

$$\begin{aligned} D_{KL}(f_c^1 \mid\mid f_c^2) &= C(f_c^1, f_c^2) - H(f_c^1) \\ &= \Big[C(f_j^1, f_j^2) - C(f_x^1, f_x^2)\Big] - \Big[H(f_j^1) - H(f_x^1)\Big] \\ &= C(f_j^1, f_j^2) - H(f_j^1) - C(f_x^1, f_x^2) + H(f_x^1) \\ &= D_{KL}(f_j^1 \mid\mid f_j^2) - D_{KL}(f_x^1 \mid\mid f_x^2). \end{aligned} \tag{5.5}$$

In conclusion, it is worth noting that, albeit being commonly used as dissimilarity measures between probability distributions, neither cross-entropy nor KL divergences are truly metrics, since they do not satisfy the triangle inequality. Furthermore, they are not symmetric, as $C(f^1, f^2) \neq C(f^2, f^1)$.

5.3 Methodology

5.3.1 Optimizing the KL Constraint through Meta-MDPs

In Section 3.5, we presented TRPO, which introduces a new hyper-parameter λ to constrain the trust region at each step: the fact that this constraint is usually kept as fixed throughout the whole learning session suggests that there will probably be some inefficiencies that can be solved only through an adaptive selection (as in our case), or through line-search heuristics: as seen, TRPO implementation in Algorithm 4 transforms the KL constraint into a line-search procedure, with double checks to deal with the approximations introduced.

Consequently, with our contribution, we aim to improve TRPO by reducing the conservativity on the step sizes adopted: as mentioned in Section 5.2, we will work on a modified version of the algorithm, denoted as xTRPO, that does not shrink the step size when the surrogate loss does not predict an improvement; in other words, we are deactivating the feature that attempts to mitigate the effects of a bad trust region, since we aim to provide the optimal trust region at each step automatically. Our approach is in line with the previous chapter: We want to apply the meta-MDP devised in Section 4.4 to the problem of HO of the trust radius λ, which will be the main meta-action of the agent.

Update Rule the first and most straightforward modification we need to apply with respect to the previous implementation (aimed at optimizing the step size) consists in the replacement of the update rule: our update rule f is one step of xTRPO: the step takes as input a batch of trajectories collected with the current policy $\pi_{\boldsymbol{\theta}_t}$ and the trust region constraint λ_t. In this way, it is possible to compute an approximation of the natural gradient and then apply a line search starting from an estimation of the maximum stepsize $\alpha(\lambda, \boldsymbol{\omega})$ defined in Equation 5.1, which then is iteratively halved until the main trust region constraint is satisfied.

Encoding the Meta-State As described in Section 5.2, we want to avoid the inclusion in the meta-observation space of an explicit representation of the task, or the complete parametrization, but a representation of the trajectory distribution $\rho_{\boldsymbol{\theta},\boldsymbol{\omega}}$ induced by the interaction of the policy and the context. This distribution provides a complete representation of the behavior of the policy, but it is not *injective*, as different parametrizations might induce the same distribution: for instance, one might think of a wide neural network parametrization that simply emulates the same behavior of a linear policy. The resulting trajectory distribution is the same, but applying a TRPO (or xTRPO) update under the same hyperparameters might lead to different results. Hence, it is relevant to keep in mind that, while the trajectory distribution is indeed useful, it implies a loss of information.
Unfortunately, there is no analytical representation of $\rho_{\boldsymbol{\theta},\boldsymbol{\omega}}$: consequently, starting from a batch of n sampled trajectories, we need to extract some useful features directly relying on $\rho_{\boldsymbol{\theta},\boldsymbol{\theta}}$:

- $H(\cdot)$: the entropy measure introduced with Definition 5.1 describes the expected amount information related to the distribution. In literature, it is often adopted to measure the amount of exploration of a policy (Hazan et al., 2019).
- $C(\cdot,\cdot)$: it might be useful to consider if previous updates provided major changes in the trajectory distributions, dependent on the return manifold w.r.t. the policy space. In some regions of the policy space, small updates can provide major changes in the policy behavior, with the consequent need for a more careful trust region to avoid worse performances. Hence, it is possible to consider the impact of the last update by directly comparing the dissimilarities of the trajectory distributions induced by two policies in consecutive learning steps.
- $D_{\mathrm{KL}}(\cdot \mid\mid \cdot)$: for the same reason as the cross entropy, the KL-divergence between different distributions induced by the policies encountered during the learning process can help to predict the effects on the performance provided by different hyperparameters.

Potentially, we can consider the entire history of meta-features encountered in the learning process up to the current time t, e.g., measure $C(\boldsymbol{\theta}_t, \boldsymbol{\theta}_{t'})$ for all $t' < t$, in a similar way as in (Li and Malik, 2016). However, this idea will scale badly with the learning horizon and would make the set of meta-features entirely dependent on the previous updates. Instead, we limit ourselves to the last update, hence considering only the *timings* t and $t-1$. Hence, we will consider the trajectory distributions ρ_t and ρ_{t-1} to denote the distributions of trajectory induced respectively by the policies $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}_{t-1}$ and their related tasks (fixed or variable). Hence we will select as meta-features the following quantities: the related entropies $H(\rho_t)$ and $H(\rho_{t-1})$, the *backward* dissimilarity measures $C(\rho_t, \rho_{t-1})$, $D_{\mathrm{KL}}(\rho_t, \rho_{t-1})$ and the same *forward* features, where the two distributions are in reverted order: $C(\rho_{t-1}, \rho_t)$, $D_{\mathrm{KL}}(\rho_{t-1}, \rho_t)$. The motivation behind the choice of both *orders* is related to the asymmetry of the dissimilarity measures, as one

	Measure	Distribution	Timing
Trajectory-based	H	$p_{(s)}, p_{(a\|s)}, p_{(s'\|a,s)}$	(t) and $(t-1)$
	C	$p_{(s)}, p_{(a\|s)}, p_{(s'\|a,s)}$	$((t),(t-1))$ and $((t-1),(t))$
	D_{KL}	$p_{(s)}, p_{(a\|s)}, p_{(s'\|a,s)}$	$((t) \mid\mid (t-1))$ and $((t-1) \mid\mid (t))$
Return-based	$\mathbb{E}[J]$		(t)
	$\mathbb{V}\text{ar}[J]$		(t)

Table 5.1: *Summary of all the meta-features ϕ used as observation for the meta-MDP.* Distribution *refers to the type of distribution considered instead of the trajectory distribution.* Timing *refers to the learning step considered for the distribution.*
for instance, for H, $p_{(s)}$ and (t) we mean $H(p_{(s)}(t))$, while for D_{KL}, $p_{(a|s)}$ and $((t) \mid\mid (t-1))$ we mean $D_{KL}(p_{(a|s)}(t) \mid\mid p_{(a|s)}(t-1))$.

of the two measures is usually more meaningful than the other.

Furthermore, the distribution of the overall trajectories of a policy in the state space is unfeasible to be computed; hence, we need a simpler set of distributions that can be more easily estimated: since a trajectory can be represented through a sequence of states and actions, we can represent the distribution on trajectories through the distribution over the states $p_{(s)} \in \Delta_{\mathcal{S}}$, the distribution over the state-action pairs $p_{(s,a)} \in \Delta_{\mathcal{S}\times\mathcal{A}}$ and finally $p_{(s,a,s')} \in \Delta_{\mathcal{S}\times\mathcal{A}\times\mathcal{S}}$, which is the distribution over the transitions (s, a, s'). However, it is more meaningful to include in the set of meta-features the conditional distributions: the probability of choosing an action given a state, measured within the distribution $p_{(a|s)}$ represents the set of choices made from the policy $\pi_{\boldsymbol{\theta}}$ and is independent of the task; in the same way, the distribution $p_{(s'|a,s)}$ represents the context-dependent transition kernel $P_{\boldsymbol{\omega}}$. Hence, these conditional distributions will be adopted instead of the joint ones while keeping the same computational complexity for the estimation thanks to the linear relation provided with Equations 5.2, 5.3 and 5.5.

At last, the trajectory distribution with all related measures does not keep into account the rewards collected: we can include information about the current policy performance since it is crucial for the dynamics of the meta-MDP: in this case, we simply opted for the inclusion of the mean and variance of the estimated returns $J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_t)$, under the current policy parametrization $\boldsymbol{\theta}_t$, which can be easily approximated from the trajectories.

In summary, the overall set of meta-features can be found in Table 5.1: for the *trajectory-based* features, we consider all the possible combinations of measure (H, C, or D_{KL}), distribution ($p_{(s)}$, $p_{(a|s)}$, or $p_{(s'|a,s)}$) and timings; in addition, the *return-based* meta-features are simply the mean and variance of the return. In this way, we have provided a set of meta-features $\phi(\rho_t, \rho_{t-1})$ dependent on the trajectories observed under *timings* t and $t-1$. In the next section, we will provide the techniques used to estimate the trajectory-based meta-features from a batch of samples τ_t and τ_{t-1} (Singh et al., 2003). Thus, we can finally apply Meta-FQI under the xTRPO update rule to optimize the trust region constraint λ_t. In the same fashion, also the collection of the training dataset must be modified accordingly, as depicted in Algorithm 6.

Algorithm 6 Meta-features Dataset Generation

```
Require: CMDP $\mathscr{M}$ (with task distribution $\psi$, policy space $\Theta$, initial policy distribution $\rho$),
  number of learning instances $K$, number of learning steps $T$, number of inner trajectories $n$.
Ensure: dataset of transitions $\mathcal{D}$
  Initialize: $\mathcal{D} = \{\}$,
  for $k = 1, \ldots, K$ do
    Sample context $\boldsymbol{\omega} \sim \psi(\Omega)$, initial policy $\boldsymbol{\theta}_0 \sim \rho(\Theta)$
    Collect $\tau_0$, from $n$ trajectories in task $\mathcal{M}_{\boldsymbol{\omega}}$ under policy $\pi(\boldsymbol{\theta}_0)$
    Estimate $J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_0)$, $\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_0)$
    Compute meta-features $x_0 = \phi(\tau_0, \tau_{-1} := \emptyset)$
    for $t = 0, \ldots, T-1$ do
      Sample meta-action $h \in \mathcal{H}$
      Update policy $\boldsymbol{\theta}_{t+1} = f(\boldsymbol{\theta}_t, h, \tau_{t-1})$
      Collect $\tau_{t+1}$, from $n$ trajectories in $(\mathcal{M}_{\boldsymbol{\omega}}, \pi(\boldsymbol{\theta}_{t+1}))$
      Estimate $J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_{t+1})$, $\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_{t+1})$
      Compute meta-features $x_{t+1} = \phi(\tau_{t+1}, \tau_t)$
      Set
            $$l = J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_{t+1}) - J_{\boldsymbol{\omega}}(\boldsymbol{\theta}_t).$$
      Append $\{(x_t, h, x_{t+1}, l)\}$ to $\mathcal{D}$
    end for
  end for
```

5.4 Meta-Features Estimation

In the previous section we analyzed how to represent informatively and in a very compressed way the complex meta-state of the meta-MDP framework, we introduced a set of meta-features which is both *policy-agnostic* and *task-agnostic*, so it can be adopted without explicit knowledge of the context, without the assumption of a fixed context for the duration of the learning sessions, and with the possibility to generalize over different policy spaces and architectures. In this section, we discuss how the trajectory-based meta-features can be estimated starting from a batch of trajectories. More specifically, we want to estimate the meta-features H, C and D_{KL} for the distributions $p_{(s)}$, $p_{(a|s)}$ and $p_{(s'|a,s)}$, using the trajectories $\tau(t)$ and $\tau(t-1)$ sampled using a policies $\pi_{\boldsymbol{\theta}_t}$ and $\pi_{\boldsymbol{\theta}_{t+1}}$.

An important remark to keep in mind is that we do not have full knowledge of the distributions we want to represent, but we can only have some samples that are iteratively drawn from a batch of trajectories. From the batch of n trajectories $\tau_i = \{s_0^i, a_0^i, r_0^i, s_1^i, \ldots, a_{T-1}^i, s_T^i\}$, we will extract the information required in the form of three new datasets, containing the states encounter, the state-action pairs and the transitions in the form (s, a, s'):

$$\begin{aligned}
\mathcal{D}_s &= \bigcup_{i=1}^{n} \bigcup_{j=0}^{T(i)} (s_j^i) \\
\mathcal{D}_{sa} &= \bigcup_{i=1}^{n} \bigcup_{j=0}^{T(i)-1} (s_j^i, a_j^i) \\
\mathcal{D}_{sas} &= \bigcup_{i=1}^{n} \bigcup_{j=0}^{T(i)-1} (s_j^i, a_j^i, s_{j+1}^i)
\end{aligned} \tag{5.6}$$

Afterward, we can consider these datasets as they were drawn from the respective joint distributions $p_{(s)}$, $p_{(s,a)}$, and $p_{(s,a,s')}$. In the following, we provide a K-Nearest Neighbors (KNN)-based method for the estimation of entropies and cross-entropies provided in (Singh et al., 2003), which implicitly rely on the estimates of densities and volumes from samples. In this way, we can easily take advantage of Equation 5.4 to retrieve the estimation of KL divergences from the approximations obtained. In conclusion, thanks to the conditional properties from 5.2, 5.3 and 5.5, we can obtain the final set of the considered meta-features.

5.4.1 Density and Volumes Estimation

In this subsection, we will provide a useful methodology to provide consistent estimators for the entropy of a distribution, and the cross-entropy among two absolutely continuous distributions in a general context. The estimation of a probability distribution $p \in \Delta_{\mathcal{X}}$ defined on a metric space $(\mathcal{X}, d_{\mathcal{X}})$ can be done by means of a *KNN* approach, as described in Singh et al. (2003): first of all, since we deal with vectors, neighbors and distances, we define the knn-distance of a point using the samples in a dataset $\mathcal{D}$ as:

$$\Delta_k^{\mathcal{D}}(x) = d_{\mathcal{X}}(x; x_k);$$

where x_k is the k-th nearest neighbor of the input $x \in \mathcal{X}$ in the dataset $\mathcal{D}$. In practice, for any x, we consider $\Delta_k^{\mathcal{D}}(x)$ as the distance from x to the k^{th} closest point in the dataset. In this case, k is a parameter, which controls the variance of the estimation. We are dealing with real vectors, hence, the metric space is $\mathbb{R}^d$, for some dimension $d > 0$, and the distance is the usual Euclidean metric. Afterwards, we can estimate the minimum volume $V_k^{\mathcal{D}}(x)$ of an hypersphere centered in x with radius $\Delta_k^{\mathcal{D}}(x)$, which should contain k samples from the dataset $\mathcal{D}$ as:

$$\widehat{V}_k^{\mathcal{D}}(x) = \frac{\pi^{\frac{d}{2}} \Delta_k^{\mathcal{D}}(x)^d}{\Gamma(\frac{d}{2} + 1)}, \tag{5.7}$$

where $\Gamma(\frac{d}{2}+1)$ is the Gamma function. Then, knowing that this is the minimum volume containing k drawn samples, a reasonable estimate of the density is provided.

Definition 5.5 (KNN-Estimator of density function)**.** *Let $\mathcal{D} = \{x_i\}_i$ be a dataset of n samples drawn from $p \in \Delta_{\mathcal{X}}$. The KNN-Estimator $\widehat{p}(x; k, \mathcal{D})$ for every $x \in \mathcal{X}$ can be* *computed as:*

$$\widehat{p}(x;k,\mathcal{D}) = \widehat{p}_k^{\mathcal{D}}(x) = \frac{1}{\frac{n}{k}\widehat{V}_k^{\mathcal{D}}(x)} = \frac{k}{n}\frac{\Gamma(\frac{d}{2}+1)}{\pi^{\frac{d}{2}}\Delta_k^{\mathcal{D}}(x)^d}$$

5.4.2 KNN-Estimator

Given the density or volume estimation, we can provide consistent estimators for the entropy, cross-entropy, and KL-divergence, by transforming the expected values into discrete sums:

Starting from the entropy, we can compute the estimation $\widehat{H}(p)$ for the entropy of the distribution $p \in \Delta_{\mathcal{X}}$, using a dataset $\mathcal{D} = \{x_i \sim p(\cdot)\}$.

Definition 5.6 (Entropy KNN-Estimator)**.** *Let $\mathcal{D} = \{x_i\}_i$ be a dataset of n samples drawn from $p \in \Delta_{\mathcal{X}}$. the KNN-Estimator $\widehat{H}(p)$ for the entropy $H(p)$ can be computed as follows:*

$$\begin{aligned}\widehat{H}(p) &:= -\frac{1}{n}\sum_{x_i \in \mathcal{D}} \log \widehat{p}_k^{\mathcal{D}}(x_i) + \ln k - \Psi(k)\\ &= \frac{1}{n}\sum_{x_i \in \mathcal{D}} \log \frac{n\widehat{V}_k^{\mathcal{D}}(x_i)}{k} + \log k - \Psi(k),\end{aligned} \tag{5.8}$$

where $\ln k + \Psi(k)$ is a bias correction term in which $\Psi(k)$ is the Digamma function.

This estimator is consistent:

Proposition 5.7 (Consistent KNN-entropy estimator, Theorems 8 and 11 in Singh et al. 2003)**.** *Let $\mathcal{D} = \{x_i\}_i$ be a dataset of n samples drawn from $p \in \Delta_{\mathcal{X}}$. Consider the KNN entropy estimator from Equation 5.8. Then:*

$$\lim_{n\to\infty} \mathbb{E}[\widehat{H}(p)] = H(p)$$
$$\lim_{n\to\infty} \mathbb{V}ar[\widehat{H}(p)] = 0$$

For the cross entropies, we need instead two datasets $\mathcal{D}_1$ and $\mathcal{D}_2$ drawn from the respective distributions p_1,$p_2 \in \Delta_{\mathcal{X}}$ for which we want to estimate the cross entropy $C(p_1, p_2)$. What changes w.r.t. the entropy estimation is that we will use the KNN-Estimator for the volume trained on dataset $\mathcal{D}_1$ to compute the information of the samples coming from $\mathcal{D}_2$. Hence we will compute the sample mean over the distribution p_2 of the information $h(p_1)$.

Definition 5.8 (Cross Entropy KNN-Estimator)**.** *Let $\mathcal{D}_1 = \{x_i\}$ be a dataset of n_1 of i.i.d samples drawn from $p_1 \in \Delta_{\mathcal{X}}$, and let $\mathcal{D}_2 = \{x_j\}$ be a dataset of n_2 samples drawn from $p_2 \in \Delta_{\mathcal{X}}$. The KNN-Estimator $\widehat{C}(p_1, p_2)$ of the cross entropy $C(p_1, p_2)$ can be computed as follows:*

$$\widehat{C}(p_1, p_2) := \frac{1}{n_2}\sum_{x_j \in \mathcal{D}_2} \log \frac{n_1\widehat{V}_k^{\mathcal{D}_1}(x_j)}{k} + \log k - \Psi(k).$$

In conclusion, we can also estimate the KL-divergence $D_{\text{KL}}(p_1 \mid\mid p_2)$ easily, remembering from 5.4 that the D_{KL} is the difference between the cross entropy $C(p_1, p_2)$ and the entropy $H(p_1)$.

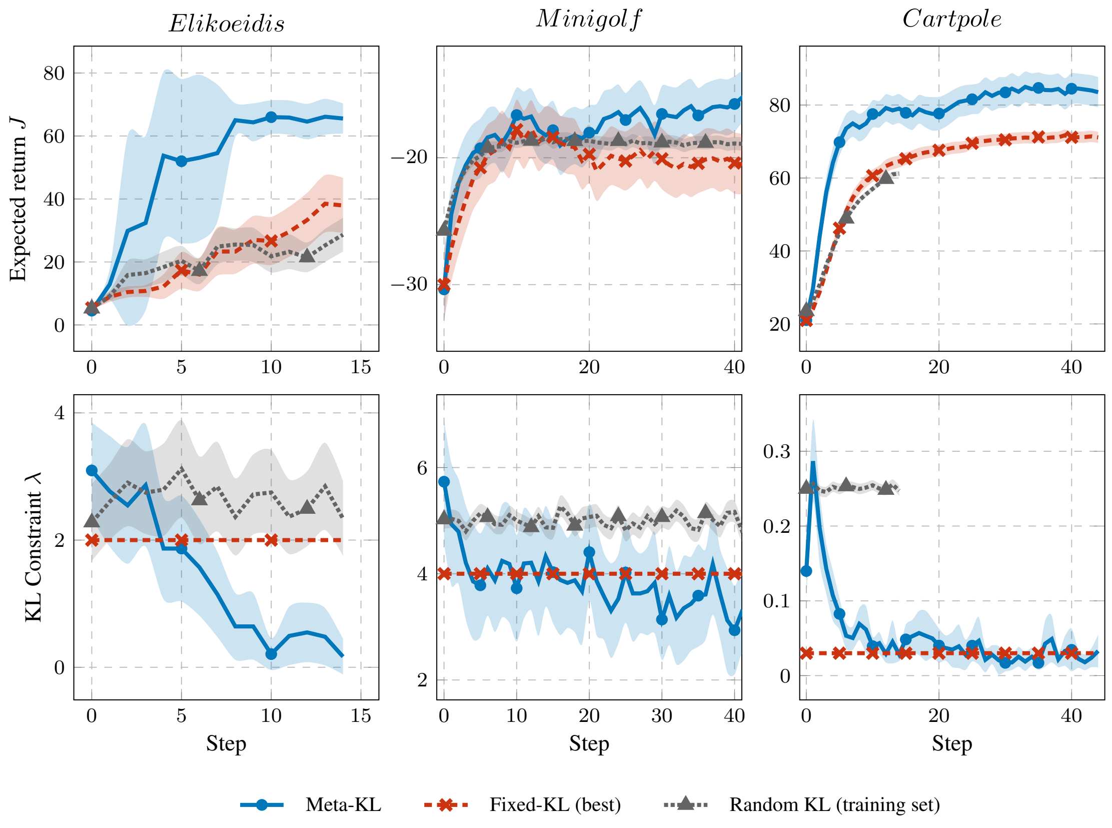


Figure 5.1: *Fine-tuned hyper-parameter TRPO vs meta-model with linear policies. Top Figure: Comparison of the average performances of the tested FQI models (Meta-KL) w.r.t the baselines (Fixed-KL) and the results from the learning trajectories in the meta-training dataset (random KL) Cartpole training dataset is composed of learning trajectories with 15 only total updates, while the evaluation process is performed on 45 learning steps. Bottom Figure: meta-action selected at each learning step. 30 runs, avg $\pm 90\%$ c.i.*

Definition 5.9 (KL-Divergence KNN-Estimator). *Let $\mathcal{D}_1 = \{x_i\}$ be a dataset of n_1 of i.i.d samples drawn from $p_1 \in \Delta_{\mathcal{X}}$, and let $\mathcal{D}_2 = \{x_j\}$ be a dataset of n_2 samples drawn from $p_2 \in \Delta_{\mathcal{X}}$. The KNN-Estimator $\widehat{D}_{KL}(p_1 \mid\mid p_2)$ of the KL-divergence $D_{KL}(p_1, p_2)$ can be computed as follows:*

$$\widehat{D}_{KL}(p_1 \mid\mid p_2) = \widehat{C}(p_1, p_2) - \widehat{H}(p_1),$$

where $\widehat{C}(p_1, p_2)$ and $\widehat{H}(p_1)$ are defined in 5.6 and 5.8.

5.5 Experiments

The experimental environments we used for our tests are *Elikoeidis*, Meta Minigolf (Tirinzoni et al., 2019) and Meta Cartpole (Penner, 2002).

While *Minigolf* and *Cartpole* are the same environments introduced in Chapter 4, with the same contextual generalizations leading to Meta Minigolf and Meta Cartpole environments, *Elikoeidis* is *specifically designed* to challenge the learning capabilities of the original TRPO algorithm, and see whether our approach could perform better with only linear policies. More technically, it is a single-state environment, which

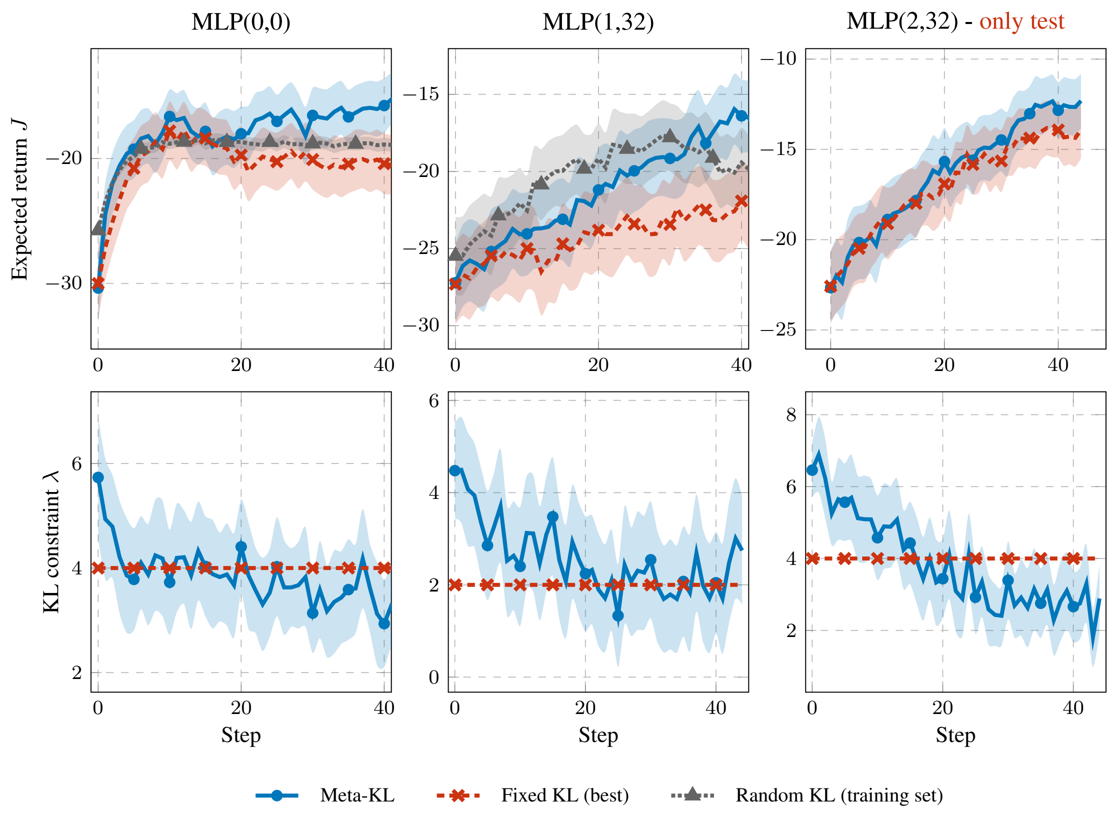


Figure 5.2: *Meta Minigolf: Performances by different Policy Parametrizations. Top figure: Comparison, across different architectures, of the metaFQI model against the baselines, including training trajectories. Bottom figure: meta-action selected at each learning step. 30 runs per architecture. MLP(h, n) denotes an MLP with h hidden layers and n neurons per layer. MLP(0,0) denotes a linear policy. MLP(2,32) (rightmost plot) is only tested, i.e. no policies with this architecture were considered in the training dataset.*

lets us control the complexity of the reward given the action through its task; the reward functions are of the kind $R(a) = \mathcal{N}(c_1 a \sin(a^{1+\frac{1}{c_2}}) + c_3 a, c_4)$, where the vector $\boldsymbol{c} = \{c_1, c_2, c_3, c_4\}$ is the task. In our case, we want to analyze the effects on a single-task MDP, with $\boldsymbol{c} = [0.5, 3, 0.5, 1]$. FQI dataset collection is made of 100 tuples, and for each one, the meta-features are extracted from sets of $n = 8$ linear policy trajectories. Instead, for Minigolf and Cartpole environments, the FQI datasets (containing respectively 7000 and 15000 tuples) are made by considering linear policies and Multi-Layer Perceptrons with different numbers of hidden layers (3 maximum) and hidden neurons. n is set to 64 for the evaluation of meta-features.

The settings are typical of Meta-Learning, with a meta-training phase and a meta-testing phase: in our case, the meta-training is composed by a meta-dataset collection phase, followed by the training phase, since FQI is an off-line algorithm; further, we also had a meta-validation phase to select the best across different meta-models, which were trained on the same datasets but using different a different number of *iterations* and *min-splits*. As commonly done in meta-learning, during the meta-tests we compare the performances of learning sessions collected using our meta-models to those collected using a fixed-value hyper-parameter fine-tuned adopting a grid search, which

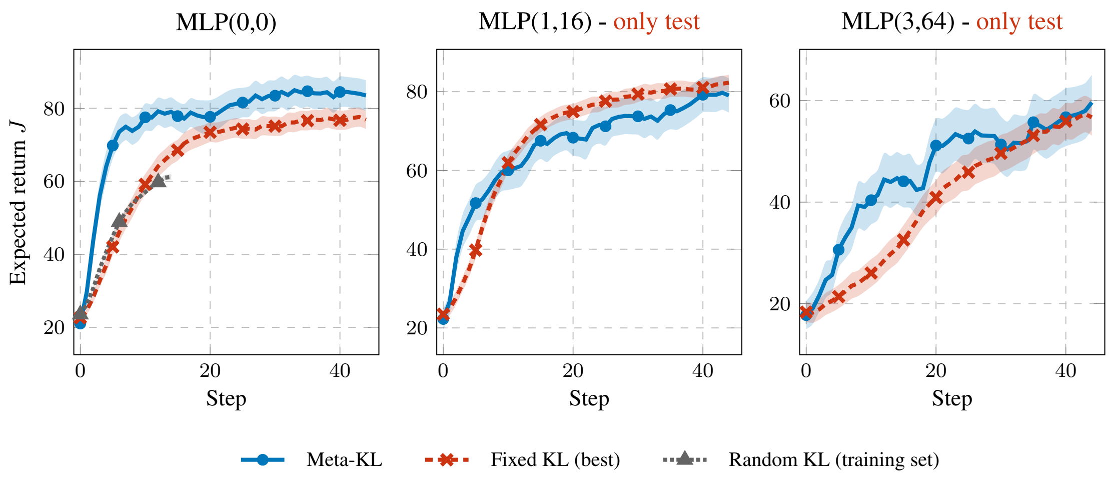


Figure 5.3: *Cartpole: Performances across different Policy Parametrizations. Comparison, split across different architectures, of the metaFQI model against the baselines, including training trajectories. 30 runs per architecture. MLP(h, n) denotes an MLP with h hidden layers and n neurons per layer. MLP(0,0) denotes a linear policy. MLP(1,16) and (3,64) are not in the training dataset, i.e. no policies with these architectures were considered for FQI training.*

are referred to as *baselines*.

In 5.1 we show the main experimental results of our work: for each iteration of the TRPO algorithm, we report the sample mean of the performances and the average selected hyper-parameter using the resulting model, compared to the *baselines*. The *Training Meta-Dataset* curves represent the average performances and λ values obtained in the dataset for training metaFQI models, with a random hyper-parameter at each step. We show it to provide a glimpse of how much information has been learned: for instance, in *Cartpole* the meta-training dataset contained only information about the first 15 steps of learning, while the meta-model has been tested on trajectories 45 steps long, requiring good generalization capabilities. The baselines are composed of a set of trajectories collected by applying a constantly constrained trust region on TRPO across the same set of test meta-tasks. Figure 5.1 depicts the performances of the best-validated FQI model, compared with the best-performing baselines. The comparison shows clearly how the meta-model can always perform as well as the best baseline found, and many times even better, by choosing an adaptive trust region, decreasing with time.

Figures 5.2 and 5.3 show the results across different policy architectures respectively regarding the Minigolf and Cartpole environments. In the first scenario, we can observe that the FQI model is able to reach better results than the baselines in the case of linear policies and with MLP with 1 hidden layer. This last case is emblematic, as also selecting a random trust region seems to bring an improvement with respect to the fixed case. The rightmost plot, instead, depicts the results on a set of instances where the selected parametrizations is an MLP with 2 hidden layers and 32 hidden neurons, which was never seen by the FQI model during training (in a sense, it is a *test architecture* or *test policy class*: the model is able to reach the same results as the best baseline, computing using a grid-search, hence proving its generalization capabilities. The same conclusions can be taken for the Cartpole case (Figure 5.3), where the resulting FQI model can outperform the baselines for the linear case (leftmost plot) and generalize well for unseen architectures (central and rightmost plots), by using only trajectories with 15 updates.

5.6 Conclusions

The general goal of this chapter is to improve TRPO by applying meta-learning to the trust region hyper-parameter optimization, with a dynamic adaptation at each learning step. We proposed a solution capable of building a *parametrization-agnostic* and *task-agnostic* meta-model, which can generalize across different policy parametrizations (or policy classes), and even being adaptive and robust w.r.t. new architectures, i.e. policy classes that were never seen during the training phase, without the need to restart the HO process. Performing better than the optimal (fixed) baseline means that we developed a method to train a meta-model that can substitute and improve the hand-crafted choice of the trust region hyper-parameter, work that must be done for each new task or policy parametrization. All the experiments carried out suggest that the meta-MDP direction is promising: the results obtained in the *Elikoeidis* environment, which was specifically designed with the intent to be easily learned by using actions provided by a meta-model on linear policies, show the feasibility of our approach, while the standard application of TRPO algorithm has far worse performances. Furthermore, the experiments on *Minigolf* and *Cartpole* show how it is possible to fully reach our goals, improving the learning algorithm over a set of tasks, and automatizing the hyper-parameter selection.

There are anyhow many drawbacks and situations in which a meta-learning approach does not bring any advantage: when the optimal fixed-size trust region does not change across different parametrizations or different tasks, it is not computationally efficient to take the effort of building the meta-model to automatize the hyper-parameter choice; on the other hand, the new degrees of freedom on the trust region may result in finding optimal models that cannot be retrieved by means of a fixed-size trust region. Moreover, in complex environments, it may require a lot of meta-training data, as in the classic applications of FQI. Alongside these considerations, our approach involved the computation of a set of informative meta-features, which were reasonable, but with no theoretical grounding: as future research directions, we may wonder how to generate and select more informative meta-features with a solid validation theory, and to develop online approaches, that can bring the advantages of a dynamic choice of learning hyperparameters.

Part II

Dynamic Step and Control Frequency

In Part I we were focused on enhancing the learning capabilities of RL models by optimizing the hyperparameter selection procedure. The contributions we made may be leveraged to automatize the fine-tuning process, that practitioners always have to perform when facing new tasks. However, they also need to carefully configure the environment, in such a way that the information gathered from the sample trajectories is useful enough to detect the most promising actions. In particular, peculiar attention is given to the *control frequency* of the environment, which defines the duration of the actions and the amount of time between two consecutive observations. Acting with the highest possible frequency allows the best control opportunities, but deeply increases the complexity of the problem.

In this part, we deal with the problem of *Control Frequency Adaptation*, with a particular focus on value-based algorithms. In Chapter 6 we introduce the main tool to work with different frequencies: *action persistence*. It consists of a parameter defining the number of repetitions of an action and helps us analyze in detail the effects of different frequencies on the trade-off between the learning opportunities and the performances of the optimal policies. In this direction, we implement a value-based algorithm, able to learn with different persistence values, and heuristic methods to learn effective frequencies. Further improvements are provided in Chapter 7, where we deal with a dynamic selection of the persistence: the action space is enlarged to the space of *persistence options*, in such a way that the agent selects both the action and its duration.

CHAPTER 6

Control Frequency Adaptation via Action Persistence

6.1 Introduction

In the previous chapters, we have described and implemented some of the most popular RL algorithms, which are all aimed at solving *discrete-time* MDPs (Puterman, 2014), introduced with Definition 2.1. In this framework, the interaction between the agent and the environment is issued in equally distanced time instants, specified by a *control frequency*. However, numerous real-world problems are more naturally defined in the continuous–time domain (Luenberger, 1979), such as robotic manipulation (Kober et al., 2013; Haarnoja et al., 2019; Kilinc et al., 2019), autonomous driving (Kiran et al., 2021) or continuous system control (Lillicrap et al., 2015; Yildiz et al., 2021). It is possible to provide ad-hoc definitions of MDPs in the continuous setting (Cont-MDPs, Section 2.2.2), and several works are focused on such setting (Munos and Bourgine, 1997; Bradtke and Duff, 1994; Doya, 2000), but the majority of research is devoted to the discrete case. Consequently, practitioners often need to deal with the problem of the selection of the *time discretization*: the choice of the control frequency of a system has a relevant impact on the ability of *reinforcement learning algorithms* to learn a highly performing policy.

In this chapter, we introduce the notion of *action persistence* that consists of the repetition of an action for a fixed number of decision steps, having the effect of modifying the control frequency. The *base MDP* we work under the assumption that there exists a *base control time step* Δt_0 inducing the dynamics of the *base MDP* $\mathcal{M}_{\Delta t_0}$. Equivalently, we can define the base control frequency as $f_0 = \frac{1}{\Delta t_0}$. In this setting, we want to select a suitable *control time step* Δt that is multiple of the base time step, i.e.,

$\Delta t = k\Delta t_0$ with $k \in \mathbb{N}^+$. Typically, a lower bound on Δt_0 is imposed by the physical limitations of the system (e.g., the frequency of the sensors and of the actuators in a physical system, or the clock time in processors). Thus, we restrict the search of Δt to the discrete set $\{k\Delta t_0\,,\, k \in \mathbb{N}^+\}$. Any choice of k generates an MDP $\mathcal{M}_{k\Delta t_0}$ according to Definition 6.2. Therefore, we describe k as the *persistence* of the action, or in other words, the number of base time intervals during which an action is kept fixed. In a sense, we can see persistence as an environmental parameter (time discretization) that can be tuned to enhance the learning capabilities of an RL algorithm. We analyze how action persistence affects the performance of the optimal policy, and then we present a novel algorithm, *Persistent Fitted Q-Iteration* (PFQI), that extends FQI (Ernst et al., 2005), with the goal of learning the optimal value function at a given persistence. Furthermore, we introduce a heuristic approach devoted to identifying the optimal persistence, which is then tested on benchmark domains to show the advantages of action persistence and prove the effectiveness of our persistence selection method.

Chapter Outline This chapter is organized as follows: the main motivation of our approach is described in Section 6.2, along with a couple of examples to underline the empirical importance related to the time discretization problem. Some of the analyses and solutions provided in the literature are provided in Section 6.3. The main contribution starts with Section 6.4, where we elaborate on the concept of action persistence: we show that it can be understood as a tunable environment parameter or as a modification of the solution class to a set of *non-Markovian* policies. A low persistence value, i.e., the repetition of the same action for a small amount of (base) time steps, reflects the selection of a high control frequency: from one side, there are more control opportunities leading to better optimal policies, but at the same time the sample complexity increases leading to more difficult learning. We provide a bound for the performance loss generated by action persistence is analyzed in Section 6.5, under Lipschitz assumptions on the environment. The result confirms the intuition that the performance loss is strictly related to how fast the environment evolves as an effect of the actions (Section 6.5). Then, we apply the notion of action persistence in the batch RL scenario, and we extend the FQI algorithm (Ernst et al., 2005), introduced in Section 3.5, to keep into account action persistence. We denote the resulting algorithm as Persistent Fitted Q-Iteration (PFQI), which estimates the optimal value function of a policy with a target persistence (Section 6.6). The value function estimation for a set of candidate persistences $\mathcal{K} \subset \mathbb{N}^+$ can then be compared to select the best performing greedy policy: thus, we introduce in Section 6.7 a persistence selection heuristic to approximate the optimal persistence. The approach is then experimentally evaluated on benchmark domains, whose goal is to confirm our theoretical findings and evaluate our persistence selection method (Section 6.8). An interesting application of our approach is provided in Section 6.9, where we trained an artificial agent to trade into the Foreign Exchange (FX) market. The experimental campaign is not enough to answer all the possible questions related to action persistence, some of which are presented and discussed in Section 6.10. The proofs of all the results and further analyses are available in Appendix B.1.

6.2 Motivations

The choice of the control frequency of a system has a relevant impact on the ability of *reinforcement learning algorithms* to learn a highly performing policy. Intuitively, a higher *control frequency* of the system allows the agent to make more frequent decisions, possibly improving the performance. This may lead to an incorrect suggestion of controlling the system at the highest possible frequency, within its physical limits. RL techniques, however, are adopted when the environment dynamics are unknown: a too-fine discretization could result in the opposite effect, making the problem harder to solve. Indeed, any RL algorithm needs to (implicitly or explicitly) evaluate the effects of the actions chosen by the agent: if the time interval between two observations is too narrow, the effect of the action may be limited, and the related advantage is consequently infinitesimal. Indeed, for standard *value-based* RL, (Tallec et al., 2019) proved than, under suitable smoothness assumption, in the limit for $\Delta t_0 \to 0$, the Q-value function collapsed to the value function V. This was empirically observed also in (Baird, 1994). This issue does not only affect value-based algorithms: Park et al. (2021) proved that, for stochastic MDPs, the variance of the return function is proportional to f_0. Consequently, a higher control frequency leads to an increase of *sample complexity*. Instead, low frequencies allow the environment to evolve for a longer time, making the effect of individual actions more easily detectable. Furthermore, in a system characterized by a “slowly evolving” dynamics, the gain obtained by increasing the control frequency might become negligible. Finally, lower frequencies help overcome some partial observability issues in robotics, like action execution delays (Kober and Peters, 2014).

Therefore, we experience a fundamental *trade–off* in the control frequency choice that involves control opportunities (higher in high–frequency) and the sample complexity (smaller in low–frequency). This problem is not only intriguing from a theoretical point of view but is also very important from a practical perspective: we now provide some scenarios as examples in which tuning the frequency is of fundamental importance.

Example 6.1 (Robotic Manipulation)**.** *One of the most straightforward frameworks where control frequency adaptation is relevant is the robotic manipulation field (Shahid et al., 2022): the dynamic systems evolve in continuous time, but the sensors and actuators are governed by a natural control frequency. In the design of the control problem, actions changed with high frequency can place a lot of strain on the robot mechanics and lead to oscillations or too dangerous positions (Kober et al., 2013). Conversely, simulated control environments often rely on Euler-like approximations of the dynamical systems, which are feasible only with small time steps. Thus, simply reducing the base control frequency might lead to inefficiencies, Persistence comes into help, as it allows to keep the system under surveillance, with frequent observations and less dangerous actions.*

Example 6.2 (Locomotion)**.** *Consider a simple locomotion task, where the agent needs to navigate to reach a goal, as a Grid maze or as a more complex race track. If an agent changes the direction with high frequency, the resulting trajectories keep returning to a region closer to the starting area, thus leading to relevant exploration issues (Park et al., 2021). By repeating the actions, it is possible to generate self-avoiding trajec-*

tories with higher probability, dramatically improving the exploration. This particular phenomenon is studied in more detail in Chapter 7.

Example 6.3 (Atari games: frame skip)**.** *In the previous examples, we often took into account near-continuous systems, which are subject to time discretization. Action persistence might be useful also in virtual environments virtually embedded with a specific base frequency, such as videogames. In some of the most famous value-based RL implementations (Bellemare et al., 2013; Mnih et al., 2015), the authors designed the environment introducing a frame-skip parameter, which controls the number of frames between two consecutive actions. This parameter is equivalent to action persistence and was introduced for empirical reasons; its importance was then analyzed more carefully in (Braylan et al., 2015; Kalyanakrishnan et al., 2021).*

Example 6.4 (Financial Trading)**.** *One last, very important framework where persistence can bring several advantages is financial trading: the application of RL in this field has been revealed to be successful since the pioneering work of (Moody and Saffell, 2001). Autonomous agents are able to detect patterns in the market and can take advantage of profitable signals that may span from a few minutes to a few hours. However, even if trading opportunities are present at many different time scales, learning them is not equally difficult. Most of the autonomous agents that populate the market today operate at a very high frequency, but their behavior is usually hard-coded since, at such time scales (milliseconds), the main opportunities are constituted by ephemeral arbitrages. On the other hand, operating at very low frequency (e.g. days, months) is impossible without including exogenous inputs regarding the market, such as economic data releases or market news. Precisely determining the best time scale for the learning process is fundamental to learning profitable policies. In fact, while higher frequencies always allow, in principle, for better control, they may present a worse signal-to-noise ratio, which can deeply impact the learning performance. Moreover, financial traders always have to deal with transaction costs: frequent portfolio modifications induce higher global costs and might lead to inefficient strategies. In the field of RL for trading, the study of the impact of the trading time scale hasn't been a primary focus. Most works do not consider changing the frequency of interaction with the environment even though the variety of time scales across papers ranges from high-frequency to daily to even longer periods. Some works highlight the effect of assumptions on the trading frequency, such as transaction costs or the agent's risk aversion (Bisi et al., 2020a). In the experimental campaign (extracted from Riva et al. 2021), we analyze the impact of persistence in the Foreign Exchange Trading scenario.*

6.3 Related Work

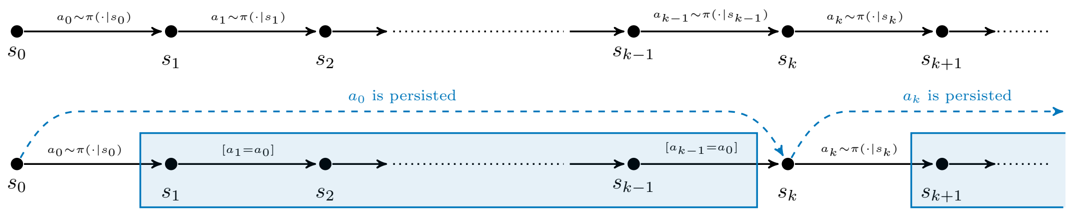
In this section, we revise the works connected to persistence, focusing on continuous–time RL and temporal abstractions.

Continuous–Time RL Among the first attempts to extend value-based RL to the continuous time domain, there is *advantage updating* (Bradtke and Duff, 1994), in which Q-learning is modified to account for infinitesimal control time steps. Instead of storing the Q-function, the authors rely on a rescaling of the advantage function $\widetilde{A}^{\pi} =$

$\frac{(Q(s,a)-V(s)}{\Delta t})$, where Δt is the time discretization. In Baird (1994), a continuous-time MDP (Cont-MDP) is addressed by means of the semi-Markov decision processes (Howard, 1963) (SemiMDP, described in Section 2.2.2) for finite–state problems. The optimal control literature has extensively studied the solution of the Hamilton-Jacobi-Bellman equation (Kirk, 2004), i.e., the continuous-time counterpart of the Bellman equation, when assuming the knowledge of the environment (Bertsekas, 2005; Fleming and Soner, 2006). The model-free case has been addressed by resorting to time discretizations (Peterson, 1993), which also provides convergence guarantees (Munos, 1997; Munos and Bourgine, 1997), and by utilizing functions function approximation (Dayan and Singh, 1995; Doya, 2000). The learning issues when dealing with the near-continuous setting have been analyzed for both policy-based algorithms (Munos, 2006; Park et al., 2021), and value-based RL (Baird, 1994; Tallec et al., 2019).

Temporal Abstractions The idea of persisting an action can be considered as a form of *temporal abstraction* (Sutton et al., 1999b; Precup, 2001). Temporally extended actions have been employed in the hierarchical RL to model different time resolutions (Singh, 1992a,b), subgoals (Dietterich, 1998), and combined with the actor–critic (Bacon et al., 2017). Persisting an action is a particular instance of a (time-based) semi-Markov *option*, where an action is temporally extended for k steps. According to the flat option representation (Precup, 2001), the option framework can be considered by setting the option initiation set as the whole state space ($\mathcal{I} = \mathcal{S}$); the intra-option policy plays deterministically the action selected when the option is issued (the non-Markovian k–persistent policy introduced in Section 6.4); finally, the termination condition is times based, as the option ends when k time steps have passed after the option started, i.e., $\beta(H_t) = \mathbb{1}_{\{t \bmod k=0\}}$. Interestingly, in Mann et al. (2015), approximate value iteration for options lasting at least a given number of steps is proposed and analyzed, sharing some similarities with persistence, and the authors show that action repetition leads to faster convergence (and lower approximation errors). The connections between control frequency adaptation and action persistence are interesting, but we consider the temporal abstraction introduced within the option framework to be more general and related to the semantics of the task to solve rather than the exploitation of the information collected by the agent via action persistence. A more detailed discussion on the relationship persistence-option framework is provided in Chapter 7

Action repetition has acquired practical relevance since the introduction of Deep RL (Mnih et al., 2013), by leveraging the *frame skip* parameter (Bellemare et al., 2013). In this direction, several works (Braylan et al., 2015; Khan et al., 2019) had shown the importance of different frame skips for Atari games on exploration and convergence speed. In Grigsby et al. 2021 the authors proposed an algorithm to automatically tune the control frequency, along with other learning hyperparameters. At last, an interesting research direction in literature regards the dynamic selection of the control frequency by employing *repetition rates*: these related works are discussed in the next chapter since the main goal in Chapter 7 is to learn an adaptive persistence rate.



$a_0 \sim \pi(\cdot|s_0)$
$a_1 \sim \pi(\cdot|s_1)$
$a_{k-1} \sim \pi(\cdot|s_{k-1})$
$a_k \sim \pi(\cdot|s_k)$
s_0
s_1
s_2
s_{k-1}
s_k
s_{k+1}
a_0 is persisted
a_k is persisted
$a_0 \sim \pi(\cdot|s_0)$
$[a_1 = a_0]$
$[a_{k-1} = a_0]$
$a_k \sim \pi(\cdot|s_k)$
s_0
s_1
s_2
s_{k-1}
s_k
s_{k+1}

Figure 6.1: *Agent-environment interaction without (top) and with (bottom) action persistence, highlighting duality. The transition generated by the k-persistent MDP $\mathcal{M}_k$ is the cyan dashed arrow, while the actions played by the k-persistent policies are inside the cyan rectangle.*

6.4 Persisting Actions in MDPs

In this section, we introduce the concept of *action persistence*. The execution of an action with a persistence value equal to $k \in \mathbb{N}^+$ is equivalent to the repetition of such action for a total number of (base) time steps equal to k. In other words, at decision step $t = 0$, the agent selects an action according to its policy $a_0 \sim \pi(\cdot|s_0)$. Action a_0 is kept fixed, or *persisted*, for the subsequent $k-1$ decision steps, i.e., actions $a_1, ..., a_{k-1}$ are all identical to a_0. Then, at decision step $t = k$, the agent queries again the policy $a_k \sim \pi(\cdot|s_k)$ and persists action a_k for the subsequent $k - 1$ decision steps and so on. Hence, the agent employs its policy only at decision steps t that are integer multiples of the persistence k ($t \bmod k = 0$). Trivially, the usual execution of π corresponds to persistence 1.

6.4.1 Duality of Action Persistence

Unsurprisingly, the execution of a Markovian stationary policy π at persistence $k > 1$ produces a behavior that, in general, cannot be represented by executing any Markovian stationary policy at persistence 1. Indeed, at any decision step t, such policy is dependent on the action pursued at the last decision step multiple of k (thus it is non-Markovian with memory k) and has to understand whether to select a new action based on t (so it is non-stationary).

Definition 6.1 (k-persistent policy). *Let $\pi \in \Pi$ be a Markovian stationary policy. For any $k \in \mathbb{N}^+$, the k-*persistent policy *induced by π is a non–Markovian, non–stationary policy, such that, for each $a \in \mathcal{A}$ and $t \in \mathbb{N}$ as:*

$$\pi_{k,t}(a|h_t) = \begin{cases} \pi(a|s_t) & \textit{if } t \bmod k = 0 \\ \delta_{a_{t-(t \bmod k)}}(a) & \textit{otherwise} \end{cases} \tag{6.1}$$

where $h_t = (s_0, a_0, \ldots, s_t)$ and δ is the Dirac distribution. Moreover, we denote with $\Pi_k = \{(\pi_{k,t})_{t \in \mathbb{N}} : \pi \in \Pi\} \subset \Pi^H$ The set of k-persistent policies.

Clearly, for $k = 1$ we recover policy π as we always satisfy the condition $t \bmod k = 0$ i.e., $\pi = \pi_{1,t}$ for all $t \in \mathbb{N}$. We refer to this interpretation of action persistence as *policy view*, as it looks at the problem from the angle of the control policy.

An alternative perspective consists in looking at the effect of the original policy π in a suitably modified MDP. To this purpose, we introduce the (state-action) persistent transition probability kernel $\pi_\delta : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}} \times \mathcal{A}$ defined for all $(s, a) \in \mathcal{S} \times \mathcal{A}$ as:

$$p_\delta(s', a'|s, a) = \int_{\mathcal{S}} P(s'|s, a)\delta_a(a'). \tag{6.2}$$

The crucial difference between p_π and p_δ is that the former samples the action a' to be executed in the next state s' according to π, whereas the latter replicates in state s' action a. We are now ready to define the k-persistent MDP.

Definition 6.2 (k-persistent MDP)**.** *Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$ be an MDP. For any $k \in \mathbb{N}^+$, the k-*persistent MDP *is defined as MDP $\mathcal{M}_k = \left(\mathcal{S}, \mathcal{A}, P_k, R_k, \gamma^k\right)$, where P_k and R_k are respectively the k-persistent transition model and reward distribution such that, for each state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, the following hold:*

$$P_k(\cdot|s, a) = \left((p_\delta)^{k-1}P\right)(\cdot|s, a), \tag{6.3}$$

$$R_k(\cdot|s, a) = \sum_{i=0}^{k-1} \gamma^i \left((p_\delta)^i R\right)(\cdot|s, a); \tag{6.4}$$

analogously, $r_k(s, a) = \int_{\mathbb{R}} x R_k(\,\mathrm{d}x|s, a) = \sum_{i=0}^{k-1} \gamma^i \left((p_\delta)^i r\right)(s, a)$ is the expected reward, uniformly bounded by $R_{\max}\frac{1-\gamma^k}{1-\gamma}$.

The k-persistent transition model P_k keeps action a fixed for $k - 1$ base steps while making the state evolve according to P. Similarly, the k-persistent reward R_k provides the cumulative discounted reward over k steps in which a is persisted. For a Markovian stationary policy $\pi \in \Pi$, we define the transition kernel p_k^π, analogously to p_π, as in Equation (2.6)[1]. In the same fashion, we will denote the expected return of a policy π in the k–persistent MDP $\mathcal{M}_k$ as $J_k(\pi)$. Clearly, for $k = 1$ we recover the base MDP, i.e., $\mathcal{M} = \mathcal{M}_1 = \mathcal{M}_{\Delta t_0}$, and we will remove the subscript Δt_0 for brevity: If $\mathcal{M}$ is the base MDP $\mathcal{M}_{\Delta t_0}$, the k–persistent MDP $\mathcal{M}_k$ corresponds to $\mathcal{M}_{k\Delta t_0}$. Therefore, executing policy π in $\mathcal{M}_k$ at persistence 1 is equivalent to executing policy π at persistence k in the original MDP $\mathcal{M}$. We refer to this interpretation of persistence as *environment view* (Figure 6.1). Thus, solving the base MDP $\mathcal{M}$ in the space of k-persistent policies Π_k (Definition 6.1), thanks to this *duality*, is equivalent to solving the k-persistent MDP $\mathcal{M}_k$ (Definition 6.2) in the space of Markovian stationary policies Π.

Remark 6.3. *It is worth noting that the persistence $k \in \mathbb{N}^+$ can be seen as an* environmental parameter*(affecting P, R, and γ, which can be externally configured with the goal to improve the learning process for the agent. In this sense, the MDP $\mathcal{M}_k$ can be seen as a Conf-MDP (Section 2.2.2) with parameter $k \in \mathbb{N}^+$ (Metelli et al., 2018): the joint tuning of the learning algorithm adopted and the persistence value might allow enhancing the overall performance of the resulting agents. Furthermore, a persistence of k induces a k-persistent MDP $\mathcal{M}_k$ with smaller discount factor γ^k. Therefore, the effective horizon in $\mathcal{M}_k$ is $\frac{1}{1-\gamma^k} < \frac{1}{1-\gamma}$. Interestingly, this effect of persistence is similar to the reduction of the planning horizon by explicitly reducing the discount factor of the task (Petrik and Scherrer, 2008; Jiang et al., 2016).*

[1]**Notation:** as the reader may notice, we have changed the notation, as the policy π is in the superscripts and the persistence value is placed in the subscript. This is made in order to avoid confusion with the usual notation for the multi-step kernel introduced in Equation 2.7 .

6.4.2 Persistent Bellman Operators

When executing policy π at persistence k in the base MDP $\mathcal{M}$, we can evaluate its performance starting from any state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, inducing a Q-function that we denote with Q_k^π and call *k-persistent action-value function* of π. In a straightforward way, we can define the k–persistent value function V_k^π as the expected action value function following π, i.e., $V_k^\pi(s) = \int_{\mathcal{A}} \pi(\,\mathrm{d}a|s) Q_k^\pi(s, a)\ \ \forall s \in \mathcal{S}$.

Thanks to duality, Q_k^π is also the action-value function of policy π when executed in the k-persistent MDP $\mathcal{M}_k$. Therefore, we can naturally extend the Bellman Expectation Operators (Definition 2.7 to their k–persistent versions, where the transition kernel P^k is taken into account:

Definition 6.4 (k–Persistent Bellman Expectation Operators)**.** *Let $\mathcal{M}$ be an MDP and $\pi \in \Pi$ be a policy. Consider a bounded, measurable function $f : \mathcal{S} \to \mathbb{R}$ and a state $s \in \mathcal{S}$. Given a persistence value $k \in \mathbb{N}^+$, the k–Persistent Bellman expectation operator for state value functions T_k^π is defined as:*

$$(T_k^\pi f)(s) := \int_{\mathcal{A}} \pi(\mathrm{d}a|s) \left(r_k(s, a) + \gamma^k \int_{\mathcal{S}} P_k(\mathrm{d}s'|s, a) f(s') \right), \tag{6.5}$$

Moreover, considering a bounded, measurable function $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and a state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, the k–Persistent Bellman expectation operator for state-action value functions T_k^π is defined as:

$$(T_k^\pi f)(s, a) := r_k(s, a) + \gamma^k \int_{\mathcal{S}} P_k(\mathrm{d}s'|s, a) \int_{\mathcal{A}} \pi(\mathrm{d}a'|s') f(s', a'), \tag{6.6}$$

Trivially, operators T_k^π are γ^k contractions in L_∞-norm., and Q_k^π and V_k^π are their fixed points. Similarly, again thanks to duality, the optimal Q-function in the space of k-persistent policies Π_k, denoted by $Q_k^\star$ and called *k-persistent optimal action-value function*, corresponds to the optimal Q-function of the k-persistent MDP, and can be formally framed generalizing Definition 2.9:

Definition 6.5 (k-Persistent Optimal Value Functions)**.** *Let $\mathcal{M}$ be an MDP. Given a persistence value $k \in \mathbb{N}^+$, a policy $\pi^\star \in \Pi$ is* k-persistent optimal *if:*

$$\begin{aligned} V_k^{\pi^\star}(s) &\geq V_k^\pi(s) \quad \forall \pi \in \Pi, \forall s \in \mathcal{S} \\ Q_k^{\pi^\star}(s, a) &\geq Q_k^\pi(s, a) \quad \forall \pi \in \Pi, \forall (s, a) \in \mathcal{S} \times \mathcal{A} \end{aligned}$$

The k-Persistent optimal state value function *and* k-Persistent optimal action-value function *are then defined according to the values attained by the optimal policy:*

$$\begin{aligned} V_k^\star(s) &:= V_k^{\pi^\star}(s) \quad \forall \pi \in \Pi, \forall s \in \mathcal{S} \\ Q_k^\star(s, a) &:= Q_k^{\pi^\star}(s, a) \quad \forall \pi \in \Pi, \forall (s, a) \in \mathcal{S} \times \mathcal{A} \end{aligned}$$

Analogously, $Q_k^\star$ is the fixed point of the Bellman Optimal Operator of $\mathcal{M}_k$, defined as k–persistent Bellman Optimal Operator:

Definition 6.6 (k–Persistent Bellman Optimality Operators)**.** *Let $\mathcal{M}$ be an MDP. Consider a bounded, measurable function $f : \mathcal{S} \to \mathbb{R}$ and a state $s \in \mathcal{S}$. Given a persistence value $k \in \mathbb{N}^+$, the k–Persistent Bellman optimality operator for state value* *functions $T_k^\star$ is defined as:*

$$(T_k^\star f)(s) \coloneqq \sup_{a\in\mathcal{A}} \left\{ r_k(s,a) + \gamma^k \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) f(s') \right\} \tag{6.7}$$

Moreover, considering a bounded, measurable function $f : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$ and a state-action pair $(s,a) \in \mathcal{S}\times\mathcal{A}$, the k–Persistent Bellman optimality operator for state-action value functions $T_k^\star$ is defined as:

$$(T_k^\star f)(s,a) \coloneqq r_k(s,a) + \gamma^k \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) \max_{a'\in\mathcal{A}} f(s',a'), \tag{6.8}$$

Again, T_k^* is a γ^k-contraction in L_∞-norm, with $Q_k^{\pi^\star}$ as fixed point.

In order to provide a complete relationship between the classic Bellman operators and their k–persistent versions, we need to define also a (one-step) *Bellman persistence operator*:

Definition 6.7 (Bellman Persistence Operators)**.** *Let $\mathcal{M}$ be an MDP and consider a state-action pair $(s,a) \in \mathcal{S}\times\mathcal{A}$. The* Bellman persistence operator *for state-action value functions T^δ is defined as:*

$$(T^\delta f)(s,a) \coloneqq r(s,a) + \gamma \int_{\mathcal{S}}\int_{\mathcal{A}} p_\delta(\,\mathrm{d}s',\,\mathrm{d}a'|s,a) f(s,a). \tag{6.9}$$

We now prove that the k-persistent Bellman operators are obtained as a composition of the base operators T^π and T^*.

Theorem 6.8. *Let $\mathcal{M}$ be an MDP, and let $\mathcal{M}_k$ be the k-persistent MDP for a persistence value $k \in \mathbb{N}^+$. Let $\pi \in \Pi$ be a Markovian stationary policy. Then, T_k^π and T_k^* can be expressed as:*

$$T_k^\pi = \left(T^\delta\right)^{k-1} T^\pi \quad \textit{and} \quad T_k^* = \left(T^\delta\right)^{k-1} T^*, \tag{6.10}$$

Proof. The result can be derived by explicitly writing the definitions of P_k and R_k in terms of P, R and γ (Equations 6.3 and 6.4) from the definition of k-persistent Bellman expectation operator T_k^π (Definition 6.4). Consider a bounded, measurable function $f : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$ and $(s,a) \in \mathcal{S}\times\mathcal{A}$:

$$\begin{aligned}
(T_k^\pi f)(s,a) &= r_k(s,a) + \gamma^k (p_k^\pi f)(s,a) \\
&= \sum_{i=0}^{k-1} \gamma^i \left((p_\delta)^i r\right)(s,a) + \gamma^k ((p_\delta)^{k-1} p_\pi f)(s,a) && (6.11) \\
&= \left(\sum_{i=0}^{k-1} \gamma^i (p_\delta)^i r + \gamma^k (p_\delta)^{k-1} p_\pi f \right)(s,a) \\
&= \left(\sum_{i=0}^{k-2} \gamma^i (p_\delta)^i r + \gamma^{k-1} (p_\delta)^{k-1} \left(r + \gamma p_\pi f\right) \right)(s,a) && (6.12) \\
&= \left(\sum_{i=0}^{k-2} \gamma^i (p_\delta)^i r + \gamma^{k-1} (p_\delta)^{k-1} T^\pi f \right)(s,a), && (6.13)
\end{aligned}$$

where (6.11) follows from Definition 6.2, line (6.12) is obtained by isolating the last term in the summation $\gamma^{k-1}(p_\delta)^{k-1}r$ and collecting $\gamma^{k-1}(p_\delta)^{k-1}$, and line (6.13) derives from the definition of the Bellman expectation operator T^π. It remains to prove that for a measurable $g : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and $(s,a) \in \mathcal{S} \times \mathcal{A}$, we have the following identity:

$$(T^\delta)^{k-1} g = \sum_{i=0}^{k-2} \gamma^i (p_\delta)^i r + \gamma^{k-1}(p_\delta)^{k-1} g. \tag{6.14}$$

We prove it by induction on $k \in \mathbb{N}^+$. For $k = 1$ we have only $g = (T^\delta)^0 g$. More formally, it holds also for $k = 2$, as $(T^\delta)g = r + \gamma(p_\delta)g$ by definition.

Let us assume that the identity hold for $k-1$, we prove the statement for k:

$$\begin{aligned}
\left((T^\delta)^k g\right)(s,a) &= (T\delta)^{k-1} T^\delta g(s,a) \\
&= \left(\sum_{i=0}^{k-2} (p_\delta)^i r + \gamma^{k-1}(p_\delta)^{k-1} p_\delta g\right)(s,a) \\
&= \left(\sum_{i=0}^{k-2} (p_\delta)^i r + \gamma^{k-1}(p_\delta)^{k-1} r + \gamma^k (p_\delta)^k g\right)(s,a) \qquad (6.15) \\
&= \left(\sum_{i=0}^{k-1} \gamma^i (p_\delta)^i r + \gamma^k (p_\delta)^k g\right)(s,a)
\end{aligned}$$

where line (6.15) derives from the definition of p_δ. We get the result by taking $g = T^\pi f$.

Concerning the k-persistent Bellman optimal operator the derivation is analogous. For simplicity, we define the $\max$-operator $M : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ defined for a bounded measurable function $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and a state $s \in \mathcal{S}$ as $(Mf)(s) = \max_{a \in \mathcal{A}} f(s,a)$. As a consequence, the Bellman optimal operator becomes: $T^* f = r + \gamma P M f$. Therefore, we have:

$$\begin{aligned}
(T^*_k f)(s,a) &= r_k(s,a) + \gamma^k \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) \max_{a' \in \mathcal{A}} f(s',a') \\
&= r_k(s,a) + \gamma^k \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) M f(s') \qquad (6.16) \\
&= \left(r_k + \gamma^k P_k M f\right)(s,a) \qquad (6.17) \\
&= \left(\sum_{i=0}^{k-1} \gamma^i (p_\delta)^i r + \gamma^k (p_\delta)^{k-1} P M f\right)(s,a) \\
&= \left(\sum_{i=0}^{k-2} \gamma^i (p_\delta)^i r + \gamma^{k-1}(p_\delta)^{k-1}\left(r + \gamma P M f\right)\right)(s,a) \qquad (6.18) \\
&= \left(\sum_{i=0}^{k-2} \gamma^i (p_\delta)^i r + \gamma^{k-1}(p_\delta)^{k-1} T^* f\right)(s,a), \qquad (6.19)
\end{aligned}$$

where line (6.16) derives from the definition of the $\max$-operator M and line (6.16) from the definition of the operator P_k. By applying Equation (6.14) we get the result. $\square$

Therefore, the fixed point equations for the k-persistent Q-functions become:

$$Q_k^\pi = \left(T^\delta\right)^{k-1} T^\pi Q_k^\pi$$
$$Q_k^\star = \left(T^\delta\right)^{k-1} T^* Q_k^\star.$$

6.5 Bounding the Performance Loss

Learning in the space of k-persistent policies Π_k can only lower the performance of the optimal policy, i.e., $Q^\star(s,a) \geq Q_k^\star(s,a)$ for $k \in \mathbb{N}^+$ (since the general policy class is reduced to Π_k. The goal of this section is to provide some measures of the performance loss induced by persistence, i.e., to bound $\|Q^\star - Q_k^\star\|_{p,\mu}$ as a function of the persistence k. To this purpose, we focus on $\|Q^\pi - Q_k^\pi\|_{p,\mu}$ for a fixed policy $\pi \in \Pi$, since denoting with $\pi^\star$ an optimal policy of $\mathcal{M}$ and with $\pi_k^\star$ an optimal policy of $\mathcal{M}_k$, we have that:

$$Q^\star - Q_k^\star = Q^{\pi^\star} - Q_k^{\pi_k^\star} \leq Q^{\pi^\star} - Q_k^{\pi^\star},$$

Since $Q_k^{\pi_k^\star}(s,a) \geq Q_k^{\pi^\star}(s,a)$. We can start with the following result, where no assumption on the structure of the underlying MDP is considered. Afterward, we will provide a more particular result in the case of Lipschitz MDPs.

Theorem 6.1. *Let $\mathcal{M}$ be an MDP and $\pi \in \Pi$ be a Markovian stationary policy. Let $\mathcal{Q}_k = \{\left(T^\delta\right)^{k-2-l} T^\pi Q_k^\pi \,:\, l \in \{0,\dots,k-2\}\}$ and for all $(s,a) \in \mathcal{S}\times\mathcal{A}$ let us define:*

$$d_{\mathcal{Q}_k}^\pi(s,a) = \sup_{f\in\mathcal{Q}_k} \left| \int_{\mathcal{S}}\int_{\mathcal{A}} \left(p_\pi(\mathrm{d}s',\mathrm{d}a'|s,a) - p_\delta(\mathrm{d}s',\mathrm{d}a'|s,a)\right) f(s',a') \right|$$

Then, for any $\mu \in \Delta_{\mathcal{S}\times\mathcal{A}}$, $p \geq 1$, and $k \in \mathbb{N}^+$, it holds that:

$$\|Q^\pi - Q_k^\pi\|_{p,\mu} \leq \frac{\gamma(1-\gamma^{k-1})}{(1-\gamma)(1-\gamma^k)} \left\|d_{\mathcal{Q}_k}^\pi\right\|_{p,\eta_k^{\mu,\pi}},$$

where $\eta_k^{\mu,\pi} \in \Delta_{\mathcal{S}\times\mathcal{A}}$ is a probability measure defined for all $(s,a) \in \mathcal{S}\times\mathcal{A}$ as:

$$\eta_k^{\mu,\pi}(s,a) = \frac{(1-\gamma)(1-\gamma^k)}{\gamma(1-\gamma^{k-1})} \sum_{\substack{i\in\mathbb{N}\\ i \bmod k \neq 0}} \gamma^i \left(\mu\left(p_\pi\right)^{i-1}\right)(s,a).$$

The bound shows that the Q-function difference depends on the discrepancy $d_{\mathcal{Q}_k}^\pi$ between the transition-kernel p_π and the corresponding persistent version p_δ, which is a form of *integral probability metric* (Müller, 1997), defined in terms of the set $\mathcal{Q}_k$. This term is averaged with the distribution $\eta_k^{\mu,\pi}$, which encodes the γ-discounted probability of visiting a state-action pair (Sutton et al., 1999a), but ignoring the visitations made at decision steps multiple of k. Indeed, in those steps, policy π is followed regardless of persistence. The dependence on k is represented in the term $\frac{1-\gamma^{k-1}}{1-\gamma^k}$. When $k \to 1$ this term displays a linear growth in k, being asymptotic to $(k-1)\log\frac{1}{\gamma}$, and, clearly, vanishes for $k=1$. Instead, when $k \to \infty$ this term tends to 1.

If no structure on the MDP/policy is enforced, the dissimilarity term $d_{\mathcal{Q}_k}^\pi$ may become large enough to make the bound vacuous, and lead to arbitrarily large losses. Indeed, it is possible to provide the following negative result:

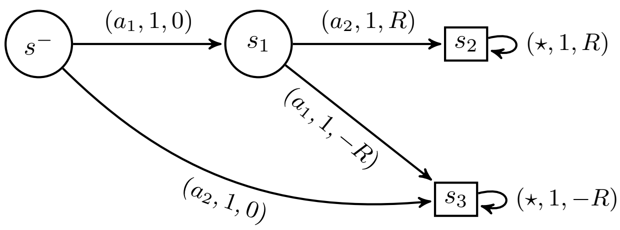


Figure 6.2: *MDP counter-example of Proposition 6.9, where $R > 0$. Each arrow connecting two states s and s' is labeled with the 3-tuple $(a, P(s'|s, a), r(s, a))$; the symbol $\star$ denotes any action in $\mathcal{A}$. While the optimal policy in the original MDP starting in s^- can avoid negative rewards by executing an action sequence of the kind $(a_1, a_2, \dots)$, every policy in the k-persistent MDP, with $k \geq 2$, inevitably ends in the negative terminal state, as the only possible action sequences are of the kind $(a_1, a_1, \dots)$ and $(a_2, a_2, \dots)$.*

Proposition 6.9. *For any MDP $\mathcal{M}$ and $k \in \mathbb{N}_{\geq 2}$ it holds that:*

$$V_k^*(s) \geq V^\star(s) - \frac{2\gamma R_{\max}}{1-\gamma}, \qquad \forall s \in \mathcal{S}. \tag{6.20}$$

Furthermore, there exists an MDP $\mathcal{M}^-$ (Figure 6.2) and a state $s^- \in \mathcal{S}$ such that the bound holds with equality for all $k \in \mathbb{N}_{\geq 2}$.

Proof. First of all, we recall that $V^\star(s) - V_k^\star(s) \geq 0$ since we cannot increase performance when executing a policy with a persistence k. Let $\pi^\star$ an optimal policy on the MDP $\mathcal{M}$, we observe that for all $s \in \mathcal{S}$:

$$V^*(s) - V_k^*(s) \leq V^{\pi^*}(s) - V_k^{\pi^*}(s), \tag{6.21}$$

since $V^{\pi^*}(s) = V^*(s)$ and $V_k^*(s) \geq V_k^{\pi^*}(s)$. Let us now consider the corresponding Q-functions $Q^{\pi^*}(s, a)$ and $Q_k^{\pi^*}(s, a)$. Recalling that they are the fixed points of the Bellman operators T^{π^*} and $T_k^{\pi^*}$ we have:

$$\begin{aligned}
Q^{\pi^*} - Q_k^{\pi^*} &= T^{\pi^*} Q^{\pi^*} - T_k^{\pi^*} Q_k^{\pi^*} \\
&= r + \gamma p_\pi Q^{\pi^*} - r_k - \gamma^k p_k^\pi Q_k^{\pi^*} \\
&= r + \gamma p_\pi Q^{\pi^*} - \sum_{i=0}^{k-1} \gamma^i \left(p_\delta\right)^i r - \gamma^k p_k^\pi Q_k^{\pi^*} \\
&= \gamma p_\pi Q^{\pi^*} - \sum_{i=1}^{k-1} \gamma^i \left(p_\delta\right)^i r - \gamma^k p_k^\pi Q_k^{\pi^*},
\end{aligned}$$

where we exploited the definitions of the Bellman expectation operators in the k-persistent MDP. As a consequence, we have that for all $(s, a) \in \mathcal{S} \times \mathcal{A}$:

$$\begin{aligned}
Q^{\pi^*}(s, a) - Q^{\pi^*}(s, a) &\leq \gamma \frac{R_{\max}}{1-\gamma} + R_{\max} \sum_{i=1}^{k-1} \gamma^i + \gamma^k \frac{R_{\max}}{1-\gamma} \\
&= \gamma \frac{R_{\max}}{1-\gamma} + R_{\max} \frac{\gamma(1-\gamma^{k-1})}{1-\gamma} + \gamma^k \frac{R_{\max}}{1-\gamma} = \frac{2\gamma R_{\max}}{1-\gamma},
\end{aligned}$$

where we considered that $\left(p_\pi Q^{\pi^\star}\right)(s,a) \leq \frac{R_{\max}}{1-\gamma}$, $\left(\left(p_\delta\right)^i r\right)(s,a) \leq R_{\max}$, and that $\left(p_k^\pi Q_k^{\pi^\star}\right) \leq R_{\max}$. The result follows since $V^{\pi^\star}(s) - V_k^{\pi^\star}(s) = \mathbb{E}\left[Q^{\pi^\star}(s,a) - Q^{\pi^\star}(s,a)\right]$, where $a \sim \pi^\star(\cdot|s)$.

Furthermore, the counterexample in Figure 6.2 proves that the bound is tight: the optimal policy must reach the terminal state s_2 yielding the positive reward $R > 0$. Thus the optimal policy plays action a_1 in state s^- and action a_2 in state s_1, generating a value function $V^*(s^-) = \frac{\gamma R}{1-\gamma}$. No persistent policies starting in state s^- can reach state s_2, since they will always end up in state s_3, yielding the negative reward $-R < 0$. Thus, the optimal value function will be $V_2^*(s^-) = -\frac{\gamma R}{1-\gamma}$ for all $k \geq 2$. □

Essentially, since action persistence repeats previous actions in new situations, it is necessary to ensure that the environment state changes relatively slowly over time and that the control policy executes similar actions in similar states. This implies that if an action is effective in a particular state, it will also likely be effective for similar states encountered in the near future. Although the condition on π is directly enforced by Assumption 3.2, we need a new notion of regularity over time for the MDP.

Assumption 6.1. *Let $\mathcal{M}$ be an MDP. $\mathcal{M}$ is L_T–*Time-Lipschitz Continuous *(L_T–TLC) if for all $(s,a) \in \mathcal{S} \times \mathcal{A}$:*

$$\mathcal{W}_1\left(P(\cdot|s,a), \delta_s\right) \leq L_T. \tag{6.22}$$

This assumption requires that the distance between the distribution of the next state s' and the deterministic distribution centered in the current state s is bounded by L_T, i.e., the system does not get too quickly far from s (see Appendix B.2.2). We can now state the following result.

Theorem 6.2. *Let $\mathcal{M}$ be an MDP and $\pi \in \Pi$ be a Markovian stationary policy. Under Assumptions 3.1, 3.2, and 6.1, if $\gamma \max\{L_P + 1, L_P(1+L_\pi)\} < 1$ and if $\mu(s,a) = \mu_{\mathcal{S}}(s)\pi(a|s)$ with $\mu_{\mathcal{S}} \in \Delta_{\mathcal{S}}$, then for any $k \in \mathbb{N}^+$:*

$$\left\|d_{\mathcal{Q}_k}^\pi\right\|_{p,\eta_k^{\mu,\pi}} \leq L_{\mathcal{Q}_k}\left[(L_\pi + 1)L_T + \sigma_p\right].$$

where:

$$\sigma_p^p = \sup_{s \in \mathcal{S}} \int_{\mathcal{A}} \int_{\mathcal{A}} d_{\mathcal{A}}\left(a, a'\right)^p \pi(\mathrm{d}a|s)\pi(\mathrm{d}a'|s),$$

$$L_{\mathcal{Q}_k} = \frac{L_r}{1 - \gamma \max\{L_P + 1, L_P(1+L_\pi)\}}.$$

The dissimilarity $d_{\mathcal{Q}_k}^\pi$ between p_π and p_δ can be bounded with four terms:

1. $L_{\mathcal{Q}_k}$ is an upper-bound of the Lipschitz constant of the functions in the set $\mathcal{Q}_k$. Indeed, under Assumptions 3.1 and 3.2 we can reduce the dissimilarity term to the Kantorovich distance (Lemma B.5):

$$d_{\mathcal{Q}_k}^\pi(s,a) \leq L_{\mathcal{Q}_k}\mathcal{W}_1\left(p_\pi(\cdot|s,a), p_\delta(\cdot|s,a)\right);$$

2. $(L_\pi + 1)$ accounts for the Lipschitz continuity of the policy, i.e., policies that prescribe similar actions in similar states have a small value of this quantity;

Algorithm 7 Persistent Fitted Q-Iteration PFQI(k).

```
Require: $k$ persistence, $J$ number of iterations ($T \bmod k = 0$), $Q^{(0)}$ initial action-value function,
    $\mathcal{F}$ functional space, $\mathcal{D} = \{(s_i, a_i, s'_i, r_i)\}_{i=1}^{|\mathcal{D}|}$ batch samples
Ensure: greedy policy $\widetilde{\pi}$
1:  for $j = 0, \dots, J-1$ do
2:      Build $TS = \{(x_i, y_i)\}_i$:
            $x_i = (s_i, a_i)$,
3:      if $t \bmod k = 0$ then                                          Phase 1
4:          $y_i = \widehat{T}^{\star} Q^{(j)}(s_i, a_i)$
5:      else
6:          $y_i = \widehat{T}^{\delta} Q^{(j)}(s_i, a_i)$
7:      end if
8:      Perform regression on $TS$ to induce $Q^{(j+1)}$:
9:          $Q^{(j+1)} \in \operatorname{arginf}_{f \in \mathcal{F}} \|f - y\|_{2,\mathcal{D}}^2$      Phase 2
10: end for
11:     $\widetilde{\pi}(s) \in \arg\max_{a \in \mathcal{A}} Q^{(J)}(s, a), \quad \forall s \in \mathcal{S}$      Phase 3
```

3. L_T represents the speed at which the environment state evolves over time;

4. σ_p denotes the average distance (in L_p-norm) between two actions prescribed by the policy in the same state. This term is zero for deterministic policies and can be related to the maximum policy variance (Lemma B.6).

6.6 Persistent Fitted Q-Iteration

In this section, we introduce an extension of FQI, introduced in Section 3.5 and adopted in the previous chapters, that employs the notion of persistence. Consequently, we will assume that the action space is discrete ($|\mathcal{A}| < +\infty$). *Persisted Fitted Q-Iteration* (PFQI(k)) takes as input a *target persistence* $k \in \mathbb{N}^+$ and its goal is to approximate the k-persistent optimal action-value function $Q_k^\star$. Starting from an initial estimate $Q^{(0)}$, at each iteration we compute the next estimate $Q^{(j+1)}$ by performing an approximate application of k-persistent Bellman optimal operator (definition 6.6) to the previous estimate $Q^{(j)}$, i.e., $Q^{(j+1)} \approx T_k^\star Q^{(j)}$.

Trivially, when samples are collected from the k-persistent MDP $\mathcal{M}_k$, the process outlined above becomes equivalent to the standard FQI. However, our algorithm needs to be able to estimate $Q_k^\star$ for different values of k, using the same dataset of samples collected in the base MDP $\mathcal{M}$ (at persistence 1, or with base control time step Δt_0).

For this purpose, we can exploit the decomposition $T_k^\star = (T^\delta)^{k-1} T^\star$ of Theorem 6.8 to reduce a single application of $T_k^\star$ to a sequence of k applications of the 1-persistent operators. Specifically, at each iteration j with $j \bmod k = 0$, given the current estimate $Q^{(j)}$, we need to perform (in this order) a single application of $T^\star$ followed by $k-1$ applications of T^δ, leading to the sequence of approximations:

$$Q^{(j+1)} \approx \begin{cases} T^* Q^{(j)} & \text{if } j \bmod k = 0 \\ T^\delta Q^{(j)} & \text{otherwise} \end{cases}. \tag{6.23}$$

To estimate the Bellman operators, we have access to a dataset of transitions $\mathcal{D} = \{(s_i, a_i, s_i', r_i)\}_{i=1}^{|\mathcal{D}|}$ collected, as in standard FQI, in the base MDP $\mathcal{M}$. In particular, $(s_i, a_i) \sim \nu$, where $\nu \in \Delta_{\mathcal{S}\times\mathcal{A}}$ is the sampling distribution; moreover, $s_i' \sim P(\cdot|s_i, a_i)$, $r_i \sim R(\cdot|s_i, a_i)$ following the MDP dynamics. We employ $\mathcal{D}$ to compute the *empirical Bellman operators* (Farahmand, 2011) defined for e bounded, measurable function $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ as:

$$(\widehat{T}^* f)(s_i, a_i) = r_i + \gamma \max_{a \in \mathcal{A}} f(s_i', a)$$
$$(\widehat{T}^\delta f)(s_i, a_i) = r_i + \gamma f(s_i', a_i).$$

These operators are unbiased conditioned to $\mathcal{D}$ (Farahmand, 2011):

$$\mathbb{E}[(\widehat{T}^* f)(s_i, a_i)|s_i, a_i] = (T^* f)(s_i, a_i)$$
$$\mathbb{E}[(\widehat{T}^\delta f)(s_i, a_i)|s_i, a_i] = (T^\delta f)(s_i, a_i).$$

The pseudocode of PFQI(k) is summarized in Algorithm 7. At each iteration $j = 0, \dots J-1$, the target values are computed by applying the empirical Bellman operators, $\widehat{T}^*$ or $\widehat{T}^\delta$, on the current estimate $Q^{(j)}$ (Phase 1). Then, the targets are projected onto the functional space $\mathcal{F}$ by solving the least squares problem (Phase 2):

$$Q^{(j+1)} \in \operatorname*{arginf}_{f \in \mathcal{F}} \|f - y\|_{2,\mathcal{D}}^2 = \frac{1}{n} \sum_{i=1}^{n} |f(s_i, a_i) - y_i|^2 .$$

Finally, the policy $\widetilde{\pi}$, which acts greedily w.r.t. $Q^{(J)}$ is extracted to approximate the optimal policy (Phase 3).

6.6.1 Theoretical Analysis

In this section, we present the computational complexity analysis and the study of the error propagation in PFQI(k).

Computational Complexity The computational complexity of PFQI(k) decreases monotonically with the persistence k. Whenever applying $\widehat{T}^\delta$, we need a single evaluation of $Q^{(j)}$, while $|\mathcal{A}|$ evaluations are needed for $\widehat{T}^*$ due to the $\max$ over $\mathcal{A}$. Thus, the overall complexity of J iterations of PFQI(k) with n samples, disregarding the cost of regression and assuming that a single evaluation of $Q^{(j)}$ takes constant time, is given by $\mathcal{O}\left(Jn\left(1 + \frac{|\mathcal{A}|-1}{k}\right)\right)$ (Proposition B.1).

Persistence trade-off We now consider the error propagation in PFQI(k). As in standard FQI, we have two sources of approximation: the representation of the Q-function through a functional space $\mathcal{F}$ to represent $Q^{(j)}$, and the approximate estimation of the (persistent) Bellman optimal operator $T_k^\star$ from samples.

Given the sequence of Q-functions estimates $\{Q^{(j)}\}_{j=0}^J \subset \mathcal{F}$ produced by PFQI(k), we define the approximation error at each iteration $j = 0, \dots, J-1$ as:

$$\epsilon^{(j)} = \begin{cases} T^* Q^{(j)} - Q^{(j+1)} & \text{if } j \bmod k = 0 \\ T^\delta Q^{(j)} - Q^{(j+1)} & \text{otherwise} \end{cases}. \tag{6.24}$$

The goal of this analysis is to bound the distance between the k–persistent optimal Q-function $Q_k^\star$ and the Q-function $Q_k^{\pi^{(J)}}$ of the greedy policy $\pi^{(J)}$ w.r.t. $Q^{(J)}$, after J iterations of PFQI(k). The following result extends Theorem 3.4 of Farahmand (2011) to account for action persistence.

Theorem 6.3 (Error Propagation). *Let $p \geq 1$, $k, J \in \mathbb{N}^+$ with $J \bmod k = 0$ and $\mu \in \Delta_{\mathcal{S}\times\mathcal{A}}$. Then for any sequence $\{Q^{(j)}\}_{j=0}^J \subset \mathcal{F}$ uniformly bounded by $Q_{\max} \leq \frac{R_{\max}}{1-\gamma}$, the corresponding $\{\epsilon^{(j)}\}_{j=0}^{J-1}$ defined in Equation* (6.24) *and for any $r \in [0,1]$ and $q \in [1, +\infty]$ it holds that:*

$$\left\| Q_k^\star - Q_k^{\pi^{(J)}} \right\|_{p,\mu} \leq \frac{2\gamma^k}{(1-\gamma)(1-\gamma^k)} \left[\frac{2}{1-\gamma} \gamma^{\frac{J}{p}} R_{\max} \right.$$
$$\left. + C_{\mathrm{VI},\mu,\nu}^{\frac{1}{2p}}(J, r, q) \mathcal{E}^{\frac{1}{2p}}(\epsilon^{(0)}, \dots, \epsilon^{(J-1)}; r, q) \right].$$

The expression of $C_{\mathrm{VI},\mu,\nu}(J; r, q)$ and $\mathcal{E}(\cdot; r, q)$ can be found in Appendix B.1.2, where the proof of the theorem is also provided.

We immediately observe that for $k = 1$ we recover Theorem 3.4 of Farahmand (2011). The term $C_{\mathrm{VI},\mu,\nu}(J; r, q)$ is defined in terms of suitable *concentrability coefficients* (Definition B.8) and encodes the distribution shift between the sampling distribution ν and the one induced by the greedy policy sequence $\{\pi^{(j)}\}_{j=0}^J$ encountered along the execution of PFQI(k). $\mathcal{E}(\cdot; r, q)$ incorporates the approximation errors $\{\epsilon^{(j)}\}_{j=0}^{J-1}$. In principle, it is hard to compare the values of these terms for different persistences k since both the greedy policies and the regression problems are different. Nevertheless, it is worth noting that the multiplicative term $\frac{\gamma^k}{1-\gamma^k}$ decreases in $k \in \mathbb{N}^+$. Thus, other things being equal, the bound value decreases with increasing persistence.

Thus, we can formally state the trade-off in the choice of control frequency, which motivates action persistence: the goal is finding the persistence $k \in \mathbb{N}^+$ that, for a fixed J, allows learning a policy $\pi^{(J)}$ whose Q-function $Q_k^{\pi^{(J)}}$ is the closest to $Q^\star$. Consider the decomposition:

$$\left\| Q^\star - Q_k^{\pi^{(J)}} \right\|_{p,\mu} \leq \| Q^\star - Q_k^\star \|_{p,\mu} + \left\| Q_k^\star - Q_k^{\pi^{(J)}} \right\|_{p,\mu}.$$

The term $\|Q^\star - Q_k^\star\|_{p,\mu}$ accounts for the performance degradation due to action persistence: it is algorithm–independent, and it increases in k (Theorem 6.1). Instead, the second term $\|Q_k^\star - Q_k^{\pi^{(J)}}\|_{p,\mu}$ decreases with k and depends on the algorithm (Theorem 6.3). Unfortunately, optimizing their sum is hard since the individual bounds

Algorithm 8 Heuristic Persistence Selection.

```
Require: Batch samples $\mathcal{D} = \{\tau_i)\}_{i=1}^m$, set of persistences $\mathcal{K}$,
    set of Q-functions $\{Q_k : k \in \mathcal{K}\}$, regressor Reg
Ensure: Approximate optimal persistence $\widetilde{k}$
1: for $k \in \mathcal{K}$ do
2:     $\widehat{J}_k = \frac{1}{m}\sum_{i=1}^m V_k(s_0^i)$
3:     Use Reg to get an estimate $\widetilde{Q}_k$ of $T_k^\star Q_k$
4:     $$\|\widetilde{Q}_k - Q_k\|_{1,\mathcal{D}} = \frac{\sum_{i=1}^m \sum_{t=0}^{T(\tau_i)-1} |\widetilde{Q}_k(s_t^i, a_t^i) - Q_k(s_t^i, a_t^i)|}{\sum_{i=1}^m H_i}$$
5: end for
6: $$\widetilde{k} \in \operatorname*{arg\,max}_{k \in \mathcal{K}} B_k = \widehat{J}_k - \frac{1}{1-\gamma^k}\|\widetilde{Q}_k - Q_k\|_{1,\mathcal{D}}.$$
```

contain terms that are not known in general (e.g., Lipschitz constants, $\epsilon^{(j)}$). The next section proposes heuristics to overcome this problem.

6.7 Persistence Selection

In this section, we analyze the problem of persistence selection (or control frequency adaptation), i.e., how to select k in a set $\mathcal{K} \subset \mathbb{N}^+$ of candidate persistences, when we are given a set of estimated Q-functions: $\{\widehat{Q}_k\}_{k \in \mathcal{K}}$. For instance, the $\widehat{Q}_k$ can be obtained by executing PFQI(k) with different target persistences $k \in \mathcal{K}$. Each $\widehat{Q}_k$ induces a greedy policy π_k. Our goal is to find the persistence $k \in \mathcal{K}$ such that π_k has the maximum expected return in the corresponding k–persistent MDP $\mathcal{M}_k$:

$$k^\star \in \operatorname*{arg\,max}_{k \in \mathcal{K}} J_k(\pi_k). \tag{6.25}$$

In principle, we could execute π_k in $\mathcal{M}_k$ to get an estimate of $J_k(\pi_k)$ and employ it to select the persistence k. However, in the batch setting, further interactions with the environment might be not allowed. On the other hand, directly using the estimated Q-function $\widetilde{Q}_k$ is inappropriate, since we should take into account the approximation error w.r.t. the true value function $Q_k^{\pi_k}$. This trade-off is encoded in the following result, which makes use of the *expected Bellman residual*.

Lemma 6.10. *Let $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be a bounded, measurable action-value function, and let π be a greedy policy w.r.t. Q. Let $J = \int \mu(\mathrm{d}s)V(s)$, with $V(s) = \max_{a \in \mathcal{A}} Q(s,a)$ for all $s \in \mathcal{S}$. Then, for any $k \in \mathbb{N}^+$, it holds that:*

$$J_k(\pi) \geq J(\pi) - \frac{1}{1-\gamma^k}\|T_k^\star Q - Q\|_{1,\eta_k^{\mu,\pi}}, \tag{6.26}$$

where $\eta_k^{\mu,\pi} = (1-\gamma^k)\mu\pi\left(\mathrm{Id} - \gamma^k p_k^\pi\right)^{-1}$, is the γ-discounted stationary distribution induced by policy π and initial distribution μ in MDP $\mathcal{M}_k$.

Some simplifications are needed in order to obtain a practical bound. First, we assume that the batch of tuples $\mathcal{D} \sim \nu$ is composed of m *trajectories*, i.e., $\mathcal{D} = \{\tau_i\}_{i=1}^m$,

Table 6.1: *Results of PFQI in different environments and persistences. For each persistence k, we report the sample mean and the standard deviation of the estimated return of the last policy $\widehat{J}_k(\pi_k)$. For each environment, the persistence with the highest average performance and the ones not statistically significantly different from that one (Welch's t-test with $p < 0.05$) are in bold. The last column reports the mean and the standard deviation of the performance loss δ between the optimal persistence and the one selected by the index B_k (Equation (6.27)).*

Environment	Expected return at persistence k ($\widehat{J}_k(\pi_k)$, mean $\pm$ std)						
	$k=1$	$k=2$	$k=4$	$k=8$	$k=16$	$k=32$	$k=64$
Cartpole	169.9 ± 5.8	176.5 ± 5.0	$\mathbf{239.5 \pm 4.4}$	10.0 ± 0.0	9.8 ± 0.0	9.8 ± 0.0	9.8 ± 0.0
MountainCar	-111.1 ± 1.5	-103.6 ± 1.6	-97.2 ± 2.0	$\mathbf{-93.6 \pm 2.1}$	$\mathbf{-94.4 \pm 1.8}$	$\mathbf{-92.4 \pm 1.5}$	-136.7 ± 0.9
LunarLander	-165.8 ± 50.4	-12.8 ± 4.7	$\mathbf{1.2 \pm 3.6}$	$\mathbf{2.0 \pm 3.4}$	-44.1 ± 6.9	-122.8 ± 10.5	-121.2 ± 8.6
Pendulum	$\mathbf{-116.7 \pm 16.7}$	$\mathbf{-113.1 \pm 16.3}$	$\mathbf{-153.8 \pm 23.0}$	-283.1 ± 18.0	-338.9 ± 16.3	-364.3 ± 22.1	-377.2 ± 21.7
Acrobot	-89.2 ± 1.1	$\mathbf{-82.5 \pm 1.7}$	$\mathbf{-83.4 \pm 1.3}$	-122.8 ± 1.3	-266.2 ± 1.9	-287.3 ± 0.3	-286.7 ± 0.6
Swimmer	21.3 ± 1.1	$\mathbf{25.2 \pm 0.8}$	$\mathbf{25.0 \pm 0.5}$	$\mathbf{24.0 \pm 0.3}$	22.4 ± 0.3	12.8 ± 1.2	14.0 ± 0.2
Hopper	58.6 ± 4.8	61.9 ± 4.2	62.2 ± 1.7	59.7 ± 3.1	60.8 ± 1.0	66.7 ± 2.7	$\mathbf{73.4 \pm 1.2}$
Walker 2D	61.6 ± 5.5	37.6 ± 4.0	62.7 ± 18.2	$\mathbf{80.8 \pm 6.6}$	$\mathbf{102.1 \pm 19.3}$	$\mathbf{91.5 \pm 13.0}$	$\mathbf{97.2 \pm 17.6}$

Environment	Performance loss (δ mean $\pm$ std)
Cartpole	0.0 ± 0.0
MountainCar	1.88 ± 0.85
LunarLander	2.12 ± 4.21
Pendulum	3.52 ± 0.0
Acrobot	0.80 ± 0.27
Swimmer	2.69 ± 1.71
Hopper	5.33 ± 2.32
Walker 2D	5.10 ± 3.74

where the initial states s_0^i are sampled from μ. In this way, the expected returns can be estimated from samples as $\widehat{J} = \frac{1}{m}\sum_{i=1}^{m} V(s_0^i)$. Moreover, since we are unable to compute expectations over $\eta_k^{\mu,\pi}$, we replace it with the sampling distribution ν and, by means of an approach similar to (Farahmand and Szepesvári, 2011), we assume to have a regressor $\mathtt{Reg}$ able to output an approximation $\widetilde{Q}_k$ of $T_k^\star Q$, thanks to which we can we replace $\|T_k^\star Q - Q\|_{1,\nu}$ with $\|\widetilde{Q}_k - Q\|_{1,\mathcal{D}}$. The discussion regarding these approximation steps is delicate, and a more detailed discussion is provided in Appendix C.1 in Metelli et al. 2020. In practice, we set $Q_k = Q_k^{(J)}$ from PFQI(k) with target persistence k, and we obtain $\widetilde{Q}_k$ through k additional iterations of the algorithm, setting $\widetilde{Q}_k = Q^{(J+k)}$. Thus, the procedure (Algorithm 8) reduces to optimizing the index:

$$\widetilde{k} \in \operatorname*{arg\,max}_{k \in \mathcal{K}} B_k = \widehat{J}_k - \frac{1}{1-\gamma^k} \left\| \widetilde{Q}_k - Q_k \right\|_{1,\mathcal{D}}. \tag{6.27}$$

6.8 Experimental Evaluation

In this section, we provide the empirical evaluation of PFQI, with different goals: proving that a persistence $k > 1$ can boost learning, possibly leading to more profitable policies, and assessing the quality of our persistence selection heuristics, We invite the reader to refer to Appendix B.3 for details regarding the experimental settings, the hyperparameters adopted, and further results. Moreover, in the original paper Metelli et al., 2020, further results are provided as, for instance, an analysis of the influence of the batch size on the performance of PFQI policies for different persistences. We

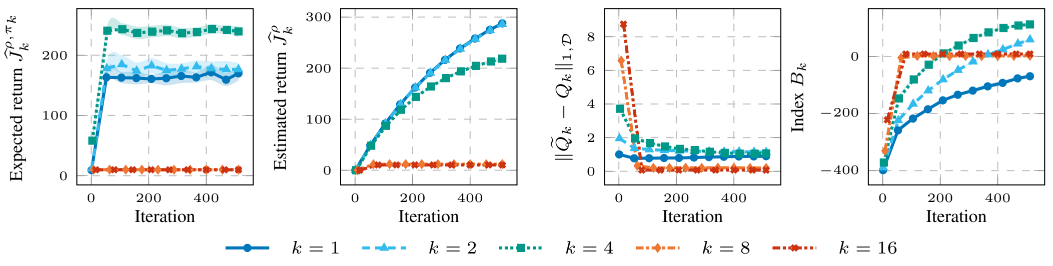


Figure 6.3: *Expected return $\widehat{J}_k(\pi_k)$, estimated return $\widehat{J}_k$, estimated expected Bellman residual $\|\widetilde{Q}_k - Q_k\|_{1,\mathcal{D}}$, and persistence selection index B_k in the Cartpole experiment as a function of the number of iterations for different persistences. 20 runs, 95 % c.i.*

train PFQI, using ExtraTrees (Geurts et al., 2006) as a regression model, for J iterations and different values of k, starting with the same dataset $\mathcal{D}$ collected at persistence 1. To compare the performance of the learned policies π_k at the different persistences, we estimate their expected return $\widehat{J}_k(\pi_k)$ with 10 runs in the corresponding MDP $\mathcal{M}_k$. Table 6.1 shows the results for different continuous environments and different persistences averaged over 20 runs of PFQI(k). We highlight in bold the persistence with the highest average performance and the ones that are not statistically significantly different from that one. Across the different environments, we observe some common trends in line with our theory: persistence 1 seldom leads to the best performance, and excessively increasing persistence prevents control at all. In Cartpole (Barto et al., 1990), we easily identify a persistence ($k = 4$) that outperforms all the others. In the Lunar Lander (Brockman et al., 2016) persistences $k \in \{4, 8\}$ are the only ones that lead to positive return (i.e., the lander does not crash) and in the Acrobot domain (Geramifard et al., 2015) we identify $k \in \{2, 4\}$ as optimal persistences. A qualitatively different behavior is displayed in Mountain Car (Moore, 1991), Pendulum (Brockman et al., 2016), and Swimmer (Coulom, 2002), where we observe a plateau of three persistences with similar performance. An explanation for this phenomenon is that, in those domains, the optimal policy tends to persist actions on its own, making the difference less evident. Intriguingly, more complex Mujoco domains, like Hopper and Walker 2D (Erickson et al., 2019), seem to benefit from the higher persistence values.

To test the quality of our persistence selection method, we compare the performance of the estimated optimal persistence, i.e., the one with the highest estimated expected return $\widehat{k} \in \arg\max \widehat{J}_k(\pi_k)$, and the performance of the persistence $\widetilde{k}$ selected by maximizing the index B_k (from (6.27)). For each run $i = 1, \dots, 20$, we compute the *performance loss* $\delta_i = \widehat{J}_{\widehat{k}}(\pi_{\widehat{k}}) - \widehat{J}_{\widetilde{k}_i}(\pi_{\widetilde{k}_i})$ and we report the mean and standard deviations in the “Performance loss” column of Table 6.1. In the Cartpole experiment, we observe a zero loss, which means that our heuristic always selects the optimal persistence ($k = 4$). Differently, non–zero loss occurs in the other domains, which means that sometimes the index B_k mispredicts the optimal persistence. Nevertheless, in almost all cases the average performance loss is significantly smaller than the magnitude of the return, proving the effectiveness of our heuristics.

In Figure 6.3, we provide an in-depth analysis of the learning curves for the Cartpole environment, highlighting the components that contribute to the index B_k. The first plot reports the estimated *expected return* $\widehat{J}_k(\pi_k)$, obtained from 10 trajectories executing

π_k in the environment $\mathcal{M}_k$ throughout PFQI(k) iterations, which confirms that $k = 4$ is the optimal persistence. In the second plot, we report the *estimated return* $\widehat{J}_k$ obtained by averaging the Q-function Q_k learned with PFQI(k), over the initial states sampled from μ. We can see that for $k \in \{1, 2\}$, PFQI(k) tends to overestimate the return, while for $k = 4$ we notice a slight underestimation. The overestimation phenomenon can be explained by the fact that, with small persistences, we perform a large number of applications of the operator $\widehat{T}^\star$, which involves a maximization over the action space, injecting an overestimation bias. By combining this curve with the expected Bellman residual (third plot), we get the value of our persistence selection index B_k (fourth plot). Finally, we observe that B_k is able to correctly rank persistences 4 and 8, but overestimates persistences 8 and 16, compared to persistence 1. The results related to the other environments are reported in Appendix B.3.

6.9 Application to Foreign Exchange Trading

In this Section, we provide an application of the proposed PFQI approach in the financial field. As shown in Example 6.4, RL applications to finance have drawn more and more attention for its goal being well aligned with trading objectives (Fischer, 2018; Meng and Khushi, 2019; Bacoyannis et al., 2018). Due to the diversity of market participants and investment strategies (Mantegna and Stanley, 1999; Bouchaud and Potters, 2003), it is reasonable to consider that the market is built upon different time scales. It is therefore of interest, for the designer of a trading algorithm, to select the elementary scale at which the algorithm will interact with the market. However, in the field of RL for trading, the study of the impact of the time-scale hasn't been a primary focus: thus we wonder what is the impact of persistence for artificial traders acting on FX, one of the most liquid markets. We extend a previous work (Bisi et al., 2020a), where FQI was adopted to retrieve an agent acting every minute on a single FX currency pair. The following results are extracted from Riva et al. 2021.

6.9.1 MDP Model for FX Trading

We model a generic trading task on a single asset as an MDP with a *discrete* action set. In the simplest case, *three* possible allocations are sufficient: $Long, Short,$ or $Flat$. These actions are referred to a fixed quantity of an asset we want to trade. The reward is modeled as follows:

$$R_{t+1} = \underbrace{a_t(P_{t+1} - P_t)}_{\text{P\&L from market changes}} - \underbrace{f_t|a_t - a_{t-1}|}_{\text{transaction costs}} \tag{6.28}$$

where s is the portfolio, a is the action, P is the price of the asset expressed in some currency, and f is the fee multiplier, set to 1\$ for a fixed total allocation of $100k$\$. The first part consists of the gain (loss) derived from trades, and the second one corresponds to costs due to changing allocation. We assume that the dynamic of the MDP is not controllable for what concerns the exchange rates, meaning that the allocations are not large enough to move the market. The only feature of the state which is affected by the agent's actions is its current allocation. The considered episodes are only one business day long, from 8:00 to 18:00 CET, with 1-minute long time steps; hence, we use the

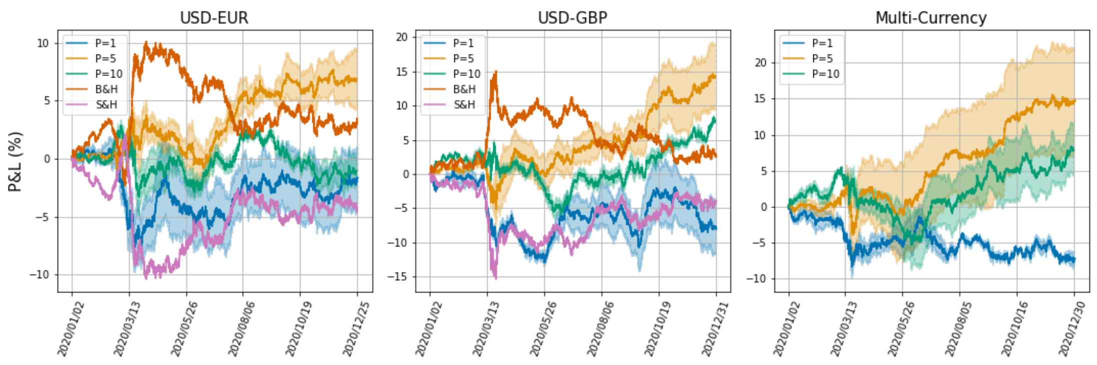


Figure 6.4: *Cumulative return (P&L stands for Profit&Loss) in test for each of the different currency pairs combinations and different persistence values. Results are shown together with B&H and S&H baselines. Performances are reported as percentages w.r.t. the invested amount.*

undiscounted setting (i.e. $\gamma = 1$). The features adopted to represent the state are the following:

- the **last 60 exchange rate variations** between consecutive minutes;
- the corresponding **time of the day**, expressed in minutes;
- the current **portfolio position** w.r.t the currency pairs, equivalent to the previous action selected as allocation.

Three-Currencies Model In this work, we consider also a three-currencies scenario, which can be seen as a trading task in which two assets can be traded (e.g., USD-EUR and USD-GBP, where USD is considered as *domestic* currency). We disallow the positions involving simultaneous allocations on different foreign currencies, i.e., it is only possible to be long or short w.r.t to one pair at each timestep. Therefore, we allow the agent to take the 5 possible positions which correspond to being long (or short) w.r.t. each of the foreign currencies, or to being flat w.r.t both. The agent can switch from one currency pair to the other one in just one step, but such an operation would involve a doubled transaction cost.

Results We trained PFQI models using data regarding USD-EUR and USD-GBP pairs from 2017 to 2020 and built the full transition tuples associating to market values all the possible portfolio-action configurations and the correspondent rewards, computed using Equation 6.28. In order to analyze the impact of the persistence on the trading algorithm performances, we chose to consider three different persistence values (1, 5, and 10) w.r.t. a 1-minute sampling frequency, both in the multi-currency scenario and in the single-currency one. For each persistence value and scenario, PFQI was trained in 2017-2018 using different min_split thresholds and with 5 maximum iterations, and the best models were selected using 2019 market values as validation set. Finally, we tested the performance of these models in 2020, and the results are shown in Figure 6.4. The returns are denoted as *P&L* and represent the incurred *Profit&Loss*. We compared the performance of the two-currencies models with two benchmark strategies: the Buy&Hold and the Sell&Hold. Both are passive strategies that consist in keeping

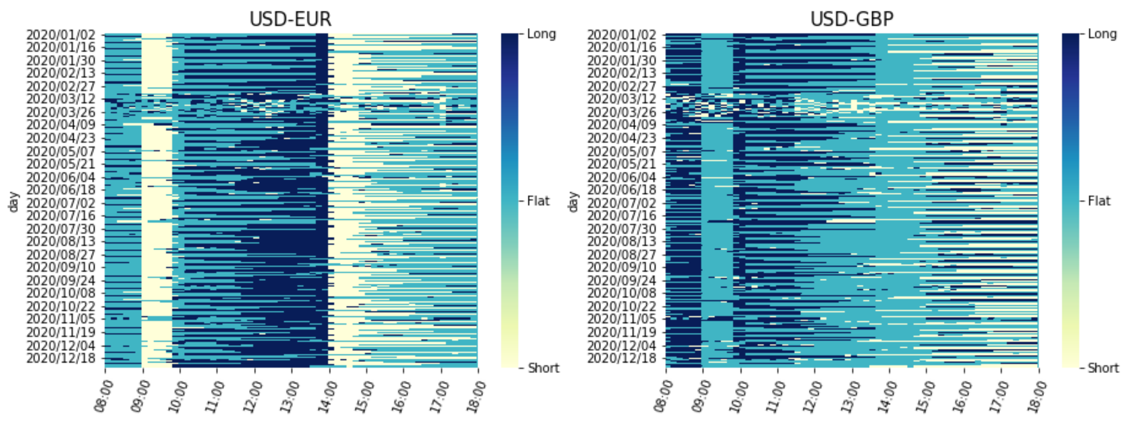


Figure 6.5: *Portfolio allocation chosen by the three-currency agent trained with persistence equal to 10. Each row corresponds to a different business day, and each column is specific for a trading minute.*

a constant position, respectively, long or short. As shown in Figure 6.4, models trained with persistence equal to 5 outperform all the other models and both the benchmark strategies. On the other hand, models trained with persistence equal to 1 are those characterized by the lowest cumulative returns at the end of the year. Moreover, looking at the policies learned by the models, we can also notice that the higher the persistence is, the better the agent exploits temporal patterns. As we can observe in Figure 6.5, these patterns can be identified by looking for vertical stripes of the same color in the allocation heatmaps. For instance, the agent of the three-currencies model trained with persistence equal to 10 learned to be long w.r.t USD-GBP during the first hour of most of the days, then it usually changes the portfolio allocation moving to a short position w.r.t USD-EUR and keeping it until 10:00. Some of these patterns are associated with particular events which characterize the trading day: when American traders enter in the FX market around 14:00, the agent usually changes its position with respect to USD-EUR from long to short. Finally, it is worth noting that the performances of all the models are strongly affected by multiple drawdowns registered between March and May of 2020, which might be related to the high volatility and unpredictability of the Forex market due to the spread of the Covid-19 pandemic. This strong impact of the pandemic can also be observed by looking at the portfolio allocations displayed in Figure 6.5, where it can be easily noticed how the solid temporal patterns learned by the agent do not hold during the whole month of March when the pandemic exploded. Nevertheless, higher persistence models were able to recover from the drawdown, ending up with a positive cumulated return. For further analysis and discussions, we invite the reader to refer to Riva et al. (2021).

6.10 Conclusions

In this chapter, we formalized the notion of action persistence, i.e., the repetition of a single action for a fixed number k of decision epochs, having the effect of altering the control frequency of the system. We have shown that persistence leads to the definition of new Bellman operators and that we are able to bound the induced performance loss, under some regularity conditions on the MDP. Based on these considerations, we presented and analyzed a novel batch RL algorithm, PFQI, able to approximate the value function at a given persistence. The experimental evaluation justifies the introduction of persistence, since reducing the control frequency can lead to an improvement when dealing with a limited number of samples. Furthermore, we introduced a persistence selection heuristic, which can identify good persistence in most cases. As a realistic application of our approach, we also performed PFQI to train artificial traders on the FX market: the resulting agent acting once every 5 minutes could better exploit the control frequency trade-off, detecting patterns more easily than the one acting every minute and obtaining more profitable strategies than the one persisting its position for 10 minutes. This works makes a step towards understanding why repeating actions may be useful for solving complex control tasks. Numerous questions remain unanswered, leading to several appealing future research directions, some of which are discussed in Appendix B.4. For instance, we may wonder how well the same sampling policy (e.g., the uniform policy over $\mathcal{A}$), executed at different persistences, explores the environment. A clear example is provided by Mountain Car, where high persistences increase the probability of reaching the goal, generating more informative datasets (preliminary results in Appendix B.4.1). These preliminary results act as the main motivation for considering an adaptive control frequency, as will be discussed in more detail in Chapter 7. Furthermore, we empirically analyzed what happens when a policy is learned by PFQI with a certain persistence level k and executed later on with a different persistence level $k' \neq k$. This helped us understand that policies that are learned on an MDP with a high persistence can be useful also in the environment with base time discretization.

CHAPTER 7

All-persistence Bellman Update

7.1 Introduction

In Chapter 6, we introduced *action persistence* to deal with the selection of the best choice of time discretization: indeed, the performance of RL learning agents is highly sensitive to the frequency of the interaction with the environment. Agents acting at high frequencies have the best control opportunities, along with some drawbacks, for instance, the increasing sample complexity due to the vanishing of the action advantages. Another possible limitation of a high-frequency agent is the inefficiency of exploration and the vanishing of action advantages. As seen, the repetition of the actions through *action persistence* comes into help, as it allows the agent to visit wider regions of the state space and improve the estimation of the action effects. One of the main limitations related to the previous approach is related to the selection of a fixed persistence value throughout the entire learning session: in this chapter, we wonder whether it is possible to consider a *dynamic persistence* i.e., an adaptive selection of the duration of the selected actions. To this extent, we derive a novel *All-Persistence Bellman Operator*, which allows for effective use of both the low-persistence experience, by decomposition into sub-transition, and the high-persistence experience, thanks to the introduction of a suitable *bootstrap* procedure. In this way, we employ transitions collected at *any* time scale to update simultaneously the action values of the considered persistence set. This novel operator maintains all the contraction properties of its standard versions (the Bellman Expectation and Optimality Operators), hence we have the chance to build learning algorithms with useful convergence guarantees. Therefore, we start by extending classic Q-learning, and we can empirically analyze the main advantages introduced with a dynamic selection of action persistence: more efficient exploration and faster propagation of the updates. Furthermore, we provide also an extension of one of the most known value-based algorithms, DQN. In conclusion, we experimentally evaluate our approach in both tabular contexts and more challenging frameworks, including some Atari games.

Chapter Outline This chapter is organized as follows: In Section 7.4 we introduce the *All-persistence Bellman Operator*. We prove that such an operator enjoys a contraction property analogous to that of the traditional optimal Bellman operator. Consequently, in Section 7.5 we embed the All-persistence Bellman operator into the classic Q-learning algorithm, obtaining *Persistent Q-learning* (Per$Q$-learning). This novel algorithm, through effective use of the transitions sampled at different persistences, displays two main advantages. First, since each transition is employed to update the value function estimates at different persistences, we experience a faster convergence. Second, the execution of persistent actions, given the nature of a large class of environments, fosters exploration of the state space, with a direct effect on the learning speed (Section 7.6). Furthermore, to deal with more complex domains, we move, in Section 7.7, to the Deep RL scenario, extending the Deep Q-Network (DQN) algorithm to its persistent version, called *Persistent Deep Q-Network* (PerDQN). Finally, in Section 7.8, we evaluate the proposed algorithms, in comparison with state-of-the-art approaches, on illustrative and complex domains, highlighting strengths and weaknesses. Further discussions and results are reported in Appendix C.

7.2 Motivations

As seen in the previous chapters, sequential decision-making problems are typically modeled as a MDP, a formalism that addresses the agent-environment interactions through *discrete-time* transitions. This holds also for *continuous-time* control problems, which are usually addressed by means of time discretization, which induces a specific control frequency (Park et al., 2021). From Chapter 6, we considered it as an environment hyperparameter, which may have dramatic effects on the learning process. Indeed, higher frequencies allow for greater control opportunities, but they have significant drawbacks. Among the most important drawbacks we have reported in Section 6.2, we remark on the vanishing effect on the advantage functions (Baird, 1994; Tallec et al., 2019). Another consequence of the use of high frequencies is related to the *inefficiency of exploration*: a random uniform policy played at high frequency may not be adequate, as in some classes of environments, including the majority of real-world control problems, it tends to visit only a local neighborhood of the initial state (Amin et al., 2021; Park et al., 2021; Yu et al., 2021). This is problematic, especially in goal-based or sparse rewards environments, where the most informative states may never be visited. On the other hand, larger time discretizations benefit from a higher probability of reaching far states, but they also deeply modify the transition process, hence a possibly large subspace of states may not be reachable.

As seen, one of the solutions to achieve the advantages related to exploration and sample complexity, while keeping the control opportunity loss bounded, consists in *action persistence*, introduced in the previous chapter and also adopted in (Schoknecht and Riedmiller, 2003; Braylan et al., 2015; Lakshminarayanan et al., 2017). Thus, the agent can achieve, in some environments, a more effective exploration, better capture the consequences of each action, and fasten convergence to the optimal policy. In the previous chapter, we exploited this trade-off to detect a *static* action persistence, i.e., the most effective duration of the actions, fixed for all states, and throughout the whole learning instances. Instead, in this chapter we aim to enhance learning by selecting a *dynamic* persistence: we propose a value-based approach in which the agent does not only choose the *action*, but also its *persistence*, with the goal of making the most effective use of samples collected at different persistences. Indeed, the information collected from the interaction with the environment at *one* persistence is used to improve the action value function estimates of *all* the considered possible persistences. On one hand, the multi-step transitions can be decomposed to consider trajectories of reduced length and used to update knowledge related to lower persistence values. On the other hand, these same transitions represent partial information for the estimation of the effects of higher persistent actions. Therefore, they can be employed to update the estimates by using a suitable *bootstrapping* procedure of the missing information. Thus, all value function estimates are updated simultaneously for each of the available persistences $k \in \mathcal{K}$.

7.3 Related Work

In this section, we revise some approaches that take into account adaptive action durations. We recall that some related works on general temporal abstraction and temporally extended actions in RL are presented in Section 6.3. Among the first works extending RL agents with a dynamic action repetition go back to Schoknecht and Riedmiller 2003. In this paper, the authors introduce Multi-Step Action (MSA)s, which are equivalent to action persistence since they “consist of a sequence of the same primitive action that is applied for consecutive tome steps”. They show that making the time scale coarser through MSAs helps reduce the number of decisions needed to reach the goal by lowering its *First Passage Time*. The exploration boost obtained with temporally extended actions is fundamental also in Dabney et al. 2020, which extends the definition of $\epsilon-$greedy policy (Definition 3.1) with a random exploratory variable deciding the duration of each action (*$\epsilon z-$greedy exploration*). In recent works, researchers have been trying to include the possibility to *dynamically* learn the control frequency during learning: in Augmented-DQN (Lakshminarayanan et al., 2017), the action space is duplicated to be able to choose actions with two previously selected repetition rates. A different approach is proposed in Sharma et al. 2017, which introduces the concept of *skip network*, a second network used to select the persistence in a specific state, regardless of the chosen action. However, the independence of the second network to the selected action leads only to learning persistences that work well on average for all actions. One way to differentiate persistences with actions is proposed by TempoRL (Biedenkapp et al., 2021), where the skip network depends on both state and action (and the estimated Q-value functions) to evaluate the effects for different possible frequencies. The dedicated network is trained to learn the persistence at each state-action pair and employs a standard replay buffer, ignoring the persistence at which samples have been collected. In G. Bellemare et al. 2016 *persistent advantage learning* is proposed, in which advantage learning from (Baird, 1994) is overridden by a Q-learning update with repeated actions when the latter promises better Q-values.

In the framework of policy-gradient methods, persistence is introduced in Yu et al. 2021, with the introduction of a secondary policy for choosing whether to repeat the previous action or to change it according to the principal agent. A completely different approach is presented by Park et al. 2021: the authors show that when $\delta \to 0$ policy-based methods tend to degrade. Thanks to the introduction of a *safe region*, the agent keeps repeating an action until the distance of the visited states overcomes a certain threshold. This state locality can guarantee reactivity but may lead to unsafe behaviors.

7.4 All-Persistence Bellman Update

In this section, we introduce our approach to make effective use of the samples collected at *any* persistence. We first introduce the notion of *persistence option* and then, we present the *all-persistence Bellman operator*.

7.4.1 Persistence Options

We formalize the decision process in which the agent chooses a *primitive* action a together with its persistence k, belonging to a *persistence space* $\mathcal{K}$, which we assume to be discrete and limited, i.e., there exist a *maximum persistence* $K_{max} \in \mathbb{N}^+$ such that $\mathcal{K} = [K_{max}]$. To this purpose, we introduce the *persistence option*.

Definition 7.1. *Let $\mathcal{A}$ be the space of* primitive *actions of an MDP $\mathcal{M}$ and $\mathcal{K} := \{1, \dots, K_{\max}\}$ be the persistence space, where $K_{\max} \in \mathbb{N}^+$. A* persistence option *$o := (a, k)$ is the decision of playing primitive action $a \in \mathcal{A}$ with persistence $k \in \mathcal{K}$. We denote with $\mathcal{O}^{(k)} := \{(a, k) : a \in \mathcal{A}\}$ the set of options with fixed persistence $k \in \mathcal{K}$ and $\mathcal{O} := \bigcup_{k \in \mathcal{K}} \mathcal{O}^{(k)} = \mathcal{A} \times \mathcal{K}$.*

The decision process works as follows. At time $t = 0$, the agent observes $s_0 \in \mathcal{S}$, selects a persistence option $o_0 = (a_0, k_0) \in \mathcal{O}$, observes the sequence of states $(s_1, \dots, s_{k_0})$ generated by repeating primitive action a_0 for k_0 times, such that $s_{i+1} \sim P(\cdot|s_i, a_0)$ for $i \in \{0, \dots, k_0 - 1\}$, and the sequence of rewards $(r_1, \dots, r_{k_0})$ with $r_{i+1} = r(s_i, a_0)$ for $i \in \{0, \dots, k_0 - 1\}$. Then, in state s_{k_0} the agent selects another option $o_1 = (a_1, k_1) \in \mathcal{O}$ and the process is repeated. During the execution of the persistence option, the agent is not allowed to change the primitive action.[1]

Remark 7.1. (**Persistence and Options**) *The persistence option (Definition 7.1) is in all regards a* semi-Markov option *(Precup, 2001), where the initiation set is the set of all states $\mathcal{S}$, the termination condition is only time-dependent, and the intra-option policy is constant. Indeed, the described process generates a* semi-Markov decision process *(Puterman, 2014), fully determined by the behavior of $\mathcal{M}$, as shown in Sutton et al. (1999b).*

Remark 7.2. (**Persistence Options vs Augmented Action Space**) *There is an important difference between using persistence options $\mathcal{O}$ in the original MDP $\mathcal{M}$ and defining an augmented MDP $\mathcal{M}_{\mathcal{K}}$ with new action space $\mathcal{A} \times \mathcal{K}$ and properly redefined transition model and reward function (Lakshminarayanan et al., 2017; Biedenkapp et al., 2021): when executing a persistence option $o_t = (a_t, k_t) \in \mathcal{O}$ at time t, we observe the* full *sequence of states $(s_{t+1}, \dots, s_{t+k_t})$ and rewards $(r_{t+1}, \dots, r_{t+k_t})$. Instead, in the augmented MDP $\mathcal{M}_{\mathcal{K}}$ we* only *observe the last state s_{k_t} and the cumulative reward $r^k_{t+1} = \sum_{i=0}^{k-1} \gamma^i r_{t+i+1}$. We will heavily exploit the particular option structure, re-using fragments of experience to perform intra-option learning.*

[1]From this definition, it follows that $\mathcal{A}$ is isomorphic to $\mathcal{O}^{(1)}$.

We now extend the policy and state-action value function definitions to consider this particular form of options. A *Markovian stationary policy over persistence options* $\psi : \mathcal{S} \to \Delta_{\mathcal{O}}$ is a mapping between states and probability measures over persistence options. We denote with Ψ the set of policies of this nature. We can therefore extend Definition 2.12 of action-value function to the set of persistence options:

Definition 7.3 (State-Option Value function). *Let $\mathcal{M}$ be an MDP, ψ a policy in Ψ and (s, a, k) any state-action pair in $\mathcal{S} \times \mathcal{A} \times \mathcal{K}$. The state-option value function $Q^{\psi} : \mathcal{S} \times \mathcal{A} \times \mathcal{K} \to \mathbb{R}$ is defined as the expected return starting from (s, a, k), and then following the policy ψ:*

$$Q^{\psi}(s,a,k) \coloneqq \mathop{\mathbb{E}}_{\psi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a, k_0 = k\right]. \tag{7.1}$$

In the same fashion, we can define the optimal action-value function:

$$Q^{\star}_{\mathcal{K}}(s,a,k) = \sup_{\psi \in \Psi} Q^{\psi}(s,a,k).$$

7.4.2 All-Persistence Bellman Operator

Our goal is to leverage any $\overline{\kappa}$-persistence transition to learn $Q^{\star}_{\mathcal{K}}(\cdot,\cdot,k)$ for *all* the possible action-persistences in $k \in \mathcal{K}$. Suppose that $\overline{\kappa} \geq k$, then, we can exploit any sub-transition of k steps from the $\overline{\kappa}$-persistence transition to update the value $Q^{\star}_{\mathcal{K}}(\cdot,\cdot,k)$. Thus, we extend the Bellman optimal operator from Definition 2.13 to persistence options $T^{\star}$, defined over functions in $\mathcal{S} \times \mathcal{O}$:

Definition 7.4 (Bellman Optimality Operator for Persistence Options).
Let $\mathcal{M}$ be an MDP and $\mathcal{K}$ a persistence space. Consider a bounded, measurable function $f : \mathcal{S} \times \mathcal{A} \times \mathcal{K} \to \mathbb{R}$ and a state-option pair $(s, a, k) \in \mathcal{S} \times \mathcal{A} \times \mathcal{K}$. The Bellman optimality operator for state-option value functions *$T^{\star}$ is defined as:*

$$\left(T^{\star} f\right)(s,a,k) = r_k(s,a) + \gamma^k \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) \max_{(a',k') \in \mathcal{O}} f(s',a',k'). \tag{7.2}$$

If, instead, $\overline{\kappa} < k$, in order to update the value $Q^{\star}_{\mathcal{K}}(\cdot,\cdot,k)$, we partially exploit the $\overline{\kappa}$-persistent transition, but then, we need to *bootstrap* from a lower persistence Q-value, to compensate the remaining $k - \overline{\kappa}$ steps. To this end, we introduce the *bootstrapping* operator $T^{\overline{\kappa}}$:

Definition 7.5 (Bellman Bootstrap Operator on Persistence Options). *Let $\mathcal{M}$ be an MDP. Let $\mathcal{K}$ be a persistence space, with $\overline{\kappa} \in \mathcal{K}$. Consider a bounded, measurable function $f : \mathcal{S} \times \mathcal{A} \times \mathcal{K} \to \mathbb{R}$ and a state-option pair $(s, a, k) \in \mathcal{S} \times \mathcal{A} \times \mathcal{K}$, with $k > \overline{\kappa}$. The* Bellman Bootstrap operator for state-option value functions *$T^{\overline{\kappa}}$ is defined as:*

$$\left(T^{\overline{\kappa}} f\right)(s,a,k) = r_{\overline{\kappa}}(s,a) + \gamma^{\overline{\kappa}} \int_{\mathcal{S}} P_{\overline{\kappa}}(\mathrm{d}s'|s,a) f(s',a,k-\overline{\kappa}). \tag{7.3}$$

By combining these two operators, we obtain the *All-Persistence Bellman operator* $\mathcal{H}_{\overline{\kappa}}$ defined for every bound measurable functions $f : \mathcal{S} \times \mathcal{A} \times \mathcal{K} \to \mathbb{R}$ as:

Definition 7.6 (All-Persistence Bellman operator)**.** *Let $\mathcal{M}$ be an MDP. Let $\mathcal{K}$ be a persistence space, with $\overline{\kappa} \in \mathcal{K}$. Consider a bounded, measurable function $f : \mathcal{S} \times \mathcal{A} \times \mathcal{K} \to \mathbb{R}$ and a state-option pair $(s, a, k) \in \mathcal{S} \times \mathcal{A} \times \mathcal{K}$. The* All-Persistence Bellman operator for state-option value functions $T^{\overline{\kappa}}$ *is defined as:*

$$(\mathcal{H}^{\overline{\kappa}} f)(s, a, k) = \left((\mathbb{1}_{k \leq \overline{\kappa}} T^{\star} + \mathbb{1}_{k > \overline{\kappa}} T^{\overline{\kappa}}) f\right)(s, a, k), \tag{7.4}$$

where $T^{\star}$ and $T^{\overline{\kappa}}$ are the operators respectively defined in Definitions 7.4 and 7.5.

Thus, given a persistence $\overline{\kappa} \in \mathcal{K}$, $\mathcal{H}^{\overline{\kappa}}$ allows updating all the Q-values with $k \leq \overline{\kappa}$ by means of $T^{\star}$, and all the ones with $k > \overline{\kappa}$ by means of $T^{\overline{\kappa}}$. We want to demonstrate its soundness by means of contraction properties. First, we need the following useful Lemma:

Lemma 7.7 (Decomposition of r_k)**.** *Let $r_k(s, a)$ the expected k-persistent reward. Let $k' < k$, then it holds that:*

$$r_k(s, a) = r_{k'}(s, a) + \gamma^{k'} \int_{\mathcal{S}} P_{k'}(\mathrm{d}s'|s, a) r_{k-k'}(s', a) \qquad \forall (s, a) \in \mathcal{S} \times \mathcal{A}.$$

Proof. From the definition of r_k, it holds that $r_k(s, a) = \sum_{i=0}^{k-1} \gamma^i \left((p_\delta)^i r\right)(s, a)$, where p_δ is the Persistent Kernel defined in equation 6.2. Hence:

$$\begin{aligned}
r_k(s, a) &= \sum_{i=0}^{k-1} \gamma^i \left((p_\delta)^i r\right)(s, a) \\
&= \sum_{i=0}^{k'-1} \gamma^i \left((p_\delta)^i r\right)(s, a) + \sum_{i=k'}^{k-1} \gamma^i \left((p_\delta)^i r\right)(s, a) \\
&= r_{k'}(s, a) + \sum_{i=k'}^{k-1} \gamma^i \int_{\mathcal{S}} P_i(\mathrm{d}s'|s, a) r(s', a) \\
&= r_{k'}(s, a) + \sum_{i=k'}^{k-1} \gamma^i \int_{\mathcal{S}} P_{k'}(\mathrm{d}s''|s, a) \int_{\mathcal{S}} P_{i-k'}(\mathrm{d}s'|s'', a) r(s', a) \\
&= r_{k'}(s, a) + \gamma^{k'} \int_{\mathcal{S}} P_{k'}(\mathrm{d}s''|s, a) \sum_{j=0}^{k-k'-1} \gamma^j \int_{\mathcal{S}} P_j(\mathrm{d}s'|s'', a) r(s', a) \\
&= r_{k'}(s, a) + \gamma^{k'} \int_{\mathcal{S}} P_{k'}(\mathrm{d}s''|s, a) r_{k-k'}(s'', a).
\end{aligned}$$

□

Thanks to the previous Lemma, we can provide the following result:

Theorem 7.1. *The all-persistence Bellman operator $\mathcal{H}^{\overline{\kappa}}$ fulfills the following properties:*

1. *$\mathcal{H}^{\overline{\kappa}}$ is a γ-contraction in L_∞ norm;*

2. *$Q^{\star}_{\mathcal{K}}$ is its unique fixed point;*

3. *$Q^\star_{\mathcal{K}}$ is monotonic in k, i.e., for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ if $k \leq k'$ then*

$$Q^\star_{\mathcal{K}}(s,a,k) \geq Q^\star_{\mathcal{K}}(s,a,k').$$

Proof. (i) First, we prove the contraction property: we consider the L_∞-norm applied to the state-action-persistence space $\mathcal{S} \times \mathcal{A} \times \mathcal{K}$:

$$\begin{aligned}
\|\mathcal{H}^{\overline{\kappa}} Q_1 - \mathcal{H}^{\overline{\kappa}} Q_2\|_\infty &= \sup_{s,a,k \in \mathcal{S}\times\mathcal{A}\times\mathcal{K}} \left|\mathcal{H}^{\overline{\kappa}} Q_1(s,a,k) - \mathcal{H}^{\overline{\kappa}} Q_2(s,a,k)\right| \\
&= \sup_{s,a,k} \Big| \mathbb{1}_{k\leq\overline{\kappa}} \left(\left(T^\star Q_1\right)(s,a,k) - \left(T^\star Q_2\right)(s,a,k) \right) \\
&\qquad\qquad + \mathbb{1}_{k\leq\overline{\kappa}} \left(\left(T^{\overline{\kappa}} Q_1\right)(s,a,k) - \left(T^{\overline{\kappa}} Q_2\right)(s,a,k) \right) \Big| \\
&= \sup_{s,a,k} \Big| \gamma^k \mathbb{1}_{k\leq\overline{\kappa}} \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) \left[\sup_{a',k'} Q_1(s',a',k') - \sup_{a',k'} Q_2(s',a',k') \right] \\
&\qquad + \gamma^{\overline{\kappa}} \mathbb{1}_{k>\overline{\kappa}} \int_{\mathcal{S}} P_{\overline{\kappa}}(\mathrm{d}s'|s,a) \left[Q_1(s',a,k-\overline{\kappa}) - Q_2(s',a,k-\overline{\kappa}) \right] \Big| \\
&\leq \sup_{s,a,k} \Big\{ \gamma^k \mathbb{1}_{k\leq\overline{\kappa}} \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) \left| \sup_{a',k'} Q_1(s',a',k') - \sup_{a',k'} Q_2(s',a',k') \right| \\
&\qquad + \gamma^{\overline{\kappa}} \mathbb{1}_{k>\overline{\kappa}} \int_{\mathcal{S}} P_{\overline{\kappa}}(\mathrm{d}s'|s,a) \left| Q_1(s',a,k-\overline{\kappa}) - Q_2(s',a,k-\overline{\kappa}) \right| \Big\} \\
&\leq \sup_{s,a,k} \Big\{ \gamma^k \mathbb{1}_{k\leq\overline{\kappa}} \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) \sup_{\widetilde{a},\widetilde{k}} \left| Q_1(s',\widetilde{a},\widetilde{k}) - Q_2(s',\widetilde{a},\widetilde{k}) \right| \\
&\qquad + \gamma^{\overline{\kappa}} \mathbb{1}_{k>\overline{\kappa}} \int_{\mathcal{S}} P_{\overline{\kappa}}(\mathrm{d}s'|s,a) \sup_{\widetilde{s},\widetilde{a}} \left| Q_1(\widetilde{s},\widetilde{a},k-\overline{\kappa}) - Q_2(\widetilde{s},\widetilde{a},k-\overline{\kappa}) \right| \Big\} \\
&\leq \sup_{s,a,k} \Big\{ \gamma^k \mathbb{1}_{k\leq\overline{\kappa}} \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) \sup_{\widetilde{s},\widetilde{a},\widetilde{k}} \left| Q_1(\widetilde{s},\widetilde{a},\widetilde{k}) - Q_2(\widetilde{s},\widetilde{a},\widetilde{k}) \right| \\
&\qquad + \gamma^{\overline{\kappa}} \mathbb{1}_{k>\overline{\kappa}} \int_{\mathcal{S}} P_{\overline{\kappa}}(\mathrm{d}s'|s,a) \sup_{\widetilde{s},\widetilde{a},\widetilde{k}} \left| Q_1(\widetilde{s},\widetilde{a},\widetilde{k}) - Q_2(\widetilde{s},\widetilde{a},\widetilde{k}) \right| \Big\} \\
&\leq \sup_{s,a,k} \left\{ \left(\gamma^k \mathbb{1}_{k\leq\overline{\kappa}} + \gamma^{\overline{\kappa}} \mathbb{1}_{k>\overline{\kappa}} \right) \|Q_1 - Q_2\|_\infty \right\} \\
&= \|Q_1 - Q_2\|_\infty \sup_{k\in\mathcal{K}} \gamma^{\min\{k,\overline{\kappa}\}} = \gamma \|Q_1 - Q_2\|_\infty.
\end{aligned}$$

(ii): Since the contraction property holds, and being $(\mathcal{S} \times \mathcal{A} \times \mathcal{K}, d_\infty)$ a complete metric space (with d_∞ being the distance induced by L_∞ norm), the Banach Fixed-point theorem holds, guaranteeing convergence to a unique fixed point.

We now show that $Q^\star_{\mathcal{K}}$ is a fixed point of $T^\star$. We first need to define the extended Bellman expectation operators in Definition 2.7 to T^ψ with $f : \mathcal{S} \times \mathcal{A} \times \mathcal{K} \to \mathbb{R}$ being a bounded and measurable and $\psi \in \Psi$:[2]

$$
\begin{aligned}
(T^{\psi} f)(s,a,k) &= r_k(s,a) + \gamma^k \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) V(s'),\\
V(s) &= \sum_{(a,k)\in\mathcal{O}} \psi(a,k|s) f(s,a,k)
\end{aligned}
$$

As with standard Bellman operators, it trivially holds that $T^{\psi} Q^{\psi} = Q^{\psi}\ \forall \psi \in \Psi$. Thus, we can take into account the definition of value function V^{ψ} of a policy ψ and the standard Bellman Equations:

$$
\begin{aligned}
Q^{\psi}(s,a,k) &= r_k(s,a) + \gamma^k \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) V^{\psi}(s')\\
V^{\psi}(s) &= \sum_{(a,k)\in\mathcal{O}} \psi(a,k|s) Q^{\psi}(s,a,k)
\end{aligned}
\tag{7.5}
$$

Following the same argument as in Puterman (2014), it holds that the optimal operator $T^{\star}$ improves the action-value function, i.e. $T^{\star} Q^{\psi} \geq Q^{\psi}$, and consequently $Q^{\star}_{\mathcal{K}}$ is the (unique) fixed point for $T^{\star}$, i.e., $T^{\star} Q^{\star}_{\mathcal{K}} = Q^{\star}$ by contraction mapping theorem.

Moreover, it holds that $T^{\overline{\kappa}} Q^{\psi} = Q^{\psi}$:

$$
\begin{aligned}
&(T^{\overline{\kappa}} Q^{\psi})(s,a,k) = r_{\overline{\kappa}}(s,a) + \gamma^{\overline{\kappa}} \int_{\mathcal{S}} P_{\overline{\kappa}}(\mathrm{d}s'|s,a) Q^{\psi}(s',a,k-\overline{\kappa})\\
&= r_{\overline{\kappa}}(s,a) + \gamma^{\overline{\kappa}} \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) \left[r_{k-\overline{\kappa}}(s',a) + \gamma^{k-\overline{\kappa}} \int_{\mathcal{S}} P_{k-\overline{\kappa}}(\mathrm{d}s''|s',a) V^{\psi}(s') \right]\\
&= \underbrace{r_{\overline{\kappa}}(s,a) + \gamma^{\overline{\kappa}} \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) r_{k-\overline{\kappa}}(s',a)}_{r_k(s,a)} + \gamma^k \int_{\mathcal{S}} P_{\overline{\kappa}}(\mathrm{d}s'|s,a) \int_{\mathcal{S}} P_{k-\overline{\kappa}}(\mathrm{d}s''|s',a) V^{\psi}(s'')
\end{aligned}
\tag{7.6}
$$

$$
= r_k(s,a) + \gamma^k \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) V^{\psi}(s') = Q^{\psi}(s,a,k),
$$

where in Equation (7.6) we used Lemma 7.7.

In conclusion,

$$
\begin{aligned}
\mathcal{H}^{\overline{\kappa}} Q^{\star}_{\mathcal{K}} &= \left(\mathbb{1}_{k\leq\overline{\kappa}} T^{\star} + \mathbb{1}_{k>\overline{\kappa}} T^{\overline{\kappa}} \right) Q^{\star}_{\mathcal{K}}\\
&= \mathbb{1}_{k\leq\overline{\kappa}} T^{\star} Q^{\star}_{\mathcal{K}} + \mathbb{1}_{k>\overline{\kappa}} T^{\overline{\kappa}} Q^{\star}_{\mathcal{K}}\\
&= \mathbb{1}_{k\leq\overline{\kappa}} Q^{\star}_{\mathcal{K}} + \mathbb{1}_{k>\overline{\kappa}} Q^{\star}_{\mathcal{K}} = Q^{\star}_{\mathcal{K}}.
\end{aligned}
$$

(iii) We provide the proof of monotonic property: given $(s,a) \in \mathcal{S} \times \mathcal{A}$, and given

[2]The following can be extended without loss of generality to a continuous action space.

$k \le k'$, we have:

$$\begin{aligned} Q^\star_{\mathcal{K}}(s,a,k) &= (T^\star Q^\star_{\mathcal{K}})(s,a,k) \\ &= r_k(s,a) + \gamma^k \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) \max_{\overline{a},\overline{k} \in \mathcal{A}\times\mathcal{K}} Q^\star_{\mathcal{K}}(s',\overline{a},\overline{k}) \\ &\ge r_k(s,a) + \gamma^k \int_{\mathcal{S}} P_k(\mathrm{d}s'|s,a) Q^\star_{\mathcal{K}}(s',a,k'-k) \\ &= T^k Q^\star_{\mathcal{K}}(s,a,k') = Q^\star_{\mathcal{K}}(s,a,k'). \end{aligned}$$

□

Thus, operator $\mathcal{H}^{\overline{\kappa}}$ contracts to the optimal action-value function $Q^\star_{\mathcal{K}}$, which, thanks to monotonicity, has its highest value at the lowest possible persistence. In particular, we report the following corollary:

Corollary 7.1 (Equivalence of $Q^\star_{\mathcal{K}}$ and $Q^\star$)**.** *For all $(s,a) \in \mathcal{S}\times\mathcal{A}$, the optimal action-value function $Q^\star$ and the optimal option-value function restricted to the primitive actions coincide, i.e.*

$$Q^\star_{\mathcal{K}}(s,a,1) = Q^\star(s,a)$$

Proof. Trivially, $Q^\star_{\mathcal{K}}$, defined on Ψ, coincides with the classic $Q^\star$ defined on the *primitive* policies $\pi \in \Pi$: in a first instance, we remark that $\Pi \subset \Psi$, hence all the policies defined on the space of primitive actions belong to the set of persistent policies. Furthermore, we can consider property (iii) of Theorem 7.1. As a consequence $Q^\star_{\mathcal{K}}(s,a,1) \ge Q^\star(s,a,k) \forall k \in \mathcal{K}$, i.e., for each state $s \in \mathcal{S}$ there is at least one optimal primitive action $a \in \mathcal{A}$ which is optimal among all the option set $\mathcal{O}$. Consequently, the two optimal action-value functions coincide. □

Corollary 7.1 shows that, by fixing the persistence to $k = 1$, we retrieve the optimal Q-function in the original MDP, and consequently, we can reconstruct a greedy optimal policy. This highlights that the primitive action space leads to the same optimal Q-function as with persistence options. Persistence, nevertheless, is helpful for exploration and learning, but for an optimal persistent policy $\psi^\star$, there exists a primitive policy $\pi^\star$ with the same performance.

7.5 Persistent Q-learning

The advantages of $\mathcal{H}^{\overline{\kappa}}$ over traditional updates may not be immediately clear. These become apparent with its empirical counterpart $\widehat{\mathcal{H}}^{\overline{\kappa}}_t = \mathbb{1}_{k\le\overline{\kappa}} \widehat{T}^\star_t + \mathbb{1}_{k>\overline{\kappa}} \widehat{T}^{\overline{\kappa}}_t$, where:

$$\begin{aligned} \left(\widehat{T}^\star_t Q\right)(s_t,a_t,k) &= r^k_{t+1} + \gamma^k \max_{(a',k')\in\mathcal{O}} Q(s_{t+k},a',k'), \\ \left(\widehat{T}^{\overline{\kappa}}_t Q\right)(s_t,a_t,k) &= r^{\overline{\kappa}}_{t+1} + \gamma^{\overline{\kappa}} Q(s_{t+k},a_t,k-\overline{\kappa}). \end{aligned}$$

These empirical operators depend on the current *partial history*, which we define as: $H^{\overline{\kappa}}_t := (s_t, a_t, r_{t+1}, s_{t+1}, r_{t+2}, \dots, s_{t+\overline{\kappa}})$, used by Algorithm 9 to update each persistence in a backward fashion. At timestep t, given a sampling persistence $\overline{\kappa}_t$, for all subtransitions of $H^{\overline{\kappa}}_t$, starting at $t+i$ and ending in $t+j$, we apply $\widehat{\mathcal{H}}^{j-i}_t$ to $Q(s_{t+i}, a_t, k+d)$,

Algorithm 9 All Persistence Bellman Update

```
Require: Sampling persistence $\overline{\kappa}_t$, partial history $H_t^{\overline{\kappa}_t}$, Q-function $Q$, stepsize $\alpha$.
Ensure: Updated $Q$-function $Q$
1: for $j{=}\overline{\kappa}_t,\overline{\kappa}_t{-}1...,1$ do
2:     for $i{=}j{-}1,j{-}2,...,0$ do
3:         $k{\leftarrow}j{-}i$
4:         $Q(s_{t+i},a_t,k){\leftarrow}(1{-}\alpha)Q(s_{t+i},a_t,k){+}\alpha\widehat{T}^{\star}_{t+i}Q(s_{t+i},a_t,k)$
5:         for $d{=}1,2,...,K_{\max}{-}k$ do
6:             $Q(s_{t+i},a_t,k{+}d){\leftarrow}(1{-}\alpha)Q(s_{t+i},a_t,k{+}d){+}\alpha\widehat{T}^{k}_{t+i}Q(s_{t+i},a_t,k{+}d)$
7:         end for
8:     end for
9: end for
```

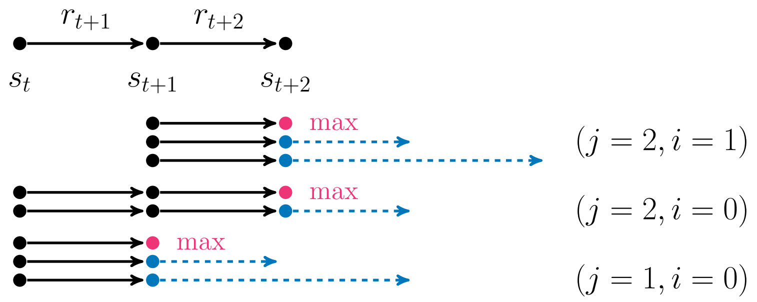


Figure 7.1: *An example of the update order for Algorithm 9 with $\overline{\kappa}_t = 2$ and $K_{\max} = 3$. Applications of $\widehat{T}_t^{\star}$ and $\widehat{T}_t^{\overline{\kappa}_t}$ are denoted, respectively, by magenta and blue nodes, while dashed arrows represent the bootstrap persistence.*

for all $d \leq K_{\max} - k$, where $k = j - i$. In Figure 7.1 we present a scheme that shows an example of the update order for Algorithm 10.

With these tools, it is possible to extend Q-learning (Watkins, 1989) to obtain the Persistent Q-learning algorithm (abbreviated as Per$Q$-learning), described in Algorithm 10. The agent follows a policy ψ_Q^{ϵ}, which is ϵ-greedy w.r.t. the option space and the current Q-function.

This approach extends the MSA-Q-learning (Schoknecht and Riedmiller, 2003), by bootstrapping higher persistence action values from lower ones. More precisely, both methods apply the update related to $\widehat{T}^{\star}$, but MSA-Q-learning does not use $\widehat{T}^{\kappa}$ instead. As shown in the empirical analysis, in some domains this difference can be crucial to speed up the convergence. Similarly to MSA-Q-learning, we perform backward updates to allow for an even faster propagation of values. The proposed approach also differs from TempoRL Q-learning (Biedenkapp et al., 2021), where action-persistence is selected using a dedicated value function, learned separately from the Q-function. The asymptotic convergence of Persistent Q-learning to $Q_K^{\star}$ directly follows Singh et al. (2000), being $\mathcal{H}^{\overline{\kappa}}$ a contraction and since their (mild) assumptions are satisfied.

7.6 Empirical Advantages of Persistence

In this section, we provide some numerical simulations to highlight the benefits of our approach. The settings are illustrative, to ease the detection of the individual advantages

Algorithm 10 Persistent Q-learning (Per$Q$-learning)

```
Require: Learning rate $\alpha$, exploration coefficient $\epsilon$, number of episodes $N$
Ensure: $Q$-function estimation
 1: Initialize $Q$ arbitrarily, except $Q(terminal, \cdot, \cdot) = 0$
 2: for $episode = 1, \dots, N$ do
 3:     $t \leftarrow 0$
 4:     while $s_t$ is not $terminal$ do
 5:         $a_t, \overline{\kappa}_t \sim \psi_Q^{\epsilon}(s_t)$
 6:         for $\tau = 1, \dots, \overline{\kappa}_t$ do
 7:             Take action $a_t$, observe $s_{t+\tau}, r_{t+\tau}$
 8:         end for
 9:         Store partial history $H_t^{\overline{\kappa}_t}$
10:         Update $Q$ according to Alg. 9
11:         $t \leftarrow t + \overline{\kappa}_t$
12:     end while
13: end for
```

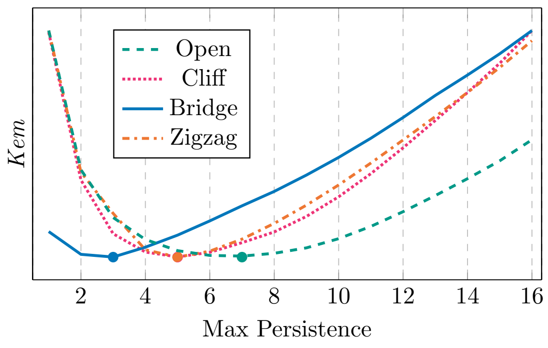


Figure 7.2: *Normalized Kemeny's constant in tabular environments as function of $K_{\max}$. Bullets represent the minimum.*

of persistence, before presenting more complex applications.

Exploration One of the main advantages of persistence is related to faster exploration, especially in goal-based environments (e.g., robotics and locomotion tasks). Indeed, persisting an action allows for reaching faster states far from the starting point and, consequently, propagating faster the reward. The reason is due to the increased chances of 1-persistent policies of getting stuck in specific regions. As explained in Amin et al. (2021), persistence helps to achieve *self-avoiding* trajectories, by increasing the expected return time in previously visited states. Hence, we study the effects of a persisted exploratory policy on the MDP, i.e., a policy $\psi \in \Psi$ over persistence options $\mathcal{O}$ (details in Appendix C.1.1).

To this purpose, we compute the *Kemeny's constant* (Catral et al., 2010; Patel et al., 2015), which corresponds to the expected first passage time from an arbitrary starting state s to another one s' under the stationary distribution induced by ψ. We consider four discrete tabular environments: *Open* is a 10x10 grid with no obstacles, while the others, presented in Biedenkapp et al. (2021), are depicted in Figure C.6. In Figure 7.2, we plot the variations of Kemeny's constant as a function of the maximum persistence $K_{\max}$, while following a uniform policy ψ over $\mathcal{O}$. We observe that increasing $K_{\max}$ promotes exploration and highlights the different $K_{\max}$ attaining the minimum value of

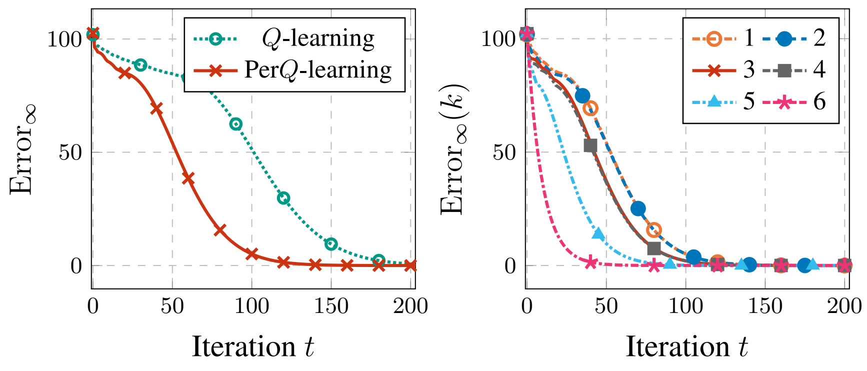


Figure 7.3: *L_∞ error on 6x6 grid-world between synchronous Q-learning and PerQ-learning (left) and for different persistence options $k \in \{1, ..., 6\}$ of PerQ-learning (right). (100 runs, avg ± 95 % c.i.)*

the constant, due to the different complexity of the environments.

Sample Complexity The second relevant effect of persistence concerns sample complexity. The intuition behind persistence relies on the fact that the most relevant information propagates faster through the state-action space, thanks to multi-step updates. Moreover, these updates are associated with a lower discount factor, for which it is possible to obtain better convergence rates, as seen in Chapter 6, in which the sample complexity in a $k-$persistent MDP is reduced by a factor $(1-\gamma^k)/(1-\gamma) > 1$. In order to evaluate the sample efficiency of Per$Q$-learning, separately from its effects on exploration, we considered a *synchronous* setting (Kearns and Singh, 1999; Sidford et al., 2018) in a deterministic 6x6 Gridworld. At each iteration t, the agent has access to a set of independent samples for each state-action pair. In standard Q-learning, for each $(s, a) \in \mathcal{S} \times \mathcal{A}$, $Q(s, a)$ is updated. In Per$Q$-learning, the samples are combined to obtain each possible set of $\overline{\kappa}$-persistent transitions, i.e., the tuples related to each possible $(s, a, k) \in \mathcal{S} \times \mathcal{O}$, with $K_{\max} = 6$; finally, the persistent Q function is updated.

In Figure 7.3 left, we compare the L_∞ error of Q-learning estimating $Q^\star(s, a)$, i.e., $\max_{s,a \in \mathcal{S}\times\mathcal{A}} |Q_t(s, a) - Q^\star(s, a)|$, and that of Per$Q$-learning estimating $Q^\star_{\mathcal{K}}(s, a, k)$, i.e., $\max_{s,a,k \in \mathcal{S}\times\mathcal{O}} |Q_t(s, a, k) - Q^\star_{\mathcal{K}}(s, a, k)|$, as a function of the number of iterations t. We observe that, although estimating a higher-dimensional function (as $Q^\star_{\mathcal{K}}(s, a, k)$ is a function of the persistence k too), Per$Q$-learning converges faster than Q-learning. In Figure 7.3 right, we plot the L_∞ error experienced by Per$Q$-learning for the different persistence options $\mathcal{O}^{(k)}$, i.e., Error$_\infty(k) := \max_{s,a \in \mathcal{S}\times\mathcal{A}} |Q_t(s, a, k) - Q^\star(s, a, k)|$ for $k \in \mathcal{K}$. As expected, higher values of k lead to faster convergence; consequently, the persistent Bellman operator helps improve the estimations also for the lower option sets. Indeed, we can see that also $Q_t(\cdot, \cdot, 1)$, the primitive actions Q-function, converges faster than classic Q-learning (details in Appendix C.1.3).

7.7 Persistent Deep Networks

In this section, we develop the extension of Per$Q$-learning to high-dimensional settings. Deep RL methods are becoming of fundamental importance when learning on real systems, as well as the research of methods to improve exploration and learning speed. It is straightforward to exploit DQN, introduced in Chapter 3, for learning

Algorithm 11 Multiple Replay Buffer Storing

```
Require: Maximum persistence $K_{\max}$, replay buffers $(\mathcal{D}_k)_{k=1}^{K_{\max}}$, transition tuple $(s_t, a_t, \overline{\kappa}_t, H_t^{\overline{\kappa}_t})$.
1: for $k = 1, \dots, K_{\max}$ do
2:     for $\tau = 0, \dots, \max\{\overline{\kappa}_t - k, 0\}$ do
3:         $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup (s_{t+\tau}, a_t, s_{t+\tau+k}, r_{t+1+\tau}^k, k)$
4:     end for
5:     for $\tau = 1, \dots, \min\{\overline{\kappa}_t, k-1\}$ do
6:         $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup (s_{t+\overline{\kappa}_t-\tau}, a_t, s_{t+\overline{\kappa}_t}, r_{t+1+\overline{\kappa}_t-\tau}^\tau, \tau)$
7:     end for
8: end for
```

in the options space $\mathcal{O}$. Standard DQN is augmented with $K_{\max}$ distinct sets of action outputs, to represent Q-value of the options space $\mathcal{O} = \mathcal{A} \times \mathcal{K}$, while the first layers are shared, similarly to previous works (Arulkumaran et al., 2016; Lakshminarayanan et al., 2017; Biedenkapp et al., 2021). The resulting algorithm, *Persistent Deep Q-Network* (PerDQN) is obtained by exploiting the application of the empirical all-persistence Bellman operator. The main differences between PerDQN and standard DQN are two:

1. A modified ϵ-greedy strategy, which is equivalent to the one described for its tabular version, in the same way as in (Dabney et al., 2020). Moreover, as in the common DQN implementation, the ϵ probability is reduced over time to ensure convergence.

2. The use of *multiple* replay buffers accounting for persistence.

Persistence Replay Buffers Whenever an option $o_t = (a_t, \overline{\kappa}_t)$ is executed, the partial history $H_t^{\overline{\kappa}_t}$ is decomposed in all its sub-transitions, which are used to update Q-values at any persistence, as shown in Section 7.5. The sub-transitions are stored in multiple replay buffers $\mathcal{D}_k$, one for each persistence $k \in \mathcal{K}$. Specifically, $\mathcal{D}_k$ stores tuples in the form $(s, a_t, s', r, \overline{\kappa})$, as summarized in Algorithm 11, where s and s' are the first and the last state of the sub-transition, r is the $\overline{\kappa}$-persistent reward, and $\overline{\kappa}$ is the true length of the sub-transition, which will then be used to suitably apply $\widehat{\mathcal{H}}_t^{\overline{\kappa}}$.

Finally, the gradient update is computed by sampling a mini-batch of experience tuples from each replay buffer $\mathcal{D}_k$, in equal proportion. Given the current network and target parametrizations $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^-$, the temporal difference error of a sample $(s, a, r, s', \overline{\kappa})$ is computed as $\widehat{\mathcal{H}}^{\overline{\kappa}} Q_{\boldsymbol{\theta}^-}(s, a, k) - Q_{\boldsymbol{\theta}}(s, a, k)$. Our approach differs from TempoRL DQN (Biedenkapp et al., 2021), which uses a dedicated network to learn the persistence at each state and employs a standard replay buffer, ignoring the persistence at which samples have been collected. This affects the learning procedure, especially in the case of mini-batch sampling prioritizing tuples with higher temporal difference errors since, trivially, predictions related to longer transitions are more difficult and are usually related to higher errors.

7.8 Experimental Evaluation

In this section, we show the empirical analysis of our approach on both the tabular setting (Per$Q$-learning) and the function approximation one (PerDQN).

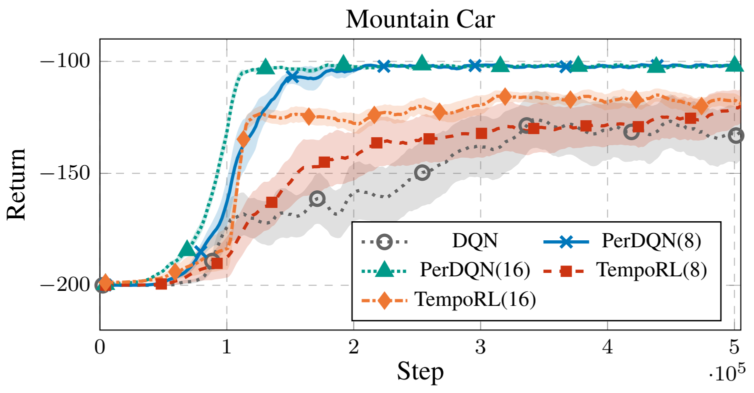


Figure 7.4: *MountainCar results. Parentheses in the legend denote $K_{\max}$. 20 runs (avg$\pm$ 95% c.i.).*

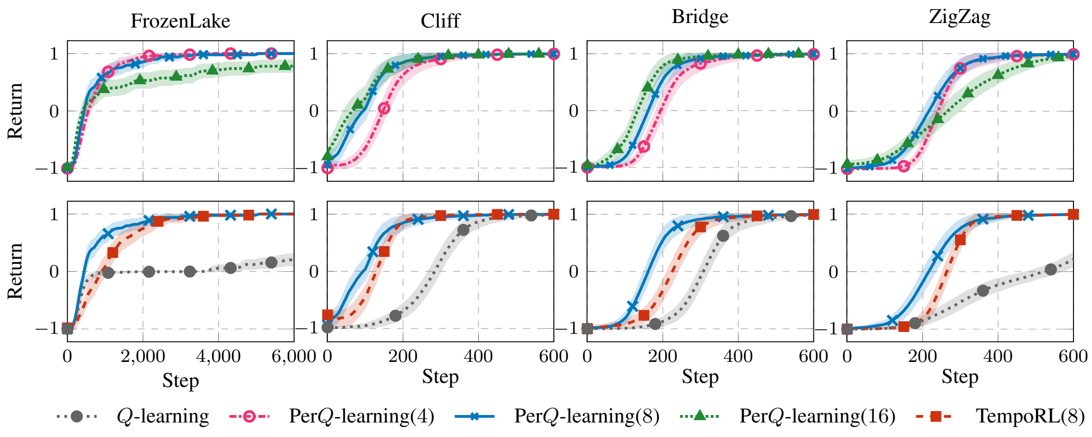


Figure 7.5: *Results on tabular environments. Top row: performances with different maximum persistences. In the legend, parentheses denote the selected $K_{\max}$. Bottom row: PerQ-learning and TempoRL comparison, $K_{\max} = 8$. 50 runs (avg$\pm$ 95% c.i.).*

Per$Q$-learning We present the results of the experiments in tabular environments, particularly suited for testing Per$Q$-learning because of the sparsity of rewards. We start with the deterministic 6x10 grid-worlds introduced by Biedenkapp et al. 2021. In these environments, the episode ends if either the goal or a hole is reached, with $+1$ or -1 points respectively. In all the other cases, the reward is 0, and the episode continues (details in Appendix C.2.1). Moreover, we experiment with 16x16 FrozenLake, from OpenAI Gym benchmark (Brockman et al., 2016), with rewards and transition processes analogous to the previous case, but with randomly generated holes at the beginning of the episode. The results are shown in Figure 7.5. In the top row, we compare the results on the performance when applying Per$Q$-learning with different $K_{\max} \in \{4, 8, 16\}$. We can detect a faster convergence when passing from $K_{\max} = 4$ to 8. However, the largest value of $K_{\max}$ is not always the best one: while Bridge and Cliff show a slight improvement, performances in ZigZag and FrozenLake degrade. This is probably due to the nature of the environment: when there are many obstacles, high persistences might be inefficient, as the agent can get stuck or reach holes more easily. In the bottom plots of Figure 7.5 we selected the results with $K_{\max} = 8$ and compared them with TempoRL (with the same maximum *skip-length* $J = 8$) and clas-

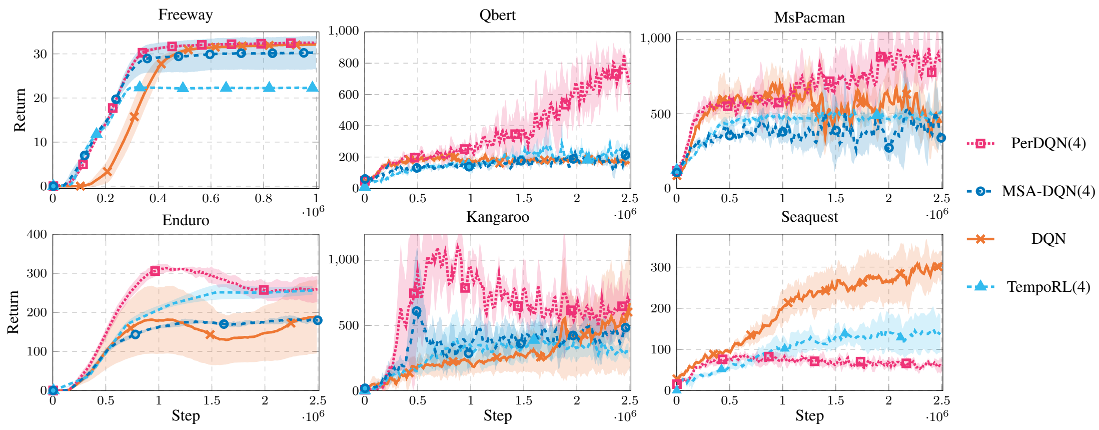


Figure 7.6: *Atari games results. Parentheses in the legend denote the maximum persistence $K_{\max}$. 5 runs (avg± 95% c.i.).*

sic Q-learning. In all cases, Per$Q$-learning outperforms the other methods, especially Q-learning, whose convergence is significantly slower. exploration is very small. With this maximum persistence value Per$Q$-learning outperforms TempoRL. Further experiments with different values of $K_{\max}$ have been reported in Appendix C.3.1. In general, Per$Q$-learning shows faster rates of improvements than TempoRL, especially in the first learning iterations. However, this advantage may not be consistent for every value of $K_{\max}$, and every environment, as also shown in Appendix C.3.1.

PerDQN Our implementation of PerDQN is based on OpenAI Gym (Brockman et al., 2016) and Baselines (Dhariwal et al., 2017) Python toolkits. We start with Mountain-Car (Moore, 1991), as it is perhaps the most suited to evaluate the performance of persistence options. As shown in Table B.1, 1-step explorative policies usually fail to reach the goal. Figure 7.4 shows that TempoRL and DQN cannot converge to the optimal policy, as already noticed in Biedenkapp et al. (2021), while PerDQN attains the optimal solution, that reaches the top of the mountain with the minimum loss.

The algorithm is then tested in the challenging framework of Atari 2600 games, where we want to validate that action persistence is beneficial to speed up the initial phases of learning also in large environments.

The same architecture from Mnih et al. (2013), suitably modified as in Section 7.7, is used for all environments. For a fair comparison with TempoRL and standard DQN, persistence is implemented on top of the *frame skip*. Thus, a one-step transition corresponds to 4 frame skips. In Figure 7.6 we compare PerDQN with TempoRL and classic DQN. In five games out of six, our PerDQN displays a faster learning curve thanks to its ability of reusing experience, although in some cases (e.g. Kangaroo) PerDQN seems to inherit the same instability issues of DQN, we conjecture due to the overestimation bias (van Hasselt et al., 2016). In order to better understand which beneficial effects are provided by action persistence alone and which ones derive from the use of the bootstrap operator, we run an ablation experiment on the same tasks removing the latter one. The resulting algorithm is then similar to the Deep RL version of MSA-Q-learning (Schoknecht and Riedmiller, 2003), which we called MSA-DQN. The results show that PerDQN always dominates over its counterpart without bootstrap. The crucial importance of the bootstrap operator is confirmed also in the MountainCar setting where removing this feature causes a performance decrease, making its score comparable to TempoRL (see Appendix C.3.2). Finally, we notice that in Seaquest persistence seems to be detrimental to learning, as DQN outperforms PerDQN. In this task, agents have to choose either to move or to shoot some moving targets. Persisting the shooting action, thus, may force the agent to stay still for a long time, hitting nothing. A possible solution could consist in the introduction of *interrupting persistence*, in a similar fashion to interrupting options (Sutton et al., 1999b; Mankowitz et al., 2014), which is an interesting future research direction. Further discussions, analyses, and results are provided in Appendix C.3.

7.9 Conclusions

In this chapter, we considered RL policies that implement action persistence, modeled as *persistence options*, selecting a primitive action and its duration. We defined the *all-persistence* Bellman operator, which allows for effective use of the experience collected at any time scale, as action-value function estimates can be updated simultaneously on the whole persistence set. In particular, low persistences (and primitive actions) can be updated by splitting the samples into their sub-transitions; high persistences can instead be improved by *bootstrap*, i.e. by estimating the partial missing information. After proving that the new operator is a contraction, we extended classic Q-learning and DQN to their persistent version. The empirical analysis underlines the benefits of the new operator for exploration and estimation. Furthermore, the experimental campaign on tabular and deep RL settings demonstrated the effectiveness of our approach and the importance of considering temporally extended actions, as well as some limitations. There are still numerous open questions and future research directions, such as the possibility to introduce persistence interruption and techniques to overcome the overestimation bias. Furthermore, one could investigate the use of the operator in the actor-critic framework to cope with continuous actions. Some preliminary results in this direction can be found in Appendix D, where the persistence selection is entrusted to a *persistor*, i.e., a specifically designed secondary policy, with the possibility to *learn when to act* via stochastic gradient ascent.

CHAPTER 8

Discussion and Conclusions

The conclusion of this thesis is devoted to providing a summary of our contributions, with further discussions on the advantages and the limitations, and inspiring some ideas for future research directions.

In this dissertation, we addressed different approaches to enhance the learning capabilities of RL techniques providing both algorithmic and theoretical contributions. We are motivated by the challenges faced by practitioners in the RL field, where real-world applications need careful engineering of both the environment and the algorithm selected. We explained the basic concepts related to the interaction between the agent and the environment (Chapter 2) and presented the most important algorithmic solutions in Chapter 3. In particular, we have shown that the fundamental theoretical guarantees play a very important role in policy-based techniques, but they do not help in the selection of the hyperparameters. Consequently, in Part I we aimed to enhance the learning capabilities of this class of algorithms: we framed the HO problem as a Sequential Decision Process and designed a solution that allows selecting a dynamic sequence of hyperparameters, adaptive to the policy and the context of the MDP. On the other hand, environment configuration is paramount to make RL algorithms able to detect the most promising actions: in this sense, the duration of the actions is significant, as it defines the control opportunities and the sample complexity. Part II is therefore devoted to making further steps in the control frequency analysis and to fostering adaptive algorithms with the introduction of the persistence concept.

8.1 Hyperparameter Optimization through Meta RL

Part I of this thesis is focused on *Hyperparameter Optimization through Meta Reinforcement Learning*. This field addresses the challenges and the new opportunities that arise with a dynamic selection of hyperparameters, that can be adapted with respect to the task determining the dynamics and reward process in the environment. Indeed, in Chapter 4 we introduced the *Contextual Markov Decision Processes* (Hallak et al., 2015), where a variable context defines the properties of the environment in a classic Meta-Learning framework. In the first instance, we considered some assumptions related to the smoothness of the environment with respect to the task and obtained interesting bounds on the difference in terms of performances within two different tasks. This result explains the behavior of related Meta-Learning approaches (e.g., Finn et al. 2017), whose models empirically show bad generalization capabilities when training and test tasks are drawn from different distributions. Successively, we took into account the hyperparameter selection problem, and we framed it as a Sequential Decision Making problem. The independence of the return function to previous policy updates (when the current policy is accessible) is equivalent to the Markov assumption requested for standard RL approaches. Hence, the whole problem can be addressed as a Meta-Markov Decision Process, where the reward function is the performance gain. This concept has interesting connections with the *learning to learn* paradigm predominant in Meta-Learning literature, but differs from the common approach of the maximization of the immediate (meta) reward, in the same fashion as MDPs differ from bandits. We then applied the Meta-MDP concept to the selection of the step size for policy gradient approaches and included in the observations provided to the meta agent the explicit parametrization of the context, of the policy and its gradient, computed on a batch of trajectories. In this way, we could rely on FQI (Ernst et al., 2005), a batch value-based RL algorithm, to learn the expected performance gain for each policy-stepsize pair. The proposed method is then tested on simulated domains, showing an improvement concerning the most common learning rate schedules. Moreover, the stepsizes selected are adaptive to the selected task, and decrease throughout the learning instances without explicit knowledge of the number of steps, meaning that the model is capable of detecting the distance from (local) optima from the observations provided.

Further improvements are obtained in Chapter 5, where the Meta-MDP design is deeply modified to overcome the curse of dimensionality related to the inclusion of the whole policy parametrization in the observation space. Indeed, we leveraged Information Theory concepts to retrieve a set of informative features that can be approximated by employing KNN-based estimators (Singh et al., 2003). In this way, the resulting models can gather information based only on the trajectories generated by the interaction of the current policy with the environment: therefore, they are *context-agnostic* and *policy-agnostic*, in the sense that there is no explicit parametrization of the context or the policy. Therefore, they can potentially be applied to any learning algorithm with any hyperparameter, and are able to generalize across different policy architectures (in the case of policy-based methods). The resulting approach is then applied to the optimization of the trust-region constraint, the main hyperparameter introduced in one of the most common RL policy-based algorithms, TRPO (Schulman et al., 2015). All the experiments carried out suggest that the meta-MDP direction is promising, as the increased number of degrees of freedom obtained through a dynamic selection of the hyperparameter can help improve the learning capabilities across different policy architectures and unseen contexts, without the need of restarting hyperparameter tuning procedures from scratch for each new task.

8.1.1 Limitations and Future Works

In the first part of this dissertation, we have investigated the automatic tuning of one hyperparameter, i.e., the stepsize or the trust radius. However, in the majority of the algorithms, the dimensionality of the tuning procedure is way larger. As a simple example, the same TRPO algorithm approximates the value function to provide a critic for the selected actions, and depends on the learning rate for the value function updates, the number of conjugate gradient steps for the natural gradient approximation, the entropy coefficient to enhance exploration, just to mention a few. The meta-MDP formulation allows to generalize to any number of hyperparameters but, of course, the complexity of value-based algorithms increases accordingly. This is an important limitation, especially if we consider the large amount of data required to train our meta-FQI models (and to train meta-RL models, in general). In this direction, a possible solution can be detected with online approaches, similar to Xu et al. 2018: however, the high sensitivity of the current approaches to the new meta-hyperparameter risks making the problem vacuous, as the models might be even more difficult to train and requiring more data. Alongside these considerations, the set of informative meta-features we detected in Chapter 5 was reasonable, but it is not justified by any theoretical grounding: as future research directions, we may wonder how to generate and select more informative meta-features with a solid validation theory. Furthermore, the smoothness analysis we included was interesting but unrelated to the solution we proposed: as a future research direction, the proved bounds can be leveraged to detect more robust models with respect to the context provided. Finally, we focused our research on HO for RL, but we may analyze the effects of our contributions on other Machine Learning fields, such as Supervised Learning.

8.2 Dynamic Step and Control Frequency

Part II of the dissertation is devoted to enhancing the learning capabilities of RL by configuring the control frequency of a system. We have shown its importance, especially in robotic manipulation problems, although in literature particular attention is also devoted to the frame-skip parameters in image-based environments (Lakshminarayanan et al., 2017), with the same meaning. On one side, this parameter has a deep impact on the control opportunities; on the other, rapid action changes may prevent the agent to be able to detect (and reward) the most useful ones. Under the assumption of the existence of a *base frequency*, we introduced the concept of action persistence, i.e., the repetition of an action for a fixed amount of base steps, having the effect of altering the control frequency of the system. We have shown in Chapter 6 that persistence can be described as a context of the environment *with the definition of the* $k-$*persistence MDP, with appropriate transition processes and reward functions*, or as a non-Markovian policy working at the highest frequency. We derived a bound of the performance loss induced by persistence under some regularity conditions. In particular, we assumed standard Lipschitz continuity properties on the MDP, with the inclusion of an additional hypothesis on the speed at which the environment evolves over time (Time-Lipschitz Continuity). Based on these considerations, we provided an algorithmic contribution with

PFQI, an extension of FQI to take into account action persistence. Furthermore, we leveraged some error propagation bounds to retrieve a persistence selection heuristic. The experimental evaluation justifies the introduction of persistence, since increasing the action duration can lead to an improvement when dealing with a limited number of samples. Furthermore, the persistence selection heuristic often identified reliable persistence values. An important application of our approach was detected in the problem of financial trading, where persistence helped detect profitable policies and patterns in market behavior.

While Chapter 6 was devoted to comparing the performances when acting at different frequencies, in 7 we focused our attention to learn a dynamic persistence selection. We implemented the selection of the action duration through the mechanism of *persistence options*, giving the agent the ability to select both an action and its persistence. We devised a new operator, the *All-persistence Bellman operator*, which allows for effective use of the experience collected at any time scale. Hence, action value function estimates for the different possible persistences can be updated simultaneously: the value of acting with high frequencies can be learned by splitting the transitions into their components; instead, the expected return related to high persistence values can be updated by bootstrapping the partial missing information by resorting to Bootstrap operations. After proving the contraction properties of the new operator, we extended standard Q-learning and DQN to operate within the persistence option framework. A first experimental campaign was devoted to underlining the benefits of our approach, in terms of better exploration (since the average time to reach far states is reduced) and faster information propagation. The proposed algorithms were then tested on benchmark environments, including some Atari games, which highlighted the effectiveness of temporally extended actions as well as some limitations.

8.2.1 Limitations and Future Works

The experimental sessions performed in Chapter 7 showed that persistence is not always useful: indeed, it is always possible to design environments where the repetition of an action provides only bad effects. Furthermore, we work under the hypothesis of the existence of a base control frequency: our approach hence does not consider the divergence effects emerging in the limit for time discretization going to 0, as shown for example in Tallec et al. 2019. The provided algorithms do not scale well with this limit, since one should greatly increase the persistence rates to keep the exploration benefits: this significantly increases the computational requirements. A first idea to overcome this limitation can be detected by relying on *advantage updating*, similarly as in Bradtke and Duff 1994. Further research directions can be detected by recalling the similarity of our approach with the option framework (Precup, 2001): advances within this framework can be leveraged to extend the set of possible *macro-action* or to better exploit the advantages of the multi-step transitions. In conclusion, with our work, we focused on value-based approaches: an interesting research direction consists in designing a secondary policy that learns the action duration with a policy-gradient framework. Some preliminary results and discussions are provided in Appendix D.

8.3 Final Remarks

To conclude, we have highlighted the potential and the limitations of current RL algorithms in view of real-world applications, where the interaction between the agent, the environment, and the learning models must be carefully and continuously tuned. In this dissertation, we attempted to enhance learning in a twofold direction: improving the algorithm by optimizing the hyperparameters tuning procedure, or improving the model, by resorting to action persistence. We believe that our work makes some small steps towards the design of more efficient learning processes, and we offered some insights for appealing research directions.

Appendices

APPENDIX $\mathcal{A}$

Additional Results of Chapter 4

A.1 Proofs

A.1.1 Lipschitz Continuity of the Action-Value Function

Before describing the proof for Theorem 4.1, we need to recall the Bellman Expectation Operator T^{π} from equation 2.18 applied on the Action Value Function $Q^{\pi}_{\boldsymbol{\omega}}$ in the context $\boldsymbol{\omega} \in \Omega$:

$$\begin{aligned} T^{\pi}Q^{\pi}_{\boldsymbol{\omega}}(s,a) &= r_{\boldsymbol{\omega}}(s,a) + \gamma \int_{\mathcal{S}} P_{\boldsymbol{\omega}}(s'|s,a) \int_{\mathcal{A}} Q^{\pi}_{\boldsymbol{\omega}}(s',a')\pi(a'|s')dads' \\ &= r_{\boldsymbol{\omega}}(s,a) + \gamma \int_{\mathcal{S}} P_{\boldsymbol{\omega}}(s'|s,a)V^{\pi}_{\boldsymbol{\omega}}(s')ds' \end{aligned}$$

where $Q^{\pi}_{\boldsymbol{\omega}}$ is the fixed point. Moreover, let's consider as preliminary result the LC-continuity of the value functions (Lemma 3.12, Rachelson and Lagoudakis 2010) presented in section 3.6. We provide a generalization of this result on Lipschitz CMDPs.

Theorem 4.1. *Let $\mathscr{M}$ be a $(L_{\omega_P}, L_{\omega_r})$-CLC CMDP for which $\mathcal{M}(\boldsymbol{\omega})$ is $(L_P(\omega), L_R(\omega))$-LC $\forall\boldsymbol{\omega} \in \Omega$. Given a L_{π}-LC policy π, the action value function $Q^{\pi}_{\boldsymbol{\omega}}(s,a)$ is L_{ω_Q}-CLC w.r.t. the context $\boldsymbol{\omega}$, i.e.:*

$$\left|Q^{\pi}_{\boldsymbol{\omega}}(s,a) - Q^{\pi}_{\overline{\boldsymbol{\omega}}}(s,a)\right| \leq L_{\omega_Q}(\pi)d_{\Omega}(\boldsymbol{\omega},\overline{\boldsymbol{\omega}}) \;\; \forall(s,a) \in \mathcal{S}\times\mathcal{A},\; \forall\boldsymbol{\omega},\overline{\boldsymbol{\omega}} \in \Omega;$$

where

$$L_{\omega_Q}(\pi) = \frac{L_{\omega_r} + \gamma L_{\omega_p} L_{V_{\pi}}(\omega)}{1-\gamma}, \qquad L_{V_{\pi}}(\omega) = \frac{L_r(\omega)(1+L_{\pi})}{1-\gamma L_P(\omega)(1+L_{\pi})} \tag{4.1}$$

Proof. We follow the same ideas as in Rachelson and Lagoudakis (2010): first of all, given an $L_{\boldsymbol{\omega}_Q}$-LC continuous Q function $Q^\pi_{\boldsymbol{\omega}}$ w.r.t. the task space $\boldsymbol{\omega}$, the related value function $V^\pi_{\boldsymbol{\omega}}$ is $L_{\boldsymbol{\omega}_Q}$-LC. Indeed,

$$\begin{aligned}\left|V^\pi_{\boldsymbol{\omega}}(s) - V^\pi_{\overline{\boldsymbol{\omega}}}(s)\right| &= \left|\int_{\mathcal{A}} \pi(a|s)\left(Q^\pi_{\boldsymbol{\omega}}(s,a) - Q^\pi_{\overline{\boldsymbol{\omega}}}(s,a)\right) da\right| \\ &\leq \int_{\mathcal{A}} \pi(a|s)\left|Q^\pi_{\boldsymbol{\omega}}(s,a) - Q^\pi_{\overline{\boldsymbol{\omega}}}(s,a)\right| da \\ &\leq \max_a \left|Q^\pi_{\boldsymbol{\omega}}(s,a) - Q^\pi_{\overline{\boldsymbol{\omega}}}(s,a)\right| \leq L_{\boldsymbol{\omega}_Q} d_\Omega(\boldsymbol{\omega}, \overline{\boldsymbol{\omega}}).\end{aligned}$$

Now, we consider the iterative application of Bellman Operators, in such a way that $Q^{\pi,n+1}_{\boldsymbol{\omega}} = T^\pi Q^{\pi,n}_{\boldsymbol{\omega}}$, and we prove that $Q^{\pi,n}_{\boldsymbol{\omega}}$ is $L^n_{\boldsymbol{\omega}_Q}$-LC continuous, and that satisfies the recurrence relation:

$$L^{n+1}_{\boldsymbol{\omega}_Q} = L_{\boldsymbol{\omega}_r} + \gamma L_\pi L_V(\boldsymbol{\omega}) + \gamma L^n_{\boldsymbol{\omega}_Q}. \tag{A.1}$$

Indeed, for $n = 1$ the property holds immediately, since:

$$\left|Q^{\pi,1}_{\boldsymbol{\omega}}(s,a) - Q^{\pi,1}_{\overline{\boldsymbol{\omega}}}(s,a)\right| = \left|R_{\boldsymbol{\omega}}(s,a) - R_{\overline{\boldsymbol{\omega}}}(s,a)\right| \leq L_{\boldsymbol{\omega}_r} d_\Omega(\boldsymbol{\omega}, \overline{\boldsymbol{\omega}}).$$

Now, let us suppose the property holds for n. Then:

$$\begin{aligned}&\left|Q^{\pi,n+1}_{\boldsymbol{\omega}}(s,a) - Q^{\pi,n+1}_{\overline{\boldsymbol{\omega}}}(s,a)\right| = \\ &\left|R_{\boldsymbol{\omega}}(s,a) - R_{\overline{\boldsymbol{\omega}}}(s,a) + \gamma\int_{\mathcal{S}} P_{\boldsymbol{\omega}}(s'|s,a)\, V^{\pi,n}_{\boldsymbol{\omega}}(s')\, ds' - \gamma\int_{\mathcal{S}} P_{\overline{\boldsymbol{\omega}}}(s'|s,a)\, V^\pi_{\overline{\boldsymbol{\omega}}}(s')\, ds'\right| \\ &\leq L_{\boldsymbol{\omega}_r} d_\Omega(\boldsymbol{\omega}, \overline{\boldsymbol{\omega}}) + \gamma\left|\int_{\mathcal{S}} \left(P_{\boldsymbol{\omega}}(s'|s,a) - P_{\overline{\boldsymbol{\omega}}}(s'|s,a)\right) V^{\pi,n}_{\boldsymbol{\omega}}(s')\, ds'\right| \\ &\quad + \gamma\left|\int_{\mathcal{S}} P_{\overline{\boldsymbol{\omega}}}(s'|s,a)\left(V^{\pi,n}_{\boldsymbol{\omega}}(s') - V^{\pi,n}_{\overline{\boldsymbol{\omega}}}(s')\right) ds'\right| \\ &\leq L_{\boldsymbol{\omega}_r} d_\Omega(\boldsymbol{\omega}, \overline{\boldsymbol{\omega}}) + \gamma L_V(\boldsymbol{\omega}) \sup_{\|f\|_L \leq 1}\left\{\left|\int_{\mathcal{S}} \left(P_{\boldsymbol{\omega}}(s'|s,a) - P_{\overline{\boldsymbol{\omega}}}(s'|s,a)\right) f(s')\, ds'\right|\right\} \\ &\quad + \gamma \max_{s'}\left|V^{\pi,n}_{\boldsymbol{\omega}}(s') - V^{\pi,n}_{\overline{\boldsymbol{\omega}}}(s')\right| \\ &\leq \left(L_{\boldsymbol{\omega}_r} + \gamma L_{\boldsymbol{\omega}_P} L_V(\boldsymbol{\omega}) + \gamma L^n_{\boldsymbol{\omega}_Q}\right) d_\Omega(\boldsymbol{\omega}, \overline{\boldsymbol{\omega}}).\end{aligned}$$

Consequently, Inequality A.1 holds. Now, if the sequence $L^n_{\boldsymbol{\omega}_Q}$ is convergent, it converges to the fixed point of the recurrence equation:

$$L_{\boldsymbol{\omega}_Q} = L_{\boldsymbol{\omega}_r} + \gamma L_{\boldsymbol{\omega}_P} L_V(\boldsymbol{\omega}) + \gamma L_{\boldsymbol{\omega}_Q}.$$

Hence the limit point is the one expressed in Equation 4.1, and the sequence can be proven to be convergent since $\gamma < 1$. □

As a consequence, the Proof that $J_{\boldsymbol{\omega}}(\pi)$ is CLC under $\boldsymbol{\omega}$ is immediate:

$$\begin{aligned}\left|J_{\boldsymbol{\omega}}(\pi) - J_{\overline{\boldsymbol{\omega}}}(\pi)\right| &= \left|\int_{\mathcal{S}} \mu\left(s_0\right)\left[V^{\pi}_{\boldsymbol{\omega}}\left(s_0\right) - V^{\pi}_{\overline{\boldsymbol{\omega}}}\left(s_0\right)\right] ds_0\right| \\ &\leq \int_{\mathcal{S}\times\mathcal{A}} \mu\left(s_0\right)\pi\left(a|s_0\right)\left|Q^{\pi}_{\boldsymbol{\omega}}\left(s_0,a\right) - Q^{\pi}_{\overline{\boldsymbol{\omega}}}\left(s_0,a\right)\right| da ds_0 \\ &\leq L_{\boldsymbol{\omega}_Q}(\pi) d_{\Omega}(\boldsymbol{\omega},\overline{\boldsymbol{\omega}}).\end{aligned}$$

A.1.2 Lipschitz Continuity of the Gradient

In order to consider the Lipschitz continuity of the gradient of the return $\nabla J_{\boldsymbol{\omega}}(\boldsymbol{\theta})$, we first need to introduce two more assumptions:

Assumption A.1 (Lipschitz Parametric Policy)**.** *Let $\pi_{\boldsymbol{\theta}} \in \Pi$ be a policy parametrized in the parameters space $\boldsymbol{\theta} \in \Theta$. An LC-policy π satisfies the following conditions:*

$$\forall \boldsymbol{\theta} \in \Theta, \forall s, \overline{s} \in \mathcal{S} \quad \mathcal{W}_1\left(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_{\boldsymbol{\theta}}\left(\cdot|\overline{s}\right)\right) \leq L_{\pi_{\boldsymbol{\theta}}} d_{\mathcal{S}}\left(s,\overline{s}\right) \tag{A.2}$$

$$\forall s \in \mathcal{S}, \forall \boldsymbol{\theta}, \overline{\boldsymbol{\theta}} \in \Theta \quad \mathcal{W}_1\left(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_{\overline{\boldsymbol{\theta}}}(\cdot|s)\right) \leq L_{\pi}(\boldsymbol{\theta}) d_{\Theta}\left(\boldsymbol{\theta},\overline{\boldsymbol{\theta}}\right). \tag{A.3}$$

Assumption A.2 (Lipschitz Gradient of Policy Logarithm)**.** *The gradient of the policy logarithm must satisfy the following conditions:*

1. *Uniformly bounded gradient:* $\forall (s,a) \in \mathcal{S}\times\mathcal{A}, \forall \boldsymbol{\theta} \in \Theta, \forall i = 1,\ldots,d$
$$\left|\nabla_{\boldsymbol{\theta}_i} \log \pi_{\boldsymbol{\theta}}(a|s)\right| \leq M^i_{\boldsymbol{\theta}};$$
2. *State-action LC:* $\forall\left(s,\overline{s},a,\overline{a}\right) \in \mathcal{S}^2\times\mathcal{A}^2, \forall \boldsymbol{\theta} \in \Theta, \forall i = 1\ldots,d$
$$\left|\nabla_{\boldsymbol{\theta}_i}\log\pi_{\boldsymbol{\theta}}(a|s) - \nabla_{\boldsymbol{\theta}_i}\log\pi_{\boldsymbol{\theta}}\left(\overline{a}|\overline{s}\right)\right| \leq L^i_{\nabla\log\pi} d_{\mathcal{S}\times\mathcal{A}}\left((s,a),\left(\overline{s},\overline{a}\right)\right);$$
3. *Parametric LC:* $\forall\left(\boldsymbol{\theta},\overline{\boldsymbol{\theta}}\right) \in \Theta, \forall (s,a) \in \mathcal{S}\times\mathcal{A}, \forall i = 1,\ldots,d$
$$\left|\nabla_{\boldsymbol{\theta}_i}\log\pi_{\boldsymbol{\theta}}(a|s) - \nabla_{\boldsymbol{\theta}_i}\log\pi_{\overline{\boldsymbol{\theta}}}(a|s)\right| \leq L^i_{\nabla\log\pi}(\boldsymbol{\theta}) d_{\Theta}\left(\boldsymbol{\theta},\overline{\boldsymbol{\theta}}\right).$$

Lemma A.1 (Lemma 3 from Pirotta et al. (2015))**.** *Given Assumptions 3.1, 3.3, if $\gamma L_P(1+ L_{\pi_{\boldsymbol{\theta}}}) < 1$, the Kantorovich distance between a pair of γ-discounted feature state distributions is Parametric-LC (PLC) w.r.t. parameters $\Theta : \forall(\boldsymbol{\theta},\widehat{\boldsymbol{\theta}}) \in \Theta^2$:*

$$\mathcal{W}_1\left(\delta^{\mu}_{\boldsymbol{\theta}}, \delta^{\mu}_{\overline{\boldsymbol{\theta}}}\right) \leq L_{\delta}(\boldsymbol{\theta}) d_{\Theta}(\boldsymbol{\theta},\overline{\boldsymbol{\theta}}); \tag{A.4}$$

where $L_{\delta}(\boldsymbol{\theta}) = \frac{\gamma L_P L_{\pi}(\boldsymbol{\theta})}{1-\gamma L_P\left(1+L_{\pi_{\boldsymbol{\theta}}}\right)}$.

In the same fashion, we can now define the state occupancy measure in the task $\mathcal{M}_{\boldsymbol{\omega}}$ as $\delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}}$ and we can prove the following lemma:[1]

Lemma A.2 (L-continuity of meta state occupancy measures)**.** *Given Assumptions 3.1, 3.3 and 4.2, if $\gamma L_P(\boldsymbol{\omega})\left(1+L_{\pi_{\boldsymbol{\theta}}}\right) < 1$, then the Kantorovich distance between a pair of γ-discounted feature-state distributions is CLC w.r.t. context $\boldsymbol{\omega}$:*

$$\mathcal{W}_1\left(\delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}}, \delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\right) \leq L_{\delta}(\boldsymbol{\omega}) d_{\Omega}(\boldsymbol{\omega},\overline{\boldsymbol{\omega}}), \quad \forall(\boldsymbol{\omega},\overline{\boldsymbol{\omega}}) \in \Omega^2; \tag{A.5}$$

where $L_{\delta}(\boldsymbol{\omega}) = \frac{\gamma L_{\boldsymbol{\omega}_P}}{1-\gamma L_P(\boldsymbol{\omega})\left(1+L_{\pi_{\boldsymbol{\theta}}}\right)}$.

[1]Assumption 3.3 is not entirely required, but only Inequality A.2 is required to hold.

Proof.

$$
\begin{aligned}
&\mathcal{W}_1\left(\delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}}, \delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\right) = \sup_f \left\{ \left| \int_{\mathcal{S}} \left(\delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}}(s) - \delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}(s)\right) f(s) ds \right| : \|f\|_L \leq 1 \right\} \\
&= \sup_f \left\{ \left| \int_{\mathcal{S}} \left(\mu(s) + \gamma \int_{\mathcal{S}} \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}\left(a|s'\right) P_{\boldsymbol{\omega}}\left(s|s',a\right) \delta^{\mu}_{\boldsymbol{\omega}}\left(s'\right) da ds' \right) f(s) - \right.\right. \\
&\quad \left.\left. - \left(\mu(s) + \gamma \int_{\mathcal{A}} \int_{\mathcal{S}} \pi_{\boldsymbol{\theta}}\left(a|s'\right) P_{\overline{\boldsymbol{\omega}}}\left(s|s',a\right) \delta^{\mu}_{\overline{\boldsymbol{\omega}}}\left(s'\right) da ds' \right) f(s) ds \right| : \|f\|_L \leq 1 \right\} \\
&= \gamma \sup_{f:\|f\|_L \leq 1} \left\{ \left| \int_{\mathcal{S}} f(s) \int_{\mathcal{A}} \int_{\mathcal{S}} \left(P_{\boldsymbol{\omega}}\left(s|s',a\right) \pi_{\boldsymbol{\theta}}\left(a|s'\right) \delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}}\left(s'\right) \right.\right.\right. \\
&\quad \left.\left.\left. - P_{\overline{\boldsymbol{\omega}}}\left(s|s',a\right) \pi_{\boldsymbol{\theta}}\left(a|s'\right) \delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\left(s'\right)\right) ds' da ds \right| \right\} \\
&= \gamma \sup_{f:\|f\|_L \leq 1} \left\{ \left| \int_{\mathcal{S}} f(s) \int_{\mathcal{A}} \int_{\mathcal{S}} P_{\boldsymbol{\omega}}\left(s|s',a\right) \pi_{\boldsymbol{\theta}}\left(a|s'\right) \left(\delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}}\left(s'\right) - \delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\left(s'\right)\right) ds' da ds \right.\right. \\
&\quad \left.\left. + \int_{\mathcal{S}} f(s) \int_{\mathcal{A}} \int_{\mathcal{S}} \left(P_{\boldsymbol{\omega}}\left(s|s',a\right) - P_{\overline{\boldsymbol{\omega}}}\left(s|s',a\right)\right) \pi_{\boldsymbol{\theta}}\left(a|s'\right) \delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\left(s'\right) ds' da ds \right| \right\} \\
&\leq \underbrace{\gamma \sup_{f:\|f\|_L \leq 1} \left\{ \left| \int_{\mathcal{S}} \left(\delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}}\left(s'\right) - \delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\left(s'\right)\right) \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}\left(a|s'\right) \int_{\mathcal{S}} P_{\boldsymbol{\omega}}\left(s|s',a\right) f(s) ds da ds' \right| \right\}}_{(1)} \\
&\quad + \underbrace{\gamma \sup_{f:\|f\|_L \leq 1} \left\{ \left| \int_{\mathcal{S}} \delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\left(s'\right) \int_{\pi}^{\boldsymbol{\theta}} \left(a|s'\right) \int_{\mathcal{S}} \left(P_{\boldsymbol{\omega}}\left(s|s',a\right) - P_{\overline{\boldsymbol{\omega}}}\left(s|s',a\right)\right) f(s) ds da ds' \right| \right\}}_{(2)}.
\end{aligned}
\tag{A.6}
$$

Now, we focus on term (1):

$$
\begin{aligned}
&\sup_{f:\|f\|_L \leq 1} \left\{ \left| \int_{\mathcal{S}} \left(\delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}}\left(s'\right) - \delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\left(s'\right)\right) \underbrace{\int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}\left(a|s'\right) \int_{\mathcal{S}} P_{\boldsymbol{\omega}}\left(s|s',a\right) f(s) ds da}_{h^f_{\boldsymbol{\omega},\boldsymbol{\theta}}(s')} ds' \right| \right\} \\
&= L_P(\boldsymbol{\omega}) \left(1 + L_{\pi_{\boldsymbol{\theta}}}\right) \sup_{f:\|f\|_L \leq 1} \left\{ \left| \int_{\mathcal{S}} \left(\delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}}\left(s'\right) - \delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\left(s'\right)\right) \frac{h^f_{\boldsymbol{\omega},\boldsymbol{\theta}}\left(s'\right)}{L_P(\boldsymbol{\omega}) \left(1 + L_{\pi_{\boldsymbol{\theta}}}\right)} ds' \right| \right\} \\
&\leq L_P(\boldsymbol{\omega}) \left(1 + L_{\pi_{\boldsymbol{\theta}}}\right) \sup_{\tilde{f}:\|\tilde{f}\|_L \leq 1} \left\{ \left| \int_{\mathcal{S}} \left(\delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}}\left(s'\right) - \delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\left(s'\right)\right) \tilde{f}\left(s'\right) ds' \right| \right\} \qquad \text{(A.7)} \\
&\leq L_P(\boldsymbol{\omega}) \left(1 + L_{\pi_{\boldsymbol{\theta}}}\right) \mathcal{W}_1\left(\delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}}, \delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\right).
\end{aligned}
$$

where inequality A.7 comes from the fact that
$h^f_{\boldsymbol{\omega},\boldsymbol{\theta}}\left(s'\right) := \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}\left(a|s'\right) \int_{\mathcal{S}} P_{\boldsymbol{\omega}}\left(s|s',a\right) f(s) ds da$ is $L_P(\boldsymbol{\omega}) \left(1 + L_{\pi_{\boldsymbol{\theta}}}\right)$-PLC w.r.t. the

state space $\mathcal{S}$. From the other side, the term (2) can be bounded as follows:

$$\begin{aligned}
&\sup_{f:\|f\|_L\leq 1}\left\{\left|\int_{\mathcal{S}}\delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\left(s'\right)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}\left(a|s'\right)\int_{\mathcal{S}}\left(P_{\boldsymbol{\omega}}\left(s|s',a\right)-P_{\overline{\boldsymbol{\omega}}}\left(s|s',a\right)\right)f(s)dsdads'\right|\right\}\\
&\leq\int_{\mathcal{S}}\delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\left(s'\right)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}\left(a|s'\right)\sup_{f:\|f\|_L\leq 1}\left\{\left|\int_{\mathcal{S}}\left(P_{\boldsymbol{\omega}}\left(s|s',a\right)-P_{\overline{\boldsymbol{\omega}}}\left(s|s',a\right)\right)f(s)ds\right|\right\}dads'\\
&\leq\int_{\mathcal{S}}\delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\left(s'\right)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}\left(a|s'\right)\mathcal{W}_1\left(P_{\boldsymbol{\omega}}\left(\cdot|s',a\right),P_{\overline{\boldsymbol{\omega}}}\left(\cdot|s',a\right)\right)dads'\\
&\leq L_{\boldsymbol{\omega}_P}d_{\Omega}(\boldsymbol{\omega},\overline{\boldsymbol{\omega}}).
\end{aligned}$$

Finally, merging everything:

$$\begin{aligned}
\mathcal{W}_1\left(\delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}},\delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\right)&\leq\gamma L_P(\boldsymbol{\omega})\left(1+L_{\pi_{\boldsymbol{\theta}}}\right)\mathcal{W}_1\left(\delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}},\delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\right)+\gamma L_{\boldsymbol{\omega}_P}d_{\Omega}(\boldsymbol{\omega},\overline{\boldsymbol{\omega}})\\
&\leq\frac{\gamma L_{\boldsymbol{\omega}_P}}{1-\gamma L_P(\boldsymbol{\omega})\left(1+L_{\pi_{\boldsymbol{\theta}}}\right)}d_{\Omega}(\boldsymbol{\omega},\overline{\boldsymbol{\omega}}).
\end{aligned}$$

□

As a direct consequence, we define the joint probability $\zeta(\delta^{\mu}_{\pi},\pi)$ between the state distribution δ^{μ}_{π} and the stationary policy π. In the case of a parametric policy $\pi_{\boldsymbol{\theta}}$ and a $\boldsymbol{\omega}$-based MDP, we will denote it as $\zeta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}}$.It is then easy to prove that:

$$\mathcal{W}_1\left(\zeta^{\boldsymbol{\theta}}_{\mu,\boldsymbol{\omega}},\zeta^{\boldsymbol{\theta}}_{\mu,\overline{\boldsymbol{\omega}}}\right)\leq L_{\delta}(\boldsymbol{\omega})\left(1+L_{\pi_{\boldsymbol{\theta}}}\right)d_{\Omega}(\boldsymbol{\omega},\overline{\boldsymbol{\omega}}).$$

Proof.

$$\begin{aligned}
\mathcal{W}_1\left(\zeta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}},\zeta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\right)&=\sup_{f:\|f\|_L\leq 1}\left\{\left\|\int_{\mathcal{S}}\delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}}(s)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}(a|s)f(s,a)dads\right.\right.\\
&\qquad\left.\left.-\int_{\mathcal{S}}\delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}(s)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}(a|s)f(s,a)dads\right\|\right\}\\
&=\sup_{f:\|f\|_L\leq 1}\left\{\left\|\int_{\mathcal{S}}\left(\delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}}(s)-\delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}(s)\right)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}(a|s)f(s,a)dads\right\|_{p,\mu}\right\}\\
&\leq\left(1+L_{\pi_{\boldsymbol{\theta}}}\right)\mathcal{W}_1\left(\delta^{\mu}_{\boldsymbol{\omega},\boldsymbol{\theta}},\delta^{\mu}_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\right). \qquad (A.8)
\end{aligned}$$

where in A.8 we used the fact that, for a function f defined on $\mathcal{S}\times\mathcal{A}$ such that $\|f\|_L\leq 1$, then $\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}(a|s)f(s,a)da$ is $(1+L_{\pi_{\boldsymbol{\theta}}})$-LC. □

Lemma A.3 (L-continuity of η)**.** *Given Assumptions 3.1, 4.2, 3.3 and 3.4,* $\eta^{i}_{\boldsymbol{\omega},\boldsymbol{\theta}}(s,a):=\nabla_{\boldsymbol{\theta}_i}\log\pi_{\boldsymbol{\theta}}(s,a)Q^{\Theta}_{\boldsymbol{\omega}}(s,a)$ *is L-CLC w.r.t. the context* $\boldsymbol{\omega}$*:*

$$\begin{aligned}
\left|\eta^{\boldsymbol{\theta}}_{i,\boldsymbol{\omega}}(s,a)-\eta^{\boldsymbol{\theta}}_{i,\overline{\boldsymbol{\omega}}}(s,a)\right|&=\left|\nabla_{\Theta_i}\log\pi_{\boldsymbol{\theta}}(a|s)\left(Q^{\boldsymbol{\theta}}_{\boldsymbol{\omega}}(s,a)-Q^{\boldsymbol{\theta}}_{\overline{\boldsymbol{\omega}}}(s,a)\right)\right|\\
&\leq\mathcal{M}^{i}_{\boldsymbol{\theta}}\left|Q^{\boldsymbol{\theta}}_{\boldsymbol{\omega}}(s,a)-Q^{\Theta}_{\overline{\boldsymbol{\omega}}}(s,a)\right|\\
&\leq\mathcal{M}^{i}_{\boldsymbol{\theta}}L_{\boldsymbol{\omega}_Q}d_{\Omega}(\boldsymbol{\omega},\overline{\boldsymbol{\omega}}).
\end{aligned}$$

Moreover, η *is also L-LC w.r.t. the joint state-action space* $\mathcal{S}\times\mathcal{A}$*:*

$$\left|\eta^{i}_{\boldsymbol{\omega},\boldsymbol{\theta}}(s,a)-\eta^{i}_{\boldsymbol{\omega},\boldsymbol{\theta}}(\overline{s},\overline{a})\right|\leq L^{i}_{\eta^{\boldsymbol{\theta}}(\boldsymbol{\omega})}d_{\mathcal{S}\times\mathcal{A}}((s,a),(\overline{s},\overline{a}));$$

where $L^{i}_{\eta^{\boldsymbol{\theta}}(\boldsymbol{\omega})}=\frac{R_{\max}}{1-\gamma}L^{i}_{\nabla\log\pi_{\boldsymbol{\theta}}}+\mathcal{M}^{i}_{\boldsymbol{\theta}}L_{Q^{\boldsymbol{\theta}}(\boldsymbol{\omega})}$.

Theorem A.4 (L-continuity of the performance gradient)**.** *Finally, given Assumptions 3.1, 4.2, 3.3 and 3.4, the return gradient is CLC w.r.t. the context $\boldsymbol{\omega}$:*

$$|\nabla_{\boldsymbol{\theta}_i} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}_i} J_{\overline{\boldsymbol{\omega}}}(\boldsymbol{\theta})| \leq L_{\nabla J}(\boldsymbol{\omega}) d_\Omega(\boldsymbol{\omega}, \overline{\boldsymbol{\omega}}),$$

where $L_{\nabla J}(\boldsymbol{\omega}) = L^i_{\eta^{\boldsymbol{\theta}}_{\boldsymbol{\omega}}} \left(1 + L_{\pi_{\boldsymbol{\theta}}}\right) L_\delta(\boldsymbol{\omega}) + \mathcal{M}^i_{\boldsymbol{\theta}} L_{\boldsymbol{\omega}_Q}$.

Proof.

$$\begin{aligned}
&|\nabla_{\boldsymbol{\theta}_i} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}_i} J_{\overline{\boldsymbol{\omega}}}(\boldsymbol{\theta})| = \left| \mathop{\mathbb{E}}_{(s,a)\sim\zeta^\mu_{\boldsymbol{\omega},\boldsymbol{\theta}}} \left[\eta^i_{\boldsymbol{\omega},\boldsymbol{\theta}}(s,a)\right] - \mathop{\mathbb{E}}_{(s,a)\sim\zeta^\mu_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}} \left[\eta^i_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}(s,a)\right] \right| \\
&\leq \left| \int_{\mathcal{S}} \int_{\mathcal{A}} \left(\zeta^\mu_{\boldsymbol{\omega},\boldsymbol{\theta}} - \zeta^\mu_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\right)(s,a) \eta^i_{\boldsymbol{\omega},\boldsymbol{\theta}}(s,a) da ds \right| + \left| \mathop{\mathbb{E}}_{(s,a)\sim\zeta^\mu_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}} \left[\eta^i_{\boldsymbol{\omega},\boldsymbol{\theta}}(s,a) - \eta^i_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}(s,a)\right] \right| \\
&\leq L^i_{\eta^{\boldsymbol{\theta}}(\boldsymbol{\omega})} \mathcal{W}_1\left(\zeta^\mu_{\boldsymbol{\omega},\boldsymbol{\theta}}, \zeta^\mu_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}\right) + \left| \int_{\mathcal{S}} \delta^\mu_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}(s) \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \left(\eta^i_{\boldsymbol{\omega},\boldsymbol{\theta}}(s,a) - \eta^i_{\overline{\boldsymbol{\omega}},\boldsymbol{\theta}}(s,a)\right) \mathrm{d}a \, \mathrm{d}s \right| \\
&\leq \left[L^i_{\eta^{\boldsymbol{\theta}}_{\boldsymbol{\omega}}} \left(1 + L_{\pi_{\boldsymbol{\theta}}}\right) L_\delta(\boldsymbol{\omega}) + \mathcal{M}^i_{\boldsymbol{\theta}} L_{\boldsymbol{\omega}_Q}\right] d_\Omega(\boldsymbol{\omega}, \overline{\boldsymbol{\omega}}).
\end{aligned}$$

□

A.2 Experiment Details

In this section, we provide more details regarding the experimental campaign provided. In the following environments, all the policies considered are Gaussian, and linear w.r.t. the state observed (with bias θ_0), i.e. $\pi_{\boldsymbol{\theta}}(a|s) \sim \mathcal{N}(\theta_0 + \boldsymbol{\theta}^\top s, \sigma^2)$, where σ is fixed standard deviation, with a different setting for each environment.

A.2.1 Navigation2D Description

The Navigation2D environment consists of a 2-dimensional square space in which an agent, represented as a point, aims to reach a goal in the plane traversing the minimum distance.

At the start of the episode, the agent is placed in the initial position $s_0 = (0, 0)$. Then, at each step t the agent observes its current position and performs an action a_t corresponding to movement speeds along the x and y axes:

$$a_t = (v_x, v_y), \text{ where } v_x, v_y \in [-v^{\max}, v^{\max}]. \tag{A.9}$$

According to this action, the agent can move in every direction of the plane, with a limit on the maximum speed $v^{\max} = 0.1$ allowed in a single step. These parameters determine the minimum number of steps necessary to reach the goal and can be varied to tune the difficulty of the environment.

At each step, the environment produces a reward equal to the negative Euclidean distance from the goal:

$$r_t = \sqrt{(x_t - x_{\text{goal}})^2 + (y_t - y_{\text{goal}})^2}. \tag{A.10}$$

An episode terminates when the agent is within a threshold distance d_{thresh} from the goal or when the horizon $H = 10$ is reached.

The distribution of tasks is implemented as a CMDP $\mathcal{M}(\boldsymbol{\omega})$ in which, at each episode, a different goal point is selected at random. The context $\boldsymbol{\omega}$ is given by a 2D vector, such that:

$$\boldsymbol{\omega} = (x_{\text{goal}}, y_{\text{goal}}), \text{ where } x_{\text{goal}}, y_{\text{goal}} \sim U(-1, 1). \tag{A.11}$$

Parameters used for experiments:

- initial policy distribution $\rho = \mathcal{N}(0, 0.1)$;
- discount factor $\gamma = 0.99$;
- policy standard deviation $\sigma = 1.001$;
- task distribution $\psi = \mathcal{U}([-0.5, 0.5]^2)$;
- meta-discount factor $\widetilde{\gamma} = 1$;
- metaFQI dataset method: trajectories;
- metaFQI number of samples: $K = 4000$ with learning horizon $T = 20$;
- inner trajectories $n = 200$ with horizon $H = 10$;
- number of estimators = 50, minimum samples split = 0.01;
- step size $\mathcal{H} = [0, 8]$;
- step size sampling distribution: uniform in $\mathcal{H}$;
- step size selected in evaluation from an evenly spaced discretization of 101 values in $\mathcal{H}$.

A.2.2 Minigolf Description

In the minigolf game, the agent has to shoot a ball with radius r inside a hole of diameter D with the smallest number of strokes. The friction imposed by the green surface is modeled by a constant deceleration $d = \frac{5}{7}\rho g$, where ρ is the dynamic friction coefficient between the ball and the ground and g is the gravitational acceleration. Given the distance x of the ball from the hole, the agent must choose the force a, from which the velocity of the ball v of the ball is determined as $v = al^2(1 + \epsilon)$, where $\epsilon \sim \mathcal{N}(0, 0.25)$ and l is the putter length. For each distance x, the ball falls in the hole if its velocity v ranges from $v_{min} = \sqrt{2dx}$ to $v_{max} = \sqrt{(2D - r)^2 \frac{g}{2r} + v_{min}^2}$. In this case, the episode ends with a null reward; if $v > v_{max}$ the ball falls outside the green, and the episode ends with a reward equal to -100. Otherwise, if $v < v_{min}$, the agents get a reward equal to -1, and the episode goes on from a new position $x_{new} = x_{old} - \frac{v^2}{2d}$. At the beginning of each episode, the initial position is selected from a uniform distribution between 0m and 20m from the hole. The stochasticity of the action implies that the stronger the action chosen the more uncertain the outcome, as the effect of r.v. ϵ becomes more effective. As a result, when it is away from the hole, the agent might not prefer to try to make a hole in one shot, preferring to perform a sequence of closer shots. In this case, the context is given by the friction coefficient $\rho \in [0.065, 0.196]$ and by the putter length $l \in [0.7, 1]m$.

During the experiment, the environment parameters are set to imitate the dynamics of a realistic shot in a minigolf green, within the limits of our simplified simulation. This is the complete configuration adopted:

- horizon $H = 20$;
- discount factor $\gamma = 0.99$;
- angular velocity $\omega \in [1 \times 10^{-5}, 10]$;
- initial distance $x_0 \in [0, 20]$ meters;
- ball radius $r = 0.02135$ meters;
- hole diameter $D = 0.10$ meters;
- gravitational acceleration $g = 9.81 \frac{\text{meters}}{\text{second}^2}$.

The distribution of tasks is built as a CMDP $\mathcal{M}(\boldsymbol{\omega})$, induced by the pair $\boldsymbol{\omega} = (l, \rho)$. At each meta-episode, a new task is sampled from a multivariate uniform distribution within this ranges:

- putter length $l \sim U(0.7, 1)$ meters;
- friction coefficient $\rho \sim U(0.065, 0.196)$.

Parameters used for experiments:

- initial policy distribution $\boldsymbol{\theta} = (w, b) \sim U((-1, 2), (-2, 3.5))$;
- policy standard deviation $\sigma = 0.1$;
- meta-discount factor $\widetilde{\gamma} = 1$;
- metaFQI dataset method: generative;
- metaFQI number of samples: $K = 10000$;
- inner trajectories $n = 400$ with horizon $H = 20$;
- number of estimators = 50, minimum samples split = 0.01;
- step size space: $\mathcal{H} = [0, 1]$
- step size sampling distribution: uniform in $\mathcal{H}$;
- step size selected in evaluation from an evenly spaced discretization of 101 values in $\mathcal{H}$.

A.2.3 CartPole Description

The CartPole environment (Barto et al., 1990), also known as the Inverted Pendulum problem, consists of a pole attached to a cart by a non-actuated joint, making it an inherently unstable system. The cart can move horizontally along a frictionless track to balance the pole. The objective is to maintain the equilibrium as long as possible.

In this implementation, an episode starts with the pendulum in a vertical position. At each step, the agent observes the following 4-tuple of continuous values:

- cart position $x_{\text{cart}} \in [-4.8, 4.8]$;
- cart velocity $v_{\text{cart}} \in \mathbb{R}$;
- pole angle $\phi_{\text{pole}} \in [-0.418, 0.418]$ rad;
- pole angular velocity $\omega_{\text{pole}} \in \mathbb{R}$.

Given the state, the agent chooses an action between 0 and 1 to push the cart to the left or to the right. For each step in which the pole is in balance, the environment produces a reward of +1. An episode ends when the pole angle from the vertical position is higher than 12 degrees, the cart moves more than 2.4 units from the center, or the horizon $H = 100$ is reached.

In our experiments, we set the environment parameters to these values:

- mass of the cart $m_{cart} = 1$ kg;
- length of the pole $l_{pole} = 0.5$ m;
- force applied by the cart $F = 10$ N.

The CMDP $\mathcal{M}(\boldsymbol{\omega})$ is induced by varying two environment parameters, the pole mass m_{pole} and the pole length l_{pole}, that form the context parameterization $\omega = (m_{pole}, l_{pole})$. Each task in the meta-MDP is built by sampling $\boldsymbol{\omega}$ from a multivariate uniform distribution, within these ranges:

- pole length $l_{pole} \sim U(0.5, 1.5)\mathrm{m}$;
- pole mass $m_{pole} \sim U(0.1, 2)$ kg.

Parameters used for experiments:

- initial policy distribution $\boldsymbol{\theta_d} \sim \mathcal{N}(0, 0.01)$ for each component $\boldsymbol{\theta}_d$;
- policy standard deviation $\sigma = 1.001$;
- meta-discount factor $\widetilde{\gamma} = 1$;
- metaFQI dataset method: trajectories;
- metaFQI number of samples: $K = 3200$ with learning horizon $T = 15$;
- inner trajectories $n = 100$ with horizon $H = 100$;
- number of estimators = 150, minimum samples split = 0.05;
- step size $\mathcal{H} = [0, 10]$;
- step size sampling distribution: uniform in $\mathcal{H}$;
- step size selected in evaluation from an evenly spaced discretization of 101 values in $\mathcal{H}$.

A.2.4 Half Cheetah Description

The CMDP $\mathcal{M}(\boldsymbol{\omega})$ is induced by varying the goal velocity of the half cheetah v_{goal}, which defines the context $\boldsymbol{\omega}$, with uniform distribution $U(0, 2)$.

Parameters used for experiments:

- initial policy distribution $\boldsymbol{\theta_d} \sim \mathcal{N}(0, 0.1)$ for each component $\boldsymbol{\theta}_d$;
- policy standard deviation $\sigma = 1.001$;
- meta-discount factor $\widetilde{\gamma} = 1$;
- metaFQI dataset method: trajectories;
- metaFQI number of samples: $K = 200$ with learning horizon $T = 80$;
- inner trajectories $n = 100$ with horizon $H = 100$;

- number of estimators = 150, minimum samples split = 0.05;
- step size $\mathcal{H} = [0, 1]$;
- step size sampling distribution: uniform in $\mathcal{H}$;
- step size selected in evaluation from an evenly spaced discretization of 101 values in $\mathcal{H}$.

A.2.5 Metagrad Implementation

In Figure 4.3, we provided the comparison of the proposed approach with meta-gradient (Xu et al., 2018). The algorithm performs an online update following the gradient of the (differentiable) return function w.r.t. the hyperparameter set. In particular, after updating the current parameter set $\boldsymbol{\theta}$ to $\boldsymbol{\theta}'$ following the update rule $f(\boldsymbol{\theta}, h, \tau)$, the hyperparameter gradient of the return function on a new batch of trajectories τ' is approximated as in equation A.12:

$$\frac{\partial J(\boldsymbol{\theta}', h, \tau')}{\partial h} = \frac{\partial J(\boldsymbol{\theta}', h, \tau')}{\partial \boldsymbol{\theta}'} \frac{d\boldsymbol{\theta}'}{dh} \approx \frac{\partial J(\boldsymbol{\theta}', h, \tau')}{\partial \boldsymbol{\theta}'} z', \tag{A.12}$$

where $z \approx \frac{d\boldsymbol{\theta}}{dh}$ is update as an accumulative trace with parameter μ:

$$z' = \mu z + \frac{\partial f(\boldsymbol{\theta}, h, \tau)}{\partial h}.$$

In the original paper, the optimized hyperparameters (as well as in Yu et al. (2006) with HOOF implementation) were the discount factor γ and the exponential weight coefficient λ related to the generalized advantage estimation. In our case, the hyperparameter considered is the stepsize. Hence, from the NGA update function, the following hold:

$$\begin{aligned} \boldsymbol{\theta}' &= \boldsymbol{\theta} + h \frac{\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta})}{\|widehat\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta})\|_2} \\ z' &= \mu z + \frac{widehat\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta})}{\|\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta})\|_2}. \end{aligned}$$

After the update, the stepsize is updated through a meta-hyperparameter β and a new batch of trajectories with policy $\pi_{\boldsymbol{\theta}'}$:

$$\begin{aligned} h' &= h - \beta \frac{\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}')}{\|\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}')\|_2} z' \\ &= h - \beta \frac{\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}')}{\|\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}')\|_2} (\mu z + \frac{\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta})}{\|\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta})\|_2}) \end{aligned}$$

In the case $\mu = 0$, as adopted in Xu et al. (2018), the stepsize update function reduces to computing the cosine similarity between consecutive gradients:

$$h' = h - \beta \ \text{sim}\Big(\widehat{\nabla}_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}'), widehat\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\omega}}(\boldsymbol{\theta}))\Big),$$

where sim(x, y) denotes the cosine similarity between vectors x and y.

A.3 Other Results

In this section of the appendix, we provide more experimental results.

A.3.1 Meta Cartpole SwingUp

For the experimental session, a variant of the Cartpole presented in Section 4.7 is the Cartpole Swingup variant, introduced in Tornio and Raiko (2006) and implemented in Duan et al. (2016). The main difference is the following: classic CartPole environment provides a unitary reward per step until the end of the episode, which ends when the pole angle from the vertical axis ϕ_{pole} is more than 12 degrees from vertical, or the cart moves more than $x_{thresh} = 2.4$ units from the center. CartPole Swingup, instead, has a reward equal to $\cos(\phi_{pole})$, and equal to -100 if the cart threshold $x_{thresh} = 3$ is reached. Finally, the CMDP in this case is built by changing only the pole mass m_{pole} with a uniform distribution $\sim U(0.1, 2)$.

Parameters used for experiments:

- initial policy distribution $\boldsymbol{\theta_d} \sim \mathcal{N}(0, 0.1)$ for each component $\boldsymbol{\theta}_d$;
- policy standard deviation $\sigma = 1.001$;
- meta-discount factor $\widetilde{\gamma} = 1$;
- metaFQI dataset method: trajectories;
- metaFQI number of samples: $K = 300$ with learning horizon $T = 25$;
- inner trajectories $n = 100$ with horizon $H = 200$;
- number of estimators = 150, minimum samples split = 0.05;
- step size $\mathcal{H} = [0, 0.5]$;
- step size sampling distribution: uniform in $\mathcal{H}$;
- step size selected in evaluation from an evenly spaced discretization of 101 values in $\mathcal{H}$.

The results are depicted in Figure A.1: the model chosen is the one which maximizes the reward, i.e. $N = 1$: however, all iterations have similar performances, which resemble the choice of a fixed learning rate equal to 0.1. Indeed, the actions chosen are almost always around this value, with the exception of the first two steps, where higher step sizes are taken into account (with slightly better learning).

A.3.2 Comparison among MetaFQI Iterations.

As said, as the regression procedures are iterated in the application of the FQI algorithm, there is a trade-off between a larger planning horizon and the accumulation of new regression errors. In Figure A.2 we show some of the learning curves with different metaFQI iterations. For all the environments considered, it is possible to see that the direct regression on the meta reward (i.e. one metaFQI iteration) does not provide the best performances, while from a certain point, the results start to get worse. As far as the Meta Cartpole environment is concerned, we can clearly see that the models select progressively more cautious steps in order to improve learning, as explained in Section 4.7.

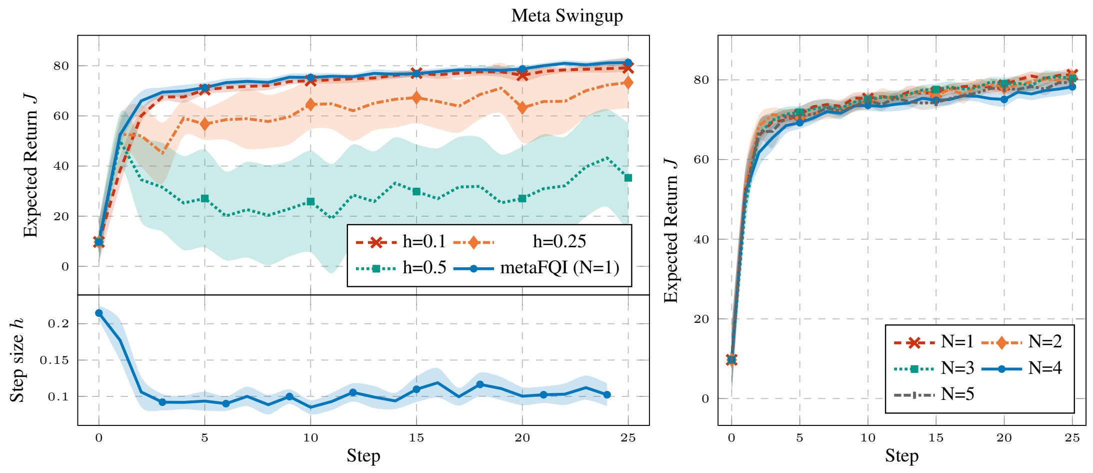


Figure A.1: *Meta Swingup metaFQI model performance on 20 random test contexts and initial policies against fixed step sizes. The top left plot shows the 95% confidence intervals of the expected returns. The bottom left plot shows the meta-action chosen through learning iterations. N represents the metaFQI iteration selected. The right plot shows the performance among different iterations.*

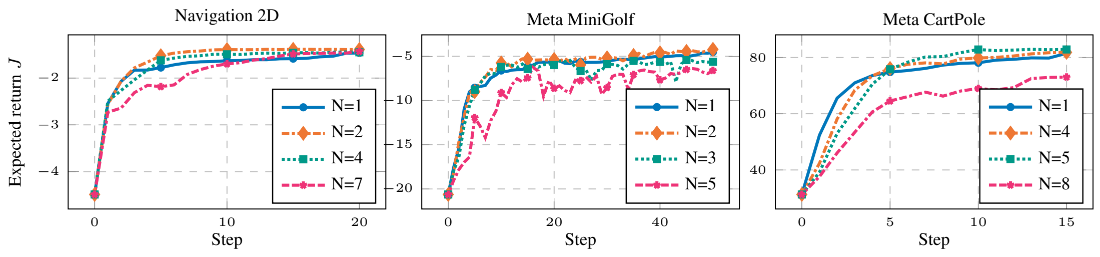


Figure A.2: *metaFQI model performance among different iterations. For the sake of clarity, only the average values are shown.*

A.3.3 Comparison with Learning Rate schedules: Details

In Figure 4.3, we compared our approach with three different baselines, where the initial learning rate (denoted as α) was tuned by grid search on 20 random test contexts and initial policies, and the best were selected for the comparison. Adam and RMSprop updates have a poor performance when applied to the natural gradient $g(\boldsymbol{\theta})$, hence they have been tuned by adopting the (standard) stochastic gradient $\widehat{\nabla}_N j(\boldsymbol{\theta})$. While for the decaying learning rate, the only hyperparameter is the initial rate, the other methods depend also on other variables, which were kept fixed to the suggested values: for RMSProp, the parameters were fixed as $\rho = 0.9, \epsilon = 1e - 7$, while for Adam the parameters were fixed as $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e - 7$. The notation for these parameters follows the one used in the implementations of the optimizers within Python Keras API (Gulli and Pal, 2017), used to perform the updates.

As far as the meta-gradient is concerned, the algorithm shows heavy dependence on the initial stepsize h_0 selected, while the impact of the meta-stepsize β is reduced. μ is always set to 0. HOOF has been tested by selecting the KL-constraintsϵ in the set $[0.0001, 0.001, 0.01, 0.02, 0.05]$, and by sampling $Z = 100$ candidate hyperparameters per iteration in the meta-action space.

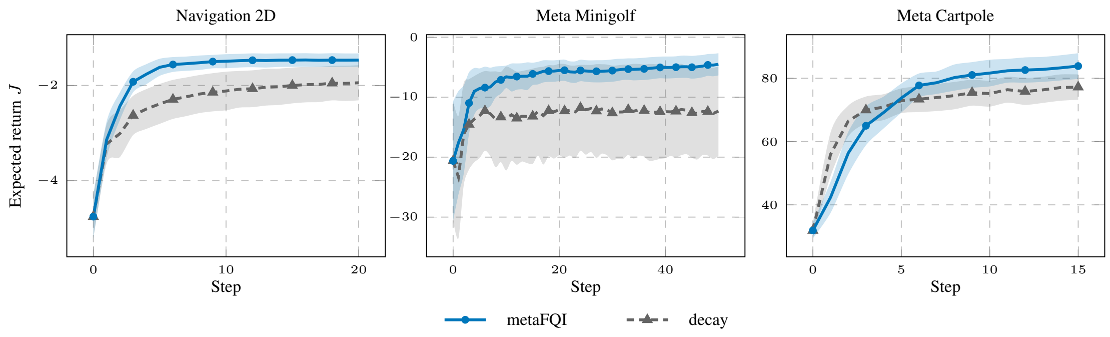


Figure A.3: *metaFQI model performance against an exponentially decreasing (decaying) learning rate. 20 runs, avg± 95% c.i.*

Table A.1: *Best initial learning rate selected. Evaluation using 20 different random tasks and policies.*

	Nav2D	Meta MiniGolf	MetaCartpole	Half-Cheetah
RMSProp	0.9	0.3	0.3	0.3
Adam	0.8	0.08	0.3	0.5
Metagrad: h_0	3	0.3	1	0.5
Metagrad: β	0.001	5	0.1	0.01
decay	5	2	7.5	N/A

Another common step size schedule adopted to grant convergence is a decaying step size $h_{t+1} = \frac{\alpha}{t}$ (similar as is an exponentially decreasing learning rate $h_{t+1} = \alpha h_t$), where h_0 is the initial learning rate. The comparison of the model trained through metaFQI and this baseline is shown in Figure A.3, while the best initial learning rates chosen for each of the environments and of the baselines (and shown in Figure 4.3) can be found in Table A.1.

A.3.4 Extension of Trajectory Length

Cartpole: In Figure 4.2 we have shown the performance of metaFQI models trained on Cartpole trajectories with horizon $T = 15$ update steps. In order to have a fair comparison, we have tested the resulting metaFQI model (and NGA with fixed step sizes) performing the same number of total updates as the training trajectories. However, as the learning curves were far from convergence, one may ask what happens if the horizon is increased: Figure A.4 depicts the performance of the same models (trained on $T = 15$-steps long trajectories) with an increased horizon of 60 steps.

Half Cheetah: One of the main contributions of the work is to introduce an offline algorithm to learn a schedule of stepsizes through metaFQI. The results on HalfCheetah metaFQI model in Figure 4.2 show that it is possible to improve and speed up learning in different contexts w.r.t. to a fixed stepsize (and other schedules) using a batch of learning trajectories with horizon $T = 80$. While the figure shows the comparison on test runs with the same horizon, convergence is again far from being reached. In Figure A.5 we show the performance of the models on an increased horizon of 500 updates. As we can see, the metaFQI model trained using trajectories with a total amount of $T = 80$ steps (red line) is dominated by NGA using a fixed stepsize $h = 0.2$ and $h = 0.4$ starting from 200 iterations: this is due to the fact that the meta-observations in

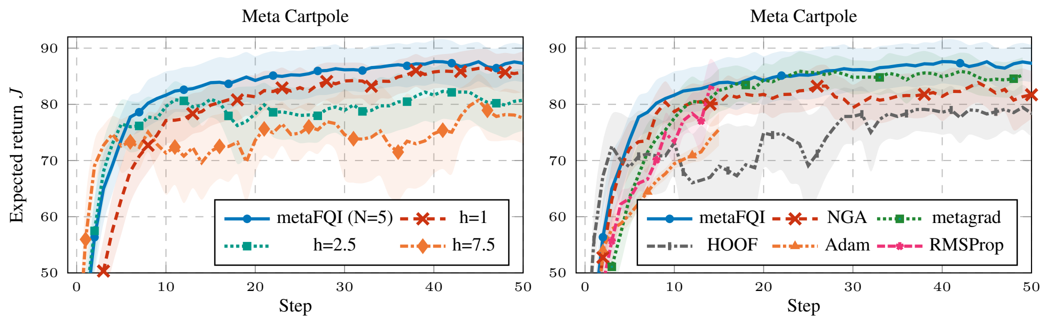


Figure A.4: *metaFQI model performance in Cartpole with extended test horizon: Train with* $T = 15$ *steps. Test on 60 updates.* Left side: *comparison against NGA with fixed size* h. Right side: *Comparison against benchmarks. 20 runs, avg± 95% c.i.*

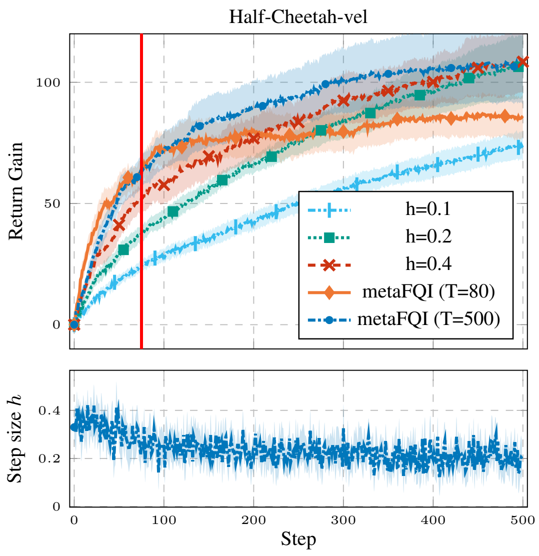


Figure A.5: *Top figure: metaFQI model performance in HalfCheetah against NGA with fixed size* h*. Test on 500 updates. The red vertical line represents the training horizon of the trajectories used to train one of the metaFQI agents (orange line). Bottom figure: meta actions chosen through learning iterations. 20 runs, avg± 95% c.i.*

this cases contain out-of-sample policy parametrizations, i.e. points in a policy space that were far from the training space. To overcome this limitation, the solution simply consists in collecting a batch of longer trajectories: when we consider a training horizon of $T = 500$ steps, the selected metaFQI model is again able to show better learning curves than using a fixed stepsize.

Explicit Knowledge of the Context: is it Informative?

In the experimental campaign, we assumed to be able to represent the parametrized context ω, as this information can be used to achieve an *implicit task-identification* by the agent. However, in some cases, the external variables influencing the process might be not observable. Hence, the Meta-MDP can be modeled by creating a different task representation. However, the gradient itself already implicitly includes information regarding the transition and reward probabilities: what is lost when we do not consider

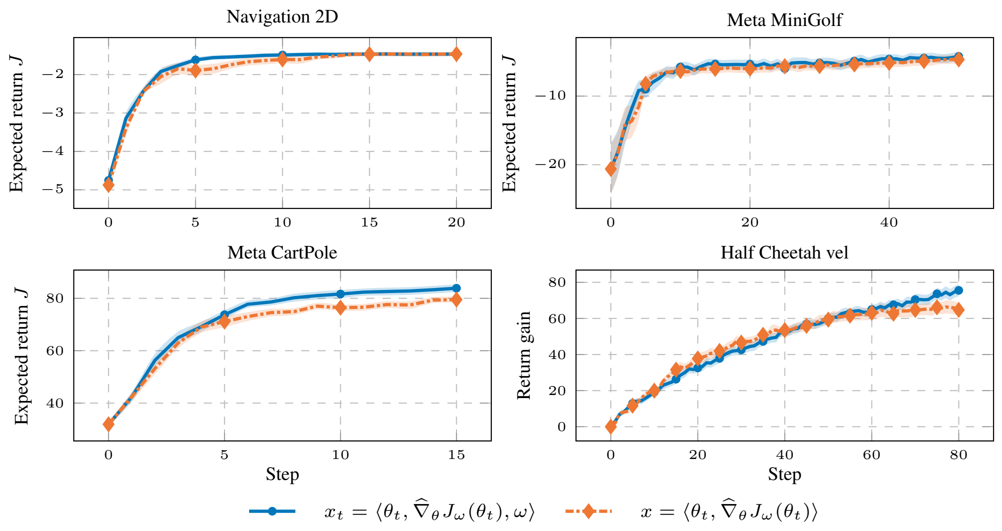


Figure A.6: *metaFQI model performance obtained by considering or excluding the explicit task parametrization ω. (95% c.i.)*

the explicit parametrization of the task? We address this question by retraining our models, and showing the results in Figure A.6: in general, there is no big loss in the performance, especially for the Minigolf environment; however, in Meta CartPole, the task parametrization seem to be informative to the choice of the step size.

Robustness of MetaFQI Regression and Effects of Double Q learning.

In Figure 4.2, we analyzed the results of a metaFQI model, evaluating the performance under different random policy initializations and tasks. One may wonder if our approach is robust with respect to the randomness included in the ExtraTrees regression. Hence, we trained different metaFQI models by setting 5 different random states (from 0 to 4), which controls the sampling of the features to apply a split and the draw of the splits. The random state is then equivalent to a seed for the Extra Trees, and it is applied up to the third metaFQI iteration. At this point, the models are tested on 20 random task/policy pairs, and their average return gain is taken for each learning step. As we can see in the left plot in Figure A.7, the 95% confidence interval, which is computed by comparing the different random states, is small enough to claim the robustness of our approach when applied to the Half-Cheetah environment.

Finally, the right plot in figure A.7 shows the effects of Clipped Double Q-Learning described in Section 4.6, which is compared with the standard FQI approach with a single Q value function (both are trained with the same number of iterations $N = 3$). The latter is still capable of choosing a dynamic learning rate obtaining better results than a fixed step while the former, as expected, provides even better return gains and lower variance over the same set of random test tasks and initial policies.

A.3.5 Experiments with Fixed Contexts

The field of application of metaFQI was presented up to this point as a contextual MDP, with a set of possible tasks. One natural question the reader might wonder is:

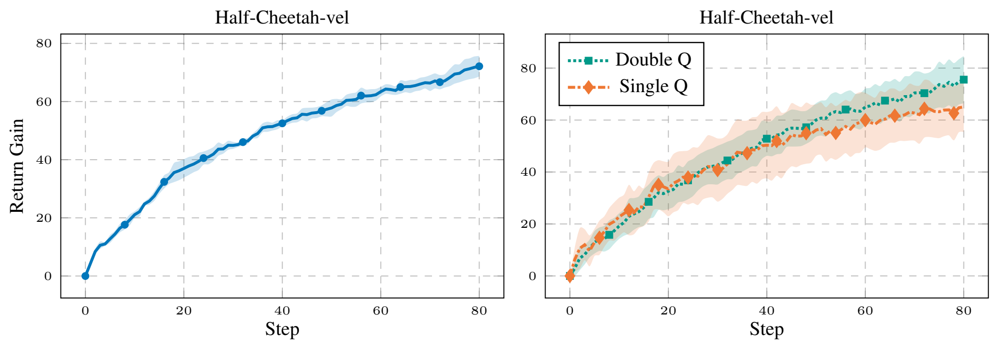


Figure A.7: *On the left: metaFQI model performance evaluated with 5 different random states (mean $\pm 95\%$ c.i. with respect to the average return gain for each random state). On the right: comparison between the adoption of a single Q and a Double Q-Learning approach for FQI (mean $\pm 95\%$ c.i. with respect to 20 different random test contexts and initial policies).*

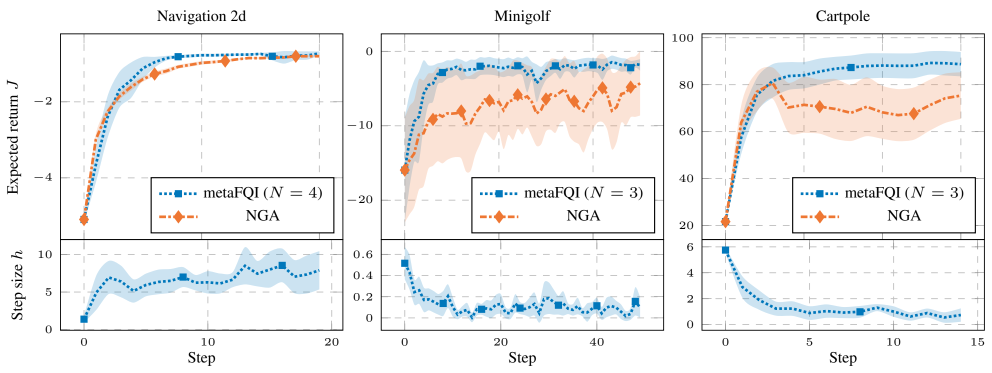


Figure A.8: *metaFQI model performance and tuned NGA on environments with fixed context.*

can we apply this approach to a standard MDP? The answer is trivially positive: we performed some experiments by fixing the task/context and the results are shown in figure A.8. the *metaFQI* curve is related to the performance in test obtained by our agent, and N denotes the iteration selected. Analogously, the NGA curve is related to the best constant learning rate, optimized by means of a grid search. As in the related meta-MDP environments, our approach can outperform the choice of a fixed step size.

A.3.6 Selection of a Single Learning Rate

One of the most important benefits of the adoption of an adaptive learning rate is provided by an increased number of degrees of freedom w.r.t. the choice of a single learning rate: hence, as an ablation study, we developed an agent capable of choosing only a fixed learning rate in the meta-MDP setting. The learning process has been conducted as follows: at first, we collected a set of trajectories by randomly varying the initial policy and the context, and selecting a random stepsize. Then, we performed a regression (with the same Extra-Trees architecture used for the meta-FQI agent) giving as input the context and the step size $x_i = (\boldsymbol{\omega}_i, h_i)$, and as output the final performance obtained in the trajectory, said J_i. Finally, in the performance evaluation, the sampled test context $\boldsymbol{\omega}_t$ is given as input to the agent, and the step size selected is then the one that attains the maximum estimated performance $h_t = \arg\max_h \widehat{J}_{\boldsymbol{\omega}_t}(\boldsymbol{\theta}_0, h)$. To have a

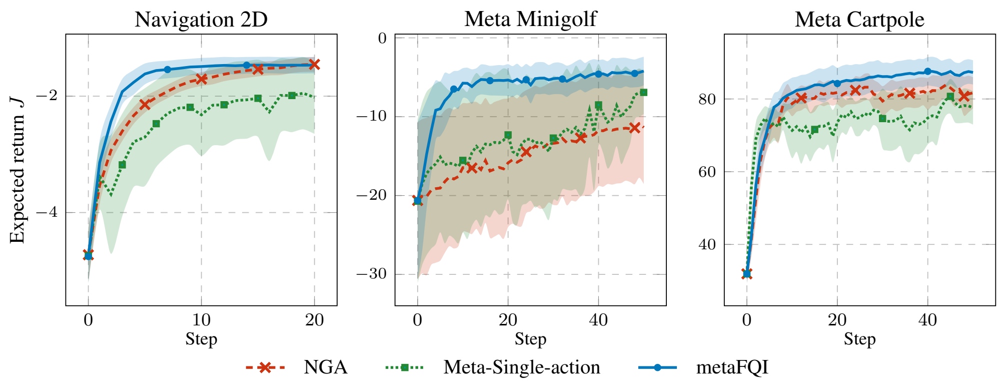


Figure A.9: *Comparison of performances obtained by a trained agent capable of choosing a fixed initial learning rate.*

fair comparison, the number of trajectories in the dataset is the same as for the Meta-FQI case. However, this new agent is only interested in the final returns, and not in the whole learning trajectory, hence the amount of data in input is reduced by a factor equal to the learning horizon H. The results are shown in figure A.9, with the curve labeled as *Meta-single-action*. As we can see the agents, even if capable of adapting the stepsize with different contexts, it is still unable to improve the NGA baseline, which has the same step size for each task. This is probably related to the fact that, to provide a fine performance estimation, the model needs a larger amount of samples, while meta-FQI uses all the single steps in the trajectory in the training dataset.

APPENDIX $\mathcal{B}$

Additional Results of Chapter 6

The contents of this Appendix can be summarized as follows:

– Appendix B.1 reports all proofs and derivations.

– Appendix B.2 provides additional considerations and discussion concerning the regularity conditions for bounding the performance loss due to action persistence.

– Appendix B.3 presents the experimental setting, together with additional experimental results (including an analysis of the effects of action persistence varying the batch size).

– Appendix B.4 reports some preliminary experiments to motivate the open questions stated in the main paper.

B.1 Proofs

In this section of the Appendix, we report the proofs of all the results presented in the main paper. In the following, we will denote as $\mathscr{B}(\mathcal{X})$ the set of bounded, measurable functions on $\mathcal{X}$.

B.1.1 Proofs of Section 6.5

Lemma B.1. *Let $\mathcal{M}$ be an MDP and $\pi \in \Pi$ be a Markovian stationary policy, then for any $k \in \mathbb{N}^+$ the following two identities hold:*

$$\begin{aligned}Q^\pi - Q^\pi_k &= \left(\mathrm{Id} - \gamma^k (p_\pi)^k\right)^{-1} \left((T^\pi)^k Q^\pi_k - \left(T^\delta\right)^{k-1} T^\pi Q^\pi_k\right) \\ &= \left(\mathrm{Id} - \gamma^k (p_\delta)^{k-1} p_\pi\right)^{-1} \left((T^\pi)^k Q^\pi - \left(T^\delta\right)^{k-1} T^\pi Q^\pi\right),\end{aligned}$$

where $\mathrm{Id} : \mathscr{B}(\mathcal{S} \times \mathcal{A}) \to \mathscr{B}(\mathcal{S} \times \mathcal{A})$ *is the identity operator over* $\mathcal{S} \times \mathcal{A}$.

Proof. We prove the equalities by exploiting the facts that Q^π and Q^π_k are the fixed points of T^π and T^π_k:

$$\begin{aligned}
Q^\pi - Q^\pi_k &= T^\pi Q^\pi - T^\pi_k Q^\pi_k \\
&= (T^\pi)^k Q^\pi - \left(T^\delta\right)^{k-1} T^\pi Q^\pi_k && \text{(B.1)} \\
&= (T^\pi)^k Q^\pi - \left(T^\delta\right)^{k-1} T^\pi Q^\pi_k \pm (T^\pi)^k Q^\pi_k && \text{(B.2)} \\
&= \gamma^k (p_\pi)^k (Q^\pi - Q^\pi_k) + \left((T^\pi)^k Q^\pi_k - \left(T^\delta\right)^{k-1} T^\pi Q^\pi_k\right), && \text{(B.3)}
\end{aligned}$$

where line (B.1) derives from recalling that $Q^\pi = T^\pi Q^\pi$ and exploiting Theorem 6.8, line (B.3) is obtained by exploiting the identity that holds for two generic bounded measurable functions $f, g \in \mathscr{B}(\mathcal{S} \times \mathcal{A})$:

$$(T^\pi)^k f - (T^\pi)^k g = \gamma^k (p_\pi)^k (f - g). \tag{B.4}$$

We prove this identity by induction. For $k = 1$ the identity clearly holds. Suppose equation (B.4) holds for $k - 1$, we prove that it holds for k too:

$$\begin{aligned}
(T^\pi)^k f - (T^\pi)^k g &= T^\pi (T^\pi)^{k-1} f - T^\pi (T^\pi)^{k-1} g \\
&= r + \gamma p_\pi (T^\pi)^{k-1} f - r - p_\pi \gamma (T^\pi)^{k-1} g \\
&= \gamma p_\pi \left((T^\pi)^{k-1} f - (T^\pi)^{k-1} g\right) && \text{(B.5)} \\
&= \gamma p_\pi \gamma^{k-1} (P^\pi)^{k-1} (f - g) && \text{(B.6)} \\
&= \gamma^k (P^\pi)^k (f - g),
\end{aligned}$$

where line (B.5) derives from the linearity of operator p_π and line (B.6) follows from the inductive hypothesis. From line (B.3) the result follows immediately, recalling that since $\gamma < 1$ the inversion of the operator is well-defined:

$$\begin{gathered}
Q^\pi - Q^\pi_k = \gamma^k (P^\pi)^k (Q^\pi - Q^\pi_k) + \left((T^\pi)^k Q^\pi_k - \left(T^\delta\right)^{k-1} T^\pi Q^\pi_k\right) \implies \\
\left(\mathrm{Id} - \gamma^k (P^\pi)^k\right) (Q^\pi - Q^\pi_k) = \left((T^\pi)^k Q^\pi_k - \left(T^\delta\right)^{k-1} T^\pi Q^\pi_k\right) \implies \\
Q^\pi - Q^\pi_k = \left(\mathrm{Id} - \gamma^k (P^\pi)^k\right)^{-1} \left((T^\pi)^k Q^\pi_k - \left(T^\delta\right)^{k-1} T^\pi Q^\pi_k\right).
\end{gathered}$$

The second identity of the statement is obtained with an analogous derivation, in which at line (B.2) we sum and subtract $\left(T^\delta\right)^{k-1} T^\pi Q^\pi$ and we exploit the identity for two bounded measurable functions $f, g \in \mathscr{B}(\mathcal{S} \times \mathcal{A})$:

$$\left(T^\delta\right)^{k-1} T^\pi Q f - \left(T^\delta\right)^{k-1} T^\pi Q g = \gamma^k (p_\delta)^{k-1} p_\pi (f - g). \tag{B.7}$$

□

Lemma B.2. *Let* $\mathcal{M}$ *be an MDP and* $\pi \in \Pi$ *be a Markovian stationary policy, then for any* $k \in \mathbb{N}^+$ *and any bounded measurable function* $f \in \mathscr{B}(\mathcal{S} \times \mathcal{A})$ *the following two* *identities hold:*

$$\begin{aligned}\left(T^{\pi}\right)^{k-1} f-\left(T^{\delta}\right)^{k-1} f &=\sum_{i=0}^{k-2} \gamma^{i+1}\left(p_{\pi}\right)^{i}\left(p_{\pi}-p_{\delta}\right)\left(T^{\delta}\right)^{k-2-i} f \\ &=\sum_{i=0}^{k-2} \gamma^{i+1}\left(p_{\delta}\right)^{i}\left(p_{\pi}-p_{\delta}\right)\left(T^{\pi}\right)^{k-2-i} f .\end{aligned}$$

Proof. We start with the first identity and we prove it by induction on k. For $k=1$, we have that the left-hand side is zero and the summation on the right-hand side has no terms. Suppose that the statement holds for $k-1$, we prove the statement for k:

$$\left(T^{\pi}\right)^{k-1} f-\left(T^{\delta}\right)^{k-1} f=\left(T^{\pi}\right)^{k-1} f-\left(T^{\delta}\right)^{k-1} f \pm\left(T^{\pi}\right)^{k-2} T^{\delta} f \tag{B.8}$$

$$=\left(\left(T^{\pi}\right)^{k-2} T^{\pi} f-\left(T^{\pi}\right)^{k-2} T^{\delta} f\right)+\left(\left(T^{\pi}\right)^{k-2} T^{\delta} f-\left(T^{\delta}\right)^{k-2} T^{\delta} f\right)$$

$$=\gamma^{k-2}\left(P^{\pi}\right)^{k-2}\left(T^{\pi} f-T^{\delta} f\right)+\left(\left(T^{\pi}\right)^{k-2} T^{\delta} f-\left(T^{\delta}\right)^{k-2} T^{\delta} f\right) \tag{B.9}$$

$$=\gamma^{k-1}\left(P^{\pi}\right)^{k-2}\left(p_{\pi}-p_{\delta}\right) f+\sum_{i=0}^{k-3} \gamma^{i+1}\left(p_{\pi}\right)^{i}\left(p_{\pi}-p_{\delta}\right)\left(T^{\delta}\right)^{k-3-i} T^{\delta} f \tag{B.10}$$

$$=\sum_{i=0}^{k-2} \gamma^{i+1}\left(p_{\pi}\right)^{i}\left(p_{\pi}-p_{\delta}\right)\left(T^{\delta}\right)^{k-2-i} f, \tag{B.11}$$

where in line (B.9) we exploited the identity at equation (B.4), line (B.10) derives from observing that $T^{\pi} f-T^{\delta} f=\gamma\left(p_{\pi}-p_{\delta}\right) f$ and by inductive hypothesis applied on $T^{\delta} f$ which is a bounded measurable function as well. Finally, line (B.11) follows from observing that the first term completes the summation up to $k-2$. The second identity in the statement can be obtained by an analogous derivation in which at line (B.8) we sum and subtract $\left(T^{\delta}\right)^{k-2} T^{\pi} f$ and, later, exploit the identity at equation (B.7). □

Lemma B.3 (Persistence Lemma). *Let $\mathcal{M}$ be an MDP and $\pi \in \Pi$ be a Markovian stationary policy, then for any $k \in \mathbb{N}^{+}$ the following two identities hold:*

$$\begin{aligned}Q^{\pi}-Q_{k}^{\pi} &=\sum_{\substack{i \in \mathbb{N} \\ i \bmod k \neq 0}} \gamma^{i}\left(p_{\pi}\right)^{i-1}\left(p_{\pi}-p_{\delta}\right)\left(T^{\delta}\right)^{k-2-(i-1) \bmod k} T^{\pi} Q_{k}^{\pi} \\ &=\sum_{\substack{i \in \mathbb{N} \\ i \bmod k \neq 0}} \gamma^{i}\left(\left(p_{\delta}\right)^{k-1} p_{\pi}\right)^{i \operatorname{div} k}\left(p_{\delta}\right)^{i \bmod k-1}\left(p_{\pi}-p_{\delta}\right)\left(T^{\pi}\right)^{k-i \bmod k} Q^{\pi},\end{aligned}$$

where for two non-negative integers $a, b \in \mathbb{N}$, we denote with $a \bmod b$ and $a \operatorname{div} b$ the remainder and the quotient of the integer division between a and b respectively.

Proof. We start proving the first identity. Let us consider the first identity of Lemma B.1:

$$
\begin{aligned}
Q^\pi - Q_k^\pi &= \left(\mathrm{Id} - \gamma^k \left(p_\pi\right)^k\right)^{-1} \left(\left(T^\pi\right)^k Q_k^\pi - \left(T^\delta\right)^{k-1} T^\pi Q_k^\pi\right) \\
&= \left(\sum_{j=0}^{+\infty} \gamma^{kj} \left(p_\pi\right)^{kj}\right) \left(\left(T^\pi\right)^k Q_k^\pi - \left(T^\delta\right)^{k-1} T^\pi Q_k^\pi\right) && \text{(B.12)} \\
&= \left(\sum_{j=0}^{+\infty} \gamma^{kj} \left(p_\pi\right)^{kj}\right) \sum_{l=0}^{k-2} \gamma^{l+1} \left(p_\pi\right)^l \left(p_\pi - p_\delta\right) \left(T^\delta\right)^{k-2-l} T^\pi Q_k^\pi && \text{(B.13)} \\
&= \sum_{j=0}^{+\infty} \gamma^{kj} \left(p_\pi\right)^{kj} \sum_{l=0}^{k-2} \gamma^{l+1} \left(p_\pi\right)^l \left(p_\pi - p_\delta\right) \left(T^\delta\right)^{k-2-l} T^\pi Q_k^\pi \\
&= \sum_{j=0}^{+\infty} \sum_{l=0}^{k-2} \gamma^{kj+l+1} \left(p_\pi\right)^{kj+l} \left(p_\pi - p_\delta\right) \left(T^\delta\right)^{k-2-l} T^\pi Q_k^\pi,
\end{aligned}
$$

where line (B.12) follows from applying the Neumann series at the first factor, line (B.13) is obtained by applying the first identity of Lemma B.2 to the bounded measurable function $T^\pi Q_k^\pi$. The subsequent lines are obtained by straightforward algebraic manipulations. Now we rename the indexes by setting $i = kj + l + 1$. Since $l \in \{0, \dots, k-2\}$ we have that $j = (i-1) \operatorname{div} k$ and $l = (i-1) \bmod k$. Moreover, we observe that i ranges over all non-negative integers values except for the multiples of the persistence k, i.e., $i \in \{n \in \mathbb{N} : n \bmod k \neq 0\}$. Now, recalling that $i \bmod k \neq 0$, we observe that for the distributive property of the modulo operator, we have $(i-1) \bmod k = (i \bmod k - 1 \bmod k) \bmod k = (i \bmod k - 1) \bmod k = i \bmod k - 1$. The second identity is obtained by an analogous derivation in which we exploit the second identities at Lemmas B.1 and B.2. $\square$

Proof of Persistence Bound (Theorem 6.1)

Theorem 6.1. *Let $\mathcal{M}$ be an MDP and $\pi \in \Pi$ be a Markovian stationary policy. Let $\mathcal{Q}_k = \{\left(T^\delta\right)^{k-2-l} T^\pi Q_k^\pi : l \in \{0, \dots, k-2\}\}$ and for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ let us define:*

$$
d_{\mathcal{Q}_k}^\pi(s,a) = \sup_{f \in \mathcal{Q}_k} \left| \int_{\mathcal{S}} \int_{\mathcal{A}} \left(p_\pi(\mathrm{d}s', \mathrm{d}a'|s,a) - p_\delta(\mathrm{d}s', \mathrm{d}a'|s,a)\right) f(s',a') \right|
$$

Then, for any $\mu \in \Delta_{\mathcal{S}\times\mathcal{A}}$, $p \geq 1$, and $k \in \mathbb{N}^+$, it holds that:

$$
\|Q^\pi - Q_k^\pi\|_{p,\mu} \leq \frac{\gamma(1-\gamma^{k-1})}{(1-\gamma)(1-\gamma^k)} \left\|d_{\mathcal{Q}_k}^\pi\right\|_{p,\eta_k^{\mu,\pi}},
$$

where $\eta_k^{\mu,\pi} \in \Delta_{\mathcal{S}\times\mathcal{A}}$ is a probability measure defined for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ as:

$$
\eta_k^{\mu,\pi}(s,a) = \frac{(1-\gamma)(1-\gamma^k)}{\gamma(1-\gamma^{k-1})} \sum_{\substack{i \in \mathbb{N} \\ i \bmod k \neq 0}} \gamma^i \left(\mu \left(p_\pi\right)^{i-1}\right)(s,a).
$$

Proof. We start from the first equality derived in Lemma B.3, and we apply the $L_p(\mu)$-norm both sides, with $p \geq 1$:

$$\|Q^\pi - Q_k^\pi\|_{p,\mu}^p = \left\| \sum_{\substack{i \in \mathbb{N} \\ i \bmod k \neq 0}} \gamma^i (p_\pi)^{i-1} (p_\pi - p_\delta) \left(T^\delta\right)^{k-2-(i-1) \bmod k} T^\pi Q_k^\pi \right\|_{p,\mu}^p$$

$$= \mu \left| \sum_{\substack{i \in \mathbb{N} \\ i \bmod k \neq 0}} \gamma^i (p_\pi)^{i-1} (p_\pi - p_\delta) \left(T^\delta\right)^{k-2-(i-1) \bmod k} T^\pi Q_k^\pi \right|^p$$

$$\leq \mu \left| \sum_{\substack{i \in \mathbb{N} \\ i \bmod k \neq 0}} \gamma^i (p_\pi)^{i-1} \sup_{f \in \mathcal{Q}_k} |(p_\pi - p_\delta) f| \right|^p \tag{B.14}$$

$$= \left(\frac{\gamma(1-\gamma^{k-1})}{(1-\gamma)(1-\gamma^k)} \right)^p \mu \left| \frac{(1-\gamma)(1-\gamma^k)}{\gamma(1-\gamma^{k-1})} \sum_{\substack{i \in \mathbb{N} \\ i \bmod k \neq 0}} \gamma^i (p_\pi)^{i-1} d_{\mathcal{Q}_k}^\pi \right|^p \tag{B.15}$$

$$\leq \left(\frac{\gamma(1-\gamma^{k-1})}{(1-\gamma)(1-\gamma^k)} \right)^p \frac{(1-\gamma)(1-\gamma^k)}{\gamma(1-\gamma^{k-1})} \mu \sum_{\substack{i \in \mathbb{N} \\ i \bmod k \neq 0}} \gamma^i (p_\pi)^{i-1} \left| d_{\mathcal{Q}_k}^\pi \right|^p \tag{B.16}$$

$$= \left(\frac{\gamma(1-\gamma^{k-1})}{(1-\gamma)(1-\gamma^k)} \right)^p \eta_k^{\mu,\pi} \left| d_{\mathcal{Q}_k}^\pi \right|^p \tag{B.17}$$

$$= \left(\frac{\gamma(1-\gamma^{k-1})}{(1-\gamma)(1-\gamma^k)} \right)^p \left\| d_{\mathcal{Q}_k}^\pi \right\|_{p,\eta^{\mu,\pi}}^p . \tag{B.18}$$

where line (B.14) is obtained by bounding $(p_\pi - p_\delta)\left(T^\delta\right)^x \leq \sup_{f \in \mathcal{Q}_k} |(p_\pi - p_\delta) f|$, recalling the definition of $\mathcal{Q}_k$ and that $x = (i-1) \bmod k \leq k-2$ for all $i \in \mathbb{N}$ and $i \bmod k \neq 0$. Then, line (B.15) follows from deriving the normalization constant in order to make the summation $\sum_{\substack{i \in \mathbb{N} \\ i \bmod k \neq 0}} \gamma^i (p_\pi)^{i-1}$ a proper probability distribution. Such a constant can be obtained as follows:

$$\sum_{\substack{i \in \mathbb{N} \\ i \bmod k \neq 0}} \gamma^i = \sum_{i \in \mathbb{N}} \gamma^i - \sum_{i \in \mathbb{N}} \gamma^{ki} = \frac{\gamma(1-\gamma^{k-1})}{(1-\gamma)(1-\gamma^k)}.$$

Line (B.16) is obtained by applying Jensen inequality recalling that $p \geq 1$. Finally, line (B.17) derives from the definition of the distribution $\eta_k^{\mu,\pi}$ and line (B.18) from the definition of $L_p(\eta_k^{\mu,\pi})$-norm. □

Lemma B.4. *Let $\mathcal{M}$ be an MDP and $\pi \in \Pi$ be a Markovian stationary policy. Let $f \in \mathscr{B}(\mathcal{S} \times \mathcal{A})$ that is L_f–LC. Then, under Assumptions 3.1 and 3.2, the following statements hold:*

i) $T^\pi f$ is $(L_r + \gamma L_P (L_\pi + 1) L_f)$–LC;

ii) $T^\delta f$ is $(L_r + \gamma (L_P + 1) L_f)$–LC;

iii) $T^\star f$ is $(L_r + \gamma L_P L_f)$–LC.

Proof. Let $f \in \mathscr{B}(\mathcal{S} \times \mathcal{A})$ be L_f-LC. Consider an application of T^π and $(s,a), (\overline{s}, \overline{a}) \in \mathcal{S} \times \mathcal{A}$:

$$\begin{aligned}
|(T^\pi f)(s,a) - (T^\pi f)(\overline{s},\overline{a})| = {} & |r(s,a) + \gamma \int_{\mathcal{S}} \int_{\mathcal{A}} P(\,\mathrm{d}s'|s,a) \pi(\,\mathrm{d}a'|s') f(s',a') \\
& - r(\overline{s},\overline{a}) - \gamma \int_{\mathcal{S}} \int_{\mathcal{A}} P(\,\mathrm{d}s'|\overline{s},\overline{a}) \pi(\,\mathrm{d}a'|s') f(s',a')| \\
\le {} & |r(s,a) - r(\overline{s},\overline{a})| \\
& + \gamma \left| \int_{\mathcal{S}} (P(\,\mathrm{d}s'|s,a) - P(\,\mathrm{d}s'|\overline{s},\overline{a})) \int_{\mathcal{A}} \pi(\,\mathrm{d}a'|s') f(s',a') \right| \qquad \text{(B.19)} \\
\le {} & |r(s,a) - r(\overline{s},\overline{a})| \\
& + \gamma (L_\pi + 1) L_f \sup_{f : \|f\|_L \le 1} \left| \int_{\mathcal{S}} (P(\,\mathrm{d}s'|s,a) - P(\,\mathrm{d}s'|\overline{s},\overline{a}))\, f(s') \right| \qquad \text{(B.20)} \\
\le {} & (L_r + \gamma L_P (L_\pi + 1) L_f)\, d_{\mathcal{S} \times \mathcal{A}} \left((s,a), (\overline{s},\overline{a}) \right), \qquad \text{(B.21)}
\end{aligned}$$

where line (B.19) follows from triangular inequality, line (B.20) is obtained from observing that the function $g_f(s') = \int_{\mathcal{A}} \pi(\,\mathrm{d}a'|s') f(s',a')$ is $(L_\pi + 1)L_f$–LC, since for any $s, \overline{s} \in \mathcal{S}$:

$$\begin{aligned}
|g_f(s) - g_f(\overline{s})| = & \left| \int_{\mathcal{A}} \pi(\,\mathrm{d}a|s) f(s,a) - \int_{\mathcal{A}} \pi(\,\mathrm{d}a|\overline{s}) f(\overline{s},a) \right| \\
= & \left| \int_{\mathcal{A}} \pi(\,\mathrm{d}a|s) f(s,a) - \int_{\mathcal{A}} \pi(\,\mathrm{d}a|\overline{s}) f(\overline{s},a) \pm \int_{\mathcal{A}} \pi(\,\mathrm{d}a|\overline{s}) f(s,a) \right| \\
\le & \left| \int_{\mathcal{A}} (\pi(\,\mathrm{d}a|s) - \pi(\,\mathrm{d}a|\overline{s}))\, f(s,a) \right| + \left| \int_{\mathcal{A}} \pi(\,\mathrm{d}a|\overline{s})\, (f(\overline{s},a) - f(s,a)) \right| \\
\le & L_f \sup_{f : \|f\|_L \le 1} \left| \int_{\mathcal{A}} (\pi(\,\mathrm{d}a|s) - \pi(\,\mathrm{d}a|\overline{s}))\, f(a) \right| \\
& \quad + \left| \int_{\mathcal{A}} \pi(\,\mathrm{d}a|\overline{s})\, (f(\overline{s},a) - f(s,a)) \right| \\
\le & L_f L_\pi d_{\mathcal{S}}(s,\overline{s}) + L_f d_{\mathcal{S}}(s,\overline{s}),
\end{aligned}$$

where we exploited the fact that L_π–LC. Finally, line (B.21) is obtained by recalling that the reward function is L_r–LC and the transition model is L_P–LC. The derivations are analogous for T^δ and $T^\star$. Concerning T^δ we have:

$$\begin{aligned}
|(T^\delta f)(s,a) - (T^\delta f)(\overline{s},\overline{a})| \le & |r(s,a) - r(\overline{s},\overline{a})| \\
& + \gamma \left| \int_{\mathcal{S}} \int_{\mathcal{A}} \left(\delta_a(\mathrm{d}a') P(\mathrm{d}s'|s,a) - \delta_{\overline{a}}(\mathrm{d}a') P(\mathrm{d}s'|\overline{s},\overline{a}) \right) f(s',a') \right| \\
\le & L_r d_{\mathcal{S}\times\mathcal{A}} \left((s,a), (\overline{s},\overline{a}) \right) \\
& + \gamma \left| \int_{\mathcal{S}} \left(P(\mathrm{d}s'|s,a) - P(\mathrm{d}s'|\overline{s},\overline{a}) \right) \int_{\mathcal{A}} \delta_a(\mathrm{d}a') f(s',a') \right| \\
& + \gamma \int_{\mathcal{S}} P(\mathrm{d}s'|\overline{s},\overline{a}) \left| \int_{\mathcal{A}} \left(\delta_a(\mathrm{d}a') - \delta_{\overline{a}}(\mathrm{d}a') \right) f(s',a') \right| \\
\le & \left(L_r + \gamma L_f L_P + \gamma L_f \right) d_{\mathcal{S}\times\mathcal{A}} \left((s,a), (\overline{s},\overline{a}) \right),
\end{aligned}$$

where we observed that $\int_{\mathcal{A}} \delta_a(\mathrm{d}a') f(s',a') = f(s',a)$ is L_f–LC and that $\int_{\mathcal{A}} |\delta_a(\mathrm{d}a') - \delta_{\overline{a}}(\mathrm{d}a')| f(s',a') \le L_f d_{\mathcal{A}}(a,\overline{a}) \le L_f d_{\mathcal{S}\times\mathcal{A}}((s,a),(\overline{a},\overline{a}))$.
Finally, considering $T^\star$, we have:

$$\begin{aligned}
|(T^\star f)(s,a) - (T^\star f)(\overline{s},\overline{a})| \le & |r(s,a) - r(\overline{s},\overline{a})| \\
& + \gamma \left| \int_{\mathcal{S}} \left(P(\mathrm{d}s'|s,a) - P(\mathrm{d}s'|\overline{s},\overline{a}) \right) \max_{a' \in As} f(s',a') \right| \\
\le & \left(L_r + \gamma L_f L_P \right) d_{\mathcal{S}\times\mathcal{A}} \left((s,a), (\overline{s},\overline{a}) \right),
\end{aligned}$$

where we observed that the function $h_f(s') = \max_{a' \in As} f(s',a')$ is L_f–LC, since:

$$\begin{aligned}
|h_f(s) - h_f(\overline{s})| &= \left| \max_{a' \in As} f(s,a') - \max_{a' \in As} f(\overline{s},a') \right| \\
&\le \max_{a' \in \mathcal{A}} |f(s,a') - f(\overline{s},a')| \\
&\le L_f d_{\mathcal{S}}(s,\overline{s}).
\end{aligned}$$

□

Lemma B.5. *Let $\mathcal{M}$ be an MDP and $\pi \in \Pi$ be a Markovian stationary policy. Then, under Assumptions 3.1 and 3.2, if $\gamma \max\{L_P + 1, L_P(L_\pi + 1)\} < 1$, the functions $f \in \mathcal{Q}_k$ are $L_{\mathcal{Q}_k}$–LC, where:*

$$L_{\mathcal{Q}_k} \le \frac{L_r}{1 - \gamma \max\{L_P + 1, L_P(L_\pi + 1)\}}. \tag{B.22}$$

Furthermore, for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ it holds that:

$$d_{\mathcal{Q}_k}(s,a) \le L_{\mathcal{Q}_k} \mathcal{W}_1 \left(p_\pi(\cdot|s,a), p_\delta(\cdot|s,a) \right). \tag{B.23}$$

Proof. First of all consider the action-value function of the k–persistent MDP Q_k^π, which is the fixed point of the operator T_k^π that decomposes into $(T^\delta)^{k-1} T^\pi$ according to Theorem 6.8. It follows that for any $f \in \mathscr{B}(\mathcal{S} \times \mathcal{A})$ we have:

$$Q_k^\pi = \lim_{j \to +\infty} (T_k^\pi)^j f = \lim_{j \to +\infty} \left((T^\delta)^{k-1} T^\pi \right)^j f.$$

We now want to bound the Lipschitz constant of Q_k^π. To this purpose, let us first compute the Lipschitz constant of $T_k^\pi f = ((T^\delta)^{k-1} T^\pi) f$ for $f \in \mathscr{B}(\mathcal{S} \times \mathcal{A})$ being an L_f–LC function. From Lemma B.4 we can bound the Lipschitz constant a_h of $(T^\delta)^h T^\pi f$ for $h \in \{0, ...k-1\}$, leading to the sequence:

$$a_h = \begin{cases} L_r + \gamma L_P (L_\pi + 1) L_f & \text{if } h = 0 \\ L_r + \gamma (L_P + 1) a_{h-1} & \text{if } h \in \{1, ...k-1\} \end{cases}.$$

Thus, the Lipschitz constant of $((T^\delta)^{k-1} T^\pi) f$ is a_{k-1}. By unrolling the recursion we have:

$$\begin{aligned} a_{k-1} &= L_r \sum_{i=0}^{k-1} \gamma^i (L_P + 1)^i + \gamma^k L_P (L_\pi + 1)(L_P + 1)^{k-1} L_f \\ &= L_r \frac{1 - \gamma^k (L_P + 1)^k}{1 - \gamma (L_P + 1)} + \gamma^k L_P (L_\pi + 1)(L_P + 1)^{k-1} L_f. \end{aligned}$$

Let us now consider the sequence b_j of the Lipschitz constants of $(T_k^\pi)^j f$ for $j \in \mathbb{N}$:

$$b_j = \begin{cases} L_f & \text{if } j = 0 \\ L_r \frac{1 - \gamma^k (L_P + 1)^k}{1 - \gamma (L_P + 1)} + \gamma^k L_P (L_\pi + 1)(L_P + 1)^{k-1} b_{j-1} & \text{if } j \in \mathbb{N}^+ \end{cases}.$$

The sequence b_j converges to a finite limit as long as $\gamma^k L_P (L_\pi + 1)(L_P + 1)^{k-1} < 1$. In such case, the limit b_∞ can be computed solving the fixed point equation:

$$\begin{aligned} b_\infty =& L_r \frac{1 - \gamma^k (L_P + 1)^k}{1 - \gamma (L_P + 1)} + \gamma^k L_P (L_\pi + 1)(L_P + 1)^{k-1} b_\infty \\ \implies b_\infty =& \frac{L_r \left(1 - \gamma^k (L_P + 1)^k\right)}{\left(1 - \gamma (L_P + 1)\right)\left(1 - \gamma^k L_P (L_\pi + 1)(L_P + 1)^{k-1}\right)}. \end{aligned}$$

Thus, b_∞ represents the Lipschitz constant of Q_k^π. It is worth noting that when setting $k = 1$ we recover the Lipschitz constant of the Q^π as in (Rachelson and Lagoudakis, 2010). To get a bound that is independent on k we define $L = \max\{L_P (L_\pi + 1), L_P + 1\}$, assuming that $\gamma L < 1$ so that:

$$b_\infty = \frac{L_r \left(1 - \gamma^k (L_P + 1)^k\right)}{\left(1 - \gamma (L_P + 1)\right)\left(1 - \gamma^k L_P (L_\pi + 1)(L_P + 1)^{k-1}\right)} \le \frac{L_r}{1 - \gamma L},$$

having observed that $\frac{1 - \gamma^k (L_P + 1)^k}{1 - \gamma (L_P + 1)} \le \frac{1 - \gamma^k L^k}{1 - \gamma L}$. Thus, we conclude that Q_k^π is also $\frac{L_r}{1 - \gamma L}$–LC for any $k \in \mathbb{N}^+$. Consider now the application of the operator T^π to Q_k^π, we have that the corresponding Lipschitz constant can be bounded by:

$$L_{T^\pi Q_k^\pi} \le L_r + \gamma L_P (L_\pi + 1) \frac{L_r}{1 - \gamma L} \le L_r + \gamma L \frac{L_r}{1 - \gamma L} = \frac{L_r}{1 - \gamma L}. \tag{B.24}$$

A similar derivation holds for the application of T^δ. As a consequence, any arbitrary sequence of applications of T^π and T^δ to Q_k^π generates a sequence of $\frac{L_r}{1 - \gamma L}$–LC functions. Even more so for the functions in the set $\mathcal{Q}_k = \left\{ \left(T^\delta\right)^{k-2-l} T^\pi Q_k^\pi \, : \, l \in\right.$

$\{0, \dots, k-2\}\}$. As a consequence, we can rephrase the dissimilarity term $d^{\pi}_{\mathcal{Q}_k}(s,a)$ as a Kantorovich distance:

$$\begin{aligned}
d^{\pi}_{\mathcal{Q}_k}(s,a) &= \sup_{f \in \mathcal{Q}_k} \left| \int_{\mathcal{S}} \int_{\mathcal{A}} \left(p_\pi(\,\mathrm{d}s',\,\mathrm{d}a'|s,a) - p_\delta(\,\mathrm{d}s',\,\mathrm{d}a'|s,a)\right) f(s',a') \right| \\
&\le L_{\mathcal{Q}_k} \sup_{f:\|f\|_L \le 1} \left| \int_{\mathcal{S}} \int_{\mathcal{A}} \left(p_\pi(\,\mathrm{d}s',\,\mathrm{d}a'|s,a) - p_\delta(\,\mathrm{d}s',\,\mathrm{d}a'|s,a)\right) f(s',a') \right| \\
&= L_{\mathcal{Q}_k} \mathcal{W}_1\left(p_\pi(\cdot|s,a), p_\delta(\cdot|s,a)\right).
\end{aligned}$$

□

Theorem 6.2. *Let $\mathcal{M}$ be an MDP and $\pi \in \Pi$ be a Markovian stationary policy. Under Assumptions 3.1, 3.2, and 6.1, if $\gamma \max\{L_P + 1, L_P(1+L_\pi)\} < 1$ and if $\mu(s,a) = \mu_{\mathcal{S}}(s)\pi(a|s)$ with $\mu_{\mathcal{S}} \in \Delta_{\mathcal{S}}$, then for any $k \in \mathbb{N}^+$:*

$$\left\| d^{\pi}_{\mathcal{Q}_k} \right\|_{p, \eta_k^{\mu,\pi}} \le L_{\mathcal{Q}_k} \left[(L_\pi + 1) L_T + \sigma_p \right].$$

where:

$$\begin{aligned}
\sigma_p^p &= \sup_{s \in \mathcal{S}} \int_{\mathcal{A}} \int_{\mathcal{A}} d_{\mathcal{A}}(a,a')^p \, \pi(\,\mathrm{d}a|s) \pi(\,\mathrm{d}a'|s), \\
L_{\mathcal{Q}_k} &= \frac{L_r}{1 - \gamma \max\{L_P + 1, L_P(1+L_\pi)\}}.
\end{aligned}$$

Proof. Let us now consider the dissimilarity term in norm:

$$\begin{aligned}
&\left\| d^{\pi}_{\mathcal{Q}_k} \right\|^p_{p, \eta_k^{\mu,\pi}} \\
&= \mathop{\mathbb{E}}_{(s,a)\sim\eta_k^{\mu,\pi}} \left[\left| \sup_{f \in \mathcal{Q}_k} \left| \int_{\mathcal{S}} \int_{\mathcal{A}} \left(p_\pi(\,\mathrm{d}s',\,\mathrm{d}a'|s,a) - p_\delta(\,\mathrm{d}s',\,\mathrm{d}a'|s,a)\right) f(s',a') \right| \right|^p \right] \\
&\le L^p_{\mathcal{Q}_k} \mathop{\mathbb{E}}_{(s,a)\sim\eta_k^{\mu,\pi}} \left[\left| \sup_{f:\|f\|_L \le 1} \left| \int_{\mathcal{S}} \int_{\mathcal{A}} \left(p_\pi(\,\mathrm{d}s',\,\mathrm{d}a'|s,a) - p_\delta(\,\mathrm{d}s',\,\mathrm{d}a'|s,a)\right) f(s',a') \right| \right|^p \right]
\end{aligned}$$

where the inequality follows from Lemma B.5. We now consider the inner term and perform the following algebraic manipulations:

$$\begin{aligned}
&\sup_{f:\|f\|_L \le 1} \left| \int_{\mathcal{S}} \int_{\mathcal{A}} \left(P^\pi(\,\mathrm{d}s',\,\mathrm{d}a'|s,a) - p_\delta(\,\mathrm{d}s',\,\mathrm{d}a'|s,a)\right) f(s',a') \right| \\
&\quad = \sup_{f:\|f\|_L \le 1} \left| \int_{\mathcal{S}} \int_{\mathcal{A}} P(\,\mathrm{d}s'|s,a)\pi(\,\mathrm{d}a'|s') f(s',a') - \int_{\mathcal{S}} \int_{\mathcal{A}} P(\,\mathrm{d}s'|s,a)\delta_a(\,\mathrm{d}a') f(s',a') \right. \\
&\qquad \left. \pm \int_{\mathcal{S}} \int_{\mathcal{A}} \delta_s(\,\mathrm{d}s')\pi(\,\mathrm{d}a'|s') \pm \int_{\mathcal{S}} \int_{\mathcal{A}} \delta_s(\,\mathrm{d}s')\delta_a(\,\mathrm{d}a') f(s',a') \right| \\
&\quad \le \sup_{f:\|f\|_L \le 1} \left| \int_{\mathcal{S}} \left(P(\,\mathrm{d}s'|s,a) - \delta_s(\,\mathrm{d}s')\right) \int_{\mathcal{A}} \pi(\,\mathrm{d}a'|s') f(s',a') \right| \\
&\qquad + \sup_{f:\|f\|_L \le 1} \left| \int_{\mathcal{S}} \left(P(\,\mathrm{d}s'|s,a) - \delta_s(\,\mathrm{d}s')\right) \int_{\mathcal{A}} \delta_a(\,\mathrm{d}a') f(s',a') \right| \\
&\qquad + \sup_{f:\|f\|_L \le 1} \left| \int_{\mathcal{S}} \delta_s(\,\mathrm{d}s') \int_{\mathcal{A}} \left(\pi(\,\mathrm{d}a'|s') - \delta_a(\,\mathrm{d}a')\right) f(s',a') \right|.
\end{aligned}$$

We now consider the first two terms:

$$\begin{aligned}
&\sup_{f:\|f\|_L \le 1} \left| \int_{\mathcal{S}} (P(\,\mathrm{d}s'|s,a) - \delta_s(\,\mathrm{d}s')) \int_{\mathcal{A}} \pi(\,\mathrm{d}a'|s') f(s',a') \right| \\
&\qquad + \sup_{f:\|f\|_L \le 1} \left| \int_{\mathcal{S}} (P(\,\mathrm{d}s'|s,a) - \delta_s(\,\mathrm{d}s')) \int_{\mathcal{A}} \delta_a(\,\mathrm{d}a') f(s',a') \right| \\
&\qquad \le (L_\pi + 1) \mathcal{W}_1 \left(P(\cdot|s,a), \delta_s\right) && \text{(B.25)} \\
&\qquad \le (L_\pi + 1) L_T,
\end{aligned}$$

where line (B.25) follows from observing that the function $g_f(s') = \int_{\mathcal{A}} \pi(\,\mathrm{d}a'|s') f(s',a')$ is L_π-LC, and function $h_f(s') = \int_{\mathcal{A}} \delta_a(\,\mathrm{d}a') f(s',a') = f(s',a)$ is 1-LC. Moreover, under Assumption 6.1, we have that $\mathcal{W}_1 \left(P(\cdot|s,a), \delta_s\right) \le L_T$. Let us now focus on the third term:

$$\begin{aligned}
&\sup_{f:\|f\|_L \le 1} \left| \int_{\mathcal{S}} \delta_s(\,\mathrm{d}s') \int_{\mathcal{A}} \left(\pi(\,\mathrm{d}a'|s') - \delta_a(\,\mathrm{d}a')\right) f(s',a') \right| \\
&\qquad = \sup_{f:\|f\|_L \le 1} \left| \int_{\mathcal{A}} \left(\pi(\,\mathrm{d}a'|s) - \delta_a(\,\mathrm{d}a')\right) f(s,a') \right| \\
&\qquad = \sup_{f:\|f\|_L \le 1} \left| \int_{\mathcal{A}} \left(\pi(\,\mathrm{d}a'|s) - \delta_a(\,\mathrm{d}a')\right) f(a') \right| && \text{(B.26)} \\
&\qquad = \sup_{f:\|f\|_L \le 1} \left| \int_{\mathcal{A}} \left(\int_{\mathcal{A}} \pi(\,\mathrm{d}a''|s) \delta_{a'}(\,\mathrm{d}a'') - \delta_a(\,\mathrm{d}a') \right) f(a') \right| && \text{(B.27)} \\
&\qquad = \sup_{f:\|f\|_L \le 1} \left| \int_{\mathcal{A}} \pi(\,\mathrm{d}a''|s) \int_{\mathcal{A}} \left(\delta_{a''}(\,\mathrm{d}a') - \delta_a(\,\mathrm{d}a')\right) f(a') \right| && \text{(B.28)} \\
&\qquad \le \int_{\mathcal{A}} \pi(\,\mathrm{d}a''|s) \sup_{f:\|f\|_L \le 1} \left| \int_{\mathcal{A}} \left(\delta_{a''}(\,\mathrm{d}a') - \delta_a(\,\mathrm{d}a')\right) f(a') \right| && \text{(B.29)} \\
&\qquad = \int_{\mathcal{A}} \pi(\,\mathrm{d}a''|s) d_{\mathcal{A}}(a,a''), && \text{(B.30)}
\end{aligned}$$

where line (B.26) follows from observing that the dependence on s for function f can be neglected because of the supremum, line (B.27) is obtained from the equality $\pi(\,\mathrm{d}a'|s) = \int_{\mathcal{A}} \pi(\,\mathrm{d}a''|s) \delta_{a'}(\,\mathrm{d}a'')$, line (B.28) derives from moving the integral over a'' outside and recalling that $\delta_{a''}(\,\mathrm{d}a') = \delta_{a'}(\,\mathrm{d}a'')$, line (B.29) comes from Jensen inequality. Finally, line (B.30) is obtained from the definition of Kantorovich distance between Dirac deltas. Now, we take the expectation w.r.t. $\eta_k^{\mu,\pi}$. Recalling that $\mu(s,a) = \mu_{\mathcal{S}}(s)\pi(a|s)$ it follows that the same decomposition holds for $\eta_k^{\mu,\pi}(s,a) = \eta_{k,\mathcal{S}}^{\mu,\pi}(s)\pi(a|s)$. Consequently, exploiting the above equation, we have:

$$\begin{aligned}
&\int_{\mathcal{S}} \eta_{k,\mathcal{S}}^{\mu,\pi}(\,\mathrm{d}s) \int_{\mathcal{A}} \pi(\,\mathrm{d}a|s) \left| \int_{\mathcal{A}} \pi(\,\mathrm{d}a''|s) d_{\mathcal{A}}(a,a'') \right|^p \\
&\le \int_{\mathcal{S}} (\eta_k^{\mu,\pi})_{\mathcal{S}}(\,\mathrm{d}s) \int_{\mathcal{A}} \pi(\,\mathrm{d}a|s) \int_{\mathcal{A}} \pi(\,\mathrm{d}a''|s) d_{\mathcal{A}}(a,a'')^p \\
&\le \sup_{s \in \mathcal{S}} \int_{\mathcal{A}} \int_{\mathcal{A}} \pi(\,\mathrm{d}a|s) \pi(\,\mathrm{d}a''|s) d_{\mathcal{A}}(a,a'')^p = \sigma_p^p,
\end{aligned}$$

where the first inequality follows from an application of Jensen inequality. An application of Minkowski inequality on the norm $\left\| d^{\pi}_{\mathcal{Q}_k} \right\|_{p,\eta_k^{\mu,\pi}}$ concludes the proof. □

Lemma B.6. *If $\mathcal{A} = \mathbb{R}^{d_{\mathcal{A}}}$, $d_{\mathcal{A}}(\mathbf{a}, \mathbf{a}') = \|\mathbf{a} - \mathbf{a}'\|_2$, then it holds that $\sigma_2^2 \le 2 \sup_{s \in \mathcal{S}} \mathbb{V}ar[A]$, with $A \sim \pi(\cdot|s)$.*

Proof. Let $s \in \mathcal{S}$ and define the mean–action in state s as:

$$\overline{\mathbf{a}}(s) = \int_{\mathcal{A}} \mathbf{a} \pi(\,\mathrm{d}\mathbf{a}|s).$$

Thus, we have:

$$\begin{aligned}
\sigma_2^2 &= \sup_{s \in \mathcal{S}} \int_{\mathcal{A}} \int_{\mathcal{A}} \|\mathbf{a} - \mathbf{a}'\|_2^2 \, \pi(\,\mathrm{d}\mathbf{a}|s)\pi(\,\mathrm{d}\mathbf{a}'|s) \\
&= \sup_{s \in \mathcal{S}} \int_{\mathcal{A}} \int_{\mathcal{A}} \|\mathbf{a} - \mathbf{a}' \pm \overline{\mathbf{a}}(s)\|_2^2 \, \pi(\,\mathrm{d}\mathbf{a}|s)\pi(\,\mathrm{d}\mathbf{a}'|s) \\
&\le \sup_{s \in \mathcal{S}} \int_{\mathcal{A}} \int_{\mathcal{A}} \|\mathbf{a} - \overline{\mathbf{a}}(s)\|_2^2 \, \pi(\,\mathrm{d}\mathbf{a}|s)\pi(\,\mathrm{d}\mathbf{a}'|s) \\
&\quad + \sup_{s \in \mathcal{S}} \int_{\mathcal{A}} \int_{\mathcal{A}} \|\mathbf{a}' - \overline{\mathbf{a}}(s)\|_2^2 \, \pi(\,\mathrm{d}\mathbf{a}|s)\pi(\,\mathrm{d}\mathbf{a}'|s) \\
&= \sup_{s \in \mathcal{S}} \int_{\mathcal{A}} \|\mathbf{a} - \overline{\mathbf{a}}(s)\|_2^2 \, \pi(\,\mathrm{d}\mathbf{a}|s) + \sup_{s \in \mathcal{S}} \int_{\mathcal{A}} \|\mathbf{a}' - \overline{\mathbf{a}}(s)\|_2^2 \, \pi(\,\mathrm{d}\mathbf{a}'|s) \\
&= 2 \sup_{s \in \mathcal{S}} \int_{\mathcal{A}} \|\mathbf{a} - \overline{\mathbf{a}}(s)\|_2^2 \, \pi(\,\mathrm{d}\mathbf{a}|s) = 2 \sup_{s \in \mathcal{S}} \mathbb{V}\mathrm{ar}[A].
\end{aligned}$$

□

Remark B.7 (On the choice of $d_{\mathcal{A}}$ when $|\mathcal{A}| < +\infty$)**.** *When the action space $\mathcal{A}$ is finite but is a subset of a metric space, e.g., $\mathbb{R}^{d_{\mathcal{A}}}$ we can employ the same metric as $d_{\mathcal{A}}$. Otherwise, we use the* discrete metric $d_{\mathcal{A}}(a, a') = \mathbb{1}\{a \neq a'\}$.

B.1.2 Proofs of Section 6.6

Proposition B.1. *Assuming that the evaluation of the estimated Q-function in a state action pair has computational complexity $\mathcal{O}(1)$, the computational complexity of J iterations of PFQI(k) run with a dataset $\mathcal{D}$ of n samples, neglecting the cost of the regression, is given by:*

$$\mathcal{O}\left(Jn \left(1 + \frac{|\mathcal{A}| - 1}{k} \right) \right).$$

Proof. Let us consider an iteration $j = 0, \dots, J-1$. If $j \bmod k = 0$, we perform an application of $\widehat{T}^*$ which requires performing $n|\mathcal{A}|$ evaluations of the next-state value function in order to compute the maximum over the actions. On the contrary, when $j \bmod k \neq 0$, we perform an application of $\widehat{T}^{\delta}$ which requires just n evaluations, since the next-state value function is evaluated in the persistent action only. By the definition of PFQI(k), J must be an integer multiple of the persistence k. Recalling that a single evaluation of the approximate Q-function is $\mathcal{O}(1)$, we have that the overall complexity is:

$$
\begin{aligned}
&\mathcal{O}\left(\sum_{j \in \{0,\dots,J-1\} \wedge j \bmod k = 0} n|\mathcal{A}| + \sum_{j \in \{0,\dots,J-1\} \wedge j \bmod k \neq 0} n \right) \\
&= \mathcal{O}\left(\frac{J}{k} n|\mathcal{A}| + \frac{J(k-1)}{k} n \right) \\
&= \mathcal{O}\left(Jn \left(1 + \frac{|\mathcal{A}|-1}{k} \right) \right).
\end{aligned}
$$

□

Theorem 6.3 (Error Propagation). *Let $p \geq 1$, $k, J \in \mathbb{N}^+$ with $J \bmod k = 0$ and $\mu \in \Delta_{\mathcal{S}\times\mathcal{A}}$. Then for any sequence $\{Q^{(j)}\}_{j=0}^{J} \subset \mathcal{F}$ uniformly bounded by $Q_{\max} \leq \frac{R_{\max}}{1-\gamma}$, the corresponding $\{\epsilon^{(j)}\}_{j=0}^{J-1}$ defined in Equation* (6.24) *and for any $r \in [0,1]$ and $q \in [1,+\infty]$ it holds that:*

$$
\begin{aligned}
\left\| Q_k^{\star} - Q_k^{\pi^{(J)}} \right\|_{p,\mu} \leq \frac{2\gamma^k}{(1-\gamma)(1-\gamma^k)} \Bigg[& \frac{2}{1-\gamma} \gamma^{\frac{J}{p}} R_{\max} \\
& + C_{\mathrm{VI},\mu,\nu}^{\frac{1}{2p}}(J,r,q) \mathcal{E}^{\frac{1}{2p}}(\epsilon^{(0)},\dots,\epsilon^{(J-1)};r,q) \Bigg].
\end{aligned}
$$

The expression of $C_{\mathrm{VI},\mu,\nu}(J;r,q)$ and $\mathcal{E}(\cdot;r,q)$ can be found in Appendix B.1.2, where the proof of the theorem is also provided.

Before proving the main result, we need to introduce a variation of the *concentrability* coefficients (Antos et al., 2008; Farahmand, 2011) to account for action persistence.

Definition B.8 (Persistent Expected Concentrability). *Let $\mu, \nu \in \Delta_{\mathcal{S}\times\mathcal{A}}$, $L \in \mathbb{N}^+$, and an arbitrary sequence of stationary policies $(\pi^{(l)})_{l=1}^{L}$. Let $k \in \mathbb{N}^+$ be the persistence. For any $m_1, m_2, m_3 \in \mathbb{N}^+$ and $q \in [1,+\infty]$, we define:*

$$
c_{\mathrm{VI}_1,k,q,\rho,\nu}(m_1,m_2,m_3;\pi) = \mathbb{E}\left[\left| \frac{\mathrm{d}\left(\mu (p_k^{\pi})^{m_1} (p_k^{\pi_k^{\star}})^{m_2} (p_\delta)^{m_3}\right)}{\mathrm{d}\nu}(s,a) \right|^{\frac{q}{q-1}} \right]^{\frac{q-1}{q}},
$$

$$
c_{\mathrm{VI}_2,k,q,\rho,\nu}(m_1,m_2;(\pi^{(l)})_{l=1}^{L}) = \mathbb{E}\left[\left| \frac{\mathrm{d}\left(\mu (p_k^{\pi^{(L)}})^{m_1} p_k^{\pi^{(L-1)}} \dots p_k^{\pi^{(1)}} (p_\delta)^{m_2}\right)}{\mathrm{d}\nu}(s,a) \right|^{\frac{q}{q-1}} \right]^{\frac{q-1}{q}},
$$

with $(s,a) \sim \nu$. If $\mu (p_k^{\pi})^{m_1} (p_k^{\pi_k^{\star}})^{m_2} (p_\delta)^{m_3}$ (resp. $\mu (p_k^{\pi^{(L)}})^{m_1} p_k^{\pi^{(L-1)}} \dots p_k^{\pi^{(1)}} (p_\delta)^{m_2}$) is not absolutely continuous w.r.t. to ν, then we take $c_{\mathrm{VI}_1,\mu,\nu}(m_1,m_2,m_3;\pi,k) = +\infty$ (resp. $c_{\mathrm{VI}_2,\mu,\nu}(m_1,m_2;(\pi^{(l)})_{l=1}^{L},k) = +\infty$).

This definition is a generalization of that provided in Farahmand (2011), that can be recovered by setting $k=1$, $q=2$, $m_3=0$ for the first coefficient and $m_2=0$ for the second coefficient..

Proof. The proof follows most of the steps of Theorem 3.4 of Farahmand (2011). We start by deriving a bound relating $Q^\star - Q^{(J)}$ to $(\epsilon^{(j)})_{j=0}^{J-1}$. To this purpose, let us first define the cumulative error over k iterations for every $j \bmod k = 0$:

$$\epsilon_k^{(j)} = T_k^\star Q^{(j)} - Q^{(j+k)}. \tag{B.31}$$

Let us denote with $\pi_k^\star$ one of the optimal policies of the k-persistent MDP $\mathcal{M}_k$. We have:

$$\begin{aligned}
Q_k^\star - Q^{(j+k)} &= T_k^{\pi_k^\star} Q_k^\star \pm T_k^{\pi_k^\star} Q^{(j)} - T_k^\star Q^{(j)} + \epsilon_k^{(j)} \leq \gamma^k p_k^{\pi_k^\star}(Q_k^\star - Q^{(j)}) + \epsilon_k^{(j)}, \\
Q_k^\star - Q^{(j+k)} &= T_k^\star Q_k^\star \pm T_k^{\pi^{(j)}} Q^\star - T_k^\star Q^{(j)} + \epsilon_k^{(j)} \geq \gamma^k p_k^{\pi^{(j)}}(Q_k^\star - Q^{(j)}) + \epsilon_k^{(j)},
\end{aligned}$$

where we exploited the fact that $T_k^\star Q^{(j)} \geq T_k^{\pi_k^\star} Q^{(j)}$, the definition of greedy policy $\pi^{(j)}$ that implies that $T_k^{\pi^{(j)}} Q^{(j)} = T_k^\star Q^{(j)}$ and the definition of $\epsilon_k^{(j)}$. By unrolling the expression derived above, we have that for every $J \bmod k = 0$:

$$\begin{aligned}
Q_k^\star - Q^{(J)} &\leq \sum_{h=0}^{\frac{J}{k}-1} \gamma^{J-k(h+1)} \left(p_k^{\pi_k^\star}\right)^{\frac{J}{k}-h-1} \epsilon_k^{(j)} + \gamma^J \left(p_k^{\pi_k^\star}\right)^{\frac{J}{k}} (Q_k^\star - Q^{(0)}) \\
Q_k^\star - Q^{(J)} &\geq \sum_{h=0}^{\frac{J}{k}-1} \gamma^{J-k(h+1)} \left(p_k^{\pi^{(J-k)}} p_k^{\pi^{(J-2k)}} \dots p_k^{\pi^{(k(h+1))}}\right) \epsilon_k^{(j)} \\
&\quad + \gamma^J \left(p_k^{\pi^{(J)}} p_k^{\pi^{(J-k)}} \dots p_k^{\pi^{(k)}}\right) (Q_k^\star - Q^{(0)}).
\end{aligned} \tag{B.32}$$

We now provide the following bound relating the difference $Q_k^\star - Q_k^{\pi^{(J)}}$ to the difference $Q_k^\star - Q^{(J)}$:

$$\begin{aligned}
Q_k^\star - Q_k^{\pi^{(J)}} &= T_k^{\pi_k^\star} Q_k^\star \pm T_k^{\pi_k^\star} Q^{(J)} \pm T_k^\star Q^{(J)} - T_k^{\pi^{(J)}} Q_k^{\pi^{(J)}} \\
&\leq T_k^{\pi_k^\star} Q_k^\star - T_k^{\pi_k^\star} Q^{(J)} + T_k^\star Q^{(J)} - T_k^{\pi^{(J)}} Q_k^{\pi^{(J)}} \\
&= \gamma^k p_k^{\pi_k^\star} (Q^\star - Q^{(J)}) + \gamma^k p_k^{\pi^{(J)}} (Q^{(J)} - Q_k^{\pi^{(J)}}) \\
&= \gamma^k p_k^{\pi_k^\star} (Q^\star - Q^{(J)}) + \gamma^k p_k^{\pi^{(J)}} (Q^{(J)} - Q_k^\star + Q_k^\star - Q_k^{\pi^{(J)}}),
\end{aligned}$$

where we exploited $T_k^\star Q^{(J)} \geq T_k^{\pi_k^\star} Q^{(J)}$ and observed that $T_k^\star Q^{(J)} = T_k^{\pi^{(J)}} Q^{(J)}$. By using Lemma 4.2 of Munos (2007) we can derive:

$$Q_k^\star - Q_k^{\pi^{(J)}} \leq \gamma^k \left(\mathrm{Id} - \gamma^k p_k^{\pi^{(J)}}\right)^{-1} \left(p_k^{\pi_k^\star} - p_k^{\pi^{(J)}}\right) (Q^\star - Q^{(J)}). \tag{B.33}$$

By plugging equation (B.32) into equation (B.33):

$$\begin{aligned}
Q_k^\star - Q_k^{\pi^{(J)}} &\leq \gamma^k \left(\mathrm{Id} - \gamma^k p_k^{\pi^{(J)}}\right)^{-1} \\
&\times \Bigg[\sum_{h=0}^{\frac{J}{k}-1} \gamma^{J-k(h+1)} \left(\left(p_k^{\pi_k^\star}\right)^{\frac{J}{k}-h} - \left(p_k^{\pi^{(J)}} p_k^{\pi^{(J-k)}} p_k^{\pi^{(J-2k)}} \dots p_k^{\pi^{(k(h+1))}}\right) \right) \epsilon_k^{(j)} \\
&\quad + \gamma^J \left(\left(p_k^{\pi_k^\star}\right)^{\frac{J}{k}+1} - \left(p_k^{\pi^{(J)}} p_k^{\pi^{(J)}} p_k^{\pi^{(J-k)}} \dots p_k^{\pi^{(k)}}\right) \right) (Q_k^\star - Q^{(0)}) \Bigg].
\end{aligned} \tag{B.34}$$

Now, we need to relate the cumulative errors $\epsilon_k^{(j)}$ to the single-step errors $\epsilon^{(j)}$:

$$\begin{aligned}
\epsilon_k^{(j)} &= T_k^\star Q^{(j)} - Q^{(j+k)} \\
&= (T^\delta)^{k-1} T^* Q^{(j)} - (T^\delta)^{k-1} Q^{(j+1)} + (T^\delta)^{k-1} Q^{(j+1)} - Q^{(j+k)} \\
&= \gamma^{k-1} (p_\delta)^{k-1} \left(T^* Q^{(j)} - Q^{(j+1)} \right) + (T^\delta)^{k-1} Q^{(j+1)} - Q^{(j+k)} \\
&= \gamma^{k-1} (p_\delta)^{k-1} \epsilon^{(j)} + (T^\delta)^{k-1} Q^{(j+1)} - Q^{(j+k)}.
\end{aligned}$$

Let us now consider the remaining term $(T^\delta)^{k-1} Q^{(j+1)} - Q^{(j+k)}$:

$$\begin{aligned}
(T^\delta)^{k-1} Q^{(j+1)} - Q^{(j+k)} &= (T^\delta)^{k-1} Q^{(j+1)} \pm (T^\delta)^{k-2} Q^{(j+2)} - Q^{(j+k)} \\
&= \gamma^{k-2} (p_\delta)^{k-2} \left(T^\delta Q^{(j+1)} - Q^{(j+2)} \right) + (T^\delta)^{k-2} Q^{(j+2)} - Q^{(j+k)} \\
&= \gamma^{k-2} (p_\delta)^{k-2} \epsilon^{(j+1)} + (T^\delta)^{k-2} Q^{(j+2)} - Q^{(j+k)} \\
&= \sum_{l=2}^{k} \gamma^{k-l} (p_\delta)^{k-l} \epsilon^{(j+l-1)},
\end{aligned}$$

where the last step is obtained by unrolling the recursion. Putting everything together, we get:

$$\epsilon_k^{(j)} = \sum_{l=1}^{k} \gamma^{k-l} (p_\delta)^{k-l} \epsilon^{(j+l-1)}. \tag{B.35}$$

Consequently, we can rewrite part of the RHS in equation (B.34) as follows:

$$\begin{aligned}
&\sum_{h=0}^{\frac{J}{k}-1} \gamma^{J-k(h+1)} \left(\left(p_k^{\pi_k^\star} \right)^{\frac{J}{k}-h} - \left(p_k^{\pi^{(J)}} p_k^{\pi^{(J-k)}} p_k^{\pi^{(J-2k)}} \dots p_k^{\pi^{(k(h+1))}} \right) \right) \\
&\quad \times \sum_{l=1}^{k} \gamma^{k-l} (p_\delta)^{k-l} \epsilon^{(j+l-1)} \\
&= \sum_{h=0}^{\frac{J}{k}-1} \sum_{l=1}^{k} \gamma^{J-kh-l} \left(\left(p_k^{\pi_k^\star} \right)^{\frac{J}{k}-h} - \left(p_k^{\pi^{(J)}} p_k^{\pi^{(J-k)}} p_k^{\pi^{(J-2k)}} \dots p_k^{\pi^{(k(h+1))}} \right) \right) \qquad \text{(B.36)} \\
&\quad \times (p_\delta)^{k-l} \epsilon^{(j+l-1)} \\
&= \sum_{j=0}^{J-1} \gamma^{J-j-1} \left(\left(p_k^{\pi_k^\star} \right)^{\frac{J}{k}-j \text{ div } k} - \left(p_k^{\pi^{(J)}} p_k^{\pi^{(J-k)}} p_k^{\pi^{(J-2k)}} \dots p_k^{\pi^{(J-k(j \text{ div } k+1))}} \right) \right) \\
&\quad \times (p_\delta)^{k-j \bmod k-1} \epsilon^{(j)} \qquad \text{(B.37)}
\end{aligned}$$

where line (B.36) derives from rearranging the two summations, line (B.37) is obtained from a redefinition of the indexes. Specifically, we observed that $h = j \text{ div } k$, $j + 1 = kh + l$, and $l = j \bmod k + 1$. Finally, we can apply the absolute value to the RHS and use Jensen inequality to obtain:

$$
\begin{aligned}
Q_k^\star - Q_k^{\pi^{(J)}} \leq \Big(\mathrm{Id} - \gamma^k p_k^{\pi^{(J)}}\Big)^{-1} \\
\Bigg[\sum_{j=0}^{J-1} \gamma^{J-j-1} \bigg(\Big(p_k^{\pi_k^\star}\Big)^{\frac{J}{k}-j \text{ div } k} + \Big(p_k^{\pi^{(J)}} p_k^{\pi^{(J-k)}} p_k^{\pi^{(J-2k)}} \dots p_k^{\pi^{(J-k(j \text{ div } k+1))}}\Big)\bigg) \\
\times (p_\delta)^{k-j \bmod k-1} \left|\epsilon^{(j)}\right| \\
+ \gamma^J \bigg(\Big(p_k^{\pi_k^\star}\Big)^{\frac{J}{k}+1} + \Big(p_k^{\pi^{(J)}} p_k^{\pi^{(J)}} p_k^{\pi^{(J-k)}} \dots p_k^{\pi^{(k)}}\Big)\bigg) \left|Q_k^\star - Q^{(0)}\right|\Bigg]
\end{aligned}
$$

We now introduce the following terms: For all $0 \leq j < J$:

$$
\begin{aligned}
A_j = \frac{1-\gamma^k}{2} \Big(\mathrm{Id} - \gamma^k p_k^{\pi^{(J)}}\Big)^{-1} \bigg((p_k^{\pi_k^\star})^{\frac{J}{k}-j \text{ div } k} \\
+ (p_k^{\pi^{(J)}} p_k^{\pi^{(J-k)}} p_k^{\pi^{(J-2k)}} \dots p_k^{\pi^{(J-k(j \text{ div } k+1))}})\bigg) (p_\delta)^{k-j \bmod k-1}
\end{aligned}
$$

For $j = J$, we define the following:

$$
A_J = \frac{1-\gamma^k}{2} \Big(\mathrm{Id} - \gamma^k p_k^{\pi^{(J)}}\Big)^{-1} \bigg(\Big(p_k^{\pi_k^\star}\Big)^{\frac{J}{k}+1} + \Big(p_k^{\pi^{(J)}} p_k^{\pi^{(J)}} p_k^{\pi^{(J-k)}} \dots p_k^{\pi^{(k)}}\Big)\bigg)
$$

Now, from the definition of α_j as in Farahmand (2011):

$$
\alpha_j = \begin{cases} \frac{(1-\gamma)\gamma^{J-j-1}}{1-\gamma^{J+1}} & \text{if } 0 \leq j < J \\ \frac{(1-\gamma)\gamma^J}{1-\gamma^{J+1}} & \text{if } j = J \end{cases} . \tag{B.38}
$$

Recalling that $\left|Q_k^\star - Q^{(0)}\right| \leq Q_{\max} + \frac{R_{\max}}{1-\gamma} \leq \frac{2R_{\max}}{1-\gamma}$ and applying Jensen inequality we get:

$$
Q_k^\star - Q_k^{\pi^{(J)}} \leq \frac{2\gamma^k(1-\gamma^{J+1})}{(1-\gamma^k)(1-\gamma)} \left[\sum_{j=0}^{J-1} \alpha_j A_j \left|\epsilon^{(j)}\right| + \alpha_J \frac{2R_{\max}}{1-\gamma} \mathbf{1}\right],
$$

where $\mathbf{1}$ denotes the constant function on $\mathcal{S} \times \mathcal{A}$ with value 1. We now take the $L_p(\mu)$–norm both sides, recalling that $\sum_{j=1}^J \alpha_j = 1$ and that the terms A_j are positive linear operators $A_j : \mathscr{B}(\mathcal{S} \times \mathcal{A}) \to \mathscr{B}(\mathcal{S} \times \mathcal{A})$ such that $A_j \mathbf{1} = \mathbf{1}$. Thus, by Lemma 12 of Antos et al. (2008), we can apply Jensen inequality twice (once w.r.t. α_j and once w.r.t. A_j), getting:

$$
\left\|Q_k^\star - Q_k^{\pi^{(J)}}\right\|_{p,\mu}^p \leq \left(\frac{2\gamma^k(1-\gamma^{J+1})}{(1-\gamma^k)(1-\gamma)}\right)^p \mu \left[\sum_{j=0}^{J-1} \alpha_j A_j \left|\epsilon^{(j)}\right|^p + \alpha_J \left(\frac{2R_{\max}}{1-\gamma}\right)^p \mathbf{1}\right].
$$

Consider now the individual terms $\mu A_j \left|\epsilon^{(j)}\right|^p$ for $0 \leq j < J$. By the properties of the

Neumann series we have:

$$\begin{aligned}
\mu A_j \left|\epsilon^{(j)}\right|^p &= \frac{1-\gamma^k}{2}\mu\left(\mathrm{Id} - \gamma^k p_k^{\pi^{(J)}}\right)^{-1} \\
&\quad \times \left(\left(p_k^{\pi_k^\star}\right)^{\frac{J}{k} - j \text{ div } k} + \left(p_k^{\pi^{(J)}} p_k^{\pi^{(J-k)}} p_k^{\pi^{(J-2k)}} \dots p_k^{\pi^{(J-k(j \text{ div } k+1))}}\right)\right) \\
&\quad \times (p_\delta)^{k-j \bmod k-1} \left|\epsilon^{(j)}\right|^p \\
&= \frac{1-\gamma^k}{2}\mu\Bigg[\sum_{m=0}^{+\infty} \gamma^{km}\Bigg(\left(p_k^{\pi^{(J)}}\right)^m \left(p_k^{\pi_k^\star}\right)^{\frac{J}{k} - j \text{ div } k} \\
&\quad + \left(\left(p_k^{\pi^{(J)}}\right)^{m+1} p_k^{\pi^{(J-k)}} p_k^{\pi^{(J-2k)}} \dots p_k^{\pi^{(J-k(j \text{ div } k))}}\right)\Bigg)\Bigg] \\
&\quad \times (p_\delta)^{k-j \bmod k-1} \left|\epsilon^{(j)}\right|^p .
\end{aligned}$$

We now aim at introducing the concentrability coefficients and for this purpose, we employ the following inequality. For any measurable function $f \in (\mathcal{X}) \to \mathbb{R}$, and the probability measures $\mu_1, \mu_2 \in \mathscr{P}(\mathcal{X})$ such that μ_2 is absolutely continuous w.r.t. μ_1, we have the following Hölder inequality, for any $q \in [1, +\infty]$:

$$\int_{\mathcal{X}} f \,\mathrm{d}\mu_1 \leq \left(\int_{\mathcal{X}} \left|\frac{\mathrm{d}\mu_1}{\mathrm{d}\mu_2}\right|^{\frac{q}{q-1}} \mathrm{d}\mu_2\right)^{\frac{q-1}{q}} \left(\int_{\mathcal{X}} |f|^q \,\mathrm{d}\mu_2\right)^{\frac{1}{q}}. \tag{B.39}$$

We now focus on a single term $\mu\left(p_k^{\pi^{(J)}}\right)^m \left(p_k^{\pi_k^\star}\right)^{\frac{J}{k} - j \text{ div } k} \left|\epsilon^{(j)}\right|^p$ and we apply the above inequality:

$$\begin{aligned}
&\mu\left(p_k^{\pi^{(J)}}\right)^m \left(p_k^{\pi_k^\star}\right)^{\frac{J}{k} - j \text{ div } k} (p_\delta)^{k-j \bmod k-1} \left|\epsilon^{(j)}\right|^p \\
&\leq \left(\int_{\mathcal{S}\times\mathcal{A}} \left|\frac{\mathrm{d}\mu\left(p_k^{\pi^{(J)}}\right)^m \left(p_k^{\pi_k^\star}\right)^{\frac{J}{k} - j \text{ div } k} (p_\delta)^{k-j \bmod k-1}}{\mathrm{d}\nu}\right|^{\frac{q}{q-1}} \mathrm{d}\nu\right)^{\frac{q-1}{q}} \\
&\quad \times \left(\int_{\mathcal{S}\times\mathcal{A}} \left|\epsilon^{(j)}\right|^{pq} \mathrm{d}\nu\right)^{\frac{1}{q}} \\
&= c_{\mathrm{VI}_1,k,q,\rho,\nu}\left(m, \frac{J}{k} - j \text{ div } k, k - j \bmod k - 1; \pi^{(J)}\right) \left\|\epsilon^{(j)}\right\|_{pq,\nu}^p .
\end{aligned}$$

Proceeding in an analogous way for the remaining terms, we get to the expression:

$$\left\|Q_k^\star - Q_k^{\pi^{(J)}}\right\|_{p,\mu}^p \leq \left(\frac{2\gamma^k(1-\gamma^{J+1})}{(1-\gamma^k)(1-\gamma)}\right)^p \left[\frac{1-\gamma^k}{2}\sum_{j=0}^{J-1}\sum_{m=0}^{+\infty}\gamma^{km}\right.$$
$$\left(c_{\mathrm{VI}_1,k,q,\rho,\nu}\left(m, \frac{J}{k} - j \text{ div } k, k - j \bmod k - 1; \pi^{(J)}\right)\right.$$
$$\left. + c_{\mathrm{VI}_2,k,q,\rho,\nu}\left(m+1, k - j \bmod k - 1; \{\pi^{(J-lk)}\}_{l=1}^{j \text{ div } k}\right)\right) \left\|\epsilon^{(j)}\right\|_{pq,\nu}^p$$
$$\left. + \alpha_J \left(\frac{2R_{\max}}{1-\gamma}\right)^p\right].$$

To separate the concentrability coefficients and the approximation errors, we apply Hölder inequality with $s \in [1, +\infty]$:

$$\sum_{j=0}^{J} a_j b_j \leq \left(\sum_{j=0}^{J} |a_j|^s\right)^{\frac{1}{s}} \left(|b_j|^{\frac{s}{s-1}}\right)^{\frac{s-1}{s}}. \tag{B.40}$$

Let $r \in [0,1]$, we set

$$a_j = \alpha_j^r \left\|\epsilon^{(j)}\right\|_{pq,\nu}^p$$
$$b_j = \alpha_j^{1-r}\frac{1-\gamma^k}{2}\sum_{j=0}^{J-1}\sum_{m=0}^{+\infty}\gamma^{km}\left(c_{\mathrm{VI}_1,k,q,\rho,\nu}\left(m, \frac{J}{k} - j \text{ div } k, k - j \bmod k - 1; \pi^{(J)}\right)\right.$$
$$\left. + c_{\mathrm{VI}_2,k,q,\rho,\nu}\left(m+1, k - j \bmod k - 1; \{\pi^{(J-lk)}\}_{l=1}^{j \text{ div } k}\right)\right).$$

The application of Hölder inequality leads to:

$$\left\|Q_k^\star - Q_k^{\pi^{(J)}}\right\|_{p,\mu}^p \leq \left(\frac{2\gamma^k(1-\gamma^{J+1})}{(1-\gamma^k)(1-\gamma)}\right)^p \frac{1-\gamma^k}{2}\left[\sum_{j=0}^{J-1}\alpha_j^{\frac{s(1-r)}{s-1}}\right.$$
$$\times\left(\sum_{m=0}^{+\infty}\gamma^{km}\left(c_{\mathrm{VI}_1,k,q,\rho,\nu}\left(m, \frac{J}{k} - j \text{ div } k, k - j \bmod k - 1; \pi^{(J)}\right)\right.\right.$$
$$\left.\left.\left. + c_{\mathrm{VI}_2,k,q,\rho,\nu}\left(m+1, k - j \bmod k - 1; \{\pi^{(J-lk)}\}_{l=1}^{j \text{ div } k}\right)\right)\right)^{\frac{s}{s-1}}\right]^{\frac{s-1}{s}}$$
$$\left[\sum_{j=0}^{J-1}\alpha_j^{sr}\left\|\epsilon^{(j)}\right\|_{pq,\nu}^{sp}\right]^{\frac{1}{s}}$$
$$+\left(\frac{2\gamma^k(1-\gamma^{J+1})}{(1-\gamma^k)(1-\gamma)}\right)^p \alpha_J \left(\frac{2R_{\max}}{1-\gamma}\right)^p.$$

Since the policies $(\pi^{(J-lk)})_{l=1}^{j \text{ div } k}$ are not known, we define the following quantity by taking the supremum over any sequence of policies:

$$C_{\mathrm{VI},\mu,\nu}(J;r,s,q) = \left(\frac{1-\gamma^k}{2}\right)^s \sup_{\pi_0,\dots,\pi_J\in\Pi} \sum_{j=0}^{J-1} \alpha_j^{\frac{s(1-r)}{s-1}} \left(\sum_{m=0}^{+\infty} \gamma^{km}\left(c_{\mathrm{VI}_1,k,q,\rho,\nu}\left(m, \frac{J}{k} - j \operatorname{div} k, k - j \bmod k - 1; \pi_J\right) + c_{\mathrm{VI}_2,k,q,\rho,\nu}\left(m+1, k - j \bmod k - 1; \{\pi_l\}_{l=1}^{j \operatorname{div} k}\right)\right)\right)^{\frac{s}{s-1}}. \tag{B.41}$$

Moreover, we define the following term that embeds all the terms related to the approximation error:

$$\mathcal{E}(\epsilon^{(0)},\dots,\epsilon^{(J-1)};r,s,q) = \sum_{j=0}^{J-1} \alpha_j^{sr} \left\|\epsilon^{(j)}\right\|_{pq,\nu}^{sp}. \tag{B.42}$$

Observing that $\frac{1-\gamma}{1-\gamma^{J+1}} \le 1$ and $1-\gamma^{J-1} \le 1$, we can put everything together and taking the p–th root and recalling that the inequality holds for all $q \in [1,+\infty]$, $r \in [0,1]$, and $s \in [1,+\infty]$:

$$\left\|Q_k^\star - Q_k^{\pi^{(J)}}\right\|_{p,\mu} \le \frac{2\gamma^k}{(1-\gamma^k)(1-\gamma)}\left[\gamma^{\frac{J}{p}}\frac{2R_{\max}}{1-\gamma} + \inf_{\substack{q\in[1,+\infty]\\ r\in[0,1]\\ s\in[1,+\infty]}} C_{\mathrm{VI},\mu,\nu}(J;r,s,q)^{\frac{s-1}{ps}} \mathcal{E}(\epsilon^{(0)},\dots,\epsilon^{(J-1)};r,s,q)^{\frac{1}{ps}}\right].$$

The statement is simplified by taking $s=2$.

□

B.1.3 Proofs of Section 6.7

Lemma 6.10. *Let $Q : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$ be a bounded, measurable action-value function, and let π be a greedy policy w.r.t. Q. Let $J = \int \mu(\mathrm{d}s)V(s)$, with $V(s) = \max_{a\in\mathcal{A}} Q(s,a)$ for all $s \in \mathcal{S}$. Then, for any $k \in \mathbb{N}^+$, it holds that:*

$$J_k(\pi) \ge J(\pi) - \frac{1}{1-\gamma^k}\left\|T_k^\star Q - Q\right\|_{1,\eta_k^{\mu,\pi}}, \tag{6.26}$$

where $\eta_k^{\mu,\pi} = (1-\gamma^k)\mu\pi\left(\mathrm{Id} - \gamma^k p_k^\pi\right)^{-1}$, is the γ-discounted stationary distribution induced by policy π and initial distribution μ in MDP $\mathcal{M}_k$.

Proof. We start by providing the following equality, recalling that $T_k^\star Q = T_k^\pi Q$, being π the greedy policy w.r.t. Q:

$$\begin{aligned} Q_k^\pi - Q &= T_k^\pi Q_k^\pi - T_k^\pi Q + T_k^\star Q - Q \\ &= \gamma^k p_k^\pi\left(Q_k^\pi - Q\right) + T_k^\star Q - Q \\ &= \left(\mathrm{Id} - \gamma^k p_k^\pi\right)^{-1}\left(T_k^\star Q - Q\right), \end{aligned}$$

where the last equality follows from the properties of the Neumann series. We take the expectation w.r.t. to the distribution $\mu\pi$ on both sides. For the left-hand side, we have:

$$J_k(\pi) - J(\pi) = \mu\pi Q_k^\pi - \mu\pi Q.$$

Concerning the right-hand side, instead, we have:

$$\mu\pi \left(\mathrm{Id} - \gamma^k p_k^\pi\right)^{-1} \left(T_k^\star Q - Q\right) = \frac{1}{1-\gamma^k} \eta_k^{\mu,\pi} \left(T_k^\star Q - Q\right),$$

where we introduced the γ–discounted stationary distribution (Sutton et al., 1999a) after normalization. Putting all together, we can derive the following inequality:

$$\begin{aligned} J_k(\pi) - J(\pi) &= \frac{1}{1-\gamma^k} \eta_k^{\mu,\pi} \left(T_k^\star Q - Q\right) \\ &\geq -\frac{1}{1-\gamma^k} \eta_k^{\mu,\pi} \left|T_k^\star Q - Q\right| \\ &= -\frac{1}{1-\gamma^k} \left\|T_k^\star Q - Q\right\|_{1,\eta_k^{\mu,\pi}}. \end{aligned}$$

□

B.2 Details on Bounding the Performance Loss (section 6.5)

In this section, we report some additional material that is referenced in Section 6.5, concerning the performance loss due to the usage of action persistence.

B.2.1 On Using Divergences Other than the Kantorovich

The Persistence Bound presented in Theorem 6.1 is defined in terms of the dissimilarity index $d_{\mathcal{Q}_k}^\pi$ which depends on the set of functions $\mathcal{Q}_k$ defined in terms of the k-persistent Q-function Q_k^π and in terms of the Bellman operators T^π and T^δ. This bound is meaningful when it yields a value that is smaller than $\frac{2\gamma R_{\max}}{1-\gamma}$ that we already know to be the maximum performance degradation we experience when executing policy π with persistence (proposition 6.9). Therefore, for any meaningful choice of $\mathcal{Q}_k$, we require that, at least for $k=2$, the following condition to hold:

$$\left.\frac{\gamma(1-\gamma^{k-1})}{(1-\gamma)(1-\gamma^k)} \left\|d_{\mathcal{Q}_k}^\pi\right\|_{p,\eta_k^{\rho,\pi}} \right|_{k=2} = \frac{\gamma}{(1-\gamma^2)} \left\|d_{\mathcal{Q}_2}^\pi\right\|_{p,\eta_2^{\rho,\pi}} < \frac{2\gamma R_{\max}}{1-\gamma}. \tag{B.43}$$

If we require no additional regularity conditions on the MDP, we can only exploit the fact that all functions $f \in \mathcal{Q}_k$ are uniformly bounded by $\frac{R_{\max}}{1-\gamma}$, reducing $d_{\mathcal{Q}_k}^\pi$ to the total variation distance between p_π and p_δ:

$$\begin{aligned} d_{\mathcal{Q}_k}^\pi(s,a) &\leq \frac{R_{\max}}{1-\gamma} \sup_{f:\|f\|_\infty \leq 1} \left| \int_{\mathcal{S}} \int_{\mathcal{A}} \left(p_\pi(\,\mathrm{d}s',\,\mathrm{d}a'|s,a) - p_\delta(\,\mathrm{d}s',\,\mathrm{d}a'|s,a)\right) f(s',a') \right| \\ &= \frac{2R_{\max}}{1-\gamma} d_{\mathrm{TV}}^\pi(s,a). \end{aligned} \tag{B.44}$$

We restrict our discussion to deterministic policies and, for this purpose, we denote with $\pi(s) \in \mathcal{A}$ the action prescribed by policy π in the state $s \in \mathcal{S}$. Thus, the total variation distance as follows:

$$\begin{aligned} d_{\mathrm{TV}}^{\pi}(s,a) &= \frac{1}{2}\int_{\mathcal{S}}\int_{\mathcal{A}} |p_{\pi}(\,\mathrm{d}s',\,\mathrm{d}a'|s,a) - p_{\delta}(\,\mathrm{d}s',a'|s,a)| \\ &= \frac{1}{2}\int_{\mathcal{S}} P(\,\mathrm{d}s'|s,a)\int_{\mathcal{A}} |\pi(\,\mathrm{d}a'|s') - \delta_a(\,\mathrm{d}a')| \\ &= \frac{1}{2}\int_{\mathcal{S}} P(\,\mathrm{d}s'|s,a)\int_{\mathcal{A}} \left|\delta_{\pi(s')}(a') - \delta_{\pi(s)}(\,\mathrm{d}a')\right| \\ &= \int_{\mathcal{S}} P(\,\mathrm{d}s'|s,a)\mathbb{1}_{\{\pi(s)\neq\pi(s')\}}, \end{aligned}$$

where $\mathbb{1}_{\mathcal{X}}$ denotes the indicator function for the measurable set $\mathcal{X}$. Consequently, we can derive the norm:

$$\begin{aligned} \left\|d_{Q_2}^{\pi}\right\|_{p,\eta_2^{\rho,\pi}}^{p} &\le \frac{2R_{\max}}{1-\gamma}\int_{\mathcal{S}}\int_{\mathcal{A}} \eta_k^{\rho,\pi}(\,\mathrm{d}s,\,\mathrm{d}a)\left|\int_{\mathcal{S}} P(\,\mathrm{d}s'|s,a)\mathbb{1}_{\{\pi(s)\neq\pi(s')\}}\right|^{p} \\ &\le \frac{2R_{\max}}{1-\gamma}\int_{\mathcal{S}}\int_{\mathcal{A}} \eta_k^{\rho,\pi}(\,\mathrm{d}s,\,\mathrm{d}a)\int_{\mathcal{S}} P(\,\mathrm{d}s'|s,a)\left|\mathbb{1}_{\{\pi(s)\neq\pi(s')\}}\right|^{p} \\ &= \frac{2R_{\max}}{1-\gamma}\int_{\mathcal{S}}\int_{\mathcal{A}} \eta_k^{\rho,\pi}(\,\mathrm{d}s,\,\mathrm{d}a)\int_{\mathcal{S}} P(\,\mathrm{d}s'|s,a)\mathbb{1}_{\{\pi(s)\neq\pi(s')\}}. \end{aligned}$$

Thus, such a term depends on the expected fraction of state-next-state pairs such that their policies prescribe different actions. Consequently, considering the condition at Equation (B.43), we have that it must be fulfilled:

$$\int_{\mathcal{S}}\int_{\mathcal{A}} \eta_k^{\rho,\pi}(\,\mathrm{d}s,\,\mathrm{d}a)\int_{\mathcal{S}} P(\,\mathrm{d}s'|s,a)\mathbb{1}_{\{\pi(s)\neq\pi(s')\}} \le 1-\gamma^2.$$

However, if for every state-next-state pair the prescribed actions are different (even if very similar in some metric space), the left-hand side would be 1, and the inequality would be never satisfied. To embed the notion of closeness of actions we need to resort to distance metrics different from the total variation (e.g., Kantorovich distance). These considerations can be extended to the case of stochastic policies.

B.2.2 Time–Lipschitz Continuity for Dynamical Systems

We now draw a connection between the rate at which a dynamical system evolves and the L_T constant of Assumption 6.1. Consider a continuous-time dynamical system having $\mathcal{S} = \mathbb{R}^{d_S}$ and $\mathcal{A} = \mathbb{R}^{d_{\mathcal{A}}}$ governed by the law $\dot{\mathbf{s}}(t) = \mathbf{f}(\mathbf{s}(t),\mathbf{a}(t))$ such that $\sup_{\mathbf{s}\in\mathcal{S},\mathbf{a}\in\mathcal{A}} \|\mathbf{f}(\mathbf{s},\mathbf{a})\| \le F < +\infty$. Suppose to control the system with a discrete time step $\Delta t_0 > 0$, inducing an MDP with transition model $P_{\Delta t_0}$. Using a norm $\|\cdot\|$, Assumption 6.1 becomes:

$$\begin{aligned} \mathcal{W}_1\left(P_{\Delta t_0}(\cdot|\mathbf{s},\mathbf{a}),\delta_{\mathbf{s}}\right) &= \|\mathbf{s}(t+\Delta t_0) - \mathbf{s}(t)\| \\ &= \left\|\int_{t}^{t+\Delta t_0} \dot{\mathbf{s}}(\,\mathrm{d}t)\right\| \le F\Delta t_0. \end{aligned}$$

Thus, the Time Lipschitz constant L_T depends on: i) how fast the dynamical system evolves (F); ii) the duration of the control time step (Δt_0).

B.3 Details on Experimental Evaluation

In this section, we report the details about our experimental setting (Appendix B.3.1), together with additional plots (Appendix B.3.2).

B.3.1 Experimental Setting

Table B.1 reports the parameters of the experimental setting, which are described in the following.

Environments The implementation of the environments are the ones provided in Open AI Gym (Brockman et al., 2016).

Action Spaces For the environments with finite action space, we collect samples with a uniform policy over $\mathcal{A}$; whereas for the environments with a continuous action space, we perform a discretization, reported in the column "Action space", and we employ the uniform policy over the resulting finite action space.

Sample Collection Samples are collected in the base MDP at persistence 1, although for some of them the uniform policy is executed at a higher persistence, k_{sampling}, reported in the column "Sampling Persistence". Using a persistence greater than 1 to generate samples has been fundamental in some cases (e.g., Mountain Car) to get a better exploration of the environment and improve the learning performances.[1]

Number of Iterations In order to perform a complete application of a k-Persisted Bellman Operator in the PFQI algorithm, we need k iterations, so the total number of iterations needed to complete the training must be an integer multiple of k. In order to compare the resulting performances, we chose the persistences as a range of powers of 2. The total number of iterations J is selected empirically so that the estimated Q-function has reached convergence for all tested persistences.

Time Discretization Each environment has its own way to deal with time discretization. In some cases, in order to make the benefits of persistence evident, we needed to reduce the base control timestep of the environment w.r.t. to the original implementation. We report in the column "Original timestep" ($\Delta t_{\text{original}}$) the control timestep in the original implementation of the environment, while the base time step (Δt_0) is obtained as a fraction of $\Delta t_{\text{original}}$. The reduction of the timestep by a factor $m = \Delta t_{\text{original}}/\Delta t_0$ results in an extension of the horizon of the same factor, hence there is a greater number of rewards to sum, with the consequent need of a larger discount factor to maintain the same "effective horizon". Thus, the new horizon H (resp. discount factor γ) can be determined starting from the original horizon H_{original} (resp. original discount factor γ_{original}) as:

$$H = mH_{\text{original}}, \qquad \gamma = (\gamma_{\text{original}})^{\frac{1}{m}}, \qquad \text{where} \quad m = \frac{\Delta t_{\text{original}}}{\Delta t_0}.$$

For all the environment, the original discount factor γ_{original} has been set to 0.99.

Regressor Hyperparameters We performed regression by means of Extra Trees with the following parameters: n_estimators = 100, min_split = 5, and min_leaf = 2.

Table B.1: *Parameters of the experimental setting, used for the PFQI(k) experiments.*

Environment	$\mathcal{A}$	Sampling Persistence k_{sampling}	Original timestep $\Delta t_{\text{original}}$ (sec)	Factor $m = \Delta t_{\text{original}}/\Delta t_0$	Original Horizon H_{original}	Batch size n	Iterations J
Cartpole	$\{-1, 1\}$	1	0.02	4	128	400	512
Mountain Car	$\{-1, 0, 1\}$	8	1	2	128	20	256
Lunar Lander	{Nop, left, main, right}	1	0.02	1	256	100	256
Pendulum	$\{-2, 0, 2\}$	1	0.05	1	256	100	64
Acrobot	$\{-1, 0, 1\}$	4	0.2	4	128	200	512
Swimmer	$\{-1, 0, 1\}^2$	1	2 (frame-skip)	2	128	100	128
Hopper	$\{-1, 0, 1\}^3$	1	1 (frame-skip)	2	128	100	128
Walker 2D	$\{-1, 0, 1\}^9$	1	1 (frame-skip)	2	128	100	128

B.3.2 Additional Plots

B.4 Preliminary Results on Open Questions

In this section, we report preliminary results related to some open questions about action persistence.

B.4.1 Improving Exploration with Persistence

As we already mentioned, action persistence might have an effect on the exploration properties of distribution ν used to collect samples. To avoid this phenomenon, in this work, we assumed to feed PFQI(k) with the same dataset collected in the base MDP $\mathcal{M}$, independently on which target persistence k we are interested in. In this appendix, we want to briefly analyze what happens when we feed standard FQI with a dataset collected by executing the same policy (e.g., the uniform policy over $\mathcal{A}$) in the k–persistent MDP $\mathcal{M}_k$,[2] To estimate the corresponding k–persistence action-value function $Q^\star_k$. In this way, for each persistence k we have a different sampling distribution ν_k, but, being the dataset $\mathcal{D}_k \sim \nu_k$ collected in $\mathcal{M}_k$, we can apply standard FQI to estimate $Q^\star_k$. Refer to Figure B.2 for a graphical comparison between PFQI(k) executed in the base MDP and FQI executed in the k–persistent MDP.

When we compare the performances of the policies obtained with different persistence levels learned starting with a dataset $\mathcal{D}_k \sim \nu_k$, we should consider two different effects: i) how training samples are generated (i.e., the sampling distribution ν_k, which changes for every persistence k); ii) how they affect the learning process in FQI. Unfortunately, in this setting, we are not able to separate the two effects.

Our goal, in this appendix, is to compare for different values of $k \in \mathcal{K} = \{1, 2, \dots 64\}$ the performance of PFQI(k) and the performance of FQI run on the k–persistent MDP $\mathcal{M}_k$. The experimental setting is the same as in Appendix B.3, apart from the "sampling persistence" which is set to 1 also for the Mountain Car environment. In Figure B.3,

[1]When considering a sampling persistence $k_{\text{sampling}} > 1$, we record in the dataset all the intermediate repeated actions, so that the tuples (S_t, A_t, S'_t, R_t) are transitions of the base MDP $\mathcal{M}$.

[2]This procedure generates a different dataset compared to the case in which we use a "sampling persistence" $k_{\text{sampling}} > 1$, as illustrated in Appendix B.3. Indeed, in this case, we do not record in the dataset the intermediate repeated actions, since we want a dataset of transition of the k–persistent MDP $\mathcal{M}_k$.

we show the performance at the end of training of the policies obtained with PFQI(k), the one derived with FQI on $\mathcal{M}_k$, and the uniform policy over the action space. First of all, we observe that when $k = 1$, executing FQI on $\mathcal{M}_1$ is in all regards equivalent to executing PFQI(1) on $\mathcal{M}$, since PFQI(1) is FQI and $\mathcal{M}_1$ is $\mathcal{M}$. We can see that in the Cartpole environment, fixing a value of $k \in \mathcal{K}$, there is no significant difference in the performances obtained with PFQI(k) and FQI on $\mathcal{M}_k$. The behavior is significantly different when considering Mountain Car. Indeed, we notice that only FQI on $\mathcal{M}_k$ is able to learn a policy that reaches the goal for some specific values of $k \in \mathcal{K}$. We can justify this behavior with the fact that by collecting samples at a persistence k, like in FQI on $\mathcal{M}_k$, the exploration properties of the sampling distribution change, as we can see from the line "Uniform policy". If the input dataset contains no trajectory reaching the goal, our algorithms cannot solve the task. This is why PFQI(k), which uses persistence 1 to collect the samples, is unable to learn at all.[3]

This experiment gives a preliminary hint on how action persistence can affect exploration. More in general, we wonder which are the necessary characteristics of the environment such that the same sampling policy (e.g., the uniform policy over $\mathcal{A}$) allows performing a better exploration. More formally, we ask ourselves how persistence affects the entropy of the stationary distribution induced by the sampling policy.

Learn in $\mathcal{M}_k$ and Execute in $\mathcal{M}_{k'}$

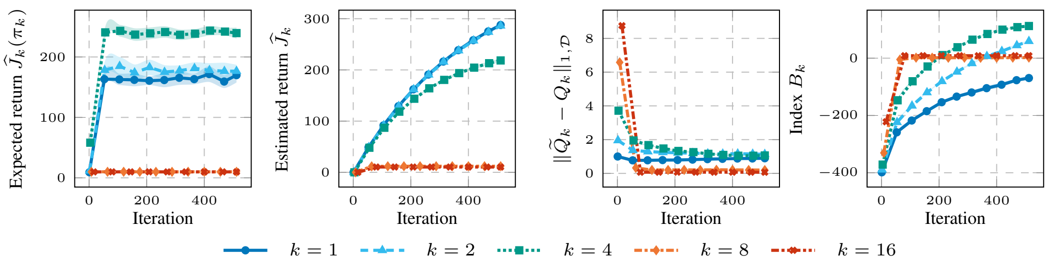
In this appendix, we empirically analyze what happens when a policy is learned by PFQI with a certain persistence level k and executed later on with a different persistence level $k' \neq k$. We consider an experiment on the Cartpole environment, in the same setting as Appendix B.3. We run PFQI(k) for $k \in \mathcal{K} = \{1, 2, \dots, 256\}$ and then for each k we execute policy π_k (i.e., the policy learned by applying the k–persistent operator) in the k'–persistent MDP $\mathcal{M}_{k'}$ for $k' \in \mathcal{K}$. The results are shown in Table B.2. Thus, for each pair (k, k'), Table B.2 shows the sample mean and the sample standard deviation over 20 runs of the expected return of policy π_k in MDP $\mathcal{M}_k$, i.e., $J_{k'}(\pi_k)$. First of all, let us observe that the diagonal of Table B.2 corresponds to the first row of Table 6.1 (apart from the randomness due to the evaluation). If we take a row k, i.e., we fix the persistence of the operator, we notice that, in the majority of the cases, the persistence k' of the MDP yielding the best performance is smaller than k. Moreover, even if we learn a policy with the operator at a given persistence k and we see that such a policy displays a poor performance in the k–persistent MDP (e.g., for $k \geq 8$), when we reduce the persistence, the performance of that policy seems to improve.

Figure B.4 compares different values of k, determining the persistence of the operator, the performance of the policy π_k when we execute it in $\mathcal{M}_k$ and the performance of π_k in the MDP $\mathcal{M}_{(k')^\star}$, where $(k')^\star \in \arg\max_{k' \in \mathcal{K}} \widehat{J}_{k'}(\pi_k)$. We see that suitably selecting the persistence k' of the MDP in which we will deploy the policy, allows reaching higher performances.

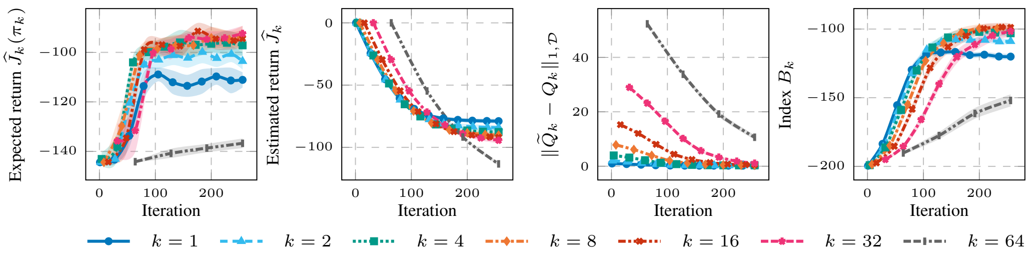
[3]Recall that in our main experiments (Appendix B.3), we had to employ for the Mountain Car a "sampling persistence" $k_{\text{sampling}} = 8$. Indeed, for $k_{\text{sampling}} \in \{1, 2, 4\}$ the uniform policy is unable to reach the goal, while for $k_{\text{sampling}} = 8$ it allows reaching the goal in the 6% of the times on average.

Table B.2: *Results of PFQI execution of the policy π_k learned with the k–persistent operator in the k'–persistent MDP $\mathcal{M}_{k'}$ in the Cartpole experiment. For each k, we report the sample mean and the standard deviation of the estimated return of the last policy $\widehat{J}_{k'}(\pi_k)$. For each k, the persistence k' with the highest average performance and the ones k' that are not statistically significantly different from that one (Welch's t-test with $p < 0.05$) are in bold.*

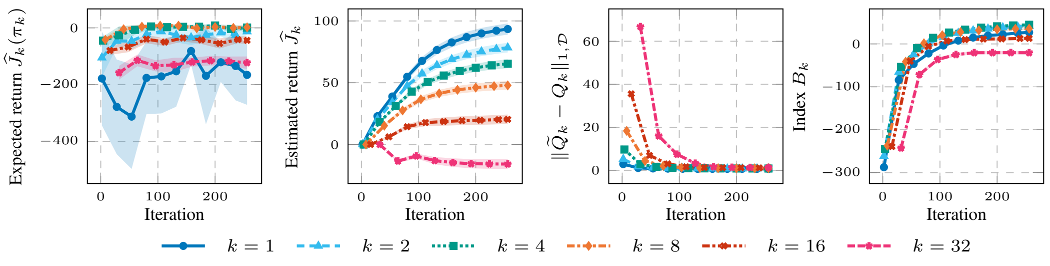
	$k' = 1$	$k' = 2$	$k' = 4$	$k' = 8$	$k' = 16$	$k' = 32$	$k' = 64$	$k' = 128$	$k' = 256$
$k = 1$	$\mathbf{172.0 \pm 6.8}$	$\mathbf{174.1 \pm 6.5}$	113.0 ± 5.3	9.8 ± 0.0	9.7 ± 0.0	9.7 ± 0.1	9.8 ± 0.0	9.7 ± 0.0	9.7 ± 0.0
$k = 2$	$\mathbf{178.4 \pm 6.7}$	$\mathbf{182.2 \pm 7.2}$	151.6 ± 5.1	9.9 ± 0.0	9.8 ± 0.0	9.8 ± 0.0	9.8 ± 0.0	9.8 ± 0.0	9.8 ± 0.0
$k = 4$	276.2 ± 3.8	$\mathbf{287.3 \pm 1.1}$	237.0 ± 5.4	10.0 ± 0.0	9.8 ± 0.0	9.8 ± 0.0	9.9 ± 0.0	9.8 ± 0.0	9.9 ± 0.0
$k = 8$	$\mathbf{284.3 \pm 1.6}$	$\mathbf{281.4 \pm 3.0}$	211.5 ± 4.0	10.0 ± 0.0	9.8 ± 0.0	9.8 ± 0.0	9.8 ± 0.0	9.8 ± 0.0	9.9 ± 0.0
$k = 16$	$\mathbf{285.9 \pm 1.1}$	$\mathbf{282.9 \pm 2.6}$	223.5 ± 3.2	10.0 ± 0.0	9.9 ± 0.0	9.8 ± 0.0	9.9 ± 0.0	9.9 ± 0.0	9.8 ± 0.0
$k = 32$	$\mathbf{285.7 \pm 1.3}$	$\mathbf{283.6 \pm 2.7}$	222.2 ± 3.6	10.0 ± 0.0	9.9 ± 0.0	9.9 ± 0.0	9.8 ± 0.0	9.9 ± 0.0	9.9 ± 0.0
$k = 64$	$\mathbf{283.6 \pm 2.3}$	$\mathbf{284.1 \pm 2.0}$	225.5 ± 4.4	10.0 ± 0.0	9.9 ± 0.0	9.8 ± 0.0	9.9 ± 0.0	9.8 ± 0.0	9.9 ± 0.0
$k = 128$	$\mathbf{282.9 \pm 2.2}$	$\mathbf{282.5 \pm 3.1}$	221.9 ± 4.7	10.0 ± 0.0	9.8 ± 0.0	9.9 ± 0.0	9.9 ± 0.0	9.9 ± 0.0	9.9 ± 0.0
$k = 256$	$\mathbf{282.5 \pm 2.3}$	$\mathbf{283.4 \pm 2.4}$	224.3 ± 3.9	10.0 ± 0.0	9.9 ± 0.0	9.9 ± 0.0	9.9 ± 0.0	9.9 ± 0.0	9.9 ± 0.0



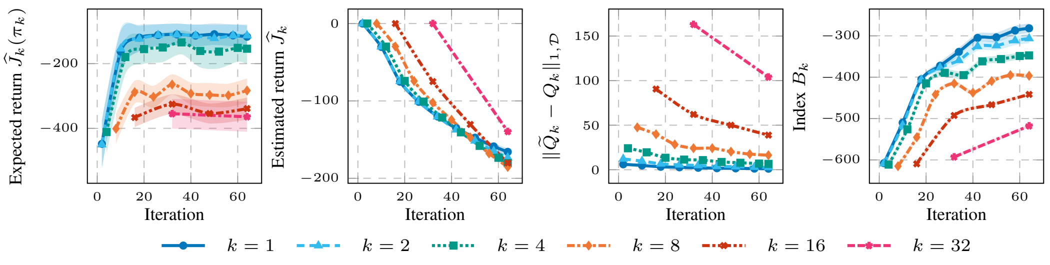
(a) *Cartpole*



(b) *Mountain Car*

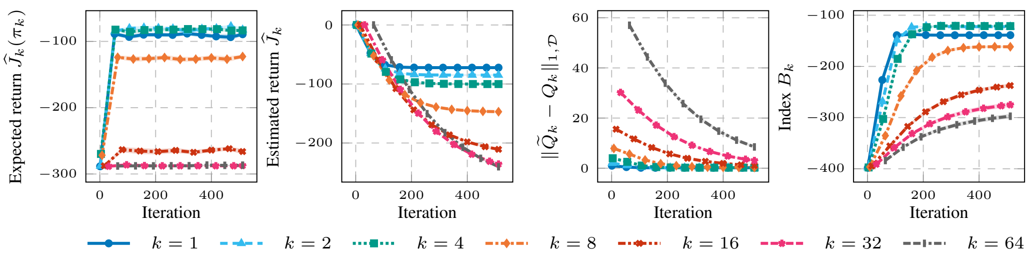


(c) *Lunar Lander*

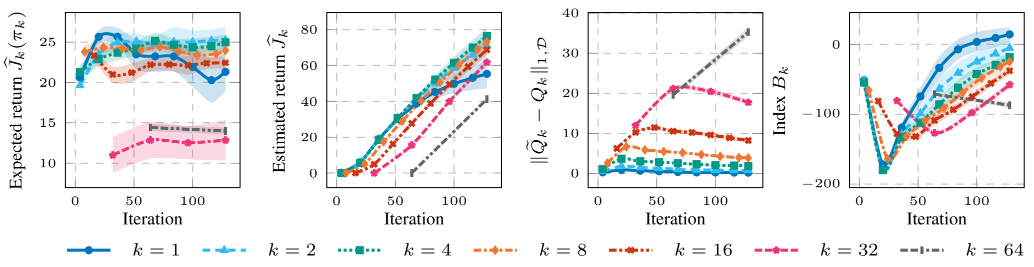


(d) *Pendulum*

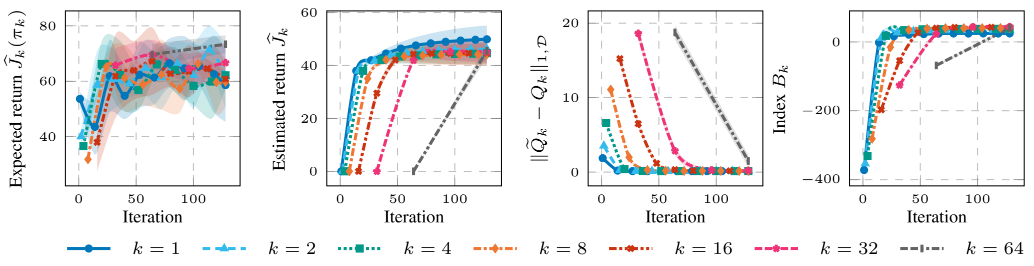
Figure B.1: *Expected return $\widehat{J}_k(\pi_k)$, estimated return J_k, estimated expected Bellman residual $\|\widetilde{Q}_k - Q_k\|_{1,\mathcal{D}}$, and persistence selection index B_k for the different experiments as a function of the number of iterations for different persistences. 20 runs, 95 % c.i.*



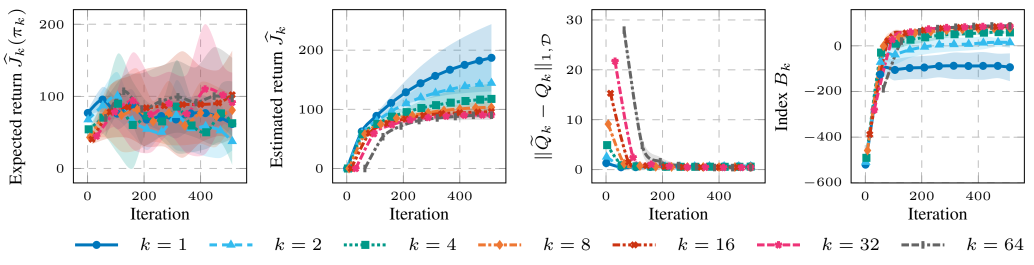
(a) *Acrobot*



(b) *Swimmer*

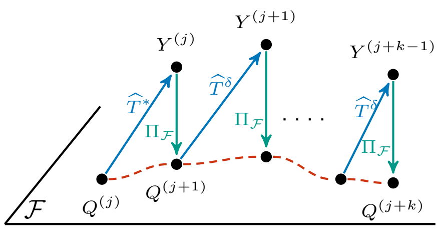


(c) *Hopper*

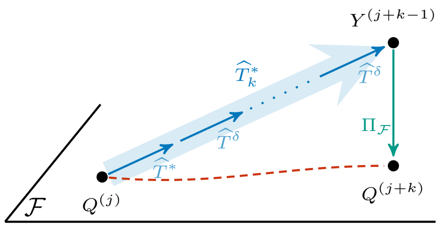


(d) *Walker 2D*

Figure B.1: *Expected return $\widehat{J}_k(\pi_k)$, estimated return $\widehat{J}_k$, estimated expected Bellman residual $\|\widetilde{Q}_k - Q_k\|_{1,\mathcal{D}}$, and persistence selection index B_k for the different experiments as a function of the number of iterations for different persistences. 20 runs, 95 % c.i.*



(a) *PFQI(k) on $\mathcal{M}$*



(b) *FQI on $\mathcal{M}_k$*

Figure B.2: *Illustration of (a) PFQI(k) executed in the base MDP $\mathcal{M}$ and (b) the standard FQI executed in the k-persistent MDP $\mathcal{M}_k$.*

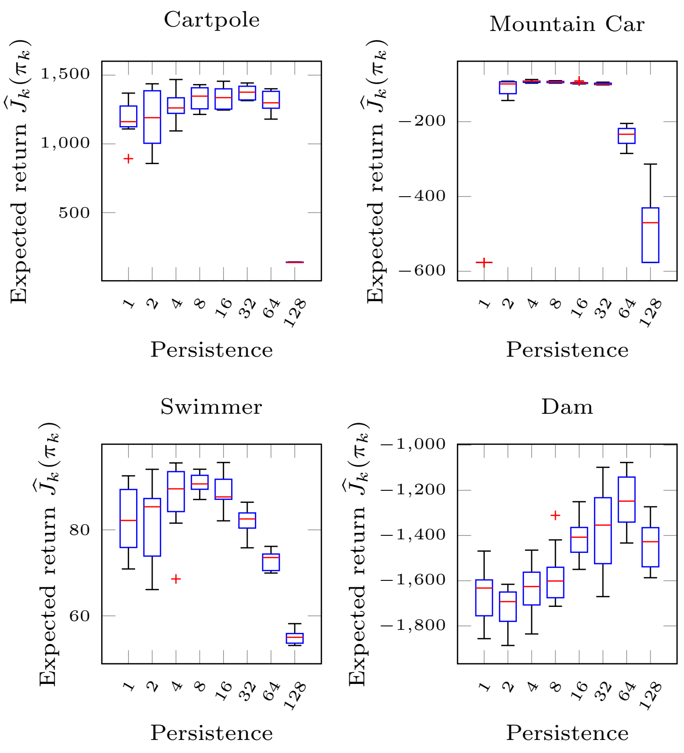


Figure B.3: *Performance of the policies learned with FQI on $\mathcal{M}_k$, PFQI(k) on $\mathcal{M}$ and of the uniform policies for different values of the persistence $k \in \mathcal{K}$. 10 runs. 95% c.i.*

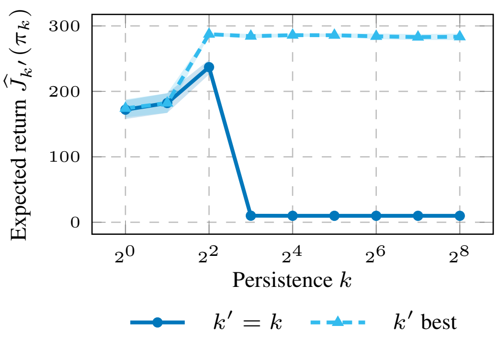


Figure B.4: *Performance of the policies π_k for $k \in \mathcal{K}$ comparing when they are executed in $\mathcal{M}_k$ and when they are executed in $\mathcal{M}_{(k')^\star}$. 20 runs, 95% c.i.*

APPENDIX *C*

Additional Results of Chapter 7

The contents of this Appendix can be summarized as follows:

– Appendix C.1 provides additional information regarding the analysis of the advantage of a dynamic persistence.

– Appendix C.2 we provide further details on the experimental framework.

– Appendix C.3 we report more experimental results and discussions on open questions.

C.1 Discussion on the Advantages of Dynamic Persistence

C.1.1 Details on Persistent Markov Chains and Kemeny Constant

In this section, we provide more details regarding how to obtain the transition Kernel implied from a persistent random variable acting on the environment, and how to compute its Kemeny's constant.

Consider an agent where actions are sampled from a generic policy $\pi(\cdot|s) =: \pi$ on the action space $\mathcal{A}$, independent from the current state, and where persistence is sampled from a discrete distribution ω with support in $\{1, \dots, K_{\max}\}$, independent from π. They constitute the policy ψ over persistence options. The k-step Transition Chain induced by π over the state space $\mathcal{S}$ is defined as $P_k^\pi(s'|s) = \int_{\mathcal{A}} P_k(s'|s,a)\pi(da|s)$. This is equivalent to the Markov chain induced by π in the k-step MDP, where the control frequency is set to k times the base duration δ. We now consider the transition probability induced by the joint probability distributions π and ω up to a maximum of $K_{\max}$ steps, which for simplicity will here be referred to as K. In order to define it, it is necessary to consider a fixed horizon H: when the total number of steps in the trajectory reaches the horizon, then the (eventual) persistence is stopped. This means that, if we start for example at the $H-j$ step, the probability of persisting j times the sampled action is equivalent to $\sum_{i\geq j}\omega_i$. This assumption is necessary for the Markov condition to hold. As a consequence, we define $\widetilde{\omega}_j = \left\{\widetilde{\omega}_{i,j}\right\}_i$ as a reduced distribution of ω to j steps:

$$\widetilde{\omega}_{i,j} := \begin{cases} \omega_i & \text{if } i<j \\ \sum_{i=j}^{K}\omega_i & \text{otherwise} \end{cases}.$$

Finally we can recursively define the transition probability in H steps, induced by π and ω as:

$$\mathbb{P}_H^{\pi,\omega} := \sum_{k=1}^{K} \widetilde{\omega}_{k,H\wedge K}\mathbb{P}_{H-k}^{\pi,\widetilde{\omega}_{H-k}} P_k^{\pi}, \tag{C.1}$$

where $\mathbb{P}_0^{\pi,\omega} = \mathbb{1}_{\mathcal{S}\times\mathcal{S}}$ and $a\wedge b = \min\{a,b\}$. Equation (C.1) is not trivial and needs some clarifications. Let's consider an example, where $K=4$ and $H=3$. In this case the persistence distribution is $\omega = \{\omega_1,\omega_2,\omega_3,\omega_4\}$.

- With probability $\omega_3+\omega_4$, the sampled persistence is equal to 3, and the related transition is P_3^{π} (since $H=3$, sampling persistence 4 leads to repeat the action for 3 times);
- With probability ω_2, the sampled persistence is equal to 2. The first two steps evolve as P_2^{π}, and the last step follows $\mathbb{P}_1^{\pi,\widetilde{\omega}_1} = P^{\pi}$;
- With probability ω_1, the action is selected only once and, at the next step, it has to be sampled again and eventually persisted for two steps w.p. $\omega_2+\omega_3+\omega_4$.

In other terms, denoting $\widetilde{\omega}_1 = \omega_1$, $\widetilde{\omega}_2 = \omega_2$, and $\widetilde{\omega}_3 = \omega_3+\omega_4$:

$$\begin{aligned}\mathbb{P}_3^{\pi,\omega} =& \widetilde{\omega}_3 P_3^{\pi} + \widetilde{\omega}_2 P^{\pi}P_2^{\pi} + \widetilde{\omega}_1[\widetilde{\omega}_1 P^{\pi}P^{\pi} + (\widetilde{\omega}_2+\widetilde{\omega}_3)P_2^{\pi}]P^{\pi} \\ =& \widetilde{\omega}_3 \mathbb{1} P_3^{\pi} + \widetilde{\omega}_2 \underbrace{\left[(\widetilde{\omega}_1+\widetilde{\omega}_2+\widetilde{\omega}_3)P^{\pi}\right]}_{=\mathbb{P}_1^{\pi,\widetilde{\omega}_1}} P_2^{\pi} + \\ &+ \widetilde{\omega}_1 \underbrace{\left[\widetilde{\omega}_1(\widetilde{\omega}_1+\widetilde{\omega}_2+\widetilde{\omega}_3)P^{\pi}P^{\pi} + (\widetilde{\omega}_1+\widetilde{\omega}_2)P_2^{\pi}\right]}_{=\mathbb{P}_2^{\pi,\widetilde{\omega}_2}} P_1^{\pi} \\ =& \widetilde{\omega}_3\mathbb{P}_0^{\pi,\omega}P_3^{\pi} + \widetilde{\omega}_2\mathbb{P}_1^{\pi,\omega}P_2^{\pi} + \widetilde{\omega}_1\mathbb{P}_2^{\pi,\omega}P_1^{\pi}\end{aligned}$$

The meaning of the modified distribution $\widetilde{\omega}$ is related to the fact that, once the trajectory evolved for k steps, the remaining $H-k$ are still sampled, but when the last step H is reached, then the agent stops repeating in any case.

Kemeny's constant computation The formula used to compute the Kemeny's constant from the transition Kernel $\mathbb{P}_H^{\pi,\omega}$ can be obtained thanks to the following Proposition (Kirkland, 2010).

Proposition C.1 (Kemeny's constant of an irreducible Markov Chain)**.** *Consider a Markov chain with an irreducible transition matrix P with eigenvalues $\lambda_1 = 1$ and $\lambda_i, i\in\{2,\dots,n\}$. The Kemeny constant of the Markov chain is given by*

$$Kem = \sum_{i=2}^{n}\frac{1}{1-\lambda_i}.$$

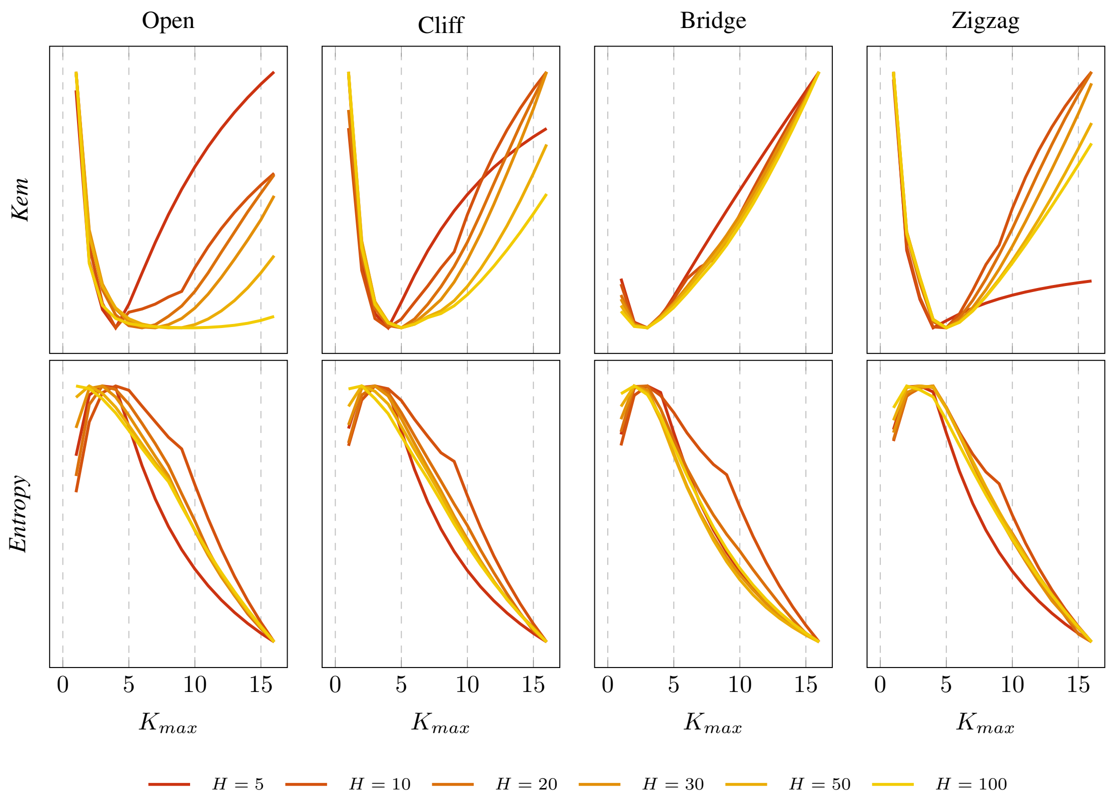


Figure C.1: *Normalized Kemeny's constant and entropy in Tabular environments as a function of the maximum persistence and horizon H.*

The introduction of the parameter H is necessary to retrieve an irreducible transition matrix $\mathbb{P}_H^{\pi,\omega}$ maintaining the Markov property.

In order to compute the curves of Kemeny's constant in Figure 7.2, we consider a K_{max} as a variable, and exploration is performed by a discrete uniform random variable in $\mathcal{O} = \bigcup_{k\in\mathcal{K}} \mathcal{O}^{(k)}$, i.e., the distribution π is uniform over the action space $\mathcal{A} = \{left, down, right, up\}$, and ω uniform over $\mathcal{K}$. In Figure C.1 we show the curves of Kemeny's constant and entropy with different values of K_{max} and H, and Figure 7.2 presented in section 7.6 refers to the same Kemeny's curves selected for $H = 30$.

As we can see in the figure, for each value of H there is a similar pattern: increasing K_{max}, the related values for Kemeny's constant initially tend to decrease, indicating that persistence helps for a faster exploration through the state space. Persisting actions for long times does not help exploration, since agents might be more frequently standing in front of walls. Consequently, depending on the different designs of the environments, Kemeny's values begin to increase. In the bottom plots of Figure C.1 we can observe also the curves related to the entropy induced by $\mathbb{P}_H^{\pi,\omega}$: again, the maximum value of entropy is attained by $K_{max} > 1$. However, the curves soon start to decrease dramatically, indicating that reaching distant states sooner is not strictly related to its visitation frequency.

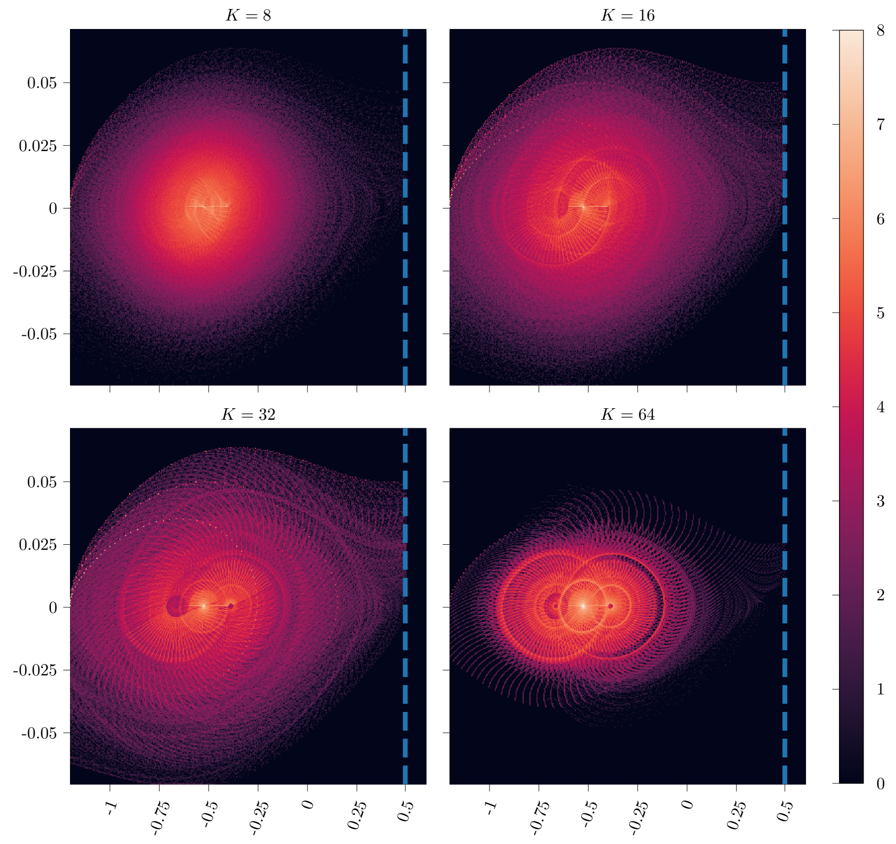


Figure C.2: *Visitation frequencies in Mountain Car environment of different random policies with fixed K persistences. The value represents in a logarithmic scale the counter of visited states. The x and y axes are respectively the position and the velocity of the car. The blue dotted line represents the goal. 10.000 episodes*

C.1.2 Further Exploration Advantages: MountainCar

In this section, we provide further evidence regarding the advantages of exploration. We study the effects of a persisted random policy on the MDP, i.e., a policy $\psi \in \Psi$ over persistence options $\mathcal{O}$ in Mountain Car. We have collected all the states traversed by a full random agent with different values of K_{max}, both with a fixed persistence (figure C.2) and a dynamic persistence (figure C.3). The figures represent the heatmaps of the visitation frequency in the state space. As we can see, when K_{max} is low the agent has less chance to reach the goal (represented by the blue dotted line). Increasing the persistence, the distribution over the states starts covering a wider region of the state space and reaching the goal with a higher probability.

We can observe that, even if the goal for some values of K is almost equally visited, with a fixed persistence we have a different distribution of visited states. Moreover, a fixed, high persistence does not provide sufficient exploration to reach the goal, as we can see in figure C.2 with $K = 64$, especially if compared to the dynamic version in figure C.3.

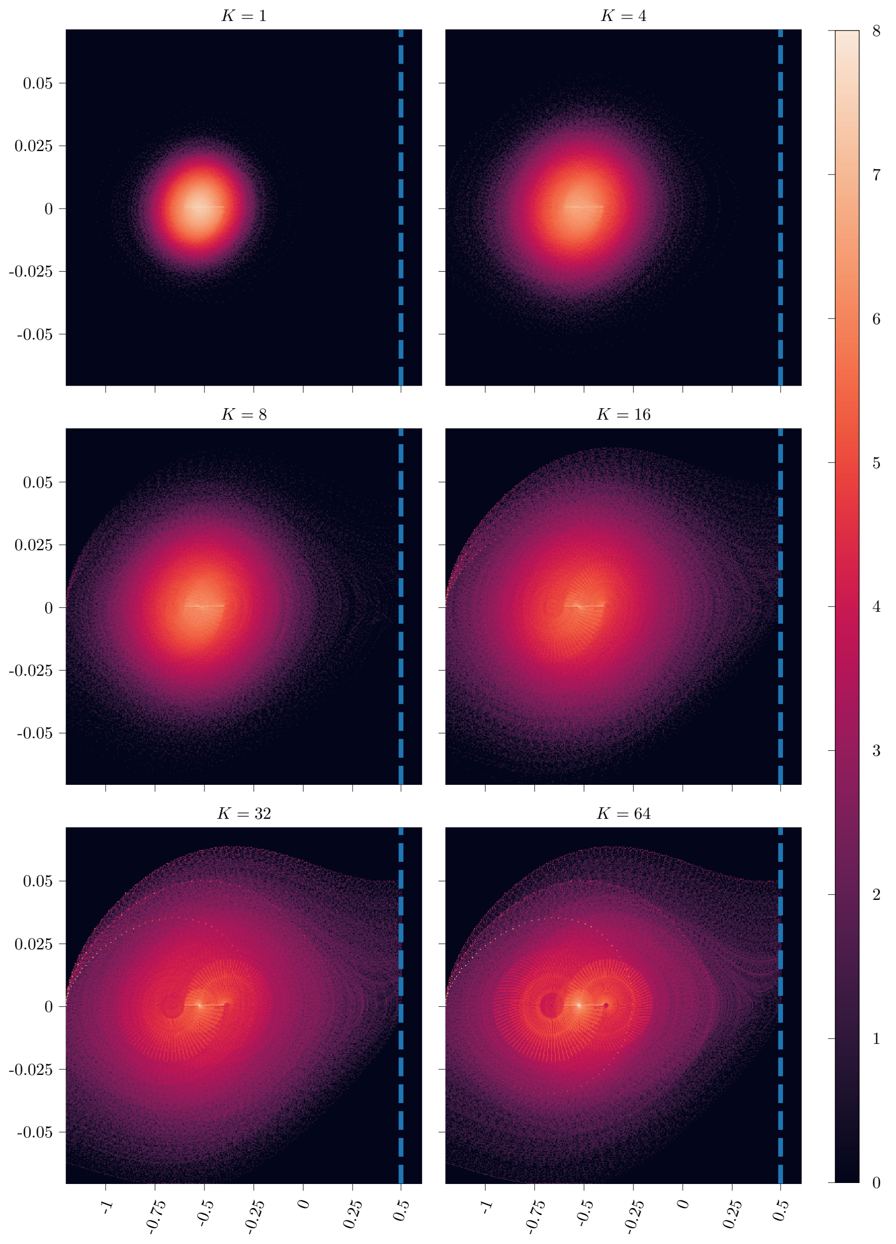


Figure C.3: *Visitation frequencies in Mountain Car environment of different random persistence options. The value represents in a logarithmic scale the counter of visited states. The x and y axes are respectively the position and the velocity of the car. The blue dotted line represents the goal. 10.000 episodes*

C.1.3 Synchronous Update: Additional Results

In this appendix, we report some additional information related to the experiments with synchronous Per$Q$-learning updates shown in Section 7.6.

The environment considered is depicted in Figure C.4: a 6x6 deterministic Gridworld, with holes. The set of (primitive) actions is $\mathcal{A} = \{left, down, right, up\}$. Transitions are deterministic. Going outside the borders and falling off a hole result in the punishment with a negative reward, respectively equal to -100 and -10 (hence, outer borders are not blocking the movement of the agent). The reward for reaching the goal instead is equal to +100. In all these cases the episode terminates; all other states result in a small negative reward (-1), to incentivize finding the shortest paths toward the goal.

In order to exploit only the convergence properties of Per$Q$-learning algorithm, without considering the exploration factor, we consider a synchronous learning framework: we assume to have access to the whole transition model, in such that, at each iteration t, we can collect an independent sample $s' \sim P(\cdot|s,a)$ for every state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$. Since the environment is deterministic, this means that we have access to the whole transition matrix P. For each simulation, the value estimation for each state-action pair (or state-option pair, in the case of Per$Q$-learning) is initialized sampling from a standard gaussian random variable. In each iteration of Q-learning, the algorithm performs a full update of the Q-function estimates. In each iteration of Per$Q$-learning, before performing the full update, the *primitive* tuples are combined, in order to collect a sample for each possible (s,a,k) pair in $\mathcal{S} \times \mathcal{A} \times \mathcal{K}$.

In the plots on Figure 7.3, we represented the overall convergence to $Q^\star$ and the convergence of the $Q_\mathcal{K}$-value function restricted on the k-persistent actions $\mathcal{O}^{(k)}$, as specified in Section 7.6.

The representations of the $k-$step value functions $V_k^\star$ are shown in Figure C.5, where $V_k^\star(s) = \max_{a\in\mathcal{A}} Q^\star(s,a,k)$. It is useful to remark that this value function does not coincide with the optimal value function in the $k-$persistent MDP $\mathcal{M}_k$, as $V_k^\star(s)$ represents the value function at the state s restricted to persist k times only the first action, and the following the optimal policy $\pi^\star$.

Parameters used for experiments:

- Initial estimation: $Q(s,a,k) \sim \mathcal{N}(0,1)\ \forall (s,a,k) \in \mathcal{S} \times \mathcal{A} \times \mathcal{K}$;
- Discount factor: $\gamma = 0.99$;
- Learning rate: $\alpha = 0.1$;
- Maximum number of iterations: 400 (plots truncated at 200).

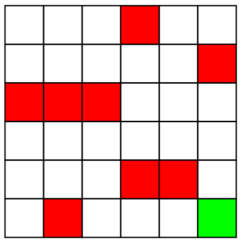


Figure C.4: *Grid environment representation. Red cells denote holes and green cells the goal.*

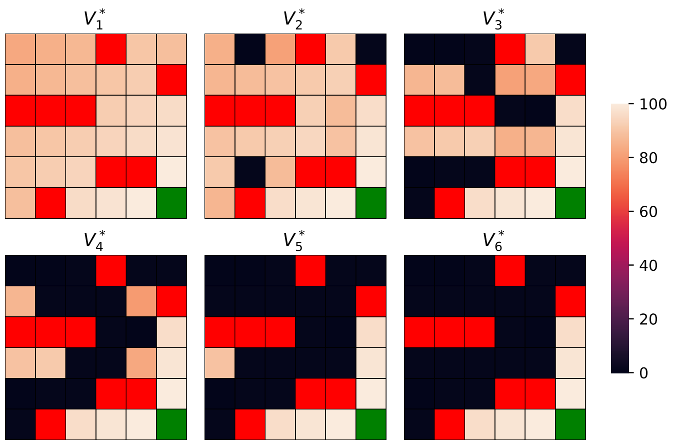


Figure C.5: *k-value function representation for different persistence values k in Gridworld environment ($K_{\max} = 6$). Red cells denote holes and green cells the goal.*

C.2 Details on Experimental Evaluation

In this appendix, we report more details about our experimental framework. In Appendix C.2.1 we provide more details about Per$Q$-learning and the tabular setting; in Appendix C.2.2 and C.2.3 we focus on PerDQN and the deep RL experiments, respectively on Mountain Car and Atari games.

C.2.1 Tabular Environments

The first environments tested are the deterministic 6x10 grids shown in Figure C.6 and presented in Biedenkapp et al. 2021. These environments are deterministic, and the outer borders block the agent from moving outside the grid (for example, an agent being at the top left cell will not move with an *Up* action). Falling in the holes (black cells in Figure C.6) results in a -1 reward, while the goal is worth a positive reward, equal to $+1$. All other states have no reward. Reaching a Hole or the Goal terminates an episode.

Along with these three environments, we tested also the results on FrozenLake, available among OpenAi gym toolkit (Brockman et al., 2016). The transition process and the rewards are the same as in the previous case; the only differences are the bigger map (16x16) and the presence of random holes, such that each run is performed on a new map. For each new random map generated, the probability for each tile to be a hole (or frozen, according to the environment description) is equal to 0.85.

Parameters:

- Initial estimation: $Q(s, a, k) \sim \mathcal{N}(0, 1)\ \forall (s, a, k) \in \mathcal{S} \times \mathcal{A} \times \mathcal{K}$;
- Discount factor: $\gamma = 0.99$;
- Learning rate: $\alpha = 0.01$;
- Maximum number of iterations: 6000 for FrozenLake, otherwise 600;
- Random policy probability: Exponentially decreasing: $\epsilon_t = 0.99^t$.

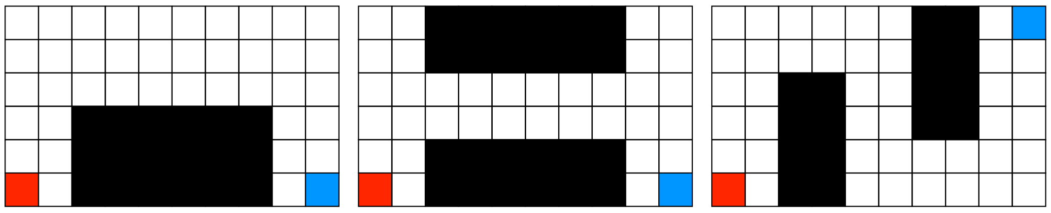

(a) *Cliff* **(b)** *Bridge* **(c)** *Zigzag*

Figure C.6: *Tabular Gridworld representation. Red cells denote the starting state and blue cells the goal state.*

C.2.2 MountainCar

For MountainCar experiments, the architecture chosen is an MLP: the first two hidden layers, consisting of 128 rectifier units, are shared among all persistences. The third hidden layer instead is diversified for each persistence value k, and each one is composed of 64 rectifier neurons and connected to three outputs, one for each action with its own persistence value.

The parameters adopted for the experiments are the following:

- Discount factor: $\gamma = 1$;
- Maximum number of iterations: 6×10^5 (truncated to 5×10^5 in the plots);
- Batch size: 32 for each persistent value;
- Replay buffer size: 50000 for each persistent value;
- Random policy probability: linearly decreasing, starting from $\epsilon_0 = 1$, to a final value $\epsilon_f = 0.01$, reached at 15% of the total number of iterations;
- Target update frequency: every 1000 steps;
- Train frequency: 1;
- Gradient clip: 10;
- Learning starts: 1000 (2000 only for Freeway);
- Loss function: Huber loss;
- Optimizer: Adam, with learning rate $\alpha = 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$;
- Prioritized replay buffers, of size 5×10^4 for each persistence value, and default prioritization coefficients $\alpha_p = 0.6$, $\beta_p = 0.4$;

C.2.3 Atari Games

For Atari games, the architecture chosen is based on that presented in Mnih et al. (2015), with three convolutional layers. the first hidden layer takes as input an $84 \times 84 \times 4$ image and convolves 32 filters of 8×8 with stride 4, with the application of rectifier nonlinearities. The second has 32 input channels, 64 output channels, a kernel size of 4 and a stride of 2, again with ReLu activations, as well as the third convolutional layer, with a kernel size of 3 and a stride of 1, and 64 output channels. The convolutional structure is shared among all persistences, while the fully-connected hidden layer, consisting of 128 rectifier units, is differentiated for each persistence value k. Each one of these layers is fully connected to the output layer, with a single output for each possible action.

The OpenAi Gym environments used are in the *deterministic-v0* version, which does not make merging operations among the 4 input frames, but considers only the last one.

The parameters adopted are the following:

- Discount factor: $\gamma = 0.99$;
- Maximum number of iterations: 2.5×10^6;
- Batch size: 32 for each persistent value;
- Replay buffer size: 50000 for each persistent value;
- Random policy probability: linearly decreasing, starting from $\epsilon_0 = 1$, to a final value $\epsilon_f = 0.01$, reached at 17% of the total number of iterations;
- Target update frequency: every 500 steps;
- Train frequency: 1;
- Gradient clip: 10;
- Learning starts: 1000 (2000 only for Freeway);
- Loss function: Huber loss;
- Optimizer: Adam, with learning rate $\alpha = 5 \times 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$;
- Prioritized replay buffers, of size 5×10^4 from each persistence value, and default prioritization coefficients $\alpha_p = 0.6$, $\beta_p = 0.4$.

C.3 Additional Results

In this appendix, we report more experimental results.

C.3.1 Tabular Environments

Here we consider the results related to the tabular environments. In Figure C.7 we present the full comparison among Per$Q$-learning-learning and TempoRL. As said in Section 7.8, we can observe a generally faster convergence of the former. However, TempoRL is more robust with higher persistences, as in complex environments such as ZigZag and FrozenLake the performance does not degrade as Per$Q$-learning learning.

Furthermore, we analyzed the impact of the Bootstrap operator: as an ablation study, we performed the same Per$Q$-learning learning evaluation without this feature. As a result, we can see that the returns are dramatically worse. Often, they perform even worse than classic Q learning: the reason for this behavior is due to the fact that, if a certain persistence value k is not feasible for a state action pair (s, a) (because of the geometry of the environment), its related value estimation $Q(s, a, k)$ will never be updated, and the algorithm may keep choosing it as the best one among the others. In TempoRL, albeit the absence of a Bootstrap, the update of the *skip-value* function is instead updated by using the standard action-value function, improving the estimation.

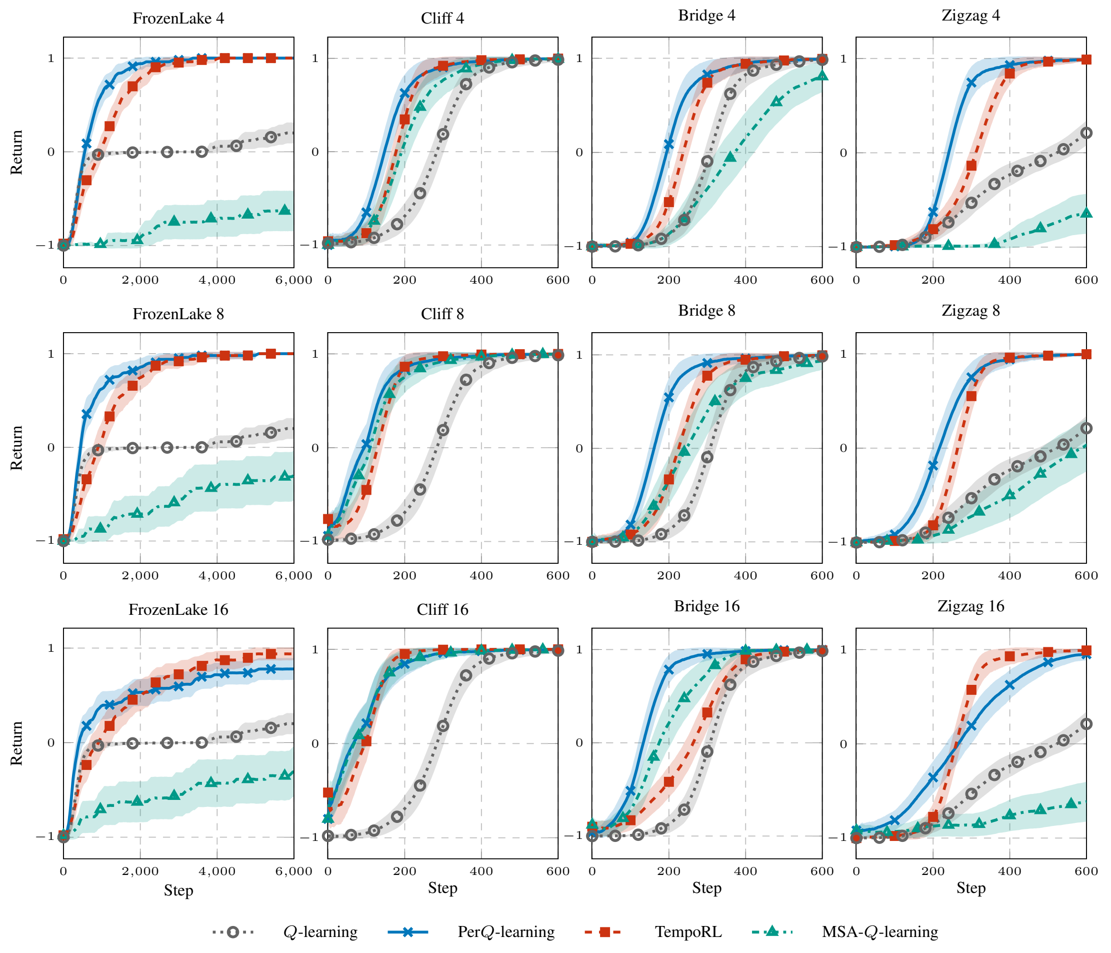


Figure C.7: *Results of Q-learning, TempoRL, PerQ-learning and MSA-Q-learning in different tabular environments and maximum persistences. On each row, a different maximum persistence is selected for both algorithms. 50 runs (avg± 95% c.i.).*

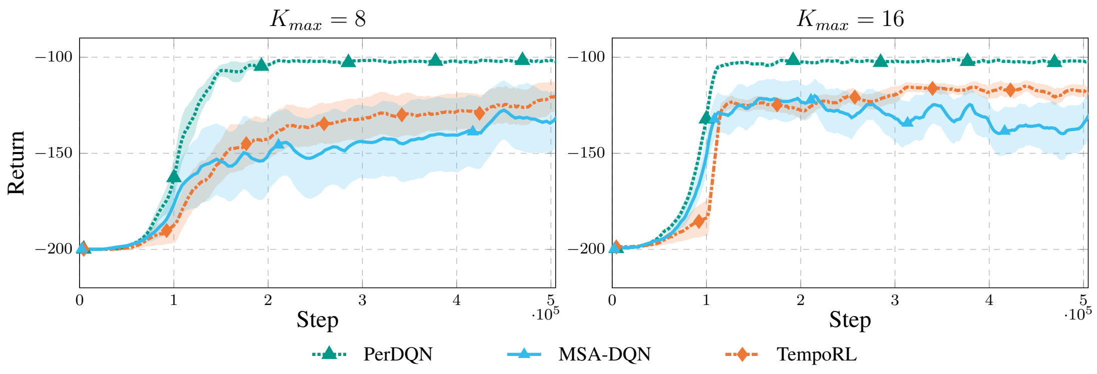


Figure C.8: *PerDQN additional results on MountainCar. Return comparison of PerDQN with and without bootstrap (MSA-DQN) and TempoRL. 20 runs (avg ± 95%c.i.)*

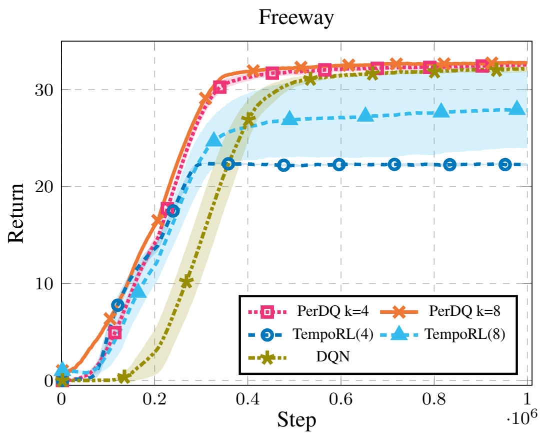


Figure C.9: *PerDQN additional results on Freeway. Return comparison of PerDQN and TempoRL with maximum persistence* 4 *and* 8*. Parentheses denote the maximum persistence chosen. 5 runs (avg ± 95%c.i.)*

C.3.2 PerDQN Additional Results

PerDQN with increased K_{max} in Atari games Here we investigate the effects of an increased value of K_{max} in Atari games: in Figure C.9 we show the results obtained on Freeway, by increasing the maximum persistence to 8 for both TempoRL and PerDQN. The same $K_{max} = 8$ for PerDQN has been tested on the other Atari games, and shown in Figure C.10 (we excluded Seaquest, as we have already shown that persistence in this environment is detrimental for learning. As we can see, an increased maximum persistence $K_{max} = 8$ does not provide improvements w.r.t. $K_{max} = 4$ for the majority of the environments. A slight improvement can be seen for Freeway, while TempoRL, albeit providing a better learning curve than the one obtained with a skip size equal to 4, is still unable to converge to the optimal policy.

MountainCar, PerDQN without Bootstrap Here we investigate the effects of the Bootstrap operator on PerDQN. To do so, we experimentally evaluated the algorithm without this feature on MountainCar and on the Atari games where persistence seemed to have beneficial effects (hence, with the exclusion of the Seaquest environment). The

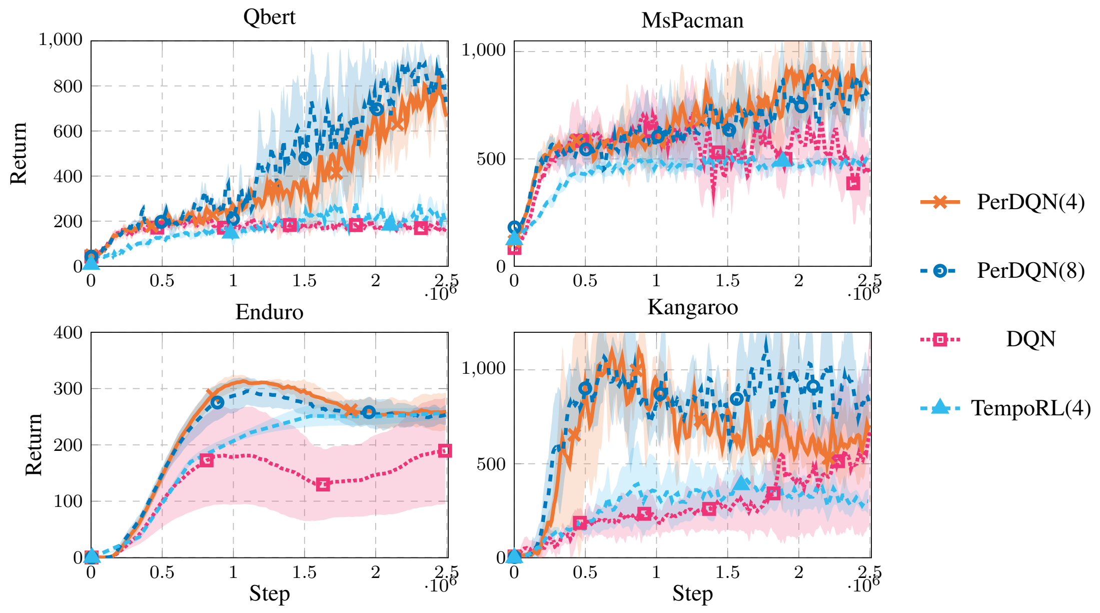


Figure C.10: *Atari games results for DQN and PerDQN, with $K_{\max} = 4$ and 8. 5 runs (avg$\pm$ 95% c.i.).*

results related to the Atari environments are shown in Figure 7.6, while Mountain-Car performances are shown in Figure C.8: the version without bootstrap recalls the same contribution brought in Schoknecht and Riedmiller (2003) with MSA-Q-learning, hence it is here denoted as MSA-DQN. As we can see, bootstrap is an essential feature for PerDQN to converge rapidly to the optimal policy and to be robust. Indeed, without the bootstrap, the performances are worse, similar to TempoRL, and their variance is dramatically higher.

C.3.3 PerQ-learning: computational times

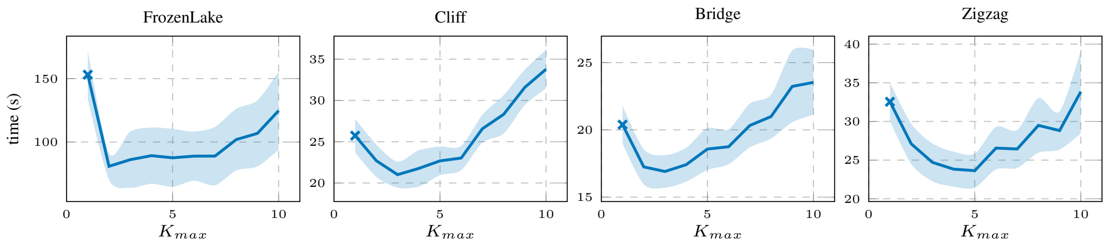


Figure C.11: *Computational times required for a complete run of PerQ-learning on tabular environments with different Maximum Persistences. 20 runs. avg $\pm$ 95% c.i.*

We ran Per$Q$-learning on the tabular environments for different values of K_{max} keeping track of the training time. We fixed the number of collected trajectories, but the trajectories can be of different lengths since the environments are goal-based. In Figure C.11, we observe that in all environments the minimum is attained by a value of $K_{max} > 1$. This means that the computational overhead due to using larger values of K_{max} is compensated by a faster learning speed, leading to shorter trajectories overall. Note that $K_{max} = 1$ corresponds to classic Q-learning.

C.3.4 Experiments with the same Number of Updates and Comparison with ez-greedy Exploration

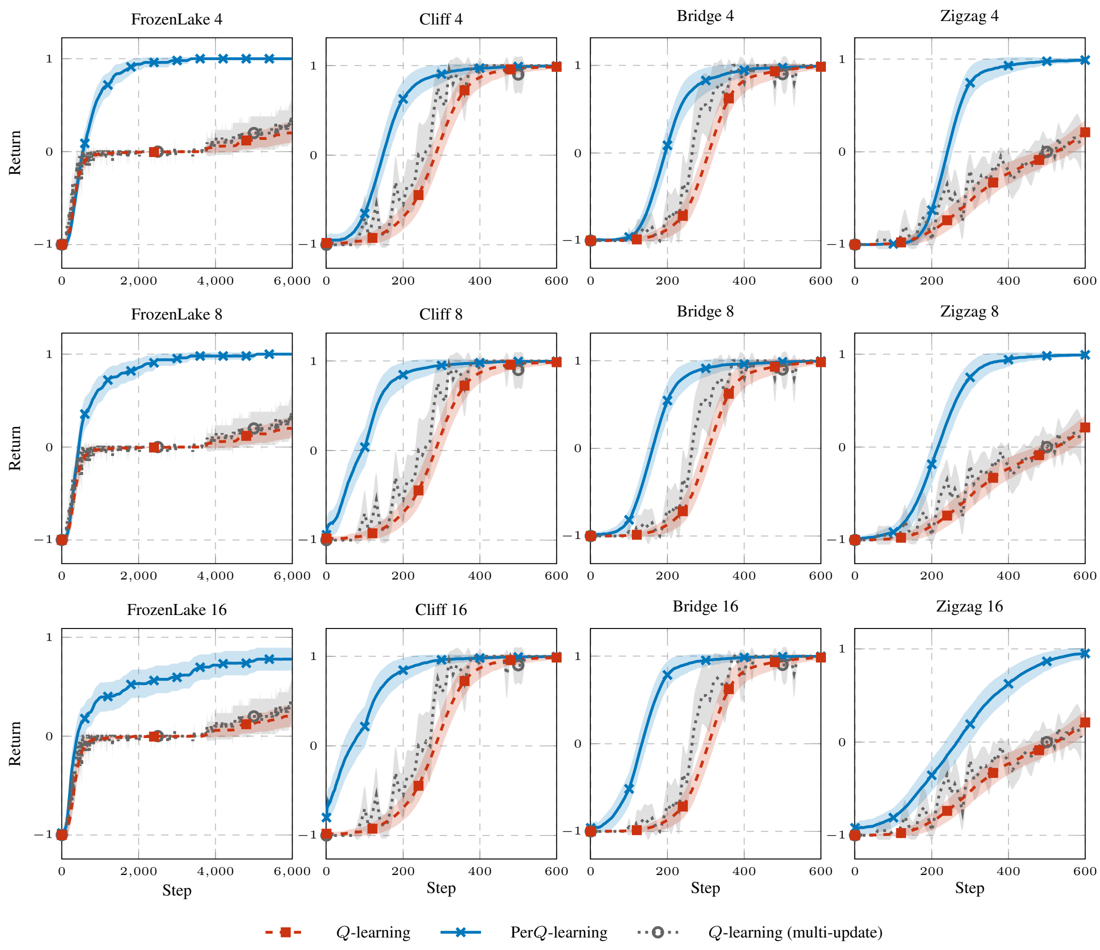


Figure C.12: *Results of Q-learning, TempoRL, Per$Q$-learning and MSA-Q-learning in different tabular environments and maximum persistences. On each row, a different maximum persistence is selected for both algorithms. 50 runs (avg± 95% c.i.).*

In Algorithm 10, we can see that the number of total updates related is $\mathcal{O}(K_{max}^3)$. One might wonder if standard Q-learning can attain the same learning advantages with an increased number of updates: in Figure C.12, we compare the results obtained for Per$Q$-learning with Q-learning, where the number of updates per each step has been increased for K_{max} times (the row denotes the value of K_{max}). Hence, the two algorithms have the same amount of updates. However, the increasing number of updates is not directly related to a learning improvement. If we consider Q-learning, increasing the number of updates (related to the same observed tuples) is equivalent to an increase in the learning rate, which is not necessarily related to better learning performances. Indeed, the performances of Q-learning in its multiple-update version are not statistically better than the single-update version, only less robust.

Furthermore, the presence of multiple replay buffers allows reducing the total number of updates linear in K_{max} in PerDQN, which is the most suited for real-world

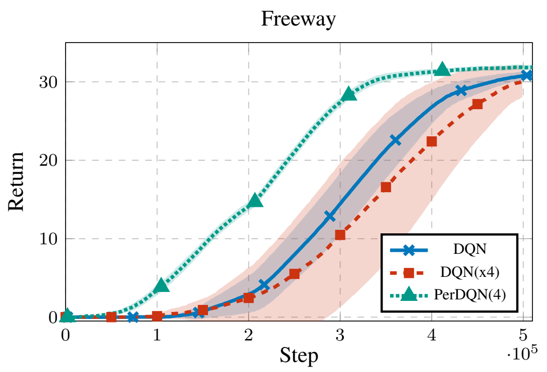


Figure C.13: *Freeway results: PerDQN comparison with DQN with the same global amount of tuples sampled per update as PerDQN (DQN(x4) denotes DQN where the batch size is 4 times the one adopted for classic DQN, hence with the same global batch size as PerDQN with $K_{max} = 4$).*

scenarios with large state spaces. In Figure C.14, we compare PerDQN with other baselines on the MountainCar environment: In the top plot, we show a comparison between our proposed approach and DQN with the same number of updates as PerDQN (denoted as DQN(xK)). In PerDQN, the batch size for each replay buffer is 32: in the new DQN runs, the total batch size for an update has been increased to 32*K_{max}.

In the middle plot, we compare PerDQN with a vanilla DQN employing ϵz-greedy exploration (Dabney et al., 2020). In the implementation of ϵz-greedy DQN, the exploration is performed similarly as in PerDQN, as the persistence is sampled from a discrete uniform distribution in $1, \ldots, K_{max}$, where K_{max} has been set to 8 and 16.

In the bottom plot, ϵz-greedy DQN is run with the same global batch size per update as in PerDQN.

In Figure C.13, we compare PerDQN (K_{max}=4) with DQN on Freeway, with an increased batch size of factor 4, in such a way that the total number of samples per update is the same.

In a similar fashion as with standard Q-learning, in Figure C.14 the DQN curve related to $8x$ the sample size is worse (on average) than the standard version, while the $16x$ experiments see a slight improvement (with no statistical evidence). In any case, PerDQN outperforms all the compared methods. The same holds for Freeway (Figure C.13), where augmenting the DQN batch size by a factor of 4 does not provide improvements in the performances.

In the middle and bottom plots of Figure C.14, we can see that the exploration with persistence alone (through ϵz-greedy exploration) is not enough to provide the same improvement in the learning capabilities, differently from PerDQN. Furthermore, increasing the number of updates can slightly help to learn, but the resulting learning curves are still largely dominated by PerDQN.

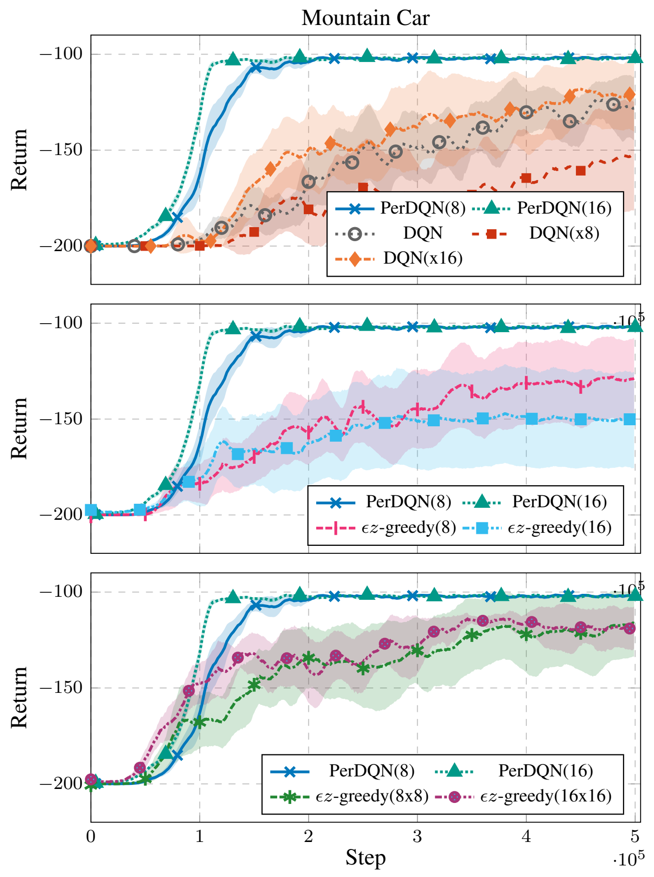


Figure C.14: *MountainCar results for DQN, PerDQN and ϵz-greedy DQN (10 runs, avg$\pm$ 95% c.i.).*
Top Figure: PerDQN comparison with DQN with the same global amount of tuples sampled per update as PerDQN (e.g. DQN(x8) denotes DQN where the batch size is 8 times the one adopted for classic DQN, hence with the same global batch size as PerDQN with $K_{max} = 8$).
Middle figure: PerDQN comparison with ϵz-greedy DQN, where parenthesis denotes the maximum persistence in the random sampling.
Bottom figure: PerDQN comparison with ϵz-greedy DQN, with the same global amount of tuples sampled per update as PerDQN (e.g. ϵz-greedy(8x8) denotes ϵz-greedy DQN with maximum persistence for exploration equal to 8, and the batch size is 8 times the one adopted for classic DQN, hence with the same global batch size as PerDQN with $K_{max} = 8$)

APPENDIX $\mathcal{D}$

Persistor Policy Gradient

In this appendix, we report some preliminary results regarding persistence selection via a policy-based approach. In this way, it is possible to extend the selection of the duration of the actions through a secondary policy, conditioned on the current state-action pair. In the following, we will show that also the selection of the persistence can be learned following the gradient of the return.

In our framework, a (Markovian, stationary) *policy* $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$ defines the probability distribution over $\mathcal{A}$ given the current state in $\mathcal{S}$. Once the action a is selected, the amount of steps it is kept is decided by a *persistor* $\xi : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{K}}$, which is defined over a *persistence space* $\mathcal{K}$. In simulated environments, the only feasible persistence space is discrete, with $\mathcal{K} \subseteq \mathbb{N}^{+}$. Hence, the persistence is related to a natural amount of discrete steps, each one performed with a fixed frequency[1]. As a consequence, we can exploit the relation between the Q and the V functions is completed thanks to the $k-$step Bellman Expectation operator defined in 6.4. The only difference is the dependency on the *policy-persistor* pair (π, ξ):

$$Q^{\pi,\xi}(s,a,k) = r_k(s,a) + \gamma^k \int_{\mathcal{S}} P_k(s'|s,a) V^{\pi,\xi}(s')\, \mathrm{d}s' \tag{D.1}$$

Furthermore, the *advantage function* $A^{\pi,\xi}$ is defined as:

$$A^{\pi,\xi}(s,a,k) = Q^{\pi,\xi}(s,a,k) - V^{\pi,\xi}(s)$$

[1]The formulation holds even considering a general setting with a Continuous-Time MDP, where $\mathcal{K} = \mathbb{R}^{+}$; in this case, the persistence of an action is equivalent to its true duration in time.

D.1 Occupancy Measures

In this section, we extend the concepts of transitions Kernels and trajectory probability introduced in Section 2.2.4 to the persistent case. This process is straightforward, with a single exception: the multi-step transitions have a variable length (being the decision process a Semi-MDP), and the related discounts depend on the persistence selected. Hence, we can ease the following computations by including the discounts in the transitions: a discounted transition Kernel induced by a pair (π, ξ) can be therefore defined as

$$\widetilde{p}_{\pi,\xi}(\cdot|s) = \int_{\mathcal{A}} \pi(\,\mathrm{d}a|s) \int_{\mathcal{K}} \xi(\,\mathrm{d}k|s,a)\gamma^k P_k(\cdot|s,a) \tag{D.2}$$

Remark D.1. *$\widetilde{p}_{\pi,\xi}$ is a measure, but not a distribution, as it is not normalized due to the presence of the mentioned discount factor. Furthermore, this distribution ignores the states visited during the persistence, but only considers the final state, regardless of the number of steps needed to reach it. The related normalized distribution trivially consists of the same definition without the discount factor:*

$$p_{\pi,\xi}(\cdot|s) = \int_{\mathcal{A}} \pi(\,\mathrm{d}a|s) \int_{\mathcal{K}} \xi(\,\mathrm{d}k|s,a) P_k(\cdot|s,a) \tag{D.3}$$

In the following sections, we will make an abuse of notation and denote as $\mathbb{E}_{x\sim p}[f(x)] = \int_X p(\,\mathrm{d}x)f(x)$, even if p is a positive non-normalized Radon measure.

The consequent $t-$step transition kernels can be then defined as in the standard way $\forall s \in \mathcal{S}$ and $\forall t \geq 1$:

$$\widetilde{p}^{1}_{\pi,\xi}(\cdot|s) = \widetilde{p}_{\pi,\xi}(\cdot|s) \tag{D.4}$$

$$\widetilde{p}^{t+1}_{\pi,\xi}(\cdot|s) = \int_{\mathcal{S}} \widetilde{p}^{t}_{\pi,\xi}(s'|s)\widetilde{p}_{\pi,\xi}(\cdot|s')ds'. \tag{D.5}$$

These sub-distributions can be used to define the discounted state-occupancy measure $\delta^s_{\pi,\xi}$:

$$\widetilde{\delta}^s_{\pi,\xi}(\cdot) = \frac{1-\gamma}{\gamma}\sum_{t=1}^{\infty} \widetilde{p}^{t}_{\pi,\xi}(\cdot|s) \tag{D.6}$$

The *state-occupancy measure* $\delta^\mu_{\pi,\xi}$ is defined $\forall s \in \mathcal{S}$ as:

$$\widetilde{\delta}^\mu_{\pi,\xi}(s) = (1-\gamma)\mu(s) + \gamma \int_{\mathcal{S}} \mu(s_0)\widetilde{\delta}^{s_0}_{\pi,\xi}(s)\,\mathrm{d}s_0 \tag{D.7}$$

Through the composition of $\delta^\mu_{\pi,\xi}$, a policy π and persistor ξ, we can also define the occupancy on $\mathcal{S}\times\mathcal{A}$ or $\mathcal{S}\times\mathcal{A}\times\mathcal{K}$:

$$\begin{aligned}\widetilde{\nu}^\mu_{\pi,\xi}(s,a) &:= \widetilde{\delta}^\mu_{\pi,\xi}(s)\pi(a|s)\\ \widetilde{\zeta}^\mu_{\pi,\xi}(s,a,k) &:= \widetilde{\delta}^\mu_{\pi,\xi}(s)\pi(a|s)\xi(k|s,a)\end{aligned}$$

We can also consider the distribution $\rho_{\pi,\xi}$ over the trajectories up to a horizon T. In this context, we need to consider the *invocation times* $t_i,\ i = 0,\dots,I(T)$, that consist in the time steps in which the action and the persistence are sampled. $I(T)$ is the total number of invocations in the trajectory such that the total steps in the trajectories reach T, and the set of invocation times is denoted as $\mathbb{T} = \{t_i\}_{i=0}^{I(T)-1}$. As a consequence, the distribution over finite trajectories induced by policy π and persistor ξ is defined as:

$$\begin{aligned}\rho_{\pi,\xi}(\tau) = \mu(s_0)\Big(\prod_{i=0}^{I(T)-2} \pi(a_{t_i}|s_{t_i})\xi(k_{t_i}|s_{t_i}, a_{t_i})P_{k_{t_i}}(s_{t_{i+1}}|s_{t_i}, a_{t_i})\Big) \\ \pi(a_{t_{I(T)-1}}|s_{t_{I(T)-1}})\xi(k_{t_{I(T)-1}}|s_{t_{I(T)-1}}, a_{t_{I(T)-1}})\end{aligned} \tag{D.8}$$

The related discounted sub-distribution is the discounted version of Equation D.8; namely:

$$\begin{aligned}\widetilde{\rho}_{\pi,\xi}(\tau) = \mu(s_0)\Big(\prod_{i=0}^{I(T)-2} \pi(a_{t_i}|s_{t_i})\xi(k_{t_i}|s_{t_i}, a_{t_i})\gamma^{k_{t_i}}P_{k_{t_i}}(s_{t_{i+1}}|s_{t_i}, a_{t_i})\Big) \\ \pi(a_{t_{I(T)-1}}|s_{t_{I(T)-1}})\xi(k_{t_{I(T)-1}}|s_{t_{I(T)-1}}, a_{t_{I(T)-1}})\gamma^{k_{t_{I(T)-1}}}\end{aligned}$$

Trivially, it holds that:

$$\frac{\widetilde{\rho}_{\pi,\xi}(\tau)}{\rho_{\pi,\xi}(\tau)} = \gamma^T \tag{D.9}$$

Finally, when expectations are taken into account, we will use the notation τ_t to denote the expectation among the space of trajectories $\mathcal{T}$ from time 0 up to time t, and with $\tau_{i:t}$ $0 < i \leq t$ the expectation on (partial) trajectories $(s_i, a_i, k_i, s_{i+k_i}, \dots, s_t, a_t, k_t)$, conditioned on s_{i-1}, a_{i-1}.

Thanks to all these definitions, it is possible to write the expected return in a compact form:

Proposition D.1.

$$J(\pi, \xi) = \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{s,a,k\sim\widetilde{\zeta}^{\mu}_{\pi,\xi}}[r_k(s,a)] = \mathop{\mathbb{E}}_{\tau\sim\rho_{\pi,\xi}}[R(\tau)] \tag{D.10}$$

Proof. Starting with the first equality: we start by applying Lemma D.6 using $f(s) = V^{\pi,\xi}(s)$ and $g(s) = \mathbb{E}_{\substack{a\sim\pi(\cdot|s)\\k\sim\xi(\cdot|s,a)}}[r_k(s,a)]$; consequently:

$$V^{\pi,\xi}(s) = g(s) + \frac{\gamma}{1-\gamma}\int_{\mathcal{S}} \widetilde{\delta}^{s}_{\pi,\xi}(s')g(s')\,\mathrm{d}s'$$

Hence, the following holds:

$$\begin{aligned}J(\pi,\xi) &= \int_{\mathcal{S}} \mu(s)V^{\pi,\xi}(s)\,\mathrm{d}s \\ &= \int_{\mathcal{S}} \mu(s)g(s)\,\mathrm{d}s + \frac{\gamma}{1-\gamma}\int_{\mathcal{S}}\mu(s_0)\int_{\mathcal{S}}\widetilde{\delta}^{s_0}_{\pi,\xi}(s)g(s)\,\mathrm{d}s\,\mathrm{d}s_0 \\ &= \int_{\mathcal{S}}\big[\mu(s) + \frac{\gamma}{1-\gamma}\int_{\mathcal{S}}\mu_{s_0}d^{s_0}_{\pi,\xi}(s)\,\mathrm{d}s_0\big]g(s)\,\mathrm{d}s \\ &= \frac{1}{1-\gamma}\int_{\mathcal{S}}\widetilde{\delta}^{\mu}_{\pi,\xi}(s)g(s)\,\mathrm{d}s \\ &= \frac{1}{1-\gamma}\int_{\mathcal{S}}\widetilde{\delta}^{\mu}_{\pi,\xi}(s)\int_{\mathcal{A}}\pi(\,\mathrm{d}a|s)\int_{\mathcal{K}}\xi(\,\mathrm{d}k|s,a)r_k(s,a)\,\mathrm{d}s,\end{aligned} \tag{D.11}$$

where in D.11 we used Equation D.7.

For the second equality, we consider Corollary D.8:

$$\begin{aligned}
J(\pi,\xi) &= \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{s,a,k\sim\zeta^{\mu}_{\pi,\xi}}[r_k(s,a)] = \sum_{i=0}^{\infty}\mathop{\mathbb{E}}_{\tau_{t_i}\sim\widetilde{\rho}_{\pi,\xi}}[r_{k_{t_i}}(s_{t_i},a_{t_i})] \\
&= \sum_{i=0}^{\infty}\mathop{\mathbb{E}}_{\tau_{t_i}\sim\rho_{\pi,\xi}}[\gamma^{t_i} r_{k_{t_i}}(s_{t_i},a_{t_i})] \qquad \text{(D.12)} \\
&= \mathop{\mathbb{E}}_{\tau\sim\rho_{\pi,\xi}}[\sum_{i=0}^{\infty}\gamma^{t_i} r_{k_{t_i}}(s_{t_i},a_{t_i})],
\end{aligned}$$

where in Equation D.12, Lemma D.9 was applied. □

D.2 Persistor Gradient Theorem

In this section, we consider *parametric policies* and *parametric persistors*: Let $\Theta \subseteq \mathbb{R}^m$ and $\Omega \subseteq \mathbb{R}^n$ be parameter spaces for some $m, n \in \mathbb{N}$. The set of policies that can be included is in the class $\Pi_\Theta = \{\pi_{\boldsymbol{\theta}} : \mathcal{S} \to \Delta_{\mathcal{A}} | \boldsymbol{\theta} \in \Theta\}$. In the same way, $\Xi_\Omega = \{\xi_{\boldsymbol{\omega}} : \mathcal{S}\times\mathcal{A} \to \Delta_{\mathcal{K}} | \boldsymbol{\omega} \in \Omega\}$ is the persistor class parametrized by Ω. A very important result in standard policy-based RL is Policy Gradient Theorem (Theorem 3.8). In the same fashion, we can extend the gradient computation to the persistor.

Notation in the following, we will often abbreviate the dependence to $\pi_{\boldsymbol{\theta}}$ and $\xi_{\boldsymbol{\omega}}$ respectively in $\boldsymbol{\theta}$ and $\boldsymbol{\omega}$. For example, $J(\pi_{\boldsymbol{\theta}}, \xi_{\boldsymbol{\omega}})$ will be denoted as $J(\boldsymbol{\theta}, \boldsymbol{\omega})$ and $\widetilde{\delta}_{\pi_{\boldsymbol{\theta}},\xi_{\boldsymbol{\omega}}}$ is abbreviated into $\widetilde{\delta}_{\boldsymbol{\theta},\boldsymbol{\omega}}$.

Theorem D.2 (Persistor-Policy Gradient Theorem)**.** *Let $\mathcal{M}$ be an MDP. Let $\pi_{\boldsymbol{\theta}} : \mathcal{S} \to \Delta_{\mathcal{A}}$ be differentiable w.r.t. $\boldsymbol{\theta}$, and $\xi_{\boldsymbol{\omega}} : \mathcal{S}\times\mathcal{A} \to \Delta_{\mathcal{K}}$ be differentiable w.r.t. $\boldsymbol{\omega}$. Then:*

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta},\boldsymbol{\omega}) = \frac{1}{1-\gamma}\int_{\mathcal{S}}\widetilde{\delta}^{\mu}_{\boldsymbol{\theta},\boldsymbol{\omega}}(\,\mathrm{d}s)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}(\,\mathrm{d}a|s)\int_{\mathcal{K}}\xi_{\boldsymbol{\omega}}(\,\mathrm{d}k|s,a)\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a|s) \quad \text{(D.13)}$$

$$\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\theta},\boldsymbol{\omega}) = \frac{1}{1-\gamma}\int_{\mathcal{S}}\widetilde{\delta}^{\mu}_{\boldsymbol{\theta},\boldsymbol{\omega}}(\,\mathrm{d}s)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}(\,\mathrm{d}a|s)\int_{\mathcal{K}}\xi_{\boldsymbol{\omega}}(\,\mathrm{d}k|s,a)\nabla_{\boldsymbol{\omega}}\log\xi_{\boldsymbol{\omega}}(k|s,a) \quad \text{(D.14)}$$

Proof. Trivially, taking into account the Bellman Expectation Equation in D.1, we obtain, for all $s \in \mathcal{S}, a \in \mathcal{A}, k \in \mathcal{K}$:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} Q^{\boldsymbol{\omega}}(s,a,k) &= \nabla_{\boldsymbol{\theta}}\big[R_k(s,a) + \gamma^k\int_{\mathcal{S}}P_k(\,\mathrm{d}s'|s,a)V^{\boldsymbol{\omega}}(s')\big] \\
&= \gamma^k\int_{\mathcal{S}}P_k(\,\mathrm{d}s'|s,a)\nabla_{\boldsymbol{\theta}}V^{\boldsymbol{\theta},\boldsymbol{\omega}}(s')
\end{aligned}$$

and the same holds for the gradient w.r.t. $\boldsymbol{\omega}$. Moreover, for all $s \in \mathcal{S}$:

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} V^{\boldsymbol{\theta},\boldsymbol{\omega}}(s) &= \nabla_{\boldsymbol{\theta}} \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(\,\mathrm{d}a|s) \underbrace{\int_{\mathcal{K}} \xi_{\boldsymbol{\omega}}(\,\mathrm{d}k|s,a) Q(s,a,k)}_{:=U^{\boldsymbol{\theta},\boldsymbol{\omega}}(s,a)} \\
&= \underbrace{\int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\,\mathrm{d}a|s) U^{\boldsymbol{\theta},\boldsymbol{\omega}}(s,a)}_{:=H(s)} + \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(\,\mathrm{d}a|s) \int_{\mathcal{K}} \xi_{\boldsymbol{\omega}}(\,\mathrm{d}k|s,a) \nabla_{\boldsymbol{\theta}} Q^{\boldsymbol{\theta},\boldsymbol{\omega}}(s,a,k) \\
&= H(s) + \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(\,\mathrm{d}a|s) \int_{\mathcal{K}} \xi_{\boldsymbol{\omega}}(\,\mathrm{d}k|s,a) \gamma^k \int_{\mathcal{S}} P_k(\,\mathrm{d}s'|s,a) \nabla_{\boldsymbol{\theta}} V^{\boldsymbol{\theta},\boldsymbol{\omega}}(s') \\
&= H(s) + \int_{\mathcal{S}} \widetilde{p}_{\boldsymbol{\theta},\boldsymbol{\omega}}(\,\mathrm{d}s'|s) \nabla_{\boldsymbol{\theta}} V^{\boldsymbol{\theta},\boldsymbol{\omega}}(s') \\
&= H(s) + \frac{\gamma}{1-\gamma} \int_{\mathcal{S}} \widetilde{\delta}^{s}_{\boldsymbol{\theta},\boldsymbol{\omega}}(\,\mathrm{d}s') \int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\,\mathrm{d}a|s') U^{\boldsymbol{\theta},\boldsymbol{\omega}}(s',a) \qquad \text{(D.15)} \\
&= H(s) + \frac{\gamma}{1-\gamma} \int_{\mathcal{S}} \widetilde{\delta}^{s}_{\boldsymbol{\theta},\boldsymbol{\omega}}(\,\mathrm{d}s') H(s'),
\end{aligned}
$$

where in D.15 we used Lemma D.6. Analogously:

$$
\begin{aligned}
\nabla_{\boldsymbol{\omega}} V^{\boldsymbol{\theta},\boldsymbol{\omega}}(s) &= \nabla_{\boldsymbol{\omega}} \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(\,\mathrm{d}a|s) \int_{\mathcal{K}} \xi_{\boldsymbol{\omega}}(\,\mathrm{d}k|s,a) Q(s,a,k) \\
&= \underbrace{\int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(\,\mathrm{d}a|s) \int_{\mathcal{K}} \nabla_{\boldsymbol{\omega}} \xi_{\boldsymbol{\omega}}(\,\mathrm{d}k|s,a) Q^{\boldsymbol{\theta},\boldsymbol{\omega}}(s,a,k)}_{:=M(s)} + \int_{\mathcal{S}} \widetilde{p}_{\boldsymbol{\theta},\boldsymbol{\omega}}(\,\mathrm{d}s'|s) \nabla_{\boldsymbol{\omega}} V^{\boldsymbol{\theta},\boldsymbol{\omega}}(s') \\
&= M(s) + \frac{\gamma}{1-\gamma} \int_{\mathcal{S}} \widetilde{\delta}^{s}_{\boldsymbol{\theta},\boldsymbol{\omega}}(\,\mathrm{d}s') M(s'). \qquad \text{(D.16)}
\end{aligned}
$$

Hence:

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta},\boldsymbol{\omega}) &= \nabla_{\boldsymbol{\theta}} \int_{\mathcal{S}} \mu(\,\mathrm{d}s_0) V^{\boldsymbol{\theta},\boldsymbol{\omega}}(s_0) \\
&= \int_{\mathcal{S}} \mu(\,\mathrm{d}s_0) \nabla_{\boldsymbol{\theta}} V^{\boldsymbol{\theta},\boldsymbol{\omega}}(s_0) \\
&= \int_{\mathcal{S}} \mu(\,\mathrm{d}s_0) \left[H(s_0) + \frac{\gamma}{1-\gamma} \int_{\mathcal{S}} \widetilde{\delta}^{s_0}_{\boldsymbol{\theta},\boldsymbol{\omega}}(\,\mathrm{d}s) H(s) \right] \\
&= \frac{1}{1-\gamma} \left[(1-\gamma) \int_{\mathcal{S}} \mu(\,\mathrm{d}s) H(s) + \gamma \int_{\mathcal{S}} \mu(\,\mathrm{d}s_0) \int_{\mathcal{S}} \widetilde{\delta}^{s_0}_{\boldsymbol{\theta},\boldsymbol{\omega}}(\,\mathrm{d}s) H(s) \right] \\
&= \frac{1}{1-\gamma} \int_{\mathcal{S}} \widetilde{\delta}^{\mu}_{\boldsymbol{\theta},\boldsymbol{\omega}}(\,\mathrm{d}s) H(s) \qquad \text{(D.17)} \\
&= \frac{1}{1-\gamma} \int_{\mathcal{S}} \widetilde{\delta}^{\mu}_{\boldsymbol{\theta},\boldsymbol{\omega}}(\,\mathrm{d}s) \int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\,\mathrm{d}a|s) U^{\boldsymbol{\theta},\boldsymbol{\omega}}(s,a) \\
&= \frac{1}{1-\gamma} \int_{\mathcal{S}} \widetilde{\delta}^{\mu}_{\boldsymbol{\theta},\boldsymbol{\omega}}(\,\mathrm{d}s) \int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\,\mathrm{d}a|s) \int_{\mathcal{K}} \xi_{\boldsymbol{\omega}}(\,\mathrm{d}k|s,a) Q(s,a,k) \qquad \text{(D.18)}
\end{aligned}
$$

where in D.17 we used the definition in Equation D.7, the conclusion is obtained via log-trick, and the same computations can be done to compute $\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\theta},\boldsymbol{\omega})$. $\square$

D.3 Persistor Classes

Among the discrete distribution classes, one might select *finite support* ones; however, the choice of a maximum persistence can be non trivial, and has to be fine-tuned by hand. Consequently, we can choose discrete distribution with infinite support. One of the straightforward choices is the Poisson distribution, with is differentiable and allows to have a mode greater than 1

D.3.1 Poisson Distribution

In the case of a Poisson distribution, the persistor has a parameter λ, called intensity, which in our context is parametrized by ω:

$$\xi_\omega(k|s,a) = \frac{\lambda_{\boldsymbol{\omega}}^k e^{-\lambda_{\boldsymbol{\omega}}}}{k!}, \qquad k \in \mathbb{N}^+$$

This function is easily differentiable and plugged into Persistor Gradient Estimator:

$$\begin{aligned}
\log \xi_{\boldsymbol{\omega}}(k|s,a) &= k\log(\lambda_{\boldsymbol{\omega}}) - \lambda_{\boldsymbol{\omega}} - \log(k!) \\
\nabla_{\boldsymbol{\omega}} \log \xi_{\boldsymbol{\omega}}(k|s,a) &= (\frac{k}{\lambda_{\boldsymbol{\omega}}} - 1)\nabla_{\boldsymbol{\omega}}\lambda_{\boldsymbol{\omega}}
\end{aligned}$$

Proposition D.3. *The KL divergence between two Poisson distributions with parameters λ_1 and λ_2 is equivalent to $\lambda_1 \log\frac{\lambda_1}{\lambda_2} - (\lambda_1 - \lambda_2)$.*

Proof. Let's call $p_1 \sim Poisson(\lambda_1), p_2 \sim Poisson(\lambda_2)$. then:

$$\begin{aligned}
D_{KL}(p_1||p_2) &= \sum_{k=0}^{\infty} p_1(k) log \frac{p_1(k)}{p_2(k)} \\
&= \sum_{k=0}^{\infty} \frac{\lambda_1^k e^{-\lambda_1}}{k!} \log \frac{\lambda_1^k e^{-\lambda_1}}{\lambda_2^k e^{-\lambda_2}} \\
&= \sum_{k=0}^{\infty} \frac{k\lambda_1^k e^{-\lambda_1}}{k!} \log\frac{\lambda_1}{\lambda_2} - \sum_{k=0}^{\infty} p_1(k)(\lambda_1 - \lambda_2) \\
&= -(\lambda_1 - \lambda_2) + \sum_{k=1}^{\infty} \frac{k\lambda_1^k e^{-\lambda_1}}{k!} \log\frac{\lambda_1}{\lambda_2} \\
&= -(\lambda_1 - \lambda_2) + \log\frac{\lambda_1}{\lambda_2} \sum_{k'=0}^{\infty} \frac{\lambda_1^{k'} e^{-\lambda_1}}{k'!} \\
&= -(\lambda_1 - \lambda_2) + \lambda_1 \log\frac{\lambda_1}{\lambda_2}
\end{aligned}$$

□

D.4 Performance Difference

As a final preliminary result, we can extend the Performance Difference Lemma (Theorem 3.9) from (Kakade and Langford, 2002) to the persistor case:

Proposition D.4 (Performance difference lemma)**.** *Let $\mathcal{M}$ be an MDP. Given two policy-persistor pairs (π', ξ') and (π, ξ), the related performance difference can be expressed as an expected advantage:*

$$J(\pi', \xi') - J(\pi, \xi) = \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{s,a,k \sim \widetilde{\zeta}^{\mu}_{\pi',\xi'}} [A^{\pi,\xi}(s, a, k)]$$

Proof.

$$\begin{aligned}
J(\pi', \xi') &= \mathop{\mathbb{E}}_{\tau \sim \widetilde{\rho}_{\pi',\xi'}} \Big[\sum_{i=0}^{I(T)-1} \gamma^{t_i} r_{k_{t_i}}(s_{t_i}, a_{t_i}) \pm V^{\pi,\xi}(s_o) \Big] \\
&= \mathop{\mathbb{E}}_{\tau \sim \widetilde{\rho}_{\pi',\xi'}} \Big[\sum_{i=0}^{I(T)-1} \gamma^{t_i} \big(r_{k_{t_i}}(s_{t_i}, a_{t_i}) + \gamma^{k_{t_i}} V^{\pi,\xi}(s_{t_{i+1}}) - V^{\pi,\xi}(s_{t_i}) \big) \Big] + J(\pi, \xi) \\
&= \mathop{\mathbb{E}}_{\tau \sim \widetilde{\rho}_{\pi',\xi'}} \Big[\sum_{i=0}^{I(T)-1} \gamma^{t_i} A^{\pi,\xi}(s_{t_i}, a_{t_i}, k_{t_i}) \Big] + J(\pi, \xi) \\
&= \mathop{\mathbb{E}}_{\tau \sim \widetilde{\rho}_{\pi',\xi'}} \Big[\sum_{i=0}^{I(T)-1} \gamma^{t_i} A^{\pi,\xi}(s_{t_i}, a_{t_i}, k_{t_i}) \Big] + J(\pi, \xi) \\
&= \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{s,a,k \sim \widetilde{\zeta}^{\mu}_{\pi',\xi'}} \big[A^{\pi,\xi}(s, a, k) \big] + J(\pi, \xi),
\end{aligned} \tag{D.19}$$

where the last equality derives from Lemma D.9 and Corollary D.8. □

While this Lemma is the direct consequence of the classic performance difference lemma (Kakade and Langford, 2002), there is an important variation: the advantage of π, ξ is evaluated on the trajectories sampled by (π', ξ'), and the consequent invocation times can be drastically changed.

D.5 Useful Lemmas

Lemma D.5 (from Lemma B.2 in Papini (2021))**.** *$\forall s_0 \in \mathcal{S}$:*

$$\begin{aligned}
\widetilde{\delta}^{s_0}_{\pi,\xi}(\cdot) &= \frac{1-\gamma}{\gamma} \widetilde{p}_{\pi,\xi}(\cdot|s_0) + \int_{\mathcal{S}} \widetilde{\delta}^{s_0}_{\pi,\xi}(\mathrm{d}s) \widetilde{p}_{\pi,\xi}(\cdot|s) \\
\widetilde{\delta}^{\mu}_{\pi,\xi}(\cdot) &= (1-\gamma)\mu(\cdot) + \gamma \int_{\mathcal{S}} \widetilde{\delta}^{\mu}_{\pi,\xi}(\mathrm{d}s) \widetilde{p}_{\pi,\xi}(\cdot|s)
\end{aligned} \tag{D.20}$$

Proof.

$$\begin{aligned}
\gamma \int_{\mathcal{S}} \widetilde{\delta}^{s_0}_{\pi,\xi}(\,\mathrm{d}s) \widetilde{p}_{\pi,\xi}(\cdot|s) &= (1-\gamma) \int_{\mathcal{S}} \sum_{t=1}^{\infty} \widetilde{p}^{t}_{\pi,\xi}(\,\mathrm{d}s|s_0) \widetilde{p}_{\pi,\xi}(\cdot|s) \\
&= (1-\gamma) \sum_{t=1}^{\infty} \int_{\mathcal{S}} \widetilde{p}^{t}_{\pi,\xi}(\,\mathrm{d}s|s_0) \widetilde{p}_{\pi,\xi}(\cdot|s) \\
&= (1-\gamma) \sum_{t=1}^{\infty} \widetilde{p}^{t+1}_{\pi,\xi}(\cdot|s_0) \\
&= (1-\gamma) \sum_{t=1}^{\infty} \widetilde{p}^{t}_{\pi,\xi}(\cdot|s_0) - (1-\gamma)\widetilde{p}_{\pi,\xi}(\cdot|s_0) \\
&= \gamma \widetilde{\delta}^{s_0}_{\pi,\xi}(\cdot) - (1-\gamma)\widetilde{p}_{\pi,\xi}(\cdot|s_0)
\end{aligned}$$

□

Lemma D.6 (From Lemma 2.1 in Papini (2021)). *Let f be any integrable function on $\mathcal{S}$ satisfying the following recursive equation:*

$$f(s) = g(s) + \int_{\mathcal{S}} \widetilde{p}_{\pi,\xi}(\,\mathrm{d}s'|s) f(s')$$

Then,

$$f(s) = g(s) + \frac{\gamma}{1-\gamma} \int_{\mathcal{S}} \widetilde{\delta}^{s}_{\pi,\xi}(\,\mathrm{d}s') g(s') \tag{D.21}$$

Proof.

$$\begin{aligned}
\int_{\mathcal{S}} \widetilde{\delta}^{s}_{\pi,\xi}(\,\mathrm{d}s') g(s') &= \int_{\mathcal{S}} \widetilde{\delta}^{s}_{\pi,\xi}(\,\mathrm{d}s') f(s') - \int_{\mathcal{S}} \widetilde{\delta}^{s}_{\pi,\xi}(\,\mathrm{d}s') \int_{\mathcal{S}} \widetilde{p}_{\pi,\xi}(\,\mathrm{d}s''|s') f(s'') \\
&= \int_{\mathcal{S}} \widetilde{\delta}^{s}_{\pi,\xi}(\,\mathrm{d}s') f(s') - \int_{\mathcal{S}} \Big[\int_{\mathcal{S}} \widetilde{\delta}^{s}_{\pi,\xi}(\,\mathrm{d}s') \widetilde{p}_{\pi,\xi}(\,\mathrm{d}s''|s') \Big] f(s'') \\
&= \int_{\mathcal{S}} \widetilde{\delta}^{s}_{\pi,\xi}(\,\mathrm{d}s') f(s') - \int_{\mathcal{S}} \Big[\widetilde{\delta}^{s}_{\pi,\xi}(s') - \frac{1-\gamma}{\gamma} \widetilde{p}_{\pi,\xi}(\,\mathrm{d}s''|s') \Big] f(s'') \qquad \text{(D.22)} \\
&= \frac{1-\gamma}{\gamma} \int_{\mathcal{S}} \widetilde{p}_{\pi,\xi}(\,\mathrm{d}s''|s') f(s'') = \frac{1-\gamma}{\gamma} \big(f(s) - g(s) \big)
\end{aligned}$$

where D.22 comes from Lemma D.5 □

Lemma D.7. *Let f be any integrable function on $\mathcal{S}$*

$$\sum_{i=0}^{\infty} \mathop{\mathbb{E}}_{\tau_t \sim \widetilde{\rho}_{\pi,\xi}} [f(s_{t_i})] = \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{s \sim \widetilde{\delta}^{\mu}_{\pi,\xi}} \big[f(s) \big] \tag{D.23}$$

Proof.

$$\begin{aligned}
\sum_{i=0}^{\infty} \mathop{\mathbb{E}}_{\tau_t\sim\widetilde{\rho}_{\pi,\xi}}[f(s_{t_i})] &= \int_{\mathcal{S}} \left[\mu(s) + \int_{\mathcal{S}} \mu(s_0) \sum_{t=1}^{\infty} \widetilde{p}^{\,t}_{\pi,\xi}(s|s_0)\,\mathrm{d}s_0\right] f(s)\,\mathrm{d}s \\
&= \int_{\mathcal{S}} \left[\mu(s) + \int_{\mathcal{S}} \mu(s_0) \frac{\gamma}{1-\gamma} \widetilde{\delta}^{s_0}_{\pi,\xi}(s)\,\mathrm{d}s_0\right] f(s)\,\mathrm{d}s \\
&= \frac{1}{1-\gamma} \int_{\mathcal{S}} \left[(1-\gamma)\mu(s) + \gamma \int_{\mathcal{S}} \mu(s_0) \widetilde{\delta}^{s_0}_{\pi,\xi}(s)\,\mathrm{d}s_0\right] f(s)\,\mathrm{d}s \\
&= \frac{1}{1-\gamma} \int_{\mathcal{S}} \widetilde{\delta}^{\mu}_{\pi,\xi}(\,\mathrm{d}s) f(s); \qquad \text{(D.24)}
\end{aligned}$$

where D.24 comes from Equation D.7. □

Corollary D.8. *Let f be any integrable function on $\mathcal{S}\times\mathcal{A}$ and g any integrable function on $\mathcal{S}\times\mathcal{A}\times\mathcal{K}$.*

$$\sum_{i=0}^{\infty} \mathop{\mathbb{E}}_{\tau_{t_i}\sim\widetilde{\rho}_{\pi,\xi}}[f(s_{t_i}, a_{t_i})] = \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{s,a\sim\widetilde{\nu}^{\mu}_{\pi,\xi}}\big[f(s,a)\big]$$

$$\sum_{i=0}^{\infty} \mathop{\mathbb{E}}_{\tau_{t_i}\sim\widetilde{\rho}_{\pi,\xi}}[g(s_{t_i}, a_{t_i}, k_{t_i})] = \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{s,a,k\sim\widetilde{\zeta}^{\mu}_{\pi,\xi}}\big[g(s,a,k)\big]$$

Proof.

$$\begin{aligned}
\sum_{i=0}^{\infty} \mathop{\mathbb{E}}_{\tau_{t_i}\sim\widetilde{\rho}_{\pi,\xi}}[f(s_{t_i}, a_{t_i})] &= \sum_{i=0}^{\infty} \mathop{\mathbb{E}}_{\tau_{t_i}\sim\widetilde{\rho}_{\pi,\xi}}\big[\mathop{\mathbb{E}}_{a\sim\pi(\cdot|s_{t_i})}[f(s_{t_i}, a)]\big] \\
&= \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{s\sim\widetilde{\delta}^{\mu}_{\pi,\xi}}\big[\mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}[f(s,a)]\big] \\
&= \frac{1}{1-\gamma} \int_{\mathcal{S}} \widetilde{\delta}^{\mu}_{\pi,\xi}(\,\mathrm{d}s) \int_{\mathcal{A}} \pi(\,\mathrm{d}a|s) f(s,a) = \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{(s,a)\sim\widetilde{\nu}^{\mu}_{\pi,\xi}}[f(s,a)]
\end{aligned}$$

$$\begin{aligned}
\sum_{i=0}^{\infty} \mathop{\mathbb{E}}_{\tau_{t_i}\sim\widetilde{\rho}_{\pi,\xi}}[g(s_{t_i}, a_{t_i}, k_{t_i})] &= \sum_{i=0}^{\infty} \mathop{\mathbb{E}}_{\tau_{t_i}\sim\widetilde{\rho}_{\pi,\xi}}\left[\mathop{\mathbb{E}}_{a\sim\pi(\cdot|s_{t_i})}\big[\mathop{\mathbb{E}}_{k\sim\xi(\cdot|s_{t_i},a)}[g(s_{t_i}, a, k)]\big]\right] \\
&= \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{s\sim\widetilde{\delta}^{\mu}_{\pi,\xi}}\left[\mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}\big[\mathop{\mathbb{E}}_{k\sim\xi(\cdot|s,a)}[g(s,a,k)]\right] \\
&= \frac{1}{1-\gamma} \int_{\mathcal{S}} \widetilde{\delta}^{\mu}_{\pi,\xi}(\,\mathrm{d}s) \int_{\mathcal{A}} \pi(\,\mathrm{d}a|s) \int_{\mathcal{K}} \xi(\,\mathrm{d}k|s,a) g(s,a,k) \\
&= \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{(s,a,k)\sim\widetilde{\zeta}^{\mu}_{\pi,\xi}}[g(s,a,k)]
\end{aligned}$$

□

Lemma D.9. *Let f be any integrable function on $\mathcal{S}\times\mathcal{A}\times\mathcal{K}$:*

$$\mathop{\mathbb{E}}_{\tau\sim\widetilde{\rho}_{\pi,\xi}}\big[\sum_{i=0}^{I(T)-1} f(s_{t_i}, a_{t_i}, k_{t_i})\big] = \mathop{\mathbb{E}}_{\tau\sim\widetilde{\rho}_{\pi,\xi}}\big[\sum_{i=0}^{I(T)-1} \gamma^{t_i} f(s_{t_i}, a_{t_i}, k_{t_i})\big]$$

Proof.

$$
\begin{aligned}
\mathop{\mathbb{E}}_{\tau\sim\widetilde{\rho}_{\pi,\xi}}\Big[\sum_{i=0}^{I(T)-1} f(s_{t_i},a_{t_i},k_{t_i})\Big] &= \sum_{t=0}^{T-1}\mathop{\mathbb{E}}_{\tau_t\sim\widetilde{\rho}_{\pi,\xi}}\Big[\mathbb{1}_{t\in\mathbb{T}} f(s_t,a_t,k_t)\Big] \\
&= \sum_{t=0}^{T-1}\mathop{\mathbb{E}}_{\tau_t\sim\rho_{\pi,\xi}}\left[\frac{\widetilde{\rho}_{\pi,\xi}(\tau_t)}{\rho_{\pi,\xi}(\tau_t)}\mathbb{1}_{t\in\mathbb{T}} f(s_t,a_t,k_t)\right] \\
&= \sum_{t=0}^{T-1}\mathop{\mathbb{E}}_{\tau_t\sim\rho_{\pi,\xi}}\left[\gamma^t\mathbb{1}_{t\in\mathbb{T}} f(s_t,a_t,k_t)\right] \qquad \text{(D.25)} \\
&= \mathop{\mathbb{E}}_{\tau\sim\rho_{\pi,\xi}}\Big[\sum_{i=0}^{I(T)-1}\gamma^{t_i} f(s_{t_i},a_{t_i},k_{t_i})\Big],
\end{aligned}
$$

where D.25 derives from D.9. $\square$

Bibliography

D. Abel, W. Dabney, A. Harutyunyan, M. K. Ho, M. Littman, D. Precup, and S. Singh. On the expressivity of markov reward. *Advances in Neural Information Processing Systems*, 34:7799–7812, 2021.

R. R. Afshar, Y. Zhang, J. Vanschoren, and U. Kaymak. Automated reinforcement learning: An overview. *arXiv preprint arXiv:2201.05000*, 2022.

A. Agarwal, N. Jiang, S. M. Kakade, and W. Sun. Reinforcement learning: Theory and algorithms. *CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep*, 2019.

S.-I. Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2): 251–276, 1998.

S. Amin, M. Gomrokchi, H. Aboutalebi, H. Satija, and D. Precup. Locally persistent exploration in continuous control tasks with sparse rewards. In *International Conference on Machine Learning*, pages 275–285. PMLR, 2021.

M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

A. Antos, C. Szepesvári, and R. Munos. Fitted q-iteration in continuous action-space mdps. *Advances in neural information processing systems*, 20, 2007.

A. Antos, C. Szepesvári, and R. Munos. Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129, 2008.

L. Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3, 1966.

P. Artzner, F. Delbaen, J.-M. Eber, and D. Heath. Coherent measures of risk. *Mathematical finance*, 9(3):203–228, 1999. Publisher: Wiley Online Library.

K. Arulkumaran, N. Dilokthanakul, M. Shanahan, and A. A. Bharath. Classifying options for deep reinforcement learning. *arXiv preprint arXiv:1604.08153*, 2016.

K. J. Åström. Optimal control of markov processes with incomplete state information. *Journal of mathematical analysis and applications*, 10(1):174–205, 1965.

P. Bacon, J. Harb, and D. Precup. The option-critic architecture. In S. P. Singh and S. Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 1726–1734. AAAI Press, 2017.

V. Bacoyannis, V. Glukhov, T. Jin, J. Kochems, and D. R. Song. Idiosyncrasies and challenges of data driven learning in electronic trading. *arXiv preprint arXiv:1811.09549*, 2018.

L. C. Baird. Reinforcement learning in continuous time: Advantage updating. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 4, pages 2448–2453. IEEE, 1994.

S. Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fund. math*, 3(1):133–181, 1922.

A. G. Barto, R. S. Sutton, and C. W. Anderson. *Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems*, pages 81–93. IEEE Press, 1990. ISBN 0818620153.

J. Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15, 2001.

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

R. Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

R. Bellman, R. Bellman, and R. Corporation. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

D. P. Bertsekas. *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific, 2005. ISBN 1886529264.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming.* Athena Scientific, Belmont, MA, 1996.

D. P. Bertsekas and J. N. Tsitsiklis. *Introduction to probability*. Optimization and computation series. Athena scientific, Belmont, 2nd ed edition, 2008. ISBN 9781886529236.

A. Biedenkapp, H. F. Bozkurt, T. Eimer, F. Hutter, and M. Lindauer. Dynamic algorithm configuration: foundation of a new meta-algorithmic framework. In *ECAI 2020*, pages 427–434. IOS Press, 2020.

A. Biedenkapp, R. Rajan, F. Hutter, and M. T. Lindauer. Temporl: Learning when to act. In *ICML*, 2021.

C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

L. Bisi, P. Liotet, L. Sabbioni, G. Reho, N. Montali, M. Restelli, and C. Corno. Foreign exchange trading: A risk-averse batch reinforcement learning approach. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–8, 2020a.

L. Bisi, L. Sabbioni, E. Vittori, M. Papini, and M. Restelli. Risk-Averse Trust Region Optimization for Reward-Volatility Reduction. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4583–4589, 2020b.

L. Bottou et al. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142, 1998.

J.-P. Bouchaud and M. Potters. *Theory of financial risk and derivative pricing: from statistical physics to risk management*. Cambridge university press, 2003.

S. Bradtke and M. Duff. Reinforcement learning methods for continuous-time markov decision problems. *Advances in neural information processing systems*, 7, 1994.

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

L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

E. Brunskill and L. Li. Sample complexity of multi-task reinforcement learning. *arXiv preprint arXiv:1309.6821*, 2013.

Q. Cai, Z. Yang, C. Jin, and Z. Wang. Provably efficient exploration in policy optimization. In *International Conference on Machine Learning*, pages 1283–1294. PMLR, 2020.

D. Calandriello, A. Lazaric, and M. Restelli. Sparse multi-task reinforcement learning. *Advances in neural information processing systems*, 27, 2014.

A. Castelletti, S. Galelli, M. Restelli, and R. Soncini-Sessa. Tree-based reinforcement learning for optimal water reservoir operation. *Water Resources Research*, 46(9), 2010.

M. Catral, S. J. Kirkland, M. Neumann, and N.-S. Sze. The kemeny constant for finite homogeneous ergodic markov chains. *Journal of Scientific Computing*, 45(1):151–166, 2010.

V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

R. Chitnis and T. Lozano-Pérez. Learning compact models for planning with exogenous processes. In *Conference on Robot Learning*, pages 813–822. PMLR, 2020.

Y. Chow, A. Tamar, S. Mannor, and M. Pavone. Risk-Sensitive and Robust Decision-Making: a CVaR Optimization Approach. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *NeurIPS 28*, pages 1522–1530. Curran Associates, Inc., 2015.

K. Chowdhary. Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649, 2020.

K. Ciosek and S. Whiteson. Expected policy gradients for reinforcement learning. *Journal of Machine Learning Research*, 21(2020), 2020.

R. Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control. (Apprentissage par renforcement utilisant des réseaux de neurones, avec des applications au contrôle moteur)*. PhD thesis, Grenoble Institute of Technology, France, 2002.

W. Dabney, G. Ostrovski, and A. Barreto. Temporally-extended ϵ-greedy exploration. In *International Conference on Learning Representations (ICLR)*, 2020.

P. Dayan and S. P. Singh. Improving policies without measuring merits. In D. S. Touretzky, M. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8, NIPS, Denver, CO, USA, November 27-30, 1995*, pages 1059–1065. MIT Press, 1995.

M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

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

P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

T. G. Dietterich. Machine learning. *Annual review of computer science*, 4(1):255–306, 1990.

T. G. Dietterich. The MAXQ method for hierarchical reinforcement learning. In J. W. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998*, pages 118–126. Morgan Kaufmann, 1998.

K. Doya. Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245, 2000.

Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338. PMLR, 2016.

R. Durrett. *Probability: theory and examples*, volume 49. Cambridge university press, 2019.

A. Eiben, M. Horvath, W. Kowalczyk, and M. C. Schut. Reinforcement learning for online control of evolutionary algorithms. In *International Workshop on Engineering Self-Organising Applications*, pages 151–160. Springer, 2007.

Z. M. Erickson, V. Gangaram, A. Kapusta, C. K. Liu, and C. C. Kemp. Assistive gym: A physics simulation framework for assistive robotics. *CoRR*, abs/1910.04700, 2019.

D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(18):503–556, 2005.

A. M. Farahmand. Regularization in reinforcement learning, 2011.

A. M. Farahmand and C. Szepesvári. Model selection in reinforcement learning. *Machine Learning*, 85(3):299–332, 2011. doi: 10.1007/s10994-011-5254-7.

A. M. Farahmand, R. Munos, and C. Szepesvári. Error propagation for approximate policy and value iteration. In *Advances in Neural Information Processing Systems*, 2010.

M. Feurer, J. Springenberg, and F. Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

T. G. Fischer. Reinforcement learning in financial markets-a survey. Technical report, FAU Discussion Papers in Economics, 2018.

W. H. Fleming and H. M. Soner. *Controlled Markov processes and viscosity solutions*, volume 25. Springer Science & Business Media, 2006.

L. Franceschi, P. Frasconi, M. Donini, and M. Pontil. A bridge between hyperparameter optimization and larning-to-learn. *stat*, 1050:18, 2017.

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

S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

M. G. Bellemare, G. Ostrovski, A. Guez, P. Thomas, and R. Munos. Increasing the action gap: New operators for reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2016. doi: 10.1609/aaai.v30i1.10303.

F. M. Garcia and P. S. Thomas. A meta-mdp approach to exploration for lifelong reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1976–1978, 2019.

A. Geramifard, C. Dann, R. H. Klein, W. Dabney, and J. P. How. Rlpy: a value-function-based reinforcement learning framework for education and research. *J. Mach. Learn. Res.*, 16:1573–1578, 2015.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

G. J. Gordon. Stable function approximation in dynamic programming. In *Machine Learning Proceedings 1995*, pages 261–268. Elsevier, 1995.

J. Grigsby, J. Y. Yoo, and Y. Qi. Towards automatic actor-critic solutions to continuous control. In *Deep RL Workshop NeurIPS 2021*, 2021.

S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. In *International conference on machine learning*, pages 2829–2838. PMLR, 2016.

A. Gulli and S. Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine. Learning to walk via deep reinforcement learning. In A. Bicchi, H. Kress-Gazit, and S. Hutchinson, editors, *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*, 2019. doi: 10.15607/RSS.2019.XV.011.

E. Hadoux, A. Beynier, and P. Weng. Sequential decision-making under non-stationary environments via sequential change-point detection. In *Learning over multiple contexts (LMCE)*, 2014.

A. Hallak, D. Di Castro, and S. Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

H. Hasselt. Double q-learning. *Advances in neural information processing systems*, 23, 2010.

E. Hazan, S. Kakade, K. Singh, and A. Van Soest. Provably efficient maximum entropy exploration. In *International Conference on Machine Learning*, pages 2681–2691. PMLR, 2019.

P. Henderson, J. Romoff, and J. Pineau. Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods. *arXiv preprint arXiv:1810.02525*, 2018.

M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. doi: 10.1609/aaai.v32i1.11796.

K. Hinderer. Lipschitz continuity of value functions in markovian decision processes. *Mathematical Methods of Operations Research*, 62(1):3–22, 2005.

R. Howard. *Dynamic programming and Markov processes*. Technology Press of Massachusetts Institute of Technology, 1960.

R. A. Howard. Semi-markov decision-processes. *Bulletin of the International Statistical Institute*, 40(2):625–652, 1963.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

M. Hutter. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer Science & Business Media, 2004.

D. J. Im, C. Savin, and K. Cho. Online hyperparameter optimization by real-time recurrent learning. *arXiv preprint arXiv:2102.07813*, 2021.

F. O. Isinkaye, Y. O. Folajimi, and B. A. Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian informatics journal*, 16(3):261–273, 2015.

S. Jastrzebski, M. Szymczak, S. Fort, D. Arpit, J. Tabor, K. Cho, and K. Geras. The break-even point on optimization trajectories of deep neural networks. *arXiv preprint arXiv:2002.09572*, 2020.

N. Jiang, A. Kulesza, S. P. Singh, and R. L. Lewis. The dependence of effective planning horizon on model accuracy. In S. Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 4180–4189. IJCAI/AAAI Press, 2016.

C. Jin, Z. Yang, Z. Wang, and M. I. Jordan. Provably efficient reinforcement learning with linear function approximation. In *Conference on Learning Theory*, pages 2137–2143. PMLR, 2020.

H. S. Jomaa, J. Grabocka, and L. Schmidt-Thieme. Hyp-rl: Hyperparameter optimization by reinforcement learning. *arXiv preprint arXiv:1906.11527*, 2019.

S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274, 2002.

S. M. Kakade. A natural policy gradient. *Advances in neural information processing systems*, 14, 2001.

S. Kalyanakrishnan, S. Aravindan, V. Bagdawat, V. Bhatt, H. Goka, A. Gupta, K. Krishna, and V. Piratla. An analysis of frame-skipping in reinforcement learning. *arXiv preprint arXiv:2102.03718*, 2021.

M. Kearns and S. Singh. Finite-sample convergence rates for q-learning and indirect algorithms. *Advances in Neural Information Processing Systems (NIPS)*, pages 996–1002, 1999.

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

M. J. Kearns and S. Singh. Bias-variance error bounds for temporal difference updates. In *COLT*, pages 142–147, 2000.

A. Khan, J. Feng, S. Liu, and M. Z. Asghar. Optimal skipping rates: training agents with fine-grained control using deep reinforcement learning. *Journal of Robotics*, 2019, 2019.

O. Kilinc, Y. Hu, and G. Montana. Reinforcement learning for robotic manipulation using simulated locomotion demonstrations. *CoRR*, abs/1910.07294, 2019.

D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014.

B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

D. E. Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2004.

S. Kirkland. Fastest expected time to mixing for a markov chain on a directed graph. *Linear Algebra and its Applications*, 433(11-12):1988–1996, 2010.

J. Kober and J. Peters. *Learning Motor Skills - From Algorithms to Robot Experiments*, volume 97 of *Springer Tracts in Advanced Robotics*. Springer, 2014. ISBN 978-3-319-03193-4. doi: 10.1007/978-3-319-03194-1.

J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013. Publisher: SAGE Publications Sage UK: London, England.

V. Konda and J. Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

A. S. Lakshminarayanan, S. Sharma, and B. Ravindran. Dynamic action repetition for deep reinforcement learning. In S. P. Singh and S. Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 2133–2139. AAAI Press, 2017.

A. Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pages 143–173. Springer, 2012.

A. Lazaric, M. Restelli, and A. Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 544–551, 2008.

S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016. Publisher: JMLR. org.

K. Li and J. Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

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. *arXiv preprint arXiv:1509.02971*, 2015.

S. A. Lippman. Semi-markov decision processes with unbounded rewards. *Management Science*, 19(7):717–731, 1973.

M. L. Littman. *Algorithms for sequential decision-making*. Brown University, 1996.

J. Lorraine, P. Vicol, and D. Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pages 1540–1552. PMLR, 2020.

D. Lu and Q. Weng. A survey of image classification methods and techniques for improving classification performance. *International journal of Remote sensing*, 28 (5):823–870, 2007.

D. G. Luenberger. Introduction to dynamic systems; theory, models, and applications. Technical report, New York: John Wiley & Sons, 1979.

D. Maclaurin, D. Duvenaud, and R. Adams. Gradient-based hyperparameter optimization through reversible learning. In *International conference on machine learning*, pages 2113–2122. PMLR, 2015.

D. J. Mankowitz, T. A. Mann, and S. Mannor. Time regularized interrupting options. In *International Conference on Machine Learning (ICML)*, 2014.

T. A. Mann, S. Mannor, and D. Precup. Approximate value iteration with temporally extended actions. *J. Artif. Intell. Res.*, 53:375–438, 2015. doi: 10.1613/jair.4676.

Y. Mansour and S. Singh. On the complexity of policy iteration. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 401–408, 1999.

R. N. Mantegna and H. E. Stanley. *Introduction to econophysics: correlations and complexity in finance*. Cambridge university press, 1999.

F. Meier, D. Kappler, and S. Schaal. Online learning of a memory for learning rates. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2425–2432. IEEE, 2018.

T. L. Meng and M. Khushi. Reinforcement learning in financial markets. *Data*, 4(3): 110, 2019.

A. M. Metelli. *Exploiting environment configurability in reinforcement learning*. PhD thesis, Politecnico di Milano, 2021.

A. M. Metelli, M. Mutti, and M. Restelli. Configurable markov decision processes. In *International Conference on Machine Learning*, pages 3491–3500. PMLR, 2018.

A. M. Metelli, F. Mazzolini, L. Bisi, L. Sabbioni, and M. Restelli. Control frequency adaptation via action persistence in batch reinforcement learning. In *International Conference on Machine Learning*, pages 6862–6873. PMLR, 2020.

T. M. Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

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

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

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 *ICML*, pages 1928–1937, 2016.

G. E. Monahan. State of the art-a survey of partially observable markov decision processes: theory, models, and algorithms. *Management science*, 28(1):1–16, 1982.

J. Moody and M. Saffell. Learning to trade via direct reinforcement. *IEEE transactions on neural Networks*, 12(4):875–889, 2001.

A. W. Moore. Efficient memory based learning for robot control. *PhD Thesis, Computer Laboratory, University of Cambridge*, 1991.

A. Müller. Integral probability metrics and their generating classes of functions. *Advances in Applied Probability*, 29(2):429–443, 1997.

R. Munos. A convergent reinforcement learning algorithm in the continuous case based on a finite difference method. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, pages 826–831. Morgan Kaufmann, 1997.

R. Munos. Error bounds for approximate value iteration. In *Proceedings of the National Conference on Artificial Intelligence*, page 1006. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

R. Munos. Policy gradient in continuous time. *Journal of Machine Learning Research*, 7:771–791, 2006.

R. Munos. Performance bounds in l_p-norm for approximate value iteration. *SIAM journal on control and optimization*, 46(2):541–561, 2007.

R. Munos and P. Bourgine. Reinforcement learning for continuous stochastic control problems. *Advances in neural information processing systems*, 10, 1997.

R. Munos and C. Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(5), 2008.

A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

G. Neu, A. Jonsson, and V. Gómez. A unified view of entropy-regularized markov decision processes. *arXiv preprint arXiv:1705.07798*, 2017.

D. Nguyen-Tuong and J. Peters. Model learning for robot control: a survey. *Cognitive processing*, 12(4):319–340, 2011.

A. Nichol and J. Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2(2):1, 2018.

J. R. Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997. doi: 10.1017/CBO9780511810633.

M. Occorso, L. Sabbioni, A. M. Metelli, and M. Restelli. Trust region meta learning for policy optimization. In P. Brazdil, J. N. van Rijn, H. Gouk, and F. Mohr, editors, *ECMLPKDD Workshop on Meta-Knowledge Transfer*, volume 191 of *Proceedings of Machine Learning Research*, pages 62–74. PMLR, 23 Sep 2022. URL `https://proceedings.mlr.press/v191/occorso22a.html`.

T. L. Paine, C. Paduraru, A. Michi, C. Gulcehre, K. Zolna, A. Novikov, Z. Wang, and N. de Freitas. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*, 2020.

M. Papini. *Safe policy optimization*. PhD thesis, Politecnico di Milano, 2021.

M. Papini, M. Pirotta, and M. Restelli. Adaptive batch size for safe policy gradients. In *NeurIPS*, pages 3591–3600, 2017.

M. Papini, M. Pirotta, and M. Restelli. Smoothing Policies and Safe Policy Gradients, 2019. _eprint: 1905.03231.

E. Park and J. B. Oliva. Meta-curvature. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

S. Park, J. Kim, and G. Kim. Time discretization-invariant safe action repetition for policy gradient methods. *Advances in Neural Information Processing Systems*, 34: 267–279, 2021.

J. Parker-Holder, R. Rajan, X. Song, A. Biedenkapp, Y. Miao, T. Eimer, B. Zhang, V. Nguyen, R. Calandra, A. Faust, et al. Automated reinforcement learning (autorl): A survey and open problems. *Journal of Artificial Intelligence Research*, 74:517–568, 2022.

R. Patel, P. Agharkar, and F. Bullo. Robotic surveillance and markov chains with minimal weighted kemeny constant. *IEEE Transactions on Automatic Control*, 60(12): 3156–3167, 2015.

S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

S. Paul, V. Kurin, and S. Whiteson. Fast efficient hyperparameter tuning for policy gradient methods. *Advances in Neural Information Processing Systems*, 32, 2019.

A. R. Penner. The physics of golf. *Reports on Progress in Physics*, 66(2):131, 2002.

J. Peters and S. Schaal. Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2219–2225. IEEE, 2006.

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

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

J. K. Peterson. On-line estimation of the optimal value function: Hjb-estimators. In *Advances in neural information processing systems*, pages 319–326, 1993.

M. Petrik and B. Scherrer. Biasing approximate dynamic programming with a lower discount factor. *Advances in neural information processing systems*, 21, 2008.

M. Pirotta, M. Restelli, and L. Bascetta. Adaptive Step-Size for Policy Gradient Methods. In *NeurIPS 26*, pages 1394–1402. Curran Associates, Inc., 2013a.

M. Pirotta, M. Restelli, A. Pecorino, and D. Calandriello. Safe policy iteration. In *ICML*, pages 307–315, 2013b.

M. Pirotta, M. Restelli, and L. Bascetta. Policy gradient in lipschitz markov decision processes. *Machine Learning*, 100(2-3):255–283, 2015. Publisher: Springer.

A. S. Polydoros and L. Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

D. Precup. *Temporal abstraction in reinforcement learning.* PhD thesis, University of Massachusetts Amherst, 2001.

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

M. L. Puterman and M. C. Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11):1127–1137, 1978.

E. Rachelson and M. G. Lagoudakis. On the locality of action domination in sequential decision making. In *International Symposium on Artificial Intelligence and Mathematics, ISAIM 2010, Fort Lauderdale, Florida, USA, January 6-8, 2010*, 2010.

K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR, 2019.

G. Ramponi, A. M. Metelli, A. Concetti, and M. Restelli. Learning in non-cooperative configurable markov decision processes. *Advances in Neural Information Processing Systems*, 34:22808–22821, 2021.

S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

M. Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European conference on machine learning*, pages 317–328. Springer, 2005.

A. Riva, L. Bisi, P. Liotet, L. Sabbioni, E. Vittori, M. Pinciroli, M. Trapletti, and M. Restelli. Learning fx trading strategies with fqi and persistent actions. In *Proceedings of the Second ACM International Conference on AI in Finance*, pages 1–9, 2021.

A. Riva, L. Bisi, P. Liotet, L. Sabbioni, E. Vittori, M. Pinciroli, M. Trapletti, and M. Restelli. Addressing non-stationarity in fx trading with online model selection of offline rl experts. In *Proceedings of the Third ACM International Conference on AI in Finance*, pages 394–402, 2022.

H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

C. P. Robert, G. Casella, and G. Casella. *Monte Carlo statistical methods*, volume 2. Springer, 1999.

G. A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*, volume 37. Citeseer, 1994.

L. Sabbioni, F. Corda, and M. Restelli. Meta learning the step size in policy gradient methods. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021.

L. Sabbioni, L. A. Daire, L. Bisi, A. M. Metelli, and M. Restelli. Simultaneously updating all persistence values in reinforcement learning. *arXiv preprint arXiv:2211.11620*, 2022.

L. Sabbioni, F. Corda, and M. Restelli. Stepsize learning for policy gradient methods in contextual markov decision processes. *arXiv preprint arXiv:2306.07741*, 2023.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

B. Scherrer. Improved and generalized upper bounds on the complexity of policy iteration. *Advances in Neural Information Processing Systems*, 26, 2013.

B. Scherrer. Approximate policy iteration schemes: a comparison. In *International Conference on Machine Learning*, pages 1314–1322. PMLR, 2014.

J. Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

R. Schoknecht and M. Riedmiller. Reinforcement learning on explicitly specified time scales. *Neural Computing & Applications*, 12(2):61–80, 2003.

J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. Trust Region Policy Optimization. In *ICML*, pages 1889–1897, 2015.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347, 2017.

A. Sehgal, H. La, S. Louis, and H. Nguyen. Deep reinforcement learning using genetic algorithm for parameter optimization. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 596–601. IEEE, 2019.

A. Seierstad. *Stochastic control in discrete and continuous time*, volume 1. Springer, 2009.

A. A. Shahid, D. Piga, F. Braghin, and L. Roveda. Continuous control actions learning and adaptation for robotic manipulation through reinforcement learning. *Autonomous Robots*, 46(3):483–498, 2022.

S. Shalev-Shwartz, S. Shammah, and A. Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

L. Shani, Y. Efroni, and S. Mannor. Adaptive trust region policy optimization: Global convergence and faster rates for regularized mdps. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):5668–5675, Apr. 2020. doi: 10.1609/aaai.v34i04.6021.

C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

S. Sharma, A. Srinivas, and B. Ravindran. Learning to repeat: Fine grained action repetition for deep reinforcement learning. *arXiv preprint arXiv:1702.06054*, 2017.

A. Sidford, M. Wang, X. Wu, L. F. Yang, and Y. Ye. Near-optimal time and sample complexities for solving markov decision processes with a generative model. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 5192–5202, 2018.

R. Silva, G. Farina, F. S. Melo, and M. Veloso. A theoretical and algorithmic analysis of configurable mdps. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 455–463, 2019.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller. Deterministic Policy Gradient Algorithms. In *ICML*, volume 32 of *JMLR and Conference Proceedings*, pages 387–395. JMLR.org, 2014.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, and others. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587):484, 2016. Publisher: Nature Publishing Group.

D. Silver, S. Singh, D. Precup, and R. S. Sutton. Reward is enough. *Artificial Intelligence*, page 103535, 2021.

H. Singh, N. Misra, V. Hnizdo, A. Fedorowicz, and E. Demchuk. Nearest neighbor estimates of entropy. *American journal of mathematical and management sciences*, 23(3-4):301–321, 2003.

S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning*, 38(3):287–308, 2000.

S. P. Singh. Reinforcement learning with a hierarchy of abstract models. In *Proceedings of the tenth national conference on Artificial intelligence*, pages 202–207, 1992a.

S. P. Singh. Scaling reinforcement learning algorithms by learning variable temporal resolution models. In *Machine Learning Proceedings 1992*, pages 406–415. Elsevier, 1992b.

S. P. Singh and R. C. Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233, 1994.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

S. Sodhani, A. Zhang, and J. Pineau. Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning*, pages 9767–9779. PMLR, 2021.

A. L. Strehl and M. L. Littman. A theoretical analysis of model-based interval estimation. In *Proceedings of the 22nd international conference on Machine learning*, pages 856–863, 2005.

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

R. S. Sutton. The quest for a common model of the intelligent decision maker. *arXiv preprint arXiv:2202.13252*, 2022.

R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0-262-19398-1.

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

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

R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999b.

C. Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

C. Tallec, L. Blier, and Y. Ollivier. Making deep q-learning methods robust to time discretization. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6096–6104. PMLR, 2019.

D. Thapa, I.-S. Jung, and G.-N. Wang. Agent based decision support system using reinforcement learning under emergency circumstances. In *International Conference on Natural Computation*, pages 888–892. Springer, 2005.

P. Thomas, G. Theocharous, and M. Ghavamzadeh. High-confidence off-policy evaluation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

T. Tieleman and G. Hinton. Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. *Technical Report.*, 2017.

A. Tirinzoni, R. Rodriguez Sanchez, and M. Restelli. Transfer of value functions via variational methods. *Advances in Neural Information Processing Systems*, 31, 2018.

A. Tirinzoni, M. Salvini, and M. Restelli. Transfer of samples in policy search via multiple importance sampling. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6264–6274. PMLR, 09–15 Jun 2019.

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

M. Tornio and T. Raiko. Variational bayesian approach for nonlinear identification and control. In *Proc. of the IFAC Workshop on Nonlinear Model Predictive Control for Fast Systems, NMPC FS06*, pages 41–46. Citeseer, 2006.

J. Tsitsiklis and B. Van Roy. Analysis of temporal-diffference learning with function approximation. *Advances in neural information processing systems*, 9, 1996.

P. Vamplew, B. J. Smith, J. Källström, G. Ramos, R. Rădulescu, D. M. Roijers, C. F. Hayes, F. Heintz, P. Mannion, P. J. Libin, et al. Scalar reward is not enough: A response to silver, singh, precup and sutton (2021). *Autonomous Agents and Multi-Agent Systems*, 36(2):1–19, 2022.

H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2016. doi: 10.1609/aaai.v30i1.10295.

H. Van Seijen, A. R. Mahmood, P. M. Pilarski, M. C. Machado, and R. S. Sutton. True online temporal-difference learning. *The Journal of Machine Learning Research*, 17 (1):5057–5096, 2016.

J. Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

T. Wang, M. Bowling, and D. Schuurmans. Dual representations for dynamic programming and reinforcement learning. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 44–51. IEEE, 2007.

Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1995–2003, New York, New York, USA, 20–22 Jun 2016. PMLR.

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

C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, University of Cambridge, 1989.

H. J. Weerts, A. C. Mueller, and J. Vanschoren. Importance of tuning hyperparameters of machine learning algorithms. *arXiv preprint arXiv:2007.07588*, 2020.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. Publisher: Springer.

C. Xu, T. Qin, G. Wang, and T.-Y. Liu. Reinforcement learning for learning rate control. *arXiv preprint arXiv:1705.11159*, 2017.

R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

Z. Xu, H. P. van Hasselt, and D. Silver. Meta-gradient reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

Y. Ye. The simplex and policy-iteration methods are strongly polynomial for the markov decision problem with a fixed discount rate. *Mathematics of Operations Research*, 36(4):593–603, 2011.

C. Yildiz, M. Heinonen, and H. Lähdesmäki. Continuous-time model-based reinforcement learning. In *International Conference on Machine Learning*, pages 12009–12018. PMLR, 2021.

H. Yu, W. Xu, and H. Zhang. Taac: Temporally abstract actor-critic for continuous control. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021.

J. Yu, D. Aberdeen, and N. N. Schraudolph. Fast online policy gradient learning with smd gain vector adaptation. In *Advances in neural information processing systems*, pages 1185–1192, 2006.

Y. Zhu, T. Hayashi, and Y. Ohsawa. Gradient descent optimization by reinforcement learning. In *The 33rd Annual Conference of the Japanese Society for Artificial Intelligence*, 2019.

B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.