Reinforcement Learning in Persistent Environments: Representation Learning and Transfer

a dissertation presented
by
Diana Borsa
to
The Department of Computer Science

in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
Machine Learning

University College London
London, UK
January 2020

In the memory of my dad, Prof. Vasile Borsa (1963-2014)

Declaration

I, Diana Borsa, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

– Diana Borsa

Impact Statement

In a nutshell, the current body of work has focused on the problem of a learning agent situated in a environment, trying to learn behaviour policies that maximise different reward signals, corresponding to different tasks in this environment. We would argue that this is a common scenario for many decision making systems in the real world. Reinforcement learning, as a modelling paradigm, has already been shown to successfully tackle complex decision making problems. Our contributions here stem from two key observations: a) in general, we would like our agents to achieve more than one goal in a given environment; b) a lot of the methods underlying even the single task setting, involve building, incrementally, multiple prediction problems that enable improvements in the agent's behaviour. Thus, an agent's journey to an optimal value function, can naturally be cast as a multitask prediction problem. The only difference between a) and b) is whether or not, one varies both the policy and the reward structure. In this work, we have focused on the more general scenario where both of these dimensions vary. In this setting we have shown the benefits of treating the above as a multitask problem and learning common representations to enable transfer between the many prediction problems encountered. Given the generality of the setup and the organic nature of the assumptions imposed, we believe the scope and applicability of this work to be quite board.

Moreover, the first part of this work has focused on an offline batch scenario, similar to ones encountered in a real application domains, where the data was generated a priori by a policy we might not have access to. Setups here might include recommender systems under multiple prediction metrics (engagement, retention, expenditure); medical data collected under different protocols and different policies; energy management systems under different manual policies. Treating these problems and data sources independently can be very expensive. In such restrictive settings, the encouraging results obtained by algorithms in Chapters 3 - 4 suggest that modelling these problems jointly can significantly improve the quality of the resulting policies and thus reduce the sample complexity needed to achieve competitive performance.

The second part of this thesis investigates a different kind of representation, particularly suitable for transferring knowledge within a more restricted set of tasks. This leads to a very effective type of generalisation in the span of tasks considered. Outside its potential applications to a multitask RL agent, this line of research has already inspired studies in other scientific fields. In particular in cognitive science and neuroscience, our colleagues have argued similar representations are formed in the brain and they have found evidence for this type of generalisation, via policy re-evaluation, being at the core of transfer behaviours in several species (including humans).

Nevertheless, the above are really just scratching the surface of the potential benefits of such representation learning in RL. Much more research is needed and we do hope that some of the work presented here would inspired other researchers, across communities, to investigate these paradigms further and improve on current solutions.

Acknowledgments

This whole process has been a long, formative journey. It is almost intimating looking back, all the way back into this chapter of my life; acknowledging all the turns, the bumps, the occasional wins, the ups and downs, the deep valleys, the scarce summits. That is truly an onerous credit-assignment problem and this is merely an attempt to do it justice.

First and foremost, I am deeply grateful to have had two wonderful supervisors, and most of all mentors, Thore Graepel and John Shawe-Taylor. Both of them have poured countless hours into my continuous growth as a researcher and beyond. I will forever be grateful for everything I have learnt from you, all the trust and encouragement I was given, the freedom to pursue my ideas – even the bad ones, the guidance to shape my understanding, the many opportunities you put in front of me. You have been absolutely instrumental throughout this journey and the reason I can call myself a researcher today. Thore, your curiosity and fascination with RL, led me to re-consider this line of the research in a very different lighting. Thank you for bring me to this paradigm, this way to see and model the world, to a complex problem that I am still engagingly pursuing. John, you are the reason I want to stay in academia. The impact I have witnessed over multiple generations cannot overstated. Your constant positivity and open-mindedness when approaching a problem, paired with critical thinking and mathematical rigour, are something I aspire to achieve.

Secondly, I was privileged enough to split my time, in the first few years, between UCL and Microsoft Research Cambridge. In many ways, MSRC was where I 'grew up' as a researcher and there are many people who contributed to this endeavour. I would like to extend a special thank you to Andy Gordon who, together with Thore, hosted me for my first project there and stayed up, late into the night, when I was submitting my first paper. Many thanks to Katja and Yoram for many fruitful discussions, for their guidance, their friendship and for sharing their PhD. survival wisdom. To the many fellow interns and friends I made during those times: Yoad, Yair, Rati, Irina, Yani, Mat and Roberto. To the many engineers that supported my work, Lucas, Tim, Dave and Matthew Johnson, who all took the time to educate a mathematician in software development. Lastly, but certainly not least, I would like to thank Chris Bishop for taking the time and interest to discuss my journey and professional development, even when it diverged away from the lab.

I would like to thank my colleagues and officemates in the CS department at UCL, Andrew, Dimitrios, Verena, Alex, Tom and Ronnie. Especially in those last years at UCL, your company, fellowship and humour made those long days at the office particularly enjoyable. Thank you also to the faculty in CSML for their teaching and occasional guidance: Iasonas Kokkinos, Mark Herbster, David Barber, Arthur Gretton and Sebastian Riedel.

The last stop in this journey was DeepMind and the i-Team. I am extremely grateful for the warm welcome the i-Team has given me from day one. Thank you to Rémi Munos and Olivier Pietquin for bringing me onto this team, for all your guidance, your trust and for always making me feel like a valuable member of this family. Rémi, I especially enjoyed our long sessions on the whiteboard exploring new operators and trying to formalise new continual learning paradigms. Outside my team, a very special thank you goes to one of the my closest collaborators and kindred minds, André. It has always been a pleasure brainstorming, collaborating and discussing new ideas with you. To all my other collaborators, Hado, Tom, Anna, Doina, Dave, Will, Georg, Bilal, Nicolas and Julien, thank you for always keeping me grounded and true to the science. You all make me a better researcher!

Thank you to Philip Treleaven, Yonita Carter and the UK Centre for Doctoral Training in Financial Computing and Analytics for their funding. Thank you also to JJ and Sarah for their commitment to making our experience at UCL more enjoyable and helping us out with the omnipresent, copious amounts of paperwork required. Thank you also to all other organisations that supported my travels to conferences, academic visits and summer schools. All of these have been highly enriching experiences.

Finally, I would like to thank my parents for all the sacrifices they made to support my academic journey, for all the training and knowledge I received from them, for always instilling a sense of wonder and curiosity about all aspects for the world. Also a special thank you to Iunia for always pushing me to do better, to achieve more, to be a better example.

Thesis advisor: Thore Graepel Diana Borsa

Reinforcement Learning in Persistent Environments: Representation Learning and Transfer

Abstract

Reinforcement learning (RL) provides a general framework for modelling and reasoning about agents capable of sequential decision making, with the goal of maximising a reward signal. In this work, we focus on the study of situated agents designed to learn autonomously through direct interaction with their environment, under limited or sparse feedback. We consider an agent in a persistent environment. The dynamics of this ’world’ do not change over time, much like the laws of physics, and the agent would need *to learn* to master a potentially vast set of tasks in this environment. To efficiently tackle learning in multiple tasks, with the ultimate goal of scaling to a life-long learning agent, we turn our attention to transfer learning. The main insight behind this paradigm is that generalisation may occur not only within tasks, but also across them. The objective of transfer in RL is to accelerate learning by building and reusing knowledge obtained in previously encountered tasks. This knowledge can be in the form of samples, value functions, policies, shared features or other abstractions of the environment or behaviour.

In this thesis, we examine different ways of learning transferable representations for value functions. We start by considering jointly learning value functions across multiple reward signals. We explore doing this by leveraging known multitask techniques to learn a shared set of features that cater to the intermediate solutions of popular iterative dynamic learning processes – like value and policy iteration. This learnt representation evolves as the individual value functions improve. At the end of this process, we obtain a shared basis for (near) optimal value functions. We show that this process benefits the learning of good policies for the tasks considered in this joint learning. This class of algorithms is potentially very general, but somewhat agnostic to the persistent environment assumption. Thus we turn to ways of building this shared basis by leveraging more explicitly the rich structure induced by this assumption. This leads to various extensions of least-squares Policy Iteration methods to the multitask scenario, under shared dynamics. Here we leverage transfer of samples and multitask regression to further improve sample efficiency in building these shared representations, capturing commonalities across optimal value functions.

The second part of the thesis introduces a different way of representing knowledge via *successor features*. In contrast to the representations learnt in the first part, these are policy dependent and serve as a basis for policy evaluations, rather than directly building optimal value functions. As such, the way to transfer knowledge to a new task changes as well. We do this by first relating the new task to previous learnt ones. In particular, we try to approximate the new reward signal as a linear combination of previous ones. Under this approximation, we can obtain approximate evaluations of the quality of previously learnt policies on the new task. This enables us to carry over knowledge about good or bad behaviour across tasks and strictly improve on previous behaviours. Here the transfer leverages the structure in policy space, with the potential of re-using partial solutions learnt in previous tasks. We show empirically that this leads to a scalable, online algorithm that can successfully re-use the common structure, if present, between a set of training tasks and a new one. Finally, we show that if one has further knowledge about the reward structure an agent would encounter, one can leverage this to learn very effectively, in an off-policy and off-task manner, a parameterised collection of successor features. These correspond to multiple (near) optimal policies for tasks hypothesized by the agent. This not only makes very efficient use of the data but proposes a parametric solution to the behaviour basis problem; namely which policies should one learn to enable transfer.

Contents

Listing of figures

The world we live in requires us to learn many things. These things obey the same physical laws, derive from the same human culture, are preprocessed by the same sensory hardware. . . . Perhaps it is the ***similarity*** *of the many tasks we learn that enables us to learn so much with so little experience.*

Rich Caruana. *Multitask Learning*, 1997.

1 Introduction

Throughout their lifetime, human beings and other intelligent agents, are generally presented with a variety of tasks and challenges they ought to grasp. Due to the diversity of scenarios and the sheer complexity of the world around us, the only scalable solution to this daunting task rests on our ability to continuously learn, adapt and transfer knowledge from one situation to another, incrementally building up expertise in both understanding and behaving in the world. This learning can be passive by simply observing the environment and trying to make predictions regarding its current and/or future states. This type of learning is mostly treated by the supervised (Vapnik, 2013) and unsupervised learning paradigms (Hinton et al., 1999; Bishop, 2006). But arguably the more interesting setting is when the agent can also meaningfully interact with the environment and take an active role in shaping and modifying the state of its world. In this case, the agent cares not only about learning about the environment, but it also needs to learn how to behave, *how to act* to drive itself or the system towards a desired outcome or avoid a negative one – sometimes being faced with negotiating multiple of these objectives at the same time.

Learning through these experiences, humans gradually develop an understanding of the world surrounding them: they are able to identify salient events, recognise which states are good or desirable, which of them might be rewarding and which should be avoided. At the same time, they can assemble a rich repertoire of behaviours and skills, grounded in this understanding of the world. Maybe most importantly, this set of behaviours can potentially be reused in different but related, future contexts and tasks. Our capacity to reason about similar tasks, reuse and adapt partial solutions and employ previously learned behaviours is an important part of our ability to generalise to new situations, re-assess and re-plan efficiently. We would want our agents to possess similar aptitudes, as we conjecture these would improve their ability to scale to more complex, longer-term learning problems.

Overarching Long-term Goal

To develop reinforcement learning agents that can *learn to achieve multiple objectives*, including answering predictive and control questions, and *meaningfully interact* with their environment, generalising and reusing knowledge when appropriate.

In this work, we will focus on the second part of the learning problem outlined above: learning how to behave in an environment. And in particular, in this thesis, we investigate the problem of *how to represent knowledge about an agent's behaviours and their consequences in the environment in such a way as to facilitate transfer between different tasks and enable reusability of partial policies*. To model this problem, we turn our attention to reinforcement learning (RL). RL provides a general framework to model this kind of sequential decision making that involves reasoning and planning over extended time horizons. The usual RL setup involves an agent placed in an unknown environment; the agent receives observations from the environment and needs to act to maximise a given reward signal. This reward signal defines the task of interest and induces a particular behaviour in the agent trying to optimise it. This reward can be part of the environment (if the agent bumps into a wall, it incurs some damage, modelled by a negative reward), can be specified by a task designer (picking up the trash, driving to a particular location), or can come from the agent itself – curiosity (Pathak et al., 2017; Burda et al., 2018), intrinsic motivation (Barto and Simsek, 2005; Singh et al., 2010; Achiam and Sastry, 2017). Given the generality of this formulation, it comes as no surprise that the RL framework has been used to model a large variety of tasks, including robotics (Kober et al., 2013), radio-controlled helicopters navigation (Kim et al., 2004), autonomous driving (Wang et al., 2018), animals behaviour (Schultz et al., 1997) and playing board (Tesauro, 1995; Silver et al., 2017), video (Mnih et al., 2015) or card (Bowling et al., 2015) games to cite just a few.

Most of the above applications and indeed most of the recent successes in RL have dealt primarily with the single task scenario. In the following, we would argue that there are many situations in which we want our agents to be able to learn to perform a variety of tasks in their given environment, not just one. For example if the task designer would be interested in multiple outcomes at different points in time: for instance, one would want a house-robot to be able to clean a surface, locate or fetch an object, water the plants, take out the rubbish ...etc. Moreover, most complex tasks, we would hope to tackle in RL, tend to naturally decompose into smaller subparts that can be learnt more easily in isolation and reused across contexts. In the above example, one could specify a more complex or abstract task such as: 'take care of the apartment', which will naturally require a collection of specific behaviours: collecting the children's toys, hoovering, taking out the trash, watering the plants, arming the alarm, securing the windows. This kind of decomposition may enable our agents to tackle more complex tasks, as long as they are able to learn and utilise this collection of behaviours appropriately. These subtasks can be modelled by different reward specifications in the same environment. Thus, as almost a prerequisite for tackling the original problem, our agents should be able to learn how to optimise for a family of reward signals. This scenario is a well-established paradigm in the literature known as multitask reinforcement learning (Taylor and Stone, 2009; Teh et al., 2017).

The above is not only a requirement for our agents but often a desirable paradigm as tasks can help each others' learning. And this is precisely the thesis of our investigations: that the learning of an RL task can be improved if we could somehow leverage the information captured by related tasks. A particular dimension of improvement we will be concerned with throughout this work is *sample complexity*. In the author's opinion, this is one of the main remaining challenges and bottlenecks facing our state-of-the-art RL agents currently (Silver et al., 2016; Hessel et al., 2018; Kapturowski et al., 2019), rendering their applicability outside simulated environments restricted at best. To give an idea, in order to learn how to play one game of Atari (Bellemare et al., 2013), an agent typically consumes an amount of data corresponding to several weeks of uninterrupted playing. In comparison, it has been shown that humans are able to reach the same performance level within 15 minutes of play (Tsividis et al., 2017).

We hypothesise that one of the main reason for this discrepancy is that, unlike humans, RL agents usually learn to perform a task essentially from scratch. This suggests that the range of problems our agents can tackle could be significantly extended if they were endowed with the appropriate mechanisms to leverage prior knowledge. In this work, we are going to explore how to develop such mechanisms and analyse their properties. More concretely, in the first part of this work, we are going to investigate the paradigm of transferring knowledge between tasks by jointly training a common representation of the world, shared by all tasks. This was mainly motivated by the rich literature and encouraging results yielded by this approach in supervised learning (Caruana, 1997; Maurer et al., 2016; Collobert and Weston, 2008). Here we consider an offline, batch learning scenario under a restricted number of samples. This is akin to what one might encounter in a real-world application, where we have access only to a relatively small set of experiences under a prescribed behaviour policy that might not be optimal for any of the tasks we are interested in pursuing (Panuccio et al., 2013; Koedinger et al., 2013; Thomas and Brunskill, 2016).

The second part of this work investigates a different paradigm of transfer. One in which the agent is first exposed to a number of tasks and then asked to learn a new one. This scenario simulates a transfer step in a more continual learning paradigm where an agent is continually confronted with new tasks and needs to be able to learn and quickly comply. In general, learning to solve a large collection of tasks, corresponding to a large collection of policies and quickly adapt or generalise to unseen combinations of reward signals is an extremely hard problem. If the tasks are unrelated and/or their number is very high, the problem becomes intractable. Thus for this problem to become at all tractable, we require some common structure amongst the tasks. In the next section, we will detail the type of shared structure we assume in this study.

1.1 Problem Setting

In particular, we consider the case in which we have a single agent in a persistent environment (in the above case, the apartment), that will need, at different times, to perform a series of tasks induced by a set of reward signals. As seen from the examples above, under these assumptions, an agent can still exhibit a large collection of rich, modular behaviours, which nevertheless share a common structure. An informal definition of what we will call a persistent RL environment is outlined in Definition 1.1.

Definition 1.1: Persistent RL environment

A persistent RL environment is characterised by the family of Markov Decision Processes that share the same state and action spaces, as well as the transition dynamics.

Now consider an RL agent in a persistent environment trying to master a number of tasks. In order for this agent to benefit from its exposure to these many tasks, it needs to be able to identify and exploit some common structure underlying these tasks. Two possible sources of structure in this scenario are: i) some similarity between the solutions of the tasks, either in the policy or in the associated value-function space, and ii) the shared dynamics of the environment. In this work, we will attempt to build an agent that can make use of both types of structure. And in particular, we would like to investigate and answer the following research questions:

- Can joint learning across multiple tasks benefit the learning of an individual task and improve its sample complexity? ([c1,2], Chapters 3-6)
- What structure is present in the value function space, under a persistent RL environment? Can learning a shared representation via conventional supervised multitask learning methods be enough to induce positive transfer between tasks in this space? (c[1,2], Chapters 3)
- What are ways of explicitly taking advantage of this structure in a persistent environment? ([c3,4], Chapters 4-6)
- Can we enable fast adaptation or even zero-shot generalisation to a different reward signal within the same environment? What would be a good representation that would enable this kind of transfer? ([c3,4], Chapters 5-6)
- Assuming additional structure across our reward family (linearity, compositionality), can we improve our generalisation properties? Can we characterise the quality of a zero-shot policy for a new task in this family? ([c3,4], Chapters 5-6)
- Can learning about multiple things (many predictions) improve the learning and generalisation abilities of an agent, even if the tasks share the same data? This would open the door for training under multiple auxiliary or fictitious tasks to improve the generalisation properties of our systems. ([c4], Chapter 6)

List of Contributions

c1 **Diana Borsa**, Thore Graepel, John Shawe-Taylor. *Learning shared structures in MDPs from multiple tasks,* (NIPS 2014, Workshop – Best Paper Award)

c2 **Diana Borsa**, Thore Graepel, John Shawe-Taylor. *Learning shared representations in multi-task reinforcement learning,* (WIML NIPS 2015).

c3 André Barreto, **Diana Borsa**, John Quan, Tom Schaul, David Silver, Matteo Hessel, Augustin Žídek, Rémi Munos. *Transfer in deep reinforcement learning using successor features and generalised policy improvement,* (ICML 2018).

c4 **Diana Borsa**, André Barreto, John Quan, Daniel Mankowitz, Rémi Munos, Hado van Hasselt, David Silver, Tom Schaul. *Universal successor features approximators,* (ICLR 2019)

List of other work and Publications conducted during this time

o5 **Diana Borsa**, Thore Graepel, Andrew Gordon. *The Wreath Process: A totally generative model of geometric shape based on nested symmetries.* (preprint arXiv:1506.03041)

o6 Alex Gaunt, **Diana Borsa**, Yoram Bachrach. *Training deep neural nets to aggregate crowdsourced responses,* (UAI 2016 – oral).

o7 Elad Yom-Tov, **Diana Borsa**, Andrew C Hayward, Rachel McKendry, Ingemar J Cox. *Automatic identification of Web-based risk markers for health events,* (Journal of medical Internet research 17 (1), e29).

o8 **Diana Borsa**, Nicolas Heess, Bilal Piot, Siqi Liu, L. Hasenclever, Rémi Munos, Oliver Pietquin. *Observational learning by reinforcement learning,* (AAMAS 2019 – oral).

o9 Anna Harutyunyan, Will Dabney, **Diana Borsa**, Nicolas Heess, Rémi Munos, Doina Precup. *The Termination Critic,* (AISTATS 2019, RLDM 2019).

o10 Tom Schaul, **Diana Borsa**, Joseph Modayil, Razvan Pascanu. *Ray Interference: a Source of Plateaus in Deep Reinforcement Learning,* (RLDM 2019).

1.2 Structure of the Thesis

In this section, we include an overview of the structure of the thesis. In Chapter 2, we review a set of preliminaries. This will include a brief introduction into reinforcement learning and core concepts underlying the learning of value functions, like dynamic programming, policy improvement methods and sample-based approximations to these solutions. The second part of this chapter should serve as a swift review of the transfer learning paradigm, its potential benefits and different mechanisms of representing and transferring knowledge in RL. We will then dive into the research part of this thesis which is organised in two parts.

In the first part (Chapter 3-4) we study the problem of multitask learning optimal action-value functions corresponding to different reward signals in a persistent environment. In this part, we opted for an offline setting where we only have access to a recorded set of experiences collected under some shared behaviour policy. Under these conditions, we wanted to see if the joint learning of the tasks can indeed lead to positive transfer between the tasks and improve the performance of the inferred control policies under restricted sample budgets. To test this, we focus our attention on learning a shared linear representation across tasks, as proposed in (Argyriou et al., 2008), to support the intermediate regression problems encountered during each iteration of multitask fitted Q-iteration (Chapter 3) and approximate policy iteration (Chapter 4) via residual methods.

In the second part (Chapter 5-6), we consider a different way of representing knowledge in a Markov decision process via *successor features*. In this part, we consider a more online scenario where the agent can interact with its environment and actively gather its experience. Here, we also look at more challenging domains that require learning of more powerful representations, beyond the linear approximators class considered previously. Given the online setting, the more natural question here becomes assessing the ability of the agent to (re)use information from the past in order to tackle more efficiently a new task. In this setting, we will be primarily concerned with generalising to a new task and speeding up the learning or adaptation phase to a new task.

Finally, Chapter 7 concludes this writeup with a summary of the work, discussing the main takeaways, limitations and future research directions for this work. The appendices are structured in two parts: one containing the derivation of some of the key results and proofs that were omitted in the main body of the thesis and the second one containing more detailed experimental outlines and additional results.

2
Theoretical Preliminaries

The first part of this chapter is meant to provide a brief introduction to Markov Decision Processes (MDPs) and the basics concepts in Reinforcement Learning (RL), the ones most relevant to the methods we will be employing to learn value functions in the next chapters. We shall start by stating the model assumptions and introducing the formalism. We will then first consider MDPs with known transition dynamics and reward functions, but we will see how these general principles translate to algorithms that only require access to samples from these MDPs. In this work, we will concern ourselves only with model-free algorithms, which can learn (optimal) value functions without an explicit model of the world, solely by interacting with the environment. Lastly, we introduce the idea of functional approximation to represent the value-functions of interest. In this context, we detail two well-known methods, Approximate Policy Iteration (API) and Fitted Q-Iteration (FQI), that use such an approximation. These methods form the basis for algorithms in Chapters 3-4. A more comprehensive overview of reinforcement learning methods and algorithms can be found in (Sutton and Barto, 1998; Szepesvári, 2010) and references therein. In the second part of this chapter, we formally introduce the paradigm of transfer learning and review the different instances of transfer one could consider in the context of

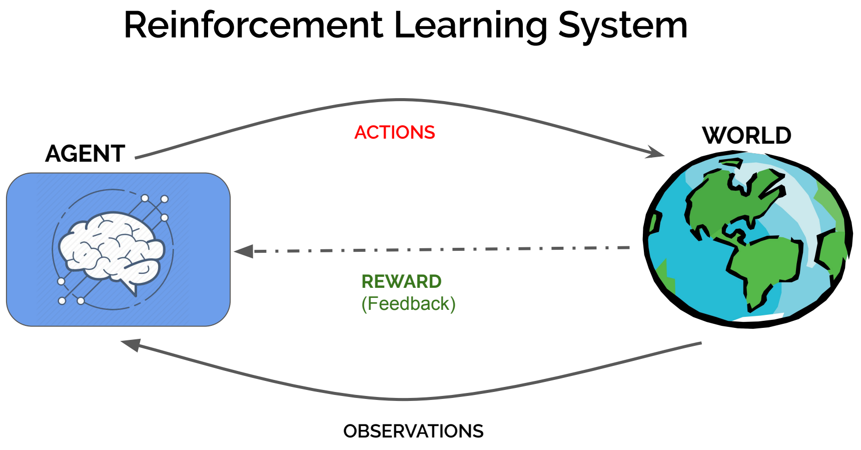


Figure 2.1.1: Depiction of a Reinforcement Learning (RL) system, describing the interaction between an agent and its environment (world).

the RL problems targeted throughout this thesis.

2.1 Reinforcement Learning

Reinforcement learning (RL) is a general formalism for reasoning about sequential decision-making problems. As depicted in Figure 2.1.1, an RL system is made up of two components: an agent and an environment. The interaction between these two components can be described as follows: the agent can execute actions in the environment; once an action is chosen, the system transitions into a new state, observable, sometimes partially, to the agent. In addition, after each such transition, the agent receives a, potentially sparse, reward signal quantifying the goodness of this transition.

2.1.1 Markov Decision Processes

More formally, in this work, we will use Markov Decision Processes (MDP-s) to describe RL systems and the interaction between the environment and our learning agent(s). As the name suggests, the common assumption in MDPs is that the environment dynamics can be described by a Markov Process, i.e. *the future is independent of the past given the* *present state* (see formal description in Definition 2.1). In other words, we assume that the current state of the system holds all the information regarding the future and knowing any additional information about the history of the process will not add any knowledge.

Definition 2.1: Markov Process

A Markov process, defined by a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$ where $\mathcal{S}$ is a set of states, $\mathcal{P} : \mathcal{S} \times \mathcal{S} \rightarrow [0, 1]$ is a state transition probability matrix describing the probability of transitioning to state s' when starting in s, $\mathcal{P}(s'|s)$, is a *memoryless random process*. That is, for any time step t:

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_1, \cdots, s_t), \forall s_i \in \mathcal{S} \tag{2.1}$$

We are interested in the interaction of the agent(s) with the environment and in most problems, we aim to identify which actions/controls are good actions and which are bad. We denote the set of all actions available to us as $\mathcal{A}$ and for now, we assumed this set to be finite. Formally, ”the goodness” of an action is quantified by an (immediate) reward signal, R, given by the environment after each action or each transition. This gives rise to the reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, which assigns a value to every state indicating the expected reward that can be achieved starting from this state, s and performing action a:

$$r(s, a) = \mathbb{E}[R|s, a] \tag{2.2}$$

Putting it all together, we now have almost all the ingredients to formally define a Markov Decision Process - see Definition 2.2 below:

Definition 2.2: Markov Decision Process

A Markov Decision Process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$ where $\mathcal{S}$ represents the set of states, $\mathcal{A}$ is the set of actions the agent can take in the environment; $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ denotes the state transition probability, $\mathcal{P}(s'|a, s)$, which fully describes how the system will evolve; $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a reward function $r(s, a)$ and $\gamma : \mathcal{S} \rightarrow [0, 1]$ is a continuation function (commonly known as discount factor).

Although in general $\gamma : \mathcal{S} \rightarrow [0, 1]$ can be an arbitrary function of the state s, throughout this work, we will consider a constant discount factor $\gamma(s) = \gamma \in [0, 1], \forall s \in \mathcal{S}$, unless s is a terminal state where $\gamma(s_{terminal}) = 0$. A terminal state marks the end of an episode, in an episodic setting, and setting γ to 0 beyond this state simply implies that we only care about what happens within the current episode and rewards are not accumulated across episodes, just within episodes.

Based on the reward signal, we can define the return, G_t as the total discounted rewards collected from time-step t onwards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t} R_T = \sum_{k=0}^{T} \gamma^k R_{t+k+1} \tag{2.3}$$

where $\gamma \in [0, 1]$ is the discount factor[1] - which says immediate rewards are more important than rewards seen further in the future (which are discounted by a factor of γ^k, after k-steps), and $T \to \infty$ for non-episodic or continuing environments.

In order to characterise the behaviour of an agent within a MDP, we will introduce a function that for every state $s \in \mathcal{S}$ returns a distribution over actions, characterising the likelihood of taking each action at a particular state s. We will call this function the policy of an agent and we will denote it as $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$:

$$\pi(s, a) = P(a|s), \forall a \in \mathcal{A}, \forall s \in \mathcal{S} \tag{2.4}$$

Depending on the nature of the policy, π, we will use the following notational convention:

- *Deterministic policies*: $\pi : \mathcal{S} \to \mathcal{A}$ and $\pi(s)$ denotes the action chosen on state s. Note that in this case, the above probability distribution will concentrate all mass on only one action $a = \pi(s)$.

- *Stochastic policies*: $\pi : \mathcal{S} \to Distr(\mathcal{A})$ and $\pi(a|s)$ denotes the probability of taking action a in state s, where $Distr(\mathcal{A})$ denotes the set of distributions on the set $\mathcal{A}$.

In most RL problems we are interested in either discovering a good (or even optimal) such policy, π or simply to evaluate a particular behaviour, π, if one is already prescribed. In the next section, we will show how one can build this type of evaluations.

[1] Note that it is sometimes possible to use undiscounted MPDs, i.e. $\gamma = 1$, if all sequences terminate. In general, we are going to work with discounted MDPs mainly for mathematical convenience, but also in the financial setting the discount factor arises naturally because of the (risk-free) interest rates: immediate rewards may earn more interest over the delayed rewards

2.1.2 Value Functions

Given an MDP, $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, and a policy π, we can define a value function, $V(s)$, quantifying how good a particular state $s \in \mathcal{S}$, is in terms of potential future rewards - in essence, this summarises the future expected interaction with the MDP, in terms of the cumulative reward. Of course, the 'goodness' of such state is at least partially related to what our agent's behaviour will be in the next steps. Thus formally we will define the *state-value function, V^π,* with respect to some policy π as the discounted return G_t that the agent is expected to collect starting from state $s_t = s$ and behaving according to π:

$$V^\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|s_t = s\right], \forall s \in \mathcal{S}. \tag{2.5}$$

Note that G_t is a random variable, similar to $\{R_\tau\}_{\tau=\overline{1,T}}$ and the above captures its expected value under the random process defined by the MDP and policy π. In a similar manner, we can define the *action-value function, $Q^\pi(s,a)$* with respected to a policy π as the expected return an agent will gain starting from state $s_t = s$, taking action $a_t = a$ in this state and thereafter acting according to policy π:

$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|s_t = s, a_t = a\right], \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.6}$$

Note that by definition, there exists a very close relationship between the two value functions above, that ensure consistency. In particular, we can write:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[G_t|s_t = s] = \mathbb{E}_{a\sim\pi(.|s)}\left[\underbrace{\mathbb{E}_\pi[G_t|s_t = s, a = a_t]}_{Q^\pi(s,a)}\right] \\ &= \mathbb{E}_{a\sim\pi(.|s)}\left[Q^\pi(s,a)\right] = \sum_{a\in\mathcal{A}} \pi(a|s) Q^\pi(s,a) \end{aligned} \tag{2.7}$$

and respectively:

$$\begin{aligned}
Q^{\pi}(s,a) &= \mathbb{E}_{\pi}[G_t|s_t = s, a_t = a] = \mathbb{E}_{\pi}\left[R_{t+1} + \sum_{k=1}^{\infty}\gamma^k R_{t+k+1}|s_t = s, a_t = a\right] \\
&= r(s,a) + \gamma\mathbb{E}_{\pi}\left[\underbrace{\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}}_{G_{t+1}}|s_t = s, a_t = a\right] \\
&= r(s,a) + \gamma\mathbb{E}_{s_{t+1}\sim\mathcal{P}(.|s_t=s,a_t=a)}\underbrace{\left[\mathbb{E}_{\pi}\left[G_{t+1}|s_{t+1}\right]\right]}_{V^{\pi}(s_{t+1})}
\end{aligned}$$

Moreover these lead to a recursive relationship for both value functions that will be at the core of many of our optimisation techniques. These recurrences are commonly known as the *Bellman Expectation Equations* (Definition 2.1) and are summarised below:

Theorem 2.1: Bellman Expectation Equations

Given an MDP, $\mathcal{M} = \langle\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma\rangle$, for any policy π, the value functions obey the following expectation equations:

$$V^{\pi}(s) = \sum_a \pi(s,a)\left[r(s,a) + \gamma\sum_{s'}\mathcal{P}(s'|a,s)V^{\pi}(s')\right] \tag{2.8}$$

$$Q^{\pi}(s,a) = r(s,a) + \gamma\sum_{s'}\mathcal{P}(s'|a,s)\sum_{a'\in\mathcal{A}}\pi(a'|s')Q^{\pi}(s',a') \tag{2.9}$$

Moreover, based on the above equations we can define an operator, T^{π}, for each policy π whose unique fixed point satisfy Theorem 2.1.

Definition 2.3: Bellman Expectation Operator

Given an MDP, $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, let $\mathcal{Q} \equiv \mathcal{Q}_{\mathcal{S},\mathcal{A}}$ be the space of bounded real-valued functions over $\mathcal{S} \times \mathcal{A}$. Then for any policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, we can the define, point-wise, the *Bellman Expectation operator* $T^{\pi}_{\mathcal{Q}} : \mathcal{Q} \to \mathcal{Q}$ as:

$$(T^{\pi}_{\mathcal{Q}} f)(s, a) = r(s, a) + \gamma \sum_{s'} \mathcal{P}(s'|a, s) \sum_{a' \in \mathcal{A}} \pi(a'|s') f(s', a')\, , \forall f \in \mathcal{Q} \qquad (2.10)$$

This operator has *one unique fixed point* which corresponds to the *action-value function* Q^{π} in our MDP $\mathcal{M}$. That is: $T^{\pi}_{\mathcal{Q}} Q^{\pi}(s, a) = Q^{\pi}(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}$.
Similarly, for the state-value function V^{π}, we can define the Bellman Expectation operator $T^{\pi}_{\mathcal{V}} : \mathcal{V} \to \mathcal{V}$, where $\mathcal{V}$ is the space of bounded real-valued functions taking values from $\mathcal{S}$, as:

$$(T^{\pi}_{\mathcal{V}} f)(s) = \sum_{a} \pi(s, a) \left[r(s, a) + \gamma \sum_{s'} \mathcal{P}(s'|a, s) f(s') \right], \forall f \in \mathcal{V} \qquad (2.11)$$

This operator as well has only *one unique fixed point* that corresponds to the state-value function V^{π}: $(T^{\pi}_{\mathcal{V}} V^{\pi})(s) = V^{\pi}(s), \forall s \in \mathcal{S}$.

The proof of the above is omitted in this exposition but can be found in (Szepesvári, 2010). Also in general, we will omit the indices $\mathcal{Q}$ and respectively $\mathcal{V}$ when talking about these operators, but it will be evident from the cardinality of the arguments the function is called with, which of them we are referring to.

2.1.3 Greedy Policy Improvement

Given an MDP $\mathcal{M}$, let Q^{π} be the action-value function for a stationary policy π. Now, let us consider the greedy policy π_{greedy} with respect to this action-value function:

$$\pi_{greedy}(a|s) = \begin{cases} 1, \text{ for } a = \arg\max_{b \in \mathcal{A}} Q^{\pi}(s, b) \\ 0, \text{ otherwise} \end{cases}$$

where we assume any ties in $\max_{b \in \mathcal{A}} Q^{\pi}(s, b)$ are resolved deterministically. In this way, we can obtain a deterministic policy $\pi_{greedy}(s)$, which *greedily* chooses at each state, the most promising action – or one of these actions if multiple achieve the same value – under policy π. This results in a policy that is guaranteed to improve the expected return at every state. The proof of this result can be found in Appendix A.1.1 (or Puterman (1994)).

Theorem 2.2: Greedy Policy Improvement

Given an MDP, $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, a policy π and its associated action-value Q^π, then if we consider the greedy policy $\pi_{greedy}(s) = \arg\max_a Q^\pi(s,a)$ we have that:

$$Q^{\pi_{greedy}}(s,a) \geq Q^\pi(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$$

with equality satisfied only when π *can not be improved any more*: $\pi(s,a) \in \arg\max_\mu Q^\mu(s,a)$, $\forall s \in \mathcal{S}, a \in \mathcal{A}$.

2.1.4 Optimality in MDPs

For control problems, we are interested in finding an optimal policy where we define an optimal policy as the policy that achieves the greatest expected return. First let us define the optimal expected return, or the optimal state-value function, V^* as:

$$V^*(s) = \max_\pi [V^\pi(s)], \forall s \in \mathcal{S} \tag{2.12}$$

That is, for each state s we define the optimal value function as the expected return if after state s we were to act optimally. Likewise one can define the optimal action-value as:

$$Q^*(s,a) = \max_\pi [Q^\pi(s,a)], \forall (s,a) \in \mathcal{S} \times \mathcal{A} \tag{2.13}$$

An optimal policy π^*, which might not be unique, is defined as any policy that attains this optimal value function:

$$\pi^* \text{ is } \mathit{optimal} \Leftrightarrow V^{\pi^*} = V^* \; (\Leftrightarrow Q^{\pi^*} = Q^*) \tag{2.14}$$

Note that such an optimal policy always exists for an given MDP. Moreover, given the optimal action-value function Q^*, one can easily construct such an optimal policy by just *acting greedily* with respect to the action-value function.

$$\pi^*_{greedy}(s) = \arg\max_a Q^*(s,a) \Rightarrow \pi^*_{greedy}(s) \text{ is optimal.} \tag{2.15}$$

The above follows immediately from the Theorem 2.2 (Greedy Policy Improvement) which states that the greedy policy on $Q^* = Q^{\pi^*}$ will improve the value function unless already at optimality, thus it follows that $Q^{\pi_{greedy}} = Q^*$ – therefore π_{greedy} is an optimal policy.

Theorem 2.3: Bellman Optimality Equations

The optimal value functions satisfy the following equations:

$$V^*(s) = \max_{a \in \mathcal{A}} \underbrace{\left[r(s,a) + \gamma \sum_{s'} \mathcal{P}(s'|a,s) V^\pi(s') \right]}_{Q^*(s,a)} \tag{2.16}$$

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} \mathcal{P}(s'|a,s) \underbrace{\max_{a' \in \mathcal{A}} \left[Q^*(s',a') \right]}_{V^*(s')} \tag{2.17}$$

for any $s \in \mathcal{S}$, and any $a \in \mathcal{A}$.

Naturally, there is a strong relationship between the optimal value function V^* and the optimal state-action value function Q^*. This can be derived by recalling the relation between the state-value function and the action-value function in Eq. 2.7 and applying it for the optimal policy $\pi = \pi^*_{greedy}$ defined in Eq. 2.15. Using this relation in conjunction with Bellman Expectation Equations we can derive the *Bellman Optimality Equations* (Theorem 2.3). As we shall see later, these play an essential role in solving value-based control problems.

As before, based on the above equations we can define an optimality operator , T^*, whose unique fixed point satisfies Theorem 2.3 – see Definition 2.4 below.

Definition 2.4: Bellman Optimality Operators

Given an MDP, $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, let $\mathcal{Q} \equiv \mathcal{Q}_{\mathcal{S},\mathcal{A}}$ be the space of bounded real-valued functions over $\mathcal{S} \times \mathcal{A}$, we can the define, point-wise, the *Bellman Optimality operator* $T^*_{\mathcal{Q}} : \mathcal{Q} \to \mathcal{Q}$ as:

$$(T^*_{\mathcal{Q}} f)(s, a) = r(s, a) + \gamma \sum_{s'} \mathcal{P}(s'|a, s) \max_{a' \in \mathcal{A}} f(s', a') , \forall f \in \mathcal{Q} \qquad (2.18)$$

This operator has *one unique fixed point* which corresponds to the *optimal action-value function* Q^* in our MDP $\mathcal{M}$. That is: $T^*_{\mathcal{Q}} Q^*(s, a) = Q^*(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}$. Similarly, for the state-value function V^*, we can define the Bellman Optimality operator $T^*_{\mathcal{V}} : \mathcal{V} \to \mathcal{V}$, where $\mathcal{V}$ is the space of bounded real-valued functions taking values from $\mathcal{S}$, as:

$$(T^*_{\mathcal{V}} f)(s) = \max_a \left[r(s, a) + \gamma \sum_{s'} \mathcal{P}(s'|a, s) f(s') \right], \forall f \in \mathcal{V} \qquad (2.19)$$

This operator as well has only *one unique fixed point* that corresponds to the optimal state-value function V^*: $(T^*_{\mathcal{V}} V^*)(s) = V^*(s), \forall s \in \mathcal{S}$.

For convenience, moving forward, we will omit the index ($\mathcal{V}$ or $\mathcal{Q}$) associated with the space of functions over which this operator is defined and simply use the notation T^*.

2.1.5 Dynamic Programming (Perfect knowledge RL)

In the following section, we assume complete knowledge of the MDP in question and we are going to explore how one can address two major classes of problems in RL: prediction (or evaluation of a particular policy) and control (finding the optimal policy). In general, please note that we usually only have access to (trajectory) samples $R, S' \sim P(r, s'|s, a)$ from the underlying MDP, but we will see that the principles highlighted in this section, can be very naturally extended and adapted to work with sampled experience.

Under perfect knowledge – that is, we have access to $\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma$ – the simplest way we can perform both prediction and control is via *dynamic programming*. This follows immediately from the Bellman Equations which expose the compositional structure present in the solutions we are targeting. A simple closer inspection of these equations reveal that one can first solve sub-problems (like $V(s')$) and then put them together to gradually build the overall solution in these optimisation problems. Note also that these sub-problems tend to reoccur and thus these partial solutions, once found and cached in, can be easily reused. This is a core principle in dynamic programming. This paired to the Markov assumption which provides a simple, step-by-step decomposition, ensures a particularly rich structure in the solution space that can be effectively exploited by RL algorithms.

2.1.5.1 Preliminaries

Most of the methods we are going to the discuss in this chapter, as well as the majority of the methods in RL trying to estimate value functions, consist of an iterative procedure of building and refining these values at every step. The convergence of these methods rely heavily on the properties of the two operators introduced in the previous sections: the expectation operator T^π and respectively the optimality operator T^*.

Definition 2.5: Contraction mapping

Let $\mathcal{B}$ be a Banach space. An operator $T : \mathcal{B} \rightarrow \mathcal{B}$ is a contraction mapping if for any $u, v \in \mathcal{B}, \exists \alpha \in [0, 1)$ such that:

$$||Tv - Tu|| \leq \alpha ||v - u||$$

One can show that both of these (T^π and T^*) are in fact contraction operators with the contraction constant given by γ (see Definition 2.5). In the light of this fact, the convergence if these iterative processes follows from the Banach's Fixed Point Theorem (Theorem 2.4). In a nutshell, the contraction property ensured that with each and every application of these operators, we get closer and closer to their respective fixed points, and under the right conditions, ultimately converging to them.

Theorem 2.4: Banach Fixed Point Theorem

Let $\mathcal{B}$ be a Banach space and $T : \mathcal{B} \to \mathcal{B}$ a γ-contraction mapping, then:

1. T has a unique fixed point $x \in \mathcal{B}$, s.t. $\exists! x^* \in \mathcal{B}$ s.t. $Tx^* = x^*$

2. $\forall x_0 \in \mathcal{B}$, the sequence $x_{n+1} = Tx_n$ converges to T's unique fixed point x^* in a geometric fashion:

$$\|x_n - x^*\| \leq \gamma^n \|x_0 - x^*\| \tag{2.20}$$

Thus $\lim_{n\to\infty} \|x_n - x^*\| \leq \lim_{n\to\infty} (\gamma^n \|x_0 - x^*\|) = 0$.

2.1.5.2 Policy Evaluation

We begin by considering the prediction problem – that is the problem of evaluating how good a particular policy is in our MDP. This is formally described as:

Problem 1 (Prediction in a known MDP)**.** *Given an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$ and a policy π, or alternatively a Markov Reward Process (MRP) $\langle \mathcal{S}, \mathcal{A}^\pi, \mathcal{P}^\pi, r^\pi, \gamma \rangle$, compute the action-value function $Q^\pi(s,a)$ (or state-value function $V^\pi(s)$).*

A solution to the above problem is known as *Policy Evaluation* and involves using dynamic programming together with the Bellman Expectation equations with respect to the policy π, we are trying to evaluate. A full description is provided in Algorithm 2.1 below.

Proof: Convergence of Iterative Policy Evaluation V^π (Algorithm 2.1.) The update equations can be re-written using the Bellman expectation operator T^π:

$$\begin{aligned} T^\pi(V)(s) &= \sum_{a\in\mathcal{A}} \pi(s,a) r(s,a) + \gamma \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} P(s'|s,a)V(s') \\ &= r^\pi(s) + \gamma \sum_{s'\in\mathcal{S}} P^\pi(s'|s)V(s') \\ \Rightarrow T^\pi(V) &= r^\pi + \gamma \mathcal{P}^\pi V \end{aligned}$$

As a reminder, note that V^π is a fixed point of T^π. Moreover this fixed point turns out to

Algorithm 2.1 Iterative Policy Evaluation

```
Require: MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$ and policy $\pi$
  Problem: Evaluate a given policy $\pi$
  Solution: Iterative backups using the Bellman Expectation Eq. (Theorem 2.1)
  Initialize: $V^{(0)}$ (or $Q^{(0)}$)
  for each $k \leq$ MAX_ITERS and for each state $s \in \mathcal{S}$ do
      $V^{(k+1)}(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \underbrace{\left( R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^{(k)}(s') \right)}_{Q^{(k)}(s,a)}$
    or if interested in the action-value function, for all $(s,a) \in \mathcal{S} \times \mathcal{A}$
      $Q^{(k+1)}(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \underbrace{\sum_{a' \in \mathcal{A}} \pi(s',a') Q^{(k)}(s',a')}_{V^{(k)}(s')}$
  end for
  return $V^{(1)} \to V^{(2)} \to V^{(3)} \to \cdots \to V^{\pi}$ (or $Q^{(1)} \to Q^{(2)} \to Q^{(3)} \to \cdots \to Q^{\pi}$)
```

be unique. This follows immediately from the fact that T^{π} is the L_{∞}-norm [2] contraction through the direct application of the *Banach's Fixed Point theorem* (Theorem 2.4). All that remains to be shown is that T^{π} is indeed a γ-contraction, with respect to the ∞-norm:

$$\begin{aligned}
\|T^{\pi}(V) - T^{\pi}(V')\|_{\infty} &= \|(r^{\pi} + \gamma \mathcal{P}^{\pi} V) - (r^{\pi} + \gamma \mathcal{P}^{\pi} V')\|_{\infty} \\
&= \|\gamma \mathcal{P}^{\pi}(V - V')\|_{\infty} \\
&\leq \gamma \|\mathcal{P}^{\pi} \|V - V'\|_{\infty}\|_{\infty} \\
&\leq \gamma \|V - V'\|_{\infty}
\end{aligned}$$

Thus, repeated applications of the Bellman Expectation operator, T^{π}, leads to:

$$\begin{aligned}
\|(T^{\pi})^{k+1}(V) - (T^{\pi})^{k+1}(V^{\pi})\|_{\infty} &\leq \gamma \|(T^{\pi})^{k}(V) - (T^{\pi})^{k}(V^{\pi})\|_{\infty} \\
&\leq \gamma^{(k+1)} \|V - V^{\pi}\|_{\infty}
\end{aligned}$$

[2] Where the ∞-norm is defined as: $\|V\|_{\infty} = \max_{s \in \mathcal{S}} V(s)$

Now remember that V^π is a fixed point for T^π, thus $(T^\pi)^{k+1}(V^\pi) = V^\pi$. Therefore we get:

$$\|(T^\pi)^{k+1}(V) - V^\pi\|_\infty \leq \gamma^{(k+1)} \|V - V^\pi\|_\infty \tag{2.21}$$

Thus, as $k \to \infty$, the repeated applications of the Bellman Expectation operator drives the value function closer and closer to the unique fixed point, $(T^\pi)^{(k+1)}(V) \to V^\pi$. □

The proof for the action-values $Q(s, a)$ rests on the same contraction-based argument to show convergence to the fixed point $Q^\pi(s, a)$, in a geometric fashion.

2.1.5.3 Value Iteration

Now we turn our attention to the harder problem of inferring how to optimally behave in a known MDP. In this case we want to find the optimal way to act in this environment, in order to maximise our expected return. The control problem can be formalise as follows:

Problem 2 (Control in a known MDP). *Given an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, compute an optimal policy π^* (or its optimal value function V^*/Q^*).*

Although one could try to optimise for the policy directly, this might be hard to do as the 'goodness' of the policy is assessed by its corresponding value function Q^π and as we have seen in the last section computing this is not immediate – in fact this is a problem of its own. An alternative way of approaching the control problem is to try to build the optimal value function itself and then, as discussed previously, one can obtain an optimal behaviour policy by greedification. The advantage of such approach is that the optimal value function is always unique – although this might correspond to multiple optimal policies. Moreover, the Bellman optimality operator turns out to be also a contraction mapping. Thus we can devise a similar procedure as in Algorithm 2.1, but now we will be using the Bellman Optimality equations to gradually improve our value-function till we reach the optimal value. A full description of this algorithm is provided in Algorithm 2.2, along with a proof of convergence to the unique optimal V^*(resp. Q^*).

Proof: Convergence of Value Iteration to V^* (Algorithm 2.2). Similar to the above proof, let

```
Algorithm 2.2 Value Iteration
Require: MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$
  Problem: Compute the optimal value function $V^*$ (or $Q^*$)
  Solution: Iterative updates/backups using the Bellman Optimality Equation. (Theorem 2.3)
  Initialize: $V^{(0)}$ (or $Q^{(0)}$)
  for each $k \leq$ MAX_ITERS and for each state $s \in \mathcal{S}$ do
      $V^{(k+1)}(s) = \max_{a \in \mathcal{A}} \underbrace{\left( r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^{(k)}(s') \right)}_{Q^{(k)}(s,a)}$
    or if interested in the action-value function, for all $(s,a) \in \mathcal{S} \times \mathcal{A}$:
      $Q^{(k+1)}(s,a) = \left( r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \underbrace{\left[ \max_{a' \in \mathcal{A}} Q^{(k)}(s',a') \right]}_{V^{(k)}(s')} \right)$
  end for
  return $V^{(1)} \to V^{(2)} \to V^{(3)} \to \cdots \to V^*$ (or $Q^{(1)} \to Q^{(2)} \to Q^{(3)} \to \cdots \to Q^*$)
```

us recall the Bellman optimality operator T^*:

$$
\begin{aligned}
(T^* V)(s) &= \max_{a \in \mathcal{A}} Q(s,a) \\
&= \max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s') \right] \\
&= \max_{a \in \mathcal{A}} [r^a + \gamma \mathcal{P}^a_{ss'} V]
\end{aligned}
$$

First, we will prove that T^* is a $\gamma-$ contraction mapping:

$$\begin{aligned}
\|T^*V - T^*U\|_\infty &= \left\|\max_{a\in\mathcal{A}}\left[r^a + \gamma\mathcal{P}^a_{ss'}V\right] - \max_{b\in\mathcal{A}}\left[r^b + \gamma\mathcal{P}^b_{ss'}U\right]\right\|_\infty \\
&\leq \left\|\max_{a\in\mathcal{A}}\left[(r^a + \gamma\mathcal{P}^a_{ss'}V) - (r^a + \gamma\mathcal{P}^a_{ss'}U)\right]\right\|_\infty \\
&\leq \left\|\max_{a\in\mathcal{A}}\left[\gamma\mathcal{P}^a_{ss'}(V-U)\right]\right\|_\infty = \gamma\left\|\max_{a\in\mathcal{A}}\left[\mathcal{P}^a_{ss'}(V-U)\right]\right\|_\infty \\
&\leq \gamma\max_{s'\in\mathcal{S}}\|V(s') - U(s')\| \qquad\qquad (\text{as } \textstyle\sum_{s'} P(s'|s,a) = 1, \forall a) \\
&\leq \gamma\left\|(V-U)\right\|_\infty
\end{aligned}$$

This implies via the Banach's Fixed Point theorem, that T^* has a unique fixed point and that repeated applications of the operator induces a sequence that converges to its fixed point, V^*:

$$V^{(k+1)} = TV^{(k)} \Rightarrow (V^{(k)})_{k\geq 0} \rightarrow_{\|.\|_\infty} V^* \tag{2.22}$$

at a geometric rate:

$$\|V^{(k+1)} - V^*\|_\infty = \|(T^*)^{k+1}V_0 - (T^*)^{k+1}V^*\|_\infty \leq \gamma^{k+1}\|V_0 - V^*\|_\infty \tag{2.23}$$

Since this fixed point is unique, we can conclude that:

The sequence $(V^{(k)})_{k\geq 0}$ defined by $V^{(k+1)} = TV^{(k)}$ converges in $\|.\|_\infty$ to V^*,

for all starting points $\forall V^{(0)}$as $k \rightarrow \infty$.

□

2.1.5.4 Generalised Policy Iteration

We have seen in the previous section how one can construct an optimal policy by first computing the optimal value function and then acting greedily based on this value. Although convergence is guaranteed in the value function space, in practice the policy tends to converge faster than the actual value-function. That is, we could have sought to act greedily with respect to an intermediate value $Q^{(k)}(s, a)$ and that could already provided an optimal policy even before the sequence of value-functions, $(Q^{(k)})_{k\geq 0}$, reaches optimality.

Building on this observation, one can construct a general procedure called *(Generalised)*

Policy Iteration (Algorithm 2.3) where we evaluate our current policy, π, by building its corresponding value function and then based on this value we seek to improve our behaviour (for instance by acting greedily with respect to the current value function V^{π}) leading to π^{new}. By iterating this procedure we continue to improve our policy and, at least in the finite case, we are guaranteed to converge to both an optimal policy and the optimal value-function. This principle actually results in a very general framework that we will use abundantly for planning in control problems, with various instantiations of the **Policy Evaluation** and **Policy Improvement** steps in Algorithm 2.3 below.

Algorithm 2.3 Generalised Policy Iteration

```
while (Not converged) do
    (Policy Evaluation): Given policy $\pi$, estimate/evaluate $Q = Q^{\pi}$.
    (e.g. by Iterative Policy Evaluation - see Algorithm 2.1)
    (Policy Improvement): Generate $\pi^{new} \geq \pi$.
    (e.g. by Greedy Policy Improvement - see Section 2.1.3)
    if $Q^{\pi} == Q^{\pi_{new}}$ then
        We converged. TERMINATE
    end if
    This becomes the new policy for the evaluation step $\pi = \pi^{new}$.
end while
```

In principle, one can employ any method of improvement in the second step, but the greedy alternative is both easy to implement and guaranteed to improve the value-function point-wise. A slightly different improvement strategy, particular suitable for the multitask scenario, is to apply a more general policy improvement step over a number of policies, if we can obtain multiple of these evaluations. We will see that some of the methods proposed here enable us to get, essentially for free, multiple evaluations of different policies for the tasks of interest. This property also translates to a potentially better improvement step for each task under consideration.

2.1.6 Sample-based Reinforcement Learning

In the previous section, we assumed full knowledge of the MDP at hand, but in practice this is rarely the case. More commonly, we would have to interact with the environment and generate experiences from which to learn. In even more restrictive cases, we would actually only have access to recorded interactions with the MDP in question. Thus, we have to either build an explicit model of the transitions' distribution and reward function based on the sampled episodes and then apply the above methodology to these approximations or attempt to approximate the value functions of interest directly from the experience. The second approach bypasses building a model explicitly and is thus refereed to in literature as model-free reinforcement learning. Throughout this work, we will be taking this latter approach to all control and prediction problems we are going to consider.

2.1.6.1 Monte Carlo (MC)

The Monte Carlo (MC) procedure implements probably the simplest estimate of the value-functions, approximating the expected return with the empirical mean return seen over a set of samples. To ensure that well-defined returns are available, here we define Monte Carlo methods only for episodic tasks. This means that our experience will consider episodes of experience: $\mathcal{D} = \{\tau_i\}_{i=1,N}$, where $\tau_i = \{s_1, a_1, r_2, s_2, a_2, \cdots, s_T\}$, $a_t \sim \pi(.|s_t)$, $s_{t+1} \sim \mathcal{P}(.|s_t, a_t)$, $\forall t = \overline{1, T}$ is the trajectory of experience seen in episode i, by interacting with an environment $\mathcal{P}, r$ while following a behaviour policy π.

$$V^{\pi}(s) = E_{\pi}[G_t | S_t = s] \approx \frac{1}{N_s} \sum_{i=1}^{N_s} G_s^{(i)} \tag{2.24}$$

where $G^{(i)}$ is the return achieved in episode i, from state s onwards and N_s is the number of visits we made to state s in the trajectory sample set $\mathcal{D}$. If state s was not encountered in an episode i, then its corresponding return $G_s^{(i)} = 0$. Moreover, there exists two versions of this estimate: i) *first-visit* MC estimate, which takes the averages over returns starting from s only in the first time s was visited in that episode and ii) *every-visit* MC estimate, which takes the averaged over all future returns starting from s, for every visit to s. Both of these estimates will converge asymptotically (as $N_s \rightarrow \infty$) to V^{π} (Singh and Sutton, 1996; Sutton and Barto, 1998). The resulting, episodic update is of the form:

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t)\right) \tag{2.25}$$

Thus, the above equation gives us a very simple model-free policy evaluation algorithm that can be used directly for prediction or as the first step of the (generalised) policy iteration procedure for control purposes.

It is worth noting that in order to build the above MC estimate for the value function V^{π}, we need access to trajectories that were *generated by acting according to the evaluated target policy π*. This also means that for control purposes, whenever we switch the evaluation target, from one policy iteration to the other, we would need to re-generate on-policy experience for our new target. Of course, one could use importance sampling corrections to facilitate reuse of data collected under a different behaviour policy than the one we are currently trying to evaluate. Nevertheless, the importance ratios here are considered throughout a full trajectory, thus the feasibility of this approach depends quite highly on the overlap between our behaviour policy and our target for the evaluation.

2.1.6.2 Temporal Difference (TD) Learning

Although MC will converge to the true value function given enough samples, the variance of this estimator given a finite sample tends to be very high. Also note that we need to wait till the end of episode to make any updates to our current estimate of the value function, which again can be very (sample) inefficient. To address these issues we will introduce a family of algorithms called *temporal-difference (TD) algorithms* that are able to learn continuously from incomplete episodes by *bootstrapping*.

TD learning methods are really a combination of Monte Carlo learning and dynamic programming(DP) divide-and-conquer ideas explored before. Like MC methods, TD methods can learn directly from *sampled experience* without a full model of the environment. And similar to DP, TD methods will update estimates based on partial (learnt) solutions, without having to wait for the final outcome of an episode.

The simplest version of temporal-difference learning, $TD(0)$, replaces the return G_t with an estimate of it, $G_{t:t+1} \approx R_{t+1} + \gamma V(S_{t+1})$, given by the Bellman Expectation Equation. Consequently the update becomes:

$$V(S_t) \leftarrow V(S_t) + \alpha(\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{G_{t:t+1}} - V(S_t)) \tag{2.26}$$

where we will commonly refer to $G_{t:t+1} = R_{t+1} + \gamma V(S_{t+1})$ as the *TD target* and we can define the *TD error* as the mismatch between this estimated target and our current value function at this state:

$$\delta_t \quad := \quad G_{t:t+1} - V(S_t) = R_{t+1} + \gamma V(S_{t+1}) - V(S_t).$$

It is important to note that by updating our value function towards the TD target, we will, in general, be introducing some bias in our estimate of the value function. Nevertheless, it was shown in (Dayan, 1992; Dayan and Sejnowski, 1994), that TD(0) does asymptotically converge to the true value function V^{π} under a decaying learning rate, following the *Robbins–Monro conditions* [3]. In general, this estimate has a much lower variance as compared to its Monte Carlo counterpart, but can introduce a bias that might be hard to unlearn. In essence, a bad initialisation could considerably delay our convergence.

Something in-between the full history return used by the MC update and the one-step TD target, is considering computing TD targets based on multiple steps returns. Let us define the n-step return:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^{n} V(S_{t+n}) \tag{2.27}$$

and making this our new TD target(n-step TD target) towards which we update our current estimate of the value function, we obtain:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_{t:t+n} - V(S_t)) \tag{2.28}$$

Note that by setting $n = 0$, we recover $TD(0)$ and if we set $n = \infty$ we recover the Monte Carlo update in Eq. 2.25. Furthermore, we can think about combining these n-step targets (which we use as a guess for $V(S_t) = \mathbb{E}_{\pi}[G_t]$) to get a new target. We define the λ*-return*, G_t^{λ} as a linear combination of the n-step returns:

$$G_t^{\lambda} = \sum_{n=1}^{\infty} a_n G_{t:t+n} \tag{2.29}$$

with $a_n = (1-\lambda)\lambda^n$, and where $\lambda \in (0,1)$[4] is the decaying factor across the n-step returns $G_{t:t+n+1}$. As defined above, in order to compute G_t^{λ} we would need to wait till the end of the episode in order to update our value function - same as for MC learning. Fortunately there is a backward view of TD(λ) that provides us with an update rule that can be immediately be applied after seeing just one additional transition, as we did in TD(0). For this, we define the eligibility traces $e(s)$ which, for each state s, will weight the corresponding TD

[3] This convergence holds for table-based representation of the value functions, but does not generally hold for arbitrary function approximation of the value functions (Dayan, 1992)

[4] Note that $\sum_{n=1}^{\infty} a_n = \sum_{n=1}^{\infty}(1-\lambda)\lambda^n = (1-\lambda)\sum_{n=1}^{\infty}\lambda^n = (1-\lambda)\cdot\frac{1}{(1-\lambda)} = 1$. Thus G_t^{λ} is actually a convex combination of the n-step returns.

errors, as follows:

$$e_0(s) = 0, e_t(s) = \gamma\lambda e_{t-1}(s) + \mathbb{I}(S_t == s) \tag{2.30}$$

In doing so, we can dynamically assign credit to the most recently and the most frequently visited states. The resulting algorithm is described in Algorithm 2.4 below and relies on re-writing the episode TD error, $G_t^\lambda - V(S_t)$, in terms of the per-step TD errors, $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$, and weighting them accordingly to these "visit counts" implemented by the eligibility traces. It has been shown that using eligibility traces, with an offline update at the end the episodes, the forward and backward view of TD(λ) are equivalent - see (Sutton and Barto, 1998) and (Singh and Sutton, 1996) for details. More recently, the same equivalence has been proven to be achieved for online updates, under a slightly different definition of the eligibility trace - please consult (Seijen and Sutton, 2014; Sutton et al., 2014) and (van Hasselt et al., 2014) for full details.

Algorithm 2.4 TD(λ) for Policy Evaluation (Result: $V \rightarrow V^\pi$)

```
Initialise $V(s) = V_0(s), \forall s$
for all episodes do
    $e_0(s) = 0, \forall s \in \mathcal{S}$
    Initialise $s = s_0 \sim \mu_0$ (starting state)
    for all steps $t = 0, T$ of an episode do
        Select action $a_t \sim \pi(.|s_t)$
        Take action $a_t$ and observe reward $r_{t+1}$ and next state $s_{t+1}$
        Update:
        $\delta_t \leftarrow r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$
        $e_t(s_t) \leftarrow e_{t-1}(s_t) + 1$
        for all $s \in \mathcal{S}$ do
            $V(s) \leftarrow V(s) + \alpha\delta_t e_t(s)$
            $e_t(s) \leftarrow \gamma\lambda e_t(s)$
        end for
    end for
end for
```

2.1.6.3 TD control

So far, we have looked at how one can approach the policy evaluation problem for an unknown MDP, where we have access only to sampled trajectories. In this section, we will revisit the control problem under the sample-based regime. As a reminder, the control problem aims to learn the optimal policy or the optimal value function Q^*. Note that in the model-free control problem we are interested in estimating the state-action value function, $Q^*(s, a)$ rather than $V^*(s)$ as this will immediately give us greedily the optimal policy without an explicit model of the immediate reward function. The simplest way to do this, is to recall the framework of *Generalised Policy Iteration* (Algorithm 2.3) and simply plug into the policy evaluation step, one of the model-free evaluation algorithms introduced in the previous section (MC, TD(0), TD(λ)). The policy improvement step can remain the same: greedy improvement (or $\varepsilon-$ greedy[5] to insure some exploration).

Another crucial observation we need to make is that in order to build the estimate Q^π at each policy evaluation step in a model-free fashion, we will need experience(samples) generated by acting out the policy we are trying to evaluate. This means that after each policy improvement step, when our policy changes, we need to interact with the environment to collect samples for the current evaluation step. This can be very sample-inefficient and in cases where environment interactions are costly, a straightforward application of on-policy TD might prove to be too expensive. An explicit focus of this work will be to increase the sample efficiency of RL algorithms. As such, most of the time, we will be looking at off-policy methods that would allow us to reuse transitions previously generated.

2.1.6.4 Q-Learning

Fortunately, it turns out that we can do model-free control without having to act out the intermediate policies encountered throughout our optimisation. Instead we can use a behaviour policy that generates diverse experiences and then employ backups according to the Bellman Optimality operator T^* to achieve convergence to Q^*. This algorithm was introduced in (Watkins and Dayan, 1992) and is commonly known under the name of *Q-learning*. This has become one of the most popular control algorithms in RL and we provide a full outline of this algorithm in Algorithm 2.5 below.

[5] An ε-greedy policy is the policy that follows the greedy policy with probability $(1-\varepsilon)$ and with probability ε selects a random action from $\mathcal{A}$.

Algorithm 2.5 Q-Learning (Off-Policy TD Control)

```
Initialize $Q(s,a) = Q_0(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$
Consider $b$ a behaviour policy
for all episodes do
    Start at $s_0 \in \mu_0$
    for all steps $t = 0, T$ of the episode do

        Take action $a_t \sim b(.|s_t)$
        Observe reward $r_{t+1}$ and next state $s_{t+1}$

        Update (One-step update based on Optimality Eq.)
        $Q(s,a) \leftarrow Q(s_t,a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)]$
    end for
end for
```

Under the assumption that every state-ation pair is visited infinitely often, it can be shown that the expectation of the TD error, $\delta_t := [R_t + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)]$ converges to 0. When the TD error approaches 0, this is equivalent to saying that $(T^* Q(s,a) - Q(s,a)) \to 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$, which implies that the obtained Q-estimate satisfies the *Bellman Optimality Equation*. Since this operator has an unique fixed point Q^*, the resulting function that obtains a zero TD-error across the state-action space, recovers the optimal action-value Q^*. Further details on the conditions and requirements of the behaviour policy, as well as a detailed proof of convergence can be found in (Szepesvári, 2010).

2.1.7 Model-Free RL with Function Approximation

So far, all the algorithms presented here assumed a table-based representation of the value functions – that is, we assumed we can enumerate($\mathcal{S}$ is finite) and treat each state(or state-action pair) separately when computing the value function. Nevertheless, this is not always feasible and generalisation is required in problems with very large state-action spaces. Naturally, we would want to extend our RL algorithms to deal with these situations. To do this, we can use a real function to approximate (as closely as possible) our target value functions, instead of an exhausted enumeration of the values for V (or Q). For every state s:

$$V(s) \approx f_\theta(s), \forall s \in \mathcal{S}, \quad f_\theta \in \mathcal{F} \text{ (some parametrisation family)} \tag{2.31}$$

As with most problems in statistical learning, we will choose an appropriate function class $\mathcal{F}$, parametrised by a set of parameters θ and we will try to learn from samples which of the functions in $\mathcal{F}$ most closely captures the value functions of interest. Such an example of a widely-used and well-studied parametrisation class are linear approximations. Thus one can consider:

$$V(s) \approx \theta^T \varphi(s) \qquad (\text{or } Q(s,a) \approx \theta^T \varphi(s,a)) \tag{2.32}$$

where $\varphi(s)$ is a feature space and θ is a real-value vector of parameters that weights the features in $\varphi(.)$. It is important to note that in general, this is just an approximation – in the best-case scenario, we will recover the function $f \in \mathcal{F}$, in our approximation class, that is closest to our real target V^{π} or V^{*}. Only if our true target belongs to the approximation family $\mathcal{F}$, we can converge to the true value function, otherwise, we hope the approximation is close enough to still provide us with a close-to-optimal behaviour.

Approximate Policy Iteration (API) methods

In the following we are going to discuss *Approximate Policy Iteration* (API) methods. As the name suggested, these methods will mimic a PI procedure, under approximation. As a reminder, these are algorithms that alternate between a policy evaluation step and a policy improvement step. Given a current policy π, the evaluation step aims to compute (or approximate) its corresponding value function Q^{π}. In most cases, this step will make use of the Bellman expectation equation (Eq. 2.8-2.9). In the approximate case, at each iteration, we aim to find the best approximation to one application of the Bellman expectation operator $T^{\pi}f$ (Theorem 2.3). Other methods will use the fact the fixed point of this operator is unique and thus will try to find directly the function f in the hypothesis class $\mathcal{F}$ that most closely satisfied its fixed point equation (Definition 2.1). In both of these cases, we end up with an approximate version of the policy evaluation step. After this step, as in the exact case (Algorithm 2.3), we can do a greedy improvement step and then iterate the alternation of these two steps. The convergence to the true value function Q^{π} has been investigated under different functional approximations and different learning criteria. Approximative PI was shown to be particularly well-behaved under a least-square projection criterion — see (Bertsekas et al., 1995; Bradtke and Barto, 1996; Bertsekas and Tsitsiklis, 1995; Lagoudakis et al., 2001), overcoming even unknown divergent cases of Fitted Q Iterations (Lagoudakis and Parr, 2003). For a general survey on error bounds for approximative policy iteration please consult (Munos, 2003; Bertsekas, 2011; Scherrer, 2014).

Fitted Q-Iteration (FQI)

Fitted Q-Iteration(FQI) (Algorithm 2.6) is a general method of extending Q-learning (Algorithm 2.5) to the approximate case. This method was first introduced in (Ernst et al., 2005; Riedmiller, 2005) and can, in principle, be used with any functional class approximating the action-value functions. Popular choices include decision trees (Ernst et al., 2005; Castelletti et al., 2012), linear approximations (Antos et al., 2007, 2008; Farahmand et al., 2009) or (deep) neural networks (Riedmiller, 2005; Gabel et al., 2011; Mnih et al., 2013, 2015); although convergence properties and training behaviour highly depend on this choice. It is worth noting that, in general, FQI is not guaranteed to converge to the best approximator in our approximation class and moreover this best approximator might not give rise to the optimal policy we are looking for. Nevertheless, in practice, it has been shown that we can achieve convergence or at least a hovering behaviour around the optimal value function and in many cases, this does indeed lead to a close-to-optimal policy.

Algorithm 2.6 Fitted Q-Iteration

```
Require: $\mathtt{D} \sim \pi$ - set of experiences
  Initialization: $Q \leftarrow Q_0$ (can be arbitrary)
  for $k = 1 : K$ do
    (Compute targets): using the Bellman Equation(s): $Y^{(k)} \approx T^*(Q^{(k)})$
    for all $(s_t, a_t, r_{t+1}, s_{t+1}) \in D^{(k)}$ do
      $y_t^{(k)} = r_{t+1} + \gamma \max_a Q^{(k)}(s_{t+1}, a)$ ( i.e. $y_t^{(k)} \approx Q(s_t, a_t)$ )
    end for
    (Q-Fitting)[6]: generate $Q^{(k+1)} = Regress(D^{(k)}, Y^{(k)})$
  end for
```

Recently, (Mnih et al., 2015) introduced Deep Q-Network (DQN), which uses a neural network to implement an off-policy non-linear function approximation of the optimal value function. The study of RL algorithms that use neural networks as function approximators has since then been dubbed deep reinforcement learning. Deep RL removes the requirement of a good handcrafted feature representation by requiring instead an effective network architecture and learning algorithm. Mnih et al. (2015) for example, when introducing DQN used a neural network composed of three hidden convolutional layers (LeCun et al., 1995, 2010) followed by a fully-connected hidden layer. These deep learning-based representations (Bengio et al., 2017) have been shown to be extremely effective, allowing us to scale up to more complex tasks and scenarios where coming up with good features is not trivial (Mnih et al., 2015; Silver et al., 2016, 2017). The network parameters are updated through gradient-descent with the following update rule:

$$\theta' \leftarrow \theta + \alpha \left[R_{t+1} + \gamma \max_{a} Q(S_{t+1}, a|\theta^{-}) - Q(S_t, A_t|\theta) \right] \nabla_{\theta} Q(S_t, A_t|\theta) \qquad (2.33)$$

where θ^{-} denote the parameters of a target network, which is a duplicate network from a past iteration that will be updated at a particular frequency, less often than the online network, to improve stability (Mnih et al., 2015). An additional insight that aids the training is the use of an experience replay buffer (Lin, 1993; Schaul et al., 2015b) to improve the sample efficiency and decorrelate observations. Since its introduction, DQN has inspired a great collection of follow-up work combining RL methods and deep neural networks (Van Hasselt et al., 2016; Horgan et al., 2018; Hessel et al., 2018; Kapturowski et al., 2019).

In this work we will investigate various instances of these algorithms, under a linear function class, given a feature space $\varphi(\mathcal{S})$ in Chapter 3 (FQI) and respectively Chapter 4 (API). And in Chapters 5-6, we will employ an $n-$step version of FQI (Peng and Williams, 1994) to learn optimal value functions represented now by a much more flexible and powerful class of approximators, deep neural networks (LeCun et al., 2015; Bengio et al., 2017) when dealing with complex observational spaces like pixels, as in DQN.

2.2 Transfer Learning

Traditional paradigms in machine learning, such as supervised learning and semi-supervised learning, assume data distributions at training and testing time remain the same. In contrast, in transfer learning, we are concerned with capturing the change, the shift in the data distribution, in the domain or task at hand. A core assumption of this paradigm, even outside RL, is that something changes in the environment, but at the same time, something remains the same: there is a connection between the previous tasks and the current task, otherwise, there is no hope for generalisation. And the aspiration here is that this relationship can be effectively exploited to aid the learning process in this new variation.

Empirical evidence suggests that even relatively small changes in the data distribution can negatively impact the predictive performance of classifiers (Shimodaira, 2000), especially in the case of neural network based architectures that are more prone to overfitting to the training data distribution due to over-parametrisation. There have been many studies showing just how fragile machine learning systems can be, when presented with data just outside the training manifold (Goodfellow et al., 2014; Gu and Rigazio, 2014; Nguyen et al., 2015). In reinforcement learning, the problem is aggravated by the fact that we are effectively using these predictions to do (long-term) planning in the environment and in this case even small errors can, over time, be amplified and policies can collapse under very small variations in the input space, as noted in (Rusu et al., 2016; Huang et al., 2017). We would like to avoid learning from scratch every time there is a change in the task conditions or specification. We would like our representations and knowledge to be robust to these shifts in the data distribution and our policy to have the ability to adapt to new scenarios.

What we are essentially saying is that i) we desire a stronger generalisation property beyond the training data distribution to a broader distribution over related tasks/domains, and ii) the ability to rapidly adapt to variations of tasks sampled from this broader distribution. In other words, we would like to avoid re-learning the commonalities between these tasks when trying to master multiple of them. Moreover, we would like to use the knowledge already acquired to boost our learning for a new instance of a task. This is partially motivated by the fact that people can intelligently reuse and apply previously accumulated knowledge to solve new problems faster or find better solutions. We would like our agents to display the same characteristics and flexibly adapt to new demands in their environment without completely re-learning/re-programming to account for these changes.

The field of transfer learning is quite broad as it aims to cover various ways in which the environment and/or the task might change. In this section, we will look at some of the variations considered in the literature and for the rest of this work, we would focus on one particular paradigm outlined in the introduction (Definition 1.1) that we believe to be the most natural in studying the development of an RL agent.

When we think about transfer learning, there are three major research questions that come up and one would need answering when developing a transfer learning algorithm. The answers to these might be dependent on the applications we are targeting, the assumptions about the domains and the variations allowed (sharing the same input space, different features, shared output space ...etc), as well as the properties of the true underlying distribution over the tasks we would like to generalise over. Regardless, the thought process and the resulting algorithm should aim to answer the following:

- **What to transfer?**

  This question targets the process of identifying the subset of the knowledge acquired, that is useful when the change in task occurs. In a sense, this is trying to separate the parts that are tasks specific and the ones that are more general and are invariant under changes observed, such as shifts in data-distribution. This is not a trivial process as it is rooted in our assumptions of how the tasks/domains may vary and 'forcing' transfer outside those assumptions may hurt the learning process, which leads us to the next question.

- **When to transfer?**

  This question asks in which scenarios transfer can be done and likewise, what are situations in which transfer is not desirable – when the change is too big and the tasks become essentially unrelated. If we are considering a series of supervised learning tasks – say sentiment analysis, cat pictures classification, cancer detection, language translation, stock market prediction – this becomes a relevant question. Which tasks are more similar to the one at hand and what we can use to map knowledge from one domain to the other. A potential answer here might be the similarity in the input domain: text (translation, sentiment analysis), pixel-based classification (cats, cancer detection).

  For an RL agent, we will usually assume some consistency in the environment it operates in, the sensory information it gets, the range of movements it can exert onto its environment. Thus in principle, most tasks will be related as they all ought to reveal information about the environment and our agents' interaction with it.

Nevertheless, *negative transfer* can still happen depending on *how* we choose to incorporate previous knowledge into the new task or *how* we constrain the learning problem on the new task to focus resources based on previous interactions. In this case, we would like to identify the tasks that are indeed most relevant to the target tasks and use information coming primarily from those.

- **How to transfer?**

  This refers to exactly how algorithmically we incorporate previous acquired samples, solutions, abstract distilled knowledge into the learning process of a related task. Although there are various ways of approaching this question and numerous instantiations of these in the literature, we remark four prominent approaches outlined as such in (Pan and Yang, 2009):

  - *Instance Transfer.* This is probably the most intuitive way of transferring information and relies on the assumption that, although the data distribution might change between tasks, there is at least a part of the training data that can be re-used for the current task.
  - *Transfer via Feature Representation.* The feature-representation transfer approach aims at finding "good" feature representations to minimize domain divergence and classification or regression model error. This is similar to *common feature learning* in the field of multi-task learning (Argyriou et al., 2007).
  - *Transfer via Parameter Sharing.* Most parameter-transfer approaches assume that individual models for related tasks should share some parameters or a prior distribution of hyper-parameters. Examples of this paradigm include regularization frameworks, parametrised shared representations and hierarchical Bayesian frameworks.
  - *Relational Knowledge Transfer.* Different from other three contexts, the relational-knowledge- transfer approach deals with transfer learning problems in relational domains, where the data are non-i.i.d. and can be represented by multiple relations, such as networked data or graph relations. This approach assumes data drawn from each domain not to be independent and tries to transfer the relationship among data from a source domain to a target domain.

In the following we would take a closer look at these ways of transferring information, but first we will introduce a formal definition of the transfer framework and some of the desired objectives – aiming to measure benefit over learning each task in isolation.

2.2.1 Definitions and Taxonomy

In this section we will introduce a formal definition of *transfer learning*. Firstly we will do this in the context of supervised learning, mainly as the literature is more abundant in this setting and the terminology is much more unified, with several surveys offering a good overview of the field (Weiss et al., 2016; Pan and Yang, 2009). Then, we will extend this formalism to reinforcement learning. Although throughout the years there have been some significant contributions in applying transfer to RL situations – a good overview of the these can be found in (Taylor and Stone, 2009; Lazaric, 2012) – the literature is still significantly sparser when compared to the supervised scenarios, with transfer being recognised as crucial problem and one of the next frontiers in machine learning and RL much more so only in the recent years.

Let us define a domain $\mathcal{D} = (\mathcal{X}, P(X))$ as a tuple of an input space $\mathcal{X}$ and a marginal probability distribution over this space $P(X)$. For a given domain, we define a task $\mathcal{T}$ as a tuple made up of the output space (or label space for classification problems) $\mathcal{Y}$ and a predictive function $f : \mathcal{X} \rightarrow \mathcal{Y}$, which in a supervised setting is to be learnt by pairs $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. We can define transfer learning as:

Definition 2.6: Transfer Learning

Given a source domain $\mathcal{D}_S = (\mathcal{X}_S, P_S(X))$ and a corresponding task $\mathcal{T}_S = (\mathcal{Y}_S, f_S)$, and a target domain $\mathcal{D}_T = (\mathcal{X}_T, P_T(X))$ with a desired target task $\mathcal{T}_T = (\mathcal{Y}_T, f_T)$, transfer learning is the process of improving the target predictive function f_T by using the information from $\mathcal{D}_S, \mathcal{T}_S$.

In the full generality of this framework the source and target domains, as well as the tasks can all be different – $\mathcal{D}_S \neq \mathcal{D}_T$, $\mathcal{T}_S \neq \mathcal{T}_T$ – but some of the most interesting situations arise when some of these components are shared. And the more they share, the more information is potentially available for transfer.

For instance, one of the most natural scenarios encountered in the literature is to assume that the input space stays the same $\mathcal{X}_S = \mathcal{X}_T$ [7], but the marginal distribution over

[7] The case where $\mathcal{X}_S = \mathcal{X}_T$ is referred to as homogeneous transfer learning, and when the input

this shared input space changes, i.e. $P_S(X) \neq P_T(X)$. This is reflective of most systems that evolve smoothly over time and it is one of the most relevant paradigm for practical applications: where the data used for training is collected somewhat in the past and then the trained classifier will be deployed at a later time into the future when the process might have changed. An example of this would be recommender systems that have to account for the ever-changing product/market conditions.

Another scenario that lends itself nicely to transfer is keeping the domains the same, but having different tasks $\mathcal{T}_S \neq \mathcal{T}_T$ – and thus the prediction functions are different. This might happen when the output spaces are different. For instance, consider as input a dataset of pictures and based on this dataset we can formulate multiple tasks: classification of objects (the output here is a probability distribution vector over the possible classes), a caption describing what is happening in the picture (the output here is natural language/text), a segmentation of the image into background and foreground (the output space there is pixel space). Although these tasks might be quite different in their predictions, it has been shown that learning jointly to predict multiple things about the same instance can improve the performance on all tasks (Kokkinos, 2016; Zhang et al., 2017).

2.2.2 Objectives and Desiderata

Previously we have argued that we can use knowledge accumulated from previous tasks to aid the learning of a new task. In this section, we aim to be more concrete about what benefits one could expect to gain from transfer learning and formalise some measurable objectives associated with these potential gains.

We will follow and adapt the formalism used in (Pan and Yang, 2009; Weiss et al., 2016) (supervised learning) and (Lazaric, 2012) (reinforcement learning). These transfer objectives and potential gains equally apply to both RL settings and more traditional supervised learning settings. Although we will argue that RL scenarios pose additional challenges, but also maybe more natural opportunities for transfer. Following (Lazaric, 2012) we will discuss the three major potential improvements a transfer learning system offers over learning from scratch:

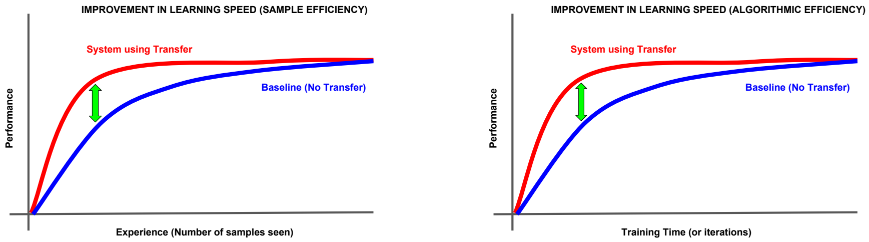
- ***Improvement in learning speed on the new task.*** Depiction of learning curves are displayed in Figures 2.2.1a, 2.2.1b. This criteria can be split into two different ob-

domain changes we refer to this as heterogeneous transfer (Weiss et al., 2016) jectives that could result in a speedup in learning:

  - *Reduction in sample complexity* (Figure 2.2.1a). This objective says that by employing transfer we could get off the ground more quickly (with fewer samples) and achieve a certain performance without seeing a vast amount of experience in the new setting. This is particularly relevant when data in the target domain/task is limited or hard to obtain, but we have related (training) tasks where data is more readily available. Sample complexity is something many RL algorithms struggle with, especially under functional approximation. Thus this is an objective we would want to improve on. Moreover, in RL the learning problem is compounded by the exploration problem. This can make the problem more challenging, but in terms of transfer, this can also be an additional opportunity. An RL system can benefit from transfer both the learning process (in the value function/policy approximation), but also to inform a better exploration in a new task.
  - *Speed-up in the training time of our algorithms.* (Figure 2.2.1b). This can happen if one could make use of training tasks to inform a more effective search in the space of solutions for the new task. Or alternatively, reduce the hypothesis space of candidate functions to be considered for the new task. It is important to note that although this might be a very effective way for speeding up learning, we are usually in a situation where we cannot recover the target function and the accuracy of our approximation is strictly dependent on the structure of this hypothesis space. Thus restricting this space too much, can lead to poor asymptotic performance and negative transfer.

  Note that one criteria does not exclude the other and a speedup in learning can come from making progress on either or both of these fronts.

- ***Jumpstart - a boost in the initial performance.*** (Figure 2.2.2a). This objective assesses the initial, often zero-shot, performance on a target task. Without transfer, the learning process usually starts with a random initialization of the parameters. If our hypothesis space is generous enough to allow good approximations, this starting point might be arbitrary bad and delay convergence. On the other hand, a good initialization has been shown to greatly help speed up convergence in Value Iteration/Policy Iteration based methods. Thus one can use the training tasks to build



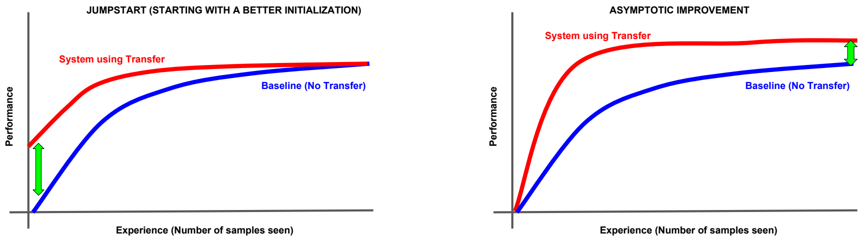
(a) Sample complexity reduction **(b)** Speed-up in the training time

Figure 2.2.1: Transfer learning objective 1: producing a speed up in the learning process for a new task.

an informative prior to better initialize the learning process for the target task. This is again a particularly important objective in an RL context, as a better initial performance can have a huge impact on the quality of data we get during exploration, which in turn has the potential of helping the learning problem itself.

A known potential caveat here is that initialisation, even one with good initial performance, can sometimes have detrimental effects on the learning process, both in terms of speed of learning and even asymptotic performance. For instance, initialising with the policy of another agent that was trained on a task that partially overlaps with the target task, could quickly converge to a suboptimal performance by exploiting only the common part of the two solutions. In order to converge to an optimal performance two things must happen: i) the approximation can escape this local (sub-optimal) solution, ii) exploration strategy will venture outside the state-space around this suboptimal path and will produce informative enough data to allow recovery of the optimal solution. Note that these two points are very distinctive in nature, one has to do with the way we are learning the target function (value/policy for RL) and the optimisation surface in the chosen hypothesis space and the second one is a bias introduced by this initial solution in the way we explore the space.

Therefore, even if we are making progress on this objective, it is good to keep in mind that this might hurt us in the long run, especially as the learning and exploration processes are generally interlinked and will be biased by this initialisation. Nevertheless, there might be situations where long-term optimal performance is



(a) Jumpstart - improving the initial performance on a new task. Some type of zero-shot transfer can happen.

(b) Asymptotic performance boost – better generalisation could obtained by incorporating past knowledge

Figure 2.2.2: Further transfer learning potential benefits and objectives

not as important as getting off the ground as quickly as possible and acting coherently in a new domain. For instance, in robotics one cares much more about having an initialisation that has a better initial performance: a policy that can move around and respond to the change in the environment, safely exploring the new domain and avoid cases where it might break the robot, even at the cost of asymptotic performance. Learn from scratch, in this scenario might lead to a better solution in the long run, but the time, effort and hardware to get there might far outweigh the asymptotic performance gap. At the end of the day, it is a question of which objective is more important to optimise so that we can properly safeguard against possible negative effects of this initialisation.

- *Asymptotic performance improvement.* (Figure 2.2.2b) This says that the final performance on the target task could be improved by transfer and we can measure this gap at the end of the training process. The final performance is one we would ideally want to optimise for – in supervised and reinforcement learning scenarios alike. But, as highlighted previously, there might be a trade-off between this objective and other improvements we can get via transfer – like initial performance (jumpstart).

  However, we would argue that in most practical applications we care more about the sample complexity, and we will, realistically, never be in the asymptotic scenario with respect to the experience seen, especially when operating in infinite state spaces. Thus this criterion might be problematic to measure fairly, as intrinsically this measure of performance disregards the time or number of samples required to reach convergence; two algorithms may eventually reach the same performance, but do so in very different ways: time-wise and sample complexity-wise. Moreover, the time to convergence may be prohibitively long and in practice, we are back to effectively comparing performances on finite sample sizes and finite wall-clock. Thus although this is a crucial objective for our transfer algorithms, setting up a 'fair' comparison might be problematic.

2.2.3 Methods of Transferring Knowledge in RL

In this section, we will re-visit the last question we posed above: *How to transfer?* and we will focus primarily on methods of transferring knowledge in reinforcement learning settings. Moreover we will focus our attention to the particular case where the source and target MDP-s share the same state and action space, as well as the dynamics – $\mathcal{M}_S = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}_S, \gamma)$, $\mathcal{M}_T = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}_T, \gamma)$. Thus the difference in the tasks is incorporate in the reward. Note that this is the scenario proposed in the introduction as the persistent environment assumption.

2.2.3.1 Instance Transfer

Considering that our tasks share the same state and action space, a natural way of transferring information is using samples from the source task(s) to improve performance on the target task. Note that most RL algorithm would require full transitions (s_t, a_t, r_t, s_{t+1}) which are not necessarily transferable unless we assume some common structure in the reward specification as well. Note that this is reasonable assumption around the reward/task distribution – for instance, most of the states would not give you any reward, or just a time penalty or some states will always give you negative reward. In this case, the question becomes when to transfer or what are the samples that are 'safe' to transfer. If the samples being transferred from the source tasks differ a lot from the target task, negative transfer can occur (Lazaric, 2012).

Even without making any assumptions about exactly the reward functions, we can sometimes reuse samples in building individual parts of our training algorithm. For instance, if we adopt a model-based approach and seek to estimate the transition model in order to plan, this estimation can be done using samples from various (training) tasks and could readily be used in the target domain. More generally, the core idea would be to identify, and if possible, separate in the learning that reward and the state-transition information –

the invariance and the task specific part. In this way we can reuse samples safely. Examples of such architectures include successor features (Barreto et al., 2017) and the multitask least-squares policy iteration method proposed in Section. 4.3 of this work.

2.2.3.2 Representation Transfer

The idea here is to use the source task(s) to infer and extract general characteristics that are likely to be present in the target task as well. The aim of the transfer algorithm here is to use these inferred characteristics to change the representation of the solution space to speed up the learning by reusing relevant parts of the policies previously inferred. At a more abstract level, it is the same idea of trying to figure out what is shared, what remains invariant throughout the source task(s) and then leverage this information to more rapidly make progress on the target task. We single out two major ways of achieving representation transfer in RL and our work will mostly revolve around these:

- **Feature transfer**.

  This transfer paradigm aims to build a set of features $\{\varphi_i\}_{i=1}^n$ that have been deemed important for the source task(s) and (re-)define the hypothesis space of the solutions considered for the target task, based on these features.

  There are multiple instances in the literature and it is the focus of the next chapters (Chapters 3-4). An example of this can be seen in (Mahadevan and Maggioni, 2007) that introduces a way of building proto-value functions using the Laplacian of the estimated graph of the source MDP. Generalisations of this work relax the constraint of the dynamics of the source and target MDP and they now can slightly vary and extend this framework to the case when the source and target MDP can have different dynamics and different reward.

  More recently, progressive networks were introduced in (Rusu et al., 2016) and these allow features learnt in previous tasks to be used if necessary for the target task. The power of this architecture lies in essentially constructing a very large set of features, from different levels of abstractions that might be helpful. On top of these, the target task can still learn from the raw input and develop a new representation that can estimate the target function, even if the source tasks are not particularly relevant or related. This results in a safe way of doing transfer, where we essentially safeguard against negative transfer if the source tasks are not a good indication of what to expect in the target task. The potential problem here is that the number of features coming from previous tasks might be overwhelmingly large, by constructions, thus the learning problem might become harder as now the learner has to distil what is important from the abundance of features coming from previous tasks and complement them with the learning of its specialized features. As the number of tasks grows, this problem is aggravated and the learning in the new task sometimes gives up on re-using the available features from the previous tasks and will essentially default to learning its own representation and use that for its approximation. This is somewhat counter-intuitive in terms of learning, as we would want our performance and transfer abilities to improve as the number of tasks we experienced increases.

- **Option or subpolicy transfer**

  Another way of transferring information in an RL setting is at the policy level. The assumption here is that the source task and the target task can be decomposed into sub-problems that at least partially can overlap. In this case, these sub-policies learnt on the training tasks can be readily used to make progress in the target task. The idea here is to exploit as much as possible the shared dynamics between the source and target domains and identify in what part of the state space the source policy is still sensible/valid and which states actually contain information about the tasks and reward. This typically allows for planning at a lower time resolution and enable the reuse to learn skills to navigate between this crucial state where (higher-level) decisions have to be made. Although most transfer algorithms in this section share this outlined idea and structure, the critical point is how to identify the subgoals and learn the options (Sutton et al., 1999) from the estimated dynamics. Inferring these sub-policies or, mostly known in the RL literature, as options (Sutton et al., 1999; Perkins et al., 1999) or subskills (Konidaris and Barto, 2009; Da Silva et al., 2012) is a notoriously hard problem. Nevertheless, options are very natural way of thinking about transfer and can potentially have major benefits as it provides a temporal abstraction that can speed up the learning and planning process as the effective time horizon for which we are optimising is decreasing substantially. Another benefit of these temporal abstractions would be using them for more informative, targeted exploration (using the previously inferred subgoals).

2.2.3.3 Parameter Sharing

In the previous methods of transferring, evolving features or options, we will assume that the training and test tasks are similar enough to allow for positive transfer and we usually try to safeguard against negative transfer – this is done typically but augmenting the hypothesis space between the common features or options, thus making sure that a (non-transfer based) solution can be recovered at least theoretically. In these methods we assume similarity, but will typically never *explicitly* model, nor measure it. In contrast, parameter-transfer algorithms do try to explicitly model the underlying distribution from which the source and target tasks are assumed to be drawn. These methods will typically parametrise the MDPs, thus $\mathcal{M}_S = \mathcal{M}_{\theta_S}$ and $\mathcal{M}_T = \mathcal{M}_{\theta_T}$, and seek to model the distribution over these parameters $\theta \sim p(.|\omega)$ and infer the hyperparameters ω. The assumption here is that the tasks seen, and consequently the target task are drawn i.i.d. from this underlying generative process. Methods here include a variety of hierarchical and Bayesian RL methods (Wilson et al., 2007; Cao and Ray, 2012). The major advance of these methods is the distillation of the knowledge gathered from previous task into the prior or the parameters the generative process of the task experience, conditioned on θ is assumed invariant and can be learnt using all task available during training.

Part I

Transferable Representation Learning

3
Representation Learning in Multitask Reinforcement Learning

In this chapter, we are going to consider a multitask Reinforcement Learning (MT-RL) scenario and explore sharing information between the multiple tasks by learning a shared representation across the value functions of interest. As the name suggests, in MT-RL we reason about scenarios in which an agent needs to learn to perform a set of different tasks. As such, our goal here will be to design algorithms that are capable of learning good policies across many of these tasks, leveraging commonalities whenever possible. As prefaced in the introduction, in this work we restrict our attention to tasks that take place in the same *persistent environment*, as described in Definition 1.1.

In this formulation, each task, indexed by j, formally gives rise to a different MDP, $\mathcal{M}_j = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r_j, \gamma_j \rangle$, that share the same state space, $\mathcal{S}$, and action space, $\mathcal{A}$, and dynamics. And the reward signal r_j can be an arbitrary functions $r_j : \mathcal{S} \times \mathcal{A} \rightarrow [R_{min}, R_{max}], \forall j$. Mastering a task j in this environment is equivalent to finding an optimal policy, π_j, for its induced MDP, $\mathcal{M}_j$. This is one of the most popular choices considered in multitask RL, although there are many studies that allow for changes in the transition model - for instance (Ravindran and Barto, 2003) and (Ferguson and Mahadevan, 2006). Note that the methods developed in this chapter would be applicable to this extended setting as well, although we leave this as a subject for future investigations.

Out of the many various attempts of phrasing the multitask learning problem in reinforcement learning and formalising the relativeness between tasks, we distinguish mainly two prominent ideas in the literature:

- *Tasks are drawn from some (un)known distribution.* This is the most common scenario treated in the literature and the one we will operate under at this stage. An early work by (Tanaka and Yamamura, 2003) introduced that idea of maintaining a sampling distribution over the tasks and implicitly over possible MDPs, an agent would need to solve in its lifetime. They then used very basic statistics, like mean and standard deviation, to better initialise the agent's value tables for the following task. Unfortunately, the modelling of the common structure in the tasks is very limited and transfer of the summary statistics of the value functions are not always the most informative for a different task - for instance, if the task distributions are multi-modal.

- *Tasks are constructed and presented to the agent in a progressive fashion.* Madden and Howley (2004) introduced a way of transferring experience between progressively more difficult environments and tasks. They propose mastering partial or simpler tasks at first, then within an introspection phase, the agent rationalises the experience gained and formulates a rule-based policy that can readily be used in a more complex situation. This kind of task learning seems to be more intuitive and aligned with the way humans acquire knowledge and expertise, but the applicability of the proposed method is quite limited. Nevertheless, there has been some recent work along the lines of formalising the idea of lifelong learning and curriculum learning (Thrun and Mitchell, 1995; Eaton and Ruvolo, 2013; Ruvolo and Eaton, 2013) to the RL framework (Brunskill and Li, 2014; Tessler et al., 2017).

The methods we will propose and investigate in this part of the thesis, could be readily applicable to either these scenarios when considering tasks within the same persistent environment. In other words, whether or not the reward functions are drawn from some known distribution or potentially increasing in their complexity (in terms of the tasks specified by these reward signals), the methods proposed in the next two chapters do not rely on any of these assumptions; although we expect empirically that the performance and the benefits of transfer to be dependent on the nature of the underlying problem. In this first part, we will be looking at methods that perform transfer *within a set of tasks*, rather than transferring knowledge to a new task in order to speed up learning. Thus, since we are not testing the generalisation outside the training set (in terms of tasks), the nature of the underlying generative process, giving rise to these tasks, is less important. In this first part, we are looking to methods to would improve generalisation across a set of given tasks. Typically this is usually the case when we have data already collected for a set of tasks and would want to use the communality of the environment and state-action space to improve learning across these tasks.

3.1 Shared Representation Learning for MDPs

As a brief reminder, in RL an agent interacts with an environment and selects actions to maximise the expected amount of cumulative reward received $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$, where $R_t = r(S_t, A_t)$ (Sutton and Barto, 1998). We model this scenario using the formalism of *Markov Decision Processes* and as discussed in Section 2.1.5, a principled way to address the RL problem is to use methods derived from *dynamic programming* (DP) (Puterman, 1994). These usually compute the *action-value function* of a policy π, defined as:

$$Q^{\pi}(s,a) \equiv \mathrm{E}_{\pi}\left[G_t \,|\, S_t = s, A_t = a\right], \tag{3.1}$$

where $\mathrm{E}_{\pi}[\cdot]$ denotes expectation over the sequences of transitions induced by π.

The computation of $Q^{\pi}(s,a)$ is called *policy evaluation*. Once a policy π has been evaluated, we can compute a *greedy* policy $\pi'(s) \in \text{argmax}_a Q^{\pi}(s,a)$ that is guaranteed to perform at least as well as π, that is: $Q^{\pi'}(s,a) \geq Q^{\pi}(s,a)$ for any $(s,a) \in \mathcal{S} \times \mathcal{A}$.

In this work we are interested in the problem of *transfer* in RL (Taylor and Stone, 2009; Lazaric, 2012). Specifically, we ask the question: given a set of MDPs that only differ in their reward function, how can we *leverage knowledge gained by interacting with all these MDPs to speed up the learning or improve the quality of the resulting solutions*?

We are going to assume that our agent has access to a high-dimensional, highly structured observational space (for instance, pixels) which also doubles as the state of the agent, S_t and that was generated by a much lower dimensional variable. While this is generally the case with all high-dimensional input, we do not claim nor explicitly target recovering this true structure that gave rise to our agent's observations. Instead we relax the above, by requiring that there exists a low-dimensional Y_t that can compactly capture all the relevant information needed for our tasks – these will be defined later. That is, we require that $\exists f_e : \mathcal{S} \rightarrow \mathbb{R}^d$, s.t. $\forall S_t \in \mathcal{S}$:

$$f_e : \mathcal{S} \rightarrow \mathcal{Y}, \text{s.t. } Y_t = f_e(S_t) \qquad \text{(Low-dim embbeding of the input } S_t\text{)}$$

While this trivially holds for the "true" low-dimensional structure underlying our observations, it is not crucial for us that the reconstruction mapping exists. Thus the projection f_e can be lossy, as long as it supports the evaluation and control problems we are considering. This also means any information, or artefacts present in the observation space can be eliminated if they do not improve our ability to approximate the value functions of interest.

3.1.1 Generalisation over Tasks

We want to achieve good generalisation and transfer of knowledge between tasks. To do so, in this chapter, we hypothesise we can achieve this by declaring the representation Y_t (or an abstraction on the joint (Y_t, A_t) space), to be invariant across tasks. In other words, we will train a representation of the environment that is powerful enough to accommodate learning good approximations, for all of the tasks under consideration.

In the following, we show how DP-based paradigms of learning optimal policies can be extended to multi-task RL and in particular to the case where we are interested in learning a common abstraction of the state-action space to be shared across all value-functions of interest, for all tasks $j \in \overline{1, J}$. A graphical depiction of the set of dynamical systems under this constraint is available in Figure 3.1.1. We will be essentially learning a shared representation for the action-value functional space. Moreover, we will see that the emerging structure is quite specialised and encodes the shared structure among expert policies. Maybe unsurprisingly in hindsight, this turns out to be a very efficient way of representing the state-action space for the computation of optimal value functions.

3.1.2 Multi-Task Fitted Q-Iteration

In this section, we outline a general framework of using *approximate value iteration* to compute the optimal Q-values (and optimal policies) for a set of tasks, in a given environment following the MT-RL setup previously introduced. The proposed algorithm is an exten-

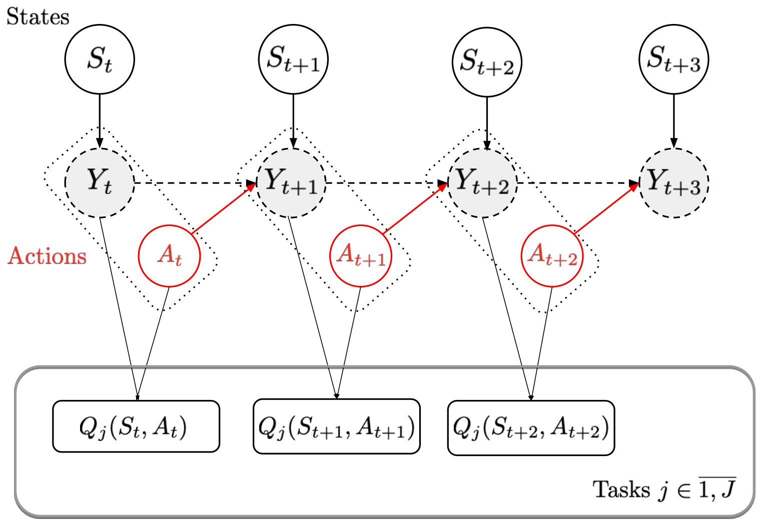


Figure 3.1.1: Enforcing a shared representation of the state-action space, to be used in modelling all action-value functions, Q_j, across a set of tasks $j = 1, J$.

sion of *Fitted Q-Iteration* (Algorithm 2.6) that allows for joint learning and transfer across tasks. Following the recipe of FQI, at each step in the iteration loop and for each sample in our experience set $\mathtt{D} = \{(s, a, r, s')|s' \sim \mathcal{P}(.|s, a)\}$, we compute the one-step TD target based on our current estimate of the value function. This is can be done online or from a recorded set of experiences. Then, treating these estimates as ground truth, we obtain a regression problem from the state-action space onto the TD targets; in the simplest case T^*Q. In the case of MT-RL, we obtain such a regression problem for each task j. Now we could, in principle, solve all these regression problems independently for each task, which would amount to applying FQI individually for each task. But our assumption here is that there is shared structure between tasks and we would like to make use of this common ground to aid the learning process. Thus we propose that the resulting regression problems are to be solved jointly, accounting for such a common representation. A detailed description of the proposed procedure can be found outlined in Algorithm 3.1.

Note that, in the spirit of generality, we do not specify a particular algorithm for the multi-task learning step (MTL in Algorithm 3.1). This is on purpose, as there exists an extensive literature of how to deal with the resulting multi-task regression problem and

Algorithm 3.1 Multi-task Fitted Q-Iteration

```
Require: $\mathtt{D} = \{\mathtt{D}_j\}_{j=1}^{J} \sim \mu, \mathcal{P}$ - set of experiences/episodes for each task $j$

  Initialize $\Theta = \Theta_0, k = 0$ (parameters of our considered approximation)

  while convergence not reached $(d\Theta < \varepsilon$ or $k < \mathtt{MaxIter})$ do
    Compute Targets:
    $\mathcal{Y}_j^{(k+1)} = \{y_j^{(k+1)}(s,a) = r_j(s,a) + \gamma \max_{a'} Q^{(k)}(s',a') | (s,a,s') \in \mathtt{D}_j\}, \forall j = 1, J$
    Multi-task Learning:
    $\Theta^{(k+1)} = \mathrm{MTL}\left(\mathtt{D} = \{\mathtt{D}_j\}_{j=1}^{J}, \mathcal{Y}^{(k)} = \{\mathcal{Y}_j^{(k)}\}_{j=1}^{J}\right)$
    $d\Theta = \|\Theta^{(k+1)} - \Theta^{(k)}\|, k = k + 1$
  end while

  return $\Theta = \{\theta_j\}_{j=1}^{J}$ $(\Rightarrow Q_j(s,a) = f_{\theta_j}(s,a), \forall s,a \in \mathcal{S} \times \mathcal{A})$
```

exploit shared structure between tasks in purely supervised settings. In this work, we will take a look at only one particular instance of this step to serve as proof-of-concept.

3.2 Learning Linear Representations for Value Functions

In the following sections, we will restrict our attention to the class of linear functional approximations of the value functions. Thus we assume there exists a representation $\Phi(s,a)$ such that for any policy π the corresponding value function can be well approximated by a inner product of this representation with some parameter vector.

$$Q^{\pi}(s,a) = \Phi(s,a)^T w^{\pi} \tag{3.3}$$

Now let us consider a class of reward functions: $\{r_j | j = \overline{1,J}\}$. These reward functions will induce, a set of optimal Q-functions one for each task j, $\{Q_j^*\}_{\overline{1,J}}$ that specify the optimal behaviour for that task. Our objective is to infer a small set of features $\Phi(s,a)$, that are powerful enough to approximate any of these optimal value-function $Q_j^*, \forall j = \overline{1,J}$. Unfortunately, we do not know the optimal value function in advance and thus will have to iteratively build up those values and a good feature representation that can model the intermediate and optimal values. This leads to the following set of equations:

$$\begin{cases} \Phi(s,a)^T w_1 & \approx T^* Q_1(s,a) \\ \Phi(s,a)^T w_2 & \approx T^* Q_2(s,a) \\ & \cdots \\ \Phi(s,a)^T w_J & \approx T^* Q_J(s,a) \end{cases} \tag{3.4}$$

$$\Leftrightarrow \begin{bmatrix} - & \Phi(s,a) & - \end{bmatrix} \begin{bmatrix} | & | & \cdots & | \\ w_1 & w_2 & \cdots & w_J \\ | & | & \cdots & | \end{bmatrix} \approx \begin{bmatrix} T^* Q_1 & T^* Q_2 & \cdots & T^* Q_J \end{bmatrix}$$

And in particular for the optimal value functions, the above implies that there exist a set of weights $\{w_j^*\}_{j=1,J}$ that can capture $\begin{bmatrix} Q_1^* & Q_2^* & \cdots & Q_J^* \end{bmatrix} \approx \begin{bmatrix} T^* Q_1^* & T^* Q_2^* & \cdots & T^* Q_J^* \end{bmatrix}$.

Without further constraints solving the above problem is equivalent to performing inference over the optimal value function for each task individually, without any exchange of information between tasks. Instead, we would like to use data from all tasks to inform and improve all individual learning problems by exploiting relationships between tasks.

In the following section, we will look at a particular method of performing the multi-task learning step in MT-FQI when considering linear approximations of the value functions. We specify a feature space $\Phi(s,a)$ that can be used to express all value functions, for all tasks, and then using MT-FQI we can learn the corresponding weight vectors. Although the feature space considered should be able to fit any value function, we will show that it only needs to be able to fit good/optimal strategies — and this is precisely the representation we want to capture. In general, we try to start with a very general representation of the input space and then progressively specialise and compress this representation to fit efficiently optimal policies across tasks. In the following, we present a method of evolving this representation that have been proposed and have been studied extensively in the multi-task learning literature for supervised tasks.

3.2.1 Learning Shared Linear Representations via a Convex Reformulation

In terms of estimating the action-value functions, the joint problem we are trying to solve can be formalised as inferring

$$\{w_j\}_{j=1}^{J} = \operatorname*{argmin}_{W} \left[\sum_j \mathcal{L}\left(w_j^T \Phi(s,a), y_j \right) + \mathcal{H}(W) \right] \tag{3.5}$$

where $y_j = r_j(s,a) + \gamma \max_b Q(s', b)$ and $\mathcal{H}(W)$ is a regulariser on the weight vectors, that encourages feature sharing. At the same time, we wish to learn a more compact abstraction of the state-action space, that will be shared among tasks. To make this a bit more formal, let $Q_{j,w} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, $Q_{j,w}(s,a) := \langle \Phi(s,a), w_j \rangle$. Then our assumption can be expressed as follows: $\exists$ a small set of features $\Psi = \{\psi_i\}_{i=1,N_\psi}$ obtained as linear combinations of the original set of features Φ – note this is just a change of basis, through a matrix U. Thus we obtain a reparametrisation of our value functions, where for every task j its corresponding value function can be re-written as a linear combination in the low-dim representation $\Psi = U\Phi$:

$$Q_j(s,a) \approx \sum_i \alpha_{ji} \psi_i(s,a) = \langle \alpha_j, U^T \Phi(s,a) \rangle.$$

Since we want the above representation to be sparse, for each task j, one could try to solve an optimisation problem of the following type:

$$Q_j \quad = \quad \operatorname*{argmin}_{f_j = \langle \alpha_j, U^T \Phi \rangle} \left[\sum_j \mathcal{L}_{\mathrm{D}_j}\left(f_j(s,a), y_j \right) + ||\alpha_j||_1^2 \right] \tag{3.6}$$

Nevertheless, it has been proposed that we can look at such problems jointly, under a similar sparsity regulariser that *also promotes features sharing among tasks*. This leads to a joint optimisation problem of the form:

$$\begin{aligned} (U^*, A^*) = & \qquad \operatorname*{argmin}_{A,U} \left[\varepsilon(U,A) + \lambda \mathcal{H}(A) \right] \qquad \text{(Problem 3.1)} \\ = & \quad \operatorname*{argmin}_{A,U} \left[\sum_j \mathcal{L}_{\mathrm{D}_j}\left(\langle \alpha_j, U^T \Phi(s,a) \rangle, y_j \right) + \lambda \mathcal{H}(A) \right] \end{aligned}$$

where $A = [\alpha_1, \cdots, \alpha_j, \cdots, \alpha_J]$ and $\mathcal{H}(A) = \lambda ||A||_{2,1}^2$. The above equation can be broken up into two parts: the empirical error in fitting $T^* Q_j$ and the regulariser $\mathcal{H}(A)$. There are detailed below:

$$\varepsilon(U, A) = \sum_j \frac{1}{m} \sum_{l=1}^{L} \mathcal{L}(\underbrace{r_j(s^{(l)}, a^{(l)}) + \gamma \max_a Q_j(s'^{(l)}, a)}_{y_j^{(l)}}, \langle \underbrace{\varphi(s^{(l)}, a^{(l)})}_{x_j^{(l)}}, \underbrace{[UA_j]}_{w_j^\pi} \rangle) \tag{Empirical error}$$

$$= \sum_j \frac{1}{m} \sum_{l=1}^{L} \mathcal{L}(\underbrace{r_j(s^{(l)}, a^{(l)}) + \gamma \max_a Q_j(s'^{(l)}, a)}_{y_j^{(l)}}, \langle \underbrace{U^T \varphi(s^{(l)}, a^{(l)})}_{\psi(s^{(l)}, a^{(l)})}, A_j \rangle)$$

where we used subscript (l) to index the samples from our datasets $(s^{(l)}, a^{(l)}, r^{(l)}), s'^{(l)}) \in \mathtt{D}_j$, and

$$\mathcal{H}(A) = \|A\|_{2,1}^2 = \sum_{k=1}^{d} \sqrt{\sum_j A_{k,j}^2} = \sum_{k=1}^{d} ||A_k||_2 \text{ where } A_{k,j} \text{ is the element in } A \text{ at position } (k, j). \tag{Regulariser}$$

It was shown in (Argyriou et al., 2008) that Problem 3.1 is equivalent to solving the following *convex* problem:

$$(W^*, D^*) = \arg\inf \left[\varepsilon(W) + \lambda Tr(W^T D^{-1} W) \right] \text{ s.t. } D \succ 0, Tr(D) \leq 1 \tag{Problem 3.2}$$

where we denote by $\varepsilon(W)$ the joint empirical loss in Eq. 3.5 and by D^{-1} the inverse of D.

More precisely if (U^*, A^*) is a solution for Problem 3.1 then $W^* = U^* A^*$ is an optimal solution for Problem 3.2 (Theorem 1 in Argyriou et al. (2008)). Moreover, given a fixed W, the optimal D^* is given by:

$$D^*(W) = \frac{(WW^T)^{\frac{1}{2}}}{Tr(WW^T)^{\frac{1}{2}}} \tag{3.8}$$

as discussed in Appendix A.2 and (Argyriou et al., 2008).

Now keeping D fixed, we get another convex problem that we can now solve for W. In doing so, note that the above amounts to a minimisation over w_j that can be *carried out* *independently for each task j as both objectives factorise over tasks.*

$$W^* = \arg \inf_{W=[w_{1:T}]} \left[\sum_j \varepsilon(w_j) + \lambda \sum_j \langle w_j, D^{-1} w_j \rangle \right] \qquad \text{(Problem 3.2a)}$$

This leads to a very intuitive alternating minimisation algorithm proposed in (Argyriou et al., 2008), that we outline in Algorithm 3.2. Note that we prefer the non-perturbed version of this alternating algorithm as we empirically observed convergence even under these conditions. More details about the algorithm can be found in Appendix A.2 and original works (Argyriou et al., 2007, 2008). Moreover, keep in mind this optimisation is to be performed at every iteration k of the MT-FQI algorithm. Thus partial, rapid solutions are preferred. In the description of the algorithm below, we assume the input of this sub-procedure to be a regression-like dataset, which contains state-action pairs from $\mathtt{D}$ and 1-step sampled targets $T^* Q_j^{(k-1)}(s, a)$ based on our current estimates $Q_j^{(k-1)} = \Phi w_j^{(k-1)}$.

Algorithm 3.2 Joint MT feature learning for MT-FQI

```
Require: The dataset corresponding to the joint regression problem at iteration $k$:
    $\{\mathtt{D}_j^{(k)}, Y^{(k)} | j = 1, J\}$ for each task $j$ as define in Alg. 3.1.
  Initialisation: $D = \frac{1}{d} Id$
  while representation has not converged do
    for $j = 1 : J$ do
      Solve for each vector $w_j$ under a sparse regularisation weighted by $D$:
        $w_j = \operatorname{argmin}_w \left[ \mathcal{L}_{\mathtt{D}_j^{(k)}} \left( \langle w, \varphi(s_j^{(l)}, a_j^{(l)}) \rangle, y_j^{(l)} \right) + \lambda \langle w, D^+ w \rangle \right]$
    end for
    Compute new relation matrix $D$:
        $D = \frac{(WW^T)^{\frac{1}{2}}}{Tr(WW^T)^{\frac{1}{2}}}$
  end while
  return Rotation matrix $U$, $D$ and weight vector matrix $W$.
```

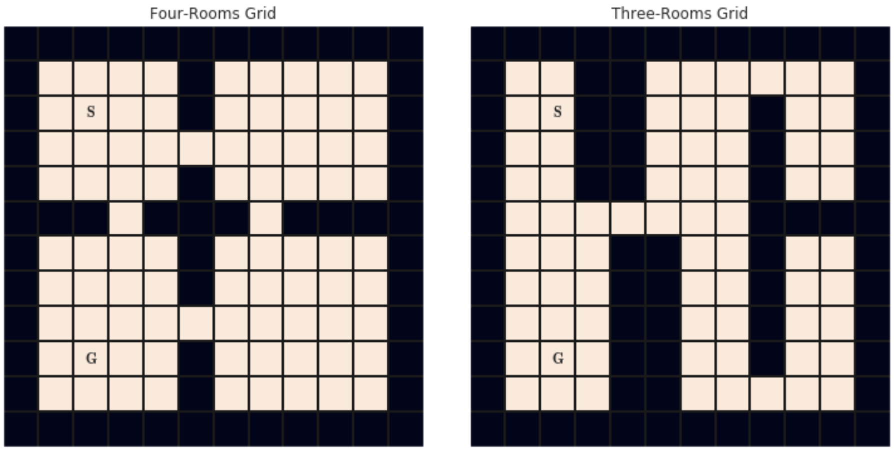
3.3 Experiments

In this section, we present an empirical evaluation of the previously proposed algorithms and try to validate to what extent learning a shared representation over our set of value functions is possible and whether this indeed leads to better generation and increased sample efficiency. Throughout these experiments, unless specified otherwise, we will use a batch set of experiences $\mathtt{D} = \cup_{j=1}^{J} \mathtt{D}_j$, created under a uniformly random behaviour policy $\mu : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, with $\mu(a|s) = \frac{1}{|\mathcal{A}|}, \forall a \in \mathcal{A}, s \in \mathcal{S}$, where $|\mathcal{A}|$ denotes the cardinality of our action set $\mathcal{A}$. For each task j, we have a fixed budget of $n_{episodes}$ episodes with a maximum length of 25 steps. These will be trajectories generated from a random start location while following the behaviour policy μ. As we are going to be looking at fairly restrictive budgets when generating the datasets for each task j, we make sure that $\mathtt{D}_j$ contains at least one transition to the goal location G_j, otherwise, the agent would have no information where the location of the goal is. This data will be generated upfront, at the beginning of the experiment and all algorithms presented will use the same dataset $\mathtt{D}$. Thus comparisons between the algorithms will be purely based on how well each of them manages to utilise this predefined dataset $\mathtt{D}$.

3.3.1 Task Description: Multi-goal Navigation

In this experiment, we consider two gridworld navigation tasks: a variation of the 4-Rooms problem proposed in (Sutton et al., 1999) and a 3-Rooms world, that breaks some of the potential symmetries present in the first scenario. The environments are depicted in Figure 3.3.1.

The agent can move deterministically in all the cardinal directions $\mathcal{A} = \{up(\uparrow), down(\downarrow), left(\leftarrow), right(\rightarrow)\}$ throughout the empty space (orange squares in the Figure 3.3.1). Bouncing into walls (black squares) has no effect on the position of the agent. In this maze-like environment, we consider several navigation tasks to different goal locations in the maze. We consider a sparse reward signal, where the agent gets a positive reward ($+10$) when it reaches the active goal location and 0 (no signal) otherwise. Thus in our setting, the goal location will fully specify the task. To generate multiple tasks, we simply generated different goal locations $\{G_j\}_{j=1,J}$, sampled uniformly at random from the achievable positions in the grid. Once a goal is reached, on a particular task, the episode ends and the agent re-spawns in a new random initial position. Note that this final transition will be

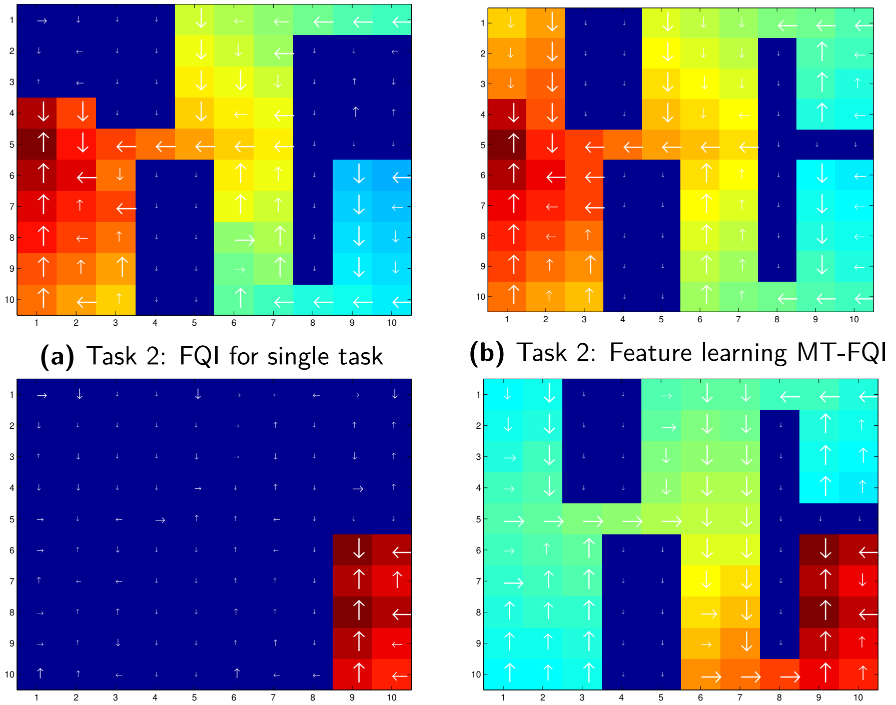


(a) 4-Rooms Grid **(b)** 3-Rooms Grid

Figure 3.3.1: Environments considered for our navigation task. Starting states **S** are chosen at random and the goal states **G** describe a particular task j.

particular to each task and thus the considered MDP will share all transitions, apart from this one. Thus, technically for this terminal state we are violating the shared dynamics assumption. At the same time, this is a terminating transition, which is modelled by $\gamma = 0$, as we are going to be bootstrapping on the immediate (terminal) reward only. Thus in this case the future transitions do not matter for the learning problem, as we are not bootstrapping beyond termination. It follows that, in the learning problem we are going to be using transitions (s, a, s') that always share the probability of transitioning to s', from s, after executing action a.

The observation space of the agent is a one-hot embedding of the agents' grid location. No other information about the environment is provided: in particular, note that the walls and goals locations are not part of the observation space. Thus the agent needs to discover the environment and reward structure, purely through the samples in $\mathtt{D}$. This makes the inference problem more challenging as FQI will take quite a few iterations to propagate that weak signal throughout the environment into parts of the state space away from the goal. Moreover, under a restricted budget of interactions, the algorithm is exposed to only a small set of transitions that actually lead to a reward in any given task. That is why, under this regime, the single-task FQI might fail to recover the optimal value. In this case, jointly inferring a shared structure of the state-action space might help this generalisation problem



(a) Task 2: FQI for single task

(b) Task 2: Feature learning MT-FQI

(c) Task 17: FQI for single task

(d) Task 17: Feature learning MT-FQI

Figure 3.3.2: Examples of the inferred value functions ($V(s) = \max_a Q(s, a)$) and policies (greedy) achieved by single-task FQI vs MT-FQI learning. One can see we indeed obtain a better generalisation when sharing information and representation across tasks.

and this will be precisely the setup we are considering in our experiments.

To make this a bit more explicit, in Figure 3.3.2 we provide two examples of the value functions and greedy policies obtained for two of the navigation tasks considered in the 3-Rooms Grid. The first row corresponds to value functions inferred for the second task, which achieved a good performance on all training algorithms (Task 2); the second row illustrates a case where single-task learning fails to generalise properly beyond the immediate vicinity of its goal state and this results in a very poor empirical performance (Task 17). For this experiment, we trained an agent on $J = n_{tasks} = 20$ tasks, randomly sampled from the set of all reachable positions in the MDP, and we provide a budget of 50 episodes per task of maximum 25 interactions. Please note that all algorithms have access to the same training sample set and are run for 100 Q-iterations, at which point they all exhibit convergence. Thus, the superior performance of MT-FQI is due to its ability to transfer knowledge across the state-action space by sharing a common subspace with the other tasks.

Nevertheless, this is just one example illustrating qualitatively the difference of behaviour we get under the single-task versus multitask FQI. Next, we are going to look at a larger quantitative study to assess the benefits in performance, one could expect under a multi-task training procedure. In particular, we were interested in assessing how the number of tasks n_{tasks} considered and regularisation parameter λ_D (Eq. 1.1) impact the MT-FQI performance. In order to assess this more broadly, we vary also the sample budget for each task: $n_{episodes} = \{25, 50, 100\}$. Results are summarised in Figure 3.3.3. These were obtained in the 4-Rooms environment and we are showing the (undiscounted) cumulative reward obtained by our agents, averaged over the tasks and over 3 random seeds. These averages were computed in an evaluation phase: for each iteration of learning, for each task j we compute the greedy policy $\tilde{\pi}_j$ on the current estimate of the value function $\tilde{Q}_j$ and evaluate its performance in the environment. We compute this performance by an average Monte Carlo return over 100 evaluation episodes with random starting locations.

Looking at Figure 3.3.3, within one row – that is for a given number of tasks n_{tasks} – for each random seed, we sample the tasks (goal locations) once and we keep this set fixed over the different sample budgets. Thus we get the expected monotonic increase in performance as the sample size increases. Across all variations, we can clearly see that there are always instances, depending on the regularisation parameter λ_D, where MT-FQI substantially improves over its single-task counterpart. Although empirically we have found that we can obtain positive results for a generous range of λ_D-s, there are still values that could hurt the performance and going beyond certain values, the performance degrades considerably, leading to a form of negative transfer. We can see signs of that for $\lambda_D \geq 0.001$.

3.3.2 Learnt Shared Representations

Probably the most interesting phenomenon encountered in learning these shared representations is the nature of the low dimensional representations inferred. We visualise the inferred set of shared features (Figure 3.3.4) and their respective weights in the value-function (Figure 3.3.5). These were produced via MT-FQI, training on a set of $J = 30$ training tasks in our 4-Rooms Grid. We obtain strong activations (significant weights in the a matrix) only for the top 3 features inferred $\psi_{1:3}$. As a reminder, the learnt shared representations $\psi(s, a) = U^T \varphi(s, a)$ aim to represent linearly the value functions for each

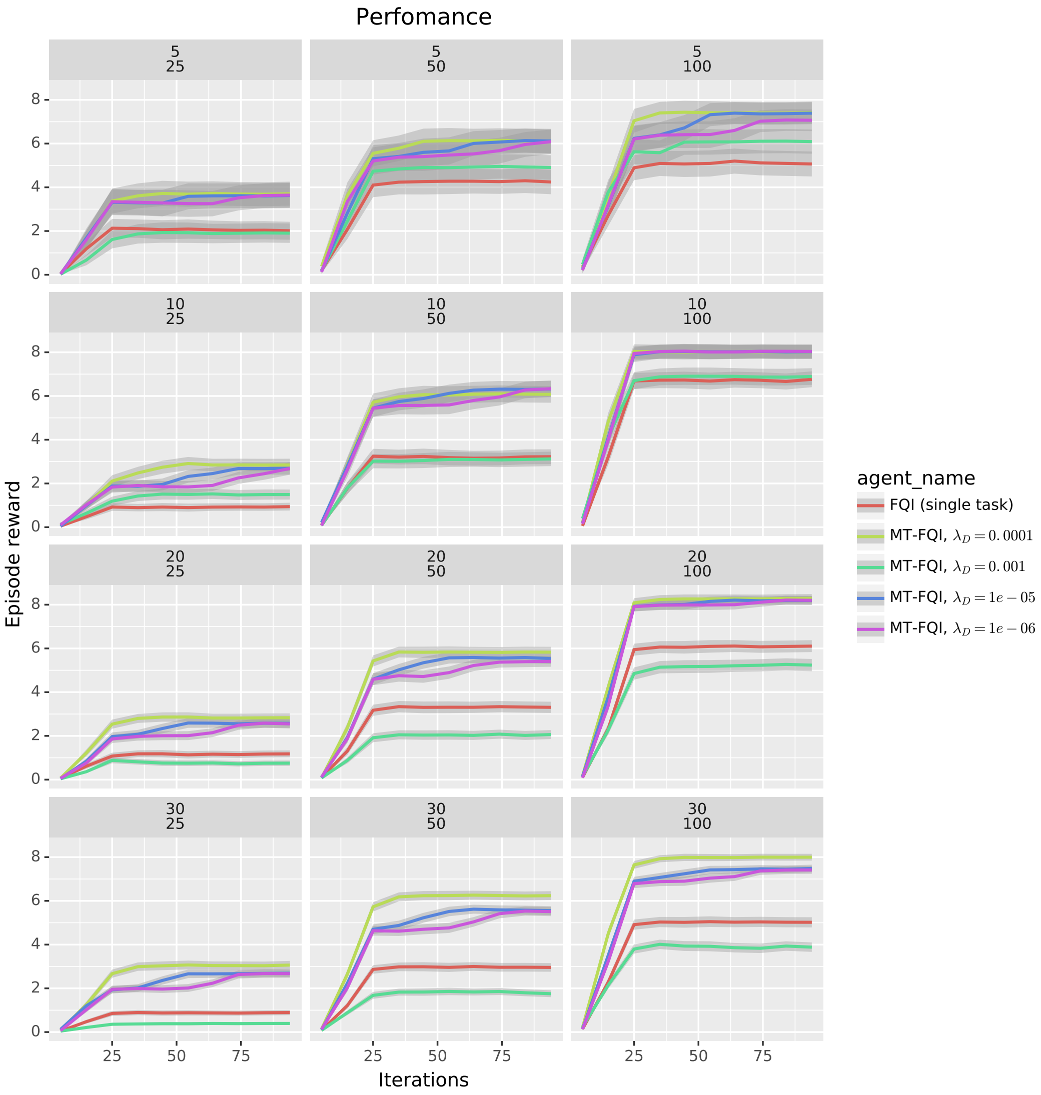


Figure 3.3.3: Performance comparison between a single-task FQI and MT-FQI agents with different λ_D, under different conditions: number of tasks $n_{tasks} \in \{5, 10, 20, 30\}$ (top entry in the title of the cell) and the budget per task, $n_{episodes} \in \{25, 50, 100\}$ (bottom entry in the title of the cell). This illustrate that data from different tasks and be used effectively by Alg. 3.1 to overcome a small budget of per-task iterations. Results were averaged over 3 seeds (data generation). Tasks were samples once for each n_{tasks}.

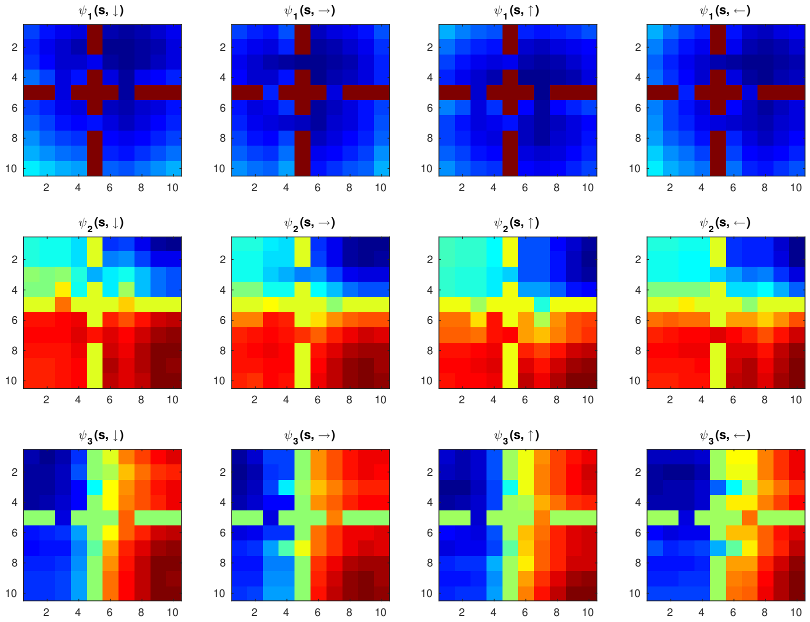


Figure 3.3.4: [To be read **row-wise**] The first three most relevant shared features $\psi_{1:3}(s, a)$ – corresponding to the top three eigenvalues – learnt via MT-FQI under 30 tasks randomly sampled in the 4-Rooms Grid. Please note that these already enable the navigation between any pair of rooms.

task j:

$$\tilde{Q}_j(s, a) = \psi(s, a)^T a_j, \forall j$$

A similar phenomenon happens in the 3-Rooms Grid – check Figures B.1.1-B.1.2 in the Appendix B.1. Thus, it seems that across different situations, the learnt representation seems to *converge to a very low dimensional compression*, that at the same time seems to be expressive enough to effectively approximate the desired optimal value functions.

Moreover, if we take a closer look at Figure 3.3.4, we can see that the learnt representation recovers very intuitive features. As reminder this is the linear representation obtained as $\Psi(s, a) = U\Phi(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}$, where transformation matrix U and corresponding coefficients a_j where obtained by a eigen-decomposition of matrix D defined in Eq. 3.8. The most prominent feature, corresponding to the highest eigenvalue, captures

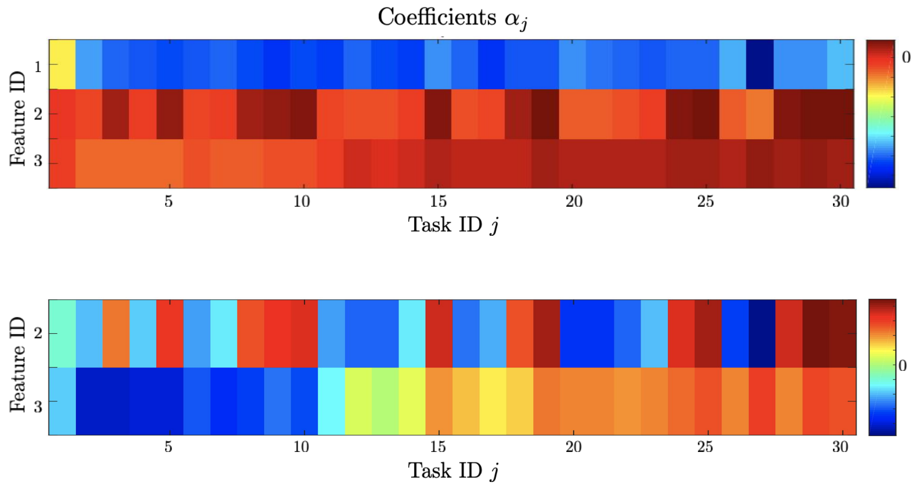


Figure 3.3.5: Weighting Coefficients $\alpha_{1:3,j}$ and $\alpha_{2:3,j}$ for the above three most prominent shared features. We can see from these values that the first feature clearly dominated in all tasks. **Bottom:** Rescaled version of $\alpha_{2:3,j}$ such that we can see the activation of the other two prominent features. Blue corresponds to negative activation and red to positive ones. Given the nature of the features one can readily read out, just by looking at the sign of the weight, which room that task's goal state is. For instance, if we look at second task: negative activation for both $\psi_2 \Rightarrow$ north-side of the environment and $\psi_3 \Rightarrow$ west-side of the environment. The goal G_2 is indeed located at position $(2, 1)$ in the top-left room.

the layout of the environment, while the second and third features seems to encode the navigation between different parts of the state space. For instance, the second feature ψ_2 has a high value in the lower part of the state space, supporting value functions that need to encode transitions to the south rooms. At the same time, note that if the weight $\alpha_{2,j}$ for this feature is negative, this feature also encodes the opposite navigation pattern to the north part of the grid. A similar thing can be said for the third feature ψ_3 which encodes navigation from the left to right and vice-versa depending on the sign of the weight associated with it. We can see that just by looking at these features and the sign of their corresponding weights, Figure 3.3.5, one can immediately read off the room in which the goal is located, for a particular task. For instance, for the second task, both activation are negative, thus we can infer that the room where the value function is highest would the NW room; likewise for the last task, both activations are positive, thus its goal is situated in the SE room. Similar patterns have been observed in the other scenarios (Mahadevan and Maggioni, 2007; Konidaris et al., 2011; Machado et al., 2017a). What this analysis is showing, maybe not surprisingly, is that indeed the optimal value function space for MDPs that shared the same state-action space and transition dynamics has a lot of common structure. Moreover, what is more encouraging is that this structure can be learnt as part of a multitask learning procedure, as we have done here and can promote positive transfer between the tasks and lead to better generalisation.

3.3.3 Connection to Options

As discussed previously, the learnt shared representation seems to account for the general topology and dynamics of the environment in the value functions. They nicely partition the environment into relevant regions to facilitate the global navigation to a local neighbourhood of the goal. Some of these characteristics are reminiscent of the options (Sutton et al., 1999), subskills (Konidaris and Barto, 2007) or macro-actions literature (Dietterich, 2000) and have the potential to drastically improve the efficiency and scalability of RL methods (Barto and Mahadevan, 2003; Hengst, 2002). And much like our shared representation, one of the appealing qualities of options is that they could be shared and re-used across different tasks. Thus, we would like to investigate this connection further.

Following the formulation in (Sutton et al., 1999), an option $o = \langle \mathcal{I}, \mu_o, \beta \rangle$, is a generalisation of primitive actions $a \in \mathcal{A}$ to a temporally extended course of action. $\mathcal{I}$ is the initiation set $\mathcal{I} \subseteq \mathcal{S}$ from which the option is available, μ is the policy we are going to follow once the options is triggered and $\beta : \mathcal{S} \rightarrow [0, 1]$ is the probability of termination. In this case, the action-value function takes the form:

$$Q_j^{\pi_j}(s, o) = \mathbb{E}\left[\underbrace{\sum_{k=0}^{K} \gamma^k r_{j,k}}_{r_j(s,o)} + \gamma^{K+1} \sum_{s'} P(s'|s, o) V_j^{\pi_j}(s')\right]$$

where s' is in the termination state of options o. We denote $P^o_{ss'} = \sum_k^\infty p_o(s', k)\gamma^k$, where $p_o(s', k)$ is the probability that options o will terminate in state s' after exactly k steps. Note that this term does not depend on the reward function and only accounts for the transition dynamics, the policy of the option and its termination criteria – all of which are task-invariant. Moreover note that for us, $r_t(s, o)$ is generally 0 unless the option happens to hit the goal. Thus the above equation, simplifies to:

$$Q_j^{\pi_j}(s,o) = \sum_{s'} \left[\underbrace{P^o_{ss'}}_{\text{task independent}} \cdot \underbrace{V^{\pi_j}(s')}_{\text{task dependent}(j,\pi_j)} \right]$$

This is a linear combination between the option transition models $P^o_{ss'} = \varphi^{\mu_o}(s,s')$ in the termination set (subgoals of the option) – which is independent of task j and π_j, it only depends on μ_o – weighted by the value function of the termination states $V^{\pi_j}(s')$, for each task – which incorporates the dependency on task and the individual policy employed after the option has terminated. This is very similar to the parametrisation we assumed in Eq. 3.4. This also suggests that the learnt representation is able to capture and represent efficiently some option-like transition models without us specifying any subgoals, sub-policies nor initial states. We hypothesise that *the learnt shared space is actually a compressed basis for these option-transition models*. In order to test this hypothesis, we consider an intuitive set of options $\mathcal{O}$ (like navigation to a particular room) and test if this learnt basis can span $P^o_{ss'}$ for some option $o \in \mathcal{O}$ and can successfully represent an option-policy.

We define an option o_1 to be navigating to a specific room, say room 1 (NW). The initialisation set $\mathcal{I}_{o_1}$ is the set of states outside the room and termination set contains all states in the desired room. We also can define an MDP that maintains the same transition dynamics, state and action space, but now the reward signal is zero outside the target room and a constant positive reward in any of the desired termination states s' in room 1. Note that the value function corresponding to this newly defined semi-MDP (Sutton et al., 1999) is given by: $Q^\pi(s,o) = \sum_{s'} P^o_{ss'}$, as $V^\pi(s') = const$. In this semi-MDP, we run FQI and indeed see that we are able to construct a value function, based solely on the learnt 5-dim feature space ψ_s, that successfully completes the newly specified task. Results for all such navigation options are available in Figure 3.3.6.

Please note that the abovely defined options are quite extended ones. Simpler ones would include finding a way out of a particular room – these are along the lines of the options defined in (Sutton et al., 1999) and (Stolle and Precup, 2002) – can be easily recovered as well. Actually, for these simpler options, we require very few samples to obtain the desired behaviour (10-30 samples), although they might not be optimal. The fact that we are able to express a whole variety of such intuitively defined options – much more than the dimensionality of the common subspace on which we are building – is a clear indica-

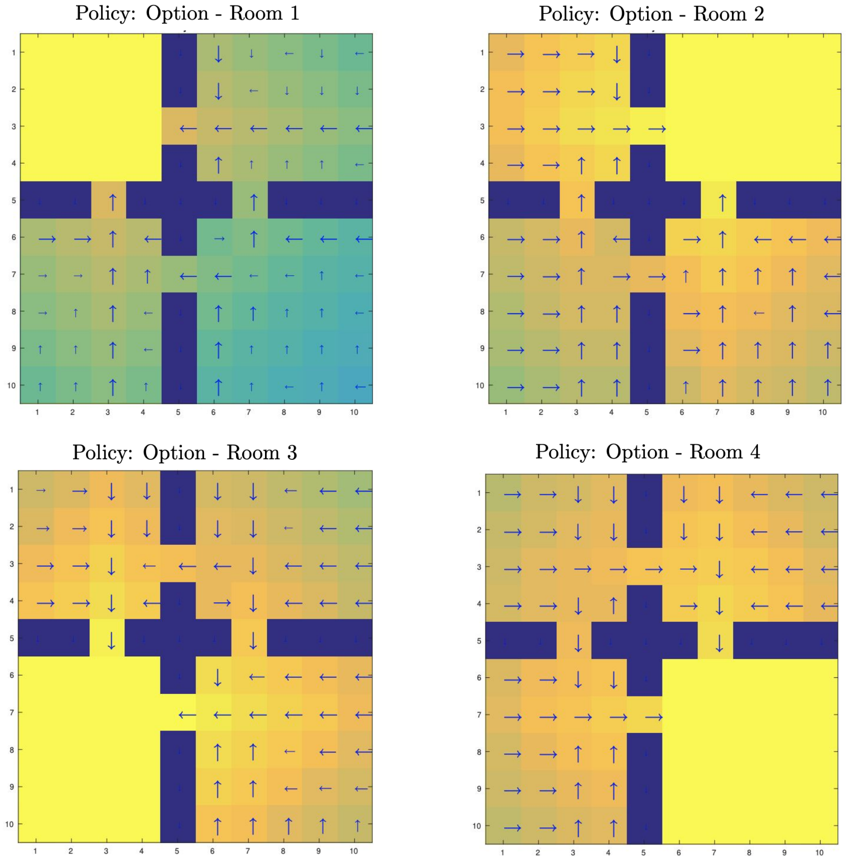


Figure 3.3.6: Learned greedy policies (as indicated by the arrows) and value functions ($V(s) = \max_a(Q(s,a))$) enabling navigation to any of the four rooms, based only on the share feature subspace discovered in the multi-task value function learning of 30 goals randomly sampled in the environment. The value functions were learnt using (single-task) FQI on the top 5 features $\psi(s,a)$ and required $\approx$ 10 episodes to recover option-like policies that enable the agent to reach the desired room.

tion of the expressiveness of this shared representation and its potential transferability in aiding learning of new tasks within the same environment.

3.3.4 Towards On-policy Data Generation

The last study we conducted for this work, is going towards a more on-policy data collection scenario, where after each iteration of Q-fitting, we go back to the environment and act according to an ε-greedy policy (with $\varepsilon = 0.1$) based on our current estimate of the per-task action-value function. Thus after each iteration, we would collect new data and use this new dataset in the next iteration of the algorithm. One could also consider a more additive approach to this data collection process, where the new experience is added to the old one, but that would mean that over time our dataset would increase considerably. Due to this effect, it would be hard to compare the efficiency in learning under a constrained budget. Thus, we opted for having a constant budget of episodes and populating our experience buffer with the most recent experiences only. This is similar to having a replay buffer, where only the new on-policy transitions are stored.

With this modification, we repeat the experiments on our $4-$Rooms environment (Figure 3.3.1), training agent in both a single task setting and under a multitask regime using MT-FQI. We do this varying the number of tasks $n_{task} \in \{5, 10, 20, 30\}$ under a fairly restrictive budget of $n_{epsiodes} = 50$ episodes or maximum 25 steps. The results are summarised in Figure 3.3.7. One thing to notice here is that this new regime of data collection might be quite restrictive for the single-task agent – what we have noticed is that under these conditions, the single-task agent will need on average $300 - 500$ episodes to solve the task. This is mainly because the on-policy data will be quite biased around regions of the policy that the agent currently thinks are good.

Nevertheless, we can see that the multitask agents can do substantially better. This might be in part because they are making better usage of the data, as the previous experiments have shown. The other cause might be credited to the different data generation. Although still on-policy as in the single-task setting, the multitask algorithm looks at different datasets collected under different behaviour policies. Thus even though these policies are fairly limited in their coverage of the state-space, the combination of the stationary distributions induced by all of these policies might still result in diverse coverage of the state-action space. In general, it would hard to distinguish how much of the improvement is due to better learning and how much is due to better data – the latter still possibly

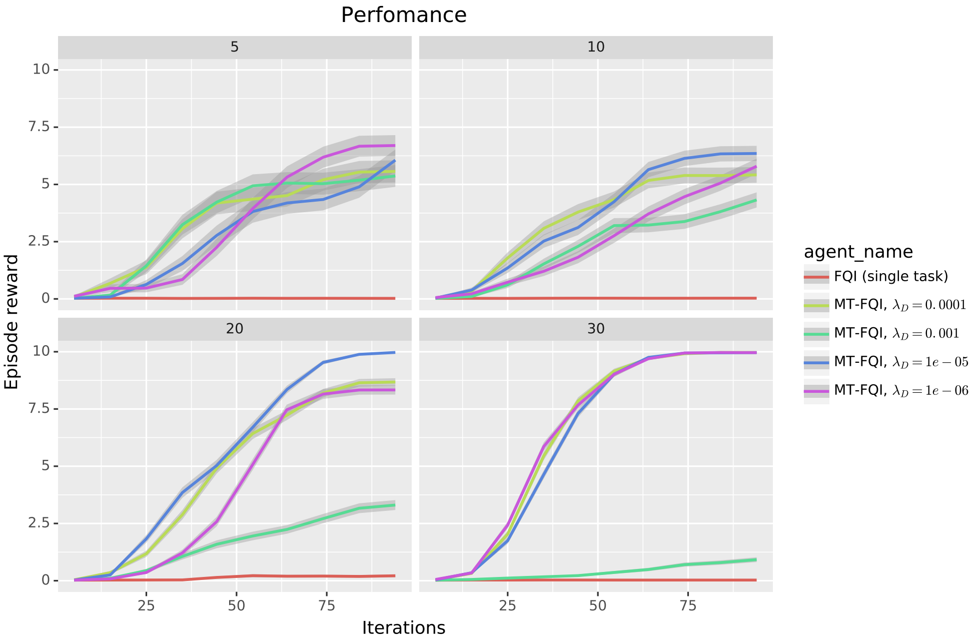


Figure 3.3.7: Empirical analysis of the performance achieved by the FQI single-task vs. MT-FQI agents, on the 4-Rooms Grid, in a setting where the dataset is refreshed on-policy (ε-greedy on the current estimate) at each iteration in the optimisation. The budget is restricted at 50 episodes worth of samples. We can see again that the MT agents manage to achieve a better performance and this effect is now amplified by on-policy collection of that data that now can produce more useful experiences as the agent's competence increases. Results are averaged over 3 random seeds.

powered by better learning in the previous iteration. Nevertheless, this is more of a sanity check that seems to indicate that the insights we have established in the offline batch setting, could have similar benefits and implications in a more online, on-policy setting.

3.4 Conclusion

3.4.1 Related Work

Multitask and transfer learning are one of the most important problems in RL in our ambition to scale up our agent to perform more and more complex tasks. As such, over the years there have been various attempts at modelling this problem – a fairly extensive survey can be found in (Lazaric, 2012; Taylor and Stone, 2009). All in all, there are at least four different strains of multitask reinforcement learning that have been explored in the literature: off-policy learning of many predictions about the same stream of experience (Sutton et al., 2011; Gruslys et al., 2017; Mankowitz et al., 2018; Borsa et al., 2019) most of them building multiple value function predictions, but also other auxiliary tasks as in (Jaderberg et al., 2016; Pathak et al., 2017), continual learning of a sequence of tasks (Thrun, 1996, 2012; Ammar et al., 2014; Ruvolo and Eaton, 2013) that aim to draw on previous knowledge to learn a new task while trying to protect the previously trained policies (Rusu et al., 2016; Kirkpatrick et al., 2017), distillation of task-specific experts into a single shared model (Parisotto et al., 2015; Rusu et al., 2015; Teh et al., 2017; Berseth et al., 2018), and joint learning of multiple tasks at once (Caruana, 1997; Caruana and O'Sullivan, 1998; Sharma and Ravindran, 2017; Du et al., 2018). In this work, we have focused on the latter under a batch mode assumption. Nevertheless, as mentioned previously, the proposed algorithm is applicable under a different, online behaviour policy as long as we still have access to a replay of experience per task, from which one can sample corresponding datasets $\mathtt{D}_j$.

One of the methods investigated in (Calandriello et al., 2014) as part of a study on sparsity in multitask RL, is very closely related to our learning procedure and this work can be seen as a generalisation of that method, although the focus and model assumptions we are making are quite different. Perhaps the most relevant prior work that shares our vision and some of the modelling assumptions is the approach in (Schaul et al., 2015a) which models a shared state representation between goals and assumes a linear factorisation between this state embedding and task or goal embedding. In our work, this can be seen as the a learnt to specify the task in the learnt shared representation.

Some other works investigating joint training of multiple value functions include (Wilson et al., 2007) and (Lazaric and Ghavamzadeh, 2010) which both employ a hierarchical Bayesian model to learn a prior over value functions. More recently (Laroche and Barlier, 2017), a follow-up work on ours, proposed a model exploiting the same shared structure property induced by the shared dynamics across our tasks. A different, but related line of work has been exploring structure in the policy space, rather than values (Dimitrakakis and Rothkopf, 2011). The properties of those representations can be very different as they model commonalities in a different space, but in principle these two could be used together in an actor-critic-like algorithm (Witten, 1977; Barto et al., 1983; Grondman et al., 2012), trying to share representation across both functional approximation classes. Along these line, IMPALA (Espeholt et al., 2018), an importance weighted actor learner, has recently achieved state of the art for multitask RL (on both the Atari suite and across the 30 DeepMind lab levels). Nevertheless, these results are far from the human level performance demonstrated by deep RL agents on the same domains, when trained on each task individually. Building upon this work, later on, (Hessel et al., 2019) manage to improve these results by stabilising the learning across multiple objectives by PopArt (van Hasselt et al., 2016). This is notably one of the only results in literature where training over such a diverse set of tasks is not only possible – previous works have struggled to even obtain a solution that caters to all tasks (Parisotto et al., 2015; Du et al., 2018) – but improves the single-task performance, which was the main motivation behind our work as well as the wider literature in multitask RL (Thrun, 1996).

3.4.2 Summary

In this chapter, we studied the problem of learning multiple optimal value functions in a multitask RL scenario where the class of MDPs we considered for generalisation shares the environment and actuators specification but differ in their reward. In this scenario, we explored the feasibility of learning a joint representation over our value functions and assess the ability to transfer knowledge between tasks by treating the estimation of the value functions as a joint multitask regression problem (see Section 3.1, Algorithm 3.1). We proposed a very general extension to the popular fitted-Q learning iterative algorithm to encompass this joint learning. Next, focusing on linear representations, we studied an instance of this algorithm with a popular multitask learning procedure proposed in(Argyriou et al., 2008). We tested the resulting procedure experimentally in a batch learning scenario with a restrictive sample budget and showed that we can indeed get positive transfer between the tasks through this multitask procedure. We then examined the resulting representation that enabled this transfer. As presented in the experimental section, the learnt representation seems to capture nicely the joint structure in the optimal action-value space and due to their linearity leads to quite interpretable features.

It is worth noting that, in principle, Algorithm 3.1 can be instantiated with any multitask regression solver and depending on the tasks and MDPs, there might be better choices out there, including some that deal with non-linear features. The purpose of this work was mostly to show that formalising the problem in this way and treating the backup steps of the optimal Bellman operators jointly can lead to positive transfer. It is worth mentioning that we did investigate one other multitask learning mechanism here, based on (Ando and Zhang, 2005). The study and results can be found in (Borsa et al., 2016). This was prompted by the observation that in some scenarios tasks can benefit from having a small and sparse set of features that represent the particularities of each individual task on top of a low-dimensional shared subspace. This is definitely the case in many practical applications and had been observed in purely supervised settings as well. It is sometimes simply too restricted to constrain all tasks to be using a single shared structure. Thus researchers have come up with various ways of incorporating task-specific components — see (Zhou et al., 2011; Chen et al., 2012) and reference therein – and showed that modelling these explicitly can improve both the learning (in accuracy and speed) and interpretability of the resulting representations. We showed that these results translated to the RL setting and result in an improved performance.

4

Least-Squared Methods for Multitask Policy Iteration

In this chapter, we are going to explore a series of methods following the general paradigm of Approximate Policy Iteration (API), introduced in Section 2.1.7. In particular, we will take a close look at least-squares approximations of the policy evaluation step in API (Buşoniu et al., 2012). One of the most iconic algorithms in this class is Least Square Policy Iteration (LSPI) introduced in Lagoudakis and Parr (2003), which extends the use of the least-squares temporal-difference(LSTD) learning algorithm proposed in Bradtke and Barto (1996). In this work we are going to briefly re-visit and re-phase those in a more general light that will allow us to extend these to our multitask scenario, taking advantage of the rich common structure present in a persistent environment, shared across our tasks.

4.1 Background

As before, we are interested in a family of MDPs that differ only in their reward specification $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, r)$, where the reward signal r can be an arbitrary function

$r : \mathcal{S} \times \mathcal{A} \to [R_{min}, R_{max}]$. Given multiple of these MDP $\{\mathcal{M}_j\}_{j=1,J}$, our goal is to learn optimal policies $\pi_j^* : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that maximise the value functions, V_j^π (or Q_j^π), associated with these reward signals r_j.

In the single MDP setting, Policy Iteration(PI) (see Section 2.1.5.4 or Sutton and Barto (1998)) represents a particular class of methods that aim to build optimal value functions (and therefore optimal policies) within this MDP. PI methods rely on an iterative procedure that alternates between a policy evaluating step and a policy improvement step. As the name suggests, the policy evaluation step will build the value function associated with the current policy – evaluating 'goodness' of the current policy. Then, the policy improvement step computes a new and improved policy, usually based on the previously built value function. The procedure continues until we can improve the policy no more. When $\mathcal{S}$ and $\mathcal{A}$ are finite and we use exact representation for the value functions – no approximation error incurred – a greedy policy improvement step is guaranteed to produce a strictly better policy than the one considered at the previous iteration. Since in the finite state-action space, there are only a finite number of deterministic policies, PI is guaranteed to converge to the optimal value function (and associated greedy optimal policy) in a finite number of iterations.

Since the improvement step can be addressed greedily, let us focus our attention onto the policy evaluation step instead. This step consists of building the Q^π-function associated with a policy π. In order to do this, we recall that Q^π satisfies the Bellman expectation equation:

$$\begin{aligned} Q^\pi(s,a) &= T^\pi Q^\pi(s,a) && (4.1) \\ &= r(s,a) + \gamma \sum_{s',a'} P(s'|s,a)\pi(a'|s')Q^\pi(s',a') && (4.2) \end{aligned}$$

Based on the above equation, there are a number of algorithms that we can employ to obtain Q^π, such as directly solving the linear system induced by Eq. 4.2 over all states and actions, iterative application of the Bellman operator, temporal-difference methods, etc.

Nevertheless, in most problems of interest, we will not be able to represent the value function of interest exactly. Instead, we will try to approximate our values within a functional approximation class $\mathcal{F}$. And we would settle on finding a good approximation within this class. In doing so, we distinguish two main types of approaches to approximatively solve the Bellman equation above (Eq. 4.2): *projected policy evaluation* methods and *Bellman residual minimisation (BRM)* methods (Buşoniu et al., 2012).

Projected policy evaluation methods will approximately enforce Eq. 4.2 via a projection of the Bellman operator onto the space of approximations $\mathcal{F}$. More specifically given an approximation $\tilde{Q} \in \mathcal{F}$, we apply the Bellman operator T^{π} which would likely result in a function outside our approximation space. Then we project this function back onto the space of representable functions $\mathcal{F}$ and require this projection to be close to the original point $\tilde{Q}$. Thus methods in this class will try to solve the following approximate problem:

$$\tilde{Q}^{\pi} = \underset{\tilde{Q} \in \mathcal{F}}{\operatorname{argmin}} \|\tilde{Q} - \Pi_{\mathcal{F}}(T^{\pi}\tilde{Q})\| \tag{4.3}$$

where $\|.\|$ denotes a norm, or measure of distance in the functional space and the mapping $\Pi_{\mathcal{F}} : \mathcal{Q} \rightarrow \mathcal{F}$ is a projection operator from the space of all Q-functions, $\mathcal{Q}$, onto the approximation space $\mathcal{F}$. Note that this is solving the projected Bellman equation:

$$\tilde{Q} \approx \Pi_{\mathcal{F}}(T^{\pi}\tilde{Q}) \tag{4.4}$$

which gives the name for this class of methods. It is also worth noting that for linear approximation, $\mathcal{F} = \mathcal{F}_{\Phi} = \{Q = \Phi^T w | w \in \mathbb{R}^d\}$, the above equation can be solved exactly, whereas for arbitrary $\mathcal{F}$, this might not be the case. We will revisit this case shortly.

The other class of methods are *Bellman residual minimisation* methods. These do not involve a projection step, instead they simply try to solve directly the Bellman equation (Eq. 4.2) in an approximate sense:

$$\tilde{Q} \approx T^{\pi}\tilde{Q} \tag{4.5}$$

As mentioned before, $T^{\pi}\tilde{Q}$ might not be in the same space as $\tilde{Q}$, thus the above problem will typically not admit an exact solution. Instead, this class of methods will try to find an approximation, $\tilde{Q}^{\pi}$, such that:

$$\tilde{Q}^{\pi} = \underset{\tilde{Q} \in \mathcal{F}}{\operatorname{argmin}} \|\tilde{Q} - (T^{\pi}\tilde{Q})\| \tag{4.6}$$

The difference $\|\tilde{Q} - T^{\pi}\tilde{Q}\|$ is referred to as the Bellman residual indicating the degree to which the approximation violates the Bellman expectation equation. As the name suggests, BRM methods will try to find the approximation that minimises this residual.

4.2 Least-Squares Methods for Policy Evaluation

In this section, we will take a closer look at the policy evaluation step and review some classical takes on using least-square methods to provide (approximate) solutions to this step. We start with a somewhat idealised setting where we consider discrete states and actions: $(s,a) \in \mathcal{S} \times \mathcal{A}$ with $|\mathcal{S}| < \infty, |\mathcal{A}| < \infty$. This is mainly for the sake of clarity in exposition, but we will show that the studied algorithms have online, incremental, sample-based versions that support their applicability to more complex, possibly uncountable and continuous state spaces. The above assumption also enables us to use the matrix form notation associated with the Bellman Equations.

We consider again the linear approximation of the value functions: $Q(s,a) = \varphi(s,a)^T w$ where $w \in \mathbb{R}^d$ is a set of weights and φ are a set of d independent features $\varphi(s,a) = [\varphi_1(s,a), \varphi_2(s,a), \cdots, \varphi_d(s,a)]^T \in \mathbb{R}^d$ for all $s \in \mathcal{S}, a \in \mathcal{A}$. Thus in this case the class of functional approximators considered is $\mathcal{F} = \mathcal{F}_\varphi$ which is the span of the features φ. For convenience, let us denote the matrix of features φ for all state-action pairs as Φ:

$$\Phi = \begin{bmatrix} - & \varphi(s_1, a_1)^T & - \\ & \vdots & \\ - & \varphi(s,a)^T & - \\ & \vdots & \\ - & \varphi(s_{|\mathcal{S}|}, a_{|\mathcal{A}|})^T & - \end{bmatrix} \tag{4.7}$$

For the class of methods reviewed in this section we would consider a (weighted) Euclidean norm. More formally, given a probability distribution over the state-actions space $\rho : \mathcal{S} \times \mathcal{A} \to [0,1]$ the corresponding weighted Euclidean norm is defined as:

$$||f||_\rho^2 = \sum_{s,a} \rho(s,a)|f(s,a)|^2 \tag{4.8}$$

And we denote the corresponding (weighted) least-square projection operator, Π^ρ:

$$\Pi^\rho(f) = \arg\min_{g \in \mathcal{F}} ||g - f||_\rho^2 \tag{4.9}$$

Moreover, for $\mathcal{F} = \mathcal{F}_\varphi$, the above operator has a known close-form solution $\Pi^\rho_{\mathcal{F}_\varphi} = \Phi(\Phi^T\Delta_\rho\Phi)^{-1}\Phi^T\Delta_\rho$, where we define Δ_ρ to be the diagonal matrix with entries $\rho(s,a)$. This property makes linear approximations particularly appealing.

4.2.1 Projected Policy Evaluation

Let us now revisit Eq. 4.3-4.4 under a linear parametrisation of the value functions. Again we would want to approximate the value function corresponding to a given policy π and we will do so by finding the approximation that minimises the distance to the projected Bellman operator. Using the matrix form, one can write:

$$\tilde{Q} \approx \Pi_{\mathcal{F}_\varphi}(T^\pi\tilde{Q}) \Rightarrow \Phi w \approx \Pi_{\mathcal{F}_\varphi}(T^\pi(\Phi w)) \tag{4.10}$$

For linear parametrisations, the above can be solved exactly $\Pi^\rho_{\mathcal{F}_\varphi} = \Phi(\Phi^T\Delta_\rho\Phi)^{-1}\Phi^T\Delta_\rho$:

$$\begin{aligned} \tilde{Q}^\pi &= \Phi(\Phi^T\Delta_\rho\Phi)^{-1}\Phi^T\Delta_\rho\left(T_\pi\tilde{Q}^\pi\right) && (4.11)\\ \Rightarrow \Phi w^\pi &= \Phi(\Phi^T\Delta_\rho\Phi)^{-1}\Phi^T\Delta_\rho\left(R+\gamma P^\pi\Phi w^\pi\right) && (4.12) \end{aligned}$$

which leads to:

$$\underbrace{\underbrace{\Phi^T\Delta_\rho}_{k\times|\mathcal{S}||\mathcal{A}|}\;\underbrace{(\Phi-\gamma P^\pi\Phi)}_{|\mathcal{S}||\mathcal{A}|\times k}}_{k\times k}\underbrace{w^\pi}_{k\times 1} = \underbrace{\Phi^T\Delta_\rho}_{k\times|\mathcal{S}||\mathcal{A}|}\;\underbrace{R}_{|\mathcal{S}||\mathcal{A}|\times 1} \tag{4.13}$$

And finally:

$$w^\pi = \left(\underbrace{\Phi^T\Delta_\rho(\Phi-\gamma P^\pi\Phi)}_{B^\pi}\right)^{-1}\underbrace{\Phi^T\Delta_\rho R}_{b_r} \tag{4.14}$$

In order to compute the above solution, we only need B^π and b_r, which are independent of the original dimensionality of the problem, independent of the size of state and action space, and only depend on d, the dimensionality of the feature set φ. Notice that B^π and $b_r = \Phi^T R$ can be computed from the samples, making this approach extendable beyond finite state and action spaces. Moreover note that b_r is essentially policy independent (thus for each task, we need to compute this vector only once) and B^π is task-independent as its computation involves just the considered policy π and the dynamics of the environment, which is considered shared across tasks. This will become relevant later in the chapter.

4.2.2 Bellman Residual Minimisation

Let us now revisit the Bellman residual minimisation problem Eq. 4.6, under linear parametrisation and a $\mathcal{L}_{2,\rho}$ norm:

$$\min_{\tilde{Q}\in\mathcal{F}_\varphi}||\tilde{Q}-T^\pi\tilde{Q}|| \Rightarrow \min_w||\Phi w - T^\pi\Phi w||_\rho^2 \tag{4.15}$$

Expanding and re-arranging terms, we obtain:

$$\begin{aligned}\min_w||\Phi w-(T^\pi\Phi w)||_\rho^2 &= \min_w||\Phi w-(R+P^\pi\Phi w)||_\rho^2\\ &= \min_w||(\Phi-P^\pi\Phi)w-R||_\rho^2\\ &= \min_w||Z_\pi w-R||_\rho^2 \end{aligned}\tag{4.16}$$

Note that the above is now a least-squares (weighted) linear regression problem. And thus its minimiser has again a closed-form solution:

$$Z_\pi^T\Delta_\rho Z_\pi w = Z_\pi^T\Delta_\rho R \Rightarrow w = \left(Z_\pi^T\Delta_\rho Z_\pi\right)^{-1}Z_\pi^T\Delta_\rho R \tag{4.17}$$

Based on the above equation, one can see that these methods enjoy the same scalability properties as the projected policy ones. In particular, one can easily show that $Z_\pi^T\Delta_\rho Z_\pi \in \mathbb{R}^{d\times d}$ and $Z_\pi^T\Delta_\rho R \in \mathbb{R}^d$ scale in the number of features d, not in the state-action space and can be built from samples (s,a,r,s') drawn from ρ. A small caveat here is that in order to get a consistent sample-estimate for $Z_\pi^T\Delta_\rho Z_\pi$ we might require a double sample for $s' \sim P(.|s,a)$. In the following we will restrict our attention for deterministic environments and as such this will not be a concern for us.

4.3 Multitask - LSPI via Transfer of Samples

In this section, we revisit our previous discussion on Projected Policy Evaluation (Section 4.2.1), now in the context of a multitask RL scenario. In this case, we are interested in learning how to behave in a collection of MDPs that share the same dynamics, but differ in their reward and we adopt a linearly parametrised functional approximation space $\mathcal{F}_\varphi$ induced by a given set of d features $\varphi : \mathcal{S}\times\mathcal{A}\rightarrow\mathbb{R}^d$. This can be seen as the sensory system of our agent and will therefore be persistent over all tasks considered. This means that now we would like to find in the span of φ, an approximation for all value functions

Q^π for the tasks considered and policies encountered at different iterations of PI:

$$\tilde{Q}_j^\pi(s,a) = \varphi(s,a)^T w_j^\pi, \forall s \in \mathcal{S}, a \in \mathcal{A}, \text{ and } \forall j \in \{1, \cdots, J\} \tag{4.18}$$

Revisiting the Projected Expectation equation, for each task $j \in \{1, \cdots, J\}$ we obtain:

$$\tilde{Q}_j^\pi \approx \Pi_{\mathcal{F}_\varphi}(T^\pi \tilde{Q}_j^\pi) \Rightarrow \Phi w_j^\pi \approx \Pi_{\mathcal{F}_\varphi}(T^\pi(\Phi w_j^\pi)) \tag{4.19}$$

And following the argument above, for every task j, be obtained the parameters w_j^π that most closely minimise the distance between the LHS and RHS of the equation above is:

$$w_j^\pi = \left(\Phi^T \Delta_\rho (\Phi - \gamma P^\pi \Phi)\right)^{-1} \Phi^T \Delta_\rho R_j = (B^\pi)^{-1} b_j \tag{4.20}$$

To obtain the matrices B^π and vectors b_j in the equation above, we would ideally need access to the transition function $P(s'|s,a)$ of the MDP and reward functions associated with each task r_j, which are usually unknown in a reinforcement learning context. Fortunately, sample-based version of these algorithms exists, especially when the density ρ under which we can consider the projection in Eq. 4.19 matches the sampling distribution under which our trajectories have been collected in the environment. In this work we are going to consider a batch setting, where we have access only to a fixed set of samples collected under some behaviour policy μ, under each of these tasks j. We denote these datasets generated by interacting with each task j by $\mathtt{D}_j = \{(s, a, r_j, s') | s' \sim P(.|s,a), a \sim \mu, s \sim d^\mu\}$. Revisiting the expressions of B^π and b_j in Eq. 4.20, we get:

$$\begin{aligned} B^\pi &= \Phi^T \Delta_\rho (\Phi - \gamma P \Pi_\pi \Phi) \\ &= \sum_{s,a} \rho(s,a)\varphi(s,a)(\varphi(s,a) - \gamma \mathbb{E}_{s' \sim P(.|s,a)} \mathbb{E}_{a' \sim \pi(.|s')}[\varphi(s', a')])^T \\ &= \mathbb{E}_{s,a\sim\rho}\left[\varphi(s,a)(\varphi(s,a) - \gamma \mathbb{E}_{s' \sim P(.|s,a)} \mathbb{E}_{a' \sim \pi(.|s')}[\varphi(s', a')])^T\right] \end{aligned} \tag{4.21}$$

and respectively

$$b_j = \Phi^T \Delta_\rho \mathrm{R} = \sum_{s,a} \rho(s,a)\varphi(s,a) r_j(s,a) = \mathbb{E}_{s,a\sim\rho}\left[\varphi(s,a) r_j(s,a)\right] \tag{4.22}$$

And now we can see that B^π and b_j can be easily approximated via samples in $\mathtt{D}_j$. To make this explicit, if we assume we have access to a dataset for each of the tasks of interest $\mathtt{D}_j$, we can use this data to:

- **Build** b_j, for each task j:

$$b_{j,\mathtt{D}_j} = \sum_{s,a} \varphi(s,a) r_j, \forall \text{ samples } (s,a,r_j,s') \in \mathtt{D}_j \text{ on task } j \tag{4.23}$$

- **Build** B^π, for **any policy** π:

$$B^\pi_{\mathtt{D}_j} = \sum_{s,a} \varphi(s,a)[\varphi(s,a) - \gamma\varphi(s',\pi(s'))]^T, \forall \text{ samples } (s,a,r_j,s') \in \mathtt{D}_j \text{ on task } j \tag{4.24}$$

  where by $\varphi(s',\pi(s'))$ we denote the expected feature value at state s' under policy π: $\varphi(s',\pi(s')) = \mathbb{E}_{a'\sim\pi(.|s')}[\varphi(s',a')]^T = \sum_{a'} \pi(a'|s')\varphi(s',a')$. Note that in order to compute this expectation we only need access to the π which is the behaviour we are trying to evaluate. In this chapter we would look primarily at deterministic strategies that were computed via a greedy improvement over previous estimates. In this case, the computation of the above expectation is particular simple as we need to evaluate the feature set φ at $(s',\pi(s'))$.

Thus for any policy π given these two estimates, we can compute w^π_j via Eq. 4.20 and obtain $\tilde{Q}^\pi_j$ – which represents the least-squares solution to the (projected) policy evaluation. If we consider this procedure in the context of Policy Iteration we obtain Algorithm 4.1. This is a straight-forward application of LSPI (Lagoudakis and Parr, 2003) to each task j independently, that uses the batch collected on task j to build the B^{π_j}, b_j.

The assumption in a multitask setting is that the J independent problems in Algorithm 4.1 may share some common structure and we could do better by considering the joint learning problem instead. We have already seen evidence of that in the last chapter, where we showed how one can learn a shared representation across tasks to benefit all tasks. In this chapter we are going to do something similar, but taking explicit advantage of the persistence property across the transition kernels. The main insight here is to notice that computing the parameters w^π_j involve two estimates B^π and b_j that nicely decouple the environment (and policy) and the task (reward signal). In particular, note that the computation of the matrices B^π do not contain any information particular to any of the tasks and for any policy π, in order to build an estimate of B^π we only need access to samples (s,a,s'), where (s,a) are drawn for the stationary distribution under the behaviour policy μ and

Algorithm 4.1 Least Squares Policy Iteration for Individual tasks

```
Require: $\mathtt{D} = \cup_{j=1}^{J} \mathtt{D}_j$, set of experiences for each task $j$

  Initialise $\Theta = \Theta_0, k = 0$

  Precompute for all tasks $j$:
  $b_{j,\mathtt{D}_j} = \sum_{s,a} \varphi(s,a) r_j, \ \forall (s,a,r_j,s') \in \mathtt{D}_j$

  while convergence not reached $(d\Theta < \varepsilon || k < \text{MaxIter})$ do
    Policy Improvement:
    $\pi_j^{(k)}(a|s) = \arg\max_a (\varphi(s,a)^T w_j^{(k-1)})$
    Policy Evaluation:
    $B_{\mathtt{D}_j}^{\pi_j} = \sum_{s,a} \varphi(s,a)[\varphi(s,a) - \gamma \varphi(s', \pi_j^{(k)}(s'))]^T, \ \forall (s,a,r_j,s') \in \mathtt{D}_j$
    $w_j^{(k)} = \left( B_{\mathtt{D}_j}^{\pi_j} \right)^{-1} b_{j,\mathtt{D}_j}$
  end while

  return $\Theta = \{w_j\}_{j=1}^{J}$ and learnt policies $\Pi = \{\pi_j\}_{j=1}^{J} \approx \Pi^*$
```

$s' \sim P(.|s,a)$. Thus *any* of the samples in $\mathtt{D} = \cup_{j=1}^{J} \mathtt{D}_j$ could be used for this estimate. This essentially means that we can use all samples collected under any task to build a much better estimate of B^π in Eq. 4.22.

$$B_{\mathtt{D}}^{\pi} = \sum_{s,a} \varphi(s,a)[\varphi(s,a) - \gamma \varphi(s', \pi(s'))]^T, \forall \text{ samples } (s,a,r_*,s') \in \mathtt{D} \qquad (4.25)$$

The only problem now is that in order to compute w_j^π we would need $b_{j,\mathtt{D}}$ and for this computation we would need $r_j(s,a)$ for all samples $(s,a) \in \mathtt{D}$. If one has access to the full set of reward functions via the task specifications, one could compute the rewards associated with samples $(s,a) \in \mathtt{D} \setminus \mathtt{D}_j$ on demand. For instance, if the task is defined as navigation to a target position g, the reward function can be simply $z > 0$ at $s = g$ and 0 for any other transition. In this case, we can compute the missing reward evaluation $r_j(s,a)$ for each tuple $(s,a) \in \mathtt{D} \setminus \mathtt{D}_j$ and it should be clear that we can effectively extend the task datasets to include all samples across tasks and apply Algorithm 4.1 for the extended sets $\bar{\mathtt{D}}_j = \{(s,a,r_j,s') | (s,a,s') \in \mathtt{D}, r_j = r_j(s,a)\}$.

Nevertheless, in general we might not have access to the functional form of r_j for the tasks of interests. In this case, we will build an estimate of $b_{j,\mathtt{D}}$ based on previous estimations of w_j^π across the tasks. To see how one can do this, let us recall the Bellman Expectation equation:

$$\tilde{Q}_j^\pi(s,a) = r_j(s,a) + \gamma \mathbb{E}_{s' \sim P(.|s,a)} \left[\tilde{Q}_j^\pi(s', \pi(s')) \right]$$

Rewriting this, we can see that one can express the reward function $r_j(s,a)$ as:

$$\begin{aligned}
\Rightarrow r_j(s,a) &= \tilde{Q}_j^\pi(s,a) - \gamma \mathbb{E}_{s' \sim P(.|s,a)} \left[\tilde{Q}_j^\pi(s', \pi(s')) \right] \\
&= \left(\varphi(s,a) - \gamma \mathbb{E}_{s' \sim P(.|s,a)} \left[\varphi(s', \pi(s')) \right] \right)^T w_j^\pi \qquad (4.26) \\
\tilde{r}_j(s,a) &\approx \left[\varphi(s,a) - \gamma \varphi(s', \pi(s')) \right]^T w_j^\pi, \text{ for } s' \sim P(.|s,a) \qquad (4.27)
\end{aligned}$$

Note that in the above equation, the only term that depends on the task j is w_j^π and all others depend solely on samples $(s,a,s') \sim (d^\mu, \mu(.|s), P({}'|s,a))$. This holds for any policy π and any task j. Throughout learning, we would generally have access to $w_j^{(k)}$ which corresponds to evaluating policy $\pi_j = \text{argmax}_a \varphi(s,a)^T w_j^{(k-1)}$. Thus we build term $[\varphi(s,a) - \gamma\varphi(s', \pi(s'))]^T$ for $\pi_j = \text{argmax}_a \varphi(s,a)^T w_j^{(k-1)}$, for all experience $(s,a,s') \in \mathtt{D}$ and use $w_j^{(k)}$ to get an estimate for the reward function $\tilde{r}_j$. Moreover, it is worth noting that we do not need to compute $\tilde{r}_j$ explicitly, but only its summed projections:

$$\begin{aligned}
b_{j,\mathtt{D}} &= \sum_{(s,a,s') \in \mathtt{D}} \varphi(s,a) r_j(s,a) \\
&\approx \sum_{(s,a,s') \in \mathtt{D}} \varphi(s,a) \tilde{r}_j(s,a) \\
&= \sum_{(s,a,s') \in \mathtt{D}} \varphi(s,a) \left[\varphi(s,a) - \gamma \varphi(s', \pi(s')) \right]^T w_j^\pi = B_{\mathtt{D}}^\pi w_j^\pi \qquad (4.28)
\end{aligned}$$

All in all, the above give us a way of approximating $b_{j,\mathtt{D}}$ and this particular form enjoys the same scalability properties associated with LSPI. In particular, note that all matrices and vectors we need to build and remember between iteration depend only on d, the dimension of the feature set φ and not the number of samples.

Based on the above, we can formulated a multitask LSPI (MT-LSPI) algorithm that can effectively use the data across all tasks. Before we do that, we make a final observation, which is: if we do have access to r_j and can build $\bar{\mathtt{D}}_j$ as explained above one ought to do so and use the real reward $r_j(s,a)$. When that is not the case, Eq. 4.28 gives us a way of effectively inferring $\tilde{r}_j \approx r_j$. In our considered scenario, *on-task* samples, coming from $\mathtt{D}_j$, will contain the actual reward function $r_j(s,a)$ and for all other *off-task* samples we will use the estimation scheme proposed in Eq. 4.28. This results in the following combined estimate:

$$\begin{aligned} b_{j,\mathtt{D}} &= \sum_{(s,a,s')\in\mathtt{D}} \varphi(s,a) r_j(s,a) \\ &\approx \underbrace{\sum_{(s,a,s')\in\mathtt{D}} \varphi(s,a) r_j(s,a)}_{b_{j,D_j}} + \underbrace{\sum_{(s,a,s')\in\mathtt{D}\setminus\mathtt{D}_j} \varphi(s,a)\tilde{r}_j(s,a)}_{\tilde{b}_{j,\mathtt{D}\setminus\mathtt{D}_j}} \\ &= b_{j,D_j} + B^{\pi}_{\mathtt{D}\setminus\mathtt{D}_j} w^{\pi}_j \end{aligned} \tag{4.29}$$

Based on this last modification we can now introduce our proposed algorithm *MT-LSPI with reward prediction*, detailed in Algorithm 4.2 below. Note that the estimation of $b_{j,\mathtt{D}}$ needs to be done only once and then re-used for the all the subsequent iterations of PI. This is akin to the pre-computation of $b_{j,\mathtt{D}_j}$ in Algorithm 4.1. In Algorithm 4.2 we propose revisiting this computation at each iteration, but we indicate that this is an optional step, once we have computed at least one *off-task reward* estimate $b_{j,\mathtt{D}\setminus\mathtt{D}_j}$. The reason behind revisiting this estimate is that we expect later iterations of PI to produce better estimates to the target evaluations, which could result in a better estimate of the inferred rewards $\tilde{r}_j$. Finally, we note that this is more likely to happen in a online setting and we will see in the experimental section that this would have almost no effect when considering a fixed dataset. Other alternatives one could consider here are averaging over estimates produced at each iteration or simply selecting the one which produces the smallest residual.

4.4 Multitask - LSPI: A Residual Minimisation Approach

In this section we will pick up our discussion from Section 4.2.2 on least-squares methods for Bellman residual minimisation and extend it to our considered multitask RL setting. The idea behind this section is to investigate how one can approach the individual BRM minimisation problems as a (joint) multitask learning problem predicting the reward associated with each task. As before, we are going to be approximating the action-value function via a linear approximation in the span of a d dimensional feature space $\varphi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^d$.

Although we are considering a *Policy Iteration* procedure, the multitask learning prob-

Algorithm 4.2 Multitask Least Squares Policy Iteration (MT - LSPI)

```
Require: $\mathtt{D} = \{\mathtt{D}_j\}_{j=1}^{J}$, set of experiences for each task $j$
  Initialise $\Theta = \Theta_0, k = 0$
  Precompute for all tasks $j$:
  $b_{j,\mathtt{D}_j} = \sum_{s,a} \varphi(s,a) r_j, \ \forall (s,a,r_j,s') \in \mathtt{D}_j, \ b_{j,\mathtt{D}\backslash\mathtt{D}_j} = 0$
  while convergence not reached ($d\Theta < \varepsilon || k <$ MaxIter) do
    for all tasks $j = 1 : J$ do
      Policy Improvement:
      $\pi_j^{(k)}(a|s) = \arg\max_a(\varphi(s,a)^T w_j^{(k-1)})$
      Policy Evaluation:
      1) On-task Evaluation:
      $B_{\mathtt{D}_j}^{\pi^{(k)}} = \sum_{s,a,s'} \varphi(s,a)[\varphi(s,a,s') - \gamma\varphi(s',\pi_j^{(k)}(s'))]^T, \ \forall(s,a,r_j,s') \in \mathtt{D}_j$
      2) Off-task Evaluation:
      $B_{\mathtt{D}_{j'}}^{\pi^{(k)}} = \sum_{s,a,s'} \varphi(s,a)[\varphi(s,a) - \gamma\varphi(s',\pi_j^{(k)}(s'))]^T, \ \forall(s,a,r_{j'},s') \in \mathtt{D}_{j'}, \forall j' \neq j$
      $B_{\mathtt{D}\backslash\mathtt{D}_j}^{\pi^{(k)}} = \sum_{j' \neq j} B_{\mathtt{D}_{j'}}^{\pi^{(k)}}$
      3) New parameters based on both on- and off-task evaluations:
      $w_j^{(k)} = \left(B_{\mathtt{D}_j}^{\pi^{(k)}} + B_{\mathtt{D}\backslash\mathtt{D}_j}^{\pi^{(k)}}\right)^{-1} (b_{j,\mathtt{D}_j} + b_{j,\mathtt{D}\backslash\mathtt{D}_j})$
      4) [Optional for $k > 1$] Off-task Reward Prediction:
      $b_{j,\mathtt{D}\backslash\mathtt{D}_j} = B_{\mathtt{D}\backslash\mathtt{D}_j}^{\pi^{(k)}} w_j^{(k)}$
    end for
  end while
  return $\Theta = \{w_j\}_{j=1}^{J}$ and learnt policies $\Pi = \{\pi_j\}_{j=1}^{J} \approx \Pi^*$
```

lems we will be considering occur in the Policy Evaluation step. In the multitask scenario, every (full) step of policy iteration, k, will involve coming up with approximations $\{\tilde{Q}_j^{\pi_j} = \Phi w_j^{\pi_j}\}_{j=1,J}$ for a set of policies $\Pi^{(k)} = \{\pi_j\}_{j=1,J}$. This is akin to the MT-FQI algorithms proposed in the last chapter –Section 3.2, but in this section we will explore a different joint evaluation problem, solved via *Bellman Residual minimisation*.

Let us first recall that the state-action value function Q_j^π representing the target for our policy evaluation steps, is the solution of the Bellman Expectation equation, for a given policy π and reward signal r_j:

$$Q_j^\pi(s,a) = r_j(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s') Q_j^\pi(s',a') \tag{4.30}$$

A way of finding a good approximation to Eq. 4.30, is by finding the approximation $\tilde{Q} \in \mathcal{F}_\varphi$ that satisfies the above equation as closely as possible: $\min_{\tilde{Q}\in\mathcal{F}_\varphi} \|\tilde{Q} - (T_{r_j}^\pi \tilde{Q})\|_2$. This is the idea behind BRM methods. Following this reasoning, substituting the approximation $\tilde{Q}_j^\pi$ in the above equation, we obtain:

$$\begin{aligned}
\tilde{Q}_j^\pi(s,a) &\approx r_j(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s') \tilde{Q}_j^\pi(s',a') \\
\Phi(s,a) w_j^\pi &\approx r_j(s,a) + \gamma \mathbb{E}_{s'\sim P(.|s,a)}\left[\Phi(s',\pi(s'))\right] w_j^\pi \\
\underbrace{\left(\Phi(s,a) - \gamma \mathbb{E}_{s'\sim P(.|s,a)}\left[\Phi(s',\pi(s'))\right]\right)}_{:=z_\pi(s,a)} w_j^\pi &\approx r_j(s,a)
\end{aligned}$$

where, as before, we denote by $f(s,\pi(s))$ the expectation of the function f with respect to the policy π: $f(s,\pi(s)) = \mathbb{E}_{a\sim\pi}[f(s,a)]$. Leading to the following linear regression problem:

$$z_\pi(s,a) w_j^\pi \approx r_j(s,a), \forall (s,a) \in \mathcal{A} \times \mathcal{S} \tag{4.31}$$

And note that the least-square solution of the above is:

$$w_j^\pi = \left(Z_\pi^T Z_\pi\right)^{-1} Z_\pi^T \mathrm{R}_j \tag{4.32}$$

where Z_π, R_j denotes the vectors $z_\pi(s,a)$, respectively $r_j(s,a)$ across the space $\mathcal{S} \times \mathcal{A}$. Note that the above recovers the known BRM solution introduced in Section 4.2.2. Hence, if we were to consider each policy evaluation problem independently, the PE step would involve building matrices Z_{π_j} for each policy j and then regressing on the reward vector associated with the tasks R_j.

As before, in the multitask/multi-policy setting, we assume that the evaluation problems considered are related and the scope of this work is to find a way to expose and exploit the common structure. To do so, let us first take a moment to observe that $z_\pi(s,a) = \left(\varphi(s,a) - \gamma\mathbb{E}_{s'\sim P(.|s,a)}[\varphi(s',\pi(s'))]\right)$ is *task-invariant*, but *policy dependent*, as we require that the action at the next state s' is chosen according to policy π which we are currently trying to evaluate. Based on this observation we will proceed to formulate two different multitask problems one can tackle for a multi-policy, multitask evaluation step. Let us first revise the evaluation problem we are interested in for such an MT-evaluation step:

$$\begin{cases} Z_{\pi_1} w_1^{\pi_1} & \approx R_1 \\ Z_{\pi_2} w_2^{\pi_2} & \approx R_2 \\ & \dots \\ Z_{\pi_J} w_J^{\pi_J} & \approx R_J \end{cases} \Leftrightarrow \begin{cases} z_{\pi_1}(s,a) w_1^{\pi_1} & \approx r_1(s,a) \\ z_{\pi_2}(s,a) w_2^{\pi_2} & \approx r_2(s,a) \\ & \dots \\ z_{\pi_J}(s,a) w_J^{\pi_J} & \approx r_J(s,a) \end{cases}, \forall (s,a) \in \mathcal{S} \times \mathcal{A} \tag{4.33}$$

In the following, we propose two methods for solving the above regression problem under slightly different assumptions on the common structure present in the problem. The first proposal will deal with the above problem directly, inferring $w^{\pi_j}{}_j$ and the second one will focus on z_π for a particular policy π and will try to learn commonalities of this policy across tasks, inferring $\{w_j^\pi\}$ for every $\pi \in \Pi = \{\pi_i\}_{i=1,J}$.

4.4.1 Learning Shared Representation over Control Policies

In this section we explore tackling the multitask problem in Eq. 4.33 and a fairly standard way of doing this is by a regularised version of this objective. In particular, the optimisation problem we are trying to solve can be expressed as:

$$W = \arg\min_W \left[\sum_{j=1}^{J} \|Z_{\pi_j} w_j^{\pi_j} - R_j\|_\rho^2 + \lambda \mathcal{H}(W) \right] \tag{4.34}$$

where $W = [w_1^{\pi_1}, w_2^{\pi_2}, \cdots, w_J^{\pi_J}] \in \mathbb{R}^{d\times J}$ and $\mathcal{H}(W)$ is a regulariser over the set of weight vectors $w_j^{\pi_j}$, modelling the structure between the individual learning problems. The above problem characterises each step of multi-policy, multitask evaluation step for a given set of policies Π. In general we will encounter these types of problems in the inner loop of Policy Iteration and this will be the focus of our investigation. The proposed algorithm, in the context of PI, is detailed in Algorithm 4.3 below.

Algorithm 4.3 Least Squares Policy Iteration via Multitask Bellman Residual minimisation (LSPI-MTBRM)

```
Require: $\mathtt{D} = \{\mathtt{D}_j\}_{j=1}^{J}$, set of experiences for each task $j$
  Initialise $\Theta = \Theta_0, k = 0$
  while convergence not reached $(d\Theta < \varepsilon || k < \text{MaxIter})$ do
    Policy Improvement:
    Compute improved policies $\Pi^{(k)} = \left[\pi_1, \cdots, \pi_j, \cdots, \pi_J\right]$.:
        $\pi_j(a|s) = \arg\max_a(\varphi(s,a)^T w_j^{(k-1)})$ for each task $j$
    where $w_j^{(k-1)}$ is the $j$-s column of $W^{(k-1)}$.
    Policy Evaluation:
    1) Compute $z_\pi$ for all policies $\pi \in \Pi^{(k)}$ :
        $z_{\pi_j}(s,a) = \left[\varphi(s,a) - \gamma\varphi(s', \pi_j(s'))\right] \ \forall (s,a,s') \in \mathtt{D}_j, \forall j = \{1, \cdots, J\}$
    2) Compute $W = \left[w_1^{\pi_1}, \cdots, w_j^{\pi_j}, \cdots, w_J^{\pi_J}\right]$ via Alg. 3.2:
        $W^{(k)} = \arg\min_{W=\{w_j^{\pi_j}\}_{j=1:J}} \left[\sum_j \mathcal{L}_{\mathtt{D}_j}\left(\langle z_{\pi_i}(s,a), w_j^{\pi_j}\rangle, r_j(s,a)\right) + \mathcal{H}(W)\right]$
  end while
  return $\Theta = \{w_j^{\pi_j}\}_{j=1}^{J}$ and learnt policies $\Pi = \{\pi_j\}_{j=1}^{J} \approx \Pi^*$
```

A particularly effective way of doing so was discussed in the previous chapter: $\mathcal{H}(W) = ||W||_{tr}^2 = Tr(WW^T)$. We will adopt this as our multitask learning procedure for the rest of this chapter, although other approaches can be swapped in. We have seen in Section 3.2 that in the above formulation one can learn compact linear representation of the state-action space shared across all tasks. As a reminder, the assumption we were making there was that: there exists a more compact representation ψ based on φ that can span the value functions we are interested in representing:

$$\exists U \text{ s.t. } \psi(s,a) = U^T\varphi(s,a) \Rightarrow \tilde{Q}_j = \langle \varphi(s,a), w_j \rangle = \langle U^T\varphi(s,a), Uw_j \rangle = \langle \psi(s,a), a_j \rangle \tag{4.35}$$

where we impose that this representation supports a sharing structure across tasks, encouraged by a $\mathcal{L}_{2,1}$ regularisation on matrix $A = [a_1, \cdots, a_J]$. In (Argyriou et al., 2008) was shown that solving the optimisation problem in Eq. 4.34 is equivalent to solving:

$$\arg\min_{A,U} = \left[\sum_{j=1}^{J} \|Z_{\pi_j} U a_j^{\pi_j} - R_j\|_\rho^2 + \lambda \mathcal{H}(A) \right] \tag{4.36}$$

which leads to an alternating minimisation algorithm that factorises over tasks. For further details, we would refer the reader to Section 3.2 and the original work in (Argyriou et al., 2008).

Note that the assumption on the joint representation considered here is essentially the same as the one made in the last chapter, Section 3.2, as if $\exists U$, unitary matrix s.t. $\psi(s,a) = U\varphi(s,a)$ then:

$$\begin{aligned} \varphi(s,a) = U^T\psi(s,a) \Rightarrow z_\pi &= \left(\varphi(s,a) - \mathbb{E}_{s'}[\varphi(s', \pi(s'))]\right) \\ &= U^T \underbrace{\left(\psi(s,a) - \mathbb{E}_{s'}[\psi(s', \pi(s'))]\right)}_{z_{\pi,\psi}} \end{aligned} \tag{4.37}$$

Thus there exists a compact representation $z_{\pi,\psi} = U^T z_\pi$ via the same shared rotation U. Nevertheless, the nature of the (multitask) regression problems is different: one was considering iterative backups of the Optimality Bellman operator, as the other is considering a regression towards the reward signals associated with each task. Thus here the PE step this is not an iterative process any more, it is a one-shot solution that yields the BRM least-squares solution. Also worth noting that this rotation matrix U is independent of the policy π, but at the same time it will only need to accommodate the current set of policies up for evaluation at the current iteration. Note that although, we assumed a universal U for all policies we might encounter, in practice this transformation U can and will evolve throughout iterations as the estimation problems it is trying to support do too. In the end, this learnt transformation will represent the common structure in the optimal policies, if achieved in the improvement process.

4.4.2 Learning Policy-dependent Representations

In this section, we propose a different way of approaching the multi-task, multi-policy evaluation step. In the last section, we proposed solving the MT-evaluation problem in Eq. 4.33 by finding a common linear transformation U. Note that in this formulation the input representation of each task z_{π_j} is policy dependent – thus for the first time in this work, we were tackling a MT problem that has a task-dependent input. Nevertheless, we propose a learning procedure based on the factorisation presented in the solution space W and the duality of this transformation, acting as a shared representation for the common feature space φ. Note that this transformation is universal as it applies to any set of policies. We would argue here that, in general, it might be hard to learn such with a common transformation over a set of task-dependent features, especially during the early stages of PI. Instead, in this section, we propose tackling the MT-evaluation step via a series of simpler problems that look at evaluating just one policy at the time, but using all data available. This is partially inspired by Section 4.3, where we have looked at re-using data off-task to help build the evaluation for policy π on a given task. The idea here is similar, but the way we are going to tackle this evaluation problem, per policy π, is a bit different.

More specifically, for any given policy π we are trying to evaluate, we are going to be considering the following joint optimisation problem:

$$W^{\pi} = \arg\min_{W} \left[\sum_{j=1}^{J} \|Z_{\pi} w_j^{\pi} - R_j\|_{\rho}^{2} + \mathcal{H}(W^{\pi}) \right] \tag{4.38}$$

where $W^{\pi} = [w_1^{\pi}, w_2^{\pi}, \cdots, w_J^{\pi}] \in \mathbb{R}^{d \times J}$. Now the difference with respect to the previously investigated problem in Eq. 4.34 is that, in the above, we are considering representation of z_{π} for one policy π, rather than across the batch of policies we are trying to evaluation at this PE step. Although we might not use all of the entries in W^{π}, the above formulation let us use all data across all tasks to learn about policy π, under different reward signals.

Note that we would have one such problem for every policy $\pi \in \Pi$ for all tasks for which we are interested in finding the optimal policy. As the number of tasks increase, so does the number of problems we will need to solve at each evaluation step. Nevertheless, as argued above, these individual problems are much simpler as they do not need to model the relationship between the optimal/intermediate solutions, but only need to account for the similarities across tasks ,under a fixed policy π.

Algorithm 4.4 Least Squares Policy Iteration via Multitask Bellman Residual minimisation over Policy Representations (LSPI-MTBRM-PR)

```
Require: $\mathtt{D} = \{\mathtt{D}_j\}_{j=1}^{J}$, set of experiences for each task $j$
  Initialise $\Theta = \Theta_0, k = 0$
  while convergence not reached $(d\Theta < \varepsilon || k < \text{MaxIter})$ do
    Policy Improvement:
    Compute improved policies $\Pi^{(k)} = \left[\pi_1, \cdots, \pi_j, \cdots, \pi_J\right]$
        $\pi_j(a|s) = \arg\max_a(\varphi(s,a)^T w_j^{(k-1)})$ for each task $j$
    where $w_j^{(k-1)}$ is the $j$-s diagonal entry of $W^{(k-1)}$.
    Policy Evaluation:
    1) Compute $z_\pi$ for all policies $\pi \in \Pi^{(k)}$ :
        $z_\pi(s,a) = \left[\varphi(s,a) - \gamma\varphi(s', \pi(s'))\right] \ \forall (s,a,s') \in \mathtt{D}$
    2) Compute $W^{(k)} = \left[w_1^{\pi_1}, \cdots, w_j^{\pi_j}, \cdots, w_J^{\pi_J}\right]$
    for all tasks $j'$ in $1:J$ do
      Compute $W^\pi = \left[w_1^\pi, \cdots, w_j^\pi, \cdots, w_J^\pi\right]$ for $\pi = \pi_{j'}$ via Alg. 3.2:
        $W^\pi = \underset{W=\{w_j^\pi\}_{j=1:J}}{\operatorname{argmin}} \left[\sum_j \mathcal{L}_{\mathtt{D}_j}\left(\langle z_\pi(s,a), w_j^\pi\rangle, r_j(s,a)\right) + \mathcal{H}(W^\pi)\right]$
    end for
    Store $W^\pi = \{w_j^\pi\}_{j=1:J}$ in the $j'$ column of $W^{(k)}$.
  end while
  return $\Theta = \{w_j^{\pi_j}\}_{j=1}^{J}$ and learnt policies $\Pi = \{\pi_j\}_{j=1}^{J} \approx \Pi^*$
```

Thus the proposed algorithm follows a similar structure as Alg. 4.3, but now the optimisation step in the policy evaluation will involve learning a shared representation that is policy dependent. The full algorithm is detailed in Alg. 4.4.To be consistent with previous algorithms we opted not to change the improvement step. Nevertheless, we could in principle employ an improvement step that looks at all the current policies Π and tries to act greedily with respect to the best one under current evaluation. This is a generalisation of the normal greedy improvement step, and readily applicable to any situations where one has access to multiple policy evaluations (Barreto et al., 2017). This is indeed the case for the Algorithm 4.4 proposed here, as the set $\{W^{\pi}\}_{\pi\in\Pi}$ estimated at each iteration, readily provides estimations for evaluations $\{\tilde{Q}_j^{\pi} = \Phi w_j^{\pi}\}$, for every $\pi \in \Pi$.

4.5 Experiments

In this section, we empirically validate and investigate the effectiveness of the algorithms proposed in the last sections. In particular, we show that our multi-task variants consistently improve performance over their single-task counterparts – LPSI and PI with least-squares BRM. As before, we study these problems under a constrained per-task budget – this is the regime in which we expect these type of algorithms to be helpful. The more we increase the budget (within a task), the less the learning has to rely on information coming from the other tasks.

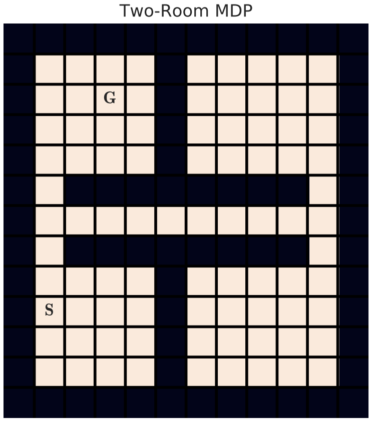


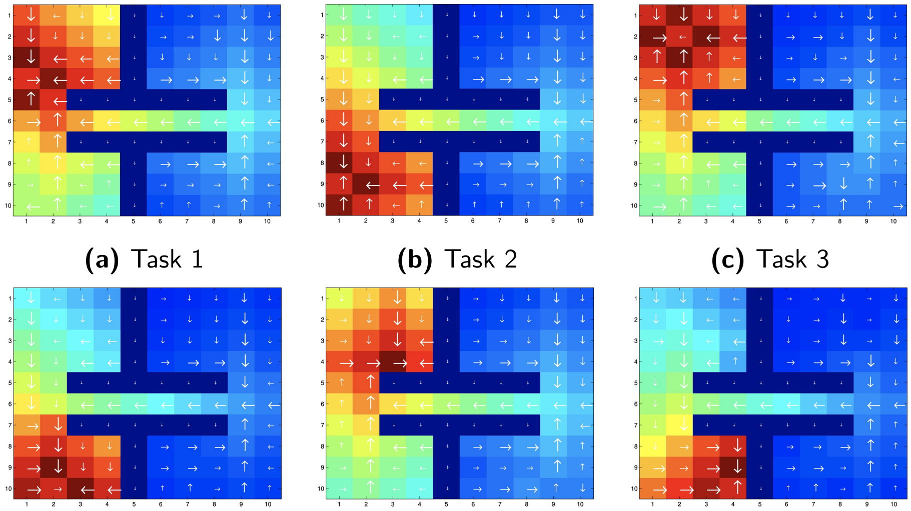
Figure 4.5.1: Depiction of the Two-Room MDP. Starting states, **S** and Goals states, **G** were sampled randomly. The goal states define the different tasks.

Similar to the experimental setup proposed for MT-FQI, we consider a series of navigation tasks in a 10 × 10 maze-like environment – see Figure 4.5.1 – to test the proposed algorithms. We sample n_{tasks} goal locations, at random from the set of reachable positions and we provide the agent with a positive reward ($+10$) once the goal is reached, and no reward signal otherwise. All transitions in this environment are deterministic and interaction with the walls is purely elastic - has no effect on the agent's position nor does it provide any (negative) reward. We vary the number of tasks n_{tasks}, but for each choice, we sample the corresponding goals once and keep this selection fixed throughout our experiments. This is mostly to make the different version of the algorithms and baselines comparable, as the inter-task variance is quite high, especially for small n_{tasks}. Across multiple experiments conducted, the trends remain the same: we consistently improve over the single-task algorithm, but the magnitude of this improvement is dependent on the set of tasks selected, the data gathered under each of these tasks and the amount of information the data collected under the different tasks brings to each on-task optimisation problem.

As before, we opt for a batch-mode validation[1]. This means we provide the agent with $n_{episodes}$ worth of experience for each task $j \in \{1, \cdots, J\}$ – each of these episodes is at most 25 steps in the environment, after which the episode ends and the agent is re-spawned in a random location in the grid. These episodes were sampled a priori under a behaviour policy μ which in our case was the uniformly random policy and this (per-task) dataset, $\mathtt{D} = \cup_j \mathtt{D}_j$ will not be revisited or augmented with online interaction with the environment. Throughout these experiments we restrict the number of samples in $\mathtt{D}_j$ to a small enough number as that most of the time, we are in a sample regime in which the single task methods, based solely in $\mathtt{D}_j$ for each task j, fail to recover the optimal policies/value functions.

As a teaser, in Figure 4.5.2 we present some of the value functions and policies MT-LSPI is able to learn in this environment. We can see that the learnt value functions give rise to nearly optimal policies for all tasks considered, even when the per-task data is not enough to learn the full value function. These were policies learnt via one of the MT Bellman Residual minimisation method proposed in Section 4.4. In particular, the above value functions were obtained by running Alg. 4.3, for $n_{tasks} = 10$ tasks, under a budget of $n_{epsiodes} = 100$ episodes per task. The training curves for this experiment can be found in Figure 4.5.5 in the results' discussion below. Looking at this figure (reporting aggregated performance across tasks), one can see that, under these conditions, the single-task BRM

[1] Our proposed algorithms extend easily to an online/on-policy interaction. Nevertheless, for this investigation, we opted to focus on the benefits for the MT treatment of the problem and the better generalisation that this formulation can bring about in estimating value functions, eliminating effects due to exploration.



(a) Task 1 **(b)** Task 2 **(c)** Task 3

(d) Task 4 **(e)** Task 5 **(f)** Task 6

Figure 4.5.2: Samples of the value functions and policy learnt via MT-LSPI

although it obtains a reasonable performance, fails to solve the task 30% of the time, while Alg. 4.3 manages to further improve on this performance and achieve close to optimal behaviour – as illustrated by the estimated value function in Figure 4.5.2. In the following, we will go through more empirical results generated by the different algorithms proposed in this chapter.

4.5.1 Results: Transfer of Samples

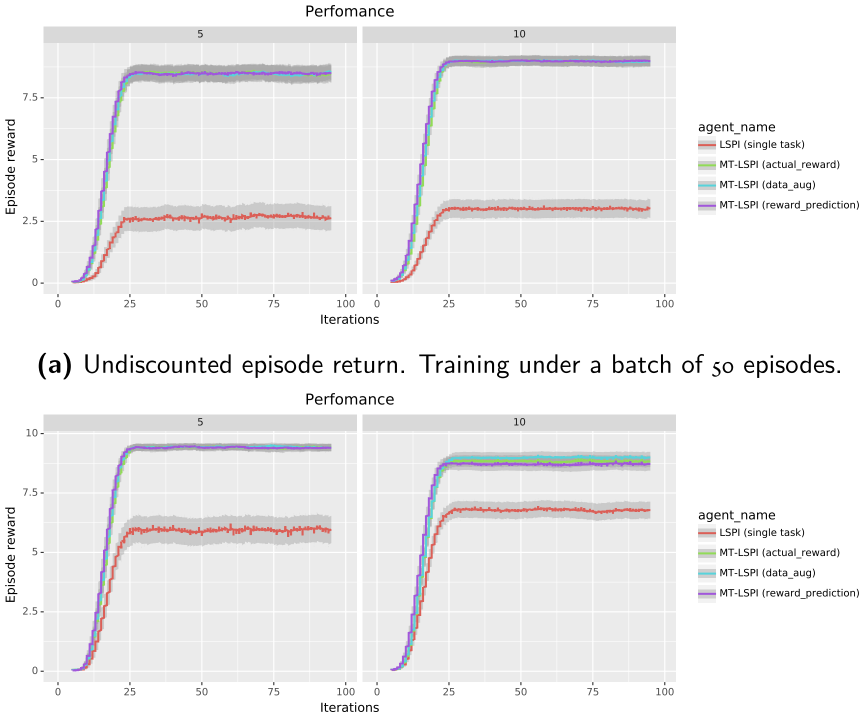
We start with Alg. 4.2, which is an extension of the popular LSPI algorithm where we propose a more effective way of using data across tasks in the (MT) policy evaluation step. This method relies heavily on the assumption that our agent resides in a persistent environment which enable us to transfer samples of the form $(s, a, s') \in \mathtt{D}$ to build the estimate of the matrices B^{π} for any policy π. But as discussed in Section 4.3, in order to compute the parameters, w_j^{π}, of the approximation $\tilde{Q}_j^{\pi} = \Phi w_j^{\pi} \approx Q_j^{\pi}$, we would need the corresponding approximation of b_j under the same sample set. Thus if we want to use the whole dataset, generated by aggregating the experience across all tasks, we will need to come up with estimates for r_j or all samples outside $\mathtt{D}_j$.

Now, if we have access to the analytic form of the reward functions r_j we can easily evaluate all samples in $\mathtt{D}$ and effectively create a much larger on-task data set. This is indeed the case in our example, as the reward structure is very simple and re-assessing samples collected under different tasks requires just a comparison with the goals of the task we are pursuing now. Thus, in our case, this re-evaluation is easily feasible and this will be one of our baselines. We denote this baseline as **MT-LSPI (actual reward)** as it has access to the actual reward functions and we do not need to infer it. Note that in principle this is an upper bound of what we can do by using the whole data in $\mathtt{D}$ if one has access to r_j.

The more interesting, yet more challenging case is where we have access only to on-task samples of this reward function and we have to resort to inferring the missing rewards for samples in $\mathtt{D} \setminus \mathtt{D}_j$. This is precisely what Alg. 4.2 proposes. And we will look at two variants of this algorithm. In the first one, we build the estimate of $b_{j,\mathtt{D}}$ for each task j just once at the beginning of learning, based on our first iteration of policy iteration and keep this estimate fixed for future iterations. We refer to this variant as **MT-LSPI (data aug)**, as for all iterations but the first one, this acts as a per-task data augmentation procedure. The second one will constantly revisit and recompute the estimation of $b_{j,\mathtt{D}}$ based on the parameters values estimates at the latest policy iteration. We denote is variant of Alg. 4.2 as **MT-LSPI (reward prediction)**.

We ran the above variants and a single-task LSPI baseline on the experimental setup outlined in the previous section. The results are compiled in Figure 4.5.3. We include the results for two types of budget $n_{epsidoes} \in \{50, 100\}$ and different number of tasks $n_{tasks} \in \{5, 10\}$ (this indicated in the title of the cell in each of the plots). The first thing to notice is that across all conditions we are substantially improving over the single-task counterpart. Moreover, if we look at the performance of **MT-LSPI (actual reward)** which uses the same data, but has access to the true reward functions, we see that in this scenario, both versions using the reward prediction match the performance of this upper bound. We hypothesise this is mostly due to the simple structure the reward signals have in our tasks. Furthermore, as discussed in Section 4.3, we expected the two variants of reward prediction to have very similar performance under a fixed dataset, as the first iteration contains as much information about the reward as do the subsequent ones. Moreover, one can see that we can obtain optimal performance much sooner (with less data), as long as we have a decent budget and increased number of tasks (regimes in the bottom-right corner).

In Figure 4.5.4 we include further results on varying the budget and the number of tasks. We can clearly see a monotonic increase in performance as more data is available. This is



(a) Undiscounted episode return. Training under a batch of 50 episodes.

(b) Undiscounted episode return. Training under a batch of 100 episodes.

Figure 4.5.3: Performance of different variants of MT-LSPI with data augmentation and reward prediction, as compared with a single task LSPI and the actual reward for all data in the individual on-task batches. Results were average over the number of tasks (top of the plots) and over 5random generations of these set of tasks.

more noticeable for the single-task learning, as we can see that the multitask methods, making use of all data available across tasks manage to get very good performance even under very reduced budgets (see left column in Figure 4.5.4).

More surprisingly maybe is that we can achieve the same performance as knowing the actual reward associated with each transition in the off-task datasets $\mathtt{D} \setminus \mathtt{D}_j$. We hypothesise this might due to a couple of effects, but one of the main ones could be the particular reward structure we assumed for our tasks. The only non-zero rewards in this environment are at the goal state for each task and we make sure that the on-task dataset $\mathtt{D}_j$ has at least one instance of this positive reward. The rest of the rewards are zero and are shared between the many tasks considered. Thus in a sense, all the information needed to predict the reward

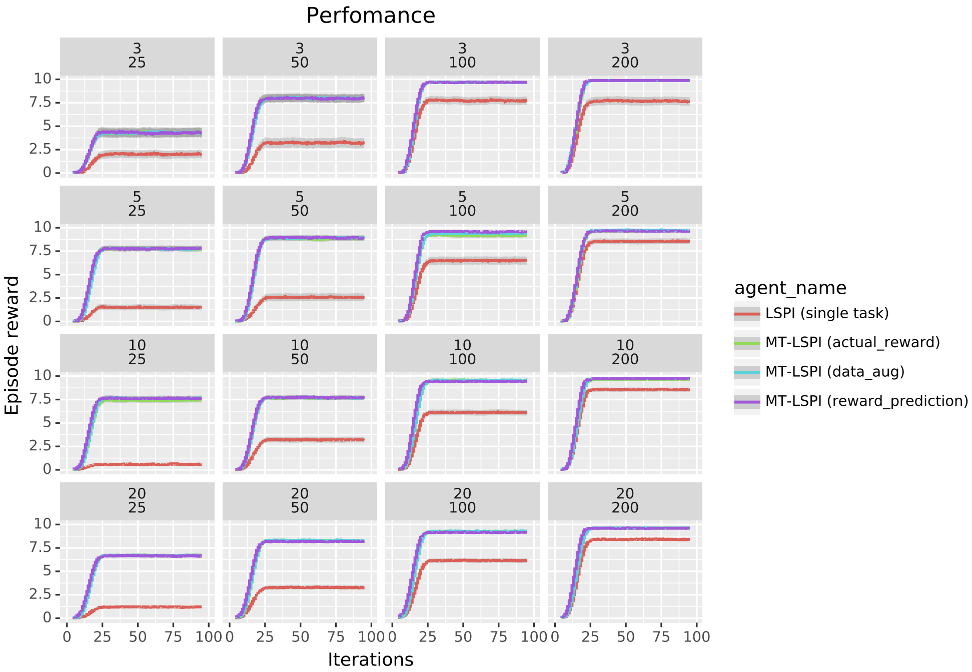


Figure 4.5.4: Empirical analysis of the dependence on number of tasks $n_{tasks} \in \{3, 5, 10, 20\}$ (top entry in the title of the cell) and the budget per task, $n_{episodes} \in \{25, 50, 100, 200\}$ (bottom entry in the title of the cell). This illustrate that data from different tasks and be used effectively by Alg. 4.2 to overcome a small budget of per-task iterations. Results were averaged over 5 seeds (data generation). Tasks were samples once for each n_{tasks}

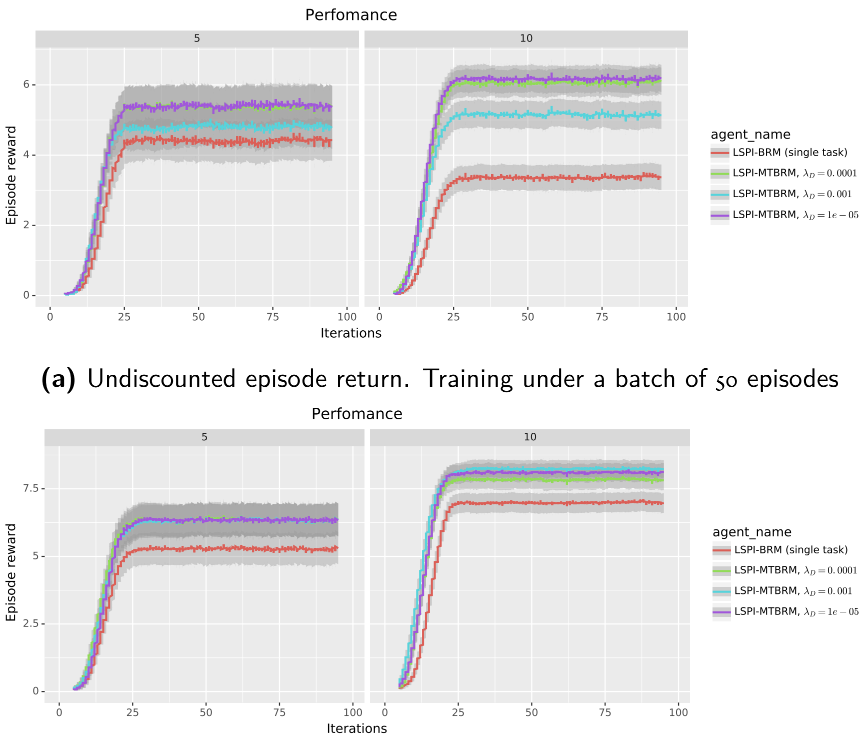
.

for a given task j is captured in $\mathtt{D}_j$ and the generalisation for the other states is trivial. The sparsity and determinism in reward signals benefit our inference problem in $b_{j,\mathtt{D}\setminus\mathtt{D}_j}$.

4.5.2 Results: Multitask Bellman Residual Minimisation

Next, we are going to turn our attention to evaluating our multitask Bellman residual methods proposed in Section 4.4. We consider the same experimental setup outlined at the beginning of our experiments section. Our baseline for this set of experiments will be the single-task LSPI seeking to minimise the Bellman residual (**LSPI-BRM single task**) as described in Eq. 4.16 with the least-squares solution given by Eq. 4.17.

Firstly, let us consider the multitask algorithm outlined in Alg. 4.3 (**LSPI-MTBRM**). This algorithm at each iteration of PI will try to solve the joint multitask regression problem Eq. 4.35 by learning a linear common representation that would support the current policies up for evaluation. As in this algorithm, we are not going to re-use any of the off-task data, the transfer of information between our task can only happen through this shared representation. This is similar to our setup in the MT-FQI case (Chapter 3), but the nature of the regression problem here is different. In the previous case, we were performing iterative backups of the per-task optimality operators, whereas here BRM methods aim to directly come up an approximation $\tilde{Q}_j^{\pi_j^{(k)}}$ that minimises the residual for the current set of policies $\Pi^{(k)} = \{\pi_j^{(k)}\}$.



(a) Undiscounted episode return. Training under a batch of 50 episodes

(b) Undiscounted episode return. Training under a batch of 100 episodes

Figure 4.5.5: Performance of different variants of LSPI-MTBRM under different regularisation strengths for $\mathcal{H}(W)$ in (Eq. 4.34). Results were average over the number of tasks (top of the plots) and over 3 random generations of $\mathtt{D}_j$ under a fixed set of tasks.

We trained our agents over 3 random selections of tasks and data generations. As before, we varied the number of tasks $n_{tasks} \in \{5, 10\}$ and the sample budgets $n_{episodes} \in \{50, 100\}$

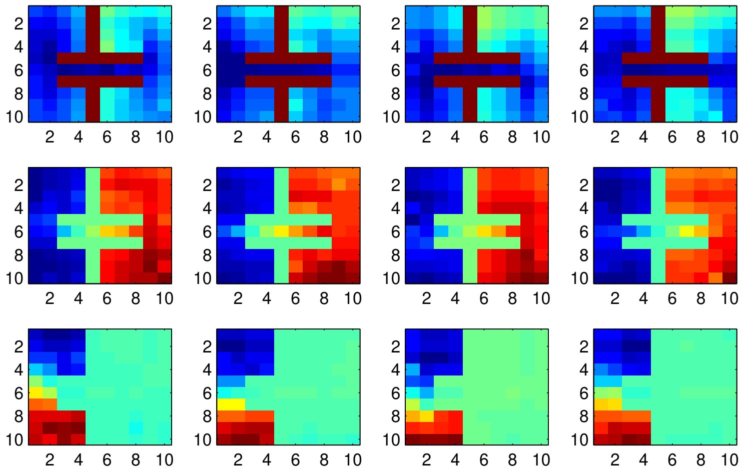
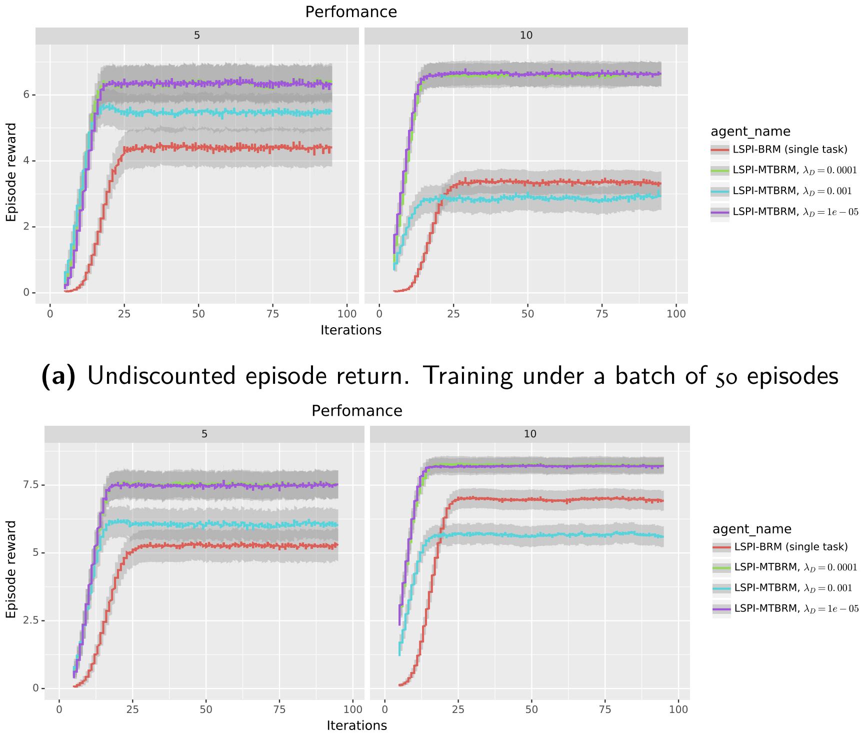


Figure 4.5.6: Top 3 features learnt by LSPI-MTBRM (end of training). Feature maps are presented row-wise. Each of the four columns corresponds to the value of the feature for each state in the grid, with respect to the four actions available: $\varphi_i(s, \rightarrow), \varphi_i(s, \uparrow), \varphi_i(s, \leftarrow), \varphi_i(s, \downarrow)$.

and consider multiple value for the regularisation parameter λ_D for the multitask regression problem. We include a selection of this results in Figure 4.5.5. For larger value of λ_D, the performance degrades considerably. This is may be due to the fact that we keep this parameter fixed throughout the learning process, although the regression problems at each iteration might be very different and the similarity between them is a combination between reward similarity and policy similarity.

Finally, it is worth taking a look at the learnt representations achieved by MT-BRM for control policies – solving problem Eq. 4.33. We record about 6 relevant dimensions to account for the shared subspace ψ - these correspond to eigenvalue greater than $\varepsilon_\lambda = 1e{-}3$. We visualise the top 3 eigenvectors obtained in Figure 4.5.6. Note that they exhibit the same compressed and highly informative structure that we encountered in the previous sections, learnt by MT-FQI. Nevertheless, this is an analysis of the representation at the end of training, where policies up for evaluation are (hopefully) competent, even close to optimal. Throughout learning, this effective dimensionality of the representation changes, but quite quickly converges to a low-dimensionality representation $(4-5)$ that supports the evaluation problems of interest. Note that is not a massive reduction as the number



(a) Undiscounted episode return. Training under a batch of 50 episodes

(b) Undiscounted episode return. Training under a batch of 100 episodes

Figure 4.5.7: Performance of different variants of LSPI-MTBRM-PR under different regularisation strengths for $\mathcal{H}(W^{\pi})$ in (Eq. 4.33). Results were averaged over the number of tasks (top of the plots) and over 3 random generations of $\mathtt{D}_j$ under a fixed set of tasks.

of tasks considered here are relatively small, but effective enough to result in performance and generalisation boost outside the individual training sets $\mathtt{D}_j$.

Lastly in this section, we are going to look at the performance and potential benefits of solving a policy dependent multitask optimisation problem as proposed in Alg. 4.4 (**LSPI-MTBRM-PR**). For comparison reasons, we maintain the same experimental setup and even the same datasets used in the previous set of experiments. The results are summarised in Figure 4.5.7. We can see that the results both in performance and speed of convergence are similar to the ones we obtained in the previous experiment, using LSPI-MTBRM. The top-performing regularisation parameters λ_D, converge to the same asymptotic best performance. We originally hypothesised that the multitask regression problems the agent needs to solve at each iteration might be simpler in the case of LSPI-MTBRM-PR, as the only varying component here is the reward structure while the environment and policy are shared by all value functions under construction. We do not seem to observe this here, but we believe this might be an artefact of the non-overlapping reward structure considered here, as most policies π_j that would be attempting to achieve a goal G_j will have a zero value under a different task j', $Q_{j'}^{\pi_j}(s, a) = 0$ outside the transition (s, a) leading to the goal, unless the goal $G_{j'}$ happens to be on the path to G_j. Thus, although the problem appears to be simpler, there may be limited transfer that can happen between these value functions. In particular, in the previous versions of the multitask problems considered we always had a mixture of policies and thus transfer of subpaths could naturally occur through enforcing the common representation. This cannot happen here anymore as the policy remains fixed in this MT problem. Nevertheless, we believe this might be a promising way of building representation when the reward signal share a lot more commonalities, as the ones we are going to explore in the second part of this thesis (Chapters 5-6). Actually, this type of joint inference problems will be the focus on the next part: successor features provide an evaluation of a common policy under different reward signals present in the environment.

4.6 Conclusion

4.6.1 Related Work

In this chapter we studied the same multitask RL problem as proposed in Chapter 3 and the relevant literature around this scenario has already been reviewed in detail in that chapter, Section 3.4.1. As such, we will instead focus here on reviewing the relevant work dealing with the underlying RL paradigm explored here: policy iteration (PI) and its proposed solutions. In particular, we will focus on the policy evaluation step of this procedure, as this is where the learning happens. It is worth noting that in recent years, most of the work in RL has focused on Q-learning based methods. Nevertheless, many studies have shown the power of a PI procedure especially for the multitask scenarios (Barreto et al., 2018; Zhang et al., 2017; Borsa et al., 2019) and representation learning (Barreto et al., 2014).

First of all, it is worth noting that one can extend the procedure proposed in Chapter 3 to deal with a policy evaluation step. This would result in an iterative procedure of applying the Bellman expectation operator and solving jointly the corresponding regression problems. As part of our study in (Borsa et al., 2016), we have tried this approach and managed to recover similar results as with MT-FQI. Nevertheless in this scenario, every policy evaluation step is an iterative problem in itself. In small data regimes, this might not be such a problem, but in this chapter, we have effectively looked at other ways of tackling this step that is much more computationally efficient under linear parametrisations. In particular, for the multitask scenario considered in the work, where the transition dynamics are shared, we have seen that we can use the compositionality in projected solution and directly leverage the experience from all tasks in building one of the terms needed in this estimation.

The idea of minimising the Bellman residual was proposed as early as in (Schweitzer and Seidmann, 1985) and several variants have been studied for instance in (Lagoudakis and Parr, 2003; Antos et al., 2008). Residual algorithms are tempting as they propose a more direct approach to the evaluation problem, rather than an iterative one and are guaranteed to convergence (Baird, 1995). Nevertheless, this class of algorithms do require a double sample, unless the environment and rewards are deterministic. Thus rendering the applicability of these methods, outside the deterministic case or model-based RL, fairly limited. It should be noted that there exist variants of BRM approaches that eliminate the need for double sampling by proposing a change in the original minimisation problem in Eq. 4.5 (Antos et al., 2008) .

In (Thiery and Scherrer, 2010), one can find a study comparing BRM approaches with projected approaches. In general, no strong statements can be made about the relative quality of their solution, but empirical experiments seem to suggest that projected policy evaluation may outperform BRM more often than not (Thiery and Scherrer, 2010).

4.6.2 Summary

In this chapter, we revised the general framework of policy iteration methods, in the context of multitask learning under linear parametrisations of the targeted value functions. In particular, we have looked at different ways one can jointly model and address the multi-policy evaluation phase in the multitask-RL problem considered. When considering a normal policy evaluation step, under functional approximation, there are two prominent ways of phrasing the learning problem: projected policy evaluation methods (described by Eq. 4.3) and Bellman residual methods (described by Eq. 4.5).

The first class of methods tries to find the solution that most closely satisfied the projected Bellman equation (Eq. 4.3). The first thing we did here is to see how these optimisation problems look under our multitask RL objective. Under a linear parametrisation of the value functions, one can easily obtain a closed-form solution that depends on two terms that can be built from samples: one depending on the environment and one depending on the reward. This factorisation in the solution is particularly useful for us as the dynamics of the environment are shared across tasks and thus the first term can be built across tasks, using all of the experience. This is a particularly simple form of transfer, but a very powerful one as we have seen in the experimental section, exploiting the structure in our MDP. As a consequence, the policy evaluation step for each of the tasks can use all the transition experience ($\{(s, a, s')\}$) generated by all tasks and can be augmented by a reward prediction step that will try to build a projected model of the reward for each task.

The second approach to policy evaluation we investigated in this chapter is minimising the Bellman residual (Eq. 4.5). In this case, inspired by the positive results obtained in chapter 3, we explore learning a common representation across the multiple policy evaluation steps under consideration. In doing so, we aim to leverage the different data collected by different tasks and any commonalities that the policies might share. We proposed two instances of the joint evaluation step: one that looks at multiple policies under the same task (**MTBRM-PR**) and one that only looks at the evaluation problems we are actually interested in, the evaluations of a particular policy π_j under its task j (**MTBRM**). Although both of these algorithms are targeting estimating the optimal value functions at convergence, they will tackle the (joint) policy evaluation steps very differently. Notably, the multitask problems solved and the resulting shared representations will be different. One, **LSPI-MTBRM**, will be learning a representation that supports the estimation of optimal value functions across the class of MDP considered, much like how we did in Section 3.1 – in a sense, this is just a different way of constructing a similar representation (Section 3.3). The other one, **LSPI-MTBRM-PR**, will build, for each policy, a shared representation across multiple reward signals; thus supporting in a sense this policy's evaluation under different reward structures. A complementary idea has been recently explored in (Dadashi et al., 2019; Bellemare et al., 2019). Since the nature of this problem is different so is the generalisation they are targeting: one is across MDPs and (optimal) policies, the for a particular policy, across MDPs. For this work, we used the same multitask algorithm (Argyriou et al., 2008) as in chapter 3, but as with the previous chapter, this is plug-in choice and other multitask regression methods can be utilised here.

As we mentioned in the main text, in this work we have focused mainly on the benefits that come out of a joint learning procedure and the transfer enabled across the multiple policy evaluation steps. Nevertheless, one could also think of using the other tasks, and in particular the other policies, in the improvement steps as well. This is especially compatible with algorithms Alg. 4.2 and Alg. 4.4, where we can easily get the evaluations of all policies considered at this iteration with respect to all tasks, thus enabling us to perform a potentially larger improvement step via Generalised Policy Improvement (Theorem 5.1), as we will explore further in the next chapters. Intuitively, this would be possible if any of the current policies surpasses as any state the performance of its optimising policy. Although this advantage might be of limited use further into the policy iteration procedure – as we expect policies learnt to optimise for that task to dominate these evaluations. It might be much more beneficial in the early phases of training and maybe more importantly in the scenario in which we consider tackling a new task in our MDP class. This kind of transfer will immediately provide us with a good starting point. We refrain from more details at this point, as we will delve into these topics in the next chapters (Section 5.1.2 - 5.2), as part of a different policy evaluation scheme via successor features (Barreto et al., 2017).

Part II

Transfer in Policy Space

5
Scalable Policy-based Transfer for Deep Reinforcement Learning via Successor Features

In the previous part, we have looked into learning a common representation which then can be used to more easily build value functions for future tasks. The intuition there came from the evidence provided by examples in supervised transfer learning literature showing tasks may benefit from a joint representation both in prediction performance but also in better generalising to a related task in the same/similar family. We have seen previously that some of these intuitions can be brought over to RL tasks. In particular, we have shown how one could adapt multitask supervised techniques to jointly learn a transferable common representation for value functions corresponding to different control problems. As a reminder, in this work, we have been and will be looking at control problems that differ in reward specification only, but otherwise maintain state and action specifications as well as observe a constant dynamics of the environment. Although some of the methods presented in Chapter 2 are widely applicable even outside this shared dynamics assumption, we have seen in Chapter 3 that we can do better by taking advantage of these assumptions and exploiting them to get reuse samples $s' \sim \mathcal{P}(.|s, a)$, increasing our sample efficiency at transfer. In this part of the thesis, we will take a close look at explicitly leveraging the shared dynamics in the tasks considered. At the same time, we will be relaxing slightly one of the constraints on the common representation we will be learning. We will show that relaxing the linearity constraint and moving instead towards building a policy-based representation leads to more scalable, practical algorithms that pair well with non-linear, state-of-the-art, neural networks-based representation learning and more principled transfer opportunities.

Why policy-based representations? Policies are a central and key concept in any control or prediction problem in an MDP. They fully describes the behaviour of an agent and highly impact the quantities we might want to estimate or questions we might want to answer. For instance, the prediction problem of where our agent is going to be in T steps is usually dependent not only on the transition kernel but also on the policy this agent is following: $\mathcal{P}^{\pi}(s_{t+T}|s_t)$. And the complexity of this prediction problem is implicitly linked to the complexity of π. Here we are assuming a long term sequential learning problem, in which the agent can and will influence the environment and the distribution of the states it will visit through its actions. As such, the trajectories an agent can observe in the same environment can be very different under different policies, π_i: $P^{\pi_1}(\{s_{t+1}, \ldots, s_{t+T}\}|s_t) \neq P^{\pi_2}(\{s_{t+1}, \ldots s_{t+T}\}|s_t)$. Nevertheless, as the transition kernel $\mathcal{P}$ does not change, π_1 and π_2 being close to each other (in the sense that their selection differs only slightly) will likely imply the induced prediction problems are closely related as well. Although one could easily construct examples where even one simple deviation in the policy can lead to very different futures, we would argue that in general, a measure on π induces a measure on $\mathcal{P}^{\pi}$ and respectively on the corresponding predictions and value functions V^{π}(or Q^{π}). More precisely, the assumption is that common structure in one space would imply structure in the induced prediction problems.

The concept of policy and the dependency of our estimates on it is very specific to RL – not present in supervised or unsupervised learning problems, at least not stationary ones. In the following chapters, we will start to investigate some of the peculiarities present in transfer problems in reinforcement learning tasks – in particular, the reusable structure present in policy space. This type of structure becomes primarily relevant if we approach new control problems or complex planning tasks. We first try to break down the problem into manageable sub-tasks that we have experience solving or for which we have already 'cached in' solutions. In other words, we devise a plan relying on transferring previously learnt skills or partial policies and applying them, whenever appropriate, to a new situation.

In the literature, this kind of hierarchical decision making based on subskills (Dietterich, 2000; Konidaris and Barto, 2007), sometimes called options (Sutton et al., 1999; Precup, 2000), has been thoroughly investigated and shown to be a very effective way of planning at a higher temporal horizon (Brunskill and Li, 2014). Nonetheless defining – or discovering – a collection of re-usable subskills that can be used effectively for planning remains largely an elusive open question (Hengst, 2017; Barto and Mahadevan, 2003; Diuk and Littman, 2009; Konidaris and Barto, 2007; Brunskill and Li, 2014). Setting this option discovery problem aside, for the time being, it should become clear by now that to attempt to reproduce this kind of functionality and transferability in new control problems, our agents should display some core capabilities:

1. The ability to *lean to solve multiple tasks* and remember or *store these partial solutions*. We have already argued for this in the past chapters and the focus of this work has been largely dedicated to investigating precisely this multitask learning problem.

2. The ability to devise *a plan using these learnt subpolicies*, in the context of a new task. This includes the ability to invoke past policies whenever they are appropriate and decide which of the available subskills are best suited for the task at hand.

The second point here implies that the ability to transfer knowledge from previous tasks in RL problems can be multifaceted, exploiting different kinds of structure. In particular, in the first chapters, we have focused on learning common features based on which we can estimate multiple (optimal) value functions. These exploit the common observational space shared by the tasks and can *implicitly* capture some of the structure induced by the shared dynamics and smoothness in policy space. In the following chapters, we will try to capture this structure more *explicitly* and argue for transfer via re-using previous policies for planning. We will also show that being able to reason explicitly about multiple policies at the same time and what they would achieve under different tasks can lead to very effective transfer in policy space. This ability to evaluate multiple policies will naturally provide us with a mechanism to decide which previously learnt policies are pertinent to use at any point in time and when to trust one versus the other. It is worth noting that, in general, building such evaluation under different tasks can be very expensive and not necessarily simpler than solving the new RL problem from scratch. Nevertheless, under the same dynamics, this problem becomes much simpler as the futures we are trying to learn about stay the same under a given policy π. Thus, switching to a new task (new reward signal) boils down to reward re-evaluation under a fixed transition kernel $\mathcal{P}^{\pi}$. Fortunately for us, there exists a particular policy-based representation, called Successor Representations (Dayan, 1993), that can do this re-evaluation instantaneously. In Section 5.1, we formally introduce these representations and their feature generalisations, called Successor Features (Barreto et al., 2017) - SFs in short, and discuss their potential for transfer.

As previously mentioned, these new representations are policy-dependent. On the one hand, this allows us to exploit the structure in the dynamic programming problems more effectively; on the other hand, we have to build these representations for each of the policies we would be interested in evaluating. Unfortunately, the space of all possible policies one could consider in a given environment scales with the number of states and actions. Thus answering even (prediction) questions like the one above for an arbitrary policy π can easily become prohibitively hard, let alone the problem of coming up with a suitable control policy for combining previous ones. Chapter 6 will propose an elegant and scalable solution to this problem, where based on a given structure in the reward space, we can learn off-policy and off-task to build SFs that generalise over a policy embedding space, closely linked to our reward space. In this chapter, we will be looking at how to discover and exploit additional structure in reward space. We will show how by relating our current task to the previously observed reward signals we can provide the same level of transfer as one could if the underlying structure is observed or provided by the user. Before moving further, we introduce two (mild) assumptions that will make the transfer problem less intractable if we can discover and leverage this structure:

- We will *restrict the space of policies* we are interested in representing or answer questions about. In this work, we specify this subset of policies by instead specifying the subspace of reward signals we are interested in generalising over – the space of tasks we are trying to handle. We will also argue that in order to deal with this family of rewards, we only need to concern ourselves with policies that are trying to optimise one or a combination of these reward signals. There is ample evidence in the literature that the structure induced by optimal policies (Schaul et al., 2015a) across a selection of tasks can be much more compact, capturing only the essential information needed to generalise to other optimal behaviours: e.g. the shortest path between two states.

- As already hinted above, the second assumption will be that the induced space of value functions, under the above set of policies, *shares a common structure* that can be learnt and leveraged for generalisation from one policy to the other. Note that this is bound to be true under the shared dynamics assumption. Furthermore, what we will show in the next chapter is that this property could be further enhanced by discovering and exploiting the structure presented in reward space – if any.

5.1 Background: Successor Representations and Features

In this section, we will define Successor Representations (SRs) and their extension Successor Features and see how one could use such representations for transfer (Barreto et al., 2017). Successor representations were introduced by Dayan (Dayan, 1993) as a way to improve generalisation for temporal difference methods within one task. This original version relied mostly on 'transferring knowledge' between different policies occurring at different value iteration steps. SRs *are defined as the expected discounted number of visits to a particular future state, starting from a given initial state, under a policy π.*

More recently, in (Barreto et al., 2017), Barreto et al proposed a generalisation of these representations in a feature space that can potentially deal with a much larger state space, including the continuous case as well as functional approximation. The next two chapters will be about extending and generalising this framework. As a reminder, we are interested in generalising for a class of MDPs that share the state and action space, as well as the transition dynamics, but differ in the reward functions which solely specify the tasks of interest:

$$\mathcal{M}(\mathcal{S},\mathcal{A},\mathcal{P},\gamma) \equiv \{M(\mathcal{S},\mathcal{A},\mathcal{P},\cdot,\gamma)\}. \tag{5.1}$$

Specifically, the restricted scenario considered in (Barreto et al., 2017) puts an additional constraint on the reward signals one can expect to see. They looked at MDPs whose expected one-step reward can be written as

$$r(s,a,s') = \boldsymbol{\varphi}(s,a,s')^\top \mathbf{w}, \tag{5.2}$$

where $\boldsymbol{\varphi}(s,a,s') \in \mathbb{R}^d$ are features of (s,a,s') and $\mathbf{w} \in \mathbb{R}^d$ are weights. In general $\boldsymbol{\varphi}(s,a,s')$ can be any abstract function of (s,a,s'), but in order to build some intuition it helps to think of them as salient events, potentially rewarding events, that may be desirable or undesirable to the agent. Based on (5.2) one can define an environment $(\mathcal{S},\mathcal{A},\mathcal{P},\gamma)$ as

$$\mathcal{M}^{\varphi} \equiv \{M(\mathcal{S},\mathcal{A},\mathcal{P},r,\gamma) | r(s,a,s') = \boldsymbol{\varphi}(s,a,s')^\top \mathbf{w}\}, \tag{5.3}$$

that is, $\mathcal{M}^{\varphi}$ is the set of MDPs induced by $\boldsymbol{\varphi}$ through all possible instantiations of $\mathbf{w}$. We call each $M \in \mathcal{M}^{\varphi}$ a *task*. Given a task $M_i \in \mathcal{M}^{\varphi}$ defined by $\mathbf{w}_i \in \mathbb{R}^d$, we will use Q_i^{π} to refer to the value function of π on M_i. Now looking at the value function Q^{π} of task w under policy π one can observe the following decomposition:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}^{\pi}\left[\sum_k \gamma^t r_{t+k} | S_t = s, A_t = a\right] = \mathbb{E}^{\pi}\left[\sum_k \gamma^t \varphi_{t+k}^T \mathbf{w} | S_t = s, A_t = a\right] \\ &= \mathbb{E}^{\pi}\left[\sum_k \gamma^t \varphi_{t+k}^T | S_t = s, A_t = a\right] \mathbf{w} = \psi^{\pi}(s,a)^T \mathbf{w} \end{aligned}$$

And from hereon we will refer to $\psi^{\pi}(s,a)$ as the successor feature(SF) at (s,a) under policy π. Thus a SF, ψ^{π}, is a vector of policy evaluation value functions under different reward signals corresponding to the different dimensions of φ. And as one can easily see from the equation above, this representation captures the sufficient information needed to re-evaluate any reward signal in the $\mathcal{M}^{\varphi}$. Moreover, it is also easy to show that

$$\boldsymbol{\psi}^{\pi}(s,a) = \mathrm{E}^{\pi}[\boldsymbol{\varphi}_{t+1} + \gamma \boldsymbol{\psi}^{\pi}(S_{t+1}, \pi(S_{t+1})) \mid S_t = s, A_t = a], \tag{5.4}$$

which means SFs satisfy a Bellman equation in which φ_i play the role of rewards – one can easily check this corresponds to the Bellman Expectation Equations for policy π and rewards given by each entry in φ. Therefore, SFs can be learned using any conventional RL method (Szepesvári, 2010; Sutton and Barto, 1998).

5.1.1 How General are ψ^{π}?

Given these very appealing properties – zero-shot inference of policy evaluations over a collection of tasks and suitability to TD-based learning methods – the question one might ask is what are we giving away here or essentially how general are these representations? And the answer is *very tightly linked to the choice* of $\varphi(s,a)$. To provide some intuition let us take a look at two particular scenarios, at different ends of the spectrum.

For instance, if S and A were both finite and $\varphi(s,a,s')$ is a $|\mathcal{S}|^2|\mathcal{A}|$ dimensional one-hot vector, ψ^{π} is the discounted sum of occurrences of this particular transition (s,a,s'), under policy π. This not only recovers the original SRs but also this representation is able to handle any reward function in the extended class $\mathcal{M}$, as any reward signal r can be trivially expressed as a linear combination in this tabular representation, where $\mathbf{w} \in \mathbb{R}^{|\mathcal{S}|^2|\mathcal{A}|}$ is equal to r: $\mathbf{w} = r$. However, coverage of the $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is simply not feasible for most interesting problems. Nevertheless, as argued in (Barreto et al., 2017; Dayan, 1993), our rewards are usually somewhat sparse and so is the space of states we can reach in one step. This implies that the number of features φ that we would need to represent can be substantially less than $|\mathcal{S} \times \mathcal{A} \times \mathcal{S}|$.

On the other hand, if the reward signal in the current task is not well represented in φ this can lead to wrong evaluations – that could be either non-informative, but also misleading. And as we will be using these evaluation for decision making – as we will outline in the next section – this could lead to very poor decisions. A particular case of this scenario is when the reward signal is orthogonal to φ in which case, all of our evaluations will be zero and thus we would have no knowledge that we can transfer from previous behaviour nor any way to distinguish between their applicability ('goodness') in this new task.

5.1.2 Transfer via SFs and GPI

Let us take a look at learning and generalising to tasks in $\mathcal{M}^{\varphi}$. Remember the goal here is to show how learning about a subset of the tasks in $\mathcal{M}^{\varphi}$ can help us come up with a policy suitable for a new, unseen task from the same family. (Barreto et al., 2017) propose SF& GPI as a way to promote transfer between tasks in $\mathcal{M}^{\varphi}$. As the name suggests, GPI is a generalisation of the policy improvement step described in Section 5.1. The difference is that in GPI the improved policy is computed based on a *set* of value functions rather than on a single one. In words, GPI implies that, if the agent acts greedily with respect to the maximum over a set of value functions, the resulting policy is no worse than the policies that originated the value functions. More formally, let $Q^{\pi_1}, Q^{\pi_2}, \ldots Q^{\pi_n}$ be the action-value functions of n policies defined on a given MDP, and let $Q^{\max} = \max_i Q^{\pi_i}$. If we define

$$\pi_{GPI}(s) \in \operatorname{argmax}_a Q^{\max}(s, a), \forall s \in \mathcal{S} \tag{5.5}$$

then $Q^{\pi_{GPI}}(s, a) \geq Q^{\max}(s, a)$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. The result also extends to the scenario where we replace Q^{π_i} with approximations $\tilde{Q}^{\pi_i}$, in which case the lower bound on $Q^{\pi_{GPI}}(s, a)$ gets looser with the approximation error, as in approximate DP (Bertsekas and Tsitsiklis, 1996). For the convenience of the reader, we restate (Barreto et al., 2017) GPI theorem below and we include the proof in the Appendix A.3.1.

Theorem 5.1: Generalised Policy Improvement

Let $\pi_1, \pi_2, ..., \pi_n$ be n decision policies and let $\tilde{Q}^{\pi_1}, \tilde{Q}^{\pi_2}, ..., \tilde{Q}^{\pi_n}$ be approximations of their respective action-value functions such that

$$|Q^{\pi_i}(s,a) - \tilde{Q}^{\pi_i}(s,a)| \leq \varepsilon \text{ for all } s \in S, a \in A, \text{ and } i \in \{1,2,\ldots,n\}.$$

Define

$$\pi(s) \in \operatorname*{argmax}_a \max_i \tilde{Q}^{\pi_i}(s,a).$$

Then,

$$Q^{\pi}(s,a) \geq \max_i Q^{\pi_i}(s,a) - \frac{2}{1-\gamma}\varepsilon$$

for any $s \in S$ and any $a \in A$, where Q^{π} is the action-value function of π.

In the context of transfer, GPI has been shown in (Barreto et al., 2017) to leverage knowledge accumulated over time, across multiple tasks, to learn a new task faster. Suppose that the agent has access to n policies $\pi_1, \pi_2, \ldots, \pi_n$. These can be arbitrary policies, but for the sake of argument, let us assume they are solutions for tasks $M_1, M_2, ..., M_n$. Suppose also that when exposed to a new task $M_{n+1} \in \mathcal{M}^{\varphi}$ the agent computes $Q^{\pi_i}_{n+1}$—the value functions of the policies π_i under the reward function induced by $\mathbf{w}_{n+1}$. In this case, applying GPI to the set $\{Q^{\pi_1}_{n+1}, Q^{\pi_2}_{n+1}, \ldots, Q^{\pi_n}_{n+1}\}$ will result in a policy that performs at least as well as any of the policies π_i, even if one is allowed to choose a *different* baseline policy π_i in each $s \in \mathcal{S}$.

Clearly, the approach above is appealing only if we have a way to quickly compute the value functions of the policies π_i on the task M_{n+1}. This is where SFs come in handy. SFs make it possible to compute the value of a policy π on *any* task $M_i \in \mathcal{M}^{\varphi}$ by simply plugging in the representation the vector $\mathbf{w}_i$ defining the task. Specifically, if we substitute (5.2) in the definition of the action-value function we have

$$\begin{aligned} Q^{\pi}_i(s,a) &= \mathrm{E}^{\pi}\left[\textstyle\sum_{i=t}^{\infty} \gamma^{i-t} \boldsymbol{\varphi}_{i+1} \,|\, S_t = s, A_t = a\right]^{\top} \mathbf{w}_i \\ &\equiv \boldsymbol{\psi}^{\pi}(s,a)^{\top} \mathbf{w}_i, \end{aligned} \tag{5.6}$$

where $\boldsymbol{\varphi}_t = \boldsymbol{\varphi}(s_t, a_t, s_{t+1})$ and $\boldsymbol{\psi}^{\pi}(s,a)$ are the SFs of (s,a) under policy π. As one can see, SFs decouple the dynamics of the MDP M_i from its rewards (Dayan, 1993). One benefit of doing this is that if we replace $\mathbf{w}_i$ with $\mathbf{w}_j$ in (5.6) we immediately obtain the evaluation of π on task M_j. In other words, if we switch to a new MDP in this family, we can readily evaluate π on this new MDP.

The combination of SFs and GPI provides a general framework for transfer in environments of the form (5.3). Suppose that we have learned the functions $Q_i^{\pi_i}$ using the representation scheme (5.6). When exposed to the task defined by $r_{n+1}(s, a, s') = \boldsymbol{\phi}(s, a, s')^\top \mathbf{w}_{n+1}$, as long as we have $\mathbf{w}_{n+1}$ we can immediately compute $Q_{n+1}^{\pi_i}(s, a) = \boldsymbol{\psi}^{\pi_i}(s, a)^\top \mathbf{w}_{n+1}$. This reduces the computation of all $Q_{n+1}^{\pi_i}$ to the problem of determining $\mathbf{w}_{n+1}$, which can be posed as a supervised learning problem whose objective is to minimise some loss derived from (5.2). Once $Q_{n+1}^{\pi_i}$ have been computed, we can apply GPI to derive a policy π that is no worse, and possibly better, than $\pi_1, \ldots, \pi_n$ on task M_{n+1}.

5.2 Extending the Notion of Environment

As reviewed above, (Barreto et al., 2017) proposed a framework for transfer based on two ideas: generalised policy improvement (GPI), a generalisation of the classic dynamic-programming operation, and successor features (SFs), a representation scheme that makes it possible to quickly evaluate a policy across many tasks. This approach to transfer is appealing for two reasons: it allows transfer to take place between any two tasks, regardless of their temporal order, and it integrates almost seamlessly within the RL framework.

In this work we extend this framework in two ways. First, we will argue that its applicability is broader than initially shown. SF&GPI was designed for the scenario where each task corresponds to a different reward function; one of the basic assumptions in the original formulation was that the rewards of all tasks can be computed as a linear combination of a fixed set of features. We show that such an assumption is not strictly necessary, and in fact it is possible to have guarantees on the performance of the transferred policy even on tasks that are not in the span of the features.

The realisation above adds some flexibility to the problem of computing features that are useful for transfer. Specifically, by looking at the associated approximation from a slightly different angle, we show that one can replace the features with actual rewards. This makes it possible to apply SF&GPI online at scale, bypassing, in a sense, the discovery problem while maintaining the same level of generality.

Previous work focused on environments of the form (5.3). In this work we will adopt a more general notion of environment and the changes in the reward signal that can occur:

$$\mathcal{M}(\mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma) \equiv \{M(\mathcal{S}, \mathcal{A}, \mathcal{P}, \cdot, \gamma)\}. \tag{5.7}$$

$\mathcal{M}$ contains *all* MDPs that share the same $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$, and γ, regardless of whether their rewards can be computed as a linear combination of the features $\boldsymbol{\varphi}$. Clearly, $\mathcal{M} \supset \mathcal{M}^{\varphi}$. Now our goal is to devise a transfer framework for environment $\mathcal{M}$.

As reviewed in the first section of this chapter, (Barreto et al., 2017) provides theoretical guarantees on the performance of SF&GPI applied to any task $M \in \mathcal{M}^{\varphi}$. In this section we show that one can generalise slightly this result and derive guarantees for any task in $\mathcal{M}$. We will do so by considering projection of the reward signals in $\mathcal{M}$ onto $\mathcal{M}^{\varphi}$. In some sense, we are going to approximate a general reward function with its closest point in $\mathcal{M}^{\varphi}$and show that taking the corresponding GPI policy for the approximation can still lead to a good policy for the original task. In the following, we formally related the optimal value function of an arbitrary reward signal r and the performance of the GPI policy induced by its 'closest' point in $\mathcal{M}^{\varphi}$. The general result is stated below:

Proposition 5.1

Let $M \in \mathcal{M}$ (MDP we are trying to generalise to) and let $Q_i^{\pi_j^*}$ be the action-value function of an optimal policy of $M_j \in \mathcal{M}$, π_j^*, when executed in an $M_i \in \mathcal{M}$. Given approximations $\{\tilde{Q}_i^{\pi_1}, \tilde{Q}_i^{\pi_2}, \ldots, \tilde{Q}_i^{\pi_n}\}$ such that $\left|Q_i^{\pi_j^*}(s,a) - \tilde{Q}_i^{\pi_j}(s,a)\right| \leq \varepsilon$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, and $j \in \{1, 2, \ldots, n\}$, let

$$\pi(s) \in \text{argmax}_a \max_j \tilde{Q}_i^{\pi_j}(s,a). \tag{5.8}$$

Then,

$$\|Q^* - Q^{\pi}\|_{\infty} \leq \frac{2}{1-\gamma}\left(\|r - r_i\|_{\infty} + \min_j \|r_i - r_j\|_{\infty} + \varepsilon\right), \tag{5.9}$$

where Q^* is the optimal value of M, Q^{π} is the value function of π in M, and $\|f - g\|_{\infty} = \max_{s,a} |f(s,a) - g(s,a)|$.

Proof. The result is a direct application of Theorem 5.1 and intermediate results Lemmas 2 and 3 present in Appendix A.3.1. Let π^* be an optimal value function of M. Then,

$$\begin{aligned}
Q^*(s,a) - Q^\pi(s,a) &= Q^{\pi^*}(s,a) - Q^\pi(s,a) \\
&= Q^{\pi^*}(s,a) - Q_i^{\pi_i^*} + Q_i^{\pi_i^*} - Q^\pi(s,a) \\
&= Q^{\pi^*}(s,a) - Q_i^{\pi_i^*} + Q_i^{\pi_i^*} - Q_i^\pi + Q_i^\pi - Q^\pi(s,a) \\
&\leq |Q^{\pi^*}(s,a) - Q_i^{\pi_i^*}| + Q_i^{\pi_i^*} - Q_i^\pi + |Q_i^\pi - Q^\pi(s,a)|
\end{aligned}$$

From Lemma 3, we know that

$$|Q^{\pi^*}(s,a) - Q_i^{\pi_i^*}| \leq \frac{\max_{s,a} |r(s,a) - r_i(s,a)|}{1-\gamma}.$$

From Theorem 5.1 we know that, for any $j \in \{1, 2, \dots, n\}$, we have

$$\begin{aligned}
Q_i^{\pi_i^*}(s,a) - Q_i^\pi(s,a) &\leq Q_i^{\pi_i^*}(s,a) - Q_i^{\pi_j^*}(s,a) + \frac{2}{1-\gamma}\varepsilon && \text{(Theorem 5.1)} \\
&= Q_i^{\pi_i^*}(s,a) - Q_j^{\pi_j^*}(s,a) + Q_j^{\pi_j^*}(s,a) - Q_i^{\pi_j^*}(s,a) + \frac{2}{1-\gamma}\varepsilon \\
&\leq |Q_i^{\pi_i^*}(s,a) - Q_j^{\pi_j^*}(s,a)| + |Q_j^{\pi_j^*}(s,a) - Q_i^{\pi_j^*}(s,a)| + \frac{2}{1-\gamma}\varepsilon \\
&\leq \frac{2}{1-\gamma} \max_{s,a} \left| r_i(s,a) - r_j(s,a) \right| + \frac{2}{1-\gamma}\varepsilon && \text{(Lemmas 2, 3 in Appendix).}
\end{aligned} \tag{5.10}$$

Finally, from Lemma 2, we know that

$$|Q_i^\pi - Q^\pi(s,a)| \leq \frac{\max_{s,a} |r(s,a) - r_i(s,a)|}{1-\gamma}.$$

□

Our result provides guarantees on the performance of the GPI policy (5.8) applied to an MDP M with *arbitrary* reward function $r(s,a)$. Note that, although the theorem does not restrict any of the tasks to be in $\mathcal{M}^\varphi$, in order to compute the GPI policy (5.8) we still need an efficient way of evaluating the policies π_j on task M_i. As explained in Section 5.1, one way to accomplish that is to assume that M_i and all M_j appearing in the statement of the theorem belong to $\mathcal{M}^\varphi$; this allows us to use SFs to quickly compute $\tilde{Q}_i^{\pi_j}(s,a)$.

To build some intuition of the implications of this result, let us take a closer look at Proposition 5.1 under the assumption that all MDPs belong to $\mathcal{M}^\varphi$for some φ, except perhaps for M. Our application of this proposition relies on a reference MDP $M_i \in \mathcal{M}^\varphi$. One should think of M_i as the MDP in $\mathcal{M}^\varphi$ that is "closest" to M in some sense. Specifically, if we define $r_i(s,a,s') = \boldsymbol{\varphi}(s,a,s')^\top \mathbf{w}_i$, we can think of $\mathbf{w}_i$ as the vector that provides the best approximation $\boldsymbol{\varphi}(s,a,s')^\top \mathbf{w}_i \approx r(s,a,s')$ under some well-defined criterion. The first term of (5.9) can thus be seen as the "distance" between M and $\mathcal{M}^\varphi$, which suggests that the performance of the GPI policy based on this approximation r_i should degrade gracefully as we move away from the original set $\mathcal{M}^\varphi$. Also note that in the particular case where $M \in \mathcal{M}^\varphi$ this first term of (5.9) vanishes, and we recover Theorem 2 in (Barreto et al., 2017).

The two remaining terms in the bound quantify the approximation error ε incurred by our value estimates and GPI's ability to generalise within $\mathcal{M}^\varphi$. In particular, this remaining term could be minimised if $r_i = r_j$, for some $j \in \{1, 2, \cdots n\}$. In this case, we would be effectively looking for which of the previous tasks is most similar to our current one. Once this task is identified, one could, in principle, try to employ its corresponding learnt policy and see how well it does on our current task. Note that if this policy is not already optimal for r_i, by considering instead the GPI policy, as the proposition proposes, we would be doing something more general that has the potential to further improve this policy by bringing knowledge from all other policies in our collection. This is a particular way in which knowledge from later tasks can readily incorporated when revisiting a task.

5.2.1 Uncovering the Structure of the Environment

In this work we are interested in the transfer scenario. Thus ideally we would want to solve a subset of the tasks in $\mathcal{M}$ and use GPI to promote transfer between these tasks. As argued in the previous section in order to do so efficiently, one would benefit from access to a collection of features $\boldsymbol{\varphi}$ that cover the tasks of interest as much as possible. If we were to rely on (Barreto et al., 2017) original results, we would have guarantees on the performance of the GPI policy only if $\boldsymbol{\varphi}$ spanned *all* the tasks of interest. Proposition 5.1 allows us to remove this requirement, since now we have theoretical guarantees for any choice of $\boldsymbol{\varphi}$. In practice, though, we want features $\boldsymbol{\varphi}$ such that the first term of (5.9) is small for all tasks of interest.

There might be contexts in which we have direct access to features $\boldsymbol{\varphi}(s,a,s')$ that satisfy (5.2), either exactly or approximately. And we will be exploring this particular case in the next chapter, where we propose a model that can very efficiently capitalize on this known structure. Nevertheless, in this chapter we propose tackling the more general scenario where this structure is not available, nor given to the agent in any form. In this case the use of SFs requires a way of unveiling such a structure in order to ground the learning for our value functions. If we assume the latent existence of a $\boldsymbol{\varphi} \in \mathbb{R}^d$ that satisfy (5.2) exactly, we can formulate the problem of computing an approximate $\tilde{\boldsymbol{\varphi}}$ as a supervised multi-task learning problem. The problem is decomposed into D regressions, each one associated with a task. For reasons that will become clear shortly, we will call these tasks *base tasks* and denote them by $\mathcal{B} \equiv \{M_1, M_2, \ldots, M_D\} \subset \mathcal{M}^{\varphi}$. The multi-task problem thus consists in solving the approximations:

$$\tilde{\boldsymbol{\varphi}}(s, a, s')^\top \tilde{\mathbf{w}}_i \approx r_i(s, a, s'), \text{ for } i = 1, 2, \ldots, D, \tag{5.11}$$

where r_i is the reward of M_i (Caruana, 1997; Baxter, 2000).

In this section we argue that (5.11) can be replaced by a much simpler approximation problem by simply rotating the problem. Suppose for a moment that we know a function $\boldsymbol{\varphi}$ and D vectors $\mathbf{w}_i$ that satisfy (5.11) exactly. If we then stack the vectors $\mathbf{w}_i$ to obtain a matrix $\mathbf{W} \in \mathbb{R}^{D\times d}$, we can write $\mathbf{r}(s, a, s') = \mathbf{W}\boldsymbol{\varphi}(s, a, s')$, where the ith element of $\mathbf{r}(s, a, s') \in \mathbb{R}^D$ is $r_i(s, a, s')$. Now, as long as we have d linearly independent tasks $\mathbf{w}_i$, we can write

$$\boldsymbol{\varphi}(s, a, s') = (\mathbf{W}^\top \mathbf{W})^{-1} \mathbf{W}^\top \mathbf{r}(s, a, s')$$

Since $\boldsymbol{\varphi}$ is given by a linear transformation of $\mathbf{r}$, any task representable by the former can also be represented by the latter. To see why this is so, note that for any task in $\mathcal{M}^{\varphi}$ we have

$$r(s, a, s') = \mathbf{w}^\top \boldsymbol{\varphi}(s, a, s') = \mathbf{w}^\top \left[(\mathbf{W}^\top \mathbf{W})^{-1} \mathbf{W}^\top \right] \mathbf{r}(s, a, s') = (\tilde{\mathbf{w}})^\top \mathbf{r}(s, a, s').$$

Therefore, we can use the rewards $\mathbf{r}$ themselves as features, which means that we can replace (5.11) with the much simpler approximation:

$$\tilde{\boldsymbol{\varphi}}(s, a, s') = \tilde{\mathbf{r}}(s, a, s') \approx \mathbf{r}(s, a, s'). \tag{5.12}$$

One potential drawback of directly approximating $\mathbf{r}(s, a, s')$ is that we no longer have the flexibility of distinguishing between d, the dimension of the environment $\mathcal{M}^{\varphi}$, and D, the number of base tasks in $\mathcal{B}$. Since in general we will solve (5.11) or (5.12) based on data, having $D > d$ may lead to a better approximation. As the number of base tasks D increases, we might be overparameterising the above regression problem. As D becomes larger than the intrinsic dimensionality of the task space d, a new reward signal can be represented in a non-unique way. Nevertheless, due to linearity, all of these solutions will be equivalent both in representing the rewards and corresponding value functions associated with the policies $\{\pi_i\}_{i=1,D}$.

On the other hand, using $\tilde{\mathbf{r}}(s, a, s')$ as features has a number of advantages that in many scenarios of interest should largely outweigh this loss in flexibility. One such advantage becomes clear when we look at the scenario above from a different perspective. If we assume that there exists indeed a set of $\boldsymbol{\varphi}$ for which (5.11) is satisfied, then given any set of D tasks with associated reward signals $r_i(s, a, s')$, $k_D \leq D$ of which are linearly independent, our new set of features $\tilde{\boldsymbol{\varphi}}$ will span a k_D dimensional subset of the original, unknown space $\mathcal{R}^{\varphi}$. If $k_D = d \leq D$, we recover the whole original space $\mathcal{R}^{\varphi}$. Thus, instead of assuming we know a set of $\boldsymbol{\varphi}$ and a $\mathbf{W}$ that satisfy (5.11), we can simply use the approximate rewards $\tilde{\boldsymbol{\varphi}}$ ($\tilde{W} = I_D$) and apply the above reasoning without modification. This highlights a benefit of replacing (5.11) with (5.12): the fact that we can work directly in the space of tasks, which in many cases may be more intuitive. When we think of $\boldsymbol{\varphi}$ as a non-observable, abstract, quantity, it can be difficult to define what the base tasks should be. In contrast, when we work on the reward signal $\mathbf{r}$ directly the question we are trying to answer becomes: is it possible to approximate the future reward function of all tasks of interest as a linear combination of the previous reward functions r_i? One can see that in this view, the set $\mathcal{B}$ acts as a basis for future tasks, and thus the name "base tasks".

Another interesting consequence of using an approximation $\tilde{\boldsymbol{\varphi}}(s, a, s') \approx \mathbf{r}(s, a, s')$ as features is that the resulting SFs are nothing but ordinary action-value functions. Specifically, the j$^{\text{th}}$ component of $\tilde{\boldsymbol{\psi}}^{\pi_i}(s, a)$ is simply $\tilde{Q}_j^{\pi_i}(s, a)$. Next we discuss how this gives rise to a simple approach that can be combined with deep learning in a stable way.

5.3 Transfer in Deep Reinforcement Learning

In this work, we are interested in using SF&GPI to build scalable agents that can be combined with state-of-the-art deep learning-based state representations in a stable way. Since deep learning generally involves vast amounts of data whose storage is impractical, we will mostly be focusing on methods that *work online*. Thus, our second contribution is to show a simple and natural way of using the previous insights to propose a scalable, online method of transferring knowledge from previously learnt tasks. The proposed method attempts transfer in two ways: 1) through a shared representation (as in previous chapters), but now this is a learnt non-linear representation which the SFs are built upon and these mappings are not assumed linear; 2) through GPI on a collection of approximate policy evaluations based on previously learnt policies. Note that these two methods of transferring knowledge exploit commonalities in different spaces: the first one takes advantage of the shared sensory/observational space and the other one exploits the structure present in the dynamic programming nature of the problem and the policy space shared by the multiple control problems considered. Thus, there is reason to believe these can be complementary and leveraging both could further increase our sample efficiency in a new task.

In order to verify this claim, we revisit one of (Barreto et al., 2017) experiments in a much more challenging format, replacing a fully observable 2-dimensional environment with a 3-dimensional domain where observations are images from a first-person perspective. We show that the transfer promoted by SF&GPI leads to reasonable policies on unseen tasks almost instantaneously. We do so by computing a zero-shot transfer policy based on a guess of what the current task entails. This estimate will then be *refined online* as we get more data about the current task – the more rewards the agent experiences, the more it can refine this estimate. Furthermore, we will show that this GPI policy can be a good starting point for learning a more specialised policy for the new task. Lastly, we show how to learn these specilised policies in a way that allows them to be added to our agent's ever-growing set of skills, a crucial ability for continual learning (Thrun, 1996).

To make this more precise and as a motivating example, we show in Algorithm 5.1 how SFs and GPI can be used for acting in a new task and combined with Q-learning (Watkins and Dayan, 1992) to extend the collection of SFs 'cached in' for transfer.[1] This algorithm assumes we have already obtained a collection of features $\tilde{\boldsymbol{\phi}}$ and a corresponding set of SFs $\tilde{\boldsymbol{\psi}}$ built on top of these. Given a new task, we are going to estimate $\tilde{\mathbf{w}}$ to approximate the new reward signal as a linear combination of $\tilde{\boldsymbol{\phi}}$. Given this estimate we have two choices: act according to the GPI policy induced by $(\tilde{\boldsymbol{\psi}}, \tilde{\mathbf{w}})$ or try to build a specialised policy π_{n+1} for this task and its SFs to add to our collection. In our case π_{n+1} is learnt to be the optimal policy for the our current task estimate $\tilde{\mathbf{w}}$ (line 12).

In essence, Algorithm 5.1 is similar to standard Q-learning for the estimated task and can indeed be reduced to that if none of the previously learnt SFs provide any information about this new task. Nevertheless, we would like to draw attention to two main ways it dif-

[1] We use $x \overset{\alpha}{\leftarrow} y$ meaning $x \leftarrow x + \alpha y$. We also use $\nabla_{\boldsymbol{\theta}} f(x)$ to denote the gradient of $f(x)$ with respect to the parameters $\boldsymbol{\theta}$.

Algorithm 5.1 Acting and learning in a new task via SF&GPI

```
Require: { $\tilde{\boldsymbol{\phi}},\ \tilde{\Psi} \equiv \{\tilde{\boldsymbol{\psi}}^{\pi_1}, \ldots, \tilde{\boldsymbol{\psi}}^{\pi_n}\}$   features, SFs
         { extend_basis                                   learn a new SF?
         { $\alpha_\psi, \alpha_w, \varepsilon$, ns                              hyper-parameters
 1: if extend_basis then
 2:     create $\tilde{\boldsymbol{\psi}}^{\pi_{n+1}}$ parametrised by $\boldsymbol{\theta}_\psi$
 3:     $\tilde{\Psi} \leftarrow \tilde{\Psi} \cup \{\tilde{\boldsymbol{\psi}}^{\pi_{n+1}}\}$
 4: end if
 5: select initial state $s \in \mathcal{S}$
 6: for ns steps do
 7:     if Bernoulli($\varepsilon$)=1 then $a \leftarrow$ Uniform($\mathcal{A}$) {exploration}
 8:     else $a \leftarrow \mathrm{argmax}_b \max_i \tilde{\boldsymbol{\psi}}^{\pi_i}(s, b)^\top \tilde{\mathbf{w}}$ {GPI }
 9:     Execute action $a$ and observe $r$ and $s'$.
10:     $\tilde{\mathbf{w}} \xleftarrow{\alpha_w} \left[r(s, a, s') - \tilde{\boldsymbol{\phi}}(s, a, s')^\top \tilde{\mathbf{w}}\right] \tilde{\boldsymbol{\phi}}(s, a, s')$ {learn $\tilde{\mathbf{w}}$ }
11:     if extend_basis then {will learn new SFs}
12:         $a' \leftarrow \mathrm{argmax}_b \tilde{\boldsymbol{\psi}}^{\pi_{n+1}}(s, b)^\top \tilde{\mathbf{w}}$
13:         for $i \leftarrow 1, 2, \ldots, d$ do
14:             $\delta_i \leftarrow \tilde{\boldsymbol{\phi}}_i(s, a, s') + \gamma \tilde{\boldsymbol{\psi}}_i^{\pi_{n+1}}(s', a') - \tilde{\boldsymbol{\psi}}_i^{\pi_{n+1}}(s, a)$
15:             $\boldsymbol{\theta}_\psi \xleftarrow{\alpha_\psi} \delta_i \nabla_{\boldsymbol{\theta}_\psi} \tilde{\boldsymbol{\psi}}_i^{\pi_{n+1}}(s, a)$ {learn $\tilde{\boldsymbol{\psi}}^{\pi_{n+1}}$ }
16:         end for
17:     end if
18:     if $s'$ is not terminal then $s \leftarrow s'$
19:     else select initial state $s \in \mathcal{S}$
20: end for
```

fers from Q-learning. First, instead of selecting actions based on the value function being learned, the behaviour of the agent is determined by GPI (line 8). Depending on the set of SFs $\tilde{\Psi}$ used by the algorithm, this can be a significant improvement over the greedy policy induced by $\tilde{Q}^{\pi_{n+1}}$, which usually is the main determinant of a Q-learning agent's behaviour.

Algorithm 5.1 also deviates from conventional Q-learning in the way a policy is learned. There are two possibilities here. One of them is for the agent to rely exclusively on the GPI policy computed over $\tilde{\Psi}$ (when the variable extend_basis is set to false). In this case no specialised policy is learned for the current task, which reduces the RL problem to the supervised problem of determining $\tilde{\mathbf{w}}$ (solved in line 10 as a least-squares regression). The other possibility is to use data collected by the GPI policy to learn a policy π_{n+1} specifically tailored for the task. As shown in lines 14 and 15, this comes down to solving equation (5.4). When π_{n+1} is learned the function $\tilde{Q}^{\pi_{n+1}}(s, a) = \tilde{\boldsymbol{\psi}}^{\pi_{n+1}}(s, a)^\top \tilde{\mathbf{w}}$ also takes part in GPI. This means that, if the approximations $\tilde{Q}^{\pi_i}(s, a)$ are reasonably accurate, the policy computed by Algorithm 5.1 should be strictly better than Q-learning's counterpart. This is likely to be true especially at the beginning of training in a new task. The SFs $\tilde{\boldsymbol{\psi}}^{\pi_{n+1}}$ can then be added to the set $\tilde{\Psi}$, and therefore a subsequent execution of Algorithm 5.1 will result in an even stronger agent. After multiple iterations we expect $Q^{\pi_{n+1}}$ to start dominating the other policies, thus at this point we would be essentially reverting to Q-learning. This ability to build and continually refine a set of skills is widely regarded as a desirable feature for continual (or lifelong) learning (Thrun, 1996).

There is a caveat, though. In order to apply Algorithm 5.1—or any variant of the SF& GPI framework—, we need not only the set $\tilde{\Psi}$, but also a way to compute features $\tilde{\boldsymbol{\phi}}(s, a, s')$ that characterise our environment. We will now discuss why this is not a trivial problem and propose a strategy to compute $\tilde{\Psi}$ and $\tilde{\boldsymbol{\phi}}(s, a, s')$ online in a stable way.

5.3.1 Challenges Involved in Building Features

In order to use Algorithm 5.1, or any variant of the SF&GPI framework, we need the features $\tilde{\boldsymbol{\phi}}(s, a, s')$. A natural way of addressing the computation of $\tilde{\boldsymbol{\phi}}(s, a, s')$ is to see it as a multi-task problem, as in (5.11). The solution of (5.11) requires data coming from D base tasks. In principle, we could look at the collection of samples as a completely separate process. However, here we are assuming that the base tasks $\mathcal{B}$ are part of the RL problem, that is, we want to collect data using policies that are competent in $\mathcal{B}$. In order to maximise the potential for transfer, while learning the policies π_i for the base tasks M_i we should also learn the associated SFs $\tilde{\boldsymbol{\psi}}^{\pi_i}$; this corresponds to building the initial set $\tilde{\Psi}$ used by Algorithm 5.1. Unfortunately, learning value functions in the form (5.6) while solving (5.11) can be problematic. Since the SFs $\tilde{\boldsymbol{\psi}}^{\pi_i}$ depend on $\tilde{\boldsymbol{\phi}}$, learning the former while refining the latter can clearly lead to undesirable and unstable solutions (this is akin to having a non-stationary reward function). Although rewards $\tilde{\boldsymbol{\phi}}$ might change slowly – we can, in principle, control for that via the learning rate, small changes in $\tilde{\boldsymbol{\phi}}$ can result in significant changes in $\tilde{\boldsymbol{\psi}}$ as these will be integrating over multiple of these small changes. On top of that, the computation of $\tilde{\boldsymbol{\phi}}$ itself depends on the SFs $\tilde{\boldsymbol{\psi}}^{\pi_i}$, for the latter ultimately define the data distribution used to solve (5.11). This circular dependency can make the process of concurrently learning $\tilde{\boldsymbol{\phi}}$ and $\tilde{\boldsymbol{\psi}}^{\pi_i}$ very unstable – something we have often observed in practice.

One possible solution is to use a conventional value function representation while solving (5.11) and only learn SFs for the subsequent tasks (Barreto et al., 2017). This has the disadvantage of not reusing the policies learned in $\mathcal{B}$ for transfer. Alternatively, one can store all the data and learn the SFs associated with the base tasks only *after* $\tilde{\boldsymbol{\phi}}$ has been learned, but this may be difficult in scenarios involving large amounts of data. Besides, any approximation errors in $\tilde{\boldsymbol{\phi}}$ will already be reflected in the initial $\tilde{\Psi}$ computed in $\mathcal{B}$.

In the next section we show that all these issues are solved when we replace (5.11) with (5.12), as proposed in Section 5.2.1.

5.3.2 Learning Features Online while Retaining Transferable Knowledge

To recapitulate, we are interested in solving D base tasks $\{M_i\}_{i=1,D}$ and, while doing so, build $\tilde{\boldsymbol{\phi}}$ and the initial set $\tilde{\Psi}$ to be used by Algorithm 5.1. We want $\tilde{\boldsymbol{\phi}}$ and $\tilde{\Psi}$ to be learned concurrently, so we do not have to store transitions, and preferably $\tilde{\Psi}$ should not reflect approximation errors in $\tilde{\boldsymbol{\phi}}$. We argued that one can accomplish all of the above by replacing (5.11) with (5.12), that is, by directly approximating $\mathbf{r}(s, a, s')$, which are observable, and adopting the resulting approximation as the features $\tilde{\boldsymbol{\phi}}$ required by Algorithm 5.1. Furthermore, in order to break the circular dependency discussed in Section 5.3.1, we propose grounding the SFs learning problem directly in the observed reward signals $\mathbf{r}(s, a, s')$. Note that this will potentially result in a slight misalignment between $\tilde{\boldsymbol{\phi}}$ and $\tilde{\Psi}$ due to the approximation errors in the reward prediction problems.

As discussed in Section 5.2.1, when using rewards as features the resulting SFs are collections of value functions: $\tilde{\boldsymbol{\psi}}^{\pi_i} = \tilde{\mathbf{Q}}^{\pi_i} \equiv [\tilde{Q}_1^{\pi_i}, \tilde{Q}_2^{\pi_i}, \ldots, \tilde{Q}_D^{\pi_i}]$. This leads to a particularly simple way of building the features $\tilde{\boldsymbol{\phi}}$ while retaining transferable knowledge in $\tilde{\Psi}$. Given a set of D base tasks M_i, while solving them we only need to carry out two extra operations: compute approximations $\tilde{r}_i(s, a, s')$ of the functions $r_i(s, a, s')$, to be used as $\tilde{\boldsymbol{\phi}}$, and evaluate the resulting policies on all base tasks—*i.e.,* compute $\tilde{Q}_j^{\pi_i}$—to build $\tilde{\Psi}$.

Before providing a practical method to compute $\tilde{\boldsymbol{\phi}}$ and $\tilde{\Psi}$, we note that, although the approximations $\tilde{r}_i(s, a, s')$ can be learned independently from each other, the computation of $\tilde{\mathbf{Q}}^{\pi_i}$ requires policy π_i to be evaluated under different reward signals. This can be accomplished in different ways; here we assume that the agent is able to interact with the tasks M_i in parallel. We can consider that at each transition the agent observes rewards from all the base tasks, $\mathbf{r} \in \mathbb{R}^D$, or a single scalar r_i associated with one of them. We will primarily assume the latter – availability to the on-task reward only, but our solution readily extends to the more informative scenario where the agent simultaneously observes D tasks.

Algorithm 5.2 shows a possible way of implementing our solution, again using Q-learning as the basic RL method. We highlight the fact that GPI can already be used in this phase, as shown in line 5, which means that the policies π_i can "cooperate" to solve each task M_i.

Algorithm 5.2 Build SF&GPI basis with ε-greedy Q-learning

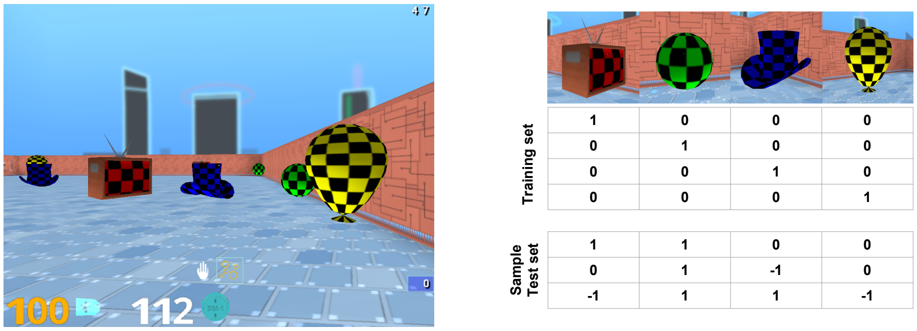
```
Require: { $M_1, M_2, \ldots, M_D$   base tasks $\mathcal{B}$
         { $\alpha_Q, \alpha_r, \varepsilon$, ns   hyper-parameters
 1: for ns steps do
 2:     select a task $k \in \{1, 2, \ldots, D\}$ and a state $s \in \mathcal{S}$
 3:     ("Pretend" you are in that task $k$ and select action accordingly)
 4:     if Bernoulli($\varepsilon$)=1 then $a \leftarrow$ Uniform($\mathcal{A}$) {exploration}
 5:     else $a \leftarrow \text{argmax}_b \max_i \mathcal{Q}_k^{\pi_i}(s, b)$ {GPI }
 6:     Execute action $a$ in $M_k$ and observe $r = r_k(s, a, s')$ and $s'$
 7:     $\boldsymbol{\theta}_r \xleftarrow{\alpha_r} [r - \tilde{r}(s, a, s')] \nabla_{\boldsymbol{\theta}_r} \tilde{r}_k(s, a, s')$
 8:     for $i \leftarrow 1, 2, \ldots, D$ do
 9:         $a' \leftarrow \text{argmax}_b \tilde{Q}_i^{\pi_i}(s, b)$ $\{a' \equiv \pi_i(s)\}$
10:         $\boldsymbol{\theta}_Q \xleftarrow{\alpha_Q} [r + \gamma \tilde{Q}_k^{\pi_i}(s', a') - \tilde{Q}_k^{\pi_i}(s, a)] \nabla_{\boldsymbol{\theta}_Q} \tilde{Q}_k^{\pi_i}(s, a)$
11:     end for
12: end for
13: return $\tilde{\boldsymbol{\varphi}} \equiv [\tilde{r}_1, \ldots, \tilde{r}_D]$ and $\tilde{\Psi} \equiv \{\tilde{\mathbf{Q}}^{\pi_1}, \ldots, \tilde{\mathbf{Q}}^{\pi_D}\}$
```

5.4 Experiments

In this section we describe the experiments used to assess whether the proposed approach can indeed promote transfer on large-scale domains. Here we focus on the most relevant aspects of the experiments; for further details please consult Appendix B.2.

5.4.1 Environment and Tasks

The environment we consider is conceptually similar to one of the problems used by (Barreto et al., 2017) to evaluate their framework: the agent has to navigate in a room picking up desirable objects while avoiding undesirable ones. Here the problem tackled is in a particularly more challenging format: instead of observing directly the state s_t at time step t, as



Training set	1	0	0	0
	0	1	0	0
	0	0	1	0
	0	0	0	1
Sample Test set	1	1	0	0
	0	1	-1	0
	-1	1	1	-1

(a) Screenshot of environment

(b) Base tasks $\mathcal{B}$ and test set $\hat{\mathcal{M}}$

Figure 5.4.1: Environment and tasks (base tasks $\mathcal{B}$, sample test tasks $\hat{\mathcal{M}}$) depiction.

in the original experiments, the agent interacts with the environment from a first-person perspective, only receiving as observation a 84 × 84 image o_t that is insufficient to disambiguate the actual underlying state of the MDP (see figure 5.4.1a).

We used DeepMind Lab platform (Beattie et al., 2016) to design our 3D environment, which works as follows. The agent finds itself in a room full of objects of different types. There are five instances of each object type: "TV", "ball", "hat", and "balloon". Whenever the agent hits an object it picks it up and another object of the same type appears at a random location in the room. This process goes on for a minute, after which the episode ends and a new one starts. The end of the episode is marked with a discount $\gamma = 0$.

The type of an object determines the reward associated with it. Thus, a task is defined (and can be referred to) by four numbers indicating the rewards attached to each object type. For example, in task 1-100 the agent is rewarded positively by objects of the first type and negatively by objects of the second type, while the other object types are irrelevant in terms of reward. Thus in this case, the agent should be interested in picking up objects of type one, while avoiding objects of type two. One can see that varying only these four numbers, the induced (optimal) behaviour can be quite complex and diverse from one task specification to the other. Nevertheless, there are also subsets of tasks that might share sub-plans/sub-routines. For instance, if we were to now consider a new task 1-1-10 we can see that the desired behaviour in this task is quite related to the first task considered 1-100. The agent in the new task cares about the same things as in 1-100 , but now it also needs to avoid objects of type three, on top of pursuing objects of type one and avoiding objects of type two. Thus, if we consider the extreme case where there are no objects of type three in our current environment, these two tasks would be equivalent. Nevertheless, the opposite scenario – where we have *only* objects of type three in the environment – illustrates that 'closeness' in the reward specification space can lead to very different optimal behaviours. In general, as we could see from these two extreme examples, there is reason to believe that the optimal policy for task `1-100` , π^*_{1-100}, could already be a good policy on task `1-1-10` and would produce good behaviour in at least some parts of the space – e.g. in parts of the state space where the number of type 3 objects is relatively low. This is precisely the intuition behind this kind of transfer we achieve in the policy space via SFs: evaluating how good previous policies are under the current task can be very valuable information.

We defined base tasks $\mathcal{B}$ that will be used by Algorithm 5.2 to build $\tilde{\boldsymbol{\phi}}$ and $\tilde{\Psi}$ in a very natural way (see Figure 5.4.1b). We focus this first investigation on "canonical" base tasks, that correspond to what our description of the environment underlying $\boldsymbol{\phi}$ would look like. Nevertheless, please note that in order to represent new reward signals, any four linearly independent set of tasks can be considered here and would result in the same generalisation. The only difference comes in the induced policies that we are going to be considering for transfer. In this respect, the chosen set of base tasks $\mathcal{B}$ would induce policies that seek one object type at a time. The hope is that these would lead to informative evaluations. The transfer ability of the algorithms will be assessed on different, unseen tasks, referred to as *test tasks*, $\hat{\mathcal{M}}$ – an illustration of such tasks can be found in Figure 5.4.1b.

5.4.2 Agents

The SF & GPI agent adopted in the experiments is a variation of Algorithms 5.1 and 5.2 that uses (Watkins, 1989) $Q(\lambda)$ to apply Q-learning with eligibility traces. Due to the complexity of the task and relatively long episodes, n-step methods were required to provide any learning for this class of control problems. As such, we needed to adapt in particular the update in Algorithm 5.2 line 10 to support trajectory-based evaluations. Since trajectories were collected under a behaviour policy that might differ from the policy we are trying to evaluate, we would cut traces whenever the two start to disagree.

The functions $\tilde{\boldsymbol{\phi}}$ and $\tilde{\Psi}$ are computed by a deep neural network whose architecture is shown in Figure 5.4.2. The network is composed of three parts. The first one uses the history of observations and actions up to time t, h_t, to compute a shared representation $f(h_t)$. The construction of $\tilde{s}_t$ can itself be broken into two stages corresponding to specific functional modules: a convolutional network (CNN) to handle the pixel-based observation o_t and a long short-term network (LSTM) to compute $f(h_t)$ in a recursive way (LeCun et al.,

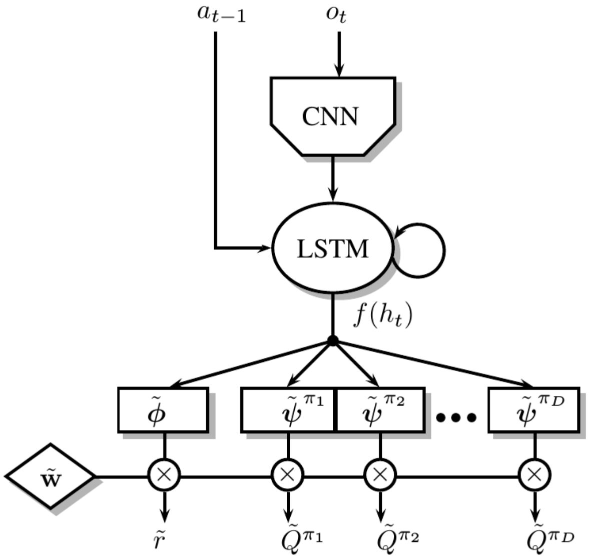


Figure 5.4.2: Schematic of the deep architecture used. Rectangles represent MLPs.

1998; Hochreiter and Schmidhuber, 1997).

The second part of the network is composed of $D + 1$ specialised blocks that receive $\tilde{s}_t$ as input and compute $\tilde{\boldsymbol{\phi}}(\tilde{s}_t, a)$ and $\tilde{\boldsymbol{\psi}}^{\pi_i}(\tilde{s}_t, a)$ for all $a \in \mathcal{A}$. Each one of these blocks is a multilayer perceptron (MLP) with a single hidden layer (Rumelhart et al., 1986). Note that the specialised blocks share the representation $f(h_t)$ representing a specific trade-off between plasticity and data efficiency. The third part of the network is simply $\tilde{\mathbf{w}}$, which combined with $\tilde{\boldsymbol{\phi}}$ and $\tilde{\boldsymbol{\psi}}^{\pi_i}$ will provide the final approximations.

The entire architecture was trained end-to-end through Algorithm 5.2 using the base tasks shown in Figure 5.4.1b. After the agent had been trained, it was tested on a test task, now using Algorithm 5.1 with the newly-learned $\tilde{\boldsymbol{\phi}}$ and $\tilde{\Psi}$. In order to handle the large number of sample trajectories needed in our environment both Algorithms 5.1 and 5.2 used the IMPALA distributed architecture (Espeholt et al., 2018). Results of this training procedure on the base tasks are included in Figure 5.4.3

Algorithm 5.1 was run with and without learning a specialised policy (as controlled by the variable extend_basis in Algorithm 5.1). We call the corresponding versions of the algorithm "SF&GPI-continual" and "SF&GPI-transfer", respectively. This is in reference

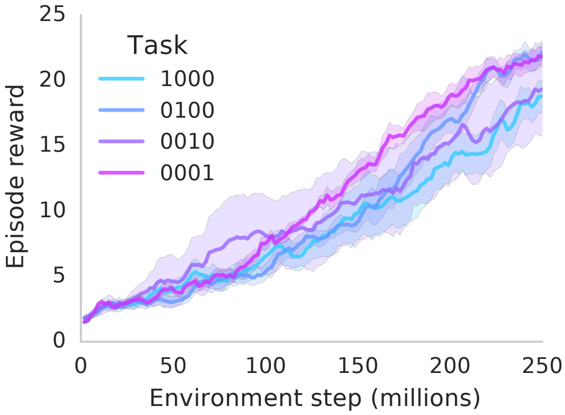
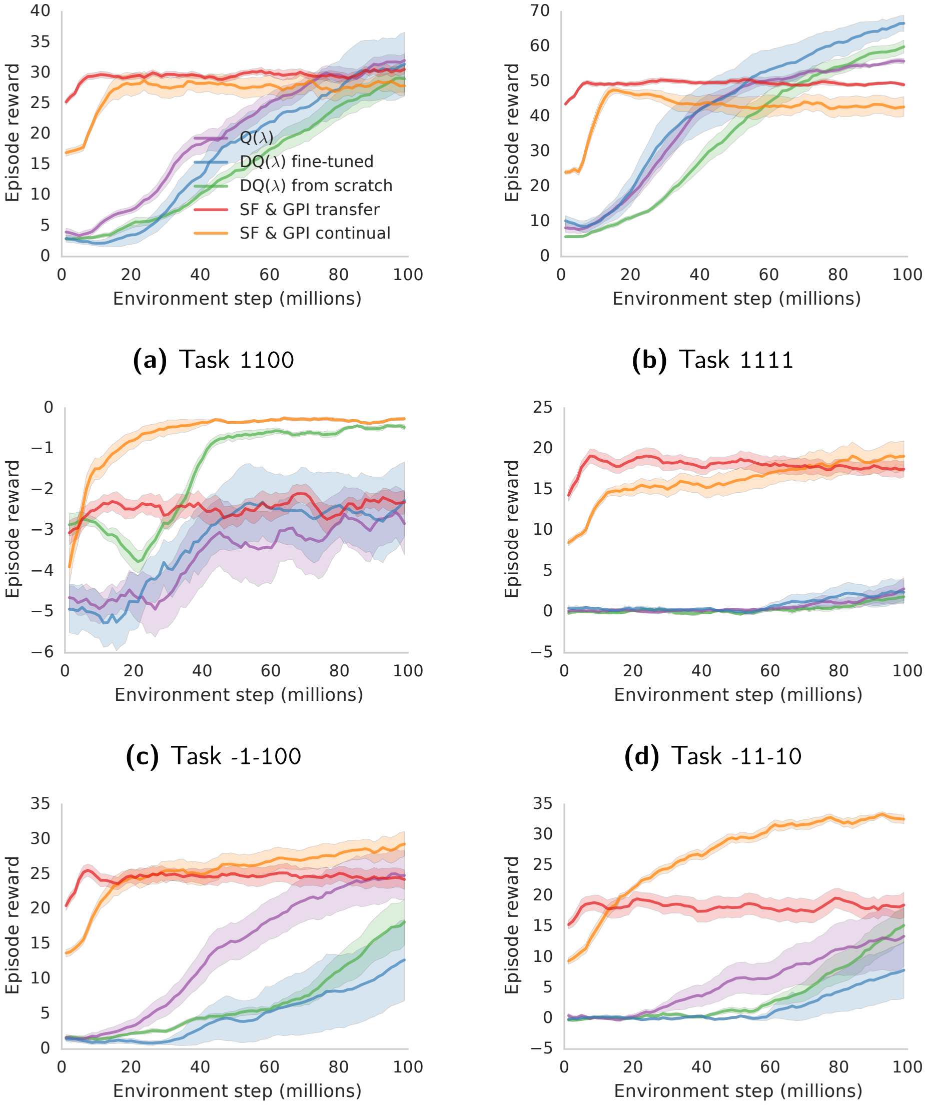


Figure 5.4.3: Base tasks $\mathcal{B}$: Training performance (average return per episode).

to the fact, that one assesses the transfer ability of GPI based only on the previous policies and the other one considers learning a new policy that can be added to our collection thus simulating one step of a continual learning process. We compare SF&GPI with baseline agents that use the same network architecture, learning algorithm, and distributed data processing. The only difference is in the way the network shown in Figure 5.4.2 is updated and used during the test phase. Specifically, we ignore the MLPs used to compute $\tilde{\boldsymbol{\varphi}}$ and $\tilde{\boldsymbol{\psi}}^{\pi_i}$ and instead add another MLP, with the exact same architecture, to be jointly trained with $\tilde{\mathbf{w}}$ through $Q(\lambda)$. We then distinguish three baselines. The first one uses the learnt shared representation $f(h_t)$ learned in the base tasks to compute an approximation $\tilde{Q}(\tilde{s}, a)$—that is, both the CNN and the LSTM are fixed. We will refer to this method simply as $Q(\lambda)$. The second baseline is allowed to modify $f(h_t)$ during the test phase, so we call it "$DQ(\lambda)$ fine tuning" as a reference to its refinement of the representation. Finally, the third baseline, "$DQ(\lambda)$ from scratch", learns its own representation $f(h_t)$ at the same time as the corresponding value function for the task at hand. This third baseline will only see data from the second phase, but its representation would also only need to cater to the test task – all the network capacity is dedicated to one task.

5.4.3 Results and Discussion

Figure 5.4.4 shows the results of SF&GPI and the baselines on a selection of test tasks $\hat{\mathcal{M}}$. The first thing that stands out is the fact that SF&GPI-transfer gives rise to reasonable poli-



(a) Task 1100

(b) Task 1111

(c) Task -1-100

(d) Task -11-10

(e) Task -1101

(f) Task -11-11

Figure 5.4.4: Average discounted return per episode on a selection of test tasks $\hat{\mathcal{M}}$. The x axes have different scales as the amount of reward available changes across tasks. Shaded regions represent one standard deviation over 10 test runs, based on three of the training runs included above.

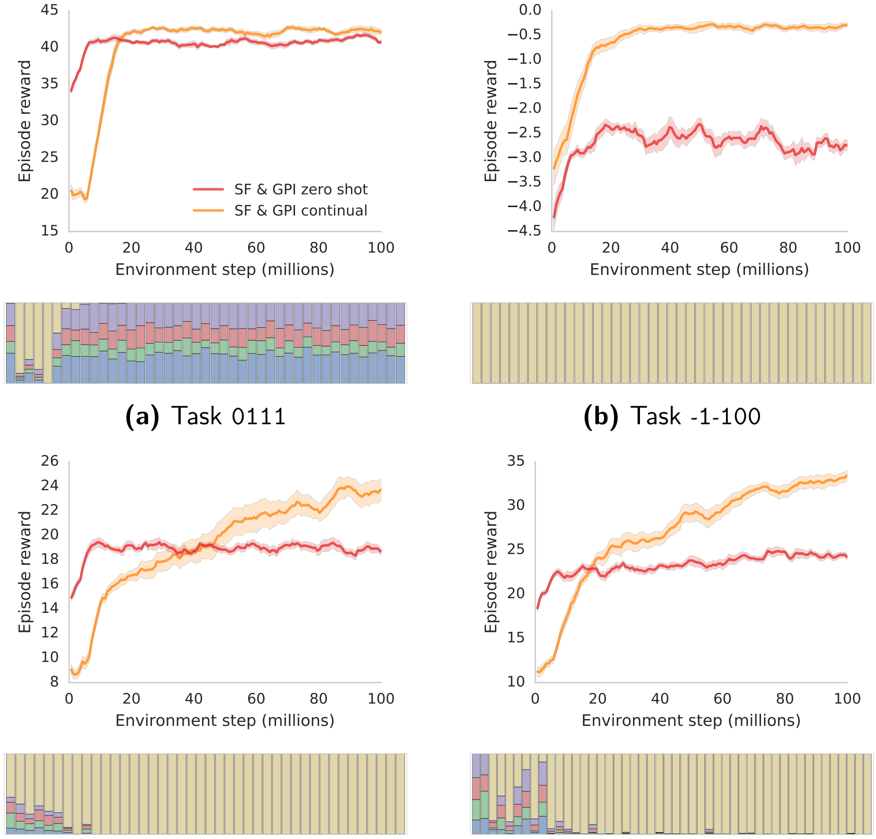
cies for the test tasks almost instantaneously. In fact, as this version of the algorithm is only solving a simple supervised learning problem, its learning progress is almost imperceptible at the scale at which the corresponding RL problem unfolds. Since the baselines *are* solving the full RL problem, in some tasks their performance eventually reaches, or even surpasses, that of the transferred policies. SF & GPI-continual combines the desirable properties of both SF & GPI-transfer and the baselines. On one hand, it still benefits from the almost instantaneous transfer promoted by SF & GPI. On the other hand, its performance keeps improving, since in this case the transferred policy is used to provide better data for learning a specialised policy. As a result, SF & GPI-continual outperforms the other methods in almost all of the tasks.[2]

Another interesting trend shown in Figures 5.4.4 is the fact that SF & GPI performs well on test tasks with negative rewards—in some cases considerably better than the alternative methods—, even though the agent only experienced positive rewards on the base tasks. This is an indication that the transferred GPI policy is combining the policies π_i for the base tasks in a non-trivial way (line 8 of Algorithm 5.1).

Ideally, our agent should rely on the GPI policy when useful but also be able to learn and use a specialised policy otherwise. In other words, whenever it is appropriate to transfer knowledge from a previously learnt policies, the agent should opt to do so, but whenever the previous tasks result in uninformative plans, it should devoted its resources to *learning* how to tackle this new scenario. Figure 5.4.5 shows that this is possible with SF & GPI-continual. Looking at Figure 5.4.5a we see that when the test task only has positive rewards the performances of SF & GPI zero-shot and continual are virtually the same. This makes sense, since in this case alternating between the policies π_i learned on $\mathcal{B}$ should lead to good performance as the test task naturally decomposes into the subtasks selected in $\mathcal{B}$. Although initially the specialised policy π_{test} does get selected by GPI a few times, eventually the policies π_i largely dominate. The figure also corroborates the hypothesis above that GPI is in general not computing a trivial policy, since even after settling on the policies π_i it keeps alternating between them.

Interestingly, when we look at the test tasks with negative rewards this pattern is no longer observed. As shown in Figures 5.4.5c and 5.4.5d, in this case SF & GPI-continual eventually outperforms SF & GPI-transfer—which is not surprising. Looking at the frequency at which policies are selected by GPI, we observe the opposite trend: now the pol-

[2] A video of SF & GPI-transfer behaviour can be found at: https://youtu.be/-dTnqfwTRMI.



(a) Task 0111

(b) Task -1-100

(c) Task -1100

(d) Task -11-11

Figure 5.4.5: Top: Comparison between SF&GPI zero-shot and continual. Shaded regions represent one standard deviation over 10 runs. Unlike in Figure 5.4.4, here all runs of SF&GPI-transfer (and continual) used the same SFs basis $\tilde{\Psi}$. **Bottom**: Coloured bar segments represent the frequency at which the policies π_i were selected by GPI in one run of SF&GPI-continual, with each colour associated with a specific policy. The policy π_{test} specialised to the task is represented in light yellow (predominant colour in figures 5.4.5c and 5.4.5d). The other colours depict the frequency of decisions that where made based on policies in $\mathcal{B}$.

icy π_{test} steadily becomes the preferred one. The fact that a specialised policy is learned and eventually dominates is reassuring, as it indicates that π_{test} does something fundamentally different and thus will contribute to the repository of skills available to the agent when added to $\tilde{\Psi}$. This also highlights the fact that SF&GPI-continual manages to provide a form of safer transfer, where if the previously learnt skills are not applicable, the agent has the option of disregarding them and specialising to this new task without much interference. This is evident from Figure 5.4.5b, where we can see that the agent quickly figures out that all previous policies, that actively seek a particular kind of object, will result in very negative returns and chooses instead to build and act according to a newly specialised policy, thus essentially reverting to $Q(\lambda)$.

5.4.4 Generalising outside the Span of $\mathcal{B}$

In this work we argued that SF&GPI can be applied even if assumption in Eq. 5.2 is not strictly satisfied. In order to illustrate this point, we reran the experiments with SF&GPI-transfer using a set of linearly-dependent base tasks, $\mathcal{B}' \equiv \{1000, 0100, 0011, 1100\}$. Clearly, $\mathcal{B}'$ can only represent tasks in which the rewards associated with the third and fourth object types are the same. We compared the results of SF&GPI-transfer using $\mathcal{B}$ and $\mathcal{B}'$ on several tasks where this is not the case. The comparison is shown in Figure 5.4.6. We can see that the resulting GPI policy under this imperfect approximation still leads to a sensible transfer policy on the new tasks – although these were somewhat adversarially chosen to be outside the span of $\mathcal{B}$' . As shown in the figure, although using a linearly-dependent set of base tasks does hinder transfer in some cases, in general it does not have a strong impact on the results. This smooth degradation of the performance is in accordance with Proposition 5.1.

The result above also illustrates an interesting distinction between the space of reward functions and the associated space of policies. Although we want to be able to represent the reward functions of all tasks of interest, this is only half of the story and does not guarantee alone that the resulting GPI policy will perform well. To see this, suppose we replace the positive rewards in $\mathcal{B}$ with negative ones. Clearly, in this case we would still have a basis spanning the same space of rewards; however, since now a policy that stands still is optimal in all tasks M_i, we should not expect GPI to give rise to good policies in tasks with positive rewards. Thus the other half of quantifying transferability here comes from the base policies used in GPI. And although in this work these policies were naturally induced by

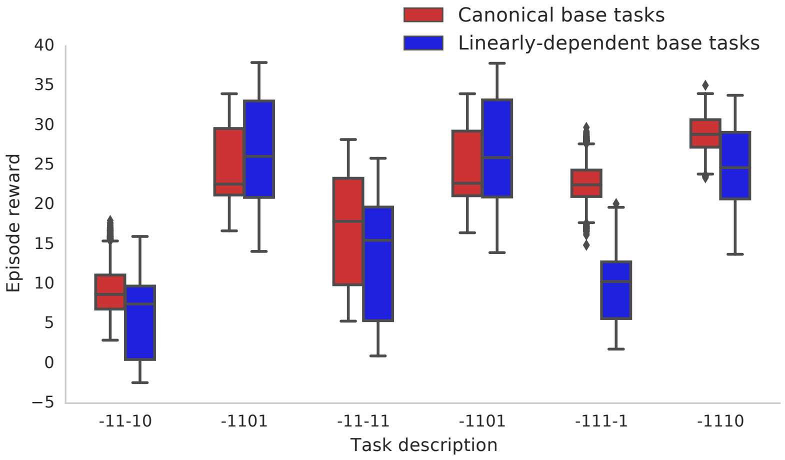


Figure 5.4.6: Performance of SF&GPI-transfer using base tasks $\mathcal{B}$ and $\mathcal{B}'$. The box plots summarise the distribution of the rewards received per episode between 50 and 100 million steps of learning.

the base tasks, one could think about separating these two learning problems. In this view, one could consider asking how to define a "behavioural basis" that leads to good policies across $\mathcal{M}$ through GPI. We leave this as an interesting open question.

5.5 Conclusion

5.5.1 Related Work

Recently, there has been a resurgence of the subject of transfer in the deep RL literature. (Teh et al., 2017) propose an approach for the multi-task problem—which in our case is the learning of base tasks—that uses a shared policy as a regulariser for specialised policies. (Finn et al., 2017) focus on fast transfer, which is akin to SF&GPI-transfer. Specifically, instead of optimising for performance on a set of tasks, they propose to learn a model that can quickly adapt to different tasks. (Rusu et al., 2016) introduce a neural-network architecture for the full continual-learning problem that consists of a growing number of specialised modules. (Kirkpatrick et al., 2017) also propose an architecture well-suited for continual learning, since it is specifically designed to avoid the problem known as "catastrophic forgetting".

Many works on the combination of deep RL and transfer propose modular network architectures that naturally induce a decomposition of the problem (Devin et al., 2017;

Heess et al., 2017; Clavera et al., 2017). Among these, a recurrent theme is the existence of sub-networks specialised in different skills that are managed by another network (Heess et al., 2016; Frans et al., 2017; Oh et al., 2017). This highlights an interesting connection between transfer learning and hierarchical RL, which has also recently re-emerged in the deep RL literature (Vezhnevets et al., 2017; Bacon et al., 2017).

Most of the approaches cited above are inherently associated with deep learning, since they exploit specific architectural designs or update rules used in the approximation. SF& GPI operate at a different level, as they do not rely on any specific feature of deep learning. Thus, rather than alternatives to our framework, we see the methods above as complementary approaches that can potentially be combined with SF&GPI.

5.5.2 Summary

In this chapter we extended the SF&GPI transfer framework in two ways. First, we showed that the theoretical guarantees supporting the framework can be extended to any set of MDPs that only differ in the reward function, regardless of whether their rewards can be computed as a linear combination of a set of features or not. In order to use SF&GPI in practice we still need the reward features, though; our second contribution is to show that these features can be the reward functions of a set of MDPs. This reinterpretation of the problem makes it possible to combine SF&GPI with deep learning in a stable way. We empirically verified this claim on a complex 3D environment that requires hundreds of millions of transitions to be solved. We showed that, by turning an RL task into a supervised learning problem, SF&GPI-transfer is able to provide skilful, non-trivial, policies almost instantaneously. We also showed how these policies can be used by SF&GPI-continual to learn specialised policies, which can then be added to the agent's set of skills. Together, these concepts can help endow an agent with the ability to build, refine, and use a set of skills while interacting with the environment.

6

Universal Successor Features Approximators

6.1 Introduction

In the previous chapter, we have seen how one can build and use successor features (SFs) for transferring knowledge in policy space. In particular, we have seen that SFs can provide us with a reliable evaluation of previous policies on a new task pertaining to the same span. These become a base for a policy improvement step, that can now be done over a set of policies. We have seen that as long as the future expectations captured by these base policies are at least partially compatible with the task at hand, we can effectively transfer the (long-term) consequences learnt under these policies. This kind of transfer explicitly exploits two commonalities assumed over the set of MDPs we are trying to obtain generalisation over the shared dynamics and the linear structure in the reward space. The combination of these two assumptions results in a nice decomposition, where one component accounts for the environment dynamics and the behaviour of the agent and the other one accounts for the reward function we seek to maximise. This decomposition is particularly useful as it supports a very easy recombination where we can now specify a new reward signal and get an instantaneous evaluation of a previous behaviour with respect to this new task. To obtain this property, the new reward signal has to be a combination of a set of features $\boldsymbol{\varphi}$, or at least well approximated by a point in the span of $\boldsymbol{\varphi}$ (Proposition 5.1). In the previous chapter, we have shown that we can circumvent the discovery of this set of features, by using instead rewards coming from tasks in the span of this underlying, unobserved $\boldsymbol{\varphi}$. Nevertheless, there are problems in which this structure (or a suitable proxy) is available to us and does not need to be inferred. Specifying $\boldsymbol{\varphi}$ can also be seen as a way of specifying a prior over the set of tasks we would be interested in. In either of these cases, when the reward structure is given, we do not have to infer the underlying space and we can focus exclusively on the control problem. This is precisely what we are going to be exploring in this chapter. And we will show how we can effectively leverage knowledge of $\boldsymbol{\varphi}$ to enable off-task and off-policy learning, making the most out of the interactions with the environment.

Thus, for the time being, we will set aside the discovery problem and assume $\boldsymbol{\varphi}$ is given – either available in the observation space, specified by the user or previously learnt as in Chapter 5. And we will focus on using this extra information to boost our previously shown transfer capabilities. This will be done by exploiting the smoothness in the task space as well as the ability to propose on-the-fly fictitious tasks in the span of $\boldsymbol{\varphi}$ and try to learn optimal policies for all of these. To sum up, in this chapter we will investigate transfer capabilities and generalisation in the set of MDPs $\mathcal{M}^{\varphi}$ for an observable set of $\boldsymbol{\varphi}$-s. As before the tasks can be specified by a task vector $\mathbf{w} \in \mathbb{R}^d$ and this is how the task will be specified to the agent. As an example, let us revisit the experimental set-up in Section 5.4. A natural candidate set for $\boldsymbol{\varphi}$ are the recognisers associated with each type of object – such a φ would be off (0) most of the time and will be on (1) when the agent picks up this particular type of object. Note that this is also a very natural space to specify tasks. Having one feature associated with picking up a particular type of object, means that specifying a vector $\mathbf{w}$ is equivalent to specifying a preference over these objects, including negative ones. In this chapter, we will be investigating this situation, where this 'interface' of task specification is provided and given a new task $\mathbf{w}'$ our aim is to output an appropriate control policy. Thus, in contrast with the last chapter where we first needed to discover the task, on the fly based on observations, here we get this task specification as an instruction but we are going to be considering the much harder problem of zero-shot generalisation in $\mathcal{M}^{\varphi}$. This means we are going to be assessing our transferability based solely on knowledge ported from other tasks, without any interaction with this new task and no adaptation phase.

Two approaches recently proposed in the literature can promote this type of zero-shot learning. *Universal value function approximators* (UVFAs)(Schaul et al., 2015a) extend the notion of value functions to also include the description of a task; by doing so one can generalise to unseen tasks by simply using their description as an argument to the approximator. Another way to transfer knowledge to unseen tasks is to use the framework explored in the last chapter, which builds on two concepts (Barreto et al., 2017): *successor features*, a representation scheme that allows a policy to be evaluated on any task of a given format, and *generalised policy improvement* (GPI), a generalisation of dynamic programming's classic operator that uses a set of policies instead of a single one. UVFAs and SF&GPI generalise to new tasks in quite different, and potentially complementary, ways. UVFAs aim to generalise across the space of tasks by exploiting structure in the underlying space of value functions. In contrast, SF&GPI 's strategy is to exploit the structure of the RL problem itself.

In this chapter we propose a model that exhibits the types of generalisation provided by both UVFAs and SF&GPI. The basic insight is to note that SFs are multi-dimensional value functions, so we can extend them in the same way a universal value function extends their unidimensional counterparts. We call the resulting model *universal successor features approximators*, or USFAs for short. USFA is a strict generalisation of its precursors. Specifically, we show that by combining USFAs and GPI we can recover both UVFAs and SF& GPI as particular cases. This opens up a new spectrum of possible approaches in between these two extreme cases.

6.2 Background

In this section, we present some background material, formalise the scenario we are interested in, and briefly revisit the methods we build upon.

6.2.1 Revised Multitask Reinforcement Learning Problem

As a reminder, in this work we are interested in generalising across multiple RL tasks, taking place in the same environment $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma)$ but differing in their reward signal. Thus each task is defined by a reward function $r_{\mathbf{w}} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$; and in this chapter, we will assume that the expected one-step reward associated with transition $s \xrightarrow{a} s'$ is given by:

$$\mathrm{E}\left[R_{\mathbf{w}}(s,a,s')\right] = r_{\mathbf{w}}(s,a,s') = \boldsymbol{\phi}(s,a,s')^{\top}\mathbf{w}, \tag{6.1}$$

where $\boldsymbol{\varphi}(s,a,s') \in \mathbb{R}^d$ are features of (s,a,s') and $\mathbf{w} \in \mathbb{R}^d$ are weights. The features $\boldsymbol{\varphi}(s,a,s')$ can be thought of as salient events that may be desirable or undesirable to the agent, such as for example picking up an object, going through a door, or knocking into something. And as discussed in the introduction, from hereon we will assume that the agent is able to recognise such events—that is, $\boldsymbol{\varphi}$ is observable—, but the solution we propose can be easily extended to the case where $\boldsymbol{\varphi}$ must be learned (Barreto et al., 2017).

Thus in this work we will revisit the set up in (Barreto et al., 2017) and be considering generalising over the set of MDP induced by some *observed features* $\boldsymbol{\varphi}$, as introduced in (5.3):

$$\mathcal{M}^{\varphi} \equiv \{M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r_w, \gamma) | r_w(s,a,s') = \boldsymbol{\varphi}(s,a,s')^{\top}\mathbf{w}, \mathbf{w} \in \mathbb{R}^d\}, \tag{6.2}$$

Given $\mathbf{w} \in \mathbb{R}^d$ representing a task, the goal of the agent is to find a *policy* $\pi_{\mathbf{w}} : \mathcal{S} \mapsto \mathcal{A}$ that maximises the expected discounted sum of rewards, also called the *return* $G^{(t)}_{\mathbf{w}} = \sum_{i=0}^{\infty} \gamma^i R^{(t+i)}_{\mathbf{w}}$, where $R^{(t)}_{\mathbf{w}} = R_{\mathbf{w}}(S_t, A_t, S_{t+1})$ is the reward received at the t$^{\text{th}}$ time step. In the following, we will denote the *action-value function* of a policy π on task $\mathbf{w}$ as $Q^{\pi}_{\mathbf{w}}(s,a) \equiv \mathrm{E}^{\pi}\left[G^{(t)}_{\mathbf{w}} \mid S_t = s, A_t = a\right]$, representing the expected discounted reward that this policy will collect under task $\mathbf{w}$.

6.2.2 Knowledge Transfer in Policy Space: Parametric vs Non-parametric

Here we focus on one aspect of multitask RL: how to *transfer* knowledge to unseen tasks (Taylor and Stone, 2009; Lazaric, 2012). Specifically, we ask the following question: how can an agent leverage knowledge accumulated on a set of tasks $\mathcal{B} \subset \mathbb{R}^d$ to speed up the solution of a new task $\mathbf{w}' \notin \mathcal{B}$?

To investigate the question above we recast it using the formalism commonly adopted in learning. Specifically, we define a distribution $\mathcal{D}_{\mathbf{w}}$ over $\mathbb{R}^d$ and assume the goal is for the agent to perform as well as possible under this distribution. Note that this distribution on $\mathbf{w}$ induces a distribution over the MDPs we were are going to be considering. As usual, we assume a fixed budget of sample transitions and define a training set $\mathcal{B} \sim \mathcal{D}_{\mathbf{w}}$ that is used by the agent to learn about the tasks of interest. We also define a test set $\hat{\mathcal{M}} \sim \mathcal{D}_{\mathbf{w}}$ and use it to assess the agent's generalisation—that is, how well it performs on the distribution of MDPs induced by $\mathcal{D}_{\mathbf{w}}$.

A natural way to address the learning problem above is to use (Schaul et al., 2015a) *universal value-function approximators* (UVFAs). The basic insight behind UVFAs is to note that the concept of optimal value function can be extended to include as one of its arguments a description of the task; an obvious way to do so in the current context is to define the function $Q^*(s, a, \mathbf{w}) : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \mapsto \mathbb{R}$ as the optimal value function associated with task $\mathbf{w}$. The function $Q^*(s, a, \mathbf{w})$ is called a *universal value function* (UVF); a UVFA is then the corresponding approximation, $\tilde{Q}(s, a, \mathbf{w})$. As most parametric approximators, UVFAs assume, in their parametrization, a particular structure over the optimal value function space in $\mathcal{M}^\varphi$. The hope here is that a sufficiently expressive UVFA can identify and exploit structure across the *joint state-action-task space* $\mathcal{S} \times \mathcal{A} \times \mathbb{R}^d$. In other words, with enough data, a properly trained UVFA should be able to generalise across the space of tasks. Nevertheless, with few tasks – small set $\mathcal{B}$, our coverage of this joint space might be quite limited which is bound to impact our generalisation abilities.

A different way of generalising across tasks was explored in Chapter 5, which builds on assumption (6.1) and two core concepts: *successor features* (SFs) and *generalised policy improvement* (GPI). The SFs of a state-action pair (s, a) under policy π are given by

$$\boldsymbol{\psi}^\pi(s, a) \equiv \mathrm{E}^\pi\left[\sum_{i=t}^{\infty} \gamma^{i-t} \boldsymbol{\varphi}_{i+1} \,|\, S_t = s, A_t = a\right].$$

SFs allow one to immediately compute the value of a policy π on *any* task $\mathbf{w}$: it is easy to show that, when (6.1) holds, $Q^\pi_{\mathbf{w}}(s, a) = \boldsymbol{\psi}^\pi(s, a)^\top \mathbf{w}$. Suppose that the agent has learned the SFs $\boldsymbol{\psi}^{\pi_i}$ of policies $\pi_1, \pi_2, ..., \pi_n$. When exposed to a new task defined by $\mathbf{w}$, the agent can immediately compute $Q^{\pi_i}_{\mathbf{w}}(s, a) = \boldsymbol{\psi}^{\pi_i}(s, a)^\top \mathbf{w}$. Thus we can build evaluations of a set of policies on this new task and use these to formulate a control policy. Let the GPI policy be defined as $\pi(s) \in \mathrm{argmax}_a Q^{\max}(s, a)$, where $Q^{\max} = \max_i Q^{\pi_i}$. The GPI theorem states that $Q^\pi(s, a) \geq Q^{\max}(s, a)$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. This result also extends to the scenario where we replace Q^{π_i} with approximations $\tilde{Q}^{\pi_i}$. It is also worth noting that this holds irrespective of the set of (base) policies $\{\pi_i\}_{i=1,n}$ and one does not need to assume any structure on this policy space, for the above to hold. Nevertheless, if we do assume that this set of policies correspond to optimal policies maximising a set of tasks $\{\mathbf{w}_i\}_{i=1,n}$, then we can obtain an extra guarantee over the quality of the resulting GPI policy. This will depend on the distance of the new task $\mathbf{w}$ to the set of base tasks $\mathcal{B}$ and the maximum approximation error incurred at the policy evaluation step (Barreto et al., 2017). Note that this method of transfer does not assume a distribution of the tasks at training nor at testing time. These sets of tasks can be sampled, in principle, from different distributions

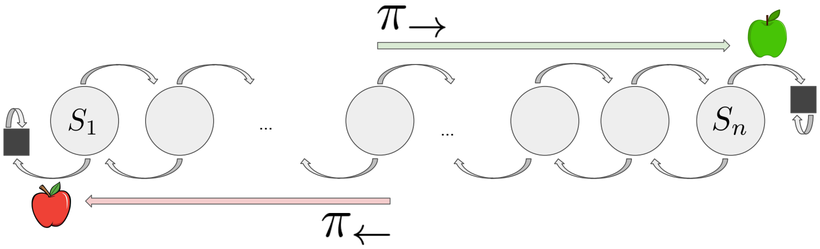


Figure 6.2.1: A simple Chain MDP with two rewards (at end states).

without affecting our guarantees and our ability to transfer information. On the other hand, this means such a procedure would not be able to exploit any structure, if present, in $\mathcal{D}_{\mathbf{w}}$. Instead, SFs will generalise over the $\mathcal{S} \times \mathcal{A}$ space and GPI will have a non-parametric approach to generalise in policy space.

In the next section we will take a look at a very simple class of MDP of the form $\mathcal{M}^{\varphi}$, and in this context try to highlight the benefits and shortcomings for addressing learning and generalisation in this class of problems by a fully parametric model like UVFAs versus SF & GPI, a semi-parametric model which we have seen previously to be a potentially powerful model for transfer.

6.2.3 A Motivating Example: UVFAs and SFs

Consider a simple chain MDP (Figure 6.2.1), with n states and two actions: left and right. For simplicity, both actions result in a deterministic transition to the neighbouring state in the direction of the chosen action and we assume a constant discount factor γ for each transition. At the two ends of the chain, one can find the two potentially rewarding transitions $(s_1, \leftarrow)$ and respectively $(s_n, \rightarrow)$ associated with picking up a particular type of apple. After collecting the associated reward, these transitions lead to a non-reward absorbing state – one can also see this as the end of an episode. Now imagine a person A might like red apples but dislike green apples and a person B that does not have a strong preference for any of these kind of apples. In this environment, generalising in $\mathcal{M}^{\varphi}$ would mean we can predict what is the best policy for each of these different preferences.

To make this more formal, we would be interested in the class of MDP of the form (6.1), where the features φ are defined as:

$$\boldsymbol{\varphi}(s,a) = \begin{cases} [1,0], \text{for } s = s_1, a = \leftarrow \\ [0,1], \text{for } s = s_n, a = \rightarrow \\ [0,0], \textit{otherwise} \end{cases} \tag{6.3}$$

Thus if the task is specified by $\mathbf{w} = [w_1, w_2]$, and let us consider $w_1, w_2 \geq 0$, one can see that the two desired outcomes are at the end of the chains. And depending on these precise values and our proximity to the two ends of the chain, the optimal strategy will prioritize one outcome over the other. Having this in mind, we can easily derive that for any state $s = s_k$, the optimal value function associated with this state, under task $\mathbf{w}$ is:

$$V^*_{\mathbf{w}}(s) = \max\left(\gamma^k w_1, \gamma^{n-k} w_2\right), \text{where } s = s_k \tag{6.4}$$

Thus for all tasks $\mathbf{w}$, the associated optimal plan will involve a cut-off point k where $(\gamma^k w_1 - \gamma^{n-k} w_2)$ changes sign. Hence, in this space, $\mathcal{M}^{\varphi} = \{M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r_{\mathbf{w}}, \gamma) | r_{\mathbf{w}}(s,a) = \varphi(s,a)^T \mathbf{w}, \mathbf{w} \in \mathbb{R}^2_+\}$, we will have at most n optimal deterministic policies – one for each cut-off point s_k. At the same time, if we look at the possible optimal value functions in $\mathcal{M}^{\varphi}$ we will see that these can take arbitrary values across $\mathbb{R}_+$.

Of course, there is a lot of structure in this space that a UVFA could easily exploit, for instance if two tasks $\mathbf{w}_1$ and $\mathbf{w}_2$ are close to each other under some measure, say $||\mathbf{w}_1 - \mathbf{w}_2||_2 \leq \delta$, their corresponding optimal value functions will take similar values. For instance, if $\mathbf{w}_1 = [1, 0]$ and $\mathbf{w}_2 = [1, \gamma^{n-1}]$, for a large enough chain, we can see that their norm is quite small $||\mathbf{w}_1 - \mathbf{w}_2||_2 = \gamma^{n-1}$ as $\gamma \in (0,1)$. One can easily see that this similarity would also be reflected in the corresponding optimal value functions $Q^*_{\mathbf{w}_1}$ and $Q^*_{\mathbf{w}_2}$. To make this more precise, one can see that the value functions agree in most states:

$$Q^*_{\mathbf{w}_1}(s,a) = Q^*_{\mathbf{w}_2}(s,a) = Q^{\pi_{\leftarrow}}(s = s_k, a) = \begin{cases} \gamma^k, \text{for } a = \leftarrow, k \leq n \\ \gamma^{k+1}, \text{for } a = \rightarrow, k \leq n-1 \end{cases},$$

and for one transition – the final transition at the end of the chain – where $Q^*_{\mathbf{w}_1}(s_n, \rightarrow) = 0$ and $Q^*_{\mathbf{w}_2}(s_n, \rightarrow) = \gamma^{n-1}$. Thus $||Q^*_{\mathbf{w}_1} - Q^*_{\mathbf{w}_2}||_\infty = \gamma^{n-1} = ||\mathbf{w}_1 - \mathbf{w}_2||_2$. And by a similar argument we can see the same holds for $\mathbf{w}'_2 = [1, \gamma^{n-2}]$: $||Q^*_{\mathbf{w}'_2} - Q^*_{\mathbf{w}_2}||_\infty = \gamma^{n-1} - \gamma^{n-2} = ||\mathbf{w}'_2 - \mathbf{w}_2||_2$, although the associated optimal policies will differ in this second case.

In spite of this, something like l_2 or other metric on $\mathbf{w}$ might not capture all the similarity present in the task space. An example of that would be $\mathbf{w}_1 = \mathbf{w}$ and $\mathbf{w}_2 = a\mathbf{w}$, where for a large a the value functions can be quite far apart in terms of value, although they are just a factor from each other $Q^*_{w_1} = aQ^*_{w_2}$. Note that this is generally true for any set of tasks $\mathcal{M}^\varphi$ for given set of $\boldsymbol{\varphi}$. We could normalise $\mathbf{w}$ -s to avoid this particular issue. Nevertheless, one could also argue that we would like our agents to be able to recognize that these two tasks are essentially the same, without us having to specify an equivalence class[1]. Furthermore, even if we were to normalise our task descriptors $\mathbf{w}$, we would still have an infinite number of value functions to represent, although we have previously established that there only a finite number of optimal policies. Thus, even with this massive reduction via normalisation, we still seem to be missing out on some of the structure present in this space. This suggests that the equivalence class above does not capture all of the rich structure available in *policy space*. This is partially because, under different reward signals, this equivalence is not readily present in the value space. Different reward signals could lead to the same optimal policy, but won't necessarily lead to the same value function. This is true in our example. And it is worth noting that this is bound to be true more widely, as the mapping between (deterministic) optimal policies and value functions is generally one-to-many. Nevertheless, this is by no means an argument against the usage of powerful functional approximations, like UVFAs, that assume and exploit smoothness assumptions (in $\mathcal{S}, \mathcal{A}$ and/or $\mathbf{w}$). Instead the above merely points out that there is more structure, particularly in the policy space, that the smoothness alone cannot account for. Thus, although very powerful, exploiting similarities in state, action and task space, *UVFAs might not be enough to exploit the rich structure in $\mathcal{M}^\varphi$.*

To see that let us consider a different example. Take tasks $\mathbf{w}_1 = [1, 0]$ and $\mathbf{w}_3 = [\gamma^{-n}, 1]$ which might not be that close to each other under some distance nor a scalar factor away (trivial equivalence), but will nevertheless induce the same optimal policy, $\pi_{\leftarrow}$, that from any state s will seek the left-most state of the chain and the associated reward. Although in this example the value functions space is fairly smooth as exposed above, the distance between these tasks $||\mathbf{w}_1 - \mathbf{w}_2|| = ||1 - \gamma^{-n-1}||$ is likely to be too large to support any generalisation between these tasks' value functions under a smoothness assumption. But even though the optimal value functions for $\mathbf{w}_1$ and $\mathbf{w}_3$ might not be as close to each other under some norm $||.||$, their associated SFs are *actually the same* across the state space, as

[1] Scaling the reward is usually not a problem for a single-task problem. Yet learning a value function on a scaled reward is not the same as generalising in the value space over a class of such rewards.

they implicitly encode the information that these two tasks will lead to the same optimal policy. Since we know there are only n optimal policies, we only have n SFs that we would need to represent in order to fully cover the space of optimal policies. This suggest that SFs might be the more compact way of representing optimal value functions. Let us look how these SFs $\{\boldsymbol{\psi}^{\pi_k}\}_{k=1,n}$, where π_k is the optimal policy that will choose to go left for all states left of s_k and choose to go right for all state on the right of s_k. The successor features corresponding to one of these policies take the form:

$$\boldsymbol{\psi}^{\pi_k}(s, \leftarrow) = \begin{cases} [1, 0], \text{for } s = s_1 \\ [\gamma^i, 0], i < k \\ [0, \gamma^{n-i+1}], i \geq k \end{cases} \qquad \boldsymbol{\psi}^{\pi_k}(s, \rightarrow) = \begin{cases} [0, 1], \text{for } s = s_n \\ [\gamma^{i+1}, 0], i < k \\ [0, \gamma^{n-i}], i \geq k \end{cases} \tag{6.5}$$

Now suppose we have learnt all n SFs, $\{\boldsymbol{\psi}^{\pi_k}\}_{k=1,n}$. A question one might ask is how do we match an arbitrary task $\mathbf{w} \in \mathbb{R}_+$ with its corresponding policy and/or optimal value function. This is where GPI comes into play. Similar to what we have done for transfer in the previous chapter, we can readily build the evaluation of all these under task $\mathbf{w}$. And one of these value functions will correspond to the optimal value function $Q^*_{\mathbf{w}}$. Moreover, it is easy to see that we can use GPI to identify this policy. If $\exists k^* \in \{1, \cdots, k\}$ s.t. π_{k^*} is the optimal policy for $\mathbf{w}$, then $Q^*_{\mathbf{w}}(s, a) = Q^{\pi_{k^*}}_{\mathbf{w}}(s, a) \geq Q^{\pi}(s, a), \forall \pi$, and thus

$$\pi_{GPI}(s) \in \max_k \max_a Q^{\pi_k}_{\mathbf{w}}(s, a) = \max_a Q^{\pi_{k^*}}_{\mathbf{w}}(s, a) = \max_a Q^*_{\mathbf{w}}(s, a). \tag{6.6}$$

Thus, more generally, if we have access to the collection of optimal SFs, we can recover the optimal policy *and* the optimal value function for any task $\mathbf{w}$. Now the question becomes how to best build this collection of SFs. In principle we can apply Algorithm 5.1 (Chapter 5) as long as we define a diverse enough set of base tasks to induce all optimal policies present in $\mathcal{M}^{\phi}$. While this is possible in this simple chain MDP example, this might not always be the case. Moreover, although the space of policy (and SFs) will be generally more compact than that of the values functions in $\mathcal{M}^{\phi}$, this set might not always be finite or it might still include a large number of policies. Even in our simple example, if we increase n the number of SFs that would be needed to represent, computation would scale accordingly. *Thus the way we were building SFs one at a time as proposed in Algorithm 5.1 might not scale well enough* to give us a diverse enough collection. Nevertheless, this is precisely where the kind of structure UVFAs exploit comes in handy. To see this, let us revisit tasks $\mathbf{w}'_2 = [1, \gamma^{n-2}]$ and $\mathbf{w}_2 = [1, \gamma^{n-1}]$ above. These are 'close' to each other, under l_2 and as argued previously so will their value functions but so would their corresponding SFs.

Nevertheless note that these two tasks will actually induce two different optimal policies $\pi_{\mathbf{w}_2} = \pi_{\mathbf{w}_3} = \pi_n$ and respectively $\pi_{\mathbf{w}'_2} = \pi_{n-1}$. Although these are different policies, they are quite related and will agree in most of the state space, except for $s = s_n$. Thus, it stands to reason that knowing about one can help us generalise to the other. And similarities like these are precisely what parametric functional approximators are good at capturing and exploiting for generalisation. Therefore, we propose to use such an approximator to build a large set of SFs that correspond to different optimal policies in $\mathcal{M}^{\phi}$ – this is akin to learning the optimal value functions, but instead, we would be using this approximator to generalise SFs across the policy space. A UVFA-like approximator for SFs should be able to capture the generalisation between $\mathbf{w}_2$ to $\mathbf{w}'_2$ and then GPI can make the leap between $\mathbf{w}_2$ and $\mathbf{w}_3$ through the evaluation provided by $\psi_{\mathbf{w}_2}$. In doing so, we obtain a more scalable method to build a large, potentially infinite set of SFs that can then be used in conjunction with property (6.6) to obtain generalisation across $\mathcal{M}^{\phi}$. Lastly note that even if we are not able to represent all optimal policies, by combining multiple of these via GPI, we can still get a very good policy for a new task – as empirically shown in the chapter.

6.3 Universal Successor Features Approximators

UVFAs and SF&GPI address the transfer problem described in Section 6.2.2 in quite different ways. With UVFAs, one trains an approximator $\tilde{Q}(s, a, \mathbf{w})$ by solving the training tasks $\mathbf{w} \in \mathcal{M}$ using any RL algorithm of choice. One can then generalise to a new task by plugging its description $\mathbf{w}'$ into $\tilde{Q}$ and then acting according to the policy $\pi(s) \in \text{argmax}_a \tilde{Q}(s, a, \mathbf{w}')$. With SF&GPI one solves each task $\mathbf{w} \in \mathcal{M}$ and computes an approximation of the SFs of the resulting policies $\pi_{\mathbf{w}}$, $\tilde{\boldsymbol{\psi}}^{\pi_{\mathbf{w}}}(s, a) \approx \boldsymbol{\psi}^{\pi_{\mathbf{w}}}(s, a)$.

The algorithmic differences between UVFAs and SF&GPI reflect the fact that these approaches exploit different properties of the transfer problem. UVFAs aim at generalising across the space of tasks by exploiting structure in the function $Q^*(s, a, \mathbf{w})$. In contrast, SF&GPI's strategy for generalising across tasks is to exploit structure in the RL problem itself. GPI builds on the general fact that a greedy policy with respect to a value function will, in general, perform better than the policy that originated the value function. SFs, in turn, exploit the structure (6.1) to make it possible to quickly evaluate policies across tasks and apply GPI efficiently.

Obviously, both UVFAs and GPI have advantages and limitations ,but their types of generalisation are in some sense complementary. Thus, it is then natural to ask if we can simultaneously have the two types of generalisation. In this work, we propose a model that provides exactly that. The main insight is actually simple: since SFs are multi-dimensional value functions, we can extend them in the same way as universal value functions extend regular value functions. In the next section, we elaborate on how exactly to do so.

6.3.1 Universal Successor Features

As discussed in Section 6.2.2, UVFs are an extension of standard value functions defined as $Q^*(s, a, \mathbf{w})$. If $\pi_{\mathbf{w}}$ is one of the optimal policies of task $\mathbf{w}$, we can rewrite the definition as $Q^{\pi_{\mathbf{w}}}(s, a, \mathbf{w})$. This makes it clear that the argument $\mathbf{w}$ plays two roles in the definition of a UVF: it determines both the task $\mathbf{w}$ and the policy $\pi_{\mathbf{w}}$ (which will be optimal with respect to $\mathbf{w}$). This does not have to be the case, though. Similarly to (Sutton et al., 2011) *general value functions* (GVFs), we could in principle define a function $Q(s, a, \mathbf{w}, \pi)$ that "disentangles" the task from the policy. This would provide a model that is even more general than UVFs. In this section we show one way to construct such a model when assumption (6.1) holds. If (6.1) is true, we can revisit the definition of SFs and write

$$Q(s, a, \mathbf{w}, \pi) = \boldsymbol{\psi}^{\pi}(s, a)^{\top}\mathbf{w}. \tag{6.7}$$

If we want to be able to compute $Q(s, a, \mathbf{w}, \pi)$ for any π, we need SFs to span the space of policies π. Thus, we define *universal successor features* as $\boldsymbol{\psi}(s, a, \pi) \equiv \boldsymbol{\psi}^{\pi}(s, a)$. Based on such definition, we call $\tilde{\boldsymbol{\psi}}(s, a, \pi) \approx \boldsymbol{\psi}(s, a, \pi)$ a *universal successor features approximator* (USFA).

According to this definition, we can see that now policies π become inputs to our functional approximator. Since these are in general functions over the whole $\mathcal{S} \times \mathcal{A}$ space, we will need to find a more compressed way of specifying this argument. Thus in practice, whenever we define a USFA we will also need to define an embedding for the policies π. Let $e : (\mathcal{S} \mapsto \mathcal{A}) \mapsto \mathbb{R}^k$ be a *policy-encoding mapping*, that is, a function that turns policies π into vectors in $\mathbb{R}^k$. We can then see USFs as a function of $e(\pi)$: $\boldsymbol{\psi}(s, a, e(\pi))$. The definition of the policy-encoding mapping $e(\pi)$ can have a strong impact on the structure of the resulting USF. We now point out a general equivalence between policies and reward functions that will provide a practical way of defining $e(\pi)$. It is well known that any reward function induces an optimal policy (Puterman, 1994), that can be recovered by acting greedily with respect to the optimal function. A point that is perhaps less immediate is that the converse is also true. Given a deterministic policy π, one can easily define a reward function r_π that induces this policy: for example, we can have $r_\pi(s, \pi(s), \cdot) = 0$ and $r_\pi(s, a, \cdot) = -1$ for any $a \neq \pi(s)$. Therefore, we can use rewards to refer to deterministic policies and vice-versa.

Since here we are interested in generalising to policies that maximise reward functions of the form (6.1), by restricting our attention to policies induced by tasks $\mathbf{z} \in \mathbb{R}^d$ we end up with a conveniently simple encoding function $e(\pi_{\mathbf{z}}) = \mathbf{z}$ that is bound to include at least some of our target policies. Thus although USFA can be defined for any policy embedding space in this work, in this work we will adopt a policy embedding that corresponds to the task embedding in $\mathcal{M}^{\varphi}$, that this policy is trying to maximise. Therefore $e(\pi) = \mathbf{z}$ if and only if π is the maximising the reward signal $r_{\mathbf{z}} = \mathbf{z}^T \boldsymbol{\varphi}$. From this encoding function it follows that $Q(s, a, \mathbf{w}, \pi_{\mathbf{z}}) = Q(s, a, \mathbf{w}, \mathbf{z})$. It should be clear that UVFs are a particular case of this definition when $\mathbf{w} = \mathbf{z}$. Going back to the definition of USFs, we can finally write $Q(s, a, \mathbf{w}, \mathbf{z}) = \boldsymbol{\psi}(s, a, \mathbf{z})^\top \mathbf{w}$. Thus, if we learn a USF $\boldsymbol{\psi}(s, a, \mathbf{z})$, we have a value function that generalises over both tasks and policies, as promised.

6.3.2 USFA Generalisation

We now revisit the question as to why USFAs should provide the benefits associated with both UVFAs and SF&GPI. We will discuss how exactly to train a USFA in the next section, but for now suppose that we have trained one such model $\tilde{\boldsymbol{\psi}}(s, a, \mathbf{z})$ using the training tasks in $\mathcal{B}$. It is then not difficult to see that we can recover the solutions provided by both UVFAs and SF&GPI. Given an unseen task $\mathbf{w}'$, let π be the GPI policy defined as

$$\pi(s) \in \text{argmax}_a \max_{\mathbf{z} \in \mathcal{C}} \tilde{Q}(s, a, \mathbf{w}', \mathbf{z}) = \text{argmax}_a \max_{\mathbf{z} \in \mathcal{C}} \tilde{\boldsymbol{\psi}}(s, a, \mathbf{z})^\top \mathbf{w}', \tag{6.8}$$

where $\mathcal{C} \subset \mathbb{R}^d$. Clearly, if we make $\mathcal{C} = \{\mathbf{w}'\}$, we get exactly the sort of generalisation associated with UVFAs. On the other hand, setting $\mathcal{C} = \mathcal{B}$ essentially recovers SF&GPI.

The fact that we can recover both UVFAs and SF&GPI opens up a spectrum of possibilities in between the two. For example, we could apply GPI over the training set augmented with the current task, $\mathcal{C} = \mathcal{B} \cup \{\mathbf{w}'\}$. In fact, USFAs allow us to apply GPI over *any* set of tasks $\mathcal{C} \subset \mathbb{R}^d$. The benefits of this flexibility are clear when we look at the theory supporting the generalisation SF&GPI provides. As revised in last chapter, (Barreto et al., 2017) provide theoretical guarantees on the performance of SF&GPI applied to any task $\mathbf{w}' \in \hat{\mathcal{M}}$ based on a fixed set of SFs. Below we state a slightly more general version of this result that highlights the two types of generalisation promoted by USFAs.

Proposition 6.1

Let $\mathbf{w}' \in \mathcal{M}'$ and let $Q^{\pi}_{\mathbf{w}'}$ be the action-value function of executing policy π on task $\mathbf{w}'$. Given approximations $\{\tilde{Q}^{\pi_\mathbf{z}}_{\mathbf{w}'} = \tilde{\boldsymbol{\psi}}(s, a, \mathbf{z})^\top \mathbf{w}'\}_{\mathbf{z} \in \mathcal{C}}$, let π be the GPI policy defined in 6.8. Then,

$$\|Q^*_{\mathbf{w}'} - Q^{\pi}_{\mathbf{w}'}\|_\infty \leq \frac{2}{1-\gamma} \left(\min_{\mathbf{z} \in \mathcal{C}} \left(\|\boldsymbol{\varphi}\|_\infty \underbrace{\|\mathbf{w}' - \mathbf{z}\|}_{\delta_d(\mathbf{z})} \right) + \max_{\mathbf{z} \in \mathcal{C}} \left(\|\mathbf{w}'\| \cdot \underbrace{\|\boldsymbol{\psi}^{\pi_\mathbf{z}} - \tilde{\boldsymbol{\psi}}(s, a, \mathbf{z})\|_\infty}_{\delta_\psi(\mathbf{z})} \right) \right) \tag{6.9}$$

where $Q^*_{\mathbf{w}'}$ is the optimal value of task $\mathbf{w}'$, $\psi^{\pi_\mathbf{z}}$ are the SFs corresponding to the optimal policy for task $\mathbf{z}$.

When we write the result in this form, it becomes clear that, for each policy $\pi_\mathbf{z}$, the right-hand side of (A.31) involves two terms: i) $\delta_d(\mathbf{z})$, the distance between the task of interest $\mathbf{w}'$ and the task that induced $\pi_\mathbf{z}$, and ii) $\delta_\psi(\mathbf{z})$, the quality of the approximation of the SFs associated with $\pi_\mathbf{z}$.

In order to get the tightest possible bound (A.31) we want to include in $\mathcal{C}$ the policy $\mathbf{z}$ that minimises $\delta_d(\mathbf{z}) + \delta_\psi(\mathbf{z})$. This is exactly where the flexibility of choosing $\mathcal{C}$ provided by USFAs can come in handy. Note that, if we choose $\mathcal{C} = \mathcal{B}$, we recover Barreto et al.'s (2017) bound, which may have an irreducible $\min_{\mathbf{z} \in \mathcal{C}} \delta_d(\mathbf{z})$ even with a perfect approximation of the SFs in $\mathcal{B}$. On the other extreme, we could query our USFA at the test point $\mathcal{C} = \{\mathbf{w}'\}$. This would result in $\delta_d(\mathbf{w}') = 0$, but can potentially incur a high cost due to the associated approximation error $\delta_\psi(\mathbf{w}')$, depending on the functional approximation's ability to capture this point.

6.3.3 How to Train a USFA

Now that we have an approximation $\tilde{Q}(s, a, \mathbf{w}, \mathbf{z})$ a natural question is how to train this model. In this section, we show that the decoupled nature of $\tilde{Q}$ is reflected in the training process, which assigns clearly distinct roles for tasks $\mathbf{w}$ and policies $\pi_\mathbf{z}$.

In the scenario considered here the transition at time t will be of the form $(s_t, a_t, \boldsymbol{\varphi}_{t+1}, s_{t+1})$. Since $\boldsymbol{\varphi}$ allows us to compute the reward of any task $\mathbf{w}$, and since our policies are encoded by tasks $\mathbf{z}$, data in the format above allows us to learn the value function of *any* policy $\pi_{\mathbf{z}}$ on *any* task $\mathbf{w}$. To see why this is so, let us define one of the fundamental quantities used in RL to learn value functions: the temporal-difference (TD) error (Sutton and Barto, 1998). Given transitions in the form above, the n-step TD error associated with policy $\pi_{\mathbf{z}}$ on task $\mathbf{w}$ will be

$$\begin{aligned}\boldsymbol{\delta}^{t,n}_{\mathbf{wz}} &= \sum_{i=t}^{t+n-1} \gamma^{i-t} r_{\mathbf{w}}(s_i, a_i, s_{i+1}) + \gamma^n \tilde{Q}(s_{t+n}, \pi_{\mathbf{z}}(s_{t+n}), \mathbf{w}, \mathbf{z}) - \tilde{Q}(s_t, a_t, \mathbf{w}, \mathbf{z}) \\ &= \left[\sum_{i=t}^{t+n-1} \gamma^{i-t} \boldsymbol{\varphi}(s_i, a_i, s_{i+1}) + \gamma^n \tilde{\boldsymbol{\psi}}(s_{t+n}, a_{t+n}, \mathbf{z}) - \tilde{\boldsymbol{\psi}}(s_t, a_t, \mathbf{z}) \right]^{\top} \mathbf{w} = (\boldsymbol{\delta}^{t,n}_{\mathbf{z}})^{\top} \mathbf{w},\end{aligned} \tag{6.10}$$

where $a_{t+n} = \text{argmax}_b \tilde{Q}(s_{t+n}, b, \mathbf{z}, \mathbf{z}) = \text{argmax}_b \tilde{\boldsymbol{\psi}}(s_{t+n}, b, \mathbf{z})^{\top} \mathbf{z}$. As it is well known, the TD error $\boldsymbol{\delta}^{t,n}_{\mathbf{wz}}$ allows us to learn the value of policy $\pi_{\mathbf{z}}$ on task $\mathbf{w}$; since here $\boldsymbol{\delta}^{t,n}_{\mathbf{wz}}$ is a function of $\mathbf{z}$ and $\mathbf{w}$ only, we can learn about any policy $\pi_{\mathbf{z}}$ on any task $\mathbf{w}$ by just plugging in the appropriate vectors.

Equation (6.10) highlights some interesting (and subtle) aspects involved in training a USFA. Since the value function $\tilde{Q}(s, a, \mathbf{w}, \mathbf{z})$ can be decoupled into two components, $\tilde{\boldsymbol{\psi}}(s, a, \mathbf{z})$ and $\mathbf{w}$, the process of evaluating a policy on a task reduces to learning $\tilde{\boldsymbol{\psi}}(s, a, \mathbf{z})$ using the vector-based TD error $\boldsymbol{\delta}^{t,n}_{\mathbf{z}}$ showing up in (6.10). Since $\boldsymbol{\delta}^{t,n}_{\mathbf{z}}$ is a function of $\mathbf{z}$ only, the updates to $\tilde{Q}(s, a, \mathbf{w}, \mathbf{z})$ will *not* depend on $\mathbf{w}$. How do the tasks $\mathbf{w}$ influence the training of a USFA, then? If sample transitions are collected by a behaviour policy, as is usually the case in online RL, a natural choice is to have this policy be induced by a task $\mathbf{w}$. When this is the case, the training tasks $\mathbf{w} \in \mathcal{M}$ will define the distribution used to collect sample transitions. Whenever we want to update $\boldsymbol{\psi}(s, a, \mathbf{z})$ for a different $\mathbf{z}$ than the one used in generating the data, we find ourselves in the *off-policy* regime (Sutton and Barto, 1998) and we need to cut traces accordingly, whenever policies disagree.

Assuming that the behaviour policy is induced by the tasks $\mathbf{w} \in \mathcal{B}$, training a USFA involves two main decisions: how to sample tasks from $\mathcal{B}$ and how to sample policies $\pi_{\mathbf{z}}$ to be trained through (6.10) or some variant. As alluded to before, these decisions may have a big impact on the performance of the resulting USFA, and in particular on the trade-offs

Algorithm 6.1 Learn USFA with ε-greedy Q-learning

```
Require: { $\mathcal{B}$            training tasks
         { $\mathcal{D}_{\mathbf{z}}$  distributions over $\mathbb{R}^d$
         { $n_{\mathbf{z}}$         number of policies
         { ns           total number of steps
         { $\varepsilon$            exploration parameter
         { $\alpha$            learning rate
Select a task $\mathbf{w} \in \mathcal{M}$
[Begin episode] Set initial state $s = s_0 \in \mathcal{S}$
for ns steps do
    for $i \leftarrow 1, 2, \ldots, n_{\mathbf{z}}$ do
        Sample $\mathbf{z}_i \sim \mathcal{D}_{\mathbf{z}}(\cdot|\mathbf{w})$ {sample policies, possibly based on current task }
        if Bernoulli($\varepsilon$)=1 then $a \leftarrow$ Uniform($\mathcal{A}$) {exploration}
        else $a \leftarrow \text{argmax}_b \max_i \tilde{\boldsymbol{\psi}}(s, b, \mathbf{z}_i)^\top \mathbf{w}$ {GPI }
        Execute action $a$ and observe $\boldsymbol{\varphi}$ and $s'$
        Sample $\mathbf{z}_i \sim \mathcal{D}_{\mathbf{z}}(\cdot|\mathbf{w})$ {sample policies for learning}
        for $i \leftarrow 1, 2, \ldots, n_{\mathbf{z}}$ do
            Update $\tilde{\boldsymbol{\psi}}$ based on sampled policies
            $a' \leftarrow \text{argmax}_b \tilde{\boldsymbol{\psi}}(s, b, \mathbf{z}_i)^\top \mathbf{z}_i$ $\{a' \equiv \pi_i(s')\}$
            $\boldsymbol{\theta} \overset{\alpha}{\leftarrow} \left[\varphi + \gamma \tilde{\boldsymbol{\psi}}(s', a', \mathbf{z}_i) - \tilde{\boldsymbol{\psi}}(s, a, \mathbf{z}_i)\right] \nabla_{\boldsymbol{\theta}} \tilde{\boldsymbol{\psi}}$
        end for
    end for
    $s \leftarrow s'$
end for
return $\boldsymbol{\theta}$
```

involved in the choice of the set of policies $\mathcal{C}$ used by the GPI policy (6.8). As a form of illustration, Algorithm 6.1 shows a possible regime to train a USFA based on particularly simple strategies to select tasks $\mathbf{w} \in \mathcal{B}$ and to sample policies $\mathbf{z} \in \mathbb{R}^d$.

One aspect of Algorithm 6.1 worth calling attention to is the fact that the distribution $\mathcal{D}_{\mathbf{z}}$ used to select policies can depend on the current task $\mathbf{w}$. This allows one to focus on specific regions of the policy space; for example, one can sample policies using a Gaussian distribution centred around $\mathbf{w}$. This will allow us to provide the functional approximator with fictitious data-points in the policy space easy to generalise to. At the same time, tasks that are close to $\mathbf{w}$ are likely to induce similar optimal policies. If that is the case, trajectories produced under $\mathbf{w}$ would allow for very efficient off-policy learning for these $\mathbf{z}$.

6.4 Experiments

In this section, we describe the experiments conducted to test the proposed architecture in a multitask setting and assess its ability to generalise to unseen tasks. To illustrate USFAs capabilities for generalisation we will look at two different sets of MDPs. The first one is an example where the structure in $\mathcal{M}^{\varphi}$ can be easily exploited by a parametric model like UVFAs, but where vanilla SF&GPI on the training tasks as presented in last chapter would not have enough coverage to represent the additional optimal policies present in $\mathcal{M}^{\varphi}$. We will see that the proposed model can overcome this limitation and leverage the parametric generalisation in policy space. As a second example, we will revisit the experimental setup in Section 5.4, where we have already seen that the kind of generalisation that GPI provides can be very effective. Moreover, we will see that this scenario is actually an example where GPI's ability to recombine policies can substantially outweigh the parametric generalisation UVFAs can provide, at least for a limited set of training tasks.

We start with a simple illustrative example to provide intuition on the kinds of generalisation provided by UVFAs and SF&GPI. We also show how in this example USFAs can effectively leverage both types of generalisation and outperform its precursors. For this, we will consider the simple two-state MDP depicted in Figure 6.4.1. To motivate the example, suppose that state s_1 of our MDP represents the arrival of a traveller to a new city. The traveller likes coffee (C) and food (F) and wants to try what the new city has to offer. In order to model that, we will use features $\boldsymbol{\varphi} \in \mathbb{R}^2$, with φ_1 representing the quality of the coffee and φ_2 representing the quality of the food, both ranging from 0 to 1. The traveler has done some research and identified the places that serve the best coffee and the best food; in our MDP these places are modelled by terminal states associated with actions 'C' and 'F' whose respective associated rewards are $\boldsymbol{\varphi}(\cdot, C) = \boldsymbol{\varphi}(C) = [1, 0]$ and $\boldsymbol{\varphi}(F) = [0, 1]$. As one can infer from these feature vectors, the best coffee place does not serve food and the best restaurant does not serve coffee (at least not a very good one). Nevertheless, other places in town serve both; as before, we will model these places by actions P_i associated with features $\boldsymbol{\varphi}(P_i)$. We assume that $\|\boldsymbol{\varphi}(P_i)\|_2 = 1$ and consider $N = 5$ alternative places P_i evenly spaced on the preference spectrum. We model how much the traveller wants coffee and food on a given day by $\mathbf{w} \in \mathbb{R}^2$. If the traveler happens to want only one of these (i.e. $\mathbf{w} \in \{[1, 0], [0, 1]\}$), she can simply choose actions 'C' or 'F' and get a reward

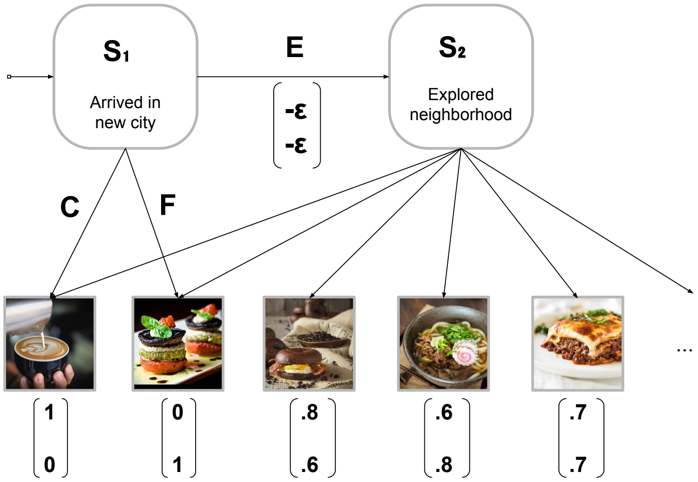


Figure 6.4.1: Trip MDP: Depiction of MDP.

$r = \boldsymbol{\varphi}(\cdot)^\top \mathbf{w} = 1$. If instead, she wants both coffee and food (i.e. if $\mathbf{w}$ is not an "one-hot" vector), it may actually be best to venture out to one of the other places. Unfortunately, this requires the traveller to spend some time researching the area, which we model by an action 'E' associated with feature $\boldsymbol{\varphi}(E) = [-\varepsilon, -\varepsilon]$. This models the fact that there is a small price to pay for choosing to explore. After choosing 'E' the traveler lands on state s_2 and can now reach any place in town: $C, F, P_1, ..., P_N$. Note that, depending on the vector of preferences $\mathbf{w}$, it may be worth paying the cost of $\boldsymbol{\varphi}(E)^\top \mathbf{w}$ to subsequently get a reward of $\boldsymbol{\varphi}(P_i)^\top \mathbf{w}$ (here $\gamma = 1$). The leftmost plot in Figure 6.4.2 depicts what the *optimal final destinations* would be under different preferences $\mathbf{w} \in [0, 1]^2$.

6.4.1 Trip MDP

In order to assess the transfer ability of UVFAs, SF&GPI and USFAs, we define a training set $\mathcal{B} = \{10, 01\}$ and $K = 50$ test tasks corresponding to directions in the two-dimensional $\mathbf{w}$-space: $\hat{\mathcal{M}} = \{\mathbf{w}' | \mathbf{w}' = [\cos(\frac{\pi k}{2K}), \sin(\frac{\pi k}{2K})], k = 0, 1, ..., K\}$. We start by analysing what SF&GPI would do in this scenario. We focus on training task $\mathbf{w}_C = [1, 0]$, but an analogous reasoning applies to task $\mathbf{w}_F = [0, 1]$. Let π_C be the optimal policy associated with task $\mathbf{w}_C$. It is easy to see that $\pi(s_1) = \pi(s_2) = C$. Thus, under π_C it should be clear that $Q^{\pi_C}_{\mathbf{w}'}(s_1, C) > Q^{\pi_C}_{\mathbf{w}'}(s_1, E)$ for all test tasks $\mathbf{w}'$. Since the exact same reason-

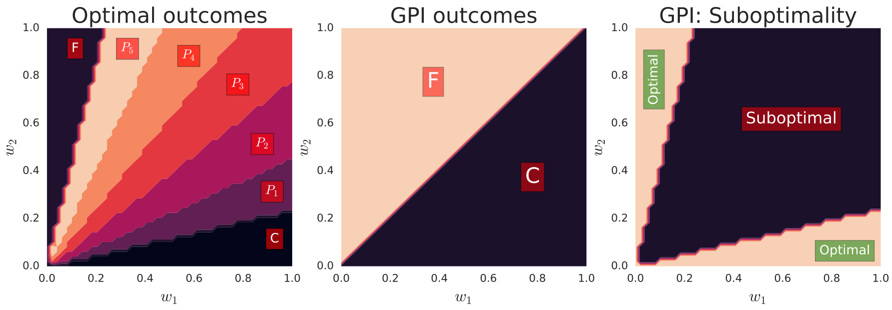


Figure 6.4.2: Trip MDP ($N = 5$): (left) Optimal policies final states. (middle) Final outcomes under the GPI policy induced by $\mathcal{B} = \{10, 01\}$. (right) GPI is suboptimal for most the task space.

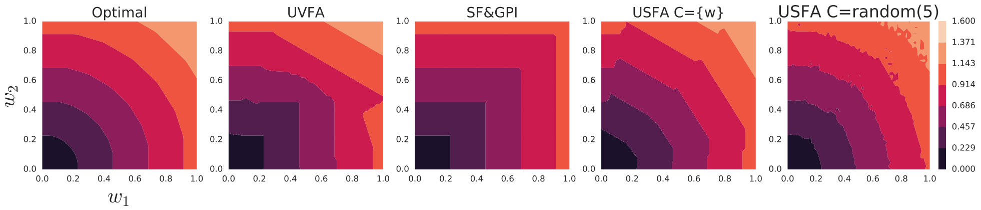


Figure 6.4.3: [Sample run] Performance of the different methods (in this order, starting with the second subplot): UVFA, SF&GPI on the perfect SFs induced by $\mathcal{B}$, USFA with $\mathcal{C} = random(5)$ and USFA with $\mathcal{C} = \{\mathbf{w}'\}$ as compared to the optimal performance one could get in this MDP (plot above). As above, the optimal performance and the *SF&GPI* were computed exactly.

ing applies to task $\mathbf{w}_F$ if we replace action C with action F, the GPI policy (6.8) computed over $\{\boldsymbol{\psi}^{\pi_C}, \boldsymbol{\psi}^{\pi_F}\}$ will be suboptimal for most test tasks in $\hat{\mathcal{M}}$. This is more generally true as illustrated in Figure 6.4.2. The middle plot illustrated the final destinations of the GPI induced policy under $\mathbf{w} \in [0, 1]^2$. Moreover if we look at the difference between the first plot and this middle one, we can see that in this case GPI will fail to recover the optimal policy for most of the $\mathbf{w}$ space. Actually it will fail to do so, for all tasks outside the ones for which the base policies are optimal – as highlighted in the last subplot in the same figure.

Training a UVFA on the same set $\mathcal{M}$, will not be perfect either due to the very limited number of training tasks. Nonetheless, the smoothness in the approximation allows for a slightly better generalisation in $\hat{\mathcal{M}}$. We include a depiction of the performance a trained UVFA will typically achieve after 100 episodes in Figure B.3.1. By comparing with the subplot on its right corresponding to the returns under the GPI policy over a perfect set

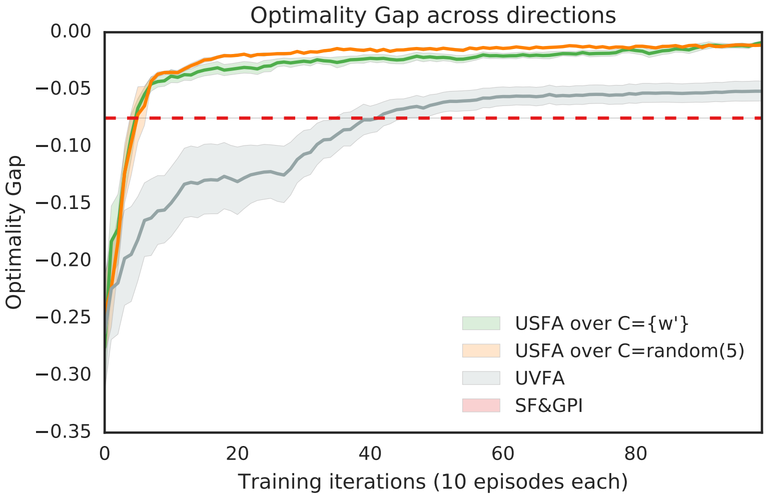


Figure 6.4.4: Trip MDP: Zero-shot performance across directions (semi-circle in the unit square): Optimality gap for $= \{\mathbf{w}'|\mathbf{w}' = [\cos(\frac{\pi k}{2K}), \sin(\frac{\pi k}{2K})], k = 0, 1, \dots, K\}$ for $K = 50$. These results were averaged over 10 runs.

of SFs, we can see that the smoothness assumption allows the functional approximator to extrapolate a bit better than its SF&GPI counterpart. This is not a perfect interpolation as the training set is very small, but note that with more tasks the UVFA will refine this representation further. The same is true for SF&GPI but in a slightly different way. As in this example, unless we include in $\mathcal{B}$ at least one point corresponding to each of the optimal policies in $\mathcal{M}^\phi$ we will not be able to recover the optimal value functions. Thus if we were to increase N (possible mixed outcomes), we would have to increase the number of SFs we represent accordingly. On the other hand, due to the structure in our outcomes, UVFA-s generalisation will be almost independent of our choice N.

Alternatively, we can use Algorithm 6.1 to train a USFA on the training set $\mathcal{B}$. In order to do so we sampled $n_{\mathbf{z}} = 5$ policies $\mathbf{z} \in \mathbb{R}^2$ using a uniformly random distribution $\mathcal{D}_{\mathbf{z}}(\cdot|\mathbf{w}) = \mathcal{U}([0,1]^2)$ (see line 5 in Algorithm 6.1). When acting on the test tasks $\mathbf{w}'$ we considered two choices for the candidates set: $\mathcal{C} = \{\mathbf{w}'\}$ and $\mathcal{C} = \{\mathbf{z}_i|\mathbf{z}_i \sim \mathcal{U}([0,1]^2), i = 1, 2, \dots, 5\}$. In the last two subplots of Figure 6.4.3 we provide an example a train USFA and its generalisation performance over the $[0,1]^2$.

Lastly, we will quantify our generalisation ability by measuring performance on the direction set defined above $\hat{\mathcal{M}}$. Aggregated results are provided in Figure 6.4.4. As a refer-

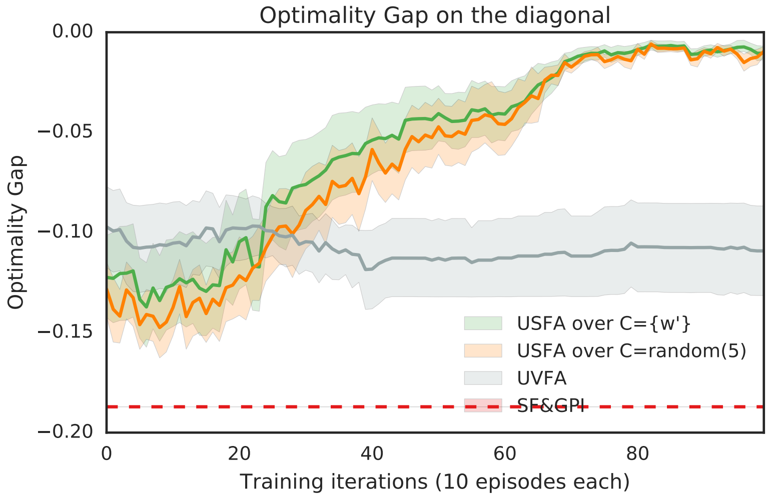
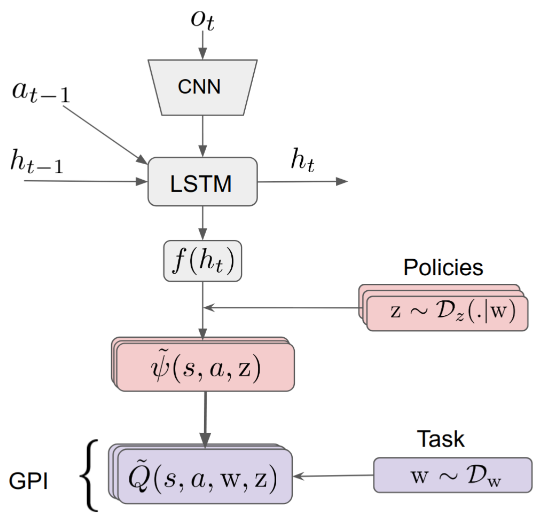


Figure 6.4.5: Trip MDP: Zero-shot performance on the diagonal: Optimality gap for $\hat{\mathcal{M}}' = \{\mathbf{w}'|w'_1 = w'_2, w'_1 \in [0,1]\}$. These results were averaged over 10 runs.

ence we report the performance of SF&GPI using the *true* $\{\boldsymbol{\psi}^{\pi_C}, \boldsymbol{\psi}^{\pi_F}\}$ – no approximation. We also show the learning curve of a UVFA. As shown in the figure, USFA clearly outperforms its precursors and quickly achieves near-optimal performance. This is due to two factors. First, contrary to vanilla SF&GPI, USFA can discover and exploit the rich structure in the policy-space, enjoying the same generalisation properties as UVFAs but now enhanced by the combination of the off-policy *and* off-task training regime. Second, the ability to sample a candidate set $\mathcal{C}$ that induces some diversity in the policies considered by GPI overcomes the suboptimality associated with the training SFs $\boldsymbol{\psi}^{\pi_C}$ and $\boldsymbol{\psi}^{\pi_F}$.

A particularly adversarial choice of test tasks for the vanilla SF&GPI would be the diagonal in the $[0,1]^2$ quadrant depicted in the plot above: $\hat{\mathcal{M}}' = \{\mathbf{w}'|w'_1 = w'_2, w'_1 \in [0,1]\}$. This is, in a sense, maximally away from the training tasks and both of the precursor models are bound to struggle in this portion of the space. This intuition was indeed empirically validated. Empirical results are provided in Figure 6.4.5. As mentioned above, this is an adversarial evaluation, mainly to point out that, in general, there might be regions of the space where the generalisation of the previous models can be very bad, but where the combination of them can still recover close to optimal performance.


o_t
CNN
a_{t-1}
h_{t-1}
LSTM
h_t
$f(h_t)$
Policies
$\mathrm{z} \sim \mathcal{D}_z(.|\mathrm{w})$
$\tilde{\psi}(s, a, \mathrm{z})$
Task
GPI
$\tilde{Q}(s, a, \mathrm{w}, \mathrm{z})$
$\mathrm{w} \sim \mathcal{D}_\mathrm{w}$

$$\pi(s) = \arg\max_a \max_z \tilde{Q}(s, a, \mathrm{w}, \mathrm{z})$$

Figure 6.4.6: USFA architecture used to the large scale, partially observable experiments in Section 6.4.2.

6.4.2 Large-scale Experiments

Environment and tasks. We used the DeepMind Lab platform to design a 3D environment consisting of one large room containing four types of objects: TVs, balls, hats, and balloons (Beattie et al., 2016). This is the same environment used in Chapter 5. A depiction of the environment through the eyes of the agent can be seen in Figure 5.4.1a. The features φ_i are indicator functions associated with object types, *i.e.*, $\varphi_i(s, a, s') = 1$ if and only if the agent hit an object of type i (say, a TV) on the transition $s \xrightarrow{a} s'$. A task is defined by four real numbers $\mathbf{w} \in \mathbb{R}^4$ indicating the rewards attached to each type of object. Note that these numbers can be negative, in which case the agent has to avoid the corresponding object type. For instance, in task $\mathbf{w} = [1\text{-}100]$ the agent is interested in objects of the first type and should avoid objects of the second type, while the other objects are irrelevant.

Agent architecture. A depiction of the architecture used for the USFA agent is illustrated in Figure 6.4.6. The architecture has three main modules: i) A fairly standard *input processing unit* composed of three convolution layers and an LSTM followed by a non-linearity (Schmidhuber, 1996); ii) A *policy conditioning module* that combines the state embedding coming from the first module, $s \equiv f(h)$, and the policy embedding, $\mathbf{z}$, and produces $|\mathcal{A}|$ outputs corresponding to the SFs of policy $\pi_{\mathbf{z}}$, $\tilde{\boldsymbol{\psi}}(s, a, \mathbf{z})$; and iii) The *evaluation module*, which, given a task $\mathbf{w}$ and the SFs $\tilde{\boldsymbol{\psi}}(s, a, \mathbf{z})$, will construct the evaluation of policy $\pi_{\mathbf{z}}$ on $\mathbf{w}$, $\tilde{Q}(s, a, \mathbf{w}, \mathbf{z}) = \tilde{\boldsymbol{\psi}}(s, a, \mathbf{z})^\top \mathbf{w}$.

Training and baselines. We trained the above architecture end-to-end using a variation of Alg. 6.1 that uses (Watkins, 1989) $Q(\lambda)$ to apply Q-learning with eligibility traces. As the distribution $\mathcal{D}_{\mathbf{z}}$ used in line 5 of Alg. 6.1 we adopted a Gaussian centred at $\mathbf{w}$: $\mathbf{z} \sim \mathcal{N}(\mathbf{w}, 0.1\,\mathbf{I})$, where $\mathbf{I}$ is the identity matrix. We used the canonical vectors of $\mathbb{R}^4$ as the training set, $\mathcal{B} = \{1000, 0100, 0010, 0001\}$. Once an agent was trained on $\mathcal{B}$ we evaluated it on a separate set of unseen tasks, $\hat{\mathcal{M}}$, using the GPI policy (6.8) over different sets of policies $\mathcal{C}$. Specifically, we used: $\mathcal{C} = \{\mathbf{w}'\}$, which corresponds to a UVFA with an architecture specialised to (6.1); $\mathcal{C} = \mathcal{B}$, which corresponds to doing GPI on the SFs of the training policies (similar to (Barreto et al., 2017)), and $\mathcal{C} = \mathcal{B} \cup \{\mathbf{w}'\}$, which is a combination of the previous two. We also included as baselines two standard UVFAs that do not take advantage of the structure (6.1); one of them was trained on-policy and the other one was trained off-policy (see Appendix B.3.2.2). The evaluation on the test tasks $\hat{\mathcal{M}}$ was done by "freezing" the agent at different stages of the learning process and using the GPI policy (6.8) to select actions. To collect and process the data we used an asynchronous scheme similar to IMPALA (Espeholt et al., 2018).

6.4.3 Results and Discussion

Figure 6.4.7 shows the results of the agents after being trained on $\mathcal{M}$. One thing that immediately stands out in the figure is the fact that *all* architectures generalise quite well to the test tasks. This is a surprisingly good result if we consider the difficulty of the scenario considered: recall that the agents are solving the test tasks *without any learning taking place*. This performance is even more impressive as we note that some test tasks contain negative rewards, something never experienced by the agents during training. When we look at the relative performance of the agents, it is clear that USFAs perform considerably better than the unstructured UVFAs. This is true even for the case where $\mathcal{C} = \{\mathbf{w}'\}$, in which USFAs essentially reduce to a structured UVFA that was trained by decoupling tasks and policies. The fact that USFAs outperform UVFAs in the scenario considered here is not particularly surprising since the former exploit the structure (5.2) while the latter do not. In any case, it is reassuring to see that our model can indeed exploit such a structure effectively. This

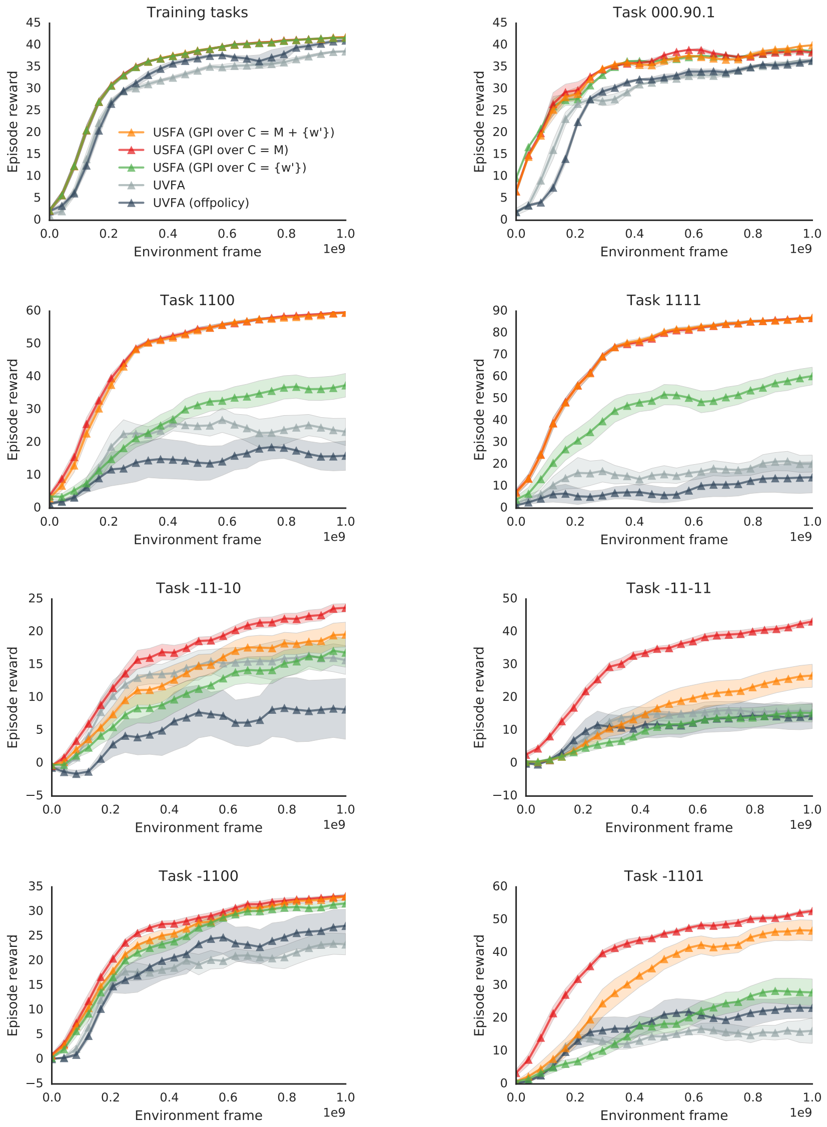


Figure 6.4.7: Learning curves for training tasks $\mathcal{B}$ and generalisation performance on a sample of test tasks $\mathbf{w}' \in \mathcal{M}'$ after training on $\mathcal{M}$. Shaded areas represent one standard deviation over 10 runs.

result also illustrates a particular way of turning prior knowledge about a problem into a favourable inductive bias in the UVFA architecture.

It is also interesting to see how the different instantiations of USFAs compare against each other. As shown in Figure 6.4.7, there is a clear advantage in including $\mathcal{M}$ to the set of policies $\mathcal{C}$ used in GPI (6.8). This suggests that, in the specific instantiation of this problem, the type of generalisation provided by SF&GPI is more effective than that associated with UVFAs. One result that may seem counter-intuitive at first is the fact that USFAs with $\mathcal{C} = \mathcal{B} + \{\mathbf{w}'\}$ sometimes perform worse than their counterparts using $\mathcal{C} = \mathcal{B}$, especially on tasks with negative rewards. Here we note two points. First, although including more tasks to $\mathcal{C}$ results in stronger guarantees for the GPI policy, strictly speaking, there are no guarantees that the resulting policy will perform better (see Barreto et al.'s Theorem 1, 2017). Another explanation that is more likely in the current scenario is that errors in the approximations $\tilde{\boldsymbol{\psi}}(s, a, \mathbf{z})$ may have a negative impact on the resulting GPI policy (6.8), for example, if the resulting $\tilde{Q}(s, a, \mathbf{w}, \mathbf{z})$ are overestimations of the actual value functions. On the other hand, comparing USFA's results using $\mathcal{C} = \mathcal{B} + \{\mathbf{w}'\}$ and $\mathcal{C} = \{\mathbf{w}'\}$, we see that by combining the generalisation of UVFAs and GPI we can boost the performance of a model that only relies on one of them.

In the above scenario, SF&GPI on the training set $\mathcal{B}$ seems to provide a more effective way of generalising, as compared to UVFAs, even when the latter has a structure specialised to (6.1). Nevertheless, with less conservative choices of $\mathcal{D}_{\mathbf{z}}$ that provide a greater coverage of the $\mathbf{z}$ space we expect the *structured* UVFA $(C = \{\mathbf{w}'\})$ to generalise better. Note that this can be done without changing $\mathcal{M}$ and is not possible with conventional UVFAs. One of the strengths of USFAs is exactly that: by disentangling tasks and policies, one can learn about the latter without ever having to actually try them out in the environment. We exploit this possibility to repeat our experiments now using $\mathcal{D}_{\mathbf{z}} = \mathcal{N}(\mathbf{w}, 0.5\,\mathbf{I})$. Results are shown in Figure6.4.8. As expected, the generalisation of the structured UVFA improves considerably, almost matching that of GPI. This shows that USFAs can operate in two regimes: i) with limited coverage of the policy space GPI over $\mathcal{M}$ will provide reliable generalisation; ii) with broader coverage of the space structured UVFAs will do increasingly better.[2]

[2] Videos of USFAs in action on the links https://youtu.be/Pn76cfXbf2Y and https://youtu.be/0afwHJofbB0.

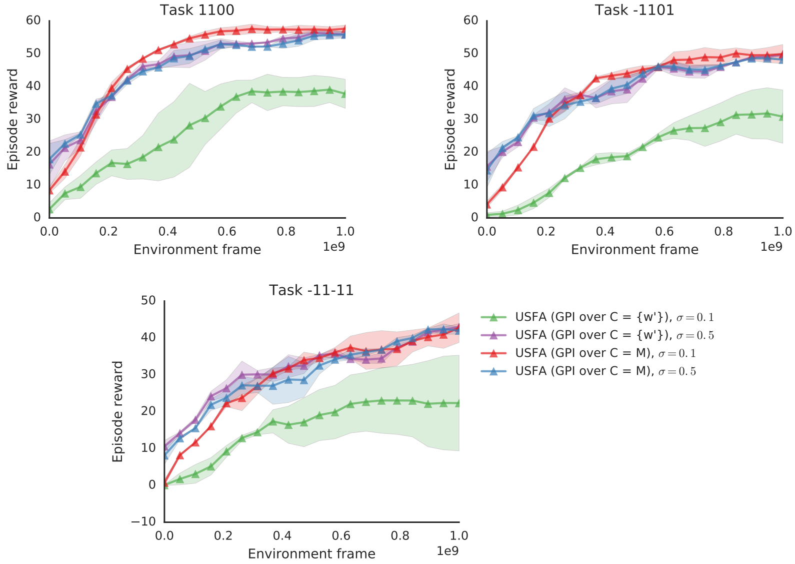


Figure 6.4.8: Generalisation performance on sample test tasks $\mathbf{w}' \in \mathcal{M}'$ after training on $\mathcal{M}$, with $\mathcal{D}_{\mathbf{z}} = \mathcal{N}(\mathbf{w}, \sigma\mathbf{I})$, for $\sigma = 0.1$ and $\sigma = 0.5$ (larger coverage of the $\mathbf{z}$ space). Average over 3 runs.

6.5 Conclusion

6.5.1 Related Work

Multitask RL is an important topic that has generated a large body of literature. Solutions to this problem can result in better performance on the training set (Espeholt et al., 2018), can improve data efficiency (Teh et al., 2017) and enable generalisation to new tasks. For a comprehensive presentation of the subject please see (Taylor and Stone, 2009) and (Lazaric, 2012) and references therein.

There exist various techniques that incorporate tasks directly into the definition of the value function for multitask learning (Kaelbling, 1993; Ashar, 1994; Sutton et al., 2011). UVFAs have been used for zero-shot generalisation to combinations of tasks (Mankowitz et al., 2018; Hermann et al., 2017), or to learn a set of fictitious goals previously encountered by the agent (Andrychowicz et al., 2017).

Many recent multitask methods have been developed for learning subtasks or skills for a hierarchical controller (Vezhnevets et al., 2017; Andreas et al., 2017; Oh et al., 2017). In this context, (Devin et al., 2017) and (Heess et al., 2016) proposed reusing and composing sub-networks that are shared across tasks and agents in order to achieve generalisation to unseen configurations. (Finn et al., 2017) uses meta-learning to acquire skills that can be fine-tuned effectively. Sequential learning and how to retain previously learned skills has been the focus of a number of investigations (Kirkpatrick et al., 2017; Rusu et al., 2016). All of these works aim to train an agent (or a sub-module) to generalise across many subtasks. These can be great use-cases for USFAs.

USFAs use a UVFA to estimate SFs over multiple policies. The main reason to do so is to apply GPI, which provides a superior zero-shot policy in an unseen task. There have been previous attempts to combine SFs and neural networks, but none of them used GPI (Kulkarni et al., 2016; Zhang et al., 2017). Recently, (Ma et al., 2018) have also considered combining SFs and UVFAs. Although this work is superficially similar to ours, it differs quite a lot in the details. Specifically, (Ma et al., 2018) do not consider GPI, and restrict the use of UVFAs to building a task-dependent set of SFs.

6.5.2 Summary

In this chapter, we presented USFAs, a generalisation of UVFAs through SFs. The combination of USFAs and GPI results in a powerful model capable of exploiting the same types of regularity exploited by its precursors: structure in the value function, like UVFAs, and structure in the problem itself, like SF & GPI. This means that USFAs can not only recover their precursors but also provide a whole new spectrum of possible models in between them. We described the choices involved in training a USFA and discussed the trade-offs associated with each alternative. To make the discussion concrete, we presented a specific way to train USFAs and put it to the test on a complex domain in which the agent has to navigate in a three-dimensional environment using only images as observations. We showed how all instantiations of our model are able to generalise to unseen tasks and discussed how we can emphasise different behaviours with simple changes in the training process.

7
Conclusions

In this thesis, we explored the scenario of a learning agent, ’living’ in a persistent environment where it needs to figure out how to behave in order to optimise a set of reward signals. We argue that this is a very natural scenario most intelligent beings are subjected to. Just to survive, animals need to learn how to navigate their surroundings, find food and water, build shelters, avoid predators, interact and cooperate with their peers. All of these behaviours are very intricate and diverse. Moreover, at the micro-scale of individual actions, all of these correspond to elaborate policies made up of smaller building blocks, like moving into a certain direction, jumping, swimming, grasping, grabbing or manipulating objects, etc. Being able to master and compose them is truly a hallmark of intelligence.

The world around us is very complex, probably beyond our abilities to fully comprehend or accurately model it. Nevertheless, we are able to meaningfully interact with our environment, predict outcomes, actively change and modify our surrounding, engage and explore in a targeted manner. This is because we are able to learn representations of the world that are sufficient for us to achieve the above feats. And this is in part what we wanted to investigate in this work: what kind of representations could support this complex, multitask (or multi-reward for us), multi-policy learning? Of course, we can think about treating all of these (RL) problems independently and learning solutions separately for all of them. Nevertheless, as Rich Caruana mentioned in his thesis, perhaps the common structure and similarities across these tasks are precisely the reasons behind our ability to learn them so efficiently, from so little experience, when compared with current state-of-the-art machine learning systems.

In the first part of this thesis, we looked at agents trying to learn policies optimising for a fixed set of rewards, given some restrictive number of per task interactions. This is a common scenario in many practical RL applications, such as designing medical treatments (Panuccio et al., 2013; Koedinger et al., 2013; Liu et al., 2018) or optimising educational curricula (Thomas and Brunskill, 2016). In these settings, the data is very scarce and expensive to procure, thus combining multiple sources of data, recorded under different behaviour policies, trying to optimise for different things is a standard scenario. In the first part, we investigated learning optimal value functions for a collection of tasks by leverage all the data collected across tasks.

Our first attempt in doing so was to extend the popular fitted-Q iteration algorithm to the multitask case and we proposed to build a common representation of the value functions being learnt (Chapter 3). To make this more concrete, we chose to build this shared representation using a popular method in linear multitask learning introduced in (Argyriou et al., 2008). We then showed empirically that the resulting algorithm indeed led to better performance under restricted budgets and one could achieve positive transfer between tasks using this method. This is by no means the only choice. In a sense, any multitask regression method that trains a common representation could be swapped in here, as discussed in Section 3.4.2. One of the benefits of opting for a linear parametrisation of the value functions was that we could easily inspect the learnt shared representation. This revealed a very compressed, informative representation that essentially encoded navigation patterns between rooms. This trained representation captures the main features needed to represent all optimal value function in this class.

In the fourth chapter, we considered the same experimental setting and explored a similar idea of jointly learning a shared linear representation for our value functions. In this chapter, we look at exploring least square methods for (approximate) policy iteration. In this case, one can observe that the policy evaluation sub-procedure can naturally be cast as a multitask problem, where we aim to build the evaluation of the n policies under their corresponding n tasks. This problem can be tackled in various ways. The first one we explored was a simple least-squares projected policy evaluation method. In this case, for linear representation, it turns out that the solution factorises into a dynamics-only model and a reward-dependent term. Due to this factorisation, under our persistent environment assumption, we can compute the first term from all of the experience, across tasks. And we have shown empirically, that there are situations in which this simple transfer via samples can be very effective. Our second approach to the multitask multi-policy evaluation problem was via Bellman residual minimisation (BRM). In particular, we cast the n residual minimisation problems as a multitask regression problem and employed the same method as in Chapter 3 to train a joint representation that would support the multi-policy evaluations. As discussed in Section 4.6.2, it is important to note that the nature of the regression problems we ended up solving in these two chapters is quite different, although at the end of this process they both should result in a common representation that support optimal value functions. As in the previous experiments, we show that phrasing and solving the joint problem in this way leads to better performance, under small sample sizes.

In the second part of this thesis, we looked at a different transfer scenario, more in-line with the way we transfer knowledge to speed-up our learning of new, but related tasks (Lazaric, 2012). The representations investigated in this second part are *successor representations*, which have been shown to achieve a computationally efficient middle ground, between model-free and model-based methods (Lehnert and Littman, 2019; Botvinick and Weinstein, 2014), and to model closely the decision making process for biological agents, like rats or even humans (Russek et al., 2017). One of the other benefits of this representation is that it exploits fully the persistent environment assumption, leveraging explicitly the shared dynamics. It is also a representation compatible with more powerful functional approximation classes, like deep neural nets, which allowed us to draw on the recent successes in deep RL. This representation enables us to perform a generalised policy evaluation step within a linear class of reward functions. This ability paired with generalised policy improvement (GPI), leads to a very natural transfer framework in behaviour space (Barreto et al., 2017).

In Chapter 5, we proceed to extend this framework in two ways. First, we extended the above setup to deal with reward functions outside the assumed span of a given feature set. We also show that it is possible to provide guarantees and characterise the quality of the resulting zero-shot policy, on a new task. Based on this we can clearly see that the quality of this transfer is based on i) the underlying reward features and how well they will approximate the current reward signal, as well as ii) the applicability of previously learnt policies to our current task. Thus the problem of specifying these reward dimensions becomes of great importance. Specifically: How could one come up, a priori or build from experience, a set of features that describe the variability of one's reward class? This is precisely where our second contribution lies. Specifically, by looking at the associated approximation from a slightly different angle, we show that one can replace the features with actual rewards that the agent has experienced so far. This is without loss of generality and makes it possible to apply SF & GPI online at scale, bypassing, in a sense, the discovery problem. Nevertheless, it is worth noting that other works (Machado et al., 2017b; Ramesh et al., 2019; Janz et al., 2019) have tried to build these features online while training the successor features (SFs). Although that can be done, we found this setting to be somewhat unstable, as the optimisation has to chase a constantly moving target. Moreover, small corrections in these reward features might lead to a large change the corresponding SFs, as these have to integrate in time all these small adjustments in the reward basis. In contrast, our approach to this problem allows us to train on stationary rewards for each of the task.

In the last chapter, we explored extending this work in a different dimension. In particular, we assumed we do have the right underlying features to span all rewards an agent would ever need to perform. This can be the case if these features are given by a user who would then use them to specify the task in this pre-described language, or we can be in the latter stages of a long-life learning scenario where the agent has already learnt what those reward features would be – building these is something we discussed in Chapter 5. In this scenario, we ask the question: how could we learn most effectively in order to generalise to all task in this class? To do so, we build on work in universal and general value functions (Schaul et al., 2015a; Sutton et al., 2011), to obtain a parametric generalisation in the successor feature space. Unlike previous work, the generalisation problem we are dealing with is one over multiple policies, but under the same reward signals – these being the reward features discussed above. This tends to be a simpler and more structured generalisation problem as it aims to generalise over policy evaluations, in the same MDP. In the proposed model, USFAs, this parametric generalisation is then paired with the direct transfer enabled by GPI, to essentially get the best of both worlds. The last contribution here is in the training procedure, where we increase our sample efficiency and potential for generalisation by using all the experience generated, to train off-policy for a collection of sampled fictitious tasks. This enables us to train many policies (and corresponding SFs), by 'pretending to be on these tasks' without ever having to execute them in the environment. This agent enjoyed the transfer properties studied in the first part of this thesis, by training jointly under multiple fictitious tasks, as well as the direct transfer in behaviour space enabled by the successor features and GPI. The resulting agent was tested in a complex 3D navigation task where the agent needs to pick up and/or avoid a set of objects. In the experiments, we have seen that these two kinds of transfer can be very different, yet complementary to each other.

Limitations and Future Directions

The material presented in this thesis can naturally be extended in a number of directions. In the following we will discuss some of these avenues, looking at applicability to different scenarios, algorithmic extensions and further studies that were left as future work.

In the first part of this thesis, we considered the offline data scenario. This was motivated mainly by real-world application scenarios, where the cost of generating new experiences by interacting in the real-world can be prohibitively expensive – domains like medicine, education, robotics. Nevertheless, the proposed methods can be straightforwardly applied to the online settings as hinted in Section 3.3.4. The performance here will now be intertwined with the exploration policy, that can now take into account the learning done by the system so far. But this could also allow the system to test its current policies and actively sample to collect better experiences to validate/invalidate its generalisations. If data can be gathered under such a learnt policy safely, a system that relies on learning from both on-policy data and historical data might exhibit the best properties.

In terms of learning methods, in this first part, we chose to concentrate on building a linear shared representation for our inferred value functions and we present one particular way of doing so. But of course there are many other ways to approach the multitask regression problems encountered in Algorithms 3.1, 4.3 and 4.4. Different MDP classes might benefit from different structural assumptions one might bake into the optimisation and in principle, any multitask regressor can be swapped in here. Along a similar line, as seen in the other chapters, maybe a more powerful function approximation could be employed to learn a shared representation across value functions. One natural candidate here are deep neural nets. Unfortunately, they tend to require a lot more experience to learn. Considering the strict constraints on the sample budget we considered here, it is not also clear how to properly regularise this functional class to induce good generalisation under limited number of samples. More generally, learning representations in deep RL under a multitask setting has proven to be challenging (Rusu et al., 2016; Kirkpatrick et al., 2017; Teh et al., 2017) and rarely delivering on the promise of positive transfer. This can stem from a few sources. A general issue in multi-task learning is that a balance must be found between the needs of multiple tasks competing for the limited resources of a single learning system. This is not particular to RL, but the nonstationary of the RL setting, even in the single task, complicates this problem further. Moreover, the usual dependency between the exploratory policy and the competency of the agent can lead to feedback cycles that can detrimentally affect the learning (Schaul et al., 2019). As future work, it would be interesting to look into different functional classes and investigate more closely the interaction between this choice of the approximations and the RL algorithm.

In the second part of this dissertation, we studied the possibility of transferring knowledge to a new task and investigated the problem of task generalisation in a persistent environment. Our way of representing and building knowledge in this part was through successor features (SFs). In a nutshell, the transfer to a different task happens through our ability to reassess a previously learnt behaviour under a new task. This reassessment is possible zero-shot, through the SFs, within the span of the reward features on which the SFs were trained. But one can argue that this really embodies a more general and powerful principle of transferring knowledge in RL, generalising the inner step of PI. If we can cheaply perform a general policy evaluation step where we can reassess previously learnt policies on our current task, this can then be paired with GPI to immediately provide us with a policy that is guaranteed to be better than any of the evaluated policies. In this view, one can argue that it worth exploring other ways of enabling such a fast re-evaluation step.

Nevertheless, in general, this step (policy evaluation) tends to be quite expensive, so we would want to limit the number of re-evaluations one needs to do. This leads us to a natural question: which policies should one learn and which policies should one use for this evaluation step? This is a very important, recurring question that remains largely unanswered. What we have seen so far in our experiments is that this choice can have drastic effects on the performance – Section 5.4 and 6.4. This makes sense, as the collection of policies we consider here are the basis of the improvement step. Thus we would want policies that are most informative to improvement. It turns out that this is a combined effect of how aligned these policies are with the new task and how precise the re-evaluation would be. This is marvellously exemplified in our last experiment Section 6.4.3 and more formally justified by Proposition 6.1. One of the advantages of USFAs is that they provide us with a choice here. We can sample, for every task, which of these points in policy space we think are likely to result in a better improvement step. And in the few choices we explored, we can definitely see the trade-off anticipated by Proposition 6.1. This choice depends also on the training procedure, in particular, which policies have we learnt about more in training. Again, for USFA that is a choice, as for each trajectory of experience, we are going to sample multiple other tasks for which we aim to learn optimal policies. We explored two options here: learning on tasks close to the one that generated the data (due to the off-policy updates, these tasks might benefit the most from this experience) and learning almost uniformly across the set of tasks we aim to generalise to. These are two extremes that will interact greatly with the decision of which policies to now sample for GPI, as the generalisation properties of these two training procedures, will be very different: in one case we have trained clusters of policies around our behaviours, in the other we have trained across broader areas of the policy space but maybe with less relevant data. Both of these sampling mechanisms, in learning and for acting, were explored only for a few initial design choices and deserve deeper investigations in their properties and their interaction.

Finally, one critical, overarching question worth a lot more investigation is which tasks should be learnt? This will dictate what are the appropriate learning methods, what are the appropriate assumptions, what structure can be exploited, what kind of transfer can we enable or how safe is the transfer of information (negative transfer, potential for interference, etc). This becomes particularly relevant in a scenario like the one in Chapter 6 where the agent can choose which tasks to focus on. This is just an example of a more general principle in which our agent could come up with their own tasks in order to build a better representation of the world, achieve better agency and control, explore in a more structured way, in the attempt to prepare itself for generalisation problems down the line. Initial studies on the benefits of auxiliary tasks (Jaderberg et al., 2016), self-supervision (Pathak et al., 2017; Kahn et al., 2018) and learning general value functions (Sutton et al., 2011; Andrychowicz et al., 2017; Riedmiller et al., 2018; Borsa et al., 2019), seem to suggest that this is a particularly promising way to approach the representation learning problem.

A
Appendix: Omitted Proofs

A.1 Background: Reinforcement Learning

A.1.1 Proof for Greedy Policy Improvement Theorem

In this section, we include a proof of the Greedy Policy Improvement theorem, restated below in Theorem 1.1. As explained in the main text, this is crucial step in many of the algorithms studied in this work, as well as a precursor the policy improvement step given by GPI (Theorem 1.2).

Theorem 1.1: Greedy Policy Improvement

Given an MDP, $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, a policy π and its associated action-value Q^π, then if we consider the greedy policy $\pi_{greedy} = \arg\max_a Q^\pi(s,a)$ we have that:

$$Q^{\pi_{greedy}}(s,a) \geq Q^\pi(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$$

with equality satisfied only when π can not be improved any more: $\pi \in \arg\max_\mu Q^\mu(s,a)$, $\forall s \in \mathcal{S}, a \in \mathcal{A}$.

Proof. We will prove that acting greedy with respect to state-action value functions, leads to a new $Q^{\pi_{greedy}}(s,a)$ that improves as each (s,a) pair value. To do that, let us first observe that:

$$Q^\pi(s, \pi_{greedy}(s)) = \max_{a \in \mathcal{A}} Q^\pi(s,a) \geq Q^\pi(s, \pi(s)) := \mathbb{E}_{a \sim \pi}[Q^\pi(s,a)].$$

Thus:

$$\begin{aligned} T^{\pi_{greedy}} Q^\pi(s,a) &= r(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{a' \in \mathcal{A}} \pi_{greedy}(s',a') Q^\pi(s',a') \\ &= r(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^\pi(s',a') \\ &\geq r(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(s',a') Q^\pi(s',a') \\ &\geq T^\pi Q^\pi(s,a) = Q^\pi(s,a) \end{aligned}$$

Now we know by the fix point theorem that repeated application of $T^{\pi_{greedy}}$ will convergence to its unique fixed point $Q^{\pi_{greedy}}$, thus we get:

$$Q^{\pi_{greedy}} = \lim_{k\to\infty} \left(T^{\pi_{greedy}}\right)^k [Q^\pi] \geq \lim_{k\to\infty} \left(\left(T^{\pi_{greedy}}\right)^{k-1} T^\pi\right) [Q^\pi] \geq \lim_{k\to\infty} \left(T^\pi\right)^k [Q^\pi] = Q^\pi \tag{A.1}$$

That is, we are guaranteed not to decrease the state-action value function by acting greedily. Note also that equality is satisfied if and only if $T^\pi Q^\pi(s,a) = T^{\pi_{greedy}} Q^\pi(s,a), \forall (s,a) \in \mathcal{S}\times\mathcal{A} \Rightarrow \left(T^{\pi_{greedy}}\right)^k [Q^\pi] = \left(T^\pi\right)^k [Q^\pi] \Rightarrow \lim_{k\to\infty} \left(T^{\pi_{greedy}}\right)^k [Q^\pi] = \lim_{k\to\infty} \left(T^\pi\right)^k [Q^\pi] \Rightarrow Q^{\pi_{greedy}} = Q^\pi \Rightarrow$ the two policies are equivalent (in terms of the value function).

Similarly, we can very easily prove that the greedily constructed policy will improve the state value function at each step:

$$\begin{aligned}
V^\pi(s) &= \sum_a \pi(s,a) Q^\pi(s,a) \leq \max_{a\in\mathcal{A}} Q^\pi(s,a) = Q^\pi(s, \pi_{greedy}(s)) \\
&\leq Q^{\pi_{greedy}}(s, \pi_{greedy}(s)) && \left(\text{proved above } Q^{\pi_{greedy}}(s,a) \geq Q^\pi(s,a), \forall s, a\right) \\
&= V^{\pi_{greedy}}(s) && \left(V^{\pi_{greedy}}(s) = Q^{\pi_{greedy}}(s, \pi_{greedy}(s))\right)
\end{aligned}$$

□

A.1.2 TD(λ): Sketch for the Derivation of the Backward View

In this section we revisit TD learning, in particular TD(λ). As a reminder, TD methods are a combination of Monte Carlo learning and dynamic programming(DP) divide-and-conquer ideas explored before. Like MC methods, TD methods can learn directly from *sampled experience* without a full model of the environment. And similar to DP, TD methods will update estimates based on partial (learnt) solutions, without having to wait for the final outcome of an episode.

The simplest version of temporal-difference learning, TD(0), replaces the return G_t with an estimate of it, $G_{t:t+1} \approx R_{t+1} + \gamma V(S_{t+1})$, given by the Bellman Expectation Equation. Consequently the update becomes:

$$V(S_t) \leftarrow V(S_t) + \alpha(\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{G_{t:t+1}} - V(S_t)) \tag{A.2}$$

where we will commonly refer to $G_{t:t+1} = R_{t+1} + \gamma V(S_{t+1})$ as the *TD target* and we can define the *TD error* as:

$$\delta_t \quad := \quad G_{t:t+1} - V(S_t) = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

Something in-between the full history return used by the MC update and the one-step TD target, is the n-step return:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) \tag{A.3}$$

Note that by setting $n = 0$, we recover TD(0) and if we set $n = \infty$ we recover the Monte Carlo update in Eq. 2.25. Furthermore, we can think about combining these n-step targets to get a new target. We define the *λ-return*, G_t^λ as a linear combination of the n-step returns:

$$G_t^\lambda = \sum_{n=1}^{\infty} a_n G_{t:t+n} \tag{A.4}$$

with $a_n = (1 - \lambda)\lambda^n$, and where $\lambda \in (0, 1)$. In this section we are going to look into the backward view of TD(λ) which provides us with an update rule that can be immediately be applied after seeing just one additional transition, as we did in TD(0). In doing so, we can dynamically assign credit to the most recently and the most frequently visited states. The resulting algorithm was described in Algorithm 2.4 and it relies on re-writing the episode TD error, $G_t^\lambda - V(S_t)$, in terms of the per-step TD errors, $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$, and weighting them accordingly, using eligibility traces. In the following, we provide a sketch of this result. Let us first look at the episodic TD error:

$$\begin{aligned}
G_t^\lambda - V(S_t) &= \sum_{n=1}^{\infty} a_n G_{t:t+n} - V(S_t) \\
&= \sum_{n=1}^{\infty} a_n \left(G_{t:t+n} - V(S_t)\right) \qquad \text{as } \textstyle\sum_{n=1}^{\infty} a_n = 1 \qquad \text{(A.5)}
\end{aligned}$$

We can now take a closer look at the individual n-step TD errors:

$$\begin{aligned}
G_{t:t+n} - V(S_t) &= R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) - V(S_t) \\
&= \sum_{k=1}^{n} \gamma^{k-1} R_{t+k} + \gamma^n V(S_{t+n}) - V(S_t) \\
&= \sum_{k=1}^{n-1} \left(\gamma^{k-1} R_{t+k} + \gamma^k V(S_{t+k}) - \gamma^k V(S_{t+k})\right) + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) - V(S_t) \\
&= \sum_{k=1}^{n} \gamma^{k-1} R_{t+k} + \sum_{k=1}^{n} \gamma^k V(S_{t+k}) - \sum_{k=0}^{n-1} \gamma^k V(S_{t+k}) \\
&= \sum_{k=1}^{n} \gamma^{k-1} R_{t+k} + \sum_{k=1}^{n} \gamma^k V(S_{t+k}) - \sum_{k=1}^{n} \gamma^{k-1} V(S_{t+k-1}) \\
&= \sum_{k=1}^{n} \gamma^{k-1} \underbrace{\left(R_{t+k} + \gamma V(S_{t+k}) - V(S_{t+k-1})\right)}_{\delta_{t+k-1}} = \sum_{k=0}^{n-1} \gamma^k \delta_{t+k} \qquad \text{(A.6)}
\end{aligned}$$

Thus, combining Eq.A.5 and Eq.A.6 we get:

$$G_t^\lambda - V(S_t) = \sum_{n=1}^{\infty} a_n \left(\sum_{k=0}^{n-1} \gamma^k \delta_{t+k}\right) = \sum_{n=0}^{\infty} \sum_{k=0}^{n} a_n \gamma^k \delta_{t+k} = \sum_{k=0}^{\infty} \sum_{n=0}^{k} a_n \gamma^k \delta_{t+k}$$

where the last equality is obtained simplify by re-arranging the terms in the two summations. Thus:

$$G_t^\lambda - V(S_t) = \sum_{k=0}^{\infty} \underbrace{\left(\sum_{n=0}^{k} a_n \gamma^k\right)}_{e_{t+k}(S_t) \text{ as defined in Eq. 2.30}} \delta_{t+k} \qquad \text{(A.7)}$$

A.2 Multitask Learning: Learning Shared Linear Representations

In this section, we continue the discussion in Section 3.2.1 and provide more details of the resulting joint optimisation problem we are attempting to solve at each step of value iteration in Algorithm 3.2.

To begin, we restate the problem we are trying to address. Our assumption can be expressed as: $\exists$ a small set of features $\{\psi_i\}_{i=1,N_\psi}$ such that $\forall j$, $Q_j(s,a) \approx \sum_i a_{ji}\psi_i(s,a)$. And in this section, we restrict our attention to features that are given as a linear combination of the input feature space $\Phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^d$ for each state and action. Thus we assume a reparametrisation of the problem as follows: there exists a transformation matrix U ($d \times d$ orthogonal matrix) such that the induced feature set $\psi(s,a) = U^T\Phi(s,a)$ induces a sparse representation across Q_j-s. Thus, in the new representation

$$Q_j(s,a) \approx \sum_i a_{ji}\psi_i(s,a) = \langle a_j, U^T\Phi(s,a)\rangle,$$

we would like the vectors a_j to be sparse (that is the approximation needs only a few of these features to represent the desirable value functions) .

Since we want the above representation to be sparse, for each task j, one could try to solve an optimisation problem of the following type:

$$Q_j^{(k)} = \operatorname*{argmin}_{f_j=\langle a_j, U^T\Phi\rangle} \left[\sum_j \mathcal{L}_{\mathtt{D}_j}\left(f_j(s,a), y_j^{(k)}\right) + \|a_j\|_1^2 \right] \qquad \text{(A.8)}$$

where $\mathtt{D}_j$ is the data set of experiences (s,a,s',r_j) recorded on task j and $y_j^{(k)}(s,a) = r_j + \gamma max_b Q_j^{(k-1)}(s',b) \approx T^* Q_j^{(k-1)}(s,a)$ is the one-step target at iteration k. We denote by superscript (k) the values of quantities at iteration k in the iterative process.

Nevertheless, it has been proposed that we can look at such problems jointly, under a similar sparsity regulariser that *also promotes features sharing among tasks*. This leads to a joint optimisation problem of the form:

$$
\begin{aligned}
(U^*, A^*) &= \arg\min[\varepsilon(U, A) + \lambda\mathcal{H}(A)] && \text{(A.9)} \\
&= \arg\min_{A,U} \left[\underbrace{\sum_j \mathcal{L}_{\mathrm{D}_j} \left(\langle a_j, U^T \Phi(s, a) \rangle, y_j^{(k)} \right)}_{\varepsilon(U,A)} + \lambda\mathcal{H}(A) \right] && \text{(A.10)}
\end{aligned}
$$

where $A = [a_1, \cdots, a_j, \cdots, a_J]$, $\mathcal{H}(A) = \lambda ||A||^2_{2,1}$ and by $\varepsilon(U, A)$ we denoted the empirical loss over all tasks. It was shown in (Argyriou et al., 2008) that A.10 is equivalent to solving the following *convex* problem:

$$
(W^*, D^*) = \arg\inf \left[\varepsilon(W) + \lambda Tr(W^T D^{-1} W) \right] \text{ s.t. } D \succ 0, Tr(D) \leq 1 \qquad \text{(Problem 2)}
$$

where we denote by $\varepsilon(W)$ the joint empirical loss in Eq. A.8. We refer the reader to the original work for the proof of this equivalence and further details on general conditions of these results (Argyriou et al., 2007, 2008).

More precisely one can show that if (U^*, A^*) is a solution for Eq. A.10 then $W^* = U^* A^*$ is an optimal solution for Problem 2 (Theorem 1 in (Argyriou et al., 2008)). More over given a fixed W, the optimal D^* that minimises the above equation is given by:

$$
D^*(W) = \frac{(WW^T)^{\frac{1}{2}}}{Tr(WW^T)^{\frac{1}{2}}} \qquad \text{(A.12)}
$$

We provide a proof for this last statement, as it will be used in the design of the alternating minimisation algorithm used to tackle Problem 2.

Proof. (Adapted from (Argyriou et al., 2008))

Consider the partial optimisation problem:

$$
D^* = \arg\inf \left[\varepsilon(W) + \lambda Tr(W^T D^{-1} W) \right] \text{ s.t. } D \succ 0, Tr(D) \leq 1 \qquad \text{(Problem 2b)}
$$

which for a fixed W, is equivalent to:

$$
D^* = \arg\inf \left[Tr(W^T D^{-1} W) \right] \text{ s.t. } D \succ 0, Tr(D) \leq 1 \qquad \text{(A.13)}
$$

Now, let us consider the corresponding Lagrangian:

$$\mathsf{L}(\alpha_1, \alpha_2) = \left[Tr(W^T D^{-1} W)\right] - \alpha_1 \left(Tr(D) - 1\right) + \alpha_2 \left(x^T (D) x - 0\right) \tag{A.14}$$

and let us compute the derivative with respect to this objective:

$$\begin{aligned}
\frac{\partial}{\partial D} Tr(W^T D^{-1} W) &= \left[\frac{\partial}{\partial W^T D^{-1} W} Tr(W^T D^{-1} W)\right] \left[\frac{\partial}{\partial D^{-1}} (W^T D^{-1} W)\right] \left[\frac{\partial}{\partial D} D^{-1}\right] \\
&= [Id] \left[WW^T\right] \left[-D^{-2}\right] \\
&= -WW^T D^{-2}
\end{aligned}$$

Thus the derivative of the whole Lagrangian is:

$$\frac{\partial}{\partial D} \mathsf{L}(\alpha_1, \alpha_2) = -WW^T D^{-2} - \alpha_1 \cdot Id + \alpha_2 \cdot xx^T$$

Putting this to 0, we explore the implications:

- If $\alpha_2 \neq 0 \Rightarrow xDx^T = 0, \forall x \in \mathbb{R}^d \Rightarrow$ impossible
- If $\alpha_1 \neq 0 \Rightarrow Tr(D) = 1$

$$WW^T D^{-2} = \alpha_1 \cdot Id \Rightarrow D^2 = \frac{1}{\alpha_1} WW^T \tag{A.15}$$

$$\Rightarrow Tr(D) = Tr\left(\frac{1}{\alpha_1^{\frac{1}{2}}} (WW^T)^{\frac{1}{2}}\right) = \frac{1}{\alpha_1^{\frac{1}{2}}} Tr\left((WW^T)^{\frac{1}{2}}\right) = 1 \Rightarrow \alpha_1^{1/2} = Tr\left((WW^T)^{\frac{1}{2}}\right)$$

$$\Rightarrow D = \frac{1}{\alpha_1^{\frac{1}{2}}} \left((WW^T)^{\frac{1}{2}}\right) = \frac{\left((WW^T)^{\frac{1}{2}}\right)}{Tr\left((WW^T)^{\frac{1}{2}}\right)} \tag{A.16}$$

□

As prefaced above, we are going to be use the above result as part of an alternating optimisation procedure to tackle Problem 2. In particular, we are going to use the fact the Problem 2 gives rise to a convex problem, in both D and W, when one of these variables is kept fixed. Thus, the idea of the algorithm is to alternate between solving these two minimisation problems. It is worth noting that convergence of this algorithm has been shown for the perturbed objective function (Argyriou et al., 2008):

$$\mathcal{R}_{\zeta}(W,D) = \left[\varepsilon(W) + \lambda Tr(W^T D^{-1} W^T + \zeta Id)\right] \tag{A.17}$$

with a small perturbation parameter $\zeta > 0$. Note that as $\zeta \rightarrow 0$, $\mathcal{R}_{\zeta}(W,D)$ recovers the original problem. Although convergence is not guaranteed for $\zeta = 0$, in practice we have observed convergence, without needing a decaying schedule for ζ.

Now let us see how we can compute the second step in the proposed alternating optimisation. Keeping D fixed, we get another convex problem and we can now solve for W. It is important to note that the above amounts to a minimisation over w_j that can be carried out independently for each task j as both objectives factorise over tasks:

$$W^* = \arg \inf_{W=[w_{1:J}]} \left[\sum_j \varepsilon(w_j) + \lambda \sum_j \langle w_j, D^{-1} w_j \rangle \right]. \tag{Problem 2a}$$

More specifically, introducing a change in variable $w'_j = D^{-1/2} w_j$ this yields a standard L_2 norm regularisation problem:

$$w_j^* = \arg \inf_{w_j} \left[\mathcal{L}_{\mathrm{D}_j}(\langle w'_j, D^{1/2}\varphi(s,a)\rangle, y_j) + \lambda \langle w'_j, w'_j \rangle \right]. \tag{Problem 2b}$$

Lastly, note that the same multitask representation learning mechanism was used to jointly solve the multiple joint optimisation problems proposed for solving in Chapter 4, as part of Algorithms 4.2- 4.4.

A.3 Transferable Representations in Policy Space

A.3.1 Generalised Policy Improvement(GPI) Proof

For the convenience of the reader we restate GPI theorem below. The result has originally proven in (Barreto et al., 2017). Here, we restate the result and adapt the proof.

Theorem 1.2: Generalised Policy Improvement(GPI) (Barreto et al., 2017)

(Generalised Policy Improvement) Let $\pi_1, \pi_2, ..., \pi_n$ be n decision policies and let $\tilde{Q}^{\pi_1}, \tilde{Q}^{\pi_2}, ..., \tilde{Q}^{\pi_n}$ be approximations of their respective action-value functions such that

$$|Q^{\pi_i}(s,a) - \tilde{Q}^{\pi_i}(s,a)| \leq \varepsilon \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}, \text{ and } i \in \{1,2,...,n\}.$$

Define

$$\pi(s) \in \operatorname*{argmax}_a \max_i \tilde{Q}^{\pi_i}(s,a).$$

Then,

$$Q^{\pi}(s,a) \geq \max_i Q^{\pi_i}(s,a) - \frac{2}{1-\gamma}\varepsilon$$

for any $s \in \mathcal{S}$ and any $a \in \mathcal{A}$, where Q^{π} is the action-value function corresponding to π.

Proof. Let $\tilde{Q}_{max}(s,a) = \max_i \tilde{Q}^{\pi_i}(s,a), \forall a \in \mathcal{A}, s \in \mathcal{S}$. Then, let us look at one application of the Bellman expectation operator T^{π} for our GPI policy:

$$\begin{aligned}
T^{\pi}\tilde{Q}_{max}(s,a) &= r(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s')\tilde{Q}_{max}(s',a') && \text{(A.18)} \\
&= r(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} \tilde{Q}_{max}(s',a') && \text{(A.19)} \\
&= r(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} \max_i \tilde{Q}^{\pi_i}(s',a') && \text{(A.20)} \\
&\geq r(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^{\pi_i}(s',a') - \gamma\varepsilon, \forall i && \text{(A.21)} \\
&\geq r(s,a) + \gamma \sum_{s'} P(s'|s,a) Q^{\pi_i}(s',\pi_i(s')) - \gamma\varepsilon, \forall i && \text{(A.22)} \\
&= T^{\pi_i}Q^{\pi_i}(s,a) - \gamma\varepsilon = Q^{\pi_i}(s,a) - \gamma\varepsilon, \forall i && \text{(A.23)}
\end{aligned}$$

If this true for all i, then: $T^{\pi}\tilde{Q}_{max}(s,a) \geq \max_i Q^{\pi_i}(s,a) - \gamma\epsilon \geq \max_i \tilde{Q}^{\pi_i}(s,a) - \epsilon - \gamma\epsilon$. And thus, by the monotonicity and contraction of the Bellman expectation operator, we have:

$$Q^{\pi}(s,a) = \lim_{k\to\infty}(T^{\pi})^k\tilde{Q}_{max} \geq \tilde{Q}_{max} - \epsilon\frac{1+\gamma}{1-\gamma} \geq \max_i Q^{\pi_i}(s,a) - \epsilon - \epsilon\frac{1+\gamma}{1-\gamma}$$

Thus:

$$Q^{\pi}(s,a) \geq \max_i Q^{\pi_i}(s,a) - \frac{2\epsilon}{1-\gamma}$$

□

A.3.2 GPI as a Strictly Improving Step

In addition, the below result shows that GPI is indeed a valid, potentially better, improvement step that can be used in a policy iteration like procedure.

Lemma 1. *Let Π be a collection of policies and let $Q_r^{\pi_i}$ be the action-value functions on task r. Then the GPI policy $\pi(s) \in \text{argmax}_a \max_i \tilde{Q}^{\pi_i}(s,a)$ is a strict improvement over every policy $\pi_i \in \Pi$, unless Π already contains the optimal policy.*

Proof. Let $Q_{max}(s,a) = \max_i Q^{\pi_i}(s,a), \forall a \in \mathcal{A}, s \in \mathcal{S}$. Then, let us look at one application of the Bellman expectation operator T^{π} for our GPI policy:

$$\begin{aligned} T^{\pi}Q_{max}(s,a) &= r(s,a) + \gamma\sum_{s'}P(s'|s,a)\sum_{a'}\pi(a'|s')Q_{max}(s',a') && \text{(A.24)}\\ &= r(s,a) + \gamma\sum_{s'}P(s'|s,a)\max_{a'}Q_{max}(s',a') && \text{(A.25)}\\ &= r(s,a) + \gamma\sum_{s'}P(s'|s,a)\max_{a'}\max_{i}Q^{\pi_i}(s',a') && \text{(A.26)}\\ &\geq r(s,a) + \gamma\sum_{s'}P(s'|s,a)\max_{a'}Q^{\pi_i}(s',a') && \text{(A.27)}\\ &\geq r(s,a) + \gamma\sum_{s'}P(s'|s,a)Q^{\pi_i}(s',\pi_i(s')) && \text{(A.28)}\\ &= Q^{\pi_i}(s,a), \forall i && \text{(A.29)} \end{aligned}$$

□

Thus we have that $T^{\pi}Q_{max} \geq Q^{\pi_i}, \forall i$. Note that this implies $T^{\pi}Q_{max} \geq Q_{max}$, which, due to the monotonicity of T^{π}, leads to:

$$Q^{\pi_i} \leq Q_{max} \leq \lim_{k \to \infty} (T^{\pi})^k Q_{max} = Q^{\pi}, \forall i \tag{A.30}$$

Now, let us investigate what are the conditions under which the above equalities hold. Firstly, we need that $Q_{max} = Q^{\pi_i}$, which, but the definition of Q_{max} can only happen if one policy in Π dominates all others on this task. In this case, the GPI improvement step reduces itself to a greedy improvement step over this dominating policy π_{i^*}. And we know that this greedy step will result in a *strict improvement*, unless the starting policy π_{i^*} is already optimal. What the above really says is that enforcing the second part of the equality, equates to $Q_{max} = (T^{\pi})^k Q_{max} \Rightarrow Q^{\pi_{i^*}} = (T^{\pi_{greedy}})^k Q^{\pi_{i^*}}, \forall k \Rightarrow Q^{\pi_{i^*}} = Q^{\pi_{greedy}}$, where $\pi_{greedy} = \arg\max_a Q^{\pi_{i^*}}(s,a)$. This can only happen if $Q^{\pi^{i^*}} = Q^{\pi^*}$ in which case the GPI policy is, as well, optimal as $Q^{\pi} = Q^{\pi^{i^*}} = Q^*$.

A.3.3 USFA Generalisation

In this section, we include the proof for Proposition 6.1, characterising the zero-shot generalisation properties of universal successor features approximators, USFAs (Chapter 6). For convenience, we begin by re-stating this result.

Proposition 1.1

Let $\mathbf{w}' \in \mathcal{M}'$ and let $Q^{\pi}_{\mathbf{w}'}$ be the action-value function of executing policy π on task $\mathbf{w}'$. Given approximations $\{\tilde{Q}^{\pi_{\mathbf{z}}}_{\mathbf{w}'} = \tilde{\boldsymbol{\psi}}(s,a,\mathbf{z})^{\top}\mathbf{w}'\}_{\mathbf{z} \in \mathcal{C}}$, let π be the GPI policy defined in 6.8. Then,

$$||Q^*_{\mathbf{w}'} - Q^{\pi}_{\mathbf{w}'}||_{\infty} \leq \frac{2}{1-\gamma}\left(\min_{\mathbf{z}\in\mathcal{C}}\left(||\boldsymbol{\varphi}||_{\infty}\underbrace{||\mathbf{w}' - \mathbf{z}||}_{\delta_d(\mathbf{z})}\right) + \max_{\mathbf{z}\in\mathcal{C}}\left(||\mathbf{w}'|| \cdot \underbrace{||\boldsymbol{\psi}^{\pi_{\mathbf{z}}} - \tilde{\boldsymbol{\psi}}(s,a,\mathbf{z})||_{\infty}}_{\delta_{\psi}(\mathbf{z})}\right)\right) \tag{A.31}$$

where $Q^*_{\mathbf{w}'}$ is the optimal value of task $\mathbf{w}'$, $\psi^{\pi_{\mathbf{z}}}$ are the SFs corresponding to the optimal policy for task $\mathbf{z}$.

Proof. We first begin by proving an intermediate result, which bounds the distance between two action value functions evaluating the same policy, under *different reward signals* $r_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and respectively $r_j : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, under the same MDP class

$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, ., \gamma \rangle$. Note this result holds for any policy π and the whole family of MDPs $\mathcal{M}$ without any additional assumptions about the structure of the reward signals themselves.

Lemma 2. *Let $\delta_{ij} = \max_{s,a} \left| r_i(s,a) - r_j(s,a) \right|$ and let π be an arbitrary policy. Then,*

$$|Q_i^\pi(s,a) - Q_j^\pi(s,a)| \leq \frac{\delta_{ij}}{1-\gamma}.$$

Proof. Define $\Delta_{ij} = \max_{s,a} |Q_i^\pi(s,a) - Q_j^\pi(s,a)|$. Then,

$$\begin{aligned}
|Q_i^\pi(s,a) - Q_j^\pi(s,a)| &= \left| r_i(s,a) + \gamma \sum_{s'} P(s'|s,a) Q_i^\pi(s', \pi(s')) - r_j(s,a) - \gamma \sum_{s'} P(s'|s,a) Q_j^\pi(s', \pi(s')) \right| \\
&= \left| r_i(s,a) - r_j(s,a) + \gamma \sum_{s'} P(s'|s,a) \left(Q_i^\pi(s', \pi(s')) - Q_j^\pi(s', \pi(s')) \right) \right| \\
&\leq \left| r_i(s,a) - r_j(s,a) \right| + \gamma \sum_{s'} P(s'|s,a) \left| Q_i^\pi(s', \pi(s')) - Q_j^\pi(s', \pi(s')) \right| \\
&\leq \delta_{ij} + \gamma \Delta_{ij}. \qquad \text{(A.32)}
\end{aligned}$$

Since (A.32) is valid for any $s, a \in S \times A$, we have shown that $\Delta_{ij} \leq \delta_{ij} + \gamma \Delta_{ij}$. Solving for Δ_{ij} we get

$$\Delta_{ij} \leq \frac{1}{1-\gamma} \delta_{ij}.$$

□

Lemma 3. *Let $\delta_{ij} = \max_{s,a} \left| r_i(s,a) - r_j(s,a) \right|$. Then,*

$$|Q_i^{\pi_i^*}(s,a) - Q_j^{\pi_j^*}(s,a)| \leq \frac{\delta_{ij}}{1-\gamma}.$$

Proof. To simplify the notation, let $Q_i^i(s,a) \equiv Q_i^{\pi_i^*}(s,a)$. Note that $|Q_i^i(s,a) - Q_j^j(s,a)|$ is the difference between the value functions of two MDPs with the same transition function but potentially different rewards. Let $\Delta_{ij} = \max_{s,a} |Q_i^i(s,a) - Q_j^j(s,a)|$. Then,

$$
\begin{aligned}
|Q_i^i(s,a) - Q_j^j(s,a)| &= \left| r_i(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_b Q_i^i(s',b) - r_j(s,a) - \gamma \sum_{s'} P(s'|s,a) \max_b Q_j^j(s',b) \right| \\
&= \left| r_i(s,a) - r_j(s,a) + \gamma \sum_{s'} P(s'|s,a) \left(\max_b Q_i^i(s',b) - \max_b Q_j^j(s',b) \right) \right| \\
&\leq \left| r_i(s,a) - r_j(s,a) \right| + \gamma \sum_{s'} P(s'|s,a) \left| \max_b Q_i^i(s',b) - \max_b Q_j^j(s',b) \right| \\
&\leq \left| r_i(s,a) - r_j(s,a) \right| + \gamma \sum_{s'} P(s'|s,a) \max_b \left| Q_i^i(s',b) - Q_j^j(s',b) \right| \\
&\leq \delta_{ij} + \gamma \Delta_{ij}. \qquad \text{(A.33)}
\end{aligned}
$$

Since (A.33) is valid for any $s, a \in S \times A$, we have shown that $\Delta_{ij} \leq \delta_{ij} + \gamma \Delta_{ij}$. Solving for Δ_{ij} we get

$$\Delta_{ij} \leq \frac{1}{1-\gamma} \delta_{ij}.$$

□

Knowing this, we can now return to the original inequality we aimed to prove. Let $\pi^*_{\mathbf{w}'}$ be an optimal value function for task $\mathbf{w}'$. And as a reminder we denoted by π the GPI policy induced by a set of approximations $\{\tilde{Q}^{\pi_{\mathbf{z}}}_{\mathbf{w}'} = \tilde{\boldsymbol{\psi}}(s,a,\mathbf{z})^\top \mathbf{w}'\}_{\mathbf{z} \in \mathcal{C}}$ for a set of policies/tasks, $\mathcal{C}$. This policy π will, of course, depend on the choice of the set $\mathcal{C}$, but in order to simplify notation in this proof, we will omit indexing this dependency. From Theorem 5.1 we know that, for any $\mathbf{z} \in \mathcal{C}$, we have

$$Q^{\pi^*_{\mathbf{w}'}}_{\mathbf{w}'}(s,a) - Q^{\pi}_{\mathbf{w}'}(s,a) \quad \leq Q^{\pi^*_{\mathbf{w}'}}_{\mathbf{w}'}(s,a) - Q^{\pi^*_{\mathbf{z}}}_{\mathbf{w}'}(s,a) + \frac{2}{1-\gamma}\varepsilon_{\mathcal{C}} \quad \text{(Theorem 5.1)} \qquad \text{(A.34)}$$

where $\varepsilon_{\mathcal{C}} = \max_{\mathbf{z} \in \mathcal{C}} \|Q^{\pi_{\mathbf{z}}}(s,a) - \tilde{Q}(s,a,\mathbf{z})\| = \max_{\mathbf{z} \in \mathcal{C}} \|Q^{\pi_{\mathbf{z}}}(s,a) - \tilde{\boldsymbol{\psi}}(s,a,\mathbf{z})^T \mathbf{w}'\|$. This leads to:

$$
\begin{aligned}
Q^{\pi^*_{\mathbf{w}'}}_{\mathbf{w}'}(s,a) - Q^{\pi}_{\mathbf{w}'}(s,a) \quad &\leq Q^{\pi^*_{\mathbf{w}'}}_{\mathbf{w}'}(s,a) - Q^{\pi^*_{\mathbf{z}}}_{\mathbf{z}}(s,a) + Q^{\pi^*_{\mathbf{z}}}_{\mathbf{z}}(s,a) - Q^{\pi^*_{\mathbf{z}}}_{\mathbf{w}'}(s,a) + \frac{2}{1-\gamma}\varepsilon_{\mathcal{C}} \\
&\leq |Q^*_{\mathbf{w}'}(s,a) - Q^*_{\mathbf{z}}(s,a)| + |Q^{\pi^*_{\mathbf{z}}}_{\mathbf{z}}(s,a) - Q^{\pi^*_{\mathbf{z}}}_{\mathbf{w}'}(s,a)| + \frac{2}{1-\gamma}\varepsilon_{\mathcal{C}} \\
&\qquad \text{(A.35)}
\end{aligned}
$$

Now, let us break-down each of these terms. Firstly, from Lemma 3, we know that

$$|Q^*_{\mathbf{w}'}(s,a) - Q^*_{\mathbf{z}}| \leq \frac{\max_{s,a} |r_{\mathbf{w}'}(s,a) - r_{\mathbf{z}}(s,a)|}{1-\gamma} = \frac{\|\mathbf{w}' - \mathbf{z}\| \cdot \|\boldsymbol{\varphi}\|_\infty}{1-\gamma}.$$

Similarly, from Lemma 2, we know that

$$|Q^{\pi^*_{\mathbf{z}}}_{\mathbf{z}}(s,a) - Q^{\pi^*_{\mathbf{z}}}_{\mathbf{w}'}(s,a)| \leq \frac{\max_{s,a} |r_{\mathbf{w}'}(s,a) - r_{\mathbf{z}}(s,a)|}{1-\gamma} = \frac{\|\mathbf{w}' - \mathbf{z}\| \cdot \|\boldsymbol{\varphi}\|_\infty}{1-\gamma}.$$

Plugging the above two equations into Eq, A.35, we obtain:

$$Q^{\pi^*_{\mathbf{w}'}}_{\mathbf{w}'}(s,a) - Q^{\pi}_{\mathbf{w}'}(s,a) \quad \leq \frac{2\|\mathbf{w}' - \mathbf{z}\| \cdot \|\boldsymbol{\varphi}\|_\infty}{1-\gamma} + \frac{2}{1-\gamma}\varepsilon_{\mathcal{C}} \tag{A.36}$$

which holds for all $\mathbf{z} \in \mathcal{C}$. Thus, we obtain:

$$\begin{aligned} Q^{\pi^*_{\mathbf{w}'}}_{\mathbf{w}'}(s,a) - Q^{\pi}_{\mathbf{w}'}(s,a) \quad & \leq \min_{\mathbf{z}\in\mathcal{C}} \left(\frac{2\|\mathbf{w}' - \mathbf{z}\| \cdot \|\boldsymbol{\varphi}\|_\infty}{1-\gamma} \right) + \frac{2}{1-\gamma}\varepsilon_{\mathcal{C}} \\ & \leq \frac{2\|\boldsymbol{\varphi}\|_\infty}{1-\gamma} \min_{\mathbf{z}\in\mathcal{C}} \left(\|\mathbf{w}' - \mathbf{z}\| \right) + \\ & + \frac{2}{1-\gamma} \max_{\mathbf{z}\in\mathcal{C}} \left(\|\boldsymbol{\psi}^{\pi_{\mathbf{z}}}(s,a)^T \mathbf{w}' - \tilde{\boldsymbol{\psi}}(s,a,\mathbf{z})^T \mathbf{w}'\| \right) \end{aligned} \tag{A.37}$$

□

B

Appendix: Additional Experimental Details

B.1 Multitask FQI: Additional Results

In this section, we include some of the experimental results studied in Chapter 3. These correspond to similar experiments done for the second MDP introduced in Section 3.3. These were omitted in the exposition of that chapter due to the fact that the results are very similar to the ones included in the main text. As a illustration, in Figure B.1.1, we display the common features learnt via the join optimisation problem. Similar to the MDP analysed in the main text, one can see the trained representation is very compact and informative for modelling the optimal value function. The same structure appear: the most prominent feature encodes the structure if the environment, while the next 2+ features encode navigation pattern within the 'rooms', or area separated by doors (or bottleneck states). For instance, the second most informative feature encodes navigation from the right-hand side of the environment to the left-most part of the environment and vice-versa (due to change in sign). We also include the weights associated with these features in the trained value functions considered. These can be found in Figure B.1.2. If we now concentrate our attention to the second row of this illustration, we can simply read off the location of the goals associated with each of the training task considered: dark red values, indicating positive values for weights α_2, correspond to goals in the left-most room; yellow values, indicating negative values for α_2, correspond to goals in right most room and for particular strong activations this would indicate locations in the bottom-right area.

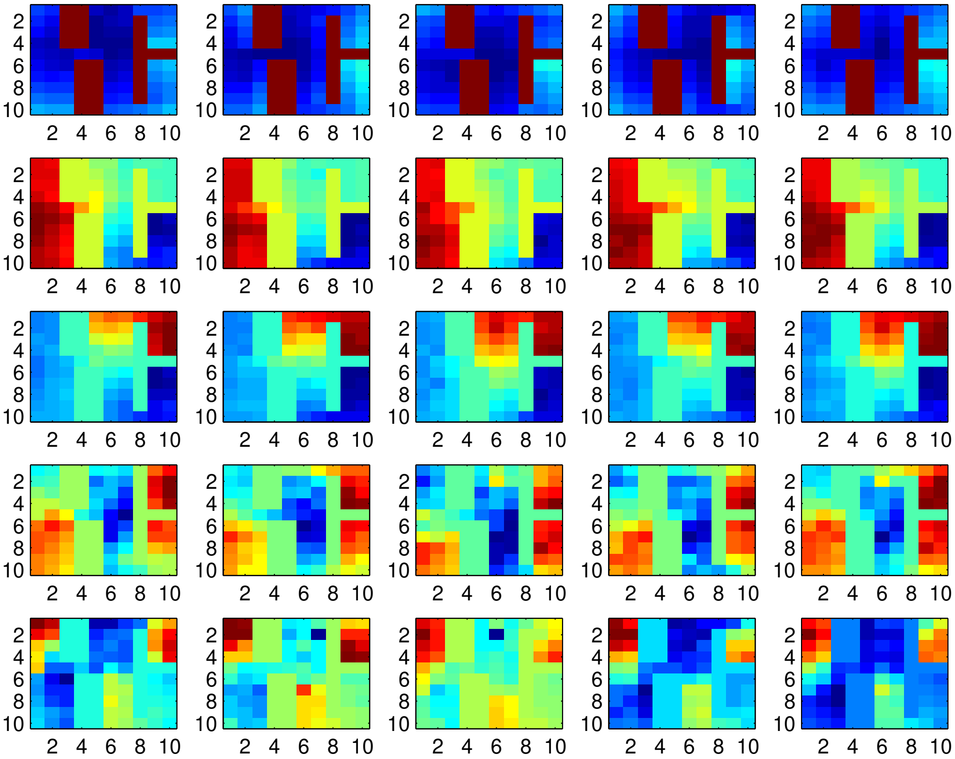


Figure B.1.1: Learnt shared features via joint multi-task learning. Feature maps are presented row-wise. Each of the first four columns corresponds to the value of the feature for each state in the grid, with respect to the four actions available. Thus row i depicts: $\varphi_i(s, \rightarrow), \varphi_i(s, \uparrow), \varphi_i(s, \leftarrow), \varphi_i(s, \downarrow)$ and the fifth column is just a max over these features $\max_a(\varphi(s, a))$. This helps visualise the potential local policies encoded in these features.

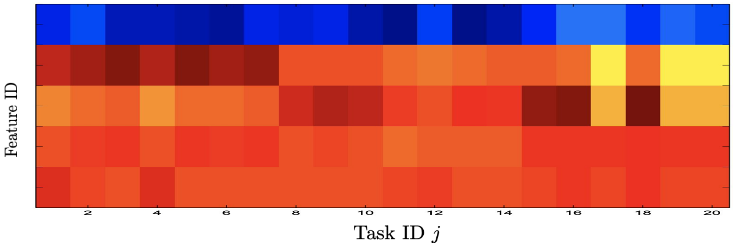


Figure B.1.2: Weight vectors a_t, highlighting the importance of each features for a given task j. To be read: column-wise = task-wise a_j. Strong activation: blue(negative activation), strong red(positive activation), yellow values (negative activation).

B.2 Successor Features at Scale and Learning on a New Task: Experimental Details and Additional Results

In this section we give some further details about the experiments presented in Section 5.4 in the main text, as well as additional results on all tasks studied whenever only a small selection could be presented in the main chapter.

B.2.1 Agents's Architecture

The CNN used in Figure 5.4.2 is identical to that used by (Mnih et al., 2015) DQN. The CNN outputs a 256-dimensional vector that serves as the LSTM state. As shown in Figure 5.4.2, the LSTM also receives the previous action of the agent as an input. The output of the LSTM is a vector of dimension 256, which in the main text we call the state signal $\tilde{s}$. The vector $\tilde{s}$ is the input of the $D + 1$ MLPs used to compute $\tilde{\boldsymbol{\varphi}}$ and $\tilde{\boldsymbol{\psi}}^{\pi_i}$. These MLPs have 100 tanh hidden units and an output of dimension $D \times |\mathcal{A}|$—that is, for each action $a \in \mathcal{A}$ the MLP outputs a D-dimensional vector representing either $\tilde{\boldsymbol{\varphi}}$ or one of the $\tilde{\boldsymbol{\psi}}^{\pi_i}$. These D-dimensional vectors are then multiplied by $\tilde{\mathbf{w}}$, leading to a $(D + 1) \times |\mathcal{A}|$ output representing $\tilde{r}$ and $\tilde{Q}^{\pi_i}$.

B.2.2 Agents's Training

The losses shown in lines 10 and 15 of Algorithm 5.1 and in lines 7 and 10 of Algorithm 5.2 were minimised using the RMSProp method, a variation of the well-known back-propagation algorithm. As parameters of RMSProp we adopted a fixed decay rate of 0.99 and $\varepsilon = 0.01$. For all algorithms we tried at least two values for the learning rate: 0.01 and 0.001. For the baselines "$DQ(\lambda)$ fine tuning" and "$DQ(\lambda)$ from scratch" we also tried a learning rate of 0.005. The results shown in the main text are those associated with the best final performance of each algorithm.

As mentioned in the main text, the agents's training was carried out using the IMPALA architecture (Espeholt et al., 2018). In IMPALA the agent is conceptually divided in two groups: "actors", which interact with the environment in parallel collecting trajectories and adding them to a queue, and a "learner", which pulls trajectories from the queue and uses them to apply the updates. On the learner side, we adopted a simplified version of IMPALA that uses $Q(\lambda)$ as the RL algorithm (*i.e.*, no parametric representation of policies nor off-policy corrections). For the distributed collection of data we used 50 actors per task. Each actor gathered trajectories of length 20 that were then added to the common queue. The collection of data followed an ε-greedy policy with a decaying ε. Specifically, the value of ε started at 0.5 and decayed linearly to 0.05 in 10^6 steps. The results shown in the main text correspond to the performance of the the ε-greedy policy (that is, they include exploratory actions of the agents).

For the results with SF & GPI-continual, in addition to the loss induced by equation (5.4), minimised in line 15 of Algorithm 5.1, we also used a standard $Q(\lambda)$ loss—that is, the gradients associated with both losses were combined through a weighted sum and then used to update $\boldsymbol{\theta}_{\psi}$. The weights for the standard $Q(\lambda)$ loss and the loss computed in line 14 of Algorithm 5.1 were 1 and 0.1, respectively. Using the standard $Q(\lambda)$ loss seems to stabilise the learning of $\tilde{\boldsymbol{\psi}}^{\pi_{n+1}}$; in this case (5.4) can be seen as a constraint for the standard RL optimisation. Obviously, if we want to add $\tilde{\boldsymbol{\psi}}^{\pi_{n+1}}$ to $\tilde{\Psi}$, we have to make sure that the SF semantics are preserved—that is, the combined updates are indeed trying to satisfy (5.4). We confirmed this fact by monitoring the loss computed in line 14 of Algorithm 5.1.figure B.2.1 shows the average of this loss computed over 10 runs of SF & GPI-continual on all test tasks; as shown in the figure, the loss is indeed minimised, which implies that the resulting $\tilde{\boldsymbol{\psi}}^{\pi_{n+1}}$ are valid SFs that can be safely added to $\tilde{\Psi}$.

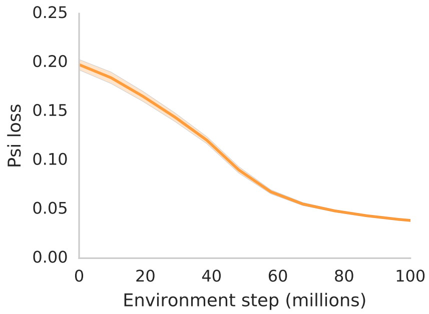


Figure B.2.1: Loss in the approximation of $\tilde{\boldsymbol{\psi}}^{\pi_{n+1}}$ (line 14 of Algorithm 5.1). Shaded region represents one standard deviation over 10 runs on all test tasks.

B.2.3 Environment

The room used in the environment and illustrated in Figure 5.4.1a was of size 13 × 13 (Beattie et al., 2016). The observations o_t were an 84×84 image with pixels re-scaled to the interval $[0, 1]$. The action space $\mathcal{A}$ contains 8 actions: move forward, move backwards, strafe left, strafe right, look left, look right, look left and move forward, and look right and move forward. Each action was repeated for 4 frames, that is, the agent was allowed to choose an action at every 4 observations.

B.2.4 Additional Results

In our experiments we defined a set of 9 test tasks in order to cover reasonably well three qualitatively distinct combinations of rewards: only positive rewards, only negative rewards, and mixed rewards. Figure B.2.2 shows the results of SF&GPI-transfer and the baselines on the test tasks that could not be included in the main text due to the space limit.

As discussed in the main text, ideally our agent should rely on the GPI policy when useful but also be able to learn and use a specialised policy otherwise. Figures B.2.3, B.2.4 and B.2.5 show that this is possible with SF&GPI-continual. Looking at Figure B.2.3 we see that when the test task only has positive rewards the performances of SF&GPI-transfer and SF&GPI-continual are virtually the same. This makes sense, since in this case alternating between the policies π_i learned on $\hat{\mathcal{M}}$ should lead to good performance. Although initially the specialised policy π_{test} does get selected by GPI a few times, eventually the policies π_i largely dominate. The figure also corroborates the hypothesis that GPI is in general not computing a trivial policy, since even after settling on the policies π_i it keeps alternating between them.

Interestingly, when we look at the test tasks with negative rewards this pattern is no longer observed. As shown in Figures B.2.4 and B.2.5, in this case SF&GPI-continual eventually outperforms SF&GPI-transfer—which is not surprising. Looking at the frequency at which policies are selected by GPI, we observe the opposite trend as before: now the policy π_{test} steadily becomes the preferred one. The fact that a specialised policy is learned and eventually dominates is reassuring, as it indicates that π_{test} will contribute to the repository of skills available to the agent when added to $\tilde{\Psi}$.

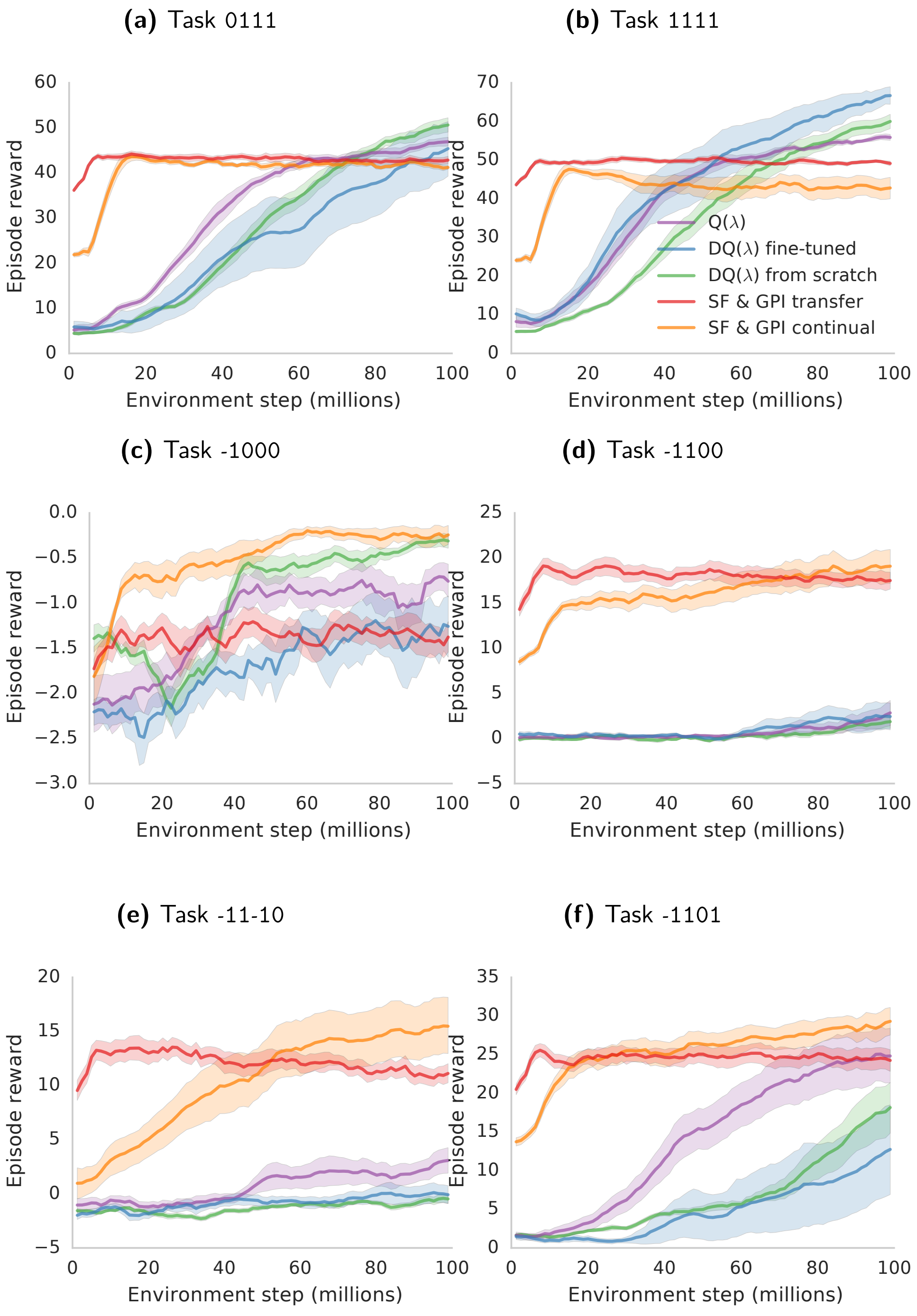


Figure B.2.2: Average reward per episode on test tasks not shown in the main text. The x axes have different scales because the amount of reward available changes across tasks. Shaded regions are one standard deviation over 10 runs.

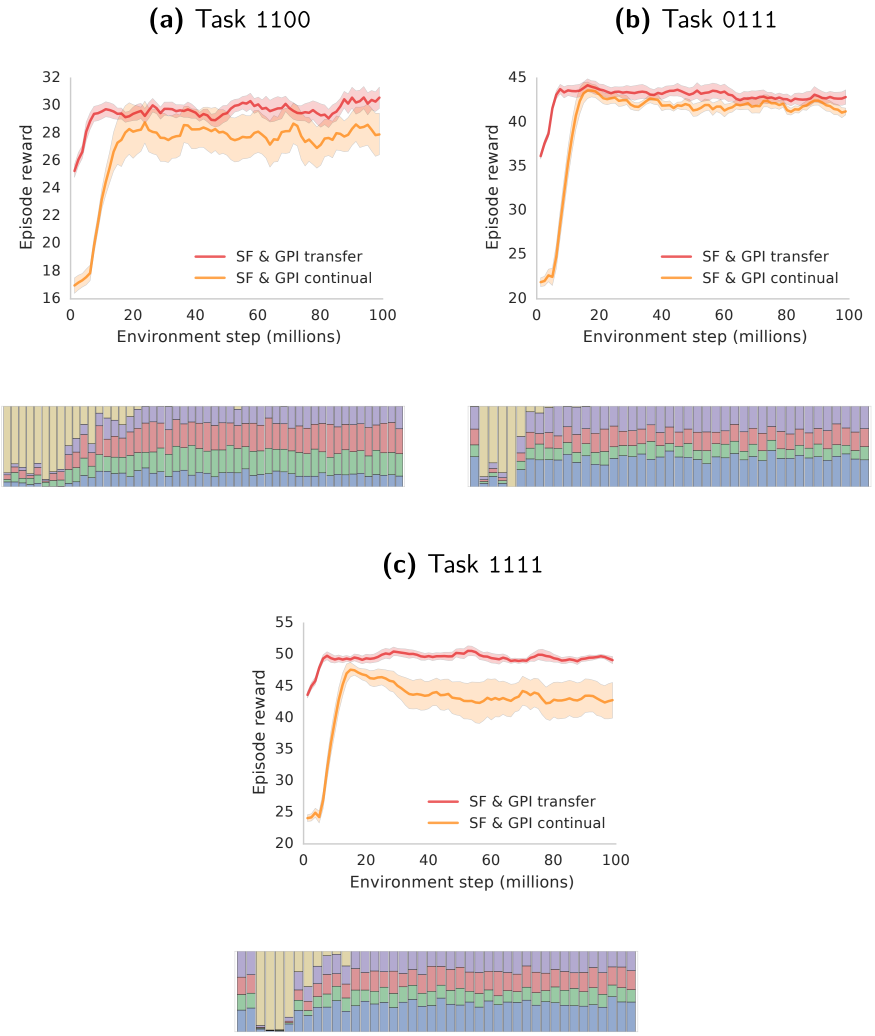
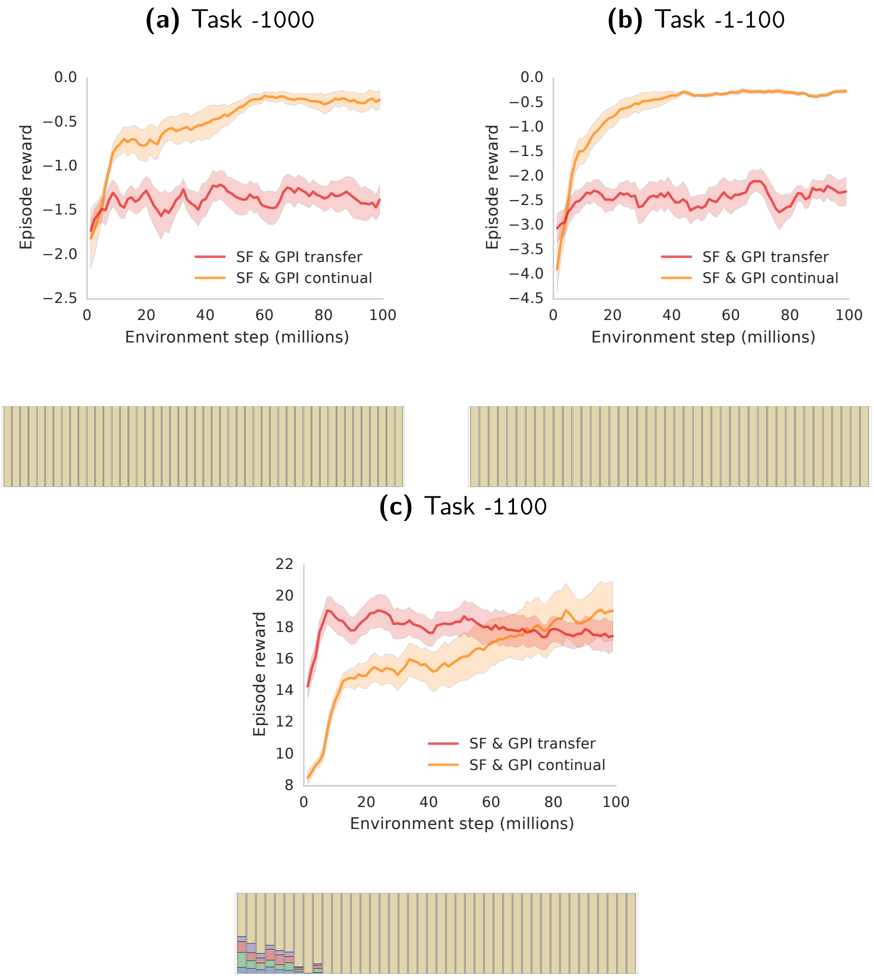


Figure B.2.3: Top figures: Comparison between SF&GPI-transfer and SF&GPI-continual. Shaded regions are one standard deviation over 10 runs. All the runs of SF&GPI-transfer and SF&GPI-continual used the same basis $\tilde{\Psi}$. **Bottom figures**: Coloured bar segments represent the frequency at which the policies π_i were selected by GPI in one run of SF&GPI-continual, with each colour associated with a specific policy. The policy π_{test} specialised to the task is represented in light yellow.

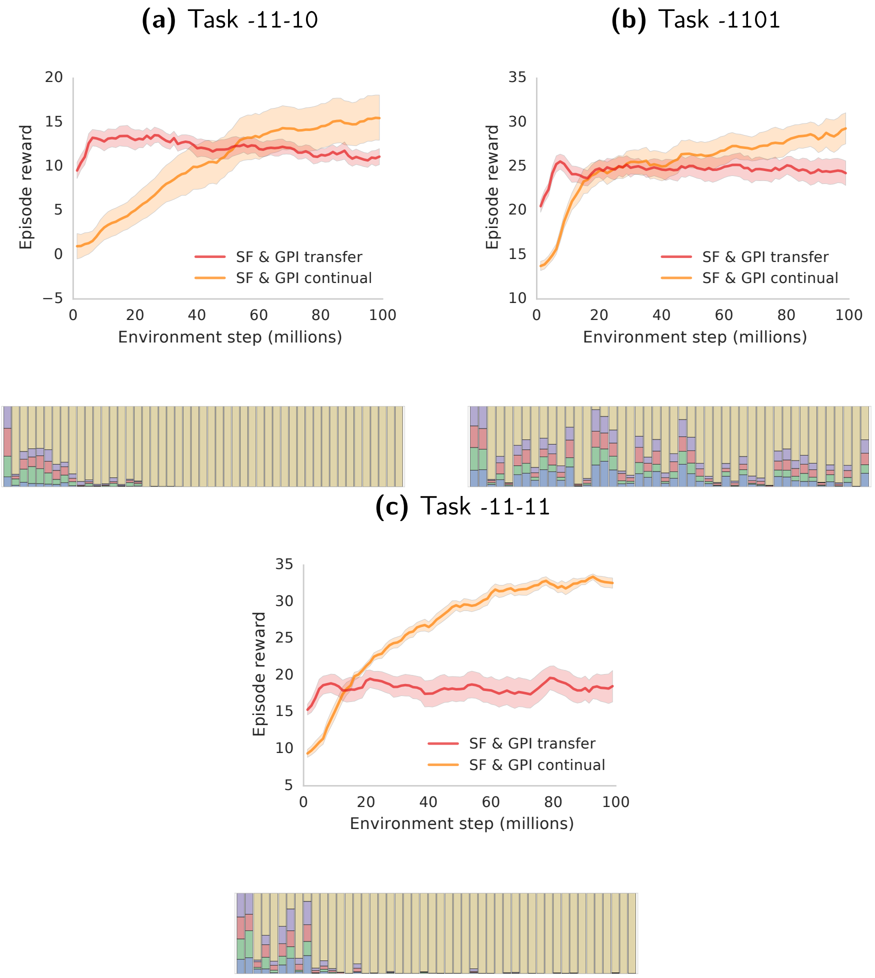


(a) Task -1000

(b) Task -1-100

(c) Task -1100

Figure B.2.4: Top figures: Comparison between SF&GPI-transfer and SF&GPI-continual. Shaded regions are one standard deviation over 10 runs. All the runs of SF&GPI-transfer and SF&GPI-continual used the same basis $\tilde{\Psi}$. **Bottom figures**: Coloured bar segments represent the frequency at which the policies π_i were selected by GPI in one run of SF&GPI-continual, with each colour associated with a specific policy. The policy π_{test} specialised to the task is represented in light yellow.



(a) Task -11-10

(b) Task -1101

(c) Task -11-11

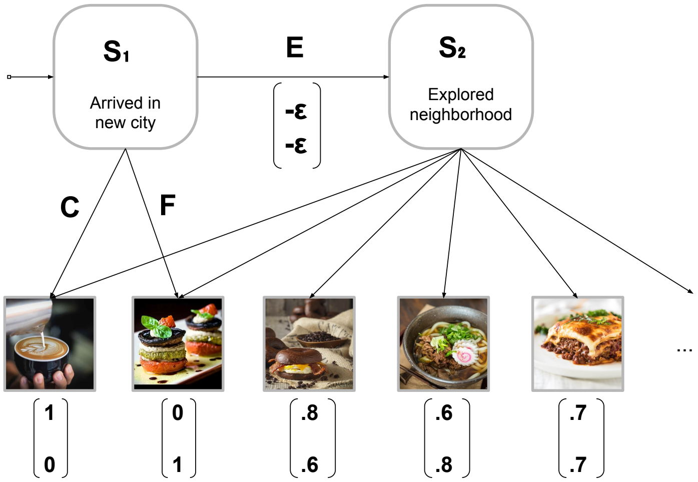
Figure B.2.5: Top figures: Comparison between SF&GPI-transfer and SF&GPI-continual. Shaded regions are one standard deviation over 10 runs. All the runs of SF&GPI-transfer and SF&GPI-continual used the same basis $\tilde{\Psi}$. **Bottom figures**: Coloured bar segments represent the frequency at which the policies π_i were selected by GPI in one run of SF&GPI-continual, with each colour associated with a specific policy. The policy π_{test} specialised to the task is represented in light yellow.

B.3 USFAs: Experimental Details and Additional Results

In this section, we give some further details regarding the experiments presented in Section 6.4 in the main text, as well as additional results and studies conducted that were omitted in the main body of the above experimental section.

B.3.1 Illustrative Example: Trip MDP

In this subsection we provide some additional analysis and results omitted from the main text. As a reminder, this is a two state MDP, where the first state is a root state, the transition from $s_1 \rightarrow s_2$ comes at a cost $r_{\mathbf{w}}(s_1, E) = \varphi(s_1, E)^T \mathbf{w} = -\epsilon(w_1 + w_2)$ and all other actions lead to a final positive reward corresponding to how much the resulting state/restaurant alligns with our preferences (our task) right now. For convenience, we provide below the depiction of the Trip MDP introduced in Section 4.1.



In the experiments run we considered a fixed set of training tasks $\mathcal{M} = \{01, 10\}$ for all methods. The set of outcomes from the exploratory state s_2 is defined as $\varphi(s_2, a) = [\cos(\theta), \sin(\theta)]$ for $\theta \in \{k\pi/2N\}_{k=0,N}$. Note that this includes the binary states for $k = 0$ and respectively $k = N$. We ran this MDP with $N = 6$, and $\epsilon = 0.05$. Thus outside the binary outcomes, the agent can select $N - 1 = 5$ other mixed outcomes and, as argued in the main text, under these conditions there will be a selection of the $\mathbf{w}$-space in which each of these outcomes will be optimal. Thus the space of optimal policies, we hope to recover,

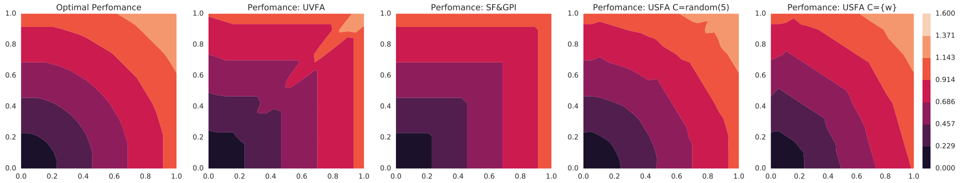


Figure B.3.1: [Sample run] Performance of the different methods (in this order, starting with the second subplot): UVFA, SF&GPIon the perfect SFs induced by $\mathcal{M}$, USFA with $\mathcal{C} = random(5)$ and USFA with $\mathcal{C} = \{\mathbf{w}'\}$ as compared to the optimal performance one could get in this MDP (first plot). These correspond to one sample run, where we trained the UVFA and USFA for 1000 episodes. The optimal performance and the SF&GPIwere computed exactly.

is generally $N + 1$. Nevertheless, there is a lot of structure in this space, that the functional approximators can uncover and employ in their generalisation.

B.3.1.1 Additional Results

In the main text, we reported the zero-shot aggregated performance over all direction $\mathcal{M}' = \{\mathbf{w}'|\mathbf{w}' = [\cos(\frac{\pi k}{2K}), \sin(\frac{\pi k}{2K})], k \in \mathbb{Z}_K\}$. This should cover most of the space of tasks/trade-offs we would be interest in. In this section we include the generalisation for other sets $\mathcal{M}'$. First in Figure B.3.1 we depict the performance of the algorithms considered across the whole $\mathbf{w}'$ space $\mathcal{M}' = [0, 1]^2$. Figure B.3.2 is just a different visualization of the previous plot, where we focus on how far these algorithms are from recovering the optimal performance. This also shows the subtle effect mentioned in the discussion in the main text, induced by the choice of $\mathcal{C}$ in the USFA evaluation.

B.3.2 Large scale Experiments: Details

B.3.2.1 Agent's Architecture

This section contains a detailed description of the USFA agent used in our experimental section. As a reminder, we include the agent's architecture below (Figure 6.4.6 in the main text). As highlighted in Section 6.4.2, our agent comprises of three main modules:

- **Input processing module**: computes a state representation $f(h_t)$ from observation

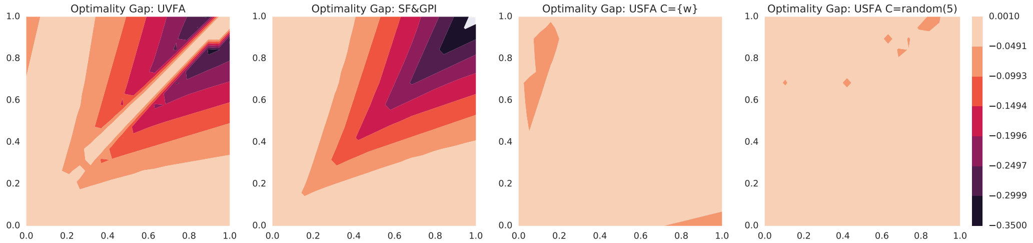
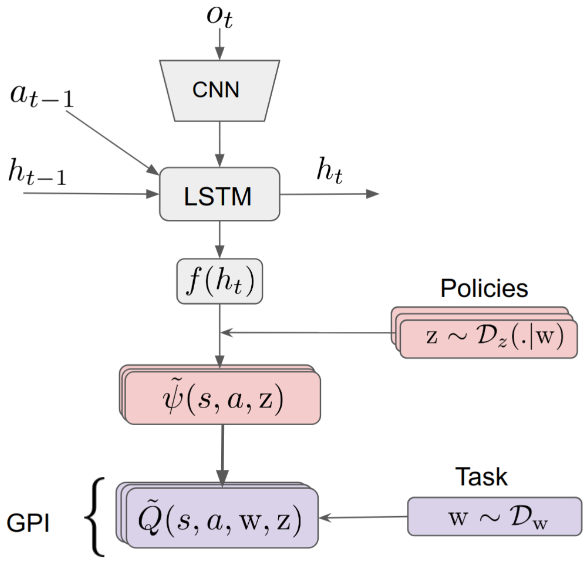


Figure B.3.2: [Sample run] Optimality gap over the whole task space. These correspond to the same sample run as above, where we trained the UVFA and USFA for 1000 episodes. We can now see more clearly that USFAs manage to recover better policies and optimality across a much greater portion of the task space. The last two plots correspond to the same USFA just using different choices of the candidate set $\mathcal{C}$. Something to note here is that by having a more diverse choice in $\mathcal{C}$, we can recover an optimal policy even in areas of the space where our approximation has not yet optimally generalised (like the upper-left corner in the **w**-space in the figures above).

o_t. This module is made up of three convolutional layers (structure identical to the one used in (Mnih et al., 2015)), the output of which then serves as input to a LSTM (256). This LSTM takes as input the previously executed action a_{t-1}. The output of the LSTM is passed through a non-linearity f (chosen here to be a ReLu) to produce a vector of 128 units, $f(h_t)$.

- **Policy conditioning module:** compute the SFs $\tilde{\boldsymbol{\psi}}(s, a, \mathbf{z})$, given a (sampled) policy embedding $\mathbf{z}$ and the state representation $f(h_t)$. This module first produces $n_{\mathbf{z}}$ number of $\mathbf{z} \sim \mathcal{D}_{\mathbf{z}}$ samples ($n_{\mathbf{z}} = 30$ in our experiments). Each of these is then transformed via a 2-layer MLP(32,32) to produce a vector of size 32, for each sample $\mathbf{z}$. This vector $g(\mathbf{z})$ gets concatenated with the state representation $f(h_t)$ and the resulting vector is further processed by a 2-layer MLP that produces a tensor of dimensions $d \times |\mathcal{A}|$ for each $\mathbf{z}$, where $d = dim(\varphi)$. These correspond to SFs $\tilde{\boldsymbol{\psi}}(s, a, \mathbf{z})$ for policy $\pi_{\mathbf{z}}$. Note that this computation can be done quite efficiently by reusing the state embedding $f(h_t)$, doing the downstream computation in parallel for each policy embedding $\mathbf{z}$.

- **Task evaluation module:** computes the value function $Q(s, a, \mathbf{z}, \mathbf{w}) = \tilde{\boldsymbol{\psi}}(s, a, \mathbf{z})^T \mathbf{w}$ for a given task description $\mathbf{w}$. This module does not have any parameters as the value functions are simply composable from $\tilde{\boldsymbol{\psi}}(s, a, \mathbf{z})$ and the task description $\mathbf{w}$


o_t
CNN
a_{t-1}
h_{t-1}
LSTM
h_t
$f(h_t)$
Policies
$\mathrm{z} \sim \mathcal{D}_z(.|\mathrm{w})$
$\tilde{\psi}(s, a, \mathrm{z})$
Task
GPI
$\tilde{Q}(s, a, \mathrm{w}, \mathrm{z})$
$\mathrm{w} \sim \mathcal{D}_\mathrm{w}$

$$\pi(s) = \arg\max_a \max_z \tilde{Q}(s, a, \mathrm{w}, \mathrm{z})$$

Figure B.3.3: USFA architecture

via assumption (1). This module with output $n_{\mathbf{z}}$ value functions that will be used to produce a behavior via GPI.

An important decision in this design was how and where to introduce the conditioning on the policy. In all experiments shown here the conditioning was done simply by concatenating the two embeddings **w** and **z**, although stronger conditioning via an inner product was tried yielding similar performance. The 'where' on the other hand is much more important. As the conditioning on the policy happens quite late in the network, most of the processing (up to $f(h_t)$) can be done only once, and we can sample multiple **z** and compute the corresponding $\tilde{\boldsymbol{\psi}}^{\pi_{\mathbf{z}}}$ at a fairly low computational cost. As mentioned above, these will be combined with the task vector **w** to produce the candidate action value functions for GPI. Note that this helps both in training and in acting, as otherwise the unroll of the LSTM would be policy conditioned, making the computation of the SFs and the off-policy n-step learning quite expensive.

UVFA baseline agents have a similar architecture, but now the task description **w** is fed in as an input to the network. The conditioning on the task of UVFAs is done in a similar fashion as we did the conditioning on the policies in USFAs, to make the computational power and capacity comparable. The input processing module is the same and now downstream, instead of conditioning on the policy embedding $\mathbf{z}$, we condition on task description $\mathbf{w}$. This conditioning if followed by a 2-layer MLP that computes the value functions $\tilde{Q}^*(s, a, \mathbf{w})$, which induces the greedy policy $\pi_{\mathbf{w}}^{(UVFA)} = \arg\max_a \tilde{Q}^*(s, a, \mathbf{w})$.

B.3.2.2 Agent's Training

The agents' training was carried out using the IMPALA architecture (Espeholt et al., 2018). On the learner side, we adopted a simplified version of IMPALA that uses $Q(\lambda)$ as the RL algorithm. In our experiments, for all agents we used $\lambda = 0.9$. Depending on the sampling distribution $D_{\mathbf{z}}$, in learning we will often be off-policy. That is, most of the time, we are going to learn about a policy $\pi_{\mathbf{z}_1}$ and update its corresponding SFs approximations $\tilde{\boldsymbol{\psi}}(s, a, \mathbf{z}_1)$, using data generated by acting in the environment according to some other policy $\pi_{\mathbf{z}_2}$. In order to account for this off-policiness, whenever computing the n-step return required in Eq. 6.10, we are going to cut traces whenever the policies start to disagree and bootstrap from this step on (Sutton and Barto, 1998):

$$e_t(s,a) = \begin{cases} 1 + \gamma\lambda e_{t-1}(s,a), & \text{if } s = s_t, a = a_t, \pi(s) = a \\ 0, & \text{if } \pi(s) \neq a \text{ (policies disagree)} \\ \gamma\lambda e_{t-1}(s,a), & \text{otherwise} \end{cases} \tag{B.1}$$

Here we can see how the data distribution induce by the choice of training tasks $\mathcal{M}$ can influence the training process. If the data distribution $\mathcal{D}_{\mathbf{z}}$ is very close to the set $\mathcal{M}$, as in our first experiment, most of the policies we are going to sample will be close to the policies that generated the data. This means that we might be able to make use of longer trajectories in this data, as the policies will rarely disagree. On the other hand, by staying close to the training tasks, we might hurt our ability to generalise in the policy space, as our first experiment suggest (see Figure 6.4.7). By having a broader distribution $\mathcal{D}_z = \mathcal{N}(\mathbf{w}, 0.5I)$, we can learn about more diverse policies in this space, but we will also increase our off-policiness. We can see from Figure 6.4.8, that our algorithm can successfully learn and operate in both of these regimes.

For the distributed collection of data we used 50 actors per task. Each actor gathered trajectories of length 32 that were then added to the common queue. The collection of data followed an ϵ-greedy policy with a fixed $\epsilon = 0.1$. The training curves shown in the main text correspond to the performance of the the ε-greedy policy (that is, they include exploratory actions of the agents).

B.3.2.3 Agent's Evaluation

All agents were evaluated in the same fashion. During the training process, periodically (every 20M frames) we will evaluate the agents performance on a test of held out test tasks. We take these intermediate snapshots of our agents and 'freeze' their parameters to assess zero-shot generalisation. Once a test task $\mathbf{w}'$ is provided, the agent interacts with the environment for 20 episodes, one minute each and the average (undiscounted) reward is recorded. These produce the evaluation curves in Figure 6.4.7. Evaluations are done with a small $\varepsilon = 0.001$, following a GPI policy with different instantiations of $\mathcal{C}$. For the pure UVFA agents, the evaluation is similar: ε-greedy on the produced value functions $\tilde{Q}^*(s, a, \mathbf{w})$, with the same evaluation $\varepsilon = 0.001$.

B.3.3 Additional Results

In our experiments we defined a set of easy test tasks (close to $\mathcal{M}$) and a set of harder tasks, in order to cover reasonably well a few distinct scenarios:

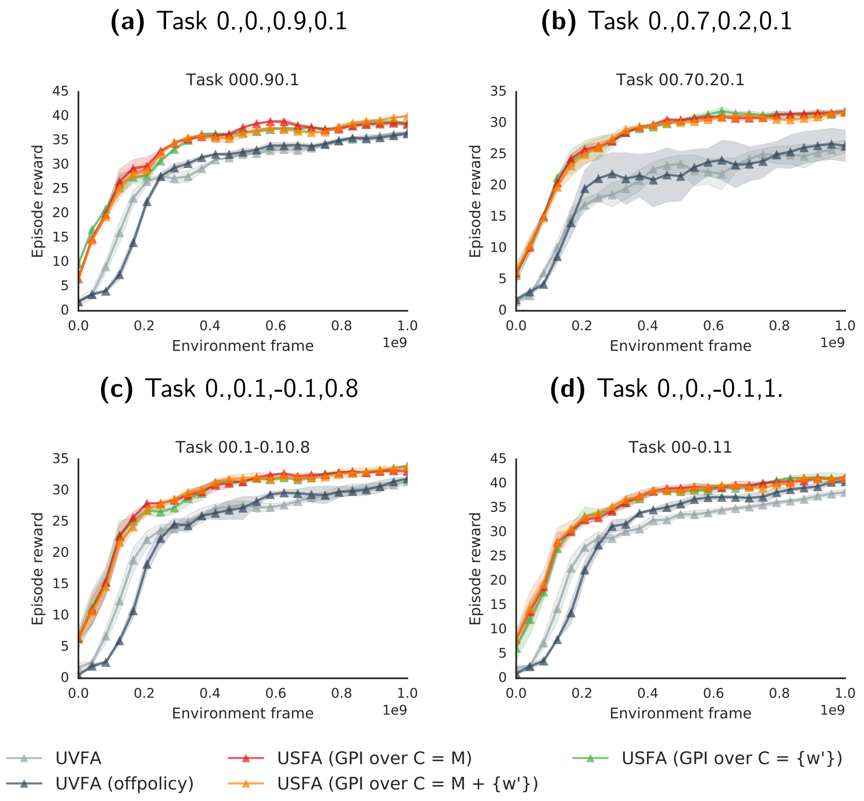
- Testing generalisation to tasks very similar to the training set, e.g. $\mathbf{w}' = [0, 0.9, 0, 0.1]$;
- Testing generalisation to harder tasks with different reward profiles: only positive rewards, only negative rewards, and mixed rewards.

In the main text, we included only a selection of these for illustrative purposes. Here we present the full results.

B.3.3.1 Canonical Basis: Zero-shot Generalisation

This section contains the complete results of the first experiment conducted. As a reminder, in this experiment we were training a USFA agent on $\mathcal{M} = \{1000, 0100, 0010, 0001\}$, with $D_{\mathbf{z}} = \mathcal{N}(\mathbf{w}, 0.1I)$ and compare its performance with two conventional UVFA agents (one trained on-policy and the other one using all the data generated to learn off-policy) on a range of unseen test tasks. Complete set of result is included below, as follows: Figure B.3.4 includes results on easy tasks, close to the tasks contained in the training set $\mathcal{M}$ (generalisation to those should be fairly straightforward); Figure B.3.5 and Figure B.3.6

present results on more challenging tasks, quite far away from the training set, testing out agents ability to generate to the whole 4D hypercube.

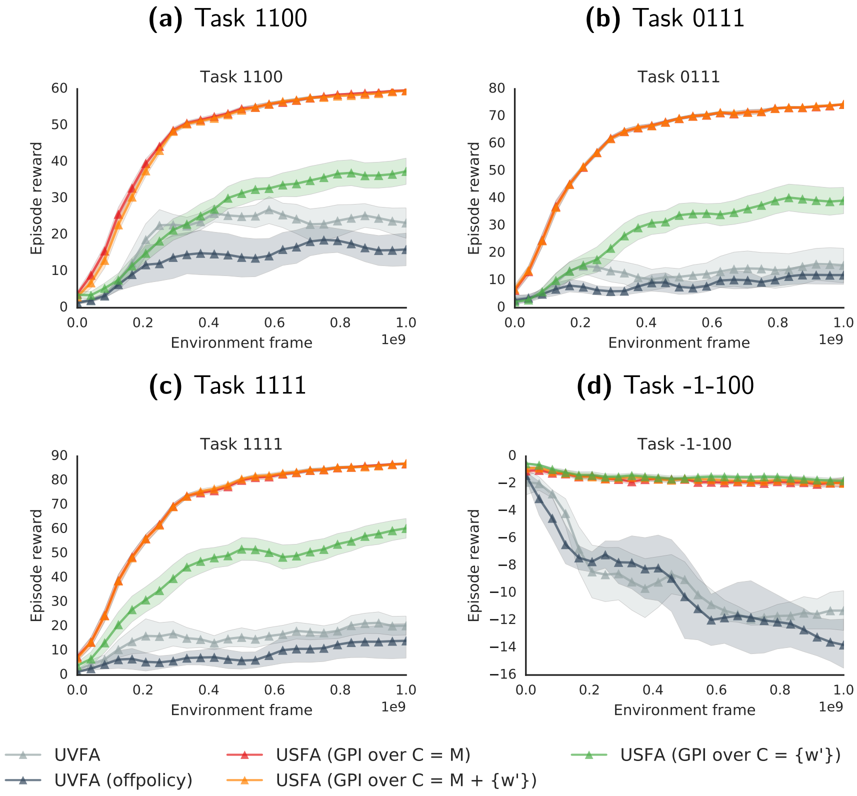

(a) Task 0.,0.,0.9,0.1

(b) Task 0.,0.7,0.2,0.1

(c) Task 0.,0.1,-0.1,0.8

(d) Task 0.,0.,-0.1,1.

Figure B.3.4: Zero-shot performance on the easy evaluation set: Average reward per episode on test tasks not shown in the main text. This is comparing a USFA agent trained on the canonical training set $\mathcal{M} = \{1000, 0100, 0010, 0001\}$, with $D_{\mathbf{z}} = \mathcal{N}(\mathbf{w}, 0.1I)$ and the two UVFA agents: one trained on-policy, one employing off-policy.



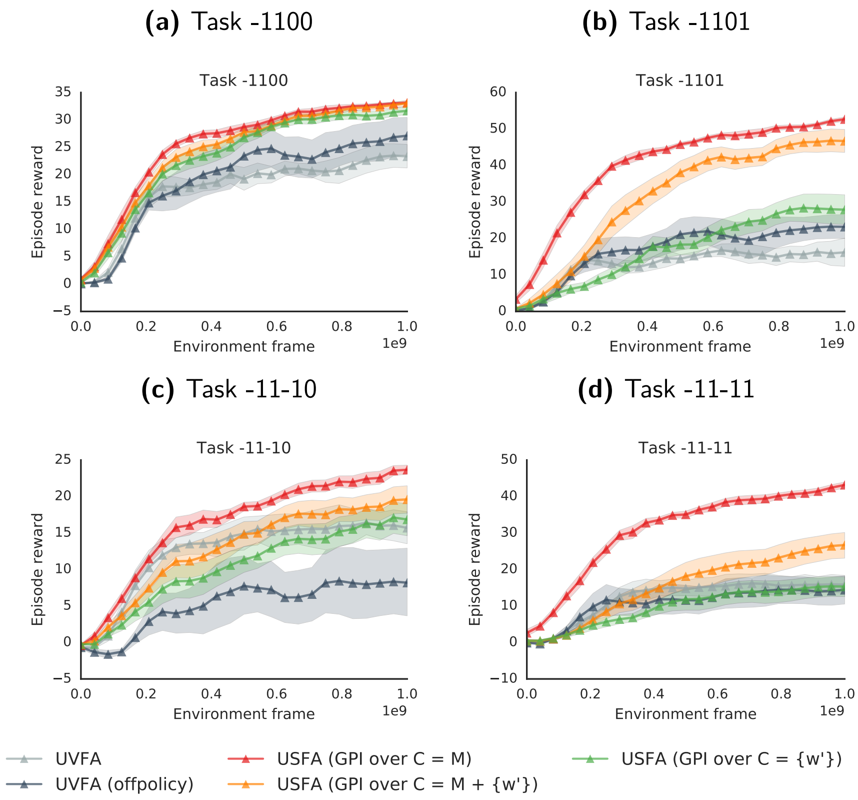
(a) Task 1100

(b) Task 0111

(c) Task 1111

(d) Task -1-100

Figure B.3.5: Zero-shot performance on harder tasks: Average reward per episode on test tasks not shown in the main text. This is comparing a USFA agent trained on the canonical training set $\mathcal{M} = \{1000, 0100, 0010, 0001\}$, with $D_{\mathbf{z}} = \mathcal{N}(\mathbf{w}, 0.1I)$ and the two UVFA agents: one trained on-policy, one employing off-policy. (Part 1)



(a) Task -1100

(b) Task -1101

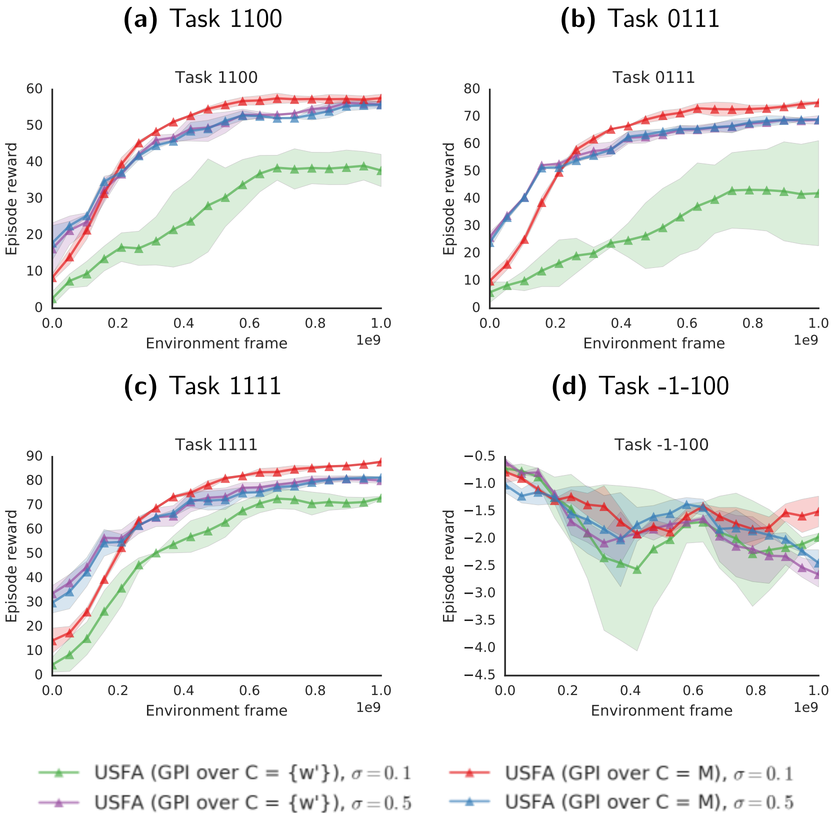
(c) Task -11-10

(d) Task -11-11

Figure B.3.6: Zero-shot performance on harder tasks: Average reward per episode on test tasks not shown in the main text. This is comparing a USFA agent trained on the canonical training set $\mathcal{M} = \{1000, 0100, 0010, 0001\}$, with $D_{\mathbf{z}} = \mathcal{N}(\mathbf{w}, 0.1I)$ and the two UVFA agents: one trained on-policy, one employing off-policy. (Part 2)

B.3.4 Canonical Basis: USFAs in Different Training Regimes.

In this section, we include the omitted results from our second experiment. As a reminder, in this experiment we were training two USFA agents on the same set of canonical tasks, but employing different distributions $D_{\mathbf{z}}$, one will low variance $\sigma = 0.1$, focusing in learning policies around the training set $\mathcal{M}$, and another one with larger variance $\sigma = 0.5$, that will try to learn about a lot more policies away from the training set, thus potentially facilitating the generalisation provided by the UVFA component. Results are displayed in Figures B.3.7-B.3.8 on all tasks in the hard evaluation set.

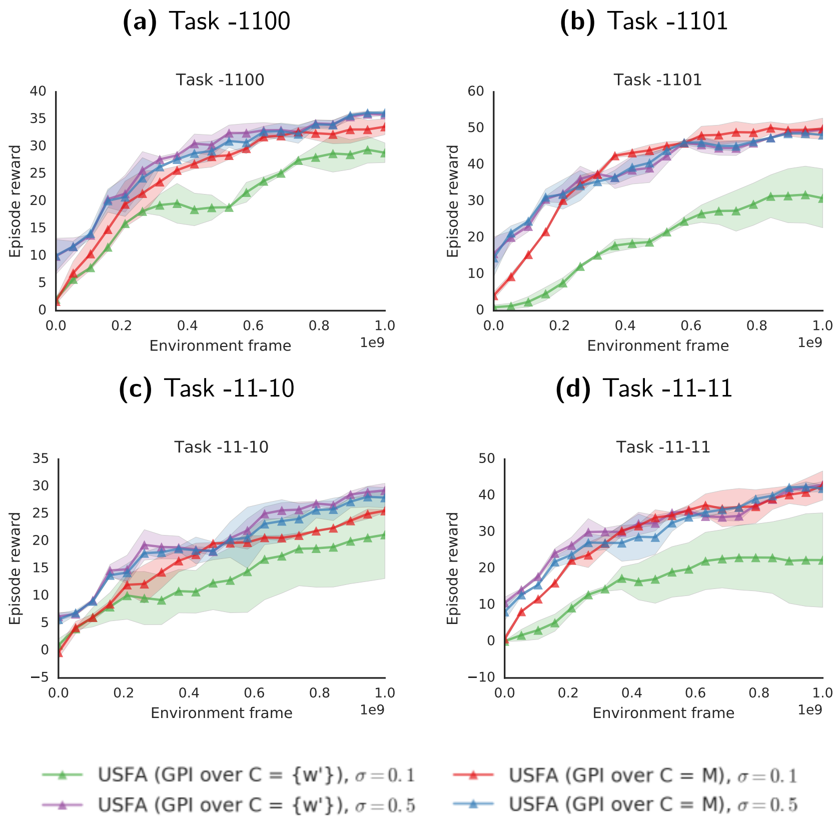

(a) Task 1100

(b) Task 0111

(c) Task 1111

(d) Task -1-100

Figure B.3.7: Different $\mathcal{D}_z$ – Zero-shot performance on harder tasks: Average reward per episode on test tasks not shown in the main text. This is comparing the generalisations of two USFA agent trained on the canonical training set $\mathcal{M} = \{1000, 0100, 0010, 0001\}$, with $D_{\mathbf{z}} = \mathcal{N}(\mathbf{w}, 0.1I)$, and $D_{\mathbf{z}} = \mathcal{N}(\mathbf{w}, 0.5I)$. (Part 1)


(a) Task -1100

(b) Task -1101

(c) Task -11-10

(d) Task -11-11

Figure B.3.8: Different $\mathcal{D}_{\mathbf{z}}$ – Zero-shot performance on harder tasks: Average reward per episode on test tasks not shown in the main text. This is comparing a USFA agent trained on the canonical training set $\mathcal{M} = \{1000, 0100, 0010, 0001\}$, with $D_{\mathbf{z}} = \mathcal{N}(\mathbf{w}, 0.1I)$, and $D_{\mathbf{z}} = \mathcal{N}(\mathbf{w}, 0.5I)$. (Part 2)

B.3.4.1 Larger Collection of Training Tasks

We also trained our USFA agent on a larger set of training tasks that include the previous canonical tasks, as well as four other tasks that contain both positive and negative reward $\mathcal{M} = \{1000, 0100, 0010, 0001, 1\text{-}100, 01\text{-}10, 001\text{-}1, \text{-}1000\}$. Thus we expect this agent to generalises better as a result of its training. A selection of these results and sample performance in training are included in Figure B.3.9.

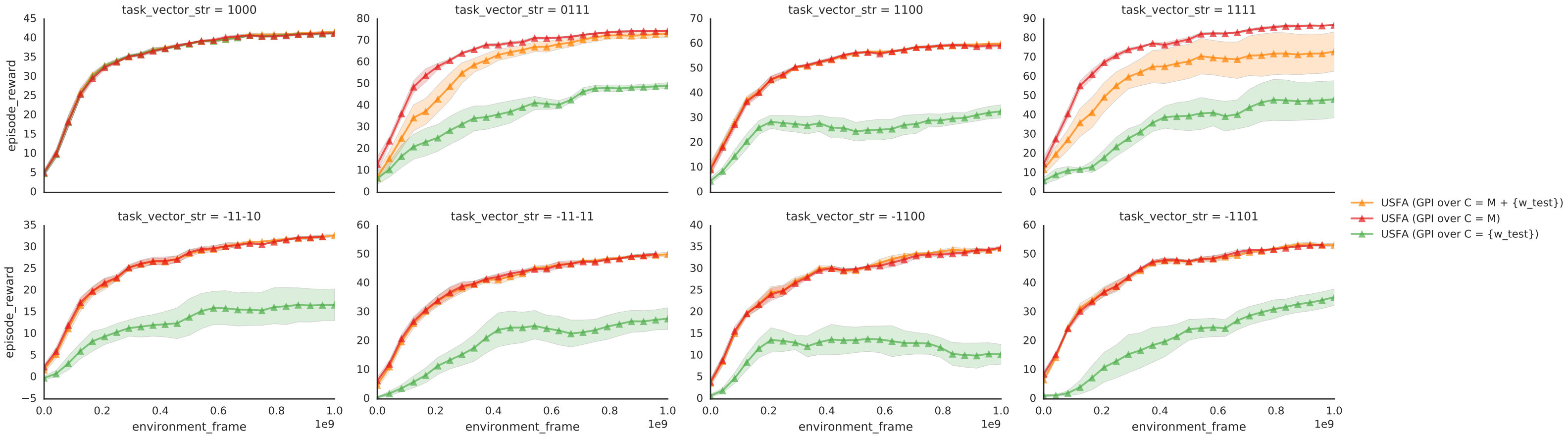


Figure B.3.9: Large $\mathcal{M}$. Learning curves for training task $[1000] \in \mathcal{M}$ and generalisation performance on a sample of test tasks $\mathbf{w}' \in \mathcal{M}'$ after training on all the tasks $\mathcal{M}$. This is a selection of the hard evaluation tasks. Results are average over 10 training runs.

References

Joshua Achiam and Shankar Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv preprint arXiv:1703.01732*, 2017.

Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew Taylor. Online multi-task learning for policy gradient methods. In *International Conference on Machine Learning*, pages 1206–1214, 2014.

Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6(Nov):1817–1853, 2005.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 166–175. JMLR. org, 2017.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

A. Antos, R. Munos, and Cs. Szepesvári. Fitted Q-iteration in continuous action-space MDPs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 9–16, 2007.

András Antos, Csaba Szepesvári, and Rémi 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.

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. In *Advances in neural information processing systems*, pages 41–48, 2007.

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, 2008.

Rachita Ashar. *Hierarchical learning in stochastic domains*. PhD thesis, Citeseer, 1994.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1726–1734, 2017.

Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

André Barreto, Will Dabney, Rémi Munos, Jonathan Hunt, Tom Schaul, Hado van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

André Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Žídek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. *Proceedings of the 35th International Conference on Machine Learning*, 2018.

André M. S. Barreto, Joelle Pineau, and Doina Precup. Policy iteration based on stochastic factorization. *Journal of Artificial Intelligence Research*, 50:763–803, 2014.

Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

Andrew G Barto and Ozgür Simsek. Intrinsic motivation for reinforcement learning systems. In *Proceedings of the Thirteenth Yale Workshop on Adaptive and Learning Systems*, pages 113–118, 2005.

Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

Jonathan Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

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, 06 2013.

Marc G Bellemare, Will Dabney, Robert Dadashi, Adrien Ali Taiga, Pablo Samuel Castro, Nicolas Le Roux, Dale Schuurmans, Tor Lattimore, and Clare Lyle. A geometric perspective on optimal representations for reinforcement learning. *arXiv preprint arXiv:1901.11530*, 2019.

Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. Citeseer, 2017.

Glen Berseth, Cheng Xie, Paul Cernek, and Michiel Van de Panne. Progressive reinforcement learning with distillation for multi-skilled motion control. *arXiv preprint arXiv:1802.04765*, 2018.

Dimitri P. Bertsekas. Approximate policy iteration: a survey and some new methods. *Journal of Control Theory and Applications*, 9(3):310–335, 2011.

Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 1, pages 560–564. IEEE, 1995.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

Diana Borsa, Thore Graepel, and John Shawe-Taylor. Learning shared representations in multi-task reinforcement learning. *arXiv preprint arXiv:1603.02041*, 2016.

Diana Borsa, Andre Barreto, John Quan, Daniel J. Mankowitz, Hado van Hasselt, Remi Munos, David Silver, and Tom Schaul. Universal successor features approximators. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1VWjiRcKX.

Matthew Botvinick and Ari Weinstein. Model-based hierarchical reinforcement learning and human action control. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 369(1655):20130480, 2014.

Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149, January 2015.

Steven J Bradtke and Andrew G Barto. Linear least-squares algorithms for temporal difference learning. *Machine learning*, 22(1-3):33–57, 1996.

Emma Brunskill and Lihong Li. Pac-inspired option discovery in lifelong reinforcement learning. In *International conference on machine learning*, pages 316–324, 2014.

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

Lucian Buşoniu, Alessandro Lazaric, Mohammad Ghavamzadeh, Rémi Munos, Robert Babuška, and Bart De Schutter. Least-squares methods for policy iteration. In *Reinforcement learning*, pages 75–109. Springer, 2012.

Daniele Calandriello, Alessandro Lazaric, and Marcello Restelli. Sparse multi-task reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 819–827, 2014.

Feng Cao and Soumya Ray. Bayesian hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 73–81, 2012.

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Rich Caruana and Joseph O'Sullivan. Multitask pattern recognition for autonomous robots. In *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No. 98CH36190)*, volume 1, pages 13–18. IEEE, 1998.

Andrea Castelletti, Francesca Pianosi, and Marcello Restelli. Tree-based fitted q-iteration for multi-objective markov decision problems. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE, 2012.

Jianhui Chen, Ji Liu, and Jieping Ye. Learning incoherent sparse and low-rank patterns from multiple tasks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5 (4):22, 2012.

Ignasi Clavera, David Held, and Pieter Abbeel. Policy transfer via modularity and reward guiding. In *International Conference on Intelligent Robots and Systems (IROS)*, 2017.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

Bruno Da Silva, George Konidaris, and Andrew Barto. Learning parameterized skills. *arXiv preprint arXiv:1206.6398*, 2012.

Robert Dadashi, Adrien Ali Taïga, Nicolas Le Roux, Dale Schuurmans, and Marc G Bellemare. The value function polytope in reinforcement learning. *arXiv preprint arXiv:1901.11524*, 2019.

Peter Dayan. The convergence of td (λ) for general λ. *Machine learning*, 8(3-4):341–362, 1992.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

Peter Dayan and Terrence J Sejnowski. Td (λ) converges with probability 1. *Machine Learning*, 14(3):295–301, 1994.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2169–2176. IEEE, 2017.

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Christos Dimitrakakis and Constantin A Rothkopf. Bayesian multitask inverse reinforcement learning. In *European workshop on reinforcement learning*, pages 273–284. Springer, 2011.

Carlos Diuk and Michael Littman. Hierarchical reinforcement learning. In *Encyclopedia of artificial intelligence*, pages 825–830. IGI Global, 2009.

Yunshu Du, Wojciech M Czarnecki, Siddhant M Jayakumar, Razvan Pascanu, and Balaji Lakshminarayanan. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224*, 2018.

Eric Eaton and Paul L Ruvolo. Ella: An efficient lifelong learning algorithm. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 507–515, 2013.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. In *Journal of Machine Learning Research*, pages 503–556, 2005.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

Amir Massoud Farahmand, Mohammad Ghavamzadeh, Csaba Szepesvári, and Shie Mannor. Regularized fitted q-iteration for planning in continuous-space markovian decision problems. In *American Control Conference, 2009. ACC'09.*, pages 725–730. IEEE, 2009.

Kimberly Ferguson and Sridhar Mahadevan. Proto-transfer learning in markov decision processes using spectral methods. *Computer Science Department Faculty Publication Series*, page 151, 2006.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. URL http://arxiv.org/abs/1703.03400.

Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *CoRR*, abs/1710.09767, 2017. URL http://arxiv.org/abs/1710.09767.

Thomas Gabel, Christian Lutz, and Martin Riedmiller. Improved neural fitted q iteration applied to a novel computer gaming and learning benchmark. In *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on*, pages 279–286. IEEE, 2011.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

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

Audrunas Gruslys, Mohammad Gheshlaghi Azar, Marc G Bellemare, and Remi Munos. The reactor: A sample-efficient actor-critic architecture. *arXiv preprint arXiv:1704.04651*, 5, 2017.

Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068*, 2014.

Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. *CoRR*, abs/1707.02286, 2017. URL http://arxiv.org/abs/1707.02286.

Bernhard Hengst. Discovering hierarchy in reinforcement learning with hexq. In *ICML*, volume 2, pages 243–250, 2002.

Bernhard Hengst. Hierarchical reinforcement learning. *Encyclopedia of Machine Learning and Data Mining*, pages 611–619, 2017.

Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojtek Czarnecki, Max Jaderberg, Denis Teplyashin, et al. Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*, 2017.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3796–3803, 2019.

Geoffrey E Hinton, Terrence Joseph Sejnowski, and Tomaso A Poggio. *Unsupervised learning: foundations of neural computation*. MIT press, 1999.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

David Janz, Jiri Hron, José Miguel Hernández-Lobato, Katja Hofmann, and Sebastian Tschiatschek. Successor uncertainties: exploration and uncertainty in temporal difference learning. *Advances in Neural Information Processing Systems*, 2019.

Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pages 1094–1099. Citeseer, 1993.

Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. *International Conference on Learning Representations*, 2019.

H Jin Kim, Michael I Jordan, Shankar Sastry, and Andrew Y Ng. Autonomous helicopter flight via reinforcement learning. In *Advances in neural information processing systems*, pages 799–806, 2004.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Kenneth R Koedinger, Emma Brunskill, Ryan SJd Baker, Elizabeth A McLaughlin, and John Stamper. New potentials for data-driven intelligent tutoring system development and optimization. *AI Magazine*, 34(3):27–41, 2013.

Iasonas Kokkinos. Ubernet: Training auniversal'convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. *arXiv preprint arXiv:1609.02132*, 2016.

George Konidaris and Andrew G Barto. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, volume 7, pages 895–900, 2007.

George Konidaris and Andrew G Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in neural information processing systems*, pages 1015–1023, 2009.

George Konidaris, Sarah Osentoski, and Philip Thomas. Value function approximation in reinforcement learning using the fourier basis. In *Twenty-fifth AAAI conference on artificial intelligence*, 2011.

Tejas D Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J Gershman. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.

Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

Michail G Lagoudakis, Ronald Parr, et al. Model-free least-squares policy iteration. In *NIPS*, volume 14, page 345. Citeseer, 2001.

Romain Laroche and Merwan Barlier. Transfer reinforcement learning with shared dynamics. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Alessandro Lazaric. *Transfer in Reinforcement Learning: A Framework and a Survey*, pages 143–173. 2012.

Alessandro Lazaric and Mohammad Ghavamzadeh. Bayesian multi-task reinforcement learning. 2010.

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256. IEEE, 2010.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436, 2015.

Lucas Lehnert and Michael L Littman. Successor features support model-based and model-free reinforcement learning. *arXiv preprint arXiv:1901.11437*, 2019.

Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

Yao Liu, Omer Gottesman, Aniruddh Raghu, Matthieu Komorowski, Aldo A Faisal, Finale Doshi-Velez, and Emma Brunskill. Representation balancing mdps for off-policy policy evaluation. In *Advances in Neural Information Processing Systems*, pages 2644–2653, 2018.

Chen Ma, Junfeng Wen, and Yoshua Bengio. Universal successor representations for transfer reinforcement learning. *arXiv preprint arXiv:1804.03758*, 2018.

Marios C Machado, Marc G Bellemare, and Michael Bowling. A laplacian framework for option discovery in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2295–2304. JMLR. org, 2017a.

Marlos C Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell. Eigenoption discovery through the deep successor representation. *arXiv preprint arXiv:1710.11089*, 2017b.

Michael G Madden and Tom Howley. Transfer of experience between reinforcement learning environments with progressive difficulty. *Artificial Intelligence Review*, 21(3-4):375–398, 2004.

Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research*, 8:2169–2231, 2007.

Daniel J Mankowitz, Augustin Žídek, André Barreto, Dan Horgan, Matteo Hessel, John Quan, Junhyuk Oh, Hado van Hasselt, David Silver, and Tom Schaul. Unicorn: Continual learning with a universal, off-policy agent. *arXiv preprint arXiv:1802.08294*, 2018.

Andreas Maurer, Massimiliano Pontil, and Bernardino Romera-Paredes. The benefit of multitask representation learning. *The Journal of Machine Learning Research*, 17(1): 2853–2884, 2016.

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

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Rémi Munos. Error bounds for approximate policy iteration. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 560–567, 2003.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. *arXiv preprint arXiv:1706.05064*, 2017.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

Gabriella Panuccio, Arthur Guez, Robert Vincent, Massimo Avoli, and Joelle Pineau. Adaptive control of epileptiform excitability in an in vitro model of limbic seizures. *Experimental neurology*, 241:179–183, 2013.

Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

Jing Peng and Ronald J Williams. Incremental multi-step q-learning. In *Machine Learning Proceedings 1994*, pages 226–232. Elsevier, 1994.

Theodore J Perkins, Doina Precup, et al. Using options for knowledge transfer in reinforcement learning. *University of Massachusetts, Amherst, MA, USA, Tech. Rep*, 1999.

Doina Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

Martin L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

Rahul Ramesh, Manan Tomar, and Balaraman Ravindran. Successor options: An option discovery framework for reinforcement learning. *arXiv preprint arXiv:1905.05731*, 2019.

Balaraman Ravindran and Andrew G Barto. Relativized options: Choosing the right transformation. In *ICML*, pages 608–615, 2003.

Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *Machine Learning: ECML 2005*, pages 317–328. Springer, 2005.

Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing-solving sparse reward tasks from scratch. *Proceedings of the 35th International Conference on Machine Learning*, 2018.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

Evan M Russek, Ida Momennejad, Matthew M Botvinick, Samuel J Gershman, and Nathaniel D Daw. Predictive representations can link model-based reinforcement learning to model-free mechanisms. *PLoS computational biology*, 13(9):e1005768, 2017.

Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016. URL http://arxiv.org/abs/1606.04671.

Paul Ruvolo and Eric Eaton. Ella: An efficient lifelong learning algorithm. In *International Conference on Machine Learning*, pages 507–515, 2013.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal Value Function Approximators. In *International Conference on Machine Learning (ICML)*, pages 1312–1320, 2015a.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015b.

Tom Schaul, Diana Borsa, Joseph Modayil, and Razvan Pascanu. Ray interference: a source of plateaus in deep reinforcement learning. *arXiv preprint arXiv:1904.11455*, 2019.

Bruno Scherrer. Approximate policy iteration schemes: a comparison. In *International Conference on Machine Learning*, pages 1314–1322, 2014.

Juergen Schmidhuber. A general method for incremental self-improvement and multi-agent learning in unrestricted environments. In *Evolutionary Computation: Theory and Applications*. Scientific Publishing Company, 1996.

Wolfram Schultz, Peter Dayan, and P Read Montague. A neural substrate of prediction and reward. *Science*, 275:1593–1599, March 1997.

Paul J Schweitzer and Abraham Seidmann. Generalized polynomial approximations in markovian decision processes. *Journal of mathematical analysis and applications*, 110 (2):568–582, 1985.

Harm V Seijen and Rich Sutton. True online td (lambda). In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 692–700, 2014.

Sahil Sharma and Balaraman Ravindran. Online multi-task learning using active sampling. 2017.

Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550, 2017.

Satinder Singh, Richard L Lewis, Andrew G Barto, and Jonathan Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2(2):70–82, 2010.

Satinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1–3):123–158, 1996.

Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *SARA*, pages 212–223. Springer, 2002.

Rich Sutton, Ashique R Mahmood, Doina Precup, and Hado V Hasselt. A new q (lambda) with interim forward view and monte carlo equivalence. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 568–576, 2014.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998. URL http://www-anw.cs.umass.edu/~rich/book/the-book.html.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112 (1-2):181–211, 1999.

Richard S. Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 761–768, 2011.

Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010.

Fumihide Tanaka and Masayuki Yamamura. Multitask reinforcement learning on the distribution of mdps. In *Computational Intelligence in Robotics and Automation, 2003. Proceedings. 2003 IEEE International Symposium on*, volume 3, pages 1108–1113. IEEE, 2003.

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.

Yee Whye Teh, Victor Bapst, Wojciech M. Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4499–4509, 2017.

Gerald J. Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3), 1995.

Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Christophe Thiery and Bruno Scherrer. Least-squares λ policy iteration: Bias-variance trade-off in control problems. 2010.

Philip Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 2139–2148, 2016.

Sebastian Thrun. Is learning the N-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems (NIPS)*, pages 640–646, 1996.

Sebastian Thrun. *Explanation-based neural network learning: A lifelong learning approach*, volume 357. Springer Science & Business Media, 2012.

Sebastian Thrun and Tom M Mitchell. Lifelong robot learning. *Robotics and autonomous systems*, 15(1-2):25–46, 1995.

Pedro A Tsividis, Thomas Pouncy, Jaqueline L Xu, Joshua B Tenenbaum, and Samuel J Gershman. Human learning in atari. In *2017 AAAI Spring Symposium Series*, 2017.

Hado van Hasselt, A Rupam Mahmood, and Richard S Sutton. Off-policy td (λ) with a true online equivalence. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence, Quebec City, Canada*, 2014.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

Hado P van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems*, pages 4287–4295, 2016.

Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal networks for hierarchical reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3540–3549, 2017.

Sen Wang, Daoyuan Jia, and Xinshuo Weng. Deep reinforcement learning for autonomous driving. *arXiv preprint arXiv:1811.11329*, 2018.

Christopher Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, England, 1989.

Christopher Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):9, 2016.

Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pages 1015–1022. ACM, 2007.

Ian H Witten. An adaptive optimal controller for discrete-time markov environments. *Information and control*, 34(4):286–295, 1977.

Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. Deep reinforcement learning with successor features for navigation across similar environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2371–2378. IEEE, 2017.

Jiayu Zhou, Jianhui Chen, and Jieping Ye. Clustered multi-task learning via alternating structure optimization. In *Advances in neural information processing systems*, pages 702–710, 2011.