Temporal Abstraction and Generalisation in Reinforcement Learning

Matthew Smith

Hertford College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2023

Abstract

The ability of agents to generalise—to perform well when presented with previously unseen situations and data—is deeply important to the reliability, autonomy, and functionality of artificial intelligence systems. The generalisation test examines an agent's ability to reason over the world in an *abstract* manner. In reinforcement learning problem settings, where an agent interacts continually with the environment, multiple notions of abstraction are possible. State-based abstraction allows for generalised behaviour across different observations in the environment that share similar properties. On the other hand, temporal abstraction is concerned with generalisation over an agent's own behaviour. This form of abstraction allows an agent to reason in a unified manner over different sequences of actions that may lead to similar outcomes. Data abstraction refers to the fact that agents may need to make use of information gleaned using data from one sampling distribution, while being evaluated on a different sampling distribution.

This thesis develops algorithmic, theoretical, and empirical results on the questions of state abstraction, temporal abstraction, and finite-data generalisation performance for reinforcement learning algorithms. To focus on data abstraction, we explore an imitation learning setting. We provide a novel algorithm for completely offline imitation learning, as well as an empirical evaluation pipeline for offline reinforcement learning algorithms, encouraging honest and principled data complexity results and discouraging overfitting of algorithm hyperparameters to the environment on which test scores are reported. In order to more deeply explore state abstraction, we provide finite-sample analysis of target network performance—a key architectural element of deep reinforcement learning. By conducting our analysis in the fully nonlinear setting, we are able to help explain the strong performance of nonlinear neural-network based function approximation. Finally, we consider the question of temporal abstraction, providing an algorithm for semi-supervised (partially reward-free) learning of skills. This algorithm improves on the variational option discovery framework—solving a key under-specification problem in the domain—by defining skills which are specified in terms of a learned, reward-dependent state abstraction.

Temporal Abstraction and Generalisation in Reinforcement Learning

Matthew Smith

Hertford College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2023

Acknowledgements

Thank you to Shimon for the opportunity to come to a new place, to conduct exciting research, and to work with the best of the best. Thank you for your trust in me, and for all of the guidance along the way.

Thank you to Kamil for your technical insight, your considerate spirit, and for your foresight.

Thanks to Jelena, Max, and Kristian for helping me become a better collaborator, and for your dedication to our project through a challenging time.

Thank you Nana and Poppa for your ever-present interest, for paving the way for me, and for standing by me through thick and thin.

Mum, Dad, and Sara, you are my leaves and my roots. You nurture my growth and support me to stand tall. Thank you for helping me find ways forward when I felt that there were none. Your unconditional love and faith make me feel that I am not only the person I am meant to be, but also the person I want to be. Thanks for all of the wonder you inspire, the calls of "indomitable spirit", for encouraging ideas "so good, someone else already thought of them", and for the reminders that "it's only life, after all". I love you and I miss you. Together, we drive safely and drive carefully, headed somewhere new, with care and compassion en route!

To Mattie, I couldn't have done any of it without you. Thank you for including me, for your brilliance, for looking after me, for laughing with me, and for inspiring me.

For all those that I love and hold dear. Thank you for the beautiful moments, the care and patience that you have held for me, and for the joy that we have shared, without which this work would have no purpose.

Abstract

The ability of agents to generalise—to perform well when presented with previously unseen situations and data—is deeply important to the reliability, autonomy, and functionality of artificial intelligence systems. The generalisation test examines an agent's ability to reason over the world in an *abstract* manner. In reinforcement learning problem settings, where an agent interacts continually with the environment, multiple notions of abstraction are possible. State-based abstraction allows for generalised behaviour across different observations in the environment that share similar properties. On the other hand, temporal abstraction is concerned with generalisation over an agent's own behaviour. This form of abstraction allows an agent to reason in a unified manner over different sequences of actions that may lead to similar outcomes. Data abstraction refers to the fact that agents may need to make use of information gleaned using data from one sampling distribution, while being evaluated on a different sampling distribution.

This thesis develops algorithmic, theoretical, and empirical results on the questions of state abstraction, temporal abstraction, and finite-data generalisation performance for reinforcement learning algorithms. To focus on data abstraction, we explore an imitation learning setting. We provide a novel algorithm for completely offline imitation learning, as well as an empirical evaluation pipeline for offline reinforcement learning algorithms, encouraging honest and principled data complexity results and discouraging overfitting of algorithm hyperparameters to the environment on which test scores are reported. In order to more deeply explore state abstraction, we provide finite-sample analysis of target network performance—a key architectural element of deep reinforcement learning. By conducting our analysis in the fully nonlinear setting, we are able to help explain the strong performance of nonlinear neural-network based function approximation. Finally, we consider the question of temporal abstraction, providing an algorithm for semi-supervised (partially reward-free) learning of skills. This algorithm improves on the variational option discovery framework—solving a key under-specification problem in the domain—by defining skills which are specified in terms of a learned, reward-dependent state abstraction.

Contents

List of Figures

List of Abbreviations

ML . . . . . Machine learning.

RL . . . . . Reinforcement learning.

SL . . . . . Supervised learning.

AI . . . . . Artificial intelligence.

TD . . . . . The temporal difference algorithm [Sutton, 1988].

MDP . . . . . Markov decision process.

SA . . . . . Stochastic approximation.

SGD . . . . . Stochastic gradient descent.

HRL . . . . . Hierarchical reinforcement learning.

SMDP . . . . . Semi-Markov decision process.

IL . . . . . Imitation learning.

BC . . . . . Behavioural cloning.

ODE . . . . . Ordinary differential equation.

FPE . . . . . Fitted policy evaluation.

OIL . . . . . Offline imitation learning.

ORL . . . . . Offline reinforcement learning.

OPE . . . . . Off-policy evaluation.

ILBRL . . . . . Imitation learning by reinforcement learning.

PFPE . . . . . Partially fitted policy evaluation.

DIVRS . . . . . Diverse in value relevant spaces algorithm.

MI . . . . . Mutual information.

Notation

$\mathcal{S}$	MDP state space.
$s,\ s'$	MDP states.
$\mathcal{A}$	MDP action space.
a	MDP action.
P	MDP Transition function.
R	MDP random reward function.
r	MDP deterministic (expected) reward function.
ρ^0	MDP initial state distribution.
$\mathbf{r}$	Accrued future rewards (return).
γ	Discount factor for discounted MDPs.
π	Agent (target) policy.
ρ^π	Steady state distribution for policy π.
V	Value function.
Q	Action-value function.
δ	TD error.
μ	Behaviour policy for off-policy learning.
ϕ	Continuous state-action embedding.
θ	Value function parameters.
ω	Option index.
$\mathcal{I}_\omega$	Option initiation set.
β_ω	Option termination function.
D	Imitation learning dataset.

t^π	Mixing time of policy π.
$\delta(\theta, \theta', \varsigma)$	TD update with target networks.
$\delta(\theta, \theta')$	Expected TD update with target networks.
ρ_ς	Off-policy distribution over i.i.d. transition data.
H	Hessian of target network fitting loss.
J_δ	Jacobian in the second parameter of TD update vector.
J_{TD}	Jacobian of the TD(0) update vector.
(C)	PFPE condition function.
σ	Upper bound on the variance of an update.
$\lambda(A)$	Eigenvalues of matrix A.
$H[X]$	Entropy of the random variable X.
$I(X, Y)$	Mutual information between X and Y.

1

Introduction

Reinforcement learning (RL) is the sub-field of machine learning (ML) dedicated to the study of sequential behaviour. It is this *sequential* characterisation that sets RL outside of other forms of learning in terms of the complexity of problems that RL agents are able to solve, and also in terms of the challenges and pathologies that arise in the development and analysis of algorithms for RL. By sequential behaviour, we mean that RL agents must learn to make intelligent decisions, despite the fact that the data an agent ultimately observes is determined by the decisions that it has previously made. This gives rise to a host of challenges unique to RL. Instead of fitting a pattern-matching function to a static dataset with a static objective—as is done in supervised learning (SL)—RL algorithms must manage endemic issues such as the need for exploration, long-term credit assignment, and random data drawn from a distribution that may change over time. However, with these challenges comes great potential: questions of resource allocation [Rust, 1987], robotic automation [Khan et al., 2020], and even human and animal learning [Botvinick et al., 2019] are naturally modelled as sequential decision making problems. Indeed RL expresses learning in its most familiar form—people, animals, plants, and microorganisms are all driven to integrate knowledge from their environment and use that information to adapt and improve their conditions.

1.1 Generalisation and Abstraction

When deploying a learning agent, it is unreasonable to expect that the conditions it will experience over its life cycle will be identical to those that it encountered during learning. Taking our own experience as an example: if we have learned to cook a delicious new dish, then variations in ingredient seasonality and quality, atmospheric pressure and moisture, and imprecise tools for measuring key variables like volume and heat will all lead to subtle changes in the flavour of the final meal. We may need to subtly adapt our process to account for these shifting conditions. This brings to light key concerns of the RL problem, the intertwined notions of generalisation and abstraction. To prepare the dish, rather than exactly mimicking a procedure we have practised before, it helps to understand at an abstract level what needs to be done to prepare well-flavoured food. This leads to a more general procedure, in which specific steps, timings and even outcomes may vary, but overall, the final dish is somehow the same in the ways that matter, leading to a tasty meal.

If abstraction allows for reasoning over only the relevant aspects of a decision at hand, then generalisation is a test, asking "how effective is this abstraction"? Due to its complex nature, RL admits many different notions of generalisation and abstraction. In this work, we focus on three forms of abstraction: *state*, *temporal*, and *data* abstraction. These different axes describe distinct but entangled ways in which RL agents can reason abstractly, each requiring the use of distinct algorithmic techniques for both learning and planning. We now provide a brief overview of these sorts of abstraction, developing intuition as to the challenges, applicable settings, and capabilities of each one.

State Abstraction Perhaps the most common notion of abstraction in RL (and especially so in the SL setting) is the concept of state abstraction or function approximation. This sort of abstraction is less directly concerned with the sequential aspect of the RL problem, and is more concerned with the computation of and reasoning over features of state that are useful for prediction or action formulation, such that distinct states that are similar under an abstraction lead to similar expected outcomes or immediate actions. This is

perhaps the most mechanically granular sort of generalisation in RL, in that the test of a good function approximator is the ability of an agent to predict the outcome of its actions or to choose the right action, on a state by state basis. To underscore the ubiquity of state abstraction in ML, we note that SL problems that deal with sequential data (e.g. in language models [Wei et al., 2023]) are largely only concerned with state abstraction, as these approaches are concerned with making accurate predictions conditioned on (an abstraction of) the state given by the realisation of the sequence thus far. In contrast, many RL methods perform updates that are use the predictions of the agent as targets, making state abstraction less stable and more complex.

Temporal Abstraction In contrast to state abstraction, temporal abstraction (also referred to as behavioural abstraction) is relatively unique to the RL problem. Temporal abstraction focuses on abstract reasoning over sequences of behaviour: if we imagine a trip to the shop, then it is temporal abstraction that allows us to recognise that following the scenic route is somehow equivalent to using a more direct path, in that they both end up reaching our destination, despite the two paths involving different sequences of actions and observations along the way. When an agent is able to reason over its own behaviour in this way, it can accurately predict outcomes, assign credit for those outcomes to the most relevant decision points, and allow for coordinated exploration of the environment without the need to compare differences in behaviour irrelevant to the task at hand. In the context of automated systems, temporal abstraction can also help lead to more interpretable agent behaviour. The generalisation test for temporal abstraction then falls along these lines: "how well does the agent explore?", "how well is the agent able to assign credit?", and "how well can the agent reason over long term outcomes?"

Data Abstraction Moving to a wider picture of abstraction in RL, we can ask about how agents are able to make use of information gathered from one sampling distribution to make predictions and decisions on data from a different distribution. This is a form of abstraction in its own right, in that the characteristics of the problem we are looking to solve must be extracted from the source data in an abstract manner, instead of concretely

memorising these solutions. In SL, data abstraction is closely related to state abstraction—the primary concern is the ability of the agent to replicate patterns sampled from a finite dataset on the population. In RL, these concerns hold as well, but there are also more demanding problems that require data abstraction. In off-policy or offline RL, an agent must make inferences about its own behaviour using online data or a batch of data gathered by one or several agents following a completely different policy. While these notions exist in supervised learning as well, in RL they are much more significant, as a slight shift in behaviour at a single state may lead to wildly different data distributions downstream. Another form of data abstraction is present in the notion of transfer learning, where agents must use learning on one problem in order to improve learning or performance on another problem. The generalisation test here is concerned with evaluating the quality of predictions and decisions made under the target policy or in the target task.

These forms of abstraction, while conceptually separate, interact with each other. For instance, in the case of data abstraction, we may be interested in the ability of an agent to evaluate the quality of its behaviour using data gathered from another behaviour—then our generalisation test will be concerned with the quality of the value function, which also may be a question about the quality of the agent's state abstraction. Similarly, as an example, our behavioural abstraction may be concerned with an agent visiting the store, but it may not matter exactly which store the agent visits. This would mean that the behavioural abstraction is specified as attainment of an abstract state. As such, when we discuss a particular form of abstraction, other abstractions are also implicated, and the connections between different sorts of abstraction are a key theme of this work.

1.2 Contributions

This thesis examines the question of generalisability of sequential decision making agents along three connected axes: offline imitation learning, off policy deep reinforcement learning, and hierarchical reinforcement learning (HRL). Each of these settings allow us to focus on a different component of abstraction as described above, leading to novel analysis and the development of novel algorithms for learning and reasoning under the

corresponding sort of abstraction. We briefly discuss each setting and its relation to abstraction in RL below.

Offline Imitation Learning (Data Abstraction) We begin our study of data abstraction in a setting which is largely independent of the quality of state or temporal abstraction, in a tabular setting for which learned information is not directly shared across states. Here, an agent must learn to imitate the behaviour of an expert using a limited amount of data gathered from the environment. While this seems like a purely SL problem, in which the agent must simply learn the pattern mapping states to actions, the need for data abstraction rears its ugly head, as our agent will ultimately be evaluated on data sampled from its own behaviour, rather than that of the expert. We provide data complexity bounds for a novel algorithm in the tabular (without state abstraction) setting, reasoning explicitly over this distribution shift. We also investigate how this distribution mismatch has led to bad practice in empirical studies, proposing a more principled approach and demonstrating the benefits of our simple algorithm along the way.

Off-policy Deep Reinforcement Learning and the Deadly Triad (State Abstraction) The use of RL agents equipped with deep neural network function approximators—which represent an extremely powerful form of state abstraction—has made it possible to solve many tasks that were previously impossible [Heimerson et al., 2022, Yu et al., 2021, Mnih et al., 2015]. However one of the core pathologies of RL, known as the *deadly triad*—the combined use of state approximation, temporal difference algorithms, and off-policy data sampling—suggests that most practical deep RL methods should not converge in general. This represents one of the broadest gaps between theory and practice in RL algorithms. We study the use of a *target network*, an architectural decision commonly misunderstood to be a heuristic fix, and how this structure enables powerful state abstraction, even when we need to generalise across data sampled from different policies.

Hierarchical Reinforcement Learning (Temporal Abstraction) In order to enable agents to reason abstractly over their own behaviour, agent policies must take on more complex forms. Usually, this involves higher-order behaviours which call on lower-order

policies instead of directly interacting with the environment. This framework, known as HRL, focuses on how to learn behaviour structured in this way. Recent approaches to this problem have been focused on preventing redundant solutions by enforcing diversity in the lower-order policies. Yet this approach ignores the close relationship between temporal abstraction and state abstraction. We look to correct this by introducing a novel algorithm for learning meaningful state abstraction and using that to derive useful behavioural abstraction. Temporal abstraction is a key vehicle for allowing agents to perform data abstraction in transfer learning, and we empirically evaluate our approach in this setting, thus bringing together all three forms of generalisation in our approach.

In the imitation learning and off-policy learning settings, we take a more theoretical approach, allowing us to precisely bound the worst-case generalisation performance of agents. These settings require agents to learn about candidate behaviours using data that does not come from that behaviour. In the temporal abstraction section, we evaluate a novel algorithm from an empirical perspective in a setting which highlights how all three forms of abstraction can be used together in order to solve challenging problems that cannot be solved in the absence of coordinated generalisation.

Taken together, it is our hope that these approaches paint a picture of the ways in which different forms of abstraction and generalisation can interact, while also contributing to algorithmic development and analysis in each setting. To do this, we must first embed our work in the technical background and literature to date in the field, which we accomplish in the next two chapters, before proceeding to our principal analysis.

2
Background

Reinforcement learning is a broad field, drawing from developments across multiple domains, including probability theory, information theory, control theory, ODEs, statistics, supervised learning, and more [Sutton and Barto, 2018]. As such, this chapter provides a technical overview of the formal concepts that are used throughout the thesis. While not all the concepts presented here are used in every chapter, together they form the mosaic-background of the theoretical underpinning of this work. Some more specific background details are left to their respective chapters.

2.1 Reinforcement Learning

Reinforcement learning is concerned with the computational learning of sequential decision making in a stochastic environment. In RL problems, agents encounter a series of states at which decisions need to be made. The states eventually observed later in the sequence depend on the states encountered and the decisions made previously. RL problems tackle the complexity of such decision making by enforcing a formal separation between the agent and the environment, in which the environment specifies states, decisions available, and outcomes of those decisions, and the agent simply must act. This abstraction is highly flexible and applicable in many practical situations. As

such, as far as this thesis is concerned, RL problems are problems in which an agent interfaces with an environment over time. While the agent is generally represented by the output of an RL algorithm, and may take many forms, the Markov decision process (MDP) is generally used to represent the environment, which we will now describe in some detail.

2.1.1 Markov Decision Processes and Dynamic Programming

The MDP provides a formal basis for describing the environment with which an RL agent interacts. It is a flexible model that can accurately represent many real-world settings and allows flexibility for representations of states and uncertainty over the consequences of actions taken. The core limitation of the MDP is the requirement is that it must obey the Markov property—the distribution over outcomes of agent decisions must depend solely on the current state of the agent. While abstractions that eschew this limitation exist [Puterman, 2014], they incur an extra layer of complexity and are not considered here.

General MDPs Formally, a general MDP is specified as the tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \rho^0 \rangle$. $\mathcal{S}$ is the set of possible states which the agent may occupy. $\mathcal{A}$ is the set of actions that the agent can take. $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the probability density function which specifies the state-to-state conditional transition dynamics of the MDP, with $P(s'|a, s)$ representing the probability that the agent transitions to state s' after taking action a in state s. $R : \mathcal{S} \times \mathcal{A} \to ([r_{\min}, r_{\max}])$ gives a random bounded reward, with $-\infty < r_{\min} \leq r_{\max} < \infty$, which specifies the signal which we look to optimise in the long run. In some chapters we use a deterministic reward corresponding to $r : \mathcal{S} \times \mathcal{A} \to [r_{\min}, r_{\max}]$, which can also be interpreted as the expected value of R. $\rho^0 : \mathcal{S} \to \mathbb{R}$ gives the initial state density. The agent interacts with the environment at discrete time steps. At step t, action a_t is taken in state s_t, $s'_t = s_{t+1}$ is sampled from $P(\cdot|s_t, a_t)$, and reward $r_t = R(s_t, a_t)$ is yielded. This interaction loop begins anew in state s_{t+1}.

The notion of optimising the scalar reward accrued in the long run after starting from an initial state, proves to already incur complexity in our model of sequential decision

making. We want to reserve the possibility of an infinite, continuing interaction between agent and environment, and in general the sum of encountered rewards:

$$\mathbf{R} = \sum_{t=0}^{\infty} R_t,$$

may not converge. As such, we generally consider either *finite horizon* MDPs, *discounted* MDPs or *average reward* MDPs, each of which modify this sum in a different way to ensure convergence.

Finite Horizon, Discounted, and Average Reward MDPs The *finite horizon* MDP gives the simplest formulation for managing long run optimisation of cumulative rewards. As the name suggests, rather than considering the rewards across an infinite horizon, only the first T rewards are considered:

$$\mathbf{R}_T = \sum_{t=0}^{T} R_t.$$

Generally, this setting assumes an eventual resetting of the agent after T timesteps, leading to resampling an initial state from ρ^0, after which another rollout begins. This is also referred to as *episodic* learning. The explicit notion of horizon leads to the simplest setting that alleviates the need to reason over rewards arbitrarily far in the future.

Discounted MDPs retain the infinite sum, but employ an effective horizon by geometrically decaying rewards received in the future. The MDP is augmented with a *discount factor*, $\gamma \in [0, 1)$, and we optimise the resulting series:

$$\mathbf{R}_\gamma = \sum_{t=0}^{\infty} \gamma^t R_t.$$

This approach is often justified as representing uncertainty over the fact that the agent will continue to interact with the environment after step t–i.e. γ represents a probability that the agent terminates operation at step t, after which it must be reset [Sutton and Barto, 2018].

Finally, *average reward* MDPs trade-off rewards equally at all time horizons through the use of the limit:

$$\mathbf{R}_\rho = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} R_t.$$

This form is powerful in that it eliminates all myopia from the optimisation target, removing any dependency on the starting state. However, this form is somewhat more unwieldy mathematically, and does not reflect the fact that in practice, we are generally concerned with the immediate behaviour of the agent, rather than purely asymptotic performance.

In this work, we generally focus on the discounted setting as it is the most well-studied, though Chapter 4 makes use of both discounted and average-reward setting. More details about the relationship between these settings can be found there.

Agents and Policies To represent agent behaviour in the MDP, we construct a *policy*, denoted $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a probability density function over actions to be taken by the agent in state s. We can view an agent as a mapping from states to a distribution over actions. An optimal policy, which solves the MDP for a particular criterion, x, is denoted π^*_x, and obeys:

$$\pi^*_x = \arg\max_{\pi} \mathbb{E}_{\pi,P} \left[\mathbf{R}_x | S_0 = s\right],$$

for all $s \in \mathcal{S}$. The expectation is taken over the potentially infinite random sequence $(A_0, R_0, S_1, A_1, ...)$, representing the random states, actions and rewards encountered by repeatedly drawing from π and P. We also introduce $P_\pi(s'|s) := \mathbb{E}_{a\sim\pi(s)} \left[P(s'|s, a)\right]$, the state-to-state conditional distribution of a policy. We say that the steady state distribution of a policy,

$$\rho_\pi(s) = \lim_{t\to\infty} \int_{s_{t-1}} P(s_t|s_{t-1}) \int_{s_{t-2}} P(s_{t-1}|s_{t-2}) \Big[...\Big] \int_{s_0} P(s_1|s_0)\rho_0(s_0) ds_0 \Big[...\Big] ds_{t-2} ds_{t-1},$$

is the limiting distribution over states induced by that policy for an MDP. In this work, we only consider *unichain* MDPs [Puterman, 2014], for which such a limiting distribution always exists.

Bellman Equations and Value Functions Fundamental to the notion of sequential decision making is the notion of dynamic programming. Dynamic programming is a paradigm in which a complex problem is recursively broken into simpler sub-problems. In general the solutions to these sub-problems may depend on a combination of the solutions

to the sub-problems which they depend on. In sequential decision problems, evaluation of a policy quality can be broken up into a sum of the immediate rewards incurred by following that policy, and then the evaluation of future rewards in accordance with the appropriate criterion. The equations formalising this relationship are known as *Bellman Equations* [Bellman, 1952].

In the finite horizon case, the expected sum of future returns, or *value* of a policy at a state, and time, denoted $V_\pi^T : \mathcal{S} \times \mathbb{N} \to \mathbb{R}$ is given by the unique solution to the Bellman Equation:

$$V_T^\pi(s,t) = \mathbb{E}_{A\sim\pi(s),r\sim R(s,A)}\left[r + \mathbb{E}_{P(S'|s,A)}\left[V_T^\pi(S',t+1)\right]\right],$$

with base case $V_T^\pi(s,T) = 0$ and S' representing the random variable for the subsequent state, with realisation s'. We see this way how information about rewards is propagated back through the states of the MDP. Clearly at the horizon, there is no future rewards to be received, so $V_T^\pi(s,T-1)$ depends only on the expected reward at state s. In earlier time steps however, values depend on both the expected rewards received and the values of the states that are expected to be reached in subsequent steps.

The discounted reward criterion value function obeys a similar form, though it no longer depends on the horizon directly:

$$V^\pi(s,\gamma) = \mathbb{E}_{A\sim\pi(s),r\sim R(s,A)}\left[r + \gamma\mathbb{E}_{P(S'|s,A)}\left[V^\pi(S',\gamma)\right]\right].$$

Now the value at a state depends on both the immediate reward, and the values of all states visited in the future, decayed geometrically at rate γ. We can observe how the geometric decay affects rewards received in the future by continuing to unroll the Bellman Equation:

$$V^\pi(s,\gamma) = \mathbb{E}_{A\sim\pi(s)}\left[r(s,A) + \gamma\mathbb{E}_{P(S'|s,A)}\left[\mathbb{E}_{A'\sim\pi(S')}\left[r(S',A') + \gamma\mathbb{E}_{P(S''|S',A')}\left[V^\pi(S'',\gamma)\right]\right]\right]\right].$$

Here S'' is the random variable representing the state visited two time steps into the future. We can see that the expected reward obtained at S' is decayed by a factor of γ, and that values (and thus rewards) obtained at states S'' and beyond will be decayed again, leading

to an overall coefficient of γ^2. If we continue to unroll this recursion to infinity, we will recover a form identical to that of $\mathbf{R}_\gamma$.

Before we continue on to the average reward formulation of the value function, we present some additional forms of Bellman equation in this context, as this is the setting we operate in for the most part. We drop the explicit dependence of value functions on γ from here on, as such dependence is generally fixed for the MDP, and clear from context. We also move to using the deterministic reward $r(s, a)$, since this simplifies notation. The *action value* function, generally denoted $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ reflects the value of taking action a in state s, then following policy π for all subsequent transitions:

$$Q^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{P(S'|s,a)} \left[V^\pi(S') \right].$$

Removing the dependency on the policy at the current state can lead to computational tractability and efficiency later on when solving MDP problems.

We also present the Bellman *optimality* equation

$$V^*(s) = \sup_{a \in \mathcal{A}} \left[r(s, a) + \gamma \mathbb{E}_{P(S'|s,a)} \left[V^*(S') \right] \right],$$

in which the expectation is replaced by the supremum over actions. For a given MDP there is only one $V^*(\cdot)$ which solves the optimality equation. It is known as the optimal value function, and it corresponds to the value of following an optimal policy in the MDP. This elucidates the core connection between the optimal policies and optimal values in an MDP: for any MDP, there exists a deterministic policy that is optimal for that MDP, which is greedy with respect to the optimal value function [Puterman, 2014].

So, an optimal policy follows naturally from the optimal value function, with:

$$\pi^*(s, a) = \mathbb{I}\left(a = \arg\max_{a \in \mathcal{A}} Q^*(s, a) \right),$$

where $\mathbb{I}(a' = a_{i+1})$ evaluates to 1 if the condition in its argument is met and 0 otherwise. $Q^*(s, a)$ is the optimal action-value function:

$$Q^*(s, a) = r(s, a) + \gamma \mathbb{E}_{P(S'|s,a)} \left[V^*(S') \right].$$

In the subsequent section, we will leverage this relationship in order to develop iterative solutions to the MDP. First, for completeness, we briefly discuss the average reward setting. For the rest of the chapter, we proceed to consider only the discounted setting.

We note that, while $\mathbf{R}_T$ and $\mathbf{R}_\gamma$, directly admit notions of value that are state dependent, for the unichain MDPs [Puterman, 2014] considered in the average reward section of this thesis, $\mathbf{R}_\rho$ is state-independent, due to the absence of an (implicit) finite horizon. In this work, such a formulation is acceptable, as it will only be used later for matching a limiting distribution of an expert policy. In the literature, this is referred to as *gain optimality*, and is the weakest of several average reward criteria. See e.g. [Mahadevan, 1996] for a more thorough discussion of this setting.

Value and Policy Iteration If the solution to an MDP is an optimal policy at each state, how do we find such a policy? The value iteration and policy iteration algorithms leverage dynamic programming paradigms to solve such a problem when the MDP dynamics, P and r, are known and tractable. These algorithms exploit the fact that the Bellman equations can be expressed as affine operators applied to the space of value functions. For a policy, π, we define the corresponding Bellman operator, $\mathcal{T}_\pi$, elementwise as follows:

$$(\mathcal{T}_\pi Q)(s,a) = r(s,a) + \gamma \mathbb{E}_{P(S'|s,a)} \mathbb{E}_{A' \sim \pi(S')} \left[Q(S', A') \right].$$

The policy iteration algorithm iterates between phases of *policy evaluation* and *policy improvement*. Starting with an arbitrary initial policy, π_0 and value function, Q_0, we repeatedly apply $\mathcal{T}_{\pi_i}$, leading to the update scheme:

$$Q_{j+1} = \mathcal{T}_{\pi_i} Q_j.$$

By Banach's fixed point theorem [Banach, 1922], and the fact that $\mathcal{T}_{\pi_i}$ is a γ-contractive operator [Puterman, 2014], iterates of this form will converge to the true value Q^{π_i}. Once we are sufficiently close to convergence (our value function is close to estimating the true

value of our provisional policy), we can apply a step of *policy improvement*, in which a new greedy policy is chosen based on the values of the previous policy:

$$\pi_{i+1}(s,a) = \mathbb{I}\left(a = \arg\max_a Q^{\pi_i}(s,a)\right).$$

Given the new policy, another round of policy evaluation is performed, and iteration proceeds until $\pi_{i+1} = \pi_i$, at which point we have the optimal policy.

Policy evaluation combines the policy improvement and policy evaluation steps by iterating directly the Bellman Optimality Operator, defined for an action value function, Q with:

$$(\mathcal{T}_* Q)(s,a) = r(s,a) + \gamma \mathbb{E}_{P(S'|s,a)}\left[\sup_{a' \in \mathcal{A}} Q(S', a')\right].$$

We note that this is also a γ-contractive affine operator in infinity norm, though the fixed point here is the value function corresponding to the optimal policy. So iterates $Q_{i+1} = \mathcal{T}_* Q_i$ converge at a geometric rate to the optimal action-value function. Once iterates have converged, we can then decide the optimal policy directly with the relation $\pi^*(s,a) = \mathbb{I}(\arg\max_a Q^*(s,a))$.

2.1.2 Approximate Dynamic Programming and Temporal Difference Methods in Tabular MDPs

In order to apply the value and policy iteration algorithms as described above, the dynamics of the MDP must be known. Specifically, to compute the relevant expectations, we need access to $r(s,a)$, and $P(s'|s,a)$ for all s, a, s'. Clearly if MDPs have large or infinite state and action spaces, or if we are interacting with an MDP that has unknown dynamics, explicit computation of such expectations is infeasable. In order to handle policy evaluation in such situations, Temporal Difference (TD) methods are generally employed. TD methods make use of intermediate value predictions as learning targets, leading to a process called bootstrapping. When state spaces are finite, TD methods act as stochastic approximation (SA) methods for policy evaluation or value iteration. Interestingly, TD methods also allow for extension of similar algorithms into MDPs with potentially infinite state and action spaces through the use of *function approximation*, which we will detail later in this section.

When the state and action spaces are small enough to be represented as a finite table, policy evaluation can be performed using the tabular TD(0) algorithm. Instead of evaluating the expectation across all states and actions simultaneously, we seek to learn from an agent actively sampling individual states and actions from an environment.

Policy Evaluation with Tabular TD(0) Again we begin with an arbitrary value function Q_0. For an agent currently occupying state s_i, we sample a_i from the policy we are trying to evaluate, $\pi(s)$. The agent receives reward $r_i = r(s_i, a_i)$, and then arrives in state s_{i+1} as sampled from $P(S'|s_i, a_i)$. After sampling the subsequent action, $a_{i+1} \sim \pi(s_{i+1})$, we can then update our value function estimate as follows. We define the TD-error at step i as:

$$\delta_i = r(s_i, a_i) + \gamma Q_i(s_{i+1}, a_{i+1}) - Q_i(s_i, a_i).$$

This encodes a sample-based estimate of the self-inconsistency of our estimate value at a particular (sampled) state. Due to the uniqueness of the fixed point, only a correct value function will encounter TD error $\delta_i = 0$ at all such states in expectation.

This error is then used to update the value at s_i, a_i in accordance with standard SA technique. Making use of a time-dependent scalar learning rate α_i, we conduct the update:

$$Q_{i+1}(s_i, a_i) = Q_i(s_i, a_i) + \alpha_i \delta_i,$$

where the sequence $[\alpha_i]$ is subject to the standard SA Robbins-Monroe constraints:

$$\sum_{i=0}^{\infty} \alpha_i = \infty, \qquad \sum_{i=0}^{\infty} \alpha_i^2 < \infty.$$

This leads the value at the encountered state to be adjusted towards the value encountered. The term $\gamma Q_i(s_{i+1})$ is referred to as a *bootstrap*, as the value of $Q_i(s_i, a_i)$ is effectively using the estimated value of the subsequent state (and the encountered reward) as an update target. A simple argument based on standard results in SA shows that this algorithm indeed will converge almost surely to the true value of the policy followed.

We note that the "0" in TD(0) comes from a generalisation of the algorithm, known as TD(λ), which implements a second set of values known as the *eligibility trace*. This

allows for trade-offs between the bootstrapped estimate of return shown here and many step Monte Carlo returns, which unroll the Bellman equation further, incorporating more of the rewards encountered before bootstrapping at the cost of higher variance. This is an important and well-studied generalisation, but this work is strictly concerned with TD(0) and related algorithms, though we expect many of our results to generalise to the use of eligibility traces. For more information on TD(λ), see Sutton and Barto [2018].

Control with Q-Learning In a control setting, where we are looking to learn an optimal policy, we can perform a similar procedure as TD(0), in which we sample a temporal difference error and update parameters accordingly. Here, the error is based on the Bellman optimality operator instead of the policy sampled variant. This leads to an algorithm known as Q-learning. Once more, starting from an arbitrary value function Q_0, and stochastic policy π_0, in state s_i, we sample a_i, r_i and s_{i+1}. Then we can compute the sample TD error for Q-learning:

$$\delta_i \coloneqq r_i + \gamma \sup_{a' \in \mathcal{A}} Q_i(s_{i+1}, a') - Q_i(s_i, a_i).$$

We see that this error also measures the gap between our best guess of the value at the current state, and the sample-based bootstrapped return. Here, our bootstrap corresponds to the action which we expect to yield highest returns under our current estimate of the value function. Notice that

$$\mathbb{E}\left[\delta_i\right] = \mathcal{T}^* Q_i(s_i, a_i) - Q_i(s_i, a_i),$$

and so we expect the error to be 0 only when we achieve the value of the optimal policy, since Q^* is the unique solution to the Bellman equation $Q = \mathcal{T}Q$. This gives us some indication that we expect Q-learning to converge to the optimal value function. The value function update:

$$Q_{i+1}(s_i, a_i) = Q_i(s_i, a_i) + \alpha_i \delta_i,$$

is then the same as in TD(0), with $[\alpha_i]$ obeying the Robbins-Monroe conditions. Based on the contractivity of the Bellman optimality operator, so long as π_0 samples all states and actions across the MDP with sufficient frequency, following similar logic to that of

TD(0), values converge to those of the optimal policy almost surely. We can then read the optimal deterministic policy $\pi^*(s,a) = \mathbb{I}(a = \arg\max_{a\in\mathcal{A}} Q^*(s,a))$.

It is important to note that in Q-learning as we have constructed it so far, states are continuously sampled from a fixed policy, while the values being learned increasingly track the values of the optimal policy. This paradigm, in which we sample from one policy but learn about another is known as *off-policy* learning. Off-policy learning is a very powerful paradigm, as in concept, it allows agents to lean about many policies from a single stream of data. However, in general, it may be very inefficient to continue to sample from π_0 indefinitely. We may want to use intermediate values to generate intermediate policies which cover the relevant regions of state space more efficiently. We will discuss this, as well as a more general algorithm for off-policy learning in the next section.

Off-Policy Methods

As mentioned previously, Q-learning is an *off-policy* method for learning the optimal policy in an MDP. Rather than maintain a fixed exploration policy until convergence, we may want to use intermediate values to gather data from parts of state space that we have learned to be more relevant to the optimal policy. The naive approach to this is to greedily follow the current best-guess optimal policy:

$$\pi_i(s,a) = \mathbb{I}(a = \arg\max_{a\in\mathcal{A}} Q_i(s,a)).$$

This corresponds to Q-learning *on-policy*, and leads to a pathological situation which fails to visit all state-action pairs. It may be the case that the value of some action in some state is underestimated, and only by taking that action can the value be updated. This means that we will not be sampling from the optimal Bellman operator anymore and may fail to learn the optimal policy. However we can take advantage of the off-policy nature of Q-learning in order to ensure that we sample all actions in all states with some minimum rate. This gives rise to the notion of ϵ-greedy policies, which sample the greedy policy with probability $1-\epsilon$, and take a random action with probability ϵ. This sort of policy ensures that underestimated values are eventually resolved, as there is some small

probability that any action will be taken in any state, giving us good coverage of the MDP once more. At the same time, the actions taken with highest probability are those of the greedy policy, so we can expect more frequent updates in the regions of state space that correspond to our current understanding of optimal behaviour.

Expected SARSA ϵ-greedy exploration targeting the deterministic optimal policy is only one particular combination in the matrix of off-policy learning algorithms. This particular approach combines the problem of control with the problem of value estimation. To explore off-policy learning more abstractly, we restrict ourselves to the policy evaluation setting. We now introduce a more general algorithm for off policy learning schemes known as Expected SARSA [Sutton and Barto, 2018].

Expected SARSA involves the use of both a target policy, π and a behaviour policy μ, both of which can change over time. The behaviour policy determines how states are sampled, while we look to learn the values of the target policy. Again we construct a temporal difference error as in Q-learning, but instead of taking the maximum over next actions, we take the expected next action according to π_i:

$$\delta_i = r_i + \gamma \sum_{a' \in \mathcal{A}} \pi_i(a'|s') Q_i(s_{i+1}, a') - Q_i(s_i, a_i).$$

Because π_i can be chosen in a time dependent way, this leads to an incredibly flexible update scheme which covers all of the algorithms we have discussed previously. For example, say we choose $\mu_i(s) = \pi_i(s)$ and $\pi_i(a'|s') = \mathbb{I}(a' = a_{i+1})$. In this case, we recover TD(0) policy evaluation. If π_i is chosen to be the greedy policy with respect to Q_i, and μ_i the ϵ-greedy policy, we recover ϵ-greedy Q-learning.

Other options are possible as well. If $\mu_i = \mu_0$ and $\pi_i = \pi_0$, we have off-policy expected policy evaluation, in which we evaluate policy π using data from μ. An additional benefit to this scheme is that, by taking the expectation over actions sampled in the next state, rather than the single sample estimate as in TD(0), we achieve a bootstrap target with reduced variance, as the only random variable is the state from which we are bootstrapping.

2.1.3 TD Methods with Function Approximation

In many practical settings—such as robotic manipulation, complex resource allocation problems, or medical decision making problems—state and action spaces become too large to be represented in a table. These settings admit infinite or continuous representations of states and actions. In such cases, new challenges arise. Firstly, we must choose a functional form and parameterisation of our value functions—these choices may constrain learning and not allow for representation of the true value function for any given policy, or fail to capture the optimal policy at every state. Secondly, our update schemes from before based on the Bellman optimality operator which require maximisation over the action space may not be possible.

Regarding the first concern—which generally revolves around large state spaces—we may be interested in learning some notion of the best possible value function approximator within our chosen function approximation class. We also may be interested in the learning behaviour of algorithms for which our chosen representation satisfies certain assumptions which may or may not hold for all practical problems. The second concern—focused on large action spaces—demands the application of different algorithmic approaches, the most popular of which are the *policy gradient approaches*. This section concerns itself mostly with continuous state spaces, while continuous action spaces are left to later chapters.

Representing Continuous State Spaces When faced with large state spaces, an important additional design decision arises: how do we represent states in the MDP such that we can learn from them? The standard approach is to introduce an additional fixed mapping $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^n$, which projects state-action pairs into n-dimensional vector space. Such vectors are referred to as feature vectors (in reference to supervised learning), though occasionally we will refer to the feature vector $\phi(s, a)$ as the state itself when context is clear, since this is the mapping of the state that the agent encounters. Implicit in such a mapping is the notion of *state similarity*: state-action pairs which are somehow similar should map to feature vectors which are close in a Euclidean sense. Furthermore, in order to maintain the Markov property, ϕ must be injective, which is to

say that distinct state-action pairs map to distinct feature vectors–we cannot allow state or action aliasing under the mapping ϕ.

In tandem with our observation function ϕ, we must construct a value function, $Q : \mathbb{R}^n \rightarrow \mathbb{R}$, which can take as input a state-action feature vector and return the (estimated) value of taking such an action in the corresponding state. We generally make use of a set of modifiable parameters, $\theta \in \mathbb{R}^m$, which encode the set of computations to be performed on $\phi(s, a)$ leading to the output of Q. To denote the dependency of Q on θ, we often denote it Q_θ, though sometimes this explicit representation is dropped where this dependency is clear.

This is a powerful and very general scheme for approximation of value functions in continuous state spaces. One simple and powerful formulation of this approach to function approximation involves the use of a linear value function, which expresses values as a linear combination of state features. Here, we take $\theta \in \mathbb{R}^n$, and define:

$$Q_\theta(\phi(s, a)) = \theta^\top \phi(s, a),$$

as the inner product between parameters and feature vectors. It is clear that this scheme is a generalisation of the tabular representation–a tabular MDP is given by $n = |\mathcal{S}| \cdot |\mathcal{A}|$, with $\phi(s, a) = \mathbf{e}_{s \cdot |\mathcal{A}| + a}$, where $\mathbf{e}_i$ is the vector with 1 at element i and 0 otherwise. We then have $\theta_{s \cdot |\mathcal{A}| + a} = Q(s, a)$ to represent our proposed value function.

TD Methods with Function Approximation Now that we are using function approximation, it is clear that the functional form of our approximator may not be sufficiently rich to represent the true value function. That is to say, there may not exist a theta for a fixed form of Q_θ such that solutions exist to:

$$Q_\theta = \mathcal{T}^\pi Q_\theta.$$

The most common alternative objective is to consider the *projected* Bellman equation, which suggests that the projection of our value function into the space of functions

representable using our function approximation class under the Bellman operator should achieve a fixed point:

$$Q_\theta = \Pi^\mu \mathcal{T}^\pi Q_\theta.$$

Π^μ represents the Euclidian projection operator with respect to the metric defined by the behaviour policy, that is:

$$\Pi^\mu(Q) \coloneqq \arg\min_{Q_\theta} \|Q_\theta - Q\|_\mu.$$

A fixed point of the projected Bellman equation represents the minimum expected TD error within our function approximation class.

Under light conditions, in the common on-policy learning with linear function approximation setting, the projected Bellman operator is also a contraction, and we can expect that repeated application of it will lead to a fixed point. While in general we may be interested in nonlinear forms of function approximation, these settings can involve nonconvex optimisation in general, and are thus much more difficult to study. In this section we focus on linear function approximation forms, leaving a discussion of practical techniques for nonlinear approximation to the next section.

In order to solve the minimisation problem described by the projection operator, many techniques are possible. Here we focus on a subset of such methods known as *fitted* TD methods. These techniques make use of a second set of parameters, θ', which are periodically fixed, and used to track the application of the Bellman operator on our candidate value function. If we set $\theta' = \theta$, then application of our Bellman operator looks like:

$$\Pi^\mu \mathcal{T}^\pi Q_\theta = \arg\min_{Q_\theta} \|Q_\theta - \mathcal{T}^\pi Q_{\theta'}\|_\mu, \tag{2.1}$$

which allows for minimisation using conventional methods. For example, we can apply stochastic gradient descent in θ to the squared distance in order to apply a single step of the projected Bellman equation. This is a standard regression problem with targets $\mathcal{T}^\pi Q_{\theta'}$.

While solving the sequence of regression problems specified as in Eq. (2.1) is possible, it is not necessary to fully complete the minimisation specified here. In fact, it is often the case

that only a single gradient descent step is needed to ensure convergence. When sampling states rather than taking expectations explicitly, this leads to the update scheme:

$$\begin{aligned}\delta_i &= r_i + \gamma Q_{\theta'_i}(s_{i+1}, a_{i+1}) - Q_{\theta_i}(s, a),\\ \theta_{i+1} &= \theta_i + \alpha_i \delta_i \nabla_{\theta_i} Q_{\theta_i}(s_i, a_i),\\ \theta'_{i+1} &= \theta_{i+1}.\end{aligned}$$

We can see that when tabular function approximation is used (represented in the linear form described above), this reduces exactly to the tabular TD update. This process is occasionally referred to as *semigradient* TD: the update matches that of TD, but also looks as though we are taking the gradient of the squared TD error with respect to the values of the current state, while treating the value of the subsequent state as fixed with respect to the value function parameters. While the semigradient description is often used, more formally, we are simply taking a single sample approximation of stochastic gradient descent (SGD) in the projected Bellman error. This will be explored further in Chapter 5.

Of course, there is a limitation in the use of function approximation. Since the projection operation depends on the data-gathering policy and the Bellman operator depends on the target policy, if these policies do not match, together they do not form a contraction and may in general be expansive. As such, in order to guarantee convergence of fitted TD methods, data must be gathered on policy, in absence of other assumptions that allow for convergence. We explore such conditions in more detail in Chapter 5.

2.1.4 Deep Reinforcement Learning

In more complex environments, even best in class linear function approximation may leave much to be desired in terms of value function representation or corresponding policy performance. In such situations, nonlinear function approximation forms are used. The most common form of nonlinear function approximation is the deep neural network [Goodfellow et al., 2016], which leads to the field of deep reinforcement learning. Due to the nonconvex optimisation sub-problems implied by the use of neural networks (i.e. optimisation of Eq. (2.1) with Q_θ specified by a neural network with weights given by θ),

fewer guarantees can be given regarding convergence properties and performance of deep RL algorithms. However, the ability of neural networks to represent massive classes of functions ensures that the true values of any policy can be better represented by neural networks than linear parameterisations. As such, many recent works have developed practical techniques for solving RL problems with DNN function approximation. Here, we investigate some of the common practices that will be discussed later in this work. The primary two techniques applied in this way are known as *experience replay* [Lin, 1992], and *target parameters* [Mnih et al., 2015]. While many novel developments have been formulated in the deep RL space since then, these techniques form the basis of most such methods, and most of the novel developments in value-based deep RL methods represent incremental tweaks to these practices.

Target Parameters As was suggested in our analysis of TD with general function approximation, in order to apply the projected Bellman operator, there exists an optimisation subproblem representing the projection step. In the deep RL literature, the practice of freezing a secondary set of target parameters, and minimising the gap between target "slow" values under the Bellman operator, and the "fast" value function parameters, is referred to as using a target network. Often, rather than freezing the parameters for a single step of optimisation, parameters remain static through many steps of updates. Alternatively, in more recent works, a Polyak averaging scheme is used to update target parameters continuously.

Historically, this practice has been justified in an ad-hoc manner as providing additional stability for the value network, by keeping targets fixed for many steps. However, we can see from the way in which we have introduced the TD update with function approximation that it is simply a small-sample estimation of an application of the projected Bellman operator. Later in this thesis, we explore in more detail the theoretical advantage that target parameters provide.

Experience Replay Rather than performing TD updates online using single samples, one can consider performing batch updates, where data is sampled i.i.d. from a fixed-length buffer of past states and actions. If the data-generating policy remains fixed and the buffer is cleared after each update, then this is equivalent to performing mini-batch optimisation of Eq. (2.1). In practice, the buffer is never cleared and policies are changed over time, though the argument is made in theoretical contexts that the value function updates are slow enough and smooth enough to warrant such characterisation. While experience replay may be generally useful [Lin, 1992], it is particularly important in the deep RL context for two reasons.

First, neural networks are difficult to optimise from single-sample SGD methods, since the variance that arises from such complex function approximation tends to be greater than that in the linear case. As such, mini-batch gradient descent methods are generally preferred [Goodfellow et al., 2016]. By storing past experiences to use in later updates, methods implemented with experience replay can average sample gradients across several state-action pairs, leading to lower variance updates and more stable learning.

In addition, neural networks can be particularly susceptible to the effects of correlated data. Due to the nonlinearities present, the order in which data is observed has significant effect on the learning process. As such, feeding the network several correlated state-action pairs may lead to parameter configurations that are inescapable. By sampling i.i.d. from a dataset of potentially millions of past state-action pairs, the data fed through the network becomes less correlated, and is thus less likely to lead to pathological parameter configurations.

While deep RL methods have allowed for powerful forms of value function representation, enabling the automatic learning of relevant forms of state abstraction, the concepts we have covered thus far provide no means for an agent to reason abstractly over its own behaviour. In the next section, we provide the formalisms needed to formulate such behavioural abstractions.

2.1.5 Temporal Abstraction in Reinforcement Learning

In RL, temporal abstraction involves reasoning over several similar sequences of actions as though they were a single, higher-order action. The biological metaphor here is highly clarifying–when people reason over their own behaviour, rather than considering raw action space consisting of coordinated muscle twitches, they think about moving their legs, and arms, picking a wildflower, or writing a letter to a friend. The field of study concerned with these processes of abstraction is generally referred to as "hierarchical" reinforcement learning (HRL).

Temporal abstraction allows for faster propagation of information through time, as sequences of actions can be considered as a single decision. Furthermore there is some evidence to suggest that HRL methods can lead to more efficient exploration of state space [Nachum et al., 2019c], and that they can help with human interpretability of agent behaviour. HRL methods are generally formulated using a modified version of the MDP known as a Semi-Markov decision process (SMDP), which we detail here.

Semi Markov Decision Processes The SMDP represents a slight modification from the MDP in which—in addition to a fixed distribution over subsequent states conditioned on the current state and action—we must be able to define a static distribution over the time that it takes to reach that subsequent state. We formalise this notion by modifying our state transition function to the multivariate form: $P' : \mathcal{S} \times \mathcal{A} \times (\mathcal{S} \times \mathbb{R}^+) \to \mathbb{R}$, where $P(s', t|a, s)$ gives the probability of arriving in state s' after duration t, after taking action a in state s. This form now describes a joint distribution between subsequent states and the time that it takes to arrive in such a state. In general, while state-to-state transition time may be represented by a distribution over positive reals, in the context of temporal abstraction for RL, we focus on arrival times given by natural numbers–representing the number of states visited in the base MDP between decision epochs–so there is an additional restriction: $t \in \mathbb{N}$.

Fortunately, at the SMDP level, value iteration and policy iteration–as well as their TD-based approximations–can be applied almost directly, requiring only a modification

of how discounting is applied in order to evaluate policies and solve SMDP problems [Puterman, 2014]. In RL, several approaches can be used to tie together an MDP with temporal abstraction in order to form an SMDP, though the most common is known as the *options framework*, which we now discuss.

The Options Framework The options framework [Sutton et al., 1999] provides the formal machinery for specification of temporal abstraction in RL. An option, ω is specified by the tuple $\langle \mathcal{I}_\omega, \pi_\omega, \beta_\omega \rangle$. $\mathcal{I}_\omega \subseteq \mathcal{S}$ is the set of states from which the option can be selected. This acts as a mechanism for which we can restrict the abstract affordances of an agent at states in which such behaviours are impossible. We cannot perform the abstract action "pick up a glass" if there are no glasses to pick up! However, in current practice it is common to disregard this sort of constraint by taking the initiation set to be the entire state space. $\pi_\omega : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a normal RL policy over primitive actions, which defines the behaviour that constitutes our temporal abstraction. $\beta_\omega : \mathcal{S} \to [0, 1]$ is a *termination function*, which maps states to the parameter of a Bernoulli random variable representing the termination of the active option, signifying the need for a new decision at a higher level of abstraction (over options).

Generally, agents are then equipped with a set of options, denoted Ω. Decisions at the abstract (SMDP) level are generally made by an additional *policy over options*, $\pi_\Omega : \mathcal{S} \times \Omega \to \mathbb{R}$, which determines a state-conditional probability distribution over options. The execution model of an agent with options can then be thought of as such: given an initial state, the agent samples from the policy over options in order to select an abstract behaviour. The agent then acts in accordance to the policy of the option selected, though at each state along the way, it also samples from the distribution specified by the termination function, $B \sim \mathfrak{B}(\beta_\omega(s))$, with $\mathfrak{B}(x)$ representing the Bernoulli distribution parameterised with x. Eventually, the agent may encounter the event $B = 1$. In such an instance, control returns to π_Ω, and another option is sampled, which is then used to select the next primitive action, beginning the loop anew.

2.2 Imitation Learning

While RL methods are effective at solving sequential decision making problems, they suffer from an important set of drawbacks. Firstly, they can be highly data inefficent due to the problem of exploration—agents must thoroughly explore MDPs in order to find the optimal policy. Perhaps more fundamentally, RL looks to optimise a single scalar reward over sequences of behaviour. While this is a fundamentally flexible and expressive model—from a design perspective—expressing a desired behaviour can be complex, with seemingly simple and intuitive reward functions leading to unexpected or dangerous optimal behaviours. Finally, in the RL framework, it is difficult to naively encode expert biases—if good behaviour for a particular MDP is known, it can be difficult to make use of such expert knowledge. *Imitation Learning* (IL) algorithms provide solutions for sequential decision making problems which mitigate all of these issues. IL algorithms do away with the scalar reward specified by the MDP in favour of a dataset of state-action transitions sampled by an expert actor. Generally, the behaviour of this expert is assumed to be an optimal or near-optimal policy, and the agent in question must learn to mimic the expert.

2.2.1 Formalism of Imitation Learning

In IL problems, an agent is provided with a fixed dataset, D, of (S, A, S')-tuples, which is sampled from an expert policy, π_E, in an MDP. The agent has no access to rewards in the MDP, and only has access to the MDP transition function implicitly via the sample transitions in the dataset. There are two formulations of the IL objective. In the first, the agent must learn a policy, π, which achieves returns close to the expert policy:

$$\mathbb{E}_{\rho_{\pi_E}}[r] - \mathbb{E}_{\rho_\pi}[r] \leq \epsilon,$$

for all possible reward functions r. In the second formulation, the distribution of states observed by the agent must closely match the distribution of states encountered by the expert. The most common notion of divergence applied to such a problem is known as the *total variation distance* and leads to the objective:

$$\|\rho_\pi - \rho_{\pi_E}\|_{TV} \coloneqq \sup_{X \in \mathcal{X}} |\rho_\pi(X) - \rho_{\pi_E}(X)| \leq \epsilon,$$

where $\mathcal{X}$ is the event space over state-action pairs.

These objectives are related, as under mild conditions, bounding the total variation distance leads directly to a bound on the return-based objective (see, e.g. [Ciosek, 2022]).

2.2.2 Behavioural Cloning

The simplest form of imitation learning employs a supervised learning technique in order to mimic expert behaviour. In the behavioural cloning (BC) algorithm, imitation is set up as a multivariate regression problem, where features are represented by states, and our regression model is fitted to the actions taken by the expert in those states. For a parameterised policy π_θ, the objective is then given by:

$$\arg\min_{\theta} \mathbb{E}_{S,A\sim D}\left[\left\|\mathbb{E}_{A'\sim\pi_\theta(S)}[A'] - A\right\|\right].$$

We simply fit the policy to the actions taken in the dataset. However, such a simple approach has a key drawback. At evaluation time, the agent must sample states according to its own behaviour. This means that if our agent encounters a state that is not in the dataset, its behaviour may not be well defined, and it may take actions that lead it to states increasingly far from those that would be encountered by the expert, leading to poor performance. This phenomenon, known as *covariate shift*, is particularly notable in data poor regimes, if our policy parameterisation class cannot fully express the behaviour of the expert, or if the MDP obeys stochastic transition dynamics. This means that few guarantees can be provided for agents trained with BC without heavy assumptions on MDP dynamics or extreme amounts of data. As such, more sophisticated approaches have been developed in recent years which explicitly consider the sequential nature of the problem at hand. These approaches are explored further in Chapter 4, in which we provide a novel algorithm and related analysis for imitation learning.

3
Related Work

In this chapter, we contextualise the contributions of our work, providing the background literature, lines of work, and concurrent developments within which our project sits.

3.1 Theory Of Reinforcement Learning

As we have discussed in the previous chapter, tying RL to its roots in dynamic programming are the *value-based* methods. These approaches seek to approximate value iteration or *policy iteration* [Howard, 1960], leveraging the fact that the *value function*, given by the expected sum of future discounted rewards can be optimised with a greedy policy. These methods employ temporal-difference (TD) learning [Sutton, 1988] in order to learn online, propagating rewards encountered by future states to those earlier in time. One particularly important value-based method is *Q-learning* [Watkins and Dayan, 1992], as described in the previous chapter. Q-learning is what is known as an *off-policy* method, in that the value function learned corresponds to the expected future rewards of the greedy policy, rather than the stochastic "exploration" policy used to gather data. Of additional theoretical value are the *least-squares temporal difference* (LSTD) algorithms, [Bradtke and Barto, 1996, Lagoudakis and Parr, 2003, Ernst et al., 2005], which have superior convergence properties to regular TD methods and can be employed in batch RL

settings (where the agent has access to a limited data set of previous interactions with the environment), at the cost of reduced computational efficiency.

Sometimes, if a stochastic policy is desirable, if optimisation is easier in policy space than in value space, or if action spaces are continuous (preventing efficient maximisation over action-values), *policy gradient* methods Williams [1992], Sutton et al. [2000], Kakade [2002] provide an alternative to the aforementioned techniques. Policy gradient methods directly optimise parameterised policies rather than forming policies implicitly through the value function. They can employ Monte Carlo estimates of future returns, as in REINFORCE [Williams, 1992], or bootstrap via learned value functions [Konda and Tsitsiklis, 2000]. An additional benefit to policy gradient methods, covered in more detail later, is that regularisation can be applied to the policy to integrate prior knowledge about useful policy structure.

3.1.1 Deep Reinforcement Learning

As general function approximators, neural networks are capable of learning their own abstraction—the model itself defines the space in which similar states lead to similar outcomes. Early work in neural network function approximation met limited success, with one particularly notable example being the TD-Gammon backgammon agent [Tesauro, 1995]. While the program was able to compete with other top-tier backgammon programs without specialised features, even with the use of general function approximation, in order to compete with human experts, the features provided to the network were specially engineered to promote meaningful generalisation. More recently, the work of Mnih et al. [2013, 2015] significantly advanced the pursuit of state abstraction in RL by training an agent to play Atari video games with superhuman performance using only the rendered pixels as input. In doing so, the agent must learn its own abstraction, in an end-to-end manner, trained entirely from reward signal. From this work, the field of "Deep RL" developed quickly. Key algorithmic advances include the parallelisation of on-policy actor-critic methods for data decorrelation [Mnih et al., 2016], the use of deep learning to train deterministic policies in environments with continuous action spaces [Lillicrap et al., 2015], trust-region style optimisation [Schulman et al., 2015a, 2017], and a

maximum-entropy objective [Haarnoja et al., 2018]. These advances have led to important accomplishments, including expert-surpassing performance in the games of Go [Silver et al., 2017b] and StarCraft [Vinyals et al., 2019], and dexterous manipulation, solving Rubik's Cube with a robotic hand [Akkaya et al., 2019]. More practical applications have included energy grid management [François-Lavet et al., 2016], and online user experience customisation [Gauci et al., 2018].

Analysis of Target Networks Key to the success in much of deep RL has been the use of target networks [Mnih et al., 2015, Fujimoto et al., 2018, Haarnoja et al., 2018], which make use of a set of lagged value function parameters to stabilise learning. This technique, which we analyse in detail in Chapter 5 has been studied before, though usually involving algorithmic changes or restrictive assumptions. Furthermore, most previous analysis makes use of two-timescale stochastic approximation techniques, which do not provide finite iteration performance bounds, as we provide in Chapter 5.

Yang et al. [2019] show convergence of a Q-learning approach using a target network that is updated using Polyack averaging with nonlinear function approximation. However their analysis—which makes use of two-timescale analysis—requires a projection step to limit the magnitude of parameters. Carvalho et al. [2020] show convergence of a related method using two-timescale analysis, though their target network update differs significantly from those used in practice. Zhang et al. [2021] analyse the use of target networks with linear function approximation, but require projection steps on both the target network and value parameters. Lee and He [2019] provide finite-iteration bounds, but are limited to on-policy data, linear function approximation, and near-perfect fitting to the target network between updates. Fan et al. [2020] analyse the use of target networks for deep Q-learning [Mnih et al., 2015] with the simplifying assumption that they are performing some form of Fitted Q Iteration.

3.1.2 Generalisation in Reinforcement Learning

Early uses of the term *generalisation* in the RL literature almost exclusively refer to the use of function approximation [Ackley and Littman, 1989, Boyan and Moore, 1995,

Sutton, 1996]. While this no doubt implies a form of generalisation, there is a lack of formal or theoretical understanding of what this term means and how to quantify it. As put by Ackley and Littman [1989], while in supervised learning there is a "'learning phase' followed by a 'generalisation test,' in reinforcement learning the search problem is a generalisation test, performed simultaneously with learning".

From this perspective, if generalisation and search are entangled, a large branch of RL theory regarding PAC learnability and regret bounds can be viewed as study in generalisation. Kearns and Singh [2002], provide the first formal sample complexity bounds on the regret of an RL algorithm (called E^3) in general MDPs. The R-max algorithm of Brafman and Tennenholtz [2002] uses an implicit exploration-exploitation tradeoff to extend these results to adversarial environments (such as zero-sum games). Kakade [2003] provides a more natural notion of PAC learning in RL settings, and demonstrates that R-max is PAC-learnable here as well. Strehl and Littman [2008] employ upper confidence bounds to ensure efficient exploration. Auer et al. [2009] introduce the notion of MDP diameter in order to achieve tighter regret bounds.

3.1.3 Performance bounds

While previously mentioned results give formal regret bounds for tabular MDPs, recent focus has been more concerned with direct performance bounds and convergence analysis, with the use of function approximation. Jiang et al. [2017] provide bounds for the broad class of Contextual Decision Processes (which includes MDPs and POMDPs as subcategories), with some restrictions. Dalal et al. [2017] give the first finite time bounds for linear TD(0), under an i.i.d. data model. Bhandari et al. [2018] provide bounds for linear TD in both the i.i.d. data setting and a correlated data setting, through analogy with SGD. Srikant and Ying [2019] approach the problem from the perspective of Ordinary Differential Equations (ODE) analysis, bounding the divergence of a Lyapunov function from the limiting point of the ODE that arises from the TD update scheme.

Fitted policy evaluation, a particular form of TD learning relevant to the analysis in Chapter 5, involves a more sophisticated iteration scheme that has been particularly well

studied with function approximation, due to more theoretically principled updates. Nedić and Bertsekas [2003] analyse the convergence of the Least-Squares Policy Evaluation (LSPE) algorithm of Bertsekas and Ioffe [1996] in an on-policy, linear function approximation setting. Analysis of LSPE shows that learning with constant step size leads to theoretical and empirical gains compared to TD and LSPE with decaying step sizes [Yu and Bertsekas, 2009].

In the context of fitted methods applied to off-policy and control problems, Munos and Szepesvári [2008] prove generalisation properties of Fitted Q Iteration [Ernst et al., 2005] for general function classes under assumptions of low projection error and limited data distribution shift. Le et al. [2019] coin the term fitted policy evaluation (FPE), and formalise the algorithm for general function approximators, with theoretical results under similar assumptions to Munos and Szepesvári [2008].

Inspired by fitted approaches, Gradient TD-style methods [Sutton et al., 2008, 2009, Maei et al., 2009] also lead to convergent, TD-style algorithms, even with off-policy sampling or nonlinear function approximation. These methods look to perform gradient descent directly on the projected Bellman error, and do so by maintaining a second set of parameters—as described in the previous chapter—which must be optimised at a faster timescale than the value parameters. However, these approaches are commonly found to be ineffective and not used in practice due to the difficulty in tuning the rate of second timescale (see, e.g. Fellows et al. [2021]), and potentially additional variance introduced by the second set of parameters [Ghiassian et al., 2020].

3.1.4 Overfitting, Transfer Learning and Deep RL

As RL has matured as a field and moved into more difficult, representationally complex domains, and with the advent of deep RL, the need for a notion of generalisation has become increasingly evident. Deep networks are well understood to be prone to overfitting [Zhang et al., 2016, Goodfellow et al., 2016], and this is likely a problem in deep RL as well, despite nonstationarity. As a result, regularisation of policy networks is applied, often in the form of entropy regularisation, as in Mnih et al. [2016], Schulman et al.

[2017], though more standard forms of L1 and L2 weight decay [Liu et al., 2019], batch normalisation [Ioffe and Szegedy, 2015], and dropout [Srivastava et al., 2014] can be used as well. While Liu et al. [2019] find that it is policy networks that need regularisation most, Farebrother et al. [2018] find that L2 regularisation of the Q-network leads to improvements in policy robustness under domain changes as well. Along similar lines, Igl et al. [2019] propose the use of information bottleneck methods to regularise policies in a multi-task learning setting.

The other common notion of generalisation in RL is concerned with the ability of agents to perform well in environments that they were not optimised for. Early origins of this perspective can be seen in value functions that generalise over many reward functions via the *successor representation* [Dayan, 1993]. Particularly important here is the notion of *environment overfitting*, where the training procedure is specialised to solve one environment, even if the agent is to be deployed in another environment. In the context of RL, this can take the form of fragile hyperparameter settings, and careful architectural choices, factors which—as an aside—harm the reproducibility of the work as well [Henderson et al., 2018]. Whiteson et al. [2011] first outline the issue of environment overfitting for the RL case, proposing the use of a distribution over environments, sampled from some common *generalised environment*. In doing so, we specify the sort of generalisation we expect from our agents, and evaluate accordingly. One successful application of this principle was done by [Packer et al., 2018], who train agents on a distribution of parameterised environments, then assess performance on held-out environments from the same distribution, as well as on environments with out-of-distribution parameters. Zhang et al. [2018a] examine the effects of training on multiple random seeds on overfitting with stochastic agents in continuous spaces. Often, environment stochasticity (such as random initialisation, or randomly repeated actions) are used to encourage robust policies [Machado et al., 2018a], however Zhang et al. [2018b] demonstrate on a procedurally generated testbed that agents are able to "overfit robustly", leading to good performance on training environments, and poor performance on testing ones sampled from the same distribution. Cobbe et al. [2019b] propose procedural generation of environments, sampled explicitly for training and testing sets. They also

introduce the CoinRun environment, which has quickly become a key benchmark for multi-task generalisation in RL. Igl et al. [2020a] find that the non-stationarity of the RL problem is harmful for generalisation performance, and propose an algorithm to combat non-stationarity.

A recent line of work is particularly promising regarding a theoretical characterisation of overfitting and generalisation in deep RL. Set in a batch RL context, François-Lavet et al. [2018], and Francois-Lavet et al. [2019] decompose the suboptimality of the expected return into an asymptotic bias term, and one which depends on the amount and quality of data observed. This decomposition suggests the possibility of a bias-overfitting, or bias-variance tradeoff in batch RL. Particularly of interest here, François-Lavet et al. [2018] suggests hierarchical learning as a potential source of generalisation, though this idea is yet to be further developed.

3.2 Imitation Learning

There exists a wide body of prior work which looks to solve the imitation learning problem. Further complicating matters is the fact that different algorithms employ slightly different problem settings. In the most strict setting, which will be the focus of this thesis, the agent only has access to a fixed dataset, without the ability to further interact with the environment. However many previous developments in IL algorithms make calls to online RL solvers, and focus more on the reward specification problem and the inductive bias provided by expert data.

Behavioural Cloning The simplest and oldest form of imitation learning is known as behavioural cloning, which, as described in Chapter 2 simply optimises policy parameters in order to match the expected state-conditional probability of taking the actions in the expert dataset. However this formulation fails to accommodate for the sequential nature of the MDP, leading to distribution shift [Ross and Bagnell, 2010], and severely increasing the amount of expert data needed to learn a good policy.

Inverse RL Inverse RL (IRL) proposes to evade this issue by learning a reward function which the expert data is (implicitly) maximising, then optimising a policy to solve the original MDP augmented with this reward function. Early formulations of this problem required full solutions to candidate MDPs at each iteration [Ng and Russell, 2000, Ziebart et al., 2008]. These approaches required heavy assumptions on transition dynamics and the form of the reward function of the MDP in order to enable this. More recent *adversarial* IRL methods [Ho and Ermon, 2016] allow for online learning use GAN-style training [Finn et al., 2016] to construct candidate reward functions and policies in order to minimise a divergence between the learned policy and the expert data distribution. However, these methods are known to be difficult to train, and require continuous interaction with the environment, which is not available in our setting, which constrains interaction with the MDP to a finite dataset.

Support Matching In order to avoid the instabilities of adversarial IRL methods, and to limit the number of calls to an RL solver needed, Wang et al. [2019] propose a phased approach, which looks to construct a reward function such that state-actions visited by (or close to those visited by) a deterministic expert are rewarding, while state-actions far from the dataset are not. A reward function is first estimated using only the expert data, either by direct labelling, or an approximate support estimation method. Then, a single call to an RL solver is made, using the rewards supported by the expert. Ciosek [2022] develop theory for support matching methods, providing formal guarantees for reward labelling on discrete MDPs, as well as a heuristic relaxation of the labelling which has been shown to work in continuous environments. Our work builds on these support matching results, but looks to do so in a strictly batch setting, without the need to call an online RL solver.

Offline Imitation Learning Several previous methods employ the setting assumed by our work, looking to learn to mimic an expert policy strictly from data. Zolna et al. [2020] propose an adaptation of adversarial IRL methods for batch settings, though this approach still suffers from training challenges and nonstationarity. More recently, Kim

et al. [2021] establish the use of a hybrid dataset composed of expert and exploratory data, making use of a marginalised density ratio method [Liu et al., 2018, Nachum et al., 2019a] in order to minimise the KL-divergence between the learned policy and expert behaviour. Concurrently, Ma et al. [2022] discuss a paradigm which leverages a more general class of f-divergences in a similar way. Our work builds on the use of both expert and exploratory data in order to train online but makes use of a simple approach more similar to the support methods mentioned above.

3.3 Temporal Abstraction in RL

Historically, temporally extended behaviours were developed to speed up long-term planning [McGovern et al., 1998, Precup et al., 1998, Mann and Mannor, 2014], with origins in traditional A.I. and hierarchical task network planners. In RL, the field of study around abstract actions is known as HRL, due to the fact that abstract actions can be viewed as "higher order" than the so-called "primitive" actions of the base MDP. HRL methods are largely motivated for promoting temporally extended exploration strategies, long-term credit assignment, and interpretable behaviour, though in some multi-task settings, HRL methods have led to generalisation benefits [Frans et al., 2017, Igl et al., 2020b].

3.3.1 Formalisms

There are many frameworks that have been developed to formalise the representation of abstract behaviours RL. Currently, the most popular is the *options framework* [Sutton et al., 1999], which will be our focus here, though we will first provide brief overviews of other forms of hierarchy.

Hierarchies of abstract machines (HAMs) are a highly general framework for representing behavioural abstraction [Parr and Russell, 1998], in that the other formalisms we discuss can be seen as instances of HAMs. HAMs represent higher order policies as finite state machines, which can call finite state machines of one order lower as subroutines.

However, this highly general formulation leads to analytical and algorithmic difficulties when dealing with anything but the most simple HAMs.

The MAXQ framework [Dietterich, 2000b] specifies hierarchical policies as solutions to increasingly refined sub problems, with the overall task being solved by the highest order policy. As such, low-level behaviours are given by a pseudo-reward function, as well as termination predicates, which dictate when a low-level policy should return control to higher ones. MAXQ develops alternate notions of optimality, and employs an additive decomposition of the value function into the value of the subtask with respect to the pseudo-reward, and a separate value of completing a subtask. While specifying the reward function can generally be easier than specifying a full low-level policy, this concision can lead to suboptimal or unexpected behaviour, as the pseudo-reward function can interact with the true reward function in surprising ways. Additionally, MAXQ policies can be difficult to analyse theoretically, as the policies which optimise the pseudo-reward may not be known in advance.

The options framework [Sutton et al., 1999] gives low-level behaviors as policies directly– as a mapping from states to action probabilities, alongside a termination function–which maps from states to the probability that the low-level behaviour returns control to the higher-order policy, and an initiation set, which dictates the set of states that a given subpolicy can be selected in. Often in more recent work, the initiation set of all options is taken to be the entire state space. While in principle, these hierarchies could be arbitrarily deep, generally there is only two levels of policy–the policy over options, and the options themselves. An MDP with a set of options induces a SMDP for the policy over options, a slight modification of the MDP formalism, where, when an action is taken in a state, instead of just specifying a distribution over next states, the environment also specifies a distribution over the durations it may take to reach that next state [Puterman, 2014]. Most of the theory of MDPs applies nearly directly to SMDPs, though care needs to be taken to ensure proper discounting. The combination of low level policies being Markov and the policy over options being semi-Markov enables a powerful form of shared learning

known as *intra option learning* [Sutton et al., 1998], which allows actions taken under one low-level policy to contribute to learning of another.

Work concerned with the theory of options is somewhat more scarce than empirical work. Jong et al. [2008] provide an empirical evaluation of the effectiveness of options with a particular structure in learning as opposed to planning. Brunskill and Li [2014] offer theoretical analysis of HRL, providing a PAC-MDP bound for learning options in a multi-task setting. Jinnai et al. [2019] provide analysis for options learned using graph-theoretical properties. Finally, Nachum et al. [2019c] give empirical results that elucidate the different aspects of HRL that may provide learning benefits.

Other forms of abstraction in tabular settings involves development of state abstractions that perform a reduction from a *base* MDP to an *abstract* MDP, based on the dynamics of the environment in question. These approaches focus on the interplay between state and temporal abstractions by defining equivalence classes between states which exhibit similar behaviour in the MDP. Advances in this line of thought are not isolated to RL, with developments visible in other domains such as motion control via motion primitives [Arikan and Forsyth, 2002, LaValle, 2006], and gridding procedures in formal verification [Esmaeil Zadeh Soudjani and Abate, 2013].

[Li et al., 2006] develop a theory of the criteria which can be used to aggregate states. Dean et al. [1997] presented an early method of state aggregation, making use concept of bisimulation in order to cluster states which exhibit similar transition and reward functions. Alongside the MAXQ framework, which includes specific allowances for state aggregation [Dietterich, 1999], Andre and Russell [2002] discuss explicit forms of hierarchical RL which aggregate states and develop policies to transition between the resulting abstract states. Abel et al. [2016] provide a modern treatment of this line of thinking, providing theoretical bounds on the effectiveness of an abstraction procedure which allows states to only exhibit similar value functions to be considered equivalent.

3.3.2 Learning Behavioural Abstraction

While behavioral abstraction is deeply useful when it is possible to be specified by hand [Precup et al., 1998, McGovern et al., 1998], the ability to discover this sort of structure automatically is seen as integral to the pursuit of HRL [Dietterich, 2000a]. Early work in this space employed heuristics over the nature of transition dynamics to help make exploration easier. Of particular historical importance is the *bottleneck* method [McGovern and Barto, 2001], which defines option policies as those which lead to "bottleneck" states–sparsely connected states that connect more dense regions of state space. These states in particular limit random exploration strategies, potentially because their presence increases hitting times between states on either side of the bottleneck [Jinnai et al., 2019]. Stolle and Precup [2002] use a random task formulation to discover bottleneck states, using Q-learning to create option policies.

Another category of option discovery algorithms leverages the relationship between MDPs and graph networks, using graph theoretical notions to define options. With these methods, the underlying MDP is represented as a graph, with states as nodes, and edges between states that have positive transition probabilities between them. Mannor et al. [2004] uses graph clustering, with options defined as policies that lead from one cluster to another. Şimşek and Barto [2009] use the *betweenness centrality* of the graph, with option policies leading to high betweenness states. Machado et al. [2017, 2018b] use proto-value functions [Mahadevan and Maggioni, 2007], a form of representation for RL based on the graph Laplacian, to define a diffusion model for options. Finally, Jinnai et al. [2019] construct a set of "point options". These options that both initiate and terminate at exactly one state, corresponding to adding a single edge to the graph of possible state transitions. Options are then chosen to minimise an upper bound on the cover time of the graph.

Another common practice involves the construction of options that reach arbitrary goal states. This paradigm is often referred to as *feudal* RL [Dayan and Hinton, 1993]. These options tend to be indexed by continuous sets, rather than the more typical discrete ones.

Option policies are trained to reach goals, either in raw state space [Nachum et al., 2018], or in a learned state space [Vezhnevets et al., 2017, Nachum et al., 2019b].

The most agnostic class of option discovery algorithms seeks to learn option structure end-to-end, directly from rewards. These approaches tend to carefully parameterise policies and leverage a state space augmented with option information, such that the resulting behaviour can be described using the options framework. Levy and Shimkin [2011] pioneered this approach, explicitly constructing a large augmented state and action space to model the learning of options as learning a flat policy. Mankowitz et al. [2016] employ gradient descent to learn a constrained class of options, where only one option can be active in any partition of state space. Bacon et al. [2017] provide a key result, deriving a policy gradient theorem for learning options. Zhang and Whiteson [2019] build on this by representing the option learning process as two augmented MDPs, allowing for off-the-shelf application of state-of-the-art RL methods. Igl et al. [2020b] learn options in a multitask setting using the information theoretic *planning as inference* framework.

An alternative method for learning options end-to-end involves treating the active option as an unobserved latent variable. Since the number of options tends to be small, marginalisation over option indices is possible, and the probability of any given option being active at any given time step can be computed. The benefit to this approach as compared to augmented state space methods is that learning can be shared across options, rather than treating different options in the same state entirely separately in order to maintain Markov state. Daniel et al. [2016] employ linear option policies, allowing them to explicitly optimise options via an expectation-maximisation procedure. Fox et al. [2017] adapt this method to the deep RL setting by alternating between expectation and gradient descent steps, rather than explicit likelihood maximisation. Smith et al. [2018] employ gradient descent through the *forward algorithm* [Baum and Petrie, 1966] to optimise option parameters directly.

Variational Skill Discovery Methods One limitation of the end-to-end approach is that the options learned are often trivial or degenerate, due to the fact that option policies provide no representational benefit in MDPs—this is to say that optimal policies can

always be represented without options [Puterman, 2014]. A novel category of algorithms thus seeks to construct options in an entirely reward-agnostic manner, with options constructed to satisfy information-theoretic criteria. Gregor et al. [2016], Eysenbach et al. [2018] learn options by optimising a variational lower bound on the mutual information between the option index and the states visited by the option policy, leading to a set of options that visit diverse states. Sharma et al. [2020] expand on this idea by using the forward form of the mutual information, and learning a dynamics model. Campos et al. [2020] also use the forward mutual information criteria, but separate the exploration and state sampling problems from the skill discovery problem. Finally Harutyunyan et al. [2019] explore the idea of learning options that optimise for consistent termination conditions, rather than the policy itself.

Transfer Learning Transfer learning, in which an agent is trained on (easy) source tasks with the goal of building on these solution to solve (challenging) target tasks, is a key motivating use case for HRL, as explored in detail in Chapter 6. As such, we provide a brief overview of alternative approaches to transfer learning.

Several previous works, such as Thrun and Schwartz [1995], Pickett and Barto [2002], Frans et al. [2017] use source tasks to learn skills for transfer. However these approaches require that the downstream target tasks can be solved by directly composing the policies learned in the source tasks.

Igl et al. [2020b]—alongside various related approaches [Teh et al., 2017, Galashov et al., 2019, Tirumala et al., 2019]—improve on this framework by making use of KL-regularisation to facilitate learning policies that are similar to those discovered on source tasks. This allows for some flexibility of policies on target tasks, with particular inductive biases introduced by skill hierarchies and choice of regulation form. Termination functions can be learned on the target tasks [Igl et al., 2020b], facilitating composition of policies learned on the source tasks. Outside of the HRL algorithmic framework, Wulfmeier et al. [2019] demonstrate that KL-regularised policies lead to positive transfer for similar sets of tasks.

These previous approaches require that solutions to the target tasks are identical or similar (in a KL-divergence sense) to the solutions to source tasks. In contrast, we will introduce a representation-learning based approach in Chapter 6 which allows skills to effectively cover the entire state space relevant to the tasks at hand, leading to skills that are not solutions to these source tasks, admitting a wider range of target tasks.

4

Imitation Learning via Offline Reinforcement Learning

This chapter builds on the work done in the paper A Strong Baseline for Batch Imitation Learning *[Smith et al., 2023].*

We begin our discussion of generalisation in sequential decision making problems and the complications that arise by studying a setting very close to supervised learning: offline imitation learning (OIL). In OIL problems, we have access to a fixed dataset of expert behaviour and must learn to mimic this expert. The naive approach, known as behavioural cloning, builds a maximum likelihood model of the expert policy over the dataset. However, these models tend not to generalise well to online execution in the environment, as they do not consider the fact that evaluation occurs in this setting, meaning that test samples are not i.i.d. As such, we need to consider more complex methods in order to provide guarantees on the generalisation performance of our learned policy.

In addition to the theoretical challenges posed by the OIL problem, the potential distribution shift incurred at test time leads to challenges in empirical evaluation. In offline RL methods more broadly, hyperparameters are generally tuned via evaluation of policies (that have been trained offline) on the environment. This practice—the equivalent of tuning a

supervised learning algorithm on the test set—may lead to significant information leak. In an extreme case, we can imagine treating all of our model parameters as hyperparameters and then tuning them based on online rollouts of the learned policy in the environment, which would violate the limited data constraints of offline RL methods, and favour methods which overfit or are difficult to tune in practice. However, evaluation of policy quality from offline data is a challenging RL problem in its own right, and any ORL evaluation pipeline needs to accommodate tuning of the policy evaluation algorithm employed. Perhaps surprisingly, far from being standard practice, to our knowledge, no such protocol has ever been described.

To address both of these issues at once, we develop a simple, novel OIL algorithm, that both exhibits strong generalisation guarantees and is easy to tune. Our bounds depend on the size of the MDP, the size of the expert dataset, and the size and quality of the exploratory data-generating policy. As a corollary to our theoretical analysis, we prove a conjecture from Kearns and Singh [1998], which to our knowledge has never been demonstrated formally until now. Alongside this, we develop a novel evaluation protocol, which includes a policy evaluation tuning phase, in order to compare ORL algorithms in a pragmatic and fair manner.

More concretely, our algorithm is motivated by support matching approaches to IL, which look to simplify the IL problem by constructing an intrinsic reward function offline which assigns high reward to policies that match the support of the expert policy in the dataset [Wang et al., 2019, Reddy et al., 2019, Ciosek, 2022]. Using this fixed reward function, a policy can then be learned by a single call to an RL solver. Previous variants of these algorithms require online interaction in order to optimise for the constructed reward. We extend these algorithms and theoretical guarantees for them in tabular environments into the OIL setting.

To do so, we need to eliminate the environmental interaction component of the support matching call to an RL solver. While ORL methods provide a natural solution to this issue, there remain concerns about the dataset that we use. In general, we imagine that the expert dataset may be expensive or difficult to collect, leading to inadequate data

for ORL. Furthermore, expert policies are often deterministic, and may not provide adequate coverage of the MDP in order to learn a robust policy. To address this, recent works in offline IL—such as [Zolna et al., 2020] and [Ma et al., 2022]—have introduced, alongside a small expert dataset, a second, much larger *exploratory* dataset which is used to learn dynamics of the Markov Decision Process (MDP) in areas potentially not covered by the expert agent. These prior works are limited by their complexity: they involve multi-stage learning procedures involving density estimation as well as off-policy RL methods. Furthermore, when they were published, the evaluation of these algorithms made use of hyper-parameter tuning on the environment, and unfair comparisons to simple baselines, inflating their performance and violating the strict data requirements of the OIL setting.

Using our novel evaluation protocol, we evaluate a relaxation of our algorithm on challenging continuous control environments, and find that despite its simplicity, our approach is able to outperform more complex state-of-the-art baselines. Evaluation is conducted in a challenging, data-poor regime in which naive approaches fail. Our results are statistically robust, making use of the cutting edge reporting methods proposed by Agarwal et al. [2021], improving confidence in our results and helping to establish a precedent for more robust benchmarking.

This chapter serves two primary functions in the context of existing literature. First, it extends the theory of support-matching imitation learning methods into the fully offline domain. This is particularly important as support-matching methods are simpler than competing IL approaches, making them more useful when faced with the practical problems—such as high variance and difficult policy evaluation—incurred in the offline RL setting. Second, our policy evaluation protocol serves as an intervention in offline RL methodology, demonstrating a practical and principled approach to the problem of hyperparameter tuning and algorithm selection, as an alternative to current practice, which makes use of information leak by evaluating policies online through interaction with the environment.

In addition to these core contributions, our empirical analysis reveals our simple algorithm to be surprisingly effective when compared with state-of-the-art baselines, marking it as a practical tool for conducting offline IL. Overall, our study finds the proposed algorithm to be both simple and effective, making it a pragmatic baseline for future work on IL in challenging, low expert data regimes.

4.1 Preliminaries and Assumptions

In the theoretical section of this chapter, we consider only tabular MDPs, with small state and action spaces. Furthermore we make the assumption that the reward function, r, is deterministic, and that $0 < r_{\min} < r_{\max} < 1$. This chapter makes use of both discounted and average reward criteria for MDPs. As such, we make use of an important identity, relating the two:

$$\mathbf{R}_{\rho_\pi} = (1-\gamma)\sum_s \rho_\pi(s)V^\pi(s,\gamma),$$

see e.g. Kakade [2001] for a proof. This powerful identity allows us to relate an approximate solution to the discounted MDP to the average reward achieved. We see that the average reward differs from the expected value over the steady state distribution by only a constant factor in $1-\gamma$.

With respect to the imitation learning formalism, in order to accommodate offline learning, we introduce, in addition to the expert dataset, which we denote D_E, we include an exploratory dataset D_X, which is used to learn the structure of the MDP. In general, D_X can be sampled from a random policy, or a policy designed to evenly sample across the state space, so long the policy visits all state-action pairs often enough, an assumption which is formalised below. To do so we introduce the notation $\rho^\pi(s,a) := \rho^\pi(s)\pi(s,a)$, the steady state-action distribution of policy π.

In addition, we introduce assumptions on the structure of the MDP, the expert policy, and our learned policy for simplicity and tractability. The assumptions made in this work are as follows:

Assumption 4.1. *The expert and imitation policies are deterministic.*

Assumption 4.2. *Expert, exploration, and imitation policies induce aperiodic and irreducible Markov chains.*

Assumption 4.3. *The Markov chain induced by the exploratory policy has a uniform lower bound on the probability that the agent will visit any given state and action:*

$$\rho_{\pi_X}(s, a) > p_{\min}.$$

Assumption 4.4. *All eigenvalues of the transition matrix obtained by following policy π^* on the MDP, P_{π^*} are distinct.*

Assumption 4.1 is needed in order for our procedure to remain agnostic to the choice of offline RL algorithm. Since any stationary MDP has a deterministic optimal solution, if the expert behaviour is stochastic, then a batch RL algorithm which yields a deterministic solution will be optimal with respect to our support-matching reward, but may not match the expert behaviour, which includes random actions. This means that a stochastic expert will generate an intrinsic reward function which will admit a deterministic optimal imitation policy. Periodic policies may be compatible with our proof, given some additional bookkeeping, leading to a relaxation of Assumption 4.2. However, the exploratory policy must be aperiodic in order to achieve appropriate sampling over the state space. Similar concerns suggest that Assumption 4.3 cannot be relaxed without introducing other assumptions. Assumption 4.4 is needed only to prove Lemma 4.5, and can, in principle, be relaxed.

In the tabular setting, some notational conveniences are possible. $\rho_\pi(s)$ can be represented by a vector, ρ_π, of length $|\mathcal{S}|$, with each entry giving the steady state probability of the agent occupying that state. Similarly $P^\pi(s'|s)$ can be given by an $|\mathcal{S}| \times |\mathcal{S}|$ matrix, with entry P_{ij} denoting the probability of transitioning from state i to state j in one step under policy π. We denote such a matrix P_π.

The quality of our exploratory policy is quantified via the *mixing time*. This quantity reflects how quickly the exploratory policy reaches the steady-state distribution of the MDP. The mixing time, denoted t^π, is given by the smallest t such that:

$$\max_s \|\mathbf{1}(s)^\top P_\pi^t - \rho^\pi\|_{TV} \leq \frac{1}{4},$$

where $\mathbf{1}$ is a vector with an entry of 1 at the argument and 0 elsewhere.

With our notation defined, assumptions made clear, and relevant quantities defined, we now introduce our algorithm.

4.2 Imitation Learning by Batch RL

Our imitation learning algorithm, Imitation Learning by Batch Reinforcement Learning (ILBRL) is simple. For discrete MDPs, we employ the *intrinsic* reward function given by:

$$\hat{r}(s,a) = \mathbb{I}((s,a) \in D_E),$$

This is the exact reward function used by other support matching approaches, such as Wang et al. [2019] and Ciosek [2022]. These works call on an arbitrary RL solver to perfectly optimise returns with respect to this intrinsic reward, making use of unlimited interaction with the environment. Here, we consider a setting in which we cannot run the algorithm in the environment before test time. This forces us to instead make a call to an *offline* RL algorithm, which in general may not be able to perfectly optimise for our intrinsic reward due to limited data. This significantly complicates analysis, as we need to manage the inefficiency and statistical dependence on finite data that comes from offline RL algorithms, as well as the fact that our call to an RL solver may lead to a suboptimal policy due to the use of finite data. Pseudocode for the overall process is given in Algorithm 1.

Algorithm 1 Imitation Learning by Batch Reinforcement Learning

Input: Expert dataset: D_E, Exploration dataset: D_X
Output: Imitation policy: π

for $s, a \in D_U$ **do**
    $\hat{r}(s,a) \leftarrow \mathbb{I}((s,a) \in D_E)$
$\pi \leftarrow \text{BatchRL}(D_U, \hat{r})$
return π

In general, our algorithm admits any batch RL algorithm which is able to learn a near-optimal policy with high probability. In order to simplify analysis, for the theory section of this chapter, we make use of an off-policy RL algorithm, developed by Kearns and

Singh [1998], known as *phased Q-Learning*. The algorithm is somewhat impractical, but its analysis is tractable. We develop favourable asymptotical guarantees on its sample complexity in subsequent sections. Later, for empirical evaluation, we will substitute phased Q-learning for a more modern offline RL algorithm, known as TD3-BC [Fujimoto and Gu, 2021].

4.3 Theory of ILBRL

This section establishes finite sample complexity bounds for ILBRLwith phased Q-learning in tabular MDPs. To do so we first reiterate and expand on the proofs provided by Kearns and Singh [1998], formalising their conjecture for agent-based sampling of data. We then join these results with the proofs of Ciosek [2022], which are concerned with the amount of expert data needed to build an effective intrinsic reward function. Together, these results give bounds on the amount of expert and exploratory data needed to ensure high probability learning.

4.3.1 Theory of Phased Q-Learning

Phased Q-learning can be seen as a sample-based form of value iteration. In this algorithm, rather than sampling states and actions individually, each action at each state across the entire environment is sampled simultaneously m times. This data generation process is known as *parallel sampling*. Parallel sampling can be simulated using standard RL sampling based on rollouts of a policy in the MDP, which we characterise theoretically later in this section. It allows us to update the estimated state-action value at each state and action simultaneously according to the optimal Bellman equation using the deterministic reward and *empirical bootstrap*, using the following iteration:

$$Q_{i+1}(s,a) = r(s,a) + \gamma \frac{1}{m} \sum_{k=1}^{m} \max_{a'} Q_i(s_k, a'),$$

where s_k is the kth sample subsequent state from taking action a in state s.

Since the empirical bootstrap will concentrate around the true value, we expect that, with enough data, this is exactly performing value iteration. Here we prove that this is an

efficient offline RL algorithm. First, we need to demonstrate that the amount of data needed to achieve bounded error at the end of the evaluation process is polynomial in the appropriate quantities, which was accomplished in the original work by Kearns and Singh [1998]. Second, we need to show that we can simulate sampling from the parallel model efficiently. Kearns and Singh [1998] postulate that such a procedure should be possible under assumptions concerning the quality of the exploration policy, but a formal proof has never been provided until now.

We use the notation from Kearns and Singh [1998]:

- m is the number of samples from the parallel sampler,
- ℓ is the number of iterations of phased Q-learning,
- $\hat{V}_i(\cdot), \hat{Q}_i(\cdot,\cdot)$ are our estimated value functions at iteration i,
- $\bar{V}_i(\cdot), \bar{Q}_i(\cdot,\cdot)$ are the true value functions of the policy greedy with respect to $\hat{Q}_i$,
- $V_i(\cdot), Q_i(\cdot,\cdot)$ is the output of i rounds of policy iteration,
- $V^*(\cdot), Q^*(\cdot,\cdot)$ are the optimal value functions.

Following the intuition that phased Q-learning is similar to value iteration, we make use of the following inequality—that our estimate of the bootstrapped value concentrates around the true bootstrap:

Claim 1. *For all s, a, and $i \leq \ell$, we have*

$$\left| \frac{1}{m} \sum_{k=1}^{m} \hat{V}_i(s'_k) - \sum_{s'} P(s'|s,a)\hat{V}_i(s') \right| \leq \eta', \tag{4.1}$$

where s'_k is the kth sampled subsequent state from the parallel sampler at s, a.

We will later take m large enough, in combination with Hoeffding's inequality, to demonstrate that Claim 1 will hold with high probability. This is done in the proof of Lemma 4.3.

With this assumption, we are able to formalise the notion that performing phased Q-learning is very close to performing true value iteration, as suggested by the following lemma.

Lemma 4.1. *Given Eq.* (4.1)*, the difference between the value output by phased Q-learning after ℓ steps and the optimal value is given by*

$$\zeta(s,a) := \left|\hat{Q}_\ell(s,a) - Q^*(s,a)\right| \leq \frac{\eta'\gamma + \gamma^\ell}{1-\gamma} \qquad \forall s, a.$$

Proof. We begin by bounding the gap between value estimated by phased-Q iteration and the value output by policy iteration. Recall the phased Q-learning update, given by

$$\hat{Q}_{i+1}(s,a) = r(s,a) + \gamma \frac{1}{m} \sum_{k=1}^{m} \hat{V}_i(s'_k).$$

For any s, a pair, this means we have

$$\begin{aligned}
\left|\hat{Q}_{i+1}(s,a) - Q_{i+1}(s,a)\right| &= \left|r(s,a) + \gamma \frac{1}{m} \sum_{k=1}^{m} \hat{V}_i(s'_k) - r(s,a) - \gamma \sum_{s'} P(s'|s,a) V_i(s')\right| \\
&\leq \gamma \left| \sum_{s'} P(s'|s,a) \left(\hat{V}_i(s') - V_i(s')\right) \right| + \gamma\eta' \\
&\leq \gamma \max_{s'} \left|\hat{V}_i(s') - V_i(s')\right| + \gamma\eta' \\
&\leq \gamma \max_{s'} \left|\max_{a'} \hat{Q}_i(s',a') - \max_{a'} Q_i(s',a')\right| + \gamma\eta' \\
&\leq \gamma \max_{s',a'} \left|\hat{Q}_i(s',a') - Q_i(s',a')\right| + \gamma\eta',
\end{aligned}$$

where the first inequality makes use of our claim in (4.1), the second uses the fact that we have a convex combination of the value gap over states, and the last from the max of a difference being greater than a difference of max. Starting from $Q_0(s,a) = \hat{Q}_0(s,a)$ and recursively applying this equation leads to the bound

$$\left|\hat{Q}_\ell(s,a) - Q_\ell(s,a)\right| \leq \frac{\eta'\gamma}{1-\gamma}, \tag{4.2}$$

by the geometric series and the fact that both algorithms are initialised identically.

Puterman [2014, Thm 6.3.3] shows that $|Q_\ell(s,a) - Q^*(s,a)| \leq \gamma^\ell/(1-\gamma)$. Combining with the above yields

$$\left|\hat{Q}_\ell(s,a) - Q^*(s,a)\right| \leq \frac{\eta'\gamma + \gamma^\ell}{1-\gamma}. \qquad \square$$

Lemma 4.1 suggests that the the value function learned by phased Q-learning differs from the optimal value function by at most a statistical error term which shrinks as m grows, and an iteration term, which shrinks as ℓ increases.

Given that our learned value function is close to the optimal value function, we now look to bound the overall suboptimality of the corresponding policy learned by phased Q-learning. We define the discounted regret of the policy at step i on the discounted MDP as the expectation of the difference between the discounted return achieved by the learned policy and that of the optimal policy, with expectation taken over the initial state distribution:

$$\varrho_\gamma(i) := \mathbb{E}_{s_0}\left[\bar{V}^*(s_0) - \bar{V}_i(s_0)\right].$$

Equipped with the pointwise value gap, we can bound the overall regret with the following lemma.

Lemma 4.2. *Given Eq.* (4.1)*, the policy output by phased Q-learning after l steps achieves regret of at most:*

$$\varrho(\ell) \leq \frac{2}{(1-\gamma)}\left(\frac{\eta'\gamma + \gamma^l}{1-\gamma}\right).$$

This means that our overall regret is simply a constant factor of the error we incur in approximation of the optimal value function.

Proof. We begin by bounding the statewise suboptimality of the learned policy using the suboptimality of a single action. Let $\pi_\ell(s) := \text{argmax}_a \hat{Q}_\ell(s,a)$ give the policy output after ℓ steps of phased Q-learning. We abuse notation slightly here by writing $a := \pi_\ell(s)$, since context is clear. Then we have

$$\begin{aligned}
\left|\bar{V}_\ell(s) - V^*(s)\right| &= \left|r(s,a) + \gamma \sum_{s'} P(s'|s,a)\bar{V}_\ell(s') - V^*(s)\right|, \\
&= \left|\gamma \sum_{s'} P(s'|s,a)\left[\bar{V}_\ell(s') - V^*(s')\right] + Q^*(s,a) - V^*(s)\right|, \\
&\leq \gamma\left|\max_{s'}\left[\bar{V}_\ell(s') - V^*(s')\right]\right| + \left|Q^*(s,a) - V^*(s)\right|.
\end{aligned}$$

Unrolling this gives rise to a geometric series in γ:

$$|\bar{V}_\ell(s) - V^*(s)| \leq \frac{1}{1-\gamma}\max_s |Q^*(s,\pi_\ell(s)) - V^*(s)|. \tag{4.3}$$

Let $\pi^*(s) := \arg\max_a Q^*(s,a)$ be the optimal policy. We bound this gap for $a = \pi_\ell(s) \neq \pi^*(s) = a^*$, since otherwise it is zero. Since, by construction, $\hat{Q}_\ell(s,a^*) \leq \hat{Q}_\ell(s,a)$, the

gap is maximised when we are both underestimating the optimal value of the optimal action and overestimating the optimal value of a suboptimal action:

$$\begin{aligned}\left|V^*(s) - Q^*(s,a)\right| &= Q^*(s,a^*) - Q^*(s,a), \\ &= Q^*(s,a^*) - \hat{Q}_\ell(s,a^*) + \hat{Q}_\ell(s,a^*) - \hat{Q}_\ell(s,a) + \hat{Q}_\ell(s,a) - Q^*(s,a), \\ &\leq |Q^*(s,a^*) - \hat{Q}_\ell(s,a^*)| + \hat{Q}_\ell(s,a^*) - \hat{Q}_\ell(s,a) + |\hat{Q}_\ell(s,a) - Q^*(s,a)|, \\ &\leq 2\left(\frac{\eta'\gamma + \gamma^\ell}{1-\gamma}\right) + \underbrace{\hat{Q}_\ell(s,a^*) - \hat{Q}_\ell(s,a)}_{\leq 0},\end{aligned}$$

where the final line is an application of Lemma 4.1. Since the both this and Eq. (4.3) apply across all states, combining this with Eq. (4.3) gives the bound on our regret. □

Since we now have a bound on the overall regret of our algorithm, we can bound this by η, our overall error margin for this phase of learning, allowing us to solve for the relevant quantities. Once we have solved for the minimum acceptable concentration error η', we can solve for m, the number of samples we need for Eq. (4.1) to hold with high probability. This is accomplished in the following lemma.

Lemma 4.3. *Taking $m\ell$ samples from the parallel model for phased Q-learning, such that*

$$m\ell \geq \log\left(\frac{2\ell SA}{\delta'}\right)\frac{2\gamma^2\ell}{(1-\gamma)^2(\eta(1-\gamma)^2 - 2\gamma^l)^2},$$

we obtain discounted regret less than η with probability at least $1-\delta'$.

The proof involves bounding by an overall acceptable error η and solving for $m\ell$ using Hoeffding's inequality. It suggests that high probability polynomial learning is possible. While this looks like a recursive bound in ℓ, we note that the constant factor on both sides allows us to set ℓ first in practice, then choose m. We have chosen this form in order to demonstrate the overall sample complexity bound.

Proof. To start, we recall Hoeffding's concentration inequality.

Lemma (Hoeffding's Inequality)**.** *Let X_j be a sequence of i.i.d. random variables uniformly bounded by $0 \leq X_j \leq a$. Define the sample mean of the sequence of length m as $\bar{X}_m := \frac{1}{m}\sum_{j=1}^{m} X_j$. Then we have*

$$P\left(\left|\bar{X}_m - \mathbb{E}[X_1]\right| \geq t\right) \leq 2\exp\left(-2\frac{t^2 m}{a^2}\right).$$

In order to bound the overall error by η, we choose the phased Q-learning concentration error as

$$\eta' = \left[\eta(1-\gamma)^2/2 - \gamma^\ell\right]/\gamma,$$

with the additional condition that:

$$\ell \geq \frac{\log\eta + 2\log(1-\gamma) - \log 2}{\log\gamma}, \tag{4.4}$$

to ensure that η' is not negative.

Substituting our choice of η' into Lemma 4.2, we see that regret obeys

$$\varrho_\gamma(\ell) \leq \frac{2}{(1-\gamma)}\left(\frac{\eta'\gamma + \gamma^\ell}{1-\gamma}\right) = \frac{2}{(1-\gamma)}\left(\frac{\frac{\eta(1-\gamma)^2/2-\gamma^\ell}{\gamma}\gamma + \gamma^\ell}{1-\gamma}\right) = \eta.$$

With our error bounded, we proceed to bound the probability with which we fail to achieve this error, which will lead to our overall bound on m. We define the concentration error as

$$\Delta_i(s,a) := \left|\frac{1}{m}\sum_{k=1}^{m}\hat{V}_i(s'_k) - \sum_{s'}P(s'|s,a)\hat{V}_i(s')\right|.$$

Next, we apply Hoeffding's Inequality for each s, a, i, substituting $X_j = \hat{V}_i(s'_k)$ and $t = \eta'$. At step i, conditioned on s and a, $\hat{V}_i$ is a deterministic function with range $[0, (1-\gamma)^{-1}]$ and the s'_k are independent draws from the MDP transition function at s, a, so Hoeffding's inequality can be used, and we obtain

$$P\left(\Delta_i(s,a) \geq \eta'\right) \leq 2\exp\left\{-\frac{2m\left([\eta(1-\gamma)^2/2 - \gamma^\ell]/\gamma\right)^2}{(1-\gamma)^{-2}}\right\}.$$

We take a union bound over state, action, and algorithm step, since the overall bound needs to hold for all of the above. Then, bounding the resulting failure probability by δ' gives

$$2SA\ell \exp\left\{-\frac{2m\left([\eta(1-\gamma)^2/2 - \gamma^\ell]/\gamma\right)^2}{(1-\gamma)^{-2}}\right\} \leq \delta'.$$

Solving for m leads to

$$m \geq \log\left(\frac{2\ell SA}{\delta'}\right) \frac{2\gamma^2}{(1-\gamma)^2(\eta(1-\gamma)^2 - 2\gamma^l)^2}$$

and the claim follows by multiplying both sides by ℓ. □

Now we only need concern ourselves with the gathering of these m independent samples from the parallel model. We do so by appealing to a classic result applying to Markov chains, which suggests that sampling the current state and action after following the chain for several steps is close to sampling from the stationary distribution, allowing for gathering of independent samples.

Lemma 4.4. *The number of sample transitions from the exploration policy needed to perfectly simulate the parallel sampling model with failure probability less than δ'' is upper bounded by*

$$\frac{2t^{\pi_X}}{\log(2)p_{\min}} \log\left(\frac{2}{p_{\min}}\right) \log\left(\frac{SA}{\delta''}\right).$$

This shows that we can ensure perfect parallel sampling with high probability using a number of samples subquadratic in $t^{\pi_X}, 1/p_{\min}, 1/\delta''$ and SA.

Proof. Let $\rho^T_{\pi_X}$ be the distribution after rolling out the exploratory policy for T steps from any starting distribution. By Levin and Peres [2017, Sec. 4.5], we know that $T \geq t^{\pi_X} \log(1/\tau)/\log(2)$ implies $\|\rho^T_{\pi_X} - \rho_{\pi_X}\|_{TV} \leq \tau$. We choose

$$T \geq t^{\pi_X} \log(2/p_{\min})/\log(2),$$

such that $\tau \leq p_{\min}/2$. Assume that we collect a sample every T steps. After collecting N samples, the probability that a given state-action pair is never observed is bounded from

above by $(1 - p_{\min} + \tau)^N \leq \exp(-Np_{\min}/2)$. Bounding this probability by δ'', taking a union bound over all state-action pairs and solving for N yields

$$N \geq 2/p_{\min} \log(SA/\delta'').$$

The claim is obtained by multiplying N and T. □

Lemma 4.4 combined with Lemma 4.3 can then be used to give the overall sample complexity of the phased-Q learning stage of our algorithm.

4.3.2 Bounding Total Variation Distance Between Expert and Imitator

With a good sense for the data requirements of our offline RL algorithm, we can proceed to ensure that we have enough expert data to craft a good reward function for ILBRL to succeed. We do so by invoking a key lemma of Ciosek [2022]. However, this lemma applies only to the average reward setting, while our results thus far concern policies optimised for discounted reward. As such we must first bound the suboptimality of the learned policy (under the discounted criterion) with respect to the average value attained. We achieve this with a relaxation of Theorem 1 from Kakade [2001]. Recall that $\mathbf{R}_{\rho_\pi}$ represents the average reward of policy π.

Lemma 4.5 (Relaxation of Theorem 1 from Kakade [2001])**.** *Suppose policy π is ϵ-suboptimal or better at each state with respect to the discounted return. Let π^* be the optimal policy under the average reward criterion. Taking Σ to be the matrix of right eigenvectors of P_{π^*} with corresponding eigenvalues: $\lambda_1 = 1 > ... \geq |\lambda_n|$ Then, under Assumption 4.4, we have*

$$\mathbf{R}_{\rho_\pi} \geq \mathbf{R}_{\rho_{\pi^*}} - \kappa(\Sigma)\|\mathbf{r}\|\frac{1-\gamma}{1-\gamma|\lambda_2|} - (1-\gamma)\epsilon,$$

where $\kappa(\Sigma) := \|\Sigma\|_2\|\Sigma^{-1}\|_2$ is the condition number of a matrix, and $\mathbf{r}$ is the vector with entries $r(s, \pi^*(s))$ for each state.

Proof. Our proof closely follows that of Kakade [2001]. Let π^{γ^*} be the optimal policy under the discounted return criterion. Then we have, for all s

$$V^{\pi^{\gamma^*}}(s) \geq V^{\pi^*}(s).$$

Subtracting ϵ from both sides gives

$$V^{\pi}(s) \geq V^{\pi^{\gamma^*}}(s) - \epsilon \geq V^{\pi^*}(s) - \epsilon,$$

for all states, which follows from the assumption that π is ϵ-suboptimal or better. From here we make use of the relationship between average reward and discounted reward:

$$\begin{aligned}
\mathbf{R}_{\rho_\pi} &= (1-\gamma)\sum_s \rho_\pi(s)V^\pi(s), \\
&\geq (1-\gamma)\sum_s \rho_\pi(s)(V^{\pi^*}(s) - \epsilon), \\
&\geq (1-\gamma)\sum_s \rho_\pi(s)(V^{\pi^*}(s)) - (1-\gamma)\epsilon, \\
&\geq \mathbf{R}_{\rho_{\pi^*}} - \kappa(\Sigma)\|\mathbf{r}\|\frac{1-\gamma}{1-\gamma|\lambda_2|} - (1-\gamma)\epsilon.
\end{aligned}$$

The second inequality comes from the fact that ρ_π sums to one across states. The final inequality follows exactly from the proof in Kakade [2001]. □

Lemma 4.5 gives us a lower bound on the suboptimality with respect to the average reward of a policy ϵ-suboptimal with respect to the discounted reward. Next, we need to ensure that it is possible to achieve a policy which performs well with respect to the intrinsic algorithm reward. This is accomplished by invoking Lemma 5 of Ciosek [2022], reproduced here for convenience.

Lemma 4.6 (Lemma 5, Ciosek [2022])**.** *Given an expert dataset consisting of $|D_E|$ points, a policy, π which maximises the average intrinsic reward achieves an expected average intrinsic reward, $\mathbf{R}^{\text{int}}_{\rho_\pi}$, of at least*

$$\mathbf{R}^{\text{int}}_{\rho_\pi} = 1 - \nu - \sqrt{\frac{8St^{\pi_E}}{|D_E|}},$$

for all error terms $\nu > 0$, with probability at least

$$1 - \delta''' := 1 - 2\exp\left(-\frac{\nu^2|D_E|}{4.5t^{\pi_E}}\right).$$

To connect high performance on the intrinsic reward with the performance on the extrinsic reward, we make use of another key lemma of Ciosek [2022]:

Lemma 4.7 (Lemma 7, Ciosek [2022])**.** *An agent which achieves an average reward of* $1-\epsilon$ *on the intrinsic reward MDP also achieves average reward of:*

$$(1-\epsilon)\mathbf{R}_{\rho_{\pi_E}} - 4t^{\pi_E}\epsilon$$

on the true MDP.

We are now ready to conclude, evaluating the overall sample complexity of our algorithm.

Theorem 4.8. *Consider an instance of Algorithm 1. Under assumptions 4.1 to 4.4, in order to achieve regret on the average reward problem of less than* ϵ*, or total variation distance between the expert and the imitation policy less than* ϵ*, with probability* $1-\delta$*, we need to sample*

$$\max\left\{\frac{1}{\epsilon^2}32St^{\pi_E}(1+4t^{\pi_E})^2, \frac{4.5t^{\pi_E}16(1+4t^{\pi_E})^2\log(4/\delta)}{\epsilon^2}\right\}$$

transitions from the expert policy, and

$$\mathcal{O}\left(\frac{t^{\pi_X}}{p_{\min}}\log\left(\frac{1}{p_{\min}}\right)\log^2\left(\frac{SA}{\delta}\right)\log\left(\frac{t^{\pi_E}\beta}{\epsilon(1-\lambda)}\right)\frac{(t^{\pi_E})^8\beta^6}{\epsilon^8(1-\lambda)^6}\right)$$

from the exploratory policy, where $\beta = \kappa(\Sigma)\|\mathbf{r}\|$*, as described in Lemma 4.5.*

Proof. For reference, we reproduce the relevant error variables and their interpretations here.

Error Variable	Interpretation
ϵ	Output error of Algorithm 1
$1-\epsilon'$	Average intrinsic reward obtained by the learned policy
ϵ''	Reward construction error
$1-\nu$	Probability controllable intrinsic reward of expert policy.
η	Discounted regret for phased Q-learning
η'	Concentration error of sampled values

Table 4.1: Error Variables

Let

$$1 - \epsilon' := \mathbb{E}_{s,a\sim\rho_\pi}[\hat{r}(s,a)] = \mathbf{R}^{\text{int}}_{\rho_\pi} \tag{4.5}$$

represent the average reward achieved by the policy learned on the intrinsic reward problem. We begin by directly evoking Lemma 4.7. This leads to our overall regret on the true problem, given by

$$\begin{aligned}\mathbf{R}_{\rho_{\pi_E}} - \mathbf{R}_{\rho_\pi} &\leq \epsilon' \mathbf{R}_{\rho_{\pi_E}} + 4t^{\pi_E}\epsilon', \\ &\leq \epsilon' \left(1 + 4t^{\pi_E}\right).\end{aligned}$$

We then wish to bound the number of samples needed to achieve ϵ' small enough such that the overall error obeys

$$\epsilon' \left(1 + 4t^{\pi_E}\right) \leq \epsilon, \tag{4.6}$$

with probability δ.

Lemma 4.5 gives us the means to decompose ϵ' in terms of the error incurred by batch learning and construction of the intrinsic reward problem in our algorithm. Let the policy learned by phased Q-learning be η-suboptimal with respect to the discounted intrinsic return across all states. Lemma 4.5 allows us to move to the average reward setting, by bounding the distance between the average intrinsic reward obtained by our learned policy and the optimal intrinsic average reward using this quantity. However the optimal intrinsic average reward itself is a function of the amount of data we have. Lemma 4.6 bounds this additional error incurred from a potential lack of expert data. Specifically, we define the reward construction error as

$$\epsilon'' := \nu + \sqrt{\frac{8St^{\pi_E}}{|D_E|}}.$$

From Lemma 4.6, we have $\mathbf{R}^{\text{int}}_{\rho_{\pi^*}} \geq 1 - \epsilon''$, with some probability $1 - \delta'''$, which we will return to bound, once ϵ'' has been bounded. Substituting $\mathbf{R}^{\text{int}}_{\rho_{\pi^*}} = 1 - \epsilon''$ and $\epsilon = \eta$ in Lemma 4.5 we have:

$$\mathbf{R}^{\text{int}}_{\rho_\pi} \geq 1 - \nu - \sqrt{\frac{8St^{\pi_E}}{|D_E|}} - \kappa(\Sigma)\|\mathbf{r}\|\frac{1-\gamma}{1-\gamma|\lambda_2|} - (1-\gamma)\eta.$$

From the definition of ϵ' in Eq. (4.5), this leads to the following bound

$$\epsilon' \leq (1-\gamma)\frac{\kappa(\Sigma)\|\mathbf{r}\|}{1-\gamma|\lambda_2|} + (1-\gamma)\eta + \nu + \sqrt{\frac{8St^{\pi_E}}{|D_E|}},$$

which we set to

$$\underbrace{(1-\gamma)\frac{\kappa(\Sigma)\|\mathbf{r}\|}{1-\gamma|\lambda_2|}}_{\text{T1}} + \underbrace{(1-\gamma)\eta}_{\text{T2}} + \underbrace{\nu}_{\text{T3}} + \underbrace{\sqrt{\frac{8St^{\pi_E}}{|D_E|}}}_{\text{T4}} \leq \frac{\epsilon}{(1+4t^{\pi_E})},$$

in order to satisfy Eq. (4.6).

Since there are four terms here—which we refer to as $\text{T1}-\text{T4}$—with largely independent algorithmic parameters, we allow error to be distributed evenly across all the terms, which allows us to bound the above expression termwise.

In order to control the first term, we need to set γ large enough to satisfy

$$\text{T1} \leq \frac{1}{4}\frac{\epsilon}{(1+4t^{\pi_E})}. \tag{4.7}$$

For convenience we define $\alpha := 4(1+4t^{\pi_E})/\epsilon$ and $\beta := \kappa(\Sigma)\|\mathbf{r}\|$. This leads us to

$$\frac{(1-\gamma)\beta}{1-\gamma|\lambda_2|} \leq \frac{1}{\alpha}. \tag{4.8}$$

We set

$$\gamma = \frac{\alpha\beta - 1}{\alpha\beta - |\lambda_2|}, \tag{4.9}$$

or, equivalently

$$(1-\gamma) = \frac{1-|\lambda_2|}{\alpha\beta - |\lambda_2|}.$$

in order to satisfy Eq. (4.8). Setting γ in this way thus allows us to minimise the error incurred by transferring from the discounted to the average reward setting, by increasing the effective horizon of the discounted problem.

Moving on to the second term, directly choose

$$(1-\gamma)\eta = \frac{1}{\alpha},$$

to satisfy T2 $\leq 1/\alpha$. We leave the error in this form, since η is multiplied by a factor of $1-\gamma$ in Lemma 4.3. Intuitively, this corresponds with the scaling of stepwise rewards in the discounted problem by a factor of $1/(1-\gamma)$.

Continuing to the terms corresponding to the definition of our intrinsic reward problem, we choose

$$\nu = \frac{1}{\alpha},$$

to satisfy T3 $\leq 1/\alpha$. Finally, we choose

$$|D_E| = \left\lceil \frac{1}{\epsilon^2} 128 S t^{\pi_E} (1 + 4t^{\pi_E})^2 \right\rceil, \tag{4.10}$$

where $\lceil \cdot \rceil$ is the ceiling function which chooses the smallest integer greater than the argument. This allows us to satisfy

$$\text{T4} = \sqrt{\frac{8St^{\pi_E}}{|D_E|}} \leq \frac{1}{\alpha}.$$

This provides us with our first bound on the number of expert samples, and completes the bounding of overall error by the relevant factors. To complete our proof, we must ensure that these errors hold with overall probability δ.

With the error of our algorithm controlled, we move on to bounding the failure probability of our algorithm. Let δ''' be the maximum probability of the expert policy failing to achieve an intrinsic average reward of $1 - \epsilon''$ as in Lemma 4.6. From Lemma 4.6, substituting $1/\alpha$ for ν, we have

$$\delta''' \leq 2 \exp\left(- \frac{\left(\frac{\epsilon}{4(1+4t^{\pi_E})}\right)^2 |D_E|}{4.5 t^{\pi_E}} \right).$$

Setting this to be less than $\delta/2$ and rearranging leads to the bound

$$|D_E| \geq \frac{72 t^{\pi_E} (1 + 4t^{\pi_E})^2 \log(4/\delta)}{\epsilon^2}. \tag{4.11}$$

This is our second bound on $|D_E|$. Taking the maximum over Eq. (4.10) and Eq. (4.11) gives our overall bound on the number of expert samples needed.

To bound the failure probability of phased Q-learning, we first bound the probability that our simulation of the parallel sampler fails. Invoking Lemma 4.4 with the substitution $\delta'' = \delta/4$, we get

$$N \geq \frac{2t^{\pi_X}}{\log(2)p_{\min}} \log\left(\frac{2}{p_{\min}}\right) \log\left(\frac{SA}{\delta/4}\right). \tag{4.12}$$

With our remaining error budget, we can look to bound the probability that phased Q learning fails. Substituting $1/\alpha$ for $(1-\gamma)\eta$ and $\delta' = \delta/4$ in Lemma 4.3 gives

$$m\ell \geq \log\left(\frac{2\ell SA}{\delta/4}\right) \frac{2\gamma^2\ell}{(1-\gamma)^2(\frac{1}{\alpha}(1-\gamma) - 2\gamma^\ell)^2}. \tag{4.13}$$

To complete the proof, we recall from Eq. (4.4) that

$$\ell \geq \log_\gamma\left(\frac{\frac{1}{\alpha}(1-\gamma)}{2}\right). \tag{4.14}$$

Together, Eqs. (4.9) and (4.11) to (4.14) give the sample complexity needed to bound the failure probability of our procedure, given by $\delta' + \delta'' + \delta'''$, below the global error δ, with the overall number of expert samples given by the maximum over Eq. (4.10) and Eq. (4.11), and the number of exploratory samples given by $|D_X| \geq N\ell m$.

While the above gives the most precise version of our bound, it is difficult to interpret the effect of changing ϵ, δ, and the parameters of the environment, largely due to the obscuring influence of γ on m and ℓ.

To fully simplify, we will choose sensible values for ℓ and γ, which will lead to a bound with a simpler form. Since $\gamma < 1$, $\log_\gamma$ is a decreasing function, so we choose

$$\ell = \log_\gamma\left(\frac{\frac{1}{\alpha}(1-\gamma)}{4}\right).$$

Substituting this for ℓ in Eq. (4.13) leads to the following bound on m:

$$\begin{aligned} m &\geq \log\left(\frac{2SA\log_\gamma\left(\frac{\frac{1}{\alpha}(1-\gamma)}{4}\right)}{\delta/4}\right) \frac{2\gamma^2}{(1-\gamma)^2\left(\frac{(1-\gamma)}{\alpha} - \frac{(1-\gamma)}{2\alpha}\right)^2} \\ &\geq \log\left(\frac{2SA\log_\gamma\left(\frac{\frac{1}{\alpha}(1-\gamma)}{4}\right)}{\delta/4}\right) \left(\frac{8\gamma^2\alpha^2}{(1-\gamma)^4}\right). \end{aligned} \tag{4.15}$$

To further simplify, we choose a sensible value for γ. Noticing that our bound on γ based on Eq. (4.8) is increasing in $\alpha\beta$, instead of Eq. (4.9), we choose

$$\gamma = \frac{2\alpha\beta - 1}{2\alpha\beta - |\lambda_2|}.$$

Which, since $2\alpha\beta \geq 1$ and $0 \leq |\lambda_2| \leq 1$, remains in $[0, 1)$, and satisfies Eq. (4.8). Substituting this into our choice of ℓ and simplifying leads to

$$\ell = \frac{\log\left(\frac{\epsilon^2(1-|\lambda_2|)}{4(1+4t^{\pi_E})(8(1+4t^{\pi_E})\beta-|\lambda_2|\epsilon)}\right)}{\log\left(\frac{8(1+4t^{\pi_E})\beta-\epsilon}{8(1+4t^{\pi_E})\beta-|\lambda_2|\epsilon}\right)}.$$

If we divide ℓ by

$$l = \left(\frac{4(1+4t^{\pi_E})\beta}{\epsilon(1-|\lambda_2|)}\right)^2,$$

and take limits, through use of a limit solver, we find that multiple applications of l'Hopital's Rule give

$$\lim_{\alpha\to\infty,\beta\to\infty,\lambda\to 1} \frac{\ell}{l} = 0,$$

which implies that:

$$\ell = \mathcal{O}\left(\left(\frac{4(1+4t^{\pi_E})\beta}{\epsilon(1-|\lambda_2|)}\right)^2\right).$$

All that is left then is to bound the second factor in Eq. (4.15). Substituting our values of γ, we have

$$\begin{aligned}\left(\frac{8\gamma^2\alpha^2}{(1-\gamma)^4}\right) &= 8\alpha^2\frac{(2\alpha\beta-1)^2(2\alpha\beta-|\lambda_2|)^2}{(1-|\lambda_2|)^4} \\ &\leq 8\alpha^2\frac{(2\alpha\beta)^4}{(1-|\lambda_2|)^4},\end{aligned}$$

where the second inequality follows from the fact that $2\alpha\beta \geq 1$. Bringing this together with our first term, this gives an overall bound for m as

$$m = \mathcal{O}\left(\log\left(\frac{SA\left(\frac{t^{\pi_E}\beta}{\epsilon(1-|\lambda|)}\right)}{\delta}\right)\frac{(t^{\pi_E})^6\beta^4}{\epsilon^6(1-\lambda)^4}.\right)$$

Multiplying with our bounds for ℓ and N, we get the overall sample complexity of the exploratory dataset

$$|D_X| = Nm\ell = \mathcal{O}\left(\frac{t^{\pi_X}}{p_{\min}}\log\left(\frac{1}{p_{\min}}\right)\log^2\left(\frac{SA}{\delta}\right)\log\left(\frac{t^{\pi_E}\beta}{\epsilon(1-\lambda)}\right)\frac{(t^{\pi_E})^8\beta^6}{\epsilon^8(1-\lambda)^6}\right).$$

The bound on total variation follows from the well-known identity

$$\mathbb{E}_{\rho_1}[f] - \mathbb{E}_{\rho_2}[f] \leq \epsilon \implies \|\rho_1 - \rho_2\|_{TV} \leq \epsilon,$$

for two discrete distributions, ρ_1, ρ_2 and a function f with range $[0, 1]$. See e.g. Ciosek [2022] for a proof. Substituting ρ_{π_E} for ρ_1, ρ_π for ρ_2, with $f = r$, gives

$$\mathbf{R}_{\rho_{\pi_E}} - \mathbf{R}_{\rho_\pi} \leq \epsilon \implies \|\rho_{\pi_E} - \rho_\pi\|_{TV} \leq \epsilon. \tag{4.16}$$

Choosing $|D_E|$ and $|D_X|$ sufficiently high as described above leads to the left side holding true as discussed. This implies that with the same number of samples, the total variation distance between expert and imitator is less than ϵ, with probability at least $1 - \delta$. □

We note that the bound on the amount of expert data needed differs from that of Ciosek [2022] only in terms of a constant factor. This is promising, insofar as it suggests that the amount of expert data needed is not significantly larger than when online learning is permitted, in order to specify an effective reward function. The burden of learning offline then falls only on the size and quality of the exploratory dataset and the size of the MDP. In practice, this exploratory data may be simpler to collect, since we need only to ensure good coverage of the MDP rather than that the best actions are taken.

While our bound is quite large in terms of $\frac{1}{\epsilon}$ compared to related bounds (such as Ma et al. [2022]), this largely is due to the setting we consider. Since we are concerned with matching the steady-state distribution of the expert rather than a discounted reward objective, the effective horizon of the problem becomes infinite, and compounding errors in the learned behaviour can no longer be dampened by the discount factor. Indeed we can see that the fact that $\frac{1}{1-\gamma}$ must be linear in ϵ in order to satisfy Eq. (4.9) leads to the extra $\mathcal{O}(\frac{1}{\epsilon^6})$ later in our proof. The use of an average-reward algorithm in place of phased Q-learning may lead to a tighter bound in ϵ, as we will not have to switch between discounted and average reward settings, but the convergence properties of average-reward algorithms require more strict conditions, and may necessitate the use of additional assumptions.

4.4 Evaluation Of Offline RL Algorithms

Equipped with theoretically sound generalisation bounds, we turn to the empirical component of this chapter, which is concerned with formulation of a reasonable and principled practice for evaluating generalisation performance of offline RL methods, as well as the practical generalisation performance of a relaxed variant of our algorithm in modern benchmark environments. We begin by developing our novel evaluation protocol for ORL algorithm comparison.

Due to the absence of an i.i.d. test set in sequential decision making problems, it is unclear how to evaluate their generalisation properties, especially when working from a fixed batch dataset. The current convention is to train on batch data, and tune hyperparameters on the true environment. However this gives batch algorithms access to conceivably unlimited data: in an extreme case we can imagine that all agent parameters are taken to be hyperparameters, and a "batch" algorithm is trained, though all optimisation happens on data from the true environment. Even more alarmingly, in the imitation learning setting this leads to an information leak, insofar as that the algorithm implicitly makes use of environment reward for parameter tuning. This makes it difficult to empirically compare the effectiveness of different batch RL algorithms, as some may be highly prone to overfitting to particular datasets or environments, but we will not observe this since they are able to be tuned on the true environment. Furthermore, the fact that the amount of data used in tuning on the true environment is both unbounded—and in common practice unreported—makes it difficult to evaluate generalisability and ease of tuning across different algorithms.

These issues have been noted recently by Paine et al. [2020], who suggest using a validation set and an independent offline policy evaluation (OPE) procedure on the policy output by the offline RL algorithm. Hyperparameters are optimised in accordance with the estimated values of the initial states in the dataset. However, little attention is given to the fact that the OPE is itself a challenging and provably difficult problem [Wang et al., 2021a], requiring hyper-parameter tuning of its own. In order to address these issues, we develop a novel evaluation protocol which enables tuning of OPE independently of the

algorithm being evaluated. This enables fair comparison of batch RL and IL algorithms, with clear data requirements, and reduced information leak.

Data Partitioning In order to prevent overfitting to our limited dataset and biasing our performance estimates, we first partition our dataset into a training set and evaluation set. The policy evaluation set is then further partitioned into an *evaluation training set*, and an *evaluation validation set*, preventing overfitting of the OPE algorithm to the evaluation data.

Tuning OPE In order to tune the OPE algorithm, at least two deterministic policies of different known quality are needed. In addition, we require that the dataset be composed of transitions which include the actions taken at the subsequent state. In this work we employ Expected SARSA [Van Seijen et al., 2009], though in principle, other methods for OPE can be used. Expected SARSA (ESARSA) is then tuned on the data with the known, "safe" policies to ensure convergence to the values of the known policies, as well as good separation of policy values. Since data is generated by multiple policies, one key hyperparameter of the policy evaluation process is the composition of the evaluation dataset—we need to find a reasonable data distribution, such that state distribution induced by our target policies are not too far from the distribution of the dataset, a key factor which leads to divergence. These hyperparameters are then used to evaluate the policies learned via offline RL. While in general this may not lead to a convergent OPE algorithm for all possible learned policies, our procedure provides a consistent evaluation protocol without the use of new data from the environment, enabling principled selection of policies and grounding of the learned values in the environment at hand.

Policy Selection Each batch RL method is trained across several hyperparameter configurations and several random seeds. Using the evaluation hyperparameters tuned on the known safe policies, evaluation of a single hyperparameter and random seed setting for the batch algorithm is then performed across several evaluation seeds in order to estimate the true value of the learned policy. Policy evaluation is run for a fixed number of steps, determined by the amount of time it took for evaluation to converge on the

known policies. Performance of a control hyperparameter setting is then given by the mean output of the learned value function on a held-out dataset consisting of initial states. The policy which achieves the highest mean convergent value across seeds and rollouts is then chosen for evaluation on the true environment, which forms the results that are reported.

Imitation Learning This chapter is largely concerned with the imitation earning setting, while the evaluation protocol developed here is meant to apply to evaluation of offline RL more broadly. We underscore that our algorithm needs no access to true rewards, but in order to implement our protocol for the purpose of evaluation, knowledge of true rewards is needed for the transitions in the dataset. These rewards are used for policy evaluation only: importantly they are not needed for ILBRL.

4.4.1 Pseudocode

Algorithm 2 represents an abstract procedure by which offline RL practitioners can tune and evaluate agents that are trained on offline data, without access to the true environment. The protocol integrates an all-important phase in which the offline policy evaluation (OPE) algorithm itself is tuned, a practice which we find to be highly necessary, and absent from existing literature altogether.

We view an offline RL algorithm, $\mathbb{A}$, as a (stochastic) function which maps from a dataset, $D \in \mathcal{D}$, and hyperparameter configuration, $\psi \in \Psi$, to a policy, $\pi \in \Pi$. Our protocol is then used to optimise over a discrete set of candidate hyperparameter configurations, returning those that lead to the highest expected performance on held-out data. Here we assume that datasets are indexed by trajectory in the first dimension, then state, though a shuffled dataset can be used, so long as initial and terminal states are labeled appropriately.

An OPE algorithm, $\mathbb{A}'$, then maps from datasets and policies, as well as OPE hyperparameters, $\vartheta \in \Theta$, to value functions, $V \in \mathcal{V}$. Our criterion for selection of OPE hyperparameters is referred to as *RankError* (RE_D), which takes a set of value functions

learned by OPE, V_k, each of which evaluates a different policy, π_k, and compares how the ranking implied by V_k compares to sample Monte Carlo estimates of V^{π_k}:

$$\text{RE}_D(\{V_k\}, \{V^{\pi_k}\}) = \sum_k \left|\text{Rank}\left[V_k(D)\right] - \text{Rank}\left[V^{\pi_k}(D)\right]\right|.$$

In our case, rankings are performed over value functions that are averaged over several random seeds. In the case where OPE diverged, the corresponding term was set to the maximum, $|\{\pi_k\}|$, with other rankings adjusted favourably to minimise RE. Ties were broken based on the total distance to the true values $\sum_k |V_k(D) - V^{\pi_k}(D)|$.

For the purposes of data partitioning in Algorithm 2, $D_{0:i}$ refers to subset of D with indices between 0 and i, $D_{l:|D|,0}$ is the subset of D with indices between l and $|D|$ intersected with the subset of D corresponding to start states.

4.4.2 Tuning Data Composition for Policy Evaluation

In our experiments, we found that tuning the policy evaluation hyperparameters played a vital role in ensuring good policy selection. Indeed, for many hyperparameter settings, policy evaluation diverged altogether. In our setting, since the data was derived from multiple policies, one key parameter was the relative mix of different sources on which to evaluate the policy. This is illustrated in Figure 4.1.

4.4.3 Algorithmic Benefits of Tuning Protocol

We note that an additional algorithmic benefit of this protocol is that, due to the instability of policy evaluation under distribution shift, our policy evaluation step represents a form of implicit regularisation on the final policy selected for deployment. Policies that lead to instability in evaluation are likely to diverge heavily from the distribution induced by the dataset, while policies that stay close to the data will lead to more stable evaluation. Since the limitation of distribution shift seems to be fundamental to current practice in batch RL, we propose that this is beneficial, especially in the context of imitation learning, in which we are looking to achieve a policy similar to the one used to generate the data.

Algorithm 2 Offline Hyperparameter Tuning for RL

Input:

Data: Dataset D, Training set fraction d,

Offline RL: Algorithm $\mathbb{A} : \mathcal{D} \times \Psi \to \Pi$, Hyperparameters $\{\psi_n\}$, random seeds $\{x_s\}$

OPE: Algorithm $\mathbb{A}' : \mathcal{D} \times \Pi \times \Theta \to \mathcal{V}$, Training fraction d', Known policies $\{\pi_k\}$, Hyperparameters $\{\vartheta_j\}$

Output: Optimal hyperparameters, ψ^*, Performance of optimised algorithm $R_{\mathbb{A}(D,\psi^*)}$

Phase 1: Data Splitting

$i \leftarrow \text{int}(|D| * d)$

$D_T \leftarrow D_{0:i}$ # Assign training set

$D_V \leftarrow D_{i:|D|}$ # Assign validation set

$l \leftarrow \text{int}(|D_V| * d')$

$D_{V_{PE}} \leftarrow D_{V\,0:l}$ # Assign Policy Evaluation Training Set

$D_{V_F} \leftarrow D_{V\,l:|D_V|,0}$ # Assign Final Performance Validation Set (initial states only)

Phase 2: Offline Policy Evaluation Tuning

for $\vartheta_j \in \{\vartheta_j\}$ **do**

  for $\pi_k \in \{\pi_k\}$ **do**

    $V_{jk} \leftarrow \mathbb{A}'(D_{V_{PE}}, \pi_k, \vartheta_j)$ # Evaluate known policy on PE training set

$\vartheta^* \leftarrow \arg\min_j \text{RE}_{D_{V_F}}(\{V_{jk}\}, \{V^{\pi_k}\})$ # Choose evaluation hparams from initial states

Phase 3: Offline RL Training

for $\psi_n \in \{\psi_n\}$ **do**

  for $x_s \in \{x_s\}$ **do**

    $\pi_{ns} \leftarrow \mathbb{A}(D_T, \psi_n)$ # Train policy via offline RL for several seeds

Phase 4: Hyperparameter Selection

for $\pi_{ns} \in \{\pi_{ns}\}$ **do**

  $V_{ns} \leftarrow \mathbb{A}'(D_{V_{PE}}, \pi_{ns}, \vartheta^*)$ # Evaluate policy with optimal PE hyperparameters

$\psi^* \leftarrow \arg\max_n \sum_s V_{ns}(D_{V_F})$ # Optimal hyperparameters maximise value of initial states

$R_{\mathbb{A}(D,\psi^*)} \leftarrow \text{EvalEnv}(\pi_{\psi^*})$ # Evaluate optimal hyperparameters averaged across seeds

return $\left(\psi^*, R_{\mathbb{A}(D,\psi^*)}\right)$

4.5 Continuous Control Task Experiments

In many realistic RL settings, tabular function approximation is not possible, due to large or continuous state and action spaces. Complimenting our theoretical guarantees provided in Section 4.3, we investigate a modification of our proposed algorithm on state-of-the-art benchmarks, making use of contemporary deep RL techniques. We apply the evaluation protocol developed in Section 4.4, in order to examine the practical efficacy of our algorithm framework as compared to other contemporary approaches. We find that, despite (or perhaps due to) its simplicity, ILBRL outperforms baseline approaches under

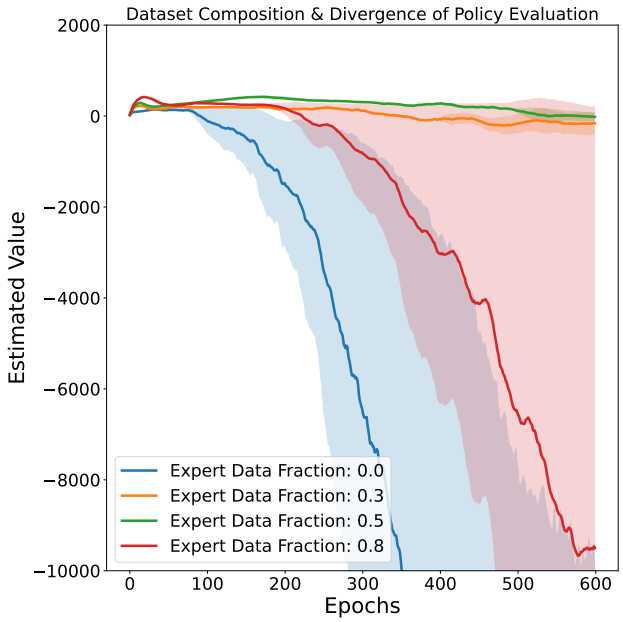


Figure 4.1: Policy Evaluation of a single policy on varying dataset compositions. We see that for more evenly mixed data, evaluation is more stable, while for less balanced data, divergence occurrs. This highlights the importance of tuning the policy evaluation protocol itself.

our fair model selection protocol.

4.5.1 Environment

The Gym MuJoCo benchmark [Brockman et al., 2016] is a standard set of challenging RL environments which cover such continuous control tasks. These locomotion tasks involve learning control policies for several robots with varying morphologies in a simulated physical environment. The agents, which are restricted to movement in a two dimensional plane, are optimised for forward velocity and maintaining their centre of mass in a prespecified vertical range (representing the agent maintaining an upright pose). Specifically, we make use of the `Hopper-v2`, `Halfcheetah-v2`, `Walker2d-v2`, and `Ant-v2` environments, as D4RL datasets [Fu et al., 2020] (described below) are available for these settings. We use `gym` version `0.24.1`, and MuJoCo version `2.1`. Additional details regarding reward function specifics, as well as state and action spaces can be found in the documentation at https://github.com/openai/gym. A discount factor of $\gamma = 0.99$ was used across all experiments.

4.5.2 Datasets

In our experiments, data comes from the D4RL MuJoCo benchmarks [Fu et al., 2020], which are standard continuous control datasets used to benchmark state-of-the-art offline RL and IL methods. The D4RL data consists of several datasets, generated by policies of varying quality. The "expert" data represents rollouts from a stochastic agent trained on the true environment online using SAC [Haarnoja et al., 2018]. This policy is taken to be optimal in the environment. We use a fraction of the D4RL expert data as our D_E. In order to simulate a risk-averse offline setting in which data has been gathered from known, safe exploratory policies for training and evaluation, our exploratory data, D_X, is derived from the D4RL "medium" dataset, which comes from a suboptimal, intermediate policy trained on the true environment. For imitation learning algorithms, we mask the true reward, normally available in the dataset, save for the evaluation protocol described above. Specifically, we use the version of D4RL available at the commit: https://github.com/rail-berkeley/d4rl/commit/d842aa194b416e564e54b0730d9f934e3e32f854.

Dataset Composition A naive behavioural cloning (BC) baseline using only the expert data was able to achieve an expert-level policy for several environments when 100 expert trajectories were used, as is done in Ma et al. [2022], Zolna et al. [2020]. We instead opted to use only 18 expert trajectories, which we found to be the maximum number such that the BC baseline began to perform suboptimally. This amount of data was chosen in order to highlight the effectiveness of our algorithm, as if there is enough expert data for BC to perform effectively, in general there is likely little benefit to using more sophisticated IL algorithms. Our total training dataset size was then taken to be $6 \cdot 10^5$ transitions, in order to leave enough data to be used for policy evaluation. Since each expert trajectory was approximately 1000 timesteps long, this means that the training datasets were composed of only 3% expert data. For policy evaluation, as described above, the dataset composition was tuned as a hyperparameter. We found that, from values we tried, a $50/50$ split was optimal. The overall size of the policy evaluation set was $7 \cdot 10^5$ transitions, with $5 \cdot 10^5$ used for training, and $2 \cdot 10^5$ reserved for validation, of which only 208 were the initial states ultimately used for validation in our experiments.

4.5.3 Adapting ILBRL for Continuous Control Tasks

In order to extend our algorithm for use in the MuJoCo, environments—which use both continuous state and action spaces—we apply a modified version of the continuous support reward relaxation of Ciosek [2022]. We employ the normalised reward:

$$r(s,a) \coloneqq \max_{s',a' \in D_E} 1 - \frac{1}{d_{\text{max}}^{\frac{1}{2}}} \left\| \phi(s,a) - \phi(s',a') \right\|_2^{\frac{1}{2}},$$

where $d_{\text{max}} \coloneqq \max_{s,a \in D_U} \min_{s',a' \in D_E} \left\| \phi(s,a) - \phi(s',a') \right\|_2$ is the maximum distance observed in the dataset. The use of normalisation ensures that rewards are bounded in $[0,1]$ and helps control instability that arises in our context due to the need for offline deep RL methods, which generally lack convergence guarantees. We can think of this as a "soft" support estimator which assumes some smoothness over the environment and agent policy: state-action pairs that are in the expert dataset will have reward 1, just as in the tabular algorithm. However, in continuous environments it is unlikely that the same state will be visited more than once, so we also reward state-action pairs that are close to the expert data.

Our agent learns via a state-of-the-art offline RL method known as TD3-BC [Fujimoto and Gu, 2021]. The algorithm is a modification of the algorithm developed in Fujimoto et al. [2018], with an additional BC regularisation term applied to the policy loss. This shifts the distribution induced by the learned policy to be closer to the data distribution and leads to more stable learning. We employ a small modification to the TD3-BC algorithm by multiplying the statewise BC loss by the intrinsic reward achieved in those states. This leads the agent to put more emphasis on imitating expert behaviour, and less weight on the imitation of the exploratory policy.

4.5.4 Baselines

We compare to two state-of-the-art offline IL algorithms: SMODICE [Ma et al., 2022] and ORIL [Zolna et al., 2020]. In addition, we compare to an expert-only BC baseline, which makes use exclusively of D_E. This stands in contrast to the dataset used for BC by Ma et al. [2022], which contains large amounts of expert data, but large amounts of random

data as well, which confounds the BC learner. In some of our experiments, we found that giving the BC learner as much data as is reported in Ma et al. [2022] leads to near optimal performance for some environments. We also provide a sample-based estimate of the D4RL policies, evaluated by rolling out the provided models on the environment several times. We use the argmax variant of the policies, which act according to the mean of their Gaussian policy. To implement our baselines, we made use of lightly modified versions of implementations made available in association with Ma et al. [2022], which can be found at: https://github.com/JasonMa2016/SMODICE

4.6 Implementation Details

Architectures All policies and value functions across our method and the baselines shared a common architecture, taken from Ma et al. [2022], but established as standard in several works prior, such as Zolna et al. [2020]. Policies and values were parameterised as multilayer perceptrons [Goodfellow et al., 2016]. Two hidden layers were used, each of size `256`. Rectified Linear Units (ReLU) [Goodfellow et al., 2016] were used as nonlinearities between layers. For policies, since MuJoCo environments have symmetrically bounded action spaces, the outputs of these networks were then passed through an elementwise `tanh` function scaled to match the environment bounds. The discriminator network used for SMODICE [Ma et al., 2022] and ORIL [Zolna et al., 2020] baselines was an MLP of the same size as the policy and value networks, though here `tanh` nonlinearities were used between layers.

Algorithm Implementation Details Both baselines and ILBRL were implemented using the PyTorch scientific computing library (https://pytorch.org/), version `1.12.0`. Experiments were run on Google Cloud Compute `n1-standard-64` VM instances with `Intel Haswell` CPUs and the `Debian 10` operating system. GPU computing was not used for our experiments.

As was suggested in Fujimoto and Gu [2021], and used in Ma et al. [2022], we employed dataset standardisation. Sample means, μ_{D_T}, and standard deviations, σ_{D_T}, were

estimated for observations, o, across the training sets, and all data passed through the networks, s, was transformed according to $s = (o - \mu_{D_T})/\sigma_{D_T}$.

The Adam [Kingma and Ba, 2014] optimiser was used for optimisation across all learned functions. Parameters were kept at the defaults used by PyTorch version `1.12.0`, except for the learning rate (gain) which was tuned as a hyperparameter. The baselines made use of a reward scaling parameter, which was also tuned as a hyperparameter.

For ILBRL, TD3-BC was selected as our offline RL algorithm as it introduces only one additional hyperparameter as compared to an online RL algorithm, alongside the fact that it learns deterministic policies. We found that this combination made TD3-BC ideal for our setting, which demanded simplicity especially due to our multi-stage evaluation protocol described above. TD3-BC can be summarised as the policy update scheme:

$$\theta' = \theta + \alpha \nabla_\theta \left(\lambda Q(s, \pi_\theta s) + \text{BC}(\pi_\theta(s), a) \right),$$

where s, a are the state and action in the dataset, α is a learning rate in $[0, 1]$, and the Q function is learned separately through Double-Q Learning [Hasselt, 2010] on the dataset, with clipped zero-mean noise added to the policy when performing the bootstrapped update. The BC term is simply the mean-squared-error between the action taken by the policy and the action in the dataset. We introduced one change to the TD3-BC algorithm, which was a re-weighting of the statewise behavioural cloning loss:

$$\text{BC}_{\text{ILBRL}}(s, a') := \text{BC}(\pi(s), a)\hat{r}(s, a),$$

where r_I is our intrinsic reward function. This re-weighting encourages imitation of the expert policy, and reduces the degree to which the exploration policy is copied. We found this led to higher performance based on our offline evaluation protocol. An ablation study can be seen below in Figure 4.4.

Off Policy Evaluation Implementation Details All methods were evaluated via Algorithm 2 combined with the Expected SARSA [Van Seijen et al., 2009] off-policy evaluation algorithm implemented using target networks [Mnih et al., 2013]. Stochastic

policies were evaluated according to the policy $\pi'(s,a) = \mathbb{E}\left[\pi(s,a)\right]$, which is common practice for Gaussian-parameterised policies such as the ones employed here [Haarnoja et al., 2018]. The expected SARSA bootstrapped update for deterministic policies with target networks is then given by:

$$\theta' = \theta + \nabla_\theta \left[Q_\theta(s,a) - r(s,a) - \gamma Q_{\bar{\theta}}\Big(s', \pi(s')\Big)\right]^2,$$

with target networks updated using the exponential moving average:

$$\bar{\theta}' = (1-\tau)\bar{\theta} + \tau\theta$$

In order to make policy evaluation more robust, updates were performed according to the mean bootstrap values of two separately parameterised value and target networks. In the equations above this corresponds to:

$$\theta_i' = \theta_i + \nabla_{\theta_i}\left(Q_{\theta_i}(s,a) - r(s,a) - \frac{1}{2}\gamma\left[Q_{\bar{\theta}_1}\Big(s', \pi(s')\Big) + Q_{\bar{\theta}_2}\Big(s', \pi(s')\Big)\right]\right)^2,$$

for $i \in \{1,2\}$, with target networks updated as before in accordance with their corresponding value network.

4.6.1 Reported Results

Each offline RL algorithm was evaluated across 10 hyperparameter configurations, with 6 random *policy seeds* per hyperparameter. Policy evaluation was then performed for each of the random seed-hyperparameter combinations. Since policy evaluation is itself a stochastic algorithm, evaluation of each configuration was tested across 3 *evaluation seeds*. Amongst evaluations that converged, the hyperparameter configuration that was then chosen was the one that had the highest learned value on a held-out dataset consisting only of environment start states.

The performance of the optimal configuration was evaluated for each of the *policy seeds* as the mean return of of 30 rollouts of 1000 steps on the true environment, averaged across 10 different random *environment seeds*. These means were taken to represent the true performance of the learned policy.

This gives 6 *policy performance* data points per algorithm, per environment (one for each policy seed). To aggregate this data, we follow the recommendations of Agarwal et al. [2021]. We perform a stratified bootstrap, sampling 6 points with replacement from the policy performance pool for each environment. To compute the IQM of a bootstrap, we discard samples in the bottom or top 25th percentile of the 6 samples in each environment. The mean is then taken over the remaining data: across environments for our confidence interval plots, and per environment for the performance profile.

In order to compute confidence intervals, we repeat this process 10 000 times, taking the lower bound of the interval to be the data point at the 2.5th percentile of IQM samples, and the upper bound to be the data point at the 97.5th percentile. This corresponds to the *percentile bootstrap assumption*

The environmentwise results in Table 4.2 were computed as the mean and 95% CI scores of the three policy seeds per environment, using the more conventional *normal assumption*.

4.6.2 Results

Algorithm	Hopper	Halfcheetah	Walker2d	Ant
ILBRL	$\mathbf{64.634} \pm 8.417$	$\mathbf{49.856} \pm 1.229$	$\mathbf{80.681} \pm 6.154$	$\mathbf{104.907} \pm 1.109$
BC	$\mathbf{69.459} \pm 15.005$	4.884 ± 0.338	27.04 ± 5.937	21.666 ± 5.17
SMODICE	53.228 ± 2.508	42.662 ± 0.422	$\mathbf{75.942} \pm 0.794$	90.05 ± 1.436
ORIL	5.7 ± 5.214	$\mathbf{55.079} \pm 4.207$	25.268 ± 14.239	35.81 ± 2.296
(Exploration)	55.62	42.686	67.371	88.333
(Expert)	111.016	94.371	107.747	121.639

Table 4.2: Mean performances on environment during evaluation for three training seeds, averaged over 30 rollouts. Error measures are given by a 95% confidence interval based on the mean performance per training seed.

The final performance results of our algorithm, evaluated on the environment after optimisation using our principled evaluation procedure can be found in Fig. 4.2 and Fig. 4.3. Our metrics were chosen based on the recommendations by Agarwal et al. [2021], for improved ease of comparison, and increased statistical robustness. Performance was measured according to the normalised D4RL score, which assigns random performance a score of 0, and expert performance a score of 100. Fig. 4.2 represents 95% confidence

intervals (CI) for the interquartile mean (IQM) score of bootstrapped samples uniformly chosen across three rollouts per algorithm and all tasks. The IQM interpolates between the median and the mean statistic. It is estimated by first computing the mean normalised score of the middle 50% of scores in a given bootstrap. Then, the reported overall means and CIs are computed over the entire set of bootstrap means. Agarwal et al. [2021] find that bootstrapped CIs are more statistically robust than the median, and less susceptible to outliers than the mean. Bootstrapping across tasks serves two purposes: it gives an aggregate notion of performance in order to efficiently compare algorithms, and allows for more statistically robust point estimates by making use of all available data at once.

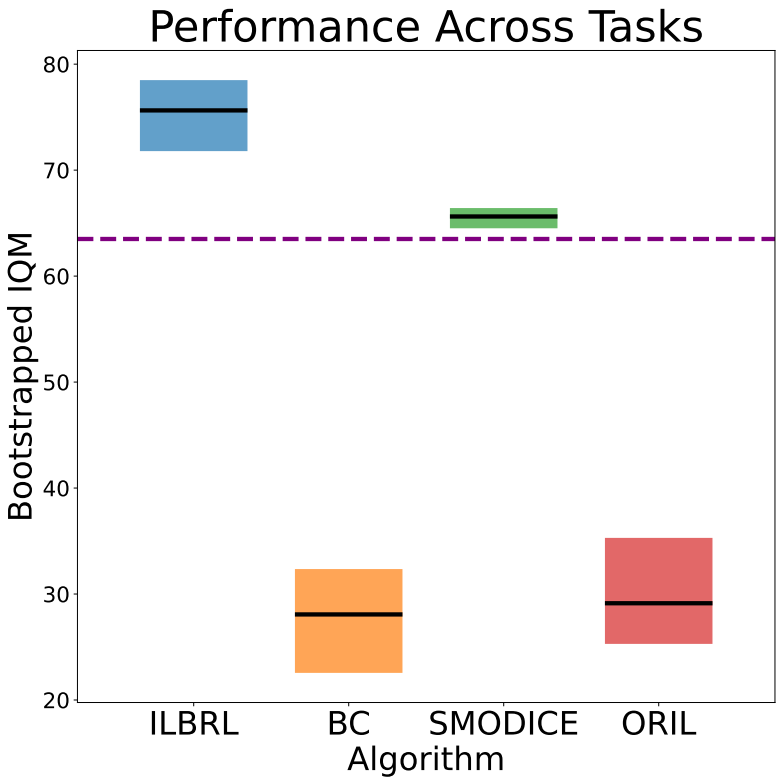


Figure 4.2: Aggregate Performance of ILBRL and baselines across D4RL tasks. Horizontal bars represent interquartile means. Shaded regions give 95% confidence intervals based on bootstrapped samples [Agarwal et al., 2021]. The purple, dashed line gives the average performance of the exploratory policy. The expert policy achieves an expected score of 100.

We note that among the algorithms we tested, only ILBRL is able to outperform the exploratory policy by a statistically significant margin. This suggests that it is able to make good use of the expert data labels and is not simply cloning the exploratory policy. Because we are in a challenging, low expert data regime, performance of the baselines

is somewhat lower than the values reported in the original publications, which make use of large amounts of expert, or near-expert data. We hypothesise that our relative effectiveness is due to the simplicity of our approach, as well as the natural scaling of our method to imbalanced datasets, while the baselines rely on density estimation, which may be challenging under highly imbalanced data regimes.

In addition to our more robust approaches to evaluation, we report complete evaluation results for all tests in Table 4.2. Our results carry over: our algorithm outperforms or competes with the baselines on all of the environments.

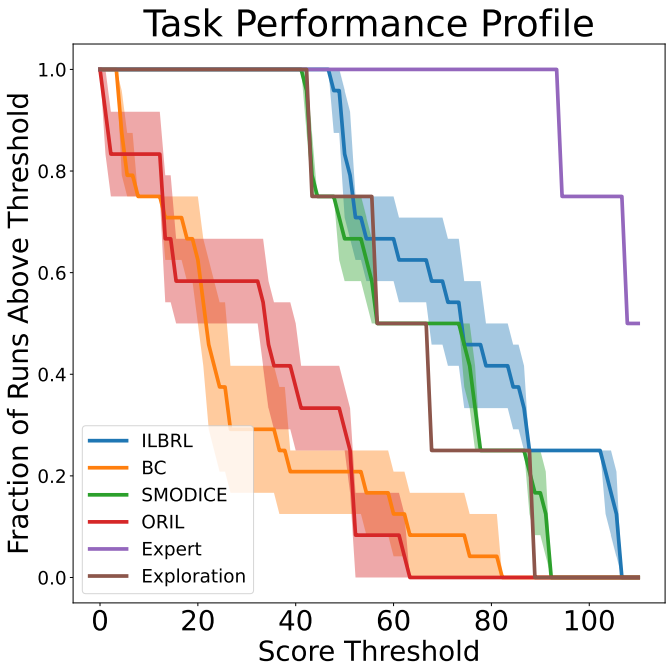


Figure 4.3: Performance profiles for ILBRL and baselines [Agarwal et al., 2021]. Curves that arc further from the origin are higher performing. ILBRL is competitive with state-of-the-art IL algorithms in difficult data regimes. It is able to achieve more tasks at a lower thresholds and reach higher performances when compared to the baselines.

For a more qualitative evaluation of our method, we make use of a performance profile, as shown in Fig. 4.3, also suggested by Agarwal et al. [2021]. On the x axis is a performance threshold, and on the y axis is the fraction of runs across environments that achieved the level of performance specified on the x axis. We see that our method is able to compete with expert performance on some tasks, as can be observed by the x-intercept reaching past a threshold of 100. Our algorithm strictly dominates the ORIL baseline, and reaches

well beyond medium performance, showing that it is able to both perform very well on easier tasks, and maintain a reasonable performance for more challenging tasks.

In addition we performed an ablation study on our modified reward function. We found it to improve performance, though not across all tasks. A performance profile and IQM bootstrap CI can be found in Fig. 4.4. In addition, despite the fact that gains are not extreme, our policy evaluation procedure is able to select for the reward-weighted ILBRL over the ablation using only offline data, further evidencing the robustness and effectiveness of our evaluation procedure.

4.6.3 Hyperparameters

This section details hyperparameters and tuning ranges to enable reproducibility of the experiments in question.

Tuning Ranges

All hyperparameters were tuned using random search. In order to evaluate fairly against the baselines, given the novel evaluation protocol, we retuned baseline hyperparameters, with ranges near the defaults from https://github.com/JasonMa2016/SMODICE. For ILBRL, hyperparameters that were not tuned were taken to be defaults from the TD3-BC implementation at https://github.com/sfujim/TD3_BC. We emphasise that policy evaluation was tuned using policies from the dataset, not from our method or the baselines.

Hyperparameter	**Range**
λ	uniform(1,4)
Critic Learning Rate	uniform($1e-5, 1e-4$)
Actor Learning Rate	uniform($1e-5, 1e-4$)
Training Epochs	400
Minibatches Per Epoch	2500
Minibatch Size	256

Table 4.3: ILBRL Hyperparameter Ranges

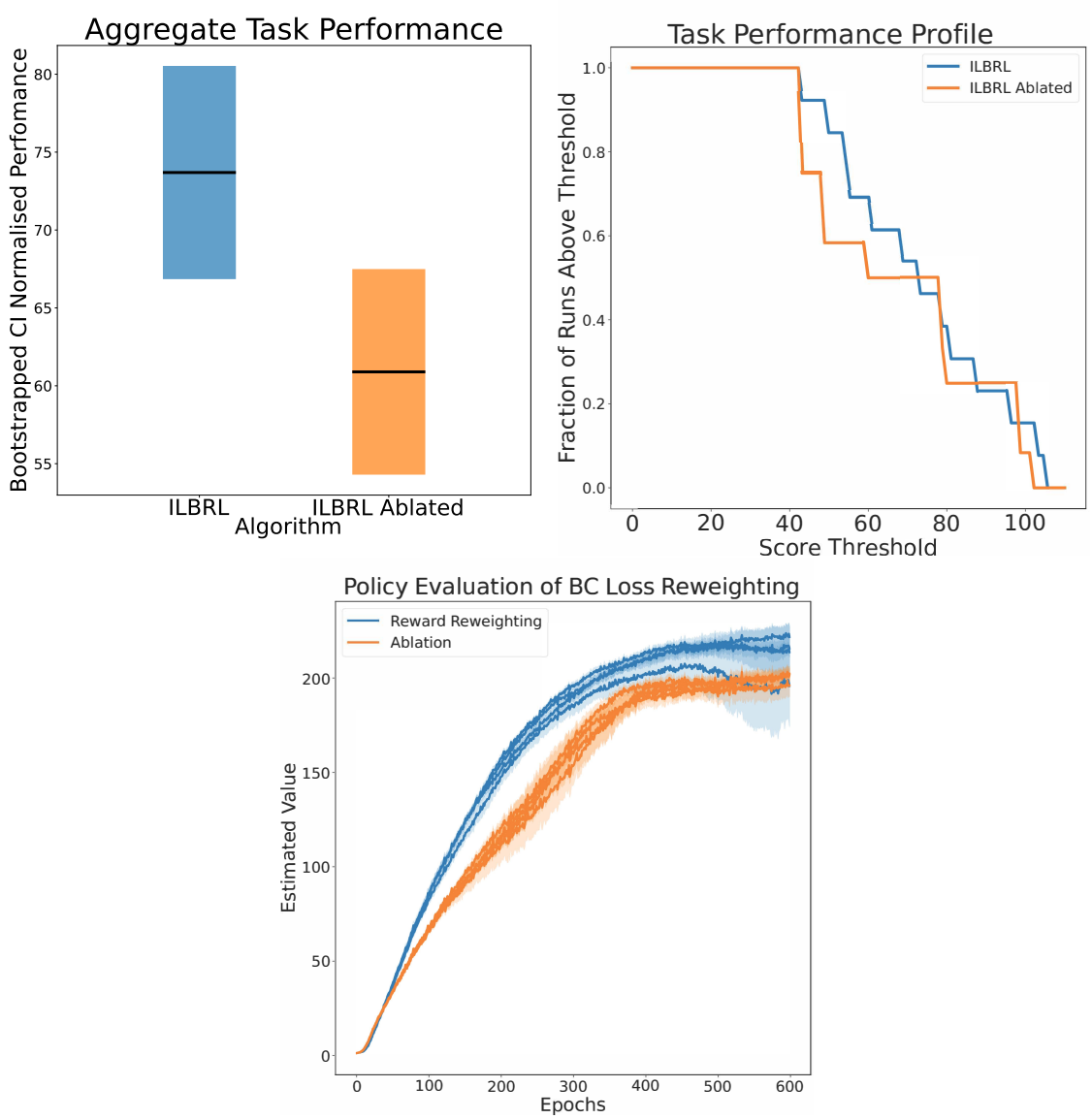


Figure 4.4: Performance of ILBRL with reward-weighted BC loss and without reward weighting (ablated). We see that reward weighting does provide some performance gains on the true environment, (left and centre figures), and that these gains are reflected in our offline policy evaluation procedure (right).

Hyperparameter	**Range**
Reward Scale	uniform(0.05, 3)
Discriminator Learning Rate	uniform($1e-4, 1e-3$)
Actor Learning Rate	uniform($1e-5, 1e-4$)

Table 4.4: SMODICE & ORIL Hyperparameter Ranges

Hyperparameter	Range
Learning Rate	loguniform$(3e-4, 3e-2)$
Training Epochs	400
Minibatches Per Epoch	2500
Minibatch Size	256

Table 4.5: Behavioural Cloning Hyperparameter Ranges

Hyperparameter	Range
Expert Dataset Fraction	choice$([0, 0.3, 0.5, 0.8])$
Target Network Update Frequency	choice$([2, 10, 16])$
Value Learning Rate	choice$([3e-6, 3e-5, 3e-4, 3e-3])$

Table 4.6: Off Policy Evaluation Hyperparameter Ranges

Final Hyperparameters

In this section we detail the hyperparameters selected for use by the policy evaluation process. We note that for reproducibility reasons, the random samples used during training were seeded identically, so the configurations for `Halfcheetah`, `Walker2d`, and `Ant` are identical for ILBRL, all environments ended up selecting the same hyper-parameters for ORIL, and hyper-parameters are identical for `Hopper` and `Walker2d` with BC.

Hyperparameter	Value
Expert Dataset Fraction	0.5
Target Network Update Frequency	16
Value Learning Rate	$3e-4$

Table 4.7: Policy Evaluation Hyperparameters

Hyperparameter	Value
λ	1.9898232924151003
Critic Learning Rate	$6.308344747206642e-05$
Actor Learning Rate	$4.279938571320915e-05$

Table 4.8: ILBRL Hyperparameters: Hopper

Hyperparameter	Value
λ	3.780707811143581
Critic Learning Rate	$3.193621317706127e-05$
Actor Learning Rate	$3.4187343220724746e-05$

Table 4.9: ILBRL Hyperparameters: Halfcheetah

Hyperparameter	Value
λ	3.780707811143581
Critic Learning Rate	$3.193621317706127e-05$
Actor Learning Rate	$3.4187343220724746e-05$

Table 4.10: ILBRL Hyperparameters: Walker2d

Hyperparameter	Value
λ	3.780707811143581
Critic Learning Rate	$3.193621317706127e-05$
Actor Learning Rate	$3.4187343220724746e-05$

Table 4.11: ILBRL Hyperparameters: Ant

Hyperparameter	Value
Reward Scale	0.132485274367925
Discriminator Learning Rate	0.0004279938571320915
Actor Learning Rate	$6.308344747206642e-05$

Table 4.12: SMODICE Hyperparameters: Hopper

Hyperparameter	Value
Reward Scale	0.09806293680143131
Discriminator Learning Rate	0.0006509491094594771
Actor Learning Rate	$8.59713615304861e-05$

Table 4.13: SMODICE Hyperparameters: Halfcheetah

Hyperparameter	Value
Reward Scale	0.17006151306334566
Discriminator Learning Rate	0.00012462906034979525
Actor Learning Rate	$1.3235001285067025e-05$

Table 4.14: SMODICE Hyperparameters: Walker2d

Hyperparameter	Value
Reward Scale	0.15584724849066423
Discriminator Learning Rate	0.0008741470800027616
Actor Learning Rate	$3.9755828061032105e-05$

Table 4.15: SMODICE Hyperparameters: Ant

Hyperparameter	Value
Reward Scale	0.1
Discriminator Learning Rate	0.0006509491094594771
Actor Learning Rate	$8.59713615304861e-05$

Table 4.16: ORIL Hyperparameters: All Environments

Hyperparameter	Value
Learning Rate	0.01463435996151319

Table 4.17: BC Hyperparameters: Hopper

Hyperparameter	Value
Learning Rate	0.017457140313552756

Table 4.18: BC Hyperparameters: Halfcheetah

Hyperparameter	Value
Learning Rate	0.01463435996151319

Table 4.19: BC Hyperparameters: Walker

Hyperparameter	Value
Learning Rate	0.005028663892514822

Table 4.20: BC Hyperparameters: Ant

4.7 Conclusions

As we have seen in this chapter, complications arise when our goal is to learn a behaviour which affects the distribution of test data observed, even when operating with a fixed dataset and clear objective function. While this setting is the closest to supervised learning that will be observed in our work, additional care needs to be taken in order to provide

sample efficiency bounds and generalisation guarantees. In our case these additional considerations lead to the need for a high quality exploratory dataset, alongside a smaller amount of expert behaviour.

Because of these additional concerns, common practice in offline RL and IL make use of hyperparameter tuning and evaluation procedures that fundamentally undermine the use cases and settings they are designed for. This makes it impossible to compare offline algorithms fairly. In order to address these problems we have developed a new protocol for analysis of offline RL methods which prevents information leak from the true environment and allows data-fair comparison across algorithms.

Despite these challenges, we are able to provide concrete guarantees for our simple algorithm in the tabular average reward setting. Our algorithm is able to generalise expert behaviour to unseen states more effectively than more complex SOTA baselines on challenging control environments with very little expert data.

5

Target Networks and Generalisation—Breaking Down the Deadly Triad

The content of this chapter was developed as part of the paper Why Target Networks Stabilise Temporal Difference Methods *[Fellows et al., 2023]*

One of the most catastrophic and pathological problems in reinforcement learning (RL) occurs when using a combination of TD methods with off-policy learning and function approximation. Styled the *deadly triad*, this seemingly natural setting has been shown to lead to divergence in simple environments [Baird, 1995]. Current theory suggests that this arises due to a misalignment between the function approximation class and the Bellman operator, which, when sampled off policy, leads to expansionary updates. This leaves us with a clear need both for algorithmic subversion of the deadly triad and for a more clear theoretical characterisation of how and why this seemingly innocuous—and highly desirable—setting leads to divergence.

Despite the lingering mystery of the deadly triad, recent decades have led to some of the most impressive empirical displays of RL efficacy in this exact setting: off-policy TD with deep neural networks as function approximation. This model class enables

automatic learning of complex state abstraction, leading to effective generalisation across the state space. However, neural networks can be challenging to optimise, requiring large amounts of data in order to generalise well to unseen states, as well as requiring stable targets in order to generalise well. This has led to the introduction of new, RL-specific techniques, in order to enable the use of this powerful function approximation class for state abstraction in RL. The most conserved of these techniques is the *target network*, which makes use of a lagged set of parameters for the value function approximator, in order to temporarily stabilise targets for value function updates. This chapter dives deeply into the theory of target networks, revealing that they serve a fundamental role in stabilising TD-style learning, providing an antidote to the deadly triad, as opposed to the conventional wisdom that they are a "trick" used to fit together neural network function approximation with RL.

Target networks—originally introduced a decade ago by Mnih et al. [2013, 2015] as a technique to limit correlation in neural network updates across states—have become a ubiquitous feature of state-of-the-art deep reinforcement learning algorithms [Lillicrap et al., 2015, Haarnoja et al., 2017, 2018, Fujimoto et al., 2018]. The conservation of this approach across off-policy deep RL methods suggests that something quite fundamental is at play. Whilst recent analysis has characterised the convergence properties of policy evaluation using target networks, existing approaches focus on asymptotic results, and usually make simplifying assumptions that neither hold in practice nor account for the true behaviour of target network-based updates. Here, in order to characterise the surprising effectiveness of state abstraction using deep neural networks in RL, we formalise the use of target networks as a class of partially fitted policy evaluation (PFPE) algorithms. These algorithms bridge the gap between traditional TD approaches [Sutton, 1988] and fitted policy evaluation (FPE) [Bertsekas and Ioffe, 1996, Le et al., 2019]. With correct hyperparameters, we show in this chapter that the use of target networks can help prevent deep RL algorithms from diverging, even in regimes where TD diverges. We also introduce a regularisation scheme that leads to guaranteed convergence of our online algorithm, without the need to resort to projection of network parameters as is done in previous work. Additionally, we establish finite-time performance bounds for

the use of target networks and general (convex) function approximation—without strong simplifying assumptions—providing theoretical justification for the empirical success that has been observed in challenging, off-policy tasks.

In this chapter, to characterise the performance of target networks, PFPE—which has traditionally been viewed through the lens of two-timescale analysis—is canonised as a single update applied only to the target network parameters. The stability of the algorithm is determined by analysing the eigenvalues of the Jacobian of this update. This formulation allows characterisation of both the limiting (asymptotic) and finite-time (non-asymptotic) convergence properties of PFPE. Furthermore, it suggests, counterintuitively, that target networks are actually the object being optimised rather than merely a means to stabilise conventional TD updates. Later in this chapter, we make use of this insight and empirically investigate a novel target parameter update scheme that uses a momentum-style update [Polyak, 1964], setting the stage for future research of practical target-based algorithms. We develop key insights into the usefulness of target networks, which we find do not improve asymptotic performance when decaying step sizes are used, but improve the conditioning of TD and fitted methods when the step size does not tend to zero, as is often implemented in practice. Under non-decaying stepsizes, our Jacobian analysis show how PFPE reconditions the TD Jacobian allowing us to prove convergence in regimes where TD is unstable, providing an explanation for the surprising efficacy of deep RL methods that make use of target networks.

This chapter represents the first known analysis of target networks without making restrictive changes to the algorithms that are used in empirical practice. Furthermore, this investigation yields the first finite-time error bounds for target networks as they are used in practice, as opposed to the asymptotic results which are more common in the literature. By characterising how target networks help bypass the deadly triad, a fundamental RL pathology, our analysis provides a foundation for future algorithmic development in this challenging setting. Our insight opens the door for such developments by fixating on the optimisation path of the target network, indicating that more sophisticated target network update schemes may be a fruitful line for future study.

5.1 Off-policy Evaluation

In this chapter, we focus only on the off-policy evaluation problem with function approximation, looking to optimise a set of value function parameters, $\theta \in \Theta$, to approximate the long term expected value of a policy, π, across all states:

$$V_\theta(s) \approx V^\pi(s).$$

However, in the off-policy setting, we are restricted to transition data sampled from a distribution that does not match the steady-state induced by π. For simplicity of notation and to accommodate the introduction of target networks in Section 5.2, we introduce some shorthand for the relevant data distributions. The data that we will be learning from is provided in the form of tuples, each of which we denote by $\varsigma := (s, a, r, s', a')$. These tuples are sampled from a fixed distribution, ρ_ς. The marginal distribution over states is given by d, and the marginal distribution over actions, conditioned on the state, is given by μ.

The following i.i.d. assumption on our data distribution will be used for clarity of exposition:

Assumption 5.1. *Each $\varsigma \sim \rho_\varsigma$ is drawn i.i.d..*

Typically, ρ_ς is the steady-state distribution of an ergodic Markov chain induced by following policy μ at all states, though in principle, data may be gathered from multiple policies, or sampled directly by other means.

5.1.1 Fitted Policy Evaluation

While direct application of online TD is occasionally possible in the off-policy evaluation setting with function approximation, more sophisticated methods may also be applied. In particular, fitted methods improve on the sample efficiency and stability of TD methods by explicitly incorporating the limitations of the function approximation class through the use of a projection operator [Szepesvári, 2010]. This is important since it may not be possible for our value function to perfectly match the true value of the target policy. Recall that these methods generally perform some variant of the iterate $Q_{\theta_{l+1}} =$

$\Pi^{\rho_S}\mathcal{T}^{\pi}Q_{\theta_l}$ where Π^{ρ_S} is the projection operator under the sampled data distribution, $\Pi^{\rho_S}Q = \arg\min_{Q'}\|Q' - Q\|_{d,\mu}$. These updates are known as fitted policy evaluation (FPE), and give the closest approximation to iteration of the Bellman operator within our function approximation class. Using a contraction argument, it has been shown that if $\mu = \pi$, i.e. we are sampling on policy, and linear function approximation is used, both TD and FPE will converge to the same fixed point [Szepesvári, 2010].

For some function approximation classes, the projection operator can be solved for in closed form. This gives rise to the LSPE algorithm [Lagoudakis and Parr, 2003] in the context of linear function approximation. However, in order to achieve more complex abstraction over the state space, richer function classes may need to be used. In this case, we must apply other solution methods for computing the projection operator. One possible solution is to introduce a second set of parameters, $\bar{\theta}_l \in \theta$, which can be used as an optimisation target, to keep track of our application of the Bellman operator. These parameters give rise to a convex inner optimisation problem which looks to solve for the application of the projection operator. Formally, at every iteration, l, we want to find the argmin of the loss:

$$\mathcal{L}(\theta; \bar{\theta}_l) \coloneqq \|Q_\theta - \mathcal{T}^{\pi}[Q_{\bar{\theta}_l}]\|_{d,\mu}. \tag{5.1}$$

One general scheme for this minimisation process is to perform stochastic gradient descent on the parameters, θ, until they converge. This can be viewed as a two-timescale optimisation process with iterates on the faster timescale obeying

$$\theta_{i+1} = \theta_i + \alpha_i \nabla_\theta \mathcal{L}(\theta_i; \bar{\theta}_l), \tag{5.2}$$

until they converge, and iterates on the slower timescale then corresponding to

$$\bar{\theta}_{l+1} = \theta_\infty, \tag{5.3}$$

where θ_∞ is the final iterate of the SGD update. While this schema works well in theory, in practice it may be too computationally expensive to iterate until convergence in the inner optimisation problem. For the remainder of this section, we discuss how classic semigradient TD represents an approximation of this procedure. In the subsequent

section we develop a family of algorithms—corresponding to the use of target networks in deep RL—which allow us to bridge the gap between FPE and its semigradient TD approximation.

5.1.2 TD(0) and FPE

Recall that the semigradient TD update is given by:

$$\theta_{l+1} = \theta_l + \alpha_l \left(r + \gamma Q_{\theta_l}(s', a') - Q_{\theta_l}(s, a)\right) \nabla_\theta Q_{\theta_l}(s, a).$$

As originally noted by [Yu and Bertsekas, 2009], this corresponds to the FPE update scheme given by Eqs. (5.2) and (5.3) if, instead of completing the inner optimisation process, we instead approximate it with a single step of SGD on $\mathcal{L}$.

For concision in this chapter, we abuse notation somewhat and will use δ to represent a vector-valued *update*, rather than the scalar value *TD error* used elsewhere in the paper. Furthermore, to unify notation for TD, FPE, and the methods that we develop later, we use a functional form which makes the dependency on the target parameters and the data explicit:

$$\delta(\theta, \theta', \varsigma) \coloneqq \left(r + \gamma Q_{\theta'}(s', a') - Q_\theta(s, a)\right) \nabla_\theta Q_\theta(s, a). \tag{5.4}$$

This allows us to write the TD parameter update as:

$$\theta_{i+1} = \theta_i + \alpha_i \delta(\theta_i, \theta_i, \varsigma). \tag{5.5}$$

We denote the expected fitted TD-error vector as: $\delta(\theta, \theta') \coloneqq \mathbb{E}_{\varsigma \sim \rho_\varsigma}[\delta(\theta, \theta', \varsigma)]$. This notation allows us to define the set of TD fixed points as:

$$\theta^\star \in \theta^\star \coloneqq \{\theta | \delta(\theta, \theta) = 0\}. \tag{5.6}$$

We recall that if a TD algorithm converges, it converges to a TD fixed point. Convergence of TD methods can be guaranteed for linear function approximators when sampling on-policy in an ergodic MDP: that is, the sampling distribution, ρ_ς, comes from a single policy, μ, and the steady state distribution of the target policy, ρ_π is the same as ρ_μ. We

characterise this convergence condition more precisely as part of our asymptotic analysis in Section 5.3.1.

With this notation, alongside an understanding of the relationship between FPE and TD, we are ready to discuss a unifying class of algorithms that approximate FPE, but also include TD as a specific case.

5.2 Partially Fitted Policy Evaluation

If FPE corresponds to iterations of the projected Bellman operator, and TD corresponds to an approximation of this via a single step of SGD on the projection loss given in Eq. (5.1), then it is easy to imagine that we can interpolate between these approaches by conducting any finite number of iterations, $1 \leq k < \infty$ on the loss in Eq. (5.1). We refer to such an algorithm as *partially fitted policy evaluation* (PFPE), as it conducts FPE-style updates, but without fully fitting to the target parameters. This corresponds to the update scheme—here presented with unified data indexing across both time scales—

$$\theta_{kl+i+1} = \theta_{kl+i} + \alpha_{kl+i}\delta(\theta_{kl+i}, \bar{\theta}_l, \varsigma), \tag{5.7}$$

$$\bar{\theta}_{l+1} = \theta_{k(l+1)}. \tag{5.8}$$

The function approximator update in Eq. (5.7) carries out k iterations of SGD on the loss before updating the target parameters using Eq. (5.8). In the limit as $k \to \infty$, assuming convergence of SGD to a global minimum, this reduces to the FPE update. By interpolating between the TD update and the FPE update, PFPE is able to achieve the improved convergence properties of FPE relative to TD, but in finite time for general function approximators, without the need for total optimisation at each step.

Without loss of generality, we assume that $\bar{\theta}_0$ is deterministic with $\|\bar{\theta}_0\| < \infty$ and $\alpha_i = \alpha_l$ for all $kl \leq i < k(l+1)$, that is stepsizes only change after updating target parameters. As the target parameters are updated to match the value parameters every k timesteps in Eq. (5.8), rather than trying to reason over two sets of parameters, it suffices to consider the target parameter update in isolation when analysing PFPE. Thus the goal of the

following section will be to reformulate our two-timescale update scheme to a single update, applied to the target parameters in the canonical form:

$$\bar{\theta}_{l+1} = g^k(\bar{\theta}_l, \mathcal{D}, \alpha_l), \quad \mathcal{D} \sim P_{\mathcal{D}}, \tag{5.9}$$

where $\mathcal{D} \coloneqq \{\varsigma_i\}_{i=1}^k$ is a set of k i.i.d. samples from ρ_ς, with joint distribution $P_{\mathcal{D}}$ and $g^k(\bar{\theta}_l, \mathcal{D}_l, \alpha_l)$ reduces the k nested updates from Eq. (5.7) into a single update for the target parameters. While we find that explicitly formulating the update in this manner is impossible for general function approximation, we can prove key properties of the update with this functional form, simplifying analysis.

5.2.1 The Jacobians of the Update

In order to begin with our analysis, we define three key matrices which will be used later to define the conditions for convergence and convergence rates for TD, FPE, and PFPE.

The Hessian of the expected loss in Eq. (5.1) is defined as

$$H(\theta, \bar{\theta}) := \nabla_\theta^2 \mathcal{L}(\theta; \bar{\theta}),$$

the Jacobian in the second argument of the expected TD error vector is given by

$$J_\delta(\theta, \bar{\theta}) := \nabla_{\theta'} \delta(\theta, \theta')|_{\theta'=\bar{\theta}},$$

and we define the Jacobian of the TD(0) update as

$$J_{\mathrm{TD}}(\bar{\theta}) := \nabla_\theta \delta(\theta, \theta)|_{\theta=\bar{\theta}}.$$

The Hessian, which we assume to be diagonalisable, will be used to characterise how sequential inner updates influence the resulting parameters. The Jacobian of the TD error vector will characterise the optimum of this fitting procedure, while the Jacobian of the TD(0) update describes the (semigradient TD-style) update that happens at the start of each PFPE iteration. We also note that, by the chain rule,

$$J_{\mathrm{TD}}(\bar{\theta}) = J_\delta(\bar{\theta}_l, \bar{\theta}_l) - H(\bar{\theta}_l; \bar{\theta}_l)$$

Our assumption on the Hessian can be made without loss of generality as an arbitrarily small perturbation may be applied to the matrix in order to make its eigenvalues distinct and therefore diagonalisable.

In order to ensure existence of these matrices, we require the expected PFPE update to be differentiable almost everywhere. This can be guaranteed by a Lipschitz assumption on the function approximators. In addition to this, we require that the variance of the updates is bounded, motivating the following regularity assumption:

Assumption 5.2 (Function Approximator Regularity)**.** *We assume that $\delta(\theta, \theta')$ is Lipschitz in θ, θ' with constant L: $\|\delta(\theta_1, \theta_1') - \delta(\theta_2, \theta_2')\| \leq L(\|\theta_1 - \theta_2\| + \|\theta_1' - \theta_2'\|)$, Θ is convex, and $\mathbb{V}_{\varsigma\sim\rho_\varsigma}[\delta(\theta, \theta, \varsigma)] \leq \sigma_\delta^2$ for some $\sigma_\delta^2 < \infty$.*

Achieving bounded variance when using unbounded function approximators is often performed in practice through the use of gradient clipping, or truncation the TD error vector.

In order to manage the nonlinearity of our function approximation scheme, we introduce the path-mean Jacobian, which corresponds to the average of all of the Jacobians along the line joining θ to a TD fixed-point, $\theta^\star$. Assumption 5.2 ensures that the line integral joining any two points in Θ always exists.

The principle elements of our analysis are the following linearisations:

$$\begin{aligned}
\bar{H}(\theta, \theta^\star; \bar{\theta}_l) &\coloneqq -\int_0^1 \nabla^2_{\theta'} \mathcal{L}(\theta' = \theta - t(\theta - \theta^\star), \bar{\theta}_l) dt, \\
\bar{J}_\delta(\theta, \theta^\star; \bar{\theta}_l) &\coloneqq \int \nabla_{\theta'} \delta(\bar{\theta}_l, \theta' = \theta - t(\theta - \theta^\star)) dt, \\
\bar{J}_{\text{TD}}(\theta, \theta^\star) &\coloneqq \int_0^1 \nabla_{\theta'} \delta(\theta', \theta')|_{\theta' = \theta - t(\theta - \theta^\star)} dt.
\end{aligned}$$

Analysis in the subsequent section will show that J_{TD} will determine the asymptotic convergence of TD, as well as PFPE with decaying step-sizes. On the other hand, when constant step sizes are used, PFPE acts more like FPE.

Stability of FPE To demonstrate the usefulness of the path-mean framework, we use our path-mean linearisations to demonstrate a convergence condition for FPE with nonlinear function approximation, a novelty in the literature.

Lemma 5.1. *Under assumption 5.2, the sequence of fitted policy evaluation updates* $\bar{\theta}_l^\star \in \arg\inf_\theta \mathcal{L}(\theta, \bar{\theta}_l)$ *satisfy:*

$$\bar{\theta}_l^\star - \theta^\star = \prod_{i=0}^{l} \bar{H}(\bar{\theta}_{i+1}^\star, \theta^\star; \bar{\theta}_i^\star)^{-1} \bar{J}_\delta(\bar{\theta}_i^\star, \theta^\star; \theta^\star)(\bar{\theta}_0 - \theta^\star).$$

Proof. Given $\bar{\theta}_l$, each FPE update $\bar{\theta}_l^\star$ must be an element of the set:

$$\bar{\theta}_l^\star \in \{\theta | \delta(\theta, \bar{\theta}_l) = 0\},$$

which we use to derive a stability condition for the projection operator:

$$\begin{aligned} \delta(\bar{\theta}_l^\star, \bar{\theta}_l) = \delta(\theta^\star, \theta^\star) = 0 \\ \implies \delta(\bar{\theta}_l^\star, \bar{\theta}_l) - \delta(\theta^\star, \bar{\theta}_l) = \delta(\theta^\star, \theta^\star) - \delta(\theta^\star, \bar{\theta}_l). \end{aligned}$$

Let $\ell_1(t) \coloneqq \bar{\theta}_l^\star - t(\bar{\theta}_l^\star - \theta^\star)$ and $\ell_2(t) \coloneqq \bar{\theta}_l - t(\bar{\theta}_l - \theta^\star)$. We introduce the notation:

$$\delta_1(t, \bar{\theta}_l) \coloneqq \delta(\ell_1(t), \bar{\theta}_l), \quad \delta_2(t, \theta^\star) \coloneqq \delta(\theta^\star, \ell_2(t)).$$

We observe that $\delta_1(0, \bar{\theta}_l) = \delta(\bar{\theta}_l^\star, \bar{\theta}_l)$ and $\delta_1(1, \bar{\theta}_l) = \delta(\theta^\star, \bar{\theta}_l)$. We also note that $\delta_2(0, \theta^\star) = \delta(\theta^\star, \bar{\theta}_l)$ and $\delta_2(1, \theta^\star) = \delta(\theta^\star, \theta^\star)$. From the fundamental theorem of calculus and assumption 5.2, it follows:

$$\begin{aligned} \delta_1(0, \bar{\theta}_l) - \delta_1(1, \bar{\theta}_l) = \delta_2(1, \theta^\star) - \delta_2(0, \theta^\star), \\ \implies -\int_0^1 \partial_t \delta(\theta = \ell_1(t), \bar{\theta}_l) dt = \int_0^1 \partial_t \delta(\theta^\star, \theta = \ell_2(t)) dt, \\ \implies \int_0^1 \nabla_\theta \delta(\theta = \ell_1(t), \bar{\theta}_l)(\bar{\theta}_l^\star - \theta^\star) dt = -\int_0^1 \nabla_\theta \delta(\theta^\star, \theta = \ell_2(t))(\bar{\theta}_l - \theta^\star) dt, \\ \implies -\int_0^1 \nabla_\theta^2 \mathcal{L}(\theta = \ell_1(t); \bar{\theta}_l)(\bar{\theta}_l^\star - \theta^\star)) dt = -\int_0^1 \nabla_\theta \delta(\theta^\star, \theta = \ell_2(t))(\bar{\theta}_l - \theta^\star) dt, \\ \implies \int_0^1 \nabla_\theta^2 \mathcal{L}(\theta = \ell_1(t); \bar{\theta}_l) dt (\bar{\theta}_l^\star - \theta^\star) = \int_0^1 \nabla_\theta \delta(\theta^\star, \theta = \ell_2(t)) dt (\bar{\theta}_l - \theta^\star), \\ \implies \bar{H}(\bar{\theta}_l^\star, \theta^\star; \bar{\theta}_l)(\bar{\theta}_l^\star - \theta^\star) = \bar{J}_\delta(\bar{\theta}_l, \theta^\star; \theta^\star)(\bar{\theta}_l - \theta^\star), \\ \implies (\bar{\theta}_l^\star - \theta^\star) = \bar{H}(\bar{\theta}_l^\star, \theta^\star; \bar{\theta}_l)^{-1} \bar{J}_\delta(\bar{\theta}_l, \theta^\star; \theta^\star)(\bar{\theta}_l - \theta^\star), \end{aligned}$$

where we applied the chain rule in deriving the third line. Recursively applying the result yields:

$$\bar{\theta}_l^\star - \theta^\star = \prod_{i=0}^{l} \bar{H}(\bar{\theta}_{i+1}^\star, \theta^\star; \bar{\theta}_i^\star)^{-1} \bar{J}_\delta(\bar{\theta}_i^\star, \theta^\star; \theta^\star)(\bar{\theta}_0 - \theta^\star),$$

as required. □

Lemma 5.1 suggests that if $\sup_{\theta,\theta'} \|\bar{H}(\theta', \theta^\star; \theta)^{-1} \bar{J}_\delta(\theta, \theta^\star; \theta^\star)\| \leq 1$, the FPE update will represent a contraction, and thus lead to convergence. This condition is important, as we can expect PFPE to obey similar dynamics if the k is taken to be sufficiently large. In the next section we derive a similar condition for convergence of PFPE under a regime of decaying step sizes.

5.3 Asymptotic Analysis

We now study the behaviour of Eq. (5.9) in the limit of $l \to \infty$ and $\alpha_l \to 0$. This represents the first analysis of target networks with nonlinear function approximation, using a single timescale stochastic approximation approach. While we will ultimately find that target networks do not improve convergence conditions compared to TD, i.e. for PFPE to converge with diminishing step sizes, TD must also converge. This surprising result contradicts popular intuition and empirical results which indicate that the use of target networks stabilise TD. However, in the following section we establish more promising results in the nonasymptotic case. We recall the standard Robbins-Munro condition for the decaying stepsizes that is a necessary condition to ensure convergence to a fixed point:

Assumption 5.3. *Each α_l is a positive scalar with $\sum_{l=0}^{\infty} \alpha_l = \infty$ and $\sum_{l=0}^{\infty} \alpha_l^2 < \infty$.*

As will become clear in our proof of Theorem 5.3, we find that the following key assumption is necessary for convergence of PFPE:

Assumption 5.4 (TD Stability)**.** *There exists a region $\mathcal{X}_{TD}(\theta^\star)$ containing a fixed point $\theta^\star$ such that $\bar{J}_{TD}(\theta, \theta^\star)$ has strictly negative eigenvalues for all $\theta \in \mathcal{X}_{TD}(\theta^\star)$.*

Notably, the necessity of assumption 5.4 suggests that the stability of PFPE under diminishing stepsizes is determined only by the eigenvalues of the single step (TD) path-mean Jacobian $\bar{J}_{\text{TD}}(\theta, \theta^\star)$, regardless of the value of k or α_l. Indeed, our stochastic approximation analysis will show these algorithms to be provably divergent if this condition cannot be satisfied [Pemantle, 1990]. From this perspective, if TD diverges then so will PFPE under *diminishing stepsizes*, hence the asymptotic stability of PFPE is independent of k and α_l, and, unlike performing smooth updates under a two-timescale regime, introducing target parameters that are updated periodically every k timesteps does not improve asymptotic convergence properties under this analysis. Once assumption 5.4 has been established, there are several approaches to prove convergence of the PFPE update under varying sampling conditions and projection assumptions. We follow the proof of [Vidyasagar, 2023].

In order to use the tools of stochastic approximation, we need to define a Martingale difference sequence that captures the behaviour of our updates. Let $\{\theta_i\}_{i=0}^{k}$ denote the intermediate function approximation parameters between target parameter updates $\bar{\theta}_{l+1}$ and $\bar{\theta}_l$, with $\theta_0 = \bar{\theta}_l$ and $\theta_k = \bar{\theta}_{l+1}$. We start by recursively expanding our target parameter updates as:

$$
\begin{aligned}
\theta_1 &= \bar{\theta}_l + \alpha_l \delta(\bar{\theta}_l, \bar{\theta}_l, \varsigma_0), \\
\theta_2 &= \theta_1 + \alpha_l \delta(\theta_1, \bar{\theta}_l, \varsigma_1), \\
&= \bar{\theta}_l + \alpha_l \left(\delta(\bar{\theta}_l, \bar{\theta}_l, \varsigma_0) + \delta(\bar{\theta}_l + \alpha_l \delta(\bar{\theta}_l, \bar{\theta}_l, \varsigma_0), \bar{\theta}_l, \varsigma_1) \right), \\
\theta_3 &= \theta_2 + \alpha_l \delta(\theta_2, \bar{\theta}_l, \varsigma_2), \\
&= \bar{\theta}_l + \alpha_l \left(\delta(\bar{\theta}_l, \bar{\theta}_l, \varsigma_0) + \delta(\bar{\theta}_l + \alpha_l \delta(\bar{\theta}_l, \bar{\theta}_l, \varsigma_0), \bar{\theta}_l, \varsigma_1) \right) \\
&\qquad + \alpha_l (\delta(\bar{\theta}_l + \alpha_l \left(\delta(\bar{\theta}_l, \bar{\theta}_l, \varsigma_0) + \delta(\bar{\theta}_l + \alpha_l \delta(\bar{\theta}_l, \bar{\theta}_l, \varsigma_0), \bar{\theta}_l, \varsigma_1) \right), \bar{\theta}_l, \varsigma_2), \\
&\vdots \\
\theta_k &= \bar{\theta}_l + \alpha_l \sum_{i=0}^{k-1} \delta(\bar{\theta}_l + \alpha_l h_i(\bar{\theta}_l, \mathcal{D}, \alpha_l), \bar{\theta}_l, \varsigma_i), \\
&= \bar{\theta}_l + \alpha_l h_k(\bar{\theta}_l, \mathcal{D}, \alpha_l),
\end{aligned}
$$

where we define $h_i(\bar{\theta}_l, \mathcal{D}, \alpha_l)$ recursively as:

$$h_i(\bar{\theta}_l, \mathcal{D}, \alpha_l) \coloneqq \sum_{j=0}^{i-1} \delta(\bar{\theta}_l + \alpha_l h_j(\bar{\theta}_l, \mathcal{D}, \alpha_l), \bar{\theta}_l, \varsigma_j).$$

and remark that $h_0(\bar{\theta}_l, \mathcal{D}, \alpha_l) = 0$ trivially. We write our target parameter update as:

$$\bar{\theta}_{l+1} = \theta_k = \bar{\theta}_l + \alpha_l \left(k\delta(\bar{\theta}_l, \bar{\theta}_l) + \mathcal{M}_{l+1} + \varepsilon_{l+1} \right),$$

where

$$\varepsilon_{l+1} \coloneqq h_k(\bar{\theta}_l, \mathcal{D}_l, \alpha_l) - \sum_{i=0}^{k-1} \delta(\bar{\theta}_l, \bar{\theta}_l, \varsigma_i),$$

and $\mathcal{M}_{l+1}$ defines the Martingale sequence:

$$\mathcal{M}_{l+1} \coloneqq \sum_{i=0}^{k-1} \delta(\bar{\theta}_l, \bar{\theta}_l, \varsigma_i) - k\delta(\bar{\theta}_l, \bar{\theta}_l)$$

Next, we demonstrate that the proof of Borkar [2008, Theorem 2.2] can be adapted to account for the additional term ε_{l+1} that arises due to the use of target networks in the updates. To this end, we introduce a new assumption which can be used to favourably characterise convergence of PFPE by uniformly bounding the gradients.

Assumption 5.5. *For all $\theta, \theta' \in \Theta$, and each transition data ς,*

$$\|\delta(\theta, \theta', \varsigma)\| \leq c_h \leq \infty,$$

almost surely.

Equipped with this assumption, which favours convergence conditions for PFPE, we now inreoduce Lemma 5.2, which demonstrates that as stepsizes tend to zero, the effect of ϵ_{l+1} becomes negligible, hence the inclusion of ε_{l+1} is also negligible to our analysis of the underlying ODE defined by the TD updates.

Lemma 5.2. *Let $\nu_{n,n+m} \coloneqq \sum_{l=n}^{m+n-1} \alpha_l \epsilon_{l+1}$ for $m \geq 1$. Under Assumptions 5.1 to 5.3, and assumption 5.5, $\lim_{n\to\infty} \sup_m \|\nu_{n,n+m}\| = 0$ almost surely.*

Proof. We start by bounding each $\|\epsilon_{i+1}\|$ using the the Lipschitzness of δ from assumption 5.2:

$$\begin{aligned}
\|\epsilon_{l+1}\| &= \left\| \sum_{i=0}^{k-1} \Big(\delta(\bar{\theta}_l + \alpha_l h_i(\bar{\theta}_l, \mathcal{D}, \alpha_l), \bar{\theta}_l, \varsigma_i) - \delta(\bar{\theta}_l, \bar{\theta}_l, \varsigma_i) \Big) \right\|, \\
&\leq \sum_{i=0}^{k-1} \left\| \delta(\bar{\theta}_l + \alpha_l h_i(\bar{\theta}_l, \mathcal{D}, \alpha_l), \bar{\theta}_l, \varsigma_i) - \delta(\bar{\theta}_l, \bar{\theta}_l, \varsigma_i) \right\|, \\
&\leq \sum_{i=0}^{k-1} L \left\| \bar{\theta}_l + \alpha_l h_i(\bar{\theta}_l, \mathcal{D}, \alpha_l) - \bar{\theta}_l \right\|, \\
&= \alpha_l L \sum_{i=0}^{k-1} \left\| h_i(\bar{\theta}_l, \mathcal{D}, \alpha_l) \right\|,
\end{aligned}$$

Using c_h from assumption 5.5, we bound $\|\epsilon_{l+1}\|$:

$$\|\epsilon_{l+1}\| \leq \alpha_l L \sum_{i=0}^{k-1} c_h = \alpha_l c_h k L,$$

almost surely. We use this result to bound $\|\nu_{n,n+m}\|$:

$$\|\nu_{n,n+m}\| \leq \sum_{l=n}^{m+n-1} \alpha_l \|\epsilon_{l+1}\| \leq c_h k L \sum_{l=n}^{m+n-1} {\alpha_l}^2. \tag{5.10}$$

Now, under assumption 5.3,

$$\lim_{n\to\infty} \sup_m \sum_{l=n}^{m+n-1} {\alpha_l}^2 = 0,$$

hence by the bound established in Eq. (5.10):

$$\lim_{n\to\infty} \sup_m \|\nu_{n,n+m}\| = 0,$$

almost surely, as required. □

Intuitively, this result tells us that taking finite k steps of PFPE becomes negligibly different from taking one TD update as the learning rate goes to zero. Equipped with the bound on the additional term induced by the target network, we are able to prove our main result in this context

Theorem 5.3. *Let Assumptions 5.1 to 5.5 hold. If there exists some fixed point $\theta^\star$ with region of contraction $\mathcal{X}_{TD}(\theta^\star)$ and timestep t such that $\bar{\theta}_l \in \mathcal{X}_{TD}(\theta^\star)$ for all $l \geq t$ the the sequence of target parameter updates in Eq.* (5.8) *converge almost surely to $\theta^\star$.*

Proof. Our update

$$\bar{\theta}_{l+1} = \bar{\theta}_l + \alpha_l \left(k\delta(\bar{\theta}_l, \bar{\theta}_l) + \mathcal{M}_{l+1} + \varepsilon_{l+1} \right),$$

is identical to the update presented in Borkar [2008, Eq. 2.1.1] with an additional term ε_{l+1}. Proof of convergence to the ODE is given by Borkar [2008, Lemma 1], which is predicated on the convergence of:

$$\Delta_{n,n+m} \coloneqq \zeta_{n+m} - \zeta_n,$$

from Borkar [2008, Eq. 2.1.6] where

$$\zeta_n = \sum_{l=0}^{n-1} \alpha_l \mathcal{M}_{l+1},$$

for $n \geq 1$, that is $\lim_{n\to\infty} \sup_m \|\Delta_{n,n+m}\| = 0$, almost surely. To adapt our updates so that Borkar [2008, Lemma 1] still applies, we recognise that the term ζ_n is now replaced in our updates with:

$$\bar{\zeta}_n = \sum_{l=0}^{n-1} \alpha_l (\mathcal{M}_{l+1} + \epsilon_{l+1}),$$

and hence $\Delta_{n,n+m}$ is replaced in our updates with:

$$\begin{aligned}
\bar{\Delta}_{n,n+m} \coloneqq & \bar{\zeta}_{n+m} - \bar{\zeta}_n, \\
= & \zeta_{n+m} - \zeta_n + \left(\sum_{l=0}^{n+m-1} \alpha_l \epsilon_{l+1} \right) - \left(\sum_{l=0}^{n-1} \alpha_l \epsilon_{l+1} \right), \\
= & \zeta_{n+m} - \zeta_n + \sum_{l=n}^{n+m-1} \alpha_l \epsilon_{l+1}, \\
= & \zeta_{n+m} - \zeta_n + \nu_{n,n+m}, \\
= & \Delta_{n,n+m} + \nu_{n,n+m},
\end{aligned}$$

where $\nu_{n,n+m}$ is defined as Lemma 5.2. All arguments of Borkar [2008, Lemma 1] remain unchanged, except Eq. 2.1.9, where we must now show that $\lim_{n\to\infty} \sup_m \|\bar{\Delta}_{n,n+m}\| = 0$:

$$\begin{aligned}
\lim_{n\to\infty} \sup_m \|\bar{\Delta}_{n,n+m}\| & \leq \lim_{n\to\infty} \sup_m \left(\|\Delta_{n,n+m}\| + \|\nu_{n,n+m}\| \right), \\
& \leq \lim_{n\to\infty} \left(\sup_m \|\Delta_{n,n+m}\| + \sup_m \|\nu_{n,n+m}\| \right), \\
& = \lim_{n\to\infty} \sup_m \|\Delta_{n,n+m}\| + \lim_{n\to\infty} \sup_m \|\nu_{n,n+m}\|.
\end{aligned}$$

Applying Lemma 5.2 yields $\lim_{n\to\infty}\sup_m\|\nu_{n,n+m}\| = 0$ almost surely, hence

$$\lim_{n\to\infty}\sup_m\|\bar{\Delta}_{n,n+m}\| \leq \lim_{n\to\infty}\sup_m\|\Delta_{n,n+m}\|,$$

which is proved in Borkar [2008, Lemma 1]. Convergence of our algorithm is thus only predicated on the convergence of the update:

$$\bar{\theta}_{l+1} = \bar{\theta}_l + \alpha_l\left(k\delta(\bar{\theta}_l, \bar{\theta}_l) + \mathcal{M}_{l+1}\right). \tag{5.11}$$

Borkar [2008, Theorem 2.2] proves convergence of Eq. (5.11) almost surely to $\theta^\star$ given the following four conditions hold:

I $k\delta(\theta, \theta)$ is Lipschitz in θ,

II Stepsizes α_l satisfy assumption 5.3,

III The sequence $\{\mathcal{M}_l, \mathcal{F}_l\}_{l\geq 0}$ is a Martingale difference sequence with respect to the increasing family of σ-algebras: $\mathcal{F}_l \coloneqq \sigma(\{\bar{\theta}_i, \mathcal{M}_i\}_{i\in\{0:l\}})$ where $\mathbb{E}\left[\mathcal{M}_{l+1}|\mathcal{F}_l\right] = 0$ and $\mathbb{E}\left[\|\mathcal{M}_{l+1}\|^2|\mathcal{F}_l\right] \leq C(1 + \|\bar{\theta}_l\|^2)$ for some positive $C < \infty$.

IV The sequence of iterates remain bounded, that is $\sup_l\|\bar{\theta}_l\| < \infty$ almost surely.

Conditions I and II hold trivially.

For Condition III, we can take expectations of the Martingale difference:

$$\begin{aligned}
\mathbb{E}\left[\mathcal{M}_{l+1}|\mathcal{F}_l\right] &= \mathbb{E}\left[\mathcal{M}_{l+1}|\mathcal{F}_l\right], \\
&= \mathbb{E}\left[\sum_{i=0}^{k-1}\delta(\bar{\theta}_l, \bar{\theta}_l, \varsigma_i) - k\delta(\bar{\theta}_l, \bar{\theta}_l)\middle|\mathcal{F}_l\right], \\
&= \sum_{i=0}^{k-1}\mathbb{E}\left[\delta(\bar{\theta}_l, \bar{\theta}_l, \varsigma_i)\middle|\mathcal{F}_l\right] - k\delta(\bar{\theta}_l, \bar{\theta}_l), \\
&= 0,
\end{aligned}$$

as required. We now show that the variance is bounded using assumption 5.2:

$$\begin{aligned}
\|\mathcal{M}_{l+1}\|^2 &= \left\|\sum_{i=0}^{k-1}\Big(\delta(\bar{\theta}_l,\bar{\theta}_l,\varsigma_i)-\delta(\bar{\theta}_l,\bar{\theta}_l)\Big)\right\|^2, \\
&\leq k^2\left\|\delta(\bar{\theta}_l,\bar{\theta}_l,\varsigma_i)-\delta(\bar{\theta}_l,\bar{\theta}_l)\right\|^2, \\
\implies \mathbb{E}\left[\|\mathcal{M}_{l+1}\|^2|\mathcal{F}_l\right] &\leq k^2\mathbb{E}\left[\left\|\delta(\bar{\theta}_l,\bar{\theta}_l,\varsigma_i)-\delta(\bar{\theta}_l,\bar{\theta}_l)\right\|^2\Big|\mathcal{F}_l\right], \\
&= k^2\mathbb{V}_{\varsigma\sim\rho_\varsigma}[\delta(\bar{\theta}_l,\bar{\theta}_l,\varsigma)], \\
&\leq k^2\sigma_\delta^2,
\end{aligned}$$

thereby satisfying Condition III.

Finally, we prove Condition IV using Vidyasagar [2023, Theorem 5], which states iterates remain bounded almost surely if:

(a) Conditions I and III hold;

(b) there exists some Lyapunov function $V:\theta\mapsto\mathbb{R}^+$ such that $a\|\theta-\theta^\star\|^2\leq V(\theta)\leq b\|\theta-\theta^\star\|^2$ for constants $a,b>0$ and $\|\nabla_\theta^2 V(\theta)\|$ is bounded, and;

(c) $\nabla_\theta V(\theta)^\top\delta(\theta,\theta)<0$ for all $\theta\in\mathcal{X}_{\text{TD}}(\theta^\star)$.

We propose $V(\theta)=\frac{1}{2}\|\theta-\theta^\star\|^2$ as a candidate Lyapunov function, which trivially satisfies (b). We now show (c) holds by applying the fundamental theorem of calculus to $\delta(\theta,\theta)$. Let $\ell(t)\coloneqq\theta-t(\theta-\theta^\star)$. Like in Lemma 5.1, it follows:

$$\begin{aligned}
\delta(\theta,\theta) &= \delta(\theta,\theta)-\underbrace{\delta(\theta^\star,\theta^\star)}_{=0}, \\
&= \delta\circ l(t=0)-\delta\circ l(t=1), \\
&= -\int_0^1\partial_t\delta\circ l(t)dt, \\
&= \int_0^1\nabla_\theta\delta\circ l(t)dt(\theta-\theta^\star),
\end{aligned}$$

hence:

$$\begin{aligned}
\nabla_\theta V(\theta)^\top\delta(\theta,\theta) &= (\theta-\theta^\star)^\top\int_0^1\nabla_\theta\delta\circ l(t)dt(\theta-\theta^\star), \\
&= (\theta-\theta^\star)^\top\bar{J}_{TD}(\theta,\theta^\star)(\theta-\theta^\star), \\
&< 0,
\end{aligned}$$

for all $\theta\in\mathcal{X}_{\text{TD}}(\theta^\star)$ under assumption 5.4, as required. □

The fact that we need assumption 5.4 in order to prove asymptotic convergence of PFPE indicates that, under a decaying stepsize regime, eventually, PFPE will behave exactly as TD and fail to converge under poor conditioning of the TD Jacobian. We now look to investigate the conditions which lead to such poor convergence, known colloquially as the deadly triad [Sutton and Barto, 2018].

5.3.1 The Deadly Triad

We have established that it is not possible to prove convergence of PFPE under diminishing stepsizes if assumption 5.4 does not hold. We now discuss how failure to adhere to assumption 5.4 formalises a phenomenon known as the deadly triad [Sutton and Barto, 2018] where it has been established that TD with function approximation cannot be proved to converge when either sampling off policy or using nonlinear function approximators.

Linear Function Approximation To control for the effect of nonlinear function approximation, we first investigate linear function approximators of the form $Q_\theta(s,a) = \phi(s,a)^\top\theta$ where $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^n$ is a feature vector. Introducing the shorthand:

$$\Phi \coloneqq \mathbb{E}_{s\sim d, a\sim\mu(s)}[\phi(s,a)\phi(s,a)^\top],$$

and

$$\Phi' \coloneqq \mathbb{E}_{s\sim d, a\sim\mu(s)}[\mathbb{E}_{s'\sim P(S'|s,a), a'\sim\pi(s')}[\phi(s',a')]\phi(s,a)^\top],$$

we can derive the linear path-mean TD Jacobian as:

$$\bar{J}_{\text{TD}}(\theta,\theta^\star,\alpha) = \gamma\Phi' - \Phi.$$

We now examine why the conditioning of $\bar{J}_{\text{TD}}(\theta,\theta^\star,\alpha)$ captures the effect of off-policy sampling on the convergence of TD(0).

For linear function approximators, one assumption that has been demonstrated to lead to convergence of off-policy TD is the "low distribution shift assumption" [Wang et al., 2021a,b]. This can be formalised as the assumption that $\gamma\|Q_\theta\|_{P_\mu,\pi} < \|Q_\theta\|_{d,\mu}$ for all θ, P_μ represents the one-step look-ahead distribution under policy μ and state distribution d.

We show here that this is a sufficient condition for $\gamma\Phi' - \Phi$ to have negative eigenvalues under assumption 5.4.

Starting from assumption 5.4 and the definition of negative definiteness, we need to show:

$$\theta^\top(\gamma\Phi' - \Phi)\theta < 0,$$

whenever $\gamma\|Q_\theta\|_{P_\mu,\pi} < \|Q_\theta\|_{d,\mu}$, for all θ. Investigating the first term by expanding the expectations, we see:

$$\begin{aligned}
\gamma\theta^\top\Phi'\theta &= \gamma\mathbb{E}_{s\sim d,a\sim\mu(s)}\left[\theta^\top\phi(s,a)\mathbb{E}_{s'\sim P(s,a),a'\sim\pi(s')}\left[\phi(s',a')^\top\theta\right]\right], \\
&= \gamma\mathbb{E}_{d,\pi,P_\mu}\left[\theta^\top\phi(s,a)\phi(s',a')^\top\theta\right], \\
&\leq \gamma\sqrt{\mathbb{E}_{d,\pi,P_\mu}\left[(\phi(s,a)^\top\theta)^2\right]\mathbb{E}_{d,\pi,P_\mu}\left[(\phi(s',a')^\top\theta)^2\right]}, \\
&\leq \gamma\sqrt{\mathbb{E}_{d,\mu}\left[(\phi(s,a)^\top\theta)^2\right]\mathbb{E}_{d,\pi,P_\mu}\left[(\phi(s',a')^\top\theta)^2\right]}, \\
&\leq \gamma\|Q_\theta\|_{d,\mu}\|Q_\theta\|_{P_\mu,\pi}.
\end{aligned}$$

This allows us to apply our assumption:

$$\theta^\top(\gamma\Phi' - \Phi)\theta \leq \gamma\|Q_\theta\|_{P_\mu,\pi}\|Q_\theta\|_{d,\mu} - \|Q_\theta\|_{d,\mu}^2 \leq \gamma\|Q_\theta\|_{d,\mu}^2 - \|Q_\theta\|_{d,\mu}^2 < 0$$

This implies that the function approximator class remains non-expansive under the one-step lookahead distribution, P_μ, thereby preventing the function approximator diverging as the Markov chain is traversed.

In the on-policy setting in an ergodic MDP, we can prove that there exists a stationary distribution d^π induced by following the target policy π—that is to say $\mu = \pi$, and $\rho^\pi = d^\pi$. Moreover it is assumed that samples come from d^π; hence by the definition of ergodicity, the one-step lookahead distribution is the stationary distribution: $\int_s P_\pi(s'|s)ds = \rho^\pi(s')$. It thus follows that $\gamma\|Q_\theta\|_{P_\mu,\pi} = \gamma\|Q_\theta\|_{d^\pi,\pi} < \|Q_\theta\|_{d^\pi,\pi}$ and hence assumption 5.4 holds automatically for on-policy TD in an ergodic MDP, thereby establishing the convergence properties as a special case via Theorem 5.3.

For off-policy data, it is not possible to prove that $\gamma\|Q_\theta\|_{P_\mu,\pi} < \|Q_\theta\|_{d,\mu}$ holds without further assumptions on the sampling policy and MDP. In general, it is not possible to show

that $\sup_{\theta\in\theta}\|\bar{J}_{\text{TD}}(\theta,\theta^\star,\alpha)\| < 1$ in the off-policy case as the distribution shift may be too high: there exist counterexample MDPs where off-policy algorithms such as Q-learning provably diverge under linear function approximation [Williams and Baird III, 1993, Baird, 1995].

Nonlinear Function Approximation Even in an on-policy regime, we cannot prove convergence of TD when nonlinear function approximators such as neural networks are used. In these cases, the path-mean Jacobian may not have a closed form solution. However, it can be bounded by the path-mean variant using the pointwise norm of the TD Jacobian. We start by bounding the maximum singular value:

$$\begin{aligned}
\sup\lambda\left(\bar{J}_{\text{TD}}(\theta',\theta^\star)\right) &= \sup_\theta \frac{\theta^\top\left(\bar{J}_{\text{TD}}(\theta',\theta^\star)\right)\theta}{\theta^\top\theta},\\
&= \sup_\theta \int_0^1 \frac{\theta^\top\left(J_{\text{TD}}(\theta'-t(\theta'-\theta^\star),\theta'-t(\theta'-\theta^\star))\right)\theta}{\theta\theta^\top}dt,\\
&\leq \int_0^1 \sup_\theta \frac{\theta^\top\left(J_{\text{TD}}(\theta'-t(\theta'-\theta^\star),\theta'-t(\theta'-\theta^\star))\right)\theta}{\theta\theta^\top}dt,\\
&\leq \int_0^1 \sup_{t\in[0,1]}\sup_\theta \frac{\theta^\top\left(J_{\text{TD}}(\theta'-t(\theta'-\theta^\star),\theta'-t(\theta'-\theta^\star))\right)\theta}{\theta\theta^\top}dt,\\
&\leq \sup_{t\in[0,1]}\sup_\theta \frac{\theta^\top\left(J_{\text{TD}}(\theta'-t(\theta'-\theta^\star),\theta'-t(\theta'-\theta^\star))\right)\theta}{\theta\theta^\top}\underbrace{\int_0^1 dt}_{=1},\\
&\leq \sup_{\theta'}\sup_\theta \frac{\theta^\top\left(J_{\text{TD}}(\theta',\alpha)-I\right)\theta}{\alpha\theta\theta^\top},\\
&= \sup_{\theta'}\lambda\left(J_{\text{TD}}(\theta',\theta')\right).
\end{aligned}$$

We now substitute for the definition of the TD Jacobian, yielding:

$$\begin{aligned}
J_{\text{TD}}(\theta,\theta) &= \nabla_\theta\delta(\theta,\theta),\\
&=\nabla_\theta\mathbb{E}_{\varsigma\sim\rho_\varsigma}\left[\left(r+\gamma Q_\theta(s',a')-Q_\theta(s,a)\right)\nabla_\theta Q_\theta(s,a)\right],\\
&=\mathbb{E}_{\varsigma\sim\rho_\varsigma}\left[\left(\gamma\nabla_\theta Q_\theta(s',a')-\nabla_\theta Q_\theta(s,a)\right)\nabla_\theta Q_\theta(s,a)\right.\\
&\qquad\left.+\left(r+\gamma Q_\theta(s',a')-Q_\theta(s,a)\right)\nabla^2_\theta Q_\theta(s,a)\right],\\
&=\mathbb{E}_{\varsigma\sim\rho_\varsigma}\left[\left(\gamma\nabla_\theta Q_\theta(s',a')-\nabla_\theta Q_\theta(s,a)\right)\nabla_\theta Q_\theta(s,a)\right.\\
&\qquad\left.+\left(\left(\mathcal{T}^\pi[Q_\theta](s,a)-Q_\theta(s,a)\right)\nabla^2_\theta Q_\theta(s,a)\right],
\end{aligned}$$

This leads us to the overall bound:

$$\sup \lambda\left(\bar{J}_{\mathrm{TD}}(\theta,\theta^{\star}\right) \leq \sup_{\theta} \sup \lambda\left(\mathbb{E}\left[(\mathcal{T}^{\pi}[Q_{\theta}]-Q_{\theta})\nabla_{\theta}^{2}Q_{\theta}\right]\right.$$
$$\left.+\mathbb{E}\left[\left(\gamma\mathbb{E}'[\nabla_{\theta}Q'_{\theta}]-\nabla_{\theta}Q_{\theta}\right)\nabla_{\theta}Q_{\theta}^{\top}\right]\right).$$

Even making the same assumption as in Section 5.3.1 of sampling on-policy in an ergodic MDP to show that $\theta^{\top}\mathbb{E}\left[\left(\gamma\mathbb{E}[\nabla_{\theta}Q'_{\theta}]-\nabla_{\theta}Q_{\theta}\right)\nabla_{\theta}Q_{\theta}^{\top}\right]\theta \leq (\gamma-1)\theta^{\top}\mathbb{E}\left[\nabla_{\theta}Q_{\theta}\nabla_{\theta}Q_{\theta}^{\top}\right]\theta < 0$, we cannot prove the negative definiteness of $J_{\mathrm{TD}}(\theta,\theta)$ required to satisfy assumption 5.4. This is because the matrix $\mathbb{E}\left[(\mathcal{T}^{\pi}[Q_{\theta}]-Q_{\theta})\nabla_{\theta}^{2}Q_{\theta}\right]$ can be arbitrarily positive definite depending on the MDP and choice of function approximator. Indeed, there exist counterexample MDPs with provably divergent nonlinear function approximators when sampling on-policy [Tsitsiklis and Van Roy, 1997].

5.4 Non-asymptotic Analysis

The asymptotic analysis in Section 5.3 shows that increasing k for PFPE does not affect the asymptotic strong convergence properties of the TD algorithm, implying that target networks do not stabilise TD if stepsizes tend to zero. We showed that the underlying reason for this was the deadly triad, which we formalised as adherence to assumption 5.4. However, hope still remains for convergence guarantees with the use of target networks, in a slightly different setting, which matches better with empirical practice and finite-time learning.

The reason why PFPE under a diminishing step-size regime is fundamentally governed by limiting ODE of TD(0) is because, as step sizes decrease, updates become increasingly close to TD updates. This is formalised by Lemma 5.2. On the other hand, intuitively, PFPE should be able to interpolate between FPE and TD(0). In order to prevent updates from converging to TD update steps, we now consider a regime in which we use a constant step-size, which invalidates the result in Lemma 5.2, and allow updates to look more like those conducted by FPE.

In this section, we conduct a non-asymptotic analysis, which allows us to obtain finite time error bounds while we investigate how the deadly triad can be broken by PFPE

when stepsizes *do not tend to zero*. With the intuition that, under the non-diminishing stepsize regime, PFPE updates should emulate those of FPE, we introduce the following assumption, which encodes our convergence condition for FPE.

Assumption 5.6 (FPE Stability)**.** *There exists a region $\mathcal{X}_{FPE}(\theta^\star)$ containing a fixed point $\theta^\star$, with $\theta_i \in \mathcal{X}_{FPE}(\theta^\star)$ for all $i \in \mathbb{N}$ such that* $\sup_{\theta,\theta' \in \mathcal{X}_{FPE}(\theta^\star)} \left\| \bar{H}(\theta', \theta^\star; \theta)^{-1} \bar{J}_\delta(\theta, \theta^\star; \theta^\star) \right\| < 1$.

This assumption may in general be either stronger or weaker than assumption 5.4, depending on the particular form of function approximation, data distribution, and MDP applicable to the problem at hand. Later, we will introduce regularisation which may introduce new fixed points, but can be used to guarantee convergence in all settings.

We begin by relating the parameter gap of the PFPE fitting updates to those of the FPE update. This will be used later to express this parameter gap in terms of the convergence conditions derived in Lemma 5.1 and Theorem 5.3. The following lemma formalises this for iterations within the inner optimisation problem.

Lemma 5.4. *Under assumption 5.2, for fitting iterations $i > 0$, the expected updates can be factored as:*

$$\mathbb{E}_{\rho_\varsigma}[\theta_{i+1} - \bar{\theta}_l^\star] = \left(I - \alpha_l \bar{H}(\theta_i, \bar{\theta}_l^\star; \bar{\theta}_l)\right)(\theta_i - \bar{\theta}_l^\star),$$
$$\mathbb{E}_{\rho_\varsigma}[\theta_{i+1} - \theta^\star] = \left(I - \alpha_l \bar{H}(\theta_i, \bar{\theta}_l^\star; \bar{\theta}_l)\right)(\theta_i - \bar{\theta}_l^\star) + \bar{\theta}_l^\star - \theta^\star.$$

and for iteration $i = 0$:

$$\mathbb{E}_{\rho_\varsigma}[\theta_1 - \bar{\theta}_l^\star] = (I + \alpha \bar{J}_{TD}(\bar{\theta}_l, \theta^\star))(\bar{\theta}_l - \theta^\star) + \theta^\star - \bar{\theta}_l^\star$$

Proof. By the definition of the expected update θ_{i+1}:

$$\mathbb{E}_{\rho_\varsigma}[\theta_{i+1} - \bar{\theta}_l^\star] = \theta_i - \bar{\theta}_l^\star + \alpha_l \delta(\theta_i, \bar{\theta}_l) - \alpha_l \underbrace{\delta(\bar{\theta}_l^\star, \bar{\theta}_l)}_{=0}.$$

Like in Lemma 5.1, let $\ell(t) \coloneqq \theta_i - t(\theta_i - \bar{\theta}_l^\star)$ define the line connecting θ_i to $\bar{\theta}_l^\star$. Using this notation, we re-write the expected update as:

$$\mathbb{E}_{\rho_\varsigma}\left[\theta_{i+1} - \bar{\theta}_l^\star\right] = \theta_i - \bar{\theta}_l^\star + \alpha_l \left(\delta(\theta = \ell(0), \bar{\theta}_l) - \delta(\theta = \ell(1), \bar{\theta}_l)\right).$$

Applying the fundamental theorem of calculus under assumption 5.2 and the chain rule yields our desired result:

$$\begin{aligned}
\mathbb{E}_{\rho_\varsigma}\left[\theta_{i+1} - \bar{\theta}_l^\star\right] &= \theta_i - \bar{\theta}_l^\star - \alpha_l \int_0^1 \partial_t \delta(\theta = \ell(t), \bar{\theta}_l) dt, \\
&= \theta_i - \bar{\theta}_l^\star - \alpha_l \int_0^1 \nabla_\theta \delta(\theta, \bar{\theta}_l)_{\theta=\ell(t)} \partial_t \ell(t) dt, \\
&= \theta_i - \bar{\theta}_l^\star + \alpha_l \left(\int_0^1 \nabla_\theta \delta(\theta, \bar{\theta}_l)_{\theta=\ell(t)} dt \right) (\theta_i - \bar{\theta}_l^\star), \\
&= \left(I - \alpha_l \bar{H}(\theta_i, \bar{\theta}_l^\star; \bar{\theta}_l) \right) (\theta_i - \bar{\theta}_l^\star).
\end{aligned}$$

Our second result follows immediately:

$$\begin{aligned}
\mathbb{E}_{\rho_\varsigma}[\theta_{i+1} - \theta^\star] &= \mathbb{E}_{\rho_\varsigma}[\theta_{i+1} - \bar{\theta}_l^\star] + \bar{\theta}_l^\star - \theta^\star, \\
&= \left(I - \alpha_l \bar{H}(\theta_i, \bar{\theta}_l^\star; \bar{\theta}_l) \right) (\theta_i - \bar{\theta}_l^\star) + \bar{\theta}_l^\star - \theta^\star.
\end{aligned}$$

While the first result applies to the initial update, in order to reveal the effect of $\bar{J}_{\text{TD}}$ on our update scheme, for the first update, it holds that:

$$\begin{aligned}
\mathbb{E}_{\rho_\varsigma}[\theta_1 - \bar{\theta}_l^\star] &= \mathbb{E}_{\rho_\varsigma}[\theta_1 - \theta^\star + \theta^\star - \bar{\theta}_l^\star], \\
&= \mathbb{E}_{\rho_\varsigma}[\theta_1 - \theta^\star] + \theta^\star - \bar{\theta}_l^\star.
\end{aligned}$$

By the definition of the expected update:

$$\mathbb{E}_{\rho_\varsigma}[\theta_1 - \theta^\star] = \bar{\theta}_l - \theta^\star + \alpha_l \delta(\bar{\theta}_l, \bar{\theta}_l) - \alpha_l \underbrace{\delta(\theta^\star, \theta^\star)}_{=0}.$$

Let $\ell(t) \coloneqq \bar{\theta}_l - t(\bar{\theta}_l - \theta^\star)$ define the line connecting $\bar{\theta}_l$ to $\theta^\star$. Using this notation we re-write the expected update as:

$$\mathbb{E}_{\rho_\varsigma}[\theta_1 - \theta^\star] = \bar{\theta}_l - \theta^\star + \alpha_l \left(\delta(\theta = \ell(0), \theta = \ell(t)) - \delta(\theta = \ell(1), \theta = \ell(1)) \right).$$

Applying the fundamental theorem of calculus under assumption 5.2 and the chain rule yields our desired result:

$$\begin{aligned}
\mathbb{E}_{\rho_\varsigma}\left[\theta_{i+1} - \bar{\theta}_l^\star\right] &= \bar{\theta}_l - \theta^\star - \alpha_l \int_0^1 \partial_t \delta(\theta = \ell(t), \theta = \ell(t)) dt, \\
&= \bar{\theta}_l - \theta^\star - \alpha_l \int_0^1 \nabla_\theta \delta(\theta, \theta)|_{\theta=\ell(t)} \partial_t \ell(t) dt, \\
&= \bar{\theta}_l - \theta^\star + \alpha_l \left(\int_0^1 \nabla_\theta \delta(\theta, \theta)|_{\theta=\ell(t)} dt \right) (\bar{\theta}_l - \theta^\star), \\
&= \left(I + \alpha_l \bar{J}_{\text{TD}}(\bar{\theta}_l^\star, \theta^\star) \right) (\theta_i - \bar{\theta}_l^\star).
\end{aligned}$$

□

With these factorisations, we are able to relate expected iterates of PFPE to iterates of FPE, alongside the difference between iterates and a fixed point.

Building on this, we now proceed to decompose the suboptimality of the parameters for a single timestep of the inner optimisation problem into the effect of the expected update plus the error induced by variance of the update. We will then manipulate this decomposition in order to reflect the functional form described in Lemma 5.4. The following lemma describes this decomposition.

Lemma 5.5. *Under assumption 5.2,*

$$\mathbb{E}_{\rho_{\varsigma_i}}\left[\|\theta_{i+1}-\theta^\star\|\right] \leq |1-\alpha_l\lambda_H^\star|\left\|\theta_i-\bar{\theta}_l^\star\right\| + \left\|\bar{\theta}_l^\star-\theta^\star\right\| + \alpha_l\sigma_\delta.$$

Proof. We start by bounding the expected norm term using Jensen's inequality: $\mathbb{E}_X[\sqrt{X^2}] \leq \sqrt{\mathbb{E}_X[X^2]}$.

$$\begin{aligned}
\mathbb{E}_{\rho_{\varsigma_i}}\left[\|\theta_{i+1}-\theta^\star\|\right] &\leq \sqrt{\mathbb{E}_{\rho_{\varsigma_i}}\left[\|\theta_{i+1}-\theta^\star\|^2\right]}, \\
&\leq \sqrt{\left\|\mathbb{E}_{\rho_{\varsigma_i}}\left[\theta_{i+1}-\theta^\star\right]\right\|^2 + \mathbb{V}_{\rho_{\varsigma_i}}\left[\theta_{i+1}-\theta^\star\right]}, \\
&\leq \sqrt{\left\|\mathbb{E}_{\rho_{\varsigma_i}}\left[\theta_{i+1}-\theta^\star\right]\right\|^2 + \mathbb{V}_{\rho_{\varsigma_i}}\left[\theta_{i+1}\right]}, \\
&\leq \left\|\mathbb{E}_{\rho_{\varsigma_i}}\left[\theta_{i+1}-\theta^\star\right]\right\| + \sqrt{\mathbb{V}_{\rho_{\varsigma_i}}\left[\theta_{i+1}\right]}, \qquad (5.12)
\end{aligned}$$

where we applied the triangle inequality to derive the final line. We bound the variance term by substituting $\theta_{i+1} = \theta_i + \alpha_l\delta(\theta_i, \bar{\theta}_l, \varsigma_i)$:

$$\begin{aligned}
\mathbb{V}_{\rho_{\varsigma_i}}\left[\theta_{i+1}\right] &= (\alpha_l)^2\mathbb{E}_{\rho_{\varsigma_i}}\left[\left\|\delta(\theta_i,\bar{\theta}_l,\varsigma_i) - \mathbb{E}_{\rho_{\varsigma_i}}\left[\delta(\theta_i,\bar{\theta}_l,\varsigma_i)\right]\right\|^2\right], \\
&= (\alpha_l)^2\mathbb{V}_{\rho_{\varsigma_i}}\left[\delta(\theta_i,\bar{\theta}_l,\varsigma_i)\right], \\
&\leq (\alpha_l\sigma_\delta)^2. \qquad (5.13)
\end{aligned}$$

Applying Lemma 5.4 to the combination of Eqs. (5.12) and (5.13) and using the triangle

inequality yields our desired result:

$$\begin{aligned}
\mathbb{E}_{\rho_{s_i}}\left[\|\theta_{i+1}-\theta^\star\|\right] &\leq \left\|\left(I-\alpha_l\bar{H}(\theta_i,\bar{\theta}_l^\star;\bar{\theta}_l)\right)(\theta_i-\bar{\theta}_l^\star)+\left(\bar{\theta}_l^\star-\theta^\star\right)\right\|+\alpha_l\sigma_\delta,\\
&\leq \left\|I-\alpha_l\bar{H}(\theta_i,\bar{\theta}_l^\star;\bar{\theta}_l)\right\|\left\|\theta_i-\bar{\theta}_l^\star\right\|+\left\|\bar{\theta}_l^\star-\theta^\star\right\|+\alpha_l\sigma_\delta,\\
&\leq \sup_{\theta_i,\bar{\theta}_l^\star,\bar{\theta}_l}\left\|I-\alpha_l\bar{H}(\theta_i,\bar{\theta}_l^\star;\bar{\theta}_l)\right\|\left\|\theta_i-\bar{\theta}_l^\star\right\|+\left\|\bar{\theta}_l^\star-\theta^\star\right\|+\alpha_l\sigma_\delta,\\
&\leq |1-\alpha_l\lambda_H^\star|\left\|\theta_i-\bar{\theta}_l^\star\right\|+\left\|\bar{\theta}_l^\star-\theta^\star\right\|+\alpha_l\sigma_\delta.
\end{aligned}$$

□

We can now use this single-step decomposition in order to bound the conditioning of the overall k-step update, $g^k(\bar{\theta}_l,\mathcal{D},\alpha_l)$, as described in Eq. (5.9). To do so, we introduce the notion of a *condition function*, which we use in the following lemma, in order to bound the stability of the overall update.

Definition 5.1 (Condition Function)**.** *For a subset $\mathcal{X}(\theta^\star)\subseteq\theta$, with fixed point $\theta^\star$, such that $\theta_i\in\mathcal{X}(\theta^\star)$ for all $i\geq 0$, let*

$$\begin{aligned}
\lambda_H^\star(\alpha_l) &\coloneqq \underset{\theta,\theta',\theta''\in\mathcal{X}(\theta^\star),\lambda'\in\lambda(\bar{H}(\theta,\theta';\theta''))}{\arg\sup}\left|1-\alpha_l\lambda'\right|,\\
\|J_{FPE}\| &\coloneqq \sup_{\theta,\theta'\in\mathcal{X}(\theta^\star)}\left\|\bar{H}(\theta',\theta^\star;\theta)^{-1}\bar{J}_\delta(\theta,\theta^\star;\theta')\right\|,\\
\|J_{TD}^\star\| &\coloneqq \sup_{\theta\in\mathcal{X}(\theta^\star)}\|I+\alpha\bar{J}_{TD}(\theta,\theta^\star)\|.
\end{aligned}$$

and define the condition function as:

$$\mathcal{C}(\alpha_l,k)\coloneqq|1-\alpha_l\lambda_H^\star(\alpha_l)|^{k-1}\|J_{TD}^\star\|+\left(1+|1-\alpha_l\lambda_H^\star(\alpha_l)|^{k-1}\right)\|J_{FPE}\|.$$

We can see that this condition function is given as a weighted combination of the key matrices introduced in Section 5.2.1, and crucially depends on k, which reveals how additional steps of optimisation without diminishing step sizes trades off the conditioning of the TD(0) update with the conditioning of the FPE update.

Using the condition function, we will now bound the overall error of the k-step update.

Theorem 5.6. *Define*

$$\sigma_k \coloneqq \left(1 - |1 - \alpha_l \lambda_H^\star|^k\right) \frac{\sigma_\delta}{\lambda_H^\star}, \tag{5.14}$$

Let Assumptions 5.1 and 5.2 hold, then:

$$\mathbb{E}\left[\|\bar{\theta}_{l+1} - \theta^\star\|\right] \leq \mathcal{C}(\alpha_l, k)\mathbb{E}\left[\|\bar{\theta}_l - \theta^\star\|\right] + \alpha_l \sigma_k. \tag{5.15}$$

Proof. Let $\{\theta_i\}_{i=0}^k$ denote the intermediate function approximation parameters between target parameter updates $\bar{\theta}_{l+1}$ and $\bar{\theta}_l$, with $\theta_0 = \bar{\theta}_l$ and $\theta_k = \bar{\theta}_{l+1}$. We define the set of samples for the inner optimisation problem up to step i as: $\mathcal{D}_i \coloneqq \{\varsigma_j\}_{j=0}^i$ with distribution $P_{\mathcal{D}_i}$, where sample ς_j has distribution ρ_{ς_j}. Using this notation, we now proceed to show:

$$\mathbb{E}_{P_{\mathcal{D}_{k-1}}}\left[\|\theta_k - \theta^\star\|\right] \leq \mathcal{C}(\alpha_l, k)\|\theta_0 - \theta^\star\| + \alpha_l \sigma_k. \tag{5.16}$$

Using assumption 5.1 to break apart the expectation, and then applying Lemma 5.5 leads to:

$$\begin{aligned}
\mathbb{E}_{P_{\mathcal{D}_{k-1}}}\left[\|\theta_k - \theta^\star\|\right] &= \mathbb{E}_{P_{\mathcal{D}_{k-2}}}\left[\mathbb{E}_{\rho_{\varsigma_{k-1}}}\left[\|\theta_k - \theta^\star\|\right]\right], \\
&\leq \mathbb{E}_{P_{\mathcal{D}_{k-2}}}\left[|1 - \alpha_l \lambda_H^\star| \left\|\theta_{k-1} - \bar{\theta}_l^\star\right\| + \left\|\bar{\theta}_l^\star - \theta^\star\right\| + \alpha_l \sigma_\delta\right], \\
&\leq |1 - \alpha_l \lambda_H^\star| \, \mathbb{E}_{P_{\mathcal{D}_{k-2}}}\left[\left\|\theta_{k-1} - \bar{\theta}_l^\star\right\|\right] + \left\|\bar{\theta}_l^\star - \theta^\star\right\| + \alpha_l \sigma_\delta, \\
&\leq |1 - \alpha_l \lambda_H^\star| \, \mathbb{E}_{P_{\mathcal{D}_{k-3}}}\left[\mathbb{E}_{\rho_{\varsigma_{k-2}}}\left[\left\|\theta_{k-1} - \bar{\theta}_l^\star\right\|\right]\right] + \left\|\bar{\theta}_l^\star - \theta^\star\right\| + \alpha_l \sigma_\delta.
\end{aligned} \tag{5.17}$$

Applying Jensen's inequality as is done in the derivation of Eq. (5.12) and using Eq. (5.13) from Lemma 5.5 to decompose the inner expectation, then applying Lemma 5.4 yields:

$$\begin{aligned}
\mathbb{E}_{\rho_{\varsigma_{k-2}}}\left[\left\|\theta_{k-1} - \bar{\theta}_l^\star\right\|\right] &\leq \left\|\mathbb{E}_{\rho_{\varsigma_{k-2}}}\left[\theta_{k-1} - \bar{\theta}_l^\star\right]\right\| + \alpha_l \sigma_\delta, \\
&\leq \left\|\left(I - \alpha_l \bar{H}(\theta_{k-2}, \bar{\theta}_l^\star; \bar{\theta}_l)\right)(\theta_{k-2} - \bar{\theta}_l^\star)\right\| + \alpha_l \sigma_\delta, \\
&\leq \sup_{\theta_{k-2}, \bar{\theta}_l^\star, \bar{\theta}_l} \left\|I - \alpha_l \bar{H}(\theta_{k-2}, \bar{\theta}_l^\star; \bar{\theta}_l)\right\| \left\|\theta_{k-2} - \bar{\theta}_l^\star\right\| + \alpha_l \sigma_\delta, \\
&\leq |I - \alpha_l \lambda_H^\star| \left\|\theta_{k-2} - \bar{\theta}_l^\star\right\| + \alpha_l \sigma_\delta.
\end{aligned} \tag{5.18}$$

Recursively applying Eq. (5.18) to Eq. (5.17) $k-1$ times leads to:

$$\begin{aligned}\mathbb{E}_{P_{\mathcal{D}_{k-1}}}\left[\|\theta_k-\theta^\star\|\right] &\leq \mathbb{E}_{\rho_{\varsigma_0}}\left[\left|1-\alpha_l\lambda_H^\star\right|^{k-1}\left\|\theta_1-\bar{\theta}_l^\star\right\|\right] \\ &\qquad + \left\|\bar{\theta}_l^\star-\theta^\star\right\| + \sum_{i=0}^{k-2}\left|1-\alpha_l\lambda_H^\star\right|^i\alpha_l\sigma_\delta, \\ &\leq \left|1-\alpha_l\lambda_H^\star\right|^{k-1}\mathbb{E}_{\rho_{\varsigma_0}}\left[\|\theta_1-\theta_l^\star\|\right] \\ &\qquad + \left\|\bar{\theta}_l^\star-\theta^\star\right\| + \sum_{i=0}^{k-2}\left|1-\alpha_l\lambda_H^\star\right|^i\alpha_l\sigma_\delta. \end{aligned} \tag{5.19}$$

Again, applying Jensen's inequality as in Eq. (5.12) and using Eq. (5.13) and Lemma 5.4, can break apart the expectation term as

$$\begin{aligned}\mathbb{E}_{\rho_{\varsigma_0}}\left[\|\theta_1-\theta_l^\star\|\right] &\leq \left\|\mathbb{E}_{\rho_{\varsigma_0}}\left[\theta_1-\theta_l^\star\right]\right\| + \alpha_l\sigma_\delta, \\ &\leq \left\|(I+\alpha\bar{J}_{\text{TD}}(\bar{\theta}_l,\theta^\star))(\bar{\theta}_l-\theta^\star)+\theta^\star-\bar{\theta}_l^\star\right\| + \alpha_l\sigma_\delta, \\ &\leq \left\|I+\alpha\bar{J}_{\text{TD}}(\bar{\theta}_l,\theta^\star)\right\|\|\bar{\theta}_l-\theta^\star\| + \|\bar{\theta}_l^\star-\theta^\star\| + \alpha_l\sigma_\delta, \\ &\leq \left\|\bar{J}_{\text{TD}}^\star\right\|\|\bar{\theta}_l-\theta^\star\| + \|\theta_l^\star-\bar{\theta}_l\| + \alpha_l\sigma_\delta. \end{aligned}$$

Substituting this into Eq. (5.19):

$$\begin{aligned}\mathbb{E}_{P_{\mathcal{D}_{k-1}}}\left[\|\theta_k-\theta^\star\|\right] &\leq \left|1-\alpha_l\lambda_H^\star\right|^{k-1}\left\|\bar{J}_{\text{TD}}^\star\right\|\left\|\bar{\theta}_l-\theta^\star\right\| \\ &\qquad + \left(1+\left|1-\alpha_l\lambda_H^\star\right|^{k-1}\right)\left\|\bar{\theta}_l^\star-\theta^\star\right\| + \sum_{i=0}^{k-1}\left|1-\alpha_l\lambda_H^\star\right|^i\alpha_l\sigma_\delta. \\ &\leq \left|1-\alpha_l\lambda_H^\star\right|^{k-1}\left\|\bar{J}_{\text{TD}}^\star\right\|\left\|\bar{\theta}_l-\theta^\star\right\| \\ &\qquad + \left(1+\left|1-\alpha_l\lambda_H^\star\right|^{k-1}\right)\left\|\bar{\theta}_l^\star-\theta^\star\right\| + \frac{1-\left|1-\alpha_l\lambda_H^\star\right|^k}{1-\left|1-\alpha_l\lambda_H^\star\right|}\alpha_l\sigma_\delta, \\ &\leq \left|1-\alpha_l\lambda_H^\star\right|^{k-1}\left\|\bar{J}_{\text{TD}}^\star\right\|\left\|\bar{\theta}_l-\theta^\star\right\| \\ &\qquad + \left(1+\left|1-\alpha_l\lambda_H^\star\right|^{k-1}\right)\left\|\bar{\theta}_l^\star-\theta^\star\right\| + \left(1-\left|1-\alpha_l\lambda_H^\star\right|^k\right)\frac{\sigma_\delta}{\lambda_H^\star}, \\ &\leq \left|1-\alpha_l\lambda_H^\star\right|^{k-1}\left\|\bar{J}_{\text{TD}}^\star\right\|\left\|\bar{\theta}_l-\theta^\star\right\| + \left(1+\left|1-\alpha_l\lambda_H^\star\right|^{k-1}\right)\left\|\bar{\theta}_l^\star-\theta^\star\right\| + \sigma_k. \end{aligned}$$

This is nearly our end result, but the second term has been expressed in terms of the FPE update. We roll this backward through one step of FPE using Lemma 5.1 in order to

derive the effect of the k step update on the target network parameters.

$$\begin{aligned}
\mathbb{E}_{P_{\mathcal{D}_{k-1}}}\left[\|\theta_k - \theta^\star\|\right] &\leq |1-\alpha_l\lambda_H^\star|^{k-1}\left\|\bar{J}_{\text{TD}}^\star\right\|\left\|\bar{\theta}_l - \theta^\star\right\| \\
&\quad + \left(1+|1-\alpha_l\lambda_H^\star|^{k-1}\right)\left\|\bar{H}(\bar{\theta}_l^\star,\theta^\star;\bar{\theta}_l)^{-1}\bar{J}_\delta(\bar{\theta}_l,\theta^\star;\theta^\star)(\bar{\theta}_l-\theta^\star)\right\| + \sigma_k, \\
&\leq |1-\alpha_l\lambda_H^\star|^{k-1}\left\|\bar{J}_{\text{TD}}^\star\right\|\left\|\bar{\theta}_l - \theta^\star\right\| \\
&\quad + \left(1+|1-\alpha_l\lambda_H^\star|^{k-1}\right)\left\|\bar{H}(\bar{\theta}_l^\star,\theta^\star;\bar{\theta}_l)^{-1}\bar{J}_\delta(\bar{\theta}_l,\theta^\star;\theta^\star)\right\|\left\|\bar{\theta}_l-\theta^\star\right\| + \sigma_k, \\
&\leq |1-\alpha_l\lambda_H^\star|^{k-1}\left\|\bar{J}_{\text{TD}}^\star\right\|\left\|\bar{\theta}_l - \theta^\star\right\| \\
&\quad + \left(1+|1-\alpha_l\lambda_H^\star|^{k-1}\right)\left\|\bar{J}_{\text{FPE}}^\star\right\|\left\|\bar{\theta}_l-\theta^\star\right\| + \sigma_k, \\
&\leq \mathcal{C}(\alpha_l,k)\left\|\bar{\theta}_l-\theta^\star\right\| + \sigma_k.
\end{aligned}$$

□

Equipped with this result, we see that the expected update will be a contraction if $\mathcal{C}(\alpha_l, k) < 1$, leading us to expect convergence in this setting. We introduce the following assumption in order to codify this behaviour.

Assumption 5.7 (Contraction Region)**.** *We assume that* $\mathcal{C}(\alpha, k) \leq c < 1$ *over* $\mathcal{X}_{FPE}(\theta^\star)$.

Later we will demonstrate that if assumption 5.6 holds, assumption 5.7 can be ensured to hold as well, leading PFPE to converge whenever FPE converges. We will also show that with regularisation, it is possible to ensure convergence, though the fixed point will ultimately be modified.

With this, the proof of finite-time convergence of PFPE follows simply, which we demonstrate as follows:

Corollary 5.6.1. *Let assumptions 5.1, 5.2 and 5.7 hold. For a fixed stepsize* $\alpha_l = \alpha > 0$*,*

$$\mathbb{E}\left[\|\bar{\theta}_l - \theta^\star\|\right] \leq \frac{\alpha\sigma_k}{1-c} + \exp(-l(1-c))\left(\left\|\bar{\theta}_0-\theta^\star\right\| - \frac{\sigma_k}{1-c}\right).$$

Proof. We start by applying Theorem 5.6:

$$\mathbb{E}\left[\|\bar{\theta}_l - \theta^\star\|\right] \leq \mathcal{C}(\alpha_l,k)\mathbb{E}\left[\|\bar{\theta}_{l-1}-\theta^\star\|\right] + \alpha_l\sigma_k.$$

As $\bar{\theta}_l \in \mathcal{X}_{\text{FPE}}(\theta^\star)$ for all $l \geq 0$, there exists a positive $c < 1$ under assumption 5.7 such that $\mathcal{C}(\alpha_l, k) \leq c$, hence:

$$\mathbb{E}\left[\|\bar{\theta}_l - \theta^\star\|\right] \leq c\mathbb{E}\left[\|\bar{\theta}_{l-1} - \theta^\star\|\right] + \alpha_l \sigma_k. \tag{5.20}$$

Now, for a fixed constant stepsize $\alpha_l = \alpha$, we can apply Eq. (5.20) l times, yielding:

$$\begin{aligned}
\mathbb{E}\left[\|\bar{\theta}_l - \theta^\star\|\right] &\leq c^l \left\|\bar{\theta}_0 - \theta^\star\right\| + \alpha\sigma_k \sum_{i=0}^{l-1} c^i, \\
&= c^l \left\|\bar{\theta}_0 - \theta^\star\right\| + \alpha\sigma_k \frac{1 - c^l}{1 - c} \\
&= c^l \left(\left\|\bar{\theta}_0 - \theta^\star\right\| - \frac{\alpha\sigma_k}{1-c}\right) + \frac{\alpha\sigma_k}{1-c}, \\
&= (1 - (1-c))^l \left(\left\|\bar{\theta}_0 - \theta^\star\right\| - \frac{\alpha\sigma_k}{1-c}\right) + \frac{\alpha\sigma_k}{1-c}.
\end{aligned}$$

Now we apply the bound $1 - x \leq \exp(-x)$, yielding our desired result:

$$\begin{aligned}
\mathbb{E}\left[\|\bar{\theta}_l - \theta^\star\|\right] &\leq \exp(-(1-c))^l \left(\left\|\bar{\theta}_0 - \theta^\star\right\| - \frac{(\alpha\sigma_k)}{1-c}\right) + \frac{\alpha\sigma_k}{1-c}, \\
&= \exp(-l(1-c)) \left(\left\|\bar{\theta}_0 - \theta^\star\right\| - \frac{\alpha\sigma_k}{1-c}\right) + \frac{\alpha\sigma_k}{1-c}.
\end{aligned}$$

□

Corollary 5.6.1 is a key result of this chapter. It demonstrates geometric decay of errors in l, to a ball of fixed radius $\frac{\alpha\sigma_k}{1-c}$. This is analogous to related work in stochastic gradient descent [Bottou et al., 2018], and matches the intuition that, without decaying stepsize, variance in the updates means that convergence to a precise fixed point does not occur, but instead parameters will oscillate around this point.

We note that the radius of this ball is decreasing in α. This supports the use of a hybrid approach, wherein a fixed step size is used until iterates are no longer improving and then reducing step size and repeating to decrease the radius of the ball of convergence whilst maintaining k as small as possible, until the desired level of precision is reached.

Corollary 5.6.1 shows that if assumption 5.7 holds, then we can expect PFPE to converge to a reasonably good solution, but this point is moot if such a region of contraction does not exist. The remainder of this section is dedicated to analysis of the condition

function, in order to demonstrate that for suitable choices of α and k, we will always occupy a region satisfying assumption 5.7 whenever FPE converges, thus also ensuring nondivergence of PFPE.

5.4.1 Analysing the Condition Function

We now investigate key properties of Eq. (5.14) to understand how target parameters can lead to convergence when classic TD methods fail. If $\bar{J}_{\text{TD}}(\theta, \theta^\star)$ is positive definite, TD is provably divergent, however our analysis reveals that even in this case, there may be values of k and α_l for which PFPE does converge.

Property 1: Lower bound $\mathcal{C}(\alpha_l, k) \geq \left\| \bar{J}^\star_{\text{FPE}} \right\|, \qquad \mathcal{C}(\alpha_l, k) \geq |1-\alpha_l \lambda^\star_H(\alpha_l)|^{k-1} \, \|J^\star_{\text{TD}}\|.$

We first investigate the conditions for which our choice of function approximators can never be used to prove convergence. Our condition function implies that if TD(0) diverges, we cannot prove convergence for any $\lambda^\star_H \leq 0$ or $\lambda^\star_H \geq \frac{2}{\alpha_l}$ as repeated applications of $|1 - \alpha_l \lambda^\star_H|$ do not reduce the effect the ill-conditioning of $\bar{J}_{\text{TD}}(\theta, \theta^\star)$. We formalise this in the following regularity assumption:

Assumption 5.8 (Eigenvalue Regularity Assumption)**.** *Given a region $\mathcal{X} \subseteq \Theta$, for all $\theta, \theta' \in \mathcal{X}$ there exists $0 < \lambda^{\min}$ and $\lambda^{\max} < \infty$ such that $\lambda^{\min} \leq \lambda(\nabla^2_\theta \mathcal{L}(\theta; \theta')) \leq \lambda^{\max}$.*

We now propose two simple fixes to avoid this issue. Recall that $\lambda^\star_H$ is an eigenvalue of the Hessian of a loss. If $\lambda^\star_H$ was negative, this would imply that the Hessian is not positive semidefinite for all θ in the region of interest; hence we cannot prove convergence of stochastic gradient descent on the loss $\mathcal{L}(\theta; \bar{\theta}_l)$, let alone the full PFPE algorithm. To remedy this problem, the eigenvalues of the matrix can be increased using regularisation, which we will describe at the end of this section. On the other hand, if $\lambda^\star_H \geq \frac{2}{\alpha_l}$, then the conditioning of the Hessian matrix is ill-suited to the chosen step-size, and an easy remedy is to decrease α_l. Our bound also shows that the condition function is lower bounded by $\|J^\star_{\text{FPE}}\|$, and so if assumption 5.6 does not hold, then convergence of PFPE is not provable. If this is the case, then again, regularisation can be applied in order to

ensure that the property holds, but will lead to convergence to a regularised fixed point, rather than the true fixed point.

Property 2: Monotonicity For $|1 - \alpha_l \lambda_H^\star| < 1, \mathcal{C}(\alpha_l, k) \leq \mathcal{C}(\alpha_l, k')$ for $k \leq k'$.

The monotonicity property ensures that $|1 - \alpha_l \lambda_H^\star| < 1$ defines the interval of Hessian eigenvalues for which there is a regime in which we can increase k in order to ensure PFPE updates are a contraction mapping. This suggests that a key role of the target network is to help mitigate the effects of the ill-conditioning of the TD Jacobian when using fixed step sizes. We now investigate how decreasing stepsizes and increasing the number of PFPE steps affect the conditioning of PFPE, which validates this hypothesis.

Property 3: Limits For any $k < \infty$, $\lim_{\alpha_l \to 0} \mathcal{C}(\alpha_l, k) = \left\| \bar{J}^\star_{\text{TD}} \right\| + 2 \left\| \bar{J}^\star_{\text{FPE}} \right\|$. For any $0 < \alpha_l < \frac{2}{\lambda_H^\star}$, $\lim_{k \to \infty} \mathcal{C}(\alpha_l, k) = \left\| \bar{J}^\star_{\text{FPE}} \right\|$.

The first limit illustrates the effects of a diminishing stepsize sequence, confirming our bound is consistent with the results of the previous section that increasing k does not improve the convergence properties of PFPE if stepsizes tend to zero and PFPE only stabilises TD for $0 < \alpha_l$. By taking the limit $k \to \infty$, we compliment our monotonicity result, obtaining a bound for how much we can improve on the stability of TD by increasing k. As expected, in the limit of $k \to \infty$, the condition function tends to $\| J^\star_{\text{FPE}} \|$. Through this insight, we interpret PFPE as mixing FPE and TD updates according the coefficient $|1 - \alpha_l \lambda_H^\star|^{k-1}$: for $k = 1$, PFPE uses only TD updates and in the limit $k \to \infty$, PFPE recovers the FPE update.

5.4.2 Breaking the Deadly Triad

We now combine all properties presented in this section into our main result, proving that through suitable regularisation and choice of α_l and k, PFPE breaks TD's deadly triad described in Section 5.3.1:

Theorem 5.7. *Let assumptions 5.6 and 5.8 hold over $\mathcal{X}_{FPE}(\theta^\star)$ from Definition 5.1. For any $\frac{1}{\alpha_l} > \frac{\lambda^{\min}+\lambda^{\max}}{2}$ such that $\alpha_l > 0$, any*

$$k > 1 + \frac{\log(1 - \|\bar{J}^\star_{FPE}\|) - \log(\|\bar{J}^\star_{TD}\| + \|\bar{J}^\star_{FPE}\|)}{\log(1 - \alpha\lambda^{min})},$$

ensures that $\mathcal{X}_{FPE}(\theta^\star)$ is a region of contraction satisfying assumption 5.7.

Proof. Now, as $|1 - \alpha_l\lambda'|$ is a symmetric function of λ with a minima at $\lambda = \frac{1}{\alpha_l}$ and $\frac{\lambda^{\min}+\lambda^{\max}}{2}$ is the mid point of $\lambda^{\min}$ and $\lambda^{\max}$, it follows:

$$\lambda^\star_H \coloneqq \sup_{\theta,\theta' \in \mathcal{X}_{\text{FPE}}(\theta^\star)} \arg\sup_{\lambda' \in \lambda(\nabla^2_\theta \mathcal{L}(\theta,\theta'))} |1 - \alpha_l\lambda'| = \lambda^{\min}.$$

Now,

$$\alpha_l < \frac{2}{\lambda^{\min} + \lambda^{\max}} \implies \lambda^\star_H \leq \frac{2}{\alpha_l} \implies |1 - \alpha_l\lambda^\star_H| < 1,$$

hence

$$\begin{aligned}\lim_{k\to\infty} \mathcal{C}(\alpha_l, k) &= \lim_{k\to\infty} \left|1 - \alpha_l\lambda^{\min}\right|^{k-1} \|J^\star_{\text{TD}}\| + \lim_{k\to\infty} \left(1 + \left|1 - \alpha_l\lambda^{\min}\right|^{k-1}\right) \left\|\bar{J}^\star_{\text{FPE}}\right\| \\ &= \left\|\bar{J}^\star_{\text{FPE}}\right\| < 1.\end{aligned}$$

Let $\left\|\bar{J}^\star_{\text{FPE}}\right\| = 1 - \epsilon$ where $0 < \epsilon < 1$. From the definition of a limit, this implies that for ϵ there exists some finite k' such that whenever $k > k'$:

$$\left|\mathcal{C}(\alpha_l, k) - \left\|\bar{J}^\star_{\text{FPE}}\right\|\right| < \epsilon \implies |\mathcal{C}(\alpha_l, k) - (1 - \epsilon)| < \epsilon \implies \mathcal{C}(\alpha_l, k) < 1,$$

as required. To find the value of k for which $\mathcal{C}(\alpha_l, k) < 1$, we set $\mathcal{C}(\alpha_l, k) = 1$ and solve:

$$\begin{aligned}1 &= \left|1 - \alpha_l\lambda^{\min}\right|^{k-1} \left\|\bar{J}^\star_{\text{TD}}\right\| + \left(1 + \left|1 - \alpha_l\lambda^{\min}\right|^{k-1}\right) \left\|\bar{J}^\star_{\text{FPE}}\right\|, \\ &\implies \left|1 - \alpha_l\lambda^{\min}\right|^{k-1} = \frac{1 - \left\|\bar{J}^\star_{\text{FPE}}\right\|}{\|\bar{J}^\star_{\text{TD}}\| + \left\|\bar{J}^\star_{\text{FPE}}\right\|}, \\ \implies (k-1)\log(\left|1 - \alpha_l\lambda^{\min}\right|) &= \log(1 - \left\|\bar{J}^\star_{\text{FPE}}\right\|) - \log(\|\bar{J}^\star_{\text{TD}}\| + \left\|\bar{J}^\star_{\text{FPE}}\right\|), \\ \implies k &= 1 + \frac{\log(1 - \|\bar{J}^\star_{\text{FPE}}\|) - \log(\|\bar{J}^\star_{\text{TD}}\| + \|\bar{J}^\star_{\text{FPE}}\|)}{\log(1 - \alpha\lambda^{\min})}.\end{aligned}$$

□

Theorem 5.7 demonstrates that appropriate values of α_l and k can be found by treating them as hyperparameters, decreasing α_l and increasing k until the algorithm is stable. The key insight of Theorem 5.7 is that even when TD is unstable due to $1 < \|I + \alpha_l \bar{J}_{\text{TD}}(\bar{\theta}_l, \theta^\star)\|$, if FPE converges, there exists a finite k such that $\mathcal{C}(\alpha_l, k) < 1$ and hence PFPE is stable, leading to a convergent algorithm without need for perfect fitting of the projected Bellman operator. We illustrate this phenomenon with a sketch in Fig. 5.1, demonstrating that increasing k ensures PFPE is provably convergent in regimes where TD cannot be proved to converge.

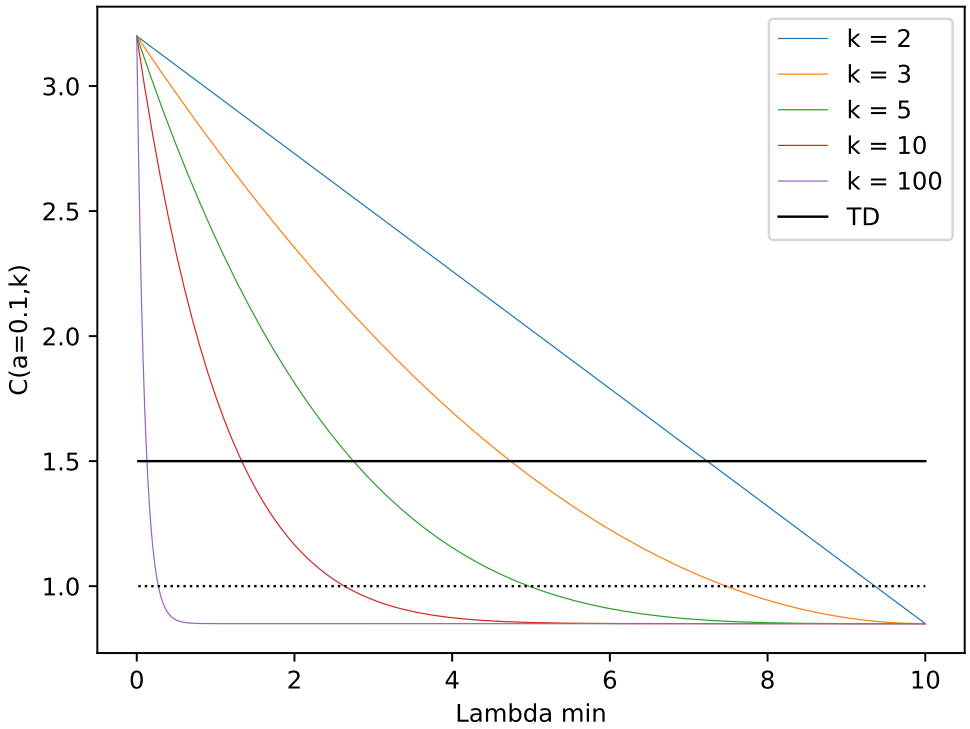


Figure 5.1: We plot $\mathcal{C}(\alpha = 0.1, k)$ for $\|\bar{J}^\star_{\text{FPE}}\| = 0.85$ and $\|\bar{J}^\star_{\text{TD}}\| \leq 1.5$ with increasing k as a function of $\lambda_{\min}$.

The key insight of our analysis is that, unlike in TD where stability can only be proved if the matrix $\bar{J}_\delta(\theta, \theta^\star; \theta^\star) - \bar{H}(\theta, \theta^\star; \theta)$ is negative definite, with suitable regularisation, the stability of PFPE can be determined by tuning α_l and k, alongside the stability of FPE. The choice of α_l and k thus becomes a trade-off between maintaining a fast rate of convergence and reducing the residual variance $(\alpha_l \sigma_k)$ in Eq. (5.14).

5.4.3 Convergence of TD(0) and FPE

It is well known that in the linear case, convergence of TD implies convergence of FPE [Szepesvári, 2010], though a converse result does not exist in the literature. This suggests that there are settings in which FPE converges but TD does not, although this remains to be formally demonstrated in future work. In the nonlinear setting, this relationship becomes less clear, as the Hessian is no longer constant across updates. This means that it may be possible to craft function approximation schemes or MDPs for which TD converges, but FPE does not. Without additional assumptions on the form of function approximation used, sampling regime, or MDP, we cannot guarantee that FPE converges as long as TD does. In order to maintain minimal assumptions in this work, we leave analysis of these convergence criteria to future work.

Instead, we introduce a regularisation scheme which can be used to ensure convergence of PFPE under all conditions.

5.4.4 Stabilising FPE with Regularisation

We now prove that assumption 5.6 can always be satisfied using regularisation schemes. To do so, we introduce the following regularised TD vector:

$$\delta_{\text{Reg}}(\theta, \theta') = \delta(\theta, \theta') + \rho(\theta, \theta'), \tag{5.21}$$

where $\rho(\theta, \theta')$ is a regularisation term. A simple choice to ensure assumption 5.6 holds is the $\rho(\theta, \theta') = -\eta\theta - \mu(\theta - \theta')$, where η and μ control the degree of regularisation. This scheme is equivalent to adding ℓ_2-regularisation to the loss $\mathcal{L}(\theta; \theta')$. Of couse, this will lead to a new fixed point, as the condition $\delta_{\text{Reg}}(\theta^\star, \theta^\star) = 0$ does not satisfy $\delta(\theta^\star, \theta^\star) = 0$ unless $\theta^\star = 0$.

We now prove that FPE can be stabilised using the regularised update by tuning η, μ:

Proposition 1. *Using the regularised TD vector:*

$$\delta_{Reg}(\theta, \theta') = \delta(\theta, \theta') - \eta\theta - \mu(\theta - \theta') \tag{5.22}$$

the path-mean Jacobians are:

$$\bar{H}_{Reg}(\theta, \theta^\star; \bar{\theta}_l) = \bar{H}(\theta, \theta^\star; \bar{\theta}_l) + (\mu + \eta)I,$$
$$\bar{J}_{\delta,Reg}(\theta, \theta^\star; \bar{\theta}_l) = \bar{J}_\delta(\theta, \theta^\star; \bar{\theta}_l) + \mu I,$$

There exists a finite $\eta > 0, \mu > 0$ such that the regularised version of assumption 5.6 is satisfied.

Proof. Taking derivatives of $\delta_{\text{Reg}}(\theta, \theta')$:

$$\begin{aligned}
-\nabla_\theta \delta_{\text{Reg}}(\theta, \theta') &= -\nabla_\theta \delta(\theta, \theta') + \mathbb{I}(\theta \neq \theta')\eta + I\mu, \\
\implies \bar{H}_{\text{Reg}}(\theta, \theta^\star; \bar{\theta}_l) &= -\int_0^1 \nabla_{\theta'} \delta_{\text{Reg}}(\theta' = \theta - t(\theta - \theta^\star), \bar{\theta}_l) dt = \bar{H}(\theta, \theta^\star; \bar{\theta}_l) + (\mu + \eta)I, \\
\nabla_{\theta'} \delta_{\text{Reg}}(\theta, \theta') &= \nabla_{\theta'} \delta(\theta, \theta') + \mu I, \\
\implies \bar{J}_{\delta,\text{Reg}}(\theta, \theta^\star; \bar{\theta}_l) &= \int_0^1 \nabla_{\theta'} \delta_{\text{Reg}}(\bar{\theta}_l, \theta' = \theta - t(\theta - \theta^\star)) dt = \bar{J}_\delta(\theta, \theta^\star; \bar{\theta}_l) + \mu I.
\end{aligned}$$

Without loss of generality, assume $\mu = na$ and $\eta = nb$ for some $0 < a, b$. Hence:

$$\left\| \bar{H}_{\text{Reg}}(\theta', \theta^\star; \theta)^{-1} \bar{J}_{\delta,\text{Reg}}(\theta, \theta^\star; \theta^\star) \right\| = \left\| (\bar{H}(\theta', \theta^\star; \theta) + n(a+b)I)^{-1} (\bar{J}_\delta(\theta, \theta^\star; \theta^\star) + naI) \right\|.$$

From the continuity of the norm, it thus follows:

$$\lim_{n \to \infty} \left\| (\bar{H}(\theta', \theta^\star; \theta) + n(a+b)I)^{-1} (\bar{J}_\delta(\theta, \theta^\star; \theta^\star) + naI) \right\| = \left| \frac{a}{a+b} \right| < 1. \tag{5.23}$$

From the definition of the limit, there exists some finite n' such that

$$\left\| \bar{H}_{\text{Reg}}(\theta', \theta^\star; \theta)^{-1} \bar{J}_{\delta,\text{Reg}}(\theta, \theta^\star; \theta^\star) \right\| < \left| \frac{a}{a+b} \right| + \epsilon,$$

for all $n > n'$. As ϵ is arbitrary, it can be chosen such that

$$\left\| \bar{H}_{\text{Reg}}(\theta', \theta^\star; \theta)^{-1} \bar{J}_{\delta,\text{Reg}}(\theta, \theta^\star; \theta^\star) \right\| < 1,$$

for all $n > n'$, as required. □

We also note that adding a positive multiple of the identity to the Hessian will eventually make the regularised version positive definite, leading to satisfaction of assumption 5.8 for appropriate choices of α_l. This means that regularisation can be used to ensure

convergence of PFPE. In general, without additional assumptions on the form of the function approximator, it is impossible to determine how the regularised fixed point will relate to the true fixed point. This said, empirical results indicate that regularisation of this sort can be useful, even in the complex function approximation case [Liu et al., 2019].

5.5 Experiments

We proceed to empirical investigation of our bounds. First, we demonstrate that the use of an infrequently updated target network leads to convergence of off-policy evaluation on the Baird's notorious counterexample. Then, we evaluate the effect of a speculative modified update rule in the `Cartpole-v0` `gym` environment [Brockman et al., 2016].

5.5.1 Baird's Counterexample

In this experiment, we demonstrate the practicality of our core claim—that for sufficiently high k and low enough α, PFPE will not diverge, even under conditions that TD does. To do so, we evaluate the use of target networks with varying update frequencies on the well known off-policy counterexample due to Baird [1995].

In this environment, depicted in Section 5.6, rewards are zero everywhere, transitions are deterministic, and the true solution lies within the linear function approximation class that we make use of. The behaviour policy is set such that all states are sampled with uniform probability. The target policy, however, always transitions to a specific state, and remains there. Due to undersampling of this absorbing state, conventional TD policy evaluation diverges, demonstrating that even in simple environments, TD can be unstable when applied off policy with function approximation.

Fig. 5.2 shows a graphical representation of the counterexample. The behaviour policy chooses between the action represented by the wavy line with probability $6/7$, and the solid line with probability 1/7, in order to ensure that all states are visited at equal rates. The behaviour policy always chooses the solid line. The linear function approximation scheme is shown in terms of the value function weights: the function approximator

evaluates the solid action using $\omega_7 + 2\omega_8$, and the wavy action by the value in the current state. Sampling off policy in this way leads to divergence of TD, but PFPE converges, as seen in Fig. 5.3.

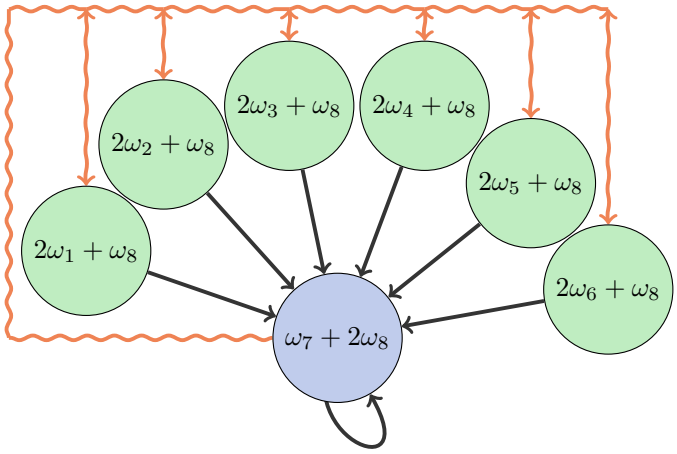


Figure 5.2: Baird's Counterexample. The solid (grey) action moves the agent to the lower state deterministically. The wavy (orange) action puts the agent into one of the upper states with equal probability

We report the stepwise (fitted) error in Fig. 5.3 across different values of k, for fixed step size $\alpha = 0.01$, and fixed discount factor $\gamma = 0.99$. We see that with $k = 1$, which is equivalent to using TD with fixed step sizes, our parameters diverge. Likewise, if k is set to 10 or 100, we are unable to overcome the conditioning of the TD Jacobian and diverge, albeit at a slower rate. Once we take $k \geq 500$, however, conditioning has improved enough to lead to convergence. This supports our theoretical conclusion: that PFPE can be used to improve the convergence conditions of TD and FPE.

5.5.2 Cartpole Experiment

One important insight of our analysis is that we can view the entire optimisation process as a sequence of updates to the target network only. This suggests investigation into alternative forms or acceleration of target network updates. Inspired by the use of optimisation methods with momentum in RL settings [Sarigül and Avci, 2018, Haarnoja et al., 2018], we investigate the effects of a target network that is updated using momentum.

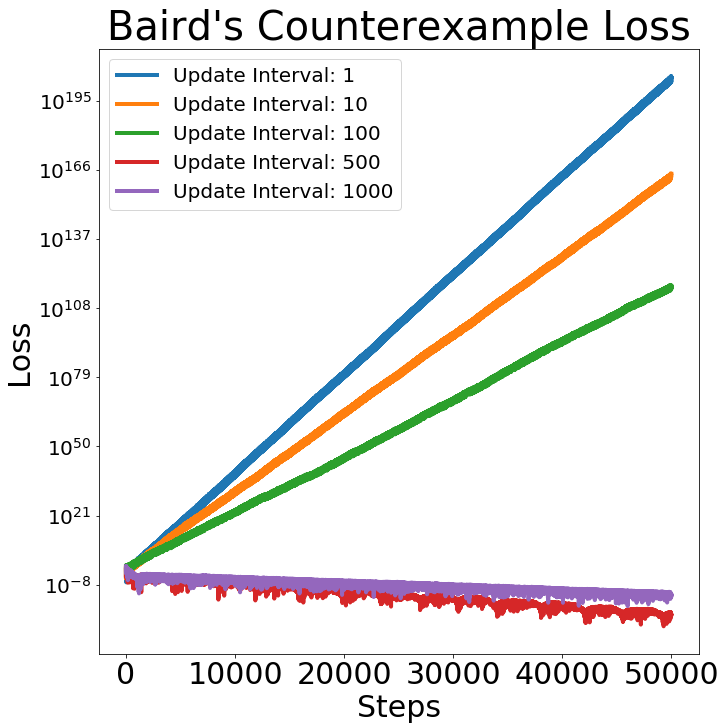


Figure 5.3: Experiment on Baird's counterexample. Decreasing the frequency of target network updates improves conditioning and leads to convergence of PFPE for suitable choices of hyperparameters.

Unlike the standard periodic target network update in Eq. (5.8), we postulate that there may be settings in which a periodic update with momentum may accelerate or stabilise convergence. This update works as follows:

$$\bar{\omega} = \begin{cases} (1-\mu)\omega_i + \mu(\omega_{i-k} - \omega_{i-2k}), & i \bmod k = 0, \\ \bar{\omega}, & \text{otherwise.} \end{cases} \tag{5.24}$$

We investigate the effects of this momentum update on the `Cartpole` domain as implemented in the OpenAI "gym" environment suite [Brockman et al., 2016]. For this experiment, we use control results in which the policy is continuously learned. This is because control problems are inherently off-policy, and induce additional instability, and thus benefit from faster and more stable convergence of values. We implement the standard DQN [Mnih et al., 2015] algorithm, with our modified target network update in order to examine its effect. The results are shown in Figure 5.4. Our proposed update indeed leads to improved learning and stability, at least for the hyperparameter ranges tested, suggesting that the momentum update has merit. As a result, we propose

investigation of more sophisticated target network update schemes as an avenue for future research.

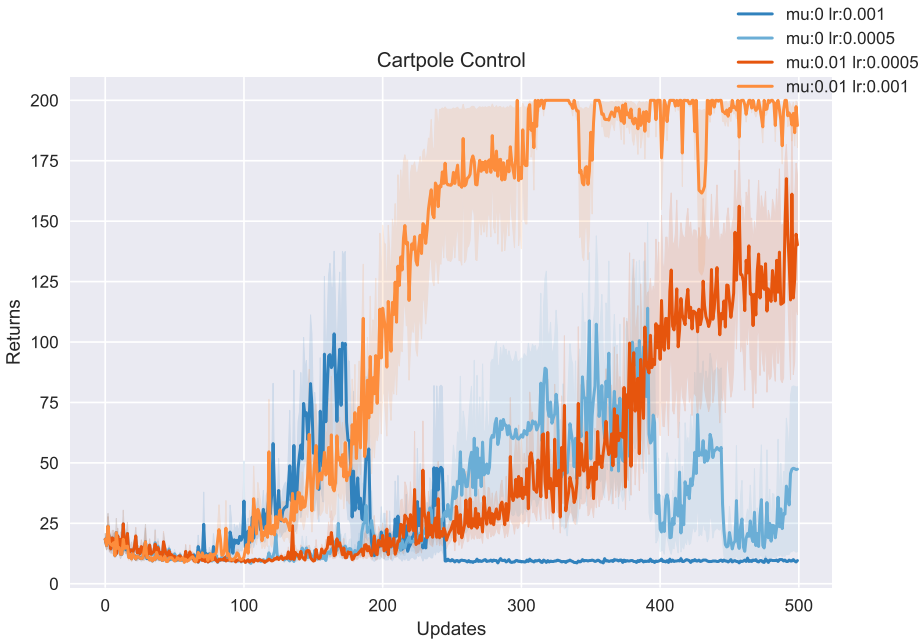


Figure 5.4: Target network with momentum experiment on the `Cartpole` environment. The agent with the momentum update is significantly more stable and able to consistently learn, while without the modified update, learning collapses.

5.6 Additional Experiment Information

For both plots, each configuration was run over 5 random seeds, with the central tendency given by the mean, and the shaded errors representing the standard error of the mean.

Hyperparameters that are not varied in the plots were optimised by grid search across either linear or logarithmic hyperparameter ranges, as is suitable. Parameters were chosen that led to the highest performance as averaged across random seeds, then relevant hyperparameters were varied, using the optimal fixed hyperparameters. Hyperparameters that were varied are denoted as lists in the tables below.

For the `Cartpole` experiment, we use a simple DQN-style setup with a small multilayer perceptron (MLP) representing the value function. A small adjustment is made from PFPE as characterised by the paper. Instead of updating value parameters on single data

points, parameter updates are averaged across a small batch. This was found to increase stability of learning, with no notable effects when comparing across independent variables. This means that, in addition to our target network, we also make use of a replay buffer which stores observed transitions. As such, data used in updates was sampled uniformly from previous transitions. The policy was ϵ-greedy, with the estimated optimal action taken with probability $1 - \epsilon$.

Parameter	Value
Environment Parameters	
γ	0.99
Architecture Parameters	
MLP Hidden Layers	2
Hidden Layer Size	32
Nonlinearity	ReLU
ϵ	0.05
Training Parameters	
Total Target Network Updates	500
Learning Rate	[0.001, 0.0005]
Momentum (μ)	[0, 0.01]
Batch Size	500
Steps per Target Network Update (k)	5
Data Gathering Steps per Update	5
Replay Buffer Size	2500

Table 5.1: Relevant Parameters for `Cartpole` Experiment

5.7 Conclusions

In this chapter, we have continued our exploration of generalisation in RL by analysing target networks–one of the core techniques for stabilising complex function approximation with TD learning. This technique leads to convergent algorithms, even when making use of complex state abstraction via nonlinear function approximation, TD learning, and sampling off-policy, all of which appear to be desirable, if not necessary, for learning with neural network function approximation.

This analysis was conducted by formalising the use of target networks through a class of methods known as partially fitted policy evaluation algorithms. PFPE generalises the more traditionally well understood TD(0) algorithm and fitted policy evaluation

algorithms. While PFPE is generally implemented using a two-timescale update scheme, a core insight derived here is that these updates can be viewed as a single, unified update to the target parameters, simplifying analysis and leading to asymptotic and finite time bounds without additional restrictive assumptions or significant changes to the algorithms used in practice.

While we cannot demonstrate convergence improvements of target networks when using asymptotically decaying step sizes, if step sizes remain bounded away from zero, our main result shows that there will always exist a finite number of update steps, k, and non-zero upper bound over stepsizes such that PFPE geometrically converges to a ball around the optimal parameters, so long as FPE converges. This setting maps well onto current practice, as most implementations of target networks with TD learning do not use a decaying step size. In this way our result helps to explain why target networks have led to empirical success, where traditional TD(0) and FPE fail.

In examining the exact conditions under which target networks help stabilise learning, we hope to encourage development of better forms of state abstraction, and to demonstrate the particular challenges of learning state abstraction in RL. Furthermore, we hope that the insight gleaned here—alongside our preliminary experiments—may lead to novel algorithms which focus directly on learning target networks more effectively, allowing for fast and stable learning of complex state abstraction and ultimately agents with better generalisation capabilities.

6

Temporal Abstraction: Learning Skills Diverse in Value-Relevant Spaces

The content of this chapter was developed as part of the paper Learning Skills Diverse in Value-Relevant Spaces *[Smith et al., 2022].*

The previous chapters have been largely concerned with capturing generalisation in performance in the form of worst-case performance bounds. Chapter 4 explored data abstraction more deeply in the offline imitation learning setting, where empirical estimation of agent performance is a challenge in its own right. Chapter 5 took a deeper dive into state abstraction, and the ways in which target networks can be used to produce good value estimates with nonlinear function approximation and off-policy data sampling. In this chapter, we approach another key notion of generalisation—unique to the RL setting—temporal abstraction. This chapter represents a methodological deviation from the preceding chapters as well, as the novel contribution here is strictly algorithmic, with primarily empirical support, rather than theoretical analysis.

Temporal abstraction—as discussed in Chapters 1 to 3—allows RL agents to reason over "skills", or entire sequences of behaviour, rather than individual actions. This leads to a reduction in the effective time horizon of RL problems and enables more coordinated exploration [Dietterich, 1998, Nachum et al., 2019c]. If skills are modular, they can

facilitate transfer learning, as each skill constitutes a part of a solution to the broader problem [Andreas et al., 2017, Frans et al., 2017, Igl et al., 2020b].

As is the case for state abstraction in supervised learning, one key goal in the development of algorithms for temporal abstraction is learning skills directly from data, rather than manually specifying them. Early approaches in HRL made use of skills specified manually, but this process can be difficult or expensive, creating a need for automated skill discovery. While many modern algorithms learn skills using carefully parameterised policies optimised for reward, another family of algorithms learns without use of a reward function, yielding skills that can be used across a variety of downstream tasks. These recent works [Gregor et al., 2016, Eysenbach et al., 2018, Sharma et al., 2020] learn sets of skills that are optimised such that it is easy to distinguish the states visited by different skills in terms of a categorical classification problem. This classifier, referred to as a discriminator network, is optimised to distinguish skills, while skill policies are optimised to be maximally classifiable by the discriminator.

However, temporal abstraction, when specified in terms of states in this way, may not be able to learn useful skills. In practice the learned skills may be trivial, representing behaviours that lead to states that are distinct, but not in ways that are meaningful to the task at hand. To put this in more concrete terms, imagine a cooking agent that employs visual input as state. In this setting, the cooking temperature or seasoning of a dish would be impossible to express explicitly in pixel space. Without access to abstract features, an agent equipped with a diversity objective would discover easy-to-learn but unhelpful skills (we can imagine it reorganising the spice cupboard in many different ways rather than learning to add them to the dish). Instead, if the agent specifically learns skills that manipulate abstract features of the food, like crunchiness or seasoning, then the agent will be more likely to learn to prepare many dishes quickly.

The question of temporal abstraction that we seek to answer in this chapter is then: "how do we construct diverse skills that abstract over the right thing"? In this way, we investigate the connections between state abstraction and behavioural abstraction, and the need for effective state abstraction in order to specify good skills.

To achieve this, we take inspiration from semi-supervised learning, in which a small amount of labelled (or in our case, reward-dependent) data is used to learn how unlabelled (reward-free) data should be abstracted. Our framework uses sample tasks with available rewards to make it easier to specify the abstract features of the environment which skills should learn to manipulate. Our approach, called DIVRS (Diverse In Value Relevant Space) approaches this problem in stages. First, making use of the reward signal on a small number of tasks, the agent learns a state abstraction which is useful for predicting performance. The agent is then able to use reward-free skill discovery methods to manipulate the space of those relevant and controllable features. These skills are then better suited for agent control in other (potentially more challenging) related tasks, compared to purely unsupervised baselines.

Relative to existing literature, this chapter provides the first concrete path towards making information-theoretic skill discovery methods practical. We do this by developing an algorithm which corrects for more typically found underspecified skills by learning a representation under which skills that visit distinct states achieve distinct returns in the tasks at hand. We validate the empirical efficacy of our algorithm both quantitatively and qualitatively. To do so, we develop a novel environment with pixel observations, which makes use of colour as a key feature. Our study finds that skills learned using representation-based pretraining to ground skills in the task at hand outperform naive counterparts on challenging locomotion tasks.

6.1 Information Theory

As the algorithm studied in this chapter makes use of quantities derived from the field of information theory, we provide a brief overview of the relevant terms.

The foundation of information theory is the study of *entropy*, H, which is used to quantify how much information is gained by realising a sample from a random variable. In this work, we focus on MDPs with continuous state and action spaces, and thus use a form of entropy known as *differential entropy*, which is given by:

$$H(X) := -\int \mathcal{P}(x) \log \mathcal{P}(x) dx,$$

where $\mathcal{P}$ denotes the density function of X. Variables with high entropy assign weight more evenly across possible outcomes, and thus carry more information when sampled.

Also important to our study is the notion of *conditional entropy*. This quantity is given by

$$H[X|Y] := -\int \mathcal{P}(x,y) \log \frac{\mathcal{P}(x|y)}{d} y dx,$$

and represents how surprising the random variable X will be on average when the variable Y takes a known quantity.

Even more central to our algorithm is the notion of *mutual information* (MI), defined as

$$I(X,Y) = H[X] + H[Y] - H[X,Y],$$

where $H[X,Y]$ is the joint entropy, representing the entropy of the joint distribution (X,Y). Intuitively, the mutual information describes correlation between two variables insofar as sampling from one variable provides information about another. If variables share information, then most of the stochasticity in the marginal samples will no longer be present in the joint sample, leading to the final term being smaller than the first two. On the other hand if the variables do not share information, then the joint sample should be nearly as random as the sum of the independent samples, leading to a low mutual information.

6.2 Variational Option Discovery

Our approach builds off of a recent trend in the use of variational option discovery (VOD) methods [Gregor et al., 2016, Eysenbach et al., 2018, Achiam et al., 2018] for unsupervised skill discovery. A skill, π_ω is defined as a policy, $\pi(a|s,\omega)$, which is conditioned on a discrete latent variable ω. We use the conditional notation to highlight that the distribution over the action chosen depend on both the state and the skill variable. In the options framework, this corresponds to an option policy, $\pi_\omega(s,a)$. VOD then take as input a statistic of agent behaviour, ξ_τ, derived from a trajectory, τ, generated by rolling

out a particular skill policy $\pi_\theta(a|s,\omega)$ for an entire episode. The option used for the rollout is generally sampled from a fixed distribution, $\mathcal{P}(\omega)$. In the language of options, this implies that the termination function is zero everywhere, the policy over options is fixed and independent of the state, and the initialisation set for all skills corresponds to the start states of the MDP, though in practice these restrictions are loosened when skills are deployed. The VALOR algorithm [Achiam et al., 2018] chooses ξ_τ to be given by a subsequence of visited states $\{s_0, s_k, s_{2k}...\}$, while DIAYN [Eysenbach et al., 2018] uses individual states. VIC [Gregor et al., 2016] selects ξ_τ to be the final state of the episode. In all cases, the optimisation target for these skills is the maximisation of the MI between ω and the statistic ξ_τ, $I(\omega;\xi_\tau)$, and in some cases, an additional maximisation of the entropy $H[\pi_\theta(\cdot|s,\omega)]$ is used to ensure that skills better cover the state space.

Since MI is symmetric in its arguments, two decompositions can be used for practical estimation:

$$I(\omega;\xi_\tau) = H(\omega) - H(\omega|\xi_\tau) \tag{6.1}$$

$$= H(\xi_\tau) - H(\xi_\tau|\omega), \tag{6.2}$$

Eq. (6.1) is often called the *reverse* MI and Eq. (6.2) the *forward* MI [Gregor et al., 2016, Campos et al., 2020]. Optimising either form directly is generally intractable, as both require integration over the entire state space.

When using the reverse form of MI, a variational lower bound can be formulated for Eq. (6.1), since $H(\omega)$ can be directly computed as options are sampled from a fixed distribution. This results in an overall entropy-regularised objective given by:

$$\max_{\theta,\phi} \mathbb{E}_{\omega\sim\mathcal{P}(\omega)}[\mathbb{E}_{\tau\sim\pi_\omega,P,\rho_0}[\log q_\phi(\omega|\xi_\tau) + \beta H(\pi_\theta(a|s,\omega))] - \log \mathcal{P}(\omega)], \tag{6.3}$$

where q_ϕ is an estimator for $\mathcal{P}(\omega|\xi_\tau)$, parameterised by learned parameters ϕ.

By contrast, the forward objective can only be bounded approximately, due to the need to integrate over all of state space in order to compute both $\log \mathcal{P}(\xi_\tau|\omega)$ and $\log \mathcal{P}(\xi_\tau)$, both

of which must be integrated over the entire state space and have opposite signs. We can approximate this similarly using a variational-style parameterisation as:

$$\max_{\theta,\phi} \mathbb{E}_{\omega\sim\mathcal{P}(\omega)}[\mathbb{E}_{\tau\sim\pi_\omega,P,\rho_0}[\log q_\phi(\xi_\tau|\omega) - \log \sum_{\omega'} q_\phi(\xi_\tau|\omega')\mathcal{P}(\omega') + \beta H[\pi_\theta(a|s,\omega)]]], \quad (6.4)$$

Despite the lack of formal bound, the forward objective has been used effectively in previous work [Sharma et al., 2020, Campos et al., 2020], indicating perhaps that the powerful neural network variational models used in practice are able to closely fit the true distribution, leading to a close approximation of the overall MI.

Two online simultaneous optimisation processes are then coordinated, alongside data collection to maximise MI:

- the variational model, referred to as the *discriminator*, is trained to minimise the negative log-likelihood loss $-\log q_\phi$ on trajectory statistics ξ_τ and labels ω
- at the same time, the skill-conditional policy used to gather the data maximises the corresponding approximation in either Eq. (6.3) or Eq. (6.4).

In other words, for the reverse objective, the discriminator is trained to predict skills from the trajectory statistics, while for the forward one it is trained to predict the trajectory statistics that are visited by the skills. In both cases policies are rewarded for producing trajectories on which the discriminator performs well. In practice, skills learned via forward MI tend to be unimodal, since each skill is specified by a single state prediction, while reverse mode skills are often multimodal [Campos et al., 2020].

In MDPs with rich state spaces, choosing the trajectory statistic directly in state space, $\xi_\tau \in \mathcal{S}^k$, where k is some integer, will not be sufficient to learn skills that are relevant for downstream tasks. Training a DIAYN agent [Eysenbach et al., 2018] on the MuJoCo Ant environment could lead the agent to take actions that affect easily-controllable features like the robot joint positions and orientation. While these skills are distinguishable in terms of the discriminator objective, they do not capture many desirable behaviours in the environment, such as different walking gaits or consistent movement in the x-y plane. This results in skills that will be useless for many tasks, including agent locomotion and navigation. This happens because some features of the state, like joint angles, are directly

influenced by actions and thereby easier to control by the policy than other aspects of state, like the location of the agent, which requires a longer sequence of coordinated actions to control.

In the literature, the most common solution to this issue has been restricting the observation space of the discriminator to some hand-crafted subset of state features, such as the position of the agent's centre of mass (referred to as an x-y prior) [Eysenbach et al., 2018, Sharma et al., 2020] or the difference between consecutive states [Achiam et al., 2018], which encourages skills that move to different states. However, for many tasks, this sort of specification of relevant features may be impossible to achieve a priori.

In an arbitrary task setting, it is difficult to imagine the possibility of this sort of restriction, or more generally, abstraction, in a purely reward-free context. How can we know what sorts of states skills should visit if we don't know anything about the reward that those skills will be eventually used to optimise? In the next section we will introduce the semi-supervised skill discovery section that will allow us to form a more general solution to this problem by learning the state abstraction needed to discover meaningful skills.

6.3 Problem Setting

To learn a diverse set of skills that manipulate feature space in ways that are relevant for downstream tasks, we are given access to an environment $\langle \mathcal{S}, \mathcal{A}, P, \rho^0 \rangle$ that can be explored during learning. Instead of seeking to learn a single task, we now use a set of tasks with different reward functions $\mathcal{M} = \{M_i\}_{i=1...N_m}$, with each task M_i specified by $\langle r_i, \gamma_i \rangle$. Our goal is to pre-train a set of discrete skills $\pi(a|s, \omega)$, $\omega \in \{1, 2, \ldots, N_\omega\}$ such that any of the tasks M_i can be solved quickly. This would be useful if we anticipate having to train multiple agents on different tasks from $\mathcal{M}$, or if particular tasks are known to be challenging (in terms of sparse reward or exploration difficulty). In these cases, we can reduce overall training costs by first learning a set of useful skills only once.

During pre-training, we make use of a small subset of representative *source* tasks $\mathcal{M}_s \subset \mathcal{M}$. In general these tasks can be randomly sampled from an overall task distribution, or chosen by hand to be easy to solve, or represent the space of tasks in $\mathcal{M}$ well. Access

to $\mathcal{M}_s$ allows us to learn skills that are relevant for solving other tasks in $\mathcal{M}$, without having to directly solve a large number of tasks in $\mathcal{M}$ individually, or solve particularly difficult tasks in $\mathcal{M}$. Underscoring the fact that pretraining will be easier than solving all of the tasks, we will see in Section 6.4 that only effective policy evaluation is needed to learn useful skills. Of course, if different tasks require manipulation of wildly different features in order to solve them, then transferring skills between tasks is unlikely to prove useful. As such, we assume the existence of a shared structure between the tasks. In the formulation presented here, we assume there is a set of low-dimensional shared state features $\psi(s)$ with $\dim(\psi(s)) << N_s$, that are useful for predicting the long-term performance on any of the tasks from $\mathcal{M}$, though in general, these features may be difficult to learn.

While this assumption will not apply to all real-world settings, there are many sets of tasks which we can imagine adhering to this setup. In robot navigation tasks, for example, we can imagine that the global position of the robot would be key to reaching any desired place. In the example of robot tasked with cooking many different dishes, the features relevant for differentiating the dishes might be given by ingredient quantities, cooking temperature, and the spiciness or acidity of the dish, etc.

Now that we have a setting in which we can conduct state abstraction in order to learn effective skills, we are ready to introduce DIVRS, an algorithm for learning skills in this setup.

6.4 Method

To approach the problem of specifying an abstract state space and thus learning useful temporal abstraction, we propose DIVRS (Diverse In Value Relevant Space). DIVRS uses the source tasks in $\mathcal{M}_s$ to learn a parameterised feature mapping $\psi_\vartheta(s) : S \to S^\psi$, that abstractly represents the variations in state that matter for the tasks of interest. If we then restrict the observation space of the discriminator to our learned latent representation, we discover skills that are not only diverse, but also diverse in relevant features. DIVRS consists of two phases of learning. The first phase makes use of the

representative tasks $M_i \in \mathcal{M}_s$ in order to learn good abstract features. After learning these features, we need to extract the representation $\psi_\vartheta(s)$ from this agent. In the second step, without any more use of the reward function, VOD methods are employed to learn skills that reach states that are *diverse* in those features. The pseudo-code for both phases can be found in Algorithm 3.

Algorithm 3 DIVRS

Input: set of source tasks $\{\mathcal{M}_1, ..., \mathcal{M}_k\}$, coverage policy μ
Output: set of skills $\pi_\theta(a|s, \omega)$

Phase 1 - Learning State Representation $\psi_\vartheta(s)$

initialise $\psi_{\hat{\vartheta}}(s), \{w_1, ..., w_k\}, f_v$
while $\psi_{\hat{\vartheta}}(s)$ is not converged **do**
    sample $\mathcal{M}_i$ from $\{\mathcal{M}_1, ..., \mathcal{M}_k\}$
    for j in n_steps **do**
        transition_batch $\leftarrow$ rollout_policy($\mathcal{M}_i$, μ)
        $\psi_{\hat{\vartheta}}(s), \{w_i\}, f_v \leftarrow$ **PEval**(transition_batch, $\psi_{\hat{\vartheta}}(s), \{w_i\}, f_v$)
$\psi_\vartheta(s) \leftarrow$ **Distillation**($\{\mathcal{M}_1, ..., \mathcal{M}_k\}$, $\psi_{\hat{\vartheta}}(s), \mu$)

Phase 2 - Learning Skills $\pi_\theta(a|s, \omega)$

fix the parameters ϑ of $\psi_\vartheta(s)$
initialise $\pi_\theta(a|s, \omega), q_\phi(\omega|\psi_\vartheta(s)), Q_\nu(s, a|\omega)$
while not converged **do**
    sample $\omega \sim \mathcal{P}(\omega)$
    for j in n_steps **do**
        exps $\leftarrow$ rollout_policy($\pi_\theta(a|s, \omega)$)
        if Using reverse MI objective **then**
            exps.reward $\leftarrow \log q_\phi(\omega|\psi_\vartheta(s_t)) - \log \mathcal{P}(\omega)$
        else if Using forward MI objective **then**
            exps.reward $\leftarrow \log q_\phi(\xi_\tau|\omega) - \log \sum_{\omega'} q_\phi(\xi_\tau|\omega')\mathcal{P}(\omega')$
        $\pi_\theta(a|s, \omega), Q_\nu(s, a|\omega) \leftarrow$ SAC_update(exps, $\pi_\theta(a|s, \omega), Q_\nu(s, a|\omega)$)
        $q_\phi(\omega|\psi_\vartheta(s)) \leftarrow$ optimise_dscr_loss(exps, $q_\phi(\omega|\psi_\vartheta(s))$)

6.4.1 Phase One

Choice of features Our learning approach requires features that are relevant to performance on the tasks of interest. Since the value function is a measure of long-term performance, we suggest that $\psi_\vartheta(s)$ be given by learned abstract features useful for prediction of values on our source tasks. We expect these features to characterise how rewarding the state that the agent is currently occupying will prove to be. If the agent

reaches states with features that predict high values, it will perform well on the given task.

The primary alternative consideration—using features that directly predict single-step rewards—is less desirable for a few reasons. If the reward is binary, for example, then the related features will only be useful for binary classification, potentially leading to discontinuities in the feature space which may make learning with those features more difficult. Comparatively, the value function is smooth, which leads to more uniform feature space. Furthermore, the value function carries more information about the long term effect of various features of the environment. Other approaches that already exist for learning features in RL, including autoencoders [Lange and Riedmiller, 2010], and successor features [Schaul et al., 2015], are focused on reward-free learning, while here we want to learn features that are closely tied to performance on the source tasks.

Value function architecture To ensure that $\psi_{\hat{\vartheta}}$ does not depend on specific tasks, but is still used across all tasks, we can choose our class of value function estimators V_i^μ (as in [Schaul et al., 2015]) such that values can be factored as a combination of a state-independent task embedding and the learned features, i.e. $V_i^\mu(s) \approx f(\psi_{\hat{\vartheta}(s)}, w_i)$, where $f(\cdot)$ is a simple function like inner product or a small neural network, μ is our data gathering policy, and w_i is the state-independent embedding of task i.

Learning features In order to learn a value function and the features useful for predicting values, we first must gather data. We make use an exploratory policy, μ, for all the tasks in the first phase and evaluate that policy, carefully parameterising our

Algorithm 4 PEval

Input: transition batch D, shared embedding $\psi_{\hat{\vartheta}}(s)$, task embedding w, value network f_v

$S, A, R, S' \leftarrow D$
$\delta \leftarrow (f(\psi_{\hat{\vartheta}}(S), w) - R - \gamma f_v(\psi_{\hat{\vartheta}}(S'), w))$
$v \leftarrow v - \delta \nabla_v f_v(\psi_{\hat{\vartheta}}(S), w)$
$\hat{\vartheta} \leftarrow \hat{\vartheta} - \delta \nabla_{\hat{\vartheta}} f_v(\psi_{\hat{\vartheta}}(S), w)$
$w \leftarrow w - \delta \nabla_w f_v(\psi_{\hat{\vartheta}}(S), w)$
return $\psi_{\hat{\vartheta}}(s)$, w, f_v

value function to ensure that a single $\psi_\vartheta(s)$ is optimised across all tasks. A random policy, a policy that has been optimised for the tasks, or a policy optimised to uniformly cover the state space are all relevant possibilities for μ. In our experiments, random policies were sufficient to derive good skills, though we investigate an optimised policy as well. One important algorithmic decision in our setup involved first training a value network with a high dimensional embedding, $\psi_{\hat{\vartheta}}(s)$, and later compressing that representation using a process known as network distillation, which is described below. This enabled learning of higher quality representations due to the increased capacity during high-variance policy evaluation. The policy evaluation procedure for learning $\psi_{\hat{\vartheta}}(s)$ is detailed in Algorithm 3 (Phase 1), and Algorithm 4. Because we are only interested in features of the value function, it is important to underscore that our approach only needs to perform policy evaluation, instead of learning potentially challenging optimal policies on source tasks.

Compression of features In order for DIVRS to work, ψ_ϑ must be a sufficiently parsimonious representation. If it is not compressed enough, and contains information from features that are not useful for predicting value, when those features are later used to specify skills, the learned skills may manipulate these irrelevant dimensions, rather than the ones useful for predicting value. For these reasons, we must make use of regularisation and distillation techniques to ensure good compression.

Information bottleneck methods [Tishby et al., 2000] encourage learning general but concise representations. These methods look to learn a minimal sufficient statistic by maximising the information conserved about the relevant prediction (in our case the value function), while also maximising the amount of information lost between state and state abstraction. Formally, this can be given by

$$\min_{\vartheta} I(s; \psi_\vartheta(s) - I(\psi_\vartheta(s); V_i^\pi(s)))$$

. While methods which directly optimise this objective exist [Alemi et al., 2016, Igl et al., 2019], this effect may also occur in neural networks with capacity constraints at an intermediate layer [Tishby and Zaslavsky, 2015, Shwartz-Ziv and Tishby, 2017].

Algorithm 5 Distillation

Input: A set of source tasks $\{\mathcal{M}_1, ..., \mathcal{M}_k\}$, target embedding $\psi_{\hat{\vartheta}}(S)$, coverage policy μ
Output: Distilled Embedding $\psi_{\vartheta}(S)$

initialise bottleneck network ϑ, reconstruction network d_v
while $\|\psi_{\hat{\vartheta}}(s) - d_\omega(\psi_\vartheta(s))\| \geq \epsilon$ **do**
    sample $\mathcal{M}_i$ from $\{\mathcal{M}_1, ..., \mathcal{M}_k\}$
    for j in n_steps **do**
        $\leftarrow$ rollout_policy($\mathcal{M}_i$, μ)
        $l \leftarrow$ RegressionLoss$(d_\omega(\psi_\vartheta(S)), \psi_{\hat{\vartheta}}(S))$
        $v \leftarrow v - \nabla_v l$
        $\vartheta \leftarrow \vartheta - \nabla_\vartheta l$
return $\psi_\vartheta(s)$

We found this approach to give sufficiently terse embeddings for skill learning. To further encourage this information bottleneck effect, we make use of regularisation during training [Cobbe et al., 2019a]. However, directly learning a value function with a small bottleneck network layer was challenging, since we found policy evaluation to require excess capacity to learn well. To tackle this problem, we make use of distillation [Rusu et al., 2015], in which a second network with an appropriate bottleneck (which will be eventually taken as ψ_ϑ) is trained on a regression objective so as to predict the outputs of $\psi_{\hat{\vartheta}}$, despite being a lower dimension embedding. This is achieved via a reconstruction network d_v that takes the compressed representation ψ_ϑ and projects it into the original representation space corresponding to $\psi_{\hat{\vartheta}}$. The pseudo-code for our distillation can be found in Algorithm 5.

6.4.2 Phase Two

Learning skills In the second phase, we make use of our learned state abstraction in order to specify the skills learned by a VOD algorithm. To this end, the compressed features ψ_ϑ learned in the first phase are used as the observation space for our discriminator network. In principle any VOD algorithm can be used here, but for the sake of simplicity, and to isolate the representation learning aspect of our contribution, in our experiments we employ either forward or reverse MI forms of DIAYN [Eysenbach et al., 2018, Campos et al., 2020]. The discriminator and skills are trained online in parallel, with the discriminator trained to minimise the prediction loss at

each state independently–i.e., $\mathcal{L}_\tau(\phi) = \sum_{t\in\{1..T\}} \mathcal{L}_t(\phi)$, where $\mathcal{L}_t(\phi) = -\log q_\phi(t)$. The discriminator loss at a timestep t is $q_\phi(t) = q_\phi(\psi_\vartheta(s_t)|\omega_t)$ for the forward MI objective, and $q_\phi(t) = q_\phi(\omega_t|\psi_\vartheta(s_t))$ for reverse MI. The resulting pseudo-reward for the skill is $r_t = \log q_\phi(t) - \log Z_t$ with normalisation term $Z_t = \sum_\omega q_\phi(\psi_\vartheta(s_t)|\omega)P(\omega)$ for the forward objective and $Z_t = P(\omega_t)$ for the reverse objective, with uniform skill prior distribution $P(\omega)$. The parameters of the state embedding $\psi(s)$ are fixed for the duration of this phase.

Skill use Once our skills are learned, they can be used to visit regions that are distinct in the value-relevant state abstraction from phase one. Two possibilities for such use include selection and fine-tuning of the best performing skills on other tasks from $\mathcal{M}$, or training a policy over options, $\pi(\omega|s)$ as in Eysenbach et al. [2018], to coordinate learned skills. These possibilities will be investigated empirically in Section 6.6.

6.5 Information-Theoretic Comparison of DIVRS and DIAYN

DIVRS learns skills that manipulate the environment in accordance with the dynamics of the value function, rather than the dynamics of the state space. In this section, we make an information-theoretic argument about how our approach is better able to capture value relevant features than the base DIAYN algorithm. In doing so, the assumptions that we make will expand on and clarify the use case of our algorithm. We show, under these assumptions, that skills learned via DIVRS have higher MI with state values than skills learned directly from the observation space.

In an MDP, for a fixed state distribution, we consider the value function for a fixed policy to be a deterministic transformation of the random variable S, corresponding to the state. This gives us a distribution over values of states, $\mathcal{P}(V(S))$. In reference to the information bottleneck model of data generation, we invert this relationship, assuming that a randomly sampled label (in our case the value function) generates a distribution over states with that value. We then consider phase one training as an optimisation process of

the information bottleneck objective [Tishby et al., 2000] for a parameterized embedding of state, $\psi_\theta(s)$:

$$\min_\theta I(\psi_\theta; S) - \beta I(\psi_\theta; V)$$

In order to capture the notion of features that are not useful for value prediction, we partition the state space into those components that can be used to predict the value, U, and those that are sampled independently of U, given by ζ (i.e. $I(\zeta, U) = 0$). While this is not always possible, it captures the notion that some components of state are not very informative of the value function (imagine joint angles on a MuJoCo task in which the agent must reach a specific point in space). During phase one, S is then encoded into a latent representation, ψ, which represents a minimal sufficient statistic of V from features in S. In order to express that this encoding is done perfectly, we introduce the following assumption:

Assumption 6.1. *ψ contains exactly the information from S needed to recreate V, and no other information, that is:*

$$H(V) = I(V; S) = I(\psi(S); S) = I(\psi(S); V).$$

Assumption 6.1 implies that both S and $\psi(S)$ can perfectly predict V, and that the only information in $\psi(S)$ is the information about V. Given that ψ is a (deterministic) embedding of S used to predict V, which is itself a deterministic function of S, this assumption is satisfied, so long as ψ does not contain spurious projections from S, and so long as ψ is rich enough to capture the complexity of $V(S)$. In our running example, the locomotion task, this assumption corresponds to the idea that $\phi(S)$ encodes the position of the agent, which is sufficient information to predict the value of S. Further, it suggests that spurious features, such as joint angles are not included in $\phi(S)$. This assumption will be satisfied based on experimental design choices, such as the dimension of $\phi(S)$, and the reward function in the source tasks.

We build skills from either the raw state space (as in DIAYN), Z_S, or from our compressed representation, leading to skills Z_ψ, where these random variables correspond to the skill

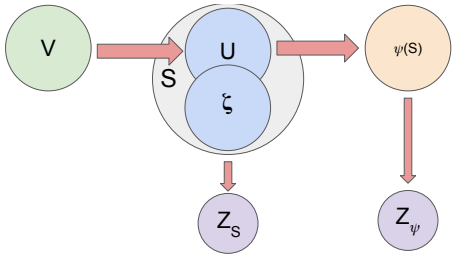


Figure 6.1: Graphical model for each time step while learning skills from both pretrained representations and raw states. $\mathcal{P}(S)$, $\mathcal{P}(Z_S)$, and $\mathcal{P}(Z_\psi)$ are all fixed.

index active at the current state. Skill indices are then distributed across states (or embeddings) to maximise the corresponding mutual information ($I(S; Z_S)$ or $I(\psi(S); Z_\psi)$) for a fixed (uniform) prior distribution over skill indices and our fixed distribution over states. For the purposes of the proof, we assume that this process occurs perfectly—the skill index can be predicted perfectly from the state or embedding that it derives from:

Assumption 6.2. *Skill indices are perfectly predictable from their corresponding state or embedding variable:*

$$H(Z_S|S) = H(Z_\psi|\psi(S)) = 0.$$

In practice, this assumption is almost never going to be perfectly realised, as model capacity is limited, and the set of initial states will always have ambiguous skill labels, since all skills can be initialised in them. However, once skills are differentiated, this will be nearly satisfied, as the agent will immediately move away from ambiguous states towards skill-specific states.

From assumption 6.2, it follows that $I(Z_S; S) = I(Z_\psi; \psi(S))$, since $H(Z_S) = H(Z_\psi)$, given that the skill indices have the same marginal distribution by construction.

Our final assumption is concerned with the fitting of the skills to the value-irrelevant features, ζ. We assume that the skill policies $\pi(s|z_s)$ are able to control features in ζ, and thus the skills Z_S learn to contain at least some information about ζ when maximizing the mutual information between states and skill indices:

Assumption 6.3. *Skills learned in raw state space manipulate features in ζ:*

$$I(Z_S; \zeta) > 0.$$

This assumption can usually be expected hold true. The only environments in which it will not is when all spurious features that are not indicative of task value also cannot be controlled by the agent.

For simplicity's sake, in our description, we have considered only the value function of a single task, though the proof that follows easily extends to a multi-task setting, with value $V' = \mathbb{E}[V]$, where the expectation is over the task distribution. The only requirement is that the features in ψ are sufficient to predict the expected value across tasks.

The dependencies between variables here can be visualised using the graphical model given by Fig. 6.1 for any given time step.

With our assumptions collected, we can move on to prove that skills learned with pretraining share more mutual information with the task values than those learned without:

Theorem 6.1. *For a fixed distribution over states and skill priors, let $\{U, \zeta\}$ be a partition of state features, such that U represents features useful to value function prediction, with MI $I(U, V) = H(V)$, while features in ζ are sampled independently of those in U, with $I(U, \zeta) = 0$. After learning sets of the same number of skills using both DIAYN and DIVRS on the fixed distribution, let Z_S represent the random variable corresponding with the DIAYN skill index active at a given time step. Let Z_ψ represent the active skill index of skills learned using DIVRS. Given assumptions assumptions 6.1 to 6.3, we have:*

$$I(V; Z_\psi) > I(V; Z_S).$$

Proof. We start by applying Bayes' rule to $H(Z_\psi|V)$:

$$\begin{aligned}
H(Z_\psi|V) &= H(Z_\psi|V,\psi(S)) + H(\psi(S)|V) - H(\psi(S)|Z_\psi,V), \\
&= H(\psi(S)|V) - H(\psi(S)|Z_\psi,V), && \text{(Assumption 6.2)} \\
&\leq H(\psi(S)|V) - H(\psi(S)|Z_\psi,V,S), \\
&\leq H(\psi(S)|S) - H(\psi(S)|S). \\
&\qquad \text{(Assumption 6.1 and } \psi \text{ is a deterministic function of } S) \\
&\leq 0.
\end{aligned}$$

However, since entropy is non-negative, it must be that $H(Z_\psi|V) = 0$. From this it follows that:

$$\begin{aligned}
I(Z_\psi;V) &= I(Z_\psi;\psi(S)) \\
&= I(Z_S;S), && \text{(Assumption 6.2)} \\
&= H(Z_S) - H(Z_S|U,\zeta), \\
&= H(Z_S) - H(\zeta|U,Z_S) - H(Z_S|U) + H(\zeta|U), \\
&= I(Z_S;U) + I(Z_S;\zeta|U), \\
&\geq I(Z_S;V) + I(Z_S;\zeta|U). \\
&\qquad \text{(Data Processing Inequality and independence of } U \text{ and } \zeta)
\end{aligned}$$

By assumptions 6.2 and 6.3, we have:

$$I(Z_S;\zeta|U) = I(Z_S;\zeta) - I(U;\zeta) > 0,$$

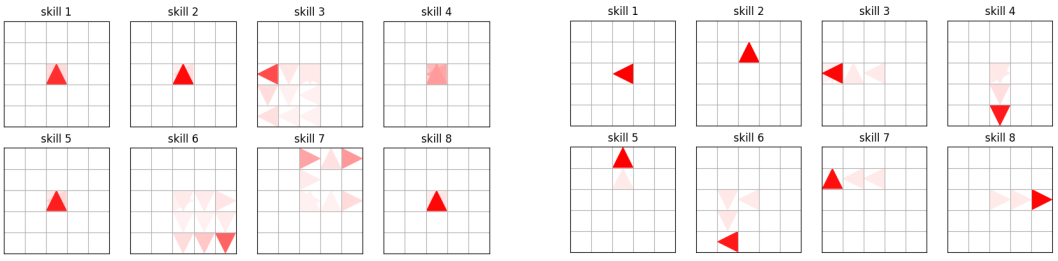
which gives us the desired result. □

Intuitively, the theorem holds because DIAYN learns skills to control elements of state space that may be uncorrelated with state values, while DIVRS constrains skills to only manipulate features relevant to the value function. Given that the methods have the same number of skills with the same sampling distribution, this means that all of the information in Z_ψ pertains to V, while some of the information in Z_S does not.

6.6 Empirical Evaluation

This section is concerned with the empirical validation of our proposed approach, investigating the ways in which learning good state abstraction leads to better specified skills. To do so, we conduct experiments in several domains: a simple gridworld, MuJoCo locomotion tasks which make use of features, and a novel MuJoCo control task with image-based state space. Our results look to both build intuition as to the sorts of skills that our pretraining procedure learns, and to establish concrete, quantitative benefits of our method as compared to baselines.

6.6.1 Gridworld Experiments



(a) DIAYN **(b)** DIVRS

Figure 6.2: Visualising state visitation frequencies of eight skills in a gridworld environment with additional confounding feature dimensions for both DIAYN [Eysenbach et al., 2018] (a) and DIVRS (b). The triangle represents the agent position, with the tip pointing in the direction of orientation, and colour intensity indicating visitation frequency of that particular state. We note that the agent trained with DIAYN falls prey to the additional dimensions and largely remains in the initial state, while DIVRS learns to navigate the grid effectively.

In order to build intuition as to the sorts of settings we expect our method to work in, our proposed approach was tested in a modified 7×7 gridworld from Chevalier-Boisvert et al. [2018]. As in the original environment, in this variation, the agent state is given by its position on the grid, as well as its orientation. The agent is able to rotate, and moves along the grid in the direction it is currently oriented. To capture the notion of spurious features as discussed in assumption 6.3, we augment our feature space with additional controllable dimensions. We add four such integer-valued "distractor" dimensions, each corresponding to the direction that the agent is facing in the plane. The action space is then augmented with two additional actions, which allow the agent to increment or

decrement the distractor dimension corresponding to the direction that it is facing. We can imagine that the agent has access to a remote control that manipulates the brightness of the wall that it is facing. However, these additional features and corresponding actions are not useful to the task at hand, which is a normal goal-reaching task in the x-y plane.

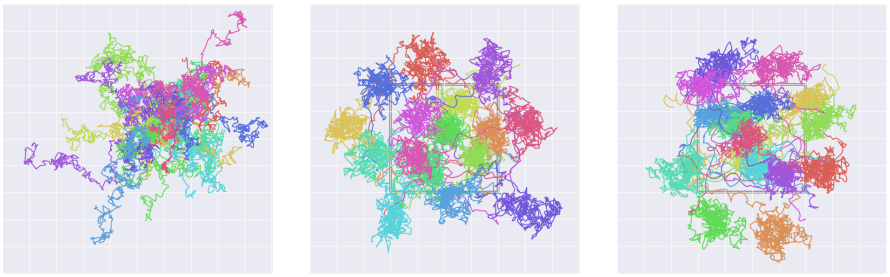
During phase one of our approach, we train an entropy-regularised PPO agent [Schulman et al., 2017] to reach one of two corners of the grid. We made use of a sparse reward, so the agent was only rewarded once it reached the active goal state. During RL training, we initialise the agent at a random state to ensure generalisation of the value function. A convolutional neural net is applied to the grid in order to embed the agent position in x-y space. The non-location observations—agent direction and the distractor dimensions—are concatenated and passed through a MLP network in order to embed these features. The two resulting embeddings are concatenated, and then fed to actor and critic network heads, alongside a task index embedding, which identifies which of the two tasks the agent is in. We did not make use of distillation for this experiment as we found it to be unnecessary for learning good skills.

In phase two, a stop-gradient operator is applied to the concatenated feature embeddings (without the task index). These fixed embeddings are used as environment features for the discriminator network during DIAYN training, where here we used reverse MI objective in Eq. (6.1). The discriminator is a single hidden layer neural network with ReLU nonlinearities applied to those features. Skill policies are trained with PPO, using an architecture similar to the one during phase one, replacing the task index embedding with a skill index embedding, which is given by fully connected layers applied to a one-hot representation of the active skill and concatenated with the location and non-location embeddings. In experiments with DIAYN, we use the same architecture for the discriminator and skills, but the discriminator takes as input the raw state space rather than the learned state embedding. Training during this phase is always initialised in the centre of the environment to produce consistent and localised skills.

Fig. 6.2 provides a visualisation of the learned skills for both DIAYN and DIVRS. We find that the additional distractor dimensions do indeed confound DIAYN, where five

of eight skills never leave the initial state, instead finding separation in the distractor dimensions. On the other hand, by making use of the pretrained embedding, the skills learned are unaffected by the distractor dimension, and reliably learn to move to different positions in the environment. This is because the state abstraction is well-formulated for the task at hand, leading to effective specification of the behavioural abstraction.

With strong qualitative intuition that our procedure is effective, we now proceed with more realistic experiments on more challenging tasks.



(a) Reverse MI. **(b)** Forward MI with x-y-prior. **(c)** Forward MI with DIVRS.

Figure 6.3: Trajectory traces in the x-y plane of 16 different Ant skills trained with varying algorithms: (a) reverse MI, (b) forward MI with x-y prior, (c) forward MI with DIVRS. Each skill was executed three times for each algorithm, with traces coloured by skill index. While DIAYN in the raw state space leads to inconsistent and undistinguished skills (a), DIVRS is able to learn skills which are consistent and easily distinguishable in visitation of different regions of the x-y plane (c), and are competitive with skills that use explicit hand-crafting of features (b).

6.6.2 MuJoCo With Features Experiments

Moving into more complicated continuous control domains, we select a variant of the MuJoCo Ant [Todorov et al., 2012, Brockman et al., 2016] environment as the ideal testbed, as in previous work, an x-y prior needed to be specified by hand in order to learn skills that can be used successfully in challenging navigation tasks [Eysenbach et al., 2018, Sharma et al., 2020]. As DIAYN is unable to formulate skills that reflect state abstraction meaningful to the task at hand in this environment, we consider this to be an ideal setting in which our algorithm can demonstrate its relative strength.

Paralleling the setup in Section 6.6.1, we make use of three navigational source tasks, each corresponding to a goal, g, in the x-y plane, which the agent must reach to obtain a reward. Each episode terminates when the agent reaches the goal, or after a fixed

number of timesteps. In these tasks, the distance between the agent and the goal broadly determines the value of the state, with other features like joint angles and agent orientation being less indicative of state value. So, in order to simultaneously capture the distance from all three goals, our learned features must effectively encode the position of the agent.

In the first phase of training, we use online policy evaluation via state-action TD learning with multiple agents running in parallel as in Mnih et al. [2016] to learn a value function that is conditioned on the goal, $V(s; g)$. GAE [Schulman et al., 2015b] and PPO-style clipping of the value function [Schulman et al., 2017] are used to stabilise and speed up learning. The value function is parameterised as $V(s; g) = f_v(\psi(s) \odot w(g))$, where the function $f_v(\cdot)$ receives an element-wise multiplication of the state and goal embeddings ($\psi(s)$ and $w(g)$ respectively). Here, we found performing policy evaluation using a randomly initialised and fixed policy network during representation learning to lead to skills as good as or better than those learned with a trained policy as in Section 6.6.1.

In this experiment, in order to effectively learn the value function, we found that we could not parameterise the value network such that $\psi(s)$ was sufficiently small to achieve good compression and learn meaningful skills. As such, we perform network distillation on $\psi(s)$ in order to ensure feature generalisation and terse representation. To do so we train a network with a narrow bottleneck layer, $\bar{\psi}(s)$, to predict the learned value representation, $\psi(s)$, where training occurs on states sampled by a randomly initialised policy. The new representation, $\bar{\psi}$, will be used as our state abstraction for VOD learning in the second phase.

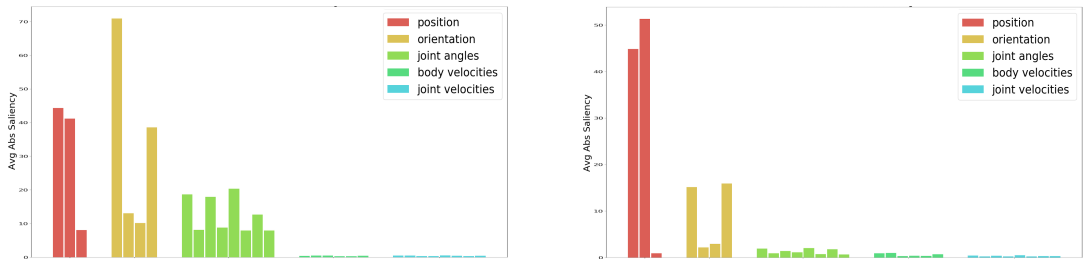
For VOD training in phase two, as in related approaches [Eysenbach et al., 2018, Sharma et al., 2020], we make use of SAC Haarnoja et al. [2018] to train the skill policies. For this experiment, we used the forward MI objective to learn skills. We chose forward MI in order to ensure skills behaved consistently across the state space. Further discussion of this decision is provided below. In order to implement the forward MI objective, we must choose a model for the skill-conditional state distribution, $q_\phi(s|z)$. In our case, we use a multivariate Gaussian in the learned feature space (or in the predefined space when using

the x-y prior). We choose the covariance to be a fixed uniform diagonal, and learn the mean. While the covariance parameters could be in principle optimised directly by the discriminator, we tuned them as additional hyperparameters.

Qualitative Skill Evaluation

In order to build intuition into the nature of skills learned by DIVRS compared to those learned by DIAYN we plot traces of the agent's position, as well as saliency maps [Simonyan and Zisserman, 2015] of the trained discriminator. In the case of the x-y traces, we can observe skillset features such as overall coverage of the x-y plane, as well as how consistently the same skills localise in the same regions of state space. On the other hand, saliency maps express exactly which features are influential in skill classification, demonstrating how state abstraction gives rise to behavioural abstraction.

Fig. 6.3 compares DIAYN, DIAYN with the x-y prior, and DIVRS in terms of agent location in x-y space. In the absence of the x-y prior, skills trained with DIAYN are not well defined in terms of locality, echoing previously established conclusions from Eysenbach et al. [2018], Sharma et al. [2020]. On the other hand, DIAYN with an x-y prior and DIVRS explore the x-y plane consistently, with well localised skills. In this way, we see that the abstraction learned by DIVRS is qualitatively similar to those that have been engineered to suit the task at hand. We note that, while our baseline DIAYN implementation makes use of a different MI formulation from the one that we



(a) DIAYN. **(b)** DIVRS.

Figure 6.4: Saliency of ant features for the trained discriminator network. We use saliency as a proxy for how relevant a feature is to the decision reached by the network. Features are grouped into position, orientation, joint angles, body valocities, and joint velocities. We observe that the position features are much more relevant for the discriminator trained with DIVRS (b) than the one trained with DIAYN (a).

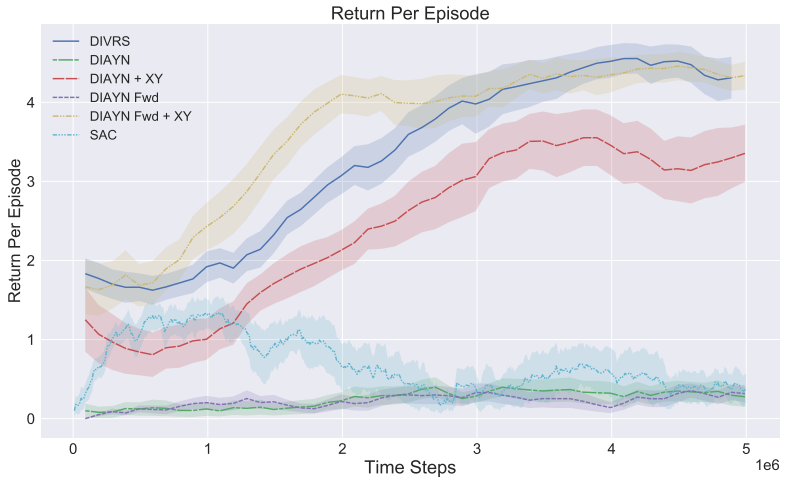


Figure 6.5: Learning curves on sparse sequential task.

used for our method and the x-y prior, we found that learning $q_\phi(s|z)$ directly with high dimensional state space was impossible due to high variance in the discriminator learning objective, preventing meaningful learning of skills. This suggests that, alongside leading to specification in the correct feature space, our learned abstraction can make learning the discriminator easier.

The saliency analysis in Fig. 6.4 demonstrates that skills learned by DIVRS are indeed largely differentiated in terms of agent position, while skills learned from DIAYN make more use of easier to control features, such as agent orientation and joint angles. This confirms our hypothesis that providing the discriminator with features pre-trained on value prediction for the source tasks lead the discriminator to specify skills based on the features relevant to those tasks.

Sparse Downstream Task with HRL

In order to evaluate skills learned by DIVRS in more concrete, quantitative terms, we test the agent on a challenging, sparse task that has been established to be difficult to solve without the use of skills. The sequential sparse navigation task from [Eysenbach et al., 2018] is ideal for this purpose, as non-hierarchical methods such as SAC [Haarnoja et al., 2018] are unable to solve the environment directly, and skill learning methods like DIAYN require handcrafted priors for the discriminator observation space. The task is sequential, requiring the agent to visit a number of goal states in a particular order to

Method	Avg Return
DIVRS	0.954 ± 0.031
DIAYN	0.360 ± 0.084
DIAYN + x-y-prior	0.972 ± 0.012
Forward MI DIAYN	0.384 ± 0.169
Forward MI DIAYN + x-y-prior	0.947 ± 0.040
SAC	0.174 ± 0.051

Table 6.1: Zero-shot performance on unseen goals.

solve the task. The goals correspond to the anticlockwise rotation of the agent around a square with side length four, with the agent receiving a sparse reward at upon reaching each corner as it traverses these goals in sequence.

To train an agent to make use of the learned skills, skills were truncated to last 50 timesteps, with this duration selected by hyperparameter tuning. The policy over options is trained using a discrete version of SAC [Christodoulou, 2019] in the Semi-Markov Decision Process (SMDP) [Puterman, 2014] induced by the learned skills and the task MDP. In order to ensure that the policy over options has enough information to accurately coordinate skills, a one-hot representation of the number of goals reached is concatenated to the agent state. Both the SAC baseline and hierarchical policies are trained for five million primitive time steps, although hierarchical methods trained at the SMDP level can only make use of data gathered at the hierarchical decision points, making the baseline favour SAC overall in terms of data use.

Figure 6.5 shows the overall results of this experiment. The skills learned by DIVRS consistently complete an entire lap of the square. This is competitive with skill discovery methods that make use of a hand-specified feature space, and surpasses the use of DIAYN with the x-y prior and reverse MI objective, even though DIVRS learns only from a binary reward signal on source tasks. We see that skills learned without any state abstraction are generally unsuccessful in the task, as they do not explore the relevant feature space efficiently. Direct training of an SAC agent is similarly unsuccessful in achieving consistent returns, as learning fails after approximately one million timesteps.

Few-Shot Task with Skill Selection

In tandem with the results on our sparse sequential task described above, in this section we look to evaluate another use case for the behavioural abstraction learned by DIVRS. Here we evaluate how skills learned can be useful when there is a large set of tasks that we wish to solve quickly. To do so, we evaluate the skills on a zero-shot learning task in which the agent must reach a goal sampled randomly from the 4×4 box centred at the origin, with a sparse reward given for reaching the goal.

As in Eysenbach et al. [2018], evaluation consists of running each skill until the episode terminates, and choosing the best performing skill as the solution to the task. We compare DIVRS with DIAYN for both forward and reverse MI objectives, either making use of the x-y prior or without it. We also compare with a skill-free baseline, which learns using SAC over the number of steps that the skill-based methods require to evaluate all of their skills.

The results are shown in Table 6.1. We report performance averaged over ten different randomly sampled goals. DIAYN skills without the x-y prior, since they are misspecified for locomotion on the plane, are infrequently able to reach the goal, only finding those near the origin. Similarly, the skill-free policy occastionally reaches goals, but the learning problem is too challenging to consistently reach goals using the limited amount of data needed for the skill-based policies to evaluate all 16 skills. Since the temporal abstraction is structured in harmony with the state abstraction needed for the task, skills learned with DIVRS—and those learned using DIAYN with the x-y prior—are able to solve nearly all of the goal-seeking tasks leading to consistently high performance.

6.6.3 Pixel MuJoCo Experiments

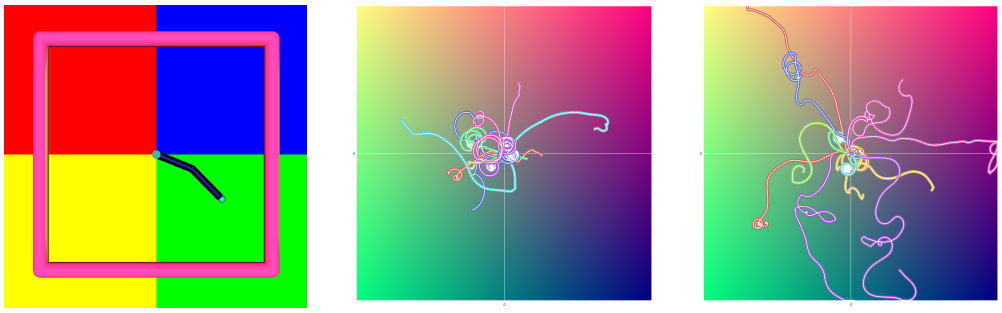
The experiments in the section thus far confirm that our procedure is able to learn a state abstraction that leads to skills that are as effective as those specified by a hand-engineered prior. However, in order to demonstrate the true capabilities of DIVRS, we conduct an evaluation on a task in which hand-specification is difficult to achieve, as the agent does not have direct access to the relevant state features. To do so, we will make use of an

environment that uses pixel-based images as state input. Loosely inspired by the cooking robot example that we have used throughout this chapter, our environment is a modified version of the MuJoCo Reacher task, though in our version, goals are specified in an abstract space, rather than the position of the reacher tip.

The environment To evaluate the ability of DIVRS to learn temporal abstraction corresponding to an abstraction of state, we augment the classic Reacher environment with an additional subspace given by the plane $[0, 1]^2$. The "position" of the agent in this subspace is represented by the colour of the Reacher tip and base. The agent position along each axis of the augmented space is given by the red and green chromaticities of the Reacher, in RGB colour space, with blue fixed to a particular value. We can thus visualise the abstract state space as a four-way colour gradient, with yellow, red, blue and green at the corners. This state subspace can thus be visualised as the background of the trace plots in Fig. 6.6 (b, c).

In order to control the position in colour space, the agent must manipulate the Reacher tip to corresponding quadrents of the original state space, as depicted in Fig. 6.6 (a). The velocity of the agent's movement in colour space corresponds to the distance between the Reacher tip and the origin. So that the agent does not learn to ignore the colour space by association with features of the original state space (like position of the tip) the controls corresponding to movement in the colour space are randomly rotated by a multiple of $90°$ each episode, so the agent must learn to move to the correct colour quadrant in order to move the right direction in colour space.

To give the agent a complete observation space, while simplifying learning, as in Hafner et al. [2019], the image resolutions of observations are 64×64 pixels of six channels, corresponding to two three-colour-channel observations of consecutive states, which allows the agent to capture the joint velocities of the agent as well as the colour controls (as visualised by the colour panels below the agent) and the colour of the agent base and tip.



(a) Pixel Reacher Environment.

(b) Skills Learned with DIAYN.

(c) Skills Learned with DIVRS.

Figure 6.6: Results on Pixel Reacher environment. The environment is illustrated in (a): the colour of Reacher's tip and base is modified towards one of the four colours, with the colour determined by quadrant currently visited by the tip. The trajectories in the colour space of eight skills obtained with DIAYN and DIVRS are shown in (b) and (c) respectively. Lines with the same colour denote different rollouts of the same skill. DIVRS improves both coverage and consistency of skills in the subspace relevant for performance. Best viewed in colour.

Learning Skills Reflecting the experiments performed on the Ant environment, for pretraining, we fix three goal positions, here reflected as particular colour configurations that the agent must reach in the abstract space. Reflecting the relative difficulty of this task as compared to the Ant navigation task, we make use of a dense reward function for pretraining, with reward given by the negative distance from the agent's colour state to the goal. Our learning architecture is identical to that used in the Ant experiments, using policy evaluation on a random policy for pretraining, and then SAC to train skills. In this setup, however, we learn only eight skills, and must use a convolutional neural network in order to capture the state embedding.

In order to analyse the learned skills, we observe the traces of the agent in colour space, given in Fig. 6.6 (b,c). In order to perform hyperparameter tuning in this setting, we select the set of skills with the highest mutual information between skills and (embedded) states as measured on test rollouts. Supporting the effectiveness of our learned abstraction procedure the skills learned by DIVRS are largely consistent across two rollouts of each skill, and manage to cover the abstract state space well, while skills learned without state abstraction are inconsistent and lead to poor coverage of colour space, corresponding instead to features of the original state space. In this way, we affirm that pretraining using value learning tasks is a general and effective process for discovering state abstraction that leads to meaningful abstract behaviours.

6.7 Supplementary Experiments and Experiment Details

In this section, we cover additional empirical evaluations which justify some of the architectural decisions that we made in evaluating DIVRS. In particular, we conduct an experiment which demonstrates that the forward MI objective is more conducive to learning consistent, composable skills, and an experiment which demonstrates the need for a distillation phase to compress our learned state abstraction.

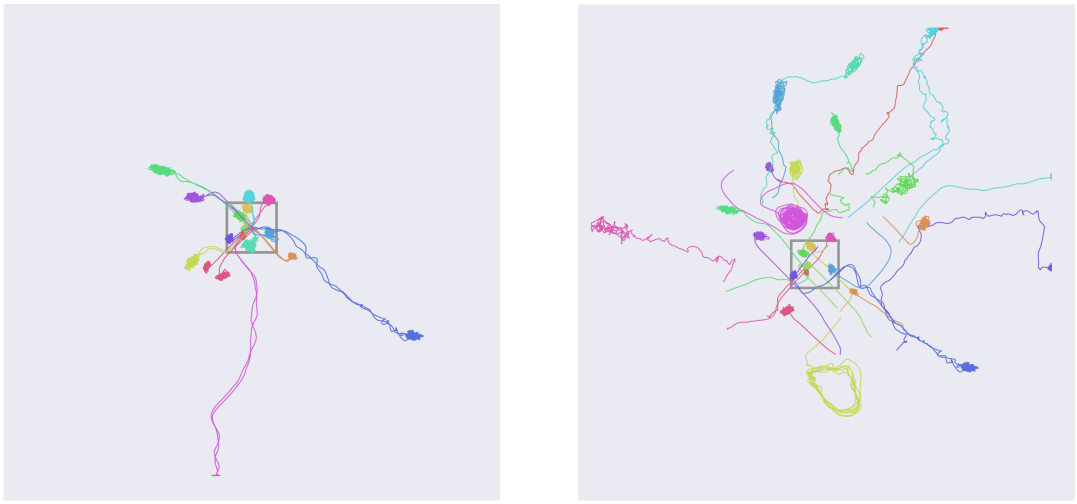
6.7.1 Forward Mutual Information and Skill Consistency

Initial experiments in the HRL environment from Section 6.6.2 made use of skills trained using the reverse MI objective. However, we found that skills learned in this way occasionally performed poorly when the policy over options selected skills in regions of state space which the skills had not been trained for.

In order to investigate the cause of this phenomenon, we conducted experiments in a simple point mass environment, in which the agent state is given only by the position in x-y space, and has direct control over its velocity in each dimension. Since the position is the only state feature, this setup controlled for confounds in representation learning, as VOD skills learned here must move the agent to distinct locations in order to be discriminable.

Figure 6.7 demonstrates that if skills trained with a reverse MI objective are learned with the agent starting in a fixed state, skill behaviour becomes unpredictable and inconsistent when applied to random initial states. Even skills that appear to have a well-defined and consistent behaviour when initialised from a consistent starting state, as in Fig. 6.7a do not exhibit consistent behaviour in regions of state space that they were not trained on, causing unexpected and erratic dynamics and leading to ineffective skills (Fig. 6.7b).

In order to correct this, we trained skills starting from a random initial state distribution. However we found that skills still did not behave consistently across state space. We found that this was caused by the use of the reverse MI objective. Since we have a powerful discriminator network optimised to classify skills by the states that they visit, it



(a) Skills learned using reverse MI (fixed initial state).

(b) The same skills, run on a random initial state distribution.

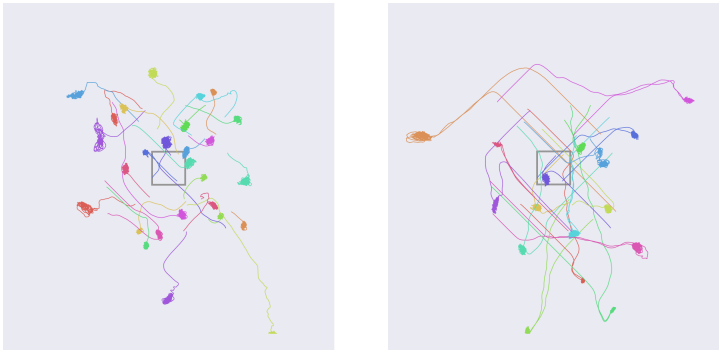
Figure 6.7: Effect of learning skills from a fixed starting state. 16 Skills were learned using the reverse mutual information (MI) objectives. Each colour corresponds to a distinct skill. Skills learned in this way lead to consistent behaviour when initialised from a fixed state, but any given skill becomes inconsistent and ill-defined on states not visited by that skill during training.

is possible for this classifier to learn a multi-modal partitioning of states which correspond to a single skill.

In contrast, for the forward mutual information objective, we can explicitly select a unimodal distribution over states as a parameterisation for each skill. We found that this choice, combined with the random initialisation described above, led to skills which were able to consistently reach specific regions of state space, regardless of where they were initialised.

We demonstrate this effect in Fig. 6.8. We see that skills learned using reverse MI trained on random starting states still does not lead to consistent regions occupied by particular skills (Fig. 6.8a). On the other hand, skills learned using the forward MI objective lead to skills which terminate in predictable regions of state space, leading to more composable skills.

In general, using a more constrained or less powerful discriminator network could have a similar effect for the reverse MI problem. However, this may lead to a more challenging learning problem due to a less powerful classifier, or, if more complex distributions of



(a) Reverse MI (random initial state).

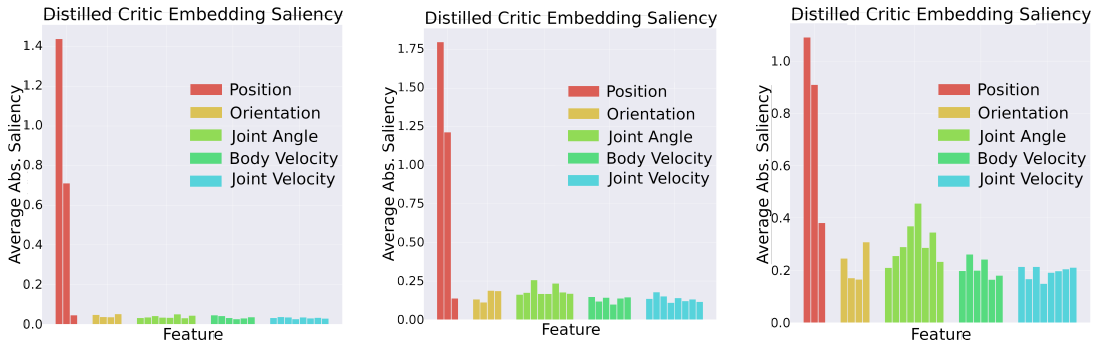
(b) Forward MI (random initial state).

Figure 6.8: The effect of using the forward MI objective as compared to the reverse MI objective, with skill learning initialised from random starting states. Skills learned by reverse MI partition the state space for a single skill into disjoint regions, which depend on the initial state. With the forward MI objectives, skills converge to consistent final states. We note that the agent tends to move diagonally, as it can control velocity in each dimension independently, leading to faster movement along diagonals.

states visited by skills are desired, it may be difficult to express the learning model and objective that correspond to such distributions in the reverse MI setting.

6.7.2 Degree of Compression Tests

Here, we investigate how compression via a distillation phase leads to better state abstraction. To do so we performed an experiment in the Ant domain (Section 6.6.2), in which a representation pretrained on the source tasks was compressed during distillation by using bottlenecks in the distilled network of varying width. We used saliency maps to provide a qualitative understanding of the compression—maps in which features are more peaked indicate better compression, as the learned representation is sensitive to only a few features of the state space. As demonstrated in Fig. 6.9, as the bottleneck gets smaller, the learned representation contains fewer projections from features that are not useful for estimating value on the source tasks. We find that this is necessary for effective skill learning, as otherwise, the irrelevant aspects of state may still heavily influence the overall skills learned.



(a) Bottleneck size: 4. **(b)** Bottleneck size: 16. **(c)** Bottleneck size: 64.

Figure 6.9: Saliency maps of embeddings distilled to varying bottleneck widths. Larger embedding sizes allow more information from aspects of state less important for value prediction to pass through to the learned representation, potentially leading to poorly specified skills during VOD.

6.8 Experiment Details

This section details some of the specifics of the architecture and training procedure used in the experiments above, alongside the hyperparameters employed.

6.8.1 Ant Experiment Details

As is done in Eysenbach et al. [2018], the actuators of the ant robot have their gear ratio reduced to 30, which leads to less erratic motion. Neural networks in this section make use of multi-layer perceptron architectures with ReLU activation functions. Adam [Kingma and Ba, 2014] is used to optimise losses. While in the normal Ant environment, the agent centre of mass is constrained to the $x - z$ plane, in this variant, movement in the y dimension is enabled. In order to keep the state Markov for the locomotion tasks, the state features are augmented with the x-y-z position of the agent.

Pretraining Representations

We learn a representation for a goal seeking agent by initially training on three fixed goals. Every episode, a goal is selected from the three, and remains constant for the duration of the episode. Goal positions act as task indices here. For a visualisation of the environment layout, see Fig. 6.10

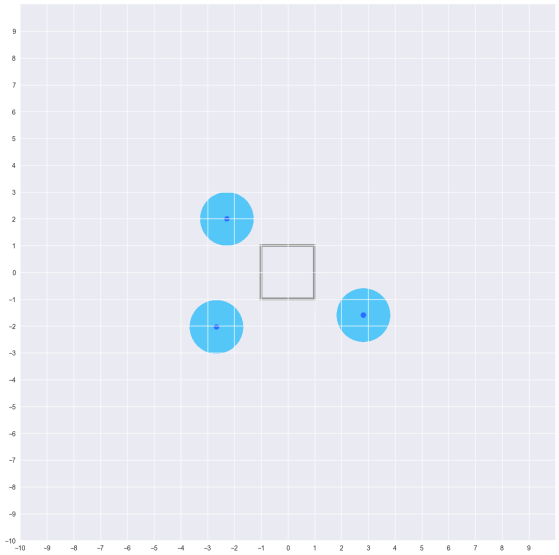


Figure 6.10: Goal positions during pretraining of Ant. Dark blue dots represent goal positions, light blue circles represent the radius at which the ant receives sparse reward and the episode terminates, if the corresponding goal is active. The grey box is for scale comparison with the trace plots in the paper and here.

Our value function network is given by the goal-dependent parameterisation: $V(s, g) = f_a(\psi(s) \odot w(g))$ ψ is a MLP with two hidden layers of size 256, and an output size of 64. This relatively small output size was chosen in order to encourage compression in the learned feature space. w is a MLP with a single hidden layer of size 128, and output of 64 to perform elementwise multiplication with the output of ψ. f_a is then a final MLP with a single hidden layer of size 64, and is trained to output the value of the state for the current task.

During this phase, we train for 20 million time steps, though this much data is not necessary. The value loss remains essentially constant after 500 000 steps. We note that it is not necessary to solve the task, or even learn the optimal value function in this phase, we only need to learn the features that correspond well with the values.

We ran this phase using five random seeds and chose the embedding to distil randomly among them. There was no observable differences between random seeds, they learned consistently, and the saliency maps of the value function were similar for all seeds.

Distillation

The distillation phase consists of training a smaller network with a tight bottleneck to predict the output of the learned embedding ψ from the state. This is done to ensure that the absolute minimal amount of information from the state is encoded in our final learned embedding, $\bar{\psi}$. The distillation was trained on states from the same distribution as the original policy was trained on. We employed an experience replay buffer of size 1 million, in order to achieve greater data efficiency during this phase.

Our distilled network was a two hidden layer MDP with 256 units in each layer, and a bottleneck size of 8. Smaller bottleneck sizes were tried, though these led to poor saliencies after learning. The output of this bottleneck was then passed through a final linear layer, giving a 64 dimensional output. The distilled network was then trained by regression of this output against the output of ϕ, using mean-squared error loss. L1 regularisation was applied to the embedding output, and weight decay regularisation was applied to the network weights, in order to ensure the most terse and well-generalising model possible. Training was performed for 5 million time steps. Four random seeds were used, and again no significant differences were found across random seeds in terms of MSE or saliencies of learned feature maps.

Skill Learning

In this phase we learn skills that are diverse in either our learned embedding from the previous phase (for DIVRS), from the original state space (DIAYN), or a hand constructed x-y space (DIAYN + XY).

Skills were trained using SAC [Haarnoja et al., 2018] with the "double Q trick" for 3 million time steps. When learning policies and values for skills on replay buffer data, the reward function was taken from the current discriminator parameters, rather than those from when the data was collected.

For policy and value networks that condition on the active skill, a one-hot representation of the active skill is passed to a small MLP with a single hidden layer of size 64 and output dimension of 32. This embedding is concatenated with the environment observation, and

passed through a three hidden-layer MLP, with the first two hidden layers of size 300, and the last of size 64, before mapping to the action space of dimension 16 for the policy network, or to a Q-value, in the case of the value network.

For reverse MI experiments, the discriminator is a two hidden layer MLP with 300 units per hidden layer. It takes in the state or embedding of state, and maps to logits for the probability that each skill is active in that state. Forward MI experiments used a lookup table corresponding to a state in the relevant space for each skill.

Five seeds were used for each skill learning procedure. Hyperparameters were selected to optimize pseudo-reward at the 3 million timestep mark. Skills were fairly consistent across seeds, though they differed somewhat in position and orientation of particular skills.

Quantitative Skill Evaluation

For quantitative evaluation of skills, we employed a sparse sequential HRL learning task, in which skills had to be composed to achieve rewards. In order to keep the state Markov, we include a one hot encoding of the number of goals reached, concatenated with the agent observation space as input to the SMDP-level policies, as well as to the flat baseline agents. Training was done for 5 million time steps. HRL agents were learned using a variant of SAC designed for discrete action spaces. This version allows for explicit computation of entropy terms, rather than the sample-based version used in continuous control.

Due to the combinatorial nature of random seeding across the several phases, for these experiments, we did not select for skills across the random seeds in the skill learning phase directly. We trained across two random seeds for 12 different hyperparameter and skillset combinations. We found that the hyperparameter ranges that we employed generally did not change learning significantly. For DIVRS, we report the average performance across hyperparameters and seeds and for baselines report the (more favorable) average of the top three most successful hyperparameter configurations, which themselves are determined by the average performance across the two seeds.

The networks used for the SMDP level policy consisted of two hidden layers of size 128, followed by a hidden layer of size 64, before outputting logits for the skill selection distribution, or for the Q values for each skill.

Ant Hyperparameters

Pretraining	Value
Environment Steps per Iteration	40
Environment Threads	64
GAE λ	0.99
Learning Rate	$8.96e-5$
Batches per Iteration	4
Gradient Clipping Norm	0.5
PPO Clipping Epsilon	0.2
Batch Size	1280

Learning Skills	Value
Learning Rate	$3e-4$
Environment Steps per Iteration	1000

6.8.2 Pixel Reacher Experiment Details

Here, we provide implementation details of the experiments performed in the Cooking Reacher environment. As was done in the Ant environment, the gear ratio of the simulated motors was lowered, this time to 80. ReLU activation functions were used for all networks, and Adam was the optimiser.

Pretraining

The pretraining procedure here is similar to the one employed in the Ant environment. First our agent is trained to reach three fixed goals in flavour space. Goal information is passed to the agent as a separate feature from the pixel state, so that our learned embedding space is not goal-dependent. Pixel information is processed via a two layer convolutional network, with a fixed kernel size of $(3, 3)$, no padding, and a stride of 2. The first layer has 64 channels, while the second has 32. From here, there is a fully connected layer of size 64. This network all together forms ϕ. The output this is elementwise multiplied by

the goal embedding, which is identical to w in the Ant experiments, though now the input is in $[0, 1]^2$ to correspond with the flavour space. This phase lasts 2m frames.

Distillation

Distillation follows identically to the process described in the Ant section, but instead of an MLP, the convolutional network described in the previous section is employed. Again, a bottleneck of size 8 was used. Distillation was done for 3m frames.

Skill Learning

As in the Ant environment, the one-hot skill index is passed through a small MLP. The resulting embedding is concatenated with the output of our pixel observation CNN, which has the same architecture as the ones used in previous phases, though here the hidden layer is of size 300. In the case of the learned Q function, the action is concatenated here as well. From here, both the critic and policy networks have a single fully connected layer of size 300, which is then mapped to either a predicted value, or a distribution over actions. For the DIAYN baseline, the discriminator is identical to the network described here, though the skill index is not concatenated.

Pixel Reacher Hyperparameters

Pretraining	Value
Environment Steps per Iteration	1000
Environment Threads	1
Learning Rate	$8e-5$
Batches per Iteration	1
Batch Size	1280

Learning Skills	Value
Learning Rate	$1e-4$
Environment Steps per Iteration	1000

6.9 Related work on Representation Learning

This chapter has focused heavily on the discovery of state abstraction and the relationship between state abstraction and specification of effective temporal abstraction. Previous works concerning state abstraction in RL include Abel et al. [2020], Li et al. [2006], Abel et al. [2016], Jong and Stone [2005], Igl et al. [2019]. These works share a common thread of looking to build effective state-action abstraction to improve the ease of learning or generalisability of action-value functions. Our approach, on the other hand, looks to learn a general state abstraction so that a VOD method may later on partition the state space in accordance with this abstract representation.

Like DIVRS, Abel et al. [2019] also utilise an information theoretic trade-off between state compression and quality of the representation, though their approach is focused on a single task setting, and involves direct solving of the abstract MDP, while the state abstraction here is used to learn behavioural abstraction. Silver et al. [2017a] and Oh et al. [2017] also learn a state abstraction, but look to use this abstraction in order to form a model of the environment, which is more challenging than the approach used here. By contrast, Liu et al. [2020] and Goyal et al. [2019] make use of a multi-task setting in order to capture goal information in the abstract space, while our approach seeks to be explicitly goal-independent. Similar to our work, Sermanet et al. [2018] make use of state abstraction to generate a pseudo-reward for policy learning, though their approach uses metric learning techniques for viewpoint independence rather than skills.

Other cosmetic similarities exist between the approach that we employ here and the successor representation framework [Schaul et al., 2015, Barreto et al., 2017, Borsa et al., 2019], as both use value functions with simple, goal-independent factorisation. However, in the sucessor representation setting, the learned subspace must capture much more state information and is used to perform policy evaluation for arbitrary possible reward functions, rather than being used to compress the state space into a form amenable for task-based partitioning via learned skills.

6.10 Limitations of DIVRS

While we have clearly demonstrated the effectiveness of DIVRS in the process of learning useful skills, it is important to note that it is not a universal solution and is best used in the setting for which it is designed. One key limitation is the fact that in order for effective learning to occur, it must be possible to select source tasks on which effective policy evaluation can be performed. If a distribution over tasks is part of the problem setting, then random sampling can be performed. While this is a drawback of much of the transfer learning literature, it may not always be clear which tasks are appropriate for pretraining, or these tasks may be challenging policy evaluation problems in their own right.

When the relevant features for the diversity objective are known and can be easily specified, DIVRS is not an appropriate method to use, as the additional burden of learning good features is unnecessary. Our multi-phased approach, while adding stability to the training process and conceptual simplicity, can also make hyperparameter tuning difficult, as hyperparameters for early phases of learning must be jointly tuned with hyperparameters in later phases, leading to combinatorial complexity in the size of the tuning space.

Finally, DIVRS relies on good coverage of the state space during both pretraining and VOD phases. We found that the use of a random policy during pretraining and entropy regularisation during skill learning were sufficient, but in general, better covering policies may be needed. We view this as an important problem, but an orthogonal one to the insight provided here on the relationship between temporal and state abstraction. In future work, we could ensure good exploratory policies through the use of methods such as marginal state matching [Lee et al., 2019] or pseudo-counts Bellemare et al. [2016].

6.11 Conclusions

Drawing on the relationship between state abstraction and behavioural abstraction, this chapter has developed a novel method, DIVRS, for automatic discovery of a feature space which leads to specification of skills that will be useful for downstream learning. In contrast to previous related methods, which operate in the MDP state space and are thus

prone to underspecification and trivial skill learning, skills learned with DIVRS make use of a few tasks and associated rewards in order to learn skills that manipulate a compressed abstract representation of state. This state abstraction, which corresponds to features useful for predicting the value of the source tasks, specifies the sorts of states that are different in ways that relate to the actual task at hand.

Our multi-task setting has allowed us to characterise generalisation in RL in a form very different from the other chapters, namely how we can build abstract structures for reasoning over our own behaviour in one environment or task—and expect this structure to lead to good performance or fast learning in an entirely different task. While theoretical characterisation of this sort of generalisation is challenging, especially without assuming that the target tasks are somehow "close" to the source tasks, we have demonstrated empirically across several environments that this sort of transfer is indeed possible. It is the explicit inclusion of a learning phase in which we discover state abstraction for the purposes of skill specification that allow the learned behaviours to generalise well without explicit guarantees on the relationship between the source and target tasks.

Options are entirely specified in terms of functions of states—termination, the option policy, and membership in the initiation set, unless trivial, will all thus depend implicitly on the state abstraction that is used. While our method approaches this question in its most fundamental form, where options are defined by membership in maximally disjoint sets of states, the broader relationship between temporal abstraction and state abstraction is largely unstudied in the literature. It is our hope that this chapter represents a starting point in a more thorough study of the dynamic interplay of these high-level modes of reasoning.

7

Future Work and Conclusions

Our contributions have sought to advance understanding of the complexity of different forms of abstraction present in RL, and perhaps more importantly, how particular forms of abstraction constrain or enable other forms of abstraction. Along the way we have made significant algorithmic, methodological, and analytical advances in the spaces of offline imitation learning, deep RL with target networks, and variational option discovery. Yet, there remains work to be done. In the next section we detail some of the open problems that relate to the work done in this thesis.

7.1 Future Work

In broad strokes, the development of language around and focus on abstraction in this thesis paves the way for a more formal study of the relationships between different forms of abstraction in RL. Theoretical discussion of optimal representations for off-policy learning would combine state and data abstraction in a powerful way. Orthogonally, there is little formal investigation of the usefulness of options for transfer learning in tabular settings, which could clarify more precisely the relationship between temporal and data abstraction, without concerns for state abstraction.

On a more technical level, the most clear open question to be drawn from this work is concerned with the development of novel target network update schemes. While Polyak averaging schemes which smooth updates have been used alongside the periodic updates discussed here, the importance of the target network as the object being optimised is a novelty which has not been previously understood. This opens the gates for algorithmic development of improved target network update schemes. Our initial experiments using target networks with a momentum-style update lend support to this.

In the space of HRL, we hope that our focus on state abstraction provides fruitful insight on the role of representation in temporal abstraction. Future work could build on this by integrating explicit representation learning components in the termination function and option policies. More directly, our skill learning setup trivialises many aspects of the options framework, including initiation sets and, more importantly, the termination function. We expect that combining some of the feature learning ideas here with learning termination functions will lead to more integrated and more composable skills.

Regarding the question of offline imitation learning, while we found empirical success with continuous environments using our heuristic approach in Chapter 4, our theoretical bounds do not extend to this setting. Generalising the bound to this setting is nontrivial, as the support for the expert data will likely not intersect with the support for the exploratory data. In addition, our data complexity bound is prohibitively high in $\frac{1}{\epsilon}$, which comes from the need to transition from the discounted setting to the average reward setting. While we consider the average reward setting of our algorithm as a novelty in comparison with other OIL approaches, we may find tighter bounds are possible if we use a discounted IL objective. On the other hand making use of an average reward RL algorithm may lead to tighter bounds with respect to the average reward IL objective, though this may prove challenging as average reward value iteration is not provably convergent in general.

Another avenue for exploration of data abstraction in RL could turn to stability analysis of RL, taking inspiration from the close relationship between stability and generalisation in supervised learning. This could be used to capture a more concrete notion of overfitting

in RL, which is integral to the study of generalisation in supervised learning, but thus far largely absent in theoretical RL.

7.2 Conclusions

Abstraction provides a foundation for learning and executing complex behaviour. If we understand abstraction, we can build more sophisticated and stable forms of learning. In this thesis, we have looked to investigate and expend these foundations across many dimensions, including data abstraction, state abstraction, and temporal abstraction.

In the case of data abstraction, we developed a simple algorithm with theoretical guarantees in the tabular offline imitation learning setting, which allowed us to focus on the data cost that comes alongside the need to evaluate a policy using data gathered from the policy itself. By setting our work in the average reward setting, we capture the most long-term notion of imitating expert behaviour, a rarity in the field to date. Alongside this, we highlighted an important connection between state and data abstraction in the need to be able to tune policy evaluation algorithms offline in order to avoid information leak. Our proposed evaluation protocol moves towards a more principled, data transparent, and fair approach to hyperparameter selection and tuning in offline RL settings.

Focusing on state abstraction, we sought to answer one of the most fundamental questions of deep RL—why do target networks stabilise deep RL, despite being subject to the deadly triad? To do so we capture a formal notion of the deadly triad, helping characterise why state abstraction and data abstraction (in the off-policy learning sense) combine to lead to divergence of TD methods. Countering this, we provide the first finite-time bounds for a TD-style algorithm with nonlinear function approximation, using the notion of a path-mean Jacobian and unifying the target network and value network updates in order to do so. Furthermore we demonstrated that when hyperparameters are correctly tuned, under fixed update step sizes, the use of a target network can lead to a nondivergent algorithm which converges to a ball of fixed radius around a local optimum, even when TD(0) diverges.

Finally we brought together all three forms of abstraction by using a pretraining approach to representation learning, leading to skills useful in a transfer setting. Here we saw that skills trained using VOD methods learn to be diverse, but they may be diverse in meaningless ways. By recognising that the issue was one of misspecification due to a lack of state abstraction, we were able to address this by developing a procedure for learning state abstraction which specifies *how* skills should differ. These skills were then used as a vehicle for curriculum learning, carrying fundamental information about agent locomotion from easily solved tasks to challenging problems which cannot be solved without the use of temporal abstraction, demonstrating the strong links between data abstraction and temporal abstraction in RL.

In closing, we simply marvel at the complex and interconnected nature of abstraction in RL. Progress in this field over the past decade has only accelerated, and as agents reason at an increasingly abstract level, they are able to tackle more and more challenging problems. We hope that our work has contributed to unravelling the deep mysteries underpinning how meaningful behaviour can be learned. More fundamentally, we hope that RL as a developing field is deployed in ways that help those most in need—with conscientiousness, care, and curiosity guiding research directions to come.

Bibliography

David Abel, David Hershkowitz, and Michael Littman. Near optimal behavior via approximate state abstraction. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2915–2923, 2016.

David Abel, Dilip Arumugam, Kavosh Asadi, Yuu Jinnai, Michael L Littman, and Lawson LS Wong. State abstraction as compression in apprenticeship learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3134–3142, 2019.

David Abel, Nate Umbanhowar, Khimya Khetarpal, Dilip Arumugam, Doina Precup, and Michael Littman. Value preserving state-action abstractions. In *International Conference on Artificial Intelligence and Statistics*, pages 1639–1650. PMLR, 2020.

Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

David H Ackley and Michael L Littman. Generalization and scaling in reinforcement learning. In *Advances in neural information processing systems*, volume 2, pages 550–557, 1989.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving Rubik's Cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. *arXiv preprint arXiv:1612.00410*, 2016.

David Andre and Stuart J. Russell. State abstraction for programmable reinforcement learning agents. In *AAAI/IAAI*, 2002. URL `https://api.semanticscholar.org/CorpusID:1479889`.

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.

Okan Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Trans. Graph.*, 21(3):483–490, jul 2002. ISSN 0730-0301. doi: 10.1145/566654.566606. URL `https://doi.org/10.1145/566654.566606`.

Peter Auer, Thomas Jaksch, and Ronald Ortner. Near-optimal regret bounds for reinforcement learning. In *Advances in neural information processing systems*, pages 89–96, 2009.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML 1995)*, pages 30–37, San Francisco, CA, USA, 1995. Morgan Kauffman. ISBN 1-55860-377-8. URL http://leemon.com/papers/1995b.pdf.

Stefan Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta mathematicae*, 3(1):133–181, 1922.

André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, pages 4055–4065, 2017.

Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563, 1966.

Marc G Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *arXiv preprint arXiv:1606.01868*, 2016.

Richard Bellman. On the theory of dynamic programming. *Proceedings of the national Academy of Sciences*, 38(8):716–719, 1952.

Dimitri P Bertsekas and Sergey Ioffe. Temporal differences-based policy iteration and applications in neuro-dynamic programming. *Lab. for Info. and Decision Systems Report LIDS-P-2349, MIT, Cambridge, MA*, 14, 1996.

Jalaj Bhandari, Daniel Russo, and Raghav Singal. A finite time analysis of temporal difference learning with linear function approximation. In *Proceedings of the 31st Conference On Learning Theory*, volume 75 of *Proceedings of Machine Learning Research*, pages 1691–1692. PMLR, 06–09 Jul 2018. URL https://proceedings.mlr.press/v75/bhandari18a.html.

V.S. Borkar. *Stochastic Approximation: A Dynamical Systems Viewpoint*. Cambridge University Press, 2008. ISBN 9780521515924. URL https://books.google.co.uk/books?id=QLxIvgAACAAJ.

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.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Matthew Botvinick, Sam Ritter, Jane X Wang, Zeb Kurth-Nelson, Charles Blundell, and Demis Hassabis. Reinforcement learning, fast and slow. *Trends in cognitive sciences*, 23(5):408–422, 2019.

Justin A Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pages 369–376, 1995.

Steven J Bradtke and Andrew G Barto. Linear least-squares algorithms for temporal difference learning. *Machine learning*, 22(1-3):33–57, 1996.

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

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Emma Brunskill and Lihong Li. Pac-inspired option discovery in lifelong reinforcement learning. In *International conference on machine learning*, pages 316–324, 2014.

Víctor Campos, Alexander Trott, Caiming Xiong, Richard Socher, Xavier Giro-i Nieto, and Jordi Torres. Explore, discover and learn: Unsupervised discovery of state-covering skills. *arXiv preprint arXiv:2002.03647*, 2020.

Diogo Carvalho, Francisco S. Melo, and Pedro Santos. A new convergent variant of Q-learning with linear function approximation. In *Advances in Neural Information Processing Systems*, volume 33, pages 19412–19421. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/e1696007be4eefb81b1a1d39ce48681b-Paper.pdf.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

Petros Christodoulou. Soft actor-critic for discrete action settings. *ArXiv*, abs/1910.07207, 2019. URL https://api.semanticscholar.org/CorpusID:204734462.

Kamil Ciosek. Imitation learning by reinforcement learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=1zwleytEpYx.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019a.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1282–1289. PMLR, 09–15 Jun 2019b. URL https://proceedings.mlr.press/v97/cobbe19a.html.

Gal Dalal, Balázs Szörényi, Gugan Thoppe, and Shie Mannor. Finite sample analysis for TD (0) with linear function approximation. *arXiv preprint arXiv:1704.01161*, 2017.

Christian Daniel, Herke Van Hoof, Jan Peters, and Gerhard Neumann. Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 104 (2-3):337–357, 2016.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278, 1993.

Thomas Dean, Robert Givan, and Sonia Leach. Model reduction techniques for computing approximately optimal solutions for markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, UAI'97, page 124–131, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1558604855.

Thomas Dietterich. State abstraction in MAXQ hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/e5a4d6bf330f23a8707bb0d6001dfbe8-Paper.pdf.

Thomas G. Dietterich. The MAXQ method for hierarchical reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998*, pages 118–126. Morgan Kaufmann, 1998.

Thomas G. Dietterich. An overview of MAXQ hierarchical reinforcement learning. In *Abstraction, Reformulation, and Approximation*, pages 26–44, Berlin, Heidelberg, 2000a. Springer Berlin Heidelberg. ISBN 978-3-540-44914-0.

Thomas G Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000b.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

Sadegh Esmaeil Zadeh Soudjani and Alessandro Abate. Adaptive and sequential gridding procedures for the abstraction and verification of stochastic processes. *SIAM Journal on Applied Dynamical Systems*, 12(2):921–956, 2013. doi: 10.1137/120871456. URL https://doi.org/10.1137/120871456.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function, 2018.

Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A theoretical analysis of deep Q-learning. In *Learning for dynamics and control*, pages 486–489. PMLR, 2020.

Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in DQN. *arXiv preprint arXiv:1810.00123*, 2018.

Mattie Fellows, Kristian Hartikainen, and Shimon Whiteson. Bayesian bellman operators. *Advances in Neural Information Processing Systems*, 34:13641–13656, 2021.

Mattie Fellows, Matthew J. A. Smith, and Shimon Whiteson. Why target networks stabilise temporal difference methods. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 9886–9909. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/fellows23a.html.

Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*, 2016.

Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017.

Vincent François-Lavet, David Taralla, Damien Ernst, and Raphaël Fonteneau. Deep reinforcement learning solutions for energy microgrids management. In *European Workshop on Reinforcement Learning (EWRL 2016)*, 2016.

Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3-4):219–354, 2018. ISSN 1935-8245. doi: 10.1561/2200000071. URL http://dx.doi.org/10.1561/2200000071.

Vincent Francois-Lavet, Guillaume Rabusseau, Joelle Pineau, Damien Ernst, and Raphael Fonteneau. On overfitting and asymptotic bias in batch reinforcement learning with partial observability. *Journal of Artificial Intelligence Research*, 65:1–30, May 2019. ISSN 1076-9757. doi: 10.1613/jair.1.11478. URL http://dx.doi.org/10.1613/jair.1.11478.

Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767*, 2017.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: Datasets for deep data-driven reinforcement learning, 2020.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/fujimoto18a.html.

Alexandre Galashov, Siddhant M Jayakumar, Leonard Hasenclever, Dhruva Tirumala, Jonathan Schwarz, Guillaume Desjardins, Wojciech M Czarnecki, Yee Whye Teh, Razvan Pascanu, and Nicolas Heess. Information asymmetry in KL-regularized RL. *arXiv preprint arXiv:1905.01240*, 2019.

Jason Gauci, Edoardo Conti, Yitao Liang, Kittipat Virochsiri, Yuchen He, Zachary Kaden, Vivek Narayanan, Xiaohui Ye, Zhengxing Chen, and Scott Fujimoto. Horizon: Facebook's open source applied reinforcement learning platform. *arXiv preprint arXiv:1811.00260*, 2018.

Sina Ghiassian, Andrew Patterson, Shivam Garg, Dhawal Gupta, Adam White, and Martha White. Gradient temporal-difference learning with regularized corrections. In *International Conference on Machine Learning*, pages 3524–3534. PMLR, 2020.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Anirudh Goyal, Riashat Islam, Daniel Strouse, Zafarali Ahmed, Matthew Botvinick, Hugo Larochelle, Yoshua Bengio, and Sergey Levine. Infobot: Transfer and exploration via the information bottleneck. *arXiv preprint arXiv:1901.10902*, 2019.

Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1352–1361. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/haarnoja17a.html.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

Anna Harutyunyan, Will Dabney, Diana Borsa, Nicolas Heess, Remi Munos, and Doina Precup. The termination critic. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2231–2240, 2019.

Hado Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. URL https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf.

Albin Heimerson, Johannes Sjölund, Rickard Brännvall, Jonas Gustafsson, and Johan Eker. Adaptive control of data center cooling using deep reinforcement learning. In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, pages 1–6, 2022. doi: 10.1109/ACSOSC56246.2022.00018.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

Ronald A Howard. *Dynamic programming and Markov processes.* John Wiley, 1960.

Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. *Advances in neural information processing systems*, 32, 2019.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. The impact of non-stationarity on generalisation in deep reinforcement learning. *arXiv preprint arXiv:2006.05826*, 2020a.

Maximilian Igl, Andrew Gambardella, Jinke He, Nantas Nardelli, N Siddharth, Wendelin Böhmer, and Shimon Whiteson. Multitask soft option learning. In *Conference on Uncertainty in Artificial Intelligence*, pages 969–978. PMLR, 2020b.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, John Langford, and Robert E Schapire. Contextual decision processes with low bellman rank are PAC-learnable. In *International Conference on Machine Learning*, pages 1704–1713. PMLR, 2017.

Yuu Jinnai, Jee Won Park, David Abel, and George Konidaris. Discovering options for exploration by minimizing cover time. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3130–3139. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/jinnai19b.html.

Nicholas K Jong and Peter Stone. State abstraction discovery from irrelevant state variables. In *IJCAI*, volume 8, pages 752–757. Citeseer, 2005.

Nicholas K Jong, Todd Hester, and Peter Stone. The utility of temporal abstraction in reinforcement learning. In *AAMAS (1)*, pages 299–306. Citeseer, 2008.

Sham Kakade. Optimizing average reward using discounted rewards. In *International Conference on Computational Learning Theory*, pages 605–615. Springer, 2001.

Sham M Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538, 2002.

Sham Machandranath Kakade. *On the sample complexity of reinforcement learning*. University of London, University College London (United Kingdom), 2003.

Michael Kearns and Satinder Singh. Finite-sample convergence rates for Q-learning and indirect algorithms. *Advances in neural information processing systems*, 11, 1998.

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

Md Al-Masrur Khan, Md Rashed Jaowad Khan, Abul Tooshil, Niloy Sikder, MA Parvez Mahmud, Abbas Z Kouzani, and Abdullah-Al Nahid. A systematic review on reinforcement learning-based robotics within the last decade. *IEEE Access*, 8:176598–176623, 2020.

Geon-Hyeong Kim, Seokin Seo, Jongmin Lee, Wonseok Jeon, HyeongJoo Hwang, Hongseok Yang, and Kee-Eung Kim. Demodice: Offline imitation learning with supplementary imperfect demonstrations. In *International Conference on Learning Representations*, 2021.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149, 2003.

Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2010.

Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. doi: 10.1017/CBO9780511546877.

Hoang Le, Cameron Voloshin, and Yisong Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, pages 3703–3712. PMLR, 2019.

Donghwan Lee and Niao He. Target-based temporal-difference learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3713–3722. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/lee19a.html`.

Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*, 2019.

David A Levin and Yuval Peres. *Markov Chains and Mixing Times*. American Mathematical Society, second edition, 2017.

Kfir Y Levy and Nahum Shimkin. Unified inter and intra options learning using policy gradient methods. In *European Workshop on Reinforcement Learning*, pages 153–164. Springer, 2011.

Lihong Li, Thomas Walsh, and Michael Littman. Towards a unified theory of state abstraction for MDPs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, 2006.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Evan Zheran Liu, Aditi Raghunathan, Percy Liang, and Chelsea Finn. Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices. *arXiv preprint arXiv:2008.02790*, 2020.

Qiang Liu, Lihong Li, Ziyang Tang, and Dengyong Zhou. Breaking the curse of horizon: Infinite-horizon off-policy estimation. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/dda04f9d634145a9c68d5dfe53b21272-Paper.pdf.

Zhuang Liu, Xuanlin Li, Bingyi Kang, and Trevor Darrell. Regularization matters in policy optimization. *arXiv preprint arXiv:1910.09191*, 2019.

Yecheng Jason Ma, Andrew Shen, Dinesh Jayaraman, and Osbert Bastani. SMODICE: Versatile offline imitation learning via state occupancy matching. *arXiv preprint arXiv:2202.02433*, 2022.

Marlos C Machado, Marc G Bellemare, and Michael Bowling. A Laplacian framework for option discovery in reinforcement learning. *arXiv preprint arXiv:1703.00956*, 2017.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61: 523–562, 2018a.

Marlos C. Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell. Eigenoption discovery through the deep successor representation. In *International Conference on Learning Representations*, 2018b. URL https://openreview.net/forum?id=Bk8ZcAxR-.

Hamid Maei, Csaba Szepesvari, Shalabh Bhatnagar, Doina Precup, David Silver, and Richard S Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. *Advances in neural information processing systems*, 22, 2009.

Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Recent advances in reinforcement learning*, pages 159–195, 1996.

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(Oct):2169–2231, 2007.

Daniel J Mankowitz, Timothy A Mann, and Shie Mannor. Adaptive skills adaptive partitions (asap). In *Advances in Neural Information Processing Systems*, pages 1588–1596, 2016.

Timothy Mann and Shie Mannor. Scaling up approximate value iteration with options: Better policies with fewer iterations. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 127–135. PMLR, 2014.

Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 71. ACM, 2004.

Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, page 361–368, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1558607781.

Amy McGovern, Doina Precup, Balaraman Ravindran, Satinder Singh, and Richard S Sutton. Hierarchical optimal control of MDPs. In *Proceedings of the Tenth Yale Workshop on Adaptive and Learning Systems*, pages 186–191, 1998.

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, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(May):815–857, 2008.

Ofir Nachum, Shixiang (Shane) Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/e6384711491713d29bc63fc5eeb5ba4f-Paper.pdf.

Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. DualDICE: Behavior-agnostic estimation of discounted stationary distribution corrections. *Advances in Neural Information Processing Systems*, 32, 2019a.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. In *International Conference on Learning Representations*, 2019b. URL https://openreview.net/forum?id=H1emus0qF7.

Ofir Nachum, Haoran Tang, Xingyu Lu, Shixiang Gu, Honglak Lee, and Sergey Levine. Why does hierarchy (sometimes) work so well in reinforcement learning? *arXiv preprint arXiv:1909.10618*, 2019c.

A Nedić and Dimitri P Bertsekas. Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, 13(1):79–110, 2003.

Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, page 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1558607072.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. *arXiv preprint arXiv:1707.03497*, 2017.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyu Wang, and Nando de Freitas. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*, 2020.

Ronald Parr and Stuart J Russell. Reinforcement learning with hierarchies of machines. In *Advances in neural information processing systems*, pages 1043–1049, 1998.

Robin Pemantle. Nonconvergence to unstable points in urn models and stochastic approximations. *The Annals of Probability*, 18(2):698 – 712, 1990. URL `https://doi.org/10.1214/aop/1176990853`.

Marc Pickett and Andrew G. Barto. Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, page 506–513, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1558608737.

Boris Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr Computational Mathematics and Mathematical Physics*, 4:1–17, 12 1964. doi: 10.1016/0041-5553(64)90137-5.

Doina Precup, Richard S Sutton, and Satinder Singh. Theoretical results on reinforcement learning with temporally abstract options. In *European conference on machine learning*, pages 382–393. Springer, 1998.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Siddharth Reddy, Anca D Dragan, and Sergey Levine. SQIL: Imitation learning via reinforcement learning with sparse rewards. *arXiv preprint arXiv:1905.11108*, 2019.

Stephane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL `https://proceedings.mlr.press/v9/ross10a.html`.

John Rust. Optimal replacement of GMC bus engines: An empirical model of Harold Zurcher. *Econometrica: Journal of the Econometric Society*, pages 999–1033, 1987.

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.

Mehmet Sarigül and Mutlu Avci. Performance comparison of different momentum techniques on deep reinforcement learning. *Journal of Information and Telecommunication*, 2(2):205–216, 2018.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/schaul15.html`.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1134–1141. IEEE, 2018.

Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJgLZR4KvH.

Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

David Silver, Hado Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, pages 3191–3199. PMLR, 2017a.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017b.

Özgür Şimşek and Andrew G. Barto. Skill characterization based on betweenness. In *Advances in Neural Information Processing Systems 21*, pages 1497–1504. Curran Associates, Inc., 2009. URL http://papers.nips.cc/paper/3411-skill-characterization-based-on-betweenness.pdf.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.1556.

Matthew Smith, Herke van Hoof, and Joelle Pineau. An inference-based policy gradient method for learning options. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4703–4712. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/smith18a.html.

Matthew Smith, Lucas Maystre, Zhenwen Dai, and Kamil Ciosek. A strong baseline for batch imitation learning. *arXiv preprint arXiv:2302.02788*, 2023.

Matthew J. A. Smith, Jelena Luketina, Kristian Hartikainen, Maximilian Igl, and Shimon Whiteson. Learning skills diverse in value-relevant features. In *Proceedings of The 1st Conference on Lifelong Learning Agents*, volume 199 of *Proceedings of Machine Learning Research*, pages 1174–1194. PMLR, 22–24 Aug 2022. URL https://proceedings.mlr.press/v199/smith22a.html.

Rayadurgam Srikant and Lei Ying. Finite-time error bounds for linear stochastic approximation and TD learning. In *Conference on Learning Theory*, pages 2803–2830. PMLR, 2019.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pages 212–223. Springer, 2002.

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

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press, 1996. URL http://papers.nips.cc/paper/1109-generalization-in-reinforcement-learning-successful-examples-usi pdf.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Richard S Sutton, Doina Precup, and Satinder P Singh. Intra-option learning about temporally abstract actions. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 556–564, 1998.

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, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

Richard S Sutton, Csaba Szepesvári, and Hamid Reza Maei. A convergent O(n) algorithm for off-policy temporal-difference learning with linear function approximation. *Advances in neural information processing systems*, 21(21):1609–1616, 2008.

Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th annual international conference on machine learning*, pages 993–1000, 2009.

Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

Yee 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*, pages 4499–4509, 2017.

Gerald Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3):58–68, 1995. ISSN 0001-0782. doi: 10.1145/203330.203343. URL https://doi.org/10.1145/203330.203343.

Sebastian Thrun and Anton Schwartz. Finding structure in reinforcement learning. In *Advances in neural information processing systems*, pages 385–392, 1995.

Dhruva Tirumala, Hyeonwoo Noh, Alexandre Galashov, Leonard Hasenclever, Arun Ahuja, Greg Wayne, Razvan Pascanu, Yee Whye Teh, and Nicolas Heess. Exploiting hierarchy for learning and transfer in KL-regularized RL. *arXiv preprint arXiv:1903.07438*, 2019.

Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5, 2015. doi: 10.1109/ITW.2015.7133169.

Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

J.N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997. doi: 10.1109/9.580874.

Harm Van Seijen, Hado Van Hasselt, Shimon Whiteson, and Marco Wiering. A theoretical and empirical analysis of expected Sarsa. In *2009 IEEE symposium on adaptive dynamic programming and reinforcement learning*, pages 177–184. IEEE, 2009.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

M Vidyasagar. Convergence of stochastic approximation via martingale and converse Lyapunov methods. *Mathematics of Control, Signals, and Systems*, pages 1–24, 2023.

Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojtek Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the real-time strategy game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019.

Ruohan Wang, Carlo Ciliberto, Pierluigi Vito Amadori, and Yiannis Demiris. Random expert distillation: Imitation learning via expert policy support estimation. In *International Conference on Machine Learning*, pages 6536–6544. PMLR, 2019.

Ruosong Wang, Dean Foster, and Sham M. Kakade. What are the statistical limits of offline RL with linear function approximation? In *International Conference on Learning Representations*, 2021a. URL `https://openreview.net/forum?id=30EvkP2aQLD`.

Ruosong Wang, Yifan Wu, Ruslan Salakhutdinov, and Sham Kakade. Instabilities of offline RL with pre-trained neural representation. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10948–10960. PMLR, 18–24 Jul 2021b. URL `https://proceedings.mlr.press/v139/wang21z.html`.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.

Chengwei Wei, Yun-Cheng Wang, Bin Wang, and C-C Jay Kuo. An overview on language models: Recent developments and outlook. *arXiv preprint arXiv:2303.05759*, 2023.

Shimon Whiteson, Brian Tanner, Matthew E Taylor, and Peter Stone. Protecting against evaluation overfitting in empirical reinforcement learning. In *2011 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*, pages 120–127. IEEE, 2011.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Ronald J Williams and Leemon C Baird III. Analysis of some incremental variants of policy iteration: First steps toward understanding actor-critic learning systems. Technical report, Citeseer, 1993.

Markus Wulfmeier, Abbas Abdolmaleki, Roland Hafner, Jost Tobias Springenberg, Michael Neunert, Tim Hertweck, Thomas Lampe, Noah Siegel, Nicolas Heess, and Martin Riedmiller. Regularized hierarchical policies for compositional transfer in robotics. *arXiv preprint arXiv:1906.11228*, 2019.

Zhuoran Yang, Zuyue Fu, Kaiqing Zhang, and Zhaoran Wang. Convergent reinforcement learning with function approximation: A bilevel optimization perspective, 2019. URL https://openreview.net/forum?id=ryfcCo0ctQ.

Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys*, 55(1), nov 2021. ISSN 0360-0300. doi: 10.1145/3477600. URL https://doi.org/10.1145/3477600.

Huizhen Yu and Dimitri P Bertsekas. Convergence results for some temporal difference methods based on least squares. *IEEE Transactions on Automatic Control*, 54(7): 1515–1531, 2009.

Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018a.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018b.

Shangtong Zhang and Shimon Whiteson. DAC: The double actor-critic architecture for learning options. In *Advances in Neural Information Processing Systems*, pages 2012–2022, 2019.

Shangtong Zhang, Hengshuai Yao, and Shimon Whiteson. Breaking the deadly triad with a target network. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12621–12631. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/zhang21y.html.

Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, page 1433–1438. AAAI Press, 2008. ISBN 9781577353683.

Konrad Zolna, Alexander Novikov, Ksenia Konyushkova, Caglar Gulcehre, Ziyu Wang, Yusuf Aytar, Misha Denil, Nando de Freitas, and Scott Reed. Offline learning from demonstrations and unlabeled experience. *arXiv preprint arXiv:2011.13885*, 2020.