Advancing Data-Efficiency in Reinforcement Learning

Sephora Madjiheurem

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department Electronic and Electrical Engineering

University College London

September 9, 2022

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

To my number one, my Mother.

Acknowledgements

First and foremost, I want to acknowledge and thank my truly amazing PhD advisor, Laura Toni. Throughout my PhD, Laura has shown me nothing but incredible kindness through her support, encouragement, and generosity with her time. Over the course of four years, I met with Laura at least once a week, nearly every week. During that time, I got to benefit from her expertise in graph learning, her inspiring curiosity for the field of reinforcement learning and her genuine passion for doing good and impactful research. I will forever cherish our occasional three to four hours long brainstorming sessions. Through excellent role modeling, she fostered a work environment that I found optimal for conducting my research. I am beyond proud to have been part of her group, as it is a safe, inclusive, stimulating and fun group. Thank you Laura, none of this would have been possible without you!

There are many other incredible people at UCL who helped me during this journey. My second advisor, Miguel Rodrigues, thank you for many enlightening chats about AI, for encouraging me with my research, but also to run my fastest 10K. The head of our group, Izzat Darwazeh, thank you for your encouragements, they always meant so much to me. Additionally, thank you for creating such a social and fun environment for all of us to thrive in. I will never forget all the dinners and other social events you initiated. To the first friends I made at UCL, Silvia Rossi, Hedaia Ghannam and Daniel Mannion, thank you so much for all the good times on and off campus. Thank you Silvia for being the joyous life of our lab, always bringing people together and putting a smile on everyone's face. You've always looked out for me, you've been such a important friend to have during this process. Hedaia, you have countless times helped me push through

those difficult moments, when research got hard. Your incredible resilience and determination have always been a huge source of inspiration. Thank you Dan for geeking out with me on topics a little outside our comfort zones, for being the most reliable dancing partner, and an overall very supportive friend. Many of our conversations helped be to feel grounded and brought me hope and comfort. To many other wonderful people with whom I had the pleasure of sharing the open office at one point or another: Temitope Odedeyi, Afroditi Papadaki, Xinyue Liu, Alan Guedes, Pedro Gomes, Eduardo Pignatelli, Kaige Yang, Nagham Osman, Aya Kayal, Andrea Piccione, Waseem Ozan, Amany Kassem, Emanuele Gruppi and Paul Anthony Haigh. Thank you for the lunch breaks, the tea and coffee breaks, the pints, the dinners. Thank you for showing me that research is better accompanied with moments of laughter and exchange of experiences.

Beyond UCL, there are a few people in academia I have looked up to and who have helped me become a better researcher. Thank you Hado van Hasselt, my supervisor during my internship at DeepMind. I feel very lucky to have had the opportunity of working with you. I have learned so much from your expertise in reinforcement learning, which directly made this thesis better. Thank you for being patient with me and so encouraging. Thank you to my other collaborators at DeepMind, Matteo Hessel for helping me improve my engineering skills, Diana Borsa, for helping me improve my maths skills, and David Silver, for helping me to truly understand what thinking outside the box means. Thank you Marc Deisenroth and Shakir Mohamed whom I had the immense pleasure of meeting at the Machine Learning Summer School 2019, for showing me that there is a place for everyone in the field of AI.

I also would like to acknowledge my dear friends outside of university, for keeping me sane and grounded. Thank you Bilan Elmi, for being and exceptional friend and showing me unconditional support, always. Thank you Briana Camacho, for being there for me even when I could not articulate that I needed a friend. Thank you Cassandra Gelmon, for getting me through the lock-downs and for cheering me on during the final phase of my PhD, when I thought it would never end. Thank

you Annika Schoene, for so much. From starting a podcast with me which helped me find a purpose during a dark time, to literally always answering my calls, thank you. Thank you to my friends in Switzerland, who have continued to support me through my journey long after I left Switzerland. Eyrùn Haraldstòttir, Samantha Juhasz, Nadia Stoppa, Tamara Di Paolo and Imane Elmghari, thank you.

Thank you to my day one supporters, my Family. Thank you to my sisters Olga Madjinodji and Solange Kaïnodji for believing in me and my research even when I failed to explain what it is that I am researching. Thank you to my brother Ader Nodjimadjiel for being the first person I ever looked up to. And, thank you mum, for being my biggest supporter through it all.

Last, but by no means least, I want to thank myself and acknowledge all my hard work. Though this journey was enriching in so many ways and mostly fun throughout, it was not always easy and required some self-discipline and a lot of patience. I did it!

Abstract

In many real-world applications, including traffic control, robotics and web system configurations, we are confronted with real-time decision-making problems where data is limited. Reinforcement Learning (RL) allows us to construct a mathematical framework to solve sequential decision-making problems under uncertainty. Under low-data constraints, RL agents must be able to quickly identify relevant information in the observations, and to quickly learn how to act in order attain their long-term objective. While recent advancements in RL have demonstrated impressive achievements, the end-to-end approach they take favours autonomy and flexibility at the expense of fast learning. To be of practical use, there is an undeniable need to improve the data-efficiency of existing systems.

Ideal RL agents would possess an optimal way of representing their environment, combined with an efficient mechanism for propagating reward signals across the state space. This thesis investigates the problem of data-efficiency in RL from these two aforementioned perspectives. A deep overview of the different representation learning methods in use in RL is provided. The aim of this overview is to categorise the different representation learning approaches and highlight the impact of the representation on data-efficiency. Then, this framing is used to develop two main research directions. The first problem focuses on learning a representation that captures the geometry of the problem. An RL mechanism that uses a scalable feature learning on graphs method to learn such rich representations is introduced, ultimately leading to more efficient value function approximation. Secondly, $ET(\lambda)$, an algorithm that improves credit assignment in stochastic environments by propagating reward information counterfactually is presented. $ET(\lambda)$ results in faster

learning compared to traditional methods that rely solely on temporal credit assignment. Overall, this thesis shows how a structural representation encoding the geometry of the state space, and counterfactual credit assignment are key characteristics for data-efficient RL.

Impact Statement

Undeniably, Artificial Intelligence (AI) has the potential to help towards wide-ranging aspects of human society. In fact, there are already existing efforts in this direction with promising results, such as the development of machine learning models for quickly and accurately forecasting floods in India and Bangladesh to keep people safe and informed [97], or the use of machine learning for automatic monitoring of viral cassava disease in Uganda to help farmers contain the spread of the disease [65]. As it happens, with the growing concerns surrounding the global climate crisis, the field of climate informatics has emerged in the past decade from collaborations between climate scientists and machine learning experts and is devoted to leveraging machine learning to develop tools to analyse complex climate data [105, 127].

While these directions are necessary and should be celebrated, there is an important aspect of AI that remains under-explored: the sustainability and accessibility of AI. Studies have shown that training and running certain types of AI systems requires enormous amounts of data and can lead to large amounts of carbon emissions [140, 50]. Deep Reinforcement Learning (RL) algorithms in particular, while having attained impressive super-human performances on some tasks [102, 135], are not particularly energy nor data efficient. This results in unequal access to AI research, due to the financial cost it incurs, as well as limited impactful deployment of AI systems on complex real-world problems characterised as small-data (such personalised healthcare or education). There is therefore a crucial need for more data-efficient, hence more accessible, AI systems capable of learning in complex domains without requiring large quantities of data.

Over the past four years, I have conducted research towards improving the data-efficiency of reinforcement learning algorithms from different angles. The central claims of this work is that

> **data-efficiency in reinforcement learning can be achieved through structural representation learning and counterfactual credit assignment**.

In this document, I support my thesis statement with an extensive literature review and analysis, different novel algorithms, theoretical justifications and experimental results.

Ultimately, the findings of this research led to more efficient and accessible reinforcement learning. Indeed, the new RL systems are equipped with the ability to process data more effectively, to propagate relevant information quicker, and to adapt faster to new tasks with minimal additional compute or data. This research will not only help to reduce carbon emissions of RL agents by replacing existing energy-hungry systems by the more efficient agents, but it will also make it possible to find solutions for problems where data is limited and exploration is constrained, thus directly contributing to improving the sustainability and accessibly of AI.

My research outcomes have resulted in conference publications and workshop presentations in highly competitive venues (AISTATS, AAAI and ICML), as well as a prestigious Distinguished Paper Award at AAAI 2021.

Contents

Acronyms

AI Artificial Intelligence.

DNN Deep Neural Network.

DP Dynamic Programming.

DQN Deep Q-Network.

GRPI General Representation Policy Iteration.

LSPI Least-Squares Policy Iteration.

MC Monte Carlo.

MDP Markov Decision Process.

ML Machine Learning.

MSE Mean Squared Error.

n2v Node2vec.

NN Neural Network.

PVFs Proto-Value Functions.

RL Reinforcement Learning.

SFs Successor Features.

SR Successor Representation.

TD Temporal-Difference.

List of Figures

List of Tables

Chapter 1

Introduction

Artificial Intelligence (AI) is the theory and development of digital computer systems to perform tasks normally associated with natural intelligence displayed by animals including humans, such as visual perception, natural language understanding, and decision-making. The field of AI emerged in the 1950's, with the successful development of the first known AI programs capable of learning checkers strategies, solving word problems in algebra and proving logical theorems [99]. However, it is not until the 2010's that AI systems developed by researchers started to gain mainstream attention and became of wide practical use. This shift was made possible not only through algorithmic improvements, but primarily by the evolution of faster computers and access to large amounts of data: data-hungry deep learning methods enjoyed a significant rise and continue to dominate accuracy benchmarks to this day. The combination of deep learning and access to large data sets unlocked solutions for complex problems such as machine translation, text-to-speech conversion, information retrieval, object detection and recognition, recommender systems and game playing.

The research undertaken in this thesis is motivated towards creating artificial learners capable of learning from experience with limited data. Data-efficient agents have the potential to benefit wide-ranging aspects of human society, in a way that is presently not attainable due to the exorbitant compute and data requirements of current AI systems. Studies have shown that training and running certain types of AI systems can lead to large amounts of carbon emissions [140, 50]. Deep Rein-

forcement Learning (RL) algorithms in particular, while having attained impressive super-human performances on some tasks [102, 135], are not particularly energy efficient. For example, it has been estimated that first artificial intelligent system to beat the world champion at the ancient game of Go [135] generated 96 tonnes of CO_2 over 40 days of training, the equivalent of approximately 1000 hours of air travel [168]. Not only does this pose a serious sustainability issue in terms of energy consumption, but it also means that this type of research is only accessible to a very limited portion of the machine learning community, due to the financial cost it incurs, and can only be used on problems with practically unlimited data. However, a vast range of problems that human societies face, such as personalised healthcare and education, are characterised as small-data problems and necessitate sample-efficient solutions. There is therefore a crucial need for more data-efficient, hence more accessible, AI systems capable of learning in complex domains without requiring large quantities of data.

Machine Learning (ML) is a fundamental concept in AI. It refers to the study of computer algorithms that automatically learn from data and experience with minimal human intervention. Supervised learning is the branch of ML which learns a mapping from input to a desired output using labeled data. The two main varieties of supervised learning are classification (used to determine what category, or class, a data point belongs to) and numerical regression (used to learn a function that describes the relationship between inputs and output). On the other hand, unsupervised learning makes sense of data by finding patterns in unlabeled data sets.

However, at the heart of learning and intelligence, there is the idea that our interactions with our environment give us knowledge about the environment itself and about ourselves. For example, an infant learns to stand up and walk by trying to move body parts and receiving negative sensorimotor feedbacks when falling down and positive feedbacks from adults around when a step forward is taken. Similarly, a chess player learns winning tactics and strategies by playing multiple games and observing the consequences of making specific moves in different scenarios. The same reasoning applies for an artificial learner, such as an adaptive routing sys-

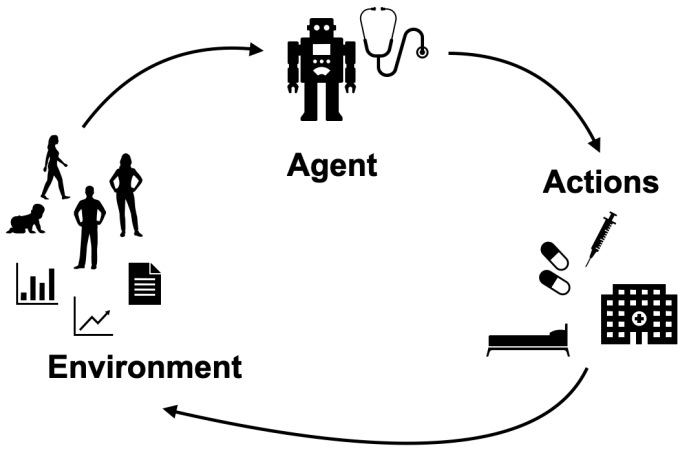


Figure 1.1: Example of RL for dynamic treatment. The agent must determine the course of actions (treatment type) based on the current health status and prior treatment history of patients to design the best long-term care plan.

tems which has to decide where to send packets in a large network to minimise the number of lost packets without knowing a priori where the weak or congested links are. Reinforcement Learning (RL) is the field of machine learning that constructs a mathematical framework for solving sequential decision-making problems under uncertainty. An agent, or decision maker, must learn how to act in an environment in order to achieve a long term goal (see example in Figure 1.1). This thesis makes contributions to the field of RL from a data-efficiency perspective.

Although in the past few years many hardware improvements have been made to lessen the compute load and to make data processing more time-efficient (with the emergence of efficient parallel processing units such as GPUs and TPUs), the fact remains that the main efficiency bottleneck of reinforcement learning is the sample size and the inability to generalise beyond unseen data: RL agents require a large number of samples in order to learn a task, even when slightly different tasks have already been previously solved successfully. This type of learning contradicts human learning, where learning happens extremely efficiently and necessitates minimal experience. This suggests that **current RL agents lack the ability to abstract information, to reason, and to generalise**. Not only does this result in unsustainable training, but it also greatly restricts the possible applications, as real world data, contrary to synthetic data, is often limited.

1.1 Main Challenges

This thesis pioneers a timely research area revolving around the question:

> *How can we improve the design of reinforcement learning algorithms in such a way that the data requirements remain reasonable while retaining high performances?*

When acting in an environment, the agent takes sequential actions and observes consequent rewards and future states. These sequences of observations improve the agent's knowledge of the environment, which must be leveraged to find the strategy that will lead to the highest long-term reward (*i.e.* the optimal policy). In most real world applications, the environment often has a large (possibly continuous) state space. In such cases, the current methods for finding the optimal policy are slow and require an extraordinary amount of data [135]. This is partly due to the following limitations:

Agents do not *understand* their environment. Artificial agents, contrarily to humans, do not have a clever way of interpreting features in the observations. An ideal RL algorithm would be able to efficiently extract the relevant features of the environment (*i.e.* the appropriate *representation*) in order to speed up the learning.

Agents lack *reasoning* capability. In complex decision-making problems, the effect of certain key decisions is not necessarily observed immediately, but can be considerably delayed. Ideally, despite the time delay between actions and outcomes, an agent would be able to reason about the links between the decisions that lead to specific outcomes, and *assign credit* accordingly to the relevant decisions.

Agents do not *adapt* past knowledge to new situations. Humans are exceptionally good at retaining information for later use, even when future tasks are yet unknown. This ability to effortlessly reuse skills has not been demonstrated in artificial agents as of yet. The data-efficiency of RL could be improved if agents were capable of *generalising* knowledge learned previously, instead of tackling new tasks from scratch.

This thesis makes important contributions towards understanding and improving data-efficiency by studying mainly the first two limitations listed above.

1.2 Contributions

The research efforts in this thesis are towards building more data-efficient and generalisable RL via representation learning eligibility traces. The main contributions of this thesis can be summarised as follows:

1. **Understanding the importance of representations learning in RL.** The field of representation learning in RL is growing rapidly, and the lack of clarity regarding advantageous representational properties hinders measurable progress towards more efficient representation in RL. This shortcoming is addressed in Chapter 3, where a review of the current state of the field of representation learning is provided, and a new categorisation of representational properties is introduced. Important links between existing representations are identified and new perspectives on desirable representational properties are proposed.

2. **Leveraging graph representation in RL.** An important limitation of state-of-the-art graph-based techniques in RL is identified in Chapter 4, and alternative graph-based approaches are successfully applied to RL problems for the first time. This work is published in [89].

3. **Analysis of a graph representation learning method for RL.** In Chapter 4, a graph representation learning method—which was not initially designed for RL applications—is proven to be powerful in efficiently solving RL problems. Chapter 5 presents key experimental and theoretical analysis of this method when applied to RL.

4. **Graph-inspired representation learning method for data-efficient RL.** Building on the finding of adopting graph-based representation for improving RL algorithms (Chapter 4), a novel graph-inspired algorithm, called state2vec, is developed and empirically proven to improve on previous findings. This work is published in [91].

5. **Data-efficient RL via counterfactual credit assignment.** Chapter 6 intro-

duces a novel concept, called expected eligibility traces, which enables fast and reliable credit assignment in RL. Expected eligibility traces are the key element in a newly proposed algorithm for counterfactual credit assignment. This work is published in [159].

1.3 Publications

Central parts of the research presented in this thesis resulted in conference and workshop publications. In particular, Chapter 4, parts of Chapter 5, and Chapter 6 are based on the following conference papers, including a Distinguished Paper Award:

- S. Madjiheurem and L. Toni. Representation learning on graphs: A reinforcement learning application. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 3391–3399. PMLR, 16–18 Apr 2019. URL `https://proceedings.mlr.press/v89/madjiheurem19a.html`

- H. van Hasselt, S. Madjiheurem, M. Hessel, D. Silver, A. Barreto, and D. Borsa. Expected eligibility traces. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, Virtual Event, February 2-9, 2021*, pages 9997–10005. AAAI Press, 2021. URL `https://ojs.aaai.org/index.php/AAAI/article/view/17200/` ***Distinguished Paper Award.***

- S. Madjiheurem and L. Toni. Disentangled predictive representation for meta-reinforcement learning. In *ICML 2021 Workshop on Unsupervised Reinforcement Learning*, 2021. URL `https://openreview.net/forum?id=VbLGbcdz16-`

The contributions in Chapters 1 and 3, and the first part of Chapter 5 are original to this thesis. The following preprints are also a direct result of the work undertaken in in this thesis.

- S. Madjiheurem and L. Toni. State2vec: Off-policy successor features approximators. *CoRR*, abs/1910.10277, 2019. URL `http://arxiv.org/abs/1910.10277`

- S. Madjiheurem, M. Bellemare, and L. Toni. On the importance of representation learning in reinforcement learning. Pending publication, 2022

1.4 Document Organisation

This thesis is organised in seven chapters. Following this introductory chapter, the rest of this document is as follows

- **Chapter 2** introduces the notation adopted throughout the document and presents the related background.

- **Chapter 3** contains discussions around the importance of representation learning in RL. A categorisation of representations in RL is proposed, and the existing representation learning methods in RL are reviewed and classified according to the newly introduced formalism. An extensive and critical discussion on the current state of the art of representation learning in RL is carried out.

- **Chapter 4** presents a new line of research that leverages graph data structures to learn representations capturing the structural geometry of underlying reinforcement learning problems. A novel algorithm, General Representation Policy Iteration, is proposed and experiments are conducted.

- **Chapter 5** contains an analysis of node2vec, a representation learning algorithm on graphs. The usefulness of node2vec in RL is studied through experiments and visualisations, and theoretical links are drawn between node2vec and state-of-the-art graph-based representation in RL. Additionally, a novel graph-inspired representation learning method for RL, called state2vec, is developed and supported by experiments and analysis.

- **Chapter 6** introduces a new approach to credit assignment, based on the concept of expected eligibility traces. A family of algorithms that use expected traces to update their predictions is presented, theoretically analysed and evaluated on various domains.

- **Chapter 7** concludes the thesis with a compact summary of these contributions and discusses important potential directions for future research.

Chapter 2

Reinforcement Learning

This thesis situates itself in the field of machine learning called Reinforcement Learning (RL). In essence, RL is the computational approach to learning from interaction. Similarly to some mechanisms observed in the human brain [104, 107], an RL *agent* discovers which actions yield the highest numerical *reward* by interacting with the *environment*. Consequently, the goal of RL is to learn how to act, i.e which sequential actions (which *policy*) an agent should take in specific situations in order to achieve a long term goal. RL problems are typically modelled as Markov Decision Processes (MDPs), which we describe in details in the following section.

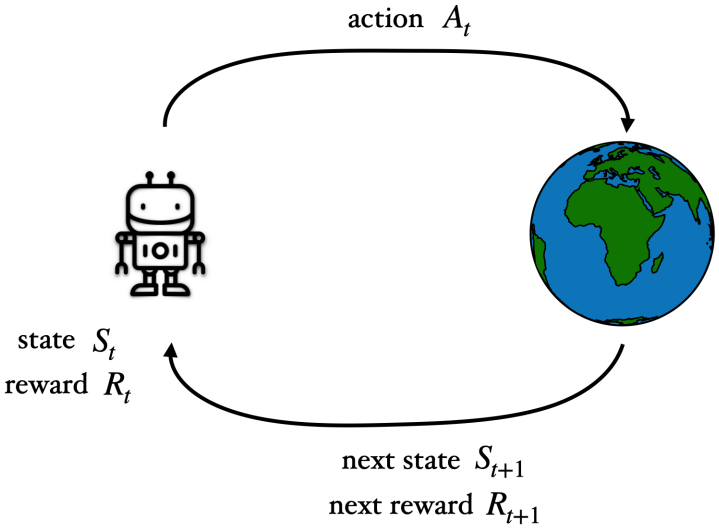


Figure 2.1: Illustration of agent-environment interaction in an MDP. At any given time t, an agent (here a robot) in a state S_t interacts with its environment (the world) by taking actions A_t and observes subsequent state S_{t+1} and reward R_t.

In addition to the agent, the environment, the reward signal and the policy, another essential element of RL is the *value function*. The problem of finding an optimal policy is quantified with value functions, which describe the long-term desirability of states, taking into account the predicted future states and the rewards available in those states. Given a good value function, the MDP is solved: the agent chooses the sequence of actions that maximises the value of each sequent state.

2.1 Notation

For consistency with prior literature, the following convention is used throughout this thesis: capital letters indicate scalar-valued random variables (*e.g.* S_t, A_t, R_t), non-bold lowercase letters are used for the value of random variables (*e.g.* s, a), bold lowercase letters indicate vectors (*e.g.* $\boldsymbol{\theta}$, $\boldsymbol{\phi}$), bold capital letters indicate matrices (*e.g.* $\boldsymbol{I}$, $\boldsymbol{M}$), scalar functions are indicated by non-bold lowercase and uppercase letters (*e.g.* f, V, Q,), a calligraphic font is used for finite sets (*e.g.* $\mathcal{S}$, $\mathcal{A}$, $\mathcal{E}$), and the blackboard bold typeset letters $\mathbb{R}$, $\mathbb{E}$, $\mathbb{V}$, and $\mathbb{P}$ indicate the set of real numbers, the expected value of random variables, the variance of random variables and a probability density function respectively.

2.2 Markov Decision Processes

Markov Decision Processes (MDPs) [9] are discrete time stochastic control processes that provide a widely-used mathematical framework for modelling sequential decision making strategies under uncertainty, where actions not only influence immediate rewards but subsequent situations as well. The decision maker is called the *agent*. At each decision opportunity (or *time step* t), the environment characteristics identify the state $s_t \in \mathcal{S}$. The agent can choose any feasible action a_t from a set of actions $\mathcal{A}$. As a consequence of the action taken, the agent finds itself in a new state s_{t+1} according to $p(s_t, a, s_{t+1})$ and observes an instantaneous reward $R_t \in \mathbb{R}$, where $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0,1]$ defines state-transition probabilities. The expected immediate reward function is given by $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, defined as $r(s,a) \doteq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a]$.

A discrete MDP can be fully described by the tuple $M = (\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where

$\mathcal{S}$ is a finite set of discrete states, $\mathcal{A}$ a finite set of actions, p describes the transition model—with $p(s,a,s')$ giving the conditional probability $\mathbb{P}(s'|s,a)$ of moving from state s to s' given action a, r is the expected reward function and $\gamma \in (0,1]$ is a discount factor. In the context of this thesis, we consider *finite* MDPs, in which the sets of states and actions have a finite number of elements. The random variable R and S have well defined discrete probability distributions that only depend on the previous state and action, hence having the Markovian property.

2.3 Policies and Value Functions

A *value function* defines how promising a state is towards achieving maximum reward. Formally, the value function of state s, denoted by $V(s)$, defines the expected cumulative reward experienced over time starting from state s. Because the expected reward depends on the choice of actions, a value function is defined with respect to a *policy*. A policy $\pi \in \Pi$ is a mapping from states to probabilities of selecting actions in $\mathcal{A}$, and Π denotes the set of all possible policies. Formally, for a stochastic policy, $\pi(a|s)$ defines the probability that the agent takes action a when the agent is in state s.

$$\pi(a|s) = \mathbb{P}(A = a, S = s)\,. \tag{2.1}$$

We write deterministic policies by $\pi(s)$, which is equal to the action that the agent takes from state s. The goal of reinforcement learning is to find a policy $\pi^* \in \Pi$ which maximises the expected discounted sum of future reward, or expected *return* G, defined as

$$G = R_0 + \gamma R_1 + \gamma^2 R_2 + \ldots = \sum_{t=0}^{\infty} \gamma^t R_t\,. \tag{2.2}$$

Given a policy $\pi \in \Pi$, a value function is a mapping $V^\pi : \mathcal{S} \mapsto \mathbb{R}$ that describes the expected long-term discounted cumulative reward observed by the agent in any given state $s \in \mathcal{S}$ when following policy π. It is formally defined as follows:

$$V^\pi(s) \doteq \mathbb{E}_\pi\Big[G \,\Big|\, S_0 = s\Big] = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^t R(s_t) \,\Big|\, S_0 = s\Big]\,. \tag{2.3}$$

The value function can also be written in recursive form,

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) \left[r(s,a) + \gamma \sum_{s' \in S} p(s,a,s') V^{\pi}(s') \right], \tag{2.4}$$

where $r(s) = \sum_{a \in A} \pi(a|s) R(s,a)$ defines the expected immediate reward at state s. The formulation in (2.4) represents the *Bellman equation* [10]. An alternative to the state value function V^{π}, is the Q-value function which considers state action pairs. Under a policy π, the Q-value function is a mapping $Q^{\pi} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ defining the value of taking action a in state s under policy π as follows

$$Q^{\pi}(s,a) \doteq \mathbb{E}_{\pi}\Big[G \Big| S_0 = s, A_0 = a \Big] = \mathbb{E}_{\pi}\Big[\sum_{k=0}^{\infty} \gamma^t R(s_t) \Big| S_0 = s, A_0 = a \Big] \tag{2.5}$$

The relationship between value functions and Q-value functions can be described as follows:

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) Q^{\pi}(a,s). \tag{2.6}$$

Solving an MDP requires to find a policy π^* that leads to the optimal value function V^* and the optimal Q-value function Q^*, defined as

$$V^*(s) = \max_{\pi} V^{\pi}(s), \quad \forall s \in \mathcal{S} \tag{2.7}$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \quad \forall s,a \in \mathcal{S} \times \mathcal{A} \tag{2.8}$$

The optimal value function and Q-value function satisfy the following respective constraints

$$V^*(s) = \max_{a} \Big(r(s,a) + \gamma \sum_{s' \in S} p(s,a,s') V^*(s') \Big) \tag{2.9}$$

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s,a,s') \max_{a'} Q^*(s',a') \tag{2.10}$$

These recursive equations are known as the *Bellman's optimality equations* [10] for V^* and Q^* respectively. The optimal policy π^* which can be written as

$$\pi^*(s) = \operatorname*{argmax}_{a} Q^*(s,a)\,. \tag{2.11}$$

is deterministic and is the unique solution to the Bellman's optimality equations. The optimal Q-value function Q^* gives the expected return for taking action a in state s an subsequently following an optimal policy. Therefore, V^* can be written in terms of Q^* as follows

$$V^*(s) = \max_a Q^*(s,a)\,. \tag{2.12}$$

Here, we have introduced the main RL concepts with the notation used throughout this thesis. Additional concepts and notations specific to each chapters will be introduced when necessary. In the following, we present the main techniques for solving the RL problems of *prediction* (computation of the value function under a fixed policy) and *control* (search for an optimal policy). We first introduce fundamental model-based and model-free algorithms in *tabular* settings (where the state action space is small enough such that the agent can maintain a set of values in a lookup table with random access). We then explain how to go beyond tabular RL by using function approximation.

2.4 Model-Based RL

In *model-based* RL, it is assumed that the agent has access to the transition dynamics p, the reward function r, and the state and action spaces $\mathcal{S}$ and $\mathcal{A}$ which define the model of the environment. This model can be either given or learned from experience. Model-based RL then consists in interacting with the model (this interaction is called *planning*) to recommend an action.

Policy Evaluation

In the tabular case, given a model of the environment, the Bellman equation (2.9) can be computed by dynamic programming DP, iteratively evaluating the value functions for all states. This approach is called *iterative policy evaluation* [146]. Initially, V_0 is chosen arbitrarily, and iterative approximations are obtained using the following update rule

$$V_{k+1}(s) \doteq \sum_{a\in A} \pi(a|s)\left[r(s,a) + \gamma \sum_{s'\in S} p(s,a,s')V_k(s')\right], \tag{2.13}$$

for all $s \in \mathcal{S}$. The sequence of approximations $\{V_0, V_1, \ldots\}$ can be proven to converge to V^π.

Policy Iteration

Policy evaluation is used in *policy iteration*, where the goal is to find an optimal policy π^*. In policy iteration, we iterate between evaluating the current policy π according to (2.13) (policy evaluation) and computing a new policy π' that improves on the current policy (policy improvement), according to

$$\pi'(s) \doteq \operatorname*{argmax}_a \sum_{s' \in S} \left[r(s,a) + \gamma p(s,a,s') V^\pi(s') \right], \tag{2.14}$$

for all $s \in \mathcal{S}$ until the policy is stable, *e.g.* no further improvement is attainable and the optimal policy is found.

2.5 Model-Free RL

When the agent does not have complete knowledge of the environment, it has to rely solely on past experience. RL methods relying only on experience to predict value functions or learn the optimal policy are referred to as *model-free* methods.

Monte Carlo Methods

Monte Carlo (MC) methods learn value estimates by averaging returns of sampled episodes. When the environment is episodic, *i.e.* when all episodes eventually terminate regardless of the sequence of actions, *Monte Carlo evaluation* solve the RL prediction problem by updating current value estimates towards sample episodes returns, according to

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)], \tag{2.15}$$

for all S_t in sampled trajectories $\tau = \{S_0, A_0, R_1, S_1, \ldots, S_t, A_t, R_{t+1}, S_{t+1}, \ldots, S_T\}$, where $\alpha > 0$ is a step size and G_t denotes the return at time step t

$$G_t \doteq G(\tau_{t:T}) = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-1} R_T \tag{2.16}$$

$$= \sum_{i=1}^{T} \gamma^{(i-1)} R_{t+i}\,, \tag{2.17}$$

where $\tau_{t:T}$ denotes the truncated trajectory τ, starting at time step t and ending at the end of the episode at the terminating time step T.

Monte Carlo policy iteration finds an optimal policy by alternating between Monte Carlo evaluation and policy improvement (2.14) on an episode-by-episode basis.

Temporal-Difference Learning

Temporal-Difference (TD) learning provides a way of learning the value function from experience, in the absence of a model and without requiring to sample full trajectories. TD learning achieves such *online learning* via bootstrapping, by updating the current value estimates towards a predicted return based on the current value estimates. The TD prediction update is as follows

$$V(S_t) \leftarrow V(S_t) + \alpha \big[\overbrace{R_{t+1} + \gamma V(s_{t+1})}^{\text{TD target}} - V(S_t) \big]_{\text{TD-error}}\,. \tag{2.18}$$

The update is made immediately upon transitioning to S_{t+1}. Comparing with the update in (2.15), we notice some similarity: the target for Monte Carlo evaluation is the return G_t, whereas TD evaluation uses the *TD target* $R_{t+1} + \gamma V(S_{t+1})$. The difference between the TD target and the current estimate at time t is called the *TD-error* and is often denoted by δ_t. This TD learning algorithm is called *TD(0)* or *one-step TD*, because only considers the one-step temporal difference. With a step size $\alpha > 0$ sufficiently small, one-step TD converges deterministically to a single answer under batch updates [143].

A variant of TD learning, *TD(λ)* [143, 146] enables online multi-step updates

by leveraging a memory variable associated with each state, known as *eligibility trace*. Given the current state s_t, the eligibility trace is defined as

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t\,, \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases} \tag{2.19}$$

with $e_0(s) = 0$ for all $s \in \mathcal{S}$, and $\lambda \in [0,1]$ is the trace-decay, controlling "how far back" in time the eligibility traces should accumulate state visitations. The update to the value estimate is then done at each time step by propagating the TD-error $\delta_t = R_{t+1} + \gamma V(S_{t+1})$ to all recently visited states, as informed by their nonzero eligibility traces:

$$V(S_t) \leftarrow V(S_t) + \alpha\delta_t e_t(S_t)\,. \tag{2.20}$$

Each time an episode terminates, the eligibility traces are reset to zero. This algorithm is known as the *backward view* of $TD(\lambda)$, because it propagates reward information backward in time.

An important TD method for control is *Q-learning* [170]. It is a way of directly approximating the optimal Q-value function. The update is quite simple

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\left[R_{t+1} + \gamma\max_a Q(S_{t+1},a) - Q(S_t,A_t)\right] \tag{2.21}$$

If all pairs continue to be updated, the estimates are guaranteed to converge.

2.6 Approximation Methods

The methods described in the previous sections suffer from the curse of dimensionality as the learning speed scales polynomially with the state and action space dimension. In addition to time complexity, the curse of dimensionality also affects the memory requirements. Indeed DP, tabular MC, and tabular TD methods update value estimates using a lookup table, with one entry for each state (or state-action pair). The size of the lookup table is therefore proportional to the cardinality of the state space (or state-action space), leading to unreasonably large lookup tables in high-dimensional problems. MC methods have the additional constraint of re-

quiring to store complete episodes, which slows the learning of problems with long episodes.

The aforementioned limitations have been addressed with function approximation [11]. Namely, instead of learning a direct mapping $s \mapsto V(s)$, we can learn a function which approximates this mapping: $V(s) \approx \tilde{V}_\theta(s)$, where the elements in vector $\boldsymbol{\theta}$ are the approximator's parameters. In the following, we review a selection of approximation methods.

2.6.1 Linear Value Function Approximation

A popular approach is to estimate the value function as a weighted sum of a set of features $\boldsymbol{\phi} = [\phi_1, \phi_2, \ldots, \phi_d]$ (called *basis functions*) [104, 93, 71, 73], as follows

$$\tilde{V}_\theta(s) \;=\; \sum_{i=1}^{d} \theta_i \phi_i(s) \;\approx\; V(s),$$

where d is the dimension of the feature space and $\boldsymbol{\theta}$ is an unknown weight vector. The basis functions $\boldsymbol{\phi}$ can be hand-crafted [145] or automatically constructed [95], and the model parameters $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_d]$ are typically learned via standard parameter estimation methods such as least-square temporal difference (LSTD) [18] or least-square policy iteration (LSPI) [73], as described in the following subsections. The accuracy of the approximation highly depends on the choice of the basis functions. How to properly select the set of basis functions for a data-efficient value function approximation process is still an open question. The main objective is to find the set of basis $\boldsymbol{\phi}$ that is *low-dimensional* (to ensure a data-efficient learning) and yet a *meaningful representation* of the MPD (to limit deviation from the true value function). Part of the efforts in this thesis is focused on answering this question. In Chapter 3, we review techniques for constructing or learning compact representation of MDPs, in Chapter 4 we investigate the use of graph-base representation in RL, and Chapter 5 introduces a novel graph-inspired mechanism for learning low-dimensional state features.

TD learning with linear function approximation

Under linear value approximation, given the basis functions $\boldsymbol{\phi}$, the parameters $\boldsymbol{\theta}$ which project the true value function onto the space spanned by the basis function need to be estimated from data. This can be achieved using sample-based methods seen in Section 2.4 generalised to function approximation. For instance, beyond lookup tables, TD learning can update arbitrary prediction functions, such as a linear function $\tilde{V}^{\pi}_{\theta}(s) = \boldsymbol{\theta}^{\top}\boldsymbol{\phi}(s)$. The $TD(0)$ update rule (2.18) with linear function approximation is given by

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[R_{t+1} + \gamma\boldsymbol{\theta}^{\top}\boldsymbol{\phi}(S_{t+1}) - \boldsymbol{\theta}^{\top}\boldsymbol{\phi}(S_t)\big]\boldsymbol{\phi}(S_t)\,. \tag{2.22}$$

It can be shown that if the system converges, it converges to the fixed point $\boldsymbol{\theta}_{TD} = \boldsymbol{M}^{-1}\boldsymbol{b}$, where

$$\begin{aligned} \boldsymbol{b} &\doteq \mathbb{E}[R\boldsymbol{\phi}(s)] \in \mathbb{R}^d \text{ and} \\ \boldsymbol{M} &\doteq \mathbb{E}\big[\boldsymbol{\phi}(s)\big(\boldsymbol{\phi}(s) - \gamma\boldsymbol{\phi}(s')\big)^{\top}\big] \in \mathbb{R}^d \times \mathbb{R}^d. \end{aligned} \tag{2.23}$$

Least-Square Temporal Difference

Least-Square Temporal Difference (LSTD) learning [18] learns an approximation $\tilde{V}^{\pi}$ given a fix set of basis $\boldsymbol{\phi}$ by computing the natural estimates of the matrix $\boldsymbol{M}$ and the vector $\boldsymbol{b}$ of the fix-point solution in (2.23):

$$\begin{aligned} \hat{b}_k &= \sum_{k=0}^{i-1} R_{k+1}\boldsymbol{\phi}(S_k) \text{ and} \\ \hat{\boldsymbol{M}}_k &= \sum_{k=0}^{i-1} \boldsymbol{\phi}(s_k)\big(\boldsymbol{\phi}(S_k) - \gamma\boldsymbol{\phi}(s_{k+1})\big)^{\top} + \varepsilon I, \end{aligned} \tag{2.24}$$

where I denotes the identity matrix, and εI ensures that $\hat{\boldsymbol{M}}_k$ is invertible. Compared to TD(0) which "wastes" data by computing the solution iteratively, LSTD is more data efficient [15] but also more expensive computationally, as the matrix $\hat{\boldsymbol{M}}_k$ needs to be inverted.

Least-Square Policy Iteration

For control problems with function approximation, Least-Square Policy Iteration (LSPI) [73] learns an optimal policy by alternating between LSTDQ (the Q-function version of LSTD), and policy improvement according to (2.14). LSTDQ learns a linear action-value approximation of the form

$$\tilde{Q}^{\pi}_{\theta}(s,a) \;=\; \sum_{i=1}^{d} \theta_i \phi_i(s,a) \;\approx\; Q^{\pi}(s,a), \tag{2.25}$$

where the $\boldsymbol{\phi}(s,a) \in \mathbb{R}^d$ are state action basis functions.

While LSTD approximates the value function of a fixed policy (policy evaluation), LSPI iteratively refines the policy such that at each iteration, the policy is a better approximation.

2.6.2 Kernel-based Function Approximation

Another type of function approximation technique used in reinforcement learning is based on the use of Kernels [110, 63]. The main idea of kernel methods is that inner products in a high-dimensional feature space can be represented by a (nonlinear) kernel function. That way, usual learning algorithms in linear domains become nonlinear algorithms without explicitly computing the inner products in the high-dimensional feature spaces. These methods are popular in various kernel-based supervised and unsupervised learning problems [129, 130].

Kernel-based reinforcement learning uses a set $\mathcal{D}$ of sample outcomes $\{s_t, a_t, s_{t+1}, r_t\}$. The value function is approximated as a kernel weighted average of the targets of all samples stored in $\mathcal{D}$:

$$\tilde{V}(s) = \sum_{s' \in \mathcal{D}} k(s,s') g(s'). \tag{2.26}$$

where $k : S \times S \mapsto \mathbb{R}$ is a kernel function and $g(s')$ denotes the sampled outcome for state s'. Kernel functions can be thought as numerical metrics that express how relevant the knowledge about a state is to another state.

Kernels allow to place problems in a high-dimensional space without having

to explicitly calculate the corresponding features. For example, the kernel-based approach leads to the same approximation as the linear parametric method when the kernel function is $k(s,s') = \boldsymbol{\phi}(s)^\top \boldsymbol{\phi}(s')$ [14]. However, these methods have a computational complexity that is quadratic in the number of samples, making it impractical when the state space is large.

2.6.3 Deep Reinforcement Learning

Artificial Neural Networks (NNs) are powerful models for nonlinear function approximation. An artificial NN consists of layers of interconnected units that have activation functions, as to mimic some of the properties of biological neurons. They offer great flexibility in their architecture in terms of the number of units, number of layers and the type of activation functions. Given sufficient amount of data, NNs enable automatic feature extraction. In RL, a deep NN (DNN) can be used to approximate the Q-value function $\tilde{Q}^\pi_\theta(s,a)$. Semi-gradient Q-learning is then used to update the NN parameters $\boldsymbol{\theta}$ according to

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \underbrace{\big[R_{t+1} + \gamma \max_a Q_\theta(S_{t+1},a)}_{\text{TD target}} - Q_\theta(S_t,A_t))\big] \nabla_\theta Q_\theta(S_{t+1},A_{t+1}), \quad (2.27)$$

where the gradient $\nabla_\theta Q_\theta(S_{t+1},A_{t+1})$ is the vector of the first partial derivatives. The deep NN approximating the Q-value is called DQN [102]. Various tactics have been adopted in practice to make the training of the DQN stable. For example, the network is not trained completely online as transitions come in. Rather, sampled transitions are saved into a replay buffer, and at each training step, the agent samples a minibatch of transitions at random and executes the update in (2.27). In addition, an additional NN, the target network $Q_{\bar{\theta}}$, is used to infer the TD target in the update. The weights of the target network are updated periodically with the weights of the main network ($\bar{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}$).

Advances in the design and training of DNNs have contributed to the achievement of remarkable performances in reinforcement learning [102, 135]. The use of DNNs to solve reinforcement learning problems is commonly known as *deep*

reinforcement learning. DQN was one of the first successful application of deep learning to RL. Since then, some improvement to original DQN architecture have been made. For instance, Hasselt et al. [51] introduced DDQN, whereby the greedy policy is evaluated according to the online network, but the target network is used to estimate its value. This helped improve the over-estimation limitation that standard DQN faces. Another improvement to DQN is known as Prioritized Experience Replay [128], which identifies important experience transitions using the TD error, and replay them more frequently, to learn more efficiently.

DNNs have been implemented in RL algorithms beyond Q-learning, such as policy gradient [173, 147]. In this case, a DNN is used to directly model the policy $\pi_{theta}(s)$, parameterised with weights θ. The parametrised policy is then optimised with respect to the expected return (long-term cumulative reward) by gradient descent. Alternatively, DNNs can model both the values and policy. This approach is known as actor-critic, where the actor decides on the action (based on the policy network) and the critic informs on how good an action is (based on the value network). Deep RL actor-critic methods include A3C [103] and Q-Prop [46].

Although deep reinforcement learning has lead to impressive performances, the training of deep neural networks remains data-hungry and requires a great amount of compute power [80, 168].

2.7 Credit Assignment

The credit assignment problem has long been a major research topic in artificial intelligence [101]. It refers to fact that in order to learn the optimal behaviour in a given situation, we need to accurately associate events, like rewards or penalties, to relevant earlier decisions or situations. This is important both for learning accurate predictions, and for making good decisions. Appropriate credit assignment is a challenging problem, as rewards can occur terribly temporally delayed. For example, in the game of chess, the individual moves do not generate any immediate reward until the last move, when the final outcome of the game is revealed. In this case, appropriate credit assignment would identify key moves in the game.

TD learning methods assign credit temporally. One-step TD updates (2.18) propagate information slowly: when a surprising value is observed, the state immediately preceding it is updated, but no earlier states or decisions are updated. *Multi-step* updates propagate information faster over longer temporal spans, speeding up credit assignment and learning. As seen previously, multi-step updates can be implemented online using *eligibility traces* [143], without incurring considerable additional computational expense, even if the time spans are long; these algorithms have computation that is independent of the temporal span of the prediction [161].

With temporal credit assignment, many steps must be performed for credit to eventually trickle backwards, however this can take many iterations. In Chapter 6, we investigate an alternative, more data-efficient way of assigning credit.

Chapter 3

The Importance of Learning Representations in RL

A major open problem in reinforcement learning (RL) is the curse of dimensionality, whereby a large state and action space makes the learning of the value function and the search for an optimal policy hard problems. When the state and action space is very large—which is the case in most real-world problems—we need to resort to function approximation as all state-action values cannot be stored in memory. The state, or *observation*, summarises the current situation of the environment accessible to the agent. This information is often high-dimensional and may contain information that is not useful for solving the task, or it might contain redundant information that could be represented in a more compact form. For this reason, a new choice of state (and action) representation is necessary to improve the performance of the learning process. Good representations exist to re-shape the state and action space to handle larger problems, and increase efficiency.

For illustration, consider the Atari game Space Invaders, which is wildly used in RL research [6, 102, 51, 54] and is depicted in Figure 3.1. In Space Invaders, the agent, or player, moves a laser cannon horizontally across the bottom of the screen and fires at aliens overhead, while the aliens randomly shoot towards the cannon. The goal is to eliminate all of the aliens by shooting them. The player starts with three lives and the game ends immediately if the invaders reach the bottom of the screen, or the player loses all their lives.

Figure 3.1: A random frame of the Atari game Space Invaders.[1]

In this environment, the observation is an RGB image of the screen, which is a 2D array of 7-bit pixels, 160 pixels wide by 210 pixels high. Because pixel encoding is partly random, no semantic or symbolic information can be inferred directly from pixel values. To make sense of the image observation, the agent must thus learn to map pixels to meaningful information relating to the game. Instead, if the observation held symbolic information (for example if a state observation was a small vector with entries relating to number of aliens, their locations, whether or not they have fired, the location of the player and the number of lives left), the agent would be able to use this observation as it is to quickly learn optimal behaviours. A symbolic representation is ideal in this scenario as it contains only information that is relevant for the game. The need for a good representation is thus undeniable. However, the questions *"What exactly is good representation for RL?"* and *"How to learn such representations?"* remain unanswered.

Prior works have made different design choices when it comes to shaping a good representation. Some works suggest that a good representation is one that allows to linearly approximate value functions with high accuracy [95, 111, 76, 152, 8], while others have focused on representations that help throughout the pol-

[1] Image credit: OpenAI [109]

icy learning process [42, 24]. Additionally, recent efforts have introduced ways of learning representations that will adapt, or transfer, to similar tasks [3, 56, 4, 89]. Other representational properties have also been deemed desirable, such as capturing specific aspect of the environment [61, 56, 131, 33] or improving exploration [85, 87, 86, 88]. Some of these properties are in direct contradiction (*e.g.* a representation that is good at evaluating a policy with high accuracy will not naturally transfer to other policies), and there is no generally accepted consensus on which necessary properties a representation must hold in order to improve the overall efficiency of RL algorithms. The field of representation learning for RL is growing rapidly, and the lack of clarity regarding advantageous representational properties hinders measurable progress towards more efficient representations for RL.

With this review, we address this shortcoming by providing an understanding of the current state of the field of representation learning in RL. We categorise existing representation learning methods for RL, highlighting what purposes they serve and what properties they hold. We identify four commonly accepted categories of representations, and discuss when representations are consistent or in conflict with one another.

The key contributions of this chapter can be summarised as follows:

- We introduce a categorisation of representations in RL with respect to their representational properties. We clearly define the class of problems implicitly referred to as representation learning in RL in the literature.

- We review and classify existing representation learning methods according to the newly introduced formalism.

- We carry out an extensive and critical discussion on the current state of the art of representation learning in RL, identifying contradictions, limitations and pitfalls.

- We provide recommended directions for future work to address the identified limitations.

This chapter is organised as follows. The notation we use in this chapter as well as the relevant background are presented in Section 3.1. Section 3.2 describes the four main approaches to representation learning in RL. The principal representational properties captured by existing methods are identified and explained in Section 3.3 and we categorise prior works according to their main properties. In Section 3.4, we identify similarities and contradictions in the different representation learning methods and discuss the main limitations of existing systems. Finally, we critically discuss the question of what a good representation for RL is, and recommend future work in Section 3.5.

3.1 Notation and Background

Discrete RL environments described by Markov Decision Processes (MDPs) are defined by a tuple $M = (\mathcal{S}, \mathcal{A}, p, r)$, where $\mathcal{S}$ is a finite set of discrete states, $\mathcal{A}$ a finite set of actions, p denotes the transition model—with $p(s,a,s')$ giving the probability of moving from state $s \in \mathcal{S}$ to state $s' \in \mathcal{S}$ given action $a \in \mathcal{A}$—and r denotes the reward function. We denote by $\boldsymbol{P}^{\pi} \in [0,1]^{|\mathcal{S}|\times|\mathcal{S}|}$ the transition matrix under policy π. The element in the i-th row and j-th column of $\boldsymbol{P}^{\pi}$ is the probability of transitioning from state s_i to s_j when behaving according to policy π.

In the context of this chapter, we consider representations for model-free reinforcement learning. The notion of representation in RL is formally defined as follows:

Definition 1. *A d-dimensional state representation is a mapping $\phi : \mathcal{S} \mapsto \mathbb{R}^d$; $\phi(s)$ is the feature vector for state s. $\Phi \in \mathbb{R}^{|\mathcal{S}|\times d}$ denotes the matrix whose rows are $\phi(\mathcal{S})$.*

Definition 2. *A d-dimensional state-action representation is a mapping $\phi : \mathcal{S}\times\mathcal{A} \mapsto \mathbb{R}^d$; $\phi(s,a)$ is the feature vector for state action pair (s,a). $\Phi \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|\times d}$ denotes the matrix whose rows are $\phi(\mathcal{S},\mathcal{A})$.*

Representations have been constructed or learned and used in various ways in reinforcement learning as shown in Figure 3.2. For example, a representation

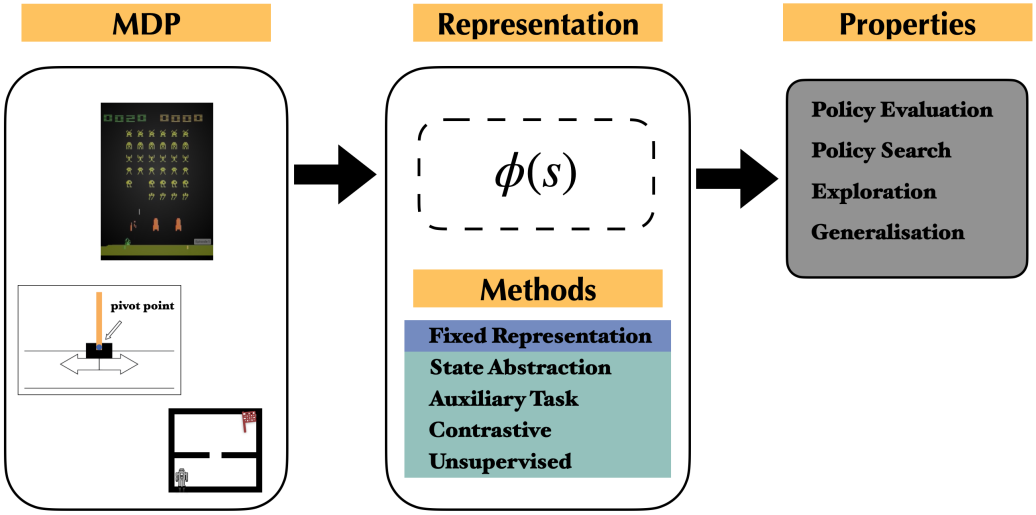


Figure 3.2: Illustrative example of the use of representations in RL and their different methodologies and properties. Given an MDP, a representation can be computed or learned using different methods to capture different properties.

can be used in **policy evaluation** [8, 95], to approximate the value function with a parametric function

$$\hat{V}^{\pi}_{\theta}(\phi(s)) \approx V^{\pi}(s)\,, \tag{3.1}$$

where θ is the vector of parameters to optimise. Often, a linear approximation is adopted

$$\hat{V}^{\pi}_{\theta}(s) = \phi(s)^{\top}\theta \approx V^{\pi}(s)\,. \tag{3.2}$$

When the dimensionality of the representation d is small compared to the state space size ($d << |\mathcal{S}|$), the size of the problem is reduced (fewer parameters to learn). As a result, efficiency is improved (sometimes at the expense of accuracy, depending on the quality of the representation).

Representations can also be used in **policy improvement** [43, 24], for example by modeling consecutively improving policies based on state action representations

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, \hat{Q}^{\pi}_{\theta}(\phi(s,a)) \tag{3.3}$$

where $\hat{Q}^{\pi}_{\theta}(\phi(s,a))$ is an approximation function for the current state action value function parameterised by θ.

Another way representations have been used in RL is to promote better **exploration** [85, 87], for example by designing an intrinsic reward function $r_i : \mathcal{S} \times \mathcal{S} \mapsto \mathbb{R}$ based on the variation in the representation space. For example

$$r_i(s, s') = |\phi(s) - \phi(s')| . \tag{3.4}$$

By learning to maximise such intrinsic reward functions, the agent is encouraged to explore different regions of the state space, ultimately increasing knowledge about the environment (depending on the quality of the representation).

Representations have also been used to facilitate **generalisation** [56, 3, 4] in RL. For example, if a representation captures information about the environment dynamics but is independent of the reward function, then the same representation can generalise across different tasks under common dynamics.

Fixed representations, or handcrafted representations, have been successfully deployed in the aforementioned different ways on problems with relatively small state-action space. Such methods include typical linear approximation architectures such as polynomial basis function and radial basis function [73], tile-coding [145], Fourier basis [71] and diffusion wavelets [94]. Existing fixed representation methods and their application to RL are discussed in detail in Section 3.3. The advantage of fixed representations is that their properties are interpretable and their performance can often be theoretically analysed. On the other hand, fixed representations make strong assumptions about the structure of the MDP (such as assuming that the state space is Euclidean, or that the value function is smooth on the state space [73, 95]) which might not hold in practice, resulting to poor learning performance. As a result, the need for more flexible representations has become further obvious, and representation learning for RL has emerged as a new field.

3.2 Representation Learning

In recent years, the field has evolved towards discovering representations that can capture information about the MDP. The problem of learning a useful representation ϕ is known as *representation learning* and it can be achieved via multiple different

methodologies. In the following, we describe those diverse representation learning methods, highlighting the key differences. For example, we show that the representation learning phase can be disentangled from the the reinforcement learning phase, or it can be learned jointly while solving the problem. We also demonstrate that the learning process can be either supervised or unsupervised. In total, we identify five main methods for generating representations in RL (fixed representation, state abstraction, auxiliary tasks, contrastive learning and unsupervised learning). Then in Section 3.3, we show how those representation methods lead to different RL agent behaviours or properties. Specifically, we identify four representational properties (a support for policy evaluation, policy search, exploration and generalisation) and we review the representation learning methods that have been used to learn representations with these different properties. In Section 3.4, we classify the reviewed works according to both the method they adopt, and the property they exhibit.

3.2.1 State Abstraction

An intuitive strategy to address the curse of dimensionality consists in reducing the size of the state space by grouping states into clusters, and assigning a similar representation for states within the same cluster. This method is known as clustering, and is a classical approach in data processing for downstream machine learning tasks. In RL, state abstractions have been proposed as a form of representation where the raw state space is mapped to a smaller finite abstract state space. By reducing the state space in this way, policy learning can be speed up substantially [79]. Figure 3.3(a) illustrates a case of state abstraction in a version of the four-room environment [141], where states within the same room are assigned an equivalent representation. Once the abstraction is computed (or learned), standard reinforcement learning techniques can be used to learn a value function and/or a policy (Figure 3.3(b)). The abstraction can also be improved online while solving the reinforcement learning problem [176, 83]. Different types of state abstractions have been proposed in RL. For example, *bisimulation* [76] is the strictest form of abstraction, grouping states that are indistinguishable with respect to reward sequences output given any action sequence. *Temporal abstraction* groups states such

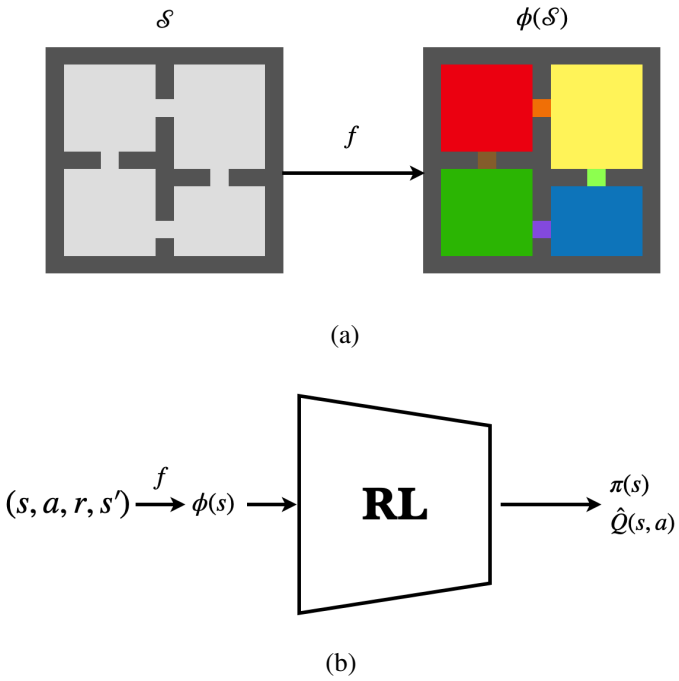


Figure 3.3: **(a)** Example of state abstraction on the four-room environment. Here, states are abstracted with respect to their room assignment. **(b)** Simplified RL system, in which the RL agent repetitively uses transition observations to improve on its current estimate of the value function and/or its policy. In this example, the state abstraction mechanism is done outside of the reinforcement learning, and the agent uses the abstraction function to preprocess state observations.

that their time scale is preserved [120]. We review the existing state abstraction methods in detail in Section 3.3. State abstractions can be powerful when the abstraction is well-aligned with the tasks at hand, and when constructing, or learning, the abstraction is feasible with a reasonable amount of data. On the other hand, if the abstraction is not aligned with the task, it can in fact harm the learning process. For example, considering the abstraction in Figure 3.3(a), if the goal is to collect objects in all corners of all the rooms, the room assignment abstraction cannot help in distinguishing between states that are corner states from those that are not, and the corresponding value function could not be accurately approximated. Learning relevant states abstractions thus requires some a priori knowledge about the task, which is not always available in practice.

3.2.2 Auxiliary Tasks

Instead of imposing a type of abstraction, some recent works have hypothesised that a good representation will emerge if we avoid optimising specific properties. These works let the system dictates the shape of the representation by training a model to optimise some additional loss along the solving of the reinforcement learning problem. This methodology is known as *auxiliary tasks*.

In classical deep RL, the representation $\mathbf{\Phi}$ is learned jointly while solving the task; the last layer of the deep neural network is the state action feature vector ϕ. As described in Chapter 2.6.3, many reinforcement algorithms have been implemented with deep neural networks. For example, one of the first successful deep RL agent, uses a Deep Q-Network (DQN) to learn to approximate its current value function while updating its policy greedily with respect to the current value function estimate [102]. Concretely, at each time step, a gradient step is performed with respect to the network's parameters θ towards minimising the following loss function

$$\ell_\theta = \left(Q_\theta(S_t, A_t) - (R_{t+1} + \gamma \max_a Q_\theta(S_{t+1}, a))\right)^2. \tag{3.5}$$

We can see the DQN as having two parts: a *core* and a linear *head* (the last hidden dense layer) each with its own parameters θ_c and θ_h respectively. We can then write the approximated value function as:

$$Q_\theta(s_t, a_t) = \phi_{\theta_c}(s_t, a_t)^\top \theta_h,$$

where $\phi_{\theta_c}(s,a)$ is the state action feature vector modeled by the core of the DQN. Figure 3.4 depicts a simple DQN.

While basic DQNs have achieved strong performances, they require an excessive amount of data and have shown to be extremely sensitive to minor perturbation in the observations (for example, an optimal policy for Atari games does not transfer to the same games when pixel colours are inverted [61]). Recent empirical findings in deep RL revealed that in order to capture rich information in a representation, the agent must learn about many aspects of the world in addition to learning a good

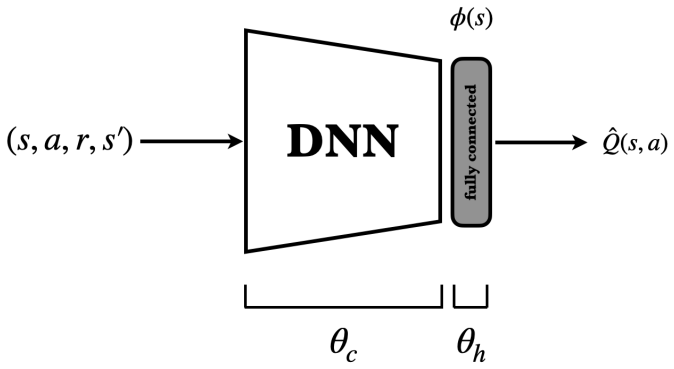


Figure 3.4: DQN pipeline. The main DQN block (the core) is typically a very deep neural network architecture, and the last layer (the head) is a fully connected layer.

policy [61, 42, 24]. As a result, much of the representation learning efforts in deep RL have been achieved through *auxiliary tasks*, whereby the agent uses its representation to learn other functions of the state such as visual reconstruction of observed states [61], inverse kinematics [113], and predicting other policies, value functions or the reward function [8, 42, 24]. The approach is similar to that of classical deep RL, where the policies and/or value functions are learned with a deep neural network. The main difference is that the RL learning objective is augmented with one or more supervised or unsupervised losses corresponding to auxiliary tasks. For example, an auxiliary loss ℓ_{aux} can be added to the loss in Equation (3.5):

$$\ell_\theta = (Q_\theta(s_t, a_t) - (r(s_t, a_t) + \gamma \max_a Q_\theta(s_{t+1}, a_{t+1})))^2 + \ell_{aux}. \tag{3.6}$$

Figure 3.5 illustrates the RL pipeline using auxiliary tasks. Several previous works that used various auxiliary tasks to learn rich representation in RL are discussed in Section 3.3. Auxiliary losses can take many forms, they can be supervised, unsupervised or semi-supervised. A particularly interesting class of loss functions have gained special attention in the RL community: contrastive losses, which we discuss next.

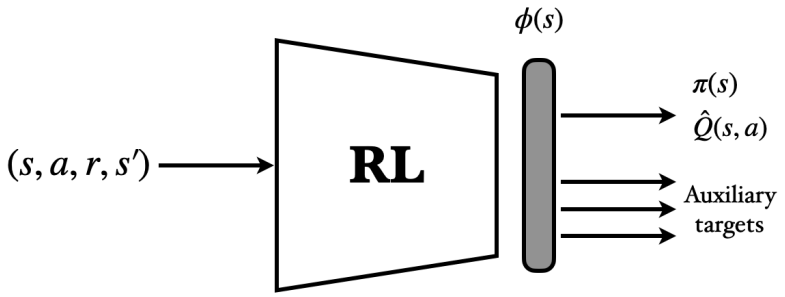


Figure 3.5: Representation learning for RL using auxiliary tasks. The RL component can be any function approximator. Typically it is a deep neural network, often made of layers of different architectures (*e.g.* convolutional or recurrent). The output layer of this module is the input of a dense layer, the representation. In addition to the RL objective (policy and/or value optimisation), auxiliary targets are jointly optimised end-to-end, shifting the representation.

3.2.3 Contrastive Learning

Contrastive learning allows to acquire knowledge about data in the absence of labels or external reward signal, by self-supervision. It has been developed specifically to learn rich representations of high dimensional data for use in various tasks, such as computer vision tasks, or natural language processing. The goal of contrastive learning is to learn a representational latent space in which data points that are close in the original space (*e.g.* the states s and s^+) with respect to a specific metric remain close to each other in the latent space ($\phi(s)$ and $\phi(s^+)$). This is often implemented by minimising a contrastive loss, where the distance between close points (*positive samples* (s, s^+)) is minimised and the distance between far apart points (*negative samples* (s, s^-)) is maximised. As the latent space projects the data points into a embedding vectors, the similarities between the anchor s and targets s^+ are often modeled with dot products ($\phi(s)^\top \phi(s^+)$) or cosine similarity metric ($\phi(s)^\top \phi(s^+) / \|\phi(s)\| \, \|\phi(s^+)\|$) [175, 52]. On way of modelling the contrastive loss function is with a softmax [45, 175]:

$$\ell_{s,s^+} = -\log \frac{\exp\big(\text{ sim}(\phi(s), \phi(s^+)\big)}{\sum_{s_k \in \mathcal{D}} \exp\big(\text{ sim}(\phi(s), \phi(s_k))\big)}, \tag{3.7}$$

where $\mathcal{D}$ is the data set of labelled samples and *sim* is the similarity function. Intuitively, minimising this loss incites the similarity between positive vectors to be as close to 1 as possible, while negative examples to be close to 0.

In RL settings, contrastive learning can be done as a preprocessing step when data is available, or it can be performed online while solving the task. In the latter case, a mechanism for generating positive and negative samples must be in place. Figure 3.6 depicts the online RL pipeline with contrastive learning.

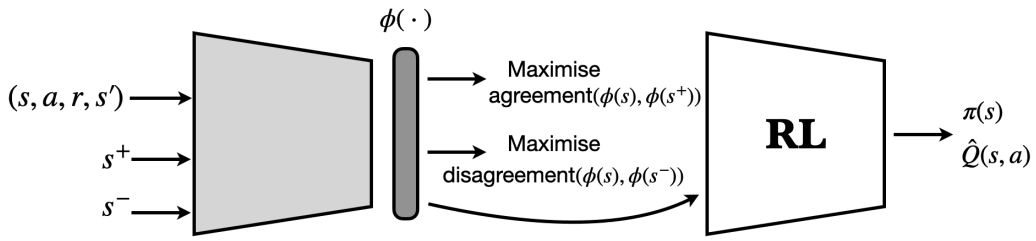


Figure 3.6: Representation learning for RL via contrastive learning. In this simplified schema, it is assumed that a positive sample s^+ and a negative sample s^- can be generated for each observed transition. The first block is the representation learning phase, which learns to maximise the agreement between the current state s and the positive sample s^+ given their respective representation while maximising the disagreement between the current state s and the negative sample s^-. The representation of the current state is simply the head of the representation learning module, and is used by the RL agent to learn a policy and/or a value function.

3.2.4 Unsupervised Learning

As discussed previously, representation learning can be performed as a separate step, prior to solving the RL task. When done in this manner, and when the learning of the representation is agnostic of the task, it is a form of *unsupervised learning*.

Unsupervised learning has seen promising progress in different AI fields, such as natural language processing and computer vision. Pre-training unsupervised models with large amount of data has enabled fine-tuning to downstream supervised learning tasks with limited labeled data, such as machine translation and image segmentation. This is particularly promising for real world RL applications, where it is often impossible to explore the entire state action space. Another motivation for

unsupervised representation learning in RL is that when the representation learning is performed online, it is susceptible to changes in the input distribution as the policy evolves. As a results, the representation needs to adapt continuously, resulting in poor learning efficiency. Figure 3.7 shows how the outputs of an unsupervised model can be used as the input of a RL model.

The representations that unsupervised algorithms yield are often referred to as *disentangled* representations, as they are disentangled from the reinforcement learning. For example, an unsupervised representation might capture information about the structure of the environment (the underlying dynamics), but will not embed any reward information: the representation is said to disentangle the reward from the transition dynamics.

Beyond unsupervised contrastive losses, recent works have introduced several methods for learning representations in an unsupervised way. We review such works in detail in Section 3.3.

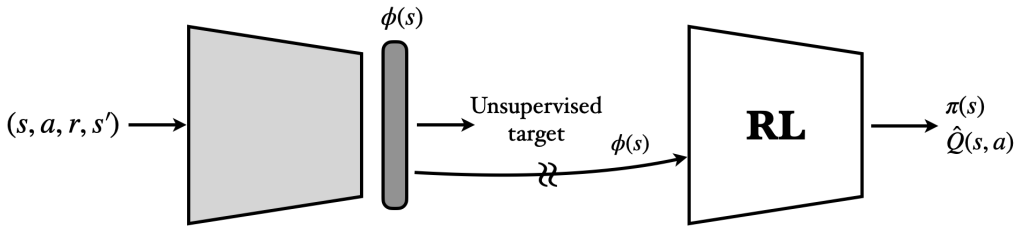


Figure 3.7: Unsupervised representation learning in RL. The interrupted arrow between the unsupervised module and the RL module signifies that the unsupervised module does not receive any feedback from the RL module (in deep learning, this equates to a stop-gradient).

3.3 Categorisation of Representational Properties

Having described *methods* for learning representations for RL, we now identify and present the four main *representational property* categories.

3.3.1 Representations supporting Policy Evaluation

We first consider the value prediction problem, where the goal is to find a good linear approximation $\hat{V}^{\pi}_{\phi,\theta} : \mathcal{S} \mapsto \mathbb{R}$ for the true value function under a policy π,

using a representation ϕ and parameters θ:

$$\hat{V}^{\pi}_{\phi,\theta}(s) = \phi(s)^{\top}\theta \approx V^{\pi}(s)\,. \tag{3.8}$$

We review state-of-the art representations, which aim to solve the following linear value function approximation objective

$$\operatorname*{argmin}_{\phi} \sum_{s\in\mathcal{S}} \left(\hat{V}^{\pi}_{\phi}(s) - V^{\pi}(s)\right)^2, \tag{3.9}$$

where $\hat{V}^{\pi}_{\phi}$ denotes the projection of V^{π} onto the linear subspace $H = \{\Phi\theta : \theta \in \mathbb{R}^d\}$. In the following, the works presented in the literature aimed at learning representations supporting policy evaluation, or value function approximation, are described against the different properties listed in Figure 3.2.

Fixed Representation. Much of the earlier representation efforts in reinforcement learning focused on constructing fixed basis architectures with different characteristics.

Proto-value functions (PVFs) [95] are representations that are constructed from the graph of states. The MDP's topology is captured in the graph of states, where each node corresponds to a state s, and is connected to another state s' by an edge if there is an action that can take the agent from s to s'. The weight on the edge usually corresponds to the transition probability $\sum_a P(s,a,s')$. Then, the value function can be viewed as a signal on the nodes of the graph. Figure 3.8 depicts the graph of state corresponding to a four-room environment [141].

The main characteristic of PVFs is that they capture the underlying dynamics of the environment. This property is motivated by the fact that value functions can be seen as smooth functions on the graph of states (see Figure 3.8), or in other words, value functions can be seen as the result of rewards diffusing through the state space under the environment dynamics [93]. Formally, PVFs are the first eigenvectors of a diffusion operator on the graph, such as the combinatorial graph Laplacian matrix

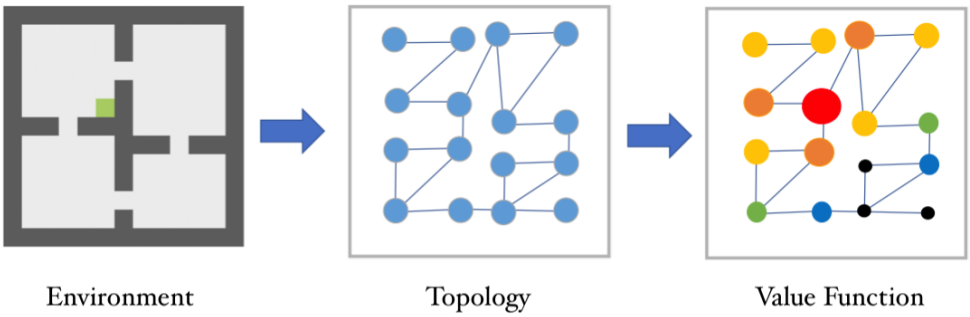


Figure 3.8: Example of a graph of states capturing the topology of a four-room environment and the value function as a function on the graph.

$$\mathcal{L} = D - A\,, \tag{3.10}$$

where A denotes the adjacency matrix of the graph of state, and D is the diagonal matrix whose entries are the row sums of A. The graph Laplacian is related in structure to the random walk diffusion model $D^{-1}A$ [see 95]. The normalised graph Laplacian $\mathcal{L} = D^{-1/2}(D-A)D^{-1/2}$ can also be used to generate PVFs. The eigenvectors of the graph Laplacian associated with different eigenvalues capture different temporal properties. By definition, the PVFs are reward agnostic and only capture information about the transition dynamics. The diagnolisation of the graph Laplacian, which is an essential operation to compute the PVFs, does not scale to large state spaces. Methods to mitigate this issue have been proposed but remain constraining as strong assumptions are required [174].

Krylov methods [116] have been proposed as an alternative way of computing basis functions for RL. The Krylov basis are powers of the transition matrix, multiplied by the reward vector $\{P^{k-1}\mathbf{r} : 1 \leq k \leq d\}$, where d denotes the number of basis functions. Krylov basis form a standard solution systems of linear equations and can be computed iteratively (by computing successive powers of the transition matrix). The relevance of Krylov methods to RL is shown in Petrik [116]. In particular, computing Krylov basis is less computationally demanding than computing the PVFs, as no spectral decomposition is needed.

Other iterative feature generation works have based their basis function con-

struction upon the current residual error. The basis functions yielded by these methods are known as Bellman Error Basis Functions (BEBF) [111]. Formally, starting with an arbitrary representation Φ_0 of dimension d, if the current estimate of the value function is given by $\Phi_k \mathbf{w}_k$, where $\mathbf{w}_k$ is the weight vector in the linear function approximation, the next BEBF is given by the Bellman error

$$\Phi_{k+1} = BE(\Phi_k) = \mathbf{r} + \gamma P \Phi_k \mathbf{w}_k - \Phi_k \mathbf{w}_k \tag{3.11}$$

The space spanned by the sequence of BEBFs is orthogonal to the real value function space. Therefore, any value function can be represented exactly with sufficient number of BEBFs.

Whereas PVFs are reward-insensitive, Krylov and BEBFs methods dilate the reward information geometrically through the transition space. Petrik [116] proposes a method for augmenting reward-specific Krylov basis with PVFs as a way of integrating localised high-frequency reward-specific features with the more global long-term information embedded in the eigenvector of the graph Laplacian.

State Abstraction. *Bisimulation* abstraction in MDPs is used to group states that have the same long-term behaviour [44]. It was developed from the notion of probabilistic bisimulation from process algebra [76]. Bisimulation is an equivalence relation that relates two states when (1) they produce the same immediate reward, and (2) they have precisely the same probability of transitioning to a class of equivalent states.

Definition 3. *(From Givan et al. [44]) Let $(\mathcal{S}, \mathcal{A}, P, r)$ be a finite Markov decision process. A stochastic bisimulation relation E is an equivalence relation on $\mathcal{S}$ if whenever sEt (s is equivalent to t with respect to equivalence relation E), the following properties are satisfied:*

1. $\forall a \in \mathcal{A}, \qquad r(s,a) = r(t,a)$

2. $\forall C \in S/E \quad \sum_{s' \in C} p(s,a,s') = \sum_{s' \in C} p(t,a,s')$

where S/E is the set of all equivalence classes in S with respect to equivalence relation R. Two states s and t are said to be bisimilar if there exists a bisimulation relation E such that sEt.

This strict notion of equivalence is problematic, as small numerical perturbations in the transition probabilities can make two equivalent states non equivalent. A *bisimulation metric*, a less sensitive way of defining behavioural similarity than bisimulation, was introduced by Ferns et al. [38].

Definition 4. *(From Ferns et al. [40]) A bisimulation metric is a distance d between states defined as*

$$d(s_i, s_j) = \max_{a \in \mathcal{A}} (1-c) \cdot |r(s_i,a) - r(s_j,a)| + c \cdot W_1(p(s_i,a,\cdot), p(s_j,a,\cdot); d). \quad (3.12)$$

with $c \in [0,1)$ and W_1 the Wasserstein-1 metric.

Bisimulation metrics have been shown to be quantitatively analogous to stochastic bisimulation, and efficient ways of calculating bisimulation metrics were introduced [39, 20].

The bisimulation notion does not take into account action similarity, thus MDP homomorphisms were introduced to address this limitation [123]. An MDP homomorphism is a transformation from an MDP M with some redundancy within the state action space to a reduced MDP M', where equivalent states in M are mapped to the same state in M', and equivalent actions in M to the same action in M'. A solution to M' yields a solution to the original MDP. Similarly to bisimulations, homomorphisms are brittle; small perturbations introduced by estimation errors will result in equivalent state-action pairs to become non equivalent. Approximate homomorphisms were proposed to allow for more flexibility in the aggregation [124], and an alternative metric, the *lax bisimulation metric*, was introduced to construct approximate homomorphism [152]. Taylor et al. [152] showed that the difference in the optimal value function of different states can be more tightly upper-bounded by the value of lax bisimulation than by that of bisimulation.

Auxiliary Tasks. *DeepMDP* [42] is a parameterised latent space model based on two auxiliary losses that are closely connected to bisimulation. Recall that bisimulation metrics define behavioural similarity between two states if they produce the same immediate reward and transition to states that are behaviourally similar. The auxiliary losses on which DeepMDP is trained directly relates to these two properties: predicting the rewards and predicting the distribution of next latent states. This relation to bisimulation allows DeepMDP to enjoy interesting guarantees. First, the minimisation of the losses guarantees that two non-bisimilar states can never collapse into a single representation. Secondly, the value functions in the DeepMDP are good approximations of value functions in the original MDP.

Bellemare et al. [8] call a representation *optimal* if it is a solution to an optimisation problem whose objective is to best approximate the value function of *all* stationary policies for a given MDP. That is, a representation ϕ that minimises the following min-max problem:

$$\min_{\phi} \max_{\pi} ||\hat{V}_{\phi}^{\pi} - V^{\pi}||_2^2 . \tag{3.13}$$

By looking at the geometry of the problem, and noting that the value function space is a polytope [25], Bellemare et al. [8] show that an expected-error relaxation of the representation learning problem in (3.13) can be restricted to a special subset of value functions, named *adversarial value functions* (AVFs), which correspond to the extremal vertices of the value function polytope. AVFs are the deterministic policies that either minimise or maximise the expected return at each state, based on the solution of the network-flow optimisation derived from an interest function. Authors show that predicting AVFs gives rise to optimal representation (in terms of value function approximation of arbitrary policies).

3.3.2 Representations supporting Policy Search

While ensuring accurate value function approximation is sufficient for prediction problems, this condition only might not be sufficient for control (the search of an optimal policy). Indeed, in control settings, the goal of the agent is not to evaluate

an arbitrary policy, rather to iteratively improve on the current policy, in order to uncover an optimal policy π^* associated with an optimal value function:

$$Q^{\pi^*}(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s,a,s') \max_{a'} Q^{\pi^*}(s',a') .$$

Standard RL algorithms solve the control problem by alternating policy evaluation with policy improvement, where a better policy π' is derived from the current value function Q^{π} according to

$$\pi'(s) = \operatorname*{argmax}_a Q^{\pi}(s,a) . \tag{3.14}$$

A representation that is optimised for specific value functions, even for the optimal ones, may be inadequate for representing the sequence of value functions leading to the optimal solution as the difference in value functions for different policies can be high [98, 79]. Standard value based RL methods such as TD(0) and Q-learning do not provide any convergence guarantees under linear function approximation, bootstrapping and off-policy data. This problem is known in reinforcement learning theory as the deadly triad [146]. Therefore, successful control requires having a mechanism for accurately approximating value functions corresponding to the polices in the sequence of improved policies. This section reviews the representation learning methods that aim at facilitating or improving the search for an optimal policy.

Fixed Representations. To help understand the conditions under which TD(0) can be trusted to converge, Ghosh and Bellemare [43] defined the following notion of stability of TD(0):

Definition 5. *(From Ghosh and Bellemare [43]) TD(0) is stable if there is a step-size* $\eta > 0$ *such that when starting from any* $\theta_0 \in \mathbb{R}^d$ *and taking updates according to*

$$\theta_{k+1} = \theta_k - \eta \nabla Q_{\theta_k}(s,a)(Q_{\theta_k}(s,a) - (r(s,a) + \gamma Q_{\theta_k}(s',a'))) ,$$

we have that

$$lim_{k\to\infty}\theta_k = \theta^*_{TD},$$

where θ^*_{TD} denotes the fixed-point solution of TD(0).

With respect to this notion of stability, authors found that TD(0) with linear value approximation is affected by the space of value functions that the representation can express and how the space is parametrised. In their analysis, they found that representations whose corresponding linear subspace is invariant of value functions that are closed under the transition dynamics of the policy are always stable. The *Schur representation*—a representation made of the first d Schur basis vectors in the Schur decomposition [78] of the transition matrix—is shown to span invariant subspaces, and therefore is stable under TD(0).

Auxiliary tasks. While guaranteed to be stable under TD(0), the Schur representation can only be constructed exactly through Schur decomposition or orthogonal iteration when the transition matrix is known. Ghosh and Bellemare [43] demonstrate how the orthogonal iteration scheme can be approximated using an auxiliary loss function and a target network: at each step, the algorithm predicts the future feature values given by a fixed target representation network, and infrequently refreshes the target representation network with the current one.

To study the problem relating to value-based methods having to handle shifts in the distribution of states and hence in their values while the policy improves, Dabney et al. [24] introduced a characterisation of value functions produced by the policy improvement process of RL algorithms:

Definition 6. *(From Dabney et al. [24]) A sequence $\{Q_{\pi_0}, Q_{\pi_1}, \ldots, Q_{\pi_*}\}$ is called a* ***value-improvement path (VIP)*** *if $Q_{\pi_{i+1}} \succeq Q_{\pi_i}$ for $i = 0, 1, \ldots$.*

Because solving a control task via policy iteration involves traversing the space of value functions in the value-improvement path, Dabney et al. [24] suggest that a good representation should allow for good approximations of all values in the algorithm's value-improvement path. They used this intuition to design auxiliary tasks

that learn to predict the value functions of past policies in the trajectory of improving policies seen during training. Authors show that this leads to a representation that spans past policies and also provide an accurate value approximation for future policies in the value improvement path.

The auxiliary task of predicting future states was proven to be successful for the policy evaluation problem as we have seen with DeepMDP [42]. Schwarzer et al. [131] proposed a similar idea to augment deep RL algorithms for control tasks. The key idea is to learn Self-Predictive Representations (SPR) by training agents to predict their *own* latent state representation multiple steps in the future in addition to maximising future reward. This auxiliary loss, combined with data augmentation to the future prediction loss result in a representation that is temporally predictive and consistent across different views of observations. The empirical results demonstrate that augmenting deep RL agents with SPR results in strong data-efficiency improvements on 26 Atari games [6].

DeepMDP [42] was discussed in the previous section as a latent space model with theoretical guarantees for good approximations of value functions. However, it is worthwhile to note that learning a DeepMDP as an auxiliary task in a standard deep RL algorithm for control (agent C51 [7]) leads to experimental improvement as well.

State Abstraction. Zhang et al. [176] proposed Deep Bisimulation for Control (DBC), another representation learning method based on the bisimulation metric, but specifically designed for control. Where the DeepMDP representation upper bounds the bisimulation distance, the distance in the latent space learned by DBC is exactly the bisimulation distance. DBC learns a bisimulation metric with gradient decent jointly while learning to solve the control task. This approach leads to a learned representation that improves as the policy improves online, making it suitable for control tasks. Authors demonstrate that DBC learns representations that capture task-relevant elements of the state and is in fact invariant to task-irrelevant information by evaluating the performance on a modified MuJoCo task [155], where the background is replaced with moving distractors and natural videos.

A recent form of state-action abstraction called Z^{π}-irrelevance was formalised in [83] and shown to be coarser than bisimulation or homomorphism. The Z^{π}-irrelevance abstraction aggregates state-action pairs with similar return distributions under policy π. Authors showed that this novel abstraction can reduce the size of the state-action space as well as approximate the state-action value functions arbitrarily accurately. Because Z^{π}-irrelevance cannot be used in practice when the full distribution is not known, Liu et al. [83] proposed Z-learning, a method approximate a Z^{π}-irrelevance abstraction from sampled returns.

Contrastive Representations. Contrastive Predictive Coding (CPC) [158] is a representation learning method designed to learn representations by predicting the future in latent space using a probabilistic contrastive loss. CPC can be summarised as follows: First, the input data in the original observational space is compressed into a compact latent embedding space. Then, an autoregressive model is used in this latent space to make prediction many steps ahead with a loss based on Noise-Contrastive Estimation [48]. Augmenting the A3C deep RL agent [103] with CPC empirically shows strong performance in five of the DeepMind Lab [5] 3D RL environments.

Contrastive Unsupervised Representation for Reinforcement Learning (CURL) [137] is a framework for learning semantic representations from visual high dimensional observations that combines contrastive learning with model-free RL. CURL trains an encoder from pixel based observations by enforcing that the representations of augmented versions of the original observations are in agreement using a contrastive loss. The positive and negative samples are constructed from the minibatch sampled for the RL update. CURL uses random cropping as data augmentation for generating positive samples, and other random images for negatives. Plugging the CURL framework into the Soft-Actor-Critic (SAC) agent [49] results in higher performance on DeepMind Control [151] and 16 Atari games [6]. However, a follow-up work suggests that the improvement brought by CURL comes from the data-augmentation only, and not from the contrastive loss [77].

Liu et al. [83] proposed Return-based Contrastive representation learning for

Reinforcement Learning (RCRL), a method that takes into account reward signals in a contrastive loss in an effort to learn state-action representations that capture only return-relevant features to accelerate RL algorithms. Given a state-action pair, RCRL selects a state-action pair with the same or similar return as positive sample and a pair with different return as negative sample. A discriminator is then trained to classify between positive and negative samples based on their representation. Empirical results demonstrate that RCRL combined with the Rainbow agent [54] achieves improved performance in Atari games [6].

Unsupervised Representations. Prior to the emergence of DQNs [102], Lange and Riedmiller [74] proposed Deep Fitted Q-Iteration (DFQ), a method for learning compact features for RL using deep learning. They integrate a deep auto-encoder [57] into RL for learning visual navigation tasks. The auto-encoder is an unsupervised mechanism that is trained to reconstruct the image observations. The agent uses the encoder to translate the collected observations into the compact feature space and applies Fitted Q-Iteration [35] to learn a policy. DFQ was shown to perform better than an experienced human player on a visual racing slot car task [75].

In addition to predicting reward signals (similarly to Liu et al. [83]), the UNsupervised Reinforcement and Auxiliary Learning (UNREAL) [61] method introduced an additional unsupervised loss for visual control tasks: predicting changes in pixel values of different regions of the input. The UNREAL agent is trained on-policy with the AC3 loss [103], and the auxiliary tasks are learned with data from a replay buffer collected while solving the task. UNREAL outperformed the vanilla A3C agent on the challenging Labyrinth task [103].

3.3.3 Representations supporting Generalisation

A natural way of thinking about data-efficient representations for RL is through the lens of generalisation. Intuitively, a representation that is able to generalise across similar states, across similar tasks or across similar environments, will help speed up the learning, as useful information can be spread to unseen states, tasks or environments through generalisation. Additionally, generalisation is also key to

designing RL agents that can be deployed in the real world, as the reality of the real world is often different from controlled training settings. Typical RL methods are trained and evaluated on the same environment, which does not reflect the reality of the vast range of practical applications.

A recent survey on generalisation in deep RL [70] captures the different types of generalisation using the notion of task distributions: Kirk et al. [70] identify Independent and Identically Distributed (IID) Environments, where the training and testing tasks are independently drawn from the same distribution (*e.g.* different instances of the same Atari game), and Out Of Distribution (OOD) Generalisation Environments, where training and testing tasks are drawn from different distributions (*e.g.* simulation vs. real world). In order for RL agents to be of practical use in either of these scenarios, the policies they learn should be able to generalise well to novel situations at deployment time. In this chapter, we review prior works that have tackled the problem of representation learning for RL from a generalisation perspective. This includes efforts to learn representations with transferable knowledge from one task to another, and representations that are tasks agnostic.

Fixed Representation Dayan [27] formalised a state representation based on temporal succession that is completely independent of the reward. The Successor Representation (SR) is a predictive representation in which states are represented by the expected count of future occupancy of successor states under a fixed policy. Formally, the SR with respect to a policy π is defined as

$$\Psi_\pi(s,s') = \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty}\gamma^t \mathbb{I}_{\{S_t=s'\}}\Big|S_0=s\Big], \tag{3.15}$$

where $\mathbb{I}$ denotes the indicator function. The representation of state s is thus an $|\mathcal{S}|$-dimensional vector whose rows are expected discounted occupancies from s to each state in the environment. The SR matrix can equivalently be written as

$$\Psi_\pi = \sum_{t=0}^{\infty}(\gamma P_\pi)^t = (\mathbf{I}-\gamma P_\pi)^{-1}, \tag{3.16}$$

where P_π is the transition matrix under policy π. With this notation, it is easy to see that the SR disentangles the transition matrix from the reward in the computation of the value function:

$$V^\pi = (\mathbf{I} - \gamma P_\pi)^{-1}\mathbf{r} = \Psi_\pi \mathbf{r}. \tag{3.17}$$

This makes the SR well suited for generalisation across tasks under the same transition dynamics and different reward functions. With the Bellman equations formulation, the SR can be estimated from samples with TD learning, where the reward function is replaced by state occupancy.

State Abstraction Castro and Precup [21] have used a modified version the lax-bisimulation metric [152] to transfer knowledge from a policy computed on an MDP $M_1 = (\mathcal{S}_1, \mathcal{A}_1, p_1, r_1)$ to a another MDP $M_2 = (\mathcal{S}_2, \mathcal{A}_2, p_2, r_2)$. Simply put, the proposed method uses lax-bisimulation to determine a state-action distance mapping d_L between the MDPs. Then, given an optimal policy π_1^* under M_1, for $t \in \mathcal{S}_2$, $\pi_L(t)$ finds the closest state $s \in \mathcal{S}_1$ to t under d_L and chooses the action b from t that is closest to $\pi_1^*(s)$. The loss incurred when using the transferred policy is tightly bounded, as shown in [21]. Lax-bisimulation metrics are computationally expensive to compute, and approximation strategies have been proposed to address the inefficiency limitation.

Agarwal et al. [1] introduced a new metric suited for generalisation across similar tasks built on the concept of bisimulation metric, called Policy Similarity Metric (PSM). The core idea is to learn a metric that can inform on which states result in similar behaviour, and which do not, without considering rewards as they can be too restrictive or too permissive and do not generalise across tasks with similar dynamics but different reward functions. PSM considers similarity between policies themselves. In Equation (3.12), the absolute reward difference is replaced by a probability pseudometric between policies, and use the optimal policy π^* as the grounding policy to ensure that the metric captures the similarity in optimal behaviour. PSM can be computed iteratively using dynamic programming, and an approximation $\hat{\pi}^*$ can be used when the optimal policy is unknown. PSM gives

an upper bound on suboptimality of policies transferred from one environment to another [see 1, Theorem 1].

The Combined Reinforcement Leaning via Abstract Representations (CRAR) [41] is a deep RL architecture based on the idea that learning a state abstraction by integrating model-based principles into model-free learning will provide features with good generalisation ability since they must perform well for both the model-based and the model-free predictions. CRAR is made of several components; a state encoder, a model-free component based on the double DQN architecture [51], and a model-based component. The model-free learner updates current Q-value estimates from past experiences stored in replay buffer, updating the parameters of both the Q-network and the encoder at each step. The model-based learning is trained to predict the reward function, the discount factor and the transition function, updating the weights of both the model-based component and the encoder. Because CRAR contains many moving parts, it makes it well suited for transfer learning. Indeed, as the task changes, some elements of the architecture can be frozen to allow only minimal re-training. Authors have shown some empirical evidence of transfer learning using CRAR on labyrinths tasks [41].

Auxiliary Tasks The Successor Representation (SR) [27] discussed previously is limited to the tabular case and cannot readily be applied to large state spaces. Barreto et al. [3] have introduced a generalisation of the SR, called Successor Features (SFs). Given a state-action representation ϕ, the SFs under policy $\boldsymbol{\pi}$ are defined as

$$\psi_{\pi}(s,a) = \mathbb{E}_{\pi}\Big[\sum_{i=0}^{\infty} \gamma^{i}\phi(S_{t+i},A_{t+i})|S_t = s, A_t = a\Big].$$

Similarly to the SR, following Bellman equations, the SFs can be learned by TD learning. [3, 4] consider the multi-task setting in which all the tasks share common dynamics and are defined by different reward functions. As the agent is learning to solve a task, authors proposed to jointly learn the SFs corresponding to the task as an auxiliary task. Then, when the agent is faced with a new task, it can use the previously computed SFs to transfer knowledge before starting to improve the

policy for the new task. Authors provide a theoretical guarantee for the notion that an agent should perform well on a novel task if it has seen a similar task before, and the experimental results show how the SFs can be used to promote this kind of transfer.

Another generalisation of the SR has been proposed by Ma et al. [84]. Using the assumption that reward functions are linear combinations of some state-action features $\phi \in \mathbb{R}^d$ and the definition of general value functions [149], the authors derive the following:

$$V_g^{\pi}(s) = \mathbb{E}^{\pi}\left[\phi(S_t, A_t)\prod_{k=0}^{t}\gamma_g(S_k)|S_0 = s\right]^{\top}\mathbf{w}_g = \psi_g^{\pi}(s)^{\top}\mathbf{w}_g$$

where g defines a task identification, $\gamma_g(s) \in [0,1]$ is a pseudo-discount function for task g ($\gamma_g(s)$ could be zero when s is a terminal state for task g and a fixed value $gamma \in (0,1]$ otherwise), $\psi_g^{\pi}(s)$ defines the Universal Successor Representation (USR) of state s, and $\mathbf{w}_g$ is a task specific weight vector. Ma et al. [84] propose an deep neural network architecture to model $\psi^{\pi}(s,g;\boldsymbol{\theta}_{\psi})$, an approximation of the USR parameterised by $\boldsymbol{\theta}_{\psi}$, the policy $\pi(s,g;\boldsymbol{\theta}_{\pi})$ parameterised by $\boldsymbol{\theta}_{\pi}$, and the weights $\mathbf{w}(g;\boldsymbol{\theta}_w)$ parameterised by $\boldsymbol{\theta}_w$. The network is such that the module learning the policy and the module approximating the USR share some parameters and are trained using an actor-critic method. The trained USR can then be used as an initialisation for exploring unseen tasks and/or to directly compute a policy for a new task.

Other types of predictive representations have been explored in the context of general representation for multitask RL. Predictions of Bootstrapped Latents (PBL) [47] learns a representation of the agent state by predicting the future latent embeddings of observation through two auxiliary tasks. The first prediction task is a forward, action-conditional prediction from histories (agent states) to future latent observations, and the sencod prediction task is a reverse prediction from latent observation to agent states. In this way, the agent states learn to predict future latent observations and at the same time future latent observations learn to predict the corresponding future agent state. The intuition is that this bootstrapping effect may

help capture contextual information about the dynamics of the environment. PBL demonstrated strong performance on DeepMind Lab environments [5] in multitask setting.

Contrastive Representation The Contrastive Metric Embeddings (CMEs) algorithm [1] is a general procedure to learn an embedding given a state similarity metric d. The set of positive and negative pairs are defined using d. The similarity metric is also used to assign importance weights to the positive and negative pairs. A contrastive loss is then minimised on this data. Policy Similarity Embeddings (PSEs) are CMEs learned with PSM as the similarity metric. PSEs can be learned jointly by an RL agent while solving a task. PSEs shows better generalisation capability than bisimulation transfer [21] on the Jumping Task [28] when evaluated on tasks unseen during training. Agarwal et al. [1].

Unsupervised Representation. The DisentAngled Representation Learning Agent (DARLA) [56] is an RL agent that was designed to generalise well under domain adaptation challenges. In domain adaptation scenarios, the agent interacts with a source domain during training, and is then deployed on a target domain. Both domains correspond to different MDPs. Typically, the state spaces of each MDP are considerably different, but the action space, transition and reward functions share structural similarities (*e.g.* in robotics, computer simulation would be the source domain and the real world would be the target domain). DARLA consists of three steps: first the agent learns to see in an unsupervised manner, training a variational autoencoder (β-VAE [55]) on observation samples collected at random, then it learns to act in the source domain, using any RL algorithm, and finally it transfers its knowledge in the target domain by running the learned policy. DARLA demonstrated robust zero-shot transfer on the DeepMind Lab 3D environment Beattie et al. [5].

3.3.4 Representations supporting Exploration

Contrary to supervised learning, the data used by reinforcement learning agents must be gathered by the agent itself. Indeed, when the agent initially starts acting in an environment, it has no prior knowledge of the environment dynamics and reward signal, and must acquire information by interacting with the environment. This induces the *exploration-exploitation* trade-off, a crucial open problem in RL, whereby the agent must decide between relying on current knowledge and risking sub-optimal decision making, or make the assumption that acquiring information on parts of the environment that is currently unknown could help discovering better rewards, risking losing the chance of gaining immediate reinforcement. Intuitively, a good exploration strategy will depend on the structure of the MDP, as well the information contained with the observation the agent has access to. For example, consider a feature space in which a state's representation contains information about states in its immediate proximity (*i.e.* reachable with few steps). Then perhaps a local exploration will not be necessary, and the agent would benefit more from exploring other regions of the state space. This motivates the study of representations that can support optimal exploration. In this section, we give an overview of prior works that have focused on designing, learning and utilising representations specifically for the discovery of better exploration strategies.

We start by introducing the *option framework*, an important concept that provides a way to implement hierarchies and macro-action in RL and has been used in the literature to design exploration strategies.

3.3.4.1 The Option Framework

Often, decision making problems involve interacting with the environment at different time scales (*e.g.* , the task of driving a car includes skills at different time scales, such as changing gears, making a left turn, or navigating from a location A to a location B). The option framework [141, 120] formalises the notion of reasoning in terms of interaction at different temporal scales; options defines actions extended in time. An option $\omega \in \Omega$ is a tuple $\omega = (\mathcal{I}_\omega, \pi_\omega, \beta_\omega)$, where $\mathcal{I}_\omega \in \mathcal{S}$ denotes the option's initiation set, $\pi_\omega : \mathcal{S} \times \mathcal{A} \mapsto [0,1]$ denotes the option's policy,

and $\beta_\omega : \mathcal{S} \mapsto [0,1]$ is the option's termination condition (the probability that ω will terminate at a given state). The option framework is a particularly useful for exploration, as options can allow for temporally-extended exploration. Some of the works review in the following have used this formalism to learn representations supporting options discovery for temporally-extended exploration.

Fixed Representation. Eigenoptions [85] are options that emerged from using PVFs [95] to discover options useful for exploration. Because PVFs capture the large-scale geometry of the environment at different time-scales, Machado et al. [85] proposed to use PVFs to define purposes for the agent: to maximise the discounted sum of the different PVFs.

Definition 7. *(From Machado et al. [85]) An eigenpurpose is the intrinsic reward function $r^{\mathbf{e}}(s,s')$ of a proto-value function $\mathbf{e} \in \mathbb{R}^{|\mathcal{S}|}$ such that*

$$r^{\mathbf{e}}(s,s') = \mathbf{e}^\top(\phi(s') - \phi(s)), \tag{3.18}$$

where $\phi(s)$ denotes the feature representation of state s.

Each eigenpurpose is then used to define an eigenoption, where the option's policy is a policy that is optimal with respect to the eigenpurpose, and the termination condition is when the purpose is achieved (when the agent reaches the state with largest value in the eigenpurpose). Machado et al. [85] show how, in the traditional four-room domain [141], eigenoptions improve exploration by reducing diffusion time (the expected number of steps required to navigate between any two states under a uniform policy).

Auxiliary Tasks. Another approach of using a representation for exploration in RL is to use the norm of the SR as an exploration bonus. Inspired by the diffusion properties exhibited by the SR, Machado et al. [86] suggest to incorporate the norm of the SR as an exploration in the Sarsa [145] update. The new Sarsa+SR update is

described by

$$\hat{Q}(s_t,a_t) \leftarrow \hat{Q}(s_t,a_t) + \alpha\big(r(s_t,a_t) + \beta\frac{1}{||\hat{\Psi}(s_t)||} + \gamma\hat{Q}(s_{t+1},a_{t+1}) - \hat{Q}(s_t,a_t)\big)\,, \tag{3.19}$$

where β is a scaling factor, and the learned SR $\hat{\Psi}$ is updated at each time step with a TD update prior to executing the Sarsa+SR update. Augmenting deep RL agents with the SR based exploration bonus demonstrate goods performance on Atari games [6] with sparse reward (thus hard exploration).

To benefit exploration, Pathak et al. [113] have taken an interesting approach to learn a representation that is curiosity-driven. They argue that in order to incentivise a type of curiosity that will help the agent explore its environment, a representation should only model the things that can be controlled by the agent or that affect the agent. The Intrinsic Curiosity Module (ICM) architecture was designed to learn a representation with the aforementioned desired properties. ICM is a deep neural network with two sub-modules. The first module encodes raw observations into feature vectors $\phi(s)$ (*forward dynamics model*), and the second module predicts the action taken by the agent to move from one state to another (*inverse dynamics model*). The ICM agent is capable of learning the exploration behavior of moving along corridors and across rooms in the VizDoom environment [67], without any external rewards from the environment. ICM is also able to achieve high external reward in VizDoom when uncontrollable distractors are added to the observation, suggesting that the representation learned by ICM are indeed robust to things that the agent cannot control or is not affected by.

Contrastive Representation Exploration-Driven Representation Learning (EDRL) was designed to address the problem that when the agent is not provided with a uniform prior over the state space (for example tasks where the initial state is always fixed), learning meaningful contrastive representations is challenging, as the data used to train the representations will most likely not reflect the entire state space. The EDRL framework handles the exploration along with the representation learning in order to preserve the representational quality. It learns a skill-based covering

strategy along with the representation. The representation is used to train directional skills as to cover the thus far explored area, while the skills discover unexplored parts of the state to provide new knowledge that will be used to refine the representation. The EDRL agent is shown to progressively explore the state space and extend the representation domain, even without a uniform prior over the state space.

Unsupervised Representation. While having shown good performance in tabular environments and environments with a provided linear feature representation, eigenoptions [85] do not naturally scale to stochastic environments with non-enumarated states. Machado et al. [87] leverage the fact that the PVFs are equal to the eigenvectors of the SR [see 139] and propose to learn SFs (as they constitute a generalisation of the SR [3]), to discover eigenoptions. The SFs are learned in an unsupervised way with a deep neural network. The neural network receives input in its original observational space and learns to estimate the successor features of a lower-dimension representation learned by the neural network. To prevent the network to collapse to the zero fixed point, the additional task of predicting the next state is added. Using this mechanism to discover eigenoptions on four Atari games [6], led to agents exhibiting interesting exploratory behaviours, visiting corners and other relevant parts of the state space.

3.4 Unifying Representations in RL

We arrange the works reviewed in Section 3.3 in Table 3.1 according to the main methodology they adopt (rows) and the main property they target (column).

We start our discussion by observing that although distinct methods have emerged from the existing body of work on representation learning for RL, the different methods are not necessarily mutually exclusive. Then, we discuss the relationship between the representational properties in the context of supporting the RL problem as a whole.

	Policy Evaluation	Policy Search	Exploration	Generalisation
Fixed Representation	PVFs [95] Krylov [116] BEBFs [111]	Schur [43]	Eigen-options [85]	SR [27]
State Abstraction	Bisim. [44, 37] Homomorph. [152]	DBC [176] Z^{π}-irrelev. [83] MICo [22]	SR [88]	CRAR [41] Bisim. [21] PSM [1]
Auxiliary Tasks	AVFs [8] DeepMDP [42]	VIP [24] SPR [131]	ICM [113] SR [86]	SF [3, 4] Univ. SR [84] PBL [47]
Contrastive Representation		CPC [158] CURL [137] RCRL [83]	EDRL [36]	PSEs [1] state2vec [89]
Unsupervised Representation		Auto-encoder [74, 75] UNREAL [61] SGI [132]	Eigen-options [87]	DARLA [56]

Table 3.1: Unifying representation approaches in RL. Columns discriminate the different representational properties while rows differentiate between the methods for computing or learning the representation. The row with **blue header** corresponds to methods that compute fixed representations, whereas the rows with **green headers** correspond to methods that learn representations from data. The **purple** entries designate methods enjoying theoretical guarantees.

3.4.1 Representation Methods Overlap

In Table 3.1, we categorise previous works based on their most obvious representation method and property, but it is important to note that there exists some overlap. For instance, Section 3.2, we established that contrastive representations are a special case of other learning methods, as they can be learned online using an auxiliary loss, or in an unsupervised manner (Figure 3.9 visualises this overlap). However, there exist other interesting links between the representation methods.

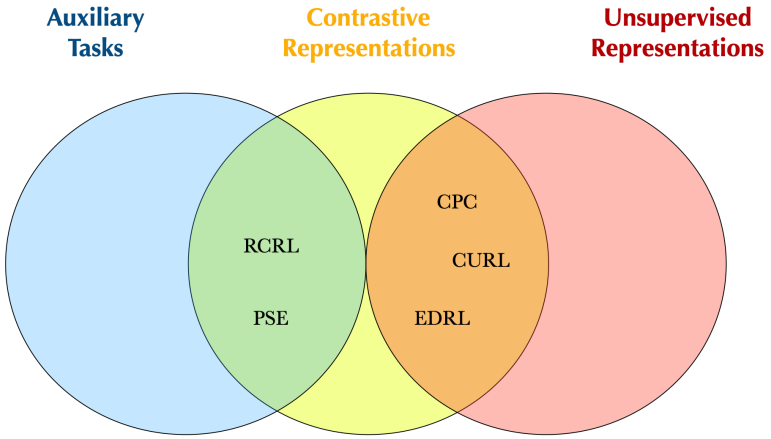


Figure 3.9: Visual depiction of the overlap between representations learned via auxiliary tasks, unsupervised learning and contrastive learning.

For example, it is important to note that some fixed representations can be approximated from data as opposed to constructed from the transition matrix and/or the reward function when these elements are unknown. For instance, Ghosh and Bellemare [43] show how the Schur representation can be learned by optimising the auxiliary objective of predicting the feature values of a fixed target representation network at the next time step (where the target network is periodically refreshed with the current representation). Similarly, a d-dimensional Krylov representation can be learned with the auxiliary task of predicting reward values at the next d time-steps [43]. Furthermore, an approximation of the Successor Representation [27] (considered fixed) can also be learned as an auxiliary task, using a version of TD learning, where the reward function is replaced by an indicator function of state occupancy. Another type of overlap worth noting is between unsupervised, or disentangled, representations and fixed representations. The PVFs [95] and the SR

[27] are a good examples of such overlap. They are constructed directly form the transition matrix while being completely insensitive to the reward. In fact, the SR can easily be shown to disentangle the reward from the transition dynamics in the value function.

These overlaps suggest that there is a broad range of representation learning methods, and that they can be highly flexible. Further investigation and comparative studies between representations with similar goals but different methods is of interest and left for future work.

3.4.2 Unifying State Abstractions

We now turn our attention to the interesting relationships between state abstraction mechanisms in use in RL. We notice a pattern of methods that develop strict to more flexible, and fine-grained to coarser abstractions. We visualise this phenomenon in Figure 3.10: abstractions are sorted with respect to their level of granularity from left to right: bisimulation equivalences and bisimulation metrics are abstraction that consider only state similarity, MDP homomorphisms, lax-bisimulations and Z^{π}-equivalences also consider action similarity, while policy similarity metrics take a step further and consider policy similarity. The vertical orientation represents the abstraction rigidity: at the top, we show the strictest form of abstractions, bisimulation equivalences and MDP homomorphisms, while their more flexible analogues are shown at the bottom. The bubbles indicate representation learning algorithms that learn the corresponding state abstraction.

The choice of state abstraction undoubtedly depends on the size and nature of the original MDP. Evaluating state abstractions with respect to their rigidity and granularity on different MDP is an interesting direction for future work.

3.4.3 Unifying Spectral and Predictive Representations

A representation equivalence that is interesting to note is between the PVFs (spectrum of the graph Laplacian) and the SR (predictive representation of future state occurrence). As discussed in [139], the eigenvectors of the normalised graph Laplacian are equal to the eigenvectors of the SR scaled by $\gamma^{-1}D^{1/2}$:

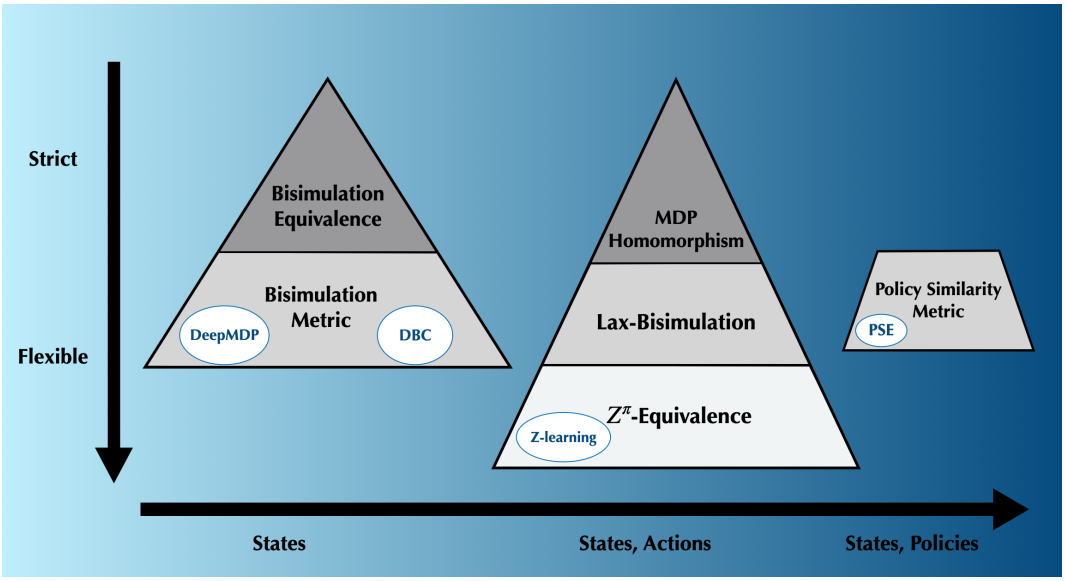


Figure 3.10: Unifying state abstractions in RL. From left to right, the abstractions are coarser, while from top to bottom the abstractions allow for more flexibility. The representation learning methods shown in bubbles are algorithms that learn to approximate the abstraction of the corresponding segment.

Theorem 1. *(From Stachenfeld et al. [139]) Let* $0 < \gamma < 1$ *such that* $\Psi = (I - \gamma P)^{-1}$ *denotes the SR matrix, and* $\mathcal{L} = D^{-1/2}(D-A)D^{-1/2}$ *denotes the normalised graph Laplacian matrix, both obtained under a uniform random policy. The* j*-th eigenvalue* $\lambda_{SR,i}$ *of* Ψ *and the* j*-th eigenvalue* $\lambda_{PVF,j}$ *of* $\mathcal{L}$ *are related as follows*

$$\lambda_{PVF,j} = \left[1 - (1 - \lambda_{SR,i}^{-1})\gamma^{-1}\right].$$

The i*-th eigenvector* $\mathbf{e}_{SR,i}$ *of the SR and the* j*-th eigenvector* $\mathbf{e}_{PVF,j}$ *of the normalised graph Laplacian, where* $i + j = |S| + 1$*, are equivalent up to scaling*

$$\mathbf{e}_{PVF,j} = (\gamma^{-1}D^{1/2})\mathbf{e}_{SR,i}.$$

This equivalence is particularly interesting for two reasons. Fist, it implies that the graph signal processing tools developed in the literature using the graph Laplacian [134] can be effortlessly extended to the SR. Secondly, from this equivalence arises the interpretation that working with the PVFs equates to working in the spectral domain, whereas working with the SR equates to working in the tem-

poral domain. Naturally, this raises the question of whether a method combining both approaches would be beneficial. This idea was in fact explored by [138] for supervised learning, yielding promising results.

Interestingly, concept of adversarial value functions (AVFs) [8] as auxiliary tasks is also closely related to PVFs. Indeed, the PVFs can be interpreted as defining a set of value-based auxiliary tasks, based on the set of value functions $\{(I-\gamma P^{\pi})^{-1} r_x\}_{s\in\mathcal{S}}$, where $r_s(s') := \mathbb{I}_{\{s'=s\}}$ defines an indicator reward function and π is the uniformly random policy.

In Figure 3.11 depicts these equivalences and similarities, as well as hierarchical relationships between spectral representations and predictive representations discussed in Section 3.3.

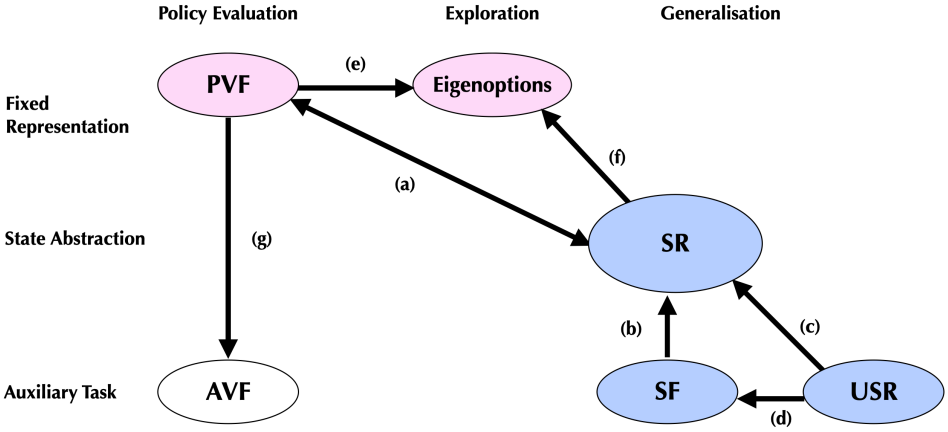


Figure 3.11: Visualising the relationship between spectral representations (**pink**) and predictive representations (**blue**). **(a)** The PVFs are equivalent to the eigenvectors of the SR up to a scaling factor. **(b)** The SFs generalises the SR from tabular to state features. **(c)** and **(d)** The USR generalises both the SR and the SF via the use of general value function, by allowing to model SRs and SFs of different policies and different tasks. **(e)** Eigenoptions are derived directly from the PVFs, or **(f)** can be computed from the SR. **(g)** AVFs share a similarity with PVFs as they both are solutions to a set of value function based auxiliary tasks.

3.5 What is a Good Representation for RL?

We now discuss open problems relating to representation learning in RL, highlighting differences, contradictions, limitations and gaps in the current state of the art.

3.5.1 Evaluating Representations in RL

To be able to answer the question of *"What is a Good Representation for RL?"* we first need a systematic way of evaluating representations. However, assessing the quality of any given representation for RL is difficult, since representation learning itself does not have a clearly defined objective beyond supporting the RL problem as a whole. As captured in this work, there are different ways to understand what makes a representation desirable for RL. Some works have sought to find a representation that will benefit value function approximation [93, 116, 111, 44, 37, 152, 42, 8], and therefore use the value function approximation error as evaluation metric. Others aimed to uncover a representation that is best suited for fast and stable control [43, 83, 22, 24, 131, 158, 137, 74, 75, 61], and evaluate performance by looking at the learning curve of the expected long term return. Representation learning efforts in RL that have investigated representations supporting generalisation [27, 21, 1, 3, 4, 84, 89, 47, 56] often use the long term return on unseen tasks as the performance metric. On the other hand, prior works focusing on learning or using a representation that will benefit exploration [86, 87, 113, 85, 36, 88], do not agree on a unified way of measuring performance (Machado et al. [85] proposed to adopt the *diffusion time* as an evaluation metric).

Although representation supporting different goals have respective (proposed) ways of evaluating the performance of the representations, there remains a lack of discussion on how the different properties relate to the overall performance.

3.5.2 What are the Desirable Representational Properties?

At a first glance, we might be tempted to think that the ideal representation would have all four properties identified in this work (to support policy evaluation, control, exploration and generalisation). However, some of these properties are in fact in direct conflict. For instance, an optimal representation for policy evaluation can harm control. Recall that given an MDP, the space of value functions (*i.e.* the set of all value functions that are attained by some policy) forms a polytope [25]. Consider the space of value functions of a two-state MDP shown in Figure 3.12. Assume we know a representation Φ that is optimal for evaluating an initial policy π_0, meaning

that the value function under π_0 lies directly on the representation space (*i.e.* zero approximation error). If the goal is to improve π_0 to reach the optimal π^* using a representation Φ, we need to ensure that V^{π^*} can be well approximated using Φ. However, in the example of Figure 3.12, the approximation error of the optimal value function is very large, which intuitively means that attempting to learn the optimal policy starting from π_0 with representation Φ will likely lead to poor results.

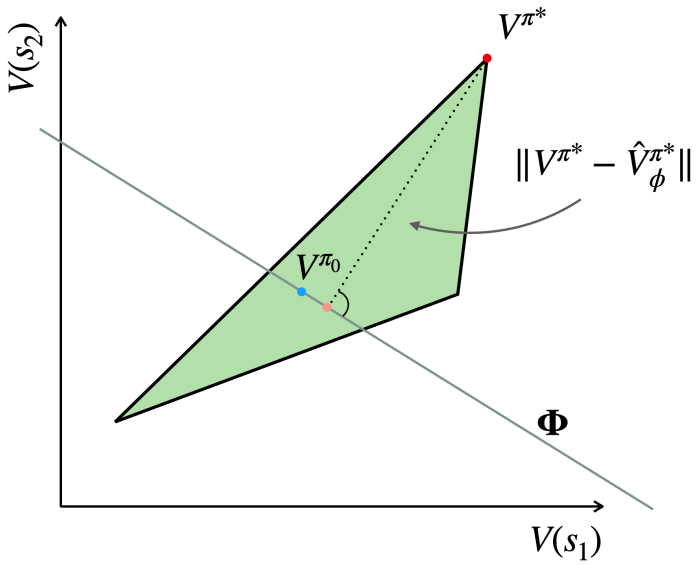


Figure 3.12: The value function polytope of a two-state MDP. Each point in the polytope represents a value function with respect to a policy. The line $\mathbf{\Phi}$ represents a representation space onto which value function can be projected. The dotted line corresponds to the projection of the optimal value function onto the representation space.

On the other hand, there are cases where a representational property can benefit another property. For example, a representation that enables efficient exploration would ultimately help policy learning (by allowing to better negotiate the exploration-exploitation trade-off in the early state of the policy learning). Similarly, the goal of general representation is to help to solve unseen tasks, either to evaluate a policy or to learn the optimal policy. Finally, it is interesting to note how exploration is related to generalisation: because pure exploration does not have for goal to maximise a specific extrinsic reward, the same exploration strategy can turn out to be beneficial for learning about different tasks within the same domain. Figure 3.13 illustrates the links between the representational properties identified in

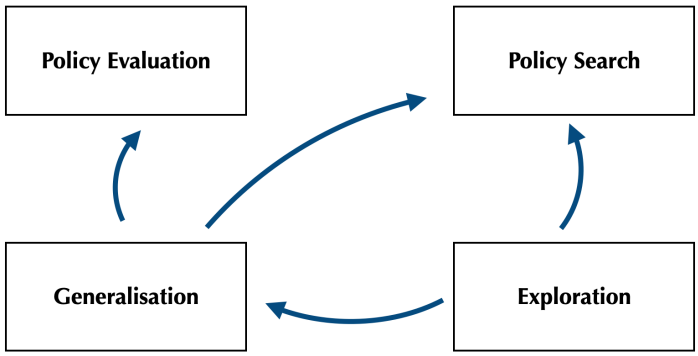


Figure 3.13: Understanding the relationship between the different representational properties. An arrow from a box A to a box B signifies that representations supporting A can also be beneficial for B. For example, a representation designed to benefit exploration can also result in improved generalisation and/or policy search [85].

this work.

While there seem to be a clear trade-off between representations supporting policy evaluation versus policy search (see Figure 3.12), the relation between representational properties bring up an important research question that has not been clearly addressed: "Is there a single representation capable of simultaneously supporting exploration, generalisation and policy search?".

3.5.3 The Effect of Representation on Data-Efficiency

In the context of this thesis, we are particularly interested in the problem of data-efficiency. We therefore give insight into link between data-efficiency in RL and representation learning.

Firstly, theoretical analysis by Du et al. [31] have shown that, in the general case, even a *good representation* (defined as one that allows to linearly approximate the value function with approximation error $\varepsilon = \Omega(\sqrt{H/d})$, where H is the horizon and d is the dimension of the representation, the agent still needs to sample an exponential number of trajectories to find a near-optimal policy, for value-based learning, policy based learning, and model-based learning alike. Moreover, even if an optimal policy can be perfectly predicted by a linear function of the given representation with a strictly positive margin, the agent still requires exponential number of trajectories to find a near-optimal policy.

However, data-efficiency can nonetheless be attained under some assumption. In particular, Jin et al. [62] present LSVI-UCB, an efficient algorithm for linear MDP (where the transition and reward functions are assumed to be linear). It is an optimistic version of Least-Squares Value Iteration (LSVI) [18] based on the Upper Confidence Bound, where a regulariser term based on a given representation $\phi(s) \in \mathbb{R}^d$ encourages exploration. LSVI-UCB's data-efficiency is bounded by d, the dimension of the representation, and not by the size of state space. This result is promising and demonstrates the importance of a good representation: if we possessed a way to learn a representation that linearise a given MDP without drastic projection loss, then efficient reinforcement algorithm exists for that MDP.

Another work introduced a regret bound that is linear in d, the dimension of some latent representation $\phi(s,a) \in \mathbb{R}^d$ of the state action space [133]. It requires assumptions that are less restrictive than the linear MDP assumption; it assumes MDP Lipschitz continuity in the observational space with respect to the value function (two points close to each other in the observational space have similar Q-values), and a Bi-Lipschitz mapping between intrinsic and external metric spaces (if two points are close in the observational space, they are close in the representational space). This interesting finding suggests clearer criteria that a good representation for data efficient RL in unrestricted MDPs must exhibit: low dimensional and Lipschitz continuous with respect to the value function.

As we have seen, there are theoretical efforts that have demonstrated that, under some assumptions and given a low dimensional representation, we can in fact achieve data-efficiency. However, they do not provide insights into how to learn such representations. Therefore, an important direction for future work consists in identifying how to construct, or learn, a representation with the key properties to achieve the aforementioned benefits.

3.6 Conclusion

In this chapter, we brought clarity to the field of representation learning in RL. We identified four main representational properties targeted by existing works in the

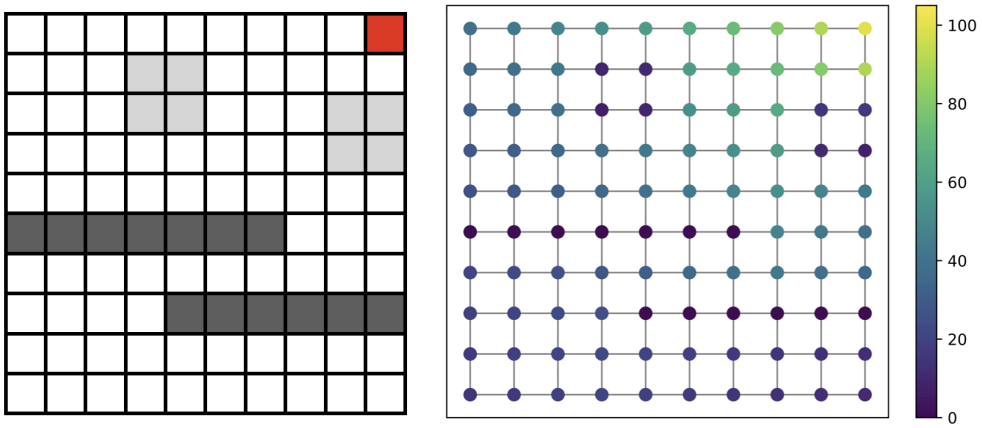
literature: to support policy evaluation, policy search, exploration or generalisation. Accordingly, we presented an overview of the current works, classifying them with respect to their representational properties and learning methods. We carried out critical discussions on the current state of the art, unifying the different representation learning approaches, identifying contradictions and limitation. From these discussions, important questions have emerged and let to ideas that we formulated as interesting future work directions.

Chapter 4

Graph-Based RL

In the context of data-efficiency in policy evaluation or policy search, exact algorithms such as policy iteration [59] or value iteration [104] have space and time complexities polynomial in the size of the state space $|S|$, making them intractable in large state spaces. On the other hand, one could easily project a high-dimensional value function $V \in \mathbb{R}^{|\mathcal{S}|}$ on a low-dimensional basis space $\mathbb{R}^d$ where $d << |\mathcal{S}|$, by selecting a set of random vectors of length d. In this case, the computational cost of constructing the basis is constant, and the cost of solving the MDP will be reduced. However, the resulting solution will most likely be far from the optimal solution due to the randomness of the vectors. There is therefore the need for meaningful basis functions that are also low-dimensional, to achieve a good compromise between accuracy and low computational cost. The problem of MDP dimensionality reduction consists of finding an "optimal" and "compact" representation Φ such that the original MDP can be represented as "accurately" and "efficiently" as possible with respect to Φ. Selecting a representation, or basis functions, requires handling the trade-off between accuracy and efficiency.

Typical linear approximation architectures such as polynomial basis functions (where each basis function is a polynomial term) and radial basis functions (where each basis function is a Gaussian with fixed mean and variance) have been studied to address this issue [73]. These architectures make the assumption that the underlying state space has Euclidiean geometry. However, in realistic scenarios, the MDP's state space is likely to exhibit irregularities. For instance, let's consider



(a) (b)

Figure 4.1: (a) Maze environment. (b) Optimal value function computed using value iteration [104].

the environment depicted in Figure 4.1 that depicts the floor plan of a maze where dark grey squares are strict walls, light grey squares are difficult access room and the read square is the goal room. From the corresponding value function shown in Figure 4.1(b), we observe that neighboring states can have values that are far apart (such as states on opposite sides of a wall). In such cases, basis functions built for Euclidean spaces do not necessarily approximate value functions in a reliable way.

To address this issue, other basis functions have been studied with the aim of preserving the geometry of the state space. Example of such methods include Fourier basis [71], diffusion wavelets, [94], Krylov basis [117] and Bellman Error Basis Function [111, 112]. While these methods take a step in the right direction for attempting to learn rich representations in non-Euclidea domains, they have some important limitations, such as requiring some a priori knowledge of the environment (such as the transition matrix), and being computationally or data hungry.

In this chapter, we propose to go a step beyond by investigating how to automatically learn representations from an approximate graph, inferred with limited data. We investigate graph-based methods for constructing (or learning) efficient basis for solving discrete MDPs. The main contributions of this work can be summarised as follows:

- We present a new line of research that challenges the smooth assumption for the value function, showing that the graph induced by the MDP can lead to suboptimal linear value function approximations under current graph-based representation.

- We propose a novel Generalized Representation Policy Iteration algorithm that allows the use of arbitrary graph-based basis function for value function approximation.

- We identify representation learning on graphs methods that are well suited for linear value function approximation.

- We discuss suggestions for promising future works in the direction of graph-based data-efficient RL.

4.1 Graphs

Many domains such as social networks, recommender systems and biological protein-protein networks underpin complex and irregular structures. These structures can be modelled with graphs, as graphs allow us to store and access relational knowledge about interacting entities. Various real world applications require to make predictions, or discover new patterns within complex domains, therefore the use of graphs becomes essential. In particular, signal processing or machine learning methods can be deployed to infer a sparse representation of relational data to facilitate the manipulation of such complex data. A major challenge is then to find an efficient way to represent, or encode, the graph structure such that downstream tasks can easily exploit these embeddings. In this chapter, we demonstrate how chapter signal processing and machine learning on graphs can help to provide tools for learning sparse representations of data lying on complex relational domains, and benefit reinforcement learning problems.

A graph G consists of a set $\mathcal{V}$ of nodes and a set $\mathcal{E}$ of edges that connect pairs of nodes. G is said to be *weighted* if a non binary value (a *weight*) is assigned to each edge. If all edges have unit cost, the graph is *unweighted*. An *undirected* graph

is a graph in which edges have no orientation; that is, if there is an edge from node i to node j, then there exists an identical edge from node j to node i. Otherwise, the graph is *directed*.

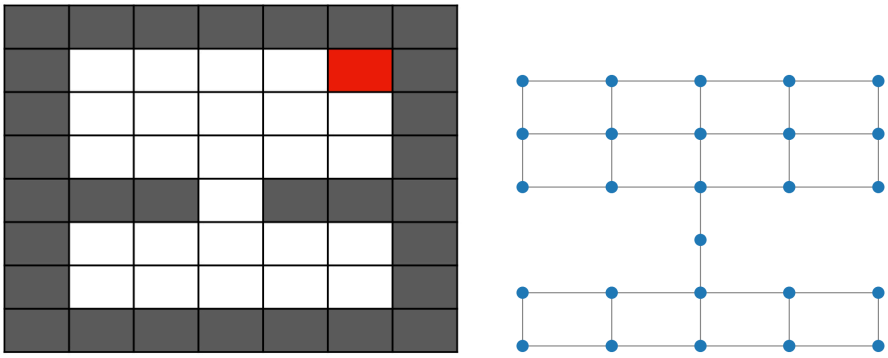
The adjacency matrix A is a square matrix whose elements indicate whether pairs of vertices are adjacent or not in the graph. For an unweighted graph, the entry at the ith column and jth row, denoted $A_{i,j}$ is equal to 1 if there exists an edge connecting nodes i and j, and $A_{i,j} = 0$ otherwise. In the case of a weighted graph, $A_{i,j}$ equals the weight on the edge connecting i to j. If the graph is undirected, then $A_{i,j} = A_{i,j}$, *i.e.* A is *symmetric*. The *degree* of a node i is defined as the number of edges incident to node i. The degree matrix D is a diagonal matrix with diagonal elements $D_{i,i}$ corresponding to the degree of the corresponding node i.

4.1.1 MDPs as Graphs

MDPs can be intuitively represented using graphs, with states being the nodes and the transition probability being the adjacency matrix. For illustration, consider the two room MDP as shown in Figure 4.2(a), which is a traditional RL environment, often used as a case study in tabular RL [27, 141, 95]. This environment is a 2D navigation task, whose state space is discretised such that a state is a specific unique location in the domain, and actions are steps taking you from one location to another. In this specific instance, there are 56 states in total, of which 30 states are inaccessible since they represent interior and exterior walls (dark grey blocks). The remaining 26 states are divided into 1 doorway state and 25 interior room states. The agent can move freely in all for cardinal directions across accessible states and is rewarded by +100 for reaching the last accessible state in the upper right-hand corner of the second room (red block). The unweighted directed graph inferred from this MDP is shown in Figure 4.2(b).

4.1.2 Diffusion on a Graph and Laplacian Eigenmaps

The notion of *diffusion*, widely used in many fields such as physics, chemistry, biology, sociology, and economics, is also useful to describe aspects of MDPs. The concept of diffusion on a graph can be understood as how information will “flow”,



(a) (b)

Figure 4.2: Two-room environment. (a) A two-room environment with 56 total states, divided into 26 accessible states (including one doorway state), and 30 inaccessible states representing exterior and interior walls. (b) The equivalent environment represented as a graph of accessible states. Two states are connected by an edge if the probability of getting from one state to the other in a single step is non zero.

or *diffuse*, from one node in the graph into the rest of the graph. For example, the random walk matrix $P_r = D^{-1}A$ is a diffusion model whose powers determine how quickly a function on the graph will converge to the long term distribution under the random walk [23]. This notion is naturally related to the value function: on a graph induced by the MDP and a given policy, the value function looks like the reward diffusing through the graph. Considering the two-room MDP [95] shown in Figure 4.2(a), this phenomenon appears evident when visualising the true value function of the random policy on the corresponding graph (Figure 4.3). Although it is an informative diffusion operator for analysing MDPs, the matrix P_r is not necessarily symmetric, therefore analysing its spectra can be computationally difficult. Luckily, the *graph Laplacian* is a symmetric matrix and is fundamentally related to the notion of diffusion on a graph, making it a good choice of diffusion operator.

The graph Laplacian is defined as $\mathcal{L} = D - A$. The spectra of the graph Laplacian reveal structural properties of undirected and directed graphs. In particular, the eigenvectors of the graph Laplacian (Laplacian Eigenmaps) are important for constructing low-dimensional embeddings of graphs [23] and for understanding representations for RL [95]. When multiplying a vector v by the Laplacian $\mathcal{L}$, each element of the vector $\mathcal{L}v$ expresses how quickly v would be changing at each node

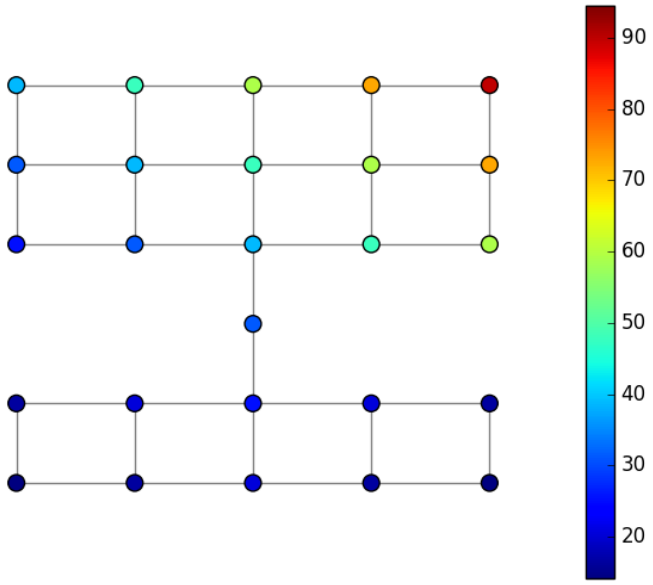


Figure 4.3: True value function on the two-room environment computed using value iteration [104].

under a time step of diffusion. In particular, for a function $f : \mathcal{V} \mapsto \mathbb{R}$ on a graph, the following is easy to show [72]:

$$\mathcal{L}f(i) = \sum_{(i,j)\in\mathcal{E}} (f(i) - f(j)), \tag{4.1}$$

meaning that the Laplacian acts as a *difference operator*.

An additional key property of the graph Laplacian is that projections of value functions on the eigenspace of the Laplacian produce the smoothest global approximation respecting the underlying graph topology. This arises from the fact that the Laplacian Eigenmaps yield a low-dimensional representation of a MDP, generating an orthogonal basis that reflects the nonlinear geometry of the state space. As an analogy to traditional signal processing, the eigenvectors of the graph Laplacian associated with the smaller and largest eigenvalues can respectively be seen as the low and high frequencies of a signal. This is reflected in Figure 4.4 where we show the values of the first few eigenvectors over the tworoom MDP. As it can be seen, the

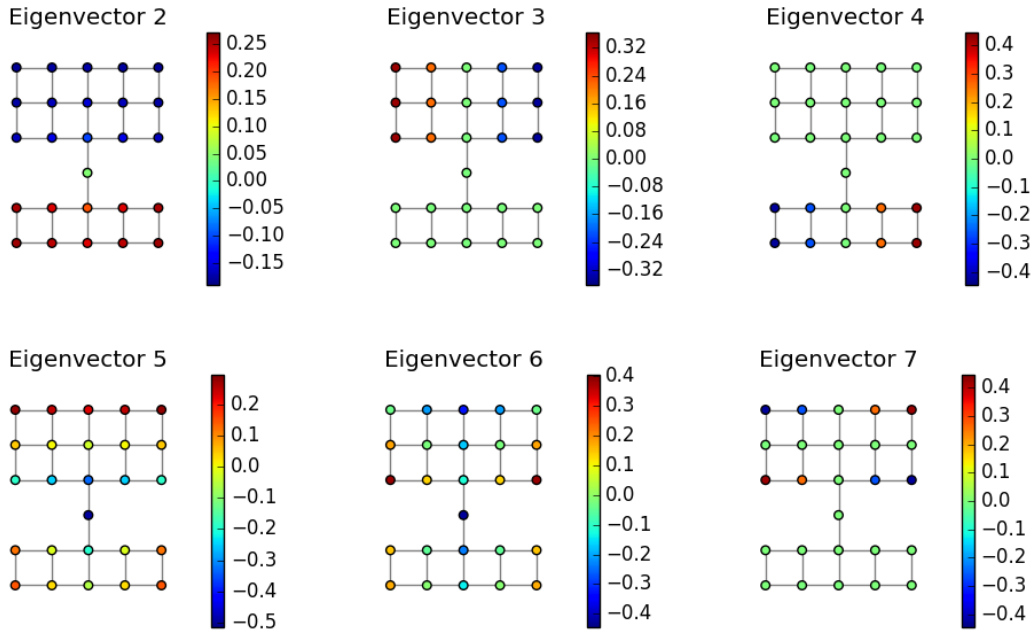


Figure 4.4: Eigenvectors 2 to 7 of the graph Laplacian induced by the tworoom MDP. The first eigenvector, not shown here, is constant across the environment because the Laplacian rows sum to 0.

second eigenvector partitions the environment into the two different rooms, while others capture different structural symmetries in the graph. For example, eigenvector 3 and 4 identify the left and right halves of each room. Eigenvector 5 organises the nodes by the distance to the bottleneck node, and eigenvector 6 captures a cardinal symmetry in the graph.

4.1.3 Representation Learning on Graph

In the majority of real world graph related applications, we are confronted with very large graphs whose adjacency matrices cannot be fully stored in memory and spectral analysis becomes infeasible. As a result, the recent years have seen increasing effort in building systems to *automatically learn to encode graph structure into low-dimensional embeddings*, as opposed to constructing features from the adjacency matrix. Successful applications include node classification [13], node clustering [29] and link prediction [81]. The potential of such graph embeddings techniques remains widely under-explored in reinforcement learning. Consequently, we propose to use node embedding models as basis functions for the value function

approximation in order to automatically learn to encode the graph structure–hence the MDP–into low-dimensional embeddings.

In the following, we identify existing node embedding methods that have proven powerful in various applications and investigate how their performance translates to RL.

4.1.3.1 Node2Vec

Node2vec [45] is an algorithmic framework for learning continuous feature representations of nodes in networks. It is inspired by the powerful language model *Skip-gram* [100] which is based on the hypothesis that words that appear in the same context share semantic meaning. In networks, the same hypothesis can be made for nodes, where the *context* of a node is derived by considering the *nodes that appear in the same random walk on the graph*. Therefore, node2vec learns the node embeddings based on random walk statistics. The key is to optimize the node embeddings so that nodes have similar embeddings if they tend to co-occur on short (biased) random walks over the graph. Moreover, it allows for a flexible definition of random walks by introducing parameters that allow interpolation between walks that are more breadth-first search or depth-first search.

Specifically, for a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{A})$ (where $\mathcal{V}$ is a set of nodes, $\mathcal{E}$ a set of edges and $\mathbf{A}$ the weight matrix) and a set $\mathcal{W}$ of T biased random walks collected under a specific sampling strategy on the graph G, node2vec seeks to maximize the log-probability of observing the network neighbourhood of each node $u \in \mathcal{V}$ conditioned on its features representations, given by f (a matrix of size $|V| \times d$ parameters, where d is the dimension of the feature space):

$$\max_{f} \sum_{w \in \mathcal{W}} \sum_{i=1}^{T} \log \mathbb{P}(N_w(u_i) | f(u_i)), \tag{4.2}$$

where $N_w(u_i)$ describes the neighborhood of the ith node in the walk w. The probability of a the neighbourhood of a node occurring given the node's representation is modeled under the assumption of conditional independence between neighbouring

nodes as follows

$$\mathbb{P}(N_w(u_i)|f(u_i)) = \prod_{u_j \in N_w(u_i)} \mathbb{P}(u_j|f(u_i)) \,. \tag{4.3}$$

The individual probabilities are modeled by a softmax, making the further assumption that a node a neighbouring node have a symmetric effect over each other:

$$\mathbb{P}(u_j|f(u_i)) = \frac{\exp(f(u_j) \cdot f(u_i))}{\sum_{v \in \mathcal{V}} \exp(f(v) \cdot f(u_i))} \,. \tag{4.4}$$

The set of random walks $\mathcal{W}$ is generated by sampling a fixed number of walks of fix length from each node in the graph observing the following sampling strategy: if the random walk just traversed edge (t,v) and now resides at node v, the next step will be made according to the unnormalised transition probability of traversing edge (v,x), defined by $\rho_{xv} = g_{pq}(t,x) \cdot A_{vx}$, with

$$g_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases} \tag{4.5}$$

where d_{tx} denotes the shortest path distance between nodes t and x. The hyperparameters $p > 0$ and $q > 0$ control the distance to the starting node and the tendency to revisit nodes.

4.1.3.2 Struc2Vec

By introducing a bias in the sampling strategy, node2vec allows us to learn representations that do not only focus on optimizing node embeddings so that nearby nodes in the graph have similar embeddings, but also consider representations that capture the structural roles of the nodes, independently of their global location on the graph. Another node embedding approach, *struc2vec*, proposed by [125] addresses the problem of specifically embedding nodes such that their structural roles are preserved. The model generates a series of weighted auxiliary graphs G_k (with

$k = 1, 2, ...$) from the original graph G, where the auxiliary graph G_k captures structural similarities between nodes k-hop neighborhoods. Formally, let $R_k(u_i)$ denote the ordered sequence of degrees of the nodes that are exactly k-hops away from u_i, the edge-weights $w_k(u_i, v_j)$, in the auxiliary graph G_k are recursively represented by the structural distance between nodes u_i and v_j defined as

$$w_k(u_i, v_j) = w_{k-1}(u_i, v_j) + d(R_k(u_i), R_k(u_j)), \tag{4.6}$$

where $w_0(u_i, v_j) = 0$ and $d(R_k(u_i), R_k(u_j))$ is the distance between the ordered degree sequences $R_k(u_i)$ and $R_k(u_j)$ computed via dynamic time warping [125].

Once the weighted auxillary graphs G_k are computed, struc2vec runs biased random walks over them and proceeds as node2vec, optimising the log-probability of observing a network neighborhood based on these random walks.

4.1.3.3 GraphWave

The *GraphWave* algorithm as proposed by [30] takes a different approach to learning structural node embeddings. It learns node representations based on the diffusion of a spectral graph wavelet centered at each node. For a graph G, $\mathcal{L} = D - A$ denotes the graph Laplacian, where A is the adjacency matrix and D is a diagonal matrix, whose entries are row sums of the adjacency matrix. Let U denote the eigenvector decomposition of the graph Laplacian $\mathcal{L} = U\Lambda U^T$ and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{|V|})$ denote the eigenvalues of $\mathcal{L}$. Given a heat kernel $g_s(\lambda) = e^{-s\lambda}$ for a given scale s, GraphWave uses U and g_s to compute a vector $\boldsymbol{\psi}_u$ representing diffusion patterns for node u as follows:

$$\boldsymbol{\psi}_u = U\text{diag}(g_s(\lambda_1), g_s(\lambda_2), \ldots, g_s(\lambda)_{|V|})U^T \delta_u$$

where δ_u is the one-hot indicator vector for node u. Then, the characteristic function for each node's coefficients $\boldsymbol{\psi}_u$ is computed as

$$\phi_u(t) = \frac{1}{|V|} \sum_{m=1}^{|V|} e^{it\Psi_{mu}}$$

Finally, to obtain the structural node embedding $f(u)$ for node u, the paramatric function $\phi_u(t)$ is sampled at d evenly spaced points $t_1,\ldots,t_d$:

$$f(u) = \big[\mathrm{Re}(\phi_u(t_i),\mathrm{Im}(\phi_u(t_i))\big]t_1,\ldots,t_d.$$

4.1.3.4 Variational Graph Auto-Encoder

An alternative is to adopt *Variational Graph Auto-Encoder* (VGAE) to learn graph features. The Variational Graph Auto-Encoder proposed by [69] is a latent variable model for graph-structure data capable of learning interpretable latent representations for undirected graphs. Considering an undirected and unweighted graph $G = (\mathcal{V},\mathcal{E},A)$ with $N = |\mathcal{V}|$ nodes and feature matrix $X \in \mathbb{R}^{N\times F}$ with the ith row representing the feature of node i, the VGAE inference model is given by

$$q(Z|X,A) = \prod_{i=1}^{N} q(z_i|X,A), \tag{4.7}$$

with $q(z_i|X,A) = \mathcal{N}(z_i|\mu_i,\mathrm{diag}(\sigma_i^2))$, wheare μ is the matrix of mean vectors and σ is the variance vector given respectively by $\mu = \mathrm{GCN}_\mu(X,A)$ and $\log\sigma = \mathrm{GCN}_\sigma(X,A)$. The parameters to the Graph Convolutional Neural Networks (GCNs) are denoted by θ_i. The matrix $Z \in \mathbb{R}^{N\times D}$ is the matrix of latent variables whose rows are the vectors z_i. The generative model is given by

$$p(A|Z) = \prod_{i=1}^{N}\prod_{j=1}^{N} p(A_{ij}|z_i.z_j) \tag{4.8}$$

with $p(A_{ij} = 1|z_i,z_j) = \sigma(z_i^T z_j)$, where $\sigma(\cdot)$ denotes the sigmoid function.

The learning phase optimises the following loss function w.r.t to the variational parameters θ_i

$$\mathcal{L} = \mathbb{E}_{q(Z|X,A)}[\log p(A|Z)] - \mathrm{KL}[q(Z|X,A)||p(Z)], \tag{4.9}$$

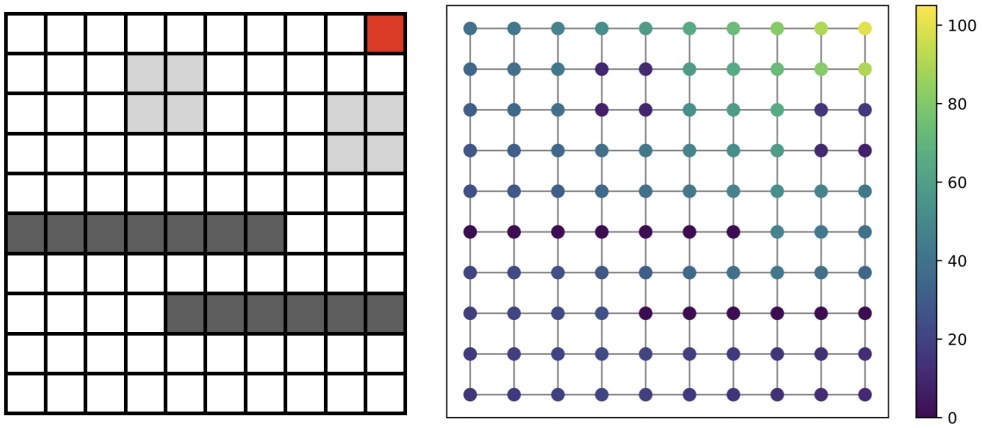
In the following chapter, we describe how we apply these graph features learning methodologies to reinforcement learning strategies.

4.2 Generalized Representation Policy Iteration

Algorithm 1 Representation Policy Iteration

Input:
π_0: sampling strategy,
N: number of random walks to sample,
T: length of each walk,
d: number of proto-value basis functions to use,
ε: convergence condition for parameter estimation algorithm.
Output: ε-optimal policy π
1. Sample Collection Phase
Collect a data set $\mathcal{D}$ of N episodes of successive samples $\{(s_i, a_i, s_{i+1}, r_i)\}$ by following sampling strategy π_0 for maximum T steps (terminating earlier if it results in an absorbing goal state).
2. Representation Learning Phase
Using samples in $\mathcal{D}$, build an undirected graph G by connecting state s_i to s_j with a unit cost edge if the pair (i, j) appear successively. Compute the normalised Laplacian $L = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$. Collect the d smoothest eigenvectors of L to form the basis matrix Φ, a $|S| \times d$ matrix.
3. Control Learning Phase
Using a parameter estimation algorithm such as LSPI or Q-learning, find an ε-optimal policy π that maximizes the action value function $Q^\pi = \boldsymbol{\phi}\theta^\pi$ within the linear span of the basis $\boldsymbol{\phi}$.

The representation policy iteration algorithm (RPI) was introduced in [95] to address the problem of jointly learning compact representations and an optimal policy. It is a three steps algorithm consisting of (1) a sample collection phase, (2) a representation learning phase and (3) a parameter estimation phase. In RPI, the representation learning phase is predefined. Namely, an undirected graph G is built from the available data set $\mathcal{D}$ collected in the initial phase of the algorithm (data collection). Then a diffusion operator, such as the normalised Laplacian L is computed on graph G and the d-dimensional basis functions $\boldsymbol{\phi} = [\phi_1, \ldots, \phi_d]$ are constructed from spectral analysis of the diffusion operator. Specifically, the ϕ_i's are the smoothest eigenvectors of the graph Laplacian and are known as *proto-value functions* (PVFs). The notion of *smoothness* on a graph relates to a function, or signal on the nodes of a graph, whose values between strongly connected nodes do not vary much. Because the graph Laplacian is a diffusion operator, its eigenvector associated with the lowest eigenvalues are smoothest on the graph. The pseudo



(a) (b)

Figure 4.5: (a) Maze environment. The dark grey squares are strict walls, while the light grey square are difficult access rooms. The red square is the goal room. (b) Optimal value function computed using value iteration [104].

code of RPI is presented in Algorithm 1.

The key intuition is that given a state-graph that perfectly represents the MDP, the value function is modelled as a diffusion process over the graph, and therefore it is a smooth function on the graph (see Figure 4.3). Hence, given the spectral properties of the Laplacian operator, PVFs are a good choice of basis functions for preserving the smoothness of the value function. However, it is not guaranteed that we can construct a graph from a limited number of samples such that its derived PVFs reflect the underlying ground truth state space. In fact, we can show that the value function is not as smooth on the estimated graph (constructed from samples) as it is on the ground truth MDP graph where the edges are weighted by the transition probability.

We consider the environment depicted in Figure 4.5, where dark grey squares are strict walls, light grey squares are difficult access room and the read square is the goal room. To construct the estimated graph $\hat{G}$, we first collect samples by running 100 independent episodes starting at a random initial state and taking successive random actions until the goal state or the maximum number of steps has been reached. We then connect temporally consecutive states with a unit cost edge. When the dynamics of the MDP is fully known, the environment can be represented

by a graph G where the states are the nodes and the edges represent actual transition probabilities (i.e. edges between accessible states have weight 1, edges between an accessible state and a wall state have weight 0, and edges between an accessible or difficult access state and a difficult access state have weight 0.2).

In order to study the smoothness of the value function on these two graphs, we use the following function to measure the global smoothness of the value function

$$\sum_{(i,j)\in\mathcal{E}} w_{ij}(V(s_i)-V(s_j))^2 = \boldsymbol{V}^\top \mathcal{L}\boldsymbol{V}\,,$$

where $\mathcal{L}$ is the graph Laplacian and w_{ij} is the weight on the edge connecting state s_i to s_j. In other words, if values $V(s_i)$ and $V(s_j)$ from a smooth function reside on two well connected nodes (i.e. w_{ij} is large), they are expected to have a small distance $(v_i - v_j)^2$, hence $\boldsymbol{V}^\top \mathcal{L}\boldsymbol{V}$ is small overall. In Table 4.1, we provide the smoothness factor measured for both the learned and ground truth graph. The analysis clearly shows a reduction of the value function smoothness when going from the ideal weighted graph to the estimated and unweighted graph (usually considered in realistic settings, when the transition probability is not known a priori)

	$\boldsymbol{V}^\top \mathcal{L}\boldsymbol{V}$
Estimated graph $\hat{G}$	14831.72
Ground truth graph G	5705.65

Table 4.1: Analysis of the smoothness of the value function on different graphs.

It follows that the lowest eigenvectors of the graph Laplacian will not be an ideal basis function for approximating the signal on the graph (*i.e.* ., the value function) since the signal is not as smooth on the estimated graph $\hat{G}$ as it is on the ground truth weighed graph G.

As results, it is expected that the smoothest PVFs of the estimated graph $\hat{G}$ on which the value function is less smooth, will not allow to reconstruct the true value function as well as the smoothest PVFs of the ideal graph. This phenomenon is verified in Figure 4.6, where we show in both cases the mean squared error (MSE) of the linear approximate value function computed in a least-square way using the true

value function (computed via value iteration [104]) for the environment shown in Figure 4.5. The figure clearly shows the loss in accuracy due to the non ideal support $\hat{G}$ on which the smoothness assumption does not hold. This motivates us to depart from constructing representation, and to instead resort to representation learning on graphs frameworks that still exploit the key structural information captured in $\hat{G}$.

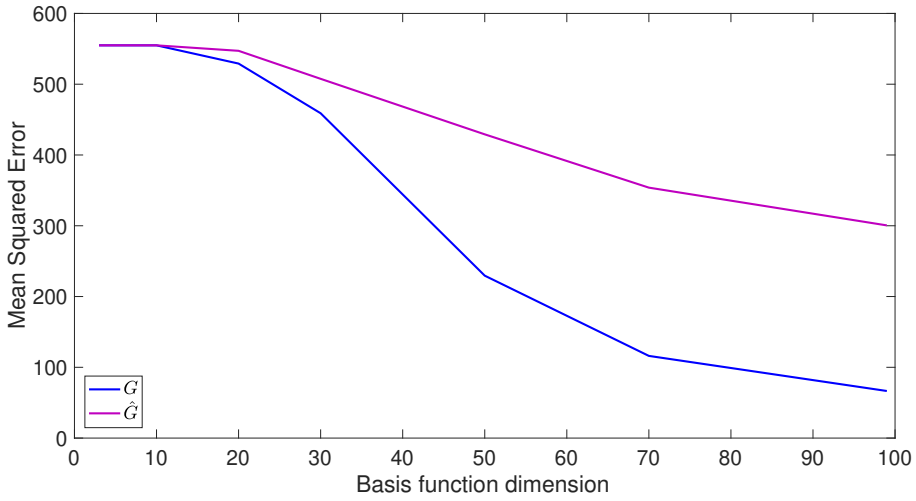


Figure 4.6: MSE between the approximate value function and the true value function when using PVFs of the ground truth graph G (**blue**) and the estimated graph $\hat{G}$ (**pink**) as basis functions. On the x axis we make the dimension of the basis function (the number of PVFs) vary.

4.3 Framework

To overcome this limitation, we propose to generalize RPI to allow different representation learning methods. In particular, we first observe that state space topologies of MDPs can be intuitively modeled as (un)directed graphs, with the nodes being the states and the transition probability matrix being the similarity matrix. When the transition probabilities are unknown, we can construct a graph from collected samples by connecting temporally consecutive states with a unit cost edge. Therefore, similarly to [93], we propose to construct the graph from collected samples of an agent acting in the environment given by the MDP. We then use the node embedding methods described in Section 4.1.3 to automatically learn representations on the graph induced by the MDP that preserves the underlying geometry of the state space. Finally, we use the learned representations to linearly approximate the

value function. We call this algorithm *Generalized Representation Policy Iteration (GRPI)* as it generalises RPI by allowing *any* representation method. It is described in Algorithm 1.

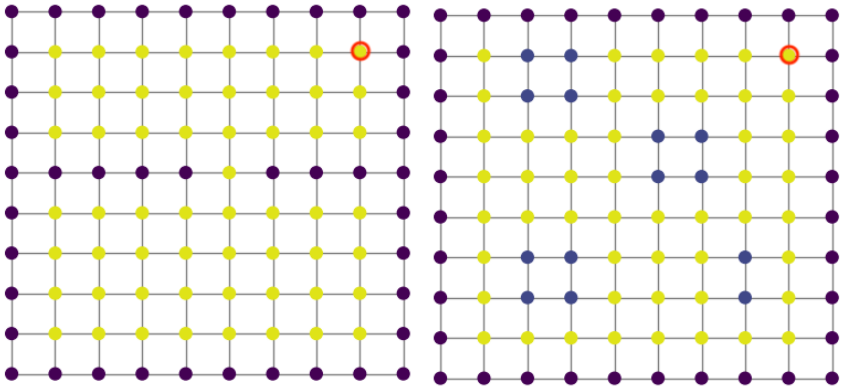
Algorithm 1 General Representation Policy Iteration

Input:
π_0: sampling strategy,
N: number of random walks to sample,
T: length of each walk,
d: dimension of the basis functions,
$embed()$: representation learning method,
ε: convergence condition for LSPI.
Output: ε-optimal policy π
1. Sample Collection Phase
Collect a data set $\mathcal{D}$ of T successive samples $\{(s_i, a_i, s_{i+1}, r_i), (s_{i+1}, a_{i+1}, s_{i+2}, r_{i+1}), \ldots\}$ by following sampling strategy π_0 for maximum T steps (terminating earlier if it results in an absorbing goal state).
2. Representation Learning Phase
Build basis function matrix $\boldsymbol{\phi} = embed(\mathcal{D}, d)$.
3. Control Learning Phase
Using a parameter estimation algorithm such as LSPI or Q-learning, find an ε-optimal policy π that maximizes the action value function $Q^\pi = \boldsymbol{\phi}\theta^\pi$ within the linear span of the basis $\boldsymbol{\phi}$.

4.3.1 Experiments

We consider the 10×10 two-room environment used in [95], shown in Figure 4.7(a). It consists of 100 states in total, divided into 57 accessible states and 43 inaccessible states representing walls. There is one goal state, marked in red and the agent is rewarded by $+100$ for reaching the goal state.

We also consider the obstacles-room environment depicted in Figure 4.7(b). In this environment, there are 100 states in total, some of which are inaccessible since they represent exterior walls and 14 of which are accessible from neighbouring states with a fixed probability of 0.2 (they represent a moving obstacle or difficult access space). All the other states are reachable with probability 0.9. The agent is rewarded by $+100$ for reaching the state located at the upper-right corner. We construct the corresponding graphs where each location is a node, and the transi-



(a) (b)

Figure 4.7: Two different maze environments. The purple nodes represent strict walls, while the blue nodes are difficult access rooms. All other nodes represent accessible rooms The node shown in red is the goal room.

tions (4 possible actions: left, right, up and down) are represented by the edges. We run and evaluate the General Representation Policy Iteration (GRPI) algorithm using embedding methods from Section 4.1.3 to compute the basis functions in the second phase of the algorithm.

Concretely, we execute the following steps:

1. We first collect a set $\mathcal{D}$ of 100 sampled random walks, each of length 100 (or terminating early when the goal state was reached). The number and length of random walks collected is empirically chosen to construct the smallest data set under which GRPI with proto-value-functions of dimension 70 (the baseline) can solve the two-room environment. The sampling dynamic is as follows: starting from a random accessible sate, the agent takes one of the four possible actions (move up, down, left or right). If a movement is possible, it succeeds with probability 0.9. Otherwise, the agent remains in the same state. If the agent reaches the gold state, it receives a reward of 100, and is randomly reset to an accessible interior state. We use off-policy sampling ($\pi_0 =$ random policy) to collect the samples, except in the case of node2vec, where we follow node2vec sampling strategy and generate samples under a biased random walk defined by (4.5). We use grid search to find the optimal hyperparameters $p = 1$ and $q = 4$ that guide the walk according to [45].

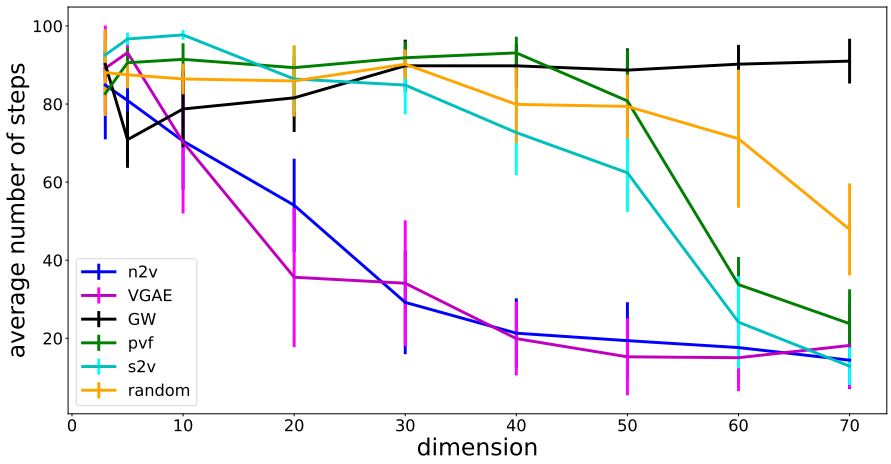
2. Following Mahadevan and Maggioni [95], we then use sample transitions in $\mathcal{D}$ to build an undirected graph where the weight matrix W is the adjacency matrix and run $embed(\mathcal{D}, d)$ with $embed \in \{$Node2vec (n2v), struc2vec (s2v), Variational Graph Auto-encoder (VGAE), GraphWave (GW)$\}$ for diffenrent choices of d. In the case of node2vec, we reuse the samples set to derive the node neighbourhoods used in the objective function.

3. We then learn the parameters of the linear value approximation from the set of samples $\mathcal{D}$ using the parameter estimation method LSPI for its demonstrated sample-efficiency [73].

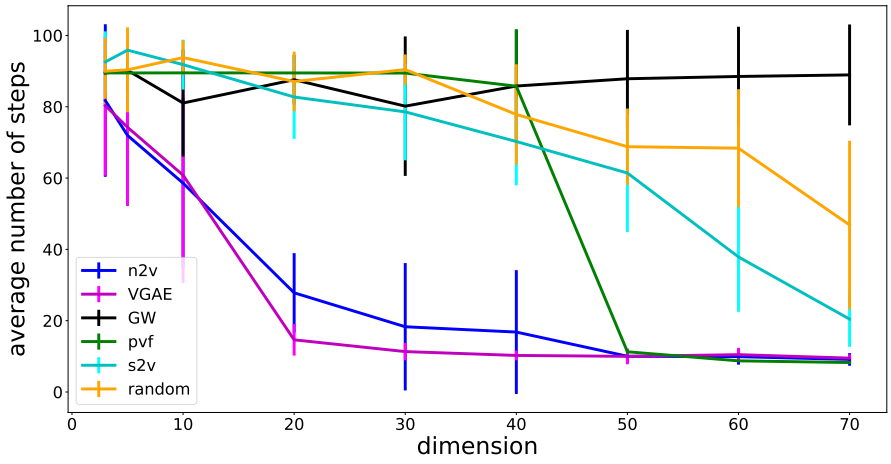
4. Finally, to evaluate the performances of the different representations, we use the policies learned by GRPI for each representation learning method to run simulations starting from each accessible states. We compare the performance of each representation learning method in terms of the average number of steps required to reach the goal. We also compare to two baselines: the traditional PVF basis functions, and random vectors. The results for the two environments, averaged over 20 independent runs, are shown in Figures 4.8(a) and 4.8(b). Each run consists of episodes of a maximum of 100 steps, where each episode is terminated earlier if the agent reached the goal state.

4.3.2 Discussion

Figures 4.8(a) and 4.8(b) show the average number of steps to reach the goal as a function of the dimension of the basis function. We first observe that the policy learned via the GraphWave basis function lead to very poor performances regardless of the dimension size. We investigate this phenomenon by looking at the approximate value function learned under these basis. The approximate state values are depicted in Figrue 4.9. Because GraphWave learns embeddings that are exclusively structural by design, we hypothesise that they fail at capturing global network properties. In fact, the embeddings learned by GraphWave for the corner states in the two-room environment are equals, making it obviously impossible to learn different state values with linear approximation. This suggests that although the GraphWave



(a)



(b)

Figure 4.8: Average number of steps required to reach the goal steps using the various basis functions. On the x axis we make the dimension of the basis functions vary. 4.8(a) corresponds to the two-room environment, while 4.8(b) corresponds to the obstacle-room environment.

is a powerful model for capturing structural information in networks, it is not a good choice of basis function for approximating value function on a graph.

On the other hand, we notice that although struc2vec was also designed to capture structural similarities between nodes, it also preserves the local properties of the graph by considering neighborhoods of different sizes [125]. Hence, struc2vec is able to accurately approximate the value function even in graphs that have sym-

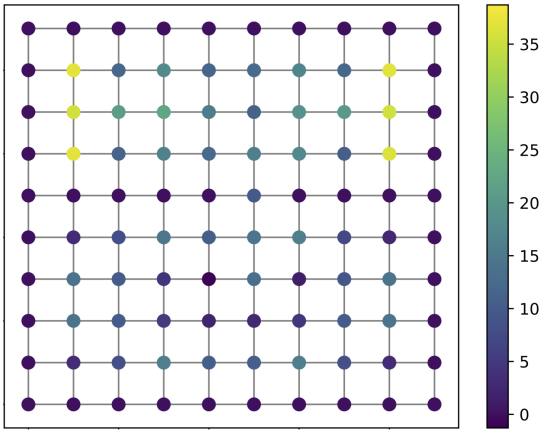


Figure 4.9: Approximate value function via GRPI using GraphWave basis function of dimension 70 on the two-room environment. Because GraphWave learns embeddings that are exclusively geometrically structural by design and fail at capturing global network properties, GRPI with GraphWave learns similar state values for states that are geometrically similar (*e.g.* corner sates), leading to large value approximation error in the two-room environment where there is a reward in one corner state only.

metrical structure such as the two-room environment.

Finally, the result show that VGAE and node2vec are good choices of basis functions for approximating the value function in low dimension. Indeed, they lead to good performances in terms of number of steps to reach the goal states with basis functions of dimension as low as 20 for VGAE and 30 for node2vec. On the contrary, we observe that the PVFs require dimension of at least 70 to reach comparable performances on the two-room domain and dimension of 50 on the obstacle-room domain. The random baseline does not allow for consistently accurate enough value approximation to solve neither the two-room or obstacle-room environment with vectors of dimension smaller or equal to 70.

We observed that the sampling strategy used in node2vec has a important impact on the performance of the learned policy. Using grid search, we find that the optimal values for the hyperparameters p and q that guide the random walks (see Equation (4.5)) are 1 and 4 respectively. We show the performances of node2vec with selected values of p and q in Figure 4.10. When $p < q$ and $q > 1$, the strategy is biased to encourage walks to backtrack a step and to visit nodes that are close to the

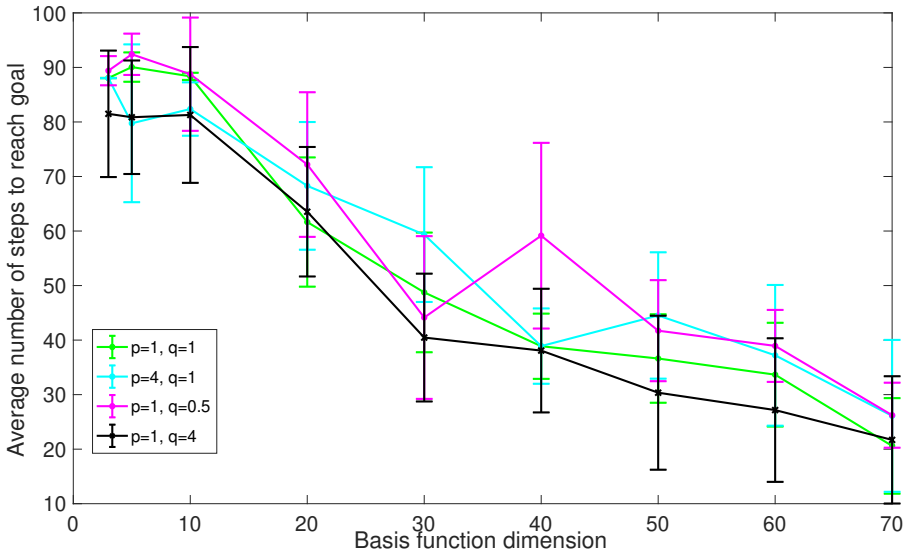


Figure 4.10: Average number of steps required to reach the goal steps using node2vec with varying parameters p and q. On the x axis we make the dimension of the basis functions vary.

current node in the walk. Therefore, it leads to walks that approximate a breadth-first search behavior, gathering a local view of the underlying graph with respect to the starting node. On the other hand, when $p > q$ and $q < 1$, the walk approximate a depth-first search behavior and lead to more outward exploration. Grover and Leskovec [45] have shown that the former type of sampling strategy allows to reflect structural equivalences of nodes whereas the second type allows to capture homophily within the network. Our results depicted in Figure 4.10 suggests that for maximising return in grid domains, structural equivalence plays a more important role than homophily.

4.3.3 Additional Results

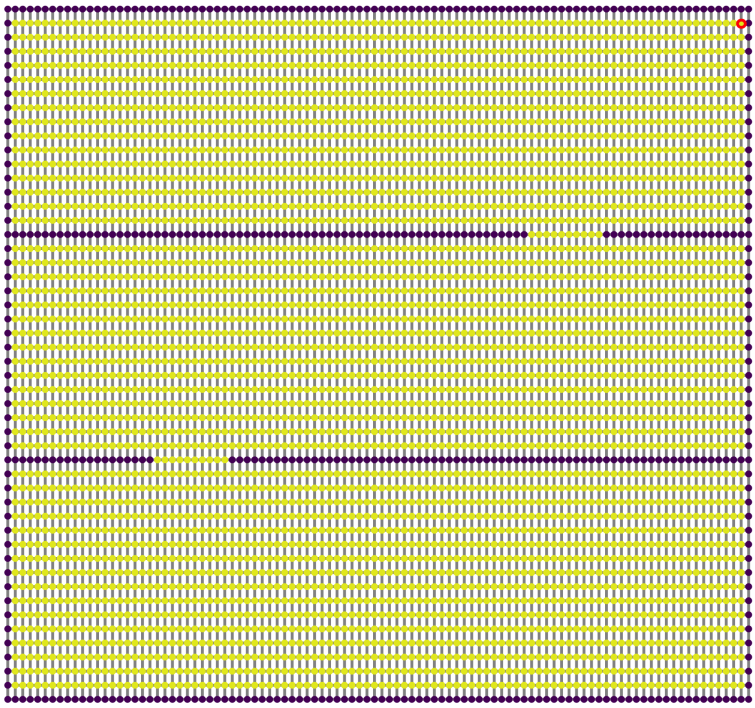
In order to investigate whether we can expect a similar behaviour in larger environments, we consider a 100×50 three-room environment. This environment is similar to the two-room environment but with two interior walls, with the upper wall having the opening more on the right and the lower wall having the opening more on the left, as shown in Figure 4.11(a).

We construct the graph from 500 collected samples of length at most 100 and

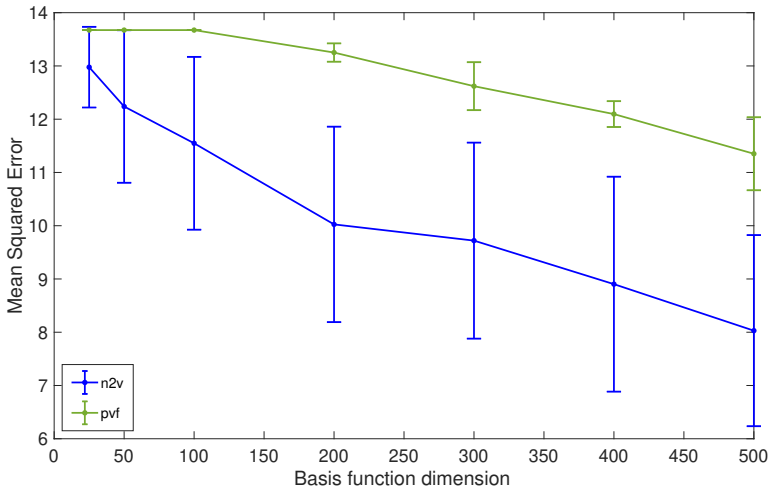
derive the PVFs and the node2vec embeddings. For each of these basis functions, we solve the linear approximation problem in the least-square sense by minimizing the following loss function with respect to the parameter $\boldsymbol{\theta}$ using the optimal value function computed via value iteration [104]:

$$L(\theta) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \Big(V(s) - \sum_{i=1}^{d} \theta_i \phi_i(s) \Big)^2. \tag{4.10}$$

Figure 4.11(b) shows the gain of adopting node2vec feature learning in reinforcement learning in high dimensional state space, suggesting that node2vec is able to capture rich information about the MDP in compact basis vectors, even in larger state spaces.



(a)



(b)

Figure 4.11: (a) Three-room environment. The purple nodes represent strict walls, while the blue nodes are difficult access rooms. All other nodes represent accessible rooms The node shown in red is the goal room. (b) Mean squared error of value function approximation. On the x axis we make the dimension of the basis functions vary.

4.3.4 Conclusion and Future Work

In this work, we have generalised the representation policy iteration algorithm to include a representation learning phase that allows to use any representation learning method for computing the basis functions in the linear value approximation. We investigated several representation learning on graphs mechanisms for learning high quality node embeddings that preserve the geometry (global and/or local) of the graph induced by the Markov decision process. We compared the performance of several representation learning methods with respect to value function approximation error and total return. Finally, we observe that models that are designed to capture the global structural geometry of the graph while preserving local properties do well at approximating the value function in low feature space dimensions, outperforming the commonly considered PVFs for this task.

The main findings of this work can be summarised as follows:

- Evidence that the smoothness assumption of the value function on an estimated unweighted graph does not necessarily hold, which leads to poor value function approximation when using smooth basis functions such as PVFs.
- Using basis functions that automatically learn to embed the geometry of the graph induced by the MPD can lead to improved performance over the PVFs.
- Such embedding methods need to capture the structural equivalence of the nodes while preserving the local properties of the graph.
- Under sampling strategies satisfying the requirements of the previous point, node2vec [45] outperforms the commonly used PVFs.
- The Variational Graph Auto-Encoder, which is a more complex system than node2vec and requires more training, leads to minor performance improvement compared to node2vec.

These findings encourage the further study of representation learning on graphs for achieving efficient and accurate policy learning for reinforcement learning. We highlight in the following relevant open questions for future work.

- Through this work, node2vec emerged as a promising and efficient representation learning algorithm for discovering rich basis function for approximating the value function. Certainly, it is therefore of great interest to conduct further experimental and theoretical analysis of the embeddings node2vec yields. Such a study will help to better understand to what extend node2vec could be useful for RL, and how it relates to other graph-based and non graph-based representations.

- Phase (1) of GRPI requires collecting data in order to construct a graph. This involves storing the entire graph in memory, which is not feasible in very large state spaces. This motivates further research on graph-inspired representation learning systems that could capture the same structural information about the MDP without necessitating a memory complexity that is linear in the state space size.

- Another limitation emerges from GRPI. Indeed, in phase (2), learning a state representation for all states requires having a graph with a node for all states, meaning that in phase (1), it is assumed that all states were explored at least once. This is an unpractical assumption to make when exploration is often limited. As a result, strategies for generating representations for states that have not been seen during exploration should be investigated.

- The graph representation learning mechanisms explored in this work were not initially designed specifically for RL. The fact that generic graph representation learning algorithms like node2vec and VGAE have proven to be useful in RL is a good motivation for designing similar systems with RL in mind. For example, by considering other aspects of the MDP in addition to the transition matrix, such as the discount factor or the reward function.

- This study focused on tabular environments, where all states are uniquely identifiable. It remains unclear how the representation learning phase would handle the case of environments with partially observable states, where the agent receive states observations in the form of features (*e.g.* images). Investi-

gating graph-based representation learning algorithm for partially observable MDPs would make an interesting future research direction.

In the next chapter, we will address the first to points. We start by conducting further analysis of node2vec, then, we derive a new graph-inspired representation learning algorithm, that does not require storing a graph in memory.

Chapter 5

Designing Graph-Inspired Representation for Data-Efficient RL

As seen in Chapter 4, graph-based representations can help to improve data-efficiency in value function approximation, by capturing the geometry of the underlying state space in compact low dimensional vectors. In particular, node2vec [45] stood out as an efficient method that outperforms the state-of-the-art graph-based representation PVFs [95] under limited data constraints. As a result, in this chapter, in an effort to understand the potential and limitation of node2vec in RL, we take a closer look at how node2vec performs in different types of environments, we study the effect of the key hyperparameters and we study the objective function of node2vec. From the theoretical analysis emerges an interesting relationship with PVFs, a relationship that is also reflected when we analyse visualisations of node2vec representations with PVFs.

We then build on the node2vec system to develop *state2vec*, an efficient yet reliable framework for learning representations that are explicitly designed for RL. We are interested in learning features that capture the underlying geometry of the state space. In particular, we seek the following properties for the features: (1) to be learned from data rather than handcrafted—to avoid structural bias (see Chapter 4, Section 4.2), (2) to have a computationally efficient learning process to scale to large state spaces (3) to be low-dimensional—to ensure a fast adaptation when used in different tasks within the same domain (*i.e.* MDPs with shared state and action

spaces), and (4) to be geometry-aware rather than task-aware—to generalize across optimal policies. To learn such features, we extend the well known node2vec algorithm [45] to infer graph embeddings capturing temporal dependencies. In other words, state2vec encodes states in low-dimensional embeddings, defining the similarity of states based on the discounted future transitions. Moreover, we impose that the data used for training is fully exploratory and independent of any specific task so that it remains reward agnostic. This allows us to use the same representation without any retraining of the features to solve tasks with varying reward functions. To solve a specific task, the agent will need to simply learn a task-aware coefficient vector to derive a value function approximation. The dimensionality of the coefficient vector is imposed by the embedding dimension, which we constraint to be low to favor sample-efficiency. We show experimentally that state2vec captures with high accuracy the structural geometry of the environment while remaining reward agnostic. The experiments also support the intuition that off-policy state2vec representations are robust low dimensional basis functions that allow to approximate well the value function.

5.1 A Closer Look at Node2vec

A key aspect of node2vec resides in the ability to control the type of geometrical information to be captured within the graph by inserting a bias in the sampling strategy used to define the notion of neighbourhood on the graph. Indeed, node2vec guides the random walks on the graph by taking a next step according to an unnormalised transition probability which is derived using the adjencency matrix A. Specifically, the probability of transitioning from state v to connected state x, when having last visited state t is defined by $\rho_{xv} = g_{pq}(t,x) \cdot A_{vx}$, with

$$g_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases} \tag{5.1}$$

where d_{tx} denotes the shortest path distance between nodes t and x. Effectively, p acts as a *return parameter*, controlling the likelihood of immediately revisiting a node in the walk. When p is chosen to by high ($> \max(q, 1)$), the walk is less likely to visit an already visited node in the following two steps (except when there are no other neighbour available). This strategy results in moderate exploration that prevents 2-hop redundancy. However, when p is set to a low value ($< \min(q, 1)$), the walk is encouraged to stay local, backtracking more often. The parameter q on the other hand acts as a *in-out parameter*, controlling a far away from the current node the walk should go. When q is large, (> 1), the walk will tend to stay closer to starting node, favouring local exploration, resulting in a strategy that resembles breath-first search (BFS). On the contrary, when $q < 1$, the walk will distance itself from the current node, leading to a search that is akin to depth-first search (DFS). In a graph containing several clusters of highly interconnected nodes, Grover and Leskovec [45] show that when sampling neighbourhoods using a BFS strategy, the embeddings learned by node2vec capture structural equivalence, *i.e.* nodes that have similar structural roles (such as acting as community hubs) are close in the embedding space. On the other hand, when the neighbourhoods are sampled using a DFS strategy, the homophily is captured in the embedding space, such that nodes within the same interconnected cluster have similar embeddings.

In Chapter 4, we have found that a sampling strategy that encourages local exploration and reflects BFS behaviour was most favourable for learning state embedding or value function approximation in the two-room environment. In the following, we investigate whether this finding is consistent across environments with different structures than the grid like structure of the two-room environment. To this end, we design three different graphs with different topologies, we run node2vec on each of these graphs using varying values for the parameter q, and we study the performance of each embeddings in terms of value function approximation error. The first environment (Figure 5.1(a)) is the four-room environment, which includes 169 states separated into four rooms connected by doorways, and there is a single reward in the corner of one of the rooms. The second environment (Figure 5.1(c))

is a low-strech spanning tree consisting of 64 nodes, where the degree of each node is in $\{1,2,3\}$, with a reward at the center of the graph. The last environment (Figure 5.1(e)) is a 1-dimensional torus of 80 nodes with a single rewarding node. The rewarding node in all the environment generates a reward of +1, all other nodes have a reward of 0, and the discount factor is $\gamma = 0.9$. Figures 5.1(a), 5.1(c) and 5.1(e) show the true value function under the uniform policy as a signal on the graph corresponding to each environment.

On each environment, we run node2vec with a dimensionality $d = |\mathcal{S}|/2$, walk length of 10, a window size of 3, and a fixed value for the return parameter $p = 1$. We experiment with varying values of q, biasing the walks from adopting a DFS behaviour to a more BFS behaviour. We then learn the parameters $\boldsymbol{\theta}$ that best approximate the true value function using the node2vec state representations as basis functions. We then evaluate the performance of the representations computing the Mean Squared Error (MSE)

$$\frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}}\left(V(s)-\boldsymbol{\phi}(s)\boldsymbol{\theta}^{\top}\right)^2 \tag{5.2}$$

for varying values of the in-out parameter q. Figures 5.1(b), 5.1(d) and 5.1(d) depict the average MSEs over 10 independent runs. Interestingly, it appears that a BFS like sampling strategy is overall better for approximating value functions (the MSE decreases as q increases). When the graph is not strongly connected, *i.e.* when the expected shortest path between any two nodes is relatively large compared to the size of the state space (like in the low-stretch tree domain or in the torus), the results suggest that the type of walk does not have a strong impact on the performance (no significant improvement from going to $q = 0.5$ to $q = 4$). On the other hand, in a grid like environment, it is clearly better to favour locality over depth when sampling the random walks.

To better understand whether the embedding learned by node2vec truly captures the geometry of the state space, we visualise the 2D PCA component of the best node2vec embeddings for each environment (that is, with the hyperparameters optimised to obtain the lowest approximation error). As seen in Figure 5.2,

node2vec clearly captures the structural notion of a room in the four room domain, and the concept of a chain in the 1-dimensional torus. Additionally, the notion of a tree seems to persist in the node2vec embedding space of the low-strech tree.

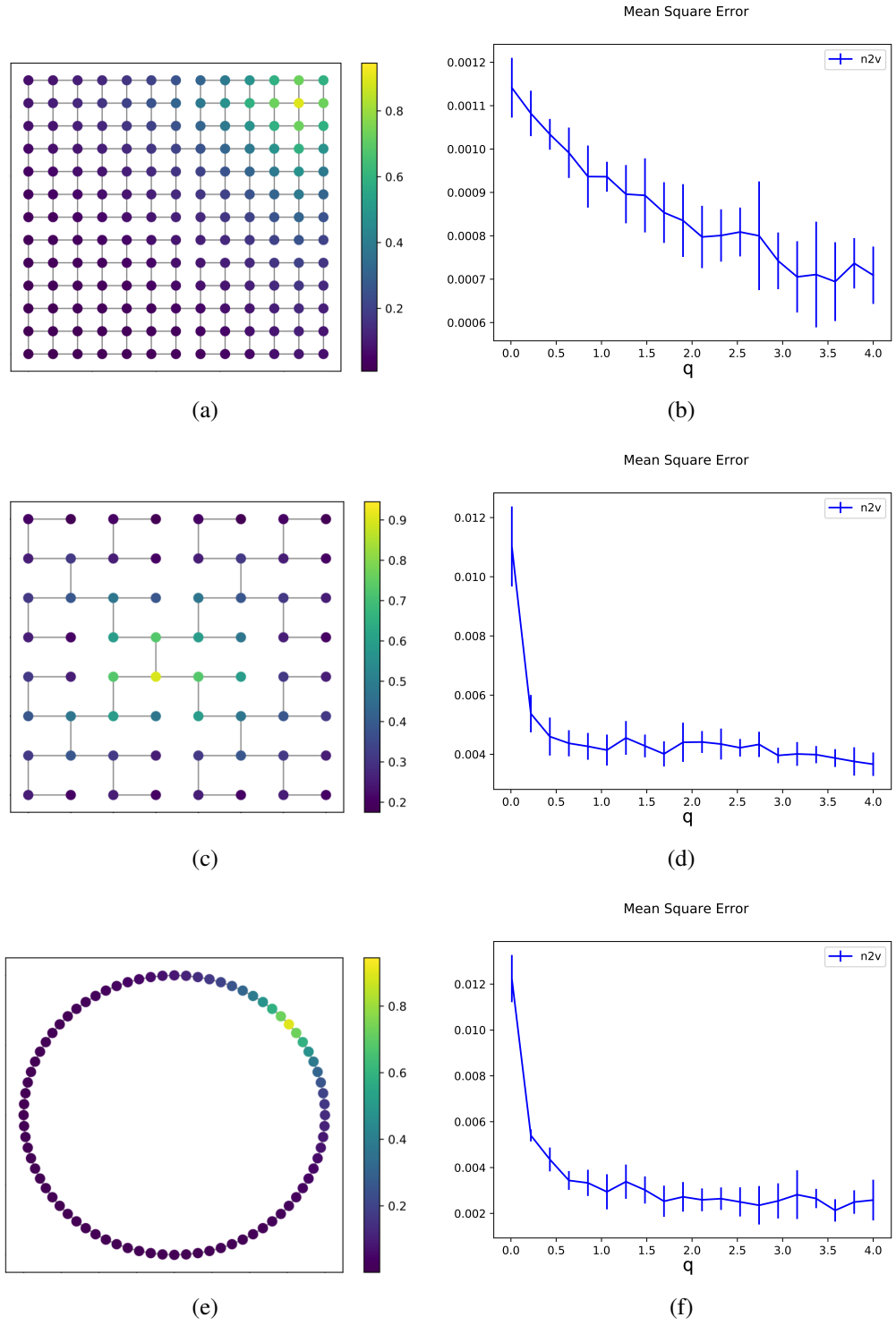


Figure 5.1: Different MDP configurations with the value function shown with colours (left) and corresponding MSEs with respect to the true value function when using node2vec with varying values of the in-out walk bias parameter q (right).

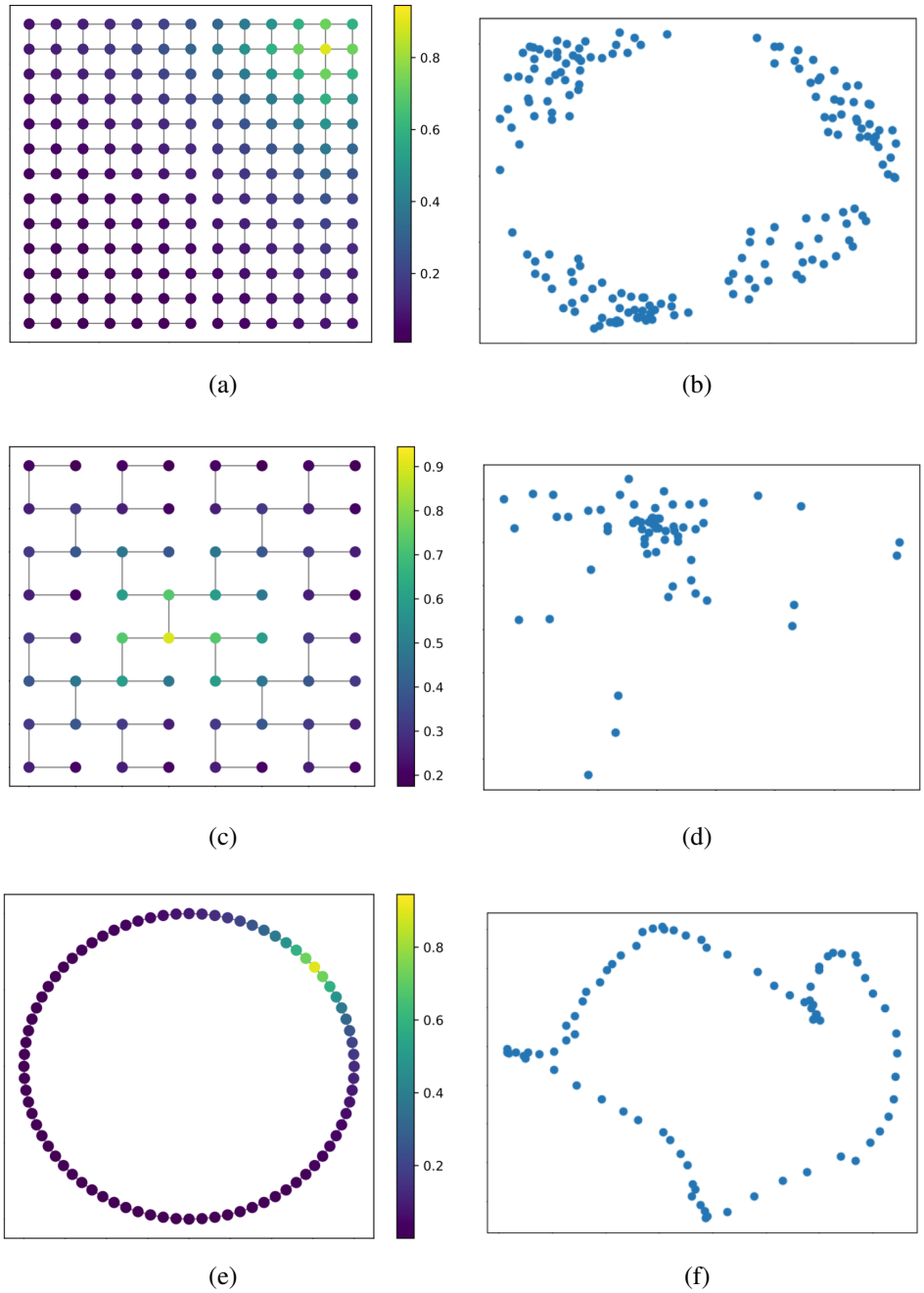


Figure 5.2: Different MDP configurations with the value function shown with colours (left) and their corresponding node2vec state embeddings projected into 2D PCA space.

5.2 Node2vec and PVFs

We now take a closer look at the objective function of node2vec to gain a better understanding of how this representation compares to existing graph-based representations used in RL.

We recall that node2vec maximises the log-probability of observing the network neighbourhood of each node $u \in \mathcal{V}$ conditioned on its features representations, given by $\mathbf{\Phi}$ (a matrix of size $|V| \times d$ parameters, where d is the dimension of the feature space). Specifically, for a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{A})$ (where $\mathcal{V}$ is a set of nodes, $\mathcal{E}$ a set of edges and $\mathbf{A}$ the weighted adjacency matrix) and a set $\mathcal{W}_\rho$ of T biased random walks collected under sampling strategy ρ on the graph G,

$$\begin{aligned}
&\max_{\boldsymbol{\phi}} \sum_{w \in \mathcal{W}_\rho} \sum_{i=1}^{T} \log \prod_{u_j \in N_w(u_i)} \frac{\exp(\boldsymbol{\phi}(u_j) \cdot \boldsymbol{\phi}(u_i))}{\sum_{u_k \in \mathcal{V}} \exp(\boldsymbol{\phi}(u_k) \cdot \boldsymbol{\phi}(u_i))} \\
=&\max_{\boldsymbol{\phi}} \sum_{w \in \mathcal{W}_\rho} \sum_{i=1}^{T} \sum_{u_j \in N_w(u_i)} \log \frac{\exp(\boldsymbol{\phi}(u_j) \cdot \boldsymbol{\phi}(u_i))}{\sum_{u_k \in \mathcal{V}} \exp(\boldsymbol{\phi}(u_k) \cdot \boldsymbol{\phi}(u_i))} \\
=&\max_{\boldsymbol{\phi}} \sum_{w \in \mathcal{W}_\rho} \sum_{i=1}^{T} \Big(\sum_{u_j \in N_w(u_i)} \boldsymbol{\phi}(u_j) \cdot \boldsymbol{\phi}(u_i) - \log(\sum_{u_k \in \mathcal{V}} \exp(\boldsymbol{\phi}(u_k) \cdot \boldsymbol{\phi}(u_i))) \Big)
\end{aligned} \tag{5.3}$$

When the window size is set to be 1 (that is, neighbours are always directly connected), we collect an infinitely long walk starting from each node ($T = |\mathcal{V}|$), and the walk strategy is uniformly random (equal chance of reaching any directly connected node), the objective in (5.3) can be written as

$$\max_{\boldsymbol{\phi}} \sum_{u_i \in \mathcal{V}} \sum_{u_j \in \mathcal{E}(i)} \Big(\boldsymbol{\phi}(u_j) \cdot \boldsymbol{\phi}(u_i) - \log \sum_{u_k \in \mathcal{V}} \exp(\boldsymbol{\phi}(u_k) \cdot \boldsymbol{\phi}(u_i)) \Big), \tag{5.4}$$

where $\mathcal{E}(i) \subseteq \mathcal{V}$ denotes the subset of nodes reachable in one hop from node u_i. If we further enforce that the state representations have unit norm (*i.e.* $|\boldsymbol{\phi}(u)| = 1$ for all $u \in \mathcal{V}$), we have that

$$\boldsymbol{\phi}(u_j) \cdot \boldsymbol{\phi}(u_i) = 1 - \frac{1}{2} ||\boldsymbol{\phi}(u_j) - \boldsymbol{\phi}(u_i)||_2^2$$

and (5.4) becomes

$$\begin{aligned}
&\max_{\boldsymbol{\phi}} \sum_{u_i \in \mathcal{V}} \sum_{u_j \in \mathcal{E}(i)} \Big(1 - \frac{1}{2}||\boldsymbol{\phi}(u_j) - \boldsymbol{\phi}(u_i)||_2^2 - \log \sum_{u_k \in \mathcal{V}} \exp(1 - \frac{1}{2}||\boldsymbol{\phi}(u_k) - \boldsymbol{\phi}(u_i)||_2^2)\Big) \\
=&\max_{\boldsymbol{\phi}} \Big(- \sum_{(i,j) \in \mathcal{E}} (\frac{1}{2}||\boldsymbol{\phi}(u_j) - \boldsymbol{\phi}(u_i)||_2^2) - \sum_{(i,j) \in \mathcal{E}} \log \sum_{u_k \in \mathcal{V}} \exp(1 - \frac{1}{2}||\boldsymbol{\phi}(u_k) - \boldsymbol{\phi}(u_i)||_2^2)\Big) \\
=&\min_{\boldsymbol{\phi}} \Big(\sum_{(i,j) \in \mathcal{E}} (\frac{1}{2}||\boldsymbol{\phi}(u_j) - \boldsymbol{\phi}(u_i)||_2^2) + \sum_{(i,j) \in \mathcal{E}} \log \sum_{u_k \in \mathcal{V}} \exp(1 - \frac{1}{2}||\boldsymbol{\phi}(u_k) - \boldsymbol{\phi}(u_i)||_2^2)\Big)
\end{aligned} \tag{5.5}$$

The first term can be written as

$$\begin{aligned}
\sum_{(i,j) \in \mathcal{E}} ||\boldsymbol{\phi}(u_j) - \boldsymbol{\phi}(u_i)||_2^2 =& \sum_{(u_i,u_j) \in \mathcal{V} \times \mathcal{V}} \mathbf{A}_{ij} ||\boldsymbol{\phi}(u_j) - \boldsymbol{\phi}(u_i)||_2^2 \\
=& Tr(\boldsymbol{\Phi}^\top (\mathbf{D} - \mathbf{A}) \boldsymbol{\Phi}) \\
=& Tr(\boldsymbol{\Phi}^\top \mathcal{L} \boldsymbol{\Phi})
\end{aligned} \tag{5.6}$$

where $\mathbf{D}$ is a diagonal matrix with degrees on the diagonal $\mathbf{D}_{uu} = \sum_{v \in \mathcal{V}} \mathbf{A}_{u,v}$ and $\mathcal{L}$ denotes the combinatorial graph Laplacian.

Thus, under the aforementioned constraints (window size of 1, uniform unbiased sampling strategy and unit norm feature vectors), the node2vec objective is

$$\min_{\boldsymbol{\Phi} \in \mathbb{R}^{n \times d}} \Big(Tr(\boldsymbol{\Phi}^\top \mathcal{L} \boldsymbol{\Phi}) + \sum_{(i,j) \in \mathcal{E}} \log \sum_{u_k \in \mathcal{V}} \exp(1 - \frac{1}{2}||\boldsymbol{\Phi}_k - \boldsymbol{\Phi}_i||_2^2)\Big), \tag{5.7}$$

where $\boldsymbol{\Phi}$ is the representation matrix, whose ith column is the representation for node u_i, $\boldsymbol{\Phi}_i = \boldsymbol{\phi}(u_i)$. Formulation (5.7) bears some resemblance with the objective of another graph-based representation, as we show next. It is interesting to observe that the nod2vec can be written as a smoothness problem (first term) augmented with an auxiliary loss (second term).

Recall that the PVFs are defined as the eigenvectors of the graph Laplacian $\mathcal{L}$. The eigendecomposition of $\mathcal{L}$ is given by $\mathcal{L} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$, where $\mathbf{U}$ is the $|\mathcal{V}| \times |\mathcal{V}|$ matrix whose columns are the eigenvectors of $\mathcal{L}$, and $\boldsymbol{\Lambda}$ is the diagonal matrix whose elements are the corresponding eigenvalues $\Lambda_{ii} = \lambda_i$. The PVFs are the columns of

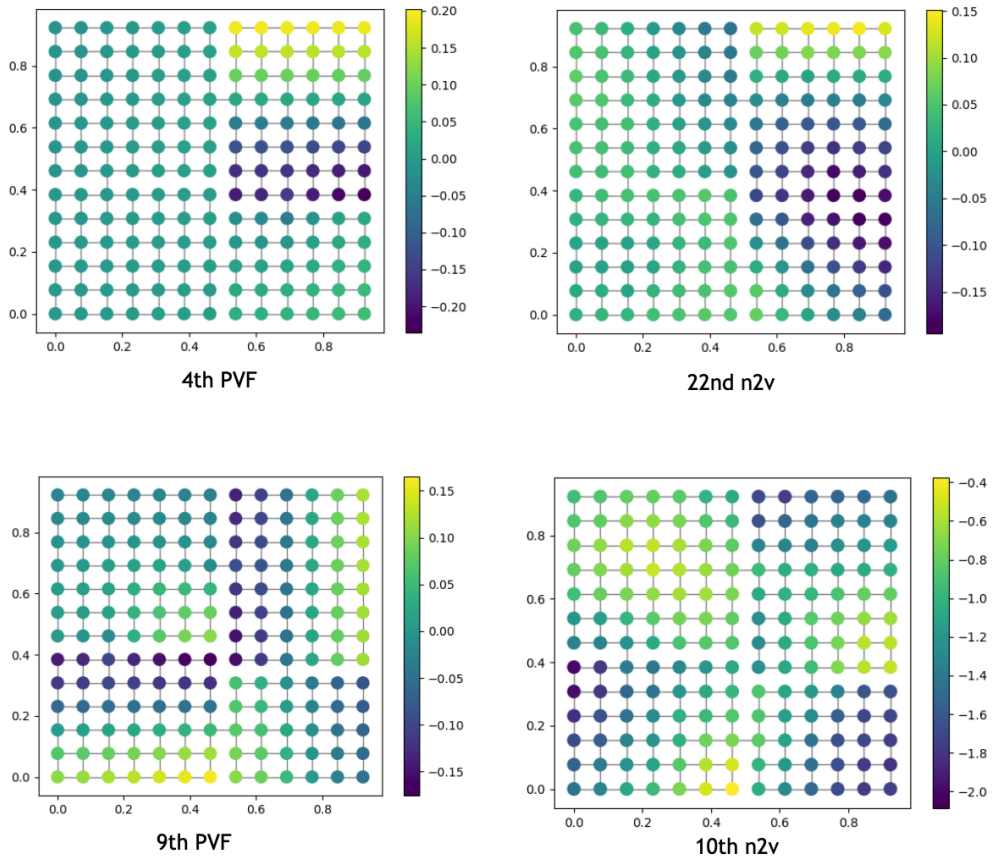


Figure 5.3: Features visualisation on the four room domains. The 4th and 9th PVFs are depicted on the left, compared against the 22nd and 10th node2vec vectors, shown on the right. More dimensions are depicted in Appendix A.4

$\mathbf{U}$ associated with the smallest eigenvalues. The following theorem emerged from spectral graph theory:

Theorem 2. *(Graph Drawing Objective, Koren [72])*

PVFs are the solution of the following objective function.

$$\min_{\mathbf{U}\in\mathbb{R}^{n\times d}} Tr(\mathbf{U}^T\mathbf{L}\mathbf{U}) \;\; s.t. \;\; \mathbf{U}\mathbf{U}^T = \mathbf{I} \tag{5.8}$$

where $\mathbf{U}\mathbf{U}^T = \mathbf{I}$ *ensures that columns of* $\mathbf{U}$ *form an orthonormal basis.*

Interestingly, the first term in (5.7) is the main term in the graph drawing objective, which is the term that imposes smoothness. This overlap can be interpreted

as follows: node2vec share a similar goal with the PVF, with an added component, which acts as a regulariser over the entire graph. Indeed, while the graph drawing objective (and the first term in (5.7)) seeks to maximise representation similarity between strongly connected nodes, the second term in (5.7) encourages to increase the difference in the representation between all pairs (u_k, u_i) when $u_k \neq u_i$. This resemblance is also identified when visualising the values of the learned node2vec representation against the value of the PVFs. In Figure 5.3, we show the values of selected representation dimensions of node2vec and PVFs that have emerged has capturing similar information. Additional visualisations are available in Appedix A.4

5.3 From Node2vec to State2vec

We have demonstrated that node2vec is a mechanism that is able to learn rich representation for RL that allows to approximate the value function in low dimension with better accuracy that state-of-the-art graph-based methods in limited data constraint. We now proceed to focus on the following limitation of this algorithm.

> **What if the graph is too large to be stored in memory? How can we define state neighbourhoods that will capture the underlying geometry of the state space?**

We address these questions by designing a graph-inspired representation learning method for RL. Our method draws inspiration from Grover and Leskovec [45]'s node2vec, we hence refer to it as *state2vec*.

As usual, we are interested in solving an MDP $M = (\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ is a finite set of discrete states, $\mathcal{A}$ a finite set of actions, p describes the transition model, with $p(s, a, s')$ giving the probability of moving from state s to s' given action a, r describes the reward function, with $r(s, a)$ expressing the immediate reward observed when the agent is in state s and takes action a thereafter, and $\gamma \in (0, 1]$ is a discount factor.

The key element of our method is that instead of sampling walks on a graph, state2vec learns state action representations based on exploratory episodes statistics (hence policy-dependent). It optimises the representations such that states that are

successors in a exploratory trajectory have similar representation. Concretely, we maintain a buffer $\mathcal{D}$ of exploratory trajectories $L = \{(s_0, a_0), (s_1, a_1), \ldots, (s_n, a_n)\}$ of maximum length $n+1$ (terminating earlier if it results in an absorbing goal state). Then, the following objective function is optimised using stochastic gradient descent with repect to the representation matrix $\mathbf{\Phi}$:

$$\max_{\mathbf{\Phi}} \sum_{L \in \mathcal{D}} \sum_{(s,a) \in L} \log \mathbb{P}(N(s,a) | \mathbf{\Phi}(s,a)), \tag{5.9}$$

where $N(s_i, a_i) = \{(s_{i+1}, a_{i+1}), (s_{i+2}, a_{i+2}), \ldots\}$ is the neighbourhood of state action pair (s_i, a_i), defined as all the successors of (s_i, a_i) in a trajectory. Similarly to Grover and Leskovec [45], we make a conditional independence assumption and model the conditional likelihood as

$$\mathbb{P}(N(s_i, a_i) | \mathbf{\Phi}(s_i, a_i)) = \prod_{(s_j, a_j) \in N(s_i, a_i)} \mathbb{P}(s_j, a_j | \mathbf{\Phi}(s_i, a_i)) \,. \tag{5.10}$$

Unlike the node2vec algorithm, we account for the fact that neighbours that are further in time should be further discounted. We do so by modelling the the likelihood of every source-neighbour pair as a sigmoid weighted by a discount factor

$$\mathbb{P}(s_j, a_j | \mathbf{\Phi}(s_i, a_i)) = \gamma^{j-i} \sigma(\mathbf{\Phi}(s_j, a_j) \cdot \mathbf{\Phi}(s_i, a_i)) \,, \tag{5.11}$$

where σ denotes the sigmoid function.

Once the state2vec representations Φ are learned, we can use them as basis functions for solving the RL problem (GRPI, Chpater 4.2). In fact, since the state2vec representations are learned from fully exploratory data, and reward-agnostic, we can use the same representation to solve different tasks with shared dynamics and varying reward functions. We thus consider the multi-task scenario, consisting of multiple MDPs $\mathcal{M} = \{M_1, M_2, \ldots\}$ where a task $M_k = (S, A, p, r_k, \gamma)$ is identifiable by its reward function r_k.

The solving of task $M_k \in \mathcal{M}$ given the structural representation $\mathbf{\Phi}$ reduces to

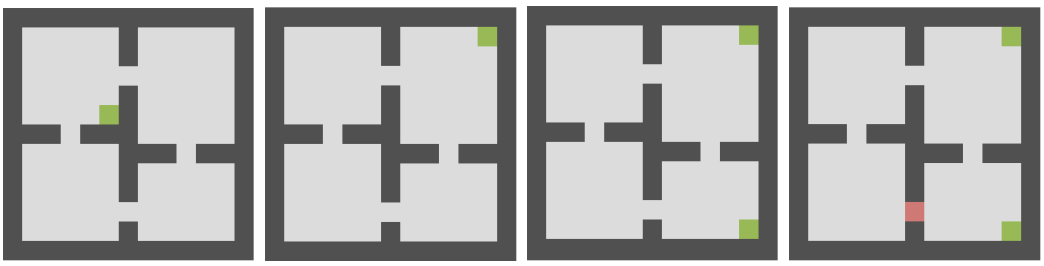
learning parameter θ_k that best approximates the the following value function

$$\hat{Q}^{\pi_k}(s,a) = \mathbf{\Phi}(s,a)^\top \theta_w \,. \tag{5.12}$$

This can be achieved using any parametric RL algorithm, such as fitted Q-learning or LSPI [126, 73].

5.4 Experiments

We consider the four-room domain [141] shown in Figure 5.4. It is a two-dimensional space quantized into 169 states, 4 of which are doorways. The agent starts at a random location, and must collect a goal object at a location defined by the task. Depending on the task, the environment also contains "dangerous" zones. The goal object's location is shown in green in Figure 5.4, while the dangerous states are depicted in red. Collecting an object gives an instantaneous reward of $+100$, and entering a dangerous state gives an instantaneous penalty of -10. The the episode terminates when a goal object is collected.



(a) (b) (c) (d)

Figure 5.4: Four-room environment with different configurations. There are 169 states in total. Goal objects are located in absorbing states shown in green (+100 reward), while states to avoid are shown in red (-10 penalty).

5.4.1 Results

First, we learn the state space's geometry by running state2vec. Note that state2 is policy dependent, since the representations are learned from trajectories collected under a policy. Choosing an exploratory policy (*e.g.* the uniform policy) allows us to learn the overall geometry of the state space. In the feature learning phase, we col-

lect 300 sample walks of length 100 and run state2vec with discount factor $\gamma = 0.8$ for varying dimensions d. Figure 5.5 visualises the low dimensional (projection onto the first two principal components) state2vec representation of original dimension $d = 50$. We can clearly see that the representations have clustered the states within the same room together, while isolating the doorway states. The learned embedding are shown to preserve the geometry of the state space and identify states that have a special structural role (e.g. doorways).

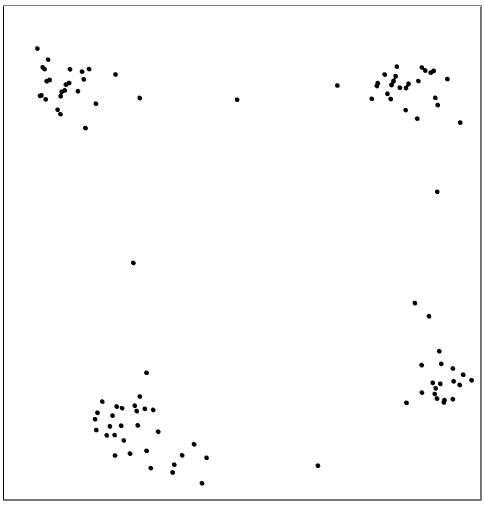


Figure 5.5: Visualisation of the state2vec representation in feature space (2D PCA projection). The original state2vec has $d = 50$ dimensions.

We then use the learned state2vec features to learn the optimal policy of each individual task in Figure 5.4 following the GRPI framework (Chapter 4.2). We collect sampled realisations of the form (s, a, s') by simulating 50 episodes of maximum length 200 (terminating earlier if the goal is reached) and run LSPI Lagoudakis [73] with state2vec representations as basis vectors to learn the weights θ_k in (5.12). Figure 5.7 shows the performance in terms of average cumulative reward for varying values of d. As it can be seen, we are able to achieve strong performance (maximum reward) for all tasks when using the pre-computed state2vec representations of dimensionality 100. Figure 5.6 shows the performance when we make the size of the data (number of simulated episodes) used to train state2vec. We observe a fast reinforcement learning, with optimal policies learned with only 50 exploratory episodes collected. These results suggest that information captured in

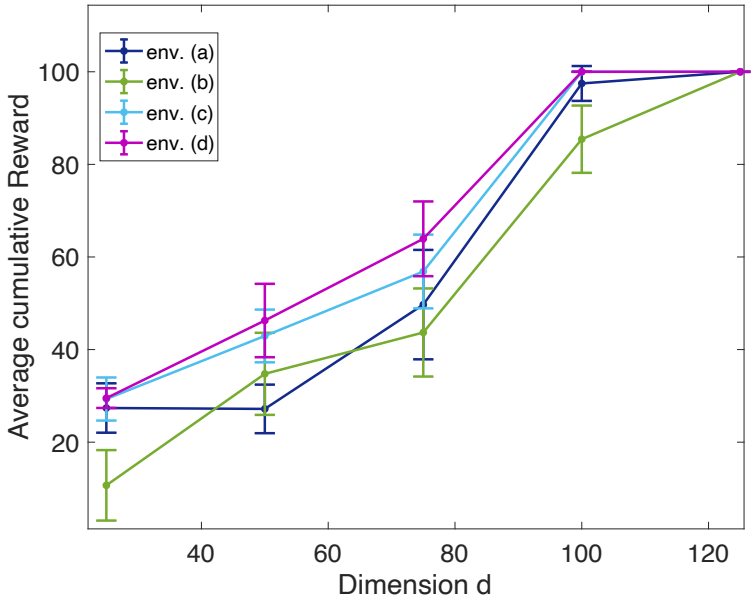


Figure 5.6: Average cumulative reward after running GRPI using pre-trained state2vec for each of the environment in Figure 5.4. Average and standard deviation over 10 independent runs.

the state2vec representation greatly benefits policy learning, reducing the need for extensive fruther exploration.

We compare the quality of state2vec embeddings against node2vec for linear value function approximation. Figure 5.8 shows an improved performance of state2vec over node2vec in terms of average cumulative reward. We suspect that the gain in performance comes for the fact that state2vec is designed for RL, whereas node2vec is a generic graph embedding algorithm. Specifically, in the objective function, the notion of neighborhood in state2vec is such that further states in time are discounted more than the immediate successors, which resonates with the definition of the value function: the expected sum of *discounted future* rewards.

5.5 State2vec and the Successor Representation

By design, the state2vec representation of a state-action pair will be similar to that of a successor state-action pair. This concept of predictivity relates to the Successor Representation (SR) [27], where a state is represented by a vector of the expected sums of discounted occupancy of future states. Formally, under a policy π, the SR

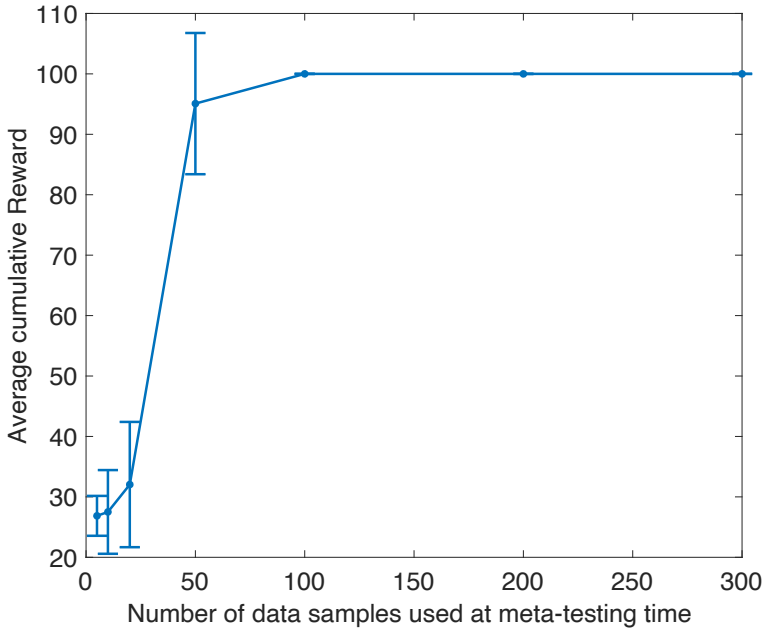


Figure 5.7: Average cumulative reward using state2vec (with $d = 100$) and LSPI for environment (5.4(a)). Average and standard deviation over 10 independent runs.

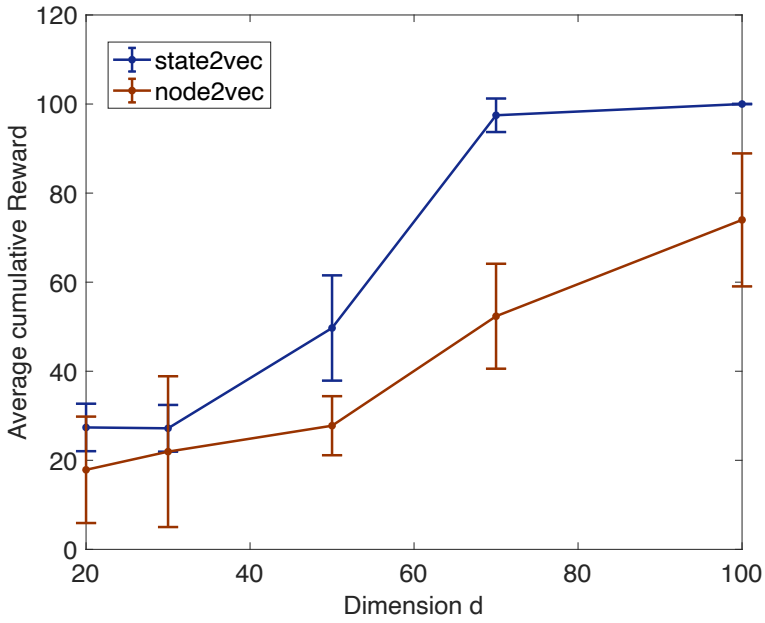


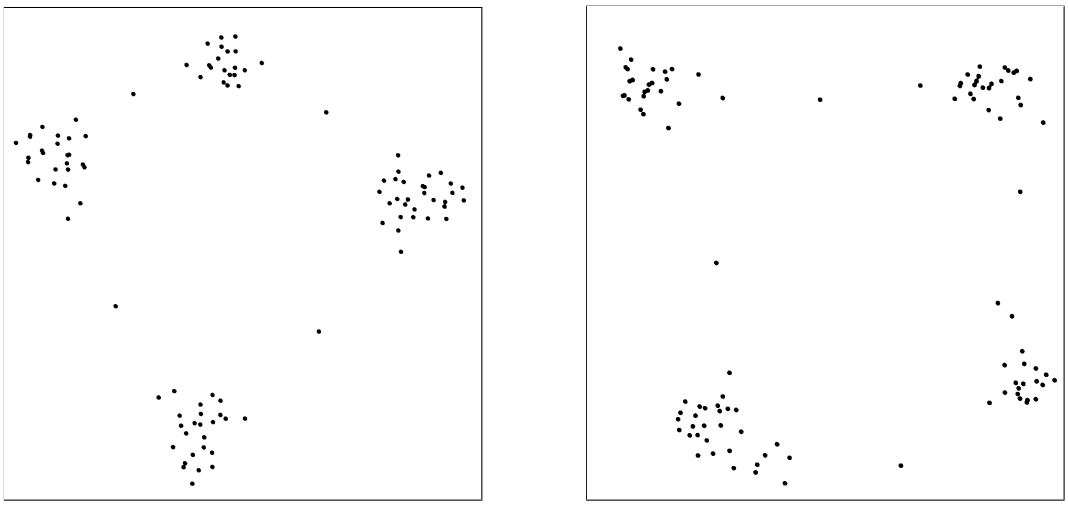
Figure 5.8: Comparison between node2vec and state2vec on environment (5.4(a)) (one goal at the corner). Average and standard deviation over 10 independent runs.

$\mathbf{\Psi}^{\pi}$ is defined, for $\gamma < 1$, as:

$$\mathbf{\Psi}^{\pi}(s,a,s') = \mathbb{E}_{\pi}\Big[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}(s_t = s') | s_0 = s, a_0 = a\Big], \tag{5.13}$$

where $\mathbb{I}(s_t = s') = 1$ if $s_t = s'$ and 0 otherwise.

The SR is a predictive type of representation, which represents a state action pair (s,a) as a feature vector $\mathbf{\Psi}^{\pi}_{s,a}$ such that, under policy π, the representation $\mathbf{\Psi}^{\pi}_{s,a}$ is similar to the feature vector of successor states. Due to their predictive nature, we investigate whether state2vec and the SR capture the same kind of information in their representational spaces. By visualising the 2D PCA projection of the SR and state2vec of the four-room environment side-by-side (Figure 5.9), it appears that state2vec is a close approximation of the exact SR. In both cases, the representations group states within the same room together, while isolating the doorway states. Both embeddings preserve the geometry of the state space.



(a) (b)

Figure 5.9: Visualisation of the state representations in reduced feature space (2D PCA projection). (a) The exact successor representation, each vector in the original feature space has dimension 169. (b) State2vec representation with original dimension $d = 50$.

5.6 Conclusion and Future Work

This chapter gave further analysis of the graph representation learning algorithm node2vec when applied to RL problems. This analysis has provided a better understanding of the kind of structural information about the MDP that is needed for approximating the value function in low dimension. We also established interesting theoretical and visual links between node2vec representations and the PVFs. Build-

ing on these promising findings, we designed state2vec, a novel graph-inspired representation learning method, that is as computationally efficient as node2vec (linear in d, the dimension of the representation), has a considerably lower memory complexity (linear in $d * L$, with L the length of the exploratory trajectories) and learns representations that are more RL specific, while remaining task-agnostic. We experimentally showed that state2vec is a good approximation of the successor feature. Additionally, we showed that training the state2vec in an unsupervised way results in embeddings that capture the geometry of the state space and ensure sample-efficiency when solving downstream RL tasks. While promising, our propose method has a few remaining limitations. The following highlights limitations and proposes important directions for future work:

- In the current framework, state representations can only be learned for states encountered during the exploratory trajectory collection phase. As a consequence, if not enough exploration is allocated, it is likely that the agent will come across unseen states for which no known representation exists. Consequently, future work should focus on extending state2vec to generalise across states (meaning we can compute good approximation of the state2vec vector for an unseen state). In the non tabular case, this could be achieved by function approximation using state observational features.

- In single task settings, it would most likely be beneficial to fine tune the representations such that they incorporate task specific information. For example, future work could consider using reward information when learning to optimise the representation, or fine-tune the representation using online data as the policy improves over time.

- We have shown that state2vec is a close approximation of the successor representation [27]. A comparative performance study between state2vec and other SR approximation strategies (such as Successor Features [3]) would help in understanding the unique advantages of state2vec.

Chapter 6

Towards Efficient Credit Assignment

Beyond the challenge of learning a rich representation of the environment, an important bottleneck in the efficiency of RL algorithms is in the way reward information is propagated through the state and action space. The problem of accurately and efficiently assigning credit (or blame) to past decisions or situations is known as the *credit assignment problem*. In Chapter 5, we have seen how a representation resembling the successor representation enables efficient learning of the value function. We now investigate how similar ideas can be beneficial to the credit assignment problem in online learning. Consider the opposite view of the successor representation [26]; the predecessor representation, which describes the possible past of a state, instead of its future. This notion appears to be important for credit assignment in reinforcement learning. The intuition is that all possible predecessor states and actions (the observed ones as well as the counterfactual ones) of a given rewarding event can contribute to observing that reward, and should therefore be assigned some credit. Following this intuition, in this chapter, we focus on the following question:

> **In online learning, while in state s, can credit assignment be made efficient by propagating reward information through the predecessor representation of state s?**

Appropriate credit assignment has long been a major research topic in artificial intelligence [101]. To make effective decisions and understand the world, we need to accurately associate events, like rewards or penalties, to relevant earlier decisions or

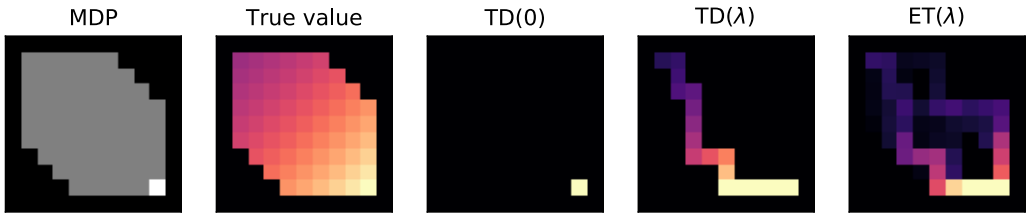


Figure 6.1: A comparison of TD(0), TD(λ), and the new expected-trace algorithm ET(λ) (with $\lambda = 0.9$). The MDP is illustrated on the left. Each episode, the agent moves randomly down and right from the top left to the bottom right, where any action terminates the episode. Reward on termination are $+1$ with probability 0.2, and zero otherwise—all other rewards are zero. We plot the value estimates after the first positive reward, which occurred in episode 7. We see a) TD(0) only updated the last state, b) TD(λ) updated the trajectory in this episode, and c) ET(λ) additionally updated trajectories from earlier (unrewarding) episodes.

situations. This is important both for learning accurate predictions, and for making good decisions.

Temporal credit assignment can be achieved with repeated temporal-difference (TD) updates [143]. One-step TD updates propagate information slowly: when a surprising value is observed, the state immediately preceding it is updated, but no earlier states or decisions are updated. *Multi-step* updates [143, 146] propagate information faster over longer temporal spans, speeding up credit assignment and learning. Multi-step updates can be implemented online using *eligibility traces* [143], without incurring important additional computational expense, even if the time spans are long; these algorithms have computation that is independent of the temporal span of the prediction [161].

Traces provide temporal credit assignment, but do not assign credit *counterfactually* to states or actions that *could* have led to the current state, but did not do so this time. Credit will eventually trickle backwards over the course of multiple visits, but this can take many iterations. As an example, suppose we collect a key to open a door, which leads to an unexpected reward. Using standard one-step TD learning, we would update the state in which the door opened. Using eligibility traces, we would also update the preceding trajectory, including the acquisition of the key. But we would not update other sequences that *could* have led to the reward, such as collecting a spare key or finding a different entrance.

The problem of credit assignment to counterfactual states may be addressed by learning a model, and using the model to propagate credit [144, 106]; however, it has often proven challenging to construct and use models effectively in complex environments [cf. 164]. Similarly, *source traces* [118] model full potential histories in tabular settings, but rely on estimated importance-sampling ratios of state distributions, which are hard to estimate in non-tabular settings.

The main contributions of this chapter can be summarised as follows:

- We introduce a new approach to counterfactual credit assignment, based on a concept inspired by the notion of predecessor features that we call *expected eligibility traces*.

- We present a family of algorithms, which we call ET(λ), that use expected traces to update their predictions. We provide a theoretical analysis of the nature of these expected traces, and illustrate their benefits empirically in several settings—see Figure 6.1 for a first illustration.

- We introduce a bootstrapping mechanism that provides a spectrum of algorithms between standard eligibility traces and expected eligibility traces, and also discuss ways to apply these ideas with deep neural networks.

- We discuss possible extensions and connections to related ideas such as successor features.

6.1 Background

As usual, we model sequential decision problems as Markov decision processes[1] (MDP) $(\mathcal{S}, \mathcal{A}, p, r)$ [122], with state space $\mathcal{S}$, action space $\mathcal{A}$, and transition function $p(s,a,s')$ and an expected reward function $r(s,a)$. An agent selects actions according to its policy π, and observes random rewards and states generated according to the MDP, resulting in trajectories $\tau_{t:T} = \{S_t, A_t, R_{t+1}, S_{t+1}, \ldots, S_T\}$. A

[1] The ideas extend naturally to POMDPs [cf. 64].

central goal is to predict *returns* of future discounted rewards [146]

$$\begin{aligned} G_t \equiv G(\tau_{t:T}) &= R_{t+1} + \gamma_{t+1} R_{t+2} + \gamma_{t+1}\gamma_{t+2} R_{t+3} + \dots \\ &= \sum_{i=1}^{T} \gamma_{t+i}^{(i-1)} R_{t+i}\,, \end{aligned}$$

where T is for instance the time the current episode terminates or $T = \infty$, and where $\gamma_t \in [0,1]$ is a (possibly constant) discount factor and $\gamma_t^{(i)} = \prod_{k=1}^{i} \gamma_{t+k}$. The value $V_\pi(s) = \mathbb{E}[G_t | S_t = s, \pi]$ of state s is the expected return. Rather than writing the return as a random variable G_t, it will be convenient to instead write it as an explicit function $G(\tau)$ of the random trajectory τ. Note that $G(\tau_{t:T}) = R_{t+1} + \gamma_{t+1} G(\tau_{t+1:T})$.

We approximate the value with a function $V_{\mathbf{w}}(s) \approx V_\pi(s)$. This can for instance be a table—with a single separate entry $w[s]$ for each state—a linear function of some input features, or a non-linear function such as a neural network with parameters $\mathbf{w}$. The goal is to iteratively update $\mathbf{w}$ with

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \Delta\mathbf{w}_t$$

such that $V_{\mathbf{w}}$ approaches the true V_π. Perhaps the simplest algorithm to do so is the Monte Carlo (MC) algorithm

$$\Delta\mathbf{w}_t \equiv \alpha(R_{t+1} + \gamma_{t+1} G(\tau_{t+1:T}) - V_{\mathbf{w}}(S_t)) \nabla_{\mathbf{w}} V_{\mathbf{w}}(S_t)\,.$$

Monte Carlo is effective, but has high variance, which can lead to slow learning. TD learning [143, 146] instead replaces the return with the current estimate of its expectation $V(S_{t+1}) \approx G(\tau_{t+1:T})$, yielding

$$\begin{aligned} \Delta\mathbf{w}_t &\equiv \alpha \delta_t \nabla_{\mathbf{w}} V_{\mathbf{w}}(S_t)\,, \\ \text{where} \quad \delta_t &\equiv R_{t+1} + \gamma_{t+1} V_{\mathbf{w}}(S_{t+1}) - V_{\mathbf{w}}(S_t)\,, \end{aligned} \tag{6.1}$$

where δ_t is called the temporal-difference (TD) error. We can interpolate between these extremes, for instance with λ-returns which smoothly mix values and sampled

returns:

$$G^{\lambda}(\tau_{t:T}) = R_{t+1} + \gamma_{t+1}\big((1-\lambda)V_{\mathbf{w}}(S_{t+1}) + \lambda G^{\lambda}(\tau_{t+1:T})\big)\,.$$

'Forward view' algorithms, like the MC algorithm, use returns that depend on future trajectories and need to wait until the end of an episode to construct their updates, which can take a long time. Conversely, 'backward view' algorithms rely only on past experiences and can update their predictions online, during an episode. Such algorithms build an *eligibility trace* [143, 146]. An example is TD(λ):

$$\Delta\mathbf{w}_t \equiv \alpha\delta_t\boldsymbol{e}_t\,, \qquad \text{with} \qquad \boldsymbol{e}_t = \gamma_t\lambda\boldsymbol{e}_{t-1} + \nabla_{\mathbf{w}}V_{\mathbf{w}}(S_t)\,,$$

where $\boldsymbol{e}_t$ is an accumulating eligibility trace. This trace can be viewed as a function $\boldsymbol{e}_t \equiv \boldsymbol{e}(\tau_{0:t})$ of the trajectory of past transitions. The TD update in (6.1) is known as TD(0), because it corresponds to using $\lambda = 0$. TD($\lambda = 1$) corresponds to an online implementation of the MC algorithm. Other variants exist, using other kinds of traces, and equivalences have been shown between these algorithms and their forward view using λ-returns: these backward-view algorithms converge to the same solution as the corresponding forward view, and can in some cases yield equivalent weight updates [143, 167, 161].

6.2 Expected Traces

The main idea is to use the concept of an *expected eligibility trace*, defined as

$$\boldsymbol{z}(s) \equiv \mathbb{E}\left[\boldsymbol{e}_t \mid S_t = s\right],$$

where the expectation is over the agent's policy and the MDP dynamics. We introduce a concrete family of algorithms, which we call ET(λ) and ET(λ, η), that learn expected traces and use them in value updates. We analyse these algorithms theoretically, describe specific instances, and discuss computational and algorithmic properties.

Algorithm 2 ET(λ)

initialise $\mathbf{w}$, $\boldsymbol{\theta}$
for M episodes **do**
    initialise $\boldsymbol{e} = \mathbf{0}$
    observe initial state S
    repeat for each step in episode m
        generate R and S'
        $\delta \leftarrow R + \gamma V_{\mathbf{w}}(S') - V_{\mathbf{w}}(S)$
        $\boldsymbol{e} \leftarrow \gamma\lambda \boldsymbol{e} + \nabla_{\mathbf{w}} V_{\mathbf{w}}(S)$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta \frac{\partial \boldsymbol{z}_{\boldsymbol{\theta}}(S)}{\partial \boldsymbol{\theta}} (\boldsymbol{e} - \boldsymbol{z}_{\boldsymbol{\theta}}(S))$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta \boldsymbol{z}_{\boldsymbol{\theta}}(S)$
    until S is terminal
end for
Return w

6.2.1 ET(λ)

We propose to learn approximations $\boldsymbol{z}_{\boldsymbol{\theta}}(S_t) \approx \boldsymbol{z}(S_t)$, with parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ (e.g., the weights of a neural network). One way to learn $\boldsymbol{z}_{\boldsymbol{\theta}}$ is by updating it towards the instantaneous trace $\boldsymbol{e}_t$, by minimizing an empirical loss $\ell(\boldsymbol{e}_t, \boldsymbol{z}_{\boldsymbol{\theta}}(S_t))$. For instance, ℓ could be a component-wise squared loss, optimized with stochastic gradient descent:

$$\begin{aligned}
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \Delta\boldsymbol{\theta}_t\,, \text{ where} \\
\Delta\boldsymbol{\theta}_t &= -\beta \frac{\partial}{\partial \boldsymbol{\theta}} \frac{1}{2} (\boldsymbol{e}_t - \boldsymbol{z}_{\boldsymbol{\theta}}(S_t))^\top (\boldsymbol{e}_t - \boldsymbol{z}_{\boldsymbol{\theta}}(S_t)) \\
&= \beta \frac{\partial \boldsymbol{z}_{\boldsymbol{\theta}}(S_t)}{\partial \boldsymbol{\theta}} (\boldsymbol{e}_t - \boldsymbol{z}_{\boldsymbol{\theta}}(S_t))\,,
\end{aligned}$$

where $\frac{\partial z_{\boldsymbol{\theta}}(S_t)}{\partial \boldsymbol{\theta}}$ is a $|\boldsymbol{\theta}| \times |\boldsymbol{e}|$ Jacobian[2] and β is a step size.

The idea is then to use $\boldsymbol{z}_{\boldsymbol{\theta}}(s) \approx \mathbb{E}[\boldsymbol{e}_t \mid S_t = s]$ in place of $\boldsymbol{e}_t$ in the value update, which becomes

$$\Delta\mathbf{w}_t \equiv \delta_t \boldsymbol{z}_{\boldsymbol{\theta}}(S_t)\,. \tag{6.2}$$

We call this ET(λ). Below, we prove that this update can be unbiased and can have lower variance than TD(λ). Algorithm 2 shows pseudo-code for a concrete instance

[2] Auto-differentiation can efficiently compute this update with comparable computation to the loss calculation.

of ET(λ).

6.2.2 Interpretation and ET(λ, η)

We can interpret TD(0) as taking the MC update and replacing the return from the subsequent state, which is a function of the future trajectory, with a state-based estimate of its expectation: $V(S_{t+1}) \approx \mathbb{E}[G(\tau_{t+1:T})|S_{t+1}]$. This becomes most clear when juxtaposing the updates

$$\Delta \mathbf{w}_t \equiv \alpha(R_{t+1} + \gamma_{t+1} G(\tau_{t+1:T}) - V_{\mathbf{w}}(S_t))\boldsymbol{\nabla}_t\,, \tag{MC}$$

$$\Delta \mathbf{w}_t \equiv \alpha(R_{t+1} + \gamma_{t+1} V_{\mathbf{w}}(S_{t+1}) - V_{\mathbf{w}}(S_t))\boldsymbol{\nabla}_t\,, \tag{TD}$$

where we used a shorthand $\boldsymbol{\nabla}_t \equiv \nabla_{\mathbf{w}} V_{\mathbf{w}}(S_t)$.

TD(λ) also uses a function of a trajectory: the trace $\boldsymbol{e}_t$. We propose replacing this as well with a function state $\boldsymbol{z}_{\boldsymbol{\theta}}(S_t) \approx \mathbb{E}[\boldsymbol{e}(\tau_{0:t})|S_t]$: the expected trace. Again juxtaposing:

$$\Delta \mathbf{w}_t \equiv \alpha \delta_t \boldsymbol{e}(\tau_{0:t})\,, \tag{TD(λ)}$$

$$\Delta \mathbf{w}_t \equiv \alpha \delta_t \boldsymbol{z}_{\boldsymbol{\theta}}(S_t)\,. \tag{ET(λ)}$$

When switching from MC to TD(0), the dependence on the trajectory was replaced with a state-based value estimate to bootstrap on. We can interpolate smoothly between MC and TD(0) via λ. This is often useful to trade off variance of the return with potential bias of the value estimate. For instance, we might not have access to the true state s, and might instead have to rely on features $\mathbf{x}(s)$. Then we cannot always represent or learn the true values $V(s)$—for instance different states may be aliased [172].

Similarly, when moving from TD(λ) to ET(λ) we replaced a trajectory-based trace with a state-based estimate. This might induce bias and, again, we can smoothly interpolate by using a recursively defined mixture trace $\boldsymbol{y}_t$, as defined

as[3]

$$\boldsymbol{y}_t = (1-\eta)\boldsymbol{z}_{\boldsymbol{\theta}}(S_t) + \eta\left(\gamma_t\lambda\boldsymbol{y}_{t-1} + \nabla_{\mathbf{w}}V_{\mathbf{w}}(S_t)\right). \tag{6.3}$$

This recursive usage of the estimates $\boldsymbol{z}_{\boldsymbol{\theta}}(s)$ at previous states is analogous to bootstrapping on future state values when using a λ-return, with the important difference that the arrow of time is opposite. This means we do not first have to convert this into a backward view: the quantity can already be computed from past experience directly. We call the algorithm that uses this mixture trace ET(λ, η):

$$\Delta\mathbf{w}_t \equiv \alpha\delta_t\boldsymbol{y}(S_t). \tag{ET(λ, η)}$$

Note that if $\eta = 1$ then $\boldsymbol{y}_t = \boldsymbol{e}_t$ equals the instantaneous trace: ET(λ, 1) is equivalent to TD(λ). If $\eta = 0$ then $\boldsymbol{y}_t = \boldsymbol{z}_t$ equals the expected trace; the algorithm introduced earlier as ET(λ) is equivalent to ET(λ, 0). By setting $\eta \in (0,1)$, we can smoothly interpolate between these extremes.

6.3 Theoretical Analysis

We now analyse the new ET algorithms theoretically. First we show that if we use $\boldsymbol{z}(s)$ directly and s is Markov then the update has the same expectation as TD(λ) (though possibly with lower variance), and therefore also inherits the same fixed point and convergence properties.

Lemma 1. *If s is Markov, then*

$$\mathbb{E}[\,\delta_t\boldsymbol{e}_t \mid S_t = s\,] = \mathbb{E}[\,\delta_t \mid S_t = s\,]\,\mathbb{E}[\,\boldsymbol{e}_t \mid S_t = s\,].$$

Proof. In Appendix A.1. □

Proposition 1. *Let $\boldsymbol{e}_t$ be any trace vector, updated in any way. Let $\boldsymbol{z}(s) = \mathbb{E}[\,\boldsymbol{e}_t \mid S_t = s\,]$. Consider the ET($\lambda$) algorithm $\Delta\mathbf{w}_t = \alpha_t\delta_t\boldsymbol{z}(S_t)$. For all Markov s the expectation of this update is equal to the expected update with instantaneous*

[3] While $\boldsymbol{y}_t$ depends on both η and λ we leave this dependence implicit, as is conventional for traces.

trace $\boldsymbol{e}_t$, and the variance is lower or equal:

$$\mathbb{E}[\,\alpha_t\delta_t\boldsymbol{z}(S_t)|S_t=s] = \mathbb{E}[\,\alpha_t\delta_t\boldsymbol{e}_t|S_t=s] \qquad \textit{and}$$
$$\mathbb{V}[\alpha_t\delta_t\boldsymbol{z}(S_t)|S_t=s] \leq \mathbb{V}[\alpha_t\delta_t\boldsymbol{e}_t|S_t=s]\,,$$

where the second inequality holds component-wise for the update vector, and is strict when $\mathbb{V}[\boldsymbol{e}_t|S_t] > 0$.

Proof. We have

$$\begin{aligned}
&\mathbb{E}[\,\alpha_t\delta_t\boldsymbol{e}_t \mid S_t=s] \\
&= \mathbb{E}[\,\alpha_t\delta_t \mid S_t=s]\,\mathbb{E}[\,\boldsymbol{e}_t \mid S_t=s] && \text{(Lemma 1)} \\
&= \mathbb{E}[\,\alpha_t\delta_t \mid S_t=s]\,\boldsymbol{z}(s) \\
&= \mathbb{E}[\,\alpha_t\delta_t\boldsymbol{z}(S_t) \mid S_t=s]\,. && (6.4)
\end{aligned}$$

Denote the i-th component of $\boldsymbol{z}(S_t)$ by $z_{t,i}$ and the i-th component of $\boldsymbol{e}_t$ by $e_{t,i}$. Then, we also have

$$\begin{aligned}
&\mathbb{E}\left[\,(\alpha_t\delta_t z_{t,i})^2|S_t=s\right] = \mathbb{E}\left[\,\alpha_t^2\delta_t^2 \mid S_t=s\right] z_{t,i}^2 \\
&= \mathbb{E}\left[\,\alpha_t^2\delta_t^2 \mid S_t=s\right]\mathbb{E}[\,e_{t,i}|S_t=s]^2 \\
&= \mathbb{E}\left[\,\alpha_t^2\delta_t^2 \mid S_t=s\right]\left(\mathbb{E}\left[\,e_{t,i}^2|S_t=s\right] - \mathbb{V}[e_{t,i}|S_t=s]\right) \\
&\leq \mathbb{E}\left[\,\alpha_t^2\delta_t^2 \mid S_t=s\right]\mathbb{E}\left[\,e_{t,i}^2 \mid S_t=s\right] \\
&= \mathbb{E}\left[\,(\alpha_t\delta_t e_{t,i})^2 \mid S_t=s\right]\,,
\end{aligned}$$

where the last step used the fact that s is Markov, and the inequality is strict when $\mathbb{V}[\boldsymbol{e}_t|S_t] > 0$. Since the expectations are equal, as shown in (6.4), the conclusion follows. □

Interpretation Proposition 1 is a strong result: it holds for any trace update, including accumulating traces [142, 143], replacing traces [136], dutch traces [167, 162, 161], and future traces that may be discovered. It implies convergence of ET(λ) under the same conditions as TD(λ) [26, 114, 156] with lower variance

when $\mathbb{V}[\boldsymbol{e}_t|S_t] > 0$, which is the common case.

Next, we consider what happens if we violate the assumptions of Proposition 1. We start by analysing the case of a learned approximation $\boldsymbol{z}_t(s) \approx \boldsymbol{z}(s)$ that relies solely on observed experience.

Proposition 2. *Let $\boldsymbol{e}_t$ an instantaneous trace vector. Then let $\boldsymbol{z}_t(s)$ be the empirical mean $\boldsymbol{z}_t(s) = \frac{1}{n_t(s)} \sum_i^{n_t(s)} \boldsymbol{e}_{t_i^s}$, where t_i^s-s denote past times when we have been in state s, that is $S_{t_i^s} = s$, and $n_t(s)$ is the number of visits to s in the first t steps. Consider the expected trace algorithm $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t \delta_t \boldsymbol{z}_t$. If S_t is Markov, the expectation of this update is equal to the expected update with instantaneous traces $\boldsymbol{e}_t$, while attaining a potentially lower variance:*

$$\mathbb{E}[\,\alpha_t \delta_t \boldsymbol{z}_t(S_t) \mid S_t\,] = \mathbb{E}[\,\alpha_t \delta_t \boldsymbol{e}_t \mid S_t\,] \qquad \textit{and}$$
$$\mathbb{V}[\alpha_t \delta_t \boldsymbol{z}_t(S_t) \mid S_t] \leq \mathbb{V}[\alpha_t \delta_t \boldsymbol{e}_t \mid S_t]\,,$$

where the second inequality holds component-wise. The inequality is strict when $\mathbb{V}[\boldsymbol{e}_t \mid S_t] > 0$.

Proof. In Appendix. □

Interpretation Proposition 2 mirrors Proposition 1 but, importantly, covers the case where we estimate the expected traces from data, rather than relying on exact estimates. This means the benefits extend to this pure learning setting. Again, the result holds for any trace update. The inequality is typically strict when the path leading to state $S_t = s$ is stochastic (due to environment or policy).

Next we consider what happens if we do not have Markov states and instead have to rely on, possibly non-Markovian, features $\mathbf{x}(s)$. We then have to pick a function class and for the purpose of this analysis we consider linear expected traces $\boldsymbol{z}_{\boldsymbol{\Theta}}(s) = \boldsymbol{\Theta}\mathbf{x}(s)$ and values $V_{\mathbf{w}}(s) = \mathbf{w}^\top \mathbf{x}(s)$, as convergence for non-linear values can not always be assured even for standard TD(λ) [157], without additional assumptions [e.g., 108, 19]. The following property of the mixture trace is used in the proposition below.

Proposition 3. *The mixture trace* $\boldsymbol{y}_t$ *defined in* (6.3) *can be written as* $\boldsymbol{y}_t = \mu \boldsymbol{y}_{t-1} + \mathbf{x}_t$ *with decay parameter* $\mu = \eta\gamma\lambda$ *and signal* $\mathbf{x}_t = (1-\eta)\boldsymbol{z}_{\boldsymbol{\theta}}(S_t) + \eta\ \nabla_{\mathbf{w}} V_{\mathbf{w}}(S_t)$, *such that*

$$\boldsymbol{y}_t = \sum_{k=0}^{t} (\eta\gamma\lambda)^k \left[(1-\eta)\boldsymbol{z}_{\boldsymbol{\theta}}(S_{t-k}) + \eta\ \nabla_{\mathbf{w}} V_{\mathbf{w}}(S_{t-k})\right]. \tag{6.5}$$

Proof. In Appendix. □

Recall $\boldsymbol{y}_t = \boldsymbol{e}_t$ when $\eta = 1$, and $\boldsymbol{y}_t = \boldsymbol{z}_{\boldsymbol{\theta}}(S_t)$ when $\eta = 0$, as can be verified by inspecting (6.5) (and using the convention $0^0 = 1$). We use this proposition to prove the following.

Proposition 4. *When using approximations* $z_{\boldsymbol{\Theta}}(s) = \boldsymbol{\Theta}\mathbf{x}(s)$ *and* $V_{\mathbf{w}}(s) = \mathbf{w}^\top \mathbf{x}(s)$ *then, if* $(1-\eta)\boldsymbol{\Theta} + \eta\mathbb{I}$ *is non-singular, ET(*λ, η*) has the same fixed point as TD(*$\lambda\,\eta$*).*

Proof. In Appendix. □

Interpretation This result implies that linear ET(λ, η) converges under similar conditions as linear TD(λ') for $\lambda' = \lambda \cdot \eta$. In particular, when $\boldsymbol{\Theta}$ is non-singular, using the approximation $\boldsymbol{z}_{\boldsymbol{\Theta}}(s) = \boldsymbol{\Theta}\mathbf{x}(s)$ in ET(λ, 0) = ET(λ) implies convergence to the fixed point of TD(0).

Though ET(λ, η) and TD($\lambda\,\eta$) have the same fixed point, the algorithms are not equivalent. In general, their updates are not the same. Linear approximations are more general than tabular functions (which are linear functions of a indicator vector for the current state), and we have already seen in Figure 6.1 that ET(λ) behaves quite differently from both TD(0) and TD(λ), and we have seen its variance can be lower in Propositions 1 and 2. Interestingly, $\boldsymbol{\Theta}$ resembles a preconditioner that speeds up the linear semi-gradient TD update, similar to how second-order optimisation algorithms [2, 96] precondition the gradient updates.

6.4 Empirical Analysis

From the insights above, we expect that ET(λ) yields lower prediction errors because it has lower variance and aggregates information across episodes better.

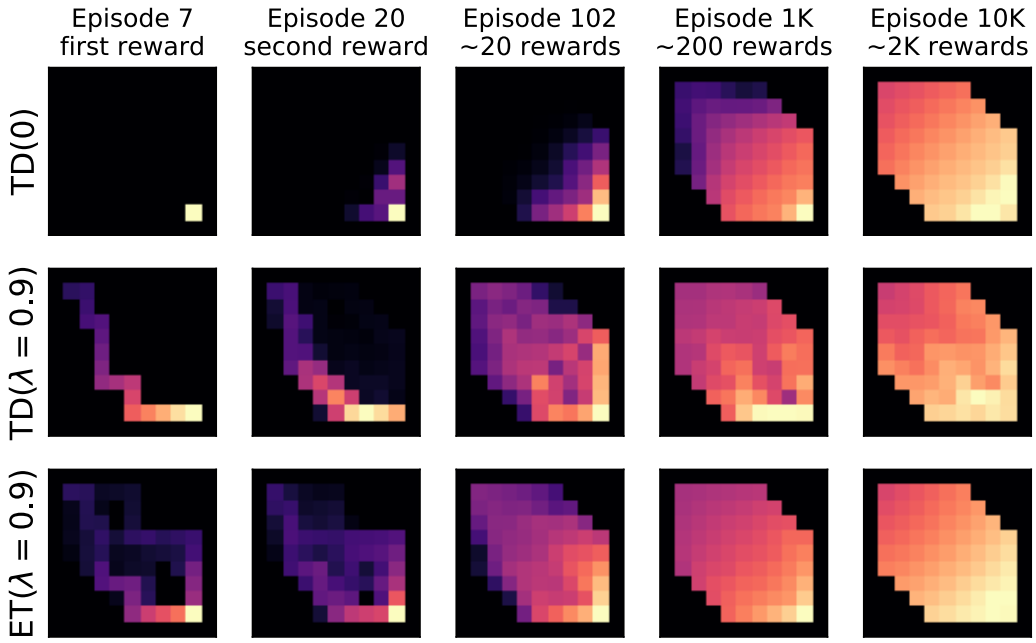


Figure 6.2: In the same setting as Figure 6.1, we show later value estimates after more rewards have been observed. TD(0) learns slowly but steadily, TD(λ) learns faster but with higher variance, and ET(λ) learns both fast and stable.

In this section we empirically investigate expected traces in several experiments. Whenever we refer to ET(λ), this is equivalent to ET(λ, 0).

6.4.1 An Open World

First consider the grid world depicted in Figure 6.1. The agent randomly moves right or down (excluding moves that would hit a wall), starting from the top-left corner. Any action in the bottom-right corner terminates the episode with $+1$ reward with probability 0.2, and 0 otherwise. All other rewards are 0.

Figure 6.1 shows the value estimates after the first positive reward, which occurred in the seventh episode. TD(0) updated a single state, TD(λ) updated earlier states in that episode, and ET(λ) additionally updated states from previous episodes. Figure 6.2 shows the values after the second reward, and after roughly 20, 200, and 2000 rewards (or 100, 1000, and $10,000$ episodes, respectively). ET(λ) converged faster than TD(0), which propagated information slowly, and than TD(λ), which had higher variance. All step sizes decayed as $\alpha = \beta = \sqrt{1/k}$, where k is the current episode number.

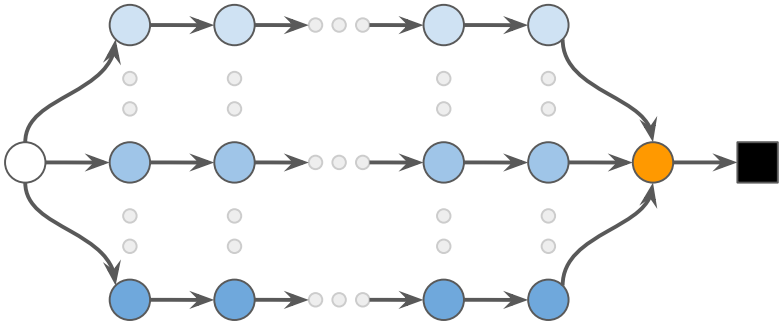


Figure 6.3: Multi-chain environment. Each episode starts in the left-most (white) state, and randomly transitions to one of m parallel (blue) chains of identical length n. After n steps, the agent always transitions to the same (orange) state, regardless of the chain it was in. The next step the episode terminates. Each reward is $+1$, except on termination when it either is $+1$ with probability 0.9 or -1 with probability 0.1.

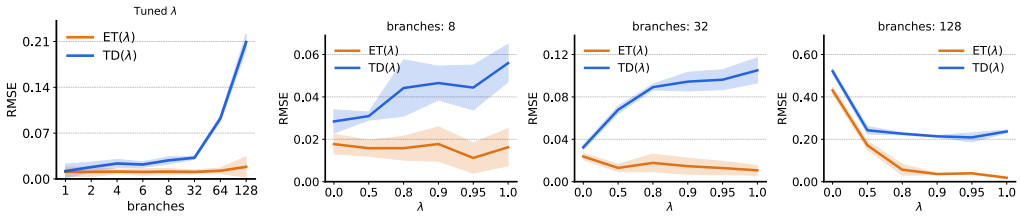


Figure 6.4: Prediction errors in the multi-chain. ET(λ) (**orange**) consistently outperformed TD(λ) (**blue**). Shaded areas depict standard errors across 10 seeds.

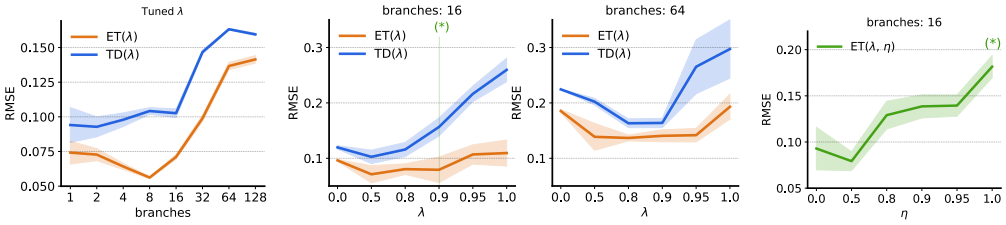


Figure 6.5: Comparing value error with linear function approximation a) as function of the number of branches (left), b) as function of λ (center two plots) and c) as function of η (right). The left three plots show comparisons of TD(λ) (**blue**) and ET(λ) (**orange**), showing ET(λ) attained lower prediction errors. The right plot interpolates between these algorithms via ET(λ, η), from ET(λ) = ET(λ, 0) to ET(λ, 1) = TD(λ), with $\lambda = 0.9$ (corresponding to a vertical slice indicated in the second plot).

6.4.2 A Multi-Chain

We now consider the multi-chain shown in Figure 6.3. We first compare TD(λ) and ET(λ) with tabular values on various variants of the multi-chain, corresponding $n = 4$ and $m \in \{1, 2, 4, 8, ..., 128\}$. The left-most plot in Figure 6.4 shows the average

root mean squared error (RMSE) of the value predictions after 1024 episodes. We ran 10 seeds for each combination of step size $1/t^d$ with $d \in \{0.5, 0.8, 0.9, 1\}$ and $\lambda \in \{0, 0.5, 0.8, 0.9, 0.95, 1\}$.

The left plot in Figure 6.4 shows value errors for different m, minimized over d and λ. The prediction error of TD(λ) (**blue**) grew quickly with the number of parallel chains. ET(λ) (**orange**) scaled better, because it updates values in multiple chains (from past episodes) upon receiving a surprising reward (e.g., -1) on termination. The other three plots in Figure 6.4 show value error as a function of λ for a subset of problems corresponding to $m \in \{8, 32, 128\}$. The dependence on λ differs across algorithms and problem instances; ET(λ) always achieved lower error than TD(λ).

To better understand the step-size sensitivity, we conduct a parameter study. Figure 6.6 contains a comparison of the performance of TD(λ) and ET(λ) across different step sizes. The data used for this figure is the same as used to generate the plots in Figure 6.4, but now we look explicitly at the effect of the step size parameter. We see that TD(0) performed best with a high step size, and that for high λ lower step sizes performed better—TD(0) with the highest step size ($\alpha_t = 1/\sqrt{n_t(S_t)}$) and TD(1) with the lowest step size ($\alpha_t = 1/n_t(S_t)$) both performed poorly. In contrast, ET(λ) here performed well for any combination of step size and trace parameter λ.

Next, we encode each state with a feature vector $\mathbf{x}(s)$ containing a binary indicator vector of the branch, a binary indicator of the progress along the chain, a bias that always equals one, and two binary features indicating when we are in the start (white) or bottleneck (orange) state. We extend the lengths of the chains to $n = 16$. Both TD(λ) and ET(λ) use a linear value function $V_{\mathbf{w}}(s) = \mathbf{w}^\top \mathbf{x}(s)$, and ET($\lambda$) uses a linear expected trace $z_{\boldsymbol{\Theta}}(s) = \boldsymbol{\Theta}\mathbf{x}(s)$. All updates use the same constant step size α. The left plot in Figure 6.5 shows the average root mean squared value error after 1024 episodes (averaged over 10 seeds). For each point the best constant step size $\alpha \in \{0.01, 0.03, 0.1\}$ (shared across all updates) and $\lambda \in \{0, 0.5, 0.8, 0.9, 0.95, 1\}$ is selected. ET(λ) (**orange**) attained lower errors across all values of m (left plot), and for all λ (center two plots, for two specific m). The right plot shows results

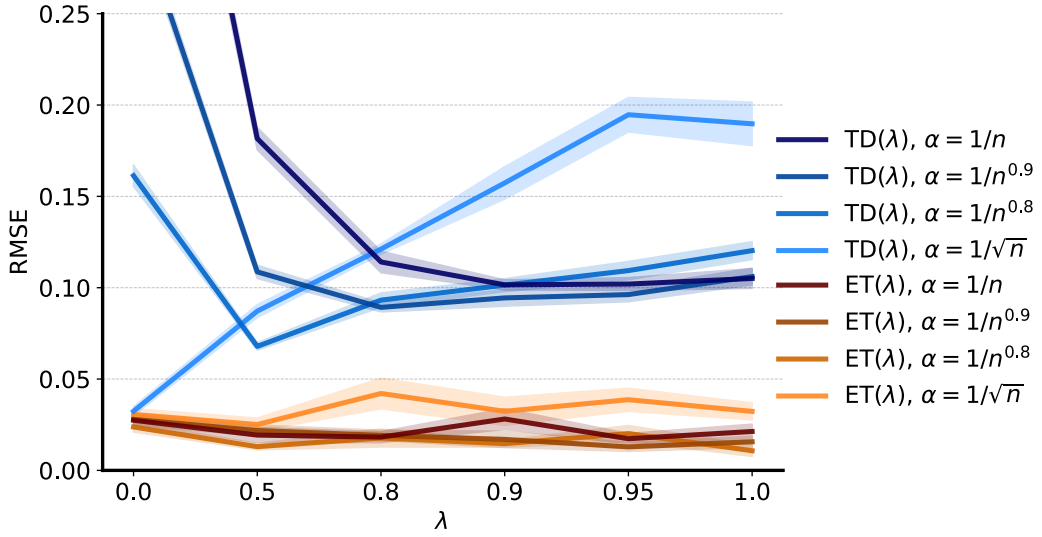


Figure 6.6: Comparison of prediction errors (lower is better) of TD(λ) and ET(λ) across different λs and different step sizes in the multi-chain world 6.3. The data underpinning these plots is the same as the data used for Figure 6.4, with 32 parallel chains. In all cases the step size was $\alpha = n_t(S_t)^d$, where $n_t(s) = \sum_{i=0}^{t} I(S_i = s)$ is the number of visits to state s in the first t time steps, and where d is a hyper-parameter. Note that the step size is *lower* when the exponent is *higher*.

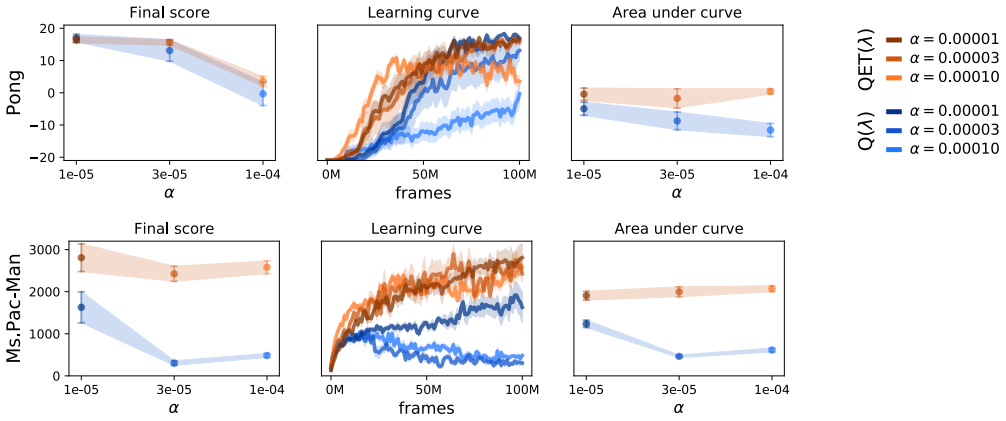


Figure 6.7: Performance of Q(λ) ($\eta = 1$, **blue**) and QET(λ) ($\eta = 0$, **orange**) on Pong and Ms.Pac-Man for various learning rates. Shaded regions show standard error across 10 random seeds. All results are for $\lambda = 0.95$.

for smooth interpolations via η, for $\lambda = 0.9$ and $m = 16$. The full expected trace ($\eta = 0$) performed well here, we expect in other settings the additional flexibility of η could be beneficial.

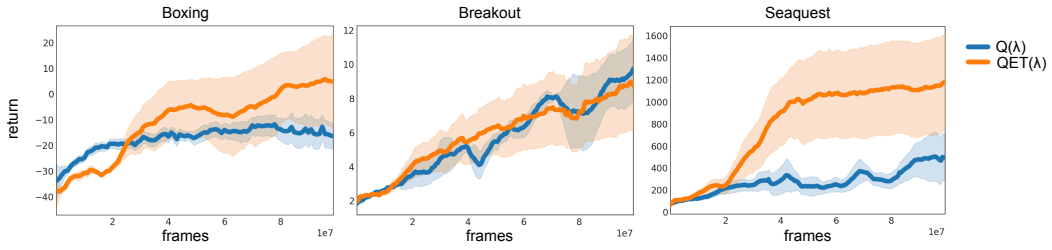


Figure 6.8: Learning curves of Q(λ) ($\eta = 1$, **blue**) and QET(λ) ($\eta = 0$, **orange**) on Boxing, Breakout and Seaquest. The Shaded regions show standard error across 10 random seeds. All results are for $\lambda = 0.95$, and the step size leading to the highest final score after 100M frames with QET(λ) was chosen for each game ($\alpha = 0.0001$ for Boxing and Seaquest and $\alpha = 0.00003$ for Breakout).

6.4.3 Expected Traces in Deep Reinforcement Learning

(Deep) neural networks are a common choice of function class in reinforcement learning [e.g., 171, 153, 154, 12, 121, 126, 160, 102, 51, 169, 135, 32, 54]. Eligibility traces are not very commonly combined with deep networks [but see 153, 34], perhaps in part because of the popularity of experience replay [82, 102, 58].

Perhaps the simplest way to extend expected traces to deep neural networks is to first separate the value function into a representation $\mathbf{x}(s)$ and a value $V_{(\mathbf{w},\boldsymbol{\xi})}(s) = \mathbf{w}^\top \mathbf{x}_{\boldsymbol{\xi}}(s)$, where $\mathbf{x}_{\boldsymbol{\xi}}$ is some (non-linear) function of the observations s.[4] We can then apply the same expected trace algorithm as used in the previous sections by learning a separate linear function $\boldsymbol{z}_{\boldsymbol{\Theta}}(s) = \boldsymbol{\Theta}\mathbf{x}(s)$ using the representation which is learned by backpropagating the value updates:

$$\begin{aligned} \boldsymbol{\xi}_{t+1} &= \boldsymbol{\xi}_t + \alpha\delta \boldsymbol{e}_t^{\boldsymbol{\xi}} \quad \text{and} \quad \mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta \boldsymbol{z}_{\boldsymbol{\Theta}}(S_t)\,, \\ \text{where} \quad \boldsymbol{e}_t^{\boldsymbol{\xi}} &= \gamma_t\lambda \boldsymbol{e}_{t-1}^{\boldsymbol{\xi}} + \nabla_{\boldsymbol{\xi}} V_{(\mathbf{w},\boldsymbol{\xi})}(S_t)\,, \\ \boldsymbol{e}_t^{\mathbf{w}} &= \gamma_t\lambda \boldsymbol{e}_{t-1}^{\mathbf{w}} + \nabla_{\mathbf{w}} V_{(\mathbf{w},\boldsymbol{\xi})}(S_t)\,, \end{aligned}$$

and then updating $\boldsymbol{\Theta}$ by minimising the sum of component-wise squared differences between $\boldsymbol{e}_t^{\mathbf{w}}$ and $\boldsymbol{z}_{\boldsymbol{\Theta}_t}(S_t)$.

To apply these idea to control settings, we use an implementation of online Q(λ) and its expected-trace variant QET(λ).

[4] Here s denotes observations to the agent, not a full environment state—s is not assumed to be Markovian.

Algorithm 3 Q(λ)

1: initialise $\mathbf{w}$
2: initialise $\boldsymbol{e} = \mathbf{0}$
3: observe initial state S
4: pick action $A \sim \pi(Q_{\mathbf{w}}(S))$
5: $V \leftarrow \max_a Q_{\mathbf{w}}(S,a)$
6: $\gamma = 0$
7: **repeat**
8: take action A, observe R, γ' and S' # $\gamma' = 0$ on a terminating transition
9: $V' \leftarrow \max_a Q_{\mathbf{w}}(S',a)$
10: $\delta \leftarrow R + \gamma v' - v$
11: $\boldsymbol{e} \leftarrow \gamma\lambda\boldsymbol{e} + \nabla_{\mathbf{w}} Q_{\mathbf{w}}(S,A)$
12: $\Delta\mathbf{w} \leftarrow \delta\boldsymbol{e} + (V - Q_{\mathbf{w}}(S,A))\nabla_{\mathbf{w}} Q_{\mathbf{w}}(S,A)$
13: $\Delta\mathbf{w} \leftarrow \text{transform}(\Delta\mathbf{w})$ # e.g., ADAM-ify
14: $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$
15: **until** done

6.4.4 Deep Q(λ)

We assume the typical setting [e.g., 102] where we have a neural network $\boldsymbol{Q}_{\mathbf{w}}$ that outputs $|A|$ numbers, such that $q(s,a) = Q_{\mathbf{w}}(s)[a]$. That is, we forward the observation s through network $Q_{\mathbf{w}}$ with weights $\mathbf{w}$ and $|A|$ outputs, and then select the a^{th} output to represent the value of taking action a.

Algorithm 3 then works as follows. For each transition, we first compute a telescoping TD error $\delta = r + \gamma' v' - v$ (line 5), where $\gamma' = 0$ on termination (and then S' is the first observation of the next episode) and, in our experiments, $\gamma' = 0.995$ otherwise. We update the trace $\boldsymbol{e}$ as usual (line 11), using accumulating traces. Note that the weights and, hence, trace will also have elements corresponding to the weights of actions that were not selected. The gradient with respect to those elements is considered to be zero, as is conventional.

Then, we compute a weight update $\Delta\mathbf{w} = \delta\boldsymbol{e} + (v - Q_{\mathbf{w}}(S,A))\nabla_{\mathbf{w}} Q_{\mathbf{w}}(S,A)$. The additional term corrects for the fact that our TD error is a telescoping error, and does not have the usual '$-q(s,a)$' term. This is akin to the Q(λ) algorithm proposed by Peng and Williams [115].

Finally, we transform the resulting update, using a transformation exactly like ADAM [68], but applied to the update $\Delta\mathbf{w}$ rather than a gradient. The hyper-

parameters were $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 0.0001$, and one of the step sizes as given in Figure 6.7. We then apply the resulting transformed update by adding it to the weights (line 14).

6.4.5 Deep QET(λ)

We now describe the expected-trace algorithm, shown in Algorithm 4, which was used for the Atari experiments. It is very similar to the Q(λ) algorithm described above, and in fact equivalent when we set $\eta = 1$.

The first main change is that we will split the computation of $q(s,a)$ into two separate parts, such that $Q_{(\mathbf{w},\boldsymbol{\xi})}(s,a) = \mathbf{w}_a^\top \mathbf{x}_{\boldsymbol{\xi}}(s)$. This is equivalent to the previous algorithm: we have just labeled separate subsets of parameters as $(\mathbf{w}, \boldsymbol{\xi})$ rather than merging all of them into a single vector $\mathbf{w}$, and we have labeled the last hidden layer as $\mathbf{x}(s)$. We keep separate traces for these subset (lines 11 and 12), but this is equivalent to keeping one big trace for the combined set.

This split in parameters helps avoid learning an expected trace for the full trace, which has millions of elements. Instead, we only learn expectations for traces corresponding to the last layer, denoted $\boldsymbol{e}^{\mathbf{w}}$. Importantly, the function $\boldsymbol{z}_{\boldsymbol{\theta}}(s,a)$ should condition on both state and action. This was implemented as a tensor $\boldsymbol{\theta} \in \mathbb{R}^{|A|\times|A|\times|\mathbf{x}|}$, such that its tensor multiplication with the features $\mathbf{x}(s)$ yields a $|A| \times |A|$ matrix $\boldsymbol{Z}$. Then, we interpret the vector $\boldsymbol{z}_a = [\boldsymbol{Z}]_a$ as the approximation to the expected trace $\mathbb{E}[\boldsymbol{e}_t \mid S_t = s, A_t = a]$, and update it accordingly, using a squared loss (and, again, ADAM-ifying the update before applying it to the parameters). The step size for the expected trace update was always $\beta = 0.1$ in our experiments, and the expected trace loss was not back-propagated into the feature representation. This can be done, but we leave any investigation of this for future work, as it would present a conflating factor for our experiments, because the expected trace update would then serve as an additional learning signal for the features that are also used for the value approximations.

Interesting challenges appear outside the fully linear case. First, the representation will itself be updated and will have its own trace $\boldsymbol{e}_t^{\boldsymbol{\xi}}$. Second, in the control case we optimise behaviour: the policy will change. Both these properties of the

Algorithm 4 QET(λ)

initialise $\mathbf{w}$, $\boldsymbol{\xi}$, $\boldsymbol{\theta}$
initialise $\boldsymbol{e} = \mathbf{0}$, $\mathbf{y} = \mathbf{0}$
observe initial state S
pick action $A \sim \pi(q(S))$
$V \leftarrow \max_a Q_{(\mathbf{w},\boldsymbol{\xi})}(S', a)$ # $Q_{(\mathbf{w},\boldsymbol{\xi})}(s,a) = \mathbf{w}_a^\top \mathbf{x}_{\boldsymbol{\xi}}(s)$, where $\mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_{|A|})$
$\gamma = 0$
repeat
    take action A, observe R, γ' and S' # $\gamma' = 0$ on any terminating transition
    $V' \leftarrow \max_a Q_{(\mathbf{w},\boldsymbol{\xi})}(S', a)$
    $\delta \leftarrow R + \gamma v' - v$
    $\boldsymbol{e}^{\mathbf{w}} \leftarrow \gamma\lambda \boldsymbol{y} + \nabla_{\mathbf{w}} Q_{(\mathbf{w},\boldsymbol{\xi})}(S,A)$
    $\boldsymbol{e}^{\boldsymbol{\xi}} \leftarrow \gamma\lambda \boldsymbol{e}^{\boldsymbol{\xi}} + \nabla_{\boldsymbol{\xi}} Q_{(\mathbf{w},\boldsymbol{\xi})}(S,A)$
    $\Delta\mathbf{w} \leftarrow \delta \boldsymbol{e}^{\mathbf{w}} + (V - Q_{(\mathbf{w},\boldsymbol{\xi})}(S,A)) \nabla_{\mathbf{w}} Q_{(\mathbf{w},\boldsymbol{\xi})}(S,A)$
    $\Delta\boldsymbol{\xi} \leftarrow \delta \boldsymbol{e}^{\boldsymbol{\xi}} + (V - Q_{(\mathbf{w},\boldsymbol{\xi})}(S,A)) \nabla_{\boldsymbol{\xi}} Q_{(\mathbf{w},\boldsymbol{\xi})}(S,A)$
    $\Delta\boldsymbol{\theta} \leftarrow \nabla_{\boldsymbol{\theta}} \| \boldsymbol{e}^{\boldsymbol{\xi}} - \boldsymbol{z}_{\boldsymbol{\theta}}(S,A) \|_2^2$
    $\Delta\mathbf{w} \leftarrow \text{transform}(\Delta\mathbf{w})$ # e.g., ADAM-ify
    $\Delta\boldsymbol{\xi} \leftarrow \text{transform}(\Delta\boldsymbol{\xi})$
    $\Delta\boldsymbol{\theta} \leftarrow \text{transform}(\Delta\boldsymbol{\theta})$
    $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$
    $\boldsymbol{\xi} \leftarrow \boldsymbol{\xi} + \Delta\boldsymbol{\xi}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
    $\boldsymbol{y} = (1 - \eta)\boldsymbol{z}_{\boldsymbol{\theta}}(s,a) + \eta \boldsymbol{e}^{\mathbf{w}}$
until done

non-linear control setting imply that the expected traces must track a non-stationary target. We found that being able to track this rather quickly improves performance: the expected trace parameters $\boldsymbol{\Theta}$ in the following experiment were updated with a step size of $\beta = 0.1$.

6.5 Experiments on Atari games

We compare online Q(λ) with QET(λ) on five Atari games. All the Atari experiments were run with the ALE [6], exactly as described in Mnih et al. [102], including using action repeats (4x), downsampling (to 84×84), and frame stacking. These experiments were conducted using Jax [16].

In all cases, we used ε-greedy exploration [cf. 146], with an ε that quickly decayed from 1 to 0.01 according to $\varepsilon_0 = 1$ and $\varepsilon_t = \varepsilon_{t-1} + 0.01(0.01 - \varepsilon_{t-1})$. Unlike Mnih et al. [102], we did not clip rewards, and we also did not apply any target nor-

malisation [cf. 163] or non-linear value transformations [119, 165]. We conjecture that such extensions could be beneficial for performance, but they are orthogonal to the main research questions investigated here and are therefore left for future work.

For the Atari experiments, we used the same preprocessing and network architecture as Mnih et al. [102], except that we used 128 channels in each convolutional layer because we ran experiments on TPUs (version 3.0, using a single core per experiment) which are most efficient when using tensors where one dimension is a multiple of 128. The experiments were written using JAX [17] and Haiku [53].

We tested this idea on five canonical Atari games: Pong and Ms. Pac-Man, Boxing, Breakout and Seaquest. The results in Figure 6.7 show that the expected traces helped speed up learning compared to the baseline which uses accumulating traces, for various step sizes. Unlike most prior work on this domain, which often relies on replay [102, 128, 58] or parallel streams of experience [103], these algorithms updated the values online from a single stream of experience. Further results in Figure 6.8 show how expected traces allow for faster or more stable learning compared to the baseline.

These experiments demonstrate that the idea of expected traces already extends to non-linear function approximation, such as deep neural networks. We consider this to be a rich area of further investigations. The results presented here are similar to earlier results [e.g., 102] and are not meant to compete with state-of-the-art performance results, which often depend on replay and much larger amounts of experience [e.g., 58].

6.6 Discussion and Extensions

We now discuss various interesting interpretations and relations, and discuss promising extensions.

6.6.1 Predecessor Features

For linear value functions the expected trace $z(s)$ can be expressed non recursively as follows:

$$\boldsymbol{z}(s) = \mathbb{E}\left[\sum_{n=0}^{\infty} \lambda_t^{(n)} \gamma_t^{(n)} \mathbf{x}_{t-n} \mid S_t = s\right], \tag{6.6}$$

where $\gamma_k^{(n)} \equiv \prod_{j=k-n}^{k} \gamma_j$. This is interestingly similar to the definition of the *successor features* [3]:

$$\boldsymbol{\psi}(s) = \mathbb{E}\left[\sum_{n=1}^{\infty} \gamma_t^{(n-1)} \mathbf{x}_{t+n} \mid S_t = s\right]. \tag{6.7}$$

The summation in (6.7) is over future features, while in (6.6) we have a sum over features already observed by the agent. This shows how linear expected traces act as *predecessor features*. A similar connection was made in the tabular setting by Pitis [118], relating source traces, which aim to estimate the source matrix $(I - \gamma P)^{-1}$, to successor representations [27]. In a sense, the above generalises this insight. In addition to being interesting in its own right, this connection allows for an intriguing interpretation of $\boldsymbol{z}(s)$ as a multidimensional value function. Like with successor features, the features $\mathbf{x}_t$ play the role of rewards, discounted with $\gamma \cdot \lambda$ rather than γ, and with time flowing backwards.

Although the predecessor interpretation only holds in the linear case, it is also of interest as a means to obtain a practical implementation of expected traces with non-linear function approximation, for instance applied only to the linear 'head' of a deep neural network. We used this 'predecessor feature trick' in our Atari experiments described earlier.

6.6.2 Relation to Model-Based Reinforcement Learning

Model-based reinforcement learning provides an alternative approach to efficient credit assignment. The general idea is to construct a model that estimates state-transition dynamics, and to update the value function based upon hypothetical transitions drawn from the model [144], for example by prioritised sweeping [106, 166].

In practice, model-based approaches have proven challenging in environments (such as Atari games) with rich perceptual observations, compared to model-free approaches that more directly update the agent's policy and predictions [164].

In some sense, expected traces also construct a model of the environment—but one that differs in several key regards from standard state-to-state models used in model-based reinforcement learning. First, expected traces estimate *past* quantities rather than *future* quantities. Second, they estimate the accumulation of gradients over a multi-step trajectory, rather than full transition dynamics, thereby focusing on those aspects that matter for the update. Third, they allow credit assignment across these potential past trajectories with a single update, without the iterative computation that is typically required when using a more explicit model. These differences may be important to side-step some of the challenges faced in model-based learning.

6.6.3 Batch Learning and Replay

We have mainly considered the online learning setting in this paper. It is often convenient to learn from batches of data, or replay transitions repeatedly, to enhance data efficiency. A natural extension is replay the experiences sequentially [e.g. 66], but perhaps alternatives exist. We now discuss one potential extension.

We defined a mixed trace $\boldsymbol{y}_t$ that mixes the instantaneous and expected traces. Optionally the expected trace $\boldsymbol{z}_t$ can be updated towards the mixed trace $\boldsymbol{y}_t$ as well, instead of towards the instantaneous trace $\boldsymbol{e}_t$. Analogously to TD(λ) we propose to then use at least one real step of data:

$$\Delta\boldsymbol{\theta}_t \equiv \beta \left(\boldsymbol{\nabla}_t + \gamma_t \lambda_t \boldsymbol{y}_{t-1} - \boldsymbol{z}_{\boldsymbol{\theta}}(S_t)\right)^\top \frac{\partial \boldsymbol{z}_{\boldsymbol{\theta}}(S_t)}{\partial \boldsymbol{\theta}}, \tag{6.8}$$

with $\boldsymbol{\nabla}_t \equiv \nabla_{\mathbf{w}} V_{\mathbf{w}}(S_t)$. This is akin to a forward-view λ-return update, with $\nabla_{\mathbf{w}} V_{\mathbf{w}}(S_t)$ in the role of (vector) reward, and $\boldsymbol{z}_{\boldsymbol{\theta}}$ of value, and discounted by $\lambda_t \gamma_t$, but reversed in time. In other words, this can be considered a sampled Bellman equation [10] but backward in time.

When we then choose $\eta = 0$, then $\boldsymbol{y}_{t-1} = z_{\boldsymbol{\theta}}(S_{t-1})$, and then the target in

(6.8) only depends on a single transition. Interestingly, that means we can then learn expected traces from *individual* transitions, sampled out of temporal order, for instance in batch settings or when using replay.

6.6.4 Application to Other Traces

We can apply the idea of expected trace to more traces than considered here. We can for instance consider the characteristic eligibility trace used in REINFORCE [173] and related policy-gradient algorithms [148].

Another appealing application is to the follow-on trace or *emphasis*, used in emphatic temporal difference learning [150] and related algorithms [e.g., 60]. Emphatic TD was proposed to correct an important issue with off-policy learning, which can be unstable and lead to diverging learning dynamics. Emphatic TD weights updates according to 1) the inherent interest in having accurate predictions in that state and, 2) the importance of predictions in that state for updating other predictions. Emphatic TD uses scalar 'follow-on' traces to determine the 'emphasis' for each update. However, this follow-on trace can have very high, even infinite, variance. Instead, we might estimate and use its expectation instead of the instantaneous emphasis. A related idea was explored by Zhang et al. [177] to obtain off-policy actor critic algorithms.

6.7 Conclusion

We have proposed a mechanism for efficient credit assignment, using the expectation of an eligibility trace. We have demonstrated this can sometimes speed up credit assignment greatly, and have analyzed concrete algorithms theoretically and empirically to increase understanding of the concept. The main findings of this work can be summarised as follows:

- The introduction of the concept of expected traces and the algorithm ET(λ) for counterfactual credit assignment.
- A theoretical analysis proving that ET(λ) results in unbiased updates with respect to traditional TD(λ) with possible lower variance.

- Empirical results demonstrating the benefit of using expected traces in some tabular and linear prediction tasks (faster learning).
- An extension of the concept of expected traces to control and nonlinear function approximation.
- A mechanism for combining expected traces with deep neural network that empirically demonstrates better performance in two Atari tasks.

These promising finding encourage further discussions. Expected traces have several interpretations. First, we can interpret the algorithm as counterfactually updating multiple possible trajectories leading up to the current state. Second, they can be understood as trading off bias and variance, which can be done smoothly via a unifying η parameter, between standard eligibility traces (low bias, high variance) and estimated traces (possibly higher bias, but lower variance). Furthermore, with tabular or linear function approximation we can interpret the resulting expected traces as predecessor states or features—object analogous to successor states or features, but time-reversed. These interpretations suggest that a variety of complementary ways to potentially extend these concepts and algorithms.

Chapter 7

General Conclusions

Designing artificial agents capable of identifying the key features in the observational space and to reason about the effect of states and actions on outcomes, has the potential to serve many aspects of human society. Data-efficient AI systems will ultimately generate a lower carbon footprint and will be more accessible than current systems due to the reduced costs of training and deploying them. Additionally, such data-efficient AI systems will be better suited for crucial problems in small-data domains such as healthcare and education in under-developed parts of the world. On this basis, this thesis studied the data-efficiency problem of RL and was set out to clarify the impact of representation learning in RL, to propose new approaches to design data-efficient RL agents. In particular, this thesis made progress towards demonstrating the claim that

> *data-efficiency in reinforcement learning can be achieved through structural representation learning and counterfactual credit assignment.*

In this final chapter, I summarise the work undertaken in this thesis that supports this claim. I conclude with general discussions, and present possible directions for future work that emerged from the findings in this thesis.

7.1 Summary of Contributions

This thesis first addressed the problem that the current RL community does not have clear answers to the questions of "What exactly is a good representation for

RL" and "How to learn such a representation?". With a deep overview of the different representation learning methods in use in RL, Chapter 3 demonstrated that prior works have made different design choices and have different, sometimes conflicting, understandings of what makes a good representation for RL. Four main representational properties that are deemed desirable were identified: from a review of the existing literature, a good representation either supports (1) policy evaluation, (2) policy search, (3) exploration, or (4) generalisation. Using this proposed categorisation, an extensive and critical discussion was carried out, identifying similarities, contradiction, limitation and pitfalls in the reviewed representation learning methods in RL. In particular, it was observed that a representation supporting policy evaluation can hinder policy search and that exploration and generalisation can be formulated as sub-goals of policy evaluation and policy search. These findings help shape the discussion around the importance of learning representation in RL, and identify key directions for future work, which are considered in the concluding remarks of Chapter 3.

Secondly, a new line of research for learning rich low dimensional state representations for value-base RL was proposed. Representation learning methods were proposed to improve the data-efficiency of RL algorithms in an innovative way. In Chapter 4, the common smoothness assumption of the value function on the graph of state was challenged: it was demonstrated that when this assumption does not hold, which happens frequently in real-world problems, the current state-of-the-art graph-based representation in RL Proto-Value Functions (PVFs) cannot approximate the value function in low dimension with satisfying accuracy. Consequently, other graph-based methods were applied to RL, and it was found that the algorithm node2vec [45] is well suited for RL. Indeed, Chapter 4 showed that node2vec efficiently generates low dimensional state representation vectors that capture the structural equivalence of the states while preserving the local properties of the environment without relying strongly on the smoothness assumption of the value function. These characteristics are shown to be key in accurately approximating value functions, and node2vec demonstrated improvements in low-dimensional value-

based RL with respect to the PVFs. Building on these findings, Chapter 5 introduces state2vec, a novel graph-inspired representation learning algorithm for RL, that overcomes the practical limitations of node2vec. State2vec efficiently learns a representation that exhibits the same characteristics as the SR, making it well suited for multi-task RL.

Lastly, this thesis addressed the data efficiency problem of RL from the credit assignment perspective. Chapter 6 introduced a novel and more efficient solution to the credit assignment problem. Where traditional RL methods assign credit to states and actions on a temporal basis, expected eligibility traces (Chapter 6) allow counterfactual credit assignment, such that credit is assigned to states or actions that could have led to the current state but did not do so this time. This is shown to be particularly helpful in stochastic environments, where the same behaviour is not guaranteed to produce the same reward. In such cases, the newly introduced expected traces enable the propagation of rare rewards faster.

7.2 Future works

The research presented in this thesis is a step towards achieving data-efficiency in RL, and opens up new interesting research avenues. This section highlights some future research directions that arose from insights gained during the development of this work.

7.2.1 Theoretical Framing of Optimal Representations in RL

The work carried out in this thesis shed light on the fact there is no globally accepted consensus on what makes a representation optimal in RL. Chapter 4 highlighted prior theoretical efforts demonstrating that under some assumptions, data-efficiency is achievable through low dimensional representations. However, the nature of such representation is yet to be defined and understood. There is an irrefutable need for a theoretical framing of optimal representations in RL. Such advancement will help us design learning systems for generating representations with optimal guarantees, hence unlocking more data-efficient RL.

7.2.2 Agents with Multiple Representations

Chapter 4 identified key representational properties for RL. It was discussed that a representation learning mechanism often targets a specific properties, sometimes at the expense of other desirable representational properties. It is remains unclear whether all desirable properties—to support policy evaluation and search, exploration, and generalisation—can be captured within a single representation. As a result, a relevant adjacent research question could be investigated: "Can agents learn from multiple representations?". Similarly to how control agents adapt their policy to trade-off between exploration and exploitation during training, we could imagine agents adapting their current representation of their world depending on which learning stage their are at. For instance, in the early stage of learning, the agent could speed up its exploration of the environment with a representation designed to support exploration. As more confidence is gained about the environment, the agent could utilise the data acquired during exploration to fine-tune a representation that is more suited for faster control or generalisation. Agents able to quickly change their representation of the world to suit their current need would have the potential to greatly improve learning efficiency.

7.2.3 Graphs and RL

This thesis demonstrated initial promising benefits of using graphs to learn a structural representation of the environment to speed up the solving of RL problems. In particular, Chapter 4 validated the importance of representation capturing structural equivalences as well as local properties in the state space for efficient RL, and Chapter 5 developed an efficient graph-inspired representation learning algorithm for RL that exhibit the aforementioned characteristics. These findings suggest that the potential of graph-based, or graph-inspired, methods in RL should be further explored.

For example, an interesting future research direction would be to investigate ways of incorporating graph-based methods into existing deep RL systems. Augmenting traditional RL algorithms, which operate in the temporal domain, with graph-based methods, which give information about the spectrum of the state and

action space, could potentially unlock novel, more efficient, RL agents.

Graph tools could also be considered in the credit assignment problem. For instance, spectral analysis on graphs could be used to identify key states (such as doorways or highly connected hubs), and use this information to assign credit based on structural characteristics. Additionally, alternative graphs could be inferred from data and used to propagate reward information. For instance, instead of relying on transitions to connect states in the graphs, state features similarity could be used: states that are close in the observation space would be highly connected in the graph. In this way, information could be propagated quickly among similar states, even if they are distant under the transition dynamics. This would have the potential of going beyond traditional temporal credit assignment, expanding the counterfactual credit assignment idea explored in Chapter 6.

Appendix A

Appendix

A.1 Proof of Lemma 1

We start with a formal definition of a Markov state:

Definition 8. ***Markov property****: we say a state s is Markov if*

$$p(R_{t+1}, S_{t+1}|A_t, S_t, R_{t-1}, A_{t-1}, S_{t-1}...) = p(R_{t+1}, S_{t+1}|A_t, S_t).$$

Next, we show that a Markov state implies a similar property for the transition probabilities induced by a policy π:

Property 1. *Let p_π be the transition probabilities induced by policy π. Then, if s is Markov, we have that*

$$p_\pi(R_{t+1}, S_{t+1}|S_t, R_{t-1}, A_{t-1}, S_{t-1}...) = p_\pi(R_{t+1}, S_{t+1}|S_t).$$

Proof.

$$p_\pi(R_{t+1}, S_{t+1}|S_t, R_{t-1}, A_{t-1}, S_{t-1}...) = \int_a \pi(a|S_t) p(S_{t+1}, R_{t+1}|A_t = a, S_t, R_{t-1}, A_{t-1}, S_{t-1}...) da \tag{A.1}$$

$$= \int_a \pi(a|S_t) p(S_{t+1}, R_{t+1}|A_t = a, S_t) da.$$

(Markov property)

□

Using the above, we can prove Lemma 1:

Lemma 1. *If s is Markov, then $\mathbb{E}[\,\delta_t \boldsymbol{e}_t \mid S_t = s] = \mathbb{E}[\,\delta_t \mid S_t = s]\,\mathbb{E}[\,\boldsymbol{e}_t \mid S_t = s]$.*

Proof. First note that the expectations above are with respect to the transition probabilities p_π as defined in (A.1). That noted, the result trivially follows from the fact that, when s is Markov, the two random variables δ_t and $\boldsymbol{e}_t$ are independent conditioned on S_t. To see why this is so, note that δ_t is defined as

$$\delta_t = R_{t+1} + \gamma_{t+1} V_{\mathbf{w}}(S_{t+1}) - V_{\mathbf{w}}(S_t). \tag{A.2}$$

Since we are conditioning on the event that $S_t = s$, the only two random quantities in the definition of δ_t are R_{t+1} and S_{t+1}. Thus, because s is Markov, we have that

$$p_\pi(\delta_t | S_t, R_t, S_{t-1}, R_{t-1}, ...) = p_\pi(\delta_t | S_t), \tag{Property 1}$$

that is, S_t fully defines the distribution of δ_t. This means that $p_\pi(\delta_t | S_t, X_{t'}) = p_\pi(\delta_t | S_t)$ for any $t' \leq t$, where $X_{t'}$ is a random variable that only depends on events that occurred up to time t'. Replacing $X_{t'}$ with $\boldsymbol{e}_t$, we have that $p_\pi(\delta_t | S_t, \boldsymbol{e}_t) = p_\pi(\delta_t | S_t)$, which implies that δ_t and $\boldsymbol{e}_t$ are independent conditioned on S_t. □

A.2 Proof of Proposition 2

Proposition 2. *Let $\boldsymbol{e}_t$ an instantaneous trace vector. Then let $\mathbf{z}_t(s)$ be the empirical mean $\mathbf{z}_t(s) = \frac{1}{n_t(s)} \sum_i^{n_t(s)} \boldsymbol{e}_{t_i^s}$, where t_i^s-s denote past times when we have been in state s, that is $S_{t_i^s} = s$, and $n_t(s)$ is the number of visits to s in the first t steps. Consider the expected trace algorithm $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t \delta_t \mathbf{z}_t$. If S_t is Markov, the expectation of this update is equal to the expected update with instantaneous traces $\boldsymbol{e}_t$, while attaining a potentially lower variance:*

$$\mathbb{E}[\,\alpha_t \delta_t \mathbf{z}_t(S_t) \mid S_t\,] = \mathbb{E}[\,\alpha_t \delta_t \boldsymbol{e}_t \mid S_t\,] \qquad \textit{and}$$
$$\mathbb{V}[\alpha_t \delta_t \mathbf{z}_t(S_t) \mid S_t] \leq \mathbb{V}[\alpha_t \delta_t \boldsymbol{e}_t \mid S_t]\,,$$

where the second inequality holds component-wise. The inequality is strict when $\mathbb{V}[\boldsymbol{e}_t \mid S_t] > 0$.

Proof. We have

$$\begin{aligned}
\mathbb{E}[\,\alpha_t\delta_t\boldsymbol{e}_t \mid S_t = s] &= \mathbb{E}[\,\alpha_t\delta_t \mid S_t = s]\,\mathbb{E}[\,\boldsymbol{e}_t \mid S_t = s] && \text{(as } s \text{ is Markov)}\\
&= \mathbb{E}[\,\alpha_t\delta_t \mid S_t = s]\,\mathbb{E}[\,\boldsymbol{z}_t \mid S_t = s] && \text{(as } \boldsymbol{z}_t = \tfrac{1}{n}\textstyle\sum_i^n \boldsymbol{e}_{t_i^s}\text{)}\\
&= \mathbb{E}[\,\alpha_t\delta_t\boldsymbol{z}_t \mid S_t = s]\,. && \square
\end{aligned}$$

Now let us look at the conditional variance for each of the dimension of the update vector $\alpha_t\delta_t\boldsymbol{z}_t$: $\mathbb{V}[\alpha_t\delta_t\boldsymbol{z}_{t,i} \mid S_t = s]$, where $\boldsymbol{z}_{t,i}$ denotes the i-th component of vector $\boldsymbol{z}_t$.

$$\begin{aligned}
&\mathbb{V}[\alpha_t\delta_t\boldsymbol{z}_{t,i} \mid S_t = s]\\
&= \mathbb{E}\left[\,(\alpha_t\delta_t\boldsymbol{z}_{t,i})^2 \mid S_t = s\right] - \mathbb{E}[\,\alpha_t\delta_t\boldsymbol{z}_{t,i} \mid S_t = s]^2\\
&= \mathbb{E}\left[\,\alpha_t^2\delta_t^2(\boldsymbol{z}_{t,i})^2 \mid S_t = s\right] - \mathbb{E}[\,\alpha_t\delta_t \mid S_t = s]^2\,\mathbb{E}[\,\boldsymbol{z}_{t,i} \mid S_t = s]^2\\
&= \mathbb{E}\left[\,\alpha_t^2\delta_t^2 \mid S_t = s\right]\mathbb{E}\left[\,(\boldsymbol{z}_{t,i})^2 \mid S_t = s\right] - \mathbb{E}[\,\alpha_t\delta_t \mid S_t = s]^2\,\mathbb{E}[\,\boldsymbol{z}_{t,i} \mid S_t = s]^2
\end{aligned}$$

By a similar argument, we have

$$\begin{aligned}
&\mathbb{V}[\alpha_t\delta_t\boldsymbol{e}_{t,i} \mid S_t = s]\\
&= \mathbb{E}\left[\,\alpha_t^2\delta_t^2 \mid S_t = s\right]\mathbb{E}\left[\,(\boldsymbol{e}_{t,i})^2 \mid S_t = s\right] - \mathbb{E}[\,\alpha_t\delta_t \mid S_t = s]^2\,\mathbb{E}[\,\boldsymbol{e}_{t,i} \mid S_t = s]^2
\end{aligned}$$

Now, we also know that $\mathbb{E}[\,\boldsymbol{z}_t \mid S_t = s] = \mathbb{E}[\,\boldsymbol{e}_t \mid S_t = s] = \mu_t$, as $\boldsymbol{z}_t$ is the empirical mean of $\boldsymbol{e}_t$. Thus we also have, component-wise,

$$\mathbb{E}[\,\boldsymbol{z}_{t,i} \mid S_t = s] = \mathbb{E}[\,\boldsymbol{e}_{t,i} \mid S_t = s] = \mu_{t,i}$$

Moreover, from the same reason we have that $\mathbb{V}(\boldsymbol{z}_{t,i}|S_t = s) = \frac{1}{n_s}\mathbb{V}(\boldsymbol{e}_{t,i}|S_t = s)$. Thus we obtain:

$$\mathbb{V}[\alpha_t\delta_t\boldsymbol{z}_{t,i} \mid S_t = s] \;=\; \mathbb{E}\left[\,\alpha_t^2\delta_t^2 \mid S_t = s\right]\mathbb{E}\left[\,\boldsymbol{z}_{t,i}(\boldsymbol{z}_{t,i})^T \mid S_t = s\right] - \mathbb{E}[\,\alpha_t\delta_t \mid S_t = s]^2\,\mu_{t,i}^2$$

Thus:

$$\begin{aligned}&\mathbb{V}[\alpha_t\delta_t \boldsymbol{z}_{t,i} \mid S_t = s] - \mathbb{V}[\alpha_t\delta_t \boldsymbol{e}_{t,i} \mid S_t = s] \\ &= \mathbb{E}\left[\alpha_t^2\delta_t^2 \mid S_t = s\right] \underbrace{\left(\mathbb{E}\left[\boldsymbol{z}_{t,i}(\boldsymbol{z}_{t,i})^T \mid S_t = s\right] - \mathbb{E}\left[\boldsymbol{e}_{t,i}(\boldsymbol{e}_{t,i})^T \mid S_t = s\right]\right)}_{\leq 0, \text{ from definition of } \boldsymbol{z}_{t,i}} \;\leq\; 0\,,\end{aligned}$$

with equality holding, if and only if:

i $\mathbb{E}\left[(\boldsymbol{z}_{t,i})^2 \mid S_t = s\right] = \mathbb{E}\left[(\boldsymbol{e}_{t,i})^2 \mid S_t = s\right] \Rightarrow \mathbb{V}(\boldsymbol{z}_{t,i}|S_t = s) = \mathbb{V}(\boldsymbol{e}_{t,i}|S_t = s)$, but $\mathbb{V}(\boldsymbol{z}_{t,i}|S_t = s) = \frac{1}{n_s}\mathbb{V}(\boldsymbol{e}_{t,i}|S_t = s)$ by definition of $\boldsymbol{z}_{t,i}$ as the running mean on samples $\boldsymbol{e}_{t,i}$. This can only happen for $n_s = 1$, or in the absence of stochasticity, for every state s. Thus, in the most general case, this implies $\mathbb{V}(\boldsymbol{z}_{t,i}|S_t = s) = \mathbb{V}(\boldsymbol{e}_{t,i}|S_t = s) = 0$; or

ii $\mathbb{E}\left[\alpha_t^2\delta_t^2 \mid S_t = s\right] = 0 \Rightarrow \delta_t = 0$

Thus, we have equality only with we have exactly one sample for the average $\boldsymbol{z}_t$ so far, or only one sample is needed (thus $\boldsymbol{z}_t$ and $\boldsymbol{e}_t$ are not actual random variables and there is only one deterministic path to s); or when the TD errors are zero for all transitions following s. □

A.3 Properties of Mixture Traces

In this section we explore and proof some of the properties of the proposed mixture trace, defined in Equation (6.3) in the main text and repeated here:

$$\boldsymbol{y}_t = (1-\eta)\boldsymbol{z}_{\boldsymbol{\theta}}(S_t) + \eta\left(\gamma_t\lambda_t\boldsymbol{y}_{t-1} + \nabla_{\mathbf{w}}V_{\mathbf{w}}(S_t)\right). \tag{6.3}$$

The proofs, in this section we will use the notation $\mathbf{x}_t$ to denote the features used in a linear approximation for the value function(s) constructed. Just note that this term can be substituted, in general, by the gradient term $\nabla_{\mathbf{w}}V_{\mathbf{w}}(S_t)$ in the equation above.

Proposition 3. *The mixture trace* $\boldsymbol{y}_t$ *defined in* (6.3) *can be written as* $\boldsymbol{y}_t = \mu\boldsymbol{y}_{t-1} + \mathbf{x}_t$ *with decay parameter* $\mu = \eta\gamma\lambda$ *and signal* $\mathbf{x}_t = (1-\eta)\boldsymbol{z}_{\boldsymbol{\theta}}(S_t) + \eta\ \nabla_{\mathbf{w}}V_{\mathbf{w}}(S_t)$,

such that

$$\boldsymbol{y}_t = \sum_{k=0}^{t} (\eta\gamma\lambda)^k \left[(1-\eta)\boldsymbol{z}_{\boldsymbol{\theta}}(S_{t-k}) + \eta\ \nabla_{\mathbf{w}} V_{\mathbf{w}}(S_{t-k})\right]. \tag{6.5}$$

Proof. As mentioned before, under a linear parameterization $\nabla_{\mathbf{w}} V_{\mathbf{w}}(S_t) = \mathbf{x}(S_t) := \mathbf{x}_t$ Let us start with the definition of the mixture trace $\boldsymbol{y}_t$:

$$\begin{aligned}
\boldsymbol{y}_t &= (1-\eta)\boldsymbol{z}_t + \eta(\gamma_t\lambda_t\boldsymbol{y}_{t-1} + \mathbf{x}_t) \\
&= [(1-\eta)\boldsymbol{z}_t + \eta\mathbf{x}_t] + \eta\gamma_t\lambda_t\boldsymbol{y}_{t-1} \\
&= [(1-\eta)\boldsymbol{z}_t + \eta\mathbf{x}_t] + \eta\gamma_t\lambda_t\left[(1-\eta)\boldsymbol{z}_{t-1} + \eta\mathbf{x}_{t-1}\right] + \eta^2\gamma_t\lambda_t\gamma_{t-1}\lambda_{t-1}\boldsymbol{y}_{t-2} \\
&= (1-\eta)\left[\boldsymbol{z}_t + \eta\gamma_t\lambda_t\boldsymbol{z}_{t-1} + \eta^2\gamma_t\lambda_t\gamma_{t-1}\lambda_{t-1}\boldsymbol{z}_{t-2} + \cdots\right] + \\
&\quad + \eta\left[\mathbf{x}_t + \eta\gamma_t\lambda_t\mathbf{x}_{t-1} + \eta^2\gamma_t\lambda_t\gamma_{t-1}\lambda_{t-1}\mathbf{x}_{t-2} + \cdots\right] \\
&= (1-\eta)\sum_{k=0}^{t}(\eta\gamma\lambda)^k\boldsymbol{z}_{t-k} + \eta\sum_{k=0}^{t}(\eta\gamma\lambda)^k\mathbf{x}_{t-k} \\
&= \sum_{k=0}^{t}(\eta\gamma\lambda)^k\left[(1-\eta)\boldsymbol{z}_{t-k} + \eta\mathbf{x}_{t-k}\right]
\end{aligned}$$

Substituting $\mathbf{x}_t$ in the above derivation by $\nabla_{\mathbf{w}} V_{\mathbf{w}}(S_t)$ leads to (6.3). □

Proposition 4. *When using approximations $z_{\boldsymbol{\Theta}}(s) = \boldsymbol{\Theta}\mathbf{x}(s)$ and $V_{\mathbf{w}}(s) = \mathbf{w}^\top\mathbf{x}(s)$ then, if $(1-\eta)\boldsymbol{\Theta} + \eta\mathbb{I}$ is non-singular, ET(λ, η) has the same fixed point as TD($\lambda\,\eta$).*

Proof. By Proposition 3 we have that $\boldsymbol{y}_t$ can be re-written as:

$$\begin{aligned}
\boldsymbol{y}_t &= \sum_{k=0}^{t}(\eta\gamma\lambda)^k\left[(1-\eta)\boldsymbol{z}_{\boldsymbol{\theta}}(S_{t-k}) + \eta\mathbf{x}(S_{t-k})\right] \\
&= \sum_{k=0}^{t}(\eta\gamma\lambda)^k\left[(1-\eta)\boldsymbol{\Theta}\mathbf{x}(S_{t-k}) + \eta\mathbf{x}(S_{t-k})\right] &\text{(A.3)} \\
&= \left[(1-\eta)\boldsymbol{\Theta} + \eta\mathbb{I}\right]\underbrace{\sum_{k=0}^{t}(\eta\gamma\lambda)^k\mathbf{x}(S_{t-k})}_{\text{instantaneous trace } \boldsymbol{e}_t^{\lambda\eta}}. &\text{(A.4)}
\end{aligned}$$

We examine the fixed point $\mathbf{w}_*$ of the algorithm using this approximation of the

expected trace:

$$\begin{aligned}\mathbb{E}[\,\delta_t \boldsymbol{y}_t\,] &= \mathbb{E}\left[\boldsymbol{y}_t(R_{t+1} + \gamma \mathbf{x}(S_{t+1})^\top \mathbf{w}_* - \mathbf{x}(S_t)^\top \mathbf{w}_*)\right] \\ &= \mathbf{0}\,.\end{aligned}$$

This implies the fixed point is

$$\mathbf{w}_* = \mathbb{E}\left[\boldsymbol{y}_t(\gamma \mathbf{x}(S_{t+1}) - \mathbf{x}(S_t))^\top\right]^{-1} \mathbb{E}[\,\boldsymbol{y}_t R_{t+1}\,]\,.$$

Now, plugging in the relation in (A.4) above, we get:

$$\begin{aligned}\mathbf{w}_* &= \mathbb{E}\left[[(1-\eta)\boldsymbol{\Theta} + \eta\mathbb{I}]\,\boldsymbol{e}_t^{\lambda\eta}(\gamma \mathbf{x}(S_{t+1}) - \mathbf{x}(S_t))^\top\right]^{-1} \mathbb{E}\left[[(1-\eta)\boldsymbol{\Theta} + \eta\mathbb{I}]\,\boldsymbol{e}_t^{\lambda\eta} R_{t+1}\right] \\ &= \mathbb{E}\left[\boldsymbol{e}_t^{\lambda\eta}(\gamma \mathbf{x}(S_{t+1}) - \mathbf{x}(S_t))^\top\right]^{-1} [(1-\eta)\boldsymbol{\Theta} + \eta\mathbb{I}]^{-1}\,[(1-\eta)\boldsymbol{\Theta} + \eta\mathbb{I}]\,\mathbb{E}\left[\boldsymbol{e}_t^{\lambda\eta} R_{t+1}\right] \\ &= \mathbb{E}\left[\boldsymbol{e}_t^{\lambda\eta}(\gamma \mathbf{x}(S_{t+1}) - \mathbf{x}(S_t))^\top\right]^{-1} \mathbb{E}\left[\boldsymbol{e}_t^{\lambda\eta} R_{t+1}\right]\,.\end{aligned}$$

This last term is the fixed point for TD($\lambda\,\eta$). □

Moreover, it is worth noting that the above equality recovers, for the extreme values of η:

- $\eta = 1 \Rightarrow \boldsymbol{y}_t = \sum_{k=0}^{t} (\gamma\lambda)^k \mathbf{x}_{t-k}$ (instantaneous trace for TD(λ))
- $\eta = 0 \Rightarrow \boldsymbol{y}_t = \sum_{k=0}^{t} (\eta\,\gamma\lambda)^k \boldsymbol{z}_{t-k} = \boldsymbol{z}_t$ (expected trace for TD(λ))

Moreover, as the extreme values already suggest, the expected update of the mixture traces follows the TD(λ) learning, in expectation, for all the intermediate values $\eta \in (0,1)$ as well, trading off variance of estimates as η approaches 0.

Proposition 5. *Let $\boldsymbol{e}_t^\lambda$ be a λ trace vector. Let $\boldsymbol{y}_t = (1-\eta)\boldsymbol{z}_t + \eta(\gamma\lambda \boldsymbol{y}_{t-1} + \mathbf{x}_t)$ (as defined in* (6.3)*). Consider the ET(λ, η) algorithm $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t \delta_t \boldsymbol{y}_t$. For all Markov states s the expectation of this update is equal to the expected update with instantaneous traces $\boldsymbol{e}_t^\lambda$:*

$$\mathbb{E}[\,\alpha_t \delta_t \boldsymbol{y}(S_t) | S_t = s] = \mathbb{E}\left[\alpha_t \delta_t \boldsymbol{e}_t^\lambda | S_t = s\right]\,,$$

for every $\eta \in [0,1]$ *and any* $\lambda \in [0,1]$.

Proof. Let us revisit Eq. 6.5 in Proposition 3:

$$\begin{aligned}
\mathbb{E}[\boldsymbol{y}_t] &= \mathbb{E}\left[\sum_{k=0}^{t}(\eta\gamma\lambda)^k[(1-\eta)\boldsymbol{z}_{t-k}+\eta\mathbf{x}_{t-k}]\right] \\
&= \mathbb{E}\left[\sum_{k=0}^{t}(\eta\gamma\lambda)^k[(1-\eta)\mathbb{E}[(\mathbf{x}_{t-k}+\gamma\lambda\boldsymbol{z}_{t-k-1})]+\eta\mathbf{x}_{t-k}]\right] \\
&= \mathbb{E}\left[\sum_{k=0}^{t}(\eta\gamma\lambda)^k\mathbf{x}_{t-k}+(1-\eta)\gamma\lambda\sum_{k=0}^{t-1}(\eta\gamma\lambda)^k\underbrace{\boldsymbol{z}_{t-k-1}}_{\mathbb{E}[(\mathbf{x}_{t-k-1}+\gamma\lambda\boldsymbol{z}_{t-k-2})]}\right] \\
&= \mathbb{E}\left[\sum_{k=0}^{t}(\eta\gamma\lambda)^k\mathbf{x}_{t-k}+(1-\eta)\gamma\lambda\sum_{k=0}^{t-1}(\eta\gamma\lambda)^k\mathbf{x}_{t-k-1}+(1-\eta)(\gamma\lambda)^2\sum_{k=0}^{t-2}(\eta\gamma\lambda)^k\underbrace{\boldsymbol{z}_{t-k-2}}_{\mathbb{E}[\mathbf{x}_{t-k-2}+\gamma\lambda\boldsymbol{z}_{t-k-3}]}\right] \\
&= \mathbb{E}\left[\sum_{k=0}^{t}(\eta\gamma\lambda)^k\mathbf{x}_{t-k}+(1-\eta)\sum_{i=1}^{t-1}(\gamma\lambda)^i\sum_{k=0}^{t-i}(\eta\gamma\lambda)^k\mathbf{x}_{t-k-i}\right]
\end{aligned}$$

Now, re-writing the sum, gathering all the weighting for each feature $\mathbf{x}_{t-k-i}$ we get:

$$\begin{aligned}
\mathbb{E}[\boldsymbol{y}_t] &= \mathbb{E}\left[\sum_{k=0}^{t}(\eta\gamma\lambda)^k\mathbf{x}_{t-k}+(1-\eta)\sum_{i=1}^{t-1}(\gamma\lambda)^i\sum_{k=0}^{t-i}(\eta\gamma\lambda)^k\mathbf{x}_{t-k-i}\right] \\
&= \mathbb{E}\left[\mathbf{x}_t+\sum_{k=1}^{t}\mathbf{x}_{t-k}\left((\eta\gamma\lambda)^k+(1-\eta)\sum_{i=1}^{k}(\gamma\lambda)^i\cdot(\gamma\lambda\eta)^{k-i}\right)\right] \\
&= \mathbb{E}\left[\mathbf{x}_t+\sum_{k=1}^{t}\mathbf{x}_{t-k}(\gamma\lambda)^k\left(\eta^k+(1-\eta)\sum_{i=1}^{k}\eta^{k-i}\right)\right] \\
&= \mathbb{E}\left[\mathbf{x}_t+\sum_{k=1}^{t}\mathbf{x}_{t-k}(\gamma\lambda)^k\left(\eta^k+(1-\eta)\frac{1-\eta^k}{(1-\eta)}\right)\right] \\
&= \mathbb{E}\left[\mathbf{x}_t+\sum_{k=1}^{t}\mathbf{x}_{t-k}(\gamma\lambda)^k\right] \\
&= \mathbb{E}\left[\sum_{k=0}^{t}(\gamma\lambda)^k\mathbf{x}_{t-k}\right]
\end{aligned}$$

Thus $\mathbb{E}[\boldsymbol{y}_t] = \mathbb{E}\left[\sum_{k=0}^{t}(\gamma\lambda)^k\mathbf{x}_{t-k}\right] = \mathbb{E}\left[\boldsymbol{e}_t^{\lambda}\right]$, where $\boldsymbol{e}_t^{\lambda}$ is the instantaneous λ trace on feature space $\mathbf{x}$. Thus $\mathbb{E}[\boldsymbol{y}(s)] = z_*^{\lambda}(s) = \mathbb{E}\left[\boldsymbol{e}_t^{\lambda}\right]$. Finally we can plug-in this result in the

expected update:

$$\begin{aligned}
\mathbb{E}[\alpha_t \delta_t \boldsymbol{y}(S_t)|S_t = s] &= \mathbb{E}[\alpha_t \delta_t|S_t = s]\,\mathbb{E}[\boldsymbol{y}(S_t)|S_t = s] \\
&= \mathbb{E}[\alpha_t \delta_t|S_t = s]\, z_*^{\lambda}(s) \\
&= \mathbb{E}[\alpha_t \delta_t|S_t = s]\,\mathbb{E}\left[\boldsymbol{e}_t^{\lambda}|S_t = s\right] \\
&= \mathbb{E}\left[\alpha_t \delta_t \boldsymbol{e}_t^{\lambda}|S_t = s\right].
\end{aligned}$$

□

Finally, please note that in this proposition and its proof we drop the time indices t for λ and γ parameters in the definition of $\boldsymbol{y}_t$. This is purely to ease the notation and promote compactness in the derivation

A.4 Visualising PVFs and Node2vec

Figures A.2, A.2, A.4, A.4, and A.5, A.6 depict different embedding dimensions of PVFs and node2vec in the four-room, low stretch tree and 1D torus domains.

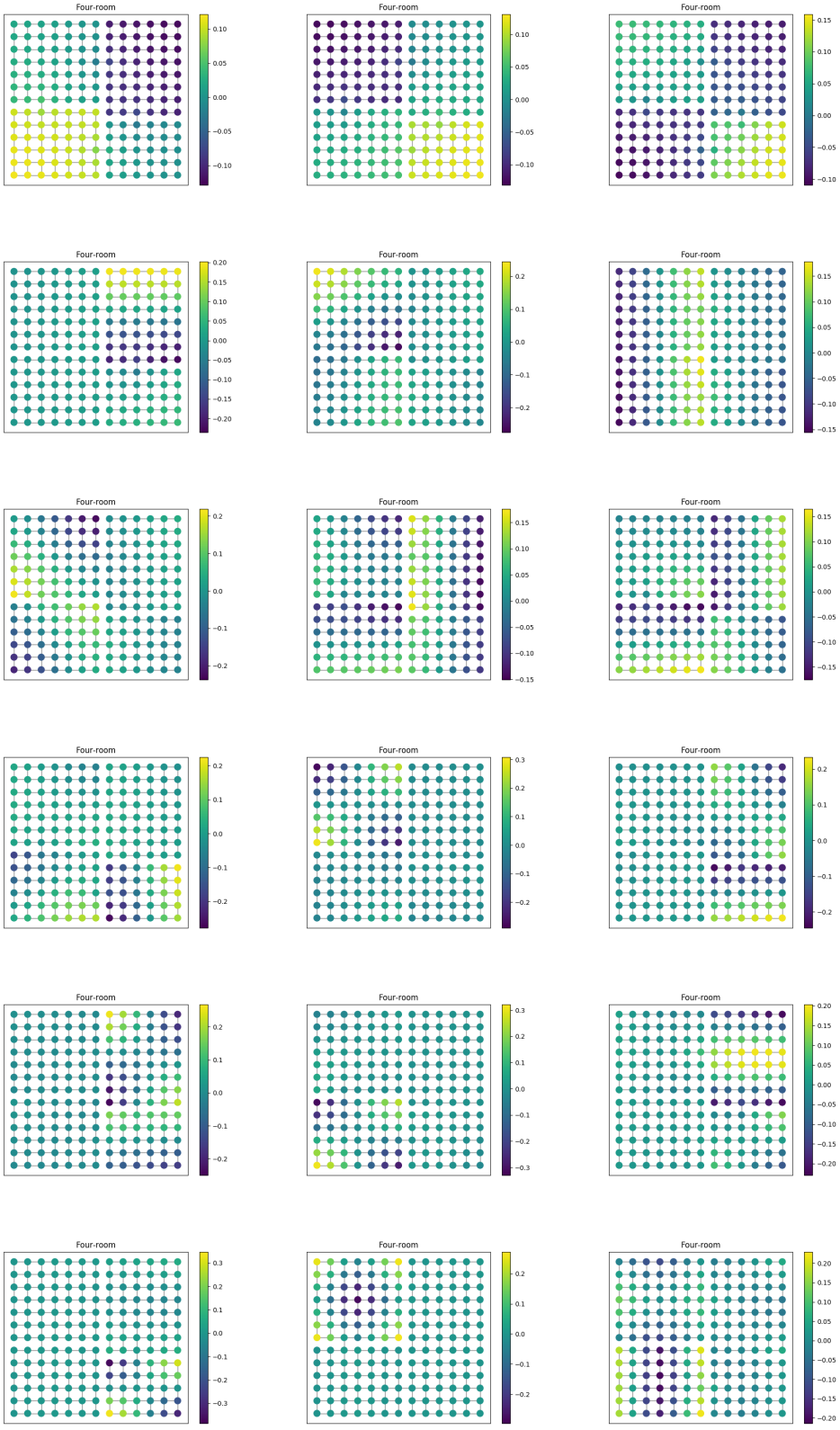


Figure A.1: First 18 PVFs on the four-room domain.

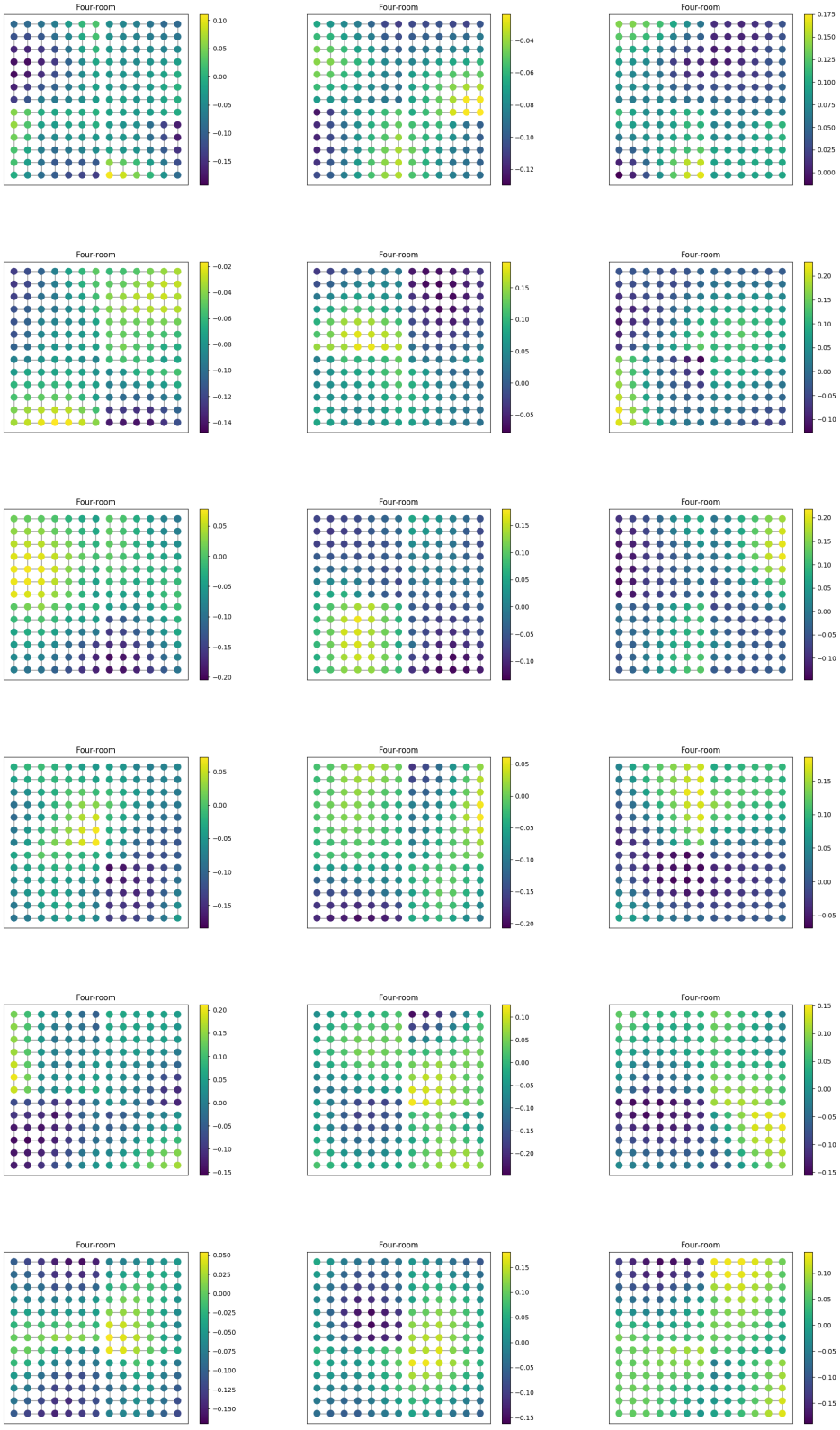


Figure A.2: 18 node2vec dimensions on the four-room domain.

Figure A.3: First 18 PVFs on the low stretch tree domain.

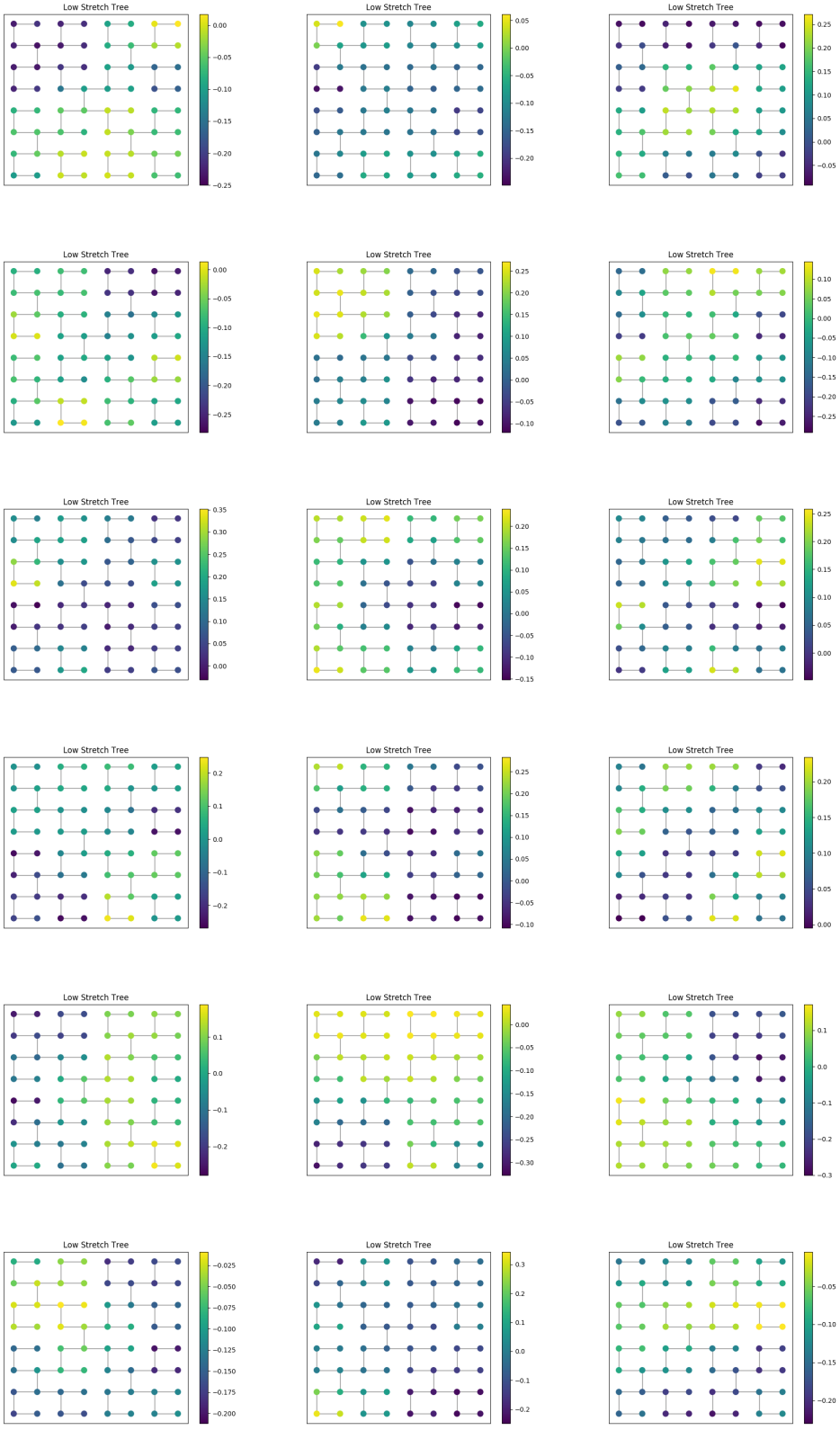


Figure A.4: 18 node2vec dimensions on the low stretch tree domain.

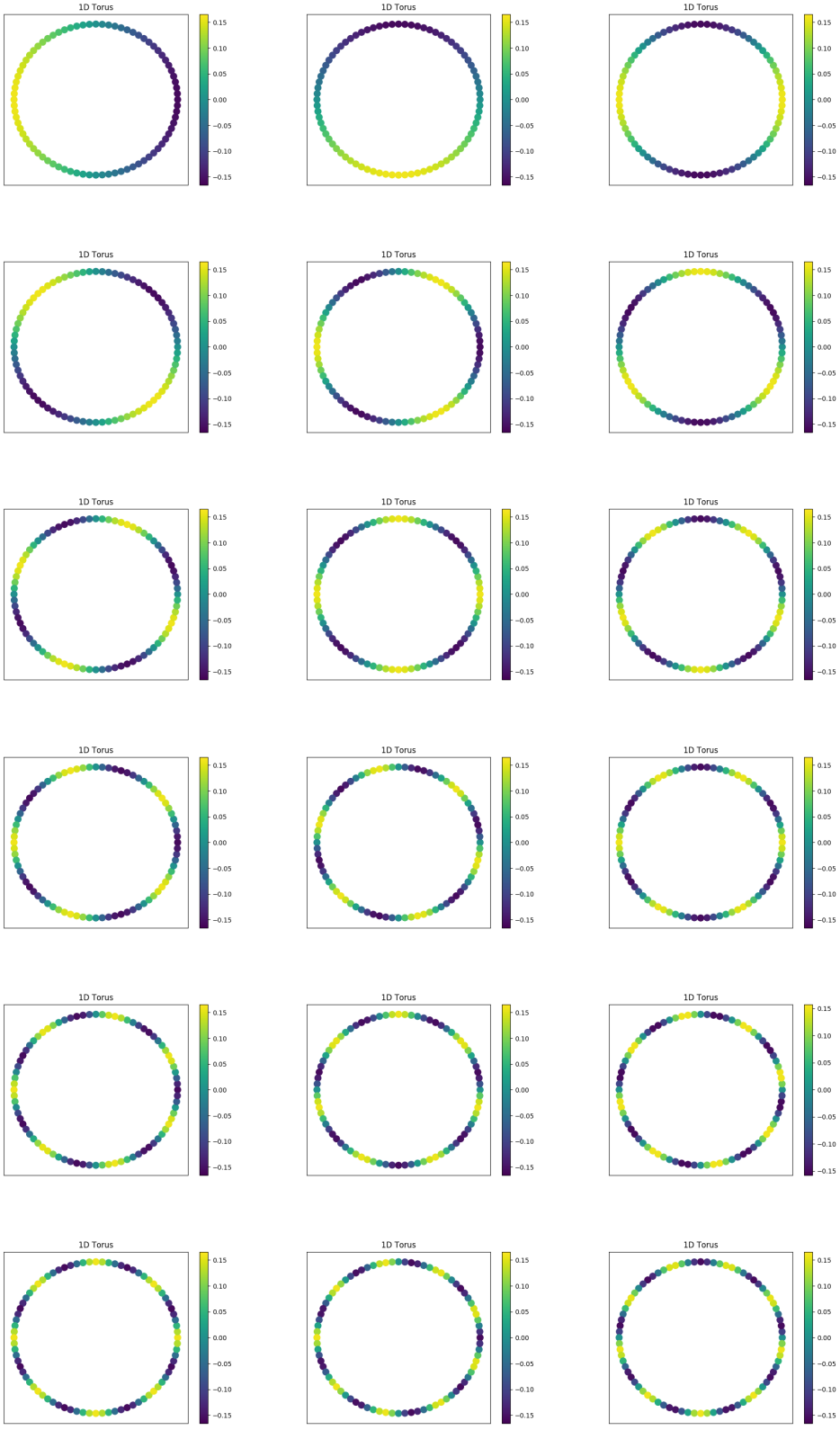


Figure A.5: First 18 PVFs on the 1D torus domain.

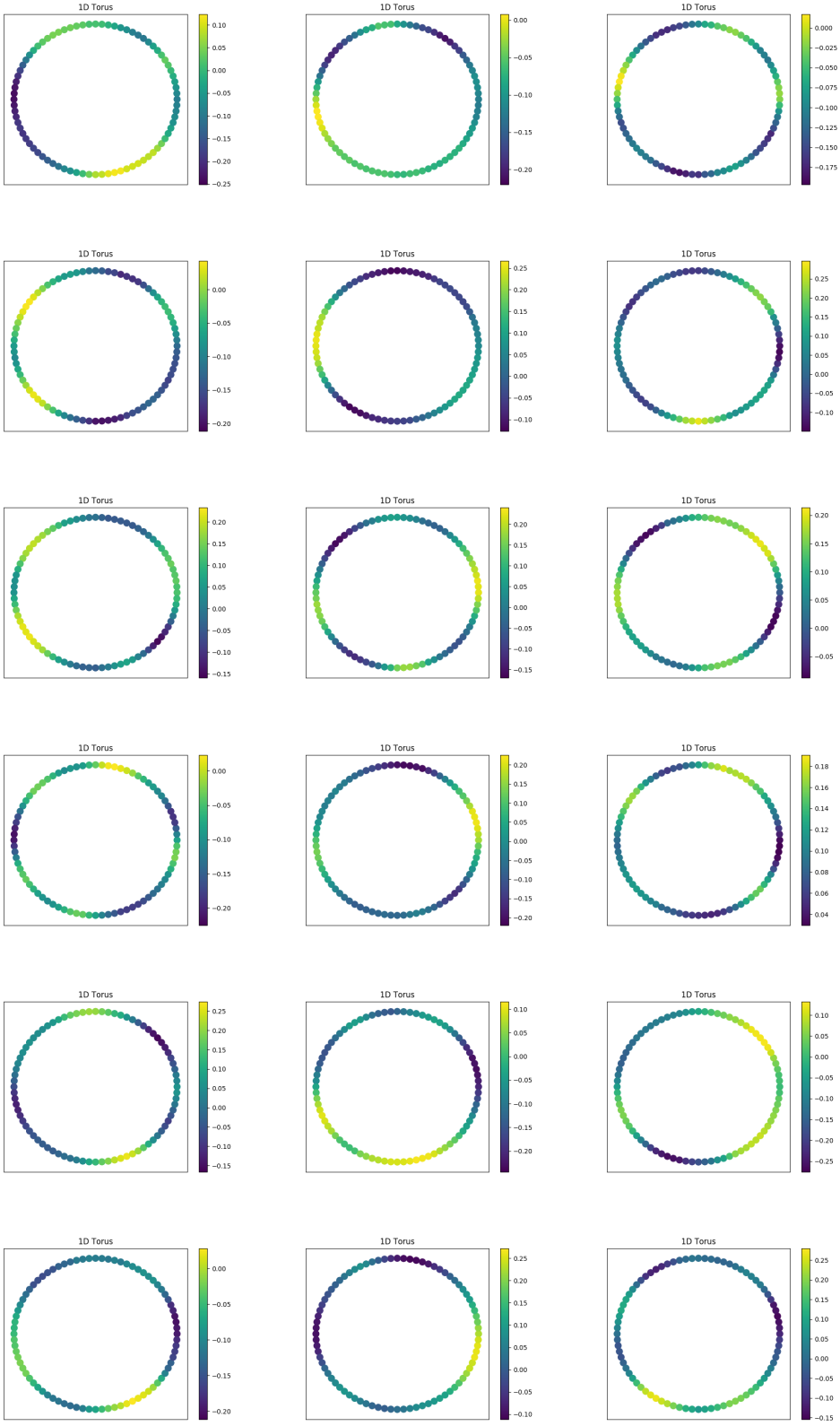


Figure A.6: 18 node2vec dimensions on the 1D torus domain.

Bibliography

[1] R. Agarwal, M. C. Machado, P. S. Castro, and M. G. Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. In *International Conference on Learning Representations*, 2021.

[2] S. I. Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998. ISSN 0899-7667.

[3] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. P. van Hasselt, and D. Silver. Successor features for transfer in reinforcement learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/350db081a661525235354dd3e19b8c05-Paper.pdf.

[4] A. Barreto, D. Borsa, J. Quan, T. Schaul, D. Silver, M. Hessel, D. Mankowitz, A. Zidek, and R. Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 501–510, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/barreto18a.html.

[5] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[6] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012. URL http://arxiv.org/abs/1207.4708.

[7] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 449–458. JMLR.org, 2017.

[8] M. G. Bellemare, W. Dabney, R. Dadashi, A. A. Taïga, P. S. Castro, N. L. Roux, D. Schuurmans, T. Lattimore, and C. Lyle. A geometric perspective on optimal representations for reinforcement learning. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4360–4371, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/3cf2559725a9fdfa602ec8c887440f32-Abstract.html.

[9] R. Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957. ISSN 00959057, 19435274. URL http://www.jstor.org/stable/24900506.

[10] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[11] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition, 1996. ISBN 1886529108.

[12] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[13] S. Bhagat, G. Cormode, and S. Muthukrishnan. *Node Classification in Social Networks*, pages 115–148. Springer US, Boston, MA, 2011. ISBN 978-1-4419-8462-3. doi: 10.1007/978-1-4419-8462-3_5. URL https://doi.org/10.1007/978-1-4419-8462-3_5.

[14] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

[15] J. Boyan. Least-squares temporal difference learning. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 49–56. Morgan Kaufmann, 1999.

[16] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

[17] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

[18] S. J. Bradtke and A. G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1):33–57, Mar 1996. ISSN 1573-0565. doi: 10.1007/BF00114723. URL `https://doi.org/10.1007/BF00114723`.

[19] D. Brandfonbrener and J. Bruna. Geometric insights into the convergence of non-linear TD learning. In *International Conference on Learning Representations*, 2020.

[20] P. S. Castro. Scalable methods for computing state similarity in deterministic markov decision processes. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(06):10069–10076, Apr. 2020. doi: 10.1609/aaai.v34i06.6564. URL `https://ojs.aaai.org/index.php/AAAI/article/view/6564`.

[21] P. S. Castro and D. Precup. Using bisimulation for policy transfer in mdps. In *AAAI*. AAAI Press, 2010.

[22] P. S. Castro, T. Kastner, P. Panangaden, and M. Rowland. MICo: Improved representations via sampling-based state similarity for markov decision processes. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=wFp6kmQELgu`.

[23] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[24] W. Dabney, A. Barreto, M. Rowland, R. Dadashi, J. Quan, M. G. Bellemare, and D. Silver. The value-improvement path: Towards better represen-

tations for reinforcement learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 7160–7168. AAAI Press, 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/16880.

[25] R. Dadashi, A. A. Taiga, N. L. Roux, D. Schuurmans, and M. G. Bellemare. The value function polytope in reinforcement learning. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1486–1495. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/dadashi19a.html.

[26] P. Dayan. The convergence of TD(λ) for general lambda. *Machine Learning*, 8:341–362, 1992.

[27] P. Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993. doi: 10.1162/neco.1993.5.4.613. URL https://doi.org/10.1162/neco.1993.5.4.613.

[28] R. T. des Combes, P. Bachman, and H. van Seijen. Learning invariances for policy generalization, 2018. URL https://openreview.net/forum?id=BJHRaK1PG.

[29] C. H. Q. Ding, X. He, H. Zha, M. Gu, and H. Simon. A min-max cut algorithm for graph partitioning and data clustering, 2001.

[30] C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec. Learning structural node embeddings via diffusion wavelets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 1320–1329, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5552-0. doi: 10.1145/3219819.3220025. URL http://doi.acm.org/10.1145/3219819.3220025.

[31] S. S. Du, S. M. Kakade, R. Wang, and L. F. Yang. Is a good representation sufficient for sample efficient reinforcement learning? In *Inter-*

national Conference on Learning Representations, 2020. URL https://openreview.net/forum?id=r1genAVKPB.

[32] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[33] D. Dwibedi, J. Tompson, C. Lynch, and P. Sermanet. Learning actionable representations from visual observations. *CoRR*, abs/1808.00928, 2018. URL http://arxiv.org/abs/1808.00928.

[34] S. Elfwing, E. Uchibe, and K. Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

[35] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(18):503–556, 2005. URL http://jmlr.org/papers/v6/ernst05a.html.

[36] A. Erraqabi, M. Zhao, M. C. Machado, Y. Bengio, S. Sukhbaatar, L. Denoyer, and A. Lazaric. Exploration-driven representation learning in reinforcement learning. In *ICML 2021 Workshop on Unsupervised Reinforcement Learning*, 2021. URL https://openreview.net/forum?id=6gEYTQFfnZh.

[37] N. Ferns and D. Precup. Bisimulation metrics are optimal value functions. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, UAI'14, page 210–219, Arlington, Virginia, USA, 2014. AUAI Press. ISBN 9780974903910.

[38] N. Ferns, P. Panangaden, and D. Precup. Metrics for finite markov decision processes. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, page 162–169, Arlington, Virginia, USA, 2004. AUAI Press. ISBN 0974903906.

[39] N. Ferns, P. S. Castro, D. Precup, and P. Panangaden. Methods for computing state similarity in markov decision processes. *ArXiv*, abs/1206.6836, 2006.

[40] N. Ferns, P. Panangaden, and D. Precup. Bisimulation metrics for continuous markov decision processes. *SIAM Journal on Computing*, 40(6):1662–1714,

2011. doi: 10.1137/10080484X. URL https://doi.org/10.1137/10080484X.

[41] V. François-Lavet, Y. Bengio, D. Precup, and J. Pineau. Combined reinforcement learning via abstract representations. *CoRR*, abs/1809.04506, 2018. URL http://arxiv.org/abs/1809.04506.

[42] C. Gelada, S. Kumar, J. Buckman, O. Nachum, and M. G. Bellemare. Deepmdp: Learning continuous latent space models for representation learning. *CoRR*, abs/1906.02736, 2019.

[43] D. Ghosh and M. G. Bellemare. Representations for stable off-policy reinforcement learning. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 3556–3565. PMLR, 2020.

[44] R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147(1):163–223, 2003. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(02)00376-4. URL https://www.sciencedirect.com/science/article/pii/S0004370202003764. Planning with Uncertainty and Incomplete Information.

[45] A. Grover and J. Leskovec. node2vec. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, pages 855–864, 2016. ISBN 9781450342322. doi: 10.1145/2939672.2939754. URL http://dl.acm.org/citation.cfm?doid=2939672.2939754.

[46] S. Gu, T. P. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. *CoRR*, abs/1611.02247, 2016. URL http://arxiv.org/abs/1611.02247.

[47] Z. D. Guo, B. Á. Pires, B. Piot, J. Grill, F. Altché, R. Munos, and M. G. Azar. Bootstrap latent-predictive representations for multitask reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3875–3886. PMLR, 2020. URL http://proceedings.mlr.press/v119/guo20g.html.

[48] M. Gutmann and A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 297–304, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/gutmann10a.html.

[49] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018. URL http://arxiv.org/abs/1812.05905.

[50] K. Hao. Training a single ai model can emit as much carbon as five cars in their lifetimes. https://www.technologyreview.com/2019/06/06/239031/, 2019. Accessed: 20 July 2021.

[51] H. v. Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100. AAAI Press, 2016.

[52] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9726–9735, 2020. doi: 10.1109/CVPR42600.2020.00975.

[53] T. Hennigan, T. Cai, T. Norman, and I. Babuschkin. Haiku: Sonnet for JAX, 2020. URL http://github.com/deepmind/dm-haiku.

[54] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017. URL http://arxiv.org/abs/1710.02298.

[55] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR (Poster)*. OpenReview.net, 2017.

[56] I. Higgins, A. Pal, A. Rusu, L. Matthey, C. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner. DARLA: Improving zero-shot transfer in reinforcement learning. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1480–1490. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/higgins17a.html.

[57] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. doi: 10.1126/science.1127647. URL https://www.science.org/doi/abs/10.1126/science.1127647.

[58] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. P. van Hasselt, and D. Silver. Distributed prioritized experience replay. In *International Conference on Learning Representations*, 2018.

[59] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[60] E. Imani, E. Graves, and M. White. An off-policy policy gradient theorem using emphatic weightings. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 96–106. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7295-an-off-policy-policy-gradient-theorem-using-emphatic-weigh
pdf.

[61] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*. OpenReview.net, 2017.

[62] C. Jin, Z. Yang, Z. Wang, and M. I. Jordan. Provably efficient reinforcement learning with linear function approximation. In J. Abernethy and S. Agarwal, editors, *Proceedings of Thirty Third Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pages 2137–2143. PMLR, 09–12 Jul 2020. URL https://proceedings.mlr.press/v125/jin20a.html.

[63] N. Jong and P. Stone. Kernel-based models for reinforcement learning in continuous state spaces. In *ICML workshop on Kernel Machines and Reinforcement Learning*, June 2006.

[64] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains, 1995. Unpublished report.

[65] P. L. Kampala. Building fertile ground for data science in uganda. `https://www.unglobalpulse.org/2016/08/building-fertile-ground-for-data-science-in-uganda/`, 2016. Accessed: 20 July 2021.

[66] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

[67] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski. Vizdoom: A doom-based AI research platform for visual reinforcement learning. *CoRR*, abs/1605.02097, 2016. URL `http://arxiv.org/abs/1605.02097`.

[68] D. P. Kingma and J. B. Adam. A method for stochastic optimization. In *International Conference on Learning Representation*, 2015.

[69] T. N. Kipf and M. Welling. Variational graph auto-encoders. *CoRR*, abs/1611.07308, 2016. URL `http://arxiv.org/abs/1611.07308`.

[70] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel. A survey of generalisation in deep reinforcement learning. *CoRR*, abs/2111.09794, 2021. URL `https://arxiv.org/abs/2111.09794`.

[71] G. Konidaris, S. Osentoski, and P. Thomas. Value Function Approximation in Reinforcement Learning using the Fourier Basis. *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, pages 380–385, 2011.

[72] Y. Koren. On spectral graph drawing. In *International Computing and Combinatorics Conference*, pages 496–508. Springer, 2003.

[73] M. Lagoudakis. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003. ISSN 15324435. doi:

10.1162/jmlr.2003.4.6.1107. URL http://dl.acm.org/citation.cfm?id=964290.

[74] S. Lange and M. Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2010. doi: 10.1109/IJCNN.2010.5596468.

[75] S. Lange, M. Riedmiller, and A. Voigtländer. Autonomous reinforcement learning on raw visual input data in a real world application. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2012. doi: 10.1109/IJCNN.2012.6252823.

[76] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991. ISSN 0890-5401. doi: https://doi.org/10.1016/0890-5401(91)90030-6. URL https://www.sciencedirect.com/science/article/pii/0890540191900306.

[77] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas. Reinforcement learning with augmented data. *CoRR*, abs/2004.14990, 2020. URL https://arxiv.org/abs/2004.14990.

[78] A. J. Laub. A schur method for solving algebraic riccati equations, 1978. URL http://hdl.handle.net/1721.1/1301.

[79] L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for mdps. In *In Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, pages 531–539, 2006.

[80] Y. Li. Deep reinforcement learning: An overview. *CoRR*, abs/1701.07274, 2017. URL http://arxiv.org/abs/1701.07274.

[81] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *J. Am. Soc. Inf. Sci. Technol.*, 58(7):1019–1031, May 2007. ISSN 1532-2882. doi: 10.1002/asi.v58:7. URL http://dx.doi.org/10.1002/asi.v58:7.

[82] L. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3):293–321, 1992.

[83] G. Liu, C. Zhang, L. Zhao, T. Qin, J. Zhu, J. Li, N. Yu, and T. Liu. Return-based contrastive representation learning for reinforcement learning. *CoRR*, abs/2102.10960, 2021. URL https://arxiv.org/abs/2102.10960.

[84] C. Ma, J. Wen, and Y. Bengio. Universal successor representations for transfer reinforcement learning. *CoRR*, abs/1804.03758, 2018. URL http://arxiv.org/abs/1804.03758.

[85] M. C. Machado, M. G. Bellemare, and M. Bowling. A laplacian framework for option discovery in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 2295–2304. JMLR.org, 2017.

[86] M. C. Machado, M. G. Bellemare, and M. Bowling. Count-based exploration with the successor representation. *CoRR*, abs/1807.11622, 2018. URL http://arxiv.org/abs/1807.11622.

[87] M. C. Machado, C. Rosenbaum, X. Guo, M. Liu, G. Tesauro, and M. Campbell. Eigenoption discovery through the deep successor representation. In *ICLR (Poster)*. OpenReview.net, 2018.

[88] M. C. Machado, A. Barreto, and D. Precup. Temporal abstraction in reinforcement learning with the successor representation. *CoRR*, abs/2110.05740, 2021. URL https://arxiv.org/abs/2110.05740.

[89] S. Madjiheurem and L. Toni. Representation learning on graphs: A reinforcement learning application. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 3391–3399. PMLR, 16–18 Apr 2019. URL https://proceedings.mlr.press/v89/madjiheurem19a.html.

[90] S. Madjiheurem and L. Toni. State2vec: Off-policy successor features approximators. *CoRR*, abs/1910.10277, 2019. URL http://arxiv.org/abs/1910.10277.

[91] S. Madjiheurem and L. Toni. Disentangled predictive representation for meta-reinforcement learning. In *ICML 2021 Workshop on Unsupervised Reinforcement Learning*, 2021. URL https://openreview.net/forum?id=VbLGbcdz16-.

[92] S. Madjiheurem, M. Bellemare, and L. Toni. On the importance of representation learning in reinforcement learning. Pending publication, 2022.

[93] S. Mahadevan. Learning Representation and Control in Markov Decision Processes: New Frontiers. *Foundations and Trends® in Machine Learning*, 1(4):403–565, 2007. ISSN 1935-8237. doi: 10.1561/2200000003. URL http://www.nowpublishers.com/article/Details/MAL-003.

[94] S. Mahadevan and M. Maggioni. Value function approximation with diffusion wavelets and laplacian eigenfunctions. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 843–850. MIT Press, 2006. URL https://proceedings.neurips.cc/paper/2005/file/2650d6089a6d640c5e85b2b88265dc2b-Paper.pdf.

[95] S. Mahadevan and M. Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *J. Mach. Learn. Res.*, 8:2169–2231, Dec. 2007. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1314498.1314570.

[96] J. Martens. *Second-order optimization for neural networks*. University of Toronto (Canada), 2016.

[97] Y. Matias. A big step for flood forecasts in india and bangladesh. https://blog.google/technology/ai/flood-forecasts-india-bangladesh/, 2020. Accessed: 20 July 2021.

[98] A. K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Computer Science Department, University of Rochester, 1996. URL ftp://ftp.cs.rochester.edu/pub/papers/robotics/97.tr611.Reinforcement_learning_with_selective_percptn_and_hdn_state.ps.gz.

[99] J. McCarthy, M. Minsky, N. Rochester, and C. Shannon. A proposal for the dartmouth summer research project on artificial intelligence. https://web.archive.org/web/20070826230310/http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html, 1955. Accessed: 30 January 2022.

[100] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. URL http://arxiv.org/abs/1301.3781.

[101] M. Minsky. Steps toward artificial intelligence. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 406–450. McGraw-Hill, New York, 1963.

[102] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

[103] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[104] P. Montague. Reinforcement Learning: An Introduction, by Sutton, R.S. and Barto, A.G. *Trends in Cognitive Sciences*, 3(9):360, 1999. ISSN 13646613. doi: 10.1016/S1364-6613(99)01331-5. URL http://linkinghub.elsevier.com/retrieve/pii/S1364661399013315.

[105] C. Monteleoni, G. A. Schmidt, and S. McQuade. Climate informatics: Accelerating discovering in climate science with machine learning. *Computing in Science Engineering*, 15(5):32–40, 2013. doi: 10.1109/MCSE.2013.50.

[106] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[107] J. P. O'Doherty, P. Dayan, K. Friston, H. Critchley, and R. J. Dolan. Temporal difference models and reward-related learning in the human brain. *Neuron*, 38(2):329–337, 2003. ISSN 08966273. doi: 10.1016/S0896-6273(03)00169-7.

[108] Y. Ollivier. Approximate temporal difference learning is a gradient descent for reversible policies. *CoRR*, abs/1805.00869, 2018.

[109] OpenAI. Randomagent on spaceinvaders-v0. `https://gym.openai.com/envs/SpaceInvaders-v0/`, 2013. Accessed: 21 January 2022.

[110] D. Ormoneit and S. Sen. Kernel-based reinforcement learning. *Mach. Learn.*, 49(2-3):161–178, Nov. 2002. ISSN 0885-6125. doi: 10.1023/A:1017928328829. URL `https://doi.org/10.1023/A:1017928328829`.

[111] R. Parr, C. Painter-Wakefield, L. Li, and M. Littman. Analyzing feature generation for value-function approximation. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 737–744, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3. doi: 10.1145/1273496.1273589. URL `http://doi.acm.org/10.1145/1273496.1273589`.

[112] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 752–759, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390251. URL `http://doi.acm.org/10.1145/1390156.1390251`.

[113] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

[114] J. Peng. *Efficient dynamic programming-based learning for control*. PhD thesis, Northeastern University, 1993.

[115] J. Peng and R. J. Williams. Incremental multi-step Q-learning. *Machine Learning*, 22:283–290, 1996.

[116] M. Petrik. An analysis of laplacian methods for value function approximation in mdps. In *IJCAI*, pages 2574–2579, 2007.

[117] M. Petrik, G. Taylor, R. Parr, and S. Zilberstein. Feature selection using regularization in approximate linear programs for markov decision processes. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 871–878, 2010. URL `http://www.icml2010.org/papers/52.pdf`.

[118] S. Pitis. Source traces for temporal difference learning. In S. A. McIlraith and K. Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 3952–3959. AAAI Press, 2018.

[119] T. Pohlen, B. Piot, T. Hester, M. G. Azar, D. Horgan, D. Budden, G. Barth-Maron, H. P. van Hasselt, J. Quan, M. Večerík, M. Hessel, R. Munos, and O. Pietquin. Observe and look further: Achieving consistent performance on Atari. *arXiv preprint arXiv:1805.11593*, 2018.

[120] D. Precup. Temporal abstraction in reinforcement learning, 2000.

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

[122] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc. New York, NY, USA, 1994.

[123] B. Ravindran and A. G. Barto. Relativized options: Choosing the right transformation. In *ICML*, pages 608–615. AAAI Press, 2003.

[124] B. Ravindran and A. G. Barto. Approximate homomorphisms: A framework for non-exact minimization in markov decision processes. In *Proceedings of the 5th International Conference on Knowledge-Based Computer Systems*, 2004.

[125] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo. Struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 385–394, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4887-4. doi: 10.1145/3097983.3098061. URL `http://doi.acm.org/10.1145/3097983.3098061`.

[126] M. Riedmiller. Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning method. In J. Gama, R. Camacho, P. Brazdil, A. Jorge, and L. Torgo, editors, *Proceedings of the 16th European Conference on Machine Learning (ECML'05)*, pages 317–328. Springer, 2005.

[127] D. Rolnick, P. L. Donti, L. H. Kaack, K. Kochanski, A. Lacoste, K. Sankaran, A. S. Ross, N. Milojevic-Dupont, N. Jaques, A. Waldman-Brown, A. Luccioni, T. Maharaj, E. D. Sherwin, S. K. Mukkavilli, K. P. Körding, C. P. Gomes, A. Y. Ng, D. Hassabis, J. C. Platt, F. Creutzig, J. T. Chayes, and Y. Bengio. Tackling climate change with machine learning. *CoRR*, abs/1906.05433, 2019. URL `http://arxiv.org/abs/1906.05433`.

[128] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *International Conference on Learning Representations*, Puerto Rico, 2016.

[129] B. Schölkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, Dec. 2002. Parts of this book, including an introduction to kernel methods, can be downloaded ¡a href="http://www.learning-with-kernels.org/sections/"¿here¡/a¿.

[130] B. Scholkopf, S. Mika, C. J. C. Burges, P. Knirsch, K. . Muller, G. Ratsch, and A. J. Smola. Input space versus feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, 10(5):1000–1017, Sep. 1999. ISSN 1045-9227. doi: 10.1109/72.788641.

[131] M. Schwarzer, A. Anand, R. Goel, R. D. Hjelm, A. C. Courville, and P. Bachman. Data-efficient reinforcement learning with self-predictive representations. In *ICLR*. OpenReview.net, 2021.

[132] M. Schwarzer, N. Rajkumar, M. Noukhovitch, A. Anand, L. Charlin, R. D. Hjelm, P. Bachman, and A. Courville. Pretraining representations for data-efficient reinforcement learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=XpSAvlvnMa`.

[133] J. Shen and L. F. Yang. Theoretically principled deep rl acceleration via nearest neighbor function approximation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(11):9558–9566, May 2021. URL `https://ojs.aaai.org/index.php/AAAI/article/view/17151`.

[134] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98, 2013.

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

[136] S. P. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[137] A. Srinivas, M. Laskin, and P. Abbeel. CURL: contrastive unsupervised representations for reinforcement learning. *CoRR*, abs/2004.04136, 2020. URL `https://arxiv.org/abs/2004.04136`.

[138] K. Stachenfeld, J. Godwin, and P. Battaglia. Graph networks with spectral message passing, 2020.

[139] K. L. Stachenfeld, M. Botvinick, and S. J. Gershman. Design principles of the hippocampal cognitive map. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL `https://proceedings.neurips.cc/paper/2014/file/dfd7468ac613286cdbb40872c8ef3b06-Paper.pdf`.

[140] E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for modern deep learning research. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 13693–13696. AAAI

Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/7123.

[141] R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[142] R. S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci., 1984.

[143] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug 1988. ISSN 1573-0565. doi: 10.1007/BF00115009. URL https://doi.org/10.1007/BF00115009.

[144] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning*, pages 216–224, 1990.

[145] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. *IEEE Trans. Neural Networks*, 9(5):1054–1054, 1998. doi: 10.1109/TNN.1998.712192. URL https://doi.org/10.1109/TNN.1998.712192.

[146] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[147] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[148] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems 13 (NIPS-00)*, 12:1057–1063, 2000.

[149] R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup. Horde: A scalable real-time architecture for learning knowledge

from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '11, page 761–768, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0982657161.

[150] R. S. Sutton, A. R. Mahmood, and M. White. An emphatic approach to the problem of off-policy temporal-difference learning. *Journal of Machine Learning Research*, 17(73):1–29, 2016.

[151] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. P. Lillicrap, and M. A. Riedmiller. Deepmind control suite. *CoRR*, abs/1801.00690, 2018. URL `http://arxiv.org/abs/1801.00690`.

[152] J. Taylor, D. Precup, and P. Panagaden. Bounding performance loss in approximate mdp homomorphisms. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2009. URL `https://proceedings.neurips.cc/paper/2008/file/6602294be910b1e3c4571bd98c4d5484-Paper.pdf`.

[153] G. Tesauro. Practical issues in temporal difference learning. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 4*, pages 259–266. San Mateo, CA: Morgan Kaufmann, 1992.

[154] G. J. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

[155] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

[156] J. N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16:185–202, 1994.

[157] 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.

[158] A. van den Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748, 2018. URL `http://arxiv.org/abs/1807.03748`.

[159] H. van Hasselt, S. Madjiheurem, M. Hessel, D. Silver, A. Barreto, and D. Borsa. Expected eligibility traces. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, Virtual Event, February 2-9, 2021*, pages 9997–10005. AAAI Press, 2021. URL `https://ojs.aaai.org/index.php/AAAI/article/view/17200/`.

[160] H. P. van Hasselt. Reinforcement learning in continuous state and action spaces. In M. A. Wiering and M. van Otterlo, editors, *Reinforcement Learning: State of the Art*, volume 12 of *Adaptation, Learning, and Optimization*, pages 207–251. Springer, 2012.

[161] H. P. van Hasselt and R. S. Sutton. Learning to predict independent of span. *CoRR*, abs/1508.04582, 2015.

[162] H. P. van Hasselt, A. R. Mahmood, and R. S. Sutton. Off-policy TD(λ) with a true online equivalence. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence*, pages 330–339, 2014.

[163] H. P. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4287–4295, 2016.

[164] H. P. van Hasselt, M. Hessel, and J. Aslanides. When to use parametric models in reinforcement learning? In *Advances in Neural Information Processing Systems*, volume 32, pages 14322–14333, 2019.

[165] H. P. van Hasselt, J. Quan, M. Hessel, Z. Xu, D. Borsa, and A. Barreto. General non-linear Bellman equations. *arXiv preprint arXiv:1907.03687*, 2019.

[166] H. van Seijen and R. S. Sutton. Planning by prioritized sweeping with small backups. In *International Conference on Machine Learning*, pages 361–369, 2013.

[167] H. van Seijen and R. S. Sutton. True online TD(λ). In *International Conference on Machine Learning*, pages 692–700, 2014.

[168] A. van Wynsberghe. Sustainable ai: Ai for sustainability and the sustainability of ai. *AI and Ethics*, 2021. doi: 10.1007/s43681-021-00043-6. URL https://doi.org/10.1007/s43681-021-00043-6.

[169] Z. Wang, N. de Freitas, T. Schaul, M. Hessel, H. P. van Hasselt, and M. Lanctot. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, New York, NY, USA, 2016.

[170] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992. doi: 10.1007/BF00992698. URL https://doi.org/10.1007/BF00992698.

[171] P. J. Werbos. A menu of designs for reinforcement learning over time. *Neural networks for control*, pages 67–95, 1990.

[172] S. D. Whitehead and D. H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83, 1991.

[173] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[174] Y. Wu, G. Tucker, and O. Nachum. The laplacian in RL: Learning representations with efficient approximations. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJlNpoA5YQ.

[175] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin. Unsupervised feature learning via non-parametric instance-level discrimination. *CoRR*, abs/1805.01978, 2018. URL http://arxiv.org/abs/1805.01978.

[176] A. Zhang, R. T. McAllister, R. Calandra, Y. Gal, and S. Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=-2FCwDKRREu.

[177] S. Zhang, W. Boehmer, and S. Whiteson. Generalized off-policy actor-critic. In *Advances in Neural Information Processing Systems*, pages 2001–2011, 2019.