Efficient Exploration in Reinforcement Learning through Time-Based Representations

by

Marlos Cholodovskis Machado

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

In the reinforcement learning (RL) problem an agent must learn how to act optimally through trial-and-error interactions with a complex, unknown, stochastic environment. The actions taken by the agent influence not just the immediate reward it observes but also the future states and rewards it will observe, implicitly requiring the agent to deal with the trade-off between short-term and long-term consequences. In this context, the problem of exploration is the problem of selecting appropriate actions to explore the state space to gather information while taking this trade-off into consideration.

In this dissertation I advocate that agents' exploration strategy can be guided by the process of representation learning. I support this claim by introducing different exploration approaches for RL algorithms that are applicable to complex environments with sparse rewards. They all use learned time-based representations, state representations that capture the temporal aspect of RL problems, implicitly encoding the temporal proximity of states. The two instantiations of time-based representations I use are proto-value functions (PVFs) and the successor representation (SR).

The first approaches I introduce are based on the idea of option-based exploration. Option-based exploration hinges on the assumption that an agent that exhibits purposeful behavior is more likely to visit states that are far from its current state than an agent that randomly selects actions at every time step. I model this purposefulness through *options*, which, in reinforcement learning, represent temporally extended courses of actions over different

time scales. I then introduce algorithms capable of discovering options autonomously through PVFs and the SR.

I also introduce count-based exploration approaches, which are based on the idea of keeping state visitation counts to ensure all states (or abstractions of a state) are visited a proper number of times. I show that the norm of the SR, while it is being learned, incorporates state visitation counts and I use this result to introduce RL algorithms that achieve state-of-the-art results in large domains that require function approximation.

I evaluate my algorithms in both tabular domains and Atari 2600 games. I use tabular domains such as the 4-room domain, RiverSwim, and SixArms in order to develop a better intuition about the proposed algorithms and to compare the proposed approaches to classic baselines in the field. I use Atari 2600 games to evaluate the scalability and generality of the proposed approaches since the state space of Atari 2600 games is too large, requiring function approximation. I discuss approaches based on linear and non-linear function approximation.

Preface

The central chapters of this dissertation are based on papers that are either published or that are currently under review. More specifically, Chapter 3, 4 and 5 are based on papers published in conference proceedings [1, 2]. Chapter 6 is based on a paper currently under review [3]. Parts of Chapter 7 are based on a paper presented at a workshop [4]. In the rest of the chapters (1, 2, and 8), the majority of the contributions are original to this dissertation.

[1] Marlos C. Machado, Marc G. Bellemare, and Michael Bowling (2017). "A Laplacian Framework for Option Discovery in Reinforcement Learning". In: Proceedings of the International Conference on Machine Learning (ICML), pp. 2295–2304.

[2] Marlos C. Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell (2018). "Eigenoption Discovery through the Deep Successor Representation". In: Proceedings of the International Conference on Learning Representations (ICLR).

[3] Marlos C. Machado, Marc G. Bellemare, and Michael Bowling (2018). "Count-Based Exploration with the Successor Representation". In: CoRR abs/1807.11622. Under review.

[4] Marlos C. Machado and Michael Bowling (2016). "Learning Purposeful Behaviour in the Absence of Rewards". In: CoRR abs/1605.07700. Presented at the ICML-16 Workshop on Abstraction in Reinforcement Learning.

My goal when writing this dissertation was to present a concise and cohesive story with all contributions revolving around the same topic. During the period in which I was developing the research presented in this dissertation I also wrote papers that have been omitted. Some of them are briefly discussed as related work [5, 6, 9, 10, 11] while others are entirely unrelated to the main

topic of this thesis [7, 8, 12].

[5] Marlos C. Machado, Sriram Srinivasan, and Michael Bowling (2016). "Domain-Independent Optimistic Initialization for Reinforcement Learning". In: CoRR abs/ 1410.4604. Presented at the AAAI-15 Workshop on Learning for General Competency in Video Games.

[6] Yitao Liang, Marlos C. Machado, Erik Talvitie, and Michael H. Bowling (2016). "State of the Art Control of Atari Games Using Shallow Reinforcement Learning". In: Proceedings of the International Conference on Autonomous Agents & Multiagent Systems (AAMAS), pp. 485–493.

[7] Craig Sherstan, Adam White, Marlos C. Machado, Patrick M. Pilarski (2016). "Introspective Agents: Confidence Measures for General Value Functions". In: Proceedings of the Conference on Artificial General Intelligence (AGI), pp. 258-261.

[8] Harm van Seijen, Ashique Rupam Mahmood, Patrick M. Pilarski, Marlos C. Machado, Richard S. Sutton (2016). "True Online Temporal-Difference Learning". In: Journal of Machine Learning Research 17:145, pp. 1-40.

[9] Miao Liu, Marlos C. Machado, Gerald Tesauro, and Murray Campbell (2018). "The Eigenoption-Critic Framework". In: CoRR abs/1712.04065. Presented at the NIPS-17 Hierarchical RL Workshop.

[10] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling (2018). "Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents". In: Journal of Artificial Intelligence Research 61, pp. 523–562.

[11] Craig Sherstan, Marlos C. Machado, Patrick M. Pilarski (2018). "Accelerating Learning in Constructive Predictive Frameworks with the Successor Representation". In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2997-3003.

[12] Jesse Farebrother, Marlos C. Machado, Michael Bowling (2018). "Generalization and Regularization in DQN". In: CoRR abs/1810.00123. Under review.

To Poliana

For all the sacrifices that she has made and

for always being by my side on this journey.

"Se quer seguir-me, narro-lhe; não uma aventura, mas experiência, a que me induziram, alternadamente, séries de raciocínios e intuições. Tomou-me tempo, desânimos, esforços. Dela me prezo, sem vangloriar-me. Surpreendo-me, porém, um tanto à-parte de todos, penetrando conhecimento que os outros ainda ignoram."

– O Espelho, Guimarães Rosa.

"If you would follow me, I will tell a story – not an adventure, but an experience that I was alternately led to by a series of reasonings and intuitions. It took me time, dismay, and effort. Proud of such an experience, I am not yet vain. I catch myself, however, a tad apart from all, fathoming knowledge others still ignore."

– The Mirror, Guimarães Rosa.
Translation by Axel Perez Trujillo.

Acknowledgements

As Guimarães Rosa wrote in Grande Sertão: Veredas,[1] "the truth is not in the setting out nor in the arriving: it comes to us in the middle of the journey". The truth is that I feel indebted to many people who were by my side at some time during my journey. This is my attempt to acknowledge them.

No words can express my gratitude to my supervisor, Michael Bowling. He was the best supervisor I could have wished for and I am still thankful he accepted me as his student. Michael deeply influenced me not only as a researcher but as a human being and he will always be one of my role-models. I am also very thankful to Marc G. Bellemare, my co-supervisor. Marc stepped in when I needed it most and he was always willing to spend several hours with me to ensure I was making progress in my research and that I was becoming a better researcher along the way.

I thank my other committee members, Rich Sutton, Dale Schuurmans, Zac Friggstad, and Emma Brunskill for their valuable comments, questions and criticism. I feel privileged I was able to interact with them during my Ph.D. and I am thankful that they were all willing to spend time meeting with me to provide a different perspective or to give me advice on future steps.

I was also fortunate to have collaborated with many people who helped me become a better researcher. Erin J. Talvitie was a close colaborator for several years. Fernando Diaz, Alekh Agarwal, Miro Dudik, Rob Schapire, and John Langford helped me widen my perspective about AI while I was interning at MSR NYC. Gerry Tesauro, Murray Campbell, Miao Liu, Xiaoxiao Guo, and Clemens Rosenbaum were my collaborators at IBM Research while

[1] The Devil to Pay in the Backlands, 1963. Translated from Portuguese by James L. Taylor and Harriet de Onís.

I was developing the work presented in Chapter 5. Vlad Mnih, Tom van de Wiele, David Warde-Farley, and Tejas Kulkarni had an important role in improving my understanding about deep reinforcement learning while I was interning at DeepMind. Jesse Farebrother provided me with the initial code I used when developing the deep reinforcement learning model I discuss in Chapter 6. Georg Ostrovski and Yuri Burda kindly provided me the data I used to generate the baseline results reported in Chapter 6 for DQN$^{\text{MMC}}$+CTS, DQN$^{\text{MMC}}$+PixelCNN, and RND. Finally, I am thankful for having worked with Craig Sherstan, Yitao Liang, Jesse Farebrother, Harm van Seijen, Joel Veness, Nicolas Carion, Patrick Pilarski, Martha White, Matthew Hausknecht, Sriram Srinivasan, Ashique Rupam Mahmood, Adam White and Rich Sutton. These collaborations allowed me to explore directions I would have never had the time to explore alone while doing my Ph.D. research.

On a personal level, I had the support of many friends and relatives who helped me when I needed them and who made my stay in Edmonton pleasant. The support from Bernardo, Filipe, Junior, Helio, Zaheen, Craig, Axel, Gabriela, Levi, Camila, Martin and Cacau was very important for me. I am very thankful for their friendship. I am also very thankful to Mario and Gabriela for their friendship and for welcoming us to Edmonton, which was essential in our first year here.

I also thank my parents and brother for always rooting for me and being absolutely sure that everything was going to be all right, even when I did not believe that. Finally, I want to thank my amazing wife, Poliana. There's a reason I dedicated this dissertation to her. I would not have gotten to this point without her support and companionship. We went through more than we expected during these past six years and she was always there for me.

My work was supported by grants from Alberta Innovates Technology Futures and the Alberta Machine Intelligence Institute (Amii). Computing resources were provided by Compute Canada through CalculQuébec.

Contents

1 Introduction **1**

1.1 Thesis statement 2

1.2 Approach 3

1.3 Contributions 5

1.4 Dissertation layout 7

2 Background **8**

2.1 Standard Reinforcement Learning 8

2.1.1 Model-Free Reinforcement Learning 10

2.1.2 Model-Based Reinforcement Learning 12

2.2 Time-Based Representations 14

2.2.1 Proto-Value Functions 14

2.2.2 The Successor Representation 17

2.3 Temporal Abstraction in RL 20

2.3.1 The Options Framework 20

2.3.2 Related Work 22

2.4 Exploration in Reinforcement Learning 26

2.4.1 Theoretical Framework for Exploration 27

2.4.2 Pseudo-Counts for Practical Exploration in RL 29

2.4.3 Related Work 31

2.5 The Arcade Learning Environment 32

3 Option-Based Exploration **36**

3.1 Option Discovery in RL 36

3.1.1 Collect Samples 38

3.1.2 Learn Linear Representation 38
3.1.3 Select k Features 38
3.1.4 Learn How to Attain each Feature 39
3.1.5 Define Option 39
3.2 Overview of the Proposed Algorithms 40

4 Option-Based Exploration with Proto-Value Functions 41
4.1 Option Discovery through the Laplacian 42
4.2 Empirical Evaluation in the Tabular Case 47
4.2.1 Discovered Options 47
4.2.2 Exploration 48
4.2.3 Accumulating Rewards 53
4.3 Approximate Option Discovery 56
4.3.1 Sample-based Option Discovery 57
4.3.2 Function Approximation 58
4.4 Discussion 60

5 Option-Based Exploration with the Successor Representation 62
5.1 Proto-Value Functions and the Eigenvectors of the Successor Representation 63
5.2 Eigenoption Discovery in the Tabular Case 65
5.3 Eigenoption Discovery in the Non-Linear Function Approximation Case 66
5.4 Empirical Evaluation 68
5.4.1 Diffusion Time in the Tabular case 68
5.4.2 Using Eigenoptions to Accumulate Reward 70
5.4.3 The Impact of Learning the SR 72
5.4.4 Non-Linear Function Approximation 75
5.5 Discussion 77

6 Count-Based Exploration with the Successor Representation 80
6.1 The Norm of the Successor Representation as an Exploration Bonus 81

6.1.1 Empirical Demonstration in Tabular Model-Free Reinforcement Learning . . . 81
6.1.2 Theoretical Justification . . . 83
6.1.3 Exploration in Model-based RL with the SSR . . . 86
6.2 Counting Feature Activations with the SR . . . 88
6.3 Evaluating the Exploration Bonus in Deep Reinforcement Learning . . . 91
6.3.1 Overal Performance and Baselines . . . 92
6.3.2 Evaluating the Impact of the Auxiliary Task . . . 94
6.3.3 On the Mismatch between the Used Norms . . . 97
6.4 Discussion . . . 99

7 Bootstrapping in Option-Based Exploration and Other Developments 102
7.1 Bootstrapping Eigenoption Discovery . . . 102
7.2 Online Computation of the Eigenvectors of the Laplacian (or the Successor Representation) . . . 106
7.3 Discussion . . . 108

8 Discussions and Future Work 109
8.1 Summary of Contributions . . . 109
8.2 Discussion and Future Directions . . . 112
8.2.1 Exploration Guided by Representation Learning Algorithms . . . 112
8.2.2 Option-based Exploration . . . 112
8.2.3 Return Maximization with Eigenoptions in Large Environments . . . 113
8.2.4 Theoretical Understanding of Option-based Exploration and of the Successor Representation . . . 114
8.3 Conclusions . . . 115

References 116

Appendix A Supporting Lemmas **127**

Appendix B Evaluating the Reconstruction Task when Learning the Representation and the SR **130**

List of Tables

4.1 Terms used when discussing the introduced options. 45

6.1 Comparison between Naïve Sarsa and Sarsa+SR. 83
6.2 Comparison between ESSR, R-Max, E^3, and MBIE. 88
6.3 Performance of DQN_e^{MMC}+SR compared to various agents on the "hard exploration" subset of Atari 2600 games. 93
6.4 DQN_e^{MMC}+SR results after different amounts of experience. . . 95
6.5 DQN_e^{MMC} results after different amounts of experience. 95
6.6 Performance of DQN_e^{MMC}+SR when using the ℓ_1-norm and ℓ_2-norm of the SR to generate the exploration bonus. 98
6.7 Performance of Sarsa+SR, in the tabular case, when using the ℓ_1-norm and ℓ_2-norm of the SR to generate the exploration bonus. 98
6.8 Results obtained with DQN_e^{MMC}+SR after different amounts of experience when using the ℓ_1-norm of the SR. 99
6.9 Results obtained with DQN_e^{MMC} after different amounts of experience when using the ℓ_1-norm in the normalization. 99

7.1 Properties of discovered options, per iteration, in a ring. . . . 105

List of Figures

2.1 Deep Q-Network as per Mnih, Kavukcuoglu, et al. (2015). . . 12
2.2 Action-Conditional Video Prediction as per Oh et al. (2015). . 13
2.3 First three PVFs in the 4-room domain. 16
2.4 Successor representation for the uniform random policy. 17

3.1 Option discovery cycle. 37

4.1 Tabular domains used for evaluation. 42
4.2 Second PVF and its corresponding option in the 4-room domain. 43
4.3 First four eigenoptions obtained in the 10×10 grid. 48
4.4 First four eigenoptions obtained in the I-Maze domain. 48
4.5 First four eigenoptions obtained in the 4-room domain. 49
4.6 Expected number of steps between any two states when following a random walk. 51
4.7 Options leading to bottleneck states. 52
4.8 Diffusion time and learning performance of eigenoptions and of random options in the 4-room domain. 53
4.9 Agents' performance accumulating reward as options are added to the action set in their behavior policy. 54
4.10 Agents' performance in different tasks when using eigenoptions, bottleneck options, and primitive actions. 55
4.11 Pre-defined start states in Atari 2600 games. 59
4.12 Eigenoptions discovered in Freeway. 60
4.13 Eigenoptions discovered in Montezuma's Revenge and Ms. Pac-Man. 61

5.1 Neural network architecture used to learn the SR. 67
5.2 Results in the 4-room domain when using the SR. 69
5.3 Learning curves obtained in different environments for different number of options obtained from the SR. 70
5.4 Different environments (varying start and goal locations) used when evaluating the agent's ability to accumulate reward. . . . 71
5.5 Agent's performance when following options obtained with estimates of the SR and the true one, $(I - \gamma P_\pi)^{-1}$. 72
5.6 Evolution of the top four eigenvectors being estimated by the SR. 73
5.7 Evolution of the top four eigenoptions being estimated by the SR. 74
5.8 Density of state visitation of eigenoptions discovered in four Atari 2600 games. 76

6.1 Domains used as testbed in the tabular case. 82
6.2 Neural network architecture used by my algorithm when learning to play Atari 2600 games. 90
6.3 $\text{DQN}_e^{\text{MMC}}$+SR and $\text{DQN}_e^{\text{MMC}}$ learning curves in the Atari 2600 games used as testbed. 96
6.4 Evaluation of the sufficiency and necessity of the auxiliary task in $\text{DQN}_e^{\text{MMC}}$+SR. 97
6.5 Learning curves for $\text{DQN}_e^{\text{MMC}}$+SR and $\text{DQN}_e^{\text{MMC}}$ when using the ℓ_1-norm of the SR. 100

7.1 Environment used to evaluate the effects of option composition. 103
7.2 Sample random walk using primitive actions and a random walk using the discovered options. 104

B.1 Final 1-step predictions in the game Bank Heist. 131
B.2 Final 1-step predictions in the game Freeway. 132
B.3 Final 1-step predictions in the game Montezuma's Revenge. 133
B.4 Final 1-step predictions in the game Ms. Pacman. 134

Chapter 1

Introduction

Artificial intelligence (AI) is often seen as an agent's ability to achieve goals in the world, with the agent being a decision maker that perceives its environment through sensors and acts upon that environment through actuators. This ability to achieve goals involves the automation of "activities such as decision-making, problem solving, learning, creating, game playing, and so on" (Bellman 1978). Sequential decision making problems are capable of modeling most of these activities. These problems are the main topic of study in this dissertation.

Sequential decision making problems take place over multiple time steps. In this dissertation I study a particular formulation of sequential decision making problems where at every time step an agent is in a state; it takes an action; and then transitions to a new state observing a real-valued reward signal. The agent's goal is to maximize a (possibly weighted) sum of future rewards. In these problems the actions taken by the agent influence not just the immediate reward it observes but also the future states and rewards it will observe. Consequently, this problem implicitly requires agents to deal with the trade-off between immediate and future rewards.

In the reinforcement learning framework we formulate sequential decision making problems as tasks where an agent must learn how to act optimally through trial-and-error interactions with a complex, unknown, stochastic environment. Reinforcement learning algorithms have been very successful in addressing sequential decision making problems. Some of these successes

include tackling problems of navigation (Banino et al. 2018), memory control optimization (Ipek et al. 2008; Martínez and Ipek 2009), game playing (Mnih, Kavukcuoglu, et al. 2015; Silver et al. 2016; Tesauro 1995), robot control (Levine et al. 2016; Stone, Sutton, and Kuhlmann 2005) and services personalization (Theocharous, Thomas, and Ghavamzadeh 2015).

In this dissertation I study the problem of exploration in reinforcement learning, which aims to reduce the number of samples (i.e., interactions) an agent needs in order to learn to perform well in the aforementioned tasks. The sample efficiency of reinforcement learning algorithms is largely dependent on how agents select actions to explore the state space. Surprisingly, the most common approach to date is to select exploratory actions uniformly at random; with many high-profile success stories in the field obtained with this strategy (e.g., Mnih, Kavukcuoglu, et al. 2015; Tesauro 1995). Nevertheless, random exploration often fails in environments with *sparse* rewards, that is, environments where the agent observes a reward signal of value zero for the majority of states.[1] This dissertation contributes and evaluates exploration approaches for reinforcement learning algorithms that are applicable to complex environments with sparse rewards. Specifically, I focus on algorithmic approaches for exploration that take long-term dependencies into consideration.

1.1 Thesis statement

The central claim of this work is that **time-based representations can be used to design domain-independent algorithms that efficiently explore complex, sparse reward environments**.

Time-based representations are state representations that capture the temporal aspect of sequential decision making problems, implicitly encoding temporal proximity of states; that is, how long an agent would take to go from

[1] In this dissertation I use the term *environments with sparse rewards* for brevity and ease of presentation. Actually, in the reinforcement learning formulation, any sequential decision making problem has dense rewards since, by definition, a reward signal is observed at every time step. By environments with sparse rewards I formally mean environments where the vast majority of transitions lead to reward signals with the same value.

one state to another given a policy. In this dissertation, as I discuss below, I focus on two different time-based representations: proto-value functions (Mahadevan 2005; Mahadevan and Maggioni 2007) and the successor representation (Dayan 1993). I discuss both of these approaches in the next chapter but, in short, proto-value functions are basis functions obtained from the eigendecomposition of the matrix encoding the environment's underlying state-transition graph. The successor representation is a representation that generalizes between states using the similarity between their successors, that is, the similarity between the states that follow the current state given the environment's dynamics and the agent's policy.

By *domain-independent algorithms* I mean reinforcement learning algorithms that are able to succeed in a variety of domains without requiring domain-specific tailoring. I am using this term in the same way as Bellemare, Naddaf, et al. (2013). By algorithms that *efficiently explore* the state space I mean algorithms that given a fixed number of samples it is likely they will visit more states than if they were selecting actions uniformly at random. The main baseline I use throughout most of this dissertation is selecting exploratory actions uniformly at random because it remains the most commonly adopted approach in complex domains.

By *complex environments* I mean large, stochastic, initially unknown environments. Often it is not tractable to enumerate all states in such environments and function approximation, discussed in the next chapter, is required. Finally, *sparse reward environments*, as aforementioned, are those environments in which the value of the observed reward signal is the same in the vast majority of the visited states. This poses a hard exploration problem because all states initially seem equally promising, even though some of these states might be closer to states with different reward signals.

1.2 Approach

I support my thesis statement in this document with different algorithms, theoretical justifications, and experimental results. The developed algorithms

are based either on proto-value functions or on the successor representation; and they tackle the problem of exploration with an approach that is either based on options, defined below, or on counting state/features visitation.

The idea of option-based exploration hinges on the assumption that an agent that exhibits purposeful behavior – i.e., that takes a sequence of actions targeting some goal – is more likely to visit states that are far from its current state than an agent that randomly selects actions at every time step.

I model this purposefulness through *options*, which, in reinforcement learning, represent temporally extended courses of actions over different time scales (Sutton, Precup, and Singh 1999). In this dissertation I consider the call-and-return option execution model and I provide evidence that exploring at a higher level of abstraction, acting according to randomly selected options instead of acting according to randomly selected actions, does reduce the expected average time an agent takes to visit every state in the environment.

In this context, I first introduce an algorithm for option discovery based on proto-value functions. As I discuss in the next chapter, proto-value functions have important properties that are useful for exploration, such as the fact that they do not depend on the environment's rewards, allowing them to be applicable to environments with sparse rewards, and that they capture different time scales in the environment. I then show that the successor representation, which can be estimated online with a low computational cost, is actually a more general version of proto-value functions. Using this insight I revisit my algorithm for option discovery based on proto-value functions and I extend it to the more general case of stochastic environments with asymmetric transitions. Importantly, such generalization also allows me to introduce an algorithm that learns representations at the same time as it discovers options.

I conclude this line of work by providing preliminary results on how we can use some of the aforementioned approaches iteratively. When doing so we can see that the discovered options are increasingly more complex in later iterations and they allow agents to navigate even farther through the state space.

In this dissertation I also explore the idea of count-based exploration by

introducing a count-based algorithm using the successor representation. This algorithm is theoretically justified in the tabular case while being extendable to settings where function approximation is required. My approach and its underlying theory is based on the substochastic successor representation, a concept I develop in this dissertation. I show that the substochastic successor representation is able to implicitly count the number of times each state (or feature) has been observed. The proposed approach is general. It is applicable to the tabular setting, where all states can be uniquely identified, and to the setting where features representing a state are learned through neural networks.

Throughout this dissertation I perform evaluations both in tabular environments and in Atari 2600 games. I use the tabular case to provide intuition about the proposed methods and I use Atari 2600 games to evaluate the generality of the proposed solutions and their applicability to complex domains that require function approximation.

1.3 Contributions

The key contributions of this dissertation are:

- **Option-based exploration.** Options have always been said to have the potential to accelerate learning. In this thesis, considering the call-and-return option execution model, I advocate that options can accelerate learning not only by accelerating the credit assignment process but also by improving exploration through the introduction of purposeful behaviors (Chapter 3). I also show that options that terminate on bottleneck states, which are often used as example of canonical options, impair exploration if used as described above (Chapter 4).

- **Proto-value functions for option-based exploration.** I introduce an algorithm that discovers options autonomously by defining them in terms of proto-value functions. This algorithm is applicable to the tabular case and to the linear function approximation case for binary features,

terms I define in the next chapter (Chapter 4).

- **The successor representation for option-based exploration.** I provide an equivalence between proto-value functions and the eigenvectors of the successor representation, when the SR is defined in a specific way, and I use such an equivalence to extend the aforementioned algorithm to a more general setting with stochastic and asymmetric transitions. I also extend this approach to settings where handcrafted features are not available by using a neural network that is capable of estimating the successor representation from raw pixels while also learning a feature representation (Chapter 5).

- **Incremental exploration with options discovered iteratively.** I present some preliminary results in the tabular case where I evaluate multiple iterations of the process of discovering options that are then used for exploration. In this loop the agent selects actions according to randomly chosen options, moving farther away in the state space. While doing so the agent gathers more experience, which is then used to discover more options that, as I show here, will push the agent even farther through the state space (Chapter 7).

- **The successor representation for count-based exploration.** I propose an extension to the successor representation, termed substochastic successor representation, and I show that it implicitly counts state visitation. This realization leads to a model-based algorithm that is competitive to the state-of-the-art in hard exploration problems in the tabular case. Moreover, it also motivated the development of a model-free algorithm that uses a neural network to learn representations from raw pixels and to estimate state-action value functions. This algorithm achieves performance competitive to the state-of-the-art in Atari 2600 games (Chapter 6).

1.4 Dissertation layout

This document contains eight chapters. Chapter 2 presents an overview of the related background and prior research. Chapter 3 introduces the idea of option-based exploration and the option discovery cycle, a general approach I introduce for option discovery. Chapter 4 contains an instantiation of the option discovery cycle using proto-value functions. Chapter 5 presents the equivalence between proto-value functions and the eigenvectors of a specific instantiation of the successor representation and it uses this equivalence to introduce algorithms for option discovery based on the successor representation. Chapter 6 introduces the substochastic successor representation and it shows how we can use this concept to create count-based exploration algorithms. Chapter 7 presents some preliminary results on the consequences of running multiple iterations of the loop that alternates between exploring the environment and discovering options. It also discusses some recent work, from other research groups, that while not my own build upon the ideas introduced in this thesis. Chapter 8 summarizes these contributions and discusses potential future research directions.

Chapter 2

Background

In this chapter I introduce the formalism behind reinforcement learning, the options framework (Sutton, Precup, and Singh 1999), as well as time-based representations (Dayan 1993; Mahadevan 2005). I elaborate on the problem of exploration in reinforcement learning and I present some of the existing algorithms that tackle this problem. I conclude this chapter discussing the Arcade Learning Environment (Bellemare, Naddaf, et al. 2013), the main platform I used when evaluating my algorithms.

Throughout this dissertation, as a convention, I will indicate scalar-valued random variables by capital letters (e.g., S_t, R_t), vectors by bold lowercase letters (e.g., $\boldsymbol{\theta}, \boldsymbol{\phi}$), functions by non-bold lowercase letters (e.g., v, q), and sets with a calligraphic font (e.g., $\mathcal{S}, \mathcal{A}$).

2.1 Standard Reinforcement Learning

Reinforcement learning (RL) is a problem formulation that allows us to tackle sequential decision making problems. In RL we consider an agent interacting with an unknown environment in a sequential manner, aiming to maximize cumulative reward. In this dissertation I assume that the environment satisfies the Markov property and that it can be modeled as a Markov decision process (MDP). An MDP is formally defined as a 4-tuple $(\mathcal{S}, \mathcal{A}, p, r)$. Starting from state $S_0 \in \mathcal{S}$, at each time step t the agent takes an action $A_t \in \mathcal{A}$, to which the environment responds with a state $S_{t+1} \in \mathcal{S}$, according to a transition prob-

ability kernel $p(s' \,|\, s, a) \doteq \Pr(S_{t+1} = s' \,|\, S_t = s, A_t = a)$, and with a reward signal $R_{t+1} \in \mathbb{R}$, where $r(s, a)$ indicates the expected reward for a transition from state s under action a, that is, $r(s, a) \doteq \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$.

The agent's goal is to learn a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that maps each state to a probability distribution over actions. The optimal policy maximizes, on expectation, the discounted cumulative sum of rewards, defined as

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.1}$$

with $\gamma \in [0, 1)$. The parameter γ is called the discount factor and it defines the relative value of future rewards.

In this dissertation I focus on value-based methods. To obtain the policy π, these algorithms aim to estimate the state-value function $v_\pi : \mathcal{S} \to \mathbb{R}$ or the state-action value function $q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The value of a state s when following a policy π, $v_\pi(s)$, is defined to be the expected sum of discounted rewards from that state: $v_\pi(s) \doteq \mathbb{E}_\pi\big[G_t | S_t = s\big]$. The state-action value function is defined similarly, but it also takes into consideration the action taken in state s, that is, $q_\pi(s, a) \doteq \mathbb{E}_\pi\big[G_t | S_t = s, A_t = a\big]$. Where the expectation in both definitions is with respect to the policy π and the probability kernel p. Importantly, these function can be defined recursively (Bellman 1957):

$$v_\pi(s) \;=\; \sum\nolimits_a \pi(a|s) \Big[r(s, a) + \gamma \sum\nolimits_{s'} p(s'|s, a) v_\pi(s') \Big], \tag{2.2}$$

and

$$\begin{aligned} q_\pi(s, a) \;&=\; r(s, a) + \gamma \sum_{s'} p(s'|s, a) v_\pi(s') \\ &=\; \sum_{s', r} p(s', r|s, a) \Big[r(s, a) + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \Big]. \end{aligned} \tag{2.3}$$

These equations can also be written in matrix form. The state-value function, for example, can be defined with $\mathbf{v}_\pi$, $\mathbf{r} \in \mathbb{R}^{|\mathcal{S}|}$ and $P_\pi \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$:

$$\mathbf{v}_\pi = \mathbf{r} + \gamma P_\pi \mathbf{v}_\pi = (I - \gamma P_\pi)^{-1} \mathbf{r}, \tag{2.4}$$

where P_π is the state to state transition probability function induced by π, that is, $P_\pi(s, s') = \sum_a \pi(a|s) p(s'|s, a)$.

These algorithms can be roughly divided into two classes: model free and model-based approaches. I further discuss these approaches below.

2.1.1 Model-Free Reinforcement Learning

In the reinforcement learning formalism we assume the agent does not know the matrix P_π nor the function r beforehand. Model-free approaches directly estimate v_π or q_π from samples (s, a, r, s'), without explicitly estimating the transition dynamics or the reward function of the environment. Two traditional model-free approaches in reinforcement learning are Q-Learning (Watkins and Dayan 1992) and Sarsa (Rummery and Niranjan 1994; Sutton and Barto 1998).

Sarsa is an on-policy algorithm, meaning that it estimates the value of the policy currently followed by the agent. Typically, the agent follows an ϵ-greedy policy with respect to the estimates $Q(S_t, A_t)$ of the state-action value function $q_\pi(S_t, A_t)$. Sarsa's update rule is

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big), \tag{2.5}$$

where $\alpha \in [0, 1)$ is the algorithm's step-size.

Sarsa and several other algorithms I discuss in this dissertation are based on temporal difference (TD) learning (Sutton 1988), which learns estimates of the value function by bootstrapping from its current estimate. This is done by updating the prediction at a given time step to bring it closer to the prediction of the same quantity at the next time step according to the TD error, δ_t. In Sarsa the TD error is

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t). \tag{2.6}$$

Q-learning is another traditional RL algorithm. It it is an off-policy algorithm, meaning that it estimates the value of a policy that can be different from the one followed by the agent. The optimal policy, for which the agent is estimating the value, is called the *target* policy while the policy the agent is following is called the *behavior* policy. Q-learning's update rule is

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)\Big), \tag{2.7}$$

where α is, again, the algorithm's step-size.

The policy that is learned by both algorithms is:

$$\pi(s) \doteq \arg\max_{a \in \mathcal{A}} Q(s, a). \tag{2.8}$$

When the value of each state (or state-action pair) is individually stored, this is a *tabular* method. Nevertheless, generalization is required in problems with large state spaces, where it is unfeasible to learn an individual value for each state. This is generally done by parametrizing a function V or Q with a set of parameters $\boldsymbol{\theta}$. We write, given the parameters $\boldsymbol{\theta}$, $V(s; \boldsymbol{\theta}) \approx v_\pi(s)$ and $Q(s, a; \boldsymbol{\theta}) \approx q_\pi(s, a)$. In the past, a common approach was to use linear function approximation where $Q(s, a; \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s, a)$, in which $\boldsymbol{\theta}$ is now a vector of weights and $\boldsymbol{\phi}(s, a)$ denotes a static feature representation of the state s when taking action a. In this case, the update rule is not very different from what I described above. For Sarsa, for example, it is:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha\Big(R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \boldsymbol{\phi}(S_{t+1}, A_{t+1}) - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}(S_t, A_t)\Big). \tag{2.9}$$

Recently, Mnih, Kavukcuoglu, et al. (2015) introduced Deep Q-Network (DQN), which uses a neural network to perform off-policy non-linear function approximation of the value function. The study of algorithms that use neural networks as function approximators has since then been dubbed *deep reinforcement learning*. Deep RL substitutes the requirement of a good handcrafted feature representation by the requirement of an effective network architecture and algorithm. Mnih, Kavukcuoglu, et al., for example, when introducing DQN used a neural network composed of three hidden convolutional layers followed by a fully-connected hidden layer (see Figure 2.1). The network parameters are updated through gradient-descent with the following update rule:

$$\begin{aligned} \boldsymbol{\theta}_{t+1} \;=\; \boldsymbol{\theta}_t + \alpha\Big[R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a; \boldsymbol{\theta}_t^-) \\ - Q(S_t, A_t, \boldsymbol{\theta}_t)\Big] \nabla_{\boldsymbol{\theta}_t}\, Q(S_t, A_t; \boldsymbol{\theta}_t), \end{aligned} \tag{2.10}$$

where $\boldsymbol{\theta}_t^-$ denotes the parameters of a duplicate network, which are updated less often for stability purposes:

$$\boldsymbol{\theta}_t^- = \begin{cases} \boldsymbol{\theta}_t, & \text{if } t \bmod U = 0, \\ \boldsymbol{\theta}_{t-1}^-, & \text{otherwise,} \end{cases}$$

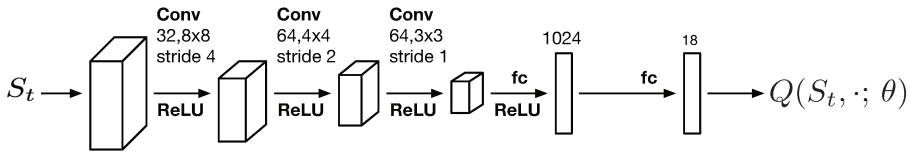


Figure 2.1: Deep Q-Network as per Mnih, Kavukcuoglu, et al. (2015).

where U is a parameter, the target network update frequency. Additional components of the algorithm include clipping the rewards between -1 and 1 and the use of experience replay (Lin 1993) to decorrelate observations. DQN has inspired much follow-up work combining reinforcement learning and deep neural networks, such as the adaptation of other algorithms to the deep RL setting (Hasselt, Guez, and Silver 2016), new approaches for parallelization when using neural networks (Mnih, Badia, et al. 2016), and new sampling strategies for updating the network (Schaul, Quan, et al. 2016). In this dissertation I will focus on DQN when discussing deep RL algorithms.

2.1.2 Model-Based Reinforcement Learning

As an alternative to estimating value functions directly from samples, one can instead use the observed samples to learn a model of the transition dynamics and reward function of the environment. When the transition probability function p and the reward function r are known, we can compute $v_\pi(s)$ recursively by solving the system of equations presented in Equations 2.2 and 2.3, for example.

Such an approach is often thought to be more sample efficient than model-free approaches in exchange for having a higher computational cost. It is also more common that model-based algorithms have stronger theoretical guarantees. Sun et al. (2018), for example, has recently introduced theoretical results confirming some of these claims. Nevertheless, model-based approaches have yet to be successful in large problems due to the difficulty in learning accurate models and the lack of algorithms capable of planning with imperfect models.

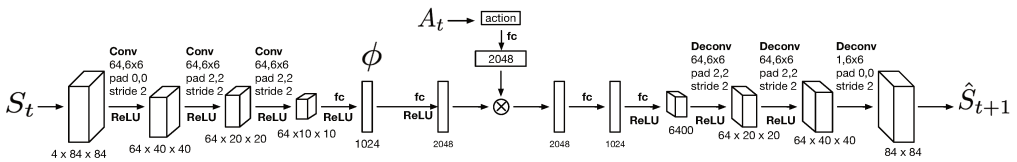


Figure 2.2: Action-Conditional Video Prediction as per Oh et al. (2015). This neural network is capable of learning how to predict grayscale Atari 2600 game screens after hundreds of time steps.

Learning generative models is a very challenging task (Bellemare, Naddaf, et al. 2013; Bellemare, Veness, and Talvitie 2014; Chiappa et al. 2017; Oh et al. 2015). In Atari 2600 games, for example, discussed below, there has been no clear demonstration of successful planning with a learned model. Learned models tend to be accurate for a small number of time steps until errors start to compound (Talvitie 2014). Probably the most successful example of model learning in Atari 2600 games is due to Oh et al. (2015) who designed a neural network capable of learning multistep models that, up to one hundred time steps, appear accurate. These models are able to assist with exploration, an indication of the models' accuracy. However, because of compounding errors, the algorithm still needs to frequently restore its model to the real state of the game. Figure 2.2 depicts the neural network Oh et al. used when learning a model capable of predicting full images as next states.

Planning with an imperfect model is a promising subject of study. Although an error-free model might be unattainable, there is plenty of evidence that even coarse value functions are sufficient for the model-free case (Veness et al. 2015). This result suggests that one might also be able to succeed when the model has flaws. Training set augmentation have shown that it is possible to improve an otherwise limited model (Talvitie 2014; Talvitie 2017; Venkatraman, Hebert, and Bagnell 2015). Similarly, Farahmand, Barreto, and Nikovski (2017) showed that better planning performance could be obtained by using a value-aware loss function when training the model. However, as aforementioned, clear demonstrations of successful planning with a learned model are still rare. Because of that, I present few results in this disserta-

tion that use model-based algorithms. The results I present in Chapter 6 are limited to the tabular case.

2.2 Time-Based Representations

In order to perform function approximation we must have access to a function capable of extracting a representation from the agent's observation (the function ϕ in Equation 2.9, for example). For decades it was quite common to use domain knowledge to handcraft feature vectors that were useful for the problem being solved. Learning feature representations became much more common in the past couple of years after the rise of deep reinforcement learning. In this section I discuss two representation learning methods that pre-date deep reinforcement learning. They learn representations with a particular structure and they are central to this dissertation.

2.2.1 Proto-Value Functions

Proto-value functions (PVFs; Mahadevan 2005) are learned representations that reflect the geometry of the environment. They are basis functions based on the notion of diffusion models (Coifman et al. 2005; Kondor and Lafferty 2002), which are models that capture how information flows in the environment by modeling it as a graph connecting nearby states. PVFs were introduced as a way to capture the underlying environment dynamics in a representation so this representation would be able to properly represent value functions linearly. This is motivated by the fact that value functions can actually be seen as the result of rewards diffusing through the state space, governed by the environment dynamics (Mahadevan and Maggioni 2007).

Formally, proto-value functions are the eigenvectors of a symmetric diffusion operator such as the *combinatorial graph Laplacian* matrix,

$$L = D - W, \tag{2.11}$$

where W is the graph's adjacency matrix and D the diagonal matrix whose entries are the row sums of W. Notice that, in theory, the matrix W can be

extended from a binary matrix to a weight matrix. Different diffusion models can be used to generate PVFs, such as the *normalized graph Laplacian*,

$$L = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}, \tag{2.12}$$

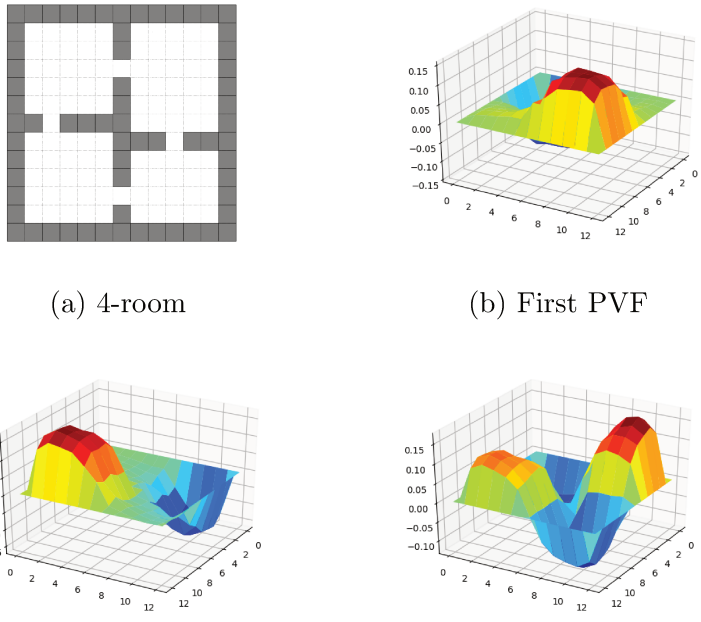
which is what I use in this dissertation.

These diffusion models are tightly related to the random walk diffusion model $D^{-1}W$. One of the reasons the random walk matrix is called a diffusion model is because powers of this matrix determine how quickly the random walk takes to converge to its stationary distribution. A diffusion model works as a surrogate that is easier to estimate than the full transition matrix, while still being useful for value function approximation since the value function can be represented as a linear combination of the eigenvectors of the transition matrix, as suggested in Equation 2.4. See the work by Mahadevan and Maggioni (2007) for a detailed discussion.

Computational reasons justify the use of one of the graph Laplacian matrices instead of simply using the random walk model. The graph Laplacian has a spectral structure that is related to the random walk diffusion model but it is symmetric, making it easier to be diagonalizable. It is fairly easy to see the relationship of the spectral structure of the graph Laplacian and of the random walk model, as Mahadevan and Maggioni (2007) showed:

$$\begin{aligned} L &= D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}WD^{-\frac{1}{2}} \\ I - L &= D^{-\frac{1}{2}}WD^{-\frac{1}{2}} \\ D^{-\frac{1}{2}}(I - L)D^{\frac{1}{2}} &= D^{-1}W \end{aligned}$$

Thus, we can then see that both the normalized graph Laplacian and the random walk matrix have the same eigenvalues. The eigenvectors of the random walk matrix are the eigenvectors of $I - L$ point-wise multipled by $D^{\frac{1}{2}}$. These basis functions are known as proto-value functions because they form "global" basis functions whose support is the entire state space. Consequently, "the set of PVFs form the *building blocks* of all value functions on a state space" (Mahadevan and Maggioni 2007). Importantly, PVFs provide a compact basis set.



(a) 4-room (b) First PVF

(c) Second PVF (d) Third PVF

Figure 2.3: First three PVFs in the 4-room domain (a). This 3D plot depicts the value assigned to the obtained eigenvector to each state. The axes are rotated for clarity, with the bottom left corner of the 4-room domain being the state closer to the reader in Figures b-d.

Intuitively, PVFs capture large-scale temporal properties of an environment, with different eigenvectors capturing different time-scales of diffusion. As value functions, they are "smooth", with the value of each state being a function of its neighbors. Figure 2.3 presents an example to ground this discussion, with the first three PVFs of the 4-room domain depicted (those with the corresponding lowest eigenvalues). Notice how the first two PVFs capture the different diagonals in the environment (bottom right corner to top left corner, for example), with the third one already having a higher frequency by having two peaks instead of one.

Since the introduction of proto-value functions, several approaches have been proposed to scale them to large, discrete, and continuous state spaces. Some of these approaches are the use of the Nyström approximation in the continuous case and the use of factorizations of the environment. Mahadevan and Maggioni (2007) provide a comprehensive early survey on these ap-

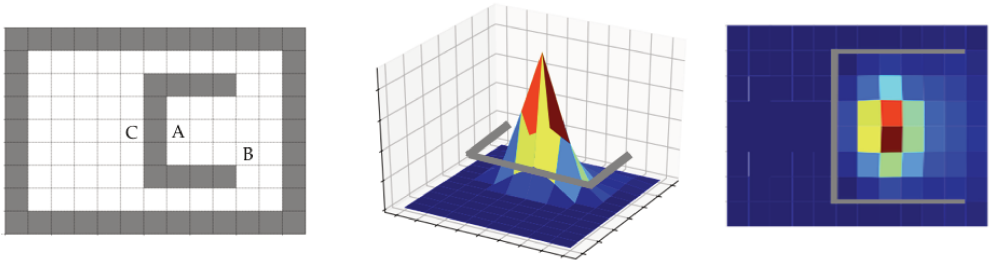


Figure 2.4: Example similar to Dayan's (1993) of the successor representation, with respect to the uniform random policy, of state A (left). The red color represents larger values while the blue color represents smaller values (states that are temporally further away). The Euclidean distance between state A and state C is smaller than the Euclidean distance between state A and state B. However, if one considers the gray tiles to be walls, an agent in state A can reach state B much *quicker* than state C. The SR captures this distinction, ensuring that state A is more similar to state B than it is to state C.

proaches. However, to the best of my knowledge, until the work presented in this dissertation PVFs had only been used as basis functions for value function approximation.

2.2.2 The Successor Representation

The successor representation (SR; Dayan 1993) determines state generalization by how similar successor states are. It is defined to be the current and expected future occupancy of state s' given the agent's policy is π and its starting state is s. It can be seen as defining state similarity in terms of time instead of space, hence the name time-based representation, which was coined by Dayan and that I use in this dissertation. Figure 2.4 contains a concrete example of this concept.

The successor representation with respect to a policy π, Ψ_π, is defined as

$$\Psi_\pi(s, s') = \mathbb{E}_{\pi,p}\left[\sum_{t=0}^{\infty} \gamma^t \mathbb{1}_{\{S_t=s'\}} \middle| S_0 = s\right],$$

where $\mathbb{1}$ denotes the indicator function and $\gamma \in [0, 1)$.

Another interesting property of the successor representation is that its

expectation can be estimated from samples with TD methods:

$$\hat{\Psi}(S_t, j) \leftarrow \hat{\Psi}(S_t, j) + \eta \Big[\mathbb{1}_{\{S_t=j\}} + \gamma \hat{\Psi}(S_{t+1}, j) - \hat{\Psi}(S_t, j) \Big], \quad (2.13)$$

for all $j \in \mathcal{S}$ and with η being the step-size. The successor representation also corresponds to the Neumann series of γP:

$$\Psi_\pi = \sum_{t=0}^{\infty} (\gamma P_\pi)^t = (I - \gamma P_\pi)^{-1}. \quad (2.14)$$

Interestingly, when looking at the equation above we see that the successor representation is actually part of the solution when computing a value function, which was first presented in Equation 2.4:

$$\mathbf{v}_\pi = (I - \gamma P_\pi)^{-1} \mathbf{r} = \Psi_\pi \mathbf{r}.$$

The successor representation is directly related to several other ideas in the field. It can be seen as a form of dual approach to dynamic programming and to value function based methods in reinforcement learning (Wang, Bowling, and Schuurmans 2007). Moreover, the eigenvectors generated from the eigendecomposition of the SR, when it is defined with respect to the uniform random policy in a deterministic and symmetric environment, are equivalent to proto-value functions (Stachenfeld, Botvinick, and Gershman 2014; 2017; Machado, Rosenbaum, et al. 2018) and to slow feature analysis (Sprekeler 2011).

Such equivalences play a central role in the algorithm I describe in Chapter 5. The successor representation may also have an important role in neuroscience. Stachenfeld, Botvinick, and Gershman (2014; 2017) recently suggested that the successor representation is encoded by the hippocampus, and that a low-dimensional basis set representing it is encoded by the enthorhinal cortex. Interestingly, both hippocampus and entorhinal cortex are believed to be part of the brain system responsible for spatial memory and navigation, which is obviously related to the problem of exploration, the main topic of study in this dissertation.

Universal value function approximators (UVFAs; Schaul, Horgan, et al. 2015) are also seemingly related to the successor representation. UVFAs

use function approximation to generalise not just over states but also over goal states. This is different from the successor representation, which can be thought of as generalizing over different reward functions given a fixed policy. Nevertheless, UVFAs lack the theoretical foundation the successor representation has and they are not easily defined in the tabular case. I do not explore UVFAs in this dissertation, although adapting the ideas proposed here to UVFAs might be an interesting future research direction.

While the definitions given for the successor representation so far have been limited to the tabular case, they can also be extended to the function approximation setting. Successor features are the natural extension of the successor representation to the function approximation setting. In this dissertation I use Barreto, Dabney, et al.'s (2017) definition of successor features for the uncontrolled case:

Definition 2.2.1 (Successor Features)**.** Let $\boldsymbol{\psi}_\pi(s) \in \mathbb{R}^d$ denote the successor features of state s when following the policy π. For a given $0 \leq \gamma < 1$ and for a feature representation $\boldsymbol{\phi}(s) \in \mathbb{R}^d$ we have:

$$\boldsymbol{\psi}_\pi(s) = \mathbb{E}_{\pi,p}\left[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{\phi}(S_t) \middle| S_0 = s\right].$$

Alternatively, in matrix form, $\Psi_\pi = \sum_{t=0}^{\infty}(\gamma P_\pi)^t \Phi = (I - \gamma P_\pi)^{-1}\Phi$, where $\Phi \in \mathbb{R}^{|\mathcal{S}| \times d}$ is a matrix encoding the feature representation of each state.

In words, $\psi_{\pi,i}(s)$ encodes the discounted expected value of the i-th feature in the vector $\boldsymbol{\phi}(\cdot)$ when the agent starts in state s and follows the policy π. The update rule presented in Equation 2.13 can be naturally extended to this definition. Importantly, the TD error in the update rule can be used as a differentiable loss function, allowing us to estimate successor features with a neural network, which is something I explore in subsequent chapters of this dissertation.

The successor representation has received a lot of attention recently. One of the results that inspired the work I present here is Kulkarni, Saeedi, et al.'s (2016), which presented a first approach for approximating the successor representation using a neural network. In this work Kulkarni, Saeedi, et al.

briefly show how one can extract bottleneck states as subgoals for options, which I discuss below, as well as how one can use the successor representation to accelerate learning in non-stationary environments where the reward function changes every now and then. Importantly, in Chapter 4 and 5; I use the squared TD-error as a loss function to learn the successor features with a neural network, which is similar to what Kulkarni, Saeedi, et al. proposed. Differently than their work, my neural networks use the definition of the successor representation in terms of states, not state-action pairs. Moreover, my neural networks do not learn a reward model and they do not use an autoencoder to learn a representation of the world.

To the best of my knowledge, the successor representation has never been used for exploration or option discovery, as I propose in this dissertation. Recently it has been extensively studied for transfer learning in reinforcement learning (Barreto, Borsa, et al. 2018; Barreto, Dabney, et al. 2017; Lehnert and Littman 2018; Lehnert, Tellex, and Littman 2017; Ma, J. Wen, and Bengio 2018). It has also been recently used to accelerate learning of general value functions (Sherstan, Machado, and Pilarski 2018) and as a motivation for the generation of new model-based algorithms (Pitis 2018).

2.3 Temporal Abstraction in RL

Sequential decision making usually involves planning, acting, and learning about temporally extended courses of actions over different time scales. In the reinforcement learning framework, *options* are a well-known formalization of the notion of actions extended in time that allow us to represent courses of actions (Sutton, Precup, and Singh 1999). I discuss this formalization below, as well as the most relevant related work to this dissertation.

2.3.1 The Options Framework

An option $\omega \in \Omega$ is a 3-tuple

$$\omega = \langle \mathcal{I}_\omega, \pi_\omega, \mathcal{T}_\omega \rangle,$$

with $\mathcal{I}_\omega \subseteq \mathcal{S}$ denoting the option's initiation set, $\pi_\omega : \mathcal{S} \times \mathcal{A} \to [0,1]$ denoting the option's policy, and $\mathcal{T}_\omega \subseteq \mathcal{S}$ denoting the option's termination set. I consider the *call-and-return* option execution model in which a meta-policy, $\mu : \mathcal{S} \to \Omega$, dictates the agent's behavior (notice $\mathcal{A} \subseteq \Omega$). After the agent decides to follow option ω from a state in $\mathcal{I}_\omega$, actions are selected according to π_ω until the agent reaches a state in $\mathcal{T}_\omega$.

In this dissertation I use the terms meta-policy and high-level policy interchangeably when referring to μ. Moreover, by stating that an agent *follows or takes* an option ω I mean that the agent commits to follow the option's policy, π_ω, until its termination. To distinguish between options and actions, in this dissertation I often refer to the actions originally defined in the problem formulation as primitive actions.

The formal definition of $q_\mu : \mathcal{S} \times \Omega \to \mathbb{R}$ comes from a natural extension of the traditional state-action values (Sutton, Precup, and Singh 1999):

$$\begin{aligned} q_\mu(s,\omega) &= \mathbb{E}\Big[R_{t+1} + \cdots + \gamma^k R_{t+k} \\ &\qquad + \gamma^k \sum_{\omega' \in \Omega} \mu(S_{t+k}, \omega') q_\mu(S_{t+k}, \omega') \mid \mathcal{E}(\omega, s, t)\Big] \\ &= R_\omega(s) + \sum_{s' \in \mathcal{S}} p(s'|s,\omega) \sum_{\omega' \in \Omega} \mu(s', \omega') q_\mu(s', \omega'), \end{aligned} \tag{2.15}$$

with $\mathcal{E}(\omega, s, t)$ being the event of ω being initiated in state s at time t, $R_\omega(s)$ denoting the expected discounted sum of rewards observed while executing option ω given $\mathcal{E}(\omega, s, t)$, and $t + k$ being the random time at which ω terminates. In this definition, γ is folded into the state-prediction part of the equation, $p(s'|s,\omega)$, such that

$$p(s'|s,\omega) = \sum_{k=1}^{\infty} p(s', k|s, \omega)\gamma^k, \tag{2.16}$$

where $p(s', k|s, \omega)$ is the probability that the option ω terminates in s' after k steps (Sutton, Precup, and Singh 1999).

The formalism I used above to introduce options is more restrictive than the original one. Instead of a termination set, Sutton, Precup, and Singh (1999) defined options with a termination condition, $\beta(S_t)$, denoting the probability

the option terminates in the state the agent is currently in, S_t. I restrict $\beta(S_t) \in \{0, 1\}$ instead. Additionally, although it is becoming increasingly popular, the call-and-return option execution model I use has limitations. In the call-and-return execution model, once an option starts to be executed it cannot be interrupted, regardless of what the agent observes or the rewards it receives. This is clearly not ideal. Although beyond the scope of this dissertation, the algorithms I introduce here could be augmented to support interruption (Sutton, Precup, and Singh 1999). The options I introduce in the next chapters, when taking the reward into consideration, could be interrupted when there is an option $\omega' \in \Omega$ such that

$$q_\mu(S_t, \omega') > q_{\pi_\omega}(S_t, a), \quad \forall a \in \mathcal{A}, \tag{2.17}$$

where $\omega \in \Omega$ is the option being executed before interruption. It is also important to acknowledge that only recently the idea of actually "executing" an option has become popular. In the past options were mostly used for faster credit assignment.

2.3.2 Related Work

The literature related to options in reinforcement learning is vast. In this section I discuss only the most relevant work related to *option discovery*, the subfield concerned with the problem of autonomously identifying good options – i.e., the elements of the tuple $\langle \mathcal{I}_\omega, \pi_\omega, \mathcal{T}_\omega \rangle$.

Traditionally, options capable of moving agents to *bottleneck* states have been sought after. Bottleneck states are those states that connect different densely connected regions of the state space (e.g., doorways; McGovern and Barto 2001; Şimşek and Barto 2004; Solway et al. 2014). They have been shown to be efficient for planning as these states are the states most frequently visited when considering the shortest distance between any two states in an MDP (Solway et al. 2014). In Chapter 4 I discuss this objective in detail and I show a setting in which bottleneck options hurt agent's performance.

Most algorithms for option discovery can be seen as *top-down* approaches.

Agents use successful trajectories that lead to non-zero rewards[1] as a starting point, decomposing and refining them into options. There are many approaches based on this principle, such as methods that use the observed rewards to generate intrinsic rewards leading to new value functions (e.g., McGovern and Barto 2001; Menache, Mannor, and Shimkin 2002; Konidaris and Barto 2009), methods that use the observed rewards to climb a gradient (e.g., Mankowitz, Mann, and Mannor 2016; Vezhnevets, Mnih, et al. 2016; Bacon, Harb, and Precup 2017), or to do probabilistic inference (Daniel et al. 2016; Fox et al. 2017). However, such approaches are not applicable in large state spaces with sparse rewards. If non-zero rewards are unlikely to be found by an agent using only primitive actions, requiring long or specific sequences of actions, options are equally unlikely to be discovered.

The approaches I propose in this dissertation can be seen as *bottom-up* approaches, in which options are constructed without taking the rewards generated by the environment into consideration. Options discovered this way tend to be task-independent, in the sense that they are potentially useful in many different tasks (Eysenbach et al. 2019; Gregor, Rezende, and Wierstra 2016). Such options can also be seen as being useful for exploration by allowing agents to commit to a behavior for an extended period of time, as I discuss in the next chapter. Importantly, other results that became available mostly after or simultaneously to the work I developed in this dissertation have also shown evidence of the usefulness of options for exploration (e.g., Eysenbach et al. 2019; Fox et al. 2017).

Among the approaches to discover options without using the rewards generated by the environment are the use of global or local graph centrality measures (Chaganty, Gaur, and Ravindran 2012; Şimşek and Barto 2004; Şimşek and Barto 2008; Şimşek, Wolfe, and Barto 2005), clustering of states (Bacon 2013; Fox et al. 2017; Lakshminarayanan et al. 2016; Mannor et al. 2004) and diversity between terminal states (Eysenbach et al. 2019). Interestingly,

[1]For clarity and ease of presentation I focus the presentation on tasks that are defined such that the agent observes a reward signal of value zero until reaching a goal. This discussion is equally applicable to other settings with sparse rewards, such as tasks where the agent observes a reward of -1 until the episode is over.

Şimşek, Wolfe, and Barto (2005) and Lakshminarayanan et al. (2016) also use the graph Laplacian in their algorithm, as I do in this dissertation. However, they use the graph Laplacian to identify bottleneck states.

The idea of explicitly using options for exploration is much less explored. Chaganty, Gaur, and Ravindran (2012) show how options can be used to create a small world effect into the environment's underlying graph, while Baranes and Oudeyer (2013) and Moulin-Frier and Oudeyer (2013) show how one can build policies to explicitly assist agents to explore the environment. The proposed algorithms self-generate subgoals in order to maximize learning progress. The policies built can be seen as options. Solway et al. (2014) formally proved that "optimal hierarchy minimizes the geometric mean number of trial-and-error attempts necessary for the agent to discover the optimal policy for any selected task". My experiments confirm this result, although I propose *diffusion time* as a different metric to evaluate how options improve exploration.

The idea of discovering options by learning to control parts of the environment is also related to my work. The options I discover encode different rates of change in the agent's representation of the world, while the corresponding policies aim at maximizing such change. Others have also proposed ways to discover options by learning to control the environment. Hengst (2002), for instance, proposes an algorithm that explicitly models changes in the variables that form the agent's representation. Gregor, Rezende, and Wierstra (2016) proposed an algorithm in which agents discover options by maximizing a notion of empowerment (Salge, Glackin, and Polani 2014), where the agent aims at getting to states with a maximal set of available intrinsic options.

Continual Curiosity driven Skill Acquisition (CCSA; Kompella, Stollenga, et al. 2017) is the closest approach to eigenoptions, which I introduce here. CCSA also discovers skills that maximize an intrinsic reward obtained by some extracted representation. While I use PVFs or the successor representation, CCSA uses Incremental Slow Feature Analysis (SFA; Kompella, Luciw, and Schmidhuber 2011) to define the intrinsic reward function. Sprekeler (2011) has shown that, given a specific choice of adjacency function, PVFs are equiv-

alent to SFA (Wiskott and Sejnowski 2002). SFA becomes an approximation of PVFs if the function space used in the SFA does not allow arbitrary mappings from the observed data to an embedding. My algorithms differ in how I define the initiation and termination sets, as well as in the objective being maximized. CCSA acquires skills that produce a large variation in the slow-feature outputs, leading to options that seek bottlenecks. The algorithms I propose do not seek bottlenecks, focusing on traversing different directions of the learned representation.

Another approach directly related to this dissertation is FeUdal Networks (Vezhnevets, Osindero, et al. 2017). At the same time I was developing the work presented in Chapter 4, Vezhnevets, Osindero, et al. (2017) also proposed to explicitly build hierarchies based on the learned latent representation of the state space. Their work is inspired by feudal reinforcement learning (Dayan and Hinton 1992), an approach that generates a hierarchy of policies by having managers dispatching jobs to workers. Similar to the work I present here, Vezhnevets, Osindero, et al. (2017) explicitly encourage the agent to traverse directions of a latent representation of the environment. They do so by using the cosine similarity between the goal state, defined by a "manager" module, and the direction, in the latent space, that the agent navigates. Nevertheless, Vezhnevets, Osindero, et al. (2017) do not explicitly build options with initiation and termination sets. Instead, they learn a hierarchy through an end-to-end learning system that does not allow us to easily retrieve options from it.

Finally, Scheduled Auxiliary Control (SAC) and Visual Reinforcement Learning with Imagined Goals (RIG) are a two recent algorithms that came after much of the work in this dissertation that propose similar ideas. SAC (Riedmiller et al. 2018) suggests that agents should learn policies to maximize an auxiliary task (e.g., to get two objects within a given distance, or to maximize the translation velocity sensor), and that these learned sub-policies should be used to drive exploration. Similar to the results I present in the next chapters, they showed that randomly selecting between sub-policies does improve exploration. Differently from my work, they did not formalize their

approach in terms of options and, more importantly, they still rely on hand-coded auxiliary tasks that, although general, do start with some semantics about the task (e.g., proximity, relative position of objects, etc). The second approach, RIG (Nair et al. 2018), proposes agents should learn a representation with a β-VAE (Higgins et al. 2017) and then use this learned representation to pose goals. It does not specifically implement options but goal-conditioned policies. RIG poses goals by sampling from the β-VAE and then learns how to achieve those goals. This process is said to generate skills that are useful when a task is posed in the form of a reward function, giving agents the purposeful exploration I talk about in this dissertation. In a high-level, both algorithms can be seen as an implementation of the option discovery cycle I discuss in the next chapter.

2.4 Exploration in Reinforcement Learning

Reinforcement learning algorithms often are not only responsible for learning accurate value estimates from the observed samples but also for determining the sample collection process. Ideally, RL algorithms should be able to traverse the state space to obtain distinct and informative samples as fast as possible. Nevertheless, it is not clear how to effectively collect these samples as the agent starts with no information about the environment it is in. This is the problem of exploration. Maximizing the return in environments with sparse rewards is one of the problems that exemplify the importance of good exploration strategies. Until the agent observes a non-zero reward it does not know that it is possible to achieve a return that is greater than zero. Therefore, it is of the agent's interest to explore the environment as fast as possible to ensure it will observe samples with non-zero rewards.

The existing approaches for exploration in model-based and model-free RL have very different flavours. In model-based RL we often have theoretical guarantees (e.g., Brafman and Tennenholtz 2002; Kearns and Singh 2002; Strehl and Littman 2008) that allow us to understand what are reasonable expectations about an algorithm in terms of sample complexity. Nevertheless,

the algorithms derived from these results have not been as succesful in larger problems where function approximation is required, with few exceptions being known (e.g., Keramati et al. 2018). Model-free approaches are mostly heuristic, but have been extremely succesful in the past (e.g. Bellemare, Srinivasan, et al. 2016; Burda et al. 2019; Ostrovski et al. 2017). Few strong theoretical results are known for model-free approaches (Strehl, Li, et al. 2006). I further discuss both settings below.

2.4.1 Theoretical Framework for Exploration

PAC-MDP (Kakade 2003) is the best-known theoretical framework in which exploration is studied. It is an adaptation of the PAC framework, originally formulated to study supervised learning algorithms (Valiant 1984). PAC stands for *probably approximate correct* and, in the supervised learning setting, its goal is to select, with high-probability (*probably*), a function that is accurate most of the time (*approximate correct*). When adapted to the reinforcement learning problem, the PAC-MDP framework can be informally described as estimating the number of times an agent executes sub-optimal actions while learning to maximize the return.

Because the agent's actions impact the samples it observes (states and rewards), there is not an obvious sampling model to be used when defining a PAC-MDP algorithm. Different models have been proposed. The model with stronger assumptions relies on a generative model that allows the agent to know the reward and the next state it will observe when acting in any particular state (Kearns, Mansour, and Ng 1999). The μ-reset model has weaker assumptions. It forces the agent to perform an online simulation of its experience, but it allows the agent to reset the current state to a state S_0 drawn according to the distribution μ (Kakade and Langford 2002). Finally, the model that is closer to the general reinforcement learning problem assumes the agent only has access to an online simulation of the MDP (Kakade 2003). The formal definition of a PAC-MDP algorithm and the notion of *approximate correctness* is presented below.

Definition 2.4.1 (from Strehl and Littman 2008)**.** Let $c = (s_1, a_1, r_1, s_2, a_2, r_2, \ldots)$ be a path generated by executing an algorithm $\mathcal{A}$ in an MDP $\mathcal{M}$. For any fixed $\epsilon > 0$, the sample complexity of exploration (sample complexity, for short) of $\mathcal{A}$ with respect to c is the number of timesteps t such that the policy at time t, $\pi_{\mathcal{A}_t}$, is not ϵ-optimal from the current state S_t at time t (formally, $v_{\pi_{\mathcal{A}_t}}(S_t) < v_*(S_t) - \epsilon$).

Definition 2.4.2 (from Strehl and Littman 2008)**.** An algorithm $\mathcal{A}$ is said to be an efficient PAC-MDP (Probably Approximately Correct in Markov Decision Processes) algorithm if, for any ϵ and δ, the per-step computational complexity and the sample complexity of $\mathcal{A}$ are less than some polynomial in the relevant quantities $\big(|\mathcal{S}|, |\mathcal{A}|, 1/\epsilon, 1/\delta, 1/(1-\gamma)\big)$, with probability at least $1 - \delta$. For convenience, we may also say that $\mathcal{A}$ is PAC-MDP.

As aforementioned, the algorithms proposed in this theoretical framework are rarely used in practice. In large problems, the explicit requirement of visiting *all* states a polynomial number of times is prohibitive.

There is also an extension of the PAC-MDP formalism to options, presenting a formal analysis of how options can affect and benefit the sample complexity of reinforcement learning algorithms (Brunskill and Li 2014). This extension is summarized by the definition below:

Definition 2.4.3 (from Brunskill and Li 2014)**.** Given history h_t at epoch t, an RL algorithm is viewed as a non-stationary policy, denoted $\mathcal{A}_t$. For any fixed ϵ, the sample complexity of exploration (or "sample complexity") of $\mathcal{A}$ is $\sum_t \tau_t \cdot \mathbb{1}_{\{v_{\mathcal{A}_t}(S_t) \leq v_*(S_t) - \epsilon\}}$, where $\mathbb{1}_{\{C\}}$ is the set-indicator function that evaluates to 1 if event C occurs and 0 otherwise.

In the definition above, $h_t \doteq (s_1, a_1, \tau_1, r_1, s_2, a_2, \tau_2, r_2, \ldots)$, where τ_t denotes the "waiting time" an option (i.e., the expected number of time steps until it terminates), and C and L are parameters of its assumed distribution. Brunskill and Li (2014) then proceed to define PAC-SMDP (Probably Approximately Correct in SMDPs) algorithms just as in Definition 2.4.2, but using the notion of sample-complexity they introduced.

An important result from Brunskill and Li (2014) that is directly related to my work is the upper bound they prove for their algorithm, SMDP-Rmax:

Theorem 2.4.1 (from Brunskill and Li 2014)**.** SMDP-Rmax *is PAC-SMDP with a sample complexity of, ignoring logarithmic terms,*

$$\tilde{\mathcal{O}}\left(\frac{v_{\max}^3}{\epsilon^3}\sum_{s,a}\frac{N_{sa}}{(1-\bar{\gamma}_s)^2}\left(\frac{1}{1-\gamma}+L+\frac{1}{\sqrt{C}}\right)\right),$$

where N_{sa} *is the number of reachable next states of* (s,a)*,* $\bar{\gamma}_s = \max_a \sum_\tau \gamma^\tau P(\tau|s,a)$*, and* $P(\tau|s,a) = \sum_{s'} P(s',\tau|s,a)$ *is the marginal waiting-time distribution.*

The result above suggests that, to be sample efficient, one should keep the number of state-action tuples small. However, the results I present in the next chapters show how I am able to improve exploration by adding a large number of options to the agents' action set. There is no contradiction between these results. In my work I only use options to define an exploration policy, without having to actually learn expected returns for each state-option pair.

In this dissertation I will not provide PAC-MDP guarantees for my algorithms, although this is an interesting future direction. I use the PAC-MDP formalism in Chapter 6 when discussing algorithms I used as baseline.

2.4.2 Pseudo-Counts for Practical Exploration in RL

Traditionally, in practice, exploration in RL algorithms is done through random action selection (ϵ-greedy strategies), or with simple heuristics such as optimistic initialization (Sutton and Barto 1998) and exploration bonuses (Sutton 1990). Recently, these approaches have been revisited and extensions to the function approximation case have been proposed (e.g., Bellemare, Srinivasan, et al. 2016; Machado, Srinivasan, and Bowling 2015; Ostrovski et al. 2017; Tang et al. 2016). Among these approaches, and many others, the use of exploration bonuses generated from pseudo-counts is probably the approach that has led to the most impressive results in large domains such as Atari 2600 games.

Introduced by Bellemare, Srinivasan, et al. (2016), pseudo-counts are a generalization of count-based exploration to the function approximation setting. Count-based exploration consists in directly using visit counts to guide an agent towards states it has visited least often in order to reduce uncertainty. Such an approach is the crux of most successes in the bandits literature (e.g., Auer, Cesa-Bianchi, and Fischer 2002) and has also led to RL algorithms with PAC-MDP guarantees (Strehl and Littman 2008).

Pseudo-counts are defined in terms of a density model, ρ, on the state space. Let $\rho_n(S)$ be the probability assigned to S after training the model on the sequence of states $S_1, \ldots, S_n$. The recoding probability $\rho'_n(S)$ is defined to be the probability the model would assign to S if it were trained on that same observation one more time. Given some assumptions, pseudo-counts are defined as

$$\hat{N}_n(S) = \frac{\rho_n(S)(1 - \rho'_n(S))}{\rho'_n(S) - \rho_n(S)}. \tag{2.18}$$

This definition comes from the expectation that, "after observing one instance of S, the density model's increase in prediction of that same S should correspond to a unit increase in pseudo-count" (Bellemare, Srinivasan, et al. 2016), such that:

$$\rho_n(S) = \frac{\hat{N}_n(S)}{\hat{n}} \qquad\qquad \rho'_n(S) = \frac{\hat{N}(S) + 1}{\hat{n} + 1}, \tag{2.19}$$

where $\hat{n}$ is the pseudo-count total. Under certain assumptions on the density model, it can be shown that pseudo-counts do grow approximately linearly with real counts. Because of that, Bellemare, Srinivasan, et al. (2016) have proposed the use of R_n^+ as a reward bonus, with

$$R_n^+(S) = \frac{1}{\sqrt{\hat{N}_n(S) + \epsilon}}, \tag{2.20}$$

where ϵ is a small constant value (e.g., 0.001).

The main caveat of such a simple approach is that the density model aforementioned is domain-dependent. In video-games, for example, it has been shown that both CTS (Bellemare, Srinivasan, et al. 2016) and PixelCNN (Ostrovski et al. 2017) are useful, reaching, at the time, some of the highest scores

ever seen in Atari 2600 games such as Montezuma's Revenge. Nevertheless, these density models are often complicated and hard to implement, making these results hard to replicate. Moreover, it is not clear what density model should be used when trying to apply this approach to a new task. The approach I propose in Chapter 6 is very related to this line of work as it can also be seen as count-based method. In a sense, it uses the norm of the successor representation as a pseudo-count. Importantly, the successor representation is a much simpler object and it is defined for any reinforcement learning problem. In Chapter 6 I use Bellemare, Srinivasan, et al. (2016)'s and Ostrovski et al. (2017)'s results as baseline.

2.4.3 Related Work

R-Max is the model-based algorithm most related to the model-based algorithm I introduce in Chapter 6. R-Max is a PAC-MDP algorithm that augments the state-space with an imaginary state and encourages the agent to visit that state by assuming its expected reward is the largest possible. By encouraging the agent to visit states visited least often R-Max is implicitly reducing the algorithm's uncertainty in the state-space. R-Max deletes the transition to this imaginary state once a state has been visited a given number of times. However, as aforementioned, it is not clear how to apply model-based RL algorithms such as R-Max, without additional domain knowledge, to large domains where function approximation is required.

Conversely, there are multiple model-free algorithms for exploration that actually work in large domains. Multiple other approaches have surfaced recently, such as the use of Thompson Sampling for exploration (Osband, Blundell, et al. 2016; Osband, Roy, and Z. Wen 2016), perturbations on the function approximator (Fortunato et al. 2018; Plappert et al. 2018), or the use of prediction errors as exploration bonuses (Burda et al. 2019; Stadie, Levine, and Abbeel 2015). Nevertheless, most of these approaches have obtained only partial success, failing in tasks with sparse rewards in large domains such as the Atari 2600 game Montezuma's Revenge.

Aside from the work by Bellemare, Srinivasan, et al. (2016) and Ostrovski

et al. (2017), it is important to mention the work by Martin et al. (2017), which showed that counting activations of fixed, handcrafted features in Atari 2600 games leads to good exploration behavior. This is relevant because, as I will discuss in later chapters, some of the algorithms I propose can be seen as not only counting *learned* features but also implicitly capturing the induced transition dynamics.

It is also worth mentioning some recent work which became available after I had developed much of the work in this dissertation. Burda et al. (2019) presents impressive results in Atari 2600 games by introducing a training process that distills a randomly initialized (target) network into a trained network and uses the prediction error as an exploration bonus. The intuition behind such an approach is that the prediction error is low on states that are similar to states the agent already visited. In Chapter 6 I use Burda et al.'s work as a baseline and I show that under a low sample-complexity regime the algorithm I introduce outperform theirs, dubbed Random Network Distillation.

Finally, another work related to what I discuss here was developed by Savinov et al. (2019), who suggest that agents should explore the environment by visiting states that they believe are far from the states they have already observed. They implemented this idea using an episodic memory with a neural network predicting how far a given state is from the states in this memory. This work is related to mine because the approaches I propose for option-based exploration can be seen as trying to move the agent to states that are far from the states it often sees. Instead of a neural network that learns a distance metric in the embedded state I use a singular value decomposition to estimate frequency of observation, as I discuss in Chapters 4 and 5.

2.5 The Arcade Learning Environment

The Arcade Learning Environment (ALE) is both a challenge problem and a platform for evaluating general competency in artificial intelligence. Originally proposed by Bellemare, Naddaf, et al. (2013), the ALE makes available dozens of Atari 2600 games for agent evaluation. The agent is expected to do

well in as many games as possible without game-specific information, generally perceiving the world through a video stream. Atari 2600 games are excellent environments for evaluating AI agents for three main reasons: 1) they are varied enough to provide multiple different tasks, requiring general competence, 2) they are interesting and challenging for humans, and 3) they are free of experimenter's bias, having been developed by an independent party. Therefore, the ALE is the main platform I use in this dissertation when evaluating the scalability and generality of my algorithms with function approximation.

I also use the MDP formalism when tackling Atari 2600 games. In the context of the ALE, an action is the composition of a joystick direction and an optional button press. There are two possible settings: (1) to use the game-dependent minimal action set where only the actions that do have an effect in that particular game are available, or the (2) full action set, where the 18 actions are available in all games. The agent observes a reward signal, which is typically the change in the player's score (the difference in score between the previous time step and the current time step), and an observation $O_t \in \mathcal{O}$ of the environment. This observation can be a single 210×160 image and/or the current 1024-bit RAM state. Because a single image typically does not satisfy the Markov property, I distinguish between observations and the environment state, with the RAM data being the real state of the emulator.[2] A frame (as a unit of time) corresponds to 1/60th of a second, the time interval between two consecutive images rendered to the television screen. Initial versions of the ALE were deterministic: given a particular emulator state, s, and a joystick input, a, there is a unique resulting next state, s'. In other words, $p(s' \mid s, a) = 1$. This is the setting in which I evaluate my algorithms in Chapter 4 and 5. The latest version of the ALE, released by Machado, Bellemare, Talvitie, et al. (2018), introduces a form of stochasticity known as sticky actions, which was used in Chapter 6. In sticky actions there is a *stickiness* parameter ς, the probability at every time step that the environment will execute the agent's previous action again, instead of the agent's new action. More specifically, at

[2]The internal emulator state also includes registers and timers, but the RAM information and joystick inputs are sufficient to infer the next emulator state.

time step t the agent decides to execute action a; however, the action A_t that the environment in fact executes is:

$$A_t = \begin{cases} a, & \text{with prob.} \quad 1-\varsigma, \\ a_{t-1}, & \text{with prob.} \quad \varsigma. \end{cases}$$

In other words, if $\varsigma = 0.25$, there is 25% chance the environment will not execute the desired action right away.

Agents interact with the ALE in an episodic fashion. An episode begins by resetting the ALE to its initial configuration, and ends at a natural endpoint of a game's playthrough (this often corresponds to the player losing their last life). The primary measure of an agent's performance is the score achieved during an episode, namely the undiscounted sum of rewards for that episode. While this performance measure is quite natural, it is important to realize that score, in and of itself, is not necessarily completely correlated to the human notion of progress. In some games, agents can maximize their score by "getting stuck" in a loop of "small" rewards, ignoring what human players would consider to be the game's main goal. Nevertheless, score is currently the most common measure of agent performance.

Beyond the minimal interface described above, almost all agents designed for the ALE implement some form of reward normalization. The magnitude of rewards can vary wildly across games; transforming the reward to fit into a roughly uniform scale makes it more feasible to find game-independent meta-parameter settings. For instance, some agents divide every reward by the magnitude of the first non-zero reward value encountered, implicitly assuming that the first non-zero reward is "typical" (e.g., Bellemare, Naddaf, et al. 2013; Liang et al. 2016). Others account only for the sign of the reward, replacing each reward value with -1, 0, or 1, accordingly (e.g., Mnih, Kavukcuoglu, et al. 2015). In this dissertation I constrain the rewards between -1 and 1, also allowing real-valued rewards. I do this by having all values smaller than -1 being set to -1 and all values greater than 1 being set to 1.

Most agents also employ some form of hard-coded preprocessing to simplify the learning and acting process. I briefly review the three most common preprocessing steps. 1) *Frame skipping* (Naddaf 2010) restricts the agent's

decision points by repeating a selected action for k consecutive frames. Frame skipping results in a simpler reinforcement learning problem and speeds up execution; values of $k = 4$ and $k = 5$ have been commonly used in the literature. 2) *Color averaging* (Bellemare, Naddaf, et al. 2013) and *frame pooling* (Mnih, Kavukcuoglu, et al. 2015) are two image-based mechanisms to flatten two successive frames into a single one in order to reduce visual artifacts resulting from limitations of the Atari 2600 hardware – by leveraging the slow decay property of phosphors on 1970s televisions, objects on the screen could be displayed every other frame without compromising the game's visual aspect (Montfort and Bogost 2009). Effectively, color averaging and frame pooling remove the most benign form of partial observability in the ALE. Finally, 3) *frame stacking* (Mnih, Kavukcuoglu, et al. 2015) concatenates previous frames with the most recent in order to construct a richer observation space for the agent. Frame stacking also reduces the degree of partial observability in the ALE, making it possible for the agent to detect the direction of motion in objects. Importantly, one can stack *and* skip frames. When this is done, the frames that are skipped are also not add to the stack.

It is not possible to concisely summarize here all the work that have used the ALE as a testbed since its introduction. In a paper that is not part of this dissertation I present a survey with a (still incomplete) discussion about related work (Machado, Bellemare, Talvitie, et al. 2018). I also present clear guidelines on how to evaluate and report results using the Arcade Learning Environment to ensure that they are comparable to other approaches and that they are reproducible. These guidelines are the ones followed when generating the results I report in Chapter 6, where I revisit the used methodology.

Chapter 3

Option-Based Exploration

The first contribution in this dissertation is the proposal of option-based exploration, the idea that agents can explore the environment more effectively if they use options to operate at a higher-level of abstraction. When following options, agents exhibit more decisive behavior in constrast to the aimless dithering and lack of intentionality commonly observed with primitive actions chosen uniformly at random. Intuitively, if one needs to explore the level of a building, it makes more sense to do so in terms of rooms than in terms of motor twitches. In this chapter I discuss the general strategy I propose for option discovery and in Chapter 4 and 5 I present algorithms that develop this intuition.

3.1 Option Discovery in RL

Regardless of whether the intuition presented above is correct or not, it is important to acknowledge that autonomous option discovery is one of the biggest open problems in reinforcement learning. Therefore, in order to defend the idea of option-based exploration I will first discuss a general approach that is capable of discovering options. Importantly, because I am advocating for options that improve exploration, the proposed approach cannot rely on first observing non-zero rewards to start discovering options. The proposed method for discovery should follow a constructivist approach (Piaget 1963), where increasingly more complex options are discovered until the desired behavior is

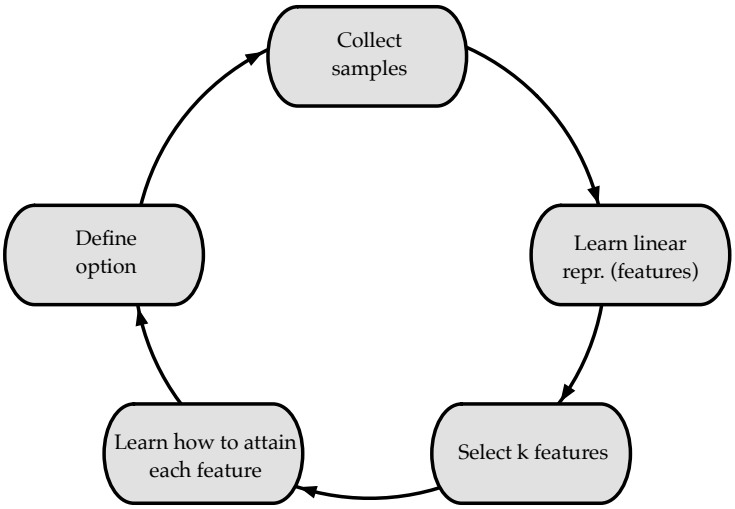


Figure 3.1: Option discovery cycle.

learned.

The general idea behind the algorithms I propose in this dissertation is summarized in Figure 3.1. This figure depicts the option discovery cycle and it tries to highlight the main steps an algorithm needs to go through in order to discover options. The two most important ideas in this diagram is that the option discovery process is guided by the process of learning a representation, and the fact that it is a cycle. I depict this process as a cycle because options discovered early in the agent's life should be used to bootstrap the option learning process. I believe successful algorithms in the future will be able to simultaneously discover representations and options. Agents will use their learned representation to discover options, which will be used to further explore the environment, improving the agent's representation. In the next chapters I present some first instantiations of this idea.

Below I discuss each one of the steps in Figure 3.1. Importantly, while I present these steps sequentially, and all the algorithms in this dissertation that implement the option discovery cycle execute these steps sequentially, it does not have to be. One could also imagine all these steps being executed concurrently, at different time scales.

3.1.1 Collect Samples

The first step in each iteration of the proposed option discovery cycle is to have the agent follow a pre-specified policy while collecting data in the form of observations. Because at the start of the agent's life it has no information about the environment it is in, selecting actions uniformly at random is an obvious first choice. At every new iteration of this cycle a new policy can be defined. Once the agent starts to be able to follow options' policies, there are more possibilities. While in this dissertation I show that selecting actions uniformly at random between the primitive actions and the option's policy is an effective strategy for exploration, one could also imagine more elaborate strategies such as selecting options with probability inversely proportional to the number of time steps the option is expected to last.

3.1.2 Learn Linear Representation

While acting in the world, the agent should learn a representation of its environment. There are multiple ways of doing so and throughout this dissertation I discuss a couple of approaches. I decided to specifically focus on time-based representations that naturally capture the dynamics of the environment, as discussed in the previous chapter, but this is not a requirement. Moreover, although I labeled this step as learning a *linear* representation, the actual requirement is that the learned representation is a vector. Therefore, the output of specific layers of a neural network is also a valid representation. In order to be able to fully explore this framework it is better if the agent does not depend on observing rewards to learn such a representation. Autoencoders (Bengio et al. 2006) and UNREAL (Jaderberg et al. 2017) are some of the many existing methods that could potentially be directly used here.

3.1.3 Select k Features

After a representation is learned, the agent should select a given number of features, k, it wants to learn to maximize or to minimize. Obviously, k could be as large as the total number of features, but it is likely that some features

simply capture noise and in that case a smaller k might be more beneficial. All the algorithms I introduce in this dissertation use singular value decomposition (or eigendecomposition) to sort features by how much they vary over time, making the task of selecting a subset of features easier.

3.1.4 Learn How to Attain each Feature

Once a representation has been learned and a subset of features has been selected, the agent needs to learn to attain those features. Each feature will generate a different policy. This step can be done in parallel with an off-policy learning algorithm, for example. The learned policy will be the option's policy. I decided to not specify here how the agent should tackle the problem of learning to attain a feature. I do so in in Chapter 4, where I introduce the concept of eigenpurpose, which is the reward function I use in the algorithms in this dissertation.

3.1.5 Define Option

Finally, after the options' policies have been learned, the last step is to properly define the option. As discussed in the previous chapter, an option consists of a policy, an initiation set, and a termination condition. While the option's policy has been learned in the previous step, the initiation and termination sets have not been defined yet. One could imagine defining the terminal condition of an option based on the value of the feature being maximized (or minimized), either via a threshold or as a function of the value of the feature. In the next chapter I introduce a theoretically sound approach to define option termination when maximizing eigenpurposes.

As discussed above, the options learned with this approach are defined even in the absence of rewards. Intuitively, when the agent observes a change in the environment through its feature representation, one of these options can learn a policy capable of reproducing that change. When such an option becomes available to the agent, the agent now can move farther in the state-space, with some events that were rare now becoming frequent and events that were "impossible" now becoming "just" infrequent.

3.2 Overview of the Proposed Algorithms

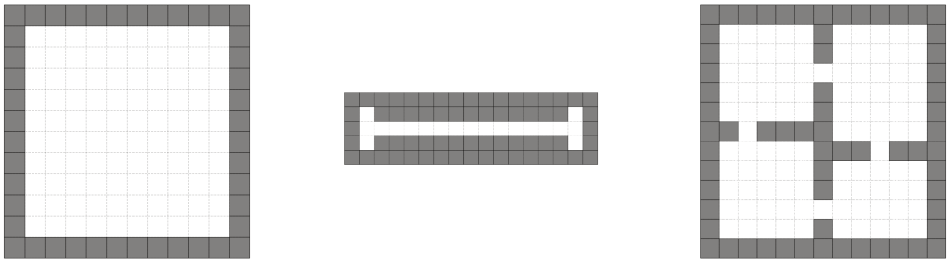
Chapter 4, 5, and 7 contain different instantiations of the option discovery cycle introduced here. The algorithms introduced in this dissertation start with a policy that selects actions randomly. If only primitive actions are available, they are selected uniformly at random. If options have been learned, the agent selects uniformly at random between all of the options and primitive actions. If an option is selected, its policy is followed until termination. While the agent is acting in the world it learns a time-based representation (proto-value functions or the successor representation). These representations can be learned in multiple ways, as I discuss throughout this document.

Chapter 4

Option-Based Exploration with Proto-Value Functions

In this chapter I introduce my first algorithm capable of option-based exploration. As discussed in the previous chapter, this algorithm implements the option discovery cycle, using the representation learning process to drive option discovery. More specifically, in this chapter I show how proto-value functions (PVFs), a particular form of learned representation, can be seen as implicitly defining purposes for the agent. I do so by introducing the concepts of *eigenpurpose* and *eigenbehavior*. Eigenpurposes are intrinsic reward functions that incentivize the agent to traverse the state space by following the principal directions of the learned representation. Each intrinsic reward function leads to a different eigenbehavior, which is the optimal policy for that reward function. I term the options discovered with this approach *eigenoptions*. Importantly, they are task-independent because, as PVFs, eigenpurposes are obtained without any information about the environment's reward structure.

Aside from being task independent, eigenoptions have two important properties that allow them to improve exploration: 1) they operate at different time scales, and 2) they can be easily sequenced. Having options that operate at different time scales allows agents to make finely timed actions while also decreasing the likelihood the agent will explore only a small portion of the state space. Moreover, because these options are defined across the whole state space, multiple options are available in every state, which allows them to be easily sequenced.



(a) 10×10 grid (b) I-Maze (c) 4-room domain

Figure 4.1: Tabular domains used for evaluation in this chapter.

The empirical evaluation of this idea in the tabular case suggests that eigenoptions can reduce the expected number of decisions an agent needs to make in order to visit every state at least once. Consequently, once the agent has access to these options, it can learn how to maximize rewards much faster than if it is using only primitive actions. Finally, at the end of this chapter I provide anecdotal evidence of the potential applicability of this idea to settings where value functions are approximated through linear functions.

4.1 Option Discovery through the Laplacian

As discussed in Chapter 2, PVFs capture the large-scale geometry of the environment, such as symmetries and bottlenecks. They are task independent, in the sense that they do not use information related to reward functions. Moreover, they are defined over the whole state space since each eigenvector induces a real-valued mapping over each state. We can imagine that options with these properties should also be useful. In this section I show how to use PVFs to discover options.

Let me start with an example. Consider the traditional 4-room domain depicted in Figure 4.1c. Gray squares represent walls and white squares represent accessible states. Four actions are available: *up*, *down*, *right*, and *left*. The transitions are deterministic and the agent is not allowed to move into a wall. Ideally, we would like to discover options that move the agent from room to room. Thus, we should be able to automatically distinguish between the

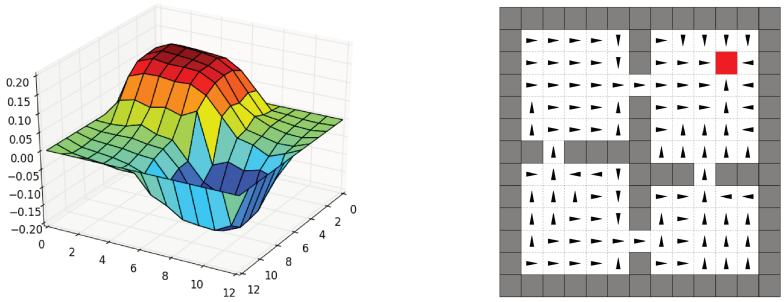


Figure 4.2: Second PVF (left) and its corresponding option (right) in the 4-room domain. Action *terminate* is depicted in red (top right corner), other actions are depicted as arrows.

different rooms in the environment. This is exactly what PVFs do, as depicted in Figure 4.2 (left). Instead of interpreting a PVF as a basis function, we can interpret the PVF in this example as a desire to reach the highest point of the plot, corresponding to the corner of the room. Importantly, the sign of an eigenvector is arbitrary. Thus, a PVF can also be interpreted as a desire to reach the lowest point of the plot, corresponding to the opposite room. In this dissertation I use the eigenvectors in both directions (i.e., both signs).

An *eigenpurpose* formalizes the interpretation above by defining an intrinsic reward function. It can be seen as defining a *purpose* for the agent, that is, to maximize the discounted sum of these rewards.

Definition 4.1.1 (Eigenpurpose)**.** An *eigenpurpose* is the reward function $r_i^{\mathbf{e}}(s, s')$ obtained from a proto-value function $\mathbf{e} \in \mathbb{R}^{|\mathcal{S}|}$ such that

$$r_i^{\mathbf{e}}(s, s') \quad = \quad \mathbf{e}^\top(\boldsymbol{\phi}(s') - \boldsymbol{\phi}(s)), \tag{4.1}$$

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

Notice that, in the tabular case, an eigenpurpose can be written as

$$r_i^{\mathbf{e}}(s, s') = \mathbf{e}[s'] - \mathbf{e}[s].$$

I can now define a new MDP to learn the option associated with the purpose, $\mathcal{M}_i^{\mathbf{e}} = \langle \mathcal{S}, \mathcal{A} \cup \{\perp\}, r_i^{\mathbf{e}}, p, \gamma \rangle$, where the reward function is defined as in

Equation 4.1 and the action set is augmented by the action *terminate* ($\perp$), which allows the agent to leave $\mathcal{M}_i^{\mathbf{e}}$ at no cost, which is equivalent to being able to terminate an episode in an episodic task. The state space and the transition probability kernel remain unchanged from the original problem. The discount rate can be chosen arbitrarily, although it impacts the timescale the option encodes by defining how myopic the agent will be with respect to the eigenpurpose.

With $\mathcal{M}_i^{\mathbf{e}}$, I define a new state-value function $v_{\pi^{\mathbf{e}}(s)}$, for policy $\pi^{\mathbf{e}}$, as the expected value of the cumulative discounted intrinsic reward if the agent starts in state s and follows policy $\pi^{\mathbf{e}}$ until termination. Similarly, I define a new action-value function $q_{\pi^{\mathbf{e}}}(s,a)$ as the expected value of the cumulative discounted intrinsic reward if the agent starts in state s, takes action a, and then follows policy $\pi^{\mathbf{e}}$ until termination. The optimal value function for any eigenpurpose obtained through $\mathbf{e}$ can also be described as

$$v_*^{\mathbf{e}}(s) = \max_{\pi} v_{\pi^{\mathbf{e}}}(s) \qquad \text{and} \qquad q_*^{\mathbf{e}}(s,a) = \max_{\pi} q_{\pi^{\mathbf{e}}}(s,a).$$

These definitions naturally lead us to *eigenbehaviors*.

Definition 4.1.2 (Eigenbehavior)**.** An *eigenbehavior* is a policy, $\chi^{\mathbf{e}}$, that is optimal with respect to the eigenpurpose $r_i^{\mathbf{e}}$, that is,

$$\chi^{\mathbf{e}}(s) = \arg\max_{a \in \mathcal{A}} q_*^{\mathbf{e}}(s,a).$$

Finding the optimal policy $\pi_*^{\mathbf{e}}$ now becomes a traditional RL problem, with a different reward function. Importantly, this reward function tends to be dense, avoiding challenging situations due to exploration issues. In this chapter I use policy iteration to solve for an optimal policy.

In order for an eigenpurpose to define an option I need to define the option's policy, initiation, and termination set. The option's policy is defined by the eigenpurpose's corresponding eigenbehavior. I now proceed to define the option's initiation and termination set to ensure that it is available in every state where it is possible to achieve its purpose, and to terminate when this purpose is achieved.

When defining the MDP to learn the option I augment the agent's action set with the *terminate* action, allowing the agent to interrupt the option anytime. I want options to terminate when the agent achieves its purpose, that is, when it is unable to accumulate further positive intrinsic rewards. With the defined reward function, this happens when the agent reaches the state with largest value in the eigenpurpose (or a local maximum when $\gamma < 1$). Any subsequent sum of rewards will be negative. I formalize this condition by defining

$$q_{\mu^{\mathbf{e}}}(s,a) = \begin{cases} q_{\chi^{\mathbf{e}}}(s,a), & \text{if } a \in \mathcal{A}, \\ 0, & \text{if } a = \perp, \end{cases}$$

where $\mu^{\mathbf{e}}$ denotes the policy of the option defined by the PVF $\mathbf{e}$. When the terminate action is selected, control is returned to the higher level policy (Dietterich 2000). An option following a policy $\mu^{\mathbf{e}}$ terminates when $q_{\mu^{\mathbf{e}}}(s,a) \leq 0$ for all $a \in \mathcal{A}$. I define the initiation set to be all states in which there exists an action $a \in \mathcal{A}$ such that $q_{\mu^{\mathbf{e}}}(s,a) > 0$. Thus, the option's policy is

$$\mu^{\mathbf{e}}(s) = \arg\max_{a \in \mathcal{A} \cup \{\perp\}} q_{\mu^{\mathbf{e}}}(s,a).$$

As aforementioned, I refer to the options discovered with this approach as *eigenoptions*. The eigenoption corresponding to the example at the beginning of this section is depicted in Figure 4.2 (right). Below I summarize the terms used in this section.

Table 4.1: Terms used when discussing the introduced options.

Term	Description
Eigenpurpose	Intrinsic reward function, $r_i^{\mathbf{e}}$, the agent maximizes when discovering the corresponding option. This reward function encourages the agent to navigate alongside a PVF $\mathbf{e}$. Formally, $r_i^{\mathbf{e}}(s,s') = \mathbf{e}^\top(\boldsymbol{\phi}(s') - \boldsymbol{\phi}(s))$.
Eigenbehavior	Optimal policy that maximizes the cumulative discounted sum of a given eigenpurpose.
Eigenoption	Option discovered when maximizing an eigenpurpose. Its initiation set, policy, and termination condition are derived from the corresponding eigenbehavior, as discussed in the main text.

For any eigenoption, there is always at least one state in which it terminates, as I show below (the supporting lemmas are available in the Appendix).

Theorem 4.1.1 (Option's Termination)**.** *Consider an eigenoption* $o = \langle \mathcal{I}_o, \pi_o, \mathcal{T}_o \rangle$ *and* $\gamma \in [0, 1)$*. Then, in an ergodic MDP with finite state space,* $\mathcal{T}_o$ *is nonempty.*

Proof. We can write the Bellman equation in the matrix form: $\mathbf{v} = \mathbf{r} + \gamma P_\pi \mathbf{v}$, for a fixed policy π, where $\mathbf{v}$ is a *finite* column vector with one entry per state encoding its value function. From Equation 4.1 we have $\mathbf{r} = P_\pi \mathbf{w} - \mathbf{w}$ with $\mathbf{w} = \Phi \mathbf{e}$, where $\mathbf{e}$ denotes the eigenpurpose of interest and $\Phi \in \mathbb{R}^{|\mathcal{S}| \times d}$ denotes the matrix representing the d-dimensional feature representation for each state. I use $r : \mathcal{S} \to \mathbb{R}$ for simplicity. This function can be seen as the expected reward in a given state. With that we have:

$$
\begin{aligned}
\mathbf{v} &= P_\pi \mathbf{w} - \mathbf{w} + \gamma P_\pi \mathbf{v} \\
\mathbf{v} + \mathbf{w} &= P_\pi \mathbf{w} + \gamma P_\pi \mathbf{v} \\
&= P_\pi \mathbf{w} + \gamma P_\pi \mathbf{v} + \gamma P_\pi \mathbf{w} - \gamma P_\pi \mathbf{w} \\
&= (1 - \gamma) P_\pi \mathbf{w} + \gamma P_\pi (\mathbf{v} + \mathbf{w}) \\
\mathbf{v} + \mathbf{w} - \gamma P_\pi (\mathbf{v} + \mathbf{w}) &= (1 - \gamma) P_\pi \mathbf{w} \\
(I - \gamma P_\pi)(\mathbf{v} + \mathbf{w}) &= (1 - \gamma) P_\pi \mathbf{w} \\
\mathbf{v} + \mathbf{w} &= (1 - \gamma)(I - \gamma P_\pi)^{-1} P_\pi \mathbf{w}
\end{aligned}
$$

where the last step is true because $(I - \gamma P_\pi)^{-1}$ is guaranteed to be nonsingular since $||P_\pi|| \leq 1$, where $||P_\pi|| = \sup_{\mathbf{v}:||\mathbf{v}||_\infty = 1} ||P_\pi \mathbf{v}||_\infty$. By the Neumann series we have $(I - \gamma P_\pi)^{-1} = \sum_{n=0}^{\infty} \gamma^n P_\pi^n$. Using the induced norm we have:

$$
\begin{aligned}
&||\mathbf{v} + \mathbf{w}||_\infty = (1 - \gamma) ||(I - \gamma P_\pi)^{-1} P_\pi \mathbf{w}||_\infty \\
&||\mathbf{v} + \mathbf{w}||_\infty \leq (1 - \gamma) ||(I - \gamma P_\pi)^{-1} P_\pi||_\infty ||\mathbf{w}||_\infty && \text{because } ||A\mathbf{x}|| \leq ||A|| \cdot ||\mathbf{x}|| \\
&||\mathbf{v} + \mathbf{w}||_\infty \leq (1 - \gamma) \frac{1}{(1 - \gamma)} ||\mathbf{w}||_\infty && \text{Lemma } A.2 \\
&||\mathbf{v} + \mathbf{w}||_\infty \leq ||\mathbf{w}||_\infty
\end{aligned}
$$

We can shift $\mathbf{w}$ by any finite constant without changing the reward, that is, $P_\pi \mathbf{w} - \mathbf{w} = P_\pi(\mathbf{w} + \boldsymbol{\delta}) - (\mathbf{w} + \boldsymbol{\delta})$ because $P_\pi \mathbf{1}\boldsymbol{\delta} = \mathbf{1}\boldsymbol{\delta}$ since $\sum_j P_{\pi_{i,j}} = 1$.

Therefore, we can assume $\mathbf{w} \geq \mathbf{0}$. Let $s^* = \arg\max_s \mathbf{w}_{s^*}$, so that $\mathbf{w}_{s^*} = ||\mathbf{w}||_\infty$. Clearly $\mathbf{v}_{s^*} \leq \mathbf{0}$, otherwise $||\mathbf{v} + \mathbf{w}||_\infty \geq |\mathbf{v}_{s^*} + \mathbf{w}_{s^*}| = \mathbf{v}_{s^*} + \mathbf{w}_{s^*} > \mathbf{w}_{s^*} = ||\mathbf{w}||_\infty$, arriving at a contradiction. □

This result is applicable in both the tabular and linear function approximation case. An algorithm that does not rely on knowing the underlying graph is provided in Section 4.3.

4.2 Empirical Evaluation in the Tabular Case

I used three MDPs in the empirical study (see Figure 4.1): an open room, an I-Maze, and the 4-room domain. Their transitions are deterministic and gray squares denote walls. Agents have access to four actions: *up*, *down*, *right*, and *left*. When an action that would have taken the agent into a wall is chosen, the agent's state does not change. I demonstrate three aspects of the proposed framework:[1]

- How the eigenoptions present specific purposes.
- How eigenoptions improve exploration by reducing the expected number of steps required to navigate between any two states.
- How eigenoptions help agents to accumulate reward faster. I show how few options may hurt the agents' performance while enough options speed up learning.

4.2.1 Discovered Options

In the PVF theory, the "smoothest" eigenvectors, corresponding to the smallest eigenvalues, are preferred (Mahadevan and Maggioni 2007). The same intuition applies to eigenoptions, with the eigenpurposes corresponding to the smallest eigenvalues being preferred. Figures 4.3, 4.4, and 4.5 depict the first eigenoptions discovered in the three domains used for evaluation.

Eigenoptions do not necessarily look for bottleneck states, allowing this algorithm to be applied in many environments in which there are no obvious,

[1] *Python* code can be found at: `https://github.com/mcmachado/options`

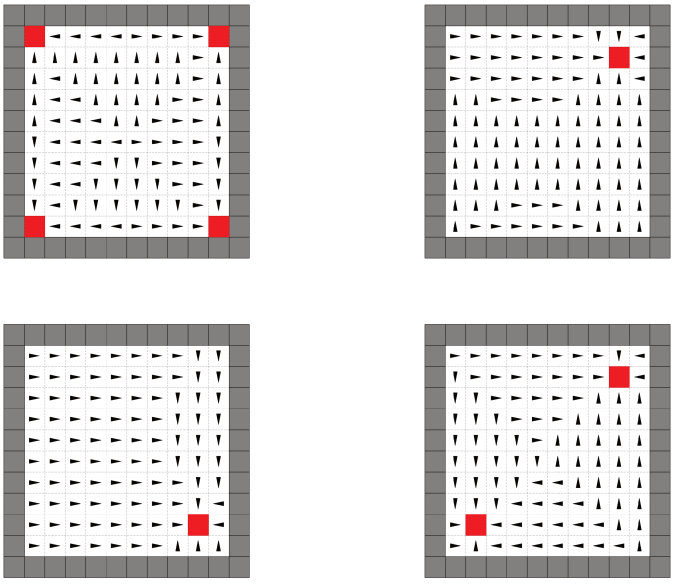


Figure 4.3: Options obtained from the four smallest eigenvectors in the 10×10 grid. Action *terminate* is depicted in red.

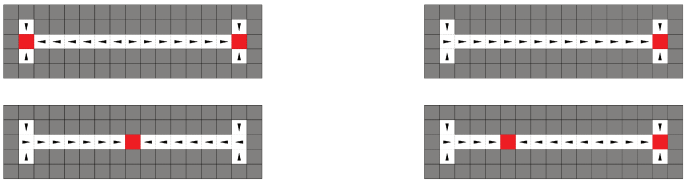


Figure 4.4: Options obtained from the four smallest eigenvectors in the I-Maze domain. Action *terminate* is depicted in red.

or meaningful, bottlenecks. It discovers meaningful options in these environments, such as walking down a corridor, or going to the corners of an open room. Interestingly, doorways are not the first options it discovers in the 4-room domain (the fifth eigenoption is the first to terminate at the entrance of a doorway). In the next sections I provide empirical evidence that eigenoptions are useful for exploration, and often more so than bottleneck options.

4.2.2 Exploration

A major challenge for agents to explore an environment is to be decisive, avoiding the dithering commonly observed in random walks (Osband, Roy, and Z. Wen 2016). Options provide such decisiveness by operating in a higher

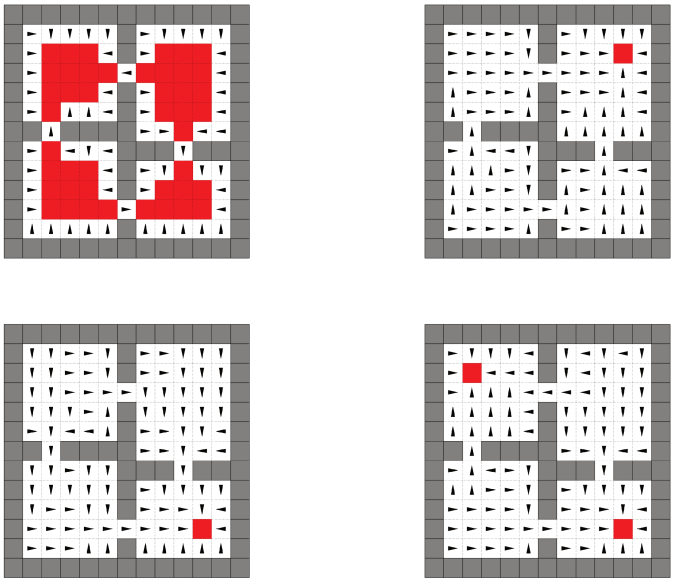


Figure 4.5: Options obtained from the four smallest eigenvectors in the 4-room domain. Action *terminate* is depicted in red.

level of abstraction. Agents performing a random walk, when equipped with options, are expected to cover larger distances in the state space, navigating back and forth between subgoals instead of dithering around the starting state. However, options need to satisfy two conditions to improve exploration: 1) they have to be available in several parts of the state space, ensuring the agent always has access to many different options; and 2) they have to operate at different time scales. For instance, in the 4-room domain, it is unlikely an agent randomly selects enough primitive actions leading it to a corner if all options move the agent between doorways. An interesting result in this section is that it is unlikely for an agent to explore the whole environment if it keeps going back and forth between similar high-level goals.

Eigenoptions satisfy both conditions. As demonstrated in Section 4.2.1, eigenoptions are often defined in the whole state space, allowing sequencing. Moreover, PVFs can be seen as a "frequency" basis, with different PVFs being associated with different frequencies (Mahadevan and Maggioni 2007). The corresponding eigenoptions also operate at different frequencies, with the length of a trajectory until termination varying. This behavior can be seen

when comparing the second and fourth eigenoptions in the 10×10 grid (Figure 4.3). The fourth eigenoption terminates, on expectation over all possible start states, twice as often as the second eigenoption.

In this section I show that eigenoptions improve exploration. I use the *diffusion time* metric to quantify the effectiveness of the exploration strategy. Diffusion time encodes the expected number of steps required to navigate between two states randomly chosen in the MDP while following a random walk. A small expected number of steps implies that it is more likely that the agent will reach all states with a random walk. Dayan and Hinton (1992) have proposed a similar metric in the past.

In tabular domains, we can easily compute the diffusion time with dynamic programming. To do so one should define a new MDP such that the value function of a state s, under a uniform random policy, encodes the expected number of steps required to navigate between state s and a chosen goal state. One can then compute the expected number of steps between any two states by averaging, for each possible goal, the value of all other states.

The MDP in which the value function of state s encodes the expected number of time steps from s to a goal state has $\gamma = 1$ and a reward function where the agent observes $+1$ at every time step in which it is not in the goal state. Policy evaluation in this case encodes the expected number of time steps the agent will take before arriving to the goal state. To compute the diffusion time we iterate over all possible states, defining them as terminal states, and averaging the value function of the other states in that MDP.

Figure 4.6 depicts, for the three environments, the diffusion time with options and the diffusion time using only primitive actions. I add options incrementally in order of increasing eigenvalue when computing the diffusion time for different sets of options.

The first options added hurt exploration, but when enough options are added, exploration is greatly improved when compared to a random walk using only primitive actions. The fact that few options hurt exploration may be surprising at first, based on the fact that few useful options are generally sought after in the literature. However, this is a major difference between

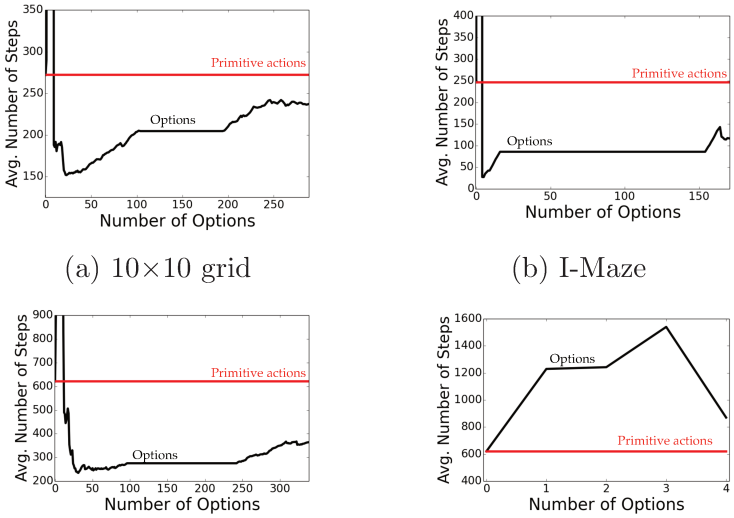


Figure 4.6: Expected number of steps between any two states when following a random walk. Figure 4.6d shows the performance of options that look for doorways in the 4-room domain.

using options for planning and for learning. In planning, options shortcut the agents' trajectories, pruning the search space. All other actions are still taken into consideration. When exploring, a uniformly random policy over options and primitive actions skews where agents spend their time. Options that are much longer than primitive actions reduce the likelihood that an agent will deviate much from the options' trajectories, since sampling an option may undo dozens of primitive actions. This biasing is often observed when fewer options are available.

The discussion above can be made clearer with an example. In the 4-room domain, if the only options available are those leading the agent to doorways (see Figure 4.7), it is less likely the agent will reach the outer corners. To do so the agent would have to select enough consecutive primitive actions without sampling an option. Also, it is very likely agents will always be moving between rooms, never really exploring inside a room. These issues are mitigated with eigenoptions. The first eigenoptions lead agents to individual rooms, but other eigenoptions operate in different time scales, allowing agents to explore different parts of rooms.

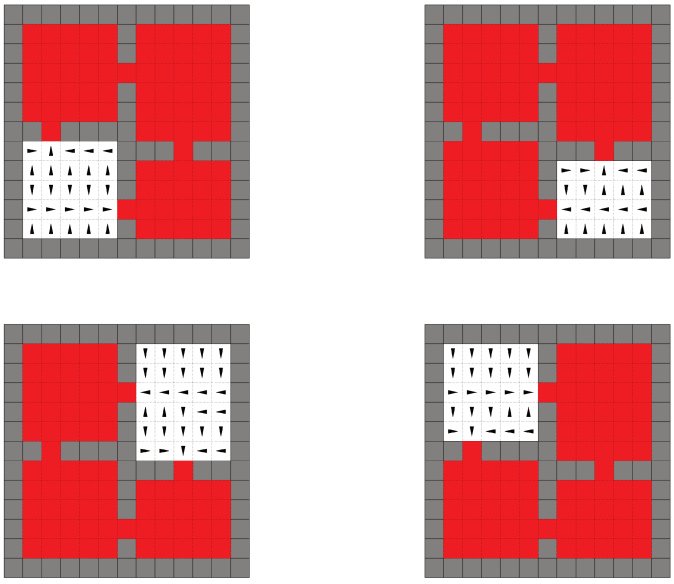
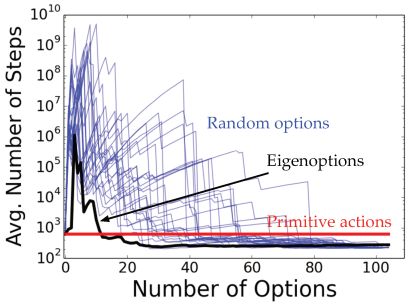


Figure 4.7: Options leading to bottleneck states. Each option is defined in a single room, moving the agent to the closest doorway.

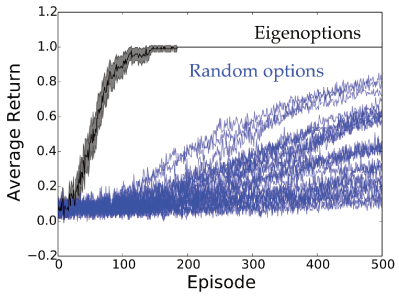
Figure 4.6d supports the intuition that options leading to bottleneck states are not sufficient, by themselves, for exploration. It shows how the diffusion time in the 4-room domain is increased when only bottleneck options are used (Jong, Hester, and Stone 2008 have also shown how bottleneck options might hurt learning, but not necessarily in the context of exploration). As in the PVF literature, the ideal number of options to be used by an agent can be seen as a model selection problem.

It is actually really important to use the information about diffusion in the environment to define the option's purposes. This information impacts the sequence of subgoal locations the options' seek after, as well as the time scales they operate at. The ordering in which the eigenoptions are discovered and the different time scales they operate at can have a major impact on the agents' performance.

We can observe the importance of using the environment's diffusion information when we compare the proposed approach to *random options*, a simple baseline that does not use such information. This baseline defines an option to be the policy, defined in the whole state space, that terminates in a randomly



(a) Diffusion time. The y-axis is in logarithmic scale.



(b) Learning curve using 64 options.

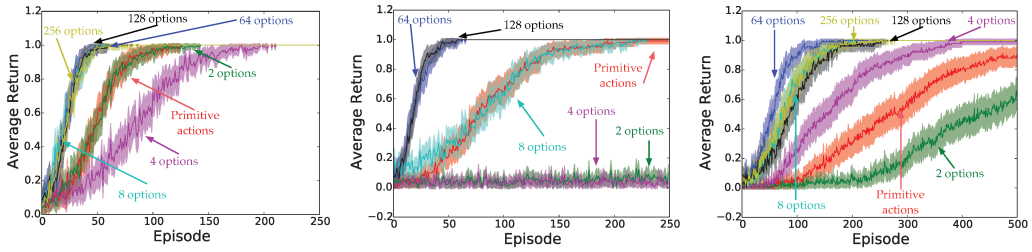
Figure 4.8: Diffusion time and learning performance of eigenoptions and of random options in the 4-room domain.

selected state of the environment. I performed these experiments in the tabular case because it is not clear how I should extend this baseline to settings in which states cannot be enumerated.

Figure 4.8a depicts the diffusion time of random options and eigenoptions in the 4-room domain. For the random options results, I added them incrementally at random to the agent's action set until having added all possible options. I repeated this process 24 times to verify the impact of adding random options in a different order. Each blue line represents the performance of one of the evaluated sequences. The results clearly show that eigenoptions do more than going to a randomly selected state. Most of the obtained sequences of random options fail to reduce the agent's diffusion time. They increase it by several orders of magnitude (notice the y-axis is in logarithmic scale) until having enough options available to the point that the graph is almost fully connected, that is, when the agent basically has an option leading it to each possible state in the MDP.

4.2.3 Accumulating Rewards

I now illustrate the usefulness of eigenoptions when the agent's goal is to accumulate reward. I also study the impact of an increasing number of options in such a task. In these experiments, the agent starts at the bottom left corner and its goal is to reach the top right corner. The agent observes a



(a) 10×10 grid (b) I-Maze (c) 4-room domain

Figure 4.9: The agents' performance accumulating reward as options are added to the action set in their behavior policy. These results use the eigenpurposes directly obtained from the eigendecomposition as well as their negation.

reward of 0 until the goal is reached, when it observes a reward of +1. I used Q-Learning ($\alpha = 0.1$, $\gamma = 0.9$; Watkins and Dayan 1992) to learn a policy over primitive actions. The behavior policy chooses uniformly over primitive actions and options, following them until termination. Figure 4.9 depicts, after learning for a given number of episodes, the average over 100 trials of the agents' final performance. Episodes were 100 time steps long, and the agent learned for 250 episodes in the 10×10 grid and in the I-Maze, and for 500 episodes in the 4-room domain.

In most scenarios eigenoptions improve performance. As in the previous section, exceptions occur when only a few options are added to the agent's action set. The best results were obtained using 64 options. Despite requiring an additional parameter, these results suggest that the agent's performance is fairly robust across different numbers of options when using the proposed approach.

Eigenoptions are task-independent by construction. The same set of eigenoptions is actually able to speed-up learning in different tasks. Figure 4.10 depicts, after learning for a pre-determined number of episodes, the average over 100 trials of the agents' final performance, as well as the starting (S) and goal (G) states. Based on the previous results, I fixed the number of used eigenoptions to 64 (32 eigenpurposes and their negations). In this set of experiments I also compare this approach to traditional bottleneck options (Figure 4.7).

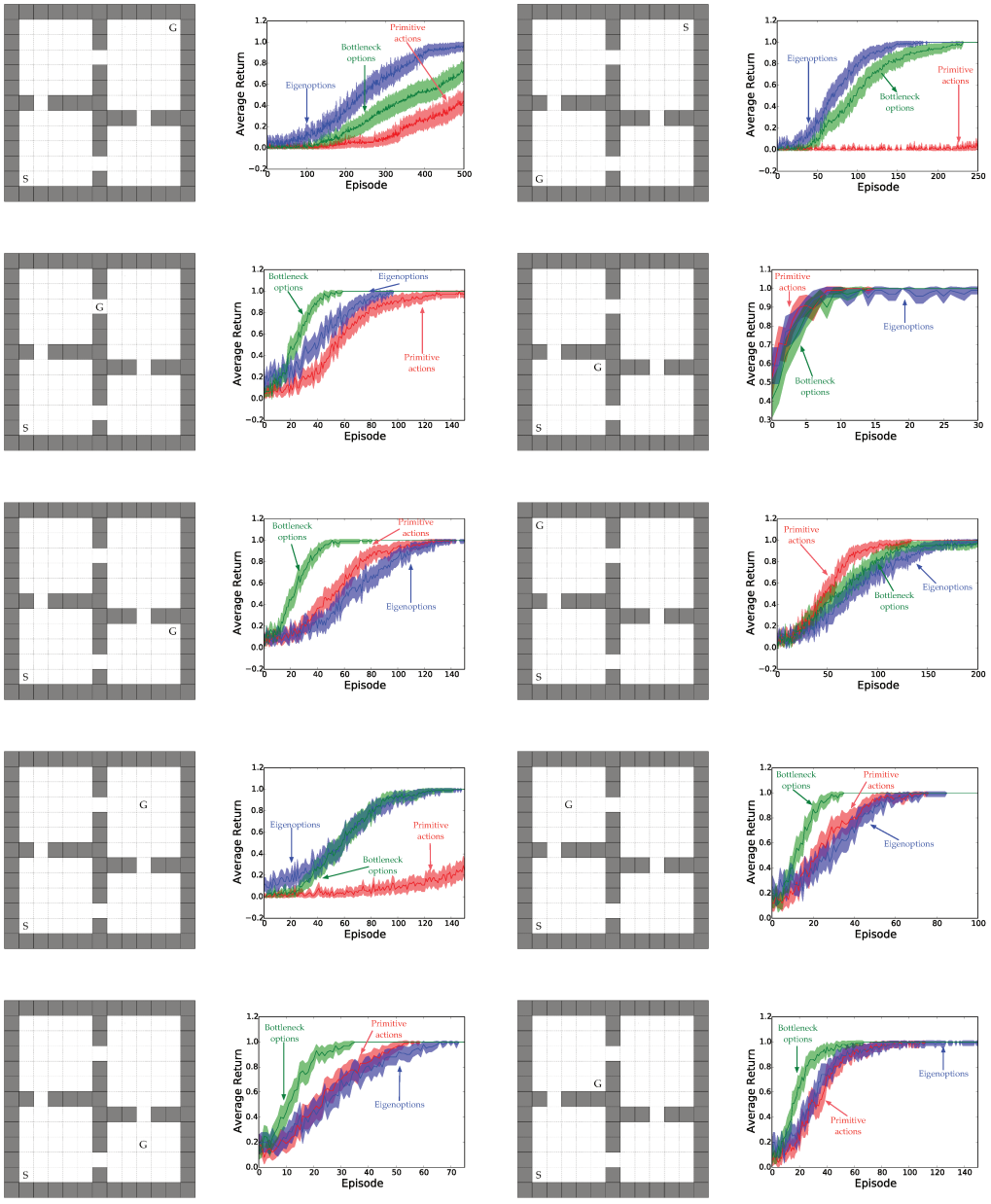


Figure 4.10: Agents' performance in different tasks when using eigenoptions, bottleneck options, and primitive actions.

The obtained results show that switching the positions of the starting and goal states have little effect in the performance of my algorithm. Also, in almost all settings, the agents augmented by eigenoptions outperfom those equipped only with primitive actions. The comparison between eigenoptions and options that look for bottleneck states is more subtle. As expected, agents

equipped with eigenoptions outperform agents equipped with options leading to bottleneck states in settings in which the goal state is far from the doorways, as aforementioned. In scenarios where the goal state is closer to bottleneck states, the options leading to doorways are more competitive. Importantly, this analysis is based on the results when using 64 eigenoptions, which may not encode all options required to go to a specific region of the state space.

Finally, I also compared the performance of eigenoptions and of random options, described in the previous section, when accumulating reward. Figure 4.8b depicts the learning curve of agents equipped with eigenoptions and of agents equipped with random options. As before, the blue lines indicate the agent's performance in individual runs. We can see that no individual run is competitive to eigenoptions. When fewer options are used (not shown), the variance across individual runs is even larger, depending on whether one of the random options terminates near the goal state. In some runs the agent never even learns to reach the goal. Therefore, as in the diffusion time, on average, random options are not competitive to eigenoptions, demonstrating the importance of the diffusion model I use.

4.3 Approximate Option Discovery

So far I have assumed that agents have access to the adjacency matrix representing the underlying MDP. However, in practical settings this is generally not true. In fact, the number of states in these settings is often so large that agents rarely visit the same state twice. These problems are generally tackled with sample-based methods and some sort of function approximation.

In this section I propose a sample-based approach for option discovery that asymptotically discovers eigenoptions. I then extend this algorithm to linear function approximation. I provide anecdotal evidence in Atari 2600 games that this relatively naïve sample-based approach to function approximation discovers purposeful options.

4.3.1 Sample-based Option Discovery

In the online setting, agents must sample trajectories. Naturally, one can sample trajectories until being able to perfectly construct the MDP's adjacency matrix, as suggested by Mahadevan and Maggioni (2007). However, this approach does not easily extend to linear function approximation. In this section I propose an approach that does not build the adjacency matrix, allowing me to extend the concept of eigenpurposes to linear function approximation.

In my algorithm, a sample transition is added to a matrix T if it was not previously encountered. The transition is added as the difference between the current and previous observations, that is, $\boldsymbol{\phi}(s') - \boldsymbol{\phi}(s)$. In the tabular case I define $\boldsymbol{\phi}(s)$ to be the one-hot encoding of state s. Once enough transitions have been sampled, I perform a singular value decomposition on the matrix of stacked transitions T such that $T = U\Sigma V^\top$. I use the columns of V, which correspond to the right-singular vectors of T, to generate the eigenpurposes. The intrinsic reward and the termination criterion for an eigenbehavior are the same as before.

Matrix T is known as the incidence matrix. If all transitions in the graph are sampled once, for tabular representations, this algorithm discovers the same options obtained with the combinatorial Laplacian. The theorem below states the equivalence between the obtained eigenpurposes (the supporting lemma is available in the Appendix).

Theorem 4.3.1. *Consider the SVD of $T = U_T \Sigma_T V_T^\top$, with each row of T consisting of the difference between observations,* i.e., *$\boldsymbol{\phi}(s') - \boldsymbol{\phi}(s)$. In the tabular case, if all transitions in the MDP have been sampled once, the orthonormal eigenvectors of L are the columns of $V_T^\top$.*

Proof. Given the SVD decomposition of a matrix $A = U\Sigma V^\top$, the columns of V are the eigenvectors of $A^\top A$ (Strang 2005). We know that $T^\top T = 2L$, where $L = D - W$ (Lemma A.3 in the Appendix). Thus, the columns of V_T are the eigenvectors of $T^\top T$, which can be rewritten as $2(D - W)$. Therefore, the columns of V_T are also the eigenvectors of L. □

There is a trade-off between reconstructing the adjacency matrix and constructing the incidence matrix. In MDPs in which states are sparsely connected, such as the I-Maze, the latter is preferred since it has fewer transitions than states. However, what makes this result interesting is the fact that this algorithm can be easily generalized to linear function approximation.

4.3.2 Function Approximation

An adjacency matrix is not available when the agent has access only to features of the state. However, I can use the intuition about the incidence matrix to propose an algorithm compatible with linear function approximation.

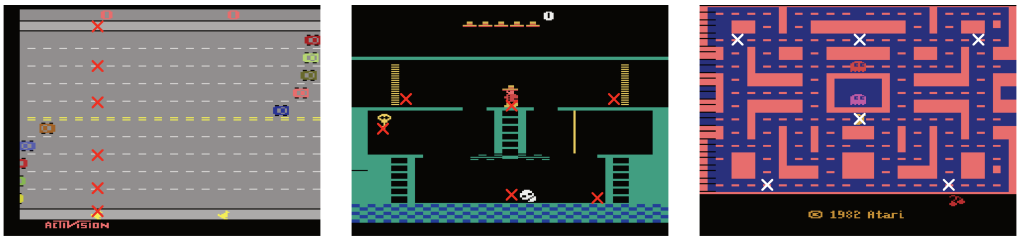
In fact, to apply the algorithm proposed in the previous section, I just need to define what constitutes a new transition. I define two vectors, $\mathbf{t}$ and $\mathbf{t}'$, to be identical if and only if $\mathbf{t} - \mathbf{t}' = \mathbf{0}$. I then use a *set* data structure to avoid duplicates when storing $\boldsymbol{\phi}(s') - \boldsymbol{\phi}(s)$. This is a naïve approach, but it provides encouraging evidence eigenoptions generalize to linear function approximation. More involved methods do perform even better, as I discuss in Chapter 7.

I tested my method in the ALE (Bellemare, Naddaf, et al. 2013). The agent's representation consists of the emulator's RAM state (1,024 bits).

I defined six different starting states in each Atari 2600 game, letting the agent take random actions from that point until termination. The agent follows a pre-determined sequence of actions leading it to each starting state. I store the observed transitions leading the agent to the start states as well as those obtained from the random actions. I provide results for Freeway, Montezuma's Revenge and Ms Pac-Man. The starting states for all three games are depicted in Figure 4.11.

The agent plays rounds of six episodes, with each episode starting from a different start state, until it observes at least 25,000 new transitions. The final incidence matrix in which I ran the SVD had 25,000 rows, which I sampled uniformly from the set of observed transitions. The agent used the deterministic version of the Arcade Learning Environment (ALE), the games' minimal action set, and a frame skip of 1.

In the tabular case I select eigenpurposes generated by the eigenvectors



(a) Freeway (b) Mont. Revenge (c) Ms. Pac-Man

Figure 4.11: Pre-defined start states in Atari 2600 games.

with smallest eigenvalue, because these are the "smoothest" ones. However, it is not clear such intuition holds here because we are in the function approximation setting and the matrix of transitions does not contain all possible transitions. Therefore, I analyzed, for each game, all 1,024 discovered options.

I approximate these options greedily ($\gamma = 0$) with the ALE emulator's look-ahead. The next action a' for an eigenpurpose $\mathbf{e}$ is selected as the $\arg\max_{b \in \mathcal{A}} \int_{s'} p(s'|s,b)\ r_i^{\mathbf{e}}(s,s')ds'$.

Even with this myopic action selection mechanism I was able to obtain options that clearly demonstrate intent. In Freeway, a game in which a chicken is expected to cross the road while avoiding cars, I observe options in which the agent clearly wants to reach a specific lane in the street. Figure 4.12 (left) depicts where the chicken tends to be when the option is executed. On the right we see a histogram representing the chicken's height during an episode. We can clearly see how the chicken's height varies for different options, and how a random walk over primitive actions (*rand*) does not explore the environment properly. Remarkably, option *#445* scores 28 points at the end of the episode, without ever explicitly taking the reward signal into consideration.

In Montezuma's Revenge, a game in which the agent needs to navigate through a room to pickup a key so it can open a door, I also observe the agent having the clear intent of reaching particular positions on the screen, such as staircases, ropes and doors (Figure 4.13, left). Interestingly, the options I discover are very similar to those handcrafted by Kulkarni, Narasimhan, et al. (2016) when evaluating the usefulness of options to tackle such a game.

Ms. Pac-Man is a game in which the agent needs to navigate through a

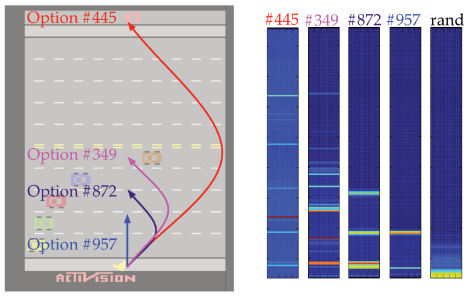


Figure 4.12: Eigenoptions discovered in FREEWAY.

maze eating pellets while avoiding ghosts. As in the other games, the agent has the clear intent of reaching particular positions in the screen, such as corners and intersections. Figure 4.13 (right) depicts the positions in which agents tend to spend most of their time on. A video of the highlighted options for all games can be found online.[2]

4.4 Discussion

In this chapter I introduced an algorithm for option discovery that is general and effective in different environments. I then used this algorithm to provide empirical evidence supporting the hypothesis that using options to navigate through the state space is a promising approach for exploration in reinforcement learning. Interestingly, the options first discovered by my approach do not necessarily find bottlenecks, which are commonly sought after. In this chapter I showed how bottleneck options can hinder exploration strategies if naïvely added to the agent's action set, and how eigenoptions can help an agent explore. Also, I have shown how we can leverage the exploratory properties of eigenoptions to maximize return.

The algorithm presented here can also be seen as a first attempt to instantiate the option discovery cycle proposed in Chapter 3. The learned representation informs the agent what are meaningful options to be sought after, with the discovered options traversing the different dimensions of the learned representation. Shortly, the agent follows the random policy while learning

[2] `https://youtu.be/2BVicx4CDWA`

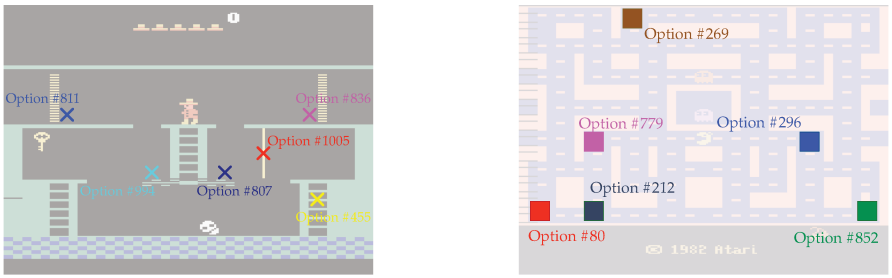


Figure 4.13: Eigenoptions discovered in Montezuma's Revenge (left) and Ms. Pac-Man (right).

proto-value functions (PVFs) as the linear representation. It selects the top k features according to the obtained eigenvalues and proceeds to learn to attain those features via eigenpurposes. Each one of the options then consists of an eigenbehavior and the initiation and termination sets defined here. I presented results for a single iteration of the cycle in this chapter. I present some initial results of multiple iterations of this cycle in Chapter 7.

Despite the promising results shown in this chapter, it is important to acknowledge the limitations of PVFs. In the tabular case, PVFs are often defined for deterministic environments, with a binary adjacency matrix being used. It is not straightforward to extend proto-value functions to stochastic environments. Moreover, there is an implicit assumption that the weight matrix is symmetric, which is often not the case. It is also not easy to incrementally update the PVFs and the extension I proposed for the linear function approximation depends on handcrafted features being provided beforehand. In the next chapter I present a new algorithm that addresses these limitations.

Chapter 5

Option-Based Exploration with the Successor Representation

In the previous chapter I introduced an algorithm for option discovery that, in the tabular case, can be used in symmetric and deterministic environments. In environments where it is infeasible to enumerate states, this algorithm is applicable when a linear representation is available beforehand. In this chapter I introduce an algorithm for eigenoption discovery that is applicable to settings with stochastic and asymmetric transitions. Motivated by the fact that methods that learn non-linear representations are more flexible, more scalable, and often lead to better performance, in this chapter I also introduce an algorithm that is capable of discovering eigenoptions while learning non-linear state representations from raw pixels. It exploits recent successes in the deep reinforcement learning literature and the fact that PVFs are equivalent to the eigenvectors of the successor representation (SR) when it is defined with respect to the uniform random policy in a symmetric and deterministic environment, a result I present here.

I evaluate this algorithm in a tabular domain as well as on Atari 2600 games. I use the tabular domain to provide intuition about the algorithm and to compare it to other algorithms in the literature. The evaluation in Atari 2600 games demonstrates that results similar to those obtained in the previous chapter can also be obtained from raw pixels. This result provides promising evidence of the applicability of this algorithm in a setting in which

a representation of the agent's observation is not available.

5.1 Proto-Value Functions and the Eigenvectors of the Successor Representation

The main result in this chapter is the realization that PVFs (the eigenvectors of the normalized Laplacian) are equal to the eigenvectors of the successor representation, for a random policy in a regular graph, scaled by $\gamma^{-1}D^{1/2}$. To the best of my knowledge, this equivalence was first mentioned by Stachenfeld, Botvinick, and Gershman (2014) but no proof was provided. Below I provide a formal statement of such an equivalence for the eingevalues and the eigenvectors of both approaches in the special case in which the SR is defined with respect to a random policy in a symmetric and deterministic environment. I then use the proof to further discuss the extent of this interchangeability.

The theorem below uses the definitions already provided in this dissertation. I assume $0 < \gamma < 1$, with $\Psi_\pi = (I - \gamma P_\pi)^{-1}$ denoting the matrix encoding the SR, and $L = D^{-1/2}(D - W)D^{-1/2}$ denoting the matrix corresponding to the normalized Laplacian, both obtained under a uniform random policy.

Theorem 5.1.1. *The i-th eigenvalue ($\lambda_{SR,i}$) of the SR, defined with respect to a uniform random policy, and the j-th eigenvalue ($\lambda_{PVF,j}$) of the normalized Laplacian are related as follows when in a symmetric and deterministic environment :*

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

The i-th eigenvector ($\mathbf{e}_{SR,i}$) of the SR and the j-th eigenvector ($\mathbf{e}_{PVF,j}$) of the normalized Laplacian, where $i + j = n + 1$, with n being the total number of rows (and columns) of matrix P_π, are related as follows:

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

Proof. Let λ_i, $\mathbf{e}_i$ denote the i-th eigenvalue and eigenvector of the SR, respectively. Using the fact that the SR converges to $(I - \gamma P_\pi)^{-1}$ (through the Neumann series), we have:

$$
\begin{aligned}
(I-\gamma P_\pi)^{-1}\mathbf{e}_i &= \lambda_i \mathbf{e}_i \\
\mathbf{e}_i &= \lambda_i (I-\gamma P_\pi)\mathbf{e}_i \\
(I-\gamma P_\pi)\mathbf{e}_i &= \lambda_i^{-1}\mathbf{e}_i \\
(I-\gamma P_\pi)\gamma^{-1}\mathbf{e}_i &= \lambda_i^{-1}\gamma^{-1}\mathbf{e}_i \\
\gamma^{-1}\mathbf{e}_i - P_\pi \mathbf{e}_i &= \lambda_i^{-1}\gamma^{-1}\mathbf{e}_i \\
P_\pi \mathbf{e}_i &= \gamma^{-1}\mathbf{e}_i - \lambda_i^{-1}\gamma^{-1}\mathbf{e}_i \\
&= (1-\lambda_i^{-1})\gamma^{-1}\mathbf{e}_i \\
I\mathbf{e}_i - P_\pi \mathbf{e}_i &= I\mathbf{e}_i - (1-\lambda_i^{-1})\gamma^{-1}\mathbf{e}_i \\
(I-P_\pi)\mathbf{e}_i &= [\gamma - (1-\lambda_i^{-1})]\gamma^{-1}\mathbf{e}_i \\
(I-P_\pi)\gamma^{-1}\mathbf{e}_i &= [1-(1-\lambda_i^{-1})\gamma^{-1}]\gamma^{-1}\mathbf{e}_i \\
(I-P_\pi)\gamma^{-1}\mathbf{e}_i &= \lambda'_j \gamma^{-1}\mathbf{e}_i \qquad (5.1) \\
(I-D^{-1}W)\gamma^{-1}\mathbf{e}_i &= \lambda'_j \gamma^{-1}\mathbf{e}_i \\
(D^{-1}(D-W))\gamma^{-1}\mathbf{e}_i &= \lambda'_j \gamma^{-1}\mathbf{e}_i \\
D^{1/2}(D^{-1}(D-W))\gamma^{-1}\mathbf{e}_i &= \lambda'_j \gamma^{-1} D^{1/2}\mathbf{e}_i \\
D^{-1/2}(D-W)\gamma^{-1}\mathbf{e}_i &= \lambda'_j \gamma^{-1} D^{1/2}\mathbf{e}_i \\
D^{-1/2}(D-W)D^{-1/2}D^{1/2}\gamma^{-1}\mathbf{e}_i &= \lambda'_j \gamma^{-1} D^{1/2}\mathbf{e}_i \\
LD^{1/2}\gamma^{-1}\mathbf{e}_i &= \lambda'_j \gamma^{-1} D^{1/2}\mathbf{e}_i \qquad \square
\end{aligned}
$$

Importantly, when using PVFs I am first interested in the eigenvectors with the corresponding smallest eigenvalues, as they are the "smoothest" ones. When using the SR I am interested in the eigenvectors with the *largest* eigenvalues. The change of variables in Equation 5.1 highlights this fact, that is, $\lambda'_j = [1-(1-\lambda_i^{-1})\gamma^{-1}]$. The indices j are sorted in the reverse order of the indices i. This distinction can be important when trying to estimate the relevant eigenvectors. Finding the largest eigenvalues/vectors is statistically more robust to noise in estimation and does not depend on the eigenvectors with smallest eigenvalues. Moreover, scaling by $D^{1/2}$ does not change the direction of the eigenvectors when the size of the action set is constant across all states. This is the case in all the RL problems being studied in this dissertation.

Alg. 1 General eigenoption discovery

$\hat{\Psi} \leftarrow$ LearnRepresentation()
$E \leftarrow$ ExtractEigenpurposes($\hat{\Psi}$)
for each eigepurpose $\mathbf{e}_i \in E$ **do**
  $\langle \mathcal{I}_{e_i}, \pi_{e_i}, \mathcal{T}_{e_i} \rangle \leftarrow$ LearnEigenoption($\mathbf{e}_i$)
end for

Alg. 2 LearnRepresentation() with the successor representation

for a given number of steps n **do**
  Observe $s \in \mathcal{S}$, take action $a \in \mathcal{A}$ selected according to $\pi(s)$, and observe a next state $s' \in \mathcal{S}$
  for each state $j \in \mathcal{S}$ **do**
    $\hat{\Psi}(s, j) \leftarrow \hat{\Psi}(s, j) + \eta\big(\mathbb{1}_{\{s=j\}} + \gamma\hat{\Psi}(s', j) - \hat{\Psi}(s, j)\big)$
  end for
end for
return $\hat{\Psi}$

5.2 Eigenoption Discovery in the Tabular Case

The structure of the algorithms capable of discovering eigenoptions is fairly straightforward. Algorithm 1 is an overview of the general approach. The agent learns (or is given) a representation that captures the diffusion model of information flow (for example, the combinatorial Laplacian). It then uses the eigenvectors of this representation to define eigenpurposes (ExtractEigenpurposes). The option's policy is the one that maximizes this new reward function, while the initiation and termination sets are defined by the agent's ability to accumulate reward, as I discussed in the previous chapter.

In the tabular case, the instantiation of the idea of using the successor representation to discover options is also fairly simple. Instead of assuming the matrix $\hat{\Psi}$ is given in the form of the graph Laplacian, or trying to estimate the graph Laplacian from samples by stacking the row vectors corresponding to the different observed transitions, we can estimate the diffusion model of information flow through the successor representation (see Algorithm 2). This idea is supported by the theorem in the previous section. The equivalence ensures that the eigenpurposes extraction and the eigenoption learning steps remain unchanged. That is, I still obtain the eigenpurposes from the eigende-

composition[1] of matrix $\hat{\Psi}$, and I still use each eigenvector $\mathbf{e}_i \in E$ to define the new learning problem in which the agent wants to maximize the eigenpurpose. Importantly, the use of the SR also generates an algorithm that has a memory cost that is independent of the number of samples drawn by the agent.

5.3 Eigenoption Discovery in the Non-Linear Function Approximation Case

The tabular case is interesting to study because it provides intuition about the problem and it is easier to analyze, both empirically and theoretically. However, the tabular case is only realizable in toy domains. In real-world situations the number of states is often very large and the ability to generalize and to recognize similar states is essential. In this section I show how one can replace Algorithm 2 by a neural network that is able to estimate the successor representation from raw pixels. Such an approach circumvents the limitation of requiring a linear feature representation to be provided beforehand. It is built on the concept of successor features, discussed in Chapter 2.

The neural network architecture I used is depicted in Figure 5.1. The reconstruction module is the same as the one introduced by Oh et al. (2015), but augmented by the successor features estimator (the three layers depicted at the bottom). The successor features estimator uses the learned latent representation as input, that is, the output of the representation learning module.

This neural network receives raw pixels as input and learns to estimate the successor features of a learned lower-dimension representation. The loss function $\mathcal{L}_{SR}$ used to learn the successor features is

$$\mathcal{L}_{SR}(s, s') = \mathbb{E}\left[\left(\boldsymbol{\phi}^{-}(s) + \gamma\boldsymbol{\psi}^{-}\big(\boldsymbol{\phi}^{-}(s')\big) - \boldsymbol{\psi}\big(\boldsymbol{\phi}(s)\big)\right)^2\right],$$

where $\boldsymbol{\phi}(s)$ denotes the feature vector encoding the learned representation of state s and $\boldsymbol{\psi}(\cdot)$ denotes the estimated successor features. In practice, $\boldsymbol{\phi}(\cdot)$ is

[1]Notice the matrix $\hat{\Psi}$ is not guaranteed to be symmetric. In that case one can define the eigenpurposes to be $\hat{\Psi}$'s right-singular eigenvectors, as I do in Section 5.3.

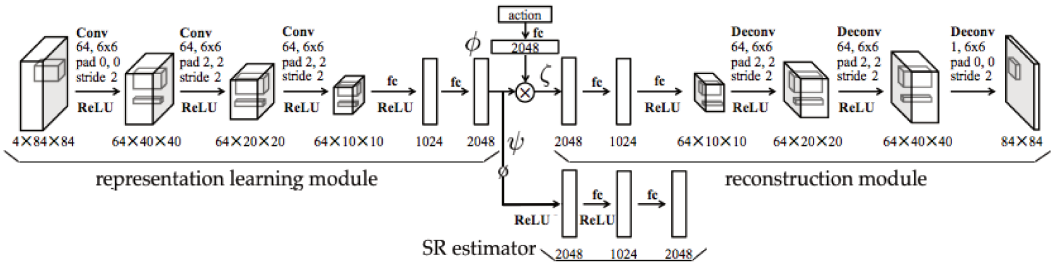


Figure 5.1: Neural network architecture used to learn the SR. The symbols $\otimes$ and $\emptyset$ denote element-wise multiplication and the fact that gradients are not propagated further back, respectively.

the output of the representation learning module and $\boldsymbol{\psi}(\cdot)$ is the output of the SR estimator, as shown in Figure 5.1. The loss function above also highlights the fact that we have two neural networks. As in Chapter 2, I use the symbol $^-$ to represent a *target* network, which is updated at a slower rate for stability purposes.

It is not ideal to directly estimate the successor features from raw pixels using only $\mathcal{L}_{SR}$ because zero is one of its fixed points when also learning ϕ. This is the reason I augmented the network with Oh et al.'s reconstruction module. It behaves as an auxiliary task (Jaderberg et al. 2017) that predicts the *next* state to be observed given the current state and action. By predicting the next state I increase the likelihood the agent will learn a representation that takes into consideration the pixels that are under its control, which has been shown to be a good bias in RL problems (Bellemare, Veness, and Bowling 2012). Such an auxiliary task is defined through the network's reconstruction error $\mathcal{L}_{RE}$:

$$\mathcal{L}_{RE}(s, a, s') = \left|\left|\boldsymbol{\zeta}\big(\boldsymbol{\phi}(s), a\big) - s'\right|\right|^2,$$

where $\boldsymbol{\zeta}(\cdot)$ denotes the output of the reconstruction module, as shown in Figure 5.1. The final loss being optimized is

$$\mathcal{L}(s, a, s') = \mathcal{L}_{RE}(s, a, s') + \mathcal{L}_{SR}(s, s').$$

Finally, to ensure that the SR will not interfere with the learned features, I zero the gradients coming from the SR estimator (represented with the symbol

ø in Figure 5.1). I trained the network with RMSProp and I followed the same protocol Oh et al. (2015) used to initialize the network.

In Algorithm 1, ExtractEigenpurposes returns eigenpurposes, which are defined in terms of a feature representation $\boldsymbol{\phi}(S_t)$ of the environment and of the eigenvectors $\mathbf{e}_i$ of the diffusion model of information flow (the successor representation in this case). I generate both with the trained network. It is trivial to obtain $\boldsymbol{\phi}(S_t)$ as I just use the output of the appropriate layer in the network as the feature representation. To obtain $\mathbf{e}_i$ I first need to generate a meaningful matrix since the network outputs a *vector* of successor features instead of a matrix. I do so by having the agent follow the uniform random policy while I store the network output, $\boldsymbol{\psi}(S_t)$, which corresponds to the network estimate of the successor features of state S_t. I then create a matrix T where row t corresponds to $\boldsymbol{\psi}(S_t)$ and I define $\mathbf{e}_i$ to be its right-singular vectors.

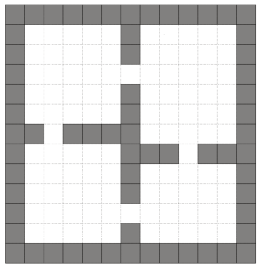
Once I have extracted the eigenpurposes, the option discovery problem is reduced to a regular RL problem where the agent aims to maximize the cumulative sum of rewards. Any learning algorithm can be used for that. In the next section I provide details about the used approach.

5.4 Empirical Evaluation

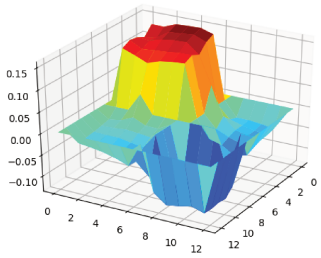
In this section I evaluate the discovered eigenoptions. I use the traditional 4-room domain to evaluate the impact of approximating the successor representation on its eigenvectors, on the discovered options and on the agent's final performance. I then use Atari 2600 games to demonstrate how the proposed network does discover purposeful options from raw pixels that are very similar to those discovered in the previous chapter.

5.4.1 Diffusion Time in the Tabular case

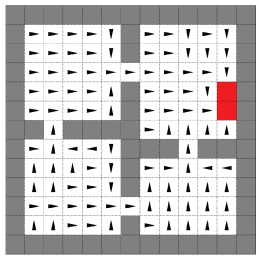
My first experiment evaluates the impact, in the agent's diffusion time, of estimating the SR from samples instead of assuming that an adjacency matrix representing the environment is available. I performed this evaluation in the



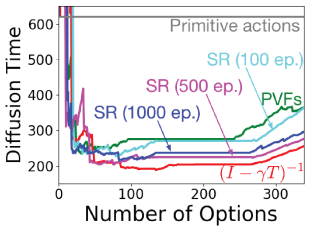
(a) 4-rooms domain



(b) First eigenvector



(c) First eigenoption

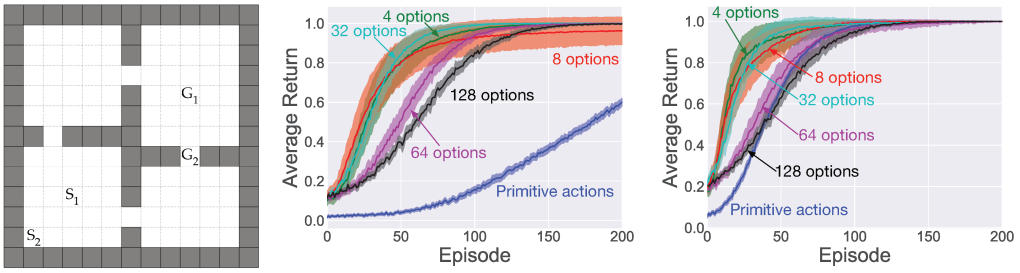


(d) Diffusion time

Figure 5.2: Results in the 4-room domain. Figure 5.2d depicts the diffusion time as eigenoptions are added to the agent's action set.

the 4-room domain (see Figure 5.2a). Figure 5.2b depicts the first eigenvector obtained from the SR while Figure 5.2c depicts the corresponding eigenoption. The agent followed the uniform random policy for 1,000 episodes to learn the SR. Episodes were 100 time steps long. I used a step-size of 0.1, and I set $\gamma = 0.9$. The estimated eigenvector is fairly close to the true one and the obtained eigenvector is fairly similar to the PVFs of this domain.

I compared the agent's diffusion time when using eigenoptions obtained with PVFs to the diffusion time when using eigenoptions obtained with estimates of the SR to evaluate whether the eigenoptions discovered with the SR still improve the agent's ability to explore. As we can see in Figure 5.2d, the eigenoptions obtained with the SR do help the agent explore. The gap between the diffusion time when using PVFs and when using the SR is due to different ways of dealing with corners. The SR implicitly models self-loops in the states adjacent to walls, since the agent takes an action and it observes it did not move. My implementation of PVFs does not model self-loops.



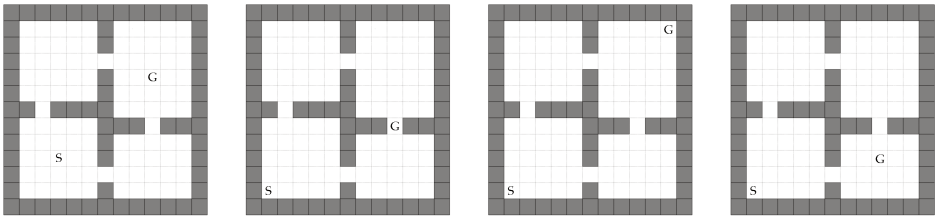
(a) Environments (b) Results for S_1, G_1 (c) Results for S_2, G_2

Figure 5.3: Different environments used in my evaluation (a), as well as the learning curves obtained in each one of these environments (b, c) for different number of options obtained from the SR when estimated after 100 episodes. Two settings are being depicted. In the first setting the agent always starts the episode in the state assigned with S_1 and the episode terminates when the agent reaches state G_1. In the second setting the start and goal states are assigned with S_2 and G_2.

Fig 5.2d also depicts how the diffusion time evolves as more episodes are used to learn the successor representation. Naturally, more episodes allow the agent to learn more accurate estimates of the SR as a more global facet of the environment is seen, since the agent has more chances to further explore the state space (see Figure 5.6 for an example). However, it seems that even the SR learned from few episodes already allow us to discover useful eigenoptions. The eigenoptions obtained from the SR learned using only 100 episodes are already capable of reducing the agent's diffusion time considerably. Finally, it is important to stress that the discovered options do more than randomly selecting subgoal states. Random options, as discussed in the previous chapter, only reduce the agent's diffusion time when hundreds of them are added to the agent's action set.

5.4.2 Using Eigenoptions to Accumulate Reward

I also evaluated the use of the discovered eigenoptions to maximize reward. The agent learned, off-policy, the greedy policy over primitive actions (target policy) while following the uniform random policy over actions and eigenoptions (behavior policy). I used Q-learning (Watkins and Dayan 1992) in these experiments – parameters $\lambda = 0$, $\alpha = 0.1$, and $\gamma = 0.9$. As before, episodes are



(a) Setting 1 (b) Setting 2 (c) Setting 3 (d) Setting 4

Figure 5.4: Different environments (varying start and goal locations) used when evaluating the agent's ability to accumulate reward.

100 time steps long. Figure 5.3 summarizes the obtained results comparing the performance of my approach to regular Q-learning over primitive actions. The eigenoptions were extracted from estimates of the SR obtained after 100 episodes. The reported results are the average over 24 independent runs when learning the SR, with each one of these runs encoding 100 runs evaluating Q-Learning. The options were added following the sorting provided by the eigenvalues. For example, *4 options* denotes an agent with the action set used in the behavior policy being composed of the four primitive actions and the four eigenoptions generated by the top 2 eigenvalues (both directions are being used). Notice that these results do not try to take the sample efficiency of my approach into consideration, they are meant to showcase how eigenoptions, once discovered, can speed up learning. The sample complexity of learning options is generally justified in lifelong learning settings where they are re-used over multiple tasks. This is beyond the scope of this dissertation.

The obtained results show that eigenoptions are not only capable of reducing the diffusion time in the environment but of also improving the agent's control performance. They do so by increasing the likelihood that the agent will cover a larger part of the state space given the same amount of time. Moreover, as before, it seems that a very accurate estimate of the successor representation is not necessary for the eigenoptions to be useful.

Similar results can be obtained for different locations of the start and goal states, and when the estimates of the SR are more accurate. Figure 5.5 summarizes the results comparing the performance of my approach to regular Q-learning over primitive actions in four environments (see Figure 5.4). I evaluate

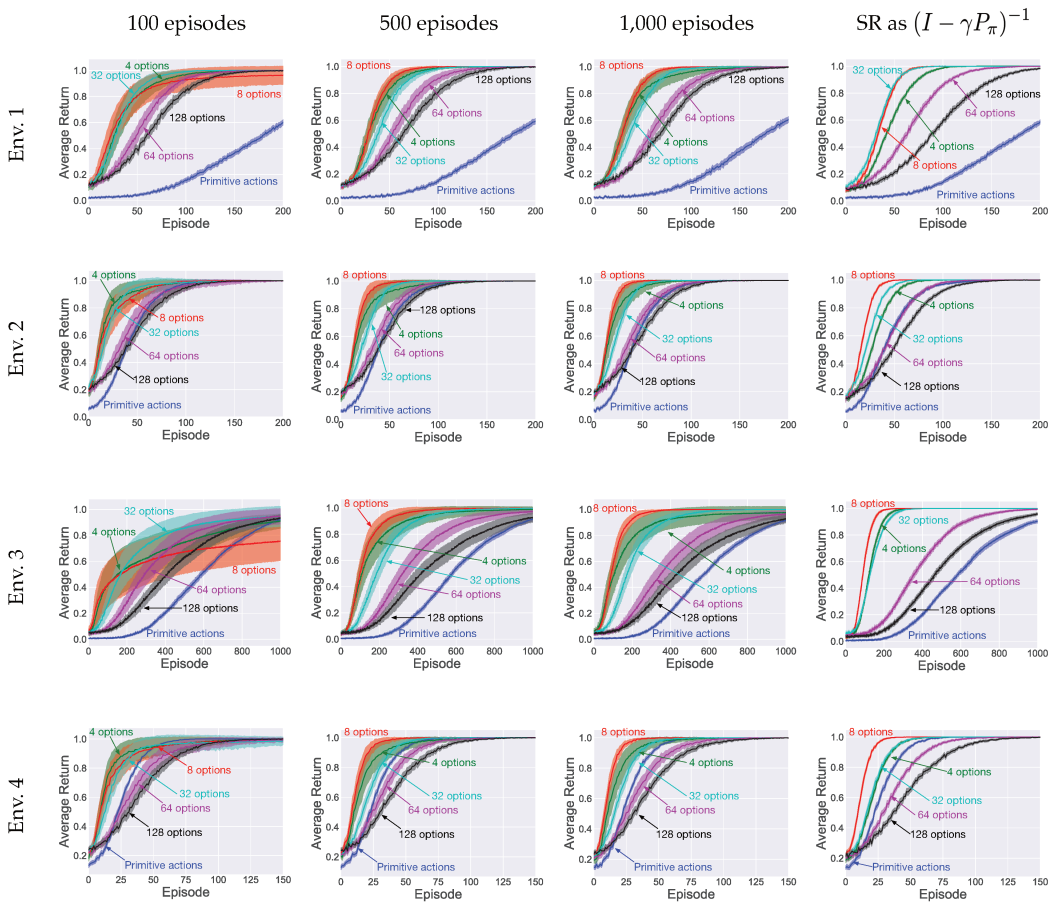


Figure 5.5: Agent's performance when following options obtained with estimates of the SR (100, 500, and 1,000 episodes) and the true one, $(I - \gamma P_\pi)^{-1}$, in four different environments.

the agent's performance when using eigenoptions extracted from estimates of the SR obtained after 100, 500, and 1000 episodes, as well eigenoptions obtained from the true SR, $(I - \gamma T)^{-1}$. The results are the average over 24 independent runs when learning the SR, with each one of these runs encoding 100 runs evaluating Q-Learning. We can see that more accurate predictions of the SR are able to further improve the agent's performance, mainly when dozens of eigenoptions are being used. The first eigenoptions to be accurately estimated are those with larger eigenvalues, which are the ones I add first.

5.4.3 The Impact of Learning the SR

So far I have discussed the impact of estimating the SR from samples

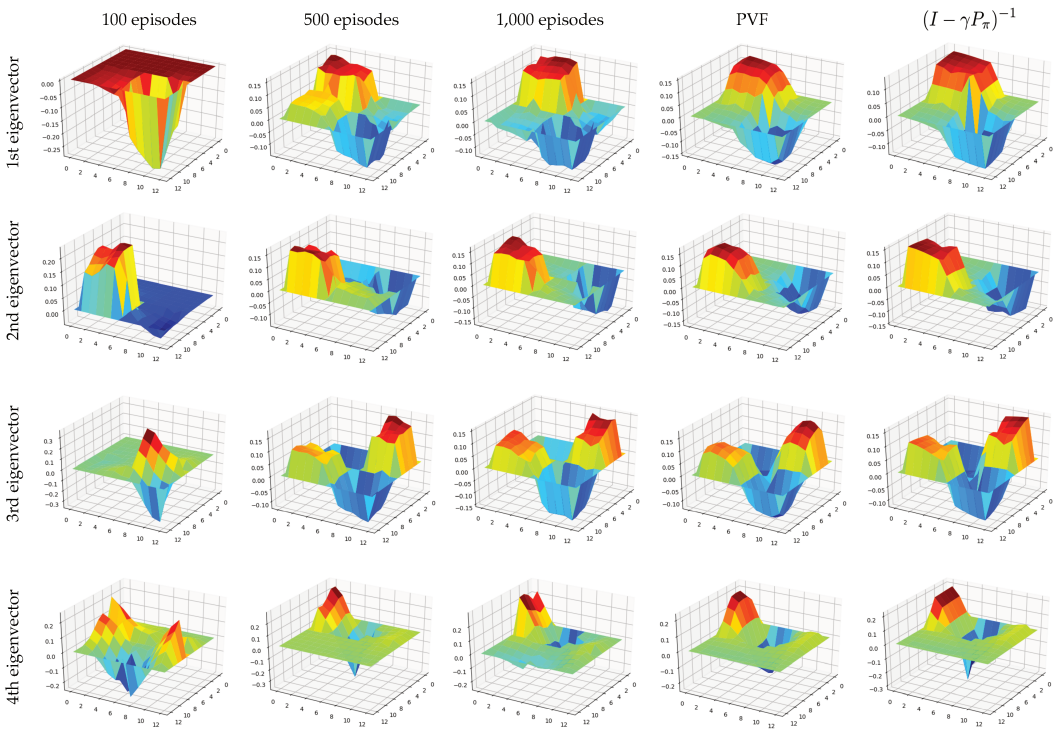


Figure 5.6: Evolution of the top four eigenvectors being estimated by the SR.

instead of assuming the agent has access to the normalized Laplacian. In this section I present the eigenvectors of the estimates of the SR at different moments and I compare them to PVFs and to the eigenvectors of the matrix $(I - \gamma T)^{-1}$.

Figure 5.6 depicts the first four eigenvectors of the successor representation in the 4-room domain, after being learned for different number of episodes (episodes were 100 time steps long, $\eta = 0.1$, $\gamma = 0.9$). I also present the corresponding eigenvectors of the $(I - \gamma T)^{-1}$ matrix, and the PVFs. Because the eigenvectors orientation (sign) is often arbitrary in an eigendecomposition, I matched their orientation to ease visualization.

Overall, after 500 episodes we can observe an almost perfect estimate of the first eigenvectors in the environment; while 100 episodes seem to not be enough to accurately learn the diffusion model of information flow in all rooms. However, learning the SR for 100 episodes seems to be enough to generate eigenoptions that reduce the agent's diffusion time, as I have shown in Figure 5.2d. This behavior can be better understood when looking at Figure 5.7,

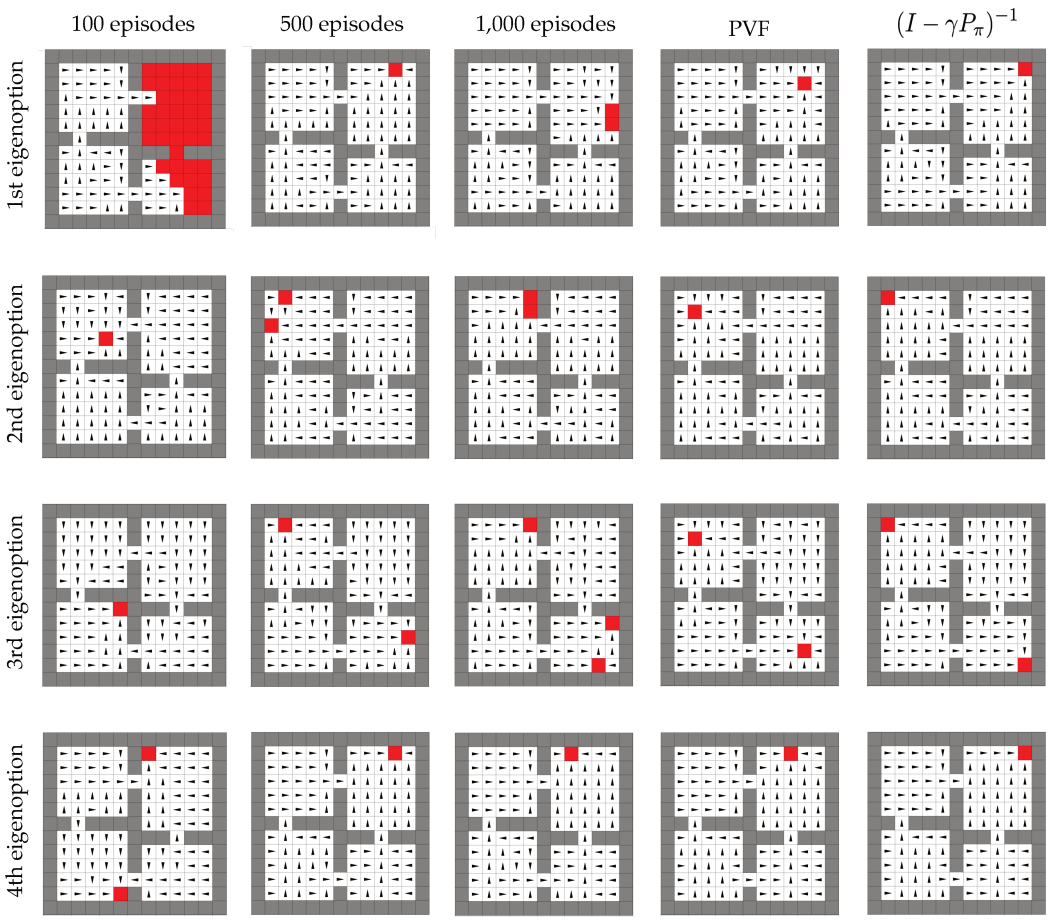


Figure 5.7: Evolution of the top four eigenoptions being estimated by the SR.

which depicts the options generated by the obtained eigenvectors.

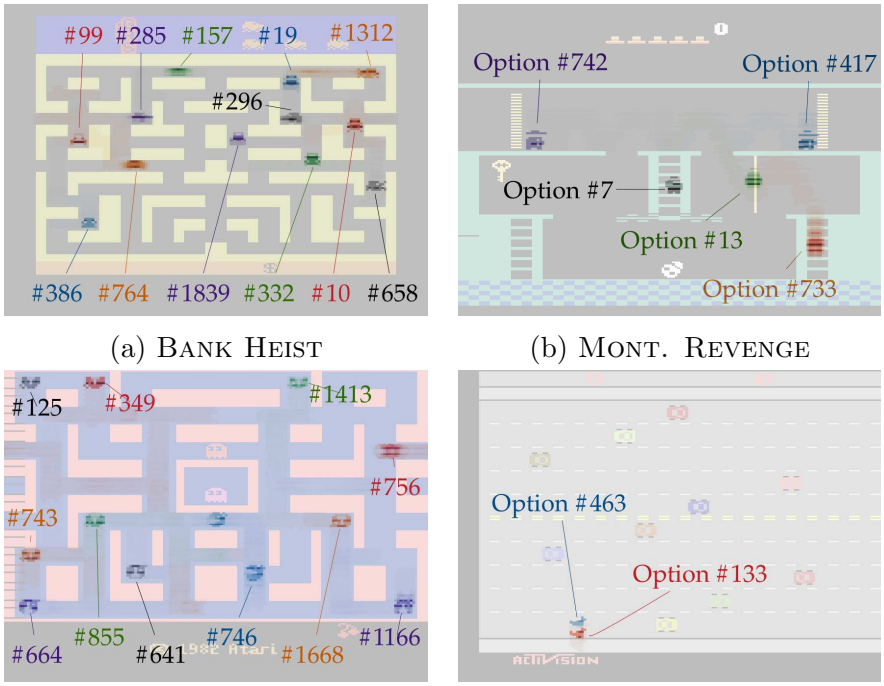
With the exception of the options generated after learning the successor representation for 100 episodes, all the eigenoptions obtained from estimates of the successor representation already move the agent towards the "correct" room(s). Naturally, they do not always hit the corners, but the general structure of the policies can be clearly seen. Finally, I use the first plot of Figure 5.7 (top left corner) to speculate why the options learned after 100 episodes are capable of reducing the agent's diffusion time. The first eigenoption learned by the agent moves it to the parts of the state space it has never been to, this may be the reason that the combination of these options is so effective. It also suggests that incremental methods for option discovery and exploration are a promising path, which is something I explore in the next chapter.

5.4.4 Non-Linear Function Approximation

In this section I evaluate the eigenoptions discovered when the successor representation is obtained from raw pixels using the neural network described in this chapter. I used four Atari 2600 games via the Arcade Learning Environment (ALE) as testbed: Bank Heist, Freeway, Montezuma's Revenge, and Ms. Pac-Man.

I trained the network in Figure 5.1 to estimate the SR under the uniform random policy. Since the network does not impact the policy being followed, I built a dataset of 500,000 samples for each game and I used this dataset to optimize the network weights. I passed through the shuffled dataset 10 times, using RMSProp with a step size of 10^{-4}. Once I was done with the training, I let the agent follow a uniform random policy for 50,000 steps while I stored the SR output by the network for each observed state as a row of matrix T. I define $\mathbf{e}$, the eigenpurposes the agent maximizes, to be the right-singular vectors of the matrix T, while $\boldsymbol{\phi}(\cdot)$ is extracted at each time step from the network in Figure 5.1. Due to computational constraints, I approximated the final eigenoptions. As in the previous chapter, I did so by using the ALE's internal emulator to do a one-step lookahead and act greedily with respect to each eigenpurpose (in practice, this is equivalent to learning with $\gamma = 0$). This is not ideal because the options I obtain are quite limited as they do not deal with delayed rewards. However, even in this limiting setting I obtained promising results, as I discuss below.

As in the previous chapter, I evaluate the discovered eigenoptions qualitatively. I execute all options following the procedure described above (greedy one-step lookahead) while tracking the avatar's position on the screen. Figure 5.8 summarizes the behavior of some of the meaningful options discovered. The trajectories generated by different options are represented by different colors and the color's intensity at a given location represents how often the agent was at that location. I argue that eigenoptions are options that generate purposeful behavior and we can clearly see that the discovered eigenoptions are indeed purposeful. They aim to reach a specific location and stay there. If



(a) Bank Heist (b) Mont. Revenge

(c) Ms. Pacman (d) Freeway

Figure 5.8: Plots of density of state visitation of eigenoptions discovered in four Atari 2600 games. States visited more frequently show darker images of the avatar. Note that an eigenoption's overwhelming mass of visitations corresponds to its terminal state, and that disparate options have different terminal states.

this was not the case the agent's trajectory would be much more visible. Instead, what we actually observe is that the mass of visitation is concentrated on one location on the screen, dominating (color intensity) all the others. The location the agent is spending most of its time on can in fact be seen as the option's terminal state. Constantly being in a state suggests the agent has arrived to a myopic local maximum for that eigenpurpose.

In three out of four games (Bank Heist, Montezuma's Revenge, Ms. Pacman) the proposed algorithm discovers options that clearly push the agent to corners and to other relevant parts of the state space, corroborating the intuition that eigenoptions also improve exploration. In Montezuma's Revenge, the terminal state of the highlighted options even correspond to what are considered good subgoals for the game (Kulkarni, Narasimhan, et al. 2016). It is likely that additional subgoals, such as the key, were not found due to

the myopic greedy approach. This approach may also explain why my algorithm was ineffective in FREEWAY. Avoiding cars may be impossible without longer-term planning or a more varied set of start states. The fact that myopic policies are able to navigate to specific locations and stay there also suggests that, as in the tabular case, the proposed approach gives rise to dense intrinsic rewards that are very informative. This is an important constrast between randomly assigned subgoals and my approach. Randomly assigned subgoals do not give rise to such dense rewards. Thus, one can argue that this approach does not only generate useful options but it also gives rise to dense eigenpurposes, making it easier to build the policies associated with them.

It is important to stress that the proposed algorithm was able to discover eigenoptions, *from raw pixels*, similar to those obtained in the previous chapter, which used the RAM state of the game as a feature representation. The RAM state of the game often uses specific bytes to encode important information of the game, such as the position of the player's avatar in the game. The algorithm in this chapter had to implicitly learn what were the meaningful parts of the screen. Also, different from before, this approach is not constrained by the dimensionality of the state representation, a limitation that prevented me in the last chapter from using the best linear features available for Atari 2600 games (Liang et al. 2016). Based on this discussion, I consider these results to be promising, even though I only depict options that have effect on the initial state of the games. I believe that in a more general setting (e.g., using DQN to learn policies) this algorithm has the potential to discover even better options.

5.5 Discussion

In this chapter I introduced a new algorithm for eigenoption discovery in RL. My algorithm uses the successor representation (SR) to estimate the diffusion model of information flow in the environment, leveraging the equivalence between proto-value functions (PVFs) and the eigenvectors of the SR when it is defined with respect to the uniform random policy in a symmetric and de-

terministic environment. This approach circumvents several limitations from the algorithm presented in the previous chapter: 1) it builds increasingly accurate estimates using a constant-cost update-rule; 2) it naturally deals with stochastic MDPs; 3) it does not depend on the assumption that the transition matrix is symmetric; and 4) it does not depend on handcrafted feature representations. The first three items were achieved by simply using the SR instead of the PVFs, while the latter was achieved by using a neural network to estimate the SR.

The algorithms presented here can also be seen as instantiations of the option discovery cycle proposed in Chapter 3 (page 37). Actually, the only step that is different from what I discussed in Chapter 4 is the representation learning step. Instead of having an agent following the random policy while learning proto-value functions (PVFs) as the linear representation, the agent now follows a random policy while learning the successor representation. The SR can be seen as a linear representation in both the tabular and non-linear function approximation cases. The agent selects the top k features according to the obtained eigenvalues and proceeds to learn to attain those features via eigenpurposes. Each one of the options then consists of an eigenbehavior and the initiation and termination sets as previously defined. As in Chapter 4, I presented results for a single iteration of the cycle in this chapter. Some initial results of multiple iterations of this cycle are discussed in Chapter 7.

Finally, although the ideas presented in this chapter address multiple limitations of the framework proposed in Chapter 4, this algorithm is still not fully practical in settings such as Atari 2600 games. The main reason behind it is the fact that learning multiple options is computationally intensive and different from the tabular case, one cannot simply rely on the sorting provided by the eigenvalues to select eigenpurposes. There are a couple of reasons for this, such as the fact that a random walk does not necessarily capture the true distribution of state visitation in a short amount of time. A better neural network architecture, that seems to be able to generate a better sorting of interesting eigenpurposes was recently proposed by Pfau et al. (2018), as I discuss in details in Chapter 7. In the next chapter I explore the observation that

the top eigenvector of the successor representation, while being learned, seems to be a good guiding signal for exploration, as mentioned in Section 5.4.3.

Chapter 6

Count-Based Exploration with the Successor Representation

In the previous chapter we have seen that the top eigenpurpose of the successor representation (SR), while it is being learned, seems to incentivize the agent to navigate to states it has visited less often. Motivated by this observation, in this chapter I introduce algorithms that use the SR to directly generate exploration bonuses. This is a simple approach that allows me to develop algorithms for the tabular case that are also extendable to settings where function approximation is required. The exploration bonus I propose here is the norm of the transient SR for the current state.

In this chapter I also introduce the substochastic successor representation (SSR) in order to try to understand the behavior of the proposed exploration bonus. The SSR behaves similarly to the SR but it is more amenable to theoretical analysis. I am able to show that the SSR implicitly counts state visitation, suggesting that maybe the exploration bonus obtained from the transient SR is also incorporating some notion of state visitation counts. I proceed to show that a standard model-based RL algorithm that uses the norm of the SSR as an exploration bonus performs as well as algorithms with provable PAC-MDP bounds in hard exploration tasks. Finally, I extend the idea of using the norm of the SR as an exploration bonus to the function approximation setting, designing a model-free deep RL algorithm that achieves state-of-the-art performance in hard exploration Atari 2600 games (Bellemare,

Srinivasan, et al. 2016; Burda et al. 2019; Ostrovski et al. 2017) when evaluated in a low sample-complexity regime.

6.1 The Norm of the Successor Representation as an Exploration Bonus

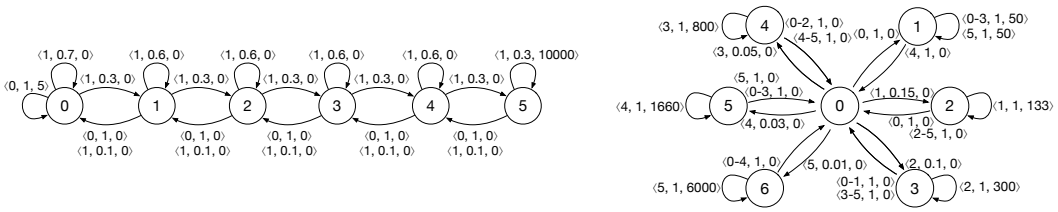
The thesis statement in this dissertation is that time-based representations can be used to guide exploration in RL. The previous chapters presented some evidence supporting this claim, showing for example that the SR incorporates diffusion properties of the environment and how it can be used to promote exploration via options. As aforementioned, in this chapter I argue that the SR can be used in a more direct way to promote exploration. I show that the magnitude of the norm of the SR, while it is being learned, behaves as an exploration bonus. In this section I first demonstrate it empirically, in the tabular case, to clearly present the idea behind the proposed algorithm. I then introduce the substochastic successor representation to provide some theoretical intuition of why the proposed exploration bonus is effective.

6.1.1 Empirical Demonstration in Tabular Model-Free Reinforcement Learning

I demonstrate the usefulness of the norm of the SR as an exploration bonus by first comparing the performance of an algorithm that uses this exploration bonus to an algorithm that uses an ϵ-greedy exploration strategy. I use the Sarsa update rule (Rummery and Niranjan 1994) to implement both approaches. I refer to the algorithm that uses an ϵ-greedy strategy as Naïve Sarsa while the one that uses the proposed exploration bonus as Sarsa+SR. The update equation for Sarsa+SR is

$$\hat{q}(S_t, A_t) \leftarrow \hat{q}(S_t, A_t) + \alpha \left(R_t + \beta \frac{1}{||\hat{\Psi}(S_t)||_1} + \gamma \hat{q}(S_{t+1}, A_{t+1}) - \hat{q}(S_t, A_t) \right), \quad (6.1)$$

where β is a scaling factor and, at each time step t, $\hat{\Psi}(S_t, \cdot)$ is updated before



(a) RiverSwim (b) SixArms

Figure 6.1: Domains used as testbed in the tabular case. The tuples in each transition should be read as ⟨action id, probability, reward⟩.

$\hat{q}(S_t, A_t)$ via temporal-difference learning, as discussed in Chapter 2:

$$\hat{\Psi}(S_t, j) \leftarrow \hat{\Psi}(S_t, j) + \eta\Big(\mathbb{1}_{\{S_t=j\}} + \gamma_{SR}\hat{\Psi}(S_{t+1}, j) - \hat{\Psi}(S_t, j)\Big). \tag{6.2}$$

I evaluated Sarsa+SR in RiverSwim and SixArms, traditional domains in the PAC-MDP literature (Kakade 2003; Strehl and Littman 2008) that are often used to evaluate provably sample-efficient algorithms (see Figure 6.1). In these domains it is very likely that an agent will first observe a small reward generated in a state that is easy to get to. If the agent does not have a good exploration policy it is likely it will converge to a suboptimal behavior, never observing larger rewards available in states that are difficult to get to. These are features often found in larger domains such as Atari 2600 games and robotics. These domains allow me to investigate the behavior of my algorithm without having to also take other aspects into consideration, such as feature learning. For SixArms, the agent starts in state 0. For RiverSwim, the agent starts in either state 1 or 2 with equal probability.

The results suggest that the proposed exploration bonus has a profound impact in the algorithm's performance. When evaluating the agent for 5,000 time steps, Naïve Sarsa obtains an average return of approximately 26,000, while Sarsa+SR obtains an approximate average return of 1.8 million! Notice that, in RiverSwim, the reward that is "easy to get" has value 5, implying that, different from Sarsa+SR, Naïve Sarsa almost never explores the state space well enough to discover larger rewards. In SixArms the trend is the same. Naïve Sarsa obtains an approximate average return of 284,000 while Sarsa+SR achieves approximately 2.2 millon. The actual numbers, which were averaged over 100 runs, are available in Table 6.1.

Table 6.1: Comparison between Naïve Sarsa and Sarsa+SR. A 95% confidence interval is reported between parentheses.

	Naïve Sarsa		Sarsa + SR	
RiverSwim	26,526	(2,351)	1,792,126	(258,709)
SixArms	284,013	(88,552)	2,176,044	(449,962)

Both algorithms acted ϵ-greedily maximizing the discounted return using $\gamma = 0.95$. For Sarsa+SR, I swept over different values of α, η, γ_{SR}, β and ϵ, with $\alpha \in \{0.01, 0.05, 0.1, 0.25, 0.5\}$, $\eta \in \{0.01, 0.05, 0.1, 0.25, 0.5\}$, $\gamma_{SR} \in \{0.5, 0.8, 0.95, 0.99\}$, $\beta \in \{1, 10, 100, 1000, 10000\}$ and $\epsilon \in \{0.01, 0.05, 0.1\}$. For Naïve Sarsa, I swept over the parameters α and ϵ. For fairness, I looked at a finer granularity for these parameters, with $\alpha \in i \times 0.005$ for i ranging from 1 to 100, and with $\epsilon \in j \times 0.01$ for j ranging from 1 to 15. The reported performance of Sarsa+SR was obtained with $\alpha = 0.1$, $\gamma_{SR} = 0.5$, $\epsilon = 0.01$, $\beta = 10000$ in both settings, with $\eta = 0.5$ in RiverSwim and $\eta = 0.25$ in SixArms. The Naïve Sarsa results were obtained with $\alpha = 0.37$ and $\epsilon = 0.12$ in RiverSwim and $\alpha = 0.43$ and $\epsilon = 0.01$ in SixArms. The value function of Naïve Sarsa was pessimistically initialized to 0 because my goal was to use random exploration as baseline, not approaches such as optimistic initialization that are not easily applicable to the function approximation setting. The reported results hold for a wide range of parameter values.

6.1.2 Theoretical Justification

It is difficult to characterize the behavior of the proposed exploration bonus because it is updated at each time step with TD learning. It is hard to analyze the transient behavior of estimates obtained with TD learning. Also, at its fixed point, the ℓ_1-norm of the SR is $\sum \gamma^t 1 = 1/(1-\gamma)$ for all states, making it hard to use the fixed point of the SR to theoretically analyze the behavior of this exploration bonus. In this section I introduce the *substochastic successor representation* (SSR) to provide some theoretical intuition of why the norm of the SR is a good exploration bonus. The SSR behaves similarly to the SR but it is simpler to analyze.

Definition 6.1.1 (Substochastic Successor Representation)**.** Let $\tilde{P}$ denote the substochastic matrix induced by the environment's dynamics and by the policy π such that $\tilde{P}(s'|s) = \frac{n(s,s')}{n(s)+1}$. For a given $0 \leq \gamma < 1$, the substochastic successor representation, $\tilde{\Psi}$, is defined as:

$$\tilde{\Psi} = \sum_{t=0}^{\infty} \gamma^t \tilde{P}^t = (I - \gamma \tilde{P})^{-1}.$$

The SSR only differs from the empirical SR in its incorporation of an additional "phantom" transition from each state, making it underestimate the real SR. Through algebraic manipulation one can show that the SSR allows us to recover an estimate of the visit counts, $n(s)$. This result, stated in Theorem 6.1.1, provides some intuition of why the exploration bonus I propose in this chapter performs so well, as exploration bonuses based on state visitation counts are known to generate proper exploration.

As aforementioned, the SSR behaves similarly to the SR. When computing the norm of the SR, while it is being learned with TD learning, it is as if a reward of 1 was observed at each time step.[1] Thus, there is little variance in the target, with the predictions slowly approaching the true value of the SR. If pessimistically initialized, as traditionally done, the estimates of the SR approach the target from below. In this sense, the number of times a prediction has been updated in a given state is a good proxy to estimate how far this prediction is from its final target. From the definition above we can see that the SSR have similar properties. It underestimates the true target but slowly approaches it, converging to the true SR in the limit. The SSR simplifies the analysis by not taking bootstrapping into consideration.

The theorem below formalizes the idea that the ℓ_1-norm of the SSR implicitly counts state visitation, shedding some light on why the exploration bonus I propose seems to work so well.

[1]When estimating the SR with TD learning, in vector form, the clause $\mathbb{1}_{\{S_t=j\}}$, from Equation 6.2, is always true for one of the states, that is, an entry in the vector representing the SR. Thus, it is as if a reward of 1 was observed at each time step.

Theorem 6.1.1. *Let $n(s)$ denote the number of times state s has been visited and let $\tilde{\Psi}$ denote the substochastic successor representation as in Definition 6.1.1. For a given $0 \leq \gamma < 1$,*

$$\frac{\gamma}{n(s)+1} - \frac{\gamma^2}{1-\gamma} \leq (1+\gamma) - ||\tilde{\Psi}(s)||_1 \leq \frac{\gamma}{n(s)+1}$$

Proof of Theorem 6.1.1. Let $\hat{P}_\pi$ be the empirical transition matrix. I first rewrite $\tilde{P}$ in terms of $\hat{P}_\pi$:

$$\begin{aligned}\tilde{P}(s,s') &= \frac{n(s,s')}{n(s)+1} = \frac{n(s)}{n(s)+1}\frac{n(s,s')}{n(s)} = \frac{n(s)}{n(s)+1}\hat{P}_\pi(s,s') \\ &= \Big(1 - \frac{1}{n(s)+1}\Big)\hat{P}_\pi(s,s').\end{aligned}$$

The expression above can also be written in matrix form: $\tilde{P} = (I-N)\hat{P}_\pi$, where $N \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ denotes the diagonal matrix of augmented inverse counts. Expanding $\tilde{\Psi}$ we have:

$$\tilde{\Psi} = \sum_{t=0}^{\gamma}(\gamma\tilde{P})^t = I + \gamma\tilde{P} + \sum_{t=2}^{\infty}(\gamma\tilde{P})^t = I + \gamma\tilde{P} + \gamma^2\tilde{P}^2\tilde{\Psi}.$$

The top eigenvector (the eigenvector with largest eigenvalue) of a stochastic matrix is the all-ones vector, $\mathbf{1}$ (Meyn and Tweedie 2012), and it corresponds to the eigenvalue 1. Using this fact and the definition of $\tilde{P}$ with respect to $\hat{P}_\pi$ we have:

$$\begin{aligned}(I+\gamma\tilde{P})\mathbf{1} + \gamma^2\tilde{P}^2\tilde{\Psi}\mathbf{1} &= \big(I + \gamma(I-N)\hat{P}_\pi\big)\mathbf{1} + \gamma^2\tilde{P}^2\tilde{\Psi}\mathbf{1} \\ &= (I+\gamma)\mathbf{1} - \gamma N\mathbf{1} + \gamma^2\tilde{P}^2\tilde{\Psi}\mathbf{1}. \qquad (6.3)\end{aligned}$$

We can now bound the term $\gamma^2\tilde{P}^2\tilde{\Psi}\mathbf{1}$ using the fact that $\mathbf{1}$ is also the top eigenvector of the SR and has eigenvalue $\frac{1}{1-\gamma}$:

$$0 \leq \gamma^2\tilde{P}^2\tilde{\Psi}\mathbf{1} \leq \frac{\gamma^2}{1-\gamma}\mathbf{1}.$$

Plugging Equation 6.3 into the definition of the successor representation we have (notice that $\Psi(s)\mathbf{1} = ||\Psi(s)||_1$):

$$(1+\gamma)\mathbf{1} - (1+\gamma)\mathbf{1} + \gamma N\mathbf{1} - \gamma^2\tilde{P}^2\tilde{\Psi}\mathbf{1} \;=\; \gamma N\mathbf{1} - \gamma^2\tilde{P}^2\tilde{\Psi}\mathbf{1} \;\leq\; \gamma N\mathbf{1}.$$

When we also use the other bound on the quadratic term we conclude that, for any state s,

$$\frac{\gamma}{n(s)+1} - \frac{\gamma^2}{1-\gamma} \leq (1+\gamma) - ||\tilde{\Psi}(s)||_1 \leq \frac{\gamma}{n(s)+1}.$$

□

In other words, the SSR, obtained after a slight change to the SR, can be used to recover state visitation counts. The intuition behind this result is that the phantom transition, represented by the +1 in the denominator of the SSR, serves as a proxy for the uncertainty about that state by underestimating the successor representation. This is due to the fact that $\sum_{s'} \tilde{P}(s, s')$ gets closer to 1 each time state s is visited.

6.1.3 Exploration in Model-based RL with the SSR

Inspired by the result above, I implemented a simple model-based algorithm that penalizes the agent for visiting commonly visited states. This algorithm is named ESSR and it uses an exploration bonus equals to $r_{\text{int}}(s) = -||\tilde{\Psi}(s)||_1$.[2] In ESSR the agent maximizes the reward function $r(s,a) + \beta r_{\text{int}}(s)$, with β being a scaling parameter. The shift $1+\gamma$ in the theorem above has no effect in the agent's policy because it is the same across all states. The agent updates its SSR estimate as in Definition 6.1.1. Its transition probability model and reward model are updated through the equations

$$\hat{P}_\pi(s'|s) = \frac{n(s,s')}{n(s)}, \qquad \hat{r}(s) = \frac{C(s,s')}{n(s)}, \tag{6.4}$$

where $n(s,s')$ denotes the number of times the transition $s \to s'$ was observed, $n(s) = \sum_{s' \in \mathcal{S}} n(s,s')$, and $C(s,s')$ denote the sum of the rewards associated with the $n(s,s')$ transitions (I drop the action in the discussion to simplify notation). Algorithm 3 depicts the pseudo-code of ESSR. I use $\hat{P}_\pi$ and $\hat{r}$ for the empirical estimates of P_π and r.

I evaluated ESSR in RiverSwim and SixArms, with the algorithm maximizing the discounted return ($\gamma = 0.95$) in both environments. I used policy

[2]The code used to generate these results is available at: `https://github.com/mcmachado/count_based_exploration_sr/tree/master/tabular`.

Algorithm 3 Exploration through the Substochastic Successor Representation (ESSR)

$n(s, s') \leftarrow 0 \qquad \forall s, s' \in \mathcal{S}$
$t(s, a, s') \leftarrow 1 \qquad \forall s, s' \in \mathcal{S}, \forall a \in \mathcal{A}$
$\hat{r}(s, a) \leftarrow 0 \qquad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$
$\hat{P}(s, a) \leftarrow 1/|\mathcal{S}| \qquad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$
$\tilde{P}(s, s') \leftarrow 0 \qquad \forall s, s' \in \mathcal{S}$
$\pi \leftarrow$ random over $\mathcal{A}$
while episode is *not* over **do**
  Observe $s \in \mathcal{S}$, take action $a \in \mathcal{A}$ selected according to $\pi(s)$, and observe a reward R and a next state $s' \in \mathcal{S}$
  $n(s, s') \leftarrow n(s, s') + 1$
  $t(s, a, s') \leftarrow t(s, a, s') + 1$
  $n(s) \leftarrow \sum_{x', b} t(s, b, x')$
  $n(s, a) \leftarrow \sum_{x'} t(s, a, x')$
  $\hat{r}(s, a, s') \leftarrow \frac{(t(s,a,s')-2)\times\hat{r}(s,a,s')+R}{t(s,a,s')-1}$
  for each state $x' \in \mathcal{S}$ **do**
    $\hat{P}(s, a, x') \leftarrow \frac{t(s,a,x')}{n(s,a)}$
    $\tilde{P}(s, x') \leftarrow \frac{n(s,x')}{n(s)+1}$
  end for
  $\tilde{\Psi} \leftarrow (I - \gamma\tilde{P})^{-1}$
  $r_{\text{int}} \leftarrow -\tilde{\Psi}\mathbf{e}$
  $\pi \leftarrow$ PolicyIteration$(\hat{P}, \hat{r} + \beta r_{\text{int}})$
end while

iteration where the policy evaluation step is terminated when the estimates of the value function change by less than 0.01. In RiverSwim β was set to 100 and in SixArms β was set to 1000. These values were obtained after evaluating the algorithm for $\beta \in \{1, 10, 100, 200, 1000, 2000\}$.

Table 6.2 depicts the performance of ESSR as well as the performance of some algorithms with polynomial sample-complexity bounds. The goal with this evaluation is not to outperform these algorithms, but to evaluate how well ESSR performs when compared to algorithms that explicitly keep visitation counts to promote exploration. ESSR performs as well as R-Max (Brafman and Tennenholtz 2002) and E^3 (Kearns and Singh 2002) on RiverSwim and it outperforms these algorithms on SixArms; while MBIE (Strehl and Littman

Table 6.2: Comparison between ESSR, R-MAX, E^3, and MBIE. The numbers reported for R-MAX, E^3, and MBIE are an estimate from the histograms presented by Strehl and Littman (2008). ESSR's performance is the average over 100 runs. A 95% confidence interval is reported between parentheses.

	E^3	R-MAX	MBIE	ESSR	
RIVERSWIM	3,000,000	3,000,000	3,250,000	3,088,924	(± 57,584)
SIXARMS	1,800,000	2,800,000	9,250,000	7,327,222	(± 1,189,460)

2008), which explicitly estimates confidence intervals over the expected return in each state, outperforms ESSR in these domains. These results clearly show that ESSR performs, on average, similarly to other algorithms with PAC-MDP guarantees, suggesting that the norm of the SSR is a promising exploration bonus.

A more careful examination of ESSR sheds some light into these results, as ESSR and R-MAX are actually very similar. As R-MAX, ESSR augments the state-space with an imaginary state and encourages the agent to visit that state, implicitly reducing the algorithm's uncertainty in the state-space. However, while R-MAX deletes the transition to this imaginary state once a state has been visited a given number of times, ESSR lets the probability of visiting this imaginary state vanish with additional visitations.

6.2 Counting Feature Activations with the SR

In large environments, where enumerating all states is not an option, model-based RL algorithms or using Sarsa+SR as described in the previous section are not viable options. Learning the SR becomes even more challenging when the representation, ϕ, is also being learned. Using neural networks to learn a representation while learning to estimate state-action value functions is the approach that currently often leads to state-of-the-art performance in the field. In this section I describe an algorithm that uses the same ideas described so far but in the function approximation setting. The proposed algorithm is inspired by the neural network introduced in the previous chapter, which learns the successor features jointly with the feature representation itself.

The neural network used to learn the agent's value function while also learning the feature representation and the successor representation is depicted in Figure 6.2. The layers used to compute the state-action value function, $\hat{q}(S_t, \cdot)$, are structured as in DQN (Mnih, Kavukcuoglu, et al. 2015), but with a different numbers of parameters (i..e, filter sizes, stride, and number of nodes). This was done to match Oh et al.'s (2015) architecture, which is known to succeed in the auxiliary task of predicting the agent's next observation, as previously discussed. From here on I will call the part of our architecture that predicts $\hat{q}(S_t, \cdot)$ DQN$_e$ to stress the difference between the parameters of this network and DQN. It is trained to minimize

$$\mathcal{L}_{\text{TD}} = \mathbb{E}\Big[\big((1-\tau)\delta(s,a) + \tau\delta_{\text{MC}}(s,a)\big)^2\Big],$$

where $\delta(s,a)$ and $\delta_{\text{MC}}(s,a)$ are defined as

$$\delta(s,a) = R_t + \beta r_{\text{int}}(s;\boldsymbol{\theta}^-) + \gamma \max_{a'} q(s',a';\boldsymbol{\theta}^-) - q(s,a;\boldsymbol{\theta}),$$
$$\delta_{\text{MC}}(s,a) = \sum_{t=0}^{\infty} \gamma^t \Big(r(S_t,A_t) + \beta r_{\text{int}}(S_t;\boldsymbol{\theta}^-)\Big) - q(s,a;\boldsymbol{\theta}).$$

This loss is known as the mixed Monte-Carlo return (MMC) and it has been used in the past by algorithms that achieved succesful exploration in deep reinforcement learning (Bellemare, Srinivasan, et al. 2016; Ostrovski et al. 2017). The distinction between $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^-$ is standard in the field, with $\boldsymbol{\theta}^-$ denoting the parameters of the target network, which is updated less often for stability purposes. As before, I use r_{int} to denote the exploration bonus obtained from the successor features of the internal representation, ϕ, which will be defined below. Moreover, to ensure all features are in the same range, I normalize the feature vector so that $||\boldsymbol{\phi}(\cdot)||_2 = 1$. In Figure 6.2 I highlight with ϕ the layer in which I normalize its output. Notice that the features are always non-negative due to the use of ReLU gates.

The successor features are computed by the two bottom layers of the network, which minimize the loss

$$\mathcal{L}_{\text{SR}} = \mathbb{E}_{\pi,p}\Big[\big(\boldsymbol{\phi}(S_t;\boldsymbol{\theta}^-) + \gamma\boldsymbol{\psi}(S_{t+1};\boldsymbol{\theta}^-) - \boldsymbol{\psi}(S_t;\boldsymbol{\theta})\big)^2\Big].$$

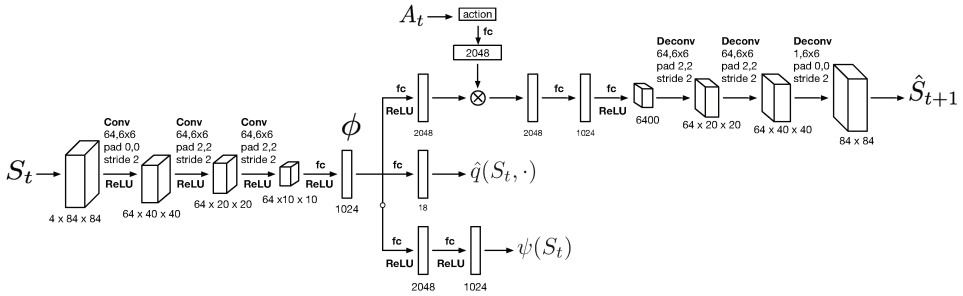


Figure 6.2: Neural network architecture used by my algorithm when learning to play Atari 2600 games.

As previously discussed, zero is a fixed point for the SR. This is particularly concerning in settings with sparse rewards as the agent might end up learning to set $\boldsymbol{\phi}(\cdot) = \mathbf{0}$ to achieve zero loss. I address this problem by not propagating $\nabla \mathcal{L}_{\text{SR}}$ to ϕ (this is depicted in Figure 6.2 as an open circle stopping the gradient), and by creating an auxiliary task (Jaderberg et al. 2017) to encourage a representation to be learned before a non-zero reward is observed. The loss the network minimizes for the auxiliary task is

$$\mathcal{L}_{\text{Recons}} = \left|\left|\hat{S}_{t+1} - S_{t+1}\right|\right|^2.$$

The overall loss minimized by the network is

$$\mathcal{L} = w_{\text{TD}}\mathcal{L}_{\text{TD}} + w_{\text{SR}}\mathcal{L}_{\text{SR}} + w_{\text{Recons}}\mathcal{L}_{\text{Recons}}.$$

The last step in describing my algorithm is to define $r_{\text{int}}(S_t; \boldsymbol{\theta}^-)$, the intrinsic reward used to encourage exploration. I choose the exploration bonus to be the inverse of the ℓ_2-norm of the vector of successor features of the current state, as in Sarsa+SR. That is,

$$r_{\text{int}}(S_t; \boldsymbol{\theta}^-) = \frac{1}{||\boldsymbol{\psi}(S_t; \boldsymbol{\theta}^-)||_2},$$

where $\boldsymbol{\psi}(S_t; \boldsymbol{\theta}^-)$ denotes the successor features of state S_t parametrized by $\boldsymbol{\theta}^-$. The exploration bonus comes from the same intuition presented in the previous section, but instead of penalizing the agent with the norm of the SR I make $r_{\text{int}}(S_t; \boldsymbol{\theta}^-)$ into a bonus (in preliminary experiments not discussed here I observed that DQN performs better when dealing with positive rewards).

Moreover, I use the ℓ_2-norm of the SR instead of the ℓ_1-norm, which was used so far. This mismatch is further discussed in Section 6.3.3. It was driven by the ℓ_2-norm leading to slightly better performance.

Finally, I initialize the network the same way Oh et al. (2015) does. I use Xavier initialization (Glorot and Bengio 2010) in all layers except the fully connected layers around the element-wise multiplication denoted by $\otimes$, which are initialized uniformly with values between -0.1 and 0.1.

6.3 Evaluating the Exploration Bonus in Deep Reinforcement Learning

I evaluated the proposed algorithm on Atari 2600 games through the Arcade Learning Environment (ALE). Following Bellemare, Srinivasan, et al.'s (2016) taxonomy, I evaluated the algorithm in the Atari 2600 games with sparse rewards that pose hard exploration problems. They are: Freeway, Gravitar, Montezuma's Revenge, Private Eye, Solaris, and Venture.[3]

I followed the evaluation protocol proposed by Machado, Bellemare, Talvitie, et al. (2018). I used Montezuma's Revenge to tune the algorithm's parameters. The reported results are the average over 10 seeds after 100 million frames. I evaluated the agents in the stochastic setting (sticky actions, $\varsigma = 0.25$) using a frame skip of 5 with the full action set ($|\mathcal{A}| = 18$). The agent learns from raw pixels, that is, it uses the game screen as input.

The reported results were obtained with the algorithm described in the previous section. I set $\beta = 0.025$ after a rough sweep over values in the game Montezuma's Revenge. I annealed ϵ in DQN's ϵ-greedy exploration over the first million steps, starting at 1.0 and stopping at 0.1 as done by Bellemare, Srinivasan, et al. (2016). I trained the network with RMSprop with a step-size of 0.00025, an ϵ value of 0.01, and a decay of 0.95, which are the standard parameters for training DQN. The discount factor, γ, is set to 0.99 and $w_{\text{TD}} = 1$, $w_{\text{SR}} = 1000$, $w_{\text{Recons}} = 0.001$. The weights w_{TD}, w_{SR}, and

[3]The code used to generate these results is available at: `https://github.com/mcmachado/count_based_exploration_sr/tree/master/function_approximation`.

w_{Recons} were set so that the loss functions would be roughly at the same scale. All other parameters are the same as those used by Mnih, Kavukcuoglu, et al. (2015).

6.3.1 Overal Performance and Baselines

Table 6.3 summarizes the results after 100 million frames. The performance of other algorithms is also provided for reference. Notice I am reporting the *learning performance* for all algorithms instead of the maximum scores achieved by the algorithm. I use the superscript $^{\text{MMC}}$ to distinguish between the algorithms that use MMC from those that do not. When comparing my algorithm, $\text{DQN}_e^{\text{MMC}}$+SR, to DQN we can see how much my approach improves over the most traditional baseline. By comparing my algorithm's performance to DQN^{MMC}+CTS (Bellemare, Srinivasan, et al. 2016) and DQN^{MMC}+PixelCNN (Ostrovski et al. 2017) I compare my algorithm to established baselines for exploration. By comparing my algorithm's performance to Random Network Distillation (RND; Burda et al. 2019) I compare my algorithm to one of the most recent papers in the field with state-of-the-art performance.

As highlighted in Section 6.2, the parameters of the network I used are different from those used in the traditional DQN network, so I also compared the performance of my algorithm to the performance of the same network my algorithm uses but without the additional modules (next state prediction and successor representation) by setting $w_{\text{SR}} = w_{\text{Recons}} = 0$ and without the intrinsic reward bonus by setting $\beta = 0.0$. The column labeled $\text{DQN}_e^{\text{MMC}}$ contains the results for this baseline. This comparison allows me to explicitly quantify the improvement provided by the proposed exploration bonus.

We can clearly see that $\text{DQN}_e^{\text{MMC}}$+SR achieves scores much higher than those achieved by DQN, which struggles in games that pose hard exploration problems. Moreover, by comparing $\text{DQN}_e^{\text{MMC}}$+SR to $\text{DQN}_e^{\text{MMC}}$ we can see that the provided exploration bonus has a big impact in the game Montezuma's Revenge, which is probably known as the hardest game among those I used in my evaluation. Interestingly, the change in architecture and the use of MMC leads to a big improvement in games such as Gravitar and Venture. It

Table 6.3: Performance of the proposed algorithm, $\text{DQN}_e^{\text{MMC}}$+SR, compared to various agents on the "hard exploration" subset of Atari 2600 games. The DQN results reported are from the work by Machado, Bellemare, Talvitie, et al. 2018 (2018) while the DQN^{MMC}+CTS and DQN^{MMC}+PixelCNN results were extracted from the learning curves available in Ostrovski et al.'s (2017) work. $\text{DQN}_e^{\text{MMC}}$ denotes another baseline used in the comparison. When available, standard deviations are reported between parentheses. See text for details.

	DQN	$\text{DQN}_e^{\text{MMC}}$	DQN^{MMC}+CTS	DQN^{MMC}+PixelCNN	RND	$\text{DQN}_e^{\text{MMC}}$+SR
Freeway	32.4 (0.3)	29.5 (0.1)	29.2	29.4	- -	29.5 (0.1)
Gravitar	118.5 (22.0)	1078.3 (254.1)	199.8	275.4	790.0 (122.9)	430.3 (109.4)
Mont. Rev.	0.0 (0.0)	0.0 (0.0)	2941.9	1671.7	524.8 (314.0)	1778.6 (903.6)
Private Eye	1447.4 (2,567.9)	113.4 (42.3)	32.8	14386.0	61.3 (53.7)	99.1 (1.8)
Solaris	783.4 (55.3)	2244.6 (378.8)	1147.1	2279.4	1270.3 (291.0)	2155.7 (398.3)
Venture	4.4 (5.4)	1220.1 (51.0)	0.0	856.2	953.7 (167.3)	1241.8 (236.0)

is likely this is due to the faster credit assignment MMC provides in settings where the reward is not extremely sparse and to the finer granularity of the convolutional layers. However, notice that the change in architecture does not have any effect in Montezuma's Revenge. The proposed exploration bonus seems to be essential in this game. The proposed exploration bonus seems to be essential in games with very sparse rewards. I also compared $\text{DQN}_e^{\text{MMC}}$+SR to DQN^{MMC}+CTS and DQN^{MMC}+PixelCNN. We can observe that, on average, it outperforms these algorithms while being simpler since it does not require a density model. Instead, my algorithm requires the SR, which is domain-independent as it is already defined for every problem since it is a component of the value function estimates, as discussed in Chapter 2.

Finally, $\text{DQN}_e^{\text{MMC}}$+SR also outperforms RND (Burda et al. 2019) when it is trained for 100 million frames. Importantly, RND is currently considered to be the state-of-the-art approach for exploration in Atari 2600 games. Burda et al. did not evaluate RND in Freeway. I computed the performance of RND from the data used by Burda et al. to plot Figure 7 of their paper. The authors shared this data with me. The performance I report is the average performance after 1,530 rollouts. Each rollout consists of 128 time steps with 4 frames per time step (128 environments were executed in parallel), leading to $1{,}530 \times 128 \times 128 \times 4 = 100{,}270{,}080$ frames. I averaged the performance over 3 seeds in the games Gravitar, Private Eye, Solaris, and Venture. The performance reported for Montezuma's Revenge is the average over 10 seeds.

In order to provide additional data about $\text{DQN}_e^{\text{MMC}}$+SR and $\text{DQN}_e^{\text{MMC}}$, regardless of their performance compared to other baselines, I also present their performance after different amounts of experience in Tables 6.4 and 6.5; and their learning curves are depicted in Figure 6.3.

6.3.2 Evaluating the Impact of the Auxiliary Task

While the results depicted in Table 6.3 allow us to clearly see the benefit of using an exploration bonus derived from the SR, they do not inform us about the impact of the auxiliary task in the results. I did evaluate the impact of

Table 6.4: Results obtained with $\text{DQN}_e^{\text{MMC}}$+SR after different amounts of experience.

Game	10M frames		50M frames		100M frames	
Freeway	24.9	(0.5)	29.5	(0.1)	29.5	(0.1)
Gravitar	244.1	(23.8)	326.4	(53.0)	430.3	(109.4)
Mont. Revenge	2.6	(7.2)	563.8	(465.7)	1778.6	(903.6)
Private Eye	99.2	(1.2)	98.5	(3.3)	99.1	(1.8)
Solaris	1547.5	(410.9)	2036.3	(339.0)	2155.7	(398.3)
Venture	26.2	(22.1)	942.0	(423.8)	1241.8	(236.0)

Table 6.5: Results obtained with $\text{DQN}_e^{\text{MMC}}$ after different amounts of experience.

Game	10M frames		50M frames		100M frames	
Freeway	25.7	(1.5)	29.6	(0.1)	29.5	(0.1)
Gravitar	229.9	(31.3)	559.3	(75.9)	1078.3	(254.1)
Mont. Revenge	0.0	(0.0)	0.0	(0.0)	0.0	(0.0)
Private Eye	216.7	(219.5)	109.1	(44.1)	113.4	(42.3)
Solaris	2230.0	(322.3)	2181.5	(292.9)	2244.6	(378.8)
Venture	63.8	(31.3)	794.1	(151.9)	1220.1	(51.0)

the introduced exploration bonus by reporting the performance of $\text{DQN}_e^{\text{MMC}}$ but I did not evaluate the impact of the auxiliary task introduced to the network yet. It is well-known that auxiliary tasks tend to improve agents' performance (Jaderberg et al. 2017), so it is important to know how much the auxiliary task is responsible for the reported results, and how much the proposed exploration bonus is. For this analysis I focus on Montezuma's Revenge because it is the game where the problem of exploration is maximized, with most algorithms not being able to do anything without an exploration bonus.

The first question to ask is whether the *auxiliary task is necessary* to $\text{DQN}_e^{\text{MMC}}$+SR. I evaluated this by dropping the reconstruction module from the network to test whether the initial random noise generated by the successor representation is enough to drive representation learning. It is not. When dropping the auxiliary task, the average performance of this baseline over 4 seeds in Montezuma's Revenge after 100 million frames was 100.0 points ($\sigma^2 = 200.0$; min: 0.0, max: 400.0). As comparison, $\text{DQN}_e^{\text{MMC}}$+SR obtains

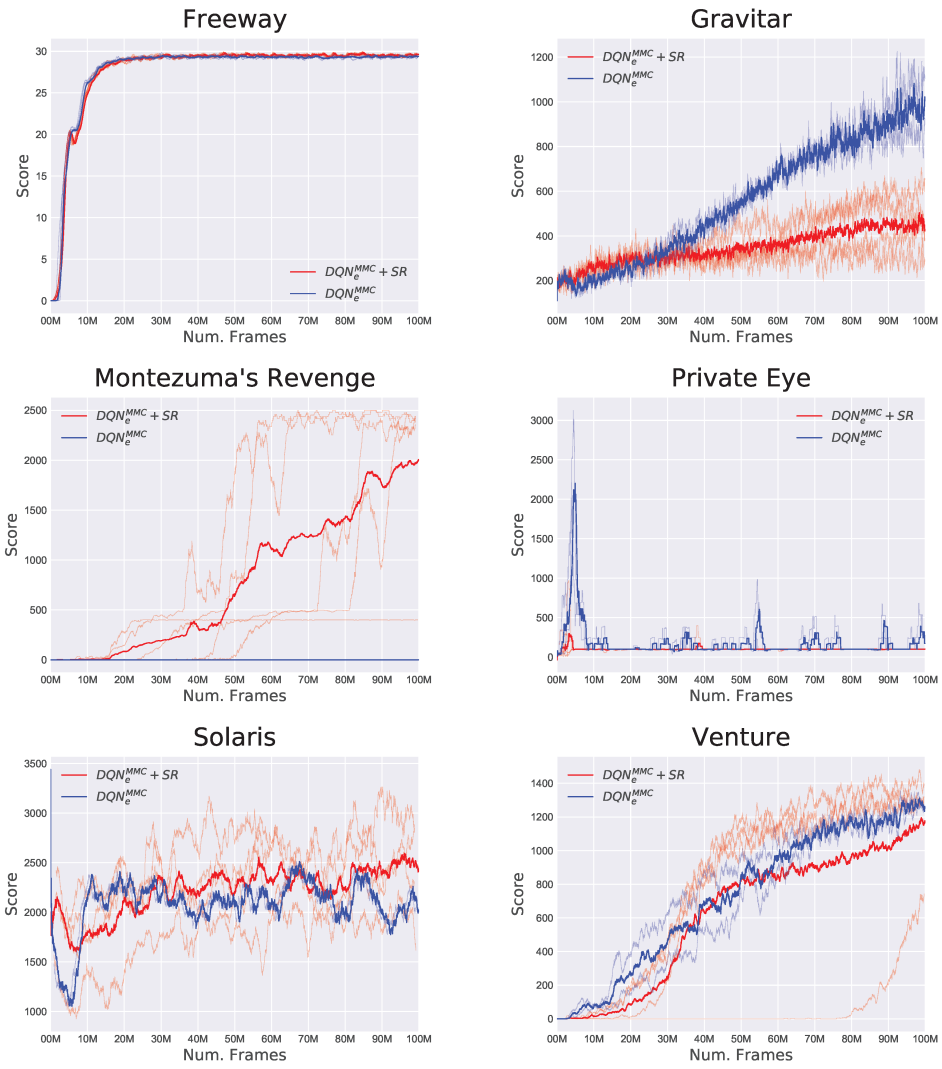


Figure 6.3: $\text{DQN}_e^{\text{MMC}}$+SR and $\text{DQN}_e^{\text{MMC}}$ learning curves in the Atari 2600 games used as testbed. The curves are smoothed with a running average computed using a window of size 100.

1778.6 points ($\sigma^2 = 903.6$, min: 400.0, max: 2500.0). These results suggest that auxiliary tasks seem to be necessary for $\text{DQN}_e^{\text{MMC}}$+SR to perform well.

I also evaluated whether the *auxiliary task was sufficient* to generate the reported results. To do so I dropped the SR module and set $\beta = 0.0$ to evaluate whether the proposed exploration bonus was actually improving the agent's performance or whether the auxiliary task was doing it. The results suggest that the exploration bonus is essential in $\text{DQN}_e^{\text{MMC}}$+SR. When dropping the exploration bonus and the successor representation module, the average per-

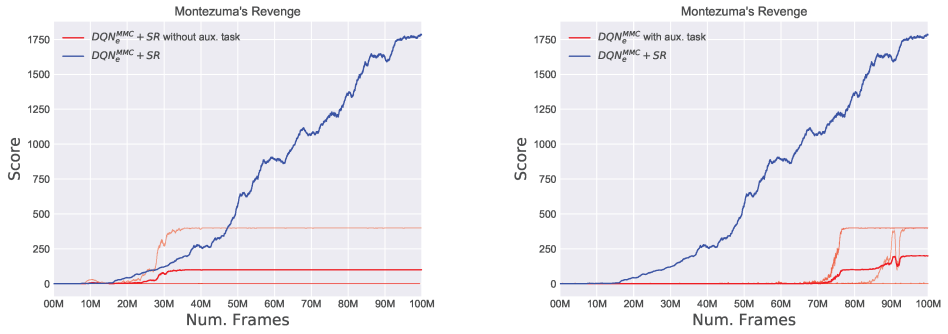


Figure 6.4: Evaluation of the sufficiency and necessity of the auxiliary task in DQN$_e^{\text{MMC}}$+SR. The learning curves are smoothed with a running average computed using a window of size 100.

formance of this baseline over 4 seeds in Montezuma's Revenge after 100 million frames was 398.5 points ($\sigma^2 = 230.1$; min: 0.0, max: 400.0). Again, clearly, the auxiliary task is not a sufficient feature for the performance I report. The reported results use the same parameters as those reported so far. Learning curves for each individual run are depicted in Figure 6.4.

6.3.3 On the Mismatch between the Used Norms

As aforementioned, there is a mismatch between theory and practice when looking at the deep RL algorithm I introduced in Section 6.2. While DQN$_e^{\text{MMC}}$+SR uses the ℓ_2-norm of the SR to generate the exploration bonus, my theoretical result is stated with respect to the ℓ_1-norm of the SR. This mismatch was driven by the fact that, empirically, DQN$_e^{\text{MMC}}$+SR exhibits a slightly better performance when using the ℓ_2-norm of the SR instead of the ℓ_1-norm. I conjecture this might be due to the fact that the ℓ_2-norm is smoother than the ℓ_1-norm, a property that is particularly important when training neural networks. Table 6.6 depicts a comparison between the performance of DQN$_e^{\text{MMC}}$+SR when using the ℓ_2-norm and the ℓ_1-norm of the SR to generate its exploration bonus ($\boldsymbol{\phi}(\cdot)$ is normalized with the respective norm).

I followed the same evaluation protocol described before, averaging the performance of DQN$_e^{\text{MMC}}$+SR with the ℓ_1-norm over 10 runs. The parameter β is the only parameter not shared by both algorithms. While $\beta = 0.025$

Table 6.6: Performance of the proposed algorithm, $\text{DQN}_e^{\text{MMC}}$+SR, when using the ℓ_1-norm and ℓ_2-norm of the SR to generate the exploration bonus. Standard deviations are reported between parenthesis.

	ℓ_1-norm		ℓ_2-norm	
Freeway	29.4	(0.1)	29.5	(0.1)
Gravitar	457.4	(120.3)	430.3	(109.4)
Mont. Rev.	1395.4	(1121.8)	1778.6	(903.6)
Private Eye	104.4	(50.4)	99.1	(1.8)
Solaris	1890.1	(163.1)	2155.7	(398.3)
Venture	1348.5	(56.5)	1241.8	(236.0)

Table 6.7: Performance of Sarsa+SR, in the tabular case, when using the ℓ_1-norm and ℓ_2-norm of the SR to generate the exploration bonus. A 95% confidence interval is reported between parentheses.

	ℓ_1-norm		ℓ_2-norm	
RiverSwim	1,792,126	(258,709)	1,989,479	(167,189)
SixArms	2,176,044	(449,962)	2,625,132	(516,804)

when using the ℓ_2-norm of the SR, $\beta = 0.05$ when using the ℓ_1-norm of the SR. These results also support the claim that the norm of the SR can be used to generate exploration bonuses. $\text{DQN}_e^{\text{MMC}}$+SR, when using the ℓ_1-norm of the SR, exhibits performance comparable to pseudo-count based methods, despite not being the best results I obtained.

I also revisit the results presented in Section 6.1.1 to evaluate the impact of the different norms in Sarsa+SR. I swept over all the parameters, as previously described. The results reported for Sarsa+SR when using the ℓ_2-norm of the SR are the average over 100 runs. The actual numbers are available in Table 6.7. Interestingly, we observe the same trend we observed in the deep RL case. The ℓ_2-norm of the SR leads to even better results. The reported performance of Sarsa+SR was obtained with $\alpha = 0.1$, $\gamma_{SR} = 0.5$, $\beta = 10000$ in both settings, with $\eta = 0.5$ and $\epsilon = 0.05$ in RiverSwim and $\eta = 0.5$ and $\epsilon = 0.01$ in SixArms.

The fact that the algorithms proposed in this chapter perform better when using the ℓ_2-norm of the SR instead of the ℓ_1-norm deserves further investigation, either empirically or theoretically. I conjecture it might be possible to derive theoretical guarantees for the ℓ_2-norm of the SR that are similar to

Table 6.8: Results obtained with $\text{DQN}_e^{\text{MMC}}$+SR after different amounts of experience when using the ℓ_1-norm of the SR.

Game	10M frames		50M frames		100M frames	
Freeway	23.5	(0.8)	29.3	(0.1)	29.4	(0.1)
Gravitar	238.8	(26.4)	343.0	(72.2)	457.4	(120.3)
Mont. Revenge	0.2	(0.4)	591.3	(870.1)	1395.4	(1121.8)
Private Eye	97.2	(5.3)	99.4	(1.9)	104.4	(50.4)
Solaris	1455.7	(146.3)	1755.9	(237.5)	1890.1	(163.1)
Venture	29.0	(30.2)	1102.4	(77.6)	1348.5	(56.5)

Table 6.9: Results obtained with $\text{DQN}_e^{\text{MMC}}$ after different amounts of experience when using the ℓ_1-norm in the normalization.

Game	10M frames		50M frames		100M frames	
Freeway	24.0	(8.4)	26.4	(9.3)	26.4	(9.3)
Gravitar	228.6	(26.9)	555.9	(35.6)	1063.1	(271.8)
Mont. Revenge	0.1	(0.3)	0.6	(1.9)	40.0	(126.5)
Private Eye	97.3	(5.1)	102.8	(13.2)	98.7	(3.2)
Solaris	1873.6	(210.6)	2175.2	(243.8)	2028.7	(143.2)
Venture	48.1	(44.9)	795.3	(167.2)	1236.0	(51.3)

those derived here. Nevertheless, these results suggest that the idea of using the norm of the SR for exploration is quite general, with the p-norm of the SR being effective for more than one value of p.

The performance of $\text{DQN}_e^{\text{MMC}}$+SR and DQN_e after different amounts of experience, when using the ℓ_1-norm of the SR, is available in Tables 6.8 and 6.9; and their learning curves are depicted in Figure 6.5.

6.4 Discussion

In this chapter I introduced a general method for exploration in RL that implicitly counts state (or feature) visitation in order to guide the exploration process. It is still guided by the successor representation but it does not require dozens or hundreds of options to be learned simultaneously before an effective exploration strategy can arise. It is compatible with representation learning and the idea can also be adapted to be applied to large domains. The results reported in this chapter are competitive to state-of-the-art results in Atari

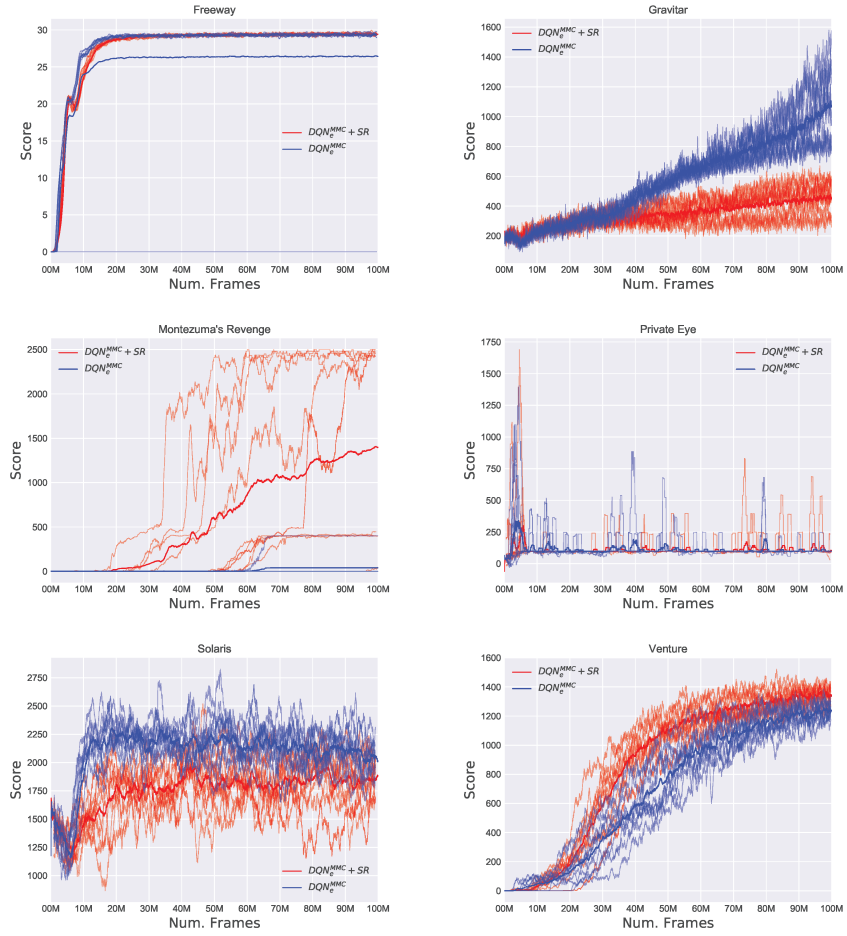


Figure 6.5: Learning curves for $\text{DQN}_e^{\text{MMC}}$+SR and $\text{DQN}_e^{\text{MMC}}$, when using the ℓ_1-norm of the SR, in the Atari 2600 games used as testbed. The curves are smoothed with a running average computed using a window of size 100.

2600 games and the use of the SR to guide exploration through exploration bonuses might be a fruitful research path in the years to come.

Importantly, $\text{DQN}_e^{\text{MMC}}$+SR can also be seen as connecting different approaches for exploration in RL. Martin et al. (2017) showed, for example, that counting activations of fixed, handcrafted features in Atari 2600 games leads to good exploration behavior. In this chapter I have shown that by using the SR we are not only counting *learned* features but we are also implicitly capturing the induced transition dynamics. On the other hand, Bellemare, Srinivasan, et al. (2016) and Ostrovski et al. (2017) have extended the idea of providing exploration bonuses based on counts to the function approximation

setting through a density model, which is often domain-specific and difficult to implement. The work presented in this chapter is obviously related to this idea, and maybe it can be seen as showing how the successor representation can instead be used to approximate state visitation counts.

Chapter 7

Bootstrapping in Option-Based Exploration and Other Developments

In this chapter I discuss some extensions of the ideas I presented in the previous three chapters. In Section 7.1 I discuss some preliminary work I developed to assess the impact of composing eigenoptions. This is an instantiation of multiple iterations of the option discovery cycle proposed in Chapter 3. In Section 7.2 I discuss some recent developments, by different research groups, that proposed ways to compute the eigenvectors of the Laplacian (or of the successor representation) online. These results are directly connected to what I have proposed in this dissertation because they can potentially pave the way to online algorithms for eigenoption discovery.

7.1 Bootstrapping Eigenoption Discovery

In Chapters 4 and 5 I described two instantiations of the option discovery cycle proposed in Chapter 3 and I presented results of a single iteration of the process. In theory, this process could have been repeated, with the discovered eigenoptions being used by the agent to collect data in the second iteration of the option discovery process. The main challenge in doing this is the high sample complexity of current reinforcement learning algorithms and the fact that off-policy learning methods are known to be unstable in domains where

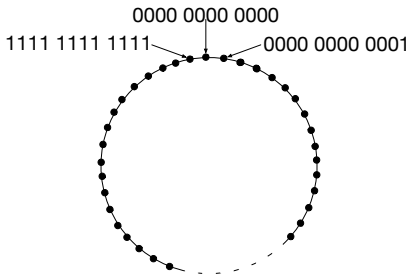


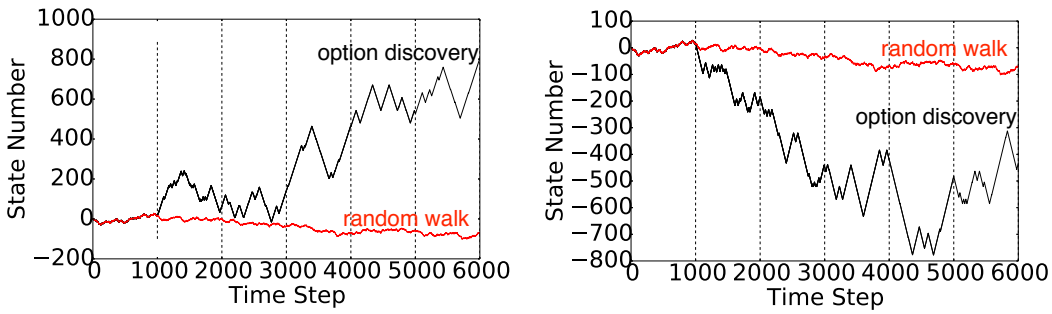
Figure 7.1: Environment used to evaluate the effects of option composition.

function approximation is required (Machado, Bellemare, Talvitie, et al. 2018). Consequently, with modest computational resources, it is unfeasible to learn the multiple eigenbehaviors generated by the proposed approaches in large domains such as Atari 2600 games.

In this section I present some preliminary results I obtained when exploring the effect of executing multiple iterations of the option discovery cycle in a small toy-domain. In this domain the value function is computed through linear function approximation. I used the algorithm described in Section 4.3. Shortly, the agent acts randomly with respect to primitive actions and options (after the first iteration) while collecting samples. These samples are stacked in a transition matrix such that row t encodes $\boldsymbol{\phi}(S_t) - \boldsymbol{\phi}(S_{t-1})$. After a given number of steps I compute the SVD of this transition matrix and use its right-singular vectors to generate the eingepurposes that will then be maximized. I test the following hypotheses with this algorithm:

- At each new iteration the discovered eigenoptions become increasingly more complex.
- More complex options move the agent farther away in the state space.
- As the agent moves farther away with newly discovered options, it observes new features, discovering different options, what creates a self-reinforcing loop.

The agent selected actions (and options) uniformly random. The domain I used was a ring of length 4096 with deterministic transitions in which the agent starts at the x coordinate 0 and at every time step it chooses between



(a) Full observability (b) Partial observability

Figure 7.2: Sample random walk using primitive actions and a random walk using the discovered options. Dashed vertical lines represent iteration boundaries.

going right or going left (see Figure 7.1). Each state is represented as a vector containing the two's complement binary encoding (12 bits long). When the agent goes left on the state 0 it goes to the state -1 ($0000\ 0000\ 0000_2 \rightarrow 1111\ 1111\ 1111_2$). Similarly, going right in state 2047 transitions to -2048. The agent observes a reward signal of value zero in every state.

All the evaluations were made in the same setting, with a discount factor $\gamma = 0.99$ when learning the eigenbehaviors. The policies were obtained through value iteration with 100 iterations. Each round of the algorithm consisted of 1,000 time steps in which the agent collected transitions to discover eigenpurposes. I ran six rounds, with only primitive actions available in round zero. All discovered options become available for the agent in a subsequent iteration.

I first evaluated this algorithm in settings with full observability, in which the agent perceives states as described above (Figure 7.2a, and Table 1). By looking at the average length of the discovered options we see that options become increasingly complex with each iteration. This added complexity allows the agent to move to farther states, as evidenced by the increasing distance between farthest point and the agent's starting state at that iteration (Max Dist. from Start). The improvement is particularly clear when comparing to a sample random walk on primitive actions (Figure 7.2a). Note that, despite options constructed in later iterations still use only primitive actions,

Table 7.1: Properties of discovered options, per iteration, when compared to a random walk in a ring. Each number is the average of 30 runs and standard deviations are reported between parentheses.

Observ.	Metric	Iter. 0		Iter. 1		Iter. 2		Iter. 3		Iter. 4		Iter. 5	
Full	Num. Options Discov.	-	(-)	5.9	(5.5)	7.7	(10.3)	8.5	(8.8)	9.2	(12.6)	9.5	(11.4)
	Avg. Opt. Length	-	(-)	12.1	(1.1)	19.2	(2.1)	21.6	(2.0)	25.5	(2.0)	27.8	(1.7)
	Max Dist. from Start	29.3	(18.2)	168.7	(222.5)	240.1	(287.3)	269.9	(311.5)	287.1	(436.9)	298.9	(320.8)
Partial	Num. Options Discov.	-	(-)	3.5	(20.9)	5.2	(14.1)	6.2	(19.4)	6.6	(30.2)	6.8	(21.6)
	Avg. Opt. Length	-	(-)	20.4	(1.8)	30.5	(1.4)	33.5	(2.1)	35.9	(1.7)	37.7	(1.7)
	Max Dist. from Start	29.3	(18.2)	212.8	(326.9)	314.9	(314.3)	301.8	(434.3)	352.4	(464.1)	301.1	(360.0)

they present more complex behaviors. This is different than typical option discovery methods, which construct hierarchies of options.

I also evaluated a setting in which the agent had partial observability. In this setting the agent does not observe the three least significant bits encoding the state. This collapses several states together and makes it much harder for the agent to observe progress. Interestingly, the same behavioral pattern as in the full observability experiment emerges. Agents still come up with options of the type "flip the i-th bit" once they discover these purposes. The only difference is that fewer options are discovered at each iteration, due to fewer observable eigenpurposes.

We can conclude that these results confirm the three aforementioned hypotheses. It is important to acknowledge that the described domain was designed to highlight the benefits of the proposed approach and that these results are fairly preliminary. That being said, these results suggest that the option discovery cycle proposed in Chapter 3 might scale well with computation.

7.2 Online Computation of the Eigenvectors of the Laplacian (or the Successor Representation)

One of the main issues I did not discuss in this dissertation is the fact that the computational cost of the eigendecompostion/SVD required by some of my algorithms is fairly high. In discrete settings, when all the data can fit in memory, the cost of an eigendecomposition is $\mathcal{O}(n^3)$. In continuous settings, the eigenfunctions need to be approximated from a fixed number of points while their value at other points is computed through the Nyström approximation (Liu et al. 2017; Mahadevan and Maggioni 2007), an operation that might also be extremely expensive depending on the size of the dataset.

While I do not adress this issue in my work, Pfau et al. (2018) and Wu, Tucker, and Nachum (2018) recently proposed approaches for estimating the eigenvectors of the graph Laplacian online. These methods approximate the

eigenfunctions of the graph Laplacian on high-dimensional function spaces via stochastic optimization. Pfau et al. (2018) do so by posing the problem of eigenfunction computation as an optimization problem and by deriving a gradient estimate which optimizes the sequential eigenfunction problem, allowing them to use neural networks to solve large-scale spectral decompositions. Wu, Tucker, and Nachum (2018) formalize the Laplacian in a more general way and, inspired by spectral graph drawing theory (Koren 2003), compute the eigenfunctions online by using an objective function that is amenable to stochastic optimization. They enforce orthonormality of the eigenvectors by introducing a penalty term into the objective function. Differently from Pfau et al. (2018), which formulated the problem with an unconstrained optimization problem objective, Wu, Tucker, and Nachum (2018) did not. The authors also claim that different from Pfau et al.'s, their approach scales linearly in the number of embedding dimensions.

Importantly, these approaches seem to outperform the algorithms I introduce in this dissertation for the estimation of the eigenvectors of the graph Laplacian (or successor representation). When considering the architecture I introduced in Chapter 5, this might be due to the fact that none of the components of my architecture pushes the network to learn a representation with a particular structure to the latent space. If one transforms the learned feature representation $\boldsymbol{\phi}(\cdot)$ with an arbitrary invertible matrix $A\boldsymbol{\phi}(\cdot)$, the reconstruction module could be unaffected as one can turn $\zeta(\boldsymbol{\phi}(\cdot), a)$ into $\zeta(A^{-1}\boldsymbol{\phi}(\cdot), a)$. However, when computing $\boldsymbol{\psi}_{\pi,i}$, this same term A could be pulled out, giving us $A\boldsymbol{\psi}(\cdot)$. Importantly, the SVD of $A\Psi(\cdot)$ can be completely different than the SVD of $\Psi(\cdot)$, depending on A, which I do not specify. The formulations proposed by Pfau et al. (2018) and Wu, Tucker, and Nachum (2018) address this issue and remove the necessity of multiple losses, computing the eigenpurposes in a fully end-to-end fashion.

Unfortunately, neither Pfau et al. nor Wu, Tucker, and Nachum evaluated their approaches in the setting I propose here. Pfau et al. computed eigenpurposes in Atari 2600 games and showed that their algorithm, when looking at the top k eigenvectors of the singular value decomposition, often obtains

eigenpurposes that are more diverse and that encode more interesting features. Nevertheless, eigenoptions were never actually learned. Wu, Tucker, and Nachum (2018) on the other hand, despite comparing the accuracy of the eigenvectors estimated by my approach to theirs, focused on the more traditional take of applying Laplacians in reinforcement learning in order to generate a feature space that reflects more accurately the geometry of the environment dynamics. They validated such a hypothesis with a shaped reward signal in goal-achieving tasks.

In conclusion, recent results suggest that there are ways to make the singular value decomposition in the approaches I proposed cheaper via online estimation. These results could potentially let me avoid having to explicitly compute the singular value decomposition of the Laplacian or of the successor representation. By doing so I would then be able to generate a fully online algorithm for option discovery. Moreover, improved neural network architectures might actually allow us to concentrate in a lower number of eingepurposes, also reducing the number of eigenbehaviors that would have to be learned.

7.3 Discussion

In this chapter I discussed different potential improvements or extensions to the ideas presented in this dissertation for option discovery. I presented preliminary results of multiple iterations of the option discovery cycle proposed in Chapter 3, and I discussed how other research groups have started to exploit some of the ideas I present in this dissertation. More specifically, Pfau et al. (2018) and Wu, Tucker, and Nachum (2018) recently have proposed online ways of computing the eigenfunctions of the graph Laplacian, an approach that could be applied to some of the algorithms I propose. In the next chapter I lay out the main conclusions obtained in this work and I discuss some promising research directions, further discussing the impact of the ideas presented here.

Chapter 8

Discussions and Future Work

This dissertation set out to demonstrate that

> *time-based representations can be used to design domain-independent algorithms that efficiently explore complex, sparse reward environments.*

In this final chapter I summarize the contributions I presented to support this statement. I discuss some possible future research directions that the contributions in this dissertation raise and I present some potential new directions for the field based on my experience developing this work.

8.1 Summary of Contributions

I started this document claiming that despite the recent successes of reinforcement learning algorithms, exploration is still a major problem in the field. This problem is particularly important in environments where the agent observes the same reward signal in the large majority of states. In environments where specific and long sequences of actions are required before the agent observes a different reward signal, the dithering behavior generated by random walks is almost hopeless. Despite all of that, random walks and ϵ-greedy strategies are still, by far, the exploration strategies mostly commonly used by practitioners.

The work I described in this dissertation tackles the problem of exploration with a new approach. I advocate that agents should use their progress when learning representations to guide their exploratory behavior, which is

somehow reminiscent of the idea of using learning progress to guide exploration (Oudeyer, Kaplan, and Hafner 2007; Schmidhuber 2008). Agents should visit places that they know are reachable, but have only been rarely visited. This information can be obtained from the learned representations. In my thesis statement I specifically advocate for the use of time-based representations to guide exploration and in the rest of this document I discussed different ways one could do that.

I first proposed option-based exploration (Chapter 3), which is the idea of using options with the call-and-return model to explore the environment. If these options operate at different time scales, and they are available at different parts of the state space, they improve exploration even when selected in a uniformly random fashion. This is because agents empowered by these options exhibit a more purposeful behavior and they operate at different levels of abstraction. Finally, I advocate that a general way of obtaining these options is to learn how to attain individual features of the agent representation.

In Chapters 4 and 5 I proceeded to introduce different algorithms for option discovery that obtain options that allow agents to better explore the state space. These options were discovered from learned time-based representations, proto-value functions (PVFs) and the successor representation. These representations naturally capture different time scales, ensuring that if different options aim at attaining different features they would indeed operate at different time scales. I define their initiation set in such a way that these options are available in the large majority of the state space as well, satisfying the conditions aforementioned.

In Chapter 4 I showed how PVFs could be used to generate algorithms that instantiate this option discovery cycle. I also proposed an extension to the original work of PVFs in order to be able to apply the same idea to settings in which a set of handcrafted linear features is available. The contributions in this chapter can also be seen as connecting two otherwise disjoint areas of research: option discovery and representation learning in reinforcement learning.

In Chapter 5 I introduced new algorithms that relax some of the assumptions made in Chapter 4, namely symmetry and determinism of the environ-

ment. The algorithms introduced in this chapter are also more general because they are sample-based, being implemented with temporal-difference learning, and because they permit the agent to simultaneously learn a non-linear representation of the environment with a neural network. All the algorithms in this chapter use the successor representation and they were inspired by the equivalence I proved between PVFs and the eigenvectors of the successor representation when it is defined with respect to a uniform random policy in a deterministic and symmetric environment.

While I do believe option-based exploration is a promising research direction, as I discuss in the next section, the computational cost of learning the policies of multiple options is sometimes prohibitive. In order to provide additional evidence of the potential of using time-based representations for exploration, in Chapter 6 I showed how the norm of the successor representation, while it is being learned, can be actually used as an exploration bonus for count-based reinforcement learning algorithms. I instantiated this idea in the tabular case in both model-free and model-based settings, and I introduced a neural network architecture for the deep reinforcement learning setting that achieves results comparable to the state-of-the-art in Atari 2600 games that pose hard exploration problems.

Finally, in Chapter 7 I returned to the idea of option-based exploration and I discussed some results that have the potential to strengthen the framework I proposed. Specifically, I presented some preliminary work I developed on bootstrapping the option-discovery process by executing multiple iterations of the option discovery cycle and I discussed some recent work, from different research groups, that show how one could compute the eigenvectors of time-based representations online. These are exciting results that show how promising the ideas proposed in this dissertation are. These result might also pave the way towards more practical algorithms for option-discovery.

8.2 Discussion and Future Directions

This dissertation opens up new interesting research directions that could either complement the work presented here or that arose from insights I gained while developing this work. I discuss some of these directions in this section.

8.2.1 Exploration Guided by Representation Learning Algorithms

With the advent of reinforcement learning algorithms that use deep neural networks as function approximators, it is now common for reinforcement learning algorithms to learn a representation of the available observations while interacting with the world. However, researchers rarely focus on using the process of representation learning as a learning signal. This dissertation provided evidence that this is indeed a promising research direction. I believe that the process of learning representations is by itself informative and it could guide agents in environments with sparse feedback. In fact, other groups have recently started exploring this idea. Burda et al. (2019), for example, recently introduced an algorithm that achieves impressive performance in some Atari 2600 games by rewarding the agent when it visits states it is not so certain about, with this uncertainty being measured by predictions of the learned latent representation. In this dissertation I explored time-based representations due to their underlying structure, but the idea of using the representation learning process to guide exploration seems to be a much more general principle. It would be interesting to see this general principle being further explored.

8.2.2 Option-based Exploration

In reinforcement learning, options are often associated to predictions or to faster credit assignment (Sutton, Precup, and Singh 1999), not so much to exploration. Nevertheless, in this dissertation I have presented evidence suggesting that the decisiveness introduced by options can actually play an important role in exploration. Simultaneously to the the work developed here, Vezhnevets, Osindero, et al. (2017) also proposed a hierarchical reinforcement

learning algorithm that uses the representation learning process to guide the hierarchy discovery and, despite not being the main topic of study in their work, they were also able to show that the purposeful behavior introduced by the uncovered hierarchy improved the agent's ability to explore. Similarly, Warde-Farley et al. (2018) recently proposed an approach that does not take the environment's reward into consideration but it is capable of learning goal-conditioned policies. The proposed neural network can be seen as learning the definition of a goal in that setting, ignoring parts of the observation that are not under the agent's control. Importantly, their approach can also be seen as learning a higher-level policy that could potentially accelerate learning. I do believe this is a promising research direction that should get more attention in the future, both theoretically and empirically.

My experience developing this work gave me a different perspective about this problem as well. The concept of exploration with options often relies on the assumption that the cost of learning the options is not taken into consideration. This is not generally true in the tasks that are traditionally used in the field (e.g., Atari 2600 games, or MuJoCo problems). It seems to me that exploration with options will have a particularly important effect in more ambitious settings such as continual learning (Ring 1997), where the cost of learning options is amortized within all the tasks an agent is expected to do (Liu et al. 2017). In those settings, a promising research direction would be to have coevolving action and representation abstractions: higher levels of action abstraction should drive the agent to improve its representation of the world and once the agent has a better representation of the world, better action abstractions should become available.

8.2.3 Return Maximization with Eigenoptions in Large Environments

The algorithms I introduced in Chapters 4 and 5 obtained encouraging results but I was still not able to use them to maximize rewards in large domains such as Atari 2600 games. This was mainly due to two problems: 1) the eigenpurposes obtained in the approximated setting do not seem to be properly

sorted according to their eigenvalues, with several of them being seemingly equivalent; and 2) the computational cost of learning the policies of dozens of options is prohibitive for most modern deep reinforcement learning algorithms. If these issues were solved I believe one could have a complete system for maximizing rewards using eigenoptions.

Fortunately, as I discussed in the previous chapter, progress on both of these issues has been made. Pfau et al. (2018) and Wu, Tucker, and Nachum (2018) have introduced methods that seem to generate a better sorting of the eigenvectors/eigenvalues. Importantly, these approaches are online, which could also potentially address the limitation in my work of not having a fully online algorithm for option discovery. In the off-learning policy setting, recent developments have generated more stable off-policy learning algorithms (e.g., Espeholt et al. 2018), which give us hope that they could actually be deployed in the option learning phase. Combining all these recent developments with the ideas I present in this dissertation is definitely a promising direction.

8.2.4 Theoretical Understanding of Option-based Exploration and of the Successor Representation

There is a lot of room for improvement in our theoretical understanding of some of the ideas presented in this dissertation. Formally understanding the impact options have in exploration in reinforcement learning is one of them. Such an understanding would be beneficial and could directly impact how we design algorithms for option discovery in the future. I conjecture that the right set of options might end up reducing the mixing time in the graph underlying the environment, either by making it somehow closer to an expander graph (Chapter 21, Arora and Barak 2009) or by introducing a small-world phenomenon (Chaganty, Gaur, and Ravindran 2012). Understanding how the transient behavior of the successor representation evolves with updates is another potentially interesting avenue, which could allow us to avoid the formalism introduced with the substochastic successor representation. Moreover, I conjecture that the use of the (substochastic) successor representation can lead to algorithms with PAC-MDP bounds (Kakade 2003).

A very interesting research task I believe is essential for the future endevours of exploration in reinforcement learning is the development of a theoretical formalism that tackles exploration under a function approximation perspective, moving away from the tabular case. It is not realistic to expect algorithms to visit all possible states in the environment in order to obtain guaranteess about their performance. Given the proper assumptions about similarity of states, we should have results about exploration in the function approximation case, when generalization comes into play. The work developed by Jiang et al. (2017) is a very promising direction towards that path.

8.3 Conclusions

In this dissertation I introduced different approaches for exploration in reinforcement learning. These approaches use the process of learning time-based representations to guide exploration. This is done by either discovering options that provide decisiveness to agents, avoiding the low-level dithering of random walks, or by generating exploration bonuses that directly incentivize the agent to visit parts of the state space that have only been rarely visited. This is a powerful framework and I believe there is still a fruitful path for further exploring the ideas presented here.

In this work I focused on using representation learning to guide exploration because we are now at a time in which we have a solid understanding of exploration in the tabular case but we still struggle to translate this knowledge to more general settings. Expecting agents to visit every state in the environment is unrelastic. We need to explicitly incorporate generalization into our algorithms to be able to properly explore large environments. It is likely that even such an approach will not be sufficient towards a fully intelligent agent. Complementary approaches such as curriculum learning, reward shaping, and imitation learning are also likely to be required in this process. It is not my intention to claim that agents should be able to learn, tabula rasa, any task specified to them, but I believe that, in tasks with some regularities, we should aim at moving away from the low-level dithering random walks provide.

References

Arora, Sanjeev and Boaz Barak (2009). *Computational Complexity: A Modern Approach*. Cambridge, United Kingdom: Cambridge University Press.

Auer, Peter, Nicolò Cesa-Bianchi, and Paul Fischer (2002). "Finite-time Analysis of the Multiarmed Bandit Problem." In: *Machine Learning* 47.2-3, pp. 235–256.

Bacon, Pierre-Luc (2013). "On the Bottleneck Concept for Options Discovery: Theoretical Underpinnings and Extension in Continuous State Spaces." Master's thesis. McGill University.

Bacon, Pierre-Luc, Jean Harb, and Doina Precup (2017). "The Option-Critic Architecture." In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Banino, Andrea, Caswell Barry, Benigno Uria, Charles Blundell, Timothy Lillicrap, Piotr Mirowski, Alexander Pritzel, Martin J. Chadwick, Thomas Degris, Joseph Modayil, Greg Wayne, Hubert Soyer, Fabio Viola, Brian Zhang, Ross Goroshin, Neil Rabinowitz, Razvan Pascanu, Charlie Beattie, Stig Petersen, Amir Sadik, Stephen Gaffney, Helen King, Koray Kavukcuoglu, Demis Hassabis, Raia Hadsell, and Dharshan Kumaran (2018). "Vector-Based Navigation using Grid-Like Representations in Artificial Agents." In: *Nature* 557.7705, pp. 429–433.

Baranes, Adrien and Pierre-Yves Oudeyer (2013). "Active Learning of Inverse Models with Intrinsically Motivated Goal Exploration in Robots." In: *Robotics and Autonomous Systems* 61.1, pp. 49–73.

Barreto, André, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel J. Mankowitz, Augustin Zidek, and Rémi Munos (2018). "Transfer in Deep Reinforcement Learning Using Successor Features and Generalised Policy Improvement." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 510–519.

Barreto, André, Will Dabney, Rémi Munos, Jonathan Hunt, Tom Schaul, David Silver, and Hado van Hasselt (2017). "Successor Features for Transfer in Reinforcement Learning." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 4058–4068.

Bellemare, Marc G., Yavar Naddaf, Joel Veness, and Michael Bowling (2013). "The Arcade Learning Environment: An Evaluation Platform for General Agents." In: *Journal of Artificial Intelligence Research* 47, pp. 253–279.

Bellemare, Marc G., Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Rémi Munos (2016). "Unifying Count-Based Exploration and Intrinsic Motivation." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 1471–1479.

Bellemare, Marc G., Joel Veness, and Michael Bowling (2012). "Sketch-Based Linear Value Function Approximation." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 2222–2230.

Bellemare, Marc G., Joel Veness, and Erik Talvitie (2014). "Skip Context Tree Switching." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1458–1466.

Bellman, Richard E. (1957). *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Bellman, Richard E. (1978). *Artificial Intelligence: Can Computers Think?* Boyd & Fraser Publishing Company.

Bengio, Yoshua, Pascal Lamblin, Dan Popovici, and Hugo Larochelle (2006). "Greedy Layer-Wise Training of Deep Networks." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 153–160.

Brafman, Ronen I. and Moshe Tennenholtz (2002). "R-MAX - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning." In: *Journal of Machine Learning Research* 3, pp. 213–231.

Brunskill, Emma and Lihong Li (2014). "PAC-inspired Option Discovery in Lifelong Reinforcement Learning." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 316–324.

Burda, Yuri, Harrison Edwards, Amos Storkey, and Oleg Klimov (2019). "Exploration by Random Network Distillation." In: *Proceedings of the International Conference on Learning Representations (ICLR)*.

Chaganty, Arun Tejasvi, Prateek Gaur, and Balaraman Ravindran (2012). "Learning in a Small World." In: *Proceedings of the International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*, pp. 391–397.

Chiappa, Silvia, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed (2017). "Recurrent Environment Simulators." In: *Proceedings of the International Conference on Learning Representations (ICLR)*.

Coifman, R. R., S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker (2005). "Geometric Diffusions as a Tool for Harmonic Analysis and Structure Definition of Data: Diffusion Maps." In: *Proceedings of the National Academy of Sciences* 102.21, pp. 7426–7431.

Daniel, Christian, Herke van Hoof, Jan Peters, and Gerhard Neumann (2016). "Probabilistic Inference for Determining Options in Reinforcement Learning." In: *Machine Learning* 104.2, pp. 337–357.

Dayan, Peter (1993). "Improving Generalization for Temporal Difference Learning: The Successor Representation." In: *Neural Computation* 5.4, pp. 613–624.

Dayan, Peter and Geoffrey E. Hinton (1992). "Feudal Reinforcement Learning." In: *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 271–278.

Dietterich, Thomas G. (2000). "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition." In: *Journal of Artificial Intelligence Research (JAIR)* 13, pp. 227–303.

Espeholt, Lasse, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu (2018). "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1406–1415.

Eysenbach, Benjamin, Abhishek Gupta, Julian Ibarz, and Sergey Levine (2019). "Diversity is All You Need: Learning Skills without a Reward Function." In: *Proceedings of the International Conference on Learning Representations (ICLR)*.

Farahmand, Amir-Massoud, André Barreto, and Daniel Nikovski (2017). "Value-Aware Loss Function for Model-based Reinforcement Learning." In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1486–1494.

Fortunato, Meire, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Volodymyr Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg (2018). "Noisy Networks for Exploration." In: *Proceedings of the International Conference on Learning Representations (ICLR)*.

Fox, Roy, Sanjay Krishnan, Ion Stoica, and Ken Goldberg (2017). "Multi-Level Discovery of Deep Options." In: *CoRR* abs/1703.08294.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the Difficulty of Training Deep Feedforward Neural Networks." In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 249–256.

Gregor, Karol, Danilo Rezende, and Daan Wierstra (2016). "Variational Intrinsic Control." In: *CoRR* abs/1611.07507.

Hasselt, Hado van, Arthur Guez, and David Silver (2016). "Deep Reinforcement Learning with Double Q-Learning." In: *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pp. 2094–2100.

Hengst, Bernhard (2002). "Discovering Hierarchy in Reinforcement Learning with HEXQ." In: *Proceedings of the International Conference on Machine Learning (ICML)*.

Higgins, Irina, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner (2017). "β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework." In: *Proceedings of the International Conference on Learning Representations (ICLR)*.

Ipek, Engin, Onur Mutlu, José F. Martínez, and Rich Caruana (2008). "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach." In: *International Symposium on Computer Architecture (ISCA)*, pp. 39–50.

Jaderberg, Max, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu (2017). "Reinforcement Learning with Unsupervised Auxiliary Tasks." In: *Proceedings of the International Conference on Learning Representations (ICLR)*.

Jiang, Nan, Akshay Krishnamurthy, Alekh Agarwal, John Langford, and Robert E. Schapire (2017). "Contextual Decision Processes with low Bellman rank are PAC-Learnable." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1704–1713.

Jong, Nicholas K., Todd Hester, and Peter Stone (2008). "The Utility of Temporal Abstraction in Reinforcement Learning." In: *Proceedings of the International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*, pp. 299–306.

Kakade, Sham (2003). "On the Sample Complexity of Reinforcement Learning." Doctoral dissertation. Gatsby Computational Neuroscience Unit, University College London.

Kakade, Sham and John Langford (2002). "Approximately Optimal Approximate Reinforcement Learning." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 267–274.

Kearns, Michael J., Yishay Mansour, and Andrew Y. Ng (1999). "A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1324–1231.

Kearns, Michael J. and Satinder Singh (2002). "Near-Optimal Reinforcement Learning in Polynomial Time." In: *Machine Learning* 49.2-3, pp. 209–232.

Keramati, Ramtin, Jay Whang, Patrick Cho, and Emma Brunskill (2018). "Strategic Object Oriented Reinforcement Learning." In: *CoRR* abs/1806.00175.

Kompella, Varun Raj, Matthew D. Luciw, and Jürgen Schmidhuber (2011). "Incremental Slow Feature Analysis." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1354–1359.

Kompella, Varun Raj, Marijn F. Stollenga, Matthew D. Luciw, and Jürgen Schmidhuber (2017). "Continual Curiosity-driven Skill Acquisition from High-Dimensional Video Inputs for Humanoid Robots." In: *Artificial Intelligence* 247, pp. 313–335.

Kondor, Risi and John D. Lafferty (2002). "Diffusion Kernels on Graphs and Other Discrete Input Spaces." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 315–322.

Konidaris, George and Andrew G. Barto (2009). "Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining." In: *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 1015–1023.

Koren, Yehuda (2003). "On Spectral Graph Drawing." In: *Proceedings of the International Computing and Combinatorics Conference (COCOON)*, pp. 496–508.

Kulkarni, Tejas D., Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum (2016). "Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 3675–3683.

Kulkarni, Tejas D., Ardavan Saeedi, Simanta Gautam, and Samuel J. Gershman (2016). "Deep Successor Reinforcement Learning." In: *CoRR* abs/1606.02396.

Lakshminarayanan, Aravind, Ramnandan Krishnamurthy, Peeyush Kumar, and Balaraman Ravindran (2016). "Option Discovery in Hierarchical Reinforcement Learning using Spatio-Temporal Clustering." In: *CoRR* abs/1605.05359. Presented at the ICML-16 Workshop on Abstraction in Reinforcement Learning.

Lehnert, Lucas and Michael L. Littman (2018). "Transfer with Model Features in Reinforcement Learning." In: *CoRR* abs/1807.01736.

Lehnert, Lucas, Stefanie Tellex, and Michael L. Littman (2017). "Advantages and Limitations of using Successor Features for Transfer in Reinforcement Learning." In: *CoRR* abs/1708.00102.

Levine, Sergey, Chelsea Finn, Trevor Darrell, and Pieter Abbeel (2016). "End-to-End Training of Deep Visuomotor Policies." In: *Journal of Machine Learning Research* 17, 39:1–39:40.

Liang, Yitao, Marlos C. Machado, Erik Talvitie, and Michael H. Bowling (2016). "State of the Art Control of Atari Games Using Shallow Reinforcement Learning." In: *Proceedings of the International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*, pp. 485–493.

Lin, Long-Ji (1993). *Reinforcement Learning for Robots Using Neural Networks*. Tech. rep. Carnegie Mellon University, School of Computer Science.

Liu, Miao, Marlos C. Machado, Gerald Tesauro, and Murray Campbell (2017). "The Eigenoption-Critic Framework." In: *NIPS-17 Workshop on Hierarchical Reinforcement Learning*.

Ma, Chen, Junfeng Wen, and Yoshua Bengio (2018). "Universal Successor Representations for Transfer Reinforcement Learning." In: *CoRR* abs/1804.03758.

Machado, Marlos C., Marc G. Bellemare, and Michael Bowling (2017). "A Laplacian Framework for Option Discovery in Reinforcement Learning." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2295–2304.

Machado, Marlos C., Marc G. Bellemare, and Michael Bowling (2018). "Count-Based Exploration with the Successor Representation." In: *CoRR* abs/1807.11622. Under review.

Machado, Marlos C., Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling (2018). "Revisiting the Arcade Learn-

ing Environment: Evaluation Protocols and Open Problems for General Agents." In: *Journal of Artificial Intelligence Research* 61, pp. 523–562.

Machado, Marlos C. and Michael Bowling (2016). "Learning Purposeful Behaviour in the Absence of Rewards." In: *CoRR* abs/1410.4604. Presented at the ICML-16 Workshop on Abstraction in Reinforcement Learning.

Machado, Marlos C., Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell (2018). "Eigenoption Discovery through the Deep Successor Representation." In: *Proceedings of the International Conference on Learning Representations (ICLR)*.

Machado, Marlos C., Sriram Srinivasan, and Michael Bowling (2015). "Domain-Independent Optimistic Initialization for Reinforcement Learning." In: *AAAI Workshop on Learning for General Competency in Video Games*.

Mahadevan, Sridhar (2005). "Proto-Value Functions: Developmental Reinforcement Learning." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 553–560.

Mahadevan, Sridhar and Mauro Maggioni (2007). "Proto-value Functions: A Laplacian Framework for Learning Representation and Control in Markov Decision Processes." In: *Journal of Machine Learning Research (JMLR)* 8, pp. 2169–2231.

Mankowitz, Daniel J., Timothy Arthur Mann, and Shie Mannor (2016). "Adaptive Skills Adaptive Partitions (ASAP)." In: *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 1588–1596.

Mannor, Shie, Ishai Menache, Amit Hoze, and Uri Klein (2004). "Dynamic Abstraction in Reinforcement Learning via Clustering." In: *Proceedings of the International Conference on Machine Learning (ICML)*.

Martin, Jarryd, Suraj Narayanan Sasikumar, Tom Everitt, and Marcus Hutter (2017). "Count-Based Exploration in Feature Space for Reinforcement Learning." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2471–2478.

Martínez, José F. and Engin Ipek (2009). "Dynamic Multicore Resource Management: A Machine Learning Approach." In: *IEEE Micro* 29.5, pp. 8–17.

McGovern, Amy and Andrew G. Barto (2001). "Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density." In: *Proceedings of the International Conference on Machine Learning (ICML)*.

Menache, Ishai, Shie Mannor, and Nahum Shimkin (2002). "Q-Cut - Dynamic Discovery of Sub-goals in Reinforcement Learning." In: *Proceedings of the European Conference on Machine Learning (ECML)*.

Meyn, Sean P and Richard L Tweedie (2012). *Markov Chains and Stochastic Stability*.

Mnih, Volodymyr, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016). "Asynchronous Methods for Deep Reinforcement Learning." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1928–1937.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis (2015). "Human-level Control through Deep Reinforcement Learning." In: *Nature* 518, pp. 529–533.

Montfort, Nick and Ian Bogost (2009). *Racing the Beam: The Atari Video Computer System*. MIT Press.

Moulin-Frier, Clément and Pierre-Yves Oudeyer (2013). "Exploration Strategies in Developmental Robotics: A Unified Probabilistic Framework." In: *Proceedings of the Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pp. 1–6.

Naddaf, Yavar (2010). "Game-independent AI Agents for Playing Atari 2600 Console Games." Master's thesis. University of Alberta.

Nair, Ashvin, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine (2018). "Visual Reinforcement Learning with Imagined Goals." In: *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 9209–9220.

Oh, Junhyuk, Xiaoxiao Guo, Honglak Lee, Richard L. Lewis, and Satinder Singh (2015). "Action-Conditional Video Prediction using Deep Networks in Atari Games." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 2863–2871.

Osband, Ian, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy (2016). "Deep Exploration via Bootstrapped DQN." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 4026–4034.

Osband, Ian, Benjamin Van Roy, and Zheng Wen (2016). "Generalization and Exploration via Randomized Value Functions." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2377–2386.

Ostrovski, Georg, Marc G. Bellemare, Aaron van den Oord, and Rémi Munos (2017). "Count-Based Exploration with Neural Density Models." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2721–2730.

Oudeyer, Pierre-Yves, Frédéric Kaplan, and Verena Vanessa Hafner (2007). "Intrinsic Motivation Systems for Autonomous Mental Development." In: *IEEE Transactions on Evolutionary Computation* 11.2, pp. 265–286.

Pfau, David, Stig Petersen, Ashish Agarwal, David Barrett, and Kim Stachenfeld (2018). "Spectral Inference Networks: Unifying Spectral Methods With Deep Learning." In: *CoRR* abs/1806.02215.

Piaget, Jean (1963). *The Origins of Intelligence in Children*. New York, NY: W. W. Norton & Company.

Pitis, Silviu (2018). "Source Traces for Temporal Difference Learning." In: *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pp. 3952–3959.

Plappert, Matthias, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychow-

icz (2018). "Parameter Space Noise for Exploration." In: *Proceedings of the International Conference on Learning Representations (ICLR)*.

Riedmiller, Martin A., Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg (2018). "Learning by Playing Solving Sparse Reward Tasks from Scratch." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 4341–4350.

Ring, M.B. (1997). "CHILD: A First Step Towards Continual Learning." In: *Machine Learning* 28.1, pp. 77–104.

Rummery, G. A. and M. Niranjan (1994). *On-line Q-Learning using Connectionist Systems*. CUED/F-INFENG/TR 166. Cambridge University Engineering Dept.

Salge, Christoph, Cornelius Glackin, and Daniel Polani (2014). "Empowerment – An Introduction." In: *Guided Self-Organization: Inception*. Springer, pp. 67–114.

Savinov, Nikolay, Anton Raichuk, Raphael Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly (2019). "Episodic Curiosity through Reachability." In: *Proceedings of the International Conference on Learning Representations (ICLR)*.

Schaul, Tom, Daniel Horgan, Karol Gregor, and David Silver (2015). "Universal Value Function Approximators." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1312–1320.

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver (2016). "Prioritized Experience Replay." In: *Proceedings of the International Conference on Learning Representations (ICLR)*.

Schmidhuber, Jürgen (2008). "Driven by Compression Progress: A Simple Principle Explains Essential Aspects of Subjective Beauty, Novelty, Surprise, Interestingness, Attention, Curiosity, Creativity, Art, Science, Music, Jokes." In: *Anticipatory Behavior in Adaptive Learning Systems, From Psychological Theories to Artificial Cognitive Systems [4th Workshop on Anticipatory Behavior in Adaptive Learning Systems (ABiALS)]*, pp. 48–76.

Seijen, Harm van, Ashique Rupam Mahmood, Patrick M. Pilarski, Marlos C. Machado, and Richard S. Sutton (2016). "True Online Temporal-Difference Learning." In: *Journal of Machine Learning Research* 17, 145:1–145:40.

Sherstan, Craig, Marlos C. Machado, and Patrick Pilarski (2018). "Accelerating Learning in Constructive Predictive Frameworks with the Successor Representation." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2997–3003.

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis (2016).

"Mastering the Game of Go with Deep Neural Networks and Tree Search." In: *Nature* 529.7587, pp. 484–489.

Şimşek, Özgür and Andrew G. Barto (2004). "Using Relative Novelty to Identify Useful Temporal Abstractions in Reinforcement Learning." In: *Proceedings of the International Conference on Machine Learning (ICML)*.

Şimşek, Özgür and Andrew G. Barto (2008). "Skill Characterization Based on Betweenness." In: *Proceedings of Advances in Neural Information Processing Systems (NIPS)*.

Şimşek, Özgür, Alicia P. Wolfe, and Andrew G. Barto (2005). "Identifying Useful Subgoals in Reinforcement Learning by Local Graph Partitioning." In: *Proceedings of the International Conference on Machine Learning (ICML)*.

Solway, Alec, Carlos Diuk, Natalia Córdova, Debbie Yee, Andrew G. Barto, Yael Niv, and Matthew M. Botvinick (2014). "Optimal Behavioral Hierarchy." In: *PLOS Computational Biology* 10.8, pp. 1–10.

Sprekeler, Henning (2011). "On the Relation of Slow Feature Analysis and Laplacian Eigenmaps." In: *Neural Computation* 23.12, pp. 3287–3302.

Stachenfeld, Kimberly, Matthew Botvinick, and Samuel Gershman (2014). "Design Principles of the Hippocampal Cognitive Map." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 2528–2536.

Stachenfeld, Kimberly, Matthew Botvinick, and Samuel Gershman (2017). "The Hippocampus as a Predictive Map." In: *Nature Neuroscience* 20, pp. 1643–1653.

Stadie, Bradly C., Sergey Levine, and Pieter Abbeel (2015). "Incentivizing Exploration in Reinforcement Learning With Deep Predictive Models." In: *CoRR* abs/1507.00814.

Stone, Peter, Richard S. Sutton, and Gregory Kuhlmann (2005). "Reinforcement Learning for RoboCup Soccer Keepaway." In: *Adaptive Behaviour* 13.3, pp. 165–188.

Strang, Gilbert (2005). *Linear Algebra and Its Applications*. Brooks Cole.

Strehl, Alexander L., Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman (2006). "PAC Model-Free Reinforcement Learning." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 881–888.

Strehl, Alexander L. and Michael L. Littman (2008). "An Analysis of Model-based Interval Estimation for Markov Decision Processes." In: *Journal of Computer and System Sciences* 74.8, pp. 1309–1331.

Sun, Wen, Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, and John Langford (2018). "Model-Based Reinforcement Learning in Contextual Decision Processes." In: *CoRR* abs/1811.08540.

Sutton, Richard S. (1988). "Learning to Predict by the Methods of Temporal Differences." In: *Machine Learning* 3, pp. 9–44.

Sutton, Richard S. (1990). "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 216–224.

Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, Richard S., Doina Precup, and Satinder Singh (1999). "Between MDPs and semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning." In: *Artificial Intelligence* 112.1–2, pp. 181–211.

Talvitie, Erik (2014). "Model Regularization for Stable Sample Rollouts." In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 780–789.

Talvitie, Erik (2017). "Self-Correcting Models for Model-Based Reinforcement Learning." In: *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pp. 2597–2603.

Tang, Haoran, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel (2016). "#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning." In: *CoRR* abs/1611.04717.

Tesauro, Gerald (1995). "Temporal Difference Learning and TD-Gammon." In: *Communications of the ACM* 38.3, pp. 58–68.

Theocharous, Georgios, Philip S. Thomas, and Mohammad Ghavamzadeh (2015). "Personalized Ad Recommendation Systems for Life-Time Value Optimization with Guarantees." In: *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1806–1812.

Valiant, Leslie G. (1984). "A Theory of the Learnable." In: *Communications of the ACM* 27.11, pp. 1134–1142.

Veness, Joel, Marc G. Bellemare, Marcus Hutter, Alvin Chua, and Guillaume Desjardins (2015). "Compress and Control." In: *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pp. 3016–3023.

Venkatraman, Arun, Martial Hebert, and J. Andrew Bagnell (2015). "Improving Multi-Step Prediction of Learned Time Series Models." In: *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pp. 3024–3030.

Vezhnevets, Alexander Sasha, Volodymyr Mnih, Simon Osindero, Alex Graves, Oriol Vinyals, John Agapiou, and Koray Kavukcuoglu (2016). "Strategic Attentive Writer for Learning Macro-Actions." In: *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 3486–3494.

Vezhnevets, Alexander Sasha, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu (2017). "FeUdal Networks for Hierarchical Reinforcement Learning." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 3540–3549.

Wang, Tao, Michael Bowling, and Dale Schuurmans (2007). "Dual Representations for Dynamic Programming and Reinforcement Learning." In: *Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 44–51.

Warde-Farley, David, Tom Van de Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Volodymyr Mnih (2018). "Unsupervised Control through Non-Parametric Discriminative Rewards." In: *CoRR* abs/1811.11359.

Watkins, Christopher J. C. H. and Peter Dayan (1992). "Technical Note: Q-Learning." In: *Machine Learning* 8.3-4.

Wiskott, Laurenz and Terrence J. Sejnowski (2002). "Slow Feature Analysis: Unsupervised Learning of Invariances." In: *Neural Computation* 14.4, pp. 715–770.

Wu, Yifan, George Tucker, and Ofir Nachum (2018). "The Laplacian in RL: Learning Representations with Efficient Approximations." In: *CoRR* abs/ 1810.04586.

Appendix A

Supporting Lemmas

In Chapter 4 I used two lemmas I did not discuss in that chapter. They are presented here, as well as their proofs.

Lemma A.1. *Suppose* $(I + A)$ *is a non-singular matrix, with* $||A|| \leq 1$. *We have:*

$$||(I + A)^{-1}|| \leq \frac{1}{1 - ||A||}.$$

Proof. [1]

$$\begin{aligned}
(I + A)(I + A)^{-1} &= I \\
I(I + A)^{-1} + A(I + A)^{-1} &= I \\
(I + A)^{-1} &= I - A(I + A)^{-1} \\
||(I + A)^{-1}|| &= ||I - A(I + A)^{-1}|| \\
&\leq ||I|| + ||A(I + A)^{-1}|| \\
&\leq 1 + ||A||||(I + A)^{-1}|| \\
||(I + A)^{-1}|| - ||A||||(I + A)^{-1}|| &\leq 1 \\
(1 - ||A||)||(I + A)^{-1}|| &\leq 1 \\
||(I + A)^{-1}|| &\leq \frac{1}{1 - ||A||} \qquad \text{if } \; ||A|| \leq 1.
\end{aligned}$$

Where the first inequality is due to the fact that $||A + B|| \leq ||A|| + ||B||$ and the second inequality comes from the fact that $||AB|| \leq ||A|| \cdot ||B||$. □

[1] This proof follows closely the proof of Parnell in lecture notes available at `http://www-solar.mcs.st-and.ac.uk/~clare/Lectures/num-analysis.html`.

Lemma A.2. *The induced infinity norm of* $(I - \gamma T)^{-1}T$ *is bounded by*

$$||(I - \gamma T)^{-1}T||_\infty \leq \frac{1}{(1-\gamma)}.$$

Proof.

$$||(I - \gamma T)^{-1}T||_\infty \leq ||(I - \gamma T)^{-1}||_\infty ||T||_\infty \quad \text{because} \quad ||AB||_\infty \leq ||A||_\infty \cdot ||B||_\infty$$

$$||(I - \gamma T)^{-1}T||_\infty \leq \frac{1}{1 - ||-\gamma T||_\infty}||T||_\infty \quad \text{Lemma 3.1}$$

$$||(I - \gamma T)^{-1}T||_\infty \leq \frac{1}{1 - \gamma||T||_\infty}||T||_\infty \quad \text{because} \quad ||\lambda B|| = |\lambda|||B||$$

$$||(I - \gamma T)^{-1}T||_\infty \leq \frac{1}{(1-\gamma)}$$

□

Lemma A.3. *In the tabular case, if all transitions in the MDP have been sampled once,* $T^\top T = 2L$.

Proof. Let t_{ij} and tt_{ij} denote the entries in the i-th row and j-th column of matrices T and $T^\top T$. We can write tt_{ij} as:

$$tt_{ij} = \sum_k t_{ik} \times t_{jk}. \tag{A.1}$$

In the tabular case, t_{ij} has three possible values:

- $t_{ij} = +1$, meaning that the agent arrived in state j at time step i,
- $t_{ij} = -1$, meaning that the agent left state j at time step i,
- $t_{ij} = 0$, meaning that the agent did not arrive nor leave state j at time step i.

We decompose $T^\top T$ in two matrices, K and Z, such that $T^\top T = K + Z$. Here Z is a diagonal matrix such that $z_{ii} = tt_{ii}$, for all i; and K contains all elements from $T^\top T$ that lie outside the main diagonal.

When computing the elements of Z we have $i = j$. Thus $z_{ii} = \sum_k t_{ik}^2$. Because we square all elements, we are in fact summing over all transitions leaving (-1^2) and arriving (1^2) in state i, counting the node's degree twice. Thus, $Z = 2D$.

When not computing the elements in the main diagonal, for the element tt_{ij}, we add all transitions that leave state i arriving in state j (-1×1), and those that leave state j arriving in state i (1×-1). We assume each transition has been sampled once, thus:

$$tt_{ij} = \begin{cases} -2, & \text{if the transition between states } i \text{ and } j \text{ exists,} \\ 0, & \text{otherwise.} \end{cases}$$

Therefore, we have $K = -2W$ and $T^\top T = K + Z = 2(D - W)$. □

Appendix B

Evaluating the Reconstruction Task when Learning the Representation and the SR

In Section 5.4.4 I presented the eigenoptions my algorithms discover in four Atari 2600 games. I did not discuss the performance of the proposed network in the auxiliary task of predicting the next screen given the current screen and the action taken. I do it here. Figures B.1–B.4 depict a comparison between the target screen that should be predicted and the network's actual prediction for ten time steps in each game. We can see that it accurately predicts the general structure of the environment and it is able to keep track of most moving sprites on the screen. The prediction is quite noisy, different from Oh et al.'s result. Still, it is interesting to see how even an underperforming network is able to learn useful representations for my algorithm. It is likely better representations would result in better options.

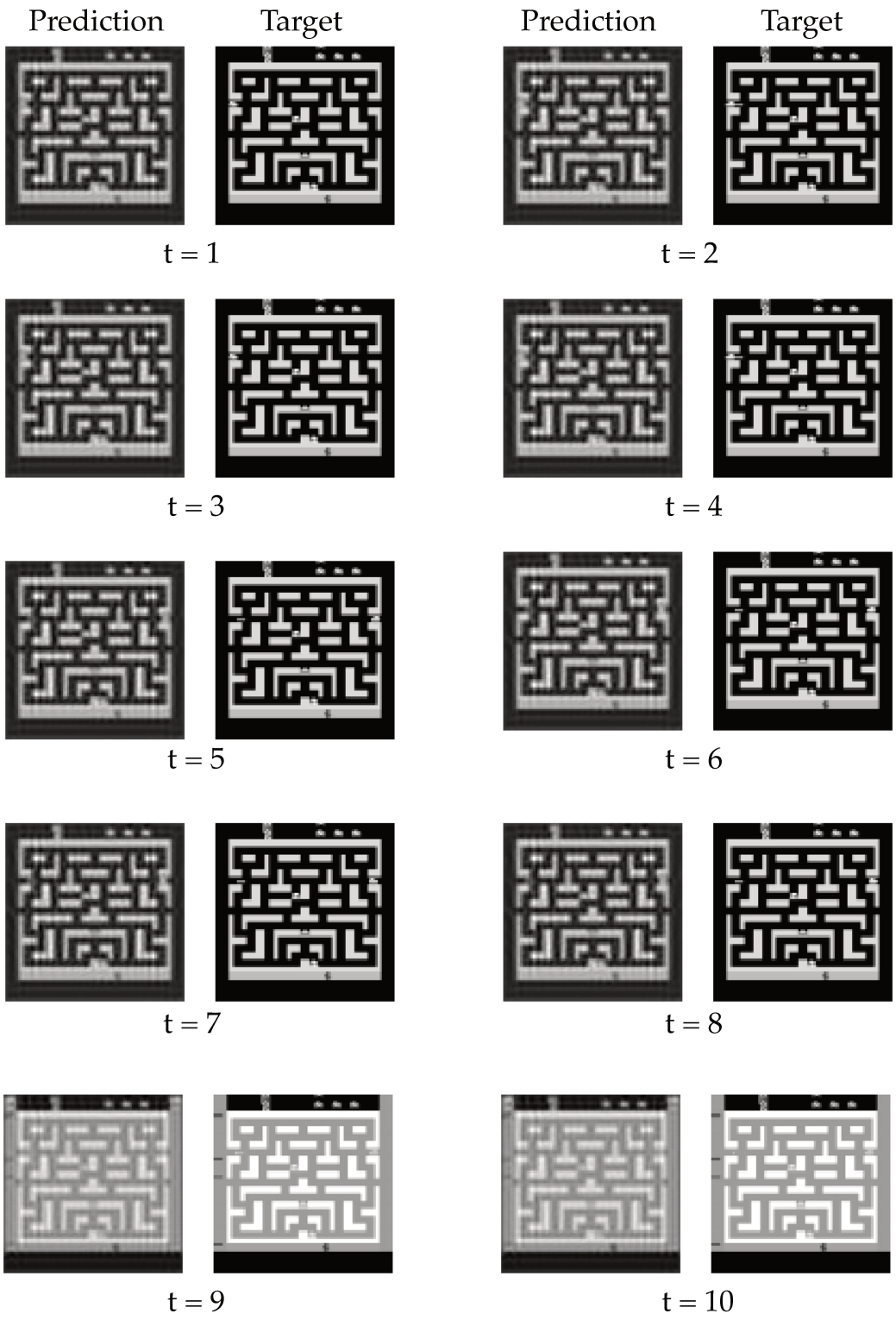


Figure B.1: Final 1-step predictions in the game Bank Heist.

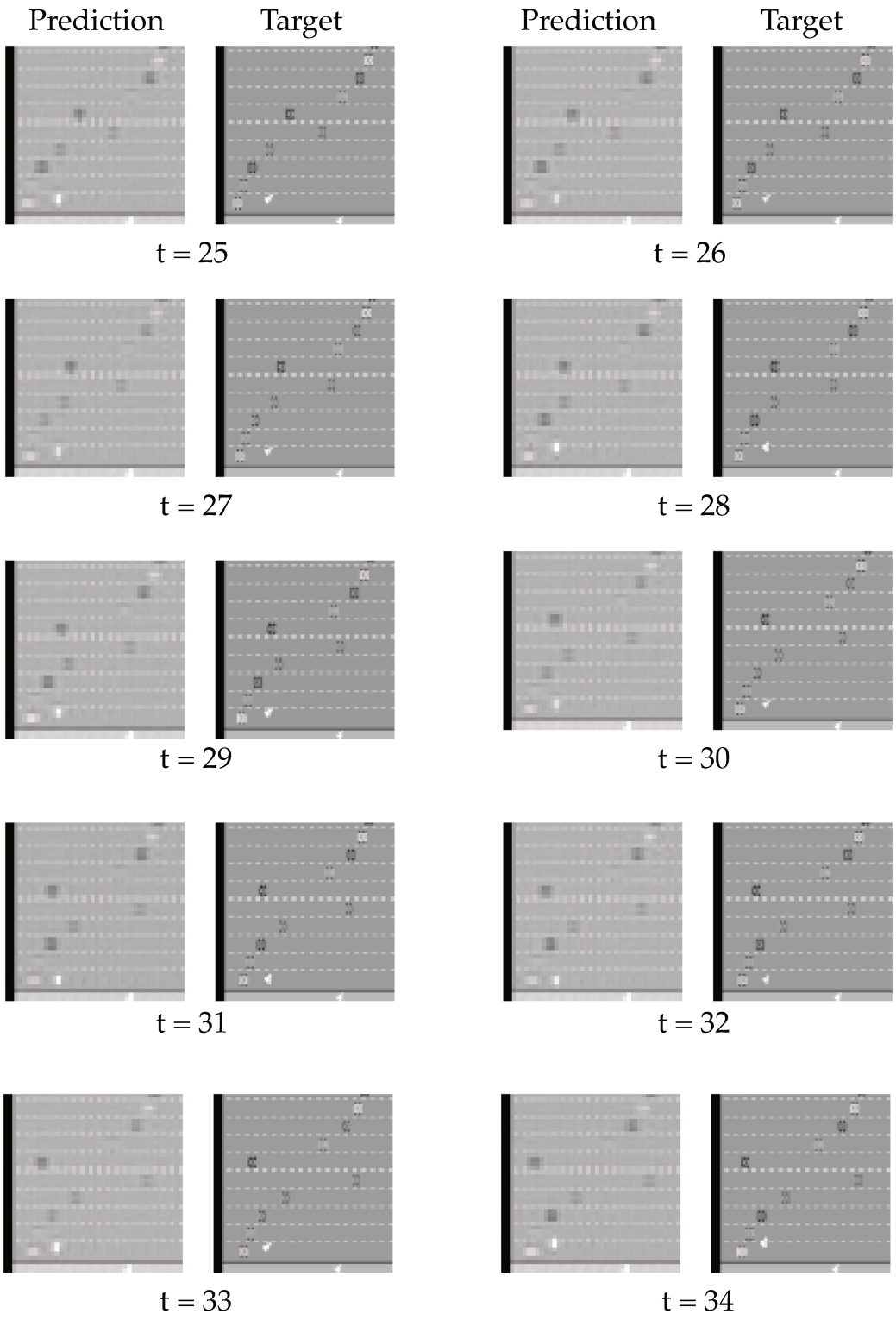


Figure B.2: Final 1-step predictions in the game FREEWAY.

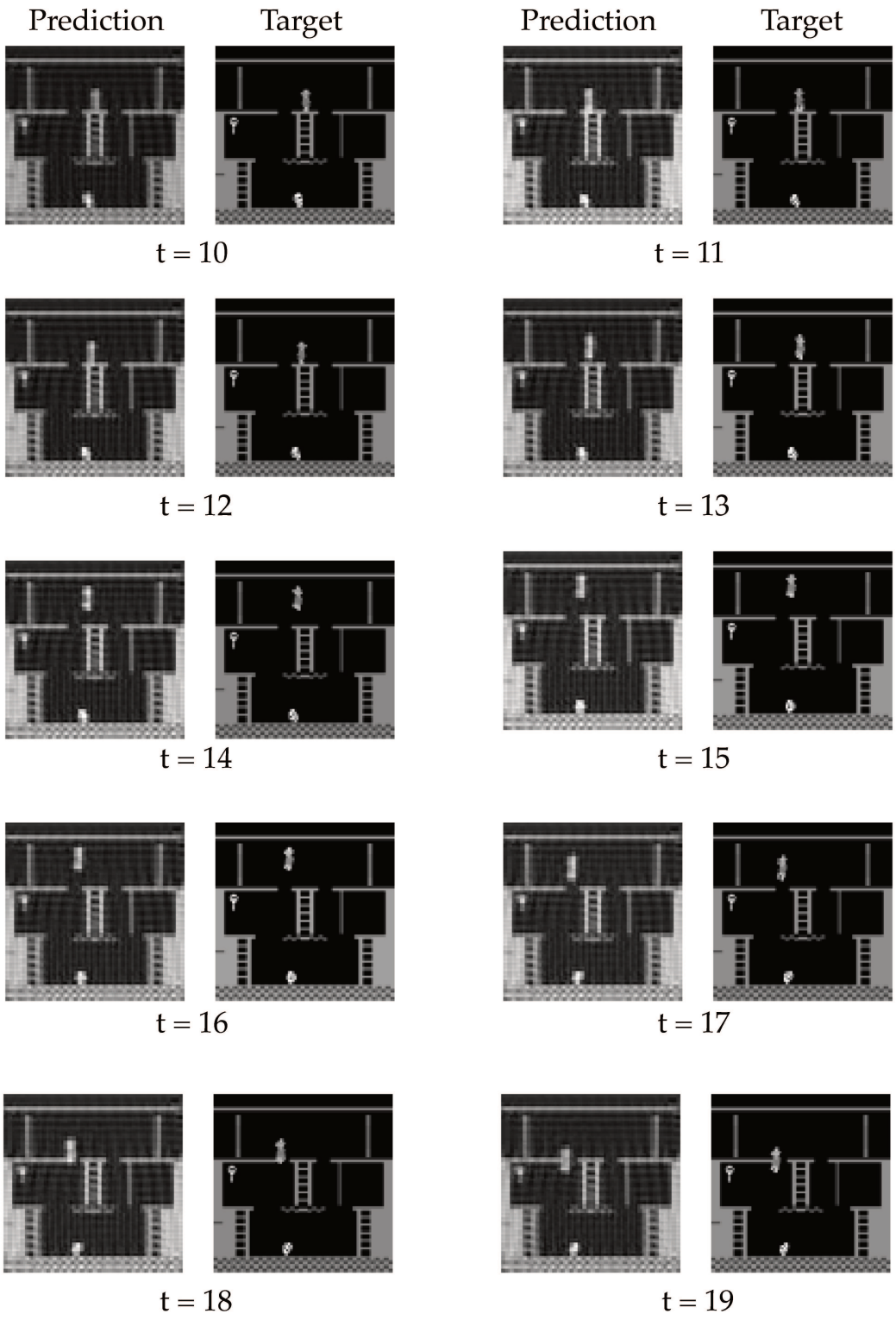


Figure B.3: Final 1-step predictions in the game Montezuma's Revenge.

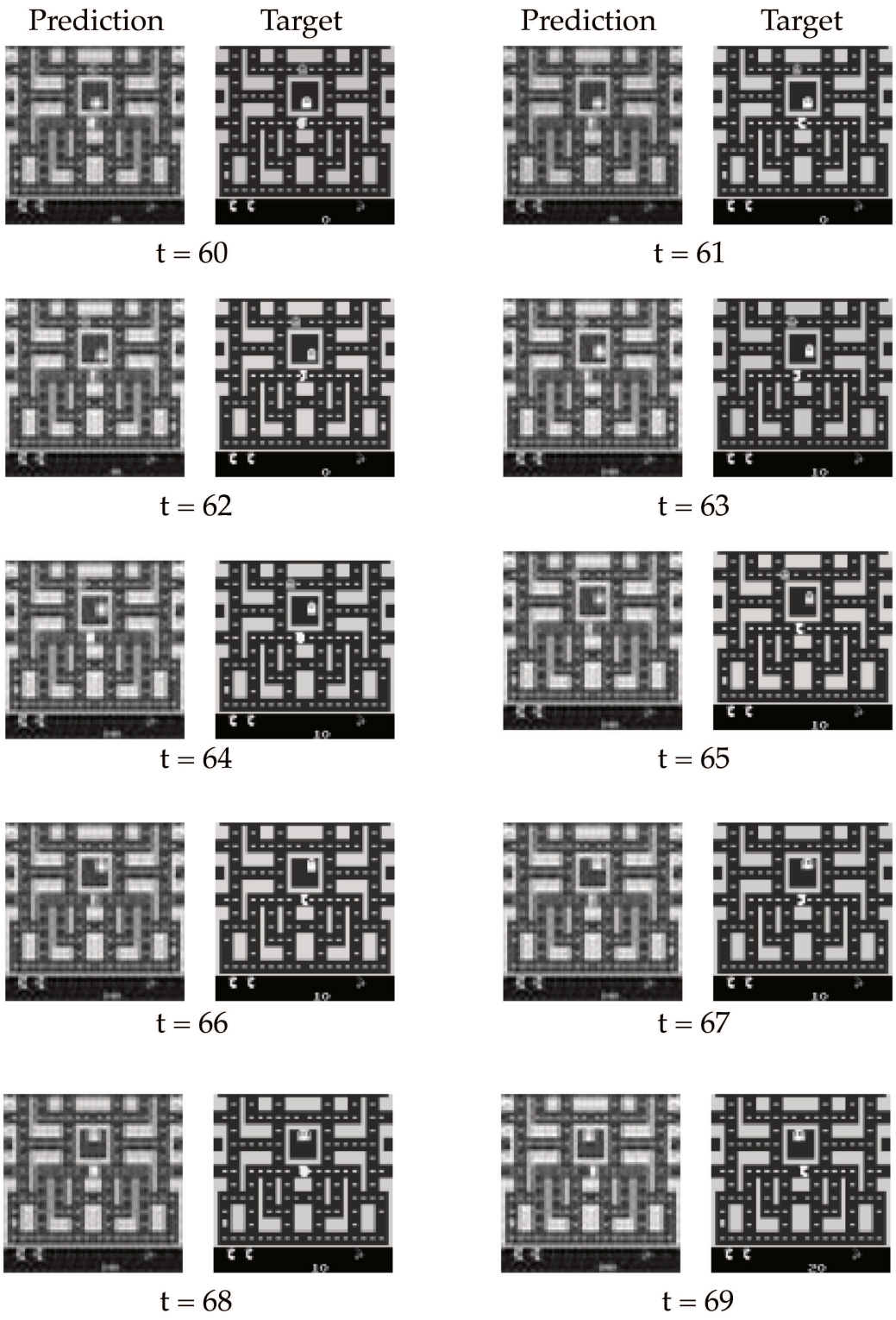


Figure B.4: Final 1-step predictions in the game Ms. Pacman.