# On Successor Representations for Value Learning:

**Efficient credit assignment through implicit models**

Anthony G.X. Chen

Computer Science
McGill University, Montreal

December 14, 2021

A thesis submitted to McGill University in partial fulfilment of the requirements of the degree of Master of Science. 

**Acknowledgements**

I would like to thank my advisors, Joelle Pineau and Blake Richards. Joelle, thank you for your wisdom, both in the technical subtleties of RL research, and in the way you provide support and guidance. Blake, you are a masterful example of how to bridge the fields of neuroscience and ML and I learn from you daily. To both, I am grateful for the freedom I had to pursue ideas in this amorphous intersection of neural-AI, as well as the always-insightful support I am able to receive in both RL and neuroscience.

I am grateful to my colleagues at Mila. Thank you for making me feel welcomed into a field that was new and intimidating, and for the always stimulating discussions that span from fundamental issues in RL algorithms, to interpreting experimental observations of specific brain regions. There were numerous people who helped in various ways: Veronica Chelu, Emmanuel Bengio, Nishanth Anand, Harsh Satija, Wesley Chung, Maxime Wabartha, Mandana Samiei, Colleen Gillon, Samuel Laferriere, Arna Ghosh, Raymond Chua, Surya Penmetsa, Chen Sun, Shahab Bakhtiari, Annik Carson, ... the list goes on. Thank you also to Pierre-Luc Bacon whose vast knowledge of RL and optimal control have both inspired and improved this thesis.

I am thankful for my friends. Your friendships have kept me sane throughout the long and at times discouraging months of research, the uncertainty about the future, and the isolation of the Covid pandemic that spanned the majority of my master's. To my family: Grace, Xenia, Kevin, Wendy, thank you for giving me a home away from home. To my parents: thank you for always supporting me, it means the world to me to know I can always turn to you in the moments when I feel the most lost.

## Abstract

Reinforcement learning is a well-established framework for sequential decision making. At its core is the *value function*—an estimate of total future reward—a quantity to be maximized.[1] A number of tools are utilized for value learning. The *lambda return* is a way of efficiently combining full-trajectory information to construct learning targets. The *successor representation* is a way of representing a decomposed value function in which dynamics are learned separately from reward information, then combined to calculate value. Typically, the lambda-return has been used for efficient learning, while the successor representation has been used for transferring to new tasks.

In this thesis, we revisit the above two core concepts. We explore the various forms of the lambda return, showing ways of decomposing out the dynamics information—in the form of successor representations / features—to be independently learned. This gives rise to a novel class of reinforcement learning algorithms which directly uses successor-like representation for efficient credit assignment. We introduce and investigate this class of algorithms, provide theoretical justifications, and empirically demonstrate how this approach results in efficient credit assignment. Finally we discuss recent neuroscience theories on the connections between successor representation and the brain, and how our new algorithm class extends theories and predictions for the neuroscience of reinforcement learning.


[1] Specifically, the value function is particularly important for the approximate dynamic programming formulation of reinforcement learning, such as temporal difference learning.

## Résumé

L'apprentissage par renforcement est un paradigme établi pour la prise de décision séquentielle. À sa base se trouve la *fonction de valeur*—une estimation de la récompense future totale—une quantité à maximiser. Un certain nombre d'outils sont utilisés pour l'apprentissage de la valeur. Le *lambda return (retour lambda)* est un moyen de combiner efficacement les informations d'une trajectoire pour construire des cibles d'apprentissage. La *successor representation (représentation de successeurs)* est un moyen de représenter une fonction de valeur factorisée dans laquelle les dynamiques de l'environnement sont apprises séparément des informations de récompense, puis combinées pour calculer la valeur. En règle générale, le retour lambda est utilisé pour un apprentissage efficace, tandis que la représentation de successeurs est utilisée pour le transfert vers de nouvelles tâches.

Dans cette thèse, nous revisitons ces deux concepts de base. Nous explorons les différentes formes du retour lambda, montrant des manières de factoriser les informations environnementales—sous la forme de *successor representation*—afin de les apprendre de manière indépendante. Cela donne naissance à une nouvelle classe d'algorithmes d'apprentissage par renforcement qui utilise directement une représentation de successeurs pour une attribution efficace des crédits. Nous présentons et étudions cette classe d'algorithmes, fournissons des justifications théoriques et démontrons empiriquement comment cette approche permet une attribution de crédit efficace. Enfin, nous discutons des théories neuroscientifiques récentes sur les liens entre la représentation de successeurs et le cerveau, et de comment notre nouvelle classe d'algorithmes étend les théories et les prédictions neuroscientifiques de l'apprentissage par renforcement.

# Contents

**Contents** **iv**

**1 Introduction** **1**

1.1 A Reward Maximizing Framework of Intelligence . . . . . 2

1.2 Objectives and Outline . . . . . 4

1.3 Summary of Contribution . . . . . 5

**2 Reinforcement Learning** **6**

2.1 Markov Decision Process . . . . . 7

2.2 Policy Evaluation . . . . . 9

2.2.1 Dynamic Programming . . . . . 11

2.2.2 Monte-Carlo Simulation . . . . . 14

2.2.3 Temporal Difference . . . . . 17

2.2.4 Lambda Return . . . . . 18

2.3 Policy Improvement . . . . . 22

2.3.1 Value Iteration . . . . . 23

2.3.2 Q-Learning . . . . . 25

**3 Representation Learning** **26**

3.1 Policy Evaluation with Function Approximation . . . . . 27

3.1.1 Feature Representation of States . . . . . 28
3.1.2 The Value Approximation Objective . . . . . 29
3.1.3 Linear Value Function . . . . . 31
3.1.4 Fixed Point of Linear TD(0) . . . . . 33
3.1.5 Nonlinear Value Function . . . . . 37
3.2 Policy Improvement with Function Approximation . . . . . 39
3.2.1 Fitted Q Iteration . . . . . 39
3.2.2 Nonlinear Control with Deep Q Network . . . . . 40
3.3 Successor Representation of States . . . . . 41
3.3.1 Learning tabular SR . . . . . 43
3.3.2 SR for Non-Tabular State Observations . . . . . 45
3.4 Successor Representation of Features . . . . . 46
3.4.1 General Value Function . . . . . 47
3.4.2 Successor Features . . . . . 48
3.5 SRs in the brain . . . . . 51

**4 Predictive Error Propagation 54**

4.1 The Lambda Return Error . . . . . 55
4.1.1 Lambda Successor Return Error ($\lambda$-SRE) . . . . . 56
4.1.2 The $\lambda$-SRE Fixed Point . . . . . 57
4.1.3 Tabular Algorithm . . . . . 59
4.1.4 Learning the $\lambda\gamma$-discounted SR . . . . . 60
4.2 Experiments . . . . . 61
4.2.1 Policy Evaluation . . . . . 62
4.2.2 Using Successor-like Representations . . . . . 64
4.3 Discussion on Machine Learning . . . . . 65
4.3.1 Successor Errors . . . . . 65
4.3.2 Related Works . . . . . 66

4.4 Discussion on Neuroscience . . . . . 66
4.5 Future Works . . . . . 67

**5 Lambda Value Function 69**

5.1 Lambda Value Function ($\lambda$-VF) . . . . . 69
5.1.1 Linearly Predictive Rewards and Features . . . . . 70
5.1.2 The $\lambda$-VF Decomposition . . . . . 72
5.1.3 Fixed Point of Linear $\lambda$-VF TD(0) . . . . . 75
5.1.4 Linear Algorithm for Prediction . . . . . 79
5.1.5 Nonlinear Control Algorithm . . . . . 81
5.2 Experiments . . . . . 81
5.2.1 Value Prediction in Deterministic Chain . . . . . 82
5.2.2 Prediction in Random Chain . . . . . 85
5.2.3 Nonlinear Control in Mini-Atari (MinAtar) . . . . . 87
5.2.4 Feature Representation Collapse . . . . . 91
5.3 Discussion on Machine Learning . . . . . 92
5.3.1 Related Works . . . . . 92
5.3.2 Discussion and Future Directions . . . . . 94
5.4 Discussion on Neuroscience . . . . . 95

**6 Discussion 96**

6.1 On Reinforcement Learning . . . . . 96
6.2 On Neuroscience . . . . . 98

**Bibliography 103**

# 1

# Introduction

It can be said that the goal of building artificial machines with human-like intelligence is the fundamental challenge for both the field of artificial intelligence (AI) and neuroscience.[1] For AI, the ability to create intelligence machines lead to powerful tools with an immense capacity to improve humanity. For neuroscience, the ability to reproduce an intelligent mind *in silico* implies attaining the highest form of scientific knowledge—in the words of the physicist Richard Feynman: "What I cannot create, I do not understand".

Since the advent of this endeavour (Turing 1950; McCarthy et al. 1955), the field of AI and neuroscience have played a complementary role in each other's mutual advancement. For example, the conception of the neuron as the basic structural and computational unit of the brain—the *neuron doctrine*, championed by Santiago Ramon y Cajal, Charles Sherrington and others around the turn of the 20th century (Yuste 2015)—formed the conceptual basis for the invention of the artificial neural network as early models of information processing in brain (Rosenblatt 1958; McClelland, Rumelhart, PDP Research Group, et al. 1986), which was subsequently developed into a powerful method for AI (for example, the impressive image classification performance of Krizhevsky, Sutskever, and G. E. Hinton 2012), then re-applied back to understand the function, dynamics and representation of neuron populations of the

[1] We use "neuroscience" to refer generally to any sciences concerned with the mind and brain: neuroscience, cognitive science, psychology, and more. We view them as analogous as they share similar goals, differing only in their approaches.

brain (Yamins et al. 2014; Richards et al. 2019). Similarly, the psychological observations of *conditioning* (Pavlov 1927) inspired the development of algorithms that perform reward-based learning (Rescorla and Wagner 1972; R. S. Sutton and Barto 1987; R. S. Sutton 1988), which is then used to explain the neuronal responses of biological system whose activities appeared to correspond to environmental rewards (Schultz, Dayan, and Montague 1997; Dabney et al. 2020). This cycle continues today and is ever fruitful: biological, cognitive and behavioural findings from the neural-sciences constrain and bias the search for models of intelligence in AI, while the development of new ideas and algorithms in AI help provide the theoretical framework to interpret empirical findings in the brain.

This thesis follows in the above scientific tradition of developing neuroscience-inspired model of artificial intelligence, with the long term goal of generating new theories back in the neural-sciences. In general, we believe that a fruitful line of research lies in designing models and algorithms that are *both* performant from an AI perspective, *and* simultaneously useful in providing better understanding of the brain.

## 1.1 A Reward Maximizing Framework of Intelligence

We use reinforcement learning (RL, see R. S. Sutton and Barto 1998) as a *framework* to understand intelligence. Under the RL framework, all learning and behaviour serve the singular goal of the maximization of reward. While this axiomatic principle may not be fully correct, we believe RL is sufficiently general to make important progress toward human-like intelligence.[2] Importantly, RL allows one to explicitly evaluate an embodied algorithm's ability to succeed in a given environment (through the total

[2] Although, for an argument in support of why reward maximization alone may indeed be sufficient in understanding intelligence, see Silver et al. (2021).

amount of reward it receives). We believe having a *performance-driven* objective is one crucial ingredient in building models of the mind (e.g. the models of R. S. Sutton and Barto 1987 and Yamins et al. 2014); after all, models that describe the mind should support behaviour similar to the mind: acting intelligently amidst complexity.

Concretely, our modelling work is inspired by a particular observation in the brain: neurons encoding for future predictive representations. Such neurons—place cells—are thought to encode an animal's current position and are located in a brain region known as the *hippocampus*. Existing neuroscience theory have argued that hippocampal place cells encode a particular mathematical object known as the "successor representation (SR)" (Stachenfeld, M. M. Botvinick, and Gershman 2017). While some physiological evidence support this observation, it is unclear from a performance-driven perspective *why* the brain would encode this. Under the assumption that the brain should only encode something if it helps in the maximization of reward, what evidence do we have that SRs are helpful for RL performance?

We argue that traditionally, SRs have seen *little* use in the maximization of reward outside of limited settings. For example, the most well-known use of the SR allows an agent to re-learn good actions quickly following a change in the reward function, albeit under the specific assumption of unchanging dynamics (Barreto, Dabney, et al. 2017). The main question we ask in this thesis is whether SRs can be useful more broadly, for general-purpose computations that underlie *all* of RL.

We answer the above question in the affirmative, by developing two new ways in which the SR can be used for more efficient learning of *any* value functions. The value function is a fundamental object in RL, and our methods uses the standard RL framework without making additional assumptions on reward, dynamics, or prior knowledge. Moreover, this novel way of using the SR is a *complement* to, rather than a replacement of previously suggested use of the SR (such as Barreto, Dabney, et al. (2017) for transferring across multiple reward functions). All in all, we have provided theoretical evidence in favour of the SR being a useful quantity for the brain

to encode, and our specific learning algorithms also hint at new testable neuroscience hypothesis for *how* a neural-representation may be used in brain computation.

## 1.2 Objectives and Outline

The main objective of this thesis is to develop mathematical and algorithmic techniques of using the successor representation for efficient credit assignment in the *single-task*, *tabula rasa* learning setting. This contrasts from previous work in both reinforcement learning and neuroscience which mainly consider SRs as being useful in the *transfer* or *multitask* settings.

The thesis is divided into the following chapters. Chapters 2 and 3 are background chapters. Chapter 2 introduces the reinforcement learning framework in depth, along with common algorithms employed under this framework. This gives us the language to describe intelligent behaviour. The methods in this chapter are "tabular", meaning that we use only *discrete* features. In chapter 3 we extend the result of chapter 2 to work with continuous features. This chapter also formally introduces the SR.

Chapters 4 and 5 are content chapters. Chapter 4 details our first method of using the SR: the $\lambda$-SRE, for re-weighing the learning error signal from future states. So far, this idea has only been developed for the tabular setting. Chapter 5 proposes a spectrum of algorithms—the $\lambda$ value function—that generalizes previous methods, namely the "model free" and "successor features" parameterization of value functions. This idea is readily applicable to the reinforcement learning setting with nonlinear function approximation. Both chapters leverage the use of predictive representations for non-local credit assignment, even if only local information from the environment can be accessed.

Finally, chapter 6 is a concluding chapter that discusses the over-arching ideas of this thesis.

## 1.3 Summary of Contribution

The main contribution of the current work to machine learning and RL is of *sample efficiency*: the proposed methods more effectively leverage the *same* amount of interactions between the agent and the environment. That is, given the same amount of (one-step) data, we show we can more effectively leverage information compared to baseline methods. We also contribute to neuroscience through expanding the space of theories and models. While we do not have new neuroscience experiments, we discuss the neuroscientific implications of our methods given the current literature. We hope this will inspire empirical experiments in the future to evaluate our predictions.

The main idea of chapter 4 was accepted for publication at the Biological and Artificial Reinforcement Learning Workshop at NeurIPS 2020. The results in chapter 5 were accepted for publication at the Association for the Advancement of Artificial Intelligence conference (AAAI 2022).

The current thesis were done under the co-supervision of Prof. Joelle Pineau and Prof. Blake Richards. I received collaborative help from Veronica Chelu for chapters 4 and 5. I was the lead author and main contributor to idea generation, experiments, theoretical derivation, writing and presentation.

# 2

# Reinforcement Learning

The ability to learn in a trial-and-error manner from experience is a fundamental cornerstone of intelligence. One may postulate on the precise objective this learning should achieve, for example, a sensible objective for learning may be to develop *behaviour* which maximizes the amount of external reward an organism receives. This is the fundamental objective that Reinforcement Learning (RL) tackles.

Broadly, consider the following setting: given an environment containing rewards, and a set of allowable actions, how should an *agent* (e.g. an animal in the wild, or an artificial agent in a simulated environment) behave in the environment so as to maximize the amount of total reward it receives? While the objective is easily described, achieving it is not so easy. Each action an agent takes can have an influence on the final outcome, and each reward received can be attributed as a consequence of *any* of the actions taken up to that point. Similarly, some actions can be informative about new ways of achieving more rewards, while others can directly result in rewards. From such complexities, figuring out the best sequence of actions to maximize total reward can quickly become intractable. Therefore, an important focus of algorithms that work under the RL framework is to solve this issue—sometimes referred to as *temporal credit assignment* (Minsky 1961; R. S. Sutton 1984)—in an efficient manner.

This chapter will first focus on formalizing the problem setting of RL as a *framework* in which intelligent behaviour can be modelled and studied. Then we will intro-

duce a variety of common methods from the literature for efficiently solving problems formulated under this framework.

## 2.1 Markov Decision Process

We begin by formalizing a general notion of "environment" and the rules within it, using the Markov Decision Process (MDP, introduced as early as Bellman (1957)). This is a common formalism for modelling decision-making with discrete time-steps (i.e. we model time as discrete "chunks" of equal length).

We use "state" ($s$) to describe the agent within the environment at a particular timestep $t$, with capitalized letter denoting random variables ($S_t$), and lowercase denoting a particular sample ($s_t$). Italicized $\mathcal{S}$ is used to denote the set of all states, $s \in \mathcal{S}$. Similarly, within each state, the agent will have access to a number of actions it can perform, denoted $a \in \mathcal{A}$. Performing action $A_t$ in state $S_t$ will result in the agent moving to a different state, $S_{t+1}$, with this transition being modelled via a transition probability function. Finally, each transition also accompanies a *reward* ($R_{t+1}$), depending on the actions taken in each state. To make a concrete example, in a maze, the states describe the locations of the agent; the actions can describe where to go next (e.g. up, down, left, or right), and the agent can receive a reward for reaching the end of the maze (i.e. reward is some positive scalar at the state that describes the end of the maze).

Importantly, the next state ($S_{t+1}$) and reward ($R_{t+1}$) the agent receives depends *only* on the current state ($S_t$) and action ($A_t$). This is known as the **Markov property**. Thus, the above framework for describing decision making processes is referred to as a Markov Decision Process, which we formally define below.

**Definition 2.1.1.** A **Markov Decision Process** (MDP) (Puterman 1994) consists of the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, r, P \rangle$. Where $\mathcal{S}$ is the set of states; $\mathcal{A}$ is the set of actions;

$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. $P : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the *transition probability function*, where $\mathcal{P}(\mathcal{S})$ is the set of probability distributions on the state space $\mathcal{S}$, and $P(\cdot|s, a)$ describes the distribution over $\mathcal{S}$ given current state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$.

We now introduce the formalism describing how to act within an MDP environment. A **policy**, $\pi$, dictates how actions are selected at each timestep ("decision epoch").[1] Formally, we consider a *randomized* policy, $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, as mapping from the space of states $\mathcal{S}$ to a set of probability distributions on the space of actions, $\mathcal{A}$. We write $\pi(A|S)$ to denote the probability of sampling action $A \in \mathcal{A}$ while being in state $S \in \mathcal{S}$ under policy $\pi$.[2] We can likewise consider the class of *deterministic* policies as deterministic functions mapping from $\pi : \mathcal{S} \to \mathcal{A}$. Deterministic policies are a subset of randomized policies. Intuitively, given any state $s \in \mathcal{S}$, a policy $A \sim \pi(\cdot|s)$ tells us how to act within that state.

Ultimately, we are interested in finding a policy to *maximize* reward. Yet, to do so, we first need to tackle a sub-problem of the amount of reward *some* given policy will receive in an MDP. In the following chapter (section 2.2), we consider this setting where the policy is fixed; i.e. we do not change the policy and estimate certain properties of a MDP given the policy. Fixing the policy within a MDP induces a Markov Reward Process.

**Definition 2.1.2.** A **Markov Reward Process** (MRP) is defined by the tuple $\mathcal{M}_\pi = \langle \mathcal{S}, r_\pi, P_\pi \rangle$. Similar to a MDP, $\mathcal{A}$ corresponds to the set of states. Given a

[1] More precisely, following the convention of Puterman (1994), a **decision rule**, $d_t$, describes how action is selected at decision epoch $t$. A policy is a set of decision rules for all time-steps, $\pi = (d_1, d_2, ...)$. However, within this thesis, we only consider the set of *stationary* policies: policies that have the same decision rule for all decision epochs, $\pi = (d, d, ...)$. Therefore, with a slight abuse of terminologies we will use "policy" and "decision rule" interchangeably.

[2] This is technically the set of *stationary, randomized* (SR) policies, $\pi \in \Pi^{SR}$: *stationary* policies that contain a single *Markovian, randomized* (MR) decision rule; $\pi = (d, d, ...)$, $d \in D^{MR}$. A decision rule is Markovian if it only depends on the current state, and it is randomized because it is a probabilistic mapping to the space of actions. This is the most widely used policy class in contemporary RL. For a full treatment extending into non-stationary policies and non-Markovian decision rules, see Puterman (1994), chapter 2.

policy $\pi$ and MDP $\mathcal{M}$, we induce an MRP as follows:

$$r_\pi(S) = \sum_{a \in \mathcal{A}} r(S, \pi(a|S)) \quad , \quad P_\pi(\cdot|S) = \sum_{a \in \mathcal{A}} P(\cdot|S, \pi(a|S)). \tag{2.1}$$

Here we describe the policy-dependant reward function $r_\pi : \mathcal{S} \to \mathbb{R}$, and transition function $P_\pi : \mathcal{S} \to \mathcal{P}(\mathcal{S})$.

In essence, we arrive at a MRP from a MDP by "absorbing" the policy information into the reward and transition functions. MRP is a convenient formalism to describe the dynamics of any given policy $\pi$ within a MDP. Specifically, while we are ultimately interested in finding the "best" policy, doing so often requires the ability to measure the "goodness" of a given policy $\pi$ in order to improve it. This process of measuring a policy's "goodness" is referred to as *policy evaluation*, which we will discuss in the following chapter.

## 2.2 Policy Evaluation

So far, we have described the formalism to describe an "environment" (the MDP, $\mathcal{M}$), and how to act within the environment (the policy, $\pi$). We now tackle the question of measuring how "good" an agent is acting within the environment. Since the ultimate goal of RL is to maximize the total amount of reward received, we can measure how much reward a given policy receives. We first introduce the concept of a **discount factor**, $\gamma$. The discount factor is a scalar, $0 \leq \gamma \leq 1$ applied to reward received further into the future. The smaller the $\gamma$, the more "myopic" the agent becomes—in favouring immediate rewards over longer-term rewards. We can now define the **return**, which describes the amount of discounted reward received by an agent in a particular trajectory of experience from time-point $t$.

**Definition 2.2.1.** A **discounted return** is the total reward received from the current

time-point $t$:

$$G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.2}$$

with future rewards being exponentially discounted by $0 \leq \gamma \leq 1$.

In the above definition we have written the *infinite horizon* return which sum over an infinitely-long future horizon. It should be noted that for many practical settings, we will instead be interested in the **episodic return**, collected from *episodes* with finite lengths $T$. In such settings we will simply sum the (discounted) reward until the end of the episode instead, $G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$.

The return is something that can be sampled empirically, such as by running the agent in the environment, collecting all rewards, and computing the return using definition 2.2.1. However, the return is a noisy estimate, subjected to the stochasticity in the policy, transition and reward functions. In fact, what we really care about is *on average* how much overall reward a policy can collect. We can define precisely a criterion to describe this—the *expected total discounted reward criterion* (Puterman (1994), chapter 5.3)—meaning we are interested in the total amount of reward received, *in expectation*, with reward further into the future being less important (i.e. $\gamma$-discounted). We formally define this below.

**Definition 2.2.2.** Given MRP $\mathcal{M}_\pi = \langle \mathcal{S}, r_\pi, P_\pi \rangle$ and discount factor $\gamma \in [0, 1]$, the **value function** of state $s \in \mathcal{S}$ is the expected return when starting from $s$ and following the MRP dynamics induced by policy $\pi$:

$$v_\pi(s) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right]. \tag{2.3}$$

We use the subscript $\mathbb{E}_\pi[\cdot]$ as shorthand denote expectation over the dynamics of the MRP induced by fixing the policy $\pi$ in MDP $\mathcal{M}$.

The value function of equation 2.3 precisely describes the *expected total discounted reward criterion*. Ultimately we will be interested in maximizing this criterion—through changing the policy to induce an environmental dynamic that results in a

higher total reward. However, for now we first deal with how to estimate the value as described in equation 2.3. This is the problem of **policy evaluation** (prediction) and we introduce various algorithms for solving this problem for the remainder of section 2.2.

### 2.2.1 Dynamic Programming

The most suitable way to do policy evaluation depends on the amount of information available and any computational constraints. We begin in the most lenient setting and consider a case where we have full access to the MRP $\mathcal{M}_\pi = \langle \mathcal{S}, r_\pi, P_\pi \rangle$, with discrete state spaces, bounded rewards, and stationary reward and transition probabilities (we borrow these standard assumptions from Puterman (1994), chapter 6). To simplify notation, we introduce the matrix notation value function here.

**Definition 2.2.3. Matrix notation**. We write MRP $\mathcal{M}_\pi = \langle \mathcal{S}, r_\pi, P_\pi \rangle$ in matrix notation. Let $\mathbf{P}_\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ be the transition matrix, $(\mathbf{P}_\pi)_{ij} = P_\pi(S' = j|S = i)$. $\mathbf{r}_\pi \in \mathbb{R}^{|\mathcal{S}|}$ be the reward vector, $(\mathbf{r}_\pi)_i = r_\pi(i)$. We can write the value function $\mathbf{v} \in \mathbb{R}^{|\mathcal{S}|}$ (definition 2.2.2) in matrix form as follows:

$$\begin{aligned}
\mathbf{v}_\pi &= \sum_{t=1}^{\infty} (\gamma \mathbf{P}_\pi)^{t-1} \mathbf{r}_\pi \,, \\
&= \mathbf{r}_\pi + \gamma \mathbf{P}_\pi \left( \mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{r}_\pi + \gamma^2 \mathbf{P}_\pi \mathbf{P}_\pi \mathbf{r}_\pi + ... \right) \,, \\
&= \mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{v}_\pi \,; \qquad (2.4)
\end{aligned}$$

where $(\mathbf{v}_\pi)_i = v_\pi(i)$ and $\gamma \in [0, 1]$.

We observe from definition 2.2.3 that the matrix form value function can be solved by the linear system of equations:

$$\mathbf{v}_\pi = \mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{v}_\pi \,, \qquad (2.5)$$

$$\Rightarrow (\mathbf{I} - \gamma \mathbf{P}_\pi) \mathbf{v}_\pi = \mathbf{r}_\pi \,. \qquad (2.6)$$

**Theorem 2.2.1.** *(Puterman (1994), theorem 6.1.1) Given* $0 \leq \gamma < 1$*, for any (stationary Markov randomized) policy* $\pi$*,* $\mathbf{v}_\pi$ *is the unique solution of*

$$\mathbf{v} = \mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{v}. \tag{2.7}$$

*Further,* $\mathbf{v}_\pi$ *may be written as*

$$\mathbf{v}_\pi = (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} \mathbf{r}_\pi. \tag{2.8}$$

*Proof.* In brief we establish that the spectral radius of $\gamma \mathbf{P}_\pi$ is less than 1, and the inverse $(\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1}$ exists. See Puterman (1994), theorem 6.1.1 for details. □

We can therefore reassure ourselves that a solution exists for the system in equation 2.5. The value function can be solved here simply through matrix inversion.

**Algorithm 1:** Matrix inversion solution for policy evaluation

1 Input: MRP parameters $\mathbf{P}_\pi$, $\mathbf{r}_\pi$, $\gamma$ ;
2 $\mathbf{v}_\pi = (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} \mathbf{r}_\pi$ ;

Due to the computational complexity of matrix inversion, we are also interested in an iterative solution. This lets us approximate with arbitrary precision the solution without having to do matrix inversion. Let us consider the space of value functions, $\mathbf{v} \in \mathbf{V}$. We introduce the **policy evaluation operator**,[3] $L_d : \mathbf{V} \to \mathbf{V}$. Application of the operator to $\mathbf{v} \in \mathbf{V}$ results in the linear transformation:

$$L_d \, \mathbf{v} \doteq \mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{v} \, . \tag{2.9}$$

It should be noted that by definition of $\mathbf{v}_\pi$ (equation 2.5), the fixed point to the operator $L_d$ is $\mathbf{v}_\pi$: $L_d \, \mathbf{v}_\pi = \mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{v}_\pi = \mathbf{v}_\pi$. We next show that repeated application of $L_d$ will converge to the fixed point $\mathbf{v}_\pi$.

[3] We use the subscript $L_d$ to denote the operator's dependence on the decision rule ($d$) evaluated. This is sometimes also referred to as the *Bellman operator*; not to be confused with the *Bellman optimality operator*, whose fixed point is the optimal value function. We introduce this later in section 2.3.1.

**Theorem 2.2.2.** *(**Banach Fixed-Point Theorem**, adopted from Puterman (1994))*

*Suppose* $\mathbf{V}$ *is a Banach space and* $T : \mathbf{V} \to \mathbf{V}$ *is a contraction mapping.*[4] *Then*

1. *there exists a unique* $\mathbf{v}^* \in \mathbf{V}$ *such that* $T\mathbf{v}^* = \mathbf{v}^*$*; and*

2. *for arbitrary initial point* $\mathbf{v}^0 \in \mathbf{V}$*, the sequence* $\{\mathbf{v}^n\}$ *defined by*

$$\mathbf{v}^{n+1} = T\mathbf{v}^n = T^{n+1}\mathbf{v}^0 \tag{2.10}$$

*converges to* $\mathbf{v}^*$*.*

*Proof.* In brief, via triangle inequality and property of contraction mappings. See Banach (1922) and Puterman (1994) (theorem 6.2.3) for details. □

**Proposition 2.2.3.** *The policy evaluation operator,* $L_d : \mathbf{V} \to \mathbf{V}$*, is a* $\gamma$*-contraction in the infinity norm,* $||\mathbf{v}||_\infty = \max_i |\mathbf{v}_i|$.[5] *Given vectors* $\mathbf{u} \in \mathbf{V}$*,* $\mathbf{v} \in \mathbf{V}$*:*

$$||L_d\,\mathbf{u} - L_d\,\mathbf{v}||_\infty \leq \gamma ||\mathbf{u} - \mathbf{v}||_\infty. \tag{2.11}$$

*Proof.* (Puterman 1994) We denote a vector of 1's of dimension $|\mathcal{S}|$, $\mathbf{1} = [1, 1, ...]^\top$:

$$\begin{aligned} ||L_d\,\mathbf{u} - L_d\,\mathbf{v}||_\infty &= ||(\mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{u}) - (\mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{v})||_\infty\,, \\ &= \gamma ||\mathbf{P}_\pi(\mathbf{u} - \mathbf{v})||_\infty\,, \\ &\leq \gamma ||\mathbf{P}_\pi \cdot (\mathbf{1}||\mathbf{u} - \mathbf{v}||_\infty)||_\infty\,, \\ &= \gamma ||\mathbf{P}_\pi \cdot \mathbf{1}||_\infty \cdot ||\mathbf{u} - \mathbf{v}||_\infty\,, \\ &= \gamma ||\mathbf{u} - \mathbf{v}||_\infty\,; \end{aligned} \tag{2.12}$$

where $||\mathbf{P}_\pi \cdot \mathbf{1}||_\infty = 1$ follows from $\mathbf{P}_\pi$ being a stochastic transition matrix. □

[4] A Banach space is a vector space with a defined norm $||\cdot||$. It is *complete* in that every *Cauchy sequence*'s limit is contained within that space (see Puterman (1994), Appendix C for details). This is a standard formalism to analyze fixed points and convergence of iterative algorithms. An operator $T$ is a contraction mapping if $||T\mathbf{u} - T\mathbf{v}|| \leq ||\mathbf{u} - \mathbf{v}||\,, \forall \mathbf{u}, \mathbf{v} \in \mathbf{V}$.

[5] An operator $L_d$ is a $\gamma$-contraction if its application contracts space by at most $\gamma \in [0, 1)$, this is defined formally in equation 2.11.

We observe that by proposition 2 we have proven $L_d$ to be a contraction, and by theorem 2.2.2 we know that iterative application of the operator $L_d$ to an arbitrary initial point $\mathbf{v}^0 \in \mathbf{V}$ will result in a sequence that converges to the unique fixed point, $\mathbf{v}_\pi$—the correct value function for the MRP. This gives us an iterative algorithm for policy evaluation, which is sometimes referred to as the **dynamic programming** (DP) solution for policy evaluation (algorithm 2).[6]

**Algorithm 2:** Iterative DP solution for policy evaluation (R. S. Sutton and Barto 2018)

Input: MRP parameters $\mathbf{P}_\pi$, $\mathbf{r}_\pi$, $\gamma$ ;
Initialize $\mathbf{v}^0$ arbitrarily ;
**for** *k=1,2,... until satisfied* **do**
  $\mathbf{v}^k \leftarrow \mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{v}^{k-1}$
**end**

### 2.2.2 Monte-Carlo Simulation

So far, the solutions discussed in section 2.2.1 assumes full access to a perfect environmental model, in the form of $\mathbf{P}_\pi$ and $\mathbf{r}_\pi$. For most cases we are interested in, we will not have access to the full MDP / MRP. Instead, we will only be able to *act* within the environment, and in doing so *sample* information about the environment. Our objective is the same—to evaluate the value function for a given policy, $v_\pi$.

Despite the added limitation, the subsequent methods still derive largely from the iterative DP solution (algorithm 2). Recall in the setting where we have access to perfect transition and reward models of the MRP $\mathcal{M}_\pi = \langle \mathcal{S}, r_\pi, P_\pi \rangle$ with discount factor $\gamma$, we can solve for the value function $\mathbf{v}_\pi = \mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{v}_\pi$ (equation 2.5) iteratively by applying the policy evaluation operator, $L_d$ (equation 2.9). We can write the $k$-th

[6] More generally, this is an application of the *method of successive approximation*.

iterative update in component form, denoting each state as $\mathbf{v}_i = v(i)$,

$$\begin{aligned} v^{(k+1)}(s) &= (1-\alpha)\, v^{(k)}(s) + \alpha \left(L_d\, v^{(k)}\right)(s), && \forall s \in \mathcal{S}, \\ &= (1-\alpha)\, v^{(k)}(s) + \alpha \left(r_\pi(s) + \gamma \sum_{s'} P_\pi(s'|s) v^{(k)}(s')\right), && \forall s \in \mathcal{S}; \end{aligned} \tag{2.13}$$

where $\alpha \in (0, 1]$ is a step-size parameter. Intuitively, for each step we are updating our current estimate, $v^{(k)}(s)$ toward the target $(L_d\, v^{(k)})(s)$. Note that $\alpha = 1$ implicitly in algorithm 2.

For the remainder of this chapter, we assume we cannot evaluate $(L_d\, v^{(k)})(s)$ exactly, instead we only have access to a random variable, $G = (L_d\, v^{(k)})(s) + \omega$, where $\omega$ is some random noise term. We can substitute $G$ in lieu of $(L_d\, v^{(k)})(s)$ as a learning *target*, resulting in the following update,

$$\begin{aligned} v^{(k+1)}(s) &= (1-\alpha)\, v^{(k)}(s) + \alpha \left((L_d\, v^{(k)})(s) + \omega\right), \\ &= v^{(k)}(s) + \alpha \left(G - v^{(k)}(s)\right). \end{aligned} \tag{2.14}$$

We refer to this class of methods as **stochastic iterative algorithms**, which solves the systems in equation 2.5 in a stochastic, component-wise fashion. All subsequent RL algorithms discussed will follow this formalism,[7] with the only difference being how we construct the random variable $G$, which will inform how we sample the environment. Indeed, it is no coincidence that the letter $G$ is used—this random variable is precisely the **return** in RL (see definition 2.2.1). We will discuss various ways of constructing a return and using it to estimate the value function for the remainder of this chapter.

The simplest way of estimating the value function (definition 2.2.2) is to directly construct the return with Monte-Carlo samples and use this as our estimator of value. Specifically, assuming we are in some state $s \in \mathcal{S}$ and interested in estimating the

[7]Here we consider the class of algorithm with tabular features (i.e. per-state values are stored as look-up tables). For the case where $v(s)$ is a parameterized function, the update will need to include the gradients of the difference with respect to each parameters. This will be discussed in greater depth in chapter 3.

value function of that state given the policy, $v_\pi(s)$, we can simply act according to to policy $\pi$ to sample a trajectory $(S_t, R_{t+1}, S_{t+1}, R_{t+2}, ...)$, and count the total amount of rewards received.

**Definition 2.2.4.** A **Monte-Carlo (MC) Return** (R. S. Sutton and Barto 2018), $G_t^{\text{MC}}$, under the expected total discounted reward criterion, given a sampled trajectory $(R_{t+1}, R_{t+2}, ..., R_T)$ and discount factor $\gamma$, is defined as:

$$G_t^{MC} \doteq R_{t+1} + \gamma R_{t_2} + ... + \gamma^{T-t-1} R_T. \tag{2.15}$$

Where $T$ is the episode length.

We know that the MC return is an estimate of the value function, since $\mathbb{E}[G_t^{MC}] = v_\pi(s_t)$. This yields our first stochastic approximation algorithm for policy evaluation, **Monte-Carlo Policy Evaluation** (algorithm 3).[8] Since the MC return requires having a full trajectory, algorithm 3 is only applicable in an episodic setting, where updates to the value estimate happens after each episode terminates.

**Algorithm 3:** Every visit Monte Carlo policy evaluation (Bertsekas and Tsitsiklis 1995)

Input: MRP $\mathcal{M}_\pi$, discount factor $\gamma \in [0, 1]$, stepsize $\alpha \in (0, 1]$ ;
Initialize value estimate $v^{(0)}(\cdot)$ arbitrarily ;
**while** *not converged* **do**
    Sample episode trajectory from $\mathcal{M}_\pi$: $(S_0, R_1, S_1, R_2, ..., S_{T-1}, R_T)$ ;
    **for** *each state encountered,* $S_t$ **do**
        Compute return: $G_t^{MC} = \sum_{k=1}^{T-t} \gamma^{k-1} R_{t+k}$ ;
        Update: $v^{(k+1)}(S_t) \leftarrow v^{(k)}(S_t) + \alpha \left( G_t^{MC} - v^{(k)}(S_t) \right)$
    **end**
**end**

[8] More precisely, algorithm 3 is the *every visit* variant of Monte-Carlo policy evaluation, which may be biased. For a detailed discussion on this and the alternative *first visit* variant, see Bertsekas and Tsitsiklis (1995), chapter 5.2.

### 2.2.3 Temporal Difference

While the Monte-Carlo return allows us to do policy evaluation while only being able to take samples from the environment, we notice each MC return requires a full trajectory to be collected before it can be constructed. We can do better by noticing the recursive nature of a return:

$$\begin{aligned}\mathbb{E}_\pi[G_t] &= \mathbb{E}_\pi\left[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ... + \gamma^{T-t-1} R_T\right], \\ &= \mathbb{E}_\pi[R_t] + \gamma v_\pi(S_{t+1}).\end{aligned} \tag{2.16}$$

We can use the current value estimate $v(\cdot)$ in lieu of the value function $v_\pi(\cdot)$. This gives us the temporal difference return.

**Definition 2.2.5.** The one-step **temporal difference (TD) return** (R. S. Sutton and Barto (2018), chapter 6.1), $G_t^{TD}$, given a sampled one-step transition $(S_t, R_{t+1}, S_{t+1})$, discount factor $\gamma$, and current value estimate $v(\cdot)$, is defined as:

$$G_t^{TD} \doteq R_{t+1} + \gamma v(S_{t+1}), \tag{2.17}$$

where $v(S_{t+1})$ is the value function estimate for state $S_{t+1}$.

Unlike the Monte-Carlo return which require the full trajectory, the one-step TD return can be computed using just the one-step sampled transition $(S_t, R_{t+1}, S_{t+1})$. This process of using one's own value estimate in constructing the learning target is known as **bootstrapping** in RL. Bootstrapping relates closely to the the dynamic programming solution for policy evaluation, and the one-step TD return (definition 2.2.5) can be viewed as a single-component sample obtained from an application of the policy evaluation operator (equation 2.9), $\mathbb{E}_\pi[G_t^{TD}|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma v(S_{t+1})|S_t = s] = (\mathbf{r}_\pi + \gamma \mathbf{P}_\pi v)(s)$.

We further define the **TD error**, commonly written as $\delta_t$. It denotes the signed error between the value estimate and the one-step TD return,

$$\delta_t = R_{t+1} + \gamma v(S_{t+1}) - v(S_t). \tag{2.18}$$

The TD return gives us the one-step TD algorithm (also known as TD(0)) for policy evaluation (algorithm 4).

**Algorithm 4:** Online incremental TD(0) for policy evaluation (R. S. Sutton and Barto 2018)

Input: MRP $\mathcal{M}_\pi$, discount factor $\gamma \in [0, 1]$, stepsize $\alpha \in (0, 1]$ ;
Initialize value estimate $v^{(0)}(\cdot)$ arbitrarily ;
**while** *not converged* **do**
    Sample environment $\mathcal{M}_\pi$, receive one-step experience tuple $(S_t, R_{t+1}, S_{t+1})$ ;
    Compute return: $G_t^{TD} = R_{t+1} + \gamma v^{(k)}(S_{t+1})$ ;
    Update: $v^{(k+1)}(S_t) \leftarrow v^{(k)}(S_t) + \alpha \left( G_t^{TD} - v^{(k)}(S_t) \right)$
**end**

### 2.2.4 Lambda Return

In the previous section we motivated the one-step TD return by replacing all future sequence of rewards (after a single step) with a current estimate of value. Similarly, we can perform the same substitution after waiting multiple steps rather than a single step:

$$\begin{aligned} \mathbb{E}_\pi[G_t] &= \mathbb{E}_\pi[R_t] + \gamma v_\pi(S_{t+1}) \,, \\ &= \mathbb{E}_\pi[R_t + \gamma R_{t+1}] + \gamma^2 v_\pi(S_{t+2}) \,, \\ &= \mathbb{E}_\pi[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2}] + \gamma^3 v_\pi(S_{t+3}) \,, \\ &\dots \,. \end{aligned} \tag{2.19}$$

**Definition 2.2.6.** The **n-step return** (R. S. Sutton and Barto 2018), $G^{(n)}$, given a sampled n-step trajectory $(S_t, R_{t+1}, S_{t+1}, R_{t+2}, ..., R_{t+n}, S_{t+n})$, discount factor $\gamma$, and current value estimate $v(\cdot)$, is defined as:

$$G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n v(S_{t+n}) \tag{2.20}$$

We note that the one-step TD return is a special case of the n-step return, when $n = 1$, while the Monte-Carlo return corresponds to $n$ approaching infinity (in the

infinite horizon case) or when $n = T - t$ (for the episodic case). In general, since the sampled rewards $R_{t+k}$ are random variables, a larger $n$ (i.e. a long trajectory) corresponds to a higher variance estimate. On the other hand, since we are *bootstrapping* using an estimate of value at the end of the n-step trajectory, using small $n$'s results in higher potential bias.

Finally, in the setting where we have access to a full trajectory, we can in fact compute *all* possible n-step returns, and average over them for a potentially more accurate estimate. The $\lambda$-return is one way to approach this, by proposing an exponential-averaging scheme.

**Definition 2.2.7.** The **$\lambda$-return** (R. S. Sutton and Barto 2018), $G_t^\lambda$, given a sampled full trajectory $(S_t, R_{t+1}, S_{t+1}, R_{t+2}, ...)$, discount factor $\gamma$, and current value estimate $v(\cdot)$, is defined as an $\lambda$-weighted exponential average over all $n$-step returns:

$$G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}\,, \tag{2.21}$$

$$= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\left((\sum_{k=1}^{n}\gamma^{k-1}R_{t+k}) + \gamma^n v(S_{t+n})\right)\,; \tag{2.22}$$

where $0 \leq \lambda \leq 1$ is a hyperparameter controlling the exponential average.

In a similar vein to the $n$-step return, when $\lambda = 0$, the $\lambda$-return reduces to the one-step TD return, and when $\lambda = 1$ the $\lambda$-return becomes the Monte-Carlo return. As one interpolates from $0 \leq \lambda \leq 1$, a similar bias-variance trade-off is had. The parameter $\lambda$ can be thought of as controlling the "recency bias" of the return—a smaller $\lambda$ more highly weigh $n$-step returns in the immediate future, while a larger $\lambda$ give more importance to $n$-step returns further into the future.

As an aside, the $\lambda$-return can be equivalently written in a number of different forms.

**Remark 2.2.4.** *Given current value estimate $v(\cdot)$, the $\lambda$-return can be written equiv-*

*alently in the following forms:*

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\left[(\sum_{k=1}^{n}\gamma^{k-1}R_{t+k}) + \gamma^n v(S_{t+n})\right] \tag{2.23}$$

$$= R_{t+1} + \gamma\left(\sum_{n=1}^{\infty}(\lambda\gamma)^{n-1}\left[(1-\lambda)v(S_{t+n}) + \lambda R_{t+n+1}\right]\right) \tag{2.24}$$

$$= v(S_t) + \sum_{n=0}^{\infty}(\lambda\gamma)^n\delta_{t+n} \tag{2.25}$$

*Proof.* We start with expanding the $\lambda$-return as an exponential average over n-step returns—noting we write the value estimate as $V_k = v(S_k)$ for brevity:

$$\begin{aligned} G_t^\lambda &= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\left[(\sum_{k=1}^{n}\gamma^{k-1}R_{t+k}) + \gamma^n V_{t+n}\right], \\ &= (1-\lambda)\lambda^0[R_{t+1} + \gamma V_{t+1}] \\ &\quad + (1-\lambda)\lambda^1[R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+2}] \\ &\quad + \dots; \end{aligned}$$

We use the fact that the factors $(1-\lambda)\sum_{k=0}^{\infty}\lambda^k = 1$ to pull out the variables $R_{t+1}$, $R_{t+2}$, ... from inside of the brackets,

$$\begin{aligned} G_t^\lambda &= R_{t+1} + (1-\lambda)\lambda^0\gamma V_{t+1} \\ &\qquad + (1-\lambda)\lambda^1\gamma[R_{t+2} + \gamma V_{t+2}] \\ &\qquad + (1-\lambda)\lambda^2\gamma[R_{t+2} + \gamma R_{t+3} + \gamma^2 V_{t+3}] \\ &\qquad + \dots, \\ &= R_{t+1} + \gamma\lambda^0(1-\lambda)V_{t+1} \\ &\quad + \gamma\lambda R_{t+2} + \gamma^2\lambda^1(1-\lambda)V_{t+2} \\ &\quad + \dots. \end{aligned} \tag{2.26}$$

Re-arranging the above gives us equation 2.24:

$$\begin{aligned} G_t^\lambda &= R_{t+1} + (\gamma\lambda^0)[(1-\lambda)V_{t+1} + \lambda R_{t+2}] \\ &\quad + (\gamma^2\lambda^1)[(1-\lambda)V_{t+2} + \lambda R_{t+3}] \\ &\quad + ... \\ &= R_{t+1} + \gamma \sum_{n=1}^{\infty} (\gamma\lambda)^{n-1}[(1-\lambda)V_{t+n} + \lambda R_{t+n+1}] \,. \end{aligned}$$

Finally, we can add the terms $(+V_t - V_t)$ to equation 2.26 without changing the result to arrive at equation 2.25 (using the TD-error definition of equation 2.18):

$$\begin{aligned} G_t^\lambda &= V_t - V_t + R_{t+1} + \gamma V_{t+1} \\ &\quad - \gamma\lambda V_{t+1} + \gamma\lambda R_{t+2} + \gamma^2\lambda V_{t+2} \\ &\quad - \gamma^2\lambda^2 V_{t+2} + ... \,, \\ &= V_t + \sum_{n=0}^{\infty} (\gamma\lambda)^n [R_{t+n+1} + \gamma V_{t+n+1} - V_{t+n}] \,. \end{aligned}$$

□

From the above, we see multiple interpretations of the $\lambda$-return. While they are mathematically equivalent, different estimators are best suited to approximate the different algebraic forms—indeed one main contribution of the current thesis is the proposal of novel algorithms inspired by the different algebraic forms. Specifically, equation 2.23 shows the $\lambda$-return as an exponential average over all future $n$-step returns—this is the original motivation for the $\lambda$-return. Equation 2.24 shows the $\lambda$-return can be separated into an immediate reward term and a discounted future term that is a cumulative mixture of future rewards and value estimates. We will use this identity extensively in chapter 5. Finally, equation 2.25 shows the $\lambda$-return can be interpreted as the current value estimate plus a cumulative discounted sum over all future TD-errors, akin to temporally regularizing the current value estimate to minimize both current and future one-step TD errors. We will use this identity in chapter 4.

Like the Monte-Carlo return, the full $\lambda$-return is a quantity which requires the full trajectory to be computed. However, in general, $\lambda$-return (with $\lambda < 1$) makes better use of the full episode trajectory as compared to the Monte-Carlo return, as well as the $n$-step return (see R. S. Sutton and Barto (2018), chapter 12.1). This way of computing the per-state $\lambda$-return after collecting a full episodic trajectory gives us the End-of-episode $\lambda$-return algorithm (algorithm 5).[9]

**Algorithm 5:** End-of-episode $\lambda$-return policy evaluation (R. S. Sutton and Barto 2018)

Input: MRP $\mathcal{M}_\pi$, discount factor $\gamma \in [0, 1]$, stepsize $\alpha \in (0, 1]$, parameter $\lambda \in [0, 1]$;
Initialize value estimate $v^{(0)}(\cdot)$ arbitrarily ;
**while** *not converged* **do**
    Sample episode trajectory from $\mathcal{M}_\pi$: $(S_0, R_1, S_1, R_2, ..., S_{T-1}, R_T)$ ;
    **for** *each state encountered,* $S_t$ **do**
        Compute return:
        $G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t} \lambda^{n-1}((\sum_{k=1}^{n} \gamma^{k-1} R_{t+k}) + \gamma^n v(S_{t+n}))$ ;
        Update: $v^{(k+1)}(S_t) \leftarrow v^{(k)}(S_t) + \alpha \left(G_t^\lambda - v^{(k)}(S_t)\right)$
    **end**
**end**

Note that various convergence results are available for the stochastic iterative algorithms introduced in this chapter. They are outside of the scope of this thesis, though we refer the interested reader to Bertsekas and Tsitsiklis (1995), chapter 5.

## 2.3 Policy Improvement

Recall that the fundamental objective of RL is to produce reward-maximizing behaviour in a particular environment. We formalized environments using MDPs, and behaviour in terms of policies. So far, we have introduced a variety of methods for evaluating *a* given policy, $\pi$, which are all based on the application of the policy

[9] We use "end-of-episode" here for clarity about when the update occurs. This algorithm is sometimes also referred to by the name *forward view* TD($\lambda$), or the "off-line" $\lambda$-return algorithm (R. S. Sutton and Barto 2018). We avoid the use of "off-line" as it has recently been used to refer to RL algorithm that work on fixed-size datasets without collecting new data.

evaluation operator, $L_d$—either in full matrix form when it is available, or via doing component-wise stochastic approximation if one can only sample the environment. We now turn our attention to the question of finding the *best* policy inside of an MDP.

We will use the same formalism as the previous chapters. The environment is modelled as a MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, r, P \rangle$. We can condition the MDP on a given fixed policy $\pi$ to give rise to a MRP $\mathcal{M}_\pi = \langle \mathcal{S}, r_\pi, P_\pi \rangle$. The MRP transition matrix and reward function in matrix form are $\mathbf{P}_\pi \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$ and $\mathbf{r}_\pi \in \mathbb{R}^{|\mathcal{S}|}$, respectively. We use $\pi^*$ to denote the **optimal policy**, which is the policy that results in the maximum amount of discounted total reward (with discount factor $\gamma \in [0, 1]$). We are interested in finding such a policy through the process of policy improvement.

### 2.3.1 Value Iteration

Similar to how the problem of policy evaluation can be solved as a system of (linear) equations (equation 2.5) whose solution is the fixed point of the policy evaluation operator (equation 2.9), we can similarly write down a system of **optimality equations** for finding the *optimal* value function (i.e. the value function corresponding to the optimal policy $\pi^*$).[10] This (nonlinear) system can be written as:[11]

$$\mathbf{v} \doteq \max_{\pi \in \Pi} \{ \mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{v} \} . \tag{2.27}$$

We can similarly write down an operator whose fixed point is the above system. Similar to the policy evaluation operator, it applies a transformation in the space of value functions, $\mathbf{v} \in \mathbf{V}$.

[10] This is referred to in Puterman (1994) as the *optimality equations* or *Bellman equation*. We will exclusively refer to this as the "(Bellman) optimality equations" to avoid confusion with the policy evaluation equations, which is sometimes also referred to as the Bellman equations. As an aside, Bellman's historical contribution is more about the optimality equations.

[11] The maximum is typically defined over the set of Markov, deterministic decision rules, $d \in D^{MD}$. As we only work with stationary policies, here we define it over the set of stationary, deterministic policies $\pi \in \Pi^{SD}$ for notation brevity. Note that the solution for the class of stationary, deterministic policies is the same as the solution for the class of stationary, randomized policies (see Puterman (1994) proposition 6.2.1).

**Definition 2.3.1.** The **Bellman Optimality Operator**, $L : \mathbf{V} \to \mathbf{V}$, applies the following transformation in the space of value functions:

$$L\mathbf{v} \doteq \max_{\pi \in \Pi} \{\mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{v}\}. \tag{2.28}$$

It can be written in component forms (for each state $s \in \mathcal{S}$) as

$$v^{(k+1)}(s) = \max_{a \in \mathcal{A}} \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v^{(k)}(s') \right\}. \tag{2.29}$$

**Theorem 2.3.1.** *For arbitrary initial point $\mathbf{v}^0 \in \mathbf{V}$, the sequence $\{\mathbf{v}^n\}$ resulting from repeated application of the operator $\mathbf{v}^{n+1} = L\mathbf{v}^n$ converges to a unique fixed point $\mathbf{v}^*$ such that $L\mathbf{v}^* = \mathbf{v}^*$.*

*Proof.* Follows directly from the Banach Fixed-Point Theorem (theorem 2.2.2) and $L$ being a contraction mapping for $0 \leq \gamma < 1$ (see proposition 6.2.4 of Puterman (1994)). □

The Bellman Optimality Operator directly gives rise to the value iteration algorithm (algorithm 6), which can be used when we have full access to the MDP to give us the optimal policy, $\pi^*$.

**Algorithm 6:** Value Iteration (Puterman 1994)

Input: MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, r, P \rangle$, discount factor $\gamma \in [0,1]$ ;
Initialize value estimate $v^{(0)}(\cdot)$ arbitrarily ;
**while** *value not converged* **do**
  For each $s \in \mathcal{S}$, compute
  $v^{(k+1)}(s) = \max_{a \in \mathcal{A}} \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v^{(k)}(s') \right\}$ ;
**end**
Behave according to the last value function iterate:
$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v^{(k+1)}(s') \right\}$ ;

### 2.3.2 Q-Learning

We now introduce a stochastic approximation algorithm based on value iteration. Let us first define the **action-value function**:

$$q_\pi(s,a) = \mathbb{E}_\pi[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')] \,, \tag{2.30}$$

$$= \mathbb{E}_\pi[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_\pi(s)] \,. \tag{2.31}$$

We can write the component form value-iteration update, at the $k$-th step, with action-value functions and step-size parameter $\alpha \in (0,1]$. For each $s \in \mathcal{S}$ and $a \in \mathcal{A}$:

$$q^{(k+1)}(s,a) = (1-\alpha) q^{(k)}(s,a) + \alpha \left( r(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a' \in \mathcal{A}} q^{(k)}(s',a') \right) . \tag{2.32}$$

Imagine the setting without direct access to the MDP dynamics (i.e. $P(\cdot)$, $r(\cdot)$, etc.), but we can only sample the experiences $(S_t, A_t, R_{t+1}, S_{t+1}, ...)$. We write a similar update with the sampled transition and rewards:

$$q^{(k+1)}(S_t,A_t) = (1-\alpha) q^{(k)}(S_t,A_t) + \alpha \left( R_{t+1} + \gamma \max_{a' \in \mathcal{A}} q^{(k)}(S_{t+1},a') \right) . \tag{2.33}$$

The above update gives us the **Q-Learning** algorithm (Watkins (1989), Bertsekas and Tsitsiklis (1995) section 5.6). It can be seen as the stochastic approximation version of value iteration, using action-value functions $q(\cdot)$.[12]

**Algorithm 7:** Online Q-learning with one-step transitions (Bertsekas and Tsitsiklis 1995)

Input: MDP $\mathcal{M}$, discount factor $\gamma \in [0,1]$, stepsize $\alpha \in (0,1]$ ;
Initialize action-value estimate $q^{(0)}(\cdot)$ arbitrarily ;
**while** *policy not converged* **do**
    Given current state $S_t$, sample action $A_t$ ;
    Take action $A_t$ and observe $R_{t+1}, S_{t+1}$ ;
    Update: $q^{(k+1)}(S_t,A_t) \leftarrow$
     $q^{(k)}(S_t,A_t) + \alpha \left( R_{t+1} + \gamma \max_{a' \in \mathcal{A}} q^{(k)}(S_{t+1},a') - q^{(k)}(S_t,A_t) \right)$ ;
**end**

[12] Convergence proof for Q-learning are outside of the scope of this thesis; though we refer the interested reader to Bertsekas and Tsitsiklis (1995), section 5.6.

# 3

# Representation Learning

The difficulty of an information processing task depends on the way information is presented. This is the problem of **representation learning**—learning to represent information in a way that is more useful for downstream processing. This chapter deals specifically with the representation of *states* in reinforcement learning.

First, state representation arises naturally when *function approximation* is used. In this setting, the state space can be large (or infinite); we no longer have access to an explicit "identifier" for individual states (this is referred to as the *tabular* setting), but instead some feature vector describing each state. Sections 3.1 and 3.2 extend the tabular methods for policy evaluation and improvement of chapter 2 to this setting. For *linear* function approximation, the feature representations of each state are given, and we approximate values on top of the given features. For *deep / nonlinear* function approximation, we learn the features as well as the value approximation in an end-to-end manner using deep neural networks.

Second, we leverage the structure of the RL problem to learn representations specific to the MDP setting. Representation in this form rely on learning predictive knowledge about observations an agent will encounter in the future, in the form of *general value functions*. One particular kind of general value function—the successor representation—encodes the states that will likely be encountered in the future.[1] Sec-

[1]This is not to be confused with the predictive state representation (PSR) of Littman, R. S. Sutton, and Singh (2001). While they share the same high-level philosophy of having predictive

tions 3.3 and 3.4 deal with the successor representation and how it can also be used to estimate value. Notably, the two types of representations introduced above are not contradictory, but complementary. Function approximation gives rise to features, while general value functions are learned on top of features.

## 3.1 Policy Evaluation with Function Approximation

In chapter 2, we introduced many methods for solving a value function $v_\pi(s)$. Consider again some MDP, $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, r, P \rangle$. So far, we have only considered value functions that are *tabular*, meaning we have explicit access to each individual states in our MDP and each state's value can therefore be stored in a "look-up table", $v(s), s \in \mathcal{S}$, where we are given the state index $s$ at each timestep.

When states grow large, the tabular representation becomes a highly inefficient or even intractable method of solving the reinforcement learning problem. For example, a single state in the game of chess describes a unique board position, resulting in over $10^{45}$ states. Furthermore, if the observation is continuous rather than discrete (for instance, the position and velocities of a self-driving car), it is also unclear how to discretize a continuous space into a tabular one as the degree of precision is arbitrary. Such settings are standard for most if not all non-trivial, real-world tasks, so applying RL to non-trivial settings necessitates working with non-tabular features. This chapter introduce *state features* and approximate value functions. As usual, we first start with the setting of policy evaluation, before moving into control.

representation, PSRs are motivated by the problem of generating sufficient statistics for partially observable systems, through designing history dependent prediction objectives ("tests"). Later in this chapter, we will discuss successor representations, which assume Markovian (i.e. non history-dependent) observations, and learn about future state occupancy using this Markovian property.

### 3.1.1 Feature Representation of States

Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, r, P \rangle$, we "describe" a given state $s \in \mathcal{S}$ using *features*.

**Definition 3.1.1.** A **feature mapping**, $\phi : \mathcal{S} \to \mathbb{R}^d$, is mapping from the set of states to a $d$-dimensional set of **features**, $\Phi$. For brevity (and with an abuse of notation), we denote each feature vector equivalently as $\phi(s_t) = \boldsymbol{\phi}_t$, where $\boldsymbol{\phi}_t \in \Phi \subset \mathbb{R}^d$.

A feature can be thought of as a $d$-dimensional vector description of the state it represents. This has the additional benefits of allowing us to possibly describe similar states in similar ways, allowing for a degree of *generalization* in learning. Take the example of the self-driving car, where the feature may describe the car's velocity and position (say distance travelled on a linear track): $[30\text{km/h}, 100\text{m}]$. We understand the states $[29.9\text{km/h}, 99.9\text{m}]$ and $[30.1\text{km/h}, 100.1\text{m}]$ to be similar, and should most likely be assigned similar values—this arises naturally from using the above feature representation. On the other hand, if we discretize the states in tabular form, the above becomes independent states whose value must be learned independently, resulting in slower learning.

Usually, we assume the space of features to be much smaller than the space of states, $d << |\mathcal{S}|$. However, it should be noted that the tabular state representation is a special case of the feature representation: if we construct features as "one-hot" vectors of size $|\mathcal{S}|$ (where it is "1" at the state index and "0" everywhere else), we have effectively recovered the tabular representation used in chapter 2. It follows that the methods introduced in this section simply degenerate into the methods in chapter 2 when tabular (one-hot) state features are used.

Finally, there is the question of where features come from? We assume each state in the environment $s \in \mathcal{S}$ give some *state observation*. There are two ways of getting features. Firstly, we can simply use the state observation as the features, in which case we assume the feature set $\Phi$ is given as part of the environment. This is the *linear* value function approximation setting discussed in section 3.1.3. Alternatively,

we can *learn* a feature mapping $\phi : \mathcal{S} \to \mathbb{R}^d$ from the state observations to some $d$-dimensional feature space. This is a more general setting which we refer to as *nonlinear* value function approximation. Within this thesis we only consider the setting of learning $\phi$ in an end-to-end manner using artificial neural networks as our nonlinear function approximator (section 3.1.5). For a broader discussion on more explicit ways of constructing features from state observations, we refer the interested reader to R. S. Sutton and Barto (2018), chapter 9.5.

### 3.1.2 The Value Approximation Objective

We return to the predictive objective of estimating the value function of a given MRP $\mathcal{M}_\pi = \langle \mathcal{S}, r_\pi, P_\pi \rangle$ with discount factor $\gamma \in [0, 1]$. Recall from equation 2.3 the value is defined as $v_\pi(s) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s], \forall s \in \mathcal{S}$. Let $v_{\boldsymbol{\theta}}(\cdot)$ denote a parameterized and differentiable value function (linear or nonlinear) with parameters $\boldsymbol{\theta}$. We optimize $\boldsymbol{\theta}$ under a least squares framework to minimize a mean squared loss given some state $s \in \mathcal{S}$ and *target* $U$,

$$\mathcal{L}(\theta) \doteq [U - v_{\boldsymbol{\theta}}(s)]^2 . \tag{3.1}$$

Generally speaking, policy evaluation with function approximation is concerned with minimizing the squared error of the value estimate, $[v_\pi(s) - v_{\boldsymbol{\theta}}(s)]^2$, weighted by the on-policy state distribution.

**Definition 3.1.2.** The **on-policy state distribution**,[2] $\mu(s)$, describes the probability of visiting any states $s \in \mathcal{S}$ in an MRP $\mathcal{M}_\pi$,

$$\mu(s) = \mathbb{E}_\pi[P(S = s)], \quad \sum_s \mu(s) = 1. \tag{3.2}$$

[2]We assume the Markov chain is *ergodic*: any state can reach any other states in a finite number of steps. This implies the existence of the state distribution $\mu$, which is sometimes also referred to as the *stationary distribution* of the Markov chain.

The squared error prediction objective weighted by $\mu$ is referred to as the **Mean Squared Value Error** (R. S. Sutton and Barto (2018), chapter 9.2),

$$\overline{\text{VE}}(\theta) \doteq \sum_{s \in \mathcal{S}} \mu(s) \left[ v_\pi(s) - v_\theta(s) \right]^2 . \tag{3.3}$$

This is a theoretical quantity since computing it requires access to $v_\pi$—the quantity we wish to learn. Nonetheless, in diagnostic settings where we can analytically compute the true value function it can be used as a evaluation metric to measure how good some value learning method is.

In practice, we construct some target random variable in the form of a *return*, $U \equiv G$, to approximate $v_\pi$. These returns are constructed in exactly the same way as in chapter 2, namely the Monte-Carlo Return of definition 2.2.4, the (one-step) TD return of definition 2.2.5, and the $\lambda$-return of definition 2.2.7. Since any of the aforementioned ways of constructing returns are viable, we do not introduce return-specific methods for the remainder of this section. Instead, we denote the return as simply the random variable $G$ (or $G_t$ to indicate the return for a particular time-step).

From the above loss equations 3.1 and 3.3, we can interpret the original dynamic programming RL problem (of chapter 2) as an equivalent loss minimization problem. Specifically, we can do stochastic gradient descent (SGD) to minimize the mean squared loss between some return $G_t$ and the current value estimate $v_{\boldsymbol{\theta}}(s_t)$, giving rise to the general form parameter update with step-size $\alpha \in (0, 1]$,

$$\theta \leftarrow \theta + \alpha \left[ G_t - v_{\boldsymbol{\theta}}(S_t) \right] \nabla_{\boldsymbol{\theta}} v_{\boldsymbol{\theta}}(S_t) . \tag{3.4}$$

This equation is general in the sense that one can choose any method to construct the return $G_t$, any differentiable value function whose gradient is computable in the form of $\nabla_{\boldsymbol{\theta}} v_{\boldsymbol{\theta}}(\cdot)$, and any sampling distribution for how states are sampled from the environment.

Additionally, in treating RL as a loss minimization method, we can also leverage the various gradient-based optimization framework which have been well-developed

in deep supervised learning. In fact this is common practice in contemporary deep reinforcement learning. However, it would be erroneous to interpret optimization for RL as identical to optimization in supervised learning. For one, all returns that use bootstrapping (i.e. all returns *except* the Monte-Carlo return) are "semi-gradient" methods rather than gradient methods.[3] Additionally, while supervised learning assumes a fixed, pre-collected dataset, RL uses agent collected data whose distribution depend on the behavioural policy. In the case of policy improvement, the changing behavioural policy induces a non-stationary data distribution over which optimization is done.[4] While optimizers for deep supervised learning have seen success for deep reinforcement learning, the analysis and design of optimization procedures specifically suited to RL is currently still an open question (E. Bengio, Pineau, and Precup 2020; Romoff et al. 2020; E. Bengio, Pineau, and Precup 2021).

For the remainder of this section, we consider more concretely the different components of equation 3.4. Section 3.1.3 considers the setting where the function approximation $v_{\boldsymbol{\theta}}(\cdot)$ is a linear function, section 3.1.4 considers linear function approximation with the one-step TD return to theoretically analyze the fixed point of this algorithm, and section 3.1.5 considers the more general case of nonlinear value functions.

### 3.1.3 Linear Value Function

We consider the simplest approximation setting where we are given the MRP $\mathcal{M}_\pi = \langle \mathcal{S}, r_\pi, P_\pi \rangle$, discount factor $\gamma \in [0, 1]$, and the state-observations are the features such that we have the feature set $\Phi$. Given some state $s_t \in \mathcal{S}$, we represent the (approximate) value estimate of each state as a linear function of the features,

$$v_\pi(s_t) \approx v_{\boldsymbol{\theta}}(s_t) \doteq \boldsymbol{\phi}_t^\top \boldsymbol{\theta} = \sum_{i=1}^{d} \phi(s_t)_i \, \boldsymbol{\theta}_i. \tag{3.5}$$

[3] We later discuss the concept of "semi-gradient" in greater details in section 3.1.3. Also see R. S. Sutton and Barto (2018), chapter 9.3.

[4] We use *behavioural policy* to refer to the data-collection policy. In the *on-policy* case, the behavioural policy is also the policy one wishes to learn about. In the *off-policy* case, the data distribution can come from (possibly many) behavioural policies that differ from the policy one wishes to learn about.

The gradient of the linear value function (equation 3.5) for some $s_t \in \mathcal{S}$ is $\nabla_{\boldsymbol{\theta}} v_{\boldsymbol{\theta}}(s_t) = \boldsymbol{\phi}_t$. We can simply substitute this into the general update (equation 3.4) to minimize the loss between target return and current value estimate:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \left( G_t - \boldsymbol{\phi}_t^\top \boldsymbol{\theta} \right) \cdot \boldsymbol{\phi}_t \,. \tag{3.6}$$

We note that in the case of tabular ("one-hot") features, equation 3.6 reduces to the chapter 2 stochastic iterative update (equation 2.14). Here we introduce a general end-of-episode algorithm for policy evaluation with linear function approximation (algorithm 8).

**Algorithm 8:** End-of-episode policy evaluation with linear function approximation and stochastic gradients.

Input: MRP $\mathcal{M}_\pi$, feature set $\Phi$, discount factor $\gamma \in [0, 1]$, stepsize $\alpha \in (0, 1]$ ;
Initialize value parameters $\boldsymbol{\theta}^{(0)} \in \mathbb{R}^d$ ;
**while** *not converged* **do**
  Sample episode trajectory from $\mathcal{M}_\pi$: $(\boldsymbol{\phi}_0, R_1, \boldsymbol{\phi}_1, R_2, ..., \boldsymbol{\phi}_{T-1}, R_T)$ ;
  **for** *each state-feature encountered,* $\boldsymbol{\phi}_t$ **do**
    Compute return $G_t$, possibly bootstrapping using value function $v_{\boldsymbol{\theta}^{(n)}}(\cdot)$ ;
    Update: $\boldsymbol{\theta}^{(n+1)} \leftarrow \boldsymbol{\theta}^{(n)} + \alpha \left( G_t - \boldsymbol{\phi}_t^\top \boldsymbol{\theta}^{(n)} \right) \cdot \boldsymbol{\phi}_t$
  **end**
**end**

So far we have remained agnostic to how the return $G_t$ is constructed. Algorithm 8 shares a very similar form to the end-of-episode algorithms of the previous chapter (Monte-Carlo policy evaluation of algorithm 3, and $\lambda$-return policy evaluation of algorithm 5), with the main difference being the updates are for the value function parameters $\boldsymbol{\theta}$ rather than a look-up table entry of a given state. One can easily turn the algorithm into an *online* algorithm (i.e. the linear approximation version of the one-step TD algorithm) using the one-step TD return (definition 2.2.5) which only requires the one-step experience tuple $(\boldsymbol{\phi}_t, R_{t+1}, \boldsymbol{\phi}_{t+1})$.

While one can construct any target $G$ used in chapter 2, the specific target used does result in different solutions. If one uses the Monte-Carlo return (definition 2.2.4),

the update of equation 3.6 corresponds to a gradient descent update and one can find a locally optimal solution (R. S. Sutton and Barto (2018), chapter 9.3).[5] Alternatively, if the return uses *bootstrapped targets* (such as the one-step TD return, the n-step return, and the $\lambda$-return with $\lambda < 1$), we only consider the gradient with respect to the current parameters, and not the value function used to construct the target. This is referred to as **semi-gradient methods** and are not equivalent to a true gradient descent method (R. S. Sutton and Barto (2018), chapter 9.3). Intuitively, this is because bootstrapping results in a "moving target" where the current target depends in part on the current value estimate, while gradient descent method perform steepest descent with respect to a static target.[6]

### 3.1.4 Fixed Point of Linear TD(0)

We can better understand the linear TD method for policy evaluation by analyzing the dynamics of its parameters over repeated applications of the update step and in the limit. Specifically, we write the linear update (equation 3.6) with the one-step TD return $G_t^{TD} \equiv R_{t+1} + \gamma v_{\boldsymbol{\theta}}(S_{t+1})$. We abuse the notation slightly and assume that we update online, thus the $t$-th iteration of the parameters $\boldsymbol{\theta}_t$ is updated with the online samples $(\boldsymbol{\phi}_t, R_{t+1}, \boldsymbol{\phi}_{t+1})$ in the form:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \left( R_{t+1} + \gamma \boldsymbol{\phi}_{t+1}^\top \boldsymbol{\theta}_t - \boldsymbol{\phi}_t^\top \boldsymbol{\theta}_t \right) \boldsymbol{\phi}_t \,, \tag{3.7}$$

$$= \boldsymbol{\theta}_t + \alpha \left( R_{t+1} \boldsymbol{\phi}_t - \boldsymbol{\phi}_t (\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1})^\top \boldsymbol{\theta}_t \right) . \tag{3.8}$$

Notably, the terms $R_{t+1}\boldsymbol{\phi}_t$ and $\boldsymbol{\phi}_t(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})^\top$ of equation 3.8 can be viewed as Markov processes on their own. The analysis of the algorithm follows from analyzing these processes in expectation. First, we define in matrix notation the terms we utilize in the analysis.

[5] Under the usual stochastic approximation conditions of decreasing step-sizes.

[6] The interested reader may refer to (Barnard (1993), Appendix I) for a proof of how in the general case with linear function approximation, there does not exist a loss function whose direction of steepest descent correspond to the parameter update of one-step TD.

**Definition 3.1.3.** Given MRP $\mathcal{M}_\pi = \langle \mathcal{S}, r_\pi, P_\pi \rangle$, discount factor $\gamma \in [0, 1)$, and feature set $\Phi$, we define them in matrix notation (similar to definition 2.2.3). Let $\mathbf{P}_\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ be the transition matrix, $(\mathbf{P}_\pi)_{ij} = P_\pi(S' = j|S = i)$; $\mathbf{r}_\pi \in \mathbb{R}^{|\mathcal{S}|}$ be the reward vector, $(\mathbf{r}_\pi)_i = r_\pi(i)$; and $\mathbf{\Phi} \in \mathbb{R}^{|\mathcal{S}|\times d}$ be the feature matrix, whose $i$-th row denote the $d$-dimensional feature of the $i$-th state. Let $\mathbf{D} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ be a diagonal matrix whose diagonal entries are the on-policy state distribution of the MRP $\mathcal{M}_\pi$,[7]

$$\mathbf{D}_{i,i} = \mu(i)\,, \quad i \in \mathcal{S}\,. \tag{3.9}$$

We can write the parameter updates of equation 3.8 as follows, using $\mathbb{E}_{\boldsymbol{\mu}}[\cdot]$ to denote expectation under steady-state distribution,

$$\mathbb{E}_{\boldsymbol{\mu}}[\boldsymbol{\theta}_{t+1}|\boldsymbol{\theta}_t] = \boldsymbol{\theta}_t + \alpha\left(\mathbf{b}_{\boldsymbol{\theta}} - \mathbf{A}_{\boldsymbol{\theta}}\boldsymbol{\theta}_t\right)\,, \tag{3.10}$$

where $\mathbf{A}_{\boldsymbol{\theta}} \in \mathbb{R}^{d\times d}$ and $\mathbf{b}_{\boldsymbol{\theta}} \in \mathbb{R}^d$ denote matrices describing the expected updates,

$$\mathbf{A}_{\boldsymbol{\theta}} = \mathbb{E}_{\boldsymbol{\mu}}[\boldsymbol{\phi}_t(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})^\top] = \mathbf{\Phi}^\top\mathbf{D}(\mathbf{I} - \gamma\mathbf{P}_\pi)\mathbf{\Phi}\,, \tag{3.11}$$

$$\mathbf{b}_{\boldsymbol{\theta}} = \mathbb{E}_{\boldsymbol{\mu}}[R_{t+1}\boldsymbol{\phi}_t] = \mathbf{\Phi}^\top\mathbf{D}\mathbf{r}_\pi\,. \tag{3.12}$$

**Lemma 3.1.1.** *The iteration in equation 3.10 has a fixed point known as the* ***TD fixed point****:*

$$\boldsymbol{\theta}_{TD} = \mathbf{A}_{\boldsymbol{\theta}}^{-1}\mathbf{b}_{\boldsymbol{\theta}} = (\mathbf{\Phi}^\top\mathbf{D}(\mathbf{I} - \gamma\mathbf{P}_\pi)\mathbf{\Phi})^{-1}\cdot\mathbf{\Phi}^\top\mathbf{D}\mathbf{r}_\pi\,. \tag{3.13}$$

*Proof.* (Adopted from R. S. Sutton and Barto (2018), chapter 9.4) Equation 3.10 has a fixed point when $\mathbb{E}_{\boldsymbol{\mu}}[\boldsymbol{\theta}_{t+1}|\boldsymbol{\theta}_t] = \boldsymbol{\theta}_t$. This occurs when:

$$\begin{aligned}
&\mathbf{A}_{\boldsymbol{\theta}}\boldsymbol{\theta}_{TD} - \mathbf{b}_{\boldsymbol{\theta}} = 0\,,\\
\Rightarrow\ &\boldsymbol{\theta}_{TD} = (\mathbf{A}_{\boldsymbol{\theta}})^{-1}\mathbf{b}_{\boldsymbol{\theta}}\,.
\end{aligned}$$

□

[7] This is the same distribution as the one used to define equation 3.3.

We use similar methods in chapter 5 to analyze the fixed point of our newly proposed algorithms. While this is outside of the scope of this thesis, one can also show that the iteration described by equation 3.10 converges to the TD fixed point under mild assumptions.[8]

Lastly, we briefly discuss the *value error* of the TD fixed point solution. Let us first define a new *weighted quadratic norm*, $||\cdot||_{\mathbf{D}}$. This norm is weighted by the visitation probability to each states of the MRP (i.e. the on-policy distribution, $\mu(\cdot)$). Specifically for some $\mathbf{v} \in \mathbb{R}^{|\mathcal{S}|}$,

$$||\mathbf{v}||_{\mathbf{D}} \doteq \mathbf{v}^\top \mathbf{D}\, \mathbf{v} = \sum_{i=1}^{|\mathcal{S}|} \mu(i)\, \mathbf{v}_i^2 \,. \tag{3.14}$$

We can express the mean squared value error (equation 3.3) using this norm: given some linear value function parameter $\boldsymbol{\theta} \in \mathbb{R}^d$ and its resulting value estimate $\mathbf{v} = \boldsymbol{\Phi}\boldsymbol{\theta}$, the value error can be written as:

$$\overline{\text{VE}}(\theta) \doteq ||\boldsymbol{\Phi}\boldsymbol{\theta} - \mathbf{v}_\pi||_{\mathbf{D}} \,, \tag{3.15}$$

where $\mathbf{v}_\pi$ is the true value function in matrix form (definition 2.2.3).

As we typically assume there are less parameters than there are states, and that the feature matrix $\boldsymbol{\Phi} \in \mathbb{R}^{|\mathcal{S}| \times d}$ has full rank, the space of representable linear value functions, $\mathbf{v} = \boldsymbol{\Phi}\boldsymbol{\theta}$, lies in a $d$-dimensional subspace within $\mathbb{R}^{|\mathcal{S}|}$, $d << |\mathcal{S}|$. Therefore, the *true* value function, $\mathbf{v}_\pi$, may not be representable. Instead, we are interested in finding the closest approximation to the function of interest in the $d$-dimensional subspace. We define a projection matrix, $\boldsymbol{\Pi} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$, that projects onto this subspace for arbitrary vector $\mathbf{v}$,

$$||\boldsymbol{\Pi}\mathbf{v} - \mathbf{v}||_{\mathbf{D}} = \min_{\boldsymbol{\theta}} ||\boldsymbol{\Phi}\boldsymbol{\theta} - \mathbf{v}||_{\mathbf{D}} \,, \quad \forall \mathbf{v} \in \mathbb{R}^{|\mathcal{S}|} \,. \tag{3.16}$$

[8] In brief, convergence is proven by showing matrix $\mathbf{A}_{\boldsymbol{\theta}}$ is positive definite, thus repeated multiplication of the matrix $(\mathbf{I} - \alpha\mathbf{A}_{\boldsymbol{\theta}})$ (this matrix can be derived algebraically by rearranging equation 3.10) with parameters $\boldsymbol{\theta}_t$ converges with probability 1 under standard assumptions (of step-size decay, ergodicity of the MRP and feature rank). The interested reader can refer to (R. S. Sutton and Barto (2018), chapter 9.4) and (Bertsekas and Tsitsiklis (1995), chapter 6.3.3) for detailed proofs.

In other words, $\mathbf{\Pi v} \in \mathbb{R}^{|\mathcal{S}|}$ is a vector that lies in the space of representable value functions which most closely approximates $\mathbf{v}$, defined in terms of minimizing the norm $||\cdot||_{\mathbf{D}}$. We are interested in finding the closest approximation to the true value function, $\mathbf{\Pi v}_\pi$. We provide an error bound for the TD fixed point solution to this value function.

**Proposition 3.1.2.** *The TD fixed point solution, $\boldsymbol{\theta}_{TD}$, corresponds to a value function with the following bounded error,*

$$||\mathbf{\Phi}\boldsymbol{\theta}_{TD} - \mathbf{v}_\pi||_{\mathbf{D}} \leq \frac{1}{1-\gamma}||\mathbf{\Pi v}_\pi - \mathbf{v}_\pi||_{\mathbf{D}}\,. \tag{3.17}$$

*Proof.* We briefly sketch out the proof. Let $\mathbf{v}^{TD} = \mathbf{\Phi}\boldsymbol{\theta}_{TD}$ denote the TD fixed point value estimate. The proof relies on the following inequality (derivable from Bertsekas and Tsitsiklis (1995), lemma 6.9),

$$||\mathbf{v}^{TD} - \mathbf{\Pi v}_\pi||_{\mathbf{D}} \leq \gamma||\mathbf{v}^{TD} - \mathbf{v}_\pi||_{\mathbf{D}}\,. \tag{3.18}$$

We can then show,

$$\begin{aligned} ||\mathbf{v}^{TD} - \mathbf{v}_\pi||_{\mathbf{D}} &\leq ||\mathbf{v}^{TD} - \mathbf{\Pi v}_\pi||_{\mathbf{D}} + ||\mathbf{\Pi v}_\pi - \mathbf{v}_\pi||_{\mathbf{D}}\,, \\ &\leq \gamma||\mathbf{v}^{TD} - \mathbf{v}_\pi||_{\mathbf{D}} + ||\mathbf{\Pi v}_\pi - \mathbf{v}_\pi||_{\mathbf{D}}\,, \end{aligned}$$

and the result follows from rearrangement. For a detailed proof we refer the reader to (Bertsekas and Tsitsiklis (1995), lemma 6.9 and proposition 6.5). □

Intuitively, proposition 3.1.2 states that the TD fixed point $\boldsymbol{\theta}^{TD}$ and its corresponding linear value function $\mathbf{v}^{TD} = \mathbf{\Phi}\boldsymbol{\theta}_{TD}$ has a mean squared value error that is upper bounded by a multiplicative factor $(1/1-\gamma)$ of the lowest achievable value error. Notably, this bound becomes very loose for higher values of $\gamma$ (corresponding to a longer horizon problem), indicating in such cases that the TD(0) solution is not guaranteed to be good and may be quite biased. Nonetheless, TD(0) can still be

a useful algorithm as it allows for online updates and faster convergence. In chapter 5 we show that our new method has the same TD fixed point as TD(0), while demonstrating even faster convergence speed empirically.

### 3.1.5 Nonlinear Value Function

We now consider the setting where we learn a transformation from the space of state-observations to features. Again consider the MRP $\mathcal{M}_\pi = \langle \mathcal{S}, r_\pi, P_\pi \rangle$ and discount factor $\gamma \in [0, 1]$, we would like to learn a parameterized feature mapping, $\phi_\xi : \mathcal{S} \to \mathbb{R}^d$ to generate our feature set $\Phi$. This learnable mapping (sometimes referred to as a *feature encoder*) has parameters $\xi$, and we use the subscript to denote the feature mapping and feature set's dependence on it. Given a feature mapping $\phi_\xi(\cdot)$, a value function can be parameterized in the same manner as in section 3.1.3. Namely for all $s \in \mathcal{S}$,

$$v_\pi(s) \approx v_{\boldsymbol{\theta},\xi}(s) \doteq \boldsymbol{\phi}_\xi(s)^\top \boldsymbol{\theta}\,. \tag{3.19}$$

We consider the setting where the feature map $\phi_\xi$ is an artificial neural network (ANN).[9] Specifically, $v_{\boldsymbol{\theta},\xi}(\cdot)$ is a multi-layer ANN, where $\boldsymbol{\theta}$ describes the parameters of the final linear layer for value prediction, and $\xi$ describes all the parameters *up to* the final layer, which we consider to be the nonlinear "feature encoder". This is somewhat of an arbitrary distinction as we can easily pick another layer of the ANN to be called the "feature layer", though much of our later results rely on value being linear with respect to the features, so we find this distinction to be both useful and powerful as we do not limit the expressivity of the nonlinear feature encoder.[10]

In practice, the ANN describing $v_{\boldsymbol{\theta},\xi} : \mathcal{S} \to \mathbb{R}$ is trained in an end-to-end manner to minimize the loss (equation 3.1). The update follows the same form as equation 3.4 for both parameters $\boldsymbol{\theta}, \xi$. Given some state observation $s \in \mathcal{S}$, the value gradient

[9] The use of multi-layer ANNs for function approximation in reinforcement learning is referred to as **deep reinforcement learning (DRL)**, for an introduction see François-Lavet et al. (2018).

[10] While outside of the scope of this thesis, there are works on nonlinear (value) functions of features, with theoretical guarantees, such as the two timescale analysis of Chung et al. (2018).

($\nabla_{\boldsymbol{\theta}} v_{\boldsymbol{\theta},\xi}(s) = \phi_\xi(s)$) and feature encoder gradient ($\nabla_\xi v_{\boldsymbol{\theta},\xi}(s)$) are both computable using standard backpropagation (Rumelhart, G. E. Hinton, and Williams 1985). We sketch out an algorithm for policy evaluation with ANN-based nonlinear function approximation below.

**Algorithm 9:** End-of-episode policy evaluation with nonlinear function approximation and stochastic gradients.

Input: MRP $\mathcal{M}_\pi$, discount factor $\gamma \in [0, 1]$, stepsizes $\alpha_\xi \in (0, 1]$, $\alpha_{\boldsymbol{\theta}} \in (0, 1]$ ;
Initialize encoder and value parameters $\xi^{(0)}, \boldsymbol{\theta}^{(0)}$ ;
**while** *not converged* **do**
  Sample episode trajectory from $\mathcal{M}_\pi$: $(S_0, R_1, S_1, R_2, ..., S_{T-1}, R_T)$ ;
  **for** *each state-observation encountered,* $S_t$ **do**
    Compute return $G_t$, possibly bootstrapping using current value estimate $v_{\boldsymbol{\theta}^{(n)},\xi^{(n)}}(\cdot)$ ;
    Update value parameter:
    $\boldsymbol{\theta}^{(n+1)} \leftarrow \boldsymbol{\theta}^{(n)} + \alpha \left(G_t - \boldsymbol{\phi}_{\xi^{(n)}}(S_t)^\top \boldsymbol{\theta}^{(n)}\right) \cdot \boldsymbol{\phi}_{\xi^{(n)}}(S_t)$ ;
    Update feature encoder parameters:
    $\xi^{(n+1)} \leftarrow \xi^{(n)} + \alpha \left(G_t - \boldsymbol{\phi}_{\xi^{(n)}}(S_t)^\top \boldsymbol{\theta}^{(n)}\right) \cdot \nabla_{\xi^{(n)}} v_{\boldsymbol{\theta}^{(n)},\xi^{(n)}}$ ;
  **end**
**end**

Algorithm 9 is nearly identical to algorithm 8 with the exception of needing to perform feature encoder parameter updates. It should be noted that algorithm 9 is just a sketch for the idea of nonlinear policy evaluation with ANNs. In practice, policy evaluation with a bootstrapped return and nonlinear architecture is *not* guaranteed to converge (see Bertsekas and Tsitsiklis (1995), chapter 6.3.2 and example 6.6). Often, additional "tricks" are required to stably train ANN-based value functions (we outline some of the tricks for nonlinear ANN-based control in section 3.2.2).

## 3.2 Policy Improvement with Function Approximation

Having extended policy evaluation to the function approximation setting, we now do the same for policy improvement.

### 3.2.1 Fitted Q Iteration

So far, we have introduced approximation algorithms using single examples in minimizing a loss between some constructed target and the current value prediction. We now consider the "batch-mode" setting in which we minimize this loss using batches or mini-batches of examples. In essence, we consider Q-learning (section 2.3.2) and train it by regressing using batches of data. This was proposed by Ernst, Geurts, and Wehenkel (2005) who combined it with tree-based supervised regression models. We illustrate a general form of their algorithm below.

**Algorithm 10:** Fitted Q Iteration Algorithm, adopted from Ernst, Geurts, and Wehenkel (2005)

**Input**:
  Discount factor $\gamma \in [0, 1]$ ;
  Batch of experience, $\{(S_k^{(i)}, A_k^{(i)}, R_{k+1}^{(i)}, S_{k+1}^{(i)}), i = 1, ..., m\}$;
  Parameterized action-value prediction function $q_{\boldsymbol{\theta}}(\cdot, \cdot)$ ;
  Parameterized action-value target function $q_{\boldsymbol{\theta}^-}(\cdot, \cdot)$ ;
  Regression loss function $\texttt{loss}(\cdot, \cdot)$ and Optimizer $\texttt{optim}(\cdot)$ ;
**while** *stopping condition not reached* **do**
  Compute target for each example $i = 1, ..., m$,
    $U^{(i)} = R_{k+1}^{(i)} + \gamma \max_{a'} q_{\boldsymbol{\theta}^-}(S_{k+1}^{(i)}, a')$ ;
  Compute batch regression loss,
    $\mathcal{L}_Q = \sum_{i=1}^{m} \texttt{loss}(\ U^{(i)}, q_{\boldsymbol{\theta}}(S_k^{(i)}, A_k^{(i)})\ )$ ;
  Parameter update to minimize batch regression loss,
    $\boldsymbol{\theta} \leftarrow \texttt{optim}(\mathcal{L}_Q)$ ;
**end**

### 3.2.2 Nonlinear Control with Deep Q Network

While Fitted Q Iteration (Ernst, Geurts, and Wehenkel 2005) is in theory compatible with arbitrary function approximation architectures of $q_{\boldsymbol{\theta}}$, and subsequent development realized a combination of the Fitted Q Iteration algorithm with artificial neural networks (Riedmiller 2005), the above methods typically relied on training from scratch their $q$ functions at each iteration using the full set of experiences collected up to the current point. While training is stable, such methods are inefficient and do not scale to large models. In contrast, (Mnih et al. 2015) sought to develop a method of stably training deep neural networks without weight reset.

Concretely, the challenge of training deep neural net (and more generally, nonlinear function approximators for control) is that weight updates for a single state-action pair can arbitrarily change the prediction of other state-action pairs, leading to instability or divergence (Riedmiller 2005). When paired with bootstrapping, this further results in a highly unstable target. (Mnih et al. 2015) sought to address such issues in two ways. First, a *replay buffer* of previous experience is introduced, and sampling mini-batches uniformly from the replay buffer reduces the correlation between individual data. Second, a *frozen target network* is used, which removes the correlation between the current $q$ estimate and the bootstrap target, in addition to providing a more stable optimization objective.

We illustrate the training procedure for the Deep Q network (Mnih et al. 2015) above in algorithm 11, where the network $q_{\boldsymbol{\theta}}$ is a deep convolutional neural network to process high-dimensional pixel input. At its core is the same fitted-Q iteration procedure as algorithm 10, though we have added the additional "tricks" of target network and uniform sampling from experience replay buffer to achieve stable training. This algorithm achieved human-level performance in the 49-game Atari 2600 environment (Bellemare et al. 2013).

**Algorithm 11:** Deep Q Network Training, adopted from Mnih et al. (2015)

**Input**:
  MDP $\mathcal{M}$ and discount factor $\gamma \in [0, 1]$ ;
**Initialize**:
  Policy network $q_{\boldsymbol{\theta}}$ and target network $q_{\boldsymbol{\theta}^-}$, $\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$ ;
  Experience replay buffer $\mathcal{B} = \{\emptyset\}$ ;
**while** *policy not good enough* **do**
  ▷ `Take single environmental step`
  Act in the environment (e.g. $\epsilon$-greedy based on $q_{\boldsymbol{\theta}}(S_t, \cdot)$) to collect experience $(S_t, A_t, R_{t+1}, S_{t+1})$ ;
  Store experience to buffer, $\mathcal{B} = \{\mathcal{B} \cup (S_t, A_t, R_{t+1}, S_{t+1})\}$ ;
  ▷ `Sample mini-batch data`
  Sample uniformly from buffer $\{(S_k^{(i)}, A_k^{(i)}, R_{k+1}^{(i)}, S_{k+1}^{(i)}),\ i = 1, ..., m\} \sim \mathcal{B}$ ;
  ▷ `Fitted Q Iteration`
  Run algorithm 10 for single iteration, given inputs:
    Discount factor $\gamma$ ;
    Mini-batch $\{(S_k^{(i)}, A_k^{(i)}, R_{k+1}^{(i)}, S_{k+1}^{(i)}),\ i = 1, ..., m\}$ ;
    Prediction function $q_{\boldsymbol{\theta}}$ and target function $q_{\boldsymbol{\theta}^-}$ ;
    Mean squared error loss with RMSProp optimizer (Tieleman and G. Hinton 2012) ;
  ▷ `Target network updates`
  For every pre-specified number of training steps, update target net parameter $\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$
**end**

## 3.3 Successor Representation of States

Up to this point, feature representations are based on *observation similarity* of a *single* time-step: we are either given a feature observation directly at each time-step, or, we are given an observation and map it to a feature. We now develop representations that encode information about the future, in the form of temporally-dependant, predictive representations about states an agent will likely encounter in the future.

Specifically, we are interested in developing representations that are useful for value learning. Let us first return to the simplest setting: policy evaluation with tabular

states (introduced in section 2.2), given MRP $\mathcal{M}_\pi = \langle \mathcal{S}, r_\pi, P_\pi \rangle$ and discount factor $\gamma \in [0, 1)$. We consider the form of the value function estimate, which is previously stored simply as a $|\mathcal{S}|$-dimensional vector, $\mathbf{v} \in \mathbb{R}^{|\mathcal{S}|}$, whose i-th entry refers to the value estimate for the i-th state, $s_i \in \mathcal{S}$ (see section 2.2.1). The value estimate of each state contains *temporal information*, as it estimates the total amount of reward the agent expects to receive in the future when starting from that state. Through the process of policy evaluation, the tabular value estimate converges to the true value function, which can be written in the following matrix form (theorem 2.2.1):

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

We see that the tabular value function is the product of two quantities: the instantaneous reward vector $\mathbf{r}_\pi \in \mathbb{R}^{|\mathcal{S}|}$, and an $|\mathcal{S}| \times |\mathcal{S}|$ matrix:

$$\mathbf{M}_\pi \doteq (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1}. \tag{3.20}$$

We refer to $\mathbf{M}_\pi$ as the **successor representation matrix**. Notably, all the temporal information in the value function is contained within $\mathbf{M}_\pi$, as $\mathbf{r}_\pi$ is simply the immediate reward. This is further demonstrated by interpreting the matrix via the following identity.

**Proposition 3.3.1.** *Given* $0 \leq \gamma < 1$*, the inverse of* $(\mathbf{I} - \gamma \mathbf{P}_\pi)$ *exists and can be written as a Neumann series:* [11]

$$(\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} = \lim_{n \to \infty} \sum_{n=0}^{N} (\gamma \mathbf{P}_\pi)^n. \tag{3.21}$$

*Proof.* See (Puterman (1994), Theorem C.2). □

Let $\mathbf{M}_\pi(s_i, s_j)$ denote the i-th row and j-th column of matrix $\mathbf{M}_\pi$. Following proposition 3.3.1, we can interpret the entries of matrix as follows:

$$\mathbf{M}_\pi(s_i, s_j) = \sum_{n=0}^{\infty} \gamma^n P_\pi(S_n = s_j | S_0 = s_i), \tag{3.22}$$

[11] A Neumann series is generalization of the geometric series using matrices.

where $P_\pi(S_n = s_j | S_0 = s_i)$ is the probability of ending up in state $s_j$ when starting from state $s_i$ and taking $n$ steps while following policy $\pi$. In other words, equation 3.22 represents the *total discounted future occupancy* between all pairs of states in $\mathcal{S}$.

Dayan (1993) proposes using the i-th *row* of the matrix $\mathbf{M}_\pi$ as the representation for the i-th state. This method of state representation is referred to as the **successor representation (SR)**, $\mathbf{M}_\pi(s_i, \cdot) \in \mathbb{R}^{|\mathcal{S}|}$. In this tabular setting, the SR represents each state as a $|\mathcal{S}|$-dimensional vector whose entries are the expected (discounted) future occupancy of all states. We can compute the value of a given state $s_i$ by:

$$\mathbf{v}_\pi(s_i) = \mathbf{M}_\pi(s_i, \cdot) \cdot \mathbf{r}_\pi. \tag{3.23}$$

Thus, if the SR is given for a MRP (or can be pre-computed), the value learning problem turns into just supervised learning of $\mathbf{r}_\pi$, which Dayan (1993) conjectures is a lower-variance estimation problem, since learning $\mathbf{r}_\pi$ is not a temporal learning problem.

### 3.3.1 Learning tabular SR

Generally, the SR is not available to an agent, although it can be learned in a similar way as a value function (section 2.2). Consider the following identity which follows from equation 3.20,

$$\mathbf{M}_\pi = (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} \quad \Rightarrow (\mathbf{I} - \gamma \mathbf{P}_\pi)\mathbf{M}_\pi = \mathbf{I}\,, \tag{3.24}$$

$$\Rightarrow \mathbf{M}_\pi = \mathbf{I} + \gamma \mathbf{P}_\pi \mathbf{M}_\pi\,. \tag{3.25}$$

Similar to equation 2.9 in chapter 2, we can define the successor representation evaluation operator, $L_{\mathbf{M}_\pi} : \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$,

$$L_{\mathbf{M}_\pi}\mathbf{M} = \mathbf{I} + \gamma \mathbf{P}_\pi \mathbf{M}\,. \tag{3.26}$$

We observe from equations 3.25 and 3.26 that $\mathbf{M}_\pi$ is the fixed point of $L_{\mathbf{M}_\pi}$. Furthermore, we can show that the operator is a contraction using a similar proof as

proposition 2.2.3. It follows from the Banach Fixed-Point Theorem (theorem 2.2.2) that repeated application of the operator $L_{\mathbf{M}_\pi}$ to an arbitrarily initialized matrix $\mathbf{M}^{(0)}$ converges to the fixed point $\mathbf{M}_\pi$. This gives us an iterative dynamic-programming (DP) algorithm to solve for the successor representation matrix (algorithm 12), similar to the DP algorithm for value function.

**Algorithm 12:** Iterative DP solution for solving the tabular successor representation matrix

```
Input: MRP parameters P_π, γ ;
Initialize M^(0) arbitrarily ;
for k=1,2,... until satisfied do
    M^(k) ← I + γ P_π M^(k−1)
end
```

It follows that the stochastic approximation methods from section 2.2 can be also applied here to solve $\mathbf{M}_\pi$. We outline one such method here which is analogous to the one-step, TD(0) algorithm for value function learning (algorithm 4). Denoting $\mathbf{M}(s,\cdot) \in \mathbb{R}^{|\mathcal{S}|}$ as the successor representation vector for state $s$ (i.e. a row vector in $\mathbf{M}$), $\mathbb{1}_s = [0,...,1,...,0]^\top \in \mathbb{R}^{|\mathcal{S}|}$ as an indicator vector for $s$ (1 for state index $s$ and 0 everywhere else), $\alpha \in (0,1]$ as the step-size, and $P_\pi$ as the MRP transition function. We can write the iterative DP update (algorithm 12) for each of its row-vector in small step-size form:

$$\mathbf{M}^{(k+1)}(s,\cdot) = (1-\alpha)\,\mathbf{M}^{(k)}(s,\cdot) + \alpha\left[\mathbb{1}_s + \gamma \sum_{s'} P_\pi(s'|s)\mathbf{M}^{(k)}(s',\cdot)\right]. \tag{3.27}$$

If we do not have access to the transition function $P_\pi$, we can sample $S_{t+1} \sim P_\pi(\cdot|S_t)$ to construct a random variable as the update target. In fact, we can write in component form the update for *any* arbitrary state-pairs $S_t, S_k \in \mathcal{S}$,

$$\mathbf{M}^{(k+1)}(S_t,S_k) \leftarrow \mathbf{M}^{(k)}(S_t,S_k) + \alpha\left(\mathbb{1}_{S_t=S_k} + \gamma\mathbf{M}^{(k)}(S_{t+1},S_k) - \mathbf{M}^{(k)}(S_t,S_k)\right), \tag{3.28}$$

where $\mathbb{1}_{S_t=S_k} = 1$ if and only if $S_t = S_k$ and is otherwise 0.[12]

[12]The indicator vector $\mathbb{1}_{S_t=S_k}$ is a random variable whose expectation is equal to the probability of $S_t = S_k$.

We provide a stochastic approximation algorithm for one-step SR learning below.

**Algorithm 13:** Online incremental TD(0) for tabular SR matrix learning

Input: MRP $\mathcal{M}_\pi$, discount factor $\gamma \in [0, 1]$, step-size $\alpha \in (0, 1]$ ;
Initialize SR matrix estimate $\mathbf{M}^{(0)}$ arbitrarily ;
**while** *not converged* **do**
  Sample environment $\mathcal{M}_\pi$, receive one-step experience tuple $(S_t, S_{t+1})$ ;
  Compute update target $\mathbf{u}_t = \mathbb{1}_{S_t} + \gamma \mathbf{M}^{(k)}(S_{t+1}, \cdot)$ ;
  Update: $\mathbf{M}^{(k+1)}(S_t, \cdot) \leftarrow \mathbf{M}^{(k)}(S_t, \cdot) + \alpha \left( \mathbf{u}_t - \mathbf{M}^{(k)}(S_t, \cdot) \right)$ ;
**end**

The similarity between this algorithm and algorithm 4 for value learning is apparent. Indeed, the SR shares a similar form as the value function, only that the SR for state $s$ is a vector of dimension $|\mathcal{S}|$, while the value is a scalar. It should be noted that estimating the value function through learning SR and reward separately is no faster than directly learning the value using TD.

### 3.3.2 SR for Non-Tabular State Observations

What happens if we are given state observations rather than (tabular) state indices? We see that learning the SR matrix $\mathbf{M}_\pi$ becomes highly non-trivial in the case of function approximation. Specifically, consider the TD update for SR learning (equation 3.28), which relies on the indicator function $\mathbb{1}_{s_t = s_k}$. The comparison $\{s_t = s_k\}$ is trivial in the tabular case (it is simply a comparison of state identities). However, in the high-dimensional setting, if the state observations are noisy, or continuous, then one may never encounter exactly the same observation twice. Thus, $\mathbb{1}_{s_t = s_k}$ is a "pseudo-reward" for SR learning that is *infinitely sparse* for many non-trivial settings.

One solution for learning SRs in the high-dimensional setting is using *successor features* (SFs, addressed later in section 3.4.2). In fact, SFs retain Dayan (1993)'s original motivation of "summarizing" a state by its cumulative future discounted states (or in this case, state features). However, unlike the *successor representation matrix*, $\mathbf{M}_\pi$, SFs do not provide the ability to *query* future occupancy between arbitrary state-

pairs. This is a desirable ability in various settings, such as for the methods in chapter 4.

As it stands, this thesis does not tackle the problem of learning the SR matrix in continuous state spaces, and currently the results of chapter 4 are limited to the tabular setting. However, exciting recent works have began to tackle the question of continuous-state SRs. As continuous states cannot be represented in matrix form $\mathbf{M}_\pi$, such works treat the problem as learning a density function over all pairs of states, given state pairs $s, s' \in \mathcal{S}$ and some normalizing scalar constant $\eta$,

$$\hat{m}(s, s') \approx \eta \sum_{n=0}^{\infty} \gamma^n P_\pi(S_n = s' | S_0 = s). \tag{3.29}$$

A number of approaches have been taken. For instance, Janner, Mordatch, and Levine (2020) treats equation 3.29 as a distribution learning problem and leverage recent tools from generative modelling. Blier, Tallec, and Ollivier (2021) bypasses the infinitely-sparse pseudo-reward issue by deriving a non-sparse gradient for the same update.

## 3.4 Successor Representation of Features

We saw in section 3.3.1 that the temporal difference algorithms can be applied to both value function learning and SR learning. The difference between the two is minute: a state's SR can be thought of as a "generalized" value function, if each state's "reward" is a $|\mathcal{S}|$-dimensional indicator vector, $\mathbb{1}_s$. We develop this idea of *general value function* further in this section, where we can learn value functions for arbitrary, multi-dimensional pseudo-rewards (section 3.4.1). We then show one specific form of general value functions that uses the current state-features as the pseudo-reward, the *successor features* (section 3.4.2).

### 3.4.1 General Value Function

Generally, we can learn *arbitrary* "pseudo-value" functions for any arbitrarily defined "pseudo-rewards". For instance, if the "pseudo-reward" is the state index vector, the "pseudo-value" function is the successor representation (section 3.3.1). Formally, we refer to all such functions as **general value functions (GVFs)** (R. S. Sutton, Modayil, et al. 2011), and the "pseudo-rewards" as **cumulants** to generalize it from the actual reward given as a part of the MDP.

Akin to the actual reward, the cumulant at time $t$ is a (possibly multi-dimensional) random variable, $C_t$. We can define the GVF as follows,

$$v(s;\pi,\gamma) = \mathbb{E}_\pi\left[\sum_{t=1}^{\infty}\gamma^{t-1}C_t \;\middle|\; S_0 = s\right]. \tag{3.30}$$

GVFs can be viewed as a form of predictive *knowledge representation* (R. S. Sutton, Modayil, et al. 2011; Schlegel, Jacobsen, et al. 2021). For instance, a (standard) value function is the answer to the question "how much total reward will I receive from this point onward". More generally, a GVF stores the answer to the question: "how much of $C$ will occur over the next $T$ steps?". To demonstrate this formally, we first show the relationship between discount factor and termination probabilities.

**Lemma 3.4.1.** *A discounted, infinite-horizon value function (equation 3.30) can be equivalently written as a undiscounted, finite-horizon value function with random termination lengths,*

$$v(s;\pi,T) = \mathbb{E}_\pi\left[\mathbb{E}\left[\sum_{t=1}^{T}C_t \;\middle|\; S_0 = s\right]\right], \tag{3.31}$$

*where $T \sim Geom(1-\gamma)$ is a random variable sampled from a geometric distribution with success (i.e. termination) probability $1-\gamma$ and support over $n \in \{1,2,3,...\}$.*

*Proof.* Adapted from Puterman (1994), Proposition 5.3.1,

$$\begin{aligned}
\mathbb{E}_\pi\left[\mathbb{E}\left[\sum_{t=1}^{T} C_t\right]\right] &= \mathbb{E}_\pi\left[\sum_{n=1}^{\infty}\gamma^{n-1}(1-\gamma)\sum_{t=1}^{n} C_t\right] && \text{[Geometric p.m.f.]}\\
&= \mathbb{E}_\pi\left[\sum_{t=1}^{\infty}\sum_{n=t}^{\infty} C_t(1-\gamma)\gamma^n\right]\\
&= \mathbb{E}_\pi\left[\sum_{t=1}^{\infty} C_t(1-\gamma)\gamma^{t-1}\sum_{n=1}^{\infty}\gamma^{n-1}\right]\\
&= \mathbb{E}_\pi\left[\sum_{t=1}^{\infty}\gamma^{t-1}C_t\right] && \text{[via } \sum_{n=1}^{\infty}\gamma^{n-1} = \frac{1}{1-\gamma}\text{]}\\
&= v(s;\pi,\gamma).
\end{aligned}$$

Note all quantities above are conditioned on $(S_0 = s)$ which we omit for brevity. □

The sequence length random variable $T$ (equation 3.31) is geometrically distributed with mean $\mathbb{E}[T] = \frac{1}{1-\gamma}$. Take the example of the successor representation: the cumulant is the state index vector $\mathbb{1}_s$, and assume a discount factor of $\gamma = 0.99$, $\frac{1}{1-0.99} = 100$. The SR matrix entry, $\mathbf{M}_\pi(s_i, s_j)$, is therefore the answer to the question: "starting from state $s_i$, how many times do I expect to see state $s_j$ over the next 100 steps?".

### 3.4.2 Successor Features

We now introduce a specific type of GVFs which use the state-features as the cumulant: the **successor feature (SFs)**. SFs can be thought of as a feature-based generalization of the SR; whereas SR requires explicit state index to work, the SF is designed to work with arbitrary $d$-dimensional state feature vectors. This offers a solution for extending SRs to continuous, high-dimensional settings. Concretely, given an MRP with features $\boldsymbol{\phi} \in \Phi$, we define the SF as follows,

$$\boldsymbol{\psi}_\pi(s) \doteq \mathbb{E}_\pi\left[\sum_{n=0}^{\infty}\gamma^n \boldsymbol{\phi}_{t+n} \,\middle|\, S_t = s\right]. \tag{3.32}$$

We show how the SF can be used to approximate the value (equation 2.3). First we define a linearly parameterized reward function, $r_{\mathbf{w}} : \mathbb{R}^d \to \mathbb{R}$, with **instantaneous**

**reward parameters w**,

$$\mathbb{E}_\pi[R_{t+1}|S_t = s_t] \approx r_{\mathbf{w}}(s_t) = \boldsymbol{\phi}_t^\top \mathbf{w}\,. \tag{3.33}$$

The reward function approximates the expected immediate reward. The following result follows.

**Remark 3.4.2.** *Given feature set $\Phi$, the value function for all states $s \in \mathcal{S}$ can be written as a linear combination of the successor feature $\boldsymbol{\psi}_\pi(s) \in \mathbb{R}^d$ and the instantaneous reward parameters $\mathbf{w} \in \mathbb{R}^d$:*

$$v_\pi(s) \approx \boldsymbol{\psi}_\pi(s)^\top \mathbf{w}\,. \tag{3.34}$$

*Proof.* By moving out the (cumulative) features from the reward parameter,

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[\textstyle\sum_{n=0}^{\infty} \gamma^n R_{t+n+1}|S_t = s]\,, \\ &\approx \mathbb{E}_\pi[\textstyle\sum_{n=0}^{\infty} \gamma^n \boldsymbol{\phi}_{t+n}^\top \mathbf{w}|S_t = s]\,, \\ &= \mathbb{E}_\pi[\textstyle\sum_{n=0}^{\infty} \gamma^n \boldsymbol{\phi}_{t+n}|S_t = s]^\top \mathbf{w} = \boldsymbol{\psi}_\pi(s)^\top \mathbf{w}\,. \end{aligned}$$

□

We refer to this linearly decomposed way of writing the value function (equation 3.34) as the **SF value function**, which also helps to distinguish it from the undecomposed *"model-free"* value function (equation 3.5). The fact that the value function can be written in the above decomposed form is perhaps unsurprising as the successor representation can reconstruct the value function in a similar way ($\mathbf{v}_\pi = \mathbf{M}_\pi \mathbf{r}_\pi$). The SFs can be thought of as a linear function approximation extension to the SR, much like how the linear value function extends the tabular value function. Indeed, in the case of tabular one-hot feature representations, SFs reduce to SRs exactly.

In general, learning a SF value function is no faster than learning a "model-free" value function (i.e. parameterized according to equation 3.5), thus it is unclear

**Algorithm 14:** Online one-step policy evaluation with SF value function.

**Input**: MRP $\mathcal{M}_\pi$, feature set $\Phi$, discount factor $\gamma \in [0, 1]$, step-sizes $\alpha_{\boldsymbol{\Xi}}, \alpha_{\mathbf{w}} \in (0, 1]$ ;
**Initialize:**
  SF function $\boldsymbol{\psi}_{\boldsymbol{\Xi}} : \mathbb{R}^d \to \mathbb{R}^d$ with parameters $\boldsymbol{\Xi}$ ;
  Reward function $r_{\mathbf{w}} : \mathbb{R}^d \to \mathbb{R}$ with parameters $\mathbf{w}$, $r_{\mathbf{w}}(\boldsymbol{\phi}_t) = \boldsymbol{\phi}_t^\top \mathbf{w}$ ;
**while** *not converged* **do**
  Sample environment $\mathcal{M}_\pi$, receive one-step experience tuple $(\boldsymbol{\phi}_t, R_{t+1}, \boldsymbol{\phi}_{t+1})$ ;
  ▷ `Update SF function parameters`
  Construct target: $\mathbf{u}_t = \boldsymbol{\phi}_t + \gamma \boldsymbol{\psi}_{\boldsymbol{\Xi}}(\boldsymbol{\phi}_{t+1})$ ;
  Update: $\boldsymbol{\Xi} \leftarrow \boldsymbol{\Xi} + \alpha_{\boldsymbol{\Xi}} \left(\mathbf{u}_t - \boldsymbol{\psi}_{\boldsymbol{\Xi}}(\boldsymbol{\phi}_t)\right) \nabla_{\boldsymbol{\Xi}} \boldsymbol{\psi}_{\boldsymbol{\Xi}}(\boldsymbol{\phi}_t)$ ;
  ▷ `Update reward function parameters`
  Update: $\mathbf{w} \leftarrow \mathbf{w} + \alpha_{\mathbf{w}} \left(R_{t+1} - \boldsymbol{\phi}_t^\top \mathbf{w}\right) \boldsymbol{\phi}_t$ ;
**end**
Compute value estimate as: $v_{\boldsymbol{\Xi},\mathbf{w}}(S_t) = \boldsymbol{\psi}_{\boldsymbol{\Xi}}(\boldsymbol{\phi}_t)^\top \mathbf{w}$ ;

whether *naively* learning a SF value function is at all useful within the single-task setting. Alternatively, one area in which SFs have been extensively used is the *multi-task* and *transfer* settings. For instance, Zhang et al. (2017) and Lehnert, Tellex, and Littman (2017) leverage the decoupled transition and reward components of the SF value function to transfer to similar dynamics and rewards by using the previously learned SF as the initialization of the new SF. Alternatively, Barreto, Dabney, et al. (2017) propose a more principled approach of *Generalized Policy Improvement*: a way of exploiting the fast policy evaluation of the SF value function given arbitrary SF and reward parameters to do policy improvement over *sets* of SFs, allowing for faster transfer to new reward functions (Barreto, Borsa, et al. 2018; Hansen, Dabney, Barreto, Warde-Farley, et al. 2019; Grimm et al. 2019). Adding to this, transfer can occur via generalization in the parameters of the function approximator (i.e. by including a task description as input to the function approximator), which is leveraged by works such as Borsa et al. (2018). Additionally, a learned SF contains useful information of the MRP, which have been used for option discovery (Machado, Bellemare, and Bowling 2017; Machado, Rosenbaum, et al. 2018) and better exploration (Janz et al.

2019; Machado, Bellemare, and Bowling 2020).

The current thesis takes a completely orthogonal, yet complementary approach to all previously proposed uses of successor-like representations. We investigate the use of SRs (chapter 4) and SFs (chapter 5) *for single task value learning*. Indeed, as we will show, when used in an informed way SRs and SFs can be leveraged for efficient credit assignment for value function learning.

## 3.5 SRs in the brain

There has been a recent surge in interest in the neuroscience community to interpret neuroscientific findings from the perspective of the brain encoding successor representations. This surge in interest was instigated by Stachenfeld, M. Botvinick, and Gershman (2014) and Stachenfeld, M. M. Botvinick, and Gershman (2017), who propose a unifying explanation for the activity patterns of the cells in the *hippocampus* region of the brain, based on previous experimental results in animals and humans. Specifically, the authors propose that individual hippocampal "place cells" correspond to specific (discrete) states in the environment: when an animal is in a state $s_i$, the firing rate of a hippocampal place cell encodes the entry of the SR matrix $\mathbf{M}_\pi(s_i, s_j)$ (where $s_j$ is the state encoded by that cell). Correspondingly, the *population activity* of the hippocampus encodes the SR of the current state, $\mathbf{M}_\pi(s_i, \cdot)$. Under this model, the authors offer explanations for why the activity field of a place cell can become distorted in space based on the transition probability between states (e.g. if an animal is trained to run in a specific direction the activity field skews against the direction of travel, and if there are barriers in the environment the activity field distorts around the boundaries), and how such activity can also exist for non-spatial tasks. Such observations were not explainable by previous theories of place cells that claim a place cell simply encode a specific spatial location. Further, the authors re-interpret the firing activity of "grid cells" in the nearby entorhinal cortex region of the brain. These

"grid cells" are hypothesized to be an eigendecomposition of the SRs, which give rise to grid-like activity patterns in spatial tasks. In a similar vein, Russek et al. (2017) compares simulated learning outcomes with the SR to previous experimental data in neuroscience. It was suggested that having a SR value function, along with a learned transition model for Dyna-like replay (R. S. Sutton 1990), provided the closest fit in terms of the tasks solvable by animals.

Additionally, Momennejad, Russek, et al. (2017) empirically evaluated human's ability to transfer to small changes in the reward and/or transition structure of the task. It was found that humans transfer well to tasks that only involve re-evaluating the reward, and marginally worse to settings where the transition structure changes (either as a result of the underlying MDP transition changing, or when the new optimal policy results in a different transition structure). The authors compare this finding to three hypothesis models: a "model-based learner" using value iteration (algorithm 6), a "SR learner" which learns the Q value using information collected only from the most recently experienced trial, and a "hybrid learner" that selects actions based on a mixture of the Q values of the two previous ("model-based" and "SR") learners. The "hybrid learner" offered the closest fit to human performance, which the authors claim supported SR as a computational substrate for human decision making.

Finally, the review papers Gershman (2018) and Momennejad (2020) cite the aforementioned works as evidence for SR being learned in the brain. Overall, the primary argument for the brain learning SR follows from the claim that doing RL with one-step transition models ("model-based RL") is computationally expensive as the policy needs to be dynamically computed during behavioural time, while directly estimating the action-value function (e.g. Q learning) is inflexible to changes in the environment. According to this argument, the SR is a potentially good "middle ground" for biological organisms to balance between flexibility and efficiency. Overall, though, little of the work in neuroscience has explored the possibility that SRs may also be useful for value learning in-and-of-itself, as opposed to just re-evaluating after reward

function changes. Thus, our results in this thesis also speak to additional potential explanations for the apparent presence of SR-like functions in the mammalian brain.

# 4

# Predictive Error Propagation

Value function learning for policy evaluation (section 2.2) involves constructing a *return* for each encountered state, which is used as a target. Many returns are available, such as the one-step TD return (definition 2.2.5), and the $\lambda$-return (definition 2.2.7). Typically, learning occurs by minimizing the error between the current return and the current value estimate.

In this chapter, we show how errors from *arbitrary* states can also be used to update the value estimate of the current state. This allows for a single error signal to be "re-used" multiple times in a form of efficient, non-local credit assignment. Specifically in an MRP, we show the contribution of the current state to errors in other states is proportional to the visitation count to the other states, when starting from the current state. This is analogous to the SR (section 3.3), which we leverage to simultaneously estimate environmental transition structure and propagate errors. We focus on better prediction (i.e. policy evaluation) only, and all of our results is in the tabular feature representation setting where SRs can be tractably learned.

## 4.1 The Lambda Return Error

Recall the lambda return (definition 2.2.7) which is a $\lambda$ weighted exponential average of all n-step returns, $\lambda \in [0, 1]$,

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \left[ (\sum_{k=1}^{n} \gamma^{k-1} R_{t+k}) + \gamma^n v_{\boldsymbol{\theta}}(S_{t+n}) \right] .$$

The $\lambda$-return is motivated from the perspective of efficient credit assignment. That is, it is able to provide a good bias variance trade off in the return—the target used to update the value estimate. The value estimate is updated to minimize the MSE loss $\mathcal{L}(\theta) = \left[ G_t^\lambda - v_{\boldsymbol{\theta}}(S_t) \right]^2$ (a special case of equation 3.1), with the SGD update taking the form of (a special case of equation 3.4),

$$\theta \leftarrow \theta + \alpha \left[ G_t^\lambda - v_{\boldsymbol{\theta}}(S_t) \right] \nabla_{\boldsymbol{\theta}} v_{\boldsymbol{\theta}}(S_t) \, . \tag{4.1}$$

In particular we are interested in the term $G_t^\lambda - v_{\boldsymbol{\theta}}(S_t)$, which we refer to as the **lambda return error**. Notably, using the identity in equation 2.25, we can write this term as follows,

$$G_t^\lambda - v_{\boldsymbol{\theta}}(S_t) = \sum_{n=0}^{\infty} (\lambda\gamma)^n \delta_{t+n} \, , \tag{4.2}$$

where $\delta_k = R_{k+1} + \gamma v_{\boldsymbol{\theta}}(S_{k+1}) - v_{\boldsymbol{\theta}}(S_k)$ is by definition the TD error (equation 2.18).

We see by equation 4.2 that the $\lambda$-return error being minimized at time-step $t$ is a (discounted) sum over all future one-step TD errors. In other words, we are simultaneously moving the value function estimate toward the one-step TD return target for *all* future states we will encounter in the trajectory, with states further into the future being less emphasized due to the discount factors $\lambda$ and $\gamma$. Further, we can interpret equation 4.2 as a general value function (GVF, section 3.4.1), where the "cumulant", $\delta(S, \boldsymbol{\theta})$, is the one-step TD error as a function of the current state and value function parameters.

### 4.1.1 Lambda Successor Return Error ($\lambda$-SRE)

The $\lambda$-return is a "theoretical" quantity: we do not have access to the full future trajectory at time $t$, and we can only construct it in an episodic setting by saving the entire trajectory to update end-of-episode. In this section, we introduce a decomposition trick to ameliorate the above weaknesses, through approximating *in expectation* the lambda return error. Given MRP $\mathcal{M}_\pi = \langle \mathcal{S}, r_\pi, P_\pi \rangle$ with a transition matrix $\mathbf{P}_\pi$, we first define a $\lambda\gamma$-discounted successor representation matrix (sometimes written as "$\lambda$-SR" in brief),

$$\mathbf{m}_\pi \doteq (\mathbf{I} - \lambda\gamma \mathbf{P}_\pi)^{-1} = \sum_{n=0}^{\infty} (\lambda\gamma\, \mathbf{P}_\pi)^n \,. \tag{4.3}$$

The above is nearly identical to the "vanilla" SR matrix definition in equation 3.20, with the only difference being the discount factor is now $\lambda\gamma$, as supposed to just $\gamma$. Thus, it also follows the same interpretation given in equation 3.22, specifically

$$\mathbf{m}_\pi(s_i, s_j) = \sum_{n=0}^{\infty} (\lambda\gamma)^n P_\pi(S_n = s_j | S_0 = s_i) \,. \tag{4.4}$$

We return to the $\lambda$-return error of equation 4.2 and treat the TD error itself as a random variable. Let $\delta_\theta(s) = \mathbb{E}_\pi[\delta_k | S_k = s] = \mathbb{E}_\pi[R_{k+1} + \gamma v_\theta(S_{k+1}) - v_\theta(S_k) | S_k = s]$ denote the expectation of this random variable. We write the $\lambda$-return error in expectation,

$$\begin{aligned}
\mathbb{E}[G_t^\lambda - v_{\boldsymbol{\theta}}(s) | S_t = s] &= \mathbb{E}\left[ \sum_{k=0}^{\infty} (\lambda\gamma)^k \, \delta_{t+k} \middle| S_t = s \right] , \\
&= \sum_{k=0}^{\infty} \sum_{s' \in \mathcal{S}} (\lambda\gamma)^k P_\pi(S_{t+k} = s' | S_t = s) \, \delta_\theta(s') \,, \\
&= \sum_{s' \in \mathcal{S}} (\sum_{k=0}^{\infty} (\lambda\gamma)^k P_\pi(S_{t+k} = s' | S_t = s)) \, \delta_\theta(s') \,, \\
&= \sum_{s' \in \mathcal{S}} \mathbf{m}_\pi(s, s') \, \delta_\theta(s') \,.
\end{aligned} \tag{4.5}$$

We can write the above using some sampling distribution $\rho(\cdot)$ with full support over the states, via $\sum_{s' \in \mathcal{S}} \mathbf{m}_\pi(s, s') \, \delta_\theta(s') = \sum_{s' \in \mathcal{S}} \frac{\rho(s')}{\rho(s')} \mathbf{m}_\pi(s, s') \, \delta_\theta(s')$,

$$\mathbb{E}[G_t^\lambda - v_{\boldsymbol{\theta}}(s) | S_t = s] = \mathbb{E}_\rho\Big[ \frac{\mathbf{m}_\pi(S, S')}{\rho(S')} \, \delta_\theta(S') \Big| S = s \Big] \,. \tag{4.6}$$

With reference to equation 4.5, the expected $\lambda$-return error can be decomposed into the $\lambda\gamma$-discounted successor representation matrix, $\mathbf{m}_\pi$, and the expected error at each state of the MRP. We refer to equation 4.5 as the **lambda successor return error** ($\lambda$**-SRE**). We emphasize it is a quantity independent of the value function parameterization $\boldsymbol{\theta}$, much like the $\lambda$-return is. That is, equation 4.5 can be used as the error for any kind of parameterization of the value function (linear, non-linear, etc.).

It should be noted that the summation in equation 4.5 is over the entire state space $\mathcal{S}$ and is typically intractable for large domains. However, equation 4.6 suggests a family of algorithms, where one can define any kind of sampling distribution $\rho$ that allow for an unbiased estimate of the $\lambda$-SRE. For example, in the case of uniform sampling of $s'$, the error at state $s$ will be directly proportional to $m(s, s')\delta_\theta(s')$.

### 4.1.2 The $\lambda$-SRE Fixed Point

We provide a brief analysis of the behaviour of iterative updates that minimize $\lambda$-SR weighted TD errors. Given the matrix form MRP, with transition matrix $\mathbf{P}_\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$, reward vector $\mathbf{r}_\pi \in \mathbb{R}^{|\mathcal{S}|}$, discount factor $\gamma \in [0, 1)$, current value estimate $\mathbf{r} \in \mathbb{R}^{|\mathcal{S}|}$, and some estimated $\lambda$-SR matrix $\mathbf{m} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$. We write the *SRE policy evaluation operator* as follows,

$$L_E\,\mathbf{v} = \mathbf{v} + \mathbf{m}\left(\mathbf{r}_\pi + \gamma\mathbf{P}_\pi\mathbf{v} - \mathbf{v}\right)\,. \tag{4.7}$$

Equation 4.7 updates the value estimates $\mathbf{v}$ to minimize the equation 4.5 error.

**Proposition 4.1.1.** *Given* any *invertible matrix* $\mathbf{m} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$*, the SRE policy evaluation operator has the true value function as its fixed point,*

$$L_E\,\mathbf{v}_E = \mathbf{v}_E = (\mathbf{I} - \gamma\mathbf{P}_\pi)^{-1}\mathbf{r}_\pi = \mathbf{v}_\pi\,. \tag{4.8}$$

*Proof.* For notation brevity, we borrow the SR notation from equation 3.20, $\mathbf{M}_\pi \doteq (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1}$. We can write the SRE policy evaluation operator as

$$\begin{aligned} L_E\, \mathbf{v} &= \mathbf{v} + \mathbf{m}\mathbf{r}_\pi - \mathbf{m}(\mathbf{I} - \gamma \mathbf{P}_\pi)\mathbf{v}\,, \\ &= \mathbf{m}\mathbf{r}_\pi + (\mathbf{I} - \mathbf{m}\mathbf{M}_\pi^{-1})\mathbf{v}\,. \end{aligned}$$

At its fixed point, we have $L_E\, \mathbf{v}_E = \mathbf{m}\mathbf{r}_\pi + (\mathbf{I} - \mathbf{m}\mathbf{M}_\pi^{-1})\mathbf{v}_E = \mathbf{v}_E$. Re-arranging,

$$\begin{aligned} \mathbf{v}_E &= (\mathbf{m}\mathbf{M}_\pi^{-1})^{-1}\mathbf{m}\mathbf{r}_\pi \\ &= (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1}\mathbf{r}_\pi = \mathbf{v}_\pi\,. \end{aligned}$$

□

**Proposition 4.1.2.** *Given invertible matrix* $\mathbf{m} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$*, and define* $\mathbf{1} = [1, 1, ...]^\top$*. We assume* $||(\mathbf{I} - \mathbf{m}\mathbf{M}_\pi^{-1}) \cdot \mathbf{1}||_\infty < 1$*. Then, SRE policy evaluation operator is a contraction in the infinity norm* $||\cdot||_\infty$*.*

*Proof.* Given arbitrary vectors in value function space, $\mathbf{u}, \mathbf{v} \in \mathbf{V}$,

$$\begin{aligned} ||L_E\, \mathbf{u} - L_E\, \mathbf{v}||_\infty &= ||(\mathbf{I} - \mathbf{m}\mathbf{M}_\pi^{-1})(\mathbf{u} - \mathbf{v})||_\infty\,, \\ &\leq ||(\mathbf{I} - \mathbf{m}\mathbf{M}_\pi^{-1})\,(\mathbf{1}||\mathbf{u} - \mathbf{v}||_\infty)||_\infty\,, \\ &= ||(\mathbf{I} - \mathbf{m}\mathbf{M}_\pi^{-1})\mathbf{1}||_\infty \cdot\, ||\mathbf{u} - \mathbf{v}||_\infty\,, \\ &< ||\mathbf{u} - \mathbf{v}||_\infty\,. \end{aligned}$$

The last step follows by the assumption $||(\mathbf{I} - \mathbf{m}\mathbf{M}_\pi^{-1}) \cdot \mathbf{1}||_\infty < 1$. □

That is, as long as the proposition 4.1.2 assumption is met, applying the SRE policy evaluation operator $L_E$ results in a sequence that converges to the true value function $\mathbf{v}_\pi$ (by Banach Fixed-Point Theorem, theorem 2.2.2). We do not prove all cases for which $\mathbf{m}$ satisfies this in our current work, but we refer the reader to (Pitis 2018) which show similar convergence results and how a true SR matrix (i.e. if $\mathbf{m} = \mathbf{m}_\pi$) satisfies a similar assumption.

### 4.1.3 Tabular Algorithm

As a proof of concept and for a fair comparison with the offline $\lambda$-return algorithm, we propose an offline algorithm using states sampled only from a single episode of on-policy trajectory.

**Algorithm 15:** Offline $\lambda$-SRE algorithm using single episode samples for policy evaluation, without correcting for sampling distribution

**Input**:
  MRP $\mathcal{M}_\pi$, discount factor $\gamma \in [0, 1]$, step-sizes $\alpha \in (0, 1]$ ;
  $\gamma\lambda$-discounted SR function, $m_\pi(\cdot, \cdot) \to \mathbb{R}$, and a way to learn it;
**Initialize:** Value function $v_{\boldsymbol{\theta}} : \mathcal{S} \to \mathbb{R}$ with parameters $\boldsymbol{\theta}$ ;
**while** *repeat episodes until satisfied* **do**
  Collect end-of-episode on-policy trajectory $\tau = (S_0, R_1, S_1, ..., R_T)$;
  ▷ `Update SR estimate with given learning method`
  Update $\lambda$-SR function, $m_\pi$ ;
  ▷ `Update to minimize λ-SRE`
  **for** $k = 0, ..., T-1$ **do**
    $\delta_k = R_{k+1} + \gamma v_{\boldsymbol{\theta}}(S_{k+1}) - v_{\boldsymbol{\theta}}(S_k)$
  **end**
  **for** $t = 0, ..., T-1$ **do**
    $\Delta\boldsymbol{\theta} = \left[\frac{1}{T-t} \sum_{k=t}^{T-1} m_\pi(S_t, S_k)\delta_k\right] \nabla_{\boldsymbol{\theta}} v_{\boldsymbol{\theta}}(s_t)$ ;
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\, \Delta\boldsymbol{\theta}$ ;
  **end**
**end**

We note that $\lambda$-SRE does not need to be an offline algorithm, but our main hypothesis is that simply *having* the $\lambda$-SRE (regardless of its implementation detail) *is better than using* $\lambda$*-return*, specifically by removing the trajectory dependence of $\lambda$-return via decomposing out the entire time component into the $\lambda\gamma$-discounted SR (Dayan 1993), which we hypothesize reduces trajectory variance (at the possible cost of increased bias in using a learned estimate of $\mathbf{m}_\pi$). Specifically, we use the on-policy distribution as the sampling distribution $\rho$ to approximate equation 4.6, without correcting for the frequency of states encountered. This is a naive approach: the lack of state frequency correction means equation 4.6 not longer exactly equal the lambda return error (equation 4.2), resulting in potential bias. Nonetheless, we find this works

reasonable well in the restricted set of empirical studies performed. We leave the development of more principled methods of correcting the return for future work. This gives rise to our offline $\lambda$-SRE algorithm (algorithm 15).

Once again note algorithm 15 is independent of the value function parameterization $\boldsymbol{\theta}$ as long as we can compute the gradient $\nabla_{\boldsymbol{\theta}} v_{\boldsymbol{\theta}}(s_t)$ (e.g. via backpropagation in a neural network), and assuming that we can learn (or are given) a $\lambda\gamma$-discounted SR function $m_\pi$ (in line 7 of algorithm 15).

### 4.1.4 Learning the $\lambda\gamma$-discounted SR

We briefly examine the question of learning the $\gamma\lambda$-discounted SR. Indeed, learning this is identical to learning the "full" ($\gamma$-discounted) SR, with a similar Bellman form:

$$m_\pi(s_t, s') = P_\pi(S_0 = s' | S_0 = s) + (\lambda\gamma)\mathbb{E}_\pi \left[ m_\pi(s_{t+1}, s') \right] . \tag{4.9}$$

Therefore, in theory we can leverage all methods from section 3.3 to learn this, substituting in the new discount factor $\lambda\gamma$.

**Algorithm 16:** End-of-episode learning of tabular $\lambda\gamma$-discounted SR

1 **Initialize:** Tabular $\lambda\gamma$-discounted SR matrix, $\mathbf{m} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$ ;
2 Given a on-policy trajectory of states from the latest episode: $\tau = (S_0, S_1, ..., S_{T-1})$ ;
3 **for** $S_t$ *where* $t = 0, ..., T-1$ **do**
4  **while** *time allows* **do**
5  Sample $S_k \sim \{S_0, S_1, ..., S_{T-1}\}$ from the trajectory, randomly with replacement;
6  $\Delta\mathbf{m}(S_t, S_k) \leftarrow \mathbb{1}_{S_t = S_k} + (\gamma\lambda)\,\mathbf{m}(S_{t+1}, S_k) - \mathbf{m}(S_t, S_k)$ ;
7  $\mathbf{m}(S_t, S_k) \leftarrow \mathbf{m}(S_t, S_k) + \alpha\Delta\mathbf{m}(S_t, S_k)$ ;
8  **end**
9 **end**

However, learning $m_\pi$ in the functional approximation (non-tabular) case is non-trivial, as previously discussed in section 3.3.2. For now, we use a tabular algorithm for learning $\mathbf{m}_\pi$ (line 7 of algorithm 15). In particular, how to best do SR learning is

not the focus of this work, but instead we illustrate that learning the SR is beneficial in the case of single-task value learning. Thus, we trivially adopt the tabular SR algorithm from Machado, Bellemare, and Bowling (2020) and include it below for completeness. To make it compatible with algorithm 15, we provide an end-of-episode update algorithm for SR learning (algorithm 16).

## 4.2 Experiments

For our experiments, we focus on the tabular setting as a proof-of-concept for $\lambda$-SRE, as it allows for tractable computation of $\mathbf{m}_\pi$ and for us to easily solve for the true values of the MDP to compare against the agents' estimates. All algorithms are evaluated on the canonical 19 states random chain task (figure 4.1, also see R. S. Sutton and Barto (2018), figure 12.3).

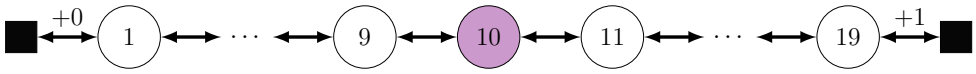


Figure 4.1: The 19-state random walk chain. Agent starts in the centre and transition randomly to adjacent states until reaching the terminal states on either end. Reward is 0 for all transitions, except for the right-side termination state, which yields a reward of +1.

We always initialize the tabular value function $\mathbf{v}$ (for the offline $\lambda$-return and the $\lambda$-SRE algorithms which contains an explicitly parameterized value function) to 0.5, the averaged value of the random walk chain MDP. This follows from the method used in Chapter 12 of R. S. Sutton and Barto (2018). For all experiments we run 50 seeds and average over them for the results shown. Error bars, when shown, denote 95% confidence interval (standard error).

Additionally, in the inner loop of algorithm 16, we sample only $n \approx (0.05 \cdot T)$ number of states, $S_k$ ($T$ is trajectory length). While larger samples can help with faster learning, small samples worked well in our task, and had better computational efficiency. Further, we note that in the tabular case, the successor representation is

upper-bounded by $\mathbf{m}_\pi(s, s') \leq \sum_{n=0}^{\infty}(\lambda\gamma)^n = \frac{1}{1-\gamma\lambda}$, for all $s, s' \in \mathcal{S}$. As a heuristic we therefore initialize the successor matrix to $\mathbf{m}_{\text{init}}(\cdot, \cdot) = 0.5 \cdot \frac{1}{1-\min(\gamma\lambda, 0.95)}$. The 0.5 constant is akin to a prior saying any state has a 0.5 chance of reaching any other states at any times, and the min is used to prevent the values from blowing up to infinity. We emphasize this is a rough heuristic we used in our experiments and we did not explore extensively the effect of initialization on learning the tabular SR matrix as this is not the main focus of this work.

### 4.2.1 Policy Evaluation

We compare all algorithms in the offline, on-policy setting with access to the full trajectory of the most recent episode. We compare three algorithms learning from scratch: (i) offline lambda return (algorithm 5), (ii) our offline $\lambda$-SRE (algorithm 15), and (iii) a SR algorithm with decoupled SR and reward representations (algorithm 16, with $\lambda = 1$). We give the SR algorithm the *true* reward function $\mathbf{r}_\pi$, such that it only needs to learn the SR matrix (as the value estimate can be recovered by the dot product $\mathbf{v} = \mathbf{M} \cdot \mathbf{r}_\pi$).[1] Despite this, the SR algorithm had a large ($> 1$) error. We therefore excluded it from figure 4.2.

Again following the set-up in R. S. Sutton and Barto (2018), we use no discounting ($\gamma = 1$) and evaluate the root mean squared error (RMSE) of each agent's value function after 10 episodes' worth of experience. We also use the same step-size for all learning procedures (e.g. for value learning and for SR learning in the $\lambda$-SRE agent).

We see that the $\lambda$-SRE agent performs similarly whether the true $\lambda$-SR matrix, $\mathbf{m}_\pi$, is given (figure 4.2, centre), *or* if it needs to be learned from scratch via algorithm 16 (figure 4.2, right), albeit the concurrently learned $\lambda$-SRE agent shows more instability across the learning rates. Both $\lambda$-SRE algorithms consistently out-perform

[1]The reward function can be estimated using supervised learning. However, in exploratory experiments, the (SF and reward) learning rates need to be set differently. Specifically, the reward learning prefers a very small learning rate, while the SR learning prefers larger ones. Using the same step-sizes resulted in either a complete lack of learning or divergence. Thus, for simplicity, we opted to just give the SR agent the true reward function and let it learn *only* the SR matrix.

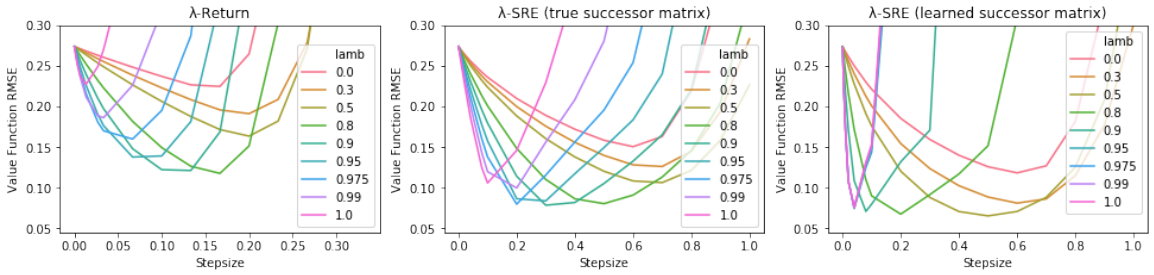


Figure 4.2: **Policy evaluation in random walk chain**. Vertical axis shows the root mean squared error (RMSE) at the end of the 10-th episode, averaged over 50 seeds. **(Left)** offline $\lambda$-return. **(Centre)** offline $\lambda$-SRE given true $\mathbf{m}_\pi$, **(Right)**: offline $\lambda$-SRE with concurrently learned $\mathbf{m}_\pi$ and value function.

the traditional $\lambda$-return algorithm (figure 4.2, left) while having access to the exactly same information, across a wide range of learning rates.

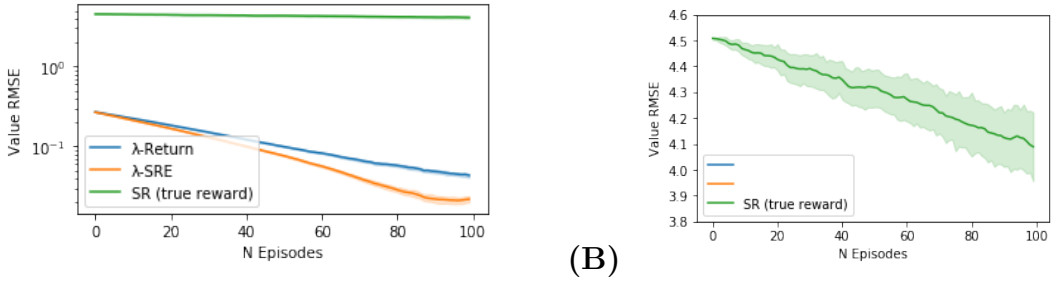


Figure 4.3: **Policy evaluation in random walk chain**. Root mean squared error (RMSE) over the first 100 episodes of training. **(A)** RMSE of the $\lambda$-return, $\lambda$-SRE and (vanilla) SR algorithms on log scale. **(B)** RMSE of just the SR algorithm on linear scale to show it is learning, albeit very slowly. All settings use 50 independent runs with error bars denoting 95% standard error.

Figure 4.3 further demonstrates the learning efficiency of the $\lambda$-SRE by plotting the 100 episodes learning curves for the three algorithms with best parameter settings. While both algorithms learn a SR, we observe that $\lambda$-SRE has lower value error as compared to the (vanilla) SR at all points of value learning. The failure of SR to learn might be due to multiple reasons. Initialization of the SR matrix plays a potential role, though this is not a problem with the $\lambda$-SRE. The undiscounted case may also be difficult to learn in as future occupancy have high value. Additionally, small errors in the SR matrix may result in large errors in the value function (L. M. White 1996).

### 4.2.2 Using Successor-like Representations

Having shown the benefit of using $\lambda$-SRE, we turn to the representation learned. While both the $\lambda$-SRE and the ("vanilla") SR algorithms learn a SR-like matrix, the SR matrix is used very differently and result in different learning efficiencies (figure 4.3). Here, we will directly compare the effect of the SR matrix error on the $\lambda$-SRE and the SR algorithms.

Specifically, we set $\lambda = 1$ for the $\lambda$-SRE agent such that both agents are learning *identical* SR matrices. We use identical learning rates of $\alpha = 0.1$ for all experiments and run for 3000 episodes, with discounting $\gamma = 0.8$ (this was chosen to make the learning process easier for the SR agent as high $\gamma$ may result in more unstable learning). We emphasize we learn from scratch the SR matrix for *both* the $\lambda$-SRE and SR algorithms here rather than using a pre-computed SR matrix.[2]


Figure 4.4: **Comparison of policy evaluation in the random walk chain.** Vertical axis shows root mean square error (RMSE) of the value function and successor matrix for the $\lambda$-SRE and SR algorithms. **(Left)** Value function error over training. **(Middle)** Successor matrix error over training. **(Right)** Correlation of value function error with successor matrix error. All settings use 50 independent runs, error bars denote 95% standard error.

We observe that the successor matrix error is identical for both algorithms throughout training (figure 4.4, middle), which is unsurprising given we explicitly set it up this way. Yet, the $\lambda$-SRE agent converges much more quickly than the SR agent (figure 4.4, left). We also see that the $\lambda$-SRE agent is robust to a much higher successor

[2]We still give the (vanilla) SR algorithm the ground truth reward function such that it only needs to do SR learning.

matrix error while the SR agent requires a highly accurate successor matrix for good value prediction (figure 4.4, right).

## 4.3 Discussion on Machine Learning

### 4.3.1 Successor Errors

Consider the tabular successor representation matrix $\mathbf{M}_\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$. Each row, $\mathbf{M}_{i,}$, is the SR for state $i$, representing the total future (discounted) visitations to all other states (Dayan 1993). We can similarly view the SR as the degree of "contribution" from state $i$ to all other states. Consequently, the expected *error* received at state $i$ from other states is proportional to its successor representation. Likewise, we can also view the matrix *columns*, $\mathbf{M}_{,j}$, as how error at state $j$ should be distributed to all other states—this is known as the *ideal source trace* and explored extensively in Pitis (2018).

In $\lambda$-return, the $\lambda$ term gives us additional flexibility to tune how much importance we wish to assign to future *errors*, which is expressed via the $\lambda$-SR matrix, $\mathbf{m}_\pi$. Similar to how the value is the expected discounted sum of future (instantaneous) rewards, the $\lambda$-return error is the discounted sum of future (one-step) errors (equation 4.2). It is perhaps unsurprising that just as one can compute the value for a state in one step using the SR and rewards, we can compute the expected $\lambda$-return error in one step using the $\lambda$-SR and TD errors. We can interpret the $\lambda$-successor return similarly to the $\lambda$-return. In the case of $\lambda = 0$, we have the lambda successor matrix $\mathbf{m}_\pi = \mathbf{I}$, meaning we only care about the *immediate* TD errors and do TD(0). In the case of $\lambda = 1$ we recover the successor representation ($\mathbf{m}_\pi = \mathbf{M}_\pi$), weighing future errors proportional to their actual ($\gamma$-discounted) occupancy.

We note that the $\gamma$ term in SR is *behaviourally relevant*: different $\gamma$ can result in different levels of myopia. Unlike $\gamma$ dependent processes like SR learning, the $\lambda$ term

is a purely learning parameter, and we are free to tune the discounting freely in $\lambda$-SRE without changing the policy myopia. It is interesting to observe that lower $\lambda$ tends to perform better in the $\lambda$-SRE agent as compared to the $\lambda$-return agent (figure 4.2), potentially pointing to lower $\lambda$ resulting in a more stable learning problem.

### 4.3.2 Related Works

The closest work to ours is Source Traces (Pitis 2018), which can be viewed as a "backward view" version of $\lambda$-SRE. While we update a current value function by a future error, $\Delta v(s_t) \propto \sum_{s'} m(s_t, s')\delta_\theta(s')$, Pitis (2018) updates *past* value functions using the present TD error: $\Delta v(s') \propto m(s', s_t)\delta(s_t)\ \forall s'$. Both works require learning the SR function $m_\pi$, which is a mutual weaknesses. However, even if $m_\pi$ can be learned, the backward view has two additional weaknesses for non-tabular cases: (i) it assumes access to the entire state space for error propagation; and (ii) it is unclear how to interpret the SR function as an eligibility trace, as $m_\pi$ maps to a scalar, while (non-tabular) eligibility traces require one to keep track of all model parameters. We note this latter perspective is explored further in H. v. Hasselt, Madjiheurem, et al. (2020), which can be viewed as "back-in-time" successor *features* for linear value functions.

## 4.4 Discussion on Neuroscience

Our work opens up the interesting question of a possible relationship between hippocampal *representation* (O'Keefe and Dostrovsky 1971; Stachenfeld, M. M. Botvinick, and Gershman 2017; Mehta, Quirk, and Wilson 2000; Alvernhe, Save, and Poucet 2011; Hollup et al. 2001), and the hippocampus as a system for *fast learning* (through replay (Gupta et al. 2010; Mattar and Daw 2018; Momennejad, Otto, et al. 2018), complementary learning (McClelland, McNaughton, and O'Reilly 1995; Kumaran, Hassabis, and McClelland 2016), and/or episodic control (Lengyel and Dayan 2008;

Blundell et al. 2016; Gershman and Daw 2017)). Through the $\lambda$-successor representation, we demonstrate a representation *to support fast learning* (of a value function) through better credit assignment. Notably, the physiological evidence supporting place fields as successor representations equally support the $\lambda$-SR hypothesis (as SR can be seen as a special case of $\lambda$-SR). We further speculate that learning the value function through $\lambda$-SRE naturally fit with methods such as experience replay (or a model), specifically through the need to sample the state space to estimate equation 4.6. It may be interesting to re-interpret hippocampal replay data in light of $\lambda$-SRE.

We reiterate a small but important nuance between the traditional SR and $\lambda$-SR: SR is a direct (decomposed) representation of the value function, while $\lambda$-SR weighs the TD errors to *update* a (separate) value function. We demonstrate through simulations that the latter can result in both faster learning and is more robust to imperfections in the learning process—a potentially desirable biological property that complements the strength of the traditional SR (e.g. for transfer). Subscribing to the $\lambda$-SR as a model for place fields, we would expect place cells to be important *during* learning, but not for value estimation after learning. Some works support this: the dorsal hippocampus is *transiently* involved in action acquisition (Bradfield et al. 2020), and dopamine release (a quantity typically associated with the one-step TD error (Schultz 2016)) in the dorsal hippocampus promoting spatial learning (Kempadoo et al. 2016). Future experiments can help delineate the interactions between reward prediction error, place fields, and learning speed, through the lens of $\lambda$-SRE.

## 4.5 Future Works

We identify two main limitations to the current work: (i) proposing a sampling distribution, $\rho$, in equation 4.6; and (ii) having a principled method of learning the $\lambda$-SR function, $m_\pi$ in high dimensional feature spaces. Point (i) is well suited to be com-

bined with an experience replay buffer, and/or a (generative) model of states and state transitions. On the other hand, point (ii) is much more difficult. Some possible direction include heuristic methods to estimate the proximity of features, or learning the function $m_\pi(\cdot, \cdot)$ as a metric using the methods discussed in section 3.3.2. This is a promising direction for future research, as the biological evidence also points to such "successor-like" representations being present in the brain, which works exclusively in high-dimensional settings.

All in all, we show an interesting connection between SR—a representation typically used for transfer, and the $\lambda$-return—a quantity for temporal credit assignment. We show our method works in tabular settings and complements the current neuroscience theories and observations, demonstrating its potential to be further investigated for future research at the intersection of biological and artificial RL.

# 5

# Lambda Value Function

In chapter 4, we developed a method for error propagation using the successor representation matrix. This was motivated by decomposing the signed error between the $\lambda$-return and the current value prediction in expectation. We motivate a similar decomposition in this chapter, using the $\lambda$-return itself. Interestingly, the decomposition results in a generalized form of value function parameterization—the $\lambda$ value function ($\lambda$-VF)—that generalizes traditionally parameterized "model-free" and "successor-feature" value functions. This decomposition is also more amenable to function approximation, being readily extendable to combine with deep neural networks.

We develop the theory of $\lambda$-VF in section 5.1 and outline algorithms. The algorithms are evaluated on illustrative and high-dimensional standard benchmarks in section 5.2. Finally, we discuss the $\lambda$-VF in relation with both machine learning and neuroscience.

## 5.1 Lambda Value Function ($\lambda$-VF)

In this section we motivate the core idea and derivation of the Lambda Value Function ($\lambda$-VF). First, we review the two commonly used ways of estimating value, namely the "model-free" and "successor features" value functions, in section 5.1.1. Then, we

show how the $\lambda$-return, in expectation, can be decomposed into the $\lambda$-VF in section 5.1.2 as a generalization of the two previous value functions. We briefly analyze the fixed-point of learning with $\lambda$-VF as a one-step target in section 5.1.3. Finally, we discuss algorithms for learning the $\lambda$-VF for linear prediction and nonlinear control in sections 5.1.4 and 5.1.5.

### 5.1.1 Linearly Predictive Rewards and Features

We focus on the policy evaluation setting and review the two previously presented methods of estimating value. Given MRP $\mathcal{M}_\pi = \langle \mathcal{S}, r_\pi, P_\pi \rangle$, discount factor $\gamma \in [0, 1)$, and feature set $\Phi$. We write $\phi(s_t) = \boldsymbol{\phi}_t$, $\boldsymbol{\phi}_t \in \mathbb{R}^d$ to denote the $d$-dimensional feature for the state sampled at time $t$, $s_t \in \mathcal{S}$. We are interested in approximating the true value, $v_\pi$, as a linear function of the features.[1]

**Model-Free (MF) Value Function** The first method *directly* estimates value as a function of the features. This corresponds to equation 3.5 in section 3.1.3, which we refer to as the **MF value estimate**:

$$v_\pi(s) \approx v^{\text{MF}}(s) \doteq v_{\boldsymbol{\theta}}(s) = \phi(s)^\top \boldsymbol{\theta}\,, \tag{5.1}$$

with $\boldsymbol{\theta} \in \mathbb{R}^d$ learnable parameters.

Methods for learning the MF value function are covered extensively in section 3.1, thus we do not review them in detail here. Generally, learning involves constructing a learning target $U_t$ and updating $v_t^{\text{MF}}$ toward the target:

$$\boldsymbol{\theta}_{(\text{new})} = \boldsymbol{\theta}_{(\text{old})} + \alpha(U_t - \boldsymbol{\phi}_t^\top \boldsymbol{\theta}_{(\text{old})})\boldsymbol{\phi}_{t+1}\,, \tag{5.2}$$

with learning rate parameter $\alpha \in (0, 1]$. Different targets can be constructed in the form of different *returns*, $G$. Different returns are reviewed in section 2.2.

[1] This also extends naturally to nonlinear ANN settings by treating the last (linear) layer of the ANN as the value function, and the layers up to the last layer as the feature encoder.

**Successor Features (SF) Value Function** We can also decouple the value estimate into reward and transition information, by decomposing into immediate rewards and SFs, as was done in many previous works (Dayan 1993; Kulkarni et al. 2016; Zhang et al. 2017; Barreto, Dabney, et al. 2017; Barreto, Borsa, et al. 2018). This decomposition is introduced in section 3.4.2.

To review, the SFs $\boldsymbol{\psi}_\pi \in \mathbb{R}^d$ (equation 3.32), are the expected cumulative discounted features under a policy $\pi$, $\boldsymbol{\psi}_\pi(s) \doteq \mathbb{E}_\pi \left[\sum_{n=0}^{\infty} \gamma^n \boldsymbol{\phi}_{t+n} \mid S_t = s\right]$. We use the SF function $\psi_{\boldsymbol{\Xi}} : \mathbb{R}^d \to \mathbb{R}^d$ to estimate to SF as a linear function the features:

$$\boldsymbol{\psi}_\pi(s) \approx \psi_{\boldsymbol{\Xi}}(s) = \boldsymbol{\Xi}^\top \phi(s)\,, \tag{5.3}$$

where $\boldsymbol{\Xi} \in \mathbb{R}^{d \times d}$ are learnable parameters.[2] Using the Bellman form $\boldsymbol{\psi}_\pi(s) = \phi(s) + \gamma \mathbb{E}_\pi \left[\boldsymbol{\psi}_\pi(S_{t+1}) | S_t = s\right]$, the SF and can be learned by any previously introduced value learning algorithms. Furthermore, recall the instantaneous reward function $r_{\mathbf{w}}(s) \doteq \phi(s)^\top \mathbf{w} \approx \mathbb{E}_\pi[R_{t+1} | S_t = s]$. We write the **SF value estimate** (equation 3.34) as:

$$v_\pi(s) \approx v^{\text{SF}}(s) \doteq v_{\boldsymbol{\Xi},\mathbf{w}}(s) = \psi_{\boldsymbol{\Xi}}(s)^\top \mathbf{w}\,. \tag{5.4}$$

We see the MF value, $v_t^{\text{MF}} = \boldsymbol{\phi}_t^\top \boldsymbol{\theta}$, linearly combines local feature information $\boldsymbol{\phi}_t$ with temporally-extended reward prediction.[3] On the other hand, the linear SF value, $v^{\text{SF}} = \boldsymbol{\phi}_t^\top \boldsymbol{\Xi} \mathbf{w}$, linearly combines temporally-extended feature prediction in $\boldsymbol{\phi}_t^\top \boldsymbol{\Xi} = \boldsymbol{\psi}_t$, with local reward information in $\mathbf{w}$. We later see the $\lambda$-VF interpolates between the extent of feature prediction versus reward prediction.

[2]Unless stated otherwise, we consider $\psi_{\boldsymbol{\Xi}} : \mathbb{R}^d \to \mathbb{R}^d$ to be a linear function of the features with parameters $\boldsymbol{\Xi}$. This linearity is not necessary—we can learn arbitrary functions for $\psi$. Nonetheless, we stick to only linear functions, in part as it allows for simple comparisons with the linear MF value function, where the two function produce identical estimates when $\boldsymbol{\Xi} \cdot \mathbf{w} = \boldsymbol{\theta}$. For learning SFs as nonlinear functions of features, see (Zhang et al. 2017; Machado, Bellemare, and Bowling 2020).

[3]Here, temporally-extended refers to the parameter encoding future information: $\boldsymbol{\theta}$ encodes cumulative future rewards.

### 5.1.2 The $\lambda$-VF Decomposition

We introduce a decomposition of the $\lambda$-return that generalizes the MF and SF value functions. Recall the $\lambda$-return (chapter 2, definition 2.2.7) uses a $\lambda$ geometrically discounted average of all future n-step estimates of the value function. Specifically, the $\lambda$-return can be written in the following way, using the identity introduced in equation 2.24,

$$G_t^\lambda = R_{t+1} + \gamma \left( \sum_{n=1}^{\infty} (\lambda\gamma)^{n-1} \left[ (1-\lambda) v(S_{t+n}) + \lambda R_{t+n+1} \right] \right) . \tag{5.5}$$

Let us define a $\lambda\gamma$-discounted successor feature,

$$\boldsymbol{\psi}_\pi^\lambda(s) \doteq \mathbb{E}_\pi \left[ \textstyle\sum_{n=0}^{\infty} (\lambda\gamma)^n \boldsymbol{\phi}_{t+n} \mid S_t = s \right] \approx \psi_{\boldsymbol{\Xi}}^\lambda(s) \,, \tag{5.6}$$

which we can separately estimate using $\psi_{\boldsymbol{\Xi}}^\lambda(s) \doteq \boldsymbol{\Xi}^\top \phi(s)$, with $\boldsymbol{\Xi} \in \mathbb{R}^{d \times d}$ (learnable) parameters. Then, given the linear MF value function $v_{\boldsymbol{\theta}}(s) = \phi(s)^\top \boldsymbol{\theta}$ and linear reward model $r_{\mathbf{w}}(s) = \phi(s)^\top \mathbf{w}$, we have the following results.

**Remark 5.1.1.** *The expected $\lambda$-return can be written approximately as*

$$\mathbb{E}_\pi[G_t^\lambda] \approx \mathbb{E}_\pi \left[ R_{t+1} + \gamma \boldsymbol{\psi}_\pi^\lambda(S_{t+1}) [(1-\lambda)\boldsymbol{\theta} + \lambda \mathbf{w}] \right] . \tag{5.7}$$

*Proof.* Follows from a similar decomposition as remark 3.4.2, which uses the linearity of the value and reward estimates to decompose out the future expected features,

$$\begin{aligned}
\mathbb{E}_\pi[G_t^\lambda] &= \mathbb{E}_\pi \left[ R_{t+1} + \gamma \left( \textstyle\sum_{n=1}^{\infty} (\lambda\gamma)^{n-1} [(1-\lambda) v_{\boldsymbol{\theta}}(S_{t+n}) + \lambda R_{t+n+1}] \right) \right] , \\
&\approx \mathbb{E}_\pi \left[ R_{t+1} + \gamma \left( \textstyle\sum_{n=1}^{\infty} (\lambda\gamma)^{n-1} [(1-\lambda) \boldsymbol{\phi}_{t+n}^\top \boldsymbol{\theta} + \lambda \boldsymbol{\phi}_{t+n}^\top \mathbf{w}] \right) \right] , \\
&= \mathbb{E}_\pi \left[ R_{t+1} + \gamma \left( \textstyle\sum_{n=1}^{\infty} (\lambda\gamma)^{n-1} \boldsymbol{\phi}_{t+n}^\top \right) [(1-\lambda)\boldsymbol{\theta} + \lambda \mathbf{w}] \right] , \\
&= \mathbb{E}_\pi \left[ R_{t+1} + \gamma \boldsymbol{\psi}_\pi^\lambda(S_{t+1}) [(1-\lambda)\boldsymbol{\theta} + \lambda \mathbf{w}] \right] .
\end{aligned}$$

□

**Definition 5.1.1.** We define the **lambda value function (λ-VF)**, $v^\lambda : \mathcal{S} \to \mathbb{R}$, as

$$v^\lambda(S_{t+1}; \boldsymbol{\Xi}, \boldsymbol{\theta}, \mathbf{w}) \doteq \boldsymbol{\psi}^\lambda_{\boldsymbol{\Xi}}(S_{t+1})^\top \left((1-\lambda)\boldsymbol{\theta} + \lambda \mathbf{w}\right) , \tag{5.8}$$

which combines the $\lambda\gamma$-discounted SF model $\boldsymbol{\psi}^\lambda_{\boldsymbol{\Xi}}$ with parameters $\boldsymbol{\Xi}$, and a *mixture* of value $\boldsymbol{\theta}$ and reward $\mathbf{w}$ parameters. We usually write $v^\lambda(\cdot) = v^\lambda(\cdot; \boldsymbol{\Xi}, \boldsymbol{\theta}, \mathbf{w})$ for brevity.

**Corollary.** *The λ-VF can be used as a one-step value learning target—the one-step* ***λ-VF target****:*

$$U_t \doteq R_{t+1} + \gamma v^\lambda(S_{t+1}) , \tag{5.9}$$

*and it follows from remark 5.1.1 that this target approximates the expected λ-return,*

$$\mathbb{E}_\pi \left[ R_{t+1} + \gamma v^\lambda(S_{t+1}) \right] \approx \mathbb{E}_\pi[G^\lambda_t] . \tag{5.10}$$

**Interpolating between "model-free" and "model-based" with $\lambda$** Similar to how $\lambda$-return interpolates between the TD and MC returns, the $\lambda$-VF is a generalization that interpolates between the "model-free" MF value and the more "model-based" SF value—the latter can be interpreted as an implicit policy-dependant model estimation of the value function.

**Remark 5.1.2.** *When $\lambda = 0$, the λ-VF is the MF value function (equation 5.1),*

$$v^{\lambda=0}(s_t) = \boldsymbol{\psi}^{\lambda=0}_\pi(s_t)^\top \left((1-0)\boldsymbol{\theta} + 0\mathbf{w}\right) = \boldsymbol{\phi}_t^\top \boldsymbol{\theta} = v^{MF}(s_t) . \tag{5.11}$$

It follows that bootstrapping with this target defaults to the one-step TD return (definition 2.2.5).

**Remark 5.1.3.** *When $\lambda = 1$, the λ-VF is the SF value function (equation 5.4), equivalent to using an implicit infinite model,*

$$v^{\lambda=1}(s_t) = \boldsymbol{\psi}^{\lambda=1}_\pi(s_t)^\top \left((1-1)\boldsymbol{\theta} + 1\mathbf{w}\right) = \boldsymbol{\psi}^\top_{\pi,t}\mathbf{w} = v^{SF}(s_t) . \tag{5.12}$$

Consequently, the $\lambda$-VF is a generalization that spans the spectrum of value function parametrizations using $\lambda \in [0, 1]$, with the traditional (MF, SF) value functions as extremes.[4]

**One-step approximate $\lambda$-return** Consider a single-step transition tuple ($S_t$, $A_t$, $R_{t+1}$, $S_{t+1}$). TD(0) propagates information locally from $S_{t+1}$ to $S_t$ by constructing a *bootstrapped target*. The MF TD target (definition 2.2.5) propagates only value information, while the SF TD target propagates only feature information. We hypothesize we can more effectively use the same one-step information by simultaneously predicting both the value *and* the features. Specifically, we can update the $\lambda$-SF (equation 5.6) and instantaneous reward parameter $\mathbf{w}$. We then construct the *one-step $\lambda$-VF target* (equation 5.9) using the current SF $\boldsymbol{\Xi}$, reward $\mathbf{w}$ and value $\boldsymbol{\theta}$ parameters: $U_t \equiv R_{t+1} + \gamma v^{\lambda}(S_{t+1}; \boldsymbol{\Xi}, \boldsymbol{\theta}, \mathbf{w})$. This target is used for further value learning to update parameter $\boldsymbol{\theta}$.

Despite not using a multi-step trajectory, the combination of the SF and value parameters allow the model to implicitly "look ahead" to access value information in the future, akin to a form of "implicit planning".[5] Therefore, we hypothesize that **the $\lambda$-VF with an intermediate lambda ($0 < \lambda < 1$)**, which combines both feature and value predictions, **uses information more effectively than both the MF ($v^{\lambda=0}$) and SF ($v^{\lambda=1}$) value estimates**, approximating the true value faster given the same amount of data (see figure 5.2 for an intuitive demonstration of this).

[4] We have reused $\lambda$ to show the analogy with the $\lambda$-return, but $\lambda$ holds a different interpretation of trading-off learning vs planning implicitly.

[5] This shares a similar intuition with the $\lambda$-return in that the $\lambda$-return can generate future value estimates. However, the $\lambda$-return constructs future value estimates by sampling a multi-step trajectory to access future features. Unlike the $\lambda$-return, the $\lambda$-VF constructs future value estimates by combining the value parameters with the successor features (which estimates expected future cumulative features), while being learned with only single-step experiences.

### 5.1.3 Fixed Point of Linear $\lambda$-VF TD(0)

We provide a fixed point analysis for iterative value learning using the *one-step $\lambda$-VF target* (equation 5.9). The analysis follows similarly from the analysis of the TD(0) fixed point in section 3.1.3.

Consider the MRP $\mathcal{M}_\pi = \langle \mathcal{S}, r_\pi, P_\pi \rangle$ with discount factor $\gamma \in [0, 1)$ and feature set $\Phi$. We use the same matrix notation as in definition 3.1.3, namely with transition matrix $\mathbf{P}_\pi \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$, reward vector $\mathbf{r}_\pi \in \mathbb{R}^{|\mathcal{S}|}$, feature matrix $\mathbf{\Phi} \in \mathbb{R}^{|\mathcal{S}| \times d}$ with linearly independent columns, and the diagonal on-policy state distribution matrix $\mathbf{D} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$.

We analyze the on-policy, one-step learning setting, where all parameters at time $t$ are updated with the online, on-policy, one-step experience tuple $(\boldsymbol{\phi}_t, R_{t+1}, \boldsymbol{\phi}_{t+1})$. As an overview for the remainder of this section, we separately analyze the fixed point of one-step value learning for parameters $\boldsymbol{\theta}$, SF learning for $\mathbf{\Xi}$, and instantaneous reward learning for $\mathbf{w}$, then analyze value learning using the one-step $\lambda$-VF target with the SF and reward parameters at their respective fixed points.

**Value fixed-point** We re-iterate the results from section 3.1.4. One-step TD learning with linear function approximation solves for the following (Parr et al. 2008),

$$\boldsymbol{\theta} = (\mathbf{\Phi}^\top \mathbf{D} \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{D} \left( \mathbf{R} + \gamma \mathbf{P}_\pi \mathbf{\Phi} \boldsymbol{\theta} \right) . \tag{5.13}$$

The solution is the *TD fixed point* (lemma 3.1.1),

$$\boldsymbol{\theta}_{TD} = (\mathbf{\Phi}^\top \mathbf{D} \mathbf{\Phi} - \gamma \mathbf{\Phi}^\top \mathbf{D} \mathbf{P}_\pi \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{D} \mathbf{R} . \tag{5.14}$$

Note the fixed point implies the following,

$$(\mathbf{\Phi}^\top \mathbf{D} \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{D} \left( \mathbf{R} + \gamma \mathbf{P}_\pi \mathbf{\Phi} \boldsymbol{\theta}_{TD} \right) = \boldsymbol{\theta}_{TD} . \tag{5.15}$$

We can write a similar system and fixed point with a $\lambda\gamma$-discounted value function;

this is a term that will appear in the $\lambda$-VF,

$$\boldsymbol{\theta}^{\lambda}_{TD} = (\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi} - \lambda\gamma\boldsymbol{\Phi}^\top \mathbf{D}\mathbf{P}_\pi\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{D}\mathbf{R}\,, \tag{5.16}$$

$$(\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{D}\left(\mathbf{R} + \lambda\gamma\mathbf{P}_\pi\boldsymbol{\Phi}\boldsymbol{\theta}^{\lambda}_{TD}\right) = \boldsymbol{\theta}^{\lambda}_{TD}\,. \tag{5.17}$$

**SF fixed-point** Now consider one-step learning of the linear $\gamma$-discounted SF function $\psi_{\boldsymbol{\Xi}}(s) \doteq \boldsymbol{\Xi}^\top\phi(s)$, with $\boldsymbol{\Xi} \in \mathbb{R}^{d\times d}$. The update is:

$$\boldsymbol{\Xi}^\top_{t+1} = \boldsymbol{\Xi}^\top_t + \alpha\left(\boldsymbol{\phi}_t + \gamma\boldsymbol{\Xi}^\top_t\boldsymbol{\phi}_{t+1} - \boldsymbol{\Xi}^\top_t\boldsymbol{\phi}_t\right)\boldsymbol{\phi}^\top_t\,. \tag{5.18}$$

Similar to value-learning with TD(0), SF learning with TD(0) corresponds to solving the following,[6]

$$\boldsymbol{\Xi} = (\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{D}\left(\boldsymbol{\Phi} + \gamma\mathbf{P}_\pi\boldsymbol{\Phi}\boldsymbol{\Xi}\right)\,. \tag{5.19}$$

The SF fixed-point is as follows,

$$\boldsymbol{\Xi}_{TD} = (\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi} - \gamma\boldsymbol{\Phi}^\top \mathbf{D}\mathbf{P}_\pi\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi}\,. \tag{5.20}$$

Similarly, we can write down the system and fixed point for a $\lambda\gamma$-discounted SF (i.e. equation 5.6), $\boldsymbol{\psi}^{\lambda}_\pi(s) = \mathbb{E}_\pi[\sum_{n=0}^{\infty}(\lambda\gamma)^n\boldsymbol{\phi}_{t+n} \mid S_t = s]$ as follows,

$$\boldsymbol{\Xi}^{\lambda}_{TD} = (\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi} - \lambda\gamma\boldsymbol{\Phi}^\top \mathbf{D}\mathbf{P}_\pi\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi}\,, \tag{5.21}$$

$$(\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{D}\left(\boldsymbol{\Phi} + \lambda\gamma\mathbf{P}_\pi\boldsymbol{\Phi}\boldsymbol{\Xi}^{\lambda}_{TD}\right) = \boldsymbol{\Xi}^{\lambda}_{TD}\,. \tag{5.22}$$

**Reward regression solution** We take the a similar approach to analyze supervised regression of the instantaneous reward function $r_{\mathbf{w}}(s) \doteq \phi(s)^\top\mathbf{w} \approx \mathbb{E}_\pi[R_{t+1}|S_t = s]$ with $\mathbf{w} \in \mathbb{R}^d$. It has the following supervised update,

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\left(R_{t+1} - \boldsymbol{\phi}^\top_t\mathbf{w}_t\right)\boldsymbol{\phi}_t\,. \tag{5.23}$$

The reward regression solution is,

$$\hat{\mathbf{w}} = (\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{D}\mathbf{R}\,. \tag{5.24}$$

[6] We can similarly analyze the *expected update* of one-step SF learning, $\mathbb{E}_{\boldsymbol{\mu}}[\boldsymbol{\Xi}^\top_{t+1}|\boldsymbol{\Xi}^\top_t] = \boldsymbol{\Xi}^\top_t + \alpha(\mathbf{b}_{\boldsymbol{\Xi}} - \boldsymbol{\Xi}^\top_t\mathbf{A}_{\boldsymbol{\Xi}})$, similar to the expected update of $\boldsymbol{\theta}$. The analysis is largely the same the one described in section 3.1.4 and R. S. Sutton and Barto (2018), chapter 9.4, with similar convergence results. We therefore exclude similar details for brevity.

**Lemma 5.1.4.** *Given the one-step SF fixed point $\boldsymbol{\Xi}_{TD}$ and the reward regression solution $\hat{\mathbf{w}}$, we recover the* TD fixed-point*,*

$$\boldsymbol{\Xi}_{TD}\hat{\mathbf{w}} = \boldsymbol{\theta}_{TD}\,. \tag{5.25}$$

*In other words, the SF value and MF value (learned with on-policy one-step TD) have identical value estimates at their fixed points, $\boldsymbol{\Phi}\boldsymbol{\Xi}_{TD}\hat{\mathbf{w}} = \boldsymbol{\Phi}\boldsymbol{\theta}_{TD}$.*

*Proof.* By algebra,

$$\begin{aligned}\boldsymbol{\Xi}_{TD}\hat{\mathbf{w}} &= (\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi} - \gamma\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{P}_\pi\boldsymbol{\Phi})^{-1}(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi})\,(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{R}\,,\\ &= (\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi} - \gamma\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{P}_\pi\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{R} \;= \boldsymbol{\theta}_{TD}\,.\end{aligned}$$

□

Similarly, we have $\boldsymbol{\Xi}^\lambda_{TD}\hat{\mathbf{w}} = \boldsymbol{\theta}^\lambda_{TD}$.

**$\lambda$-VF Fixed Point** We now consider doing value learning with the $\lambda$-VF. One-step learning with the $\lambda$-VF target has the following update,

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha\left(R_{t+1} + \gamma\,(\boldsymbol{\psi}^\lambda_{t+1})^\top[(1-\lambda)\boldsymbol{\theta}_t + \lambda\mathbf{w}] - \boldsymbol{\phi}_t^\top\boldsymbol{\theta}_t\right)\boldsymbol{\phi}_t\,,\\ &= \boldsymbol{\theta}_t + \alpha\left(R_{t+1} + \gamma\,\boldsymbol{\phi}^\top_{t+1}\boldsymbol{\Xi}[(1-\lambda)\boldsymbol{\theta}_t + \lambda\mathbf{w}] - \boldsymbol{\phi}_t^\top\boldsymbol{\theta}_t\right)\boldsymbol{\phi}_t\,.\end{aligned}$$

Written in matrix form, the above iteration solves for the following,

$$\boldsymbol{\theta} = (\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{D}\left(\mathbf{R} + \gamma\mathbf{P}_\pi\boldsymbol{\Phi}[(1-\lambda)\boldsymbol{\Xi}\boldsymbol{\theta} + \lambda\boldsymbol{\Xi}\mathbf{w}]\right)\,. \tag{5.26}$$

We will now show the above system has the *TD fixed point*, $\boldsymbol{\theta}_{TD}$ as its fixed point as well. We first note a small identity.

**Lemma 5.1.5.** *Assuming the inverse of the SF parameters matrix $\boldsymbol{\Xi}^\lambda_{TD}$ exists, $(\boldsymbol{\Xi}^\lambda_{TD})^{-1} = (\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi})^{-1}(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi} - \lambda\gamma\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{P}_\pi\boldsymbol{\Phi})$, the following identity holds,*

$$(\boldsymbol{\Xi}^\lambda_{TD})^{-1}\boldsymbol{\theta}_{TD} = (1-\lambda)\boldsymbol{\theta}_{TD} + \lambda(\boldsymbol{\Xi}^\lambda_{TD})^{-1}\boldsymbol{\theta}^\lambda_{TD}\,. \tag{5.27}$$

*Proof.* The above identity can be re-arranged to the following form,

$$\lambda(\boldsymbol{\Xi}_{TD}^{\lambda})^{-1}\boldsymbol{\theta}_{TD}^{\lambda} = (\boldsymbol{\Xi}_{TD}^{\lambda})^{-1}\boldsymbol{\theta}_{TD} - (1-\lambda)\boldsymbol{\theta}_{TD}\,.$$

We now show the left- and right-hand sides are equivalent. We first evaluate the right hand side (r.h.s.), substituting in the fixed point equations,

$$\begin{aligned}
&(\boldsymbol{\Xi}_{TD}^{\lambda})^{-1}\cdot\boldsymbol{\theta}_{TD} - (1-\lambda)\boldsymbol{\theta}_{TD}\,,\\
=&\left((\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi})^{-1}(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi} - \lambda\gamma\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{P}_\pi\boldsymbol{\Phi}) - \mathbf{I} + \lambda\mathbf{I}\right)\boldsymbol{\theta}_{TD}\,,\\
=&\lambda\left(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi}\right)^{-1}(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi} - \gamma\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{P}_\pi\boldsymbol{\Phi})\,\boldsymbol{\theta}_{TD}\,,\\
=&\lambda\left(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi}\right)^{-1}(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi} - \gamma\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{P}_\pi\boldsymbol{\Phi})\,(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi} - \gamma\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{P}_\pi\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{R}\,,\\
=&\lambda\left(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{R}\,.
\end{aligned}$$

We now evaluate the left hand side (l.h.s.),

$$\begin{aligned}
&\lambda(\boldsymbol{\Xi}_{TD}^{\lambda})^{-1}\boldsymbol{\theta}_{TD}^{\lambda}\\
=&\lambda(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi})^{-1}(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi} - \lambda\gamma\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{P}_\pi\boldsymbol{\Phi})\,(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi} - \lambda\gamma\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{P}_\pi\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{R}\,,\\
=&\lambda(\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{R}\,.
\end{aligned}$$

The l.h.s. and r.h.s. are the same. Note that all steps are invertible. □

**Proposition 5.1.6.** *Given the SF parameter is at its $\lambda\gamma$-discounted fixed point, $\boldsymbol{\Xi}_{TD}^{\lambda}$, and the reward parameter is at its supervised regression solution, $\hat{\mathbf{w}}$, on-policy one-step learning with the $\lambda$-VF target has the TD fixed point as its fixed point,*

$$\boldsymbol{\theta}_{L,TD} = (\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi} - \gamma\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{P}_\pi\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{D}\mathbf{R} = \boldsymbol{\theta}_{TD}\,. \tag{5.28}$$

*Proof.* We first substitute the fixed points $\boldsymbol{\Xi}_{TD}^{\lambda}$ and $\hat{\mathbf{w}}$ into the system which $\lambda$-VF solves (equation 5.26),

$$\begin{aligned}
\boldsymbol{\theta}_{t+1} &= (\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{D}\left(\mathbf{R} + \gamma\mathbf{P}_\pi\boldsymbol{\Phi}[(1-\lambda)\boldsymbol{\Xi}_{TD}^{\lambda}\boldsymbol{\theta}_t + \lambda\boldsymbol{\Xi}_{TD}^{\lambda}\hat{\mathbf{w}}]\right)\,,\\
&= (\boldsymbol{\Phi}^\top\mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{D}\left(\mathbf{R} + \gamma\mathbf{P}_\pi\boldsymbol{\Phi}[(1-\lambda)\boldsymbol{\Xi}_{TD}^{\lambda}\boldsymbol{\theta}_t + \lambda\boldsymbol{\theta}_{TD}^{\lambda}]\right)\,.
\end{aligned}$$

The last step uses lemma 5.1.4, where $\boldsymbol{\Xi}^{\lambda}_{TD}\hat{\mathbf{w}} = \boldsymbol{\theta}^{\lambda}_{TD}$. We verify that $\boldsymbol{\theta}_{TD}$ is the fixed-point by considering the case where $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{TD}$,

$$
\begin{aligned}
\boldsymbol{\theta}_{t+1} &= (\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{D}\left(\mathbf{R} + \gamma \mathbf{P}_\pi \boldsymbol{\Phi}[(1-\lambda)\boldsymbol{\Xi}^{\lambda}_{TD}\boldsymbol{\theta}_{TD} + \lambda\boldsymbol{\theta}^{\lambda}_{TD}]\right), \\
&= (\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{D}(\mathbf{R} \\
&\qquad + \gamma \mathbf{P}_\pi \boldsymbol{\Phi}\, \boldsymbol{\Xi}^{\lambda}_{TD}\left[(1-\lambda)\boldsymbol{\theta}_{TD} + \lambda(\boldsymbol{\Xi}^{\lambda}_{TD})^{-1}\boldsymbol{\theta}^{\lambda}_{TD}\right]), \\
&= (\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{D}(\mathbf{R} + \gamma \mathbf{P}_\pi \boldsymbol{\Phi}\, \boldsymbol{\Xi}^{\lambda}_{TD}(\boldsymbol{\Xi}^{\lambda}_{TD})^{-1}\,\boldsymbol{\theta}_{TD}), && \text{[By lemma 5.1.5]} \\
&= (\boldsymbol{\Phi}^\top \mathbf{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{D}(\mathbf{R} + \gamma \mathbf{P}_\pi \boldsymbol{\Phi}\boldsymbol{\theta}_{TD}), \\
&= \boldsymbol{\theta}_{TD}. && \text{[By equation 5.15]}
\end{aligned}
$$

We see $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_{TD}$, therefore $\boldsymbol{\theta}_{TD}$ is the fixed point of the system. □

This result is perhaps unsurprising, as the one-step TD return is a special case of the one-step $\lambda$-VF target. For the case of SF learning, indeed it has also been shown that on-policy planning with linear models converges to the same fixed point as direct linear value estimation (Schoknecht 2002; Parr et al. 2008; R. Sutton et al. 2008). However, despite the fact that the final solution is subject to the same bias as one-step TD methods (e.g. see proposition 3.1.2), our method may still benefit from substantial learning efficiency while moving towards this solution. In fact, our finite sample empirical evaluation shows exactly this.

Finally, while we leave the convergence to the above fixed-point for future work, we note that we are leveraging multiple one-step learning processes that are known to converge: SF learning, reward learning, and value learning. Empirically, we observe learning with the one-step $\lambda$-VF target robustly reduces value error.

### 5.1.4 Linear Algorithm for Prediction

We can use any method for learning the $\lambda$-SF model $\boldsymbol{\psi}^{\lambda}_{\boldsymbol{\Xi}}$ and the instantaneous reward model $r_{\mathbf{w}}$. In this paper we make the choice of using TD(0) to learn the $\lambda$-SF model, and supervised regression for the reward model, since one-step methods are ubiquitous

in contemporary RL, and require the use of only *single-step* transitions (Mnih et al. 2015; H. v. Hasselt, Guez, and Silver 2015; Lillicrap et al. 2015; Z. Wang et al. 2016; Schaul et al. 2015; Haarnoja et al. 2018). Likewise, we use the $\lambda$-VF as a one-step bootstrap target (equation 5.9) for estimating of the value parameters $\boldsymbol{\theta}$ (equation 5.2).

**Algorithm 17:** One-step $\lambda$-VF target for policy evaluation with linear function approximation.

**Result:** Value function $v_{\boldsymbol{\theta}}(s) = \boldsymbol{\phi}(s)^\top \boldsymbol{\theta}$ for behavioural policy $\pi$
**Input:** MRP $\mathcal{M}_\pi$, feature set $\Phi$, discount factor $\gamma \in [0, 1)$, parameter $\lambda \in [0, 1]$, Stepsizes $\alpha_{\boldsymbol{\theta}}$, $\alpha_{\mathbf{w}}$, $\alpha_{\boldsymbol{\Xi}} \in (0, 1]$ ;
**Initialize:**
  Value function $v_{\boldsymbol{\theta}}(s) = \boldsymbol{\phi}(s)^\top \boldsymbol{\theta}$, with parameters $\boldsymbol{\theta}$ ;
  Reward function $r_{\mathbf{w}}(s) = \boldsymbol{\phi}(s)^\top \mathbf{w}$, with parameters $\mathbf{w}$ ;
  SF function $\boldsymbol{\psi}^{\lambda}_{\boldsymbol{\Xi}}(s) = \boldsymbol{\Xi}^\top \boldsymbol{\phi}(s)$, with parameters $\boldsymbol{\Xi}$ ;
**while** *sample one-step experience tuple*$(S_t, A_t, R_{t+1}, S_{t+1})$ *under policy* $\pi$ **do**
  $\boldsymbol{\Xi}_{t+1} \leftarrow \boldsymbol{\Xi}_t + \alpha_{\boldsymbol{\Xi},t} \left( \boldsymbol{\phi}(S_t) + \lambda\gamma \boldsymbol{\psi}^{\lambda}_{\boldsymbol{\Xi}_t}(S_{t+1}) - \boldsymbol{\Xi}^\top \boldsymbol{\phi}(S_t) \right) \boldsymbol{\phi}(S_t)^\top$ ; ▷ `SF learning update`
  $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha_{\mathbf{w},t} \left( R_{t+1} - \boldsymbol{\phi}(S_t)^\top \mathbf{w}_t \right) \boldsymbol{\phi}(s)$ ; ▷ `Reward learning update`
  $v^{\lambda}_{t+1} = \boldsymbol{\phi}(S_{t+1})^\top \boldsymbol{\Xi}_t \left( (1 - \lambda) \boldsymbol{\theta}_t + \lambda \mathbf{w}_t \right)$ ; ▷ $\lambda$`-VF target estimate`
  $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha_{\boldsymbol{\theta},t} \left( R_{t+1} + \gamma v^{\lambda}_{t+1} - \boldsymbol{\phi}(S_t)^\top \boldsymbol{\theta}_t \right) \boldsymbol{\phi}(S_t)$ ; ▷ `Value learning update`
**end**

All components of the $\lambda$-VF are learnable with one-step transitions tuples of the form $(S_t, A_t, R_{t+1}, S_{t+1})$, which make these methods amenable to both the online setting and the i.i.d. setting. In the former, the algorithm is presented with an infinite sequence of state, actions, rewards $\{S_0, A_0, R_1, S_1, A_1, R_2, \dots\}$, where $A_t \sim \pi(S_t)$, $R_{t+1} = r(S_t, A_t)$, $S_{t+1} \sim P(S_t, A_t)$. In the i.i.d. setting the learner is presented with a set of transition tuples $\{(S_t, A_t, R_{t+1}, S_{t+1})\}_{t \geq 0}$.

From an algorithmic perspective, algorithm 17 describes a computationally congenial way for learning the value function online from a single stream of experience using our method. As mentioned, in the online setting, the agent has access to experience in the form of tuples $(S_t, A_t, R_{t+1}, S_{t+1})$ at each timestep $t$. The pseudo-code describes

value estimation for the linear case, with given representations. Finally, although we have chosen to focus here on one-step targets for their simplicity and ease of use, these methods can be extended to multi-step targets with their multi-step counterparts.

### 5.1.5 Nonlinear Control Algorithm

We hypothesize that efficient value prediction of the $\lambda$-VF can help in value-based control, and extend our proposed algorithm to the control setting with estimation of the action-value function $q_{\boldsymbol{\theta}}$. We build on top of the deep Q network (DQN) architecture (Mnih et al. (2015), detailed in section 3.2.2) and simply replace the bootstrap target with a $\lambda$ action-value function. Concretely, given a sampled transition $(S_t, A_t, R_{t+1}, S_{t+1})$, DQN encodes features $\boldsymbol{\phi}(S_t) = \boldsymbol{\phi}_t$, then estimates the action-values $q_{\boldsymbol{\theta}}(\boldsymbol{\phi}_t, A_t) \approx q(S_t, A_t)$. We use to same feature encoding $\boldsymbol{\phi}(\cdot)$ to learn successor features $\boldsymbol{\psi}^{\lambda}_{\boldsymbol{\Xi}}(\boldsymbol{\phi}_t) \approx \boldsymbol{\psi}^{\lambda}_t$ and reward function $r_{\mathbf{w}}(\boldsymbol{\phi}_t)$. This allows us to construct the $\lambda$-VF target for the Q-learning update of parameters $\theta$,

$$q^{\lambda}(S_{t+1}, a'; \boldsymbol{\Xi}, \boldsymbol{\theta}, \mathbf{w}) = (1-\lambda) q_{\boldsymbol{\theta}}(\boldsymbol{\psi}_{\boldsymbol{\Xi}}(S_{t+1})^{\lambda}, a') + \lambda r_{\mathbf{w}}(\boldsymbol{\psi}_{\boldsymbol{\Xi}}(S_{t+1})^{\lambda}) \,, \tag{5.29}$$

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha(R_{t+1} + \gamma \max_{a'} q^{\lambda}(S_{t+1}, a'; \boldsymbol{\Xi}, \boldsymbol{\theta}, \mathbf{w}) - q_{\boldsymbol{\theta}}(\boldsymbol{\phi}_t, A_t)) \nabla_{\boldsymbol{\theta}} q_{\boldsymbol{\theta}}(\boldsymbol{\phi}_t, A_t)) \,, \tag{5.30}$$

where $q^{\lambda}(\cdot; \boldsymbol{\Xi}, \boldsymbol{\theta}, \mathbf{w})$ is the $\lambda$-value function target. We simultaneously estimate the representation and the action-values in an end-to-end fashion. The network architecture is illustrated in figure 5.1. See algorithm 18 for a complete pseudo-code description.

## 5.2 Experiments

We start with two simple prediction examples to provide intuition about our approach, after which, we verify that our method scales by extending it to a more complex non-linear control setting.

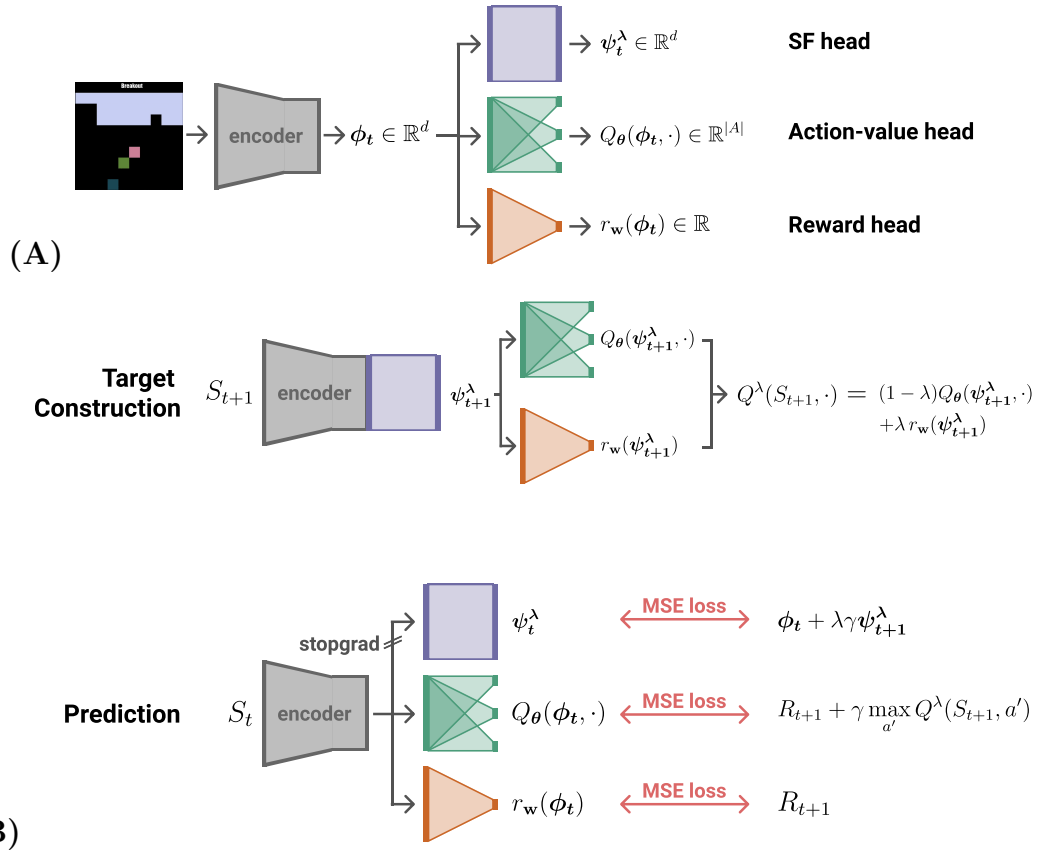


Figure 5.1: **Architecture for the $\lambda$-VF augmented Deep Q Network. (A) Base architecture**, we augment a DQN-like architecture (Mnih et al. 2015) (encoder and action-value head) with two additional heads for SF prediction and instantaneous reward prediction. **(B) Training with $\lambda$-VF**, given an experience tuple $(S_t, A_t, R_{t+1}, S_{t+1})$, we use $S_{t+1}$ to generate the $\lambda$-VF target $Q^\lambda(S_{t+1}, \cdot)$, and $S_t$ to generate the current predictions. Training is done by minimizing the (MSE) loss between the prediction and targets.

### 5.2.1 Value Prediction in Deterministic Chain

**Experiment setup** Consider the 16-state deterministic Markov reward process (MRP) with tabular features illustrated in figure 5.2-A. The agent starts in the left-most state ($s_0$), deterministically transitions right to the right-most absorbing state. The reward is 0 everywhere except for the final transition into the absorbing state, where it is +1. We apply algorithm 17 to estimate the value function in an online incremental setting. As everything is deterministic, we set the learning rate $\alpha = 1.0$ so new information can be learned right away. We use a discount factor of $\gamma = 0.9999$.

**Algorithm 18:** Deep Lambda Q Iteration

**Result:** Deep Q function $q_{\boldsymbol{\theta}}(\boldsymbol{\phi}_t, a)$ with encoder $\boldsymbol{\phi}_t \leftarrow \boldsymbol{\phi}_{\xi}(s_t)$ for control.

**Given functions:**

Feature encoder $\boldsymbol{\phi}_{\xi}(s_t) = \boldsymbol{\phi}_t$, where $\boldsymbol{\phi}_t \in \mathbb{R}^d$, with nonlinear parameters $\xi$ ;

Action value layer $q_{\boldsymbol{\theta}}(\boldsymbol{\phi}_t, a) = \boldsymbol{\phi}_t^{\top}\boldsymbol{\theta}_a$, where $q_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}^{|A|}$, with linear parameters $\boldsymbol{\theta}$ ;

Reward layer $\mathrm{r}_{\mathbf{w}}(\boldsymbol{\phi}_t) = \boldsymbol{\phi}_t^{\top}\mathbf{w}$, where $r_{\mathbf{w}} : \mathbb{R}^d \to \mathbb{R}$, with linear parameters $\mathbf{w}$ ;

SF layer $\boldsymbol{\psi}^{\lambda}_{\boldsymbol{\Xi}}(\boldsymbol{\phi}_t) = \boldsymbol{\Xi}^{\top}\boldsymbol{\phi}_t$, where $\boldsymbol{\psi}^{\lambda}_{\boldsymbol{\Xi}} : \mathbb{R}^d \to \mathbb{R}^d$, with linear parameters $\boldsymbol{\Xi}$;

**Given hyperparameters:** $\gamma \in [0, 1)$, $\lambda \in [0, 1]$

**for** *each environment step* **do**

▷ `Act in the environment`

Sample experience $(s_t, a_t, r_{t+1}, s_{t+1})$ from environment ;

Store to buffer $\mathcal{B} = \{\mathcal{B} \cup (s_t, a_t, r_{t+1}, s_{t+1})\}$ ;

▷ `Fitted Q Iteration`

Sample i.i.d. minibatch of size $n$ from buffer $\{(s_k, a_k, r_{k+1}, s_{k+1})_{i=1,\ldots,n}\} \sim \mathcal{B}$

**for** *each minibatch tuple* $(s_k, a_k, r_{k+1}, s_{k+1})_i$ **do**

Encode features: $\boldsymbol{\phi}_k$, $\boldsymbol{\phi}_{k+1} \leftarrow \boldsymbol{\phi}_{\xi}(s_k)$, $\boldsymbol{\phi}_{\xi}(s_{k+1})$ ;

Compute successor features: $\boldsymbol{\psi}_{k+1} \leftarrow \boldsymbol{\psi}^{\lambda}_{\boldsymbol{\Xi}}(\boldsymbol{\phi}_{k+1})$ ;

Copy feature with stop gradient (`sg`): $\boldsymbol{\phi}^{de}_k \leftarrow (\boldsymbol{\phi}_k)$`.sg()` ;

$\mathcal{L}_{S,i} = {}^1\!/_2 \left[[\boldsymbol{\phi}^{de}_k + \lambda\gamma\boldsymbol{\psi}_{k+1}]\texttt{.sg()} - \boldsymbol{\psi}^{\lambda}_{\boldsymbol{\Xi}}(\boldsymbol{\phi}^{de}_k)\right]^2$ ; ▷ `SF TD loss`

$\mathcal{L}_{R,i} = {}^1\!/_2 \left[r_{k+1} - \mathrm{r}_{\mathbf{w}}(\boldsymbol{\phi}_k)\right]^2$ ; ▷ `Reward supervised loss`

$q^{\lambda}(s_{k+1}, a') = (1-\lambda)q_{\theta}(\boldsymbol{\psi}_{k+1}, a') + \lambda \mathrm{r}_{\mathbf{w}}(\boldsymbol{\psi}_{k+1})$ ; ▷ $\lambda$ `Value Estimation`

$\mathcal{L}_{Q,i} = {}^1\!/_2 \left[r_{k+1} + \gamma \max_{a'} q^{\lambda}(s_{k+1}, a')\texttt{].sg()} - q_{\boldsymbol{\theta}}(s_k, a_k)\right]^2$ ; ▷ `Q learning loss`

**end**

$\mathcal{L}_{total} = \frac{1}{n}\sum_{i=1,\ldots,n} \mathcal{L}_{S,i} + \mathcal{L}_{R,i} + \mathcal{L}_{Q,i}$ ; ▷ `Overall minibatch loss`

$\xi, \boldsymbol{\theta}, \mathbf{w}, \boldsymbol{\Xi} \leftarrow$ `Optimizer`$(\mathcal{L}_{total})$ ; ▷ `Backprop and update parameters`

**end**

The main point here is to see the speed of best possible (one-step transition-based) credit propagation.

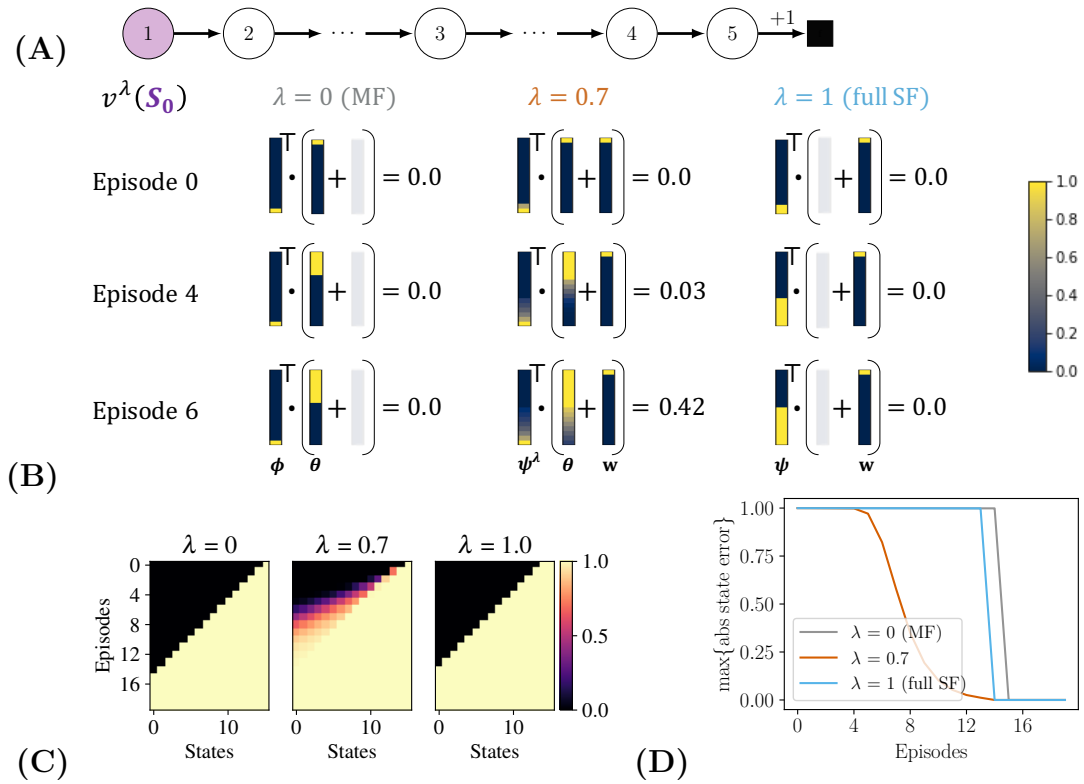


Figure 5.2: **Online value prediction in a deterministic MRP for different lambdas**. **(A)** The agent starts in the left-most state ($s_0$) and deterministically transitions right until reaching the terminal state. All rewards are 0, except for the terminal when it is +1. **(B) Parameter dynamics:** The table shows how $v^\lambda(S_0)$ is computed using $\boldsymbol{\psi}^\lambda(s_0)^\top$, $\boldsymbol{\theta}$ and $\mathbf{w}$ over the course of training for different values of $\lambda = \{0.0, 0.7, 1.0\}$. For $\lambda = 0.7$ (center) the $\lambda$-VF combines the parameters ($\boldsymbol{\theta}$) of the value function and the SF ($\boldsymbol{\psi}^\lambda$) predictions to more quickly propagate value information than either extremes. **(C) The estimated value function** for all states (columns) across learning episodes (rows). For $\lambda = 0.7$ information propagates value faster than $\lambda = 0$ and 1. **(D) Absolute value error:** for different $\lambda$ values over episodes. For $\lambda = 0.7$ error reduction is faster.

**Results** Figure 5.2-B illustrates the result of combining the successor features model $\boldsymbol{\psi}_\Xi$, with the value parameters $\boldsymbol{\theta}$, and reward parameters $\mathbf{w}$ into the value prediction $v^\lambda(s_0)$ for the starting state $s_0$, for different values of $\lambda$. In the pure model-free (MF) setting ($\lambda = 0$, identical to TD(0)), $\boldsymbol{\psi}^{\lambda=0} = \boldsymbol{\phi}$ corresponds to an unchanging feature representation, and value information (in $\boldsymbol{\theta}$) moves *backward* one state per episode. For the full successor feature (SF) value function ($\lambda = 1$), the instantaneous reward is learned immediately (parameter $\mathbf{w}$) for the final state, while the successor feature

(parameter $\boldsymbol{\psi}$) learns about one additional future state per episode. For both cases, we require $\sim 16$ episodes for the information to propagate across the entire chain and for the value estimate of $s_0$ to improve (Figure 5.2-D). However, given an intermediate value of $0 < \lambda < 1$ (Figure 5.2-B, middle, $\lambda = 0.7$ here), we are able to both propagate value information backward by bootstrapping on $\boldsymbol{\theta}$, as well as improve the predictive-features (using $\boldsymbol{\psi}^{\lambda}$) to predict farther in the forward direction. This results in an improved value estimate much earlier, as we can observe in figure 5.2-B middle, C middle, and D.

**Interpretation** In an online prediction setting, using the $\lambda$-VF (with an intermediate lambda $0 < \lambda < 1$) in place of the standard TD(0) target effectively combines both *backward* credit assignment by bootstrapping the value estimates, as well as *forward* feature prediction, to more quickly estimate the correct values.

### 5.2.2 Prediction in Random Chain

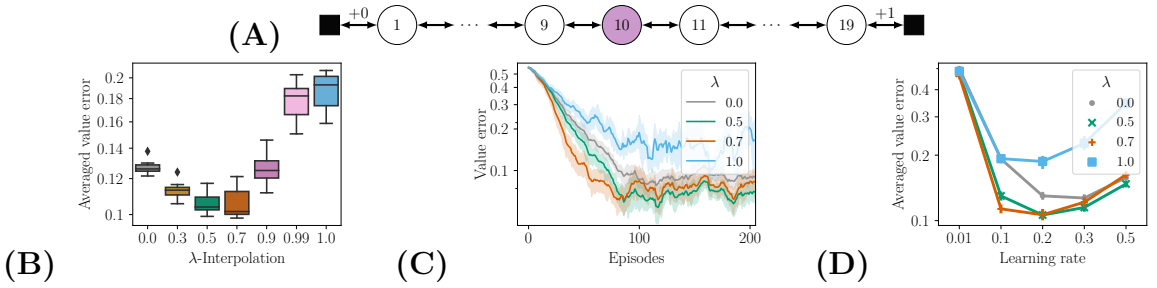


Figure 5.3: **Policy evaluation in 19-state tabular random chain. (A)** The agent starts in the center and transitions left/right randomly until either end is reached. Reward is 0 on all transitions, except the on the right-side termination, which yields a reward of +1. **(B) Parameter study for $\lambda$:** The y-axis shows the root mean squared error (RMSE) (minimized over learning rates for each $\lambda$) averaged over first 400 episodes. **(C) Learning dynamics:** The y-axis shows the RMSE for four illustrative $\lambda$ values. **(D) Parameter study for the learning rate** The y-axis shows the RMSE for four illustrative $\lambda$ values, across different learning rates. Results averages over first 400 episodes. Error bars and shaded areas denote 95 confidence intervals (some too small to see), with 10 independent seeds.

**Experiment setup** We now switch to a more difficult stochastic 19-state chain prediction task with tabular features (R. S. Sutton and Barto 2018, Example 6.2). The agent starts in the centre (state 10) and randomly transitions left or right until reaching the absorbing states at either end (figure 5.3-A). The reward is 0 everywhere except upon transitioning into the right-most terminal state, when it is +1. We train in the *online incremental* setting—the agent receives a stream of episodic experiences $(S_1, R_1, S_2, R_2, ...)$, and updates its parameter immediately upon receiving the most recent one-step experience tuple (for example, $(S_{t-1}, R_t, S_t)$ at timestep $t$). The hyperparameters sweep is done over the settings in table 5.1. Figure 5.3-B,D illustrate value error averaged over the first 400 episodes.

| Parameters | Parameter values |
|---|---|
| Value parameters learning rate, $\alpha_{\boldsymbol{\theta}}$ | Sweep over $\{0.01, 0.1, 0.2, 0.3, 0.5\}$ |
| Successor features learning rate, $\alpha_{\boldsymbol{\Xi}}$ | Always same as $\alpha_{\boldsymbol{\theta}}$ |
| Reward learning rate, $\alpha_{\mathbf{w}}$ | Always same as $\alpha_{\boldsymbol{\theta}}$ |
| $\lambda$ of $\lambda$-VF | Sweep over $\{0.0, 0.3, 0.5, 0.7, 0.9, 0.99, 1.0\}$ |
| Random seeds | $\{2, 4, 6, 8, 10, 12, 14, 16, 18, 20\}$ |

Table 5.1: Experimental parameters of random walk chain.

**Results** In figure 5.3-B, we observe that mixing with $\lambda \in [0, 1]$ results in a U-shape error curve, illustrating that an intermediate value of $\lambda$ is optimal. Note we plot the optimal learning rate $\alpha$ for each value of $\lambda$. Figure 5.3-C further confirms our hypothesis that an intermediate value (here for $\lambda = 0.5$ or 0.7) is most efficient. We also observe that intermediate $\lambda$ values show a degree of parameter robustness, having low value error over a range of different learning rates (figure 5.3-D).

**Interpretation** TD(0) using the $\lambda$-VF one-step target is robust to environment stochasticity and learns most efficiently for intermediate $\lambda$ values.

### 5.2.3 Nonlinear Control in Mini-Atari (MinAtar)

**Experiment Set-up** We test our algorithm in the Mini-Atari (MinAtar, Young and Tian 2019, GNU General Public License v3.0) environment, which is a smaller version of the Arcade Learning Environment (Bellemare et al. 2013) with 5 games (`asterix, breakout, freeway, seaquest, space invaders`) which are solved in the same way as their larger counterpart. We build our deep $\lambda$-VF agent on top of the DQN provided by (Young and Tian 2019) in `examples/dqn.py`.[7] We mimic the same DQN architecture (figure 5.1) and replicate to the best of our abilities the same hyperparameters as Young and Tian 2019, which was built to mimic the architecture and training procedure of the original DQN of (Mnih et al. 2015), albeit miniaturized for the smaller Atari environments. Unlike the original DQN, training is done every frame, using the PyTorch (Paszke et al. 2019) implementation of the RMSprop optimizer (Tieleman and G. Hinton 2012). Unless otherwise stated, we make no other changes (e.g. to policy, relay buffer, etc.), and use the same hyperparameters as the DQN in (Young and Tian 2019). Detailed training hyperparameters can be found in table 5.2.

Specifically, figure 5.4 follows exactly the hyperparameters reported in table 5.2, along with evaluations for a number of $\lambda$'s for a parameter study.[8] Each setting was conducted for 10 independent runs.[9] Figure 5.5 conducts a parameter study on the learning rates of the individual components of the $\lambda$-VF: the value head and convolutional torso ($\alpha_{\boldsymbol{\theta}}$, these two components make up exactly the "vanilla" DQN), the successor feature head ($\alpha_{\boldsymbol{\Xi}}$), and the reward prediction head ($\alpha_{\mathbf{w}}$). We investigated a range of learning rates,[10] and compare a $\lambda$-VF augmented Q network (with intermediate $\lambda = 0.4$) against a "vanilla" DQN network. Averaging is done during training by averaging over the episodic return of 10 episodes. The steps are "binned" into incre-

[7] GitHub commit:
https://github.com/kenjyoung/MinAtar/tree/8fceb584a00d86a3294c2d6ffb6fb8d93496b6a5

[8] Using $\lambda = \{0.0, 0.4, 0.5, 0.7, 0.95, 1.0\}$.

[9] With seeds $= \{2, 5, 8, 11, 14, 17, 20, 23, 26, 29\}$.

[10] Learning rates $= \{0.00025, 0.0005, 0.001, 0.0025, 0.005\}$.

| Hyperparameter | Value |
|---|---|
| discount factor ($\gamma$) | 0.99 |
| replay buffer memory size | 100000 |
| replay sampling minibatch size | 32 |
| $\epsilon$-greedy policy, initial $\epsilon$ | 1.0 |
| Initial random exploration | 5000 env steps |
| $\epsilon$-greedy policy, final $\epsilon$ | 0.1 |
| Initial to final $\epsilon$ anneal period | 100000 env steps |
| Target update period (per $n$ policy net updates) | 1000 |
| RMSprop momentum | 0.0 |
| RMSprop smoothing constant (alpha) | 0.95 |
| RMSprop eps (added to denominator for num stability) | 0.01 |
| RMSprop centered (normalized gradient) | True |
| Learning rate (conv torso and value head, $\alpha_{\boldsymbol{\theta}}$) | 0.00025 |
| Learning rate (SF head, $\alpha_{\boldsymbol{\Xi}}$) | 0.005 |
| Learning rate (reward head, $\alpha_{\mathbf{w}}$) | 0.005 |

Table 5.2: Default hyperparameters for DQN and $\lambda$-VF Q Network.

ments of length $1e4$ to account for the fact that different runs will generate episodic returns at different environmental steps, making it difficult to compute confidence interval in a "per-step" way. That is, the logged steps (x-axis of training plots) are rounded to the nearest multiple of $1e4$ for all runs.

**Intermediate $\lambda$ improves nonlinear control** Figure 5.4-A illustrates a parameter study on the mixing parameter $\lambda$ after training for 5 million environmental steps. We again observe the U-shaped performance curve as we interpolate across $\lambda$, confirming the advantage of using an intermediate $\lambda$ value. Figure 5.4-B shows the learning curves of our proposed model that uses an intermediate value of $\lambda$ in comparison to the two baseline algorithms: the model free algorithm ($\lambda = 0$, equivalent to vanilla DQN with a reward prediction auxiliary loss), and a value learning algorithm which uses the full SF value as its bootstrap target ($\lambda = 1$). The full SF baseline is remarkably unstable, while $\lambda$-VF with an intermediate $\lambda = 0.5$ outperforms both in $4/5$ games and is competitive with $\lambda = 0$ in `freeway`. It should be noted that the poor performance for higher $\lambda$ values in `freeway` is likely due to *sparse reward*, as the reward gradient

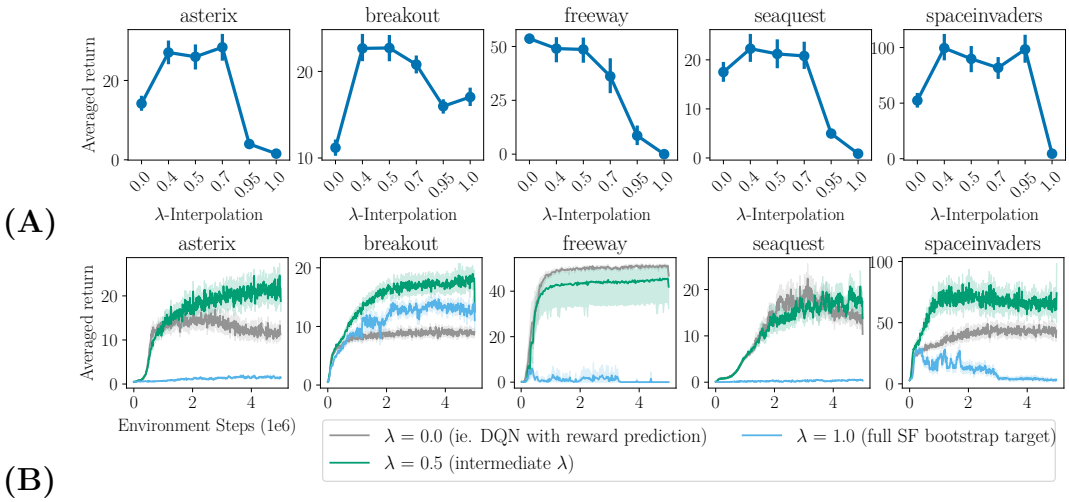


Figure 5.4: **Performance for value-based control in Mini-Atari**. **(A)** Parameter study for different values of $\lambda$. The y-axis shows the average performance over 10k timesteps and 10 seeds using an $\epsilon$-greedy policy with $\epsilon = 0.05$, after stopping training after $5e6$ learning steps. **(B)** Learning curves for 3 illustrative $\lambda$ values over the course of training. The y-axis displays the average return over 10 independent seed. Shaded area and error bars depicts 95 confidence interval.

used to shape the representation $\boldsymbol{\phi}(\cdot)$ is uninformative most of the time, leading to a collapse in representation (this is explicitly measured in section 5.2.4). This highlights a weakness of learning the feature encoding and successor features simultaneously, where poor features result in poor SF, and thus poor value estimates. The use of auxiliary losses can help ameliorate this issue (Machado, Bellemare, and Bowling 2020; Kumar et al. 2020), although it is not explored here as we found the issue to only be significant for high values of $\lambda$.

**Parameter study: robustness to SF and reward learning rates** Figure 5.5 shows parameter studies for an intermediate $\lambda$ that illustrate the sensitivity to the learning rates of the successor features and reward heads used in learning the value function. We vary the learning rates for these estimators while keeping the learning rates of the representation torso and the value function head fixed (at the same values used by (Young and Tian 2019) of $\alpha_{\boldsymbol{\theta}} = 2.5\text{e-}4$). We observe that performance is not highly dependent on the SF and reward learning rates (figure 5.5, green), but a higher

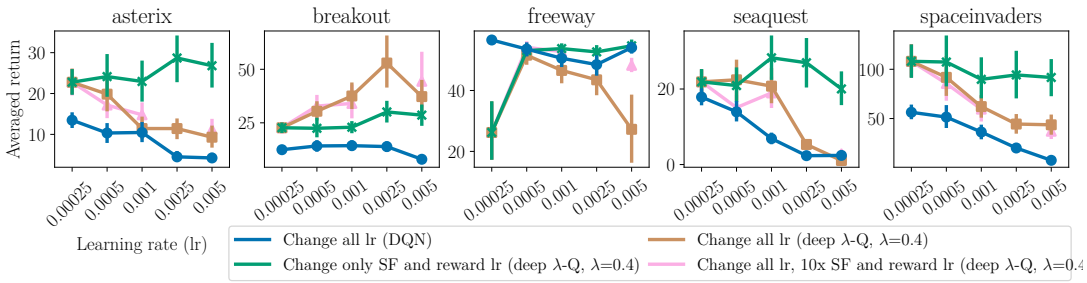


Figure 5.5: **Parameter study on the learning rates of the SF and instantaneous reward model:** The y-axis shows the average return over 10k evaluation steps using an $\epsilon$-greedy policy with $\epsilon = 0.05$, after stopping training after $5e6$ steps. For our algorithm, shown here as the Deep $\lambda$-Q algorithm (green), we sweep over the SF and reward learning rates while keeping the learning rates for the representation torso and the value function head fixed at 0.00025. For the vanilla DQN (blue), we vary the learning rates of the representation torso and the value function head. We also show the cumulative sensitivity to the parameters as we vary all the learning rates in our algorithm (yellow). Error bar denote 95 confidence intervals and each setting is ran using 3 independent seeds.

learning rate for the SF than the one used by the representation torso facilitates tracking the changes in the feature representations ($\phi$) by the SF. This choice is important in `freeway`. For comparison, we also sweep over the value and encoder learning rates of a vanilla DQN (figure 5.5, blue), and see that it is sensitive to the learning rate, i.e. performance drops as learning rate settings deviate from the recommendation of (Young and Tian 2019) (most prominently observed in `asterix`, `seaquest` and `space_invaders`, and for high learning rates in `breakout`). For additional ablation, we sweep over the learning rates of *all* parameters of the $\lambda$-Q function: either keeping all learning rates the same (figure 5.5, brown) or setting the successor feature and reward learning rates to be $10\times$ the encoder learning rates (figure 5.5, pink). Overall, we again observe that the agent is most sensitive to learning rates in the value head and encoder torso: performance decreases in all games other than `breakout`.

### 5.2.4 Feature Representation Collapse

For deep Q learning, we learn the feature representation $\boldsymbol{\phi}(\cdot)$ simultaneously to the successor features, action-values, and rewards (which are based on the learned feature layer). As our feature representation is shaped by back-propagated gradients from the action-value and reward heads (see figure 5.1 and algorithm 18), we measure the informativeness of the learned representation for different values of $\lambda$ (of the $\lambda$-VF). Concretely, we measure the *effective rank* (Yang et al. 2019; Kumar et al. 2020) of the feature learned after $5e6$ training environment steps, measured as $\texttt{srank}(\Phi) = \min\left\{k : \frac{\sum_{i=1}^{k}\sigma_i(\Phi)}{\sum_{i=1}^{d}\sigma_i(\Phi)} \geq 1-\delta\right\}$, with $\sigma_i(\Phi)$ being the $i$-th singular value of matrix $\Phi$, in decreasing order. We set $\delta = 0.01$ similar to (Kumar et al. 2020). Since we do not have access to the full feature matrix for MinAtar, we approximate $\Phi$ by sampling a large ($n = 2048$) minibatch of samples from the replay buffer and encoding them using the convolutions torso for a matrix $\hat{\Phi} \in \mathbb{R}^{n\times d}$, $n = 2048$, $d = 128$. We measure the averaged `srank` for 8 sampled minibatches per run, though standard deviation is low between the independently sampled minibatches.

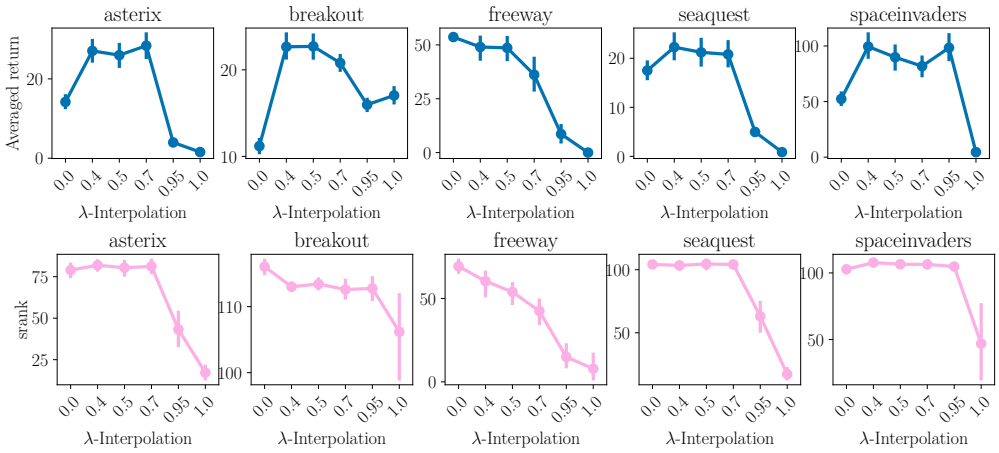


Figure 5.6: **MinAtar evaluation performance and *srank*.** **(Top)** Parameter study for different values of $\lambda$. Identical to figure 5.4-A. **(Bottom) SRank**. The y-axis displays the average srank over 10 independent seed. Shaded area and error bars depicts 95 confidence interval.

Figure 5.6 (bottom) reports the srank for the same models as figure 5.4 (we du-

plicate figure 5.4-A in the top row here). The maximum possible srank achievable is 128 (i.e. the feature dimension, $d$), with lower srank indicating the learned feature representation is less informative. We observe that in general, srank is high ($> 75$) and similar for $\lambda$'s up to $\lambda = 0.7$ (with the exception of `freeway`, to be discussed later). However, for high $\lambda$'s ($\lambda = \{0.95, 1.0\}$), we observe a decrease in srank for $^4/_5$ MinAtar games, with $\lambda = 1.0$ having srank's that tend towards 0. In the case of `freeway`, srank decreases monotically as we increase $\lambda$. We hypothesize this is the result of *sparse reward* for `freeway`. Since the feature layer is shaped in part by reward gradients, sparse reward may push the features to be less informative—an issue that is worsened as we depend more on the feature prediction rather than value prediction with higher $\lambda$'s.

Importantly, we observe the evaluation performance is related to the feature srank. Specifically, in cases where feature srank is similar, an intermediate $\lambda$ value outperforms "extreme" values of $\lambda$ (e.g. $\lambda = 0$). However, higher $\lambda$ appear to suffer from representation collapse which worsen performance, especially in sparse reward settings. The issue of learning good representation for successor feature learning can be addressed using auxiliary objectives (such as image reconstruction in (Kulkarni et al. 2016) or next-state prediction in (Machado, Bellemare, and Bowling 2020)). We leave the interplay between the $\lambda$-VF and additional feature-learning auxiliary tasks for future investigation.

## 5.3 Discussion on Machine Learning

### 5.3.1 Related Works

**Successor features**[11] (SF, equation 3.32) are an extension to the state-based successor representation (Dayan 1993), allowing for feature-based value functions to be

[11] We include this paragraph for completeness, noting it partially overlaps with section 3.4.2.

factored in a separately parameterized transition and reward model (Kulkarni et al. 2016; Lehnert and Littman 2020). A wide variety of uses have been proposed for the SF: aiding in exploration (Janz et al. 2019; Machado, Bellemare, and Bowling 2020), option discovery (Machado, Bellemare, and Bowling 2017; Machado, Rosenbaum, et al. 2018), and transferring across multiple goals (Lehnert, Tellex, and Littman 2017; Zhang et al. 2017; Ma et al. 2020; Brantley, Mehri, and Gordon 2021), in particular through the generalized policy improvement framework (Barreto, Dabney, et al. 2017; Barreto, Borsa, et al. 2018; Borsa et al. 2018; Hansen, Dabney, Barreto, Van de Wiele, et al. 2019; Grimm et al. 2019). Our method adds to this repertoire by using SF as models for implicit planning to achieve faster (*single-task*) value learning. This addition is interesting on its own since learning a (full) SF is typically slower than learning a value function directly (Lehnert, Tellex, and Littman 2017).

**Forward model-based planning** can facilitate efficient credit assignment. The algorithms that address this topic are Dyna-style methods which use explicit models to generate fictitious experience that they then leverage to improve the value function (Schoknecht 2002; Parr et al. 2008; R. Sutton et al. 2008; Yao, R. S. Sutton, et al. 2009). Closest to our method is the work by Yao, R. Sutton, et al. (2009) and Yao, R. S. Sutton, et al. (2009) which learns an explicit $\lambda$-model and uses it to generate fictitious experience for $k$-step updates to the value function. Our work is different in that the model we use in an implicit model used just to generate the value bootstrapping target. Furthermore, we extend our method to non-linear learned feature representations and combine it with batch learning algorithms (DQN) in MinAtar.

**Building state representation** is fundamental for deep RL. Our successor feature is a type of *general value function* (R. S. Sutton, Modayil, et al. 2011, section 3.4.1), which is hypothesized to be a core component in building internal representations of intelligent agents (R. S. Sutton, Modayil, et al. 2011; A. White 2015; Schlegel, Patterson, et al. 2018). Our work relates to this if we interpret the partial $\lambda$-SF model

as a new *learned* representation of the $\lambda$-VF.

### 5.3.2 Discussion and Future Directions

In this work we proposed an algorithm for implicit planning with a policy-dependent expectation model represented by the ($\lambda\gamma$-discounted) successor features. We used this implicit model to build a new target for the TD(0) algorithm, which we call $\lambda$-VF target. We showed that this method, while using the same amount of sampled experience, is more effective, resulting in more efficient value function estimation. The $\lambda$ value function we proposed can easily be used in place of the standard bootstrap target used in value-based algorithms, as we have illustrated in this work. A direct extension of this left for future work is the convergence of policy evaluation using the $\lambda$-VF target for linear function approximation, as well as its convergence rates, which can help better inform hyperparameter selection (in particular which intermediate $\lambda$ may be best suited for a particular environment). Moreover, our method is complementary to previously proposed approaches for value learning, such as multi-step returns or eligibility traces.

Many potential broad directions of investigation have also been opened for future work. (i) The $\lambda$-VF contains a successor feature estimate, which could also be further leveraged for exploration and transfer to achieve efficient single-task learning *and* multi-task transfer. (ii) Chelu, Precup, and H. V. Hasselt 2020 investigates the complementary properties of explicit forward and backward models and argues for potential of optimally combining both "forward" and "backward" facing credit assignment schemes. Further, H. v. Hasselt, Madjiheurem, et al. 2020 introduces expected eligibility traces as implicit backward models, generalizing predecessor features (time-reversed successor features). Future work can explore the differences and commonalities between implicit models in the forward and backward direction using our proposed $\lambda$-SF model and expected eligibility traces. The right balance between using

*backward* credit assignment through the use of eligibility traces, and *forward* prediction through predictive representations remains an open question with fundamental implications for learning efficiency. (iii) How to best use predictive representations to build an internal agent state is central to generalization and efficient credit assignment. Our work opens up many exciting new questions for investigation in this direction.

## 5.4 Discussion on Neuroscience

We briefly theorize how the *one-step λ-VF target* (equation 5.9) relate to brain observations. Subscribing again to the proposal of (Stachenfeld, M. M. Botvinick, and Gershman 2017) where the population activity of hippocampal place cells encode the successor representation of state $S_t$, it is sensible to assume that place fields may play the role of the SF in $\lambda$-VF. Similar to our prediction for the $\lambda$-SRE (chapter 4), we would expect the place field is again most helpful for *initial learning* of a novel task, and less so after the value function is well-learned. The transient involvement of the dorsal hippocampus for action learning again hints at this (Bradfield et al. 2020).

Unlike the $\lambda$-SRE, the predictive representation (SF) in the $\lambda$-VF is combined with value and reward information, as opposed to the TD error. Therefore, under the $\lambda$-VF framework, we might expect to observe co-activation of hippocampal place cells and neurons which represent value and reward information—possibly of the ventral (limbic) striatum, the ortibofrontal cortex, and/or the amygdala (Maia 2009).[12] The co-activity is a correlate for the $\lambda$-VF target and may also be correlated with synaptic plasticity in the above "value representation" areas as the target is constructed for value learning.

[12]To expand, Maia 2009 argues for these three areas being candidates of "value presentation" as they show neuronal activity when the animal expects to receive reward, and they project to and from the dopaminergic system which is thought to represent the TD error.

# 6

# Discussion

In this work, we have demonstrated how future predictive representations—in the form of the successor representation and successor features—can be used for better credit assignment and to more quickly estimate value functions.[1]

## 6.1 On Reinforcement Learning

The methods introduced in chapters 4 and 5 are both derived from the $\lambda$-return and learn SRs as "implicit models" over which multi-step credit assignment can occur. The SRs are factorized out of cumulative future quantities—*general value functions* (GVFs, section 3.4.1). Both GVFs have $\lambda\gamma$ as the discount factor, with different cumulants. For the $\lambda$-SRE (chapter 4), the cumulant is the one-step TD error given the current value parameters; for the $\lambda$-VF (chapter 5), the cumulant is a linear mixture of the current reward and value parameters. For both cases, the GVF plays an explicit role in value learning: the $\lambda$-SRE is the error the value function tries to minimize, while the $\lambda$-VF is part of the learning target.

Our method is orthogonal but complementary to the "usual" uses of the SRs, such as for *transfer* where a new value can be quickly reevaluated by combining a new reward function with a pre-learned SR (e.g. Barreto, Dabney, et al. (2017)).

[1] We will henceforth refer to both the successor representation and the successor features as "SR" in this chapter for brevity, as both serve the same purpose in this discussion.

This is useful in the multi-task settings if the SR have been pre-learned. In contrast, we propose ways of using the SR in the single task setting without any prior learning. Future work can explore how to best combine our single-task method with multi-task uses of the SR.

Moreover, we suspect the credit assignment benefit from our approaches arises precisely from the decomposition of the SR from the aforementioned GVFs: it allows for separate learning of a "fixed" / "slow-moving" quantity in the form of the SR (which summarize over the dynamics of the current policy), and "quickly-changing" quantities that depend on the policy dynamics (i.e. the value parameters or TD errors, which changes per value update), to be combined to recover the original GVF. We suspect this may be a general principle: in the case where the non-SR quantity is quickly-changing (and possibly non-stationary) but readily accessible, independently learning the SR and the quickly-changing quantity allows for faster learning of the overall GVF. Indeed, we observe the same process for policy evaluation in a *multi-task* setting: the decomposition of a value function into SR and reward parameters means policy evaluation can be quickly done for each task, despite the reward parameters changing quickly between tasks. In contrast, this use of the SR for the single-task setting is unhelpful: supervised learning of the reward parameters is stable and slower-moving than the SR. As SRs are generally decomposable from GVFs, we believe future work can further explore the construction of other kinds of GVFs for which the SR decomposition will help in learning.

Finally, our method relates broadly to multi-step methods, such as the $\beta$-models of R. S. Sutton (1995), $\lambda$-policy iteration of Bertsekas and Ioffe (1996), and ultimately, Van Nunen (1976).[2] One can also view this work, in particular chapter 4, through the lens of matrix preconditioning (Bacon 2018). These connections were not explored in depth within this thesis, but can be fruitful grounds for future work.

[2]See Bacon (2018), chapter 4.8, for a detailed discussion of the historical context of related multi-step methods.

## 6.2 On Neuroscience

### Value representation in the brain

Value prediction is central to RL, but how is value encoded in the brain? There is no definitive answer, although Maia (2009) proposes that the brain area encoding value must show the following characteristics: (i) exhibit increased activity during the *expectation* of reward, and (ii) project to and from the dopaminergic system, where the TD error is thoughts to be encoded (since value representation and TD error is closely related). Three brain areas satisfy the above conditions: the ventral striatum, the orbitofrontal cortex, and the amygdala. Similarly, Takahashi, Schoenbaum, and Niv (2008) maps the actor-critic architecture onto neuroanatomy, where the critic is encoded by the activity of the ventral striatum.

If value is indeed encoded in the above areas, what is the computational interaction between value and the hippocampal place cells (which Stachenfeld, M. M. Botvinick, and Gershman (2017) theorizes encode SRs; it is also worth noting that anatomically, the hippocampus project directly to all three aforementioned areas, for example see Meer et al. (2014), Zhong, Yukie, and Rockland (2006), and Pitkänen et al. (2000))? If we take the "classical" view that SRs directly parameterizes the value function (through linearly combining with rewards, see Dayan (1993)), we expect place cell activities to be multiplied with the reward estimates of each state and summed to produce value. This is plausible, albeit difficult to justify from a *performance-driven* perspective. In chapter 4 we observe small errors in the SR leads to large errors in the value estimate, while in chapter 5 using the SR directly for value computation results in a difficult feature learning problem.[3] Traditionally, a normative argument for having SRs exists only for the restricted setting of multi-task learning with similar

[3]It should be noted that feature learning may take place separate from value learning, which makes this less of an issue, although with linear function approximation where feature do not have to be learned, value estimation with "full" SR value is still the least sample efficient, see figure 5.3, high $\lambda$'s.

(environmental) dynamics.

Our current work proposes a entirely new set of hypothesis: that the SRs are not useful in parameterizing the prediction or control function, but instead most helpful for *learning* such functions.

## Learning from SR-based information propagation

Our proposed methods leverage SRs to propagate information in a way that is compliant to the environmental structure. The $\lambda$-SRE implies the following experimental predictions: the hippocampal place fields will co-activated with the TD errors in order to re-weigh and propagate them. There are two ways this can happen: the current TD error can be propagated to the other states in the form of source traces (Pitis 2018), or, arbitrary states transitions can be sampled (e.g. via replaying trajectories from memory) to compute TD errors.

On the other hand, the $\lambda$-VF predicts co-activation between hippocampal place fields and value representations. Experimental evidence indeed supports the interaction between the hippocampus and value-encoding areas. For instance, F. Wang, Schoenbaum, and Kahnt (2020) use functional magnetic resonance imaging (fMRI) in human subjects to demonstrate correlated activity between the hippocampus and orbitofrontal cortex is associated with using future state-prediction to estimate value. Similarly, electrophysiological evidence from Lansink, Goltstein, Lankelma, Joosten, et al. (2008) and Lansink, Goltstein, Lankelma, McNaughton, et al. (2009) show in rats how the ventral striatal neural activity tends to be coupled with hippocampal activity in sleep and relates to the ability to learn reward-based tasks.

More broadly speaking, there is a fundamental link between SR for credit assignment and "model-based" inference: the SR *is* a model, more specifically a compressed, policy-dependent expectation model storing future state information. The SR is also readily computable *if* one has access to a one-step transition model (i.e. since it can

be written as a summation, see proposition 3.3.1). In fact, using a one-step model to dynamically construct an SR requires "rolling-out" the transition model, which can correspond to the observation of "replay". Similarly, neuroscience evidence has implicated the hippocampus in both "model-based" inference and replay (McDannald et al. 2011; Stoianov et al. 2018; Vikbladh et al. 2019). We hope to motivate a more precise and nuanced discussion and future investigations on how a transition model may be stored and used in the brain.[4]

## Reward prediction in the brain

The highly influential paper of Schultz, Dayan, and Montague (1997) relates the relationship of the temporal difference error (Samuel 1959; R. S. Sutton 1988) to the activities of the dopamergic neurons in the ventral tegmental area (VTA) and substantia nigra areas of the brain, thus proposing a link between neurobiological response and a computational principle. This sub-field of science remains active today (e.g. Dabney et al. (2020)).

While neuroscientists have some ideas for where value and TD error might be encoded in the brain, it is interesting to note that no commonly accepted theory exists for how *value learning* is implemented in neurobiology. Our proposed algorithm offers one additional hypothesis.

## A framework for understanding the mind

Finally, we take a step back and look at the broader quest to understand the mind. Ultimately, to try to understand the mind means tackling the complexity of the biological brain. While the data collection and analysis methods of contemporary neuroscience have grown increasingly sophisticated in recent years, it is arguable if increasingly sophisticated methods and more massive datasets is sufficient for an understanding of

[4] In fact, recent RL results demonstrates that different ways of using the *same* transition model lead to drastically different performances (Chelu, Precup, and H. V. Hasselt 2020).

the brain. For example, Jonas and Kording (2017) shows contemporary neuroscience methods, even without restriction to data, cannot provide a satisfactory understanding of a computer microprocessor. Arguably, the greatest breakthroughs in brain understanding have often came when new theories have given us the perspective, language and conceptual framework to make sense of the data. The theory of dopamine as reward prediction error (Schultz, Dayan, and Montague 1997) relied on the existence of the TD algorithm (R. S. Sutton 1988); the future-predictive theory of the hippocampal place fields (Stachenfeld, M. M. Botvinick, and Gershman 2017) relied on the derivation of the successor representation (Dayan 1993); and the engineering challenges of getting convolutional neural networks to work well had to be tackled (Krizhevsky, Sutskever, and G. E. Hinton 2012) before one can make sense of the population representation in the primate visual cortex by comparing to ANNs (Yamins et al. 2014).[5] The problem, ultimately, is that the space of theoretical explanation is large: there is a degenerate mapping from the space of theories to the space of neural-observation, such that inferring brain computation from empirical observation is an underdetermined problem.[6]

Thus, if we do not sufficiently explore the space of theories, the neural-data alone may never give us the correct answer. Akin to a drunkard only searching for his keys under a street lamp,[7] the complexity of the brain means that we will only be able to find what we actively look for and what makes sense to us, while the rest will

[5]More generally speaking there has been a general trend in recent years of comparing performance-optimized neural net representation with brain recorded representations, all with promising results. This framework is described in Richards et al. (2019).

[6]To make matters even more complicated, the *same* computational principle is similarly multiply-realizable, such as the stomatogastric ganglion in crabs responsible for their chewing rhythm being implementable through a wide range of possible three-cell network motifs (Prinz, Bucher, and Marder 2004).

[7]See Freedman, David H. "Why scientific studies are so often wrong: The streetlight effect" Discover Magazine 26 (2010): 1-4. Although this analogy likely goes back much further. The joke goes: *Late at night, a police officer finds a drunk man crawling around on his hands and knees under a streetlight. The drunk man tells the officer he is looking for his keys, and they both look under the streetlight together. After a few minutes the policeman asks if he is sure he lost them here, and the drunk replies, no, and that he lost them in the park. The befuddled officer asks why he is searching here, and the drunk replies, 'this is where the light is'.*

simply appear as noise. The role of theoretical development, therefore, is to *render the invisible visible* (Gershman 2021). This thesis is an attempt to make a small stride in this direction, in adding to the set of theoretical consideration for what future-predictive representations may be used for in the brain. We hope we have increased the cone onto which the street lamp shines, even if only by a little bit.

# Bibliography


Alvernhe, Alice, Etienne Save, and Bruno Poucet (2011). "Local remapping of place cell firing in the Tolman detour task". In: *European Journal of Neuroscience* 33.9, pp. 1696–1705.

Bacon, Pierre-Luc (2018). *Temporal Representation Learning*. McGill University (Canada).

Banach, Stefan (1922). "Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales". In: *Fund. math* 3.1, pp. 133–181.

Barnard, Etienne (1993). "Temporal-difference methods and Markov models". In: *IEEE Transactions on Systems, Man, and Cybernetics* 23.2, pp. 357–365.

Barreto, Andre, Diana Borsa, et al. (2018). "Transfer in deep reinforcement learning using successor features and generalised policy improvement". In: *International Conference on Machine Learning*. PMLR, pp. 501–510.

Barreto, André, Will Dabney, et al. (2017). "Successor Features for Transfer in Reinforcement Learning." In: *NIPS*.

Bellemare, Marc G et al. (2013). "The arcade learning environment: An evaluation platform for general agents". In: *Journal of Artificial Intelligence Research* 47, pp. 253–279.

Bellman, Richard (1957). "A Markovian Decision Process". In: *Indiana Univ. Math. J.* 6 (4), pp. 679–684.

Bengio, Emmanuel, Joelle Pineau, and Doina Precup (2021). "Correcting Momentum in Temporal Difference Learning". In: *arXiv preprint arXiv:2106.03955*.

— (2020). "Interference and generalization in temporal difference learning". In: *International Conference on Machine Learning*. PMLR, pp. 767–777.

Bertsekas, Dimitri P and Sergey Ioffe (1996). "Temporal differences-based policy iteration and applications in neuro-dynamic programming". In: *Lab. for Info. and Decision Systems Report LIDS-P-2349, MIT, Cambridge, MA* 14.

Bertsekas, Dimitri P and John N Tsitsiklis (1995). "Neuro-dynamic programming: an overview". In: *Proceedings of 1995 34th IEEE conference on decision and control*. Vol. 1. IEEE, pp. 560–564.

Blier, Léonard, Corentin Tallec, and Yann Ollivier (2021). "Learning Successor States and Goal-Dependent Values: A Mathematical Viewpoint". In: *arXiv preprint arXiv: 2101. 07123*.

Blundell, Charles et al. (2016). "Model-free episodic control". In: *arXiv preprint arXiv:1606.04460*.

Borsa, Diana et al. (2018). "Universal Successor Features Approximators". In: *International Conference on Learning Representations*.

Bradfield, Laura A et al. (2020). "Goal-directed actions transiently depend on dorsal hippocampus". In: *Nature Neuroscience*, pp. 1–4.

Brantley, Kianté, Soroush Mehri, and Geoffrey J Gordon (2021). "Successor Feature Sets: Generalizing Successor Representations Across Policies". In: *arXiv preprint arXiv:2103.02650*.

Chelu, Veronica, Doina Precup, and H. V. Hasselt (2020). "Forethought and Hindsight in Credit Assignment". In: *ArXiv* abs/2010.13685.

Chung, Wesley et al. (2018). "Two-timescale networks for nonlinear value function approximation". In: *International conference on learning representations*.

Dabney, Will et al. (2020). "A distributional code for value in dopamine-based reinforcement learning". In: *Nature* 577.7792, pp. 671–675.

Dayan, Peter (1993). "Improving generalization for temporal difference learning: The successor representation". In: *Neural Computation* 5.4, pp. 613–624.

Ernst, Damien, Pierre Geurts, and Louis Wehenkel (2005). "Tree-based batch mode reinforcement learning". In: *Journal of Machine Learning Research* 6, pp. 503–556.

François-Lavet, Vincent et al. (2018). "An introduction to deep reinforcement learning". In: *arXiv preprint arXiv:1811.12560*.

Gershman, Samuel J (2021). "Just looking: The innocent eye in neuroscience". In: *Neuron*.

— (2018). "The successor representation: its computational logic and neural substrates". In: *Journal of Neuroscience* 38.33, pp. 7193–7200.

Gershman, Samuel J and Nathaniel D Daw (2017). "Reinforcement learning and episodic memory in humans and animals: an integrative framework". In: *Annual review of psychology* 68, pp. 101–128.

Grimm, Christopher et al. (2019). "Disentangled cumulants help successor representations transfer to new tasks". In: *arXiv preprint arXiv:1911.10866*.

Gupta, Anoopum S et al. (2010). "Hippocampal replay is not a simple function of experience". In: *Neuron* 65.5, pp. 695–705.

Haarnoja, Tuomas et al. (2018). "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International Conference on Machine Learning*. PMLR, pp. 1861–1870.

Hansen, Steven, Will Dabney, Andre Barreto, Tom Van de Wiele, et al. (2019). "Fast task inference with variational intrinsic successor features". In: *arXiv preprint arXiv:1906.05030*.

Hansen, Steven, Will Dabney, Andre Barreto, David Warde-Farley, et al. (2019). "Fast Task Inference with Variational Intrinsic Successor Features". In: *International Conference on Learning Representations*.

Hasselt, Hado van, Arthur Guez, and David Silver (2015). "Deep reinforcement learning with double Q-learning. CoRR abs/1509.06461 (2015)". In: *arXiv preprint arXiv:1509.06461*.

Hasselt, Hado van, Sephora Madjiheurem, et al. (2020). "Expected Eligibility Traces". In: *arXiv preprint arXiv:2007.01839*.

Hollup, Stig A et al. (2001). "Accumulation of hippocampal place fields at the goal location in an annular watermaze task". In: *Journal of Neuroscience* 21.5, pp. 1635–1644.

Janner, Michael, Igor Mordatch, and Sergey Levine (2020). "$\gamma$-Models: Generative Temporal Difference Learning for Infinite-Horizon Prediction". In: *arXiv preprint arXiv:2010.14496*.

Janz, David et al. (2019). "Successor uncertainties: exploration and uncertainty in temporal difference learning". In: *Advances in Neural Information Processing Systems* 32, pp. 4507–4516.

Jonas, Eric and Konrad Paul Kording (2017). "Could a neuroscientist understand a microprocessor?" In: *PLoS computational biology* 13.1, e1005268.

Kempadoo, Kimberly A et al. (2016). "Dopamine release from the locus coeruleus to the dorsal hippocampus promotes spatial learning and memory". In: *Proceedings of the National Academy of Sciences* 113.51, pp. 14835–14840.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25, pp. 1097–1105.

Kulkarni, Tejas D et al. (2016). "Deep successor reinforcement learning". In: *arXiv preprint arXiv:1606.02396*.

Kumar, Aviral et al. (2020). "Implicit Under-Parameterization Inhibits Data-Efficient Deep Reinforcement Learning". In: *arXiv preprint arXiv:2010.14498*.

Kumaran, Dharshan, Demis Hassabis, and James L McClelland (2016). "What learning systems do intelligent agents need? Complementary learning systems theory updated". In: *Trends in cognitive sciences* 20.7, pp. 512–534.

Lansink, Carien S, Pieter M Goltstein, Jan V Lankelma, Ruud NJMA Joosten, et al. (2008). "Preferential reactivation of motivationally relevant information in the ventral striatum". In: *Journal of Neuroscience* 28.25, pp. 6372–6382.

Lansink, Carien S, Pieter M Goltstein, Jan V Lankelma, Bruce L McNaughton, et al. (2009). "Hippocampus leads ventral striatum in replay of place-reward information". In: *PLoS biology* 7.8, e1000173.

Lehnert, Lucas and Michael L Littman (2020). "Successor features combine elements of model-free and model-based reinforcement learning". In: *Journal of Machine Learning Research* 21.196, pp. 1–53.

Lehnert, Lucas, Stefanie Tellex, and Michael L Littman (2017). "Advantages and limitations of using successor features for transfer in reinforcement learning". In: *arXiv preprint arXiv:1708.00102*.

Lengyel, Máté and Peter Dayan (2008). "Hippocampal contributions to control: the third way". In: *Advances in neural information processing systems*, pp. 889–896.

Lillicrap, Timothy P et al. (2015). "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971*.

Littman, Michael L, Richard S Sutton, and Satinder P Singh (2001). "Predictive Representations of State". In: *Advances in Neural Information Processing Systems* 14, pp. 1555–1561.

Ma, Chen et al. (2020). "Universal successor features for transfer reinforcement learning". In: *arXiv preprint arXiv:2001.04025*.

Machado, Marlos C, Marc G Bellemare, and Michael Bowling (2017). "A laplacian framework for option discovery in reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pp. 2295–2304.

Machado, Marlos C, Marc G Bellemare, and Michael Bowling (2020). "Count-based exploration with the successor representation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04, pp. 5125–5133.

Machado, Marlos C, Clemens Rosenbaum, et al. (2018). "Eigenoption Discovery through the Deep Successor Representation". In: *International Conference on Learning Representations*.

Maia, Tiago V (2009). "Reinforcement learning, conditioning, and the brain: Successes and challenges". In: *Cognitive, Affective, & Behavioral Neuroscience* 9.4, pp. 343–364.

Mattar, Marcelo G and Nathaniel D Daw (2018). "Prioritized memory access explains planning and hippocampal replay". In: *Nature neuroscience* 21.11, pp. 1609–1617.

McCarthy, John et al. (1955). "A proposal for the dartmouth summer research project on artificial intelligence". In:

McClelland, James L, Bruce L McNaughton, and Randall C O'Reilly (1995). "Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory." In: *Psychological review* 102.3, p. 419.

McClelland, James L, David E Rumelhart, PDP PDP Research Group, et al. (1986). *Parallel distributed processing*. Vol. 2. MIT press Cambridge, MA.

McDannald, Michael A et al. (2011). "Ventral striatum and orbitofrontal cortex are both required for model-based, but not model-free, reinforcement learning". In: *Journal of Neuroscience* 31.7, pp. 2700–2705.

Meer, Matthijs AA van der et al. (2014). "Hippocampal projections to the ventral striatum: from spatial memory to motivated behavior". In: *Space, Time and Memory in the Hippocampal Formation*. Springer, pp. 497–516.

Mehta, Mayank R, Michael C Quirk, and Matthew A Wilson (2000). "Experience-dependent asymmetric shape of hippocampal receptive fields". In: *Neuron* 25.3, pp. 707–715.

Minsky, Marvin (1961). "Steps toward artificial intelligence". In: *Proceedings of the IRE* 49.1, pp. 8–30.

Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pp. 529–533.

Momennejad, Ida (2020). "Learning structures: Predictive representations, replay, and generalization". In: *Current Opinion in Behavioral Sciences* 32, pp. 155–166.

Momennejad, Ida, A Ross Otto, et al. (2018). "Offline replay supports planning in human reinforcement learning". In: *Elife* 7, e32548.

Momennejad, Ida, Evan M Russek, et al. (2017). "The successor representation in human reinforcement learning". In: *Nature human behaviour* 1.9, pp. 680–692.

O'Keefe, John and Jonathan Dostrovsky (1971). "The hippocampus as a spatial map: Preliminary evidence from unit activity in the freely-moving rat." In: *Brain research*.

Parr, Ronald E. et al. (2008). "An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning". In: *ICML '08*.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 8024–8035.

Pavlov, Ivan P (1927). *Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex*.

Pitis, Silviu (2018). "Source traces for temporal difference learning". In: *Thirty-Second AAAI Conference on Artificial Intelligence*.

Pitkänen, Asla et al. (2000). "Reciprocal connections between the amygdala and the hippocampal formation, perirhinal cortex, and postrhinal cortex in rat: a review". In: *Annals of the new York Academy of Sciences* 911.1, pp. 369–391.

Prinz, Astrid A, Dirk Bucher, and Eve Marder (2004). "Similar network activity from disparate circuit parameters". In: *Nature neuroscience* 7.12, pp. 1345–1352.

Puterman, Martin L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. USA: John Wiley & Sons, Inc.

Rescorla, R. and Allan Wagner (Jan. 1972). "A theory of Pavlovian conditioning: The effectiveness of reinforcement and non-reinforcement". In: *Classical Conditioning: Current Research and Theory*.

Richards, Blake A et al. (2019). "A deep learning framework for neuroscience". In: *Nature neuroscience* 22.11, pp. 1761–1770.

Riedmiller, Martin (2005). "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method". In: *European Conference on Machine Learning*. Springer, pp. 317–328.

Romoff, Joshua et al. (2020). "TDprop: Does Jacobi Preconditioning Help Temporal Difference Learning?" In: *arXiv preprint arXiv:2007.02786*.

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1985). *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science.

Russek, Evan M et al. (2017). "Predictive representations can link model-based reinforcement learning to model-free mechanisms". In: *PLoS computational biology* 13.9, e1005768.

Samuel, Arthur L (1959). "Some studies in machine learning using the game of checkers". In: *IBM Journal of research and development* 3.3, pp. 210–229.

Schaul, Tom et al. (2015). "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952*.

Schlegel, M., Andrew Patterson, et al. (2018). "Discovery of Predictive Representations With a Network of General Value Functions". In:

Schlegel, Matthew, Andrew Jacobsen, et al. (2021). "General value function networks". In: *Journal of Artificial Intelligence Research* 70, pp. 497–543.

Schoknecht, Ralf (2002). "Optimality of Reinforcement Learning Algorithms with Linear Function Approximation". In: *NIPS*.

Schultz, Wolfram (2016). "Dopamine reward prediction error coding". In: *Dialogues in clinical neuroscience* 18.1, p. 23.

Schultz, Wolfram, Peter Dayan, and P Read Montague (1997). "A neural substrate of prediction and reward". In: *Science* 275.5306, pp. 1593–1599.

Silver, David et al. (2021). "Reward is enough". In: *Artificial Intelligence*, p. 103535.

Stachenfeld, Kimberly L, Matthew Botvinick, and Samuel J Gershman (2014). "Design principles of the hippocampal cognitive map". In: *Advances in neural information processing systems* 27, pp. 2528–2536.

Stachenfeld, Kimberly L, Matthew M Botvinick, and Samuel J Gershman (2017). "The hippocampus as a predictive map". In: *Nature neuroscience* 20.11, pp. 1643–1653.

Stoianov, Ivilin Peev et al. (2018). "Model-based spatial navigation in the hippocampus-ventral striatum circuit: A computational analysis". In: *PLoS computational biology* 14.9, e1006316.

Sutton, R. et al. (2008). "Dyna-Style Planning with Linear Function Approximation and Prioritized Sweeping". In: *UAI*.

Sutton, Richard S (1990). "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming". In: *Machine learning proceedings 1990*. Elsevier, pp. 216–224.

— (1988). "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1, pp. 9–44.

— (1995). "TD models: Modeling the world at a mixture of time scales". In: *Machine Learning Proceedings 1995*. Elsevier, pp. 531–539.

Sutton, Richard S and Andrew G Barto (1987). "A temporal-difference model of classical conditioning". In: *Proceedings of the ninth annual conference of the cognitive science society*. Seattle, WA, pp. 355–378.

Sutton, Richard S and Andrew G Barto (1998). *Introduction to reinforcement learning*. Vol. 135. EBook url: http://incompleteideas.net/book/first/ebook/the-book.html, Accessed: 2021-04-09. MIT press Cambridge.

— (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, Richard S, Joseph Modayil, et al. (2011). "Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction". In: *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 761–768.

Sutton, Richard Stuart (1984). "Temporal credit assignment in reinforcement learning". PhD thesis. University of Massachusetts Amherst.

Takahashi, Yuji, Geoffrey Schoenbaum, and Yael Niv (2008). "Silencing the critics: understanding the effects of cocaine sensitization on dorsolateral and ventral striatum in the context of an actor/critic model". In: *Frontiers in neuroscience* 2, p. 14.

Tieleman, Tijmen and Geoffrey Hinton (2012). "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* 4.2, pp. 26–31.

Turing, AM (1950). "Computing Machinery and Intelligence". In: *Mind* 59.236, pp. 433–460.

Van Nunen, J.A.E.E. (1976). "Contracting Markov decision processes". PhD thesis. Eindhoven University of Technology.

Vikbladh, Oliver M et al. (2019). "Hippocampal contributions to model-based planning and spatial memory". In: *Neuron* 102.3, pp. 683–693.

Wang, Fang, Geoffrey Schoenbaum, and Thorsten Kahnt (2020). "Interactions between human orbitofrontal cortex and hippocampus support model-based inference". In: *PLoS Biology* 18.1, e3000578.

Wang, Ziyu et al. (2016). "Dueling network architectures for deep reinforcement learning". In: *International conference on machine learning*. PMLR, pp. 1995–2003.

Watkins, Christopher John Cornish Hellaby (1989). "Learning from delayed rewards". In:

White, A. (2015). "DEVELOPING A PREDICTIVE APPROACH TO KNOWLEDGE". In:

White, Lisa M (1996). *Temporal difference learning: eligibility traces and the successor representation for actions.* Citeseer.

Yamins, Daniel LK et al. (2014). "Performance-optimized hierarchical models predict neural responses in higher visual cortex". In: *Proceedings of the national academy of sciences* 111.23, pp. 8619–8624.

Yang, Yuzhe et al. (2019). "Harnessing structures for value-based planning and reinforcement learning". In: *arXiv preprint arXiv:1909.12255*.

Yao, Hengshuai, Rich Sutton, et al. (2009). "Dyna (k): A Multi-Step Dyna Planning". In: *Abstraction in Reinforcement Learning*, p. 54.

Yao, Hengshuai, Richard S Sutton, et al. (2009). "Multi-step dyna planning for policy evaluation and control". In: *NIPS*.

Young, Kenny and Tian Tian (2019). "Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments". In: *arXiv preprint arXiv:1903.03176*.

Yuste, Rafael (2015). "From the neuron doctrine to neural networks". In: *Nature reviews neuroscience* 16.8, pp. 487–497.

Zhang, Jingwei et al. (2017). "Deep reinforcement learning with successor features for navigation across similar environments". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 2371–2378.

Zhong, Yong-Mei, Masao Yukie, and Kathleen S Rockland (2006). "Distinctive morphology of hippocampal CA1 terminations in orbital and medial frontal cortex in macaque monkeys". In: *Experimental brain research* 169.4, pp. 549–553.