Multistep Credit Assignment in Deep Reinforcement Learning

by

Brett Daley

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

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

Temporal credit assignment in reinforcement learning (RL) is the problem of associating rewards with the past actions that contributed to them. Temporal-difference (TD) learning assigns credit by comparing the observed and expected outcomes after each action, generating a TD-error signal which drives the improvement of future predictions. This incremental, 1-step learning approach eventually grasps long-term consequences but requires many repetitions to converge. In contrast, *multistep* approaches can learn much faster by reapplying TD errors across multiple actions.

Eligibility traces were the main way to implement multistep credit assignment in RL, by broadcasting errors over a fading record of recent observations. However, over the past decade, RL algorithms have largely adopted neural networks to approximate value functions. These *deep RL* methods depend on randomized experience replay to stabilize learning, precluding the use of eligibility traces.

As a result, advanced multistep credit-assignment techniques have not been widely adopted in deep RL. The vast majority of existing replay methods opt for n-step returns, which are computationally cheap but nonsmooth. On the other hand, λ-returns offer a potential surrogate for eligibility traces, but they are expensive and not commonly used in large-buffer RL agents. In both cases, off-policy corrections are typically ignored, incurring bias in the return estimation.

In this thesis, I establish new properties of compound returns—especially λ-returns—and discuss how to leverage them efficiently in deep RL to accelerate learning. I prove that compound returns generally improve the bias-variance trade-off compared to n-step returns.

Additionally, I show that all compound returns obey the recency heuristic from psychology: the influence of a reinforcing event decreases monotonically in elapsed time. Long-tailed compound returns like λ-returns, however, are most effective for credit assignment in practice. I propose a cached-trajectory replay strategy for efficiently computing λ-returns while mitigating truncation bias and sample correlations. Finally, I generalize compound returns with trajectory-aware returns, which jointly consider experiences in the replay memory to produce more efficient off-policy corrections. Throughout this thesis, I evaluate the proposed algorithms in a variety of tabular environments, imaged-based games, and robot simulations.

Preface

The main contribution chapters of this thesis are based on four published papers, ordered logically rather than chronologically. Specifically, Chapters 3 and 6 are based on papers published in the proceedings of the International Conference on Machine Learning [2, 3]; Chapter 4 is based on a paper published in the Reinforcement Learning Journal [4]; and Chapter 5 is based on a paper published in Advances in Neural Information Processing Systems [1]. The remaining chapters (1, 2, and 7) are original to this thesis.

[1] Daley, B. and Amato, C. (2019). Reconciling λ-returns with experience replay. In *Neural Information Processing Systems (NeurIPS)*.

[2] Daley, B., White, M., Amato, C., and Machado, M. C. (2023). Trajectory-aware eligibility traces for off-policy reinforcement learning. In *International Conference on Machine Learning (ICML)*.

[3] Daley, B., White, M., and Machado, M. C. (2024b). Averaging n-step returns reduces variance in reinforcement learning. In *International Conference on Machine Learning (ICML)*.

[4] Daley, B., Machado, M. C., and White, M. (2024a). Demystifying the recency heuristic in temporal-difference learning. *Reinforcement Learning Journal*, 3:1019–1036.

Chapter 5 is additionally supplemented by a short, unpublished technical report [5].

[5] Daley, B. and Amato, C. (2021). Virtual replay cache. arXiv:2112.03421.

During the course of my research presented in this thesis, I wrote papers on topics related to deep RL including temporal-difference learning [10, 13, 14], experience replay [8], partial observability [6, 9], and multi-agent RL [7, 11]. I also co-authored a paper on ads experimentation as part of my internship at Meta [12]. Discussions of these have been omitted

in order to focus on the matters of multistep temporal credit assignment—except for the work of Elelimy et al. (2025), which I briefly describe as part of future work in Section 7.2.4.

[6] Nguyen, H., Daley, B., Song, X., Amato, C., and Platt, R. (2020). Belief-grounded networks for accelerated robot learning under partial observability. In *Conference on Robot Learning (CoRL)*.*

[7] Lyu, X., Xiao, Y., Daley, B., and Amato, C. (2021). Contrasting centralized and decentralized critics in multi-agent reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. **Best Paper Award Finalist**.

[8] Daley, B., Hickert, C., and Amato, C. (2021). Stratified experience replay: Correcting multiplicity bias in off-policy reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Extended abstract.

[9] Baisero, A., Daley, B., and Amato, C. (2022). Asymmetric DQN for partially observable reinforcement learning. In *Uncertainty in Artificial Intelligence (UAI)*.

[10] Daley, B. and Chan, I. (2022). Adaptive tree backup algorithms for temporal-difference reinforcement learning. In *Multi-Disciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*.*

[11] Lyu, X., Baisero, A., Xiao, Y., Daley, B., and Amato, C. (2023). On centralized critics in multi-agent reinforcement learning. *Journal of Artificial Intelligence Research*, 77:295–354.

[12] Bakhitov, E., Zhang, C., Sherstobitov, I., Daley, B., Ting, D., Nassif, H., and Leonenkov, S. (2024). Ad clustered user randomized trials. In *Conference on Digital Experimentation @ MIT (CODE@MIT)*. Extended abstract.

[13] Elelimy, E., Daley, B., Patterson, A., Machado, M. C., White, A., and White, M. (2025). Deep reinforcement learning with gradient eligibility traces. *Reinforcement Learning Journal*. **Outstanding Paper Award on Theory of Reinforcement Learning**.*

[14] Daley, B., Nagarajan, P., White, M., and Machado, M. C. (2025). An analysis of action-value temporal-difference methods that learn state values. *Reinforcement Learning Journal*.*

*The first two authors share primary authorship.

To my parents, Elizabeth and Joseph

This thesis is a product of your hard work as much as it is of my own.

Then there's the idea of dissatisfaction. By this, I don't mean a pessimistic dissatisfaction of the world—we don't like the way things are—I mean a constructive dissatisfaction. The idea could be expressed in the words, "This is OK, but I think things could be done better. I think there is a neater way to do this. I think things could be improved a little."

– Claude Shannon

Acknowledgements

PhD students are lucky enough to have one great advisor, but I had the unusual fortune of three. My incredible advisors here at the U of A, Marlos C. Machado[1] and Martha White, always held me to the highest standards of the scientific method. I learned so much from their mentorship, life advice, and distinctive ideologies. My research was elevated as a result.

I am also indebted to Chris Amato at Northeastern University for initiating my journey into RL research. Our many years of collaboration led to several contributions in this thesis. Above all, Chris taught me how to write research papers and supported my move to Alberta.

Special thanks to my defense committee: Rich Sutton, Rupam Mahmood, and Phil Thomas. It was great engaging with their thoughtfully sharp questions. I am also grateful to Rich for serving on my supervisory committee; what an honor it was to work with an RL legend whose research has so profoundly influenced and inspired my own.

I owe it to many others for making it this far: David Kaeli (Northeastern University), Elliot Mednick (AMD), Curtis Bradford (Flex), and Oliver Staton (Tesla). Without their mentorship and letters of recommendation, graduate school would not have been a possibility. My PhD experience was enriched by internships at Sony AI with Harm van Seijen, Ishan Durugkar, and Raksha Kumaraswamy, and at Meta with Sergei Leonenkov and Houssam Nassif. Finally, I thank my many co-authors, whose names are included in the Preface.

This thesis is dedicated to my parents, Elizabeth and Joseph, for their unconditional support of my education. I am thankful for encouragement from Grandmom and my siblings too: Heather, Mark, and Rachel. Congratulations to Mark as well for successfully defending his PhD thesis at Brown University, one month before I did. He did have a head start, though.

Last but not least: Thank you, Cece. Not only did you teach me how to navigate life in Canada, but I thrived here with your unwavering support. I love you. ☺

[1] Don't forget the C!

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Artificial intelligence (AI) is intelligence exhibited by a machine. Although no definition of intelligence is universally agreed upon, McCarthy (1997) provides a succinct and satisfactory one: "Intelligence is the computational part of the ability to achieve goals in the world." Notably, a goal need not be an inherent property of an intelligent system, but can instead be a useful abstraction to describe a system's behavior from the perspective of an observer (Dennett, 1989; Sutton, 2020). The pursuit of AI can thus be roughly described as the development of algorithms that achieve arbitrary goals.

An important implication of this definition is that AI must achieve many goals with inevitably limited computational resources. The world is vast, complex, and changing; AI must therefore be computationally efficient to enable generality (Sutton et al., 2022). From this perspective, intelligence is the ability to continuously acquire goal-achieving behaviors under unavoidable constraints on calculations, time, memory, and data. Self-improvement has been considered a core component of AI since its inception (Turing, 1950; McCarthy et al., 1955), but I emphasize in this thesis that *economical* self-improvement is key. The underlying objective of my work is to understand how agents can *efficiently* acquire, or learn, new behaviors toward achieving goals.

1.1 Reinforcement Learning

Reinforcement learning (RL) is a general framework for describing agents that learn to achieve goals through trial-and-error interaction with their environments (Sutton and Barto, 2018, Ch. 1). Goals are represented by a reward signal. Each action taken by an agent

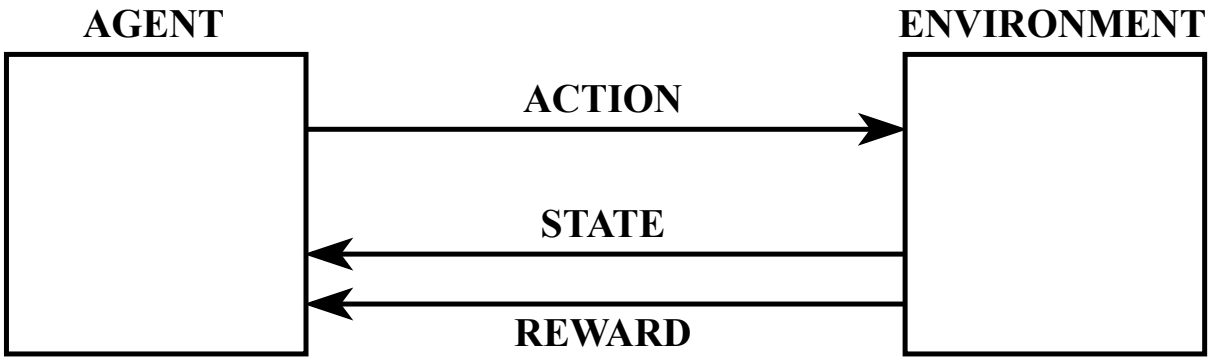
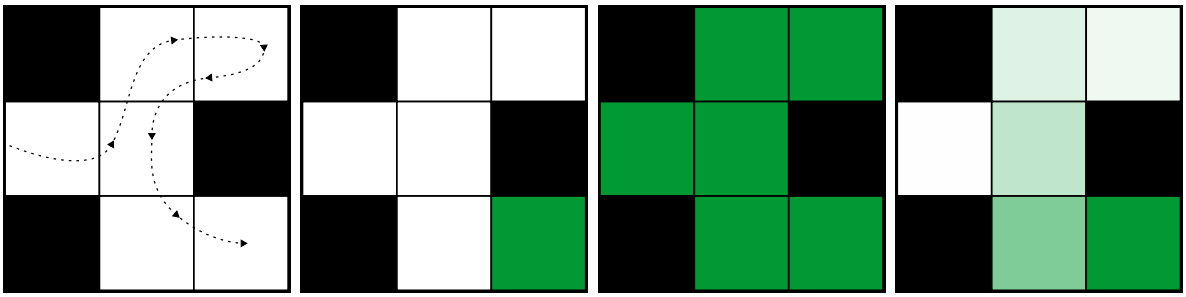


Figure 1.1: The standard RL model. An agent takes actions in an unknown environment, influencing the state and receiving rewards. The agent must learn to take actions which maximize its cumulative reward.

provides an immediate reward—which can be good or bad—and modifies the state of the environment (see Figure 1.1). The agent seeks to maximize its return (cumulative reward) in expectation.

The temporal credit-assignment problem in RL is the challenge of associating past actions with the rewards they are responsible for (Minsky, 1961; Sutton, 1984). Rewards are often delayed and partially random. When a good reward is eventually obtained, it is not evidently clear which actions contributed to it; it could be due to a specific combination of actions taken, or could be entirely unrelated to the agent's conduct. Furthermore, because actions change the state of the environment, acting greedily to obtain a good reward now may preclude opportunities to obtain better rewards later. The return of an action is therefore determined not only by the immediate reward it earns but also by its influence on the acquisition of future rewards. Addressing the temporal credit-assignment problem effectively is paramount to fast learning because the causal relationship between actions and future rewards must be well understood to refine reward-seeking behaviors.

The most widely used class of RL algorithms for temporal credit assignment is temporal-difference (TD) learning (Sutton, 1988). Basic TD methods operate using 1-step assignment; they learn the ultimate values of actions by reducing an error between the return estimates immediately before and after each obtained reward. This *TD error* is the fundamental reinforcement signal for credit assignment in TD methods. Return estimates are updated by applying the TD errors to them, nudging them in the direction of improved accuracy on av-



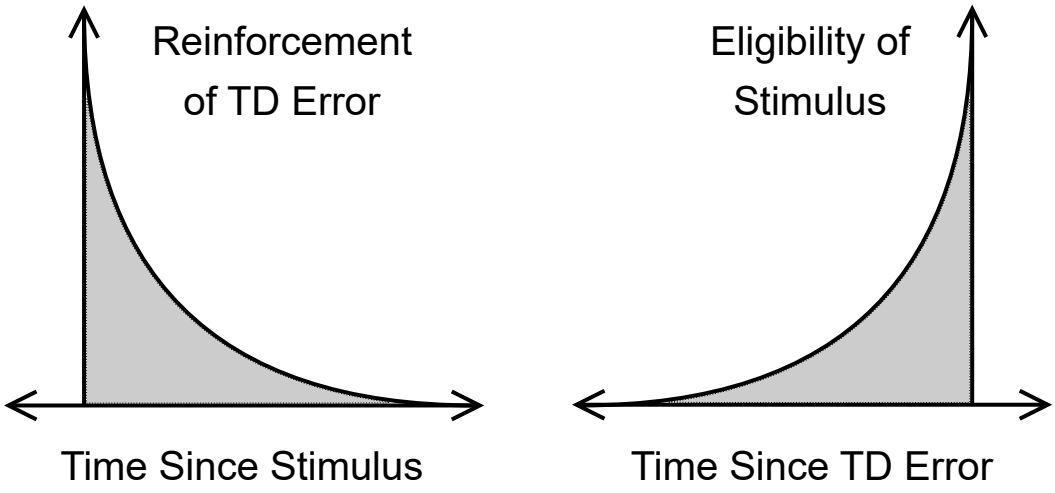
(a) Agent Path (b) 1-step TD (c) Monte Carlo (d) Eligibility Traces

Figure 1.2: Visual representation of different credit-assignment methods for an agent navigating a maze. A reward is given for reaching the bottom-right cell. Green hues represent the degree of credit assignment. (a) The agent takes a meandering, exploratory path to the reward. (b) 1-step TD assigns credit only to the final cell, requiring more repetitions to reinforce the successful behavior. (c) Monte Carlo methods assign credit to all visited cells, which is relatively uninformative and fails to identify the correct path. (d) Replacing eligibility traces (Singh and Sutton, 1996) with $\lambda = 0.5$ smoothly assign credit to the correct path while mostly ignoring less relevant cells.

erage. Since the TD-error signal itself depends on the estimates that it updates, TD methods are *bootstrapping* methods; they learn by reusing previously learned estimates.

Multistep TD methods operate on a similar principle, but they generate compound errors that span more than one time step. These compound errors can always be decomposed into a weighted combination of the fundamental 1-step TD errors described above. Furthermore, the particular way in which these TD errors are combined determines when and to what degree the learning update depends on obtained rewards directly or instead bootstraps from previous estimates. The vast space of possible bootstrapping choices differentiates how TD methods address the temporal credit-assignment problem (see the example in Figure 1.2). Mathematically, the bias of old estimates and the variance of noisy rewards pose a trade-off which must be balanced to learn efficiently.

The decomposition of compound errors into 1-step TD errors offers two nearly equivalent views of TD methods: the *forward* view and the *backward* view (see Figure 1.3). The forward view waits to collect the necessary TD errors for constructing the compound error before applying the net result. This introduces a delay between experience and reinforcement. In contrast, the backward view applies the (reweighted) TD error on each time step to past actions, gradually generating their total compound updates over time. The backward view is strongly associated with TD(λ) and eligibility traces (Sutton, 1988; see Section 1.2)



(a) Forward View (b) Backward View

Figure 1.3: The forward and backward views of TD learning. The forward view (a) concerns the reinforcement of TD errors from the reference frame of an individual stimulus. The backward view (b) concerns the eligibility of past stimuli from the reference frame of an individual TD error (negative time means stimuli precede the TD error). The total reinforcement applied to each stimulus is the same in both views, assuming value estimates are static.

because they admit an efficient update rule; however, in theory, any compound error has a backward view, even if it is not implementable by a practical algorithm (see Lemma 3.5). The two views are approximately equivalent, but differ greatly in terms of their algorithmic implementations and related computational considerations.

For a long time, the forward view was considered a purely theoretical tool for analyzing the backward view. Eligibility traces were so effective for multistep credit assignment in classic RL methods that there was no benefit to introducing a delay for forward-view updating. However, over the last 10 years, deep RL with experience replay has become ubiquitous and has gradually revealed the practical utility of the forward view. Deep RL uses neural networks to approximate return estimates. Neural networks are powerful function approximators formed by cascading multiple differentiable computational layers (see, e.g., LeCun et al., 2015; Goodfellow et al., 2016, Ch. 6) that can learn generalizable, nonlinear representations via end-to-end backpropagation (Rumelhart et al., 1986). Deep RL has achieved impressive feats in many domains including arcade games (Mnih et al., 2013, 2015), board games (Silver et al., 2016, 2017), real-time strategy games (OpenAI, 2018;

Vinyals et al., 2019), robotic control (Schulman et al., 2015a; Levine et al., 2016), 3D maze navigation (Mirowski et al., 2017), and training large language models (e.g., Ouyang et al., 2022; DeepSeek-AI, 2025). Experience replay is used to stabilize neural network training (see Section 1.3), interfering with the backward view of TD learning. In particular, the random order of reinforcement due to minibatching renders eligibility traces inapplicable, and forward-view updates become useful for implementing multistep credit assignment.

1.2 Credit Assignment with Eligibility Traces

Eligibility traces originate from Klopf's heterostatic theory of neurons in the brain (Klopf, 1972, 1982). At its core, the theory postulated that neurons are *hedonists*; they seek to maximize an analog of pleasure and to minimize an analog of pain. (These were hypothesized to be depolarization and hyperpolarization, respectively, of the membrane potential.) One of the most influential pieces of the heterostatic theory to RL was that activated synapses become temporarily *eligible* to undergo changes in efficacy. Actual changes to efficacy would then be contingent on the timely arrival of a reinforcing signal. These lingering eligibilities offered a plausible mechanism for which the Law of Effect[2] (Thorndike, 1898, 1911) could emerge from local neural activity.

Eligibility traces in the familiar, modern sense were developed in actor-critic architectures (Sutton and Barto, 1981; Barto et al., 1983; Sutton, 1984), being deeply inspired by Klopf's ideas. For practicality's sake, two major simplifications were made. First, the notion of a local pleasure-pain analog was replaced by a global reward signal to be optimized. This reward signal is not directly responsible for reinforcement but instead generates the reinforcing TD-error signal—in conjunction with bootstrapping—exactly as described in Section 1.1. Second, the eligibility traces themselves follow an idealized spike-and-decay model. Upon firing, synapses instantaneously increase their eligibilities, which then decrease exponentially afterwards. In actor-critic methods, eligibility traces implement credit assignment for the critic, which evaluates states to improve the actor's control policy. The TD(λ) algorithm (Sutton, 1988) later demonstrated that eligibility traces are a general technique for solving sequential prediction problems, independent of any particular actor. Eligibility traces be-

[2]The Law of Effect in psychology is the observation that the frequency of a behavior in animals is modulated by the desirability of the experienced consequences.

came the standard for multistep credit assignment in fundamental RL algorithms, including Q-learning (Watkins, 1989) and Sarsa (Rummery and Niranjan, 1994).

Although many algorithmic variations have been developed over the years, the basic principles governing eligibility traces have remained similar since TD(λ). At each time step, the agent receives a feature vector representing an input stimulus (typically corresponding to the current state or state-action pair). A separate vector, which stores the eligibility traces, is decayed by a constant factor, $\lambda \in [0, 1]$, and then incremented element wise by the synaptic activations induced by the stimulus. Finally, the current TD error is applied to all synapses in proportion to their eligibilities. This simultaneously reinforces all past stimuli with a strength according to the elapsed time since they were experienced. This type of gradually fading credit assignment is especially beneficial when rewards are sparse or delayed. Moreover, each input feature needs to be processed only once, making the computational and memory costs highly favorable.

Eligibility traces have been enormously successful for implementing multistep credit assignment when value estimates are represented by either a lookup table or a weighted combination of the input feature—both instances of linear function approximation. In these cases, strong learning guarantees have been established (e.g. Sutton, 1988; Watkins and Dayan, 1992; Jaakkola et al., 1993; Tsitsiklis, 1994; Tsitsiklis and Van Roy, 1997). Although eligibility traces are compatible with backpropagation (Sutton, 1989) and have achieved major successes such as learning to play Backgammon (Tesauro, 1991), they can be unstable when training neural networks. This instability arises from general training difficulties of neural networks such as catastrophic forgetting (McCloskey and Cohen, 1989) and exploding and vanishing gradients (Bengio et al., 1994), which manifest as non-contractive TD updates that sometimes cause divergence (Boyan and Moore, 1994; Tsitsiklis and Van Roy, 1997). Although these challenges are likely surmountable with new ideas, the field has largely turned to minibatched experience replay to circumvent them (see Section 1.3). Eligibility traces are not compatible with this particular mode of training because they must process experiences sequentially, and have unfortunately declined in popularity as a result.

A few recent works have attempted to revive the classic, incremental training style of eligibility traces for neural networks (see Section 7.2.4). However, experience replay remains the most widely used and successful training technique for deep RL. As of this writing, there is

no RL method that learns from a single stream of experience, without replay, that can match the performance of the state-of-the-art deep RL agents discussed in Section 1.3. As such, finding alternative ways to achieve multistep credit assignment in deep RL remains crucial.

1.3 Credit Assignment with Experience Replay

Experience replay is commonly attributed to Lin (1991, 1992), though elements of experience replay can be found in earlier RL architectures for planning (e.g., Dyna-Q; Sutton, 1990). Lin's primary motivation was data reuse in robotics, where environment interactions are expensive. The sample efficiency of RL algorithms can be greatly increased by saving and repeatedly processing past training samples. Another crucial role of experience replay is to mitigate catastrophic forgetting in neural networks by repeating past training examples, as demonstrated by Neural Fitted Q Iteration (NFQ; Riedmiller, 2005). NFQ was an early network-based RL method which combined batch training with a target network to stabilize Q-learning, laying the groundwork for later deep RL methods.

Experience replay gained mainstream popularity after the breakthrough success of Deep Q-Network (DQN; Mnih et al., 2013, 2015): a convolutional neural network (LeCun et al., 1998) trained by Q-learning to play Atari 2600 arcade games. DQN was a landmark achievement, proving that neural networks could learn human-level control policies just from raw pixel images and trial-and-error reward maximization. Central to DQN's success was its influential style of experience replay using minibatches—already a common practice in deep learning research of the time (e.g., Krizhevsky et al., 2012), but not in RL. Unlike NFQ, DQN interleaved environment interaction and training, buffering new experiences in a finite-capacity replay memory while occasionally sampling and training on minibatches of old experiences. This ensured a continual influx of fresh, exploratory experiences that would overwrite outdated, counterproductive ones. Experience replay in this manner has become a staple of modern RL algorithms including DDPG (Lillicrap et al., 2016), TD3 (Fujimoto et al., 2018), SAC (Haarnoja et al., 2018), and many others since.

Minibatched experience replay is crucial for decorrelating updates and reducing variance while keeping computational costs relatively low. Unfortunately, as discussed in Section 1.2, eligibility traces are no longer possible in this setting because experiences are processed non-

sequentially. Moreover, experience replay with multistep returns produces *off-policy* bias, owing to the fact that the replay buffer's data distribution does not match that under the agent's behavior policy. Correcting this bias with importance sampling (Kahn and Marshall, 1953) greatly exacerbates variance, which can be more detrimental than the bias itself (Precup et al., 2000). As a result, many deep RL agents—including all of the examples mentioned above—rely on simple 1-step credit assignment to avoid the complexities of multistep learning and off-policy corrections. However, this leads to slow learning in many cases.

A potential surrogate for smooth multistep credit assignment in deep RL, where eligibility traces are not possible, is λ-returns. The λ-return is the hypothetical target of TD(λ) and Q(λ) updates, equivalent to an exponentially weighted average of n-step returns (Watkins, 1989), which I discuss below. Direct computation of λ-returns (without eligibility traces) is possible using the states, actions, and rewards already stored in the replay memory. Cichosz (1994) first proposed using (truncated) λ-returns as a forward-view target for TD learning, mainly as a more efficient alternative to eligibility traces in tabular or memory-based representations (which are no longer common). Much later, van Seijen (2016) showed that truncated λ-returns could address some of the training difficulties associated with neural networks and eligibility traces. However, λ-returns are expensive to compute in minibatches because each return estimate depends on many value estimates, each of which requires an expensive forward pass through the neural network. In deep RL, several works have sought to reduce this cost by modifying the replay buffer to return minibatches of short trajectories (e.g., Munos et al., 2016; Harb and Precup, 2016; Wang et al., 2017; Kozuno et al., 2021; Tang et al., 2024).[3] This enables more efficient calculation by sharing value estimates. While trajectory replay with λ-returns does achieve eligibility-trace-like credit assignment suitable for deep RL, the downsides are excessive return truncation (which inhibits credit assignment) and harmful sample correlations within the minibatches.

A simpler and computationally cheaper alternative to λ-returns is n-step returns. The n-step return is the most basic multistep estimator: it sums the immediate n rewards and then bootstraps from the following value estimate. Compared to λ-returns, n-step returns are much less expensive, as each one needs only a single forward pass through the neural network.

[3]Often, λ-returns are conflated with eligibility traces in this setting (e.g., Harb and Precup, 2016), highlighting the general misunderstanding surrounding these concepts. True eligibility traces utilize only 1-step TD errors and have no memory beyond the trace vector itself, which is one of their most appealing features.

Again, Cichosz (1994) first proposed using n-step returns as a forward-view target—the special case of truncated λ-returns when $\lambda = 1$. Much later, in deep RL, Rainbow (Hessel et al., 2018) popularized n-step returns for multistep credit assignment in large-buffer replay methods.[4] Since then, n-step returns have become prevalent in deep RL (e.g., Horgan et al., 2018; Kapturowski et al., 2018; Schrittwieser et al., 2020; Wurman et al., 2022; Tang et al., 2022; Chebotar et al., 2023; Schwarzer et al., 2023).

Although n-step returns are easier to combine with minibatched experience replay, they are a relatively primitive form of credit assignment. More specifically, they are nonsmooth and do not integrate information across multiple value estimates. They also lack corrections for the off-policy bias in experience replay discussed previously. Although there has been extensive research into more sophisticated forms of multistep credit assignment in off-policy RL (e.g., Sutton et al., 2014; Harutyunyan et al., 2016; Munos et al., 2016; Mahmood et al., 2017; Rowland et al., 2020), they are comparatively difficult to implement in deep RL, and basic n-step returns remain commonplace. A unifying goal of my contributions in this thesis is to further develop multistep credit-assignment techniques and efficiently deploy them in large-buffer deep RL. The emphasis on the large-buffer regime is key, as on-policy actor-critic methods such as TRPO (Schulman et al., 2015a), A3C (Mnih et al., 2016), and PPO (Schulman et al., 2017) learn from short trajectories that make λ-returns feasible (Schulman et al., 2015b). However, in off-policy replay algorithms like DQN and SAC, which have replay buffers typically on the order of millions of samples, this is not the case and n-step returns are more commonly used instead.

1.4 Thesis Statement

The principal claim of this work is the following:

> **Compound returns, especially λ-returns, can be efficiently implemented in deep RL with experience replay to improve the bias-variance trade-off and increase sample efficiency in complex environments.**

Compound returns are multistep returns formed by weighted averages of two or more

[4] A3C (Mnih et al., 2016) was probably the first deep RL algorithm to apply n-step returns, but it did not use experience replay and instead relied on parallel simulated environments to randomize minibatches.

n-step returns. I coined the term compound return (Daley et al., 2024b) to refer to the target of a compound update (Sutton and Barto, 2018, Sec. 12.1): a TD update made toward a weighted average of n-step returns.[5] The λ-return is a special case of compound return with exponential weights. All compound returns are equally valid targets for TD learning in the sense that they reduce prediction error on average (Watkins, 1989, Sec. 7.2). Other examples of compound returns include γ-returns (Konidaris et al., 2011) and Ω-returns (Thomas et al., 2015), but they are less common than λ-returns, which remain popular for their efficient recursive formula. By definition, a compound return bootstraps from multiple value estimates, unlike an n-step return. Because even small networks by today's standards can have millions of parameters, the computational cost from bootstrapping becomes nontrivial. For this reason, compound returns including λ-returns have been challenging to deploy in large-buffer deep RL, and n-step returns are most widely used.

The *bias-variance trade-off* in RL is the problem of simultaneously minimizing the bias from bootstrapping and the variance from sampled rewards. This trade-off is exacerbated in off-policy learning, where the distribution mismatch contributes additional bias, but correction techniques dramatically increase variance. The bias-variance trade-off was initially hypothesized to exist for the choice of λ in TD(λ) (Sutton, 1988; Watkins, 1989, Sec. 7.2.1) and was later established formally for both n-step returns and λ-returns (Kearns and Singh, 2000). Both λ and n interpolate between 1-step TD methods ($n = 1$ or $\lambda = 0$) and Monte Carlo methods ($n \to \infty$ or $\lambda = 1$), but direct theoretical comparisons between them were not made for a long time. It was believed that n-step returns and λ-returns differed primarily by their algorithmic implementations and were otherwise equivalent means of controlling the bias-variance trade-off. In Chapter 3, I show how λ-returns, and compound returns more generally, do exhibit different (and often favorable) bias-variance properties compared to n-step returns.

Sample efficiency is a measure of learning efficiency based on the number of environment interactions an agent needs to reach a target performance level. For example, an agent that achieves a higher cumulative reward than another after the same number of environment interactions can be said to be more sample-efficient—a comparison commonly made in deep RL experiments. Other possible definitions of efficiency are formulated in terms

[5] Compound updates were called complex backups in older literature (Sutton and Barto, 1998, Sec. 7.2).

of wall-clock time or the total number of calculations. However, these can be highly dependent on implementation details such as hardware architecture, parallelization, and code optimization, making them potentially less informative as measures of learning capability. For these reasons, I mainly focus on sample efficiency as a key indicator of agent performance in this thesis, although I do analyze the real-time efficiency of different replay strategies in Chapter 5 as well.

By *complex* environments, I refer to environments with many possible states and in which value estimates cannot simply be expressed as linear combinations of the state features. Consequently, lookup tables or linear function approximation become insufficient for learning value functions. This is especially the case, for example, when states are represented as high-dimensional images (e.g., Atari 2600 games; Bellemare et al., 2013). Neural networks are able to extract useful, nonlinear features in these environments, but they become expensive with compound returns, which bootstrap from the value function more than n-step returns.

I support my thesis statement with a combination of theoretical analysis, algorithmic development, and computational experiments. I primarily focus on *value-based* deep RL agents such as DQN: those that determine their control policies from value functions learned by (off-policy) TD methods and experience replay. This allows me to more easily disentangle the effects of multistep credit assignment from those of other important issues in deep RL such as policy learning and exploration. I typically start by evaluating my algorithms in small tabular environments to demonstrate their feasibility or to illuminate certain properties. Often, the insights gained are agnostic to the particular implementation of the value function, allowing me to transfer them to the deep RL setting described above. I then use image-based games such as MinAtar (Young and Tian, 2019) and Atari 2600 (Bellemare et al., 2013), and sometimes MuJoCo robotic simulations (Todorov et al., 2012), to demonstrate the applicability and scalability of these ideas with neural networks. Notably, the algorithms I propose in this thesis are not restricted to neural networks; they equally apply to all differentiable function approximators, and are especially relevant for nonlinear and computationally expensive ones. However, I have chosen to focus the discussion on neural networks because of their modern practical significance.

1.5 Contributions by Chapter

The topics in this thesis can be roughly bifurcated into properties of multistep return estimators (Chapters 3 and 4) and algorithms for learning from trajectories of past experiences (Chapters 5 and 6). Below, I provide an overview of the contributions within each chapter.

Chapter 2: Background I provide relevant context for the topics discussed in this thesis and introduce mathematical notation accordingly.

Chapter 3: Variance Reduction via Averaging n-step Returns I introduce reasonable assumptions for modeling return variance, building upon the work of Konidaris et al. (2011). Under these assumptions, I prove that compound returns have a variance-reduction property relative to bias-equalized n-step returns. I show that this property improves sample efficiency: theoretically with linear function approximation and empirically with nonlinear function approximation.

Chapter 4: Demystifying the Recency Heuristic I mathematically formalize the notion of the recency heuristic in TD learning. I prove this heuristic is satisfied only by compound and n-step returns; other returns that violate the heuristic cause divergence in some tabular environments. Finally, I conduct experiments to understand why λ-returns are particularly effective for credit assignment, finding that the length of the return's eligibility window is more important than its smoothness.

Chapter 5: λ-Returns with Experience Replay I summarize existing approaches for implementing λ-returns in experience replay and discuss their deficiencies in terms of computational cost, return truncation, and sample correlations. I propose a new algorithm called cached-trajectory replay which ameliorates these three issues. I develop DQN(λ) based on cached-trajectory replay and show it is competitive with n-step DQN in Atari 2600 games.

Chapter 6: Trajectory-Aware Off-Policy Corrections I introduce trajectory-aware returns for off-policy learning, which generalize the per-decision estimators studied by Munos et al. (2016). I prove conditions for the convergence of the corresponding trajectory-aware

operator for prediction and control, settling an open problem in the latter case. I propose a new trajectory-aware off-policy algorithm called Recency-Bounded Importance Sampling (RBIS) and demonstrate its superiority over existing methods in several gridworld topologies.

Chapter 7: Discussions and Future Work I summarize my contributions, discuss potential avenues for future work, and survey recent attempts to combine deep RL with backward-view eligibility traces. I conclude with some remarks about AI research.

Chapter 2

Background

An RL agent interacting with its environment is commonly formalized as a Markov decision process (MDP). The MDP is specified by a tuple: $(\mathcal{S}, \mathcal{A}, p, \mathcal{R}, d_0, \gamma)$. The discrete[6] sets $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$ contain the possible environment states, agent actions, and scalar rewards, respectively. The function p represents the environment's dynamics in terms of state transitions and rewards. Taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ transitions the environment to state $s' \in \mathcal{S}$ and yields reward $r \in \mathcal{R}$ with probability $p(s', r \mid s, a)$. The transition dynamics satisfy the Markov property; they are conditionally independent of the process history, given the current state and action. The environment's initial state S_0 is sampled from $d_0(\cdot)$, where $d_0 \colon \mathcal{S} \to [0, 1]$ is the start-state distribution. The discount factor $\gamma \in [0, 1]$ determines the agent's relative preference for delayed rewards.

The agent executes a policy π, which maps states to distributions over actions. At each time step $t \geq 0$, the agent executes an action $A_t \sim \pi(\cdot \mid S_t)$, transitioning the environment to a new state $S_{t+1} \in \mathcal{S}$ and yielding a reward $R_{t+1} \in \mathcal{R}$. In continuing environments, this process repeats indefinitely and the agent experiences a stream of environment interactions: $(S_0, A_0, R_1), (S_1, A_1, R_2), \ldots$ and so on. In episodic environments, a terminal state may be encountered, in which case the next state is resampled from $d_0(\cdot)$. For ease of presentation, the equations in this thesis typically ignore episode terminations, but these details are discussed as needed for algorithms and experiments.

[6] I assume these sets are discrete for ease of exposition. In many cases, the results in this thesis extend to uncountable sets by replacing sums with integrals.

The discounted return earned by the agent at time t is defined as

$$G_t \coloneqq \sum_{i=0}^{\infty} \gamma^i R_{t+1+i} \,.$$

In continuing environments, a restriction of $\gamma < 1$ is needed to ensure finite returns. The discounted return is also called the Monte Carlo return because of its use in Monte Carlo estimation, but it has many other applications in RL.

RL agents are typically classified as one of two types: prediction or control. *Prediction* agents seek to estimate the expected discounted return earned by π in each state.[7] Because the policy is assumed to be fixed in this case, the MDP becomes a Markov chain with rewarded transitions: i.e., a Markov reward process (MRP). On the other hand, *control* agents seek to improve π until it maximizes the expected discounted return earned in each state—at which point an optimal policy has been learned. Prediction and control are closely related because accurately estimating the returns is often needed for policy improvement.

2.1 Value Functions and Bellman Operators

Most RL agents learn value functions: mappings from states or state-action pairs to their conditionally expected discounted returns. The state-value function v_π is defined as

$$v_\pi(s) \coloneqq \mathbb{E}_\pi[G_t \mid S_t = s] \,,$$

where the expectation, $\mathbb{E}_\pi$, implies actions are sampled according to π. Analogously, the action-value function q_π is defined as

$$q_\pi(s, a) \coloneqq \mathbb{E}_\pi[G_t \mid (S_t, A_t) = (s, a)] \,.$$

Estimating q_π is significant because the greedy[8] policy with respect to q_π is guaranteed to either perform better than π or be optimal. If the policy is an optimal policy (there may be multiple for a given MDP), then it is denoted by π_* and its corresponding action-value function is denoted by q_*. Every optimal policy is greedy with respect to the same q_*.

[7] More generally, prediction agents can estimate any signal, which is the idea underlying General Value Functions (GVFs; Sutton et al., 2011). However, I focus exclusively on the reward signal in this thesis.

[8] Greedy with respect to Q means executing action $\arg\max_{a \in \mathcal{A}} Q(s, a)$, where ties may be broken arbitrarily.

Value functions satisfy recursive equations known as Bellman equations (Bellman, 1957). The function v_π is the unique solution to the Bellman expectation equation for state values:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \Big(r + \gamma \, v_\pi(s') \Big) \, .$$

The function q_π is the unique solution to the Bellman expectation equation for action values:

$$q_\pi(s, a) = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \left(r + \gamma \sum_{a' \in \mathcal{A}} \pi(a' \mid s') \, q_\pi(s', a') \right) \, .$$

For mathematical convenience, value functions can be represented as vectors: i.e., $\boldsymbol{v}_\pi \in \mathbb{R}^{|\mathcal{S}|}$ and $\boldsymbol{q}_\pi \in \mathbb{R}^{|\mathcal{S} \times \mathcal{A}|}$. Each element of a vector corresponds to the value estimate for a particular state or state-action pair. (The order of the elements does not matter as long as it is consistent.) This allows the above Bellman equations to be simplified dramatically. First, I define another function which represents the expected reward for each state-action pair:

$$r(s, a) \coloneqq \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a) \, .$$

Again, this can be represented as a vector, $\boldsymbol{r} \in \mathbb{R}^{|\mathcal{S} \times \mathcal{A}|}$.

Next, I introduce several linear operators, which can be represented as matrices that transform the above vectors. Let $E_\pi \colon \mathbb{R}^{|\mathcal{S} \times \mathcal{A}|} \to \mathbb{R}^{|\mathcal{S}|}$ be defined such that

$$(E_\pi q)(s) \coloneqq \sum_{a \in \mathcal{A}} \pi(a \mid s) \, q(s, a) \, .$$

Additionally, define $P \colon \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S} \times \mathcal{A}|}$ such that

$$(Pv)(s, a) \coloneqq \sum_{s' \in \mathcal{S}} v(s') \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \, .$$

These basic operators allow me to concisely express relationships between different value functions. For example, the definitions of the state- and action-value functions imply that $\boldsymbol{v}_\pi = \boldsymbol{E}_\pi \boldsymbol{q}_\pi$ and $\boldsymbol{q}_\pi = \boldsymbol{r} + \gamma \boldsymbol{P} \boldsymbol{v}_\pi$.

Note that here and throughout my thesis, I bold value functions and linear operators when they appear as algebraic quantities. This indicates that they can be represented as vectors or matrices, respectively. In all other contexts, I do not apply boldface, even for the same objects (e.g., when invoking a value function with an argument or defining an operator element-wise).

I now define the Bellman operator, a value-function operator that represents one sweep of dynamic-programming policy evaluation. I overload T_π to denote both the state-value and action-value Bellman operators, defined respectively as

$$T_\pi \boldsymbol{v} \coloneqq \boldsymbol{E}_\pi(\boldsymbol{r} + \gamma \boldsymbol{P}\boldsymbol{v})\,,$$
$$T_\pi \boldsymbol{q} \coloneqq \boldsymbol{r} + \gamma \boldsymbol{P}\boldsymbol{E}_\pi \boldsymbol{q}\,.$$

Note that T_π is not bolded, because it is affine rather than linear. With these definitions, the Bellman expectation equations simplify to

$$\boldsymbol{v}_\pi = T_\pi \boldsymbol{v}_\pi\,,$$
$$\boldsymbol{q}_\pi = T_\pi \boldsymbol{q}_\pi\,.$$

These equations make it clear that $\boldsymbol{v}_\pi$ and $\boldsymbol{q}_\pi$ are fixed points of their respective Bellman operators. In fact, when $\gamma < 1$, T_π is a contraction mapping (Bertsekas, 2007, Sec. 1.4) and these fixed points are unique. Let exponentiation denote operator iteration such that $T_\pi^k \boldsymbol{v} \coloneqq T_\pi(T_\pi^{k-1}\boldsymbol{v})$ and $T_\pi^0 \boldsymbol{v} \coloneqq \boldsymbol{v}$. It follows that $\lim_{k\to\infty} T_\pi^k \boldsymbol{v} = \boldsymbol{v}_\pi$ or $\lim_{k\to\infty} T_\pi^k \boldsymbol{q} = \boldsymbol{q}_\pi$, for all $\boldsymbol{v} \in \mathbb{R}^{|\mathcal{S}|}$ or $\boldsymbol{q} \in \mathbb{R}^{|\mathcal{S}\times\mathcal{A}|}$, by the Banach fixed-point theorem (Banach, 1922).

Note that iterating these operators until convergence—i.e., dynamic programming—is not possible in the RL setting, since it requires knowledge of the MDP dynamics, which is assumed to be unavailable. Dynamic programming would be computationally intractable in high-dimensional environments of interest anyway. However, operators remain useful tools for understanding the convergence of RL algorithms. Prediction methods in RL approximate v_π or q_π from sampled data, assuming the policy π is fixed. Value-based control methods either track q_π as π improves (e.g., Sarsa) or attempt to directly estimate q_* (e.g., Q-learning).

2.2 Temporal-Difference Learning

TD learning (Sutton, 1988) is a class of RL algorithms that update each estimated state value $V(S_t)$ proportionally to the error with respect to a new estimate $\hat{G}_t$. The general forward-view TD update is

$$V(S_t) \leftarrow V(S_t) + \alpha_t\Big(\hat{G}_t - V(S_t)\Big)\,, \tag{2.1}$$

where $\alpha_t \in (0, 1]$ is the step size. Although α_t formally depends on time, it is commonly set to a fixed value α in practice. Eq. (2.1) is called a backup (Samuel, 1959). The arrow ($\leftarrow$) denotes reassignment of $V(S_t)$ to its new value. There are many possibilities for the target $\hat{G}_t$, the choice of which alters the learning properties of the TD algorithm (see Section 2.3).

More formally, TD learning is connected to the operator theory from Section 2.1 by considering $\hat{G}_t$ to be a sample from the noisy application of some value-function operator, $H \colon \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$. That is, for some zero-mean noise process ω_t, the return estimate is equal to

$$\hat{G}_t = (HV)(S_t) + \omega_t \,.$$

In other words, the operator H provides the expected value of the TD backup in Eq. (2.1). Thus, by analyzing the convergence properties of H in a dynamic-programming sense, similar convergence guarantees can typically be established for the TD method it represents.

For example, the fundamental 1-step TD update, also known as TD(0), constructs $\hat{G}_t$ from the immediate reward and the next state's estimated value: $\hat{G}_t = R_{t+1} + \gamma\, V(S_{t+1})$. In this case, the corresponding operator is the Bellman operator, T_π. With this choice, the error in Eq. (2.1) becomes

$$\delta_t \coloneqq R_{t+1} + \gamma\, V(S_{t+1}) - V(S_t) \,,$$

which is known as the TD error and is fundamental to value-based RL methods. When α_t is appropriately annealed (Robbins and Monro, 1951) and the policy π sufficiently explores the state space, TD(0) converges to v_π (Jaakkola et al., 1993; Tsitsiklis, 1994), because v_π is the unique fixed point of T_π.

So far, I have assumed that the value function $V(s)$ is a lookup table. In practice, environments will have many possible states, making approximation and generalization necessary. The most well-studied form of function approximation is linear. Here, each state $s \in \mathcal{S}$ is associated with a feature vector $\boldsymbol{x} \in \mathbb{R}^{N_{\boldsymbol{w}}}$, and the value function is parameterized by a weight vector $\boldsymbol{w} \in \mathbb{R}^{N_{\boldsymbol{w}}}$. The value estimate for any state $s \in \mathcal{S}$ is obtained by evaluating the dot product of its feature vector and the weight vector. Let $\boldsymbol{X}$ be the matrix whose rows are the feature vectors. The estimated value function is obtained by the multiplication $\boldsymbol{v} = \boldsymbol{X}\boldsymbol{w}$. Letting $\boldsymbol{x}_t \coloneqq \boldsymbol{X}(S_t)$, a value backup like Eq. (2.1) now becomes

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_t \Big(\hat{G}_t - \boldsymbol{w}^\top \boldsymbol{x}_t\Big) \boldsymbol{x}_t \,. \tag{2.2}$$

Because the linear function is underparameterized, the result of the Bellman operator, $T_\pi \boldsymbol{v}$, will almost surely lie outside the span of the feature vectors; the result must be projected back onto the manifold of representable value functions. Let $d\colon \mathcal{S} \to [0, 1]$ be a state-weighting distribution[9] and define $\boldsymbol{D} \coloneqq \operatorname{diag}(\boldsymbol{d})$. The linear projection operator is equal to the matrix

$$\boldsymbol{\Pi} \coloneqq \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{D} \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{D} \tag{2.3}$$

(see Tsitsiklis and Van Roy, 1997, Eq. 1). TD methods under linear function approximation therefore solve the *projected* Bellman equation,

$$\boldsymbol{\Pi} T_\pi(\boldsymbol{X}\boldsymbol{w}) = \boldsymbol{X}\boldsymbol{w}\,. \tag{2.4}$$

Analogous results for action-value methods exist as well. In this case, the value backup operates on an action-value estimate, $Q(S_t, A_t)$, and is defined as

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \Big(\hat{G}_t - Q(S_t, A_t) \Big)\,,$$

where $\hat{G}_t$ represents a sample estimate from an operator H applied to Q. Linear function approximation is achieved by extending the notions of feature vectors and the weighting distribution from states to state-action pairs.

2.3 Multistep Returns

Credit assignment with the 1-step TD error can be slow, as it suffers from high estimation bias. Multistep returns reduce bias and can significantly accelerate learning. The simplest multistep return estimator is the n-step return:

$$G_t^{(n)} \coloneqq \left(\sum_{i=0}^{n-1} \gamma^i R_{t+1+i} \right) + \gamma^n V(S_{t+n})\,. \tag{2.5}$$

The n-step return combines the immediate n discounted rewards before bootstrapping from the value function, generalizing the notion of the 1-step TD estimate discussed previously. The n-step return corresponds to a sample estimate of the n-iterated Bellman operator, T_π^n. Increasing n enhances the immediate sensitivity to future rewards and decreases the bias, but at the expense of greater variance which may require more samples to converge to the

[9] Typically, this is the stationary distribution, assuming that the MDP is aperiodic and irreducible under π.

true expectation. As $n \to \infty$, the dependency on the value function vanishes and $G_t^{(n)} \to G_t$, the Monte Carlo return. The choice of n therefore faces a bias-variance trade-off (Kearns and Singh, 2000). Although not commonly presented in this form, the n-step return is also expressible as a discounted sum of n TD errors:

$$G_t^{(n)} = V(S_t) + \sum_{i=0}^{n-1} \gamma^i \delta_{t+i} \,, \tag{2.6}$$

which is useful for proving a number of theoretical results.

The λ-return is an alternative multistep return estimator which originates from TD(λ). It is derived as a sum of discounted TD errors, which turns out to be equivalent to an exponentially weighted average of n-step returns:

$$G_t^\lambda \coloneqq V(S_t) + \sum_{i=0}^{\infty} (\gamma\lambda)^i \delta_{t+i} \,, \tag{2.7}$$

$$= (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)} \,, \tag{2.8}$$

where $\lambda \in [0, 1]$ is a decay hyperparameter. Although the infinite sum is not feasible to compute without episode termination or artificial truncation, it can be generated from sequential experiences efficiently using eligibility traces (see Section 2.4). When $\lambda = 0$, Eq. (2.7) reduces to the 1-step return.[10] When $\lambda = 1$, Eq. (2.8) reduces to the Monte Carlo return. The λ-return can thus be seen a smooth interpolation between these methods, presenting a bias-variance trade-off in the choice of λ as well (Kearns and Singh, 2000).

More generally, any weighted average of two or more n-step returns is also a well-defined return estimator (Watkins, 1989, Sec. 7.2). I call such a weighted average a compound return:

$$G_t^{\boldsymbol{c}} \coloneqq \sum_{n=1}^{\infty} c_n G_t^n \,, \tag{2.9}$$

where $\boldsymbol{c}$ is an infinite sequence of nonnegative weights such that $\sum_{n=1}^{\infty} c_n = 1$. In practice, $\boldsymbol{c}$ must be sparse or otherwise truncated to enable feasible computation. Examples of other compound returns include γ-returns (Konidaris et al., 2011) and Ω-returns (Thomas et al., 2015), though λ-returns are the most common.

[10] Technically, I must also define $0^0 \coloneqq \lim_{x \to 0^+} x^x = 1$.

2.4 Eligibility Traces

Multistep backups in the form of Eq. (2.1) are acausal; they must be delayed until the necessary quantities needed for constructing $\hat{G}_t$ are available. This is the "forward view" of TD learning. In general, this is the only way to implement arbitrary return estimates.

In the special case of λ-returns, eligibility traces provide an efficient mechanism for zero-delay learning, by incrementally generating the total update from individual TD errors as they are experienced. This is known as the "backward view" view of TD learning. Theoretically, every return estimator has a backward view, but it cannot be implemented with eligibility traces unless there is some exploitable structure like the exponential decay of λ-returns.

Recall from Eq. (2.7) that the λ-return can be expressed in terms of an infinite sum of time-decaying TD errors. The weight on each TD error is then a uniform, recursive function of the previous TD error's weight: $(\gamma\lambda)^i = \gamma\lambda \cdot (\gamma\lambda)^{i-1}$, where i is the time elapsed since the reinforced experience. Eligibility traces exploit this remarkable property to efficiently track each feature's receptiveness to the current TD error. The traces are represented by a zero-initialized vector $\boldsymbol{z} \in \mathbb{R}^{N_{\boldsymbol{w}}}$ that is decayed and incremented after each state visitation. The TD(λ) algorithm updates the value-function weights with the current TD error in proportion to the eligibility traces:

$$\boldsymbol{z} \leftarrow \gamma\lambda\boldsymbol{z} + \boldsymbol{x}_t \,,$$
$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_t\delta_t\boldsymbol{z} \,.$$

Crucially, learning now occurs on every time step, without delay. TD(λ) does not produce exactly the same updates as using G_t^λ in Eq. (2.2) does, but the discrepancy is small when the step size, α_t, also remains small. If the value-function parameters $\boldsymbol{w}$ were held fixed for all t, then these two different learning procedures would generate the same total parameter change, and in this sense the forward and backward views are equivalent.[11]

2.5 Experience Replay

Experience replay is a training technique for deep RL where each transition $(S_t, A_t, S_{t+1}, R_{t+1})$ is buffered in a first-in, first-out (FIFO) memory $\mathcal{D}$, and randomly resampled multiple times

[11] Forward-backward equivalence while the parameters change during learning is possible for true online TD methods (van Seijen and Sutton, 2014; van Seijen et al., 2016), but only with linear function approximation.

for training. The rationale is to repeatedly present training samples to the network to prevent it from drifting away from previously learned values. This is a significant departure from the previously discussed RL algorithms, which all process experiences in the order they were collected. Consequently, eligibility traces are not directly applicable here.

Before discussing the details of experience replay, I show how to generalize TD updates to nonlinear function approximators such as neural networks. To avoid confusion with the linear case, let $\boldsymbol{\theta} \in \mathbb{R}^{N_{\boldsymbol{\theta}}}$ be the network parameters. If the network were linear, then the gradient would just be $\nabla V(S_t) = \nabla(\boldsymbol{\theta}^\top \boldsymbol{x}_t) = \boldsymbol{x}_t$.[12] In fact, it turns out that making this substitution, $\boldsymbol{x}_t = \nabla V(S_t)$, in Eq. (2.2) produces the exact generalization of TD(0) for neural networks proposed by Sutton (1989):

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_t \Big(\hat{G}_t - V(S_t)\Big) \nabla V(S_t)$$

However, since value-based deep RL methods learn action values for control, I present this equation in terms of $Q(s, a)$ below. For any differentiable action-value function, a value backup has the form

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_t \Big(\hat{G}_t - Q(S_t, A_t)\Big) \nabla Q(S_t, A_t)\,.$$

However, this incremental update is not stable and is not pursued in this thesis, though I discuss it as part of future work in Section 7.2.4.

Notably, if I consider a stale copy $\boldsymbol{\theta}^-$ of the network weights—commonly referred to as a "target network" in deep RL (Mnih et al., 2015)—for computing $\hat{G}_t$, then the TD update can be written as the stochastic minimization of a mean squared error:[13]

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha_t \nabla \frac{1}{2}\Big(\hat{G}_t - Q(S_t, A_t)\Big)^2.$$

This implies that TD updates can be made independently of the order in which experiences are collected, by minimizing the loss

$$\mathcal{L}(\boldsymbol{\theta}) \coloneqq \mathbb{E}\left[\frac{1}{2}\Big(\hat{G} - Q(S, A)\Big)^2\right], \tag{2.10}$$

[12] In this thesis, the symbol ∇ is always assumed to mean the gradient with respect to the value-function parameters. It is also assumed that ∇ returns a column vector. Other inputs to the value function (e.g., S_t) are not considered differentiable arguments, to avoid the need for partial derivatives and simplify notation.

[13] Although TD resembles a stochastic gradient descent update, it does not follow the gradient of any objective function (Barnard, 1993). The derivation from a squared-error loss only works when $\hat{G}_t$ is considered a constant with respect to differentiation. For this reason, TD methods are classified as semi-gradient methods (Sutton and Barto, 2018, Sec. 9.3).

where the expectation is taken over samples uniformly distributed from the replay memory: $(S, A, \hat{G}) \sim \mathcal{U}(\mathcal{D})$. In practice, computing the expectation over the entire replay memory is too expensive, and minibatches are used instead. As a result, experiences are not reinforced in order they were collected, and may be used multiple times for learning.

2.6 Off-Policy Corrections with Importance Sampling

Off-policy prediction is where the agent learns the value function for a target policy π, while acting according to a (different) behavior policy b. Off-policy learning is nearly inevitable in control algorithms that rely on experience replay; as the agent's policy changes, the behavior data distribution in the replay buffer does not match that of the target policy.

For ease of exposition, I abstract away the details of experience replay when discussing off-policy corrections by assuming that samples are always collected from the same behavior policy, b. Applying any of the previously considered return estimators would yield biased expected returns, converging to q_b instead of q_π. Note that on-policy algorithms such as PPO avoid this issue by discarding off-policy data, but they are less efficient because they do not benefit from data reuse as much—one of the main perks of experience replay.

Importance sampling (IS; Kahn and Marshall, 1953) is the most common approach for correcting off-policy bias in RL. The IS ratio at time t is defined as $\rho_t \coloneqq \pi(A_t \mid S_t)/b(A_t \mid S_t)$. I assume that $\pi(a \mid s) > 0 \implies b(a \mid s)$, such that this ratio is always well-defined.

Before applying the IS corrections to a return estimate, I first note that each TD error can be partially corrected by computing its expected value under π:

$$\delta_t^\pi \coloneqq R_{t+1} + \gamma \left(\sum_{a' \in \mathcal{A}} \pi(a' \mid S_{t+1})\, Q(S_{t+1}, a') \right) - Q(S_t, A_t)\,.$$

That is to say, it holds that $\mathbb{E}_b[\delta_t^\pi \mid (S_t, A_t) = (s, a)] = \mathbb{E}_\pi[\delta_t^\pi \mid (S_t, A_t) = (s, a)]$, which is not true in general for the standard TD error, δ_t. The main role of IS is thus to correct the remaining discrepancy between the multistep trajectory distributions under the differing target and behavior policies.

A naive correction would multiply an entire return estimate $\hat{G}_t$ by the product of IS ratios along the trajectory, but this would greatly amplify the variance. An equivalent, lower-variance estimator uses *per-decision* corrections (Precup et al., 2000). Per-decision cor-

rections reduce variance by correcting each transition only as needed, exploiting the Markov property of the MDP. For example, the action-value λ-return is corrected as

$$G_t^\lambda = Q(S_t, A_t) + \sum_{i=0}^{\infty} (\gamma\lambda)^i \left(\prod_{j=t+1}^{t+i} \rho_j \right) \delta_{t+i}^\pi \,,$$

where $\prod_{j=t+1}^{t} \rho_j \coloneqq 1$ to handle the $i = 0$ term. Still, this estimator can have unacceptably high variance in practice.

Variance can be further reduced by modifying the IS ratios themselves. This exchanges variance for bias, allowing finer control over the bias-variance trade-off. Munos et al. (2016) introduced generalized per-decision corrections, which have the form

$$\hat{G}_t = Q(S_t, A_t) + \sum_{i=0}^{\infty} \gamma^i \left(\prod_{j=t+1}^{t+i} \tilde{\rho}_j \right) \delta_{t+i}^\pi \,, \tag{2.11}$$

where $\tilde{\rho}_j \coloneqq h(S_j, A_j)$, the function $h \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is an arbitrary weighting over state-action pairs, and $\prod_{j=t+1}^{t} \tilde{\rho}_j \coloneqq 1$ to again handle the $i = 0$ term. This generalizes the corrected λ-return by replacing $\lambda\rho_j$ with an arbitrary coefficient, $\tilde{\rho}_j$. As long as $0 \leq \tilde{\rho}_j \leq \rho_j$ always holds, any choice of h guarantees convergence to q_π (Munos et al., 2016, Theorem 1), though different choices may have beneficial properties. For example, Retrace (Munos et al., 2016) defines $\tilde{\rho}_j = \lambda \min(1, \rho_j)$ to ensure that the product of coefficients cannot grow exponentially, thereby limiting variance. Tree Backup (Precup et al., 2000) is equivalent to $\tilde{\rho}_j = \lambda\, \pi(A_j \mid S_j)$, which produces a conservative, convergent algorithm that does not require knowledge of b. These and several other off-policy examples are detailed in Chapter 6.

Chapter 3

Variance Reduction via Averaging n-step Returns

In the absence of stochasticity, learning would be easy. Monte Carlo returns would assign a deterministic value to each action that perfectly reflects its expected utility. In reality, variance from the policy, environment, and rewards greatly limits the useful length of multistep returns. Mitigating this variance is key to reliably assigning credit over longer horizons.

Compound returns are multistep estimators formed by averaging two or more n-step returns. Watkins (1989) developed the concept of compound returns by studying the theoretical target of TD(λ), an exponentially weighted average of n-step returns now known as the λ-return. More generally, he showed that any weighted average of n-step returns has an error-reduction property and is a theoretically sound target for TD learning.

Being an average of statistical estimates, it would seem that compound returns should reduce variance. However, this had never been documented in the RL literature. In the absence of any assumptions about the environment, Monte Carlo returns can actually have lower variance than 1-step returns (see Counterexample 3.3), defying the conventional wisdom of bootstrapping. Additionally, two distinct n-step returns obtained from a sampled trajectory are not independent, so averaging them does not necessarily reduce variance. Finally, when variance reduction is achieved, it does not automatically make learning faster if bias is increased.

For these reasons, the question of variance reduction turns out to be highly complex. My key insight is that the variance of two returns can be meaningfully compared when their biases are equalized. Using this and additional assumptions, I prove that compound returns exhibit a *variance-reduction property* which improves sample efficiency compared to n-step returns.

3.1 Variance Assumptions and Motivation

I start by developing reasonable models for the variance of n-step and compound returns, building on the work of Konidaris et al. (2011). Note that the real quantity of interest in this chapter is the *conditional* variance of the backup error, $\hat{G}_t - V(S_t)$. The degree of this quantity's deviation from its expected value is what ultimately impacts the performance of value-based RL backups like Eq. (2.1). Nevertheless, this turns out to be the same as the conditional variance of the return, since $V(S_t)$ contributes no randomness:

$$\mathrm{Var}[\hat{G}_t - V(S_t) \mid S_t] = \mathrm{Var}[\hat{G}_t \mid S_t]\,. \tag{3.1}$$

This equivalence allows the variances of a return and its error to be interchanged, depending on which is more computationally convenient.

Modeling a compound return's variance is challenging because it typically requires making assumptions about how the variance of n-step returns increases as a function of n, as well as how strongly correlated different n-step returns are. If these assumptions are too strong, the derived variances fail to reflect reality and lead to poorly informed algorithmic choices. For instance, consider the following compound return, which I call a *two-bootstrap return*:

$$(1-c)\,G_t^{(n_1)} + c\,G_t^{(n_2)}, \quad c \in (0,1), \quad n_1 < n_2\,. \tag{3.2}$$

Let $\sigma_n^2 \coloneqq \mathrm{Var}[G_t^{(n)} \mid S_t]$. The variance of Eq. (3.2) is

$$(1-c)^2\sigma_{n_1}^2 + c^2\sigma_{n_2}^2 + 2c(1-c)\sigma_{n_1}\sigma_{n_2}\rho\,,$$

where $\rho = \mathrm{Corr}[G_t^{(n_1)}, G_t^{(n_2)} \mid S_t]$. To evaluate this expression, it is tempting to assume either $\rho = 0$ to remove the covariance term, or $\rho = 1$ because both returns are generated from the same trajectory. However, neither assumption is fully correct. This can be seen more clearly by decomposing the returns into their constituent TD errors using Eq. (2.6):

$$\begin{aligned} G_t^{(n_1)} &= V(S_t) + \sum_{i=0}^{n_1-1} \gamma^i \delta_{t+i}\,, \\ G_t^{(n_2)} &= V(S_t) + \sum_{i=0}^{n_1-1} \gamma^i \delta_{t+i} + \sum_{i=n_1}^{n_2-1} \gamma^i \delta_{t+i}\,. \end{aligned}$$

Because the returns share the first n_1 TD errors, averaging them as in Eq. (3.2) can only reduce the variance from the last $n_2 - n_1$ TD errors (highlighted in red above). Two different

n-step returns are therefore neither uncorrelated nor perfectly correlated; they consist of various random elements, some of which are shared and cannot be averaged, and others which are not shared and can be averaged. Models that fail to account for this partial averaging lead to poor variance predictions.

To be more accurate, the variance model should start with the TD error as the fundamental unit of randomness within a return. In the following analysis, I generalize the n-step return variance assumptions made by Konidaris et al. (2011, Sec. 3) to obtain an expression for the covariance between two TD errors. I assume that there exist $\kappa > 0$ and $\rho \in [0, 1]$ such that the following hold:

Assumption 3.1 (Uniform TD-error variance)**.** $\text{Var}[\delta_{t+i} \mid S_t] \approx \kappa, \ \forall\ i \geq 0.$

Assumption 3.2 (Uniform TD-error correlation)**.** $\text{Corr}[\delta_{t+i}, \delta_{t+j} \mid S_t] \approx \rho, \ \forall\ i \geq 0, \forall\ j \neq i.$

Under these assumptions, each TD error has equal variance, and all pairs of distinct TD errors are equally correlated (implying they have the same covariance).

Although these assumptions are not completely realistic, I must make some assumptions about the TD errors because not much can be said otherwise about the variance of a return estimate. For example, without assumptions, it is possible to contrive problems in which 1-step TD learning actually has higher variance than Monte Carlo estimation, as illustrated by the following counterexample.

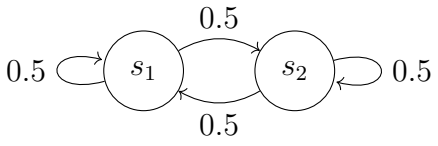


Figure 3.1: MRP in which Monte Carlo methods have lower variance than TD methods. Rewards are zero.

Counterexample 3.3. *Consider a 2-state MRP with equiprobable transitions and zero reward for all transitions (see Figure 3.1). Whenever $V(s_1) \neq V(s_2)$ and $\gamma \neq 0$, 1-step returns have strictly higher variance than Monte Carlo returns.*

In this specific counterexample, any amount of bootstrapping creates variance due to the differing value estimates, in contrast with the Monte Carlo returns which are always zero and therefore variance-free.

However, generally speaking, one would *expect* 1-step TD to have much lower variance than Monte Carlo, because summing together more TD errors (i.e., random variables) would

likely increase variance. This intuition motivates Assumptions 3.1 and 3.2, as I want a variance model that captures this behavior while remaining tractable for analysis. I further discuss the justification and historical context of these assumptions in Appendix A.1.

Now, I unify Assumptions 3.1 and 3.2 as

$$\mathrm{Cov}[\delta_{t+i}, \delta_{t+j} \mid S_t] = ((1-\rho)\mathbf{1}_{\{i=j\}} + \rho)\kappa\,, \tag{3.3}$$

where $\mathbf{1}_{\{i=j\}}$ is the indicator function. In the proposition below, I derive a variance model for the n-step return by decomposing the return into a sum of discounted TD errors and then adding up the pairwise covariances given by Eq. (3.3). For brevity, I define the function

$$\Gamma_k(n) \coloneqq \begin{cases} (1-\gamma^{kn}) \,/\, (1-\gamma^k)\,, & \text{for } \gamma < 1\,, \\ n\,, & \text{for } \gamma = 1\,, \end{cases}$$

to represent partial sums of a geometric series with common ratio γ^k.

Proposition 3.4. *Under Assumptions 3.1 and 3.2, the variance of an n-step return is*

$$\mathrm{Var}[G_t^{(n)} \mid S_t] = (1-\rho)\,\Gamma_2(n)\,\kappa + \rho\,\Gamma_1(n)^2\,\kappa\,.$$

Proof. From Eq. (2.6), the n-step error can be expressed as a finite summation of TD errors:

$$G_t^{(n)} - V(S_t) = \sum_{i=0}^{n-1} \gamma^i \delta_{t+i}\,.$$

Using this and Eq. (3.1), I calculate the covariance of two n-step returns with lengths n_1, n_2:

$$\begin{aligned}
\mathrm{Cov}[G_t^{(n_1)}, G_t^{(n_2)} \mid S_t] &= \mathrm{Cov}\left[\sum_{i=0}^{n_1-1} \gamma^i \delta_{t+i}, \sum_{j=0}^{n_2-1} \gamma^j \delta_{t+j} \;\middle|\; S_t \right] \\
&= \sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} \gamma^{i+j} \mathrm{Cov}[\delta_{t+i}, \delta_{t+j} \mid S_t] \\
&= \sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} \gamma^{i+j} ((1-\rho)\mathbf{1}_{\{i=j\}} + \rho)\kappa \\
&= (1-\rho) \sum_{i=0}^{\min(n_1,n_2)-1} \gamma^{2i}\kappa + \rho \sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} \gamma^{i+j}\kappa \\
&= (1-\rho) \sum_{i=0}^{\min(n_1,n_2)-1} \gamma^{2i}\kappa + \rho \sum_{i=0}^{n_1-1} \gamma^i \sum_{j=0}^{n_2-1} \gamma^j \kappa
\end{aligned}$$

$$= (1-\rho)\,\Gamma_2(\min(n_1,n_2))\,\kappa + \rho\,\Gamma_1(n_1)\,\Gamma_1(n_2)\,\kappa\,. \tag{3.4}$$

Because $\mathrm{Var}[G_t^{(n)} \mid S_t] = \mathrm{Cov}[G_t^{(n)}, G_t^{(n)} \mid S_t]$, then by letting $n = n_1 = n_2$, I obtain the n-step variance formula and the proof is complete. □

My n-step variance model linearly interpolates between an optimistic case where TD errors are uncorrelated ($\rho = 0$) and a pessimistic case where TD errors are maximally correlated ($\rho = 1$). In the maximum-variance undiscounted scenario, I have $\Gamma_1(n) = \Gamma_2(n) = n$, so the model becomes $(1-\rho)n\kappa + \rho n^2\kappa$: i.e., it interpolates between linear and quadratic functions. In Appendix A.2, I demonstrate that these bounds are consistent with empirical data in real environments, even when my assumptions do not fully hold.

Eq. (3.3) enables me to go beyond n-step returns and calculate variances for arbitrary compound returns. Recall from Section 2.3 that a compound return is a nonnegatively weighted average of n-step returns:

$$G_t^{\boldsymbol{c}} \coloneqq \sum_{n=1}^{\infty} c_n G_t^n\,, \tag{2.9}$$

where $\sum_{n=1}^{\infty} c_n = 1$ and at least two weights are nonzero. I compute the variance by again decomposing the return into a sum of TD errors (see the next lemma) and then applying Assumptions 3.1 and 3.2 to derive a compound variance model in the following proposition.

Lemma 3.5. *Any compound error can be written as a weighted sum of TD errors:*

$$G_t^{\boldsymbol{c}} - V(S_t) = \sum_{i=0}^{\infty} \gamma^i h_i \delta_{t+i}\,,\ \textit{where } h_i \coloneqq \sum_{n=i+1}^{\infty} c_n\,.$$

Proof. See Appendix A.3.1. □

Proposition 3.6. *Under Assumptions 3.1 and 3.2, the variance of a compound return is*

$$\mathrm{Var}[G_t^{\boldsymbol{c}} \mid S_t] = (1-\rho)\sum_{i=0}^{\infty} \gamma^{2i} h_i^2 \kappa + \rho \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} \gamma^{i+j} h_i h_j \kappa\,.$$

Proof. From Lemma 3.5 and Eq. (3.1), the variance of the compound return is

$$\mathrm{Var}\left[G_t^{\boldsymbol{c}} \mid S_t\right] = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} \mathrm{Cov}[\gamma^i h_i \delta_{t+i}, \gamma^j h_j \delta_{t+j} \mid S_t]$$

$$
\begin{aligned}
&= \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \gamma^{i+j} h_i h_j \mathrm{Cov}[\delta_{t+i}, \delta_{t+j} \mid S_t] \\
&= \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \gamma^{i+j} h_i h_j ((1-\rho)\mathbf{1}_{\{i=j\}} + \rho)\kappa \\
&= (1-\rho) \sum_{i=0}^{\infty} \gamma^{2i} h_i^2 \kappa + \rho \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \gamma^{i+j} h_i h_j \kappa \,,
\end{aligned}
$$

which establishes Prop. 3.6. □

The eligibility weights, $(h_i)_{i=0}^{\infty}$, specify the variance of a compound return. For instance, the λ-return assigns an eligibility of $h_i = \sum_{n=i+1}^{\infty}(1-\lambda)\lambda^{n-1} = \lambda^i$ to TD error δ_{t+i}, which matches TD(λ) in Eq. (2.7). Substituting this weight into Prop. 3.6 and solving the geometric series yields the modeled variance for the λ-return:

$$
\begin{aligned}
\mathrm{Var}[G_t^{\lambda} \mid S_t] &= (1-\rho) \sum_{i=0}^{\infty} (\gamma\lambda)^{2i} \kappa + \rho \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} (\gamma\lambda)^{i+j} \kappa \\
&= \frac{(1-\rho)\kappa}{1-(\gamma\lambda)^2} + \frac{\rho\kappa}{(1-\gamma\lambda)^2} \,.
\end{aligned}
$$

As expected, the variance of the λ-return is a monotonically increasing function of λ, and is unbounded when $\gamma\lambda = 1$.

3.2 Variance-Reduction Property

Prop. 3.6 provides a method for calculating variance, but I would still like to show that compound returns reduce variance relative to n-step returns. To do this, I first need a way to relate two returns in terms of their expected performance. This is because low variance by itself is not sufficient for fast learning; for example, the 1-step return has very low variance, but learns slowly.

In the discounted setting, a good candidate for expected learning speed is the return's contraction modulus. The contraction modulus is the constant factor by which the maximum value-function error between V and v_π is guaranteed to be reduced. When the contraction modulus is less than 1, the return estimator exhibits an *error-reduction property*: i.e., the maximum error decreases on average with every backup iteration of Eq. (2.1). This property is commonly used in conjunction with the Banach fixed-point theorem to prove that V

eventually converges to v_π (see, e.g., Bertsekas and Tsitsiklis, 1996, Sec. 4.3). The error-reduction property of compound returns was first identified by Watkins (1989, Sec. 7.2) and is expressed formally as

$$\max_{s\in\mathcal{S}} \left| \mathbb{E}\left[G_t^{\boldsymbol{c}} \mid S_t = s\right] - V(s) \right| \leq \beta \max_{s\in\mathcal{S}} \left| v_\pi(s) - V(s) \right|, \tag{3.5}$$

where $\beta \coloneqq \sum_{k=1}^{\infty} c_k \gamma^k$. The contraction modulus, β, is a weighted average of each individual n-step return's contraction modulus, γ^n. I can compare compound returns to n-step returns that have equivalent error-reduction properties by solving the equation

$$\gamma^n = \beta \tag{3.6}$$

for n. I call this value of n the *effective n-step* of the compound return, since the compound return reduces the worst-case error as though it were an n-step return whose length is the solution to the above equation (see Prop. 3.7 below). If the solution is not an integer, then the error-reduction property falls between that of the two closest n-step returns. In undiscounted settings, I cannot directly compare contraction moduli (because $\beta = 1$ for all compound returns), but I can still solve the limit as $\gamma \to 1$ to define the effective n-step.

Proposition 3.7 (Effective n-step of compound return)**.** *Let $G_t^{\boldsymbol{c}}$ be any compound return with contraction modulus β and let*

$$n = \begin{cases} \log_\gamma \beta\,, & \text{if } 0 < \gamma < 1\,, \\ \sum_{k=1}^{\infty} c_k k\,, & \text{if } \gamma = 1\,. \end{cases} \tag{3.7}$$

When n is an integer, $G_t^{\boldsymbol{c}}$ shares the same bound in Eq. (3.5) *as the n-step return $G_t^{(n)}$.*

Proof. When $0 < \gamma < 1$, I can take the logarithm of both sides of Eq. (3.6) to get

$$n = \log_\gamma \beta = \frac{\log\left(\sum_{k=1}^{\infty} c_k \gamma^k\right)}{\log \gamma}\,.$$

For the undiscounted case, I would like to evaluate this expression at $\gamma = 1$; however, since $\sum_{k=1}^{\infty} c_k = 1$ from the definition of a compound return, I arrive at an indeterminate form, $\frac{0}{0}$. Instead, I can apply L'Hôpital's rule to evaluate the limit as $\gamma \to 1$:

$$\lim_{\gamma\to 1} \frac{\log\left(\sum_{k=1}^{\infty} c_k \gamma^k\right)}{\log \gamma} = \lim_{\gamma\to 1} \frac{\frac{d}{d\gamma}\log\left(\sum_{k=1}^{\infty} c_k \gamma^k\right)}{\frac{d}{d\gamma}\log \gamma}$$

$$
\begin{aligned}
&= \lim_{\gamma \to 1} \frac{\left(\sum_{k=1}^{\infty} c_k \gamma^{k-1} k\right) / \left(\sum_{k=1}^{\infty} c_k \gamma^k\right)}{1 \,/\, \gamma} \\
&= \lim_{\gamma \to 1} \frac{\sum_{k=1}^{\infty} c_k \gamma^k k}{\sum_{k=1}^{\infty} c_k \gamma^k} \\
&= \frac{\sum_{k=1}^{\infty} c_k k}{\sum_{k=1}^{\infty} c_k} \\
&= \sum_{k=1}^{\infty} c_k k \,,
\end{aligned}
$$

where the last step follows again from the fact that $\sum_{k=1}^{\infty} c_k = 1$. This establishes the case where $\gamma = 1$ and completes the proof. □

With or without discounting, an n-step return and a compound return that satisfy Eq. (3.7) have the same error-reduction property. I refer to the quantity $\sum_{k=1}^{\infty} c_k k$ as the *center of mass* (COM) of a return, since it is the first moment of the weight distribution over n-step returns. Intuitively, this represents the average length into the future considered by the return—the undiscounted analog of the log contraction modulus, $\log_\gamma \beta$.

With the previous definitions, I am now ready to formalize the *variance-reduction property* of compound returns.

Theorem 3.8 (Variance-reduction property of compound returns)**.** *Let $G_t^{(n)}$ be any n-step return and let $G_t^{\mathbf{c}}$ be any compound return with the same effective n-step: i.e., Eq.* (3.7) *is satisfied. Under Assumptions 3.1 and 3.2, the inequality*

$$
\mathrm{Var}[G_t^{\mathbf{c}} \mid S_t] \leq \mathrm{Var}[G_t^{(n)} \mid S_t]
$$

always holds, and it is strict whenever TD errors are not perfectly correlated ($\rho < 1$).

Proof. Eq. (3.4) gives an expression for the covariance between two n-step returns. I use this to derive an alternative formula for the variance of a compound return:

$$
\begin{aligned}
\mathrm{Var}[G_t^{\mathbf{c}} \mid S_t] &= \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \mathrm{Cov}[c_i G_t^{(i)}, c_j G_t^{(j)} \mid S_t] \\
&= \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} c_i c_j \mathrm{Cov}[G_t^{(i)}, G_t^{(j)} \mid S_t] \\
&= \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} c_i c_j \big((1-\rho)\, \Gamma_2(\min(i,j))) + \rho\, \Gamma_1(i)\, \Gamma_1(j)\big)\kappa
\end{aligned}
$$

$$= (1-\rho)\sum_{i=1}^{\infty}\sum_{j=1}^{\infty} c_i c_j\, \Gamma_2(\min(i,j))\,\kappa + \rho\sum_{i=1}^{\infty}\sum_{j=1}^{\infty} c_i c_j\, \Gamma_1(i)\,\Gamma_1(j)\,\kappa\,.$$

I analyze both sums separately, starting with the left term. If I assume $\gamma < 1$ for now, then $\gamma^n = \sum_{i=1}^{\infty} c_i \gamma^i$ by Eq. (3.7). Further note that $\min(i,j) \leq (i+j)\,/\,2$, with the inequality strict if $i \neq j$. Because Γ_2 is monotonically increasing, it follows that

$$\begin{aligned}
\sum_{i=1}^{\infty}\sum_{j=1}^{\infty} c_i c_j\, \Gamma_2(\min(i,j)) &< \sum_{i=1}^{\infty}\sum_{j=1}^{\infty} c_i c_j\, \Gamma_2\left(\frac{i+j}{2}\right) \\
&= \sum_{i=1}^{\infty}\sum_{j=1}^{\infty} c_i c_j \left(\frac{1-\gamma^{i+j}}{1-\gamma^2}\right) \\
&= \frac{1-\sum_{i=1}^{\infty}\sum_{j=1}^{\infty} c_i c_j \gamma^{i+j}}{1-\gamma^2} \\
&= \frac{1-\gamma^{2n}}{1-\gamma^2} \\
&= \Gamma_2(n)\;.
\end{aligned}$$

The inequality is strict because at least two weights in $\boldsymbol{c}$ are nonzero by definition of a compound return, guaranteeing at least one element in the sum has $i \neq j$. If instead $\gamma = 1$, then $n = \sum_{i=1}^{\infty} c_i i$ by Eq. (3.7) and $\Gamma_2(\min(i,j)) = \min(i,j)$. Therefore, by Jensen's inequality, I have

$$\begin{aligned}
\sum_{i=1}^{\infty}\sum_{j=1}^{\infty} c_i c_j\, \Gamma_2(\min(i,j)) &= \sum_{i=1}^{\infty}\sum_{j=1}^{\infty} c_i c_j \min(i,j) \\
&< \min\left(\sum_{i=1}^{\infty} c_i i,\ \sum_{j=1}^{\infty} c_j j\right) \\
&= \min(n,n) \\
&= \Gamma_2(n)\;.
\end{aligned}$$

Again, the inequality is strict by definition of a compound return, so I conclude that

$$\sum_{i=1}^{\infty}\sum_{j=1}^{\infty} c_i c_j\, \Gamma_2(\min(i,j)) < \Gamma_2(n), \quad \text{for } 0 < \gamma \leq 1\,.$$

I now address the right term from before. I show that Γ_1 is invariant under a weighted average under my assumption that Eq. (3.7) holds. If $\gamma < 1$, then

$$\sum_{i=1}^{\infty} c_i\, \Gamma_1(i) = \sum_{i=1}^{\infty} c_i \left(\frac{1-\gamma^i}{1-\gamma}\right) = \frac{1-\sum_{i=1}^{\infty} c_i \gamma^i}{1-\gamma} = \frac{1-\gamma^n}{1-\gamma} = \Gamma_1(n)\;. \tag{3.8}$$

If $\gamma = 1$, then

$$\sum_{i=1}^{\infty} c_i \, \Gamma_1(i) = \sum_{i=1}^{\infty} c_i i = n = \Gamma_1(n) \; .$$

Thus, regardless of γ, the right term becomes

$$\sum_{i=1}^{\infty} \sum_{j=1}^{\infty} c_i c_j \, \Gamma_1(i) \, \Gamma_1(j) = \Gamma_1(n)^2 \; .$$

Putting everything together, I have shown that

$$\mathrm{Var}[G_t^{\mathbf{c}} \mid S_t] \leq (1-\rho) \, \Gamma_2(n) \, \kappa + \rho \, \Gamma_1(n)^2 \, \kappa \, ,$$

where the right-hand side is the n-step variance given by Prop. 3.4. As I showed above, this inequality is strict whenever the first term is active (i.e., $\rho < 1$), completing the proof. □

Theorem 3.8 shows that whenever a compound return has the same contraction modulus ($\gamma < 1$) or COM ($\gamma = 1$) as an n-step return, it has lower variance under Assumptions 3.1 and 3.2, as long as the TD errors are not perfectly correlated. Perfect correlation between all TD errors would not occur except in contrived, maximum-variance MDPs; thus, compound returns reduce variance in most cases. Crucially, variance reduction is achieved for any type of weighted average—although the magnitude of reduction does depend on the specific choice of weights. The exact amount, in terms of κ, can be calculated by subtracting the compound variance from the n-step variance for a given contraction modulus or COM. As an example, I bound the variance reduction of a λ-return in the following corollary.

Corollary 3.9 (Variance reduction of λ-return)**.** *Under Assumptions 3.1 and 3.2, for all $\gamma \in [0, 1]$, the magnitude of variance reduction for a λ-return is bounded such that*

$$0 \leq \mathrm{Var}[G_t^{(n)} \mid S_t] - \mathrm{Var}[G_t^{\lambda} \mid S_t] \leq (1-\rho) \frac{\lambda}{1-\lambda^2} \kappa \, .$$

Proof. See Appendix A.3.2. □

The magnitude of variance reduction increases monotonically as $\lambda \to 1$, showing that the potential for variance reduction improves as the effective n-step of the return increases. Interestingly, this is reflected in my later random-walk experiments in Section 3.4 (observe the performance gaps in Figure 3.3 when $\alpha \approx 1$). Future work can derive the weights that maximize the variance reduction in Theorem 3.8, but this will likely require more advanced techniques such as the calculus of variations (see Section 7.2.1).

3.3 Finite-Time Analysis

It might not be obvious that reducing variance leads to faster learning; indeed, in other settings such as direct policy optimization, this is not always the case (see Chung et al., 2021). I conduct a finite-time analysis of compound TD learning to prove that lower variance does lead to faster learning in this setting. I consider linear function approximation (recall Section 2.4), with tabular methods recoverable by using one-hot features. My theorem below generalizes recent analysis of 1-step TD learning (Bhandari et al., 2018, Theorem 2).

Theorem 3.10 (Finite-Time Analysis)**.** *Suppose TD learning with linear function approximation is applied under an i.i.d. state model (see Assumption A.1 in Appendix A.3.3) using compound return $G_t^{\boldsymbol{c}}$ as its target. Let β be the return's contraction modulus and let $\sigma^2 \geq 0$ be the return's variance. Assume that the features are normalized such that $\|\boldsymbol{X}(s)\|_2^2 \leq 1$, $\forall\ s \in \mathcal{S}$. Define $C \coloneqq \left(\|\boldsymbol{r}\|_\infty + (1+\gamma)\|\boldsymbol{w}^*\|_\infty\right) / (1-\gamma)$, where $\boldsymbol{w}^*$ is the minimizer of the projected Bellman error for $G_t^{\boldsymbol{c}}$. For any $T \geq (4/(1-\beta))^2$ and a constant step size $\alpha = 1/\sqrt{T}$,*

$$\mathbb{E}\left[\|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\bar{\boldsymbol{w}}_T}\|_{\boldsymbol{D}}^2\right] \leq \frac{\|\boldsymbol{w}^* - \boldsymbol{w}\|_2^2 + 2(1-\beta)^2 C^2 + 2\sigma^2}{(1-\beta)\sqrt{T}},$$

where $\bar{\boldsymbol{w}}_T \coloneqq \frac{1}{T}\sum_{t=0}^{T-1} \boldsymbol{w}_t$.

Proof. See Appendix A.3.3. □

With a constant step size, compound TD learning (and hence n-step TD learning as a special case) reduces the value-function error at the same asymptotic rate of $O(1/\sqrt{T})$ for any return estimator. However, both the contraction modulus β and the return variance σ^2 greatly influence the magnitude of the constant that multiplies this rate. Given an n-step return and a compound return with the same contraction modulus, the compound return has lower variance by Theorem 3.8; if this bound is tight, then the compound return should therefore converge faster to its respective TD fixed point. Although the two returns' fixed points may generally be different, I show that the bound on their solution quality is the same, by generalizing Lemma 6 of Tsitsiklis and Van Roy (1997) for an arbitrary return.

Proposition 3.11 (TD Solution Quality)**.** *Let $\boldsymbol{w}^*$ be the minimizer of the projected Bellman error under linear function approximation for any n-step or compound return with contraction*

modulus β. The following bound always holds:

$$\|\boldsymbol{X}\boldsymbol{w}^* - \boldsymbol{v}_\pi\|_{\boldsymbol{D}} \leq \frac{1}{1-\beta}\|\boldsymbol{\Pi}\boldsymbol{v}_\pi - \boldsymbol{v}_\pi\|_{\boldsymbol{D}}\,.$$

Proof. See Appendix A.3.4. □

This shows that the fixed-point error for an arbitrary return is always within $1 \,/\, (1-\beta)$ times the optimal error. Since β is equalized for the two returns in Theorem 3.10, the quality bound of these two fixed points is the same, but the compound return has lower variance and may reach its fixed point faster.

3.4 Comparing λ-Returns and n-step Returns

Although the λ-return is often motivated by its efficient implementation using TD(λ) and eligibility traces, my theory indicates that λ-returns can also promote faster learning via variance reduction. I provide empirical support for this by demonstrating faster learning in the random-walk experiment from Sutton and Barto (2018, Sec. 12.1). In this environment, the agent begins in the center of a linear chain of 19 connected states and can move either left or right (see Figure 3.2). The agent receives a reward only if it reaches one of the far ends of the chain (-1 for the left, $+1$ for the right), in which case the episode terminates. The agent's policy randomly moves in either direction with equal probability. I train the agents for 10 episodes, updating the value functions after each episode with offline backups like Eq. (2.1). To pair the n-step returns and λ-returns together, I derive the effective λ for an n-step return in the following proposition.

Proposition 3.12 (Effective λ of n-step return)**.** *For any $n \geq 1$ and $\gamma < 1$, the λ-return with*

$$\lambda = \frac{1-\gamma^{n-1}}{1-\gamma^n}$$

has the same contraction modulus as the n-step return. For any $n \geq 1$ and $\gamma = 1$, the λ-return with

$$\lambda = \frac{n-1}{n}$$

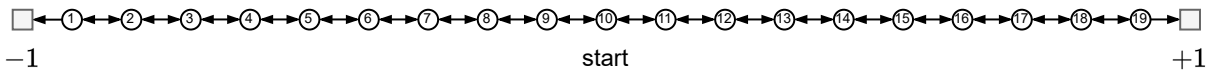


Figure 3.2: The 19-state random walk (Sutton and Barto, 2018, Sec. 12.1).

Table 3.1: Common n-step returns and λ-returns with equal COMs.

n	2	3	4	5	10	20	50	100
λ	0.5	0.67	0.75	0.8	0.9	0.95	0.98	0.99

has the same COM as the n-step return.

Proof. See Appendix A.3.5. □

Because this is an undiscounted task, I use the relationship $\lambda = (n-1)\,/\,n$ to generate several (n, λ)-pairs with equal COMs in Table 3.1.[14] For my experiment, I choose four commonly used n-step values, $\{2, 3, 5, 10\}$, which correspond to $\lambda \in \{0.5, 0.67, 0.8, 0.9\}$. In Figure 3.3, I plot the average root mean square (RMS) value error (with respect to v_π) as a function of the step size α over 100 trials.[15] I also indicate 95% confidence intervals by the shaded regions.[16]

For all of the tested (n, λ)-pairs, an interesting trend emerges. In the left half of each plot, variance is not an issue because the step size is small and has plenty of time to average out the randomness in the estimates. Learning therefore progresses at a nearly identical rate for both the n-step return and the λ-return since they have the same COM (although there is a small discrepancy due to the truncation of the episodic task). However, as $\alpha \to 1$ in the right half of each plot, variance becomes a significant factor as the step size becomes too large to mitigate the noise in the updates. This causes the n-step return's error to diverge sharply compared to the λ-return's, as the λ-return manages variance more effectively. The lowest error attained by each λ-return is also better than that of its corresponding n-step return in all cases. Notably, neither of my variance assumptions hold perfectly in this environment, demonstrating that my variance model's predictions are useful in practical settings.

I also compare λ-returns and n-step returns in a much more challenging deep RL setting: learning simulated robotic locomotion with Proximal Policy Optimization (PPO; Schulman et al., 2017). PPO is a popular actor-critic method which already uses λ-returns, also known

[14] Another way to write this relationship is $n = 1\,/\,(1-\lambda)$, which makes it clear how the effective n-step is determined by λ.

[15] I use *trial* to refer to a single, independent run of an experiment. This should not be confused with an *episode*, which is defined by the task's termination conditions. A trial typically consists of many episodes.

[16] All 95% confidence intervals in this thesis are approximated by $1.96\,\hat{\sigma}\,/\,\sqrt{m}$, where $\hat{\sigma}$ is the sample standard deviation and m is the number of trials.

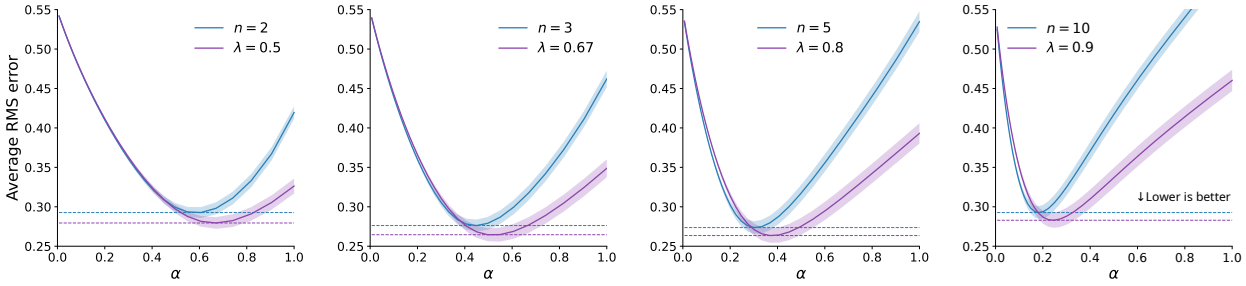


Figure 3.3: Comparing n-step returns and λ-returns, paired by COM, in a random walk. The results are averaged over 100 trials with 95% confidence intervals shaded. The dashed lines indicate the lowest errors attained.

as Generalized Advantage Estimation (GAE; Schulman et al., 2015b) in this context. This makes it easy to compare λ-returns with their equivalent n-step returns.

PPO trains a stochastic policy $\pi_{\boldsymbol{\theta}}$, parameterized by a neural network, by iteratively optimizing a clipped policy gradient objective over a sampled trajectory of on-policy experience. Let $\boldsymbol{\theta}_{\text{old}}$ be the parameters of the policy used to collect the trajectory; the importance-sampling ratio is $\rho_t(\boldsymbol{\theta}) \coloneqq \frac{\pi_{\boldsymbol{\theta}}(A_t|S_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(A_t|S_t)}$. Letting $\text{clip}(x, a, b) \coloneqq \min(\max(x, a), b)$, PPO optimizes the following objective using minibatched stochastic gradient ascent:

$$J(\boldsymbol{\theta}) \coloneqq \mathbb{E}\left[\min\Big(\rho_t(\boldsymbol{\theta})\,\text{Adv}_t,\ \text{clip}\big(\rho_t(\boldsymbol{\theta}),\ 1-\epsilon,\ 1+\epsilon\big)\Big)\right],$$

where $\text{Adv}_t \coloneqq \hat{G}_t - V(S_t)$ is the advantage estimate and $\epsilon \in (0, 1)$ is a trust-region hyperparameter. Because PPO is trained on relatively short trajectories of recent experiences, it is feasible to compute exact λ-returns for every experience, and so $\hat{G}_t = G_t^\lambda$ typically. (This is unlike the large-buffer regime of DQN and related algorithms that I discuss in Chapter 5.) However, I can also substitute any n-step return, $G_t^{(n)}$, for the purposes of this experiment.

The critic, $V(s)$, is trained by minimizing a squared-error loss, similar to the DQN loss in Eq. (2.10) but for a state-value function:

$$\mathcal{L}(\boldsymbol{\theta}) \coloneqq \mathbb{E}\left[\frac{1}{2}\Big(\hat{G}_t - V(S_t)\Big)^2\right].$$

Note that, for ease of presentation, I let $\boldsymbol{\theta}$ denote the parameters for both the policy and value function here, but they are typically implemented as separate networks.

I train the PPO agents in three MuJoCo environments (Todorov et al., 2012): Half Cheetah, Hopper, and Walker 2D. I use the CleanRL implementation (Huang et al., 2022) with

its default hyperparameters, which are similar to those of Schulman et al. (2017). The architectures for the actor and critic are 2-layer, 64-unit dense networks with tanh activations.

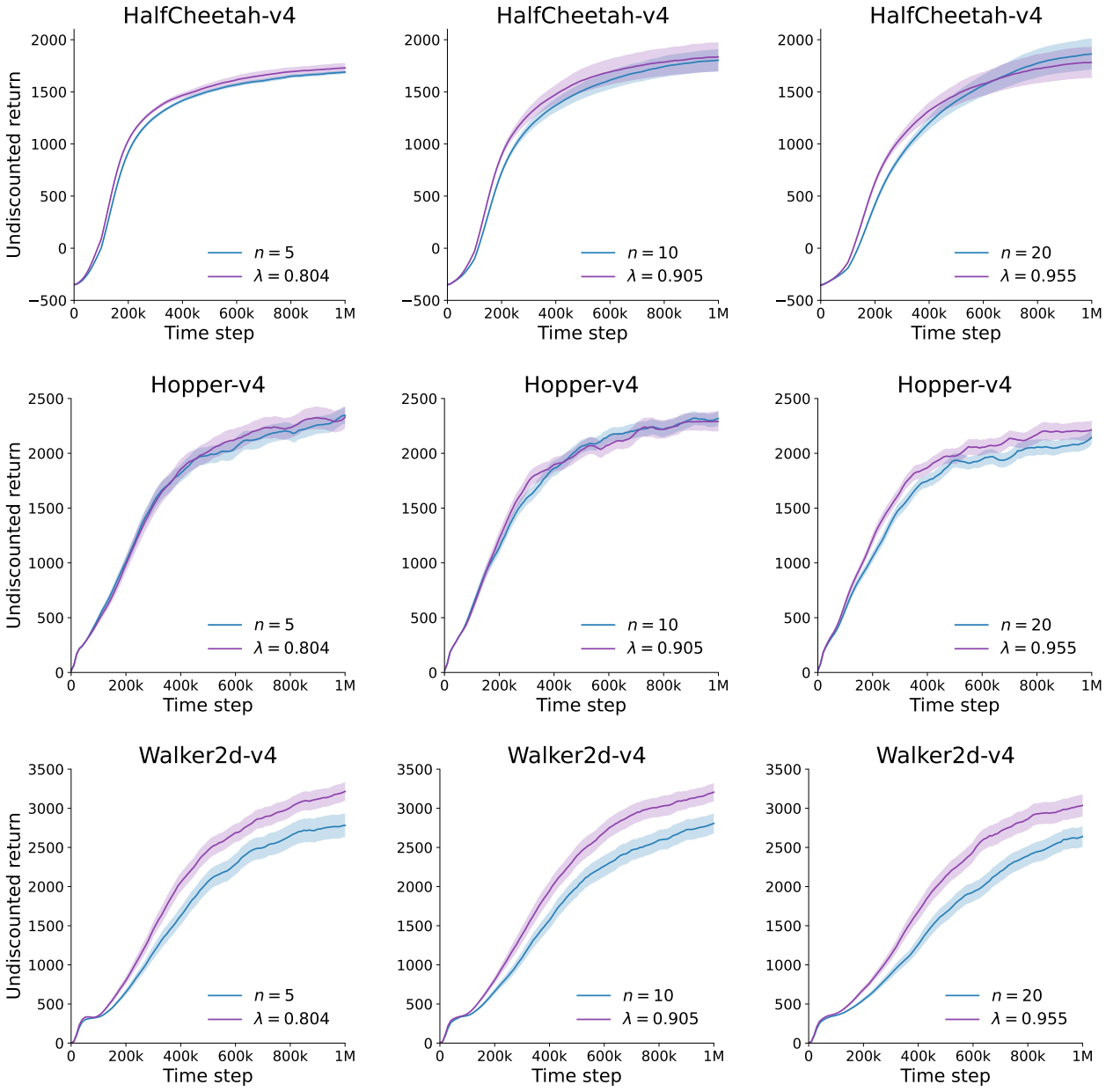


Figure 3.4: Learning curves for PPO with n-step returns and λ-returns in three MuJoCo environments. The results are averaged over 100 trials with 95% confidence intervals shaded.

I compare different returns with the values $(n, \lambda) \in (5, 0.804), (10, 0.905), (20, 0.955)$. I choose these pairs because the respective λ-returns and n-step returns have the same contraction moduli when $\gamma = 0.99$. I specifically include $n = 20$ because $\lambda = 0.95$ is a common default value for PPO. I plot the learning curves in Figure 3.4. The results are

averaged over 100 trials, with shading to indicate 95% confidence intervals. In seven out of nine of the cases, λ-returns significantly improve sample efficiency over n-step returns with respect to the 100-episode moving average of undiscounted return, again demonstrating the benefits of compound returns.

3.5 Piecewise λ-Returns

Although the previous experiment shows that λ-returns can achieve faster learning, they are expensive for large-buffer deep RL algorithms (see Chapter 5). This is because the λ-return at time t theoretically bootstraps on every time step afterwards (until the end of an episode), with each bootstrap requiring a forward pass through the neural network.

For now, I instead seek a compound return that approximates the variance-reduction property of the λ-return while being computationally efficient for minibatch replay. There are many ways I could average n-step returns together, and so I constrain my search by considering compound returns that 1) comprise an average of only two n-step returns to minimize cost; 2) preserve the contraction modulus or COM of the λ-return; and 3) place weights on the TD errors that are close to those assigned by the λ-return.

The first property constrains my estimator to have the two-bootstrap form of Eq. (3.2). Let n be the targeted effective n-step; the effective λ can be obtained from Prop. 3.12. Let me also assume $\gamma < 1$, although I include the case where $\gamma = 1$ in Appendix A.4. To preserve the contraction modulus (the second property), I must satisfy $(1-c)\gamma^{n_1} + c\gamma^{n_2} = \gamma^n$. Assuming there is freedom in the choice of n_1 and n_2, it follows that $c = \left(\gamma^n - \gamma^{n_1}\right) / \left(\gamma^{n_2} - \gamma^{n_1}\right)$.

I would thus like to find n_1 and n_2 such that the weights given to the TD errors optimize some notion of closeness to the TD(λ) weights, in order to fulfill the third and final property. Although there are many ways I could define the error, I propose to minimize the maximum absolute difference between the weights, since this ensures that no individual weight deviates too far from the TD(λ) weight. Recall that the cumulative weight given to TD error δ_{t+i} by n-step return $G_t^{(n)}$ is $h_i = 1$ if $i < n$, and is $h_i = 0$ otherwise. It follows that the two-bootstrap average assigns a weight of $h_i = 1$ if $i < n_1$; a weight of $h_i = c$ if $n_1 \leq i < n_2$ because $(1-c) \cdot 0 + c \cdot 1 = c$; and a weight of $h_i = 0$ otherwise. I then minimize the error $\max_{i \geq 0} |\gamma^i h_i - (\gamma\lambda)^i|$.

I call my approximation Piecewise λ-return (Pilar) because each weight h_i is a piecewise function whose value depends on where i lies in relation to the interval $[n_1, n_2)$. Figure 3.5 illustrates how Pilar roughly approximates the TD(λ) decay using a step-like shape. Although Pilar's TD-error weights do not form a smooth curve, they retain important properties like contraction modulus, monotonicity, and variance reduction. Crucially, Pilar is much cheaper to compute than the λ-return, making it more suitable for minibatch experience replay. In Appendix A.4, I describe a basic search algorithm for finding the lowest-error (n_1, n_2)-pair, along with a reference table of precomputed Pilars for $\gamma = 0.99$.

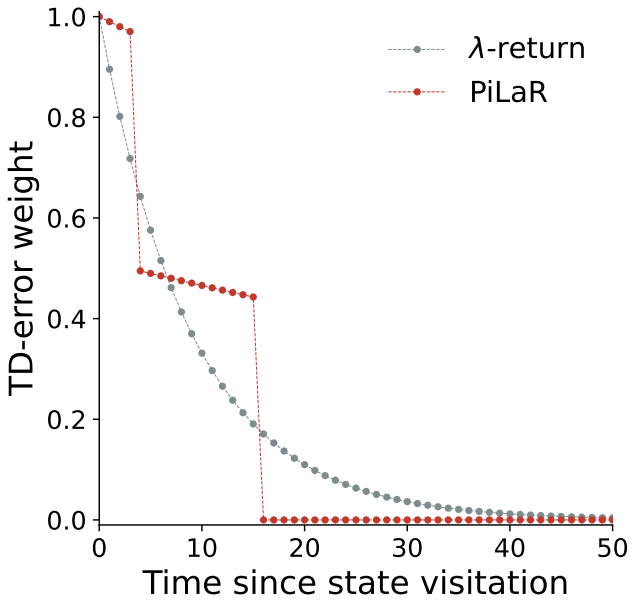


Figure 3.5: TD-error weights for a λ-return and Pilar ($\lambda = 0.904$). Both returns have the same contraction modulus as a 10-step return when $\gamma = 0.99$.

To evaluate Pilars in a challenging, high-dimensional learning environment, I compare them against n-step returns using the DQN algorithm (recall Section 2.5) in the five MinAtar games (Young and Tian, 2019): Asterix, Breakout, Freeway, Seaquest, and Space Invaders. The game states are 10×10 multi-channel binary images depicting object locations and velocity trails. A neural network processes these images to generate a vector of values corresponding to each action. The network's first layer is a 16-filter 3×3 convolutional layer, the output of which is flattened and then followed by a dense layer with 128 units. Both layers use rectified linear unit (ReLU) activations.

Table 3.2: Pilars used for MinAtar.

n-step	n_1	n_2	c
3	1	6	0.406
5	2	9	0.437

The agents are trained for 5 million time steps each. They execute a random policy for the first 5,000 time steps to prepopulate the replay buffer (capacity: 100,000 transitions), and then switch to an ϵ-greedy policy for the remainder of training, with ϵ annealed linearly from 1

to 0.1 over the next 100,000 steps. Every step, the main network is updated using a minibatch of 32 return estimates to minimize the loss in Eq. (2.10). The target network's parameters are copied from the main network every 1,000 time steps. My experiment procedure closely matches that of Young and Tian (2019); the only differences are the squared loss instead of Huber loss, the Adam optimizer (Kingma and Ba, 2015), and the use of multistep returns.

To obtain the n-step returns, the replay buffer is modified to return a minibatch of sequences of $n+1$ experiences for each return estimate (instead of the usual two experiences for DQN). The return is computed by summing the first n rewards and then adding the value-function bootstrap from the final experience, with discounting if $\gamma < 1$. If the episode terminates at any point within this trajectory, then the return is truncated and no bootstrapping is necessary, since the value of a terminal state is defined to be zero. For Pilars, the idea is the same, but the trajectories must have length n_2+1 to accommodate the lengths of both n-step returns. The two n-step returns are computed as above, and then combined by averaging them: $(1-c)\,G^{(n_1)} + c\,G^{(n_2)}$.

For $n = 3$ and $n = 5$, I compare the n-step return against the corresponding Pilar of the same contraction modulus with the given discount factor, $\gamma = 0.99$ (see Table 3.2 for specific values of n_1, n_2, and c). I choose five Adam step sizes over a logarithmic grid search and generate learning curves by plotting the 100-episode moving average of undiscounted return (game score) versus time steps. I then choose the step size for each game-estimator pair that maximizes the area under the curve over the last 1 million steps, or 20% of training. I plot the learning curves for each game in Figure 3.6. Out of the five games, Seaquest is the only one in which Pilar does not outperform n-step returns in a statistically significant way. The results are averaged over 32 trials, with shading to indicate 95% confidence intervals.

In several cases, Pilars are able to significantly improve the agent's best performance compared to n-step returns. This is in spite of the fact that each pair of n-step return and Pilar has the same contraction modulus, suggesting that the performance increase is partially due to variance reduction. Overall, Pilars outperform n-step returns for 1 in 5 games (20%) when $n = 3$, and 3 in 5 games (60%) when $n = 5$. In the remaining cases, the performance is not significantly different. Thus, Pilars improve overall performance across these games. Because the average performance gap between the estimators also widens as n increases, Pilar's benefits appears to become more pronounced for longer—and, hence,

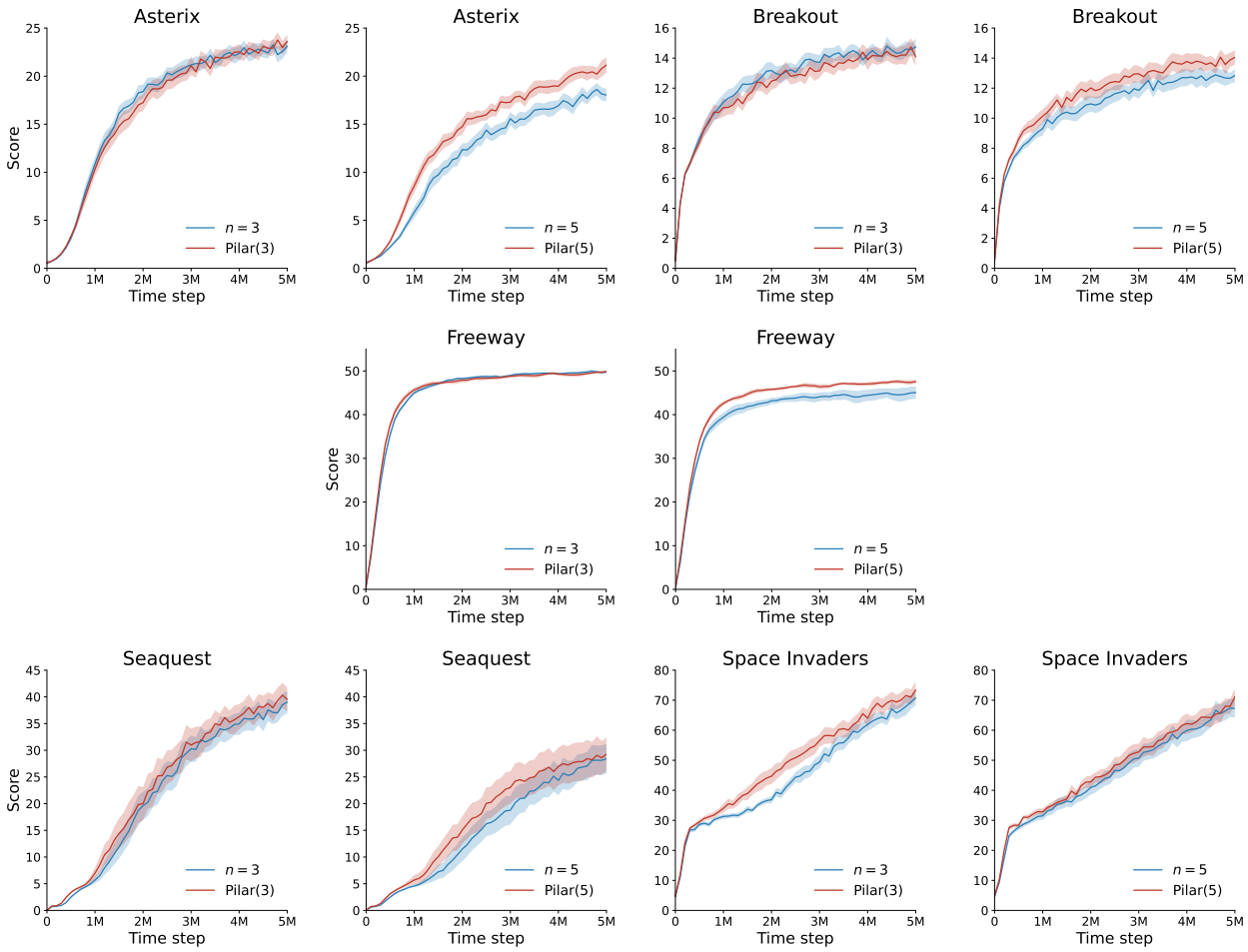


Figure 3.6: Learning curves for DQN with n-step returns and Pilars in five MinAtar games. The results are averaged over 32 trials with 95% confidence intervals shaded.

higher-variance—n-step returns.

These results corroborate my theory by showing that averaging n-step returns can accelerate learning, even with a nontrivial network architecture. I do not claim that Pilars are necessarily the best average for achieving variance reduction, but my experiments still demonstrate the practical utility of compound returns.

3.6 Discussion

I have shown that compound returns, including λ-returns, have a variance-reduction property. This is the first evidence, to my knowledge, that λ-returns have a theoretical learning advantage over n-step returns in the absence of function approximation; it was previously

believed that both were different yet equivalent ways of interpolating between TD and Monte Carlo learning. My random-walk experiments confirm that an appropriately chosen λ-return performs better than a given n-step return across a range of step sizes. In replay-based deep RL methods like DQN, where λ-returns are difficult to implement efficiently, I demonstrated with Pilar that a simpler average is still able to train neural networks with fewer samples. Since the average is formed from only two n-step returns, the additional computational cost is negligible—less expensive than adding a second target network, as is often done in recent methods (e.g., Fujimoto et al., 2018; Haarnoja et al., 2018).

Although I am able to establish strong theoretical guarantees regarding the convergence rate of multistep TD learning with linear function approximation, I note that this result does not automatically extend to nonlinear approximators. This is not a limitation of my analysis, but rather of the class of TD algorithms commonly used in deep RL. Indeed, even 1-step TD does not converge with certain nonlinear function approximators (see, e.g., Tsitsiklis and Van Roy, 1997, Sec. 10). Although my experiments have shown that compound returns often do improve performance for DQN and PPO, it is not guaranteed that these results will generalize to other methods. Additionally, I did not show that the improved performance for these algorithms is due solely to variance reduction; it is possible that other favorable properties of the compound returns also contribute to this improvement, such as convergence to a different fixed point (despite the bound that I proved in Prop. 3.11) or the tapering credit-assignment characteristic of certain compound returns (see Chapter 4). The added complexities of nonconvex optimization likely explain why the performance trends for DQN and PPO are not always as clear as my random-walk results. Nevertheless, I note that the variance-reduction property *is* a general phenomenon, since Assumptions 3.1 and 3.2 are agnostic to the choice of function approximator. This, coupled with my empirical results, indicates potential for using compound returns in other deep RL methods to improve sample efficiency.

Chapter 4

Demystifying the Recency Heuristic

The λ-return, targeted by TD(λ) and Q(λ), remains one of the most effective compound returns to date. Variations of these algorithms continue to be popular for credit assignment due to their strong empirical performance (e.g., Schulman et al., 2015b; Harb and Precup, 2016; Harutyunyan et al., 2016; Munos et al., 2016; van Seijen, 2016; Mahmood et al., 2017; Mousavi et al., 2017; Kozuno et al., 2021; Gupta et al., 2024; Tang et al., 2024). Beyond the convenient update rules of λ-returns and eligibility traces, there seems to be something particularly efficacious about smoothly fading credit assignment.

The conventional motivation behind the exponential decay of TD(λ) is that it implements the *recency heuristic*. Under the recency heuristic, an agent "assigns credit for current reinforcement to past actions according to how recently they were made" (Sutton, 1984, p. 94). The intuition is that rewards occurring sooner after an action should be reinforced more heavily, as there is presumably a stronger cause-and-effect relationship.

What makes this intuition less clear is that the recency heuristic cannot be achieved without impacting other aspects of credit assignment. Giving more weight to recent experiences means that later experiences receive comparatively less weight, potentially impacting multistep learning. The bias-variance trade-off is also affected as a result. How exactly, then, does the recency heuristic aid learning in terms of these properties?

In this chapter, I investigate the ramifications of adopting the recency heuristic in TD learning. I show that—contrary to conventional wisdom—all n-step and compound returns implement a (weak) recency heuristic: their TD-error weights are non-increasing. Violating this property with nonmonotonic weights causes divergence in simple learning problems. When TD-error weights are strictly decreasing and nonzero, as they are for TD(λ), then

credit assignment becomes smooth and asymptotically approaches zero as time elapses. I show that this smoothness is not what is beneficial on its own, but it does imply that the effective credit-assignment window extends further into the future without exacerbating worst-case variance. The cumulative effect is faster empirical learning. These findings help clarify the role of the recency heuristic and the enduring effectiveness of TD(λ).

4.1 Formalizing the Recency Heuristic

In this section, I precisely define the notion of the recency heuristic. I consider a general estimator for TD learning of the form

$$\hat{G}_t = V(S_t) + \sum_{i=0}^{\infty} \gamma^i h_i \delta_{t+i}\,, \tag{4.1}$$

where $(h_i)_{i=0}^{\infty}$ is a sequence of real numbers. Although this may appear to be restrictive, I show in Section 4.3 that it can represent every valid return target (i.e., facilitates convergence to v_π) that comprises a linear combination of future rewards and state values—thus implementable by forward-view TD prediction.

Eq. (4.1) can be thought of as an abstract form of TD(λ): one with an arbitrary stimulus-response model rather than the familiar exponential decay. At time t, the agent experiences an external stimulus modulated by the current environment state, S_t. Positive or negative reinforcement subsequently arrives in the form of TD errors $\delta_t, \delta_{t+1}, \delta_{t+2}, \ldots$. Each weight, h_i, determines the agent's receptiveness, or eligibility, to learn from the TD error that occurs exactly i steps after the initial stimulus. In this view, a return estimate, $\hat{G}_t$, is uniquely determined by the impulse response of a linear time-invariant system encoded by $(h_i)_{i=0}^{\infty}$. One possible interpretation of the recency heuristic, then, is a constraint on the impulse response such that it never increases after the initial stimulus. This gives the following definition.

Definition 4.1 (Weak Recency Heuristic)**.** *A return estimate satisfies the weak recency heuristic if and only if it has the form of Eq.* (4.1)*, and* $h_i \geq h_{i+1} \geq 0$ *holds for all* $i \geq 0$*.*

I show in Section 4.3 that this definition is closely related to the question of whether (and how fast) TD learning using this estimator converges in expectation, but it is slightly weaker than what is typically thought of as the recency heuristic. For instance, Sutton (1984, p. 94) is explicit that "Credit assigned should be a monotonically decreasing function of the time

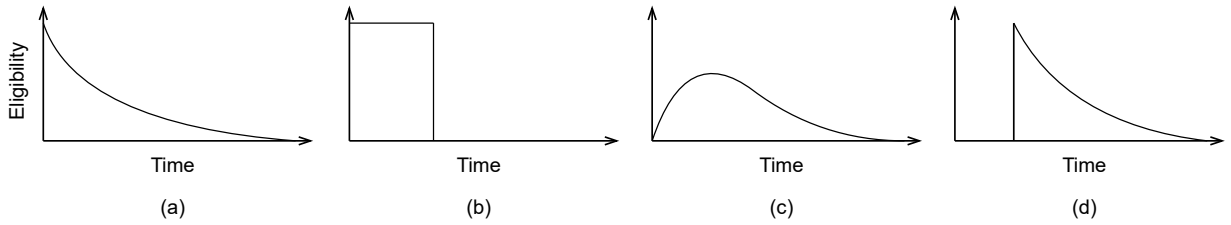


Figure 4.1: Eligibility curve illustrations for (a) λ-return, (b) n-step return, (c) inverted-U assignment inspired by Klopf (1972), and (d) time-delayed λ-return. The horizontal axis represents elapsed time since the stimulus. Neither (c) nor (d) satisfies the recency heuristic.

between action and reinforcement, approaching zero as this time approaches infinity." The credit-assignment function in Definition 4.1 is merely nonincreasing, and so I refer to it as the *weak* recency heuristic. Alternatively, I refer to the monotonically decreasing case as the *strong* recency heuristic, defined below.

Definition 4.2 (Strong Recency Heuristic)**.** *A return estimate satisfies the strong recency heuristic if and only if it has the form of Eq.* (4.1)*, and* $h_i > h_{i+1} > 0$ *holds for all* $i \geq 0$*.*

Notice that Definition 4.2 implies Definition 4.1. I make the distinction between these definitions more concrete with a few examples, which are graphed in Figure 4.1. The λ-return, used by TD(λ), is the canonical example of the strong recency heuristic; its eligibility weights in Eq. (2.7) are strictly decreasing with any $\lambda \in (0, 1)$. In contrast, the n-step return remains equally receptive to the first n TD errors, and then abruptly stops responding to the ones afterwards. However, these two updates are alike in that the weights never increase at any point: they both satisfy the weak recency heuristic. One could also imagine arbitrary weights in Eq. (4.1) that do not satisfy either definition of recency heuristic. For example, the inverted-U shape described by Klopf (1972) takes time to reach its peak value before falling back to zero, and thus violates Definitions 4.1 and 4.2. Similarly, I can take the standard spike-and-decay model of a λ-return and introduce a delay between the initial stimulus and the response. Both of these could exploit some known structure regarding the agent's environment, and may be more biologically plausible, but their mathematical implications are not yet known. Notably, there are many more possibilities in Eq. (4.1), most of which have not yet been explored.

4.2 What Happens When the Recency Heuristic Is Violated?

I conduct an experiment to demonstrate that on-policy TD learning with a tabular value function can diverge when the recency heuristic is violated. This is surprising, since one view of the TD-error weights, $(h_i)_{i=0}^{\infty}$, is that they encode a belief over the time when rewards will arrive following a stimulus. Ideally, these weights could represent any shape for the credit-assignment function and the agent would still learn the correct value function, yet this does not appear to be the case.

I test perhaps the simplest possible example of non-recent credit assignment: an update based on a single, future TD error. More specifically, I generalize TD(0) by introducing a delay of $\tau \geq 0$:

$$V(S_t) \leftarrow V(S_t) + \alpha_t \gamma^{\tau} \delta_{t+\tau} \,. \tag{4.2}$$

The impulse response for this method is generally given by $h_i = \mathbf{1}_{\{i=\tau\}}$. That is, the eligibility curve is a square pulse initiated exactly τ steps after the initial stimulus (see Figure 4.2, center). The operator corresponding to this update is $H \colon \boldsymbol{v} \mapsto \boldsymbol{v} + (\gamma \boldsymbol{E}_{\pi} \boldsymbol{P})^{\tau} (T_{\pi} \boldsymbol{v} - \boldsymbol{v})$. The fixed point of this operator is v_{π} for any value of τ because $T_{\pi} \boldsymbol{v}_{\pi} - \boldsymbol{v}_{\pi} = 0$.

Notice that this is a rather benign form of non-recent credit assignment; I have taken the simplest TD method and merely translated its impulse response along the time axis. More complex forms of non-recent credit assignment would consist of a superposition of multiple such updates, and so this example provides insight into other methods. Nevertheless, despite the simplicity of this method, I present a simple MRP that causes almost every value-function initialization to diverge away from v_{π}.

Counterexample 4.3. *Consider a 2-state MRP with reward $r(s, s') = 0$, $\forall\ s, s' \in \{s_1, s_2\}$. Let $p \in [0, 1]$ be the self-transition probability (see Figure 4.2, left) and let V_0 be the initial value function. If $\tau = 1$, $\gamma = 0.9$, and $p = 0.4$, then the TD update in Eq. (4.2) diverges whenever $V_0(s_1) \neq V_0(s_2)$.*

I give specific values of τ, γ, and p for the sake of the counterexample; however, it appears that divergence is inevitable for any $\tau > 0$ as $\gamma \to 1$ and $p \to 0$. The divergent behavior of the method is visualized in Figure 4.2 (right), where the arrows represent unit

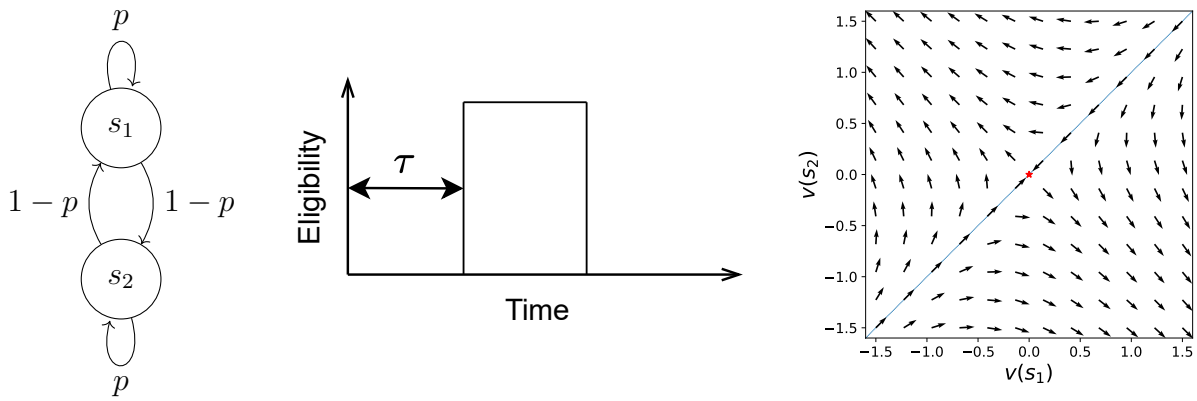


Figure 4.2: (Left) MRP for Counterexample 4.3. (Center) Credit-assignment function for delayed TD(0). (Right) Expected update directions of Eq. (4.2) for $\tau = 1$, $\gamma = 0.9$, $p = 0.4$.

vectors pointing in the direction the expected update (i.e., $H\boldsymbol{v} - \boldsymbol{v}$). Because the reward is zero for all transitions, v_π is the origin (red star) regardless of γ and p. However, every value-function initialization not on the blue line where $v(s_1) = v(s_2)$ progresses arbitrarily far away from the fixed point, v_π.

Why does violating the recency heuristic in this easy problem cause divergence? The reason becomes more clear when observing that $\gamma^\tau \delta_{t+\tau} = G_t^{(\tau+1)} - G_t^{(\tau)}$. Thus, an equivalent operator for Eq. (4.2) is $\boldsymbol{v} \mapsto \boldsymbol{v} + T_\pi^{\tau+1}\boldsymbol{v} - T_\pi^\tau \boldsymbol{v}$, whose worst-case contraction modulus is $1+\gamma^{\tau+1}+\gamma^\tau$ by the triangle inequality—greater than 1. Although this does not automatically mean the operator will diverge, it does suggest that divergence is possible, and here is one instance of it. It is important to note that this divergence is not due to sampling noise nor an uneven state distribution, as I am explicitly computing the expected result of the operator in both states. Furthermore, the phenomenon is not unique to this particular algorithm or problem, but generally arises whenever the weak recency heuristic is violated by too much. I prove this formally in the next section.

4.3 Only Convex Returns Satisfy the Weak Recency Heuristic

I introduce the notion of a *convex* return: a convex combination of n-step returns. Its formula is identical to that of a compound return, i.e.,

$$G_t^{\boldsymbol{c}} \coloneqq \sum_{n=1}^{\infty} c_n G_t^n \,, \tag{2.9}$$

but it relaxes the restriction that at least two weights must be nonzero. Thus, a convex return is either a compound return or an n-step return. In this section, I show that this definition is logically equivalent to the weak recency heuristic; Definition 4.1 is satisfied if and only if a return estimate is convex (see Prop. 4.6).

To illuminate the role of the weak recency heuristic, I first justify the general return estimator in Eq. (4.1). In particular, I show that estimates of this form correspond to the largest set of linear operators suitable for TD prediction. This allows me to later analyze how the properties of these operators are affected by the choice of the weights, $(h_i)_{i=0}^{\infty}$, especially when these weights do not satisfy the recency heuristic.

To produce a TD method in the form of Eq. (2.1) that converges to v_π under general conditions, the return estimate $\hat{G}_t$ must correspond to a maximum-norm contraction mapping, H, with its unique fixed point at v_π. In addition to these requirements, I want a *sample-realizable* operator in order to create an implementable TD method: one that can be constructed from any rewards or state values following time t. To match existing TD methods for prediction, I assume that this operator is linear with respect to these quantities, giving the following definition.

Definition 4.4. *A sample-realizable linear operator has the form*

$$H\boldsymbol{v} = \sum_{i=0}^{\infty} (\gamma \boldsymbol{E}_\pi \boldsymbol{P})^i (h_i \boldsymbol{E}_\pi \boldsymbol{r} + x_i \boldsymbol{v}) \,,$$

where $(h_i)_{i=0}^{\infty}$ and $(x_i)_{i=0}^{\infty}$ are bounded sequences of real numbers.

This definition covers all possible operators based on return estimates constructible from a linear combination of sampled experiences: i.e.,

$$\hat{G}_t = \sum_{i=0}^{\infty} \gamma^i h_i R_{t+1+i} + \gamma^i x_i V(S_{t+i}) \,. \tag{4.3}$$

However, the vast majority of these operators will not meet the necessary convergence criteria. In the following proposition, I reduce the space of operators by identifying only those whose fixed point is exactly v_π.

Proposition 4.5. *For every sample-realizable operator H whose fixed point is v_π, there exists a sequence of real numbers $(h_i)_{i=0}^{\infty}$ such that*

$$H\boldsymbol{v} = \boldsymbol{v} + \sum_{i=0}^{\infty} (\gamma \boldsymbol{E}_\pi \boldsymbol{P})^i h_i (T_\pi \boldsymbol{v} - \boldsymbol{v}) \,. \tag{4.4}$$

Letting $c_n \coloneqq h_{n-1} - h_n$ for $n \geq 1$, then H also has the equivalent form

$$H\boldsymbol{v} = \left(1 - \sum_{n=1}^{\infty} c_n\right)\boldsymbol{v} + \sum_{n=1}^{\infty} c_n T_\pi^n \boldsymbol{v}\,. \tag{4.5}$$

Proof. It is given that H is sample realizable. Eq. (4.3) provides a sample estimate of H, which I rewrite in terms of TD errors. First note that $R_{t+1+i} = \delta_{t+i} + V(S_{t+i}) - \gamma\, V(S_{t+1+i})$, and therefore

$$h_i R_{t+1+i} + x_i\, V(S_{t+i}) = h_i \delta_{t+i} + (h_i + x_i)\, V(S_{t+i}) - \gamma h_i\, V(S_{t+1+i})\,.$$

Hence, Eq. (4.3) is equal to

$$V(S_t) + \left(\sum_{i=0}^{\infty} \gamma^i h_i \delta_{t+i}\right) + (h_0 - 1 + x_0)\, V(S_t) + \left(\sum_{i=1}^{\infty} (h_i - h_{i-1} + x_i)\, V(S_{t+i})\right)\,.$$

Note that, since the leading term is $V(S_t)$, then the fixed point of this estimate is equal to v_π whenever the remaining three terms are equal to zero in expectation. The first sum is zero in expectation because v_π implies that $\mathbb{E}_\pi[\delta_{t+i} \mid S_t] = 0$ for all $i \geq 0$. The last two terms are zero only when $x_0 = 1 - h_0$ and $x_i = h_{i-1} - h_i$ for all $i \geq 1$. In other words, $(x_i)_{i=0}^{\infty}$ must be the *unique* first-order difference sequence of $(h_i)_{i=0}^{\infty}$ (with $h_{-1} = 1$); no other choice of weights would make v_π a fixed point, since at least one of the remaining terms would not vanish. I substitute this choice of weights into the previous expression, which yields the same general TD estimate introduced in Section 4.1:

$$\hat{G}_t = V(S_t) + \sum_{i=0}^{\infty} \gamma^i h_i \delta_{t+i}\,. \tag{4.1}$$

Thus, Eq. (4.1) and Eq. (4.3) are equivalent if and only if V is v_π. The operator corresponding to this sample estimate is exactly Eq. (4.4), which establishes the first part of the theorem.

To derive Eq. (4.5), I apply the fact that $h_i = \sum_{n=i+1}^{\infty} c_n$ due to the telescoping series that results from the definition of $c_n = h_{n-1} - h_n$. I complete the proof by rewriting Eq. (4.4) as

$$\begin{aligned}
H\boldsymbol{v} &= \boldsymbol{v} + \sum_{i=0}^{\infty} \left(\sum_{n=i+1}^{\infty} c_n\right) (\gamma \boldsymbol{E}_\pi \boldsymbol{P})^i (T_\pi \boldsymbol{v} - \boldsymbol{v}) \\
&= \boldsymbol{v} + \sum_{n=1}^{\infty} c_n \sum_{i=0}^{n-1} (\gamma \boldsymbol{E}_\pi \boldsymbol{P})^i (T_\pi \boldsymbol{v} - \boldsymbol{v}) \\
&= \boldsymbol{v} + \sum_{n=1}^{\infty} c_n (T_\pi^n \boldsymbol{v} - \boldsymbol{v})
\end{aligned}$$

$$= \left(1 - \sum_{n=1}^{\infty} c_n\right) \boldsymbol{v} + \sum_{n=1}^{\infty} c_n T_\pi^n \boldsymbol{v}\,.$$

The second equality interchanged the sums using the rule $\sum_{i=0}^{\infty}\sum_{n=i+1}^{\infty} = \sum_{n=1}^{\infty}\sum_{i=0}^{n-1}$. The third equality followed from the n-step Bellman operator expansion: $T_\pi^n \boldsymbol{v} = \boldsymbol{v} + \sum_{i=0}^{n-1}(\gamma \boldsymbol{E}_\pi \boldsymbol{P})^i (T_\pi \boldsymbol{v} - \boldsymbol{v})$. □

The operator in Eq. (4.4) corresponds exactly to the sample estimate in Eq. (4.1) that I considered in Section 4.1 when defining the weak recency heuristic. I refer to these as *linear* returns. Hence, every linear return with v_π as its fixed point is expressible as a weighted sum of either TD errors or n-step returns, without loss of generality.[17]

I now have a generic operator that is both sample realizable and has the correct fixed point, but it is not necessarily a contraction mapping without any conditions on $(h_i)_{i=0}^{\infty}$. Eq. (4.5) expresses the operator in terms of the n-step Bellman operators, facilitating the analysis of its contraction properties. Because $\boldsymbol{E}_\pi \boldsymbol{P}$ is a stochastic matrix, I have $\|\boldsymbol{E}_\pi \boldsymbol{P}\|_\infty = 1$, which also implies that $\|T_\pi^n \boldsymbol{v} - T_\pi^n \boldsymbol{v}'\|_\infty \leq \gamma^n \|\boldsymbol{v} - \boldsymbol{v}'\|_\infty$ for any $\boldsymbol{v}, \boldsymbol{v}' \in \mathbb{R}^{|\mathcal{S}|}$. Thus, by the triangle inequality,

$$\|H\boldsymbol{v} - H\boldsymbol{v}'\|_\infty \leq \left(\left|1 - \sum_{n=1}^{\infty} c_n\right| + \sum_{n=1}^{\infty} |c_n| \gamma^n\right) \|\boldsymbol{v} - \boldsymbol{v}'\|_\infty\,, \tag{4.6}$$

and the contraction modulus is therefore $\beta = |1 - \sum_{n=1}^{\infty} c_n| + \sum_{n=1}^{\infty} |c_n| \gamma^n$. The operator is a contraction mapping if and only if $\beta < 1$.

Eq. (4.5) consists of two terms: the original value function scaled by $1 - \sum_{n=1}^{\infty} c_n$, and a linear combination of n-step returns. The first term can be eliminated without loss of generality by normalizing the sum of weights, i.e., by adding the constraint that $\sum_{n=1}^{\infty} c_n = 1$. This is because the first term changes only the magnitude of the update, which can be absorbed into the step size, α_t, in the value backup of Eq. (2.1). With this constraint in place, it follows that the weight of the first TD error is $h_0 = 1$ because of the telescoping series: $h_0 = \sum_{n=1}^{\infty} h_{n-1} - h_n = \sum_{n=1}^{\infty} c_n = 1$. The operator is now an affine combination of n-step Bellman operators, and so I refer to such return estimates as *affine* returns. Note that, since I have $h_0 = 0$ in Eq. (4.2) when $\tau > 0$, the divergent return estimate in Counterexample 4.3

[17] This is perfectly analogous to Lemma 3.5 regarding the decomposition of compound returns into weighted TD errors, which also sets $h_i = \sum_{n=i+1}^{\infty} c_n$. This is because linear returns strictly generalize compound returns. However, here I have derived it from first principles, without assuming any relation to n-step returns.

Table 4.1: Summary of operators and sample estimates for the returns in Figure 4.3.

Name	Operator	Sample Estimate	Conditions
Linear	$\left(1-\sum_{n=1}^{\infty} c_n\right)\boldsymbol{v} + \sum_{n=1}^{\infty} c_n T_\pi^n \boldsymbol{v}$	$\left(1-\sum_{n=1}^{\infty} c_n\right)V(S_t) + \sum_{n=1}^{\infty} c_n G_t^{(n)}$	None
Affine	$\sum_{n=1}^{\infty} c_n T_\pi^n \boldsymbol{v}$	$\sum_{n=1}^{\infty} c_n G_t^{(n)}$	$\sum_{n=1}^{\infty} c_n = 1$ and $\sum_{n=1}^{\infty} \lvert c_n\rvert\gamma^n < 1$
Convex	$\sum_{n=1}^{\infty} c_n T_\pi^n \boldsymbol{v}$	$\sum_{n=1}^{\infty} c_n G_t^{(n)}$	Affine and $c_n \geq 0,\ \forall\ n \geq 1$
Compound	$\sum_{n=1}^{\infty} c_n T_\pi^n \boldsymbol{v}$	$\sum_{n=1}^{\infty} c_n G_t^{(n)}$	Convex and $\exists\ c_i, c_j > 0$
n-step	$T_\pi^n v$	$G_t^{(n)}$	$n \geq 1$

is *not* an affine return—although it is linear. Affine returns look identical to convex returns from Eq. (2.9), but they are more general because they allow for negatively weighted n-step returns. I depict the hierarchical relationship between linear, affine, convex, compound, and n-step returns in Figure 4.3, and summarize their operators and corresponding sample estimates in Table 4.1.

This analysis provides a hint of why counterexamples like the one in Section 4.2 are possible; negative weights increase the contraction modulus due to the absolute value in Eq. (4.6). It turns out that such negative weights coincide exactly with the time steps on which the weak recency heuristic is violated, and therefore only convex returns satisfy the heuristic, as I show in the next proposition.

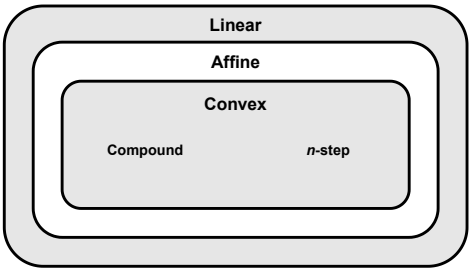


Figure 4.3: Hierarchical relationship between different return estimators. A return satisfies the weak recency heuristic if and only if it is a convex return: i.e., a compound return or n-step return.

Proposition 4.6. *An affine return satisfies the weak recency heuristic if and only if it is a convex return (i.e., a compound return or an n-step return).*

Proof. Recall that $c_n = h_{n-1} - h_n$. Therefore, the affine operator from Eq. (4.5) is equal to

$$H\boldsymbol{v} = \sum_{n=1}^{\infty} (h_{n-1} - h_n)\, T_\pi^n \boldsymbol{v}\,. \tag{4.7}$$

If the weak recency heuristic (Definition 4.1) holds, then $h_{n-1} \geq h_n \implies h_{n-1} - h_n \geq 0$, for all $n \geq 1$. Thus, Eq. (4.7) is a convex combination of n-step returns, because I have $\sum_{n=1}^{\infty} h_{n-1} - h_n = \sum_{n=1}^{\infty} c_n = 1$ for an affine return.

To complete the proof, I also show the contrapositive. Consider an affine return that is not a convex combination of n-step returns. Consequently, it must have at least one negatively weighted n-step return: there exists some $k \geq 1$ such that $c_k < 0$. However, this implies that $h_{k-1} - h_k < 0$, and therefore $h_{k-1} < h_k$, so the weak recency heuristic is violated. I conclude that an affine return satisfies the weak recency heuristic if and only if it is a convex return. □

An immediate corollary of the above is that the weak recency heuristic is a sufficient condition for convergence, since both compound returns and n-step returns are already known to correspond to contraction mappings (Watkins, 1989, Sec. 7.2). This stems from the fact that a convex combination of n-step returns, each of which is contractive with modulus γ^n, must also be contractive: i.e., $\sum_{n=1}^{\infty} c_n \gamma^n \leq \gamma$ for every choice of nonnegative weights that sum to 1. In this view, the weak recency heuristic can be seen as a convergence test for TD learning: divergence is impossible under this heuristic (for on-policy tabular methods).

On the other hand, violating the weak recency heuristic increases the contraction modulus of the return estimator, with divergence possible if the violation becomes too extreme (e.g., Counterexample 4.3). This is because any time an n-step return has a negative weight, another n-step return must have a larger positive weight to counterbalance it and ensure the weights sum to 1 overall. This necessarily increases the contraction modulus in Eq. (4.6) due to the absolute value, underscoring yet another benefit of the weak recency heuristic. A convex return is not only guaranteed to converge regardless of its weights, but also has a faster contraction than a nonconvex (affine) return constructed from the same n-step returns.

Finally, I note that these results extend to function approximation as well. In the proof of Prop. 3.11 in Appendix A.3.4, I discuss how a return estimate, $\hat{G}_t$, under *linear* function approximation corresponds to a composite operator $\mathbf{\Pi} \circ H$, where $\mathbf{\Pi}$ is the linear projection operator defined in Eq. (2.3). Furthermore, $\mathbf{\Pi}$ is nonexpansive (Tsitsiklis and Van Roy, 1997, proof of Lemma 6); hence, if H is not a contraction mapping, then $\mathbf{\Pi} \circ H$ may also fail to be one. This implies that violating the weak recency heuristic by too much can still increase the contraction modulus and cause divergence, just like in Counterexample 4.3. In the case

of *nonlinear* function approximation, the existence of counterexamples is certain, as even 1-step TD can diverge with some approximators (see Tsitsiklis and Van Roy, 1997, Fig. 1).

4.4 Are Monotonically Decreasing TD-Error Weights Necessary?

So far, I have focused on the weak recency heuristic: when the eligibility weights are nonincreasing. However, as I discussed in Section 4.1, the connotation of the recency heuristic is often that of strictly decreasing TD-error weights, i.e., the strong recency heuristic (Definition 4.2). This is why, for example, λ-returns are more strongly associated with a recency heuristic than n-step returns are. Does this distinction between weak and strong recency heuristics matter in practice? I conduct experiments indicating that the answer is yes, but in a surprising way; the smoothness of the weights do not appear to be significant, but the strong recency heuristic does imply that the return estimate consists of infinitely many n-step returns, which empirically improves credit assignment.

To test the question of whether the smoothness of the TD-error weights matters, I introduce the *sparse* λ-return, defined as

$$G_t^{\lambda,m} \coloneqq (1-\lambda)\sum_{k=1}^{\infty}\lambda^{k-1}G_t^{(m(k-1)+1)} = V(S_t) + \sum_{i=0}^{\infty}\gamma^i\lambda^{\lfloor\frac{i+m-1}{m}\rfloor}\delta_{t+i}\,,$$

where $m \geq 1$. The contraction modulus of this return is $\beta = \gamma(1-\lambda)\,/\,(1-\gamma^m\lambda)$. When $m = 1$, I simply recover the standard exponential decay of the λ-return from Eq. (2.8). However, for $m > 1$, the TD-error weights no longer satisfy the strong recency heuristic as they become more stepwise (see Figure 4.4). This implies that every $m-1$ out of m n-step returns have zero weight. For example, setting $m = 2$ generates the TD-error weight sequence $1,\ \lambda,\ \lambda,\ \lambda^2,\ \lambda^2,\ \ldots$, which produces an exponential average of the odd n-step returns: $G_t^{(1)},\ G_t^{(3)},\ G_t^{(5)},\ G_t^{(7)},\ G_t^{(9)},\ \ldots$. The reason I choose this form is because it isolates the effects of the two recency heuristics by keeping the type of weighted average consistent (i.e., exponential). If monotonicity is beneficial to learning, then I would expect to observe a performance degradation for sparse λ-returns ($m > 1$) compared to dense ($m = 1$).

My experiment setup is a discounted variation ($\gamma = 0.99$) of the 19-state random walk from Section 3.4. I test three different degrees of sparsity for the λ-returns, adjusting λ

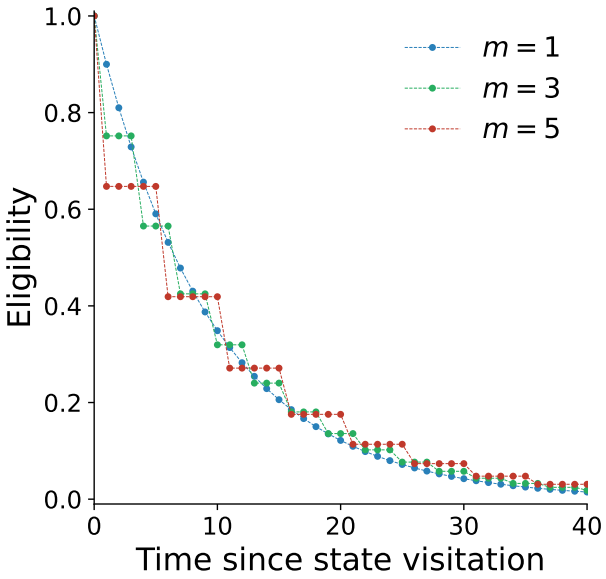


Figure 4.4: Impulse responses of λ-returns with varying degrees of sparsity.

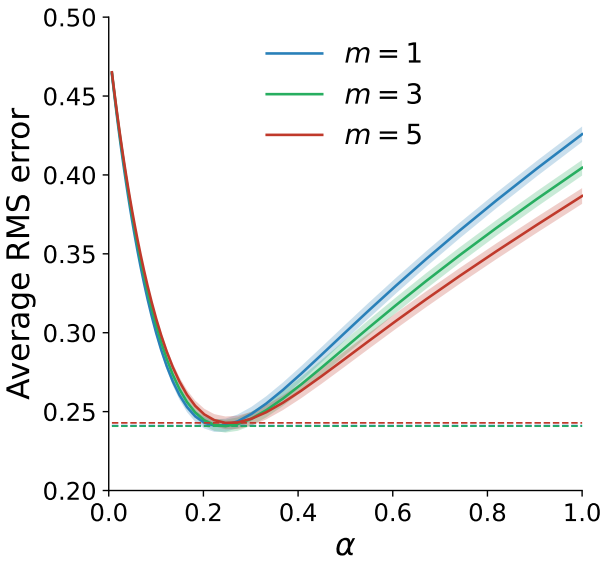


Figure 4.5: Random-walk performance of λ-returns with varying degrees of sparsity.

for each return to maintain the same contraction modulus in all cases: $(\lambda, m) \in \{(0.9, 1), (0.75, 3), (0.65, 5)\}$. The agents are trained for 10 episodes by applying offline value backups of the form Eq. (2.1) to every experience at the end of each episode. In Figure 4.5, I plot the RMS error, $\|\boldsymbol{V} - \boldsymbol{v}_\pi\|_2$, averaged over the 10 episodes, versus the step size, α, for each return. The final results are averaged over 400 trials with 95% confidence intervals indicated by shaded regions.

Because the three returns all have the same contraction modulus (i.e., expected convergence rate), their performance is nearly the same for small values of α, which are able to average out the noise in the updates. Likewise, the returns share the same lowest error, as indicated by the dashed horizontal lines in Figure 4.5. However, as α gets larger, their performances begin to separate, achieving lower average error as the sparsity of the λ-return increases. Thus, even though the eligibility curves become more step-like as the sparsity is increased and they violate the strong recency heuristic, the overall performance of the return improves. This demonstrates that the monotonicity of the eligibility curves does not directly factor into the performance of the return estimators.

The main reason for the sparse λ-return's improvement appears to be that its eligibility initially decays faster than that of the dense λ-return, but then slower as time goes on (see

Figure 4.4 again). This gives the eligibility curve a long-tailed[18] characteristic which, in turn, propagates credit back faster. In fact, every return that satisfies the strong recency heuristic must have a similar characteristic, because Definition 4.2 implies that $c_n = h_{n-1} - h_n > 0$ for all $n \geq 1$, and thus Eq. (2.9) must correspond to a positively weighted average of infinitely many n-step returns. Although this property is not unique to the strong recency heuristic (e.g., the sparse λ-return has it but does not satisfy Definition 4.2), it does suggest a practical significance for this heuristic: it implies a long horizon for credit assignment.

However, any benefit of a longer credit-assignment horizon is contingent on controlling the variance of the return. Fortunately, as I show in the following proposition, a long-tailed eligibility curve does not increase the worst-case variance when the contraction modulus is held constant.

Proposition 4.7. *Let* $\kappa_t \coloneqq \max_{i,j \geq 0} \mathrm{Cov}[\delta_{t+i}, \delta_{t+j} \mid S_t]$. *The worst-case conditional variance of any convex return* $G_t^{\mathbf{c}}$ *with contraction modulus* β *has the bound*

$$\mathrm{Var}[G_t^{\mathbf{c}} \mid S_t] \leq \left(\frac{1-\beta}{1-\gamma}\right)^2 \kappa_t \,. \tag{4.8}$$

Proof. Let $\Gamma(n) \coloneqq (1-\gamma^n) \,/\, (1-\gamma)$ be the n-th partial sum of the geometric series. Using Eq. (2.6) and Eq. (3.1), I derive an upper bound on the covariance between two n-step returns with lengths n_1 and n_2:

$$\begin{aligned}
\mathrm{Cov}[G_t^{(n_1)}, G_t^{(n_2)} \mid S_t] &= \mathrm{Cov}\left[\sum_{i=0}^{n_1-1} \gamma^i \delta_{t+i}, \sum_{j=0}^{n_2-1} \gamma^j \delta_{t+j} \;\middle|\; S_t\right] \\
&= \sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} \gamma^{i+j} \mathrm{Cov}[\delta_{t+i}, \delta_{t+j} \mid S_t] \\
&\leq \sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} \gamma^{i+j} \kappa_t \\
&= \Gamma(n_1)\, \Gamma(n_2)\, \kappa_t \,.
\end{aligned}$$

Because $\sum_{n=1}^{\infty} c_n = 1$ and $\beta = \sum_{n=1}^{\infty} c_n \gamma^n$ for a convex return, I also have

$$\sum_{n=1}^{\infty} c_n\, \Gamma(n) = \sum_{n=1}^{\infty} c_n \left(\frac{1-\gamma^n}{1-\gamma}\right) = \frac{1 - \sum_{n=1}^{\infty} c_n \gamma^n}{1-\gamma} = \frac{1-\beta}{1-\gamma} \,.$$

[18] In this context, I use *long-tailed* informally. For example, I consider exponential functions to have a long tail for credit assignment, unlike the stricter usage of the term in statistics.

Therefore, I derive the following upper bound on the variance of a convex return:

$$\begin{aligned}
\mathrm{Var}[G_t^{\mathbf{c}} \mid S_t] &= \sum_{i=1}^{\infty}\sum_{j=1}^{\infty} \mathrm{Cov}[c_i G_t^{(i)}, c_j G_t^{(j)} \mid S_t] \\
&= \sum_{i=1}^{\infty}\sum_{j=1}^{\infty} c_i c_j \mathrm{Cov}[G_t^{(i)}, G_t^{(j)} \mid S_t] \\
&\leq \sum_{i=1}^{\infty}\sum_{j=1}^{\infty} c_i c_j \, \Gamma(i)\, \Gamma(j)\, \kappa_t \\
&= \left(\frac{1-\beta}{1-\gamma}\right)^2 \kappa_t \,,
\end{aligned}$$

which completes the proof. □

This bound is rather loose, but it is general. Eq. (4.8) implies that averages of n-step returns always have finite variance, even as the n-step returns become arbitrarily long. Furthermore, this upper bound depends only on the contraction modulus of the return itself and not the chosen weights for the average. Since the contraction modulus is proportional to the worst-case bias of the return by Eq. (4.6), both the worst-case bias and worst-case variance of the λ-returns in my previous experiment remain the same regardless of sparsity. Thus, compound returns with a long-tailed eligibility curve are able to assign credit more quickly without negatively impacting the bias-variance trade-off (at least, in a worst-case sense).

To test the effect of a longer credit-assignment horizon under a controlled contraction modulus, I repeat the previous experiment but with *truncated* λ-returns (Cichosz, 1994):

$$G_{t:t+L}^{\lambda} \coloneqq V(S_t) + \sum_{i=0}^{L-1} (\gamma\lambda)^i \delta_{t+i} \tag{4.9}$$

$$= (1-\lambda) \sum_{n=1}^{L-1} \lambda^{n-1} G_t^{(n)} + \lambda^{L-1} G_t^{(L)} \,, \tag{4.10}$$

where $L \geq 1$ is the truncation length. The contraction modulus of this return is $\beta = \big((1-\gamma)(\gamma\lambda)^L + \gamma(1-\lambda)\big) \,/\, (1-\gamma\lambda)$. The eligibility curve for this return is a monotonically decreasing function, up until time L when it abruptly falls to zero (see Figure 4.6). As $L \to \infty$, I recover the true λ-return, Eq. (2.8). I test three variants of this return: $(\lambda, L) \in \{(0.99, 10), (0.93, 20), (0.9, \infty)\}$. As before, all of these values are chosen to produce approximately the same contraction modulus. I plot the average RMS error in Figure 4.7, again averaged over 400 trials with 95% confidence intervals shaded. The performance is

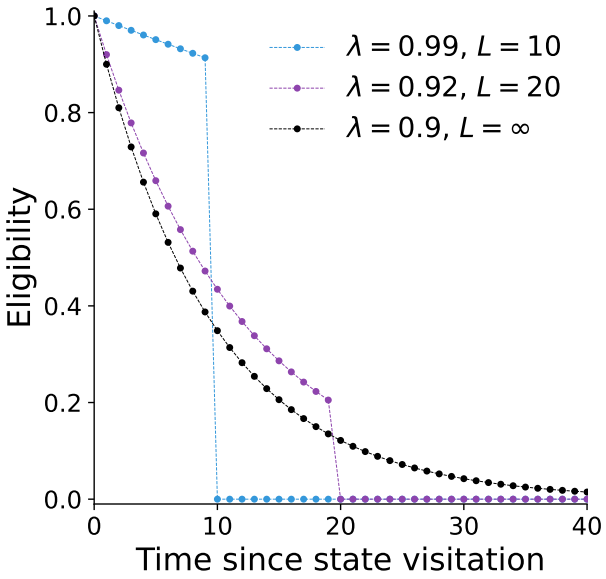


Figure 4.6: Eligibility curves of λ-returns with varying degrees of truncation.

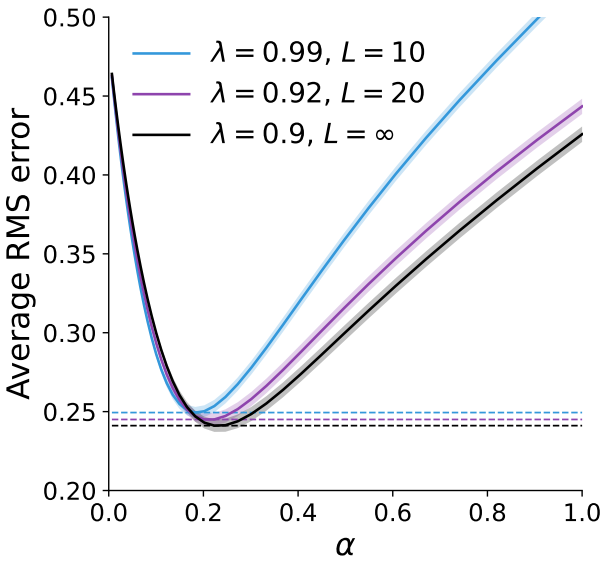


Figure 4.7: Random-walk performance of λ-returns with varying degrees of truncation.

roughly identical when α is small, since the same contraction modulus guarantees the same expected performance. However, as α gets larger, the truncated returns perform poorly compared to the full λ-return. This suggests that the performance of the returns is strongly tied to having longer n-step returns in the average, but only when the contraction moduli are equalized. This also supports my earlier hypothesis that the results observed with the sparse λ-returns in Figure 4.5 are due to their long-tail eligibility curves and not some other property such as monotonicity.

To summarize, satisfying the strong recency heuristic creates a compound return comprising infinitely many n-step returns—a long-tailed eligibility curve. This improves the effective window of credit assignment without exacerbating variance (in a conservative sense), as long as the contraction modulus is held constant. However, this property is not unique to the strong recency heuristic; for instance, sparse λ-returns violate this heuristic, but are still averages of infinitely many n-step returns, and outperform dense λ-returns in Figure 4.5. These insights help explain why smooth averages like the λ-return are often effective in practice, even if not strictly necessary for good performance.

4.5 Discussion

Although non-recent credit assignment should theoretically be possible and useful in certain learning environments, it does not seem readily compatible with our current formulation of TD learning. In particular, violating the recency heuristic manifests as negative weights on some of the n-step components of the return target. These negative weights appear to counteract learning by increasing the contraction modulus, without offering a clear benefit to learning, and potentially culminating in divergence as demonstrated by Counterexample 4.3. The fact that divergence is possible in such a favorable setting—a tabular MRP with fully observable states—points to the severity of this issue. Successfully implementing new forms of credit assignment that do not strictly follow the recency heuristic will likely require rethinking how we formulate the reinforcement signal in computational RL. My theory will provide a good starting point for algorithmic development in this direction.

Another major finding is that the recency heuristic is not merely a simple protocol for addressing the temporal credit-assignment problem, but also has intrinsic importance for learning value functions. The existence of diverging counterexamples illuminates the critical role of nonincreasing weights on the TD errors—the weak recency heuristic. The logical equivalence between this heuristic and the return estimate's ability to be expressed as a convex combination of n-step returns unifies two fundamental yet seemingly disparate ideas. More specifically, convex returns were the most general return estimates for TD learning identified before my work, and so it is surprising to find they coincide exactly with another foundational concept in RL: the recency heuristic. This appears to be a novel, unifying perspective between the forward and backward views of TD learning with arbitrary returns.

Finally, my results help to further explain the strong empirical performance and continued popularity of TD(λ), along with its many variants, for nearly four decades. My experiments suggest that the smoothness of TD(λ)'s exponential decay is not directly responsible for this success; rather, all compound returns (including λ-returns) that average an infinite number of n-step returns are able to distribute credit over a longer period without exacerbating the maximum bias or variance. These results confirm the intuition that "the fading strategy [of TD(λ)] is often the best [versus n-step TD methods]" (Sutton and Barto, 2018, p. 304), though non-exponential fading strategies are also viable.

Chapter 5

λ-Returns with Experience Replay

Beyond bias-variance properties, the amount of bootstrapping performed by deep RL is an important computational consideration. Each value estimate requires a forward pass through a neural network. Compound returns that bootstrap many times—like λ-returns—become costly when naively combined with random-access experience replay (see Section 5.1).

For this reason, most large-buffer deep RL methods rely on n-step returns for multistep credit assignment. This is unfortunate, because Chapters 3 and 4 showed that λ-returns, in particular, offer compelling benefits in terms of the bias-variance trade-off and credit assignment. Thus, it would be nice to utilize λ-returns in deep RL.

Past work has combined deep RL and λ-returns by replaying minibatches of trajectories (see Section 5.2). This amortizes the cost of the λ-returns over the length of the trajectory by sharing value estimates, just as eligibility traces do, and achieves an average computational cost equivalent to n-step returns. However, it also introduces correlations within each minibatch, and truncates the λ-returns at the ends of the trajectories. Both of these effects inhibit learning, and can largely reverse any benefits over using n-step returns.

Balancing the trade-off between computation, minibatch correlations, and truncation bias is therefore crucial to the successful deployment of λ-returns in experience replay. The purpose of this chapter is to better understand these challenges and to develop algorithms that overcome them (to the extent that is possible). I introduce *cached-trajectory replay* (see Section 5.3), a replay strategy that efficiently precomputes λ-returns in bulk for a random subset of the replay memory. Minibatches are sampled from this subset for training, reducing sample correlations and truncation bias without increasing overall computational cost. As a bonus, the stale return estimates function as stable learning targets, replacing the target network.

5.1 Naive Minibatch Replay

So far, I have not closely discussed how forward-view λ-returns are computed in practice. Eq. (2.7) comprises, in theory, an infinite number of TD errors. With eligibility traces, this is not a problem because each TD error is applied as it arrives. Without eligibility traces, however, some truncation must be applied to make the computation feasible (Cichosz, 1994), either at an episode boundary or after some fixed number of time steps.

Let $G^{\lambda}_{t:t+L}$ denote the λ-return at time t truncated to length $L \geq 1$. Truncating the λ-return formula from Eq. (2.7) yields

$$G^{\lambda}_{t:t+L} \coloneqq V(S_t) + \sum_{i=0}^{L-1} (\gamma\lambda)^i \delta_{t+i} \tag{4.9}$$

$$= (1-\lambda) \sum_{n=1}^{L-1} \lambda^{n-1} G^{(n)}_t + \lambda^{L-1} G^{(L)}_t \,, \tag{4.10}$$

which were briefly introduced in Section 4.4. Eq. (4.9) is justified because it represents the hypothetical target that TD(λ) would reinforce for time t if the episode terminated at time $t+L$. Eq. (4.10) is an equivalent form: an exponentially weighted average of n-step returns just like Eq. (2.8), but with all of the truncated weight reassigned to the final L-step return. Because the weights sum to 1, Eq. (4.10) is a compound return and remains a valid TD target regardless of the truncation length.

For the rest of this chapter, I assume that the λ-returns will be applied in a deep Q-learning agent such as DQN. The value of state s is therefore given by $V(s) = \max_{a \in \mathcal{A}} Q(s,a)$ and the TD error becomes equal to

$$\delta_t = R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(S_{t+1}, a') - \max_{a \in \mathcal{A}} Q(S_t, a) \,.$$

With this definition, Eq. (4.10) unconditionally conducts value backups using the maximizing action for each n-step return, regardless of which actions were actually selected by the behavior policy. This is the (truncated) target for Peng's Q(λ) (Peng and Williams, 1996). Peng's Q(λ) is biased because it mixes on- and off-policy backups, which has been shown to perform well empirically, but it was unknown for a long time whether this is theoretically sound. Convergence conditions for Peng's Q(λ) were established only recently (Kozuno et al., 2021).

An alternative λ-return definition for Q-learning is based on Watkins' Q(λ) (Watkins, 1989), which eliminates off-policy bias by setting $\lambda = 0$ whenever an exploratory action

is taken. Watkins' Q(λ) converges to the optimal action-value function in the tabular case (Munos et al., 2016), but truncating the λ-returns in this manner can inhibit multistep credit assignment and slow learning.

I now discuss the computational cost of λ-returns in minibatched experience replay, putting aside the slight implementation differences of Peng's and Watkins' Q(λ). Eq. (4.10) comprises L distinct n-step returns, each of which requires a single value estimate for bootstrapping. Assuming all other arithmetic operations have no cost (they are minuscule compared to the network itself), then computing one λ-return requires L forward passes through the network. A minibatch of size M would thus require $L \cdot M$ forward passes to construct one minibatch for training. However, with the same compute budget, it would make more sense to sample $L \cdot M$ independent n-step returns (of the same length) to obtain more training samples per minibatch. With randomly sampled minibatches, it is hard to justify the additional cost of λ-returns, as the non-overlapping trajectories cannot be optimized. Because of the naivety of this approach, there exist no algorithms that use it (to my knowledge). The next two sections discuss more computationally sensible approaches for replaying λ-returns.

5.2 Trajectory Replay

A more efficient and straightforward way to incorporate λ-returns into experience replay is to slightly modify the sampler to return minibatches of trajectories instead of individual experiences. This way, adjacent return estimates can share value bootstraps, reducing the total computation to roughly that of an equally sized minibatch of n-step returns.

Eligibility traces are recursive, which is what enables their highly efficient update rule. It turns out that λ-returns also have this useful property. To show this, I start by defining the truncated TD(λ) error as

$$\delta^{\lambda}_{t:t+L} \coloneqq \sum_{i=0}^{L-1} (\gamma\lambda)^i \delta_{t+i}$$

and noting that Eq. (4.9) equals $G^{\lambda}_{t:t+L} = V(S_t) + \delta^{\lambda}_{t:t+L}$. The TD($\lambda$) error is recursive because

$$\delta^{\lambda}_{t:t+L} = \delta_t + \gamma\lambda\delta^{\lambda}_{t+1:t+L}\,.$$

Adding $V(S_t)$ to both sides and substituting $\delta^{\lambda}_{t+1:t+L} = G^{\lambda}_{t+1:t+L} - V(S_{t+1})$ provides a recur-

sive formula for the λ-return:

$$G^{\lambda}_{t:t+L} = G^{(1)}_{t} + \gamma\lambda\Big(G^{\lambda}_{t+1:t+L} - V(S_{t+1})\Big)\,.$$

Finally, because $G^{(1)}_{t} = R_{t+1} + \gamma\, V(S_{t+1})$, the above can be simplified to

$$G^{\lambda}_{t:t+L} = R_{t+1} + \gamma\Big((1-\lambda)\, V(S_{t+1}) + \lambda G^{\lambda}_{t+1:t+L}\Big)\,, \tag{5.1}$$

which is slightly more efficient but otherwise equivalent. (This equation also reveals the inherent bias-variance trade-off: $\lambda = 0$ fully bootstraps and $\lambda = 1$ does not bootstrap at all.) Although the derivation here is my own, I note that these recursive formulas have been well known in the literature (see, e.g., Peng and Williams, 1996; Degris et al., 2012).

Most importantly, because $G^{\lambda}_{t:t+L}$ can be computed from $G^{\lambda}_{t+1:t+L}$ using just one bootstrap, $V(S_{t+1})$, an entire L-length sequence of λ-returns can be generated using L value estimates. In other words, only L forward passes through the network are needed, making the trajectory equal in cost to a size-L minibatch of n-step returns. The disadvantage that the experiences in the trajectory are highly correlated, which can impact impact learning based on stochastic gradient descent.

Alternatively, and more common in practice, a minibatch of several trajectories can be sampled (e.g., Munos et al., 2016; Harb and Precup, 2016; Wang et al., 2017; Kozuno et al., 2021; Tang et al., 2024). This interpolates the computational cost and correlation strength between naive minibatch replay and full trajectory replay. Trajectory replay like this is convenient because it requires minimal changes to existing deep RL methods, but the resulting truncation bias (especially when L is small) and sample correlations can still counteract the benefits of using λ-returns in the first place.

5.3 Cached-Trajectory Replay

An ideal replay algorithm for λ-returns would utilize Eq. (5.1) to compute λ-returns over very long trajectories (to maximize computational efficiency and minimize truncation) while also preserving truly randomly sampled minibatches (to minimize sample correlations). Toward this, I propose a sampling strategy called cached-trajectory replay. I exemplify the concept with DQN, although it is applicable to any value-based deep RL method that uses a periodic target network. I refer to this particular instantiation as DQN(λ) (see Algorithm 1).

Algorithm 1 DQN(λ) with cached-trajectory replay

Prepopulate replay memory $\mathcal{D}$ with at least L experiences collected arbitrarily
Randomly initialize neural network parameters $\boldsymbol{\theta}$
Initialize environment state S_0
for $t = 0, 1, 2, \ldots$ until convergence **do**
    if $t \bmod \tau = 0$ **then**
        $\boldsymbol{\theta} \leftarrow \textsc{train}(\boldsymbol{\theta}, \mathcal{D})$
    end if
    Execute action $A_t = \begin{cases} \arg\max_{a \in \mathcal{A}} Q_{\boldsymbol{\theta}}(S_t, a) & \text{with probability } 1 - \epsilon_t \\ a \sim \mathcal{U}(\mathcal{A}) & \text{with probability } \epsilon_t \end{cases}$
    Observe next state S_{t+1} and reward R_{t+1}
    Store experience $(S_t, A_t, S_{t+1}, R_{t+1})$ in $\mathcal{D}$
    if terminal(S_{t+1}) **then**
        Reset environment state S_{t+1}
    end if
end for
return $\boldsymbol{\theta}$

function $\textsc{train}(\boldsymbol{\theta}, \mathcal{D})$
    $\mathcal{C} \leftarrow \textsc{build-cache}(\mathcal{D})$
    for N / B iterations **do**
        Sample minibatch $\{(S^{(i)}, A^{(i)}, G^{\lambda(i)})\}_{i=1}^{B}$ from $\mathcal{C}$
        Update network: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \frac{\alpha}{B} \sum_{i=1}^{B} \big(G^{\lambda(i)} - Q(S^{(i)}, A^{(i)})\big) \nabla Q_{\boldsymbol{\theta}}(S^{(i)}, A^{(i)})$
    end for
    return $\boldsymbol{\theta}$
end function

function $\textsc{build-cache}(\mathcal{D})$
    Initialize empty cache $\mathcal{C}$
    for N / L iterations **do**
        Sample index $i \sim \mathcal{U}\{0, |\mathcal{D}| - L\}$
        Retrieve block $\{(S_{i+j}, A_{i+j}, S_{i+1+j}, R_{i+1+j})\}_{j=0}^{L-1}$ from $\mathcal{D}$
        for $j = L-1, L-2, \ldots, 0$ **do**
            $G_j^{\lambda} = \begin{cases} R_{j+1} & \text{if terminal}(S_{j+1}) \\ R_{j+1} + \gamma\Big((1-\lambda) \max_{a' \in \mathcal{A}} Q_{\boldsymbol{\theta}}(S_{j+1}, a') + \lambda G_{j+1}^{\lambda}\Big) & \text{otherwise} \end{cases}$
            Add training sample $(S_j, A_j, G_j^{\lambda})$ to $\mathcal{C}$
        end for
    end for
    return $\mathcal{C}$
end function

Because λ-returns are substantially more expensive than n-step returns, an ideal replay-based method minimizes the number of times each return estimate must be computed. My first modification of DQN is therefore to precompute and store λ-returns along with their corresponding transition in the replay memory, $\mathcal{D}$. The calculation of the λ-returns must be sufficiently deferred because of their dependency on future states and rewards; one choice might be to wait until a terminal state is reached and then transfer the episode's λ-returns to $\mathcal{D}$. Training then becomes a matter of sampling minibatches of λ-returns from $\mathcal{D}$ and reducing the squared error. The new loss function becomes the following:

$$\mathcal{L}(\boldsymbol{\theta}) \coloneqq \mathbb{E}\left[\frac{1}{2}\big(G^{\lambda} - Q(S, A)\big)^2\right],$$

which is just Eq. (2.10) but with $\hat{G} = G^{\lambda}$. There are two major advantages to this strategy. First, return computation is not repeated when a transition is sampled in multiple minibatches. Second, adjacent λ-returns in the replay memory can be calculated efficiently with the recursive update in Eq. (5.1). The latter point is crucial; while computing randomly accessed λ-returns is prohibitively expensive as discussed in Section 5.1, computing them in reverse chronological order requires only one forward pass per λ-return.

Storing λ-returns like this means they become outdated as the value function evolves, greatly slowing learning if the replay memory is large. However, this presents a fortunate opportunity to eliminate DQN's target network altogether. Rather than copying parameters $\boldsymbol{\theta}$ to $\boldsymbol{\theta}^{-}$ every $\tau = 10{,}000$ time steps, I propose to *refresh* the λ-returns in the replay memory using the current value function. This achieves the same effect as a target network by providing stable TD targets, but eliminates the redundant network copy.

Still, this process is too expensive for typical DQN implementations, which have a replay memory capacity on the order of millions of transitions. To make the computation invariant to the size of the replay memory, I propose a new strategy where $N \,/\, L$ contiguous "blocks" of L transitions are randomly promoted from the replay memory to build a cache $\mathcal{C}$ of size N. By refreshing only this small memory and sampling minibatches directly from it, calculations are not wasted on λ-returns that are ultimately never used. Furthermore, each block can still be efficiently refreshed using Eq. (5.1) as before. Every τ time steps, the cache is regenerated from newly sampled blocks (see Figure 5.1), once again obviating the need for a target network.

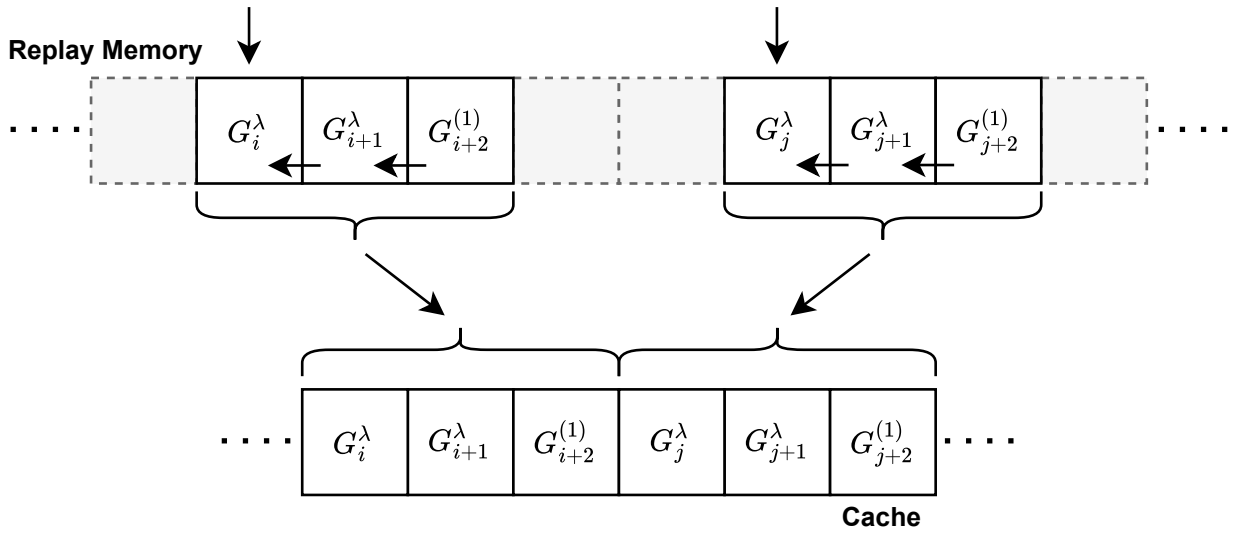
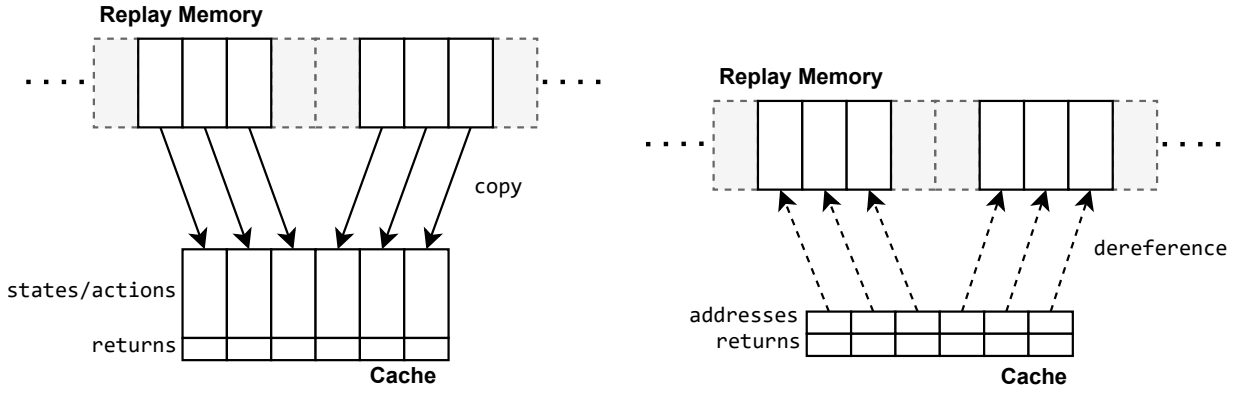


Figure 5.1: Illustration of cached-trajectory replay. For each randomly sampled index, a sequence ("block") of λ-returns is efficiently generated backwards via recursion. Together, the blocks form the new cache, which is treated as a surrogate for the replay memory for the next F time steps.

Finally, I note a possible optimization for cached-trajectory replay that I call a Virtual Replay Cache (VRC). The implementation described above consumes a large amount of memory since it duplicates state-actions pairs that are promoted into the cache from the replay memory. This extra memory cost can be avoided by using indirection. The key observation is that the storage of state-action pairs in the cache serves a purely logical function: i.e., indication of which state-action pairs are eligible for minibatch sampling. Since this data already exists in the replay memory, it is redundant to copy it into a separate data structure when it can be retrieved from the original source just as easily. The cache can instead be thought of as a useful abstraction rather than a physical data structure.

When the replay memory is implemented as an array, cache membership can be easily tracked by storing the numerical index of each experience. As a result, no experiences need to be copied at any point during the cache building process; their indices are recorded (along with their precomputed λ-returns) and then used to locate the state-action pairs later as needed. I illustrate this concept in Figure 5.2. When the memory footprint of a state-action pair is much larger than that of an integer, the VRC saves a considerable amount of memory and is slightly faster. Most importantly, in conjunction with these benefits, the VRC is a pure



(a) Physical Replay Cache

(b) Virtual Replay Cache

Figure 5.2: Graphical comparison of (a) the standard DQN(λ) cache and (b) the VRC. The standard cache wastefully duplicates state-action pairs that already exist in the replay memory—a costly operation. Instead, the VRC stores only the locations of the state-action pairs, referencing them indirectly as needed and saving substantial memory.

refactor of DQN(λ); the agent's learning performance is unaffected. For ease of exposition, I do not include these implementation details in Algorithm 1.

5.4 Atari 2600 Experiments

In order to characterize the performance of DQN(λ), I conduct experiments in six Atari 2600 games. I use OpenAI Gym (Brockman et al., 2016) to provide an interface to the Arcade Learning Environment (ALE; Bellemare et al., 2013), where observations consist of the raw frame pixels. The frames are converted to grayscale, downsized to 84×84 pixels, and stacked in groups of 4 to approximate the game state, following Mnih et al. (2015). I use the same convolutional neural network from Mnih et al. (2015) for all agents: three convolutional layers followed by two fully connected layers. The agents are trained for 10 million time steps (40 million game frames). The ϵ-greedy exploration is linearly annealed from 1 to 0.1 over the first one million time steps and then held constant. Rewards are clipped to the set $\{-1, 0, +1\}$, according to their signs, for stability.

I compare DQN(λ) against a standard target-network implementation of DQN which uses 3-step returns, which were shown to work well by Hessel et al. (2018). Overall, I match the hyperparameters from Mnih et al. (2015), except I train the neural networks with Adam

Table 5.1: Hyperparameters for DQN(λ). The upper section lists DQN hyperparameters from Mnih et al. (2015), while the lower section lists hyperparameters unique to DQN(λ).

Hyperparameter	**Symbol**	**Value**	**Description**
minibatch size	B	32	Number of samples used to compute a single gradient descent step.
replay memory size		1,000,000	Maximum number of samples that can be stored in the replay memory before old samples are discarded.
agent history length		4	Number of recent observations (game frames) simultaneously fed as input to the neural network.
refresh period	τ	10,000	Interval, measured in time steps, at which the target network is updated for DQN or the cache is rebuilt for DQN(λ).
discount factor	γ	0.99	Weighting coefficient that influences the importance of future rewards.
replay start size		50,000	Number of time steps for which to initially execute a uniform random policy and prepopulate the replay memory.
cache size	N	80,000	Number of samples used to build the cache upon refresh.
block size	L	100	Atomic length of the sampled sequences that are promoted into the cache upon refresh.

(Kingma and Ba, 2015) using $\alpha = 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-4}$. My chosen hyperparameters are summarized in Table 5.1.

When sizing the cache for DQN(λ), I make sure that the average number of training samples per time step (8:1 ratio) is preserved for fair comparison. Specifically,

$$\frac{\text{10,000 time steps}}{\text{1 refresh}} \times \frac{\text{1 minibatch}}{\text{4 time steps}} \times \frac{\text{32 samples}}{\text{1 minibatch}} = \frac{\text{80,000 samples}}{\text{1 refresh}} .$$

Hence, I use $N = 80{,}000$ for all experiments, which equates to 2,500 minibatches per cache refresh (because $80{,}000 \,/\, 32 = 2{,}500$). To help reduce bias when building the cache, I permit overlapping blocks. That is, I do not check for boundary constraints of nearby blocks, nor do I align blocks to a fixed grid (in contrast to a real computer cache). This means that multiple copies of the same experience might exist in the cache simultaneously, but each experience therefore has the same probability of being promoted.

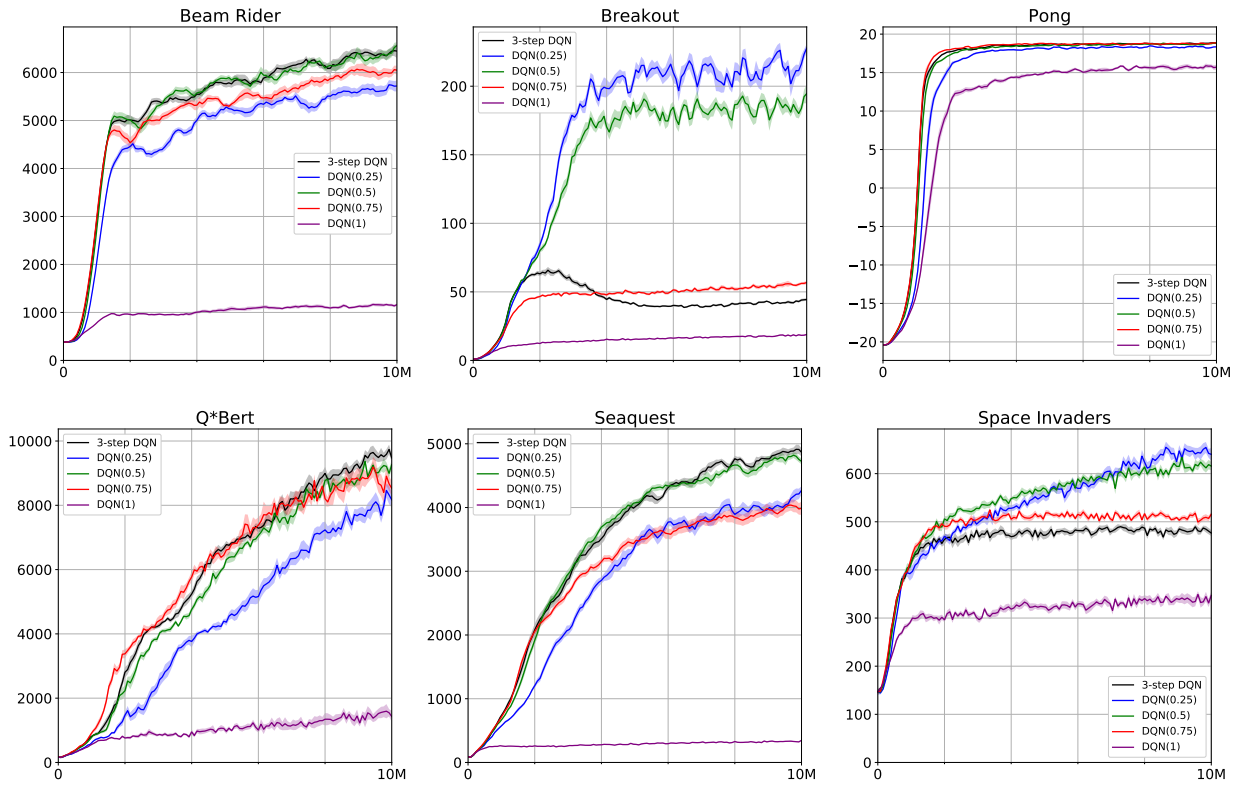


Figure 5.3: Sample efficiency of Peng's DQN(λ) with $\lambda \in \{0.25, 0.5, 0.75, 1\}$ compared against 3-step DQN in six Atari games. The horizontal axis is time steps. The results are averaged over 10 trials with standard errors of the mean shaded.

I first compare DQN(λ) using Peng's Q(λ) with $\lambda \in \{0.25, 0.5, 0.75, 1\}$ against the 3-step baseline in each of the six Atari games. In Figure 5.3, I plot the agent's performance at a given time step by averaging the earned scores of its past 100 completed episodes. The results are averaged over 10 trials with the standard error of the mean shaded. For every game, at least one λ-value matches or outperforms the 3-step return. Notably, $\lambda \in \{0.25, 0.5\}$ yields huge performance gains over the baseline on Breakout and Space Invaders. This finding is quite interesting because n-step returns have been shown to perform poorly in Breakout (Hessel et al., 2018), suggesting that λ-returns can be a better alternative.

In certain scenarios, it is desirable to train a model as quickly as possible without regard to the number of environment samples. For the best λ-value tested on each game in Figure 5.3, I re-plot the learning curve a function of wall-clock time (measured in hours) and compare it against the 3-step baseline in Figure 5.4. Notably, DQN(λ) completes training faster than DQN in five of the six games. This shows that the cache can be more computationally efficient

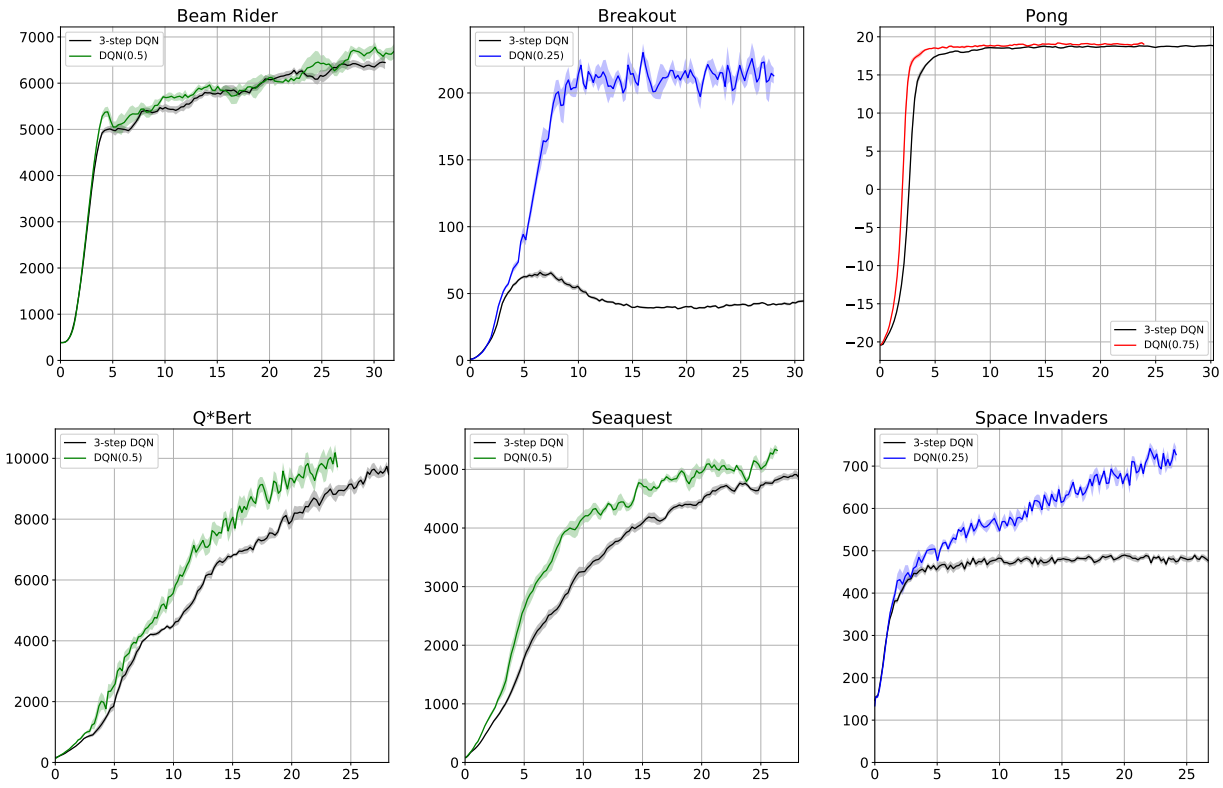


Figure 5.4: Real-time efficiency of Peng's DQN(λ) with the best λ-value from Figure 5.3 compared against 3-step DQN in six Atari games. The horizontal axis is time in hours. The results are averaged over 10 trials with standard errors of the mean shaded.

than a target network. The speedup can be mainly attributed to better parallelization on the graphics processing unit (GPU) when computing action values for the λ-returns, because the cache blocks are larger than a typical minibatch ($L = 100$ versus $B = 32$).

Because Peng's Q(λ) is a biased return estimator, I also repeat the previous experiment using Watkins' Q(λ). The results are plotted in Figure 5.5. Again, the results are averaged over 10 trials with the standard error of the mean shaded. Surprisingly, Watkins' Q(λ) fails to outperform Peng's Q(λ) in every game tested. The worse performance is due to trace cutting, which—although well motivated—slows credit assignment despite the bias correction. This suggests that long-term credit assignment is more important in these six games.

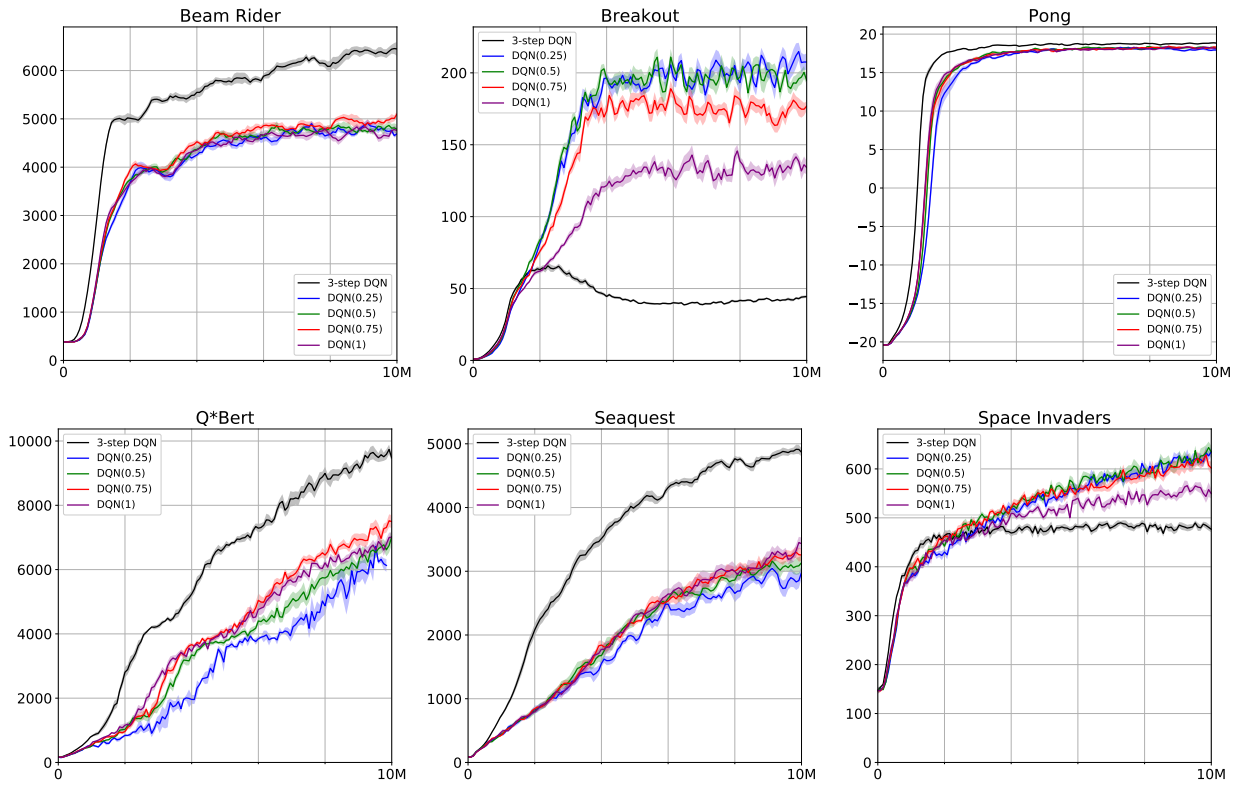


Figure 5.5: Sample efficiency of Watkins' DQN(λ) with $\lambda \in \{0.25, 0.5, 0.75, 1\}$ compared against 3-step DQN in six Atari games. The horizontal axis is time steps. The results are averaged over 10 trials with standard errors of the mean shaded.

5.5 Discussion

I proposed cached-trajectory replay, a technique that allows for the efficient integration of λ-returns in value-based RL methods with experience replay. Compared to naive minibatch replay and trajectory replay, cached-trajectory replay strikes a more favorable balance between computational efficiency, minibatch correlations, and truncation bias. Additionally, because λ-returns are stored in a periodically refreshed cache, the need for a target network is eliminated. My experiments show that replaying λ-returns in this manner can greatly increase the sample efficiency of DQN compared to using a well-tuned n-step return—although Peng's Q(λ) appears to be much more effective than Watkins' Q(λ).

While this chapter focused specifically on λ-returns, cached-trajectory replay is equally applicable for other multistep return estimators. One avenue for future work is to utilize a low-variance, bias-corrected return such as Tree Backup (Precup et al., 2000), Retrace

(Munos et al., 2016), or trajectory-aware returns (see Chapter 6) for potentially better performance. Cached-trajectory replay is a modular technique which can be useful to a wide range of off-policy RL methods by enabling the efficient computation of multistep returns beyond simple n-step returns.

Chapter 6

Trajectory-Aware Off-Policy Corrections

Off-policy learning is crucial for sample-efficient RL with experience replay. However, counteracting the off-policy bias of multistep returns without exacerbating variance is challenging. Classically, off-policy bias is corrected in a *per-decision* manner: temporal-difference errors are re-weighted by the IS ratio following each action. Many off-policy algorithms rely on this mechanism, along with differing protocols for *cutting* the IS ratios to combat the variance of the estimator (e.g., Precup et al., 2000; Harutyunyan et al., 2016; Munos et al., 2016; Mahmood et al., 2017).

If an IS ratio is ever fully cut, the effect cannot be reversed later by per-decision methods. This has led to the development of credit-assignment strategies that account for multiple past experiences simultaneously. These *trajectory-aware* methods have not been extensively analyzed, and their theoretical justification remains uncertain.

In this chapter, I propose a multistep operator that expresses both per-decision and trajectory-aware methods. I prove convergence conditions for my operator in the tabular setting, establishing the first guarantees for several existing methods as well as many new ones. Finally, I introduce Recency-Bounded Importance Sampling (RBIS), which leverages trajectory awareness to perform robustly across λ-values in several off-policy control tasks.

6.1 Trajectory-Aware Return Estimators

Recall from Section 2.6 that arbitrary per-decision corrections have the form

$$\hat{G}_t = Q(S_t, A_t) + \sum_{i=0}^{\infty} \gamma^i \left(\prod_{j=t+1}^{t+i} \tilde{\rho}_j \right) \delta^{\pi}_{t+i} \,. \tag{2.11}$$

This return estimator facilitates convergence to q_π whenever $0 \leq \tilde{\rho}_j \leq \rho_j$ for all j, where $\rho_j \coloneqq \pi(A_j \mid S_j) \,/\, b(A_j \mid S_j)$ is the IS ratio. The coefficient $\tilde{\rho}_j$ is referred to as a *trace* by Munos et al. (2016) in reference to backward-view eligibility traces, although Eq. (2.11) is written in the forward view for clarity.

While per-decision traces are convenient from a computational perspective, they require making choices about how much to cut each trace without considering the effects of previous choices. This can lead to suboptimal decisions. For example, if a trace is cut by setting $c_j = 0$ at some time step $j > t$, then state-action pair (S_t, A_t) will stop undergoing reinforcement (until it is visited again). Because of the multiplicative nature of these corrections, this effect cannot be reversed later. Regardless of whatever new experiences are encountered by the agent, experiences before time j become ineligible for additional credit assignment, potentially resulting in an opportunity cost. In fact, this exact phenomenon is why Watkins' Q(λ) often learns more slowly than Peng's Q(λ), as was shown experimentally in Section 5.4, even though the former avoids off-policy bias. The same effect (but to a lesser extent) impacts Tree Backup and Retrace, where $c_j \leq 1$ always holds in Eq. (2.11), implying that the eligibilities of past experiences cannot increase.[19]

I illustrate this phenomenon in a small, deterministic MDP that I call the Tightrope Problem (see Figure 6.1). The environment consists of n sequential, non-terminal states with two actions available: a_1 and a_2. The agent starts in state s_1 and advances from s_i to s_{i+1} whenever it takes action a_1. If $i = n$, then the episode terminates and the agent receives +1 reward. Taking action a_2 in any state immediately terminates the episode with no reward. Clearly, the optimal policy is to execute a_1 regardless of the state.

Now consider the following undiscounted off-policy learning scenario. Suppose the agent's behavior policy b is uniform random, but the target policy π is ϵ-greedy with respect to

[19] For on-policy learning, this was shown in Chapter 4 to be desirable. However, for off-policy learning, it can be beneficial to offset low-probability outcomes under the behavior policy with increased eligibilities.

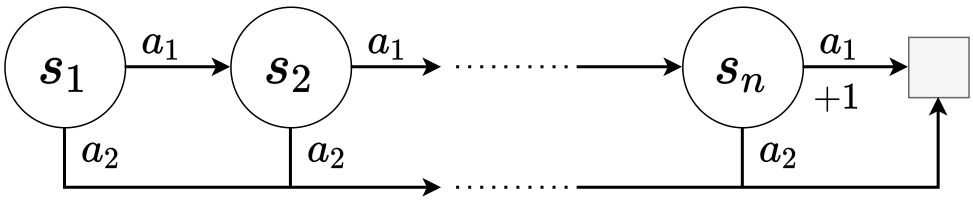


Figure 6.1: The Tightrope Problem. Starting from state s_1, the agent must take a specific sequence of n actions to receive the $+1$ reward.

$Q(s,a)$. For each state s, it follows that $\pi(a \mid s) = 1-\epsilon$ if $a = \arg\max_{a' \in \mathcal{A}} Q(s,a')$ and $\pi(a \mid s) = \epsilon$ otherwise. I assume ϵ is small in the sense that $\epsilon < \frac{1}{2}$.

Suppose now that the agent successfully receives the $+1$ reward during an episode, implying that it took action a_1 on every time step. I can compute the eligibility of the initial state-action pair (s_1, a_1) as an expression in the number $k \leq n-1$ of remaining incorrect actions in the greedy policy: i.e., where $\arg\max_{a' \in \mathcal{A}} Q(s,a') \neq a_1$ and $s \neq s_1$.

Assume $\lambda = 1$ for this example. The standard IS estimator, which does not cut traces when $\lambda = 1$, assigns an eligibility of

$$\left(\frac{1-\epsilon}{1\,/\,2}\right)^{n-1-k}\left(\frac{\epsilon}{1\,/\,2}\right)^{k} = \big(2(1-\epsilon)\big)^{n-1-k}(2\epsilon)^{k}\,. \tag{6.1}$$

This can be greater than 1 when $k \ll n-1$, suggesting that the agent's behavior should be heavily reinforced when the greedy and optimal policies agree closely. However, a per-decision method like Retrace cuts traces by setting $\tilde{\rho}_j = \min(1, \rho_j)$ in Eq. (2.11) without considering the full trajectory (see Section 6.2), ultimately assigning a much smaller eligibility:

$$\left(\min\left(1, \frac{1-\epsilon}{1\,/\,2}\right)\right)^{n-1-k}\left(\min\left(1, \frac{\epsilon}{1\,/\,2}\right)\right)^{k} = (2\epsilon)^{k}\,.$$

The eligibility in this case decays monotonically for every suboptimal action in the greedy policy, despite the fact that $\lambda = 1$, illuminating how per-decision trace cutting can lead to excessively small eligibilities—especially when ϵ is close to 0.

This issue stems from the fact that Retrace is not *aware* of its past eligibilities, and continues to decay them even when they underestimate the full IS product. This issue is not unique to Retrace, and affects other per-decision methods like Tree Backup. A *trajectory-aware* method that can actively adapt its trace-cutting behavior based on the magnitude of past eligibilities could be more efficient.

One way to obtain a trajectory-aware method is to compute the exact IS product in Eq. (6.1) and then make adjustments to it to achieve certain properties (e.g., convergence and variance reduction). For example, Truncated IS (see Section 6.2) simply imposes a fixed bound on the IS estimator:

$$\min\Big(1, \big(2(1-\epsilon)\big)^{n-1-k}(2\epsilon)^k\Big)\,.$$

Truncated IS reduces the eligibility only when it exceeds a prespecified threshold, effectively avoiding trace cuts when the true IS estimate is small. In Section 6.5, I propose an algorithm, RBIS, which achieves a similar effect using a recursive, time-decaying threshold.

As this example demonstrates, it can be advantageous to consider the agent's past experiences to produce better decisions regarding credit assignment. One of this chapter's principal contributions is the proposal and analysis of an off-policy operator $\mathcal{M}$ that encompasses this possibility. Let $\mathcal{F}_{t:t+i} \coloneqq (S_j, A_j)_{j=t}^{t+i}$ be the agent's trajectory from time t to $t+i$. I define a trajectory-aware return estimate to have the form

$$\hat{G}_t = Q(S_t, A_t) + \sum_{i=0}^{\infty} \gamma^i H_{t:t+i} \delta^\pi_{t+i}\,, \tag{6.2}$$

where $H_{t:t+i} \coloneqq h(\mathcal{F}_{t:t+i})$ and $h\colon (\mathcal{S}\times\mathcal{A})^* \to \mathbb{R}$ is an arbitrary weighting over trajectories. This corresponds to the trajectory-aware operator $\mathcal{M}\colon \mathbb{R}^{|\mathcal{S}\times\mathcal{A}|} \to \mathbb{R}^{|\mathcal{S}\times\mathcal{A}|}$ which is defined by

$$(\mathcal{M}Q)(s,a) \coloneqq Q(s,a) + \mathbb{E}_b\left[\sum_{i=0}^{\infty} \gamma^i H_{t:t+i}\delta^\pi_{t+i} \;\middle|\; (S_t, A_t) = (s,a)\right]. \tag{6.3}$$

I define $H_{t:t} \coloneqq 1$ to ensure that the first TD error, δ^π_t, is applied. In Section 6.3, I characterize the values of $H_{t:t+i}$ for $i \geq 1$ that lead to convergence.

The major challenge—and main novelty—of the trajectory-aware operator is the complex dependence on the experience sequences. This makes the operator difficult to analyze mathematically, as the terms in the series $1 + \gamma H_{t:t+1} + \gamma^2 H_{t:t+2} + \cdots$ generally share no common factors that would allow a recursive formula. Some off-policy methods, however, cannot be described by factored traces, and therefore removing this assumption is necessary to understand existing algorithms (see Section 6.2), while also paving the way for new credit-assignment methods. In the special case where $H_{t:t+i}$ *does* factor into Markov coefficients, i.e., $H_{t:t+i} = \prod_{j=t+1}^{t+i} \tilde{\rho}_j$, then Eq. (6.2) reduces to Eq. (2.11), recovering the per-decision setting studied by Munos et al. (2016). The operator $\mathcal{M}$, therefore, unifies per-decision and trajectory-aware methods.

6.2 Unifying Off-Policy Algorithms

The operator $\mathcal{M}$ is a strict generalization of the per-decision operator considered by Munos et al. (2016), allowing me to express existing algorithms in this form. I provide a non-exhaustive list of examples below with the corresponding value of $H_{t:t+i}$ used in $\mathcal{M}$. For brevity, let $\Pi_{t+1:t+i} \coloneqq \prod_{j=t+1}^{t+i} \rho_j$ for $i \geq 1$ and let $\Pi_{t+1:t} \coloneqq 1$.

Importance Sampling: $H_{t:t+i} = \lambda^i \Pi_{t+1:t+i}$ (Kahn and Marshall, 1953). The standard approach for correcting off-policy bias. Although it is the only unbiased estimator in this list (if $\lambda = 1$), it suffers from high variance, making it difficult to utilize.

$\mathbf{Q}^\pi(\lambda)$: $H_{t:t+i} = \lambda^i$ (Harutyunyan et al., 2016). A straightforward algorithm that decays the TD errors by a fixed constant. The algorithm does not require explicitly knowing b, which is desirable, but can diverge if π and b differ too much (Harutyunyan et al., 2016, Theorem 1).

Tree Backup: $H_{t:t+i} = \lambda^i \prod_{j=t+1}^{t+i} \pi(A_j \mid S_j)$ (Precup et al., 2000). A method that automatically cuts traces according to the product of probabilities under π, which forms a conservative lower bound on the IS estimate. Tree Backup converges for any behavior policy b, but it is not efficient since traces are cut excessively—especially in the on-policy case.

Retrace: $H_{t:t+i} = \lambda^i \prod_{j=t+1}^{t+i} \min(1, \rho_j)$ (Munos et al., 2016). A convergent algorithm for arbitrary policies π and b that remains efficient in the on-policy case because it does not cut traces (if $\lambda = 1$); however, the fact that $H_{t:t+i}$ never increases as $i \to \infty$ can cause the trace products to decay too quickly in practice (Mahmood et al., 2017; Rowland et al., 2020).

All of the above can be analyzed using a per-decision operator. The next two, on the other hand, have weights that depend jointly on the trajectory. I use the $\mathcal{M}$ operator to prove theoretical properties about these methods in Section 6.3.

Recursive Retrace: $H_{t:t+i} = \lambda \min(1, H_{t:t+i-1} \cdot \rho_{t+i})$ (Munos et al., 2016). A modification to Retrace conjectured to lead to faster learning. It clips products of ratios, rather than individual ratios. Its convergence for control was an open question, which I resolve in Section 6.3.

Truncated Importance Sampling: $H_{t:t+i} = \lambda^i \min(1, \Pi_{t+1:t+i})$ (Ionides, 2008). A simple but effective method to combat the variance of IS. Variations of this algorithm have been applied in the RL literature (e.g., Uchibe and Doya, 2004; Wawrzyński and Pacut, 2007; Wawrzyński, 2009), but, to my knowledge, its convergence in an MDP setting had not been studied. In Section 6.4, I show that it diverges in at least one off-policy problem.

6.3 Convergence Analysis

In this section, I study the convergence properties of the $\mathcal{M}$ operator for policy evaluation and control. It is convenient to re-express Eq. (6.3) in vector notation for my analysis. To do this, I first bring the expectation inside the sum, by linearity of expectation:

$$(\mathcal{M}q)(s,a) = q(s,a) + \sum_{i=0}^{\infty} \gamma^i \, \mathbb{E}_b\big[H_{t:t+i}\, \delta^{\pi}_{t+i} \;\big|\; (S_t, A_t) = (s,a)\big] \; . \tag{6.4}$$

To write Eq. (6.4) in vector form, I define an operator $\mathcal{H}_i$ such that

$$(\mathcal{H}_i q)(s,a) \coloneqq \mathbb{E}_b\big[H_{t:t+i}\, q(S_{t+i}, A_{t+i}) \;\big|\; (S_t, A_t) = (s,a)\big] \, ,$$

allowing me to express the $\mathcal{M}$ operator as

$$\mathcal{M}\boldsymbol{q} = \boldsymbol{q} + \sum_{i=0}^{\infty} \gamma^i \mathcal{H}_i (T_\pi \boldsymbol{q} - \boldsymbol{q}) \, . \tag{6.5}$$

$\mathcal{H}_i$ is a linear operator and hence can be represented as a matrix in $\mathbb{R}^{|\mathcal{S}\times\mathcal{A}|\times|\mathcal{S}\times\mathcal{A}|}$, the elements of which are nonnegative. Each element of $\mathcal{H}_i$, row-indexed by $(s,a) \in \mathcal{S}\times\mathcal{A}$ and column-indexed by $(s',a') \in \mathcal{S}\times\mathcal{A}$, has the form

$$\begin{aligned}\mathcal{H}_i\big((s,a),(s',a')\big) = \Pr_b\big((S_{t+i}, A_{t+i}) = (s',a') \mid (S_t, A_t) = (s,a)\big) \\ \times \mathbb{E}_b\big[H_{t:t+i} \;\big|\; (S_t, A_t) = (s,a), (S_{t+i}, A_{t+i}) = (s',a')\big] \; .\end{aligned}$$

This is because multiplying the matrix $\mathcal{H}_i$ with a vector $\boldsymbol{q}$ results in the same operation as the weighted expected value in Eq. (6.4):

$$\begin{aligned}(\mathcal{H}_i q)(s,a) &= \sum_{s'\in\mathcal{S}} \sum_{a'\in\mathcal{A}} \mathcal{H}_i\big((s,a),(s',a')\big)\, q(s',a') \\ &= \sum_{s'\in\mathcal{S}} \sum_{a'\in\mathcal{A}} \Pr_b\big((S_{t+i}, A_{t+i}) = (s',a') \mid (S_t, A_t) = (s,a)\big) \\ &\qquad \times \mathbb{E}_b\big[H_{t:t+i} \;\big|\; (S_t, A_t) = (s,a), (S_{t+i}, A_{t+i}) = (s',a')\big] \\ &\qquad \times q(s',a')\end{aligned}$$

$$= \mathbb{E}_b\left[\mathbb{E}_b\big[H_{t:t+i} \;\big|\; (S_t, A_t) = (s,a)\big] \times q(S_{t+i}, A_{t+i}) \;\middle|\; (S_t, A_t) = (s,a) \right]$$
$$= \mathbb{E}_b\big[H_{t:t+i} \times q(S_{t+i}, A_{t+i}) \;\big|\; (S_t, A_t) = (s,a)\big] \,.$$

So, when $\mathcal{H}_i$ is applied to the expected TD error, $T_\pi \boldsymbol{q} - \boldsymbol{q}$, Eq. (6.5) becomes Eq. (6.4).

Note that $\mathcal{H}_0 = \boldsymbol{I}$, the identity matrix, because of my earlier definition of $H_{t:t} \coloneqq 1$ in Eq. (6.3). In the following sections, all inequalities involving vectors or matrices should be interpreted element wise. I let $\|\boldsymbol{X}\| \coloneqq \|\boldsymbol{X}\|_\infty$ for a matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{S}\times\mathcal{A}|\times|\mathcal{S}\times\mathcal{A}|}$, which corresponds to its maximum absolute row sum. I also define $\mathbf{1} \in \mathbb{R}^{|\mathcal{S}\times\mathcal{A}|}$ to be the vector of ones, such that $\boldsymbol{X}\mathbf{1}$ returns the row sums of $\boldsymbol{X}$. To further simplify notation, I define $\boldsymbol{P}_\pi \coloneqq \boldsymbol{P}\boldsymbol{E}_\pi$, making the action-value Bellman operator equal to

$$T_\pi \boldsymbol{q} = \boldsymbol{r} + \gamma \boldsymbol{P}_\pi \boldsymbol{q} \,.$$

I start in the off-policy policy evaluation setting. Specifically, my goal is to prove that the repeated application of the $\mathcal{M}$ operator to an arbitrarily initialized vector $\boldsymbol{q} \in \mathbb{R}^{|\mathcal{S}\times\mathcal{A}|}$ converges to $\boldsymbol{q}_\pi$. I use the following condition which ensures that the trajectory-aware coefficients do not increase on average as time elapses. My theorem also requires two lemmas which I state below (proofs in Appendices B.1.1 and B.1.2, respectively).

Condition 6.1. *$H_{t:t+i} \le H_{t:t+i-1} \cdot \rho_{t+i}$ for all $t \ge 0$ and $i \ge 1$.*

Lemma 6.2. *$\boldsymbol{q}_\pi$ is a fixed point of $\mathcal{M}$. The difference between $\mathcal{M}\boldsymbol{q}$ and $\boldsymbol{q}_\pi$ is*

$$\mathcal{M}\boldsymbol{q} - \boldsymbol{q}_\pi = \boldsymbol{Z}(\boldsymbol{q} - \boldsymbol{q}_\pi) \,, \tag{6.6}$$

where $\boldsymbol{Z} \coloneqq \sum_{i=1}^{\infty} \gamma^i (\mathcal{H}_{i-1}\boldsymbol{P}_\pi - \mathcal{H}_i)$.

Lemma 6.3. *If Condition 6.1 holds, then $\boldsymbol{Z}$ has nonnegative elements and its row sums obey $\boldsymbol{Z}\mathbf{1} \le \gamma$.*

Theorem 6.4. *If Condition 6.1 holds, then $\mathcal{M}$ is a contraction mapping with $\boldsymbol{q}_\pi$ as its unique fixed point. Consequently, $\lim_{k\to\infty} \mathcal{M}^k \boldsymbol{q} = \boldsymbol{q}_\pi$, $\forall\ \boldsymbol{q} \in \mathbb{R}^{|\mathcal{S}\times\mathcal{A}|}$.*

Proof. Lemma 6.2 establishes Eq. (6.6) and that $\boldsymbol{q}_\pi$ is a fixed point of $\mathcal{M}$. Lemma 6.3 shows that $\boldsymbol{Z} \ge 0$ and $\boldsymbol{Z}\mathbf{1} \le \gamma$ using the assumption that $H_{t:t+i} \le H_{t:t+i-1} \cdot \rho_{t+i}$ for all $t \ge 0$ and

$i \geq 1$ (Condition 6.1). Consequently, $\boldsymbol{Z}(\boldsymbol{q} - \boldsymbol{q}_\pi)$ in Eq. (6.6) is a vector whose components each comprise a nonnegatively weighted combination of the components of $\boldsymbol{q} - \boldsymbol{q}_\pi$, where the weights add up to at most γ. This means $\|\mathcal{M}\boldsymbol{q} - \boldsymbol{q}_\pi\| \leq \gamma\|\boldsymbol{q} - \boldsymbol{q}_\pi\|$, and $\mathcal{M}$ is a contraction mapping. Its fixed point, $\boldsymbol{q}_\pi$, must therefore be unique by the Banach fixed-point theorem (Banach, 1922), implying that $\lim_{k\to\infty} \mathcal{M}^k\boldsymbol{q} = \boldsymbol{q}_\pi$ for every $\boldsymbol{q} \in \mathbb{R}^{|\mathcal{S}\times\mathcal{A}|}$ when $\gamma < 1$. □

Given that the ratio $H_{t:t+i}$ / H_{t+i-1} is bounded by ρ_{t+i} (Condition 6.1), the $\mathcal{M}$ operator admits convergence to $\boldsymbol{q}_\pi$. Intuitively, this ratio can be thought of as the *effective* per-decision factor at time $t + i$; convergence is guaranteed whenever this factor is no greater than ρ_{t+i}, analogous to the convergence result for the per-decision operator (Munos et al., 2016, Theorem 1). My theorem implies the existence of a space of convergent trajectory-aware algorithms, because each trace $H_{t:t+i}$ can be chosen arbitrarily (but consistently) so long as it always satisfies the bound on this ratio.

I now consider the more challenging setting of control. Given sequences of target policies $(\pi_k)_{k\geq 0}$ and behavior policies $(b_k)_{k\geq 0}$, I aim to show that the sequence of value functions $(\boldsymbol{q}_k)_{k\geq 0}$ given by $\boldsymbol{q}_{k+1} \coloneqq \mathcal{M}_k\boldsymbol{q}_k$ converges to $\boldsymbol{q}_*$. Here, $\mathcal{M}_k$ is $\mathcal{M}$ defined for π_k and b_k.

Compared to the convergence proof of the per-decision operator (Munos et al., 2016, Theorem 2), the main novelty of my proof is the fact that the traces under $\mathcal{M}$ are not Markov. Consequently, I require new techniques to establish bounds on $\boldsymbol{q} - \boldsymbol{q}_*$, since Eq. (6.3) is not representable as an infinite geometric series and so the sum does not have a closed-form expression. I additionally relax two assumptions from Munos et al. (2016), on initialization of the value function and on increasing greediness of the policy. My proof requires only that the target policies become *greedy in the limit*. A sequence of policies is greedy in the limit if $T_{\pi_k}\boldsymbol{q}_k \to T\boldsymbol{q}_k$ as $k \to \infty$, where T is the Bellman optimality operator

$$T\boldsymbol{q} \coloneqq \boldsymbol{r} + \gamma\boldsymbol{P}E\boldsymbol{q}$$

and $E\colon \mathbb{R}^{|\mathcal{S}\times\mathcal{A}|} \to \mathbb{R}^{|\mathcal{S}|}$ is defined such that $(Eq)(s) \coloneqq \max_{a\in\mathcal{A}} q(s, a)$. I discuss the significance of these relaxed assumptions at the end of this section.

First, let $\boldsymbol{C}_k \coloneqq \sum_{i=0}^{\infty} \gamma^i\mathcal{H}_i$ for the policies π_k and b_k, and write the $\mathcal{M}$ operator at iteration k as

$$\mathcal{M}_k\boldsymbol{q} = \boldsymbol{q} + \boldsymbol{C}_k(T_{\pi_k}\boldsymbol{q} - \boldsymbol{q})\,. \tag{6.7}$$

I now present my convergence theorem for control.

Theorem 6.5. *Consider a sequence of target policies $(\pi_k)_{k\geq 0}$ and a sequence of arbitrary behavior policies $(b_k)_{k\geq 0}$. Let $\boldsymbol{q}_0$ be an arbitrary vector in $\mathbb{R}^{|\mathcal{S}\times\mathcal{A}|}$ and define the sequence $\boldsymbol{q}_{k+1} \coloneqq \mathcal{M}_k \boldsymbol{q}_k$, where $\mathcal{M}_k$ is the operator defined by Eq. (6.7). Assume that $(\pi_k)_{k\geq 0}$ is greedy in the limit, and let $\epsilon_k \geq 0$ be the smallest constant such that $T_{\pi_k}\boldsymbol{q}_k \geq T\boldsymbol{q}_k - \epsilon_k\|\boldsymbol{q}_k\|\mathbf{1}$. If Condition 6.1 holds for all k, then*

$$\|\mathcal{M}_k\boldsymbol{q}_k - \boldsymbol{q}_*\| \leq \gamma\|\boldsymbol{q}_k - \boldsymbol{q}_*\| + \frac{\epsilon_k}{1-\gamma}\|\boldsymbol{q}_k\|\,, \tag{6.8}$$

and, consequently, $\lim_{k\to\infty} \boldsymbol{q}_k = \boldsymbol{q}_*$.

Proof (sketch; full proof in Appendix B.1.3). I define matrices $\boldsymbol{Z}_k$ and $\boldsymbol{Z}_k^*$, which correspond to $\boldsymbol{Z}$ in Eq. (6.6) for target policies π_k and π_*, respectively, and behavior policy b_k. I then derive the inequalities

$$\boldsymbol{Z}_k^*(\boldsymbol{q}_k - \boldsymbol{q}_*) - \epsilon_k\|\boldsymbol{q}_k\|\boldsymbol{C}_k\mathbf{1} \leq \mathcal{M}_k\boldsymbol{q}_k - \boldsymbol{q}_* \leq \boldsymbol{Z}_k(\boldsymbol{q}_k - \boldsymbol{q}_*)\,,$$

which together imply Eq. (6.8). Thus, $\mathcal{M}_k$ is nearly a contraction mapping with $\boldsymbol{q}_*$ as its unique fixed point, excepting the influence of the $O(\|\boldsymbol{q}_k\|)$ term. However, the greedy-in-the-limit target policies guarantee that $\epsilon_k \to 0$. Showing that $\|\boldsymbol{q}_k\|$ remains finite completes the proof because $\|\boldsymbol{q}_k - \boldsymbol{q}_*\| \to 0$ must follow. □

The convergence criterion for $H_{t:t+i}$ (Condition 6.1) is the same for both policy evaluation and control. In fact, the only additional assumption I need for control is the greedy-in-the-limit target policies. Crucially, the proof allows arbitrary behavior policies and an arbitrary value function initialization, which I discuss below.

To summarize, removing the Markov assumption of the per-decision operator to enable trajectory-aware methods was my primary motivation. When the trace factors are Markov, the operator $\mathcal{H}_i$ is independent of i, allowing the sum $\sum_{i=0}^{\infty}\gamma^i\mathcal{H}_i$ to be reduced to $\sum_{i=0}^{\infty}(\gamma\boldsymbol{P}_{\tilde{\rho}})^i = (\boldsymbol{I} - \gamma\boldsymbol{P}_{\tilde{\rho}})^{-1}$ for a linear operator $\boldsymbol{P}_{\tilde{\rho}}$, as was done by Munos et al. (2016). In my proofs, I avoid the Markov assumption by directly analyzing the infinite sum, which generally does not have a closed-form expression. Thus, I have established the first convergence guarantees for arbitrary trajectory-aware methods (in expectation).

I permit any initialization of $\boldsymbol{q}_0$ in the control setting. In contrast, Munos et al. (2016) made the assumption that $T_{\pi_0}\boldsymbol{q}_0 - \boldsymbol{q}_0 \geq 0$ in order to produce a lower bound on $\mathcal{R}_k\boldsymbol{q}_k - \boldsymbol{q}_*$

(where $\mathcal{R}_k$ is the per-decision operator at iteration k), accomplished in practice by a pessimistic initialization of the value function: $q_0(s,a) = -\|\boldsymbol{r}\| / (1-\gamma), \forall\ (s,a) \in \mathcal{S} \times \mathcal{A}$. Since the per-decision operator is a special case of $\mathcal{M}$ in which each coefficient $H_{t:t+i}$ factors into Markov traces, I deduce as a corollary that Retrace and all other per-decision algorithms do not require pessimistic initialization for convergence.

Finally, my requirement of greedy-in-the-limit target policies in Theorem 6.5 is less restrictive than the increasingly greedy policies proposed by Munos et al. (2016). I need only $\lim_{k\to\infty} T_{\pi_k} \boldsymbol{q}_k = T\boldsymbol{q}_k$, and I do not force the sequence of target policies to satisfy $\boldsymbol{P}_{\pi_{k+1}} \boldsymbol{q}_{k+1} \geq \boldsymbol{P}_{\pi_k} \boldsymbol{q}_{k+1}$. This implies that the agent may target non-greedy policies for any finite period of time, as long as the policies do eventually become arbitrarily close to the greedy policy. As a corollary, increasingly greedy policies are not necessary for the optimal convergence of Retrace and other per-decision methods.

6.4 Examples of Convergence and Divergence

The generality of the trajectory-aware operator means that it provides convergence guarantees for a number of credit-assignment methods that I did not discuss in Section 6.2. These include variable or past-dependent λ-values (e.g., Watkins, 1989; Singh and Sutton, 1996; Yu et al., 2018). All of these can be represented in a common form and shown to satisfy Condition 6.1. Convergence of the instantiated trajectory-aware operator for policy evaluation and control follows because Condition 6.1 is sufficient to apply Theorems 6.4 and 6.5.

Proposition 6.6. *Any trajectory-aware coefficients expressible in the form* $H_{t:t+i} = \prod_{j=t+1}^{t+i} \lambda(\mathcal{F}_{t+1:j}) \cdot \rho_j$, *where* $\lambda \colon (\mathcal{S} \times \mathcal{A})^* \to [0,1]$, *satisfy Condition 6.1.*

Proof. $H_{t:t+i} = H_{t:t+i-1} \cdot \lambda(\mathcal{F}_{t+1:t+i}) \cdot \rho_{t+i} \leq H_{t:t+i-1} \cdot \rho_{t+i}$. □

In Section 6.2, I also discussed two existing trajectory-aware methods whose convergence was unknown. I first show that Recursive Retrace satisfies the required condition.

Proposition 6.7. *Recursive Retrace satisfies Condition 6.1.*

Proof. Munos et al. (2016, Eq. 9) define the trace product for Recursive Retrace such that $H_{t:t+i} = H_{t:t+i-1} \cdot \lambda \min\left(\frac{1}{H_{t:t+i-1}}, \rho_{t+i}\right) = \lambda \min(1, H_{t:t+i-1} \cdot \rho_{t+i}) \leq H_{t:t+i-1} \cdot \rho_{t+i}$. □

Unfortunately, the coefficients for Truncated IS do not always satisfy the required bound.

Proposition 6.8. *Truncated IS may violate Condition 6.1.*

Proof. I provide a counterexample. Recall that Truncated IS defines $H_{t:t+i} = \lambda^i \min(1, \Pi_{t+1:t+i})$. Consider a trajectory $\mathcal{F}_{t:t+i}$ such that $\Pi_{t+1:t+i-1} = 2$ and $\frac{1}{2} < \rho_{t+i} < \lambda$. (It is straightforward to define an MDP, behavior policy, and target policy to create such a trajectory.) Because $\rho_{t+i} > 1 \,/\, \Pi_{t+1:t+i-1}$, then $\Pi_{t+1:t+i} = \Pi_{t+1:t+i-1} \cdot \rho_{t+i} > 1$. Thus, $H_{t:t+i} = \lambda^i$ and $H_{t:t+i-1} = \lambda^{i-1}$, and Condition 6.1 is violated because $H_{t:t+i} \,/\, H_{t:t+i-1} = \lambda \not\leq \rho_{t+i}$. □

Because Theorems 6.4 and 6.5 cannot be applied, the precise conditions under which Truncated IS converges remains an open problem. Condition 6.1 is sufficient for convergence, but probably not necessary. This is because my proofs of Theorems 6.4 and 6.5 use this assumption to guarantee that the matrix $\boldsymbol{Z}$ in Eq. (6.6) has nonnegative elements, making it straightforward to show that its row sums are sufficiently bounded for $\mathcal{M}$ to be a contraction mapping. However, $\mathcal{M}$ could remain a contraction mapping even when $\boldsymbol{Z}$ has negative elements, so long as $\|\boldsymbol{Z}\| < 1$. It could theoretically be the case that $\boldsymbol{Z}$ for Truncated IS occasionally contains negative elements but $\|\boldsymbol{Z}\|$ is still bounded enough to converge.

Nevertheless, I am able to find at least one off-policy problem for which this is not true, implying that certain initializations of the value function could ultimately cause Truncated IS to diverge.

Counterexample 6.9 (Off-Policy Truncated IS)**.** *Consider Truncated IS with $\lambda = 1$, so $H_{t:t+i} = \min(1, \Pi_{t+1:t+i})$. Assume the MDP has one state s and two actions $\{a_1, a_2\}$, the behavior policy is uniform random, and the target policy selects a_1 with probability $p \in (0, 1)$ and selects a_2 otherwise. When $p = 0.6$ and $\gamma = 0.94$, then $\|\boldsymbol{Z}\| > 1$.*

Many choices of p and γ make $\|\boldsymbol{Z}\| > 1$, but I discuss these specific values in Appendix B.2.1.

This result is surprising compared to the per-decision case, where Munos et al. (2016) showed that arbitrary trace cuts always produce a convergent algorithm. Why does the analogous result—in which $H_{t:t+i} \leq \Pi_{t+1:t+i}$ is satisfied for all time steps—not hold here? After all, clipping $H_{t:t+i}$ such that it never exceeds the IS estimate, $\Pi_{t+1:t+i}$, would be expected to simply incur bias in the return estimation.

The answer appears to be closely related to the recency heuristic. Incidentally, Condition 6.1 is the perfect off-policy analog of the weak recency heuristic (recall Definition 4.1): it implies that the eligibility of each TD error is nonincreasing on average. If, on average, the condition is violated for any transition, then the return estimate assigns a negative weight to that n-step return. This increases the contraction modulus of the return as described in Section 4.3, equivalent to increasing $\|\boldsymbol{Z}\|$ in Counterexample 6.9, and potentially pushes it to the point of divergence. However, the analysis in Section 4.3 also reveals that the weak recency heuristic can sometimes be violated without increasing the contraction modulus above 1, still allowing the operator to converge to $\boldsymbol{q}_\pi$. I therefore conclude that Condition 6.1 is *sufficient but not necessary* for $\mathcal{M}$ to converge to its fixed point.

As my next counterexample example demonstrates, violating Condition 6.1 can produce non-contractive updates even in on-policy problems.

Counterexample 6.10 (On-Policy Binary Traces)**.** *Assume the MDP has one state s and two actions $\{a_1, a_2\}$. Define a trajectory-aware method such that $H_{t:t+i} = 1$ if $A_{t+i} = a_1$ and $H_{t:t+i} = 0$ if $A_{t+i} = a_2$ (without loss of generality). Assume π and b are uniform random. When $\gamma \geq \frac{2}{3}$, then $\|\boldsymbol{Z}\| \geq 1$.*

I provide details in Appendix B.2.2. Even though $H_{t:t+i} \leq \Pi_{t+1:t+i} = 1$ always holds, non-contraction occurs. This binary method either does or does not reinforce each on-policy TD error, producing a sparse backup comprising multiple time-delayed pulses like the one from Counterexample 4.3. In this instance, the ability to examine each trajectory allows the method to strategically de-emphasize certain state-action pairs, ultimately producing a detrimental effect on learning. Notice that Condition 6.1 is indeed violated in this case because there is always some chance that $H_{t:t+i} = 1$ after $H_{t+i-1} = 0$. If I add the restriction that $H_{t+i-1} = 0 \implies H_{t:t+i} = 0$, i.e., traces are permanently cut, then convergence is reestablished by Theorem 6.4.

6.5 Recency-Bounded Importance Sampling

Theorem 6.4 guarantees convergence to $\boldsymbol{q}_\pi$ whenever Condition 6.1 holds, but I do not expect that all choices of coefficients that satisfy this condition will perform well in practice. At one extreme, if $H_{t:t+i} \leq \lambda^i \prod_{j=t+1}^{t+i} \min(1, \rho_j)$ for every sub-trajectory $\mathcal{F}_{t:t+i}$, then the method

cuts coefficients more aggressively than Retrace does; it seems unlikely that such a method would learn faster than Retrace, or other per-decision methods. At the other extreme, when $H_{t:t+i} = \Pi_{t+1:t+i}$ for every $\mathcal{F}_{t:t+i}$, the standard IS estimator is recovered, which suffers from high variance and is often ineffectual. It is therefore possible to have a method that preserves traces *too much*, to the point of being detrimental. Thus, it is important to maintain some minimum efficiency by avoiding unnecessary cuts, yet equally important to control the overall variance of the traces.

Intuitively, I want something that falls between Retrace and IS in terms of trace cutting, in order to quickly propagate credit while still managing the variance. I further hypothesize that effective trajectory-aware methods will first compute $H_{t:t+i-1} \cdot \rho_{t+i}$—i.e., the maximum trace permitted by Condition 6.1—and then apply some transformation that limits its magnitude to reduce variance. This ensures that traces are cut only as needed.

I propose one method, Recency-Bounded Importance Sampling (RBIS), which achieves this by cutting the traces only when they exceed an exponentially decaying threshold. Specifically, I define

$$H_{t:t+i} = \min(\lambda^i, H_{t+i-1} \cdot \rho_{t+i})\,. \tag{RBIS}$$

It is easy to see that RBIS always converges, by construction.

Proposition 6.11. *RBIS satisfies Condition 6.1.*

Proof. $H_{t:t+i} = \min(\lambda^i, H_{t+i-1} \cdot \rho_{t+i}) \leq H_{t+i-1} \cdot \rho_{t+i}$. □

For further insight, I unroll the recursion to obtain

$$\begin{aligned}
\min(\lambda^i, H_{t+i-1} \cdot \rho_{t+i}) &= \min\Big(\lambda^i, \min(\lambda^{i-1}, H_{t+i-2} \cdot \rho_{t+i-1}) \cdot \rho_{t+i}\Big) \\
&= \cdots \\
&= \min\Big(\lambda^i,\ \lambda^{i-1} \cdot \rho_{t+i},\ \lambda^{i-2} \cdot \rho_{t+i-1} \cdot \rho_{t+i},\ \ldots,\ \Pi_{t+1:t+i}\Big)\,. \qquad (6.9)
\end{aligned}$$

RBIS effectively takes the minimum of all past, discounted n-step IS estimates. This reveals another property of RBIS: its traces are never less than those of Retrace, because

$$\lambda^i \prod_{j=t+1}^{t+i} \min(1, \rho_j) \leq \prod_{j=t+1}^{t+i} \min(\lambda, \rho_j) \leq \lambda^{i-k} \prod_{j=t+i+1-k}^{t+i} \rho_j\,, \quad \forall\ k \in \{0, 1, \ldots, i\}\,.$$

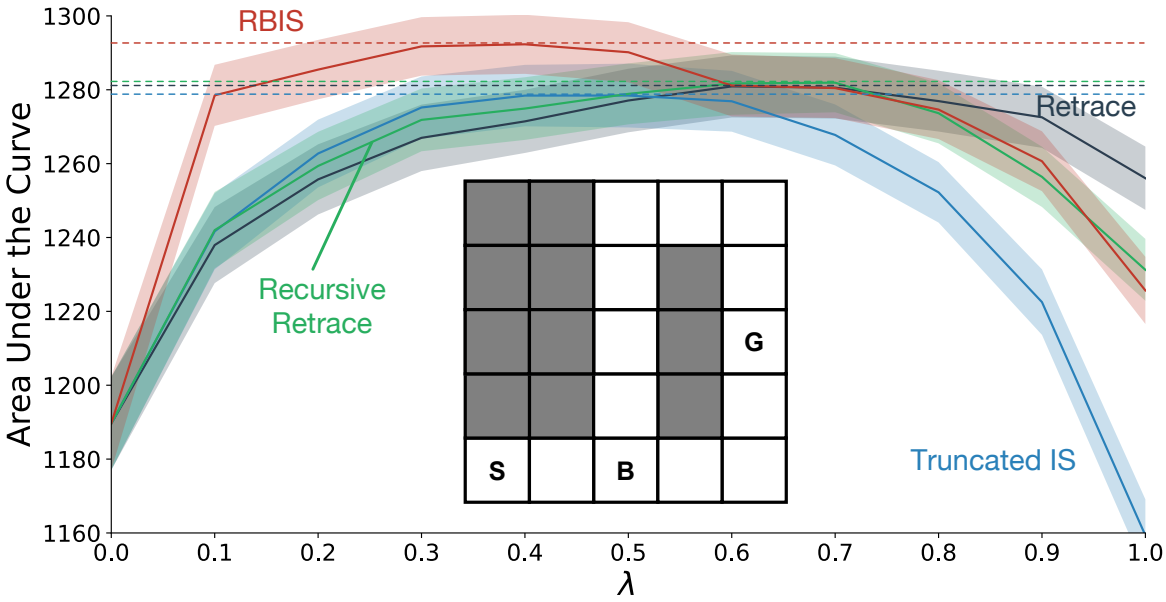


Figure 6.2: The Bifurcated Gridworld environment. The choice made at B greatly impacts the discounted return ultimately earned. The AUCs obtained by four off-policy methods are plotted across the λ-spectrum. The results are averaged over 1,000 trials with 95% confidence intervals shaded. The dashed lines mark the highest AUC achieved by each method.

Since the inequality is true for all k, Retrace's traces never exceed any of the arguments to the min function in Eq. (6.9). This achieves a method that falls somewhere between Retrace and IS in regard to trace cutting. This does not automatically mean that RBIS will outperform Retrace, since preserving the magnitude of the trace too much can lead to high variance. However, I do expect RBIS to perform well in decision-making problems in which a few critical actions largely determine the long-term outcome of an episode. In such scenarios, the agent's bottleneck to learning is its ability to assign meaningful credit to these critical actions over a potentially long time horizon.

In order to test this empirically, I construct an environment called the Bifurcated Gridworld (see Figure 6.2). This 5×5 deterministic gridworld has walls arranged such that two unequal-length paths from the start (S) to the goal (G) are available. The agent may move up, down, left, or right; taking any of these actions in the goal yields a reward of $+1$ and terminates the episode. The problem is discounted ($\gamma = 0.9$) to encourage the agent to learn the shorter path. Importantly, the action taken at the bifurcation (B) solely determines which path the agent follows, and quickly assigning credit to this state is paramount to solving the task.

I compare RBIS against Retrace, Truncated IS, and Recursive Retrace when learning this task from off-policy data. Both behavior and target policies are ϵ-greedy with respect to the action-value function, Q. The target policy uses $\epsilon = 0.1$. The behavior policy uses a piecewise schedule: $\epsilon = 1$ for the first 5 episodes and then $\epsilon = 0.2$ afterwards. The agents learn from online TD updates with tabular eligibility traces (see Appendix B.3 for pseudocode). The policies are updated only at the end of each episode, and then the discounted return obtained by a near-greedy policy ($\epsilon = 0.05$) is evaluated. The area under the curve (AUC) of each resulting learning curve is calculated, with the highest AUC achieved over a grid search of step sizes being plotted for each λ-value in Figure 6.2. I average the results over 1,000 independent trials and indicate the 95% confidence intervals by the shaded regions. In Appendix B.4, I repeat the experiment for three more gridworld topologies; the obtained results are qualitatively similar to Figure 6.2.

I make several observations regarding the results in Figure 6.2. First, the peak performance obtained by RBIS is significantly higher than that of the other three methods. This is notable because both Truncated IS and Recursive Retrace are also trajectory aware, indicating that different implementations of trajectory awareness are beneficial to varying degrees. In particular, the preservation of long-term eligibilities is not sufficient on its own to guarantee strong performance in general, as it appears that *when* and *how much* the traces are cut are important considerations as well. The role of λ as a decay hyperparameter is evidently critical for all methods to achieve their maximum performance, since $\lambda = 1$ never leads to the fastest learning. In fact, $\lambda \to 1$ is catastrophic for Truncated IS, which may be related to the divergence issue identified in Section 6.4. Finally, Retrace degrades less for larger λ, likely because it cuts traces more. Developing a trajectory-aware method that matches the robustness of RBIS but also performs better with large λ-values would be interesting for future work.

6.6 Discussion

In this chapter, I extended theory for per-decision off-policy estimators to trajectory-aware estimators. This extension allowed me to consider a broader family of algorithms, with more flexibility in obtaining off-policy corrections. Specifically, I introduced the trajectory-aware

operator $\mathcal{M}$ as a strict generalization of the per-decision operator from Munos et al. (2016) and proved a sufficient condition to ensure convergence. With this, I established the first convergence guarantees for an existing trajectory-aware method, Recursive Retrace, in the control setting. I also showed that Truncated IS sometimes violates the condition, and I provided a counterexample showing that it can potentially diverge.

I also proposed a new trajectory-aware method, RBIS, that demonstrates one instance of how trajectory awareness can be utilized for faster learning in off-policy control tasks. RBIS is able to outperform the other trajectory-aware methods that I tested in the Bifurcated Gridworld, suggesting that it possesses at least one unique property that is beneficial for long-term off-policy credit assignment. It would be interesting to search for additional beneficial properties in future work, in order to better characterize off-policy methods that reliably lead to efficient and stable learning in challenging RL environments.

I focused on convergence *in expectation*. A natural next step is to extend this theory to the stochastic-approximation algorithms used in practice. Previous results for TD learning rely primarily on the properties of the expected update, with additional conditions on the noise in the update and appropriately annealed step sizes (e.g., see Bertsekas and Tsitsiklis, 1996, Section 4.3). Similar analysis should be applicable, given that the trajectory-aware operator is a contraction mapping when Condition 6.1 is met.

Finally, another important direction is extending these methods to function approximation. The trajectory-based replay techniques described in Chapter 5 are particularly ideal for incorporating trajectory-aware methods into deep RL with experience replay.

Chapter 7

Discussions and Future Work

The thesis statement of this document was that

> **Compound returns, especially λ-returns, can be efficiently implemented in deep RL with experience replay to improve the bias-variance trade-off and increase sample efficiency in complex environments.**

I summarize my contributions in relation to this statement in Section 7.1 and offer a number of promising directions for future research in Section 7.2. I conclude this thesis in Section 7.3.

7.1 Summary of Contributions

I started by explaining how the rise of deep RL and experience replay has led to a paradigm shift in temporal credit assignment. In particular, the randomized minibatches needed to successfully train the neural networks is incompatible with classic, backward-view eligibility traces. Instead, the forward view—once a purely theoretical analysis technique—has become the principal way to achieve multistep credit assignment in deep RL. Due to the high computational cost of bootstrapping with a neural network, λ-returns and other compound returns are uncommon in deep RL. Many existing deep RL agents instead settle for 1-step or n-step returns, but these lose nice credit-assignment properties associated with more sophisticated return estimators.

In Chapters 3 and 4, I developed new theory for compound returns to motivate them over n-step returns. Under some simplifying assumptions, I proved that all compound returns have a variance-reduction property (Counterexample 3.3 showed that the variances

of multistep returns cannot be well-ordered in arbitrary MDPs without assumptions.) Additionally, long-tailed estimators like λ-returns assign credit over a larger effective horizon without adversely affecting the bias-variance trade-off. These two effects lead to higher sample efficiency in practice. For a long time, n-step returns and λ-returns were thought to be different but equivalent ways of balancing the bias-variance trade-off in RL. My theory, to the contrary, suggests that the precise construction of the return estimator has, in actuality, a large impact on how well the trade-off is satisfied.

In Chapter 5, I addressed how to efficiently implement compound returns in deep RL—with a particular focus on sequential λ-returns. Naive minibatch replay is impractical due to computational cost. Trajectory replay, while more feasible and already used in deep RL, still has drawbacks in terms of return truncation and sample correlations. I proposed an algorithm called cached-trajectory replay which mostly alleviates these three problems. My DQN(λ) algorithm, using cached-trajectory replay, is competitive with n-step DQN when playing Atari 2600 games.

In Chapter 6, I leveraged a unique opportunity presented by experience replay: computing multistep returns that are not implementable with eligibility traces and function approximation. Focusing on off-policy learning, I introduced trajectory-aware returns (a strict generalization of per-decision returns) and established their convergence properties. I showed that trajectory-aware methods generate more efficient off-policy corrections by considering multiple experiences jointly. Finally, I proposed a trajectory-aware algorithm, Recency-Bounded Importance Sampling (RBIS), which surpasses the performance of leading off-policy methods.

Collectively, the techniques developed in this thesis provide several alternatives for efficient multistep credit assignment in deep RL and experience replay. My theory and experiments provide strong support for their positive impact on the bias-variance trade-off and sample efficiency—direct outcomes of addressing temporal credit assignment more effectively. Next, I discuss further areas of improvement which I believe will lead to interesting research outcomes.

7.2 Future Directions

In this section, I discuss directions for future work: some previously mentioned and some new ones. I also discuss a recent line of work (partially my own) which tries to reconcile deep RL with backward-view eligibility traces (see Section 7.2.4), in contrast to the main approach of this thesis based on forward-view returns with experience replay.

7.2.1 Variance of Return Estimators

At the end of Section 3.2, I briefly mentioned a fascinating problem: finding the minimum-variance weights for a compound return under Assumptions 3.1 and 3.2. In my compound variance model (Prop. 3.6), only the left term impacts the variance-reduction property, because the right term is a constant for any return with a fixed contraction modulus β (this latter part can be seen at the end of the proof of Prop. 4.7). This implies that finding TD-error weights $(h_i)_{i=0}^{\infty}$ that minimize $\sum_{i=0}^{\infty}(\gamma^i h_i)^2$, subject to the constraint $\sum_{i=0}^{\infty}|h_i - h_{i+1}|\gamma^{i+1} = \beta$, will also minimize the variance. The shape of this curve poses an optimization problem which may be solvable, for example, by using the calculus of variations. It would then be interesting to see if the solution corresponds to a known compound return or a new one, and if it can be efficiently implemented in algorithms. I conjecture that the solution will be exponential, meaning that λ-returns theoretically optimize the bias-variance trade-off under Assumptions 3.1 and 3.2, but I have not formally investigated this.

The variance-reduction property is a key result in my thesis, as it provides a compelling reason to prefer compound returns over n-step returns. Although the uniformity assumptions (Assumptions 3.1 and 3.2) I made to derive this property are a significant improvement over past work that modeled variance in RL (see Appendix A.1), they are still stringent in the sense that a randomly sampled MDP will almost never satisfy them. Appendix A.2 contains experiments that show these assumptions do approximately hold in practice; along with the empirical results in Chapter 3, this suggests that compound returns often do improve sample efficiency by reducing variance. On the other hand, Counterexample 3.3 shows that the variance-reduction property does not apply to *all* MDPs. It would therefore be useful to characterize general conditions for when the variance-reduction property holds, for example by bounding the range of permissible TD-error variances instead of assuming uniform values.

A related and interesting question is whether averaging many n-step returns together reduces variance more than averaging few (e.g., λ-returns versus two-bootstrap returns). My proof of the variance-reduction property in Theorem 3.8 makes use of Jensen's inequality to show variance reduction for all possible weighted averages, but consequently it is not informative in terms of how much the variance is reduced. Although it is possible to bound the variance reduction of specific types of compound returns (like I did for λ-returns in Corollary 3.9) and then compare them, a general result would be much more powerful. The key innovation here would be some stronger analysis, perhaps based on concentration inequalities or the like, that could quantify the specific degree of variance reduction. As before, this result would be highly dependent on the variance assumptions (otherwise Counterexample 3.3 comes into play again), and so these must be chosen judiciously.

7.2.2 Convergence of TD Methods

An open question from Chapter 6 is the *online* convergence of trajectory-aware methods with eligibility traces, mirroring the algorithms tested experimentally in Section 6.5. Given that the trajectory-aware operator is a contraction mapping whenever Condition 6.1 holds, it is likely that the same condition will permit online convergence in the tabular setting. A good approach would likely be to adapt the proof of Prop. 5.2 from Bertsekas and Tsitsiklis (1996) regarding the convergence of online TD(λ). The main concern would be to show the eligibility traces are always finite, which is likely guaranteed under Condition 6.1 for discounted continuing problems as well as undiscounted episodic problems. The case of undiscounted continuing problems is less certain if $\lambda = 1$.

Counterexample 4.3 indicates a peculiarity of multistep TD learning. Nonmonotonic credit assignment like that suggested by Klopf (1972) is not conducive to learning and causes divergence in benign on-policy tabular settings. This indicates that the familiar TD error is not an absolute error signal which can be integrated arbitrarily over time, but instead a relative one which hinges on the telescopic behavior of temporally adjacent TD errors. Does this possibly suggest that there is an alternative error signal which would be a more accurate and useful analog of a secondary reinforcer in trial-and-error learning agents? Interestingly, the n-step error in Eq. (2.6) has the property that it *can* be integrated arbitrarily and still produce a convergent update (as long as the area under the curve is equal to 1, i.e., a weighted

average). However, this would merely be equivalent to a compound return and would not offer any new properties beyond what has already been discussed in this thesis. Are there alternative models of learning which would offer better properties? These questions tug deeply at the fabric of TD learning, but unfortunately I have no satisfying answers to them at this time.

7.2.3 Compound Returns for Deep RL

I proposed two main ways to implement variance-reducing compound returns in deep RL with experience replay. The first was to combine naive minibatch replay with two-bootstrap returns; this has the benefit of being computationally efficient and requiring minimal changes to existing deep RL training loops, but did not necessary deliver the full benefits of more complex returns like λ-returns. The second was to utilize cached-trajectory replay for λ-returns. These ideas were developed several years apart, and I never conducted experiments directly comparing them. To make proper comparisons between these different algorithms, Prop. 3.7 would ideally be used to find hyperparameters that achieve the same effective n-step for all tested return estimators. For instance, to make proper comparisons against 3-step DQN in Atari 2600 games (recall Section 5.4), one should test DQN(λ) with $\lambda \approx 0.67$ (Prop. 3.12 would give the exact value) and Pilar with $n_1 = 1$, $n_2 = 6$, and $c = 0.406$ (based on Table A.1). Given the beneficial properties of λ-returns outlined in Chapters 3 and 4, I suspect that DQN(λ) would perform the best. It may be the case, however, that cache blocks of greater length, and in greater numbers, are needed to sufficiently mitigate the sample correlations and observe performance gains.

A natural algorithmic combination for multistep off-policy credit assignment would be trajectory-aware returns and cached-trajectory replay. The fixed-length trajectories that compose the replay cache are ideal for making trajectory-aware off-policy adjustments that are not feasible with eligibility traces. In fact, this possibility was my original motivation for pursuing trajectory-aware estimators and developing RBIS. Unfortunately, I never implemented and tested trajectory-aware methods at scale in deep RL.

Finally, there is the matter of adaptive strategies for selecting n or λ (e.g., White and White, 2016). These were not directly discussed in this thesis, but some of their convergence properties were established in Chapter 6. The appeal of these methods is to eliminate the bur-

den of hyperparameter tuning and to enhance generality across varying environments. However, adaptive strategies must be cautiously approached. Unless they can be sensibly derived from first principles, they can be more brittle than non-adaptive methods by introducing extra latent hyperparameters in the form of schedules or other decision criteria. In my own experience, I have found the pursuit of robust, fixed-value methods to be more productive.

7.2.4 Deep RL with Eligibility Traces

I have described several ways to efficiently learn from multistep returns in deep RL, assuming that randomized experience replay is unavoidable for training. Indeed, minibatched experience replay does appear to be the best currently known training technique for deep RL. However, it comes with a number of disadvantages that are rarely addressed: a long delay between experience collection and learning, exorbitant memory consumption, and high computational cost.

In contrast, the *incremental*[20] training style commonly associated with classic TD methods and eligibility traces, where each experience is processed exactly once, is much more efficient in terms of computation and memory. From a purely theoretical view, there is also great interest in developing stable incremental methods that can learn complex representations from a continuous stream of perceptual experiences like humans do. Relatedly, the way in which modern deep RL methods store millions of high-fidelity observations and then replay them in a random order is rather biologically implausible. Thus, for both practical and scientific reasons, there is growing interest in reviving incremental RL for neural networks.

Unfortunately, incremental training is challenging to combine successfully with neural networks in practice. Despite the appeal of incremental training, minibatching seems to be important for achieving good practical performance with deep RL—at least with our current understanding. When DQN was first published, an ablation study for five games was included to highlight the benefit of experience replay (see Mnih et al., 2015, Extended Data Table 3). However, it is notable that the replay-free baselines could still learn somewhat decent policies for these games. This indicates that experience replay is not strictly necessary

[20]More recently, particularly in the deep RL setting, this training style has started to become known as *streaming* RL (Elsayed et al., 2024). Both terms are descriptive, but I use *incremental* to align with the traditional nomenclature (e.g., Sutton and Barto, 2018, Sec. 2.4) and to emphasize the cumulative, constructive nature of the sequential learning updates.

for deep RL, although it does help greatly. In any case, the massive success of DQN steered research attention away from pursuing incremental training further.

Some later papers (e.g., Mousavi et al., 2017; Young and Tian, 2019; van Hasselt et al., 2021) also included replay-free versions of DQN as baselines in their experiments—essentially just Q(0) or Q(λ) with neural networks. These experiments have generally shown that incremental algorithms can technically learn to play MinAtar or Atari games, but still not particularly well compared to DQN. This confirms that experience replay is important for sample efficiency, but not an absolute requirement for learning.

Incremental prediction also seems to be easier than incremental control. For instance, Javed et al. (2023) trained modified recurrent neural networks using TD(λ) to accurately estimate the returns obtained by pretrained policies in Atari 2600. This suggests that certain issues unique to off-policy control, such as distributional changes or the action-value function's parameterization, could be partly to blame for sample inefficiency in Q(λ) methods.

Recent work has found that various types of normalization can aid incremental training. There is growing evidence that in-network normalization like Batch Normalization (Ioffe and Szegedy, 2015) can stabilize deep RL without using target networks (e.g., Bhatt et al., 2024). Furthermore, recent works (Vasan et al., 2024; Elsayed et al., 2024) have combined other forms of normalization, including Layer Normalization (Ba et al., 2016), to successfully train incremental RL agents without target networks or experience replay. These preliminary results are encouraging, but it remains unclear how competitive these new incremental methods are against existing strong deep RL baselines, which could potentially benefit from normalization as well. Ultimately, although normalization appears to make learning easier in general, it is unlikely to fully address the root causes of instability in incremental training.

In work conducted concurrently with the writing of this thesis, my co-authors and I have developed QRC(λ) (Elelimy et al., 2025), an eligibility-trace algorithm that builds upon a long line of work on Gradient TD methods (Sutton et al., 2008, 2009; Ghiassian et al., 2020; Patterson et al., 2022). Gradient TD methods leverage the properties of stochastic gradient descent to remain convergent under off-policy sampling and function approximation, unlike classic (semi-gradient) TD methods which may diverge (see Baird, 1995; Tsitsiklis and Van Roy, 1997). When tested with the same normalized training setup proposed by Elsayed et al. (2024), QRC(λ) significantly improves performance compared to semi-gradient

baselines. However, just like normalization, enhanced off-policy stability alone is unlikely to resolve the core issues of incremental training. After all, replay algorithms like DQN face off-policy instability too, so this problem is not unique to the incremental setting.

Overall, there is more progress being made toward incremental deep RL methods than ever before. However, I expect that a truly incremental deep RL agent—one which learns steadily and reliably from a continuous stream of experiences—is still far away. Part of the immense difficulty of this problem is that it remains rather unclear why incremental learning is hard and why experience replay is so effective. Pinpointing the exact answers to both questions is essential to making progress.

For the foreseeable future, I expect minibatched experience replay to continue to prevail; it is sample-efficient, stable, low-variance, and easy to parallelize. Furthermore, even if a high-performance incremental method is eventually developed, it presumably could utilize some form of episodic replay to further increase its sample efficiency. I therefore expect experience replay will remain relevant for a long time. However, pure incremental learning is a tantalizing pursuit, and there will likely be interesting algorithmic developments in the coming years. Maybe, after all, incremental training will ultimately prove to be more efficient than experience replay in the long run, once RL is better understood.

7.3 Conclusion

Multistep credit assignment in deep RL has been relatively underexplored. With this thesis, I have tried to offer a cohesive and concise collection of efficient techniques for experience replay, with strong supporting theory and experiments. Others have also done great work in this area and I have attempted to acknowledge them wherever possible. There remain many interesting avenues for new research to pursue: with experience replay, eligibility traces, or completely new modes of learning.

I chose to focus this thesis on experience replay because, more than 10 years since the landmark DQN algorithm, it remains the most effective way to train neural networks for RL. At the same time, it is important not to get stuck in old ways of thinking. Machine learning is a fast-moving field; algorithms that are convenient for today's hardware or perform well on today's benchmarks lead us into a tempting local optimum which ultimately may

not stand the test of time. Above all else, general principles of learning and intelligence must be pursued—independently of the modern specifics of function approximation and backpropagation. Continued progress toward the development of real AI requires us to constantly question conventions and ingrained views—especially our own.

References

Ba, J., Kiros, J., and Hinton, G. E. (2016). Layer normalization. arXiv:1607.06450.

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In *International Conference on Machine Learning (ICML)*.

Baisero, A., Daley, B., and Amato, C. (2022). Asymmetric DQN for partially observable reinforcement learning. In *Uncertainty in Artificial Intelligence (UAI)*.

Bakhitov, E., Zhang, C., Sherstobitov, I., Daley, B., Ting, D., Nassif, H., and Leonenkov, S. (2024). Ad clustered user randomized trials. In *Conference on Digital Experimentation @ MIT (CODE@MIT)*. Extended abstract.

Banach, S. (1922). Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta Mathematicae*, 3(1):133–181.

Barnard, E. (1993). Temporal-difference methods and Markov models. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(2):357–365.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):834–846.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Bertsekas, D. P. (2007). *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 3rd edition.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Bhandari, J., Russo, D., and Singal, R. (2018). A finite time analysis of temporal difference learning with linear function approximation. In *Conference on Learning Theory (COLT)*.

Bhatt, A., Palenicek, D., Belousov, B., Argus, M., Amiranashvili, A., Brox, T., and Peters, J. (2024). CrossQ: Batch normalization in deep reinforcement learning for greater sample efficiency and simplicity. In *International Conference on Learning Representations (ICLR)*.

Boyan, J. and Moore, A. (1994). Generalization in reinforcement learning: Safely approximating the value function. In *Neural Information Processing Systems (NeurIPS)*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. arXiv:1606.01540.

Chebotar, Y., Hausman, K., Xia, F., Lu, Y., Irpan, A., Kumar, A., Yu, T., Herzog, A., Pertsch, K., Gopalakrishnan, K., et al. (2023). Q-Transformer: Scalable offline reinforcement learning via autoregressive Q-functions. In *Conference on Robot Learning (CoRL)*.

Chung, W., Thomas, V., Machado, M. C., and Le Roux, N. (2021). Beyond variance reduction: Understanding the true impact of baselines on policy optimization. In *International Conference on Machine Learning (ICML)*.

Cichosz, P. (1994). Truncating temporal differences: On the efficient implementation of TD(λ) for reinforcement learning. *Journal of Artificial Intelligence Research*, 2:287–318.

Daley, B. and Amato, C. (2019). Reconciling λ-returns with experience replay. In *Neural Information Processing Systems (NeurIPS)*.

Daley, B. and Amato, C. (2021). Virtual replay cache. arXiv:2112.03421.

Daley, B. and Chan, I. (2022). Adaptive tree backup algorithms for temporal-difference reinforcement learning. In *Multi-Disciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*.

Daley, B., Hickert, C., and Amato, C. (2021). Stratified experience replay: Correcting multiplicity bias in off-policy reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Extended abstract.

Daley, B., Machado, M. C., and White, M. (2024a). Demystifying the recency heuristic in temporal-difference learning. *Reinforcement Learning Journal*, 3:1019–1036.

Daley, B., Nagarajan, P., White, M., and Machado, M. C. (2025). An analysis of action-value temporal-difference methods that learn state values. *Reinforcement Learning Journal*.

Daley, B., White, M., Amato, C., and Machado, M. C. (2023). Trajectory-aware eligibility traces for off-policy reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Daley, B., White, M., and Machado, M. C. (2024b). Averaging n-step returns reduces variance in reinforcement learning. In *International Conference on Machine Learning (ICML)*.

DeepSeek-AI (2025). DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. arXiv:2501.12948.

Degris, T., White, M., and Sutton, R. S. (2012). Off-policy actor-critic. In *International Conference on Machine Learning (ICML)*.

Dennett, D. C. (1989). *The Intentional Stance*. MIT Press.

Elelimy, E., Daley, B., Patterson, A., Machado, M. C., White, A., and White, M. (2025). Deep reinforcement learning with gradient eligibility traces. *Reinforcement Learning Journal*. **Outstanding Paper Award on Theory of Reinforcement Learning**.

Elsayed, M., Vasan, G., and Mahmood, A. R. (2024). Streaming deep reinforcement learning finally works. arXiv:2410.14606.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*.

Ghiassian, S., Patterson, A., Garg, S., Gupta, D., White, A., and White, M. (2020). Gradient temporal-difference learning with regularized corrections. In *International Conference on Machine Learning (ICML)*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Gupta, D., Jordan, S. M., Chaudhari, S., Liu, B., Thomas, P. S., and da Silva, B. C. (2024). From past to future: Rethinking eligibility traces. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*.

Harb, J. and Precup, D. (2016). Investigating recurrence and eligibility traces in deep Q-networks. In *NeurIPS Deep Reinforcement Learning Workshop*.

Harutyunyan, A., Bellemare, M. G., Stepleton, T., and Munos, R. (2016). Q(λ) with off-policy corrections. In *International Conference on Algorithmic Learning Theory (ALT)*.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., van Hasselt, H., and Silver, D. (2018). Distributed prioritized experience replay. In *International Conference on Learning Representations (ICLR)*.

Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., and Araújo, J. G. (2022). CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*.

Ionides, E. L. (2008). Truncated importance sampling. *Journal of Computational and Graphical Statistics*, 17(2):295–311.

Jaakkola, T., Jordan, M., and Singh, S. (1993). Convergence of stochastic iterative dynamic programming algorithms. In *Neural Information Processing Systems (NeurIPS)*.

Javed, K., Shah, H., Sutton, R. S., and White, M. (2023). Scalable real-time recurrent learning using columnar-constructive networks. *Journal of Machine Learning Research*, 24(256):1–34.

Kahn, H. and Marshall, A. W. (1953). Methods of reducing sample size in Monte Carlo computations. *Journal of the Operations Research Society of America*, 1(5):263–278.

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. (2018). Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

Kearns, M. J. and Singh, S. (2000). Bias-variance error bounds for temporal difference updates. In *Conference on Learning Theory (COLT)*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Klopf, A. H. (1972). Brain function and adaptive systems: A heterostatic theory. Technical report, Air Force Cambridge Research Laboratories.

Klopf, A. H. (1982). *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence*. Hemisphere Publishing Corporation.

Konidaris, G., Niekum, S., and Thomas, P. S. (2011). TD_γ: Re-evaluating complex backups in temporal difference learning. In *Neural Information Processing Systems (NeurIPS)*.

Kozuno, T., Tang, Y., Rowland, M., Munos, R., Kapturowski, S., Dabney, W., Valko, M., and Abel, D. (2021). Revisiting Peng's Q(λ) for modern reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Neural Information Processing Systems (NeurIPS)*.

LeCun, Y., Bengio, Y., and Hinton, G. E. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

Lin, L.-J. (1991). Programming robots using reinforcement learning and teaching. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and reaching. *Machine Learning*, 8:293–321.

Lyu, X., Baisero, A., Xiao, Y., Daley, B., and Amato, C. (2023). On centralized critics in multi-agent reinforcement learning. *Journal of Artificial Intelligence Research*, 77:295–354.

Lyu, X., Xiao, Y., Daley, B., and Amato, C. (2021). Contrasting centralized and decentralized critics in multi-agent reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. **Best Paper Award Finalist**.

Mahmood, A. R., Yu, H., and Sutton, R. S. (2017). Multi-step off-policy learning without importance sampling ratios. arXiv:1702.03006.

McCarthy, J. (1997). What is artificial intelligence? `http://jmc.stanford.edu/artificial-intelligence/what-is-ai/index.html`.

McCarthy, J., Minsky, M. L., Rochester, N., and Shannon, C. E. (1955). A proposal for the Dartmouth summer research project on artificial intelligence. `http://jmc.stanford.edu/articles/dartmouth/dartmouth.pdf`.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165.

Minsky, M. L. (1961). Steps toward artificial intelligence. In *Proceedings of the Institute of Radio Engineers (IRE)*.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A. J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., et al. (2017). Learning to navigate in complex environments. In *International Conference on Learning Representations (ICLR)*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. arXiv:1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Mousavi, S. S., Schukat, M., Howley, E., and Mannion, P. (2017). Applying Q(λ)-learning in deep reinforcement learning to play Atari games. In *AAMAS Adaptive Learning Agents Workshop*.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. G. (2016). Safe and efficient off-policy reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.

Nguyen, H., Daley, B., Song, X., Amato, C., and Platt, R. (2020). Belief-grounded networks for accelerated robot learning under partial observability. In *Conference on Robot Learning (CoRL)*.

OpenAI (2018). OpenAI Five. `https://blog.openai.com/openai-five/`.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. (2022). Training language models to follow instructions with human feedback. In *Neural Information Processing Systems (NeurIPS)*.

Patterson, A., White, A., and White, M. (2022). A generalized projected Bellman error for off-policy value estimation in reinforcement learning. *Journal of Machine Learning Research*, 23(145):1–61.

Peng, J. and Williams, R. J. (1996). Incremental multi-step Q-Learning. *Machine Learning*, 22:226–232.

Precup, D., Sutton, R. S., and Singh, S. (2000). Eligibility traces for off-policy policy evaluation. In *International Conference on Machine Learning (ICML)*.

Riedmiller, M. (2005). Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning (ECML)*.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407.

Rowland, M., Dabney, W., and Munos, R. (2020). Adaptive trade-offs in off-policy learning. In *Artificial Intelligence and Statistics (AISTATS)*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, University of Cambridge.

Russell, S. J. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering Atari, Go, Chess and Shogi by planning with a learned model. *Nature*, 588(7839):604–609.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In *International Conference on Machine Learning (ICML)*.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. arXiv:1707.06347.

Schwarzer, M., Ceron, J. S. O., Courville, A., Bellemare, M. G., Agarwal, R., and Castro, P. S. (2023). Bigger, better, faster: Human-level Atari with human-level efficiency. In *International Conference on Machine Learning (ICML)*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

Sutton, R. S. (1989). Implementation details of the TD(λ) procedure for the case of vector predictions and backpropagation. Technical report, GTE Laboratories.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *International Conference on Machine Learning (ICML)*.

Sutton, R. S. (2020). John McCarthy's definition of intelligence. *Journal of Artificial General Intelligence*, 11(2):66–67.

Sutton, R. S. and Barto, A. G. (1981). Toward a modern theory of adaptive networks: expectation and prediction. *Psychological Review*, 88(2):135.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, 1st edition.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition.

Sutton, R. S., Bowling, M., and Pilarski, P. M. (2022). The Alberta plan for AI research. arXiv:2208.11173.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *International Conference on Machine Learning (ICML)*.

Sutton, R. S., Mahmood, A. R., Precup, D., and Hasselt, H. (2014). A new Q(λ) with interim forward view and Monte Carlo equivalence. In *International Conference on Machine Learning (ICML)*.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Sutton, R. S., Szepesvári, C., and Maei, H. R. (2008). A convergent $O(n)$ algorithm for off-policy temporal-difference learning with linear function approximation. In *Neural Information Processing Systems (NeurIPS)*.

Tang, Y., Munos, R., Rowland, M., Pires, B. Á., Dabney, W., and Bellemare, M. G. (2022). The nature of temporal difference errors in multi-step distributional reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*.

Tang, Y., Rowland, M., Munos, R., Pires, B. Á., and Dabney, W. (2024). Off-policy distributional Q(λ): Distributional RL without importance sampling. arXiv:2402.05766.

Tesauro, G. (1991). Practical issues in temporal difference learning. In *Neural Information Processing Systems (NeurIPS)*.

Thomas, P. S., Niekum, S., Theocharous, G., and Konidaris, G. (2015). Policy evaluation using the Ω-return. In *Neural Information Processing Systems (NeurIPS)*.

Thorndike, E. L. (1898). Animal intelligence: An experimental study of the associative processes in animals. *Psychological Review Monograph Supplements*, 2:1–109.

Thorndike, E. L. (1911). *Animal Intelligence: Experimental Studies*. The Macmillan Company.

Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and Q-Learning. *Machine Learning*, 16(3):185–202.

Tsitsiklis, J. N. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460.

Uchibe, E. and Doya, K. (2004). Competitive-cooperative-concurrent reinforcement learning with importance sampling. In *International Conference on Simulation of Adaptive Behavior (SAB)*.

van Hasselt, H., Madjiheurem, S., Hessel, M., Silver, D., Barreto, A., and Borsa, D. (2021). Expected eligibility traces. In *AAAI Conference on Artificial Intelligence (AAAI)*.

van Seijen, H. (2016). Effective multi-step temporal-difference learning for non-linear function approximation. arXiv:1608.05151.

van Seijen, H., Mahmood, A. R., Pilarski, P. M., Machado, M. C., and Sutton, R. S. (2016). True online temporal-difference learning. *Journal of Machine Learning Research*, 17(145):5057–5096.

van Seijen, H. and Sutton, R. S. (2014). True online TD(λ). In *International Conference on Machine Learning (ICML)*.

Vasan, G., Elsayed, M., Azimi, S. A., He, J., Shahriar, F., Bellinger, C., White, M., and Mahmood, A. R. (2024). Deep policy gradient methods without batch updates, target networks, or replay buffers. In *Neural Information Processing Systems (NeurIPS)*.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudik, M., Kapturowski, S., Lanctot, M., Piot, B., Sylvain, L., Baker, L., Xu, K., Li, X., Mnih, V., Silver, D., et al. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2017). Sample efficient actor-critic with experience replay. In *International Conference on Learning Representations (ICLR)*.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, University of Cambridge.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292.

Wawrzyński, P. (2009). Real-time reinforcement learning by sequential actor-critics and experience replay. *Neural Networks*, 22(10):1484–1497.

Wawrzyński, P. and Pacut, A. (2007). Truncated importance sampling for reinforcement learning with experience replay. In *International Multiconference on Computer Science and Information Technology (IMCSIT)*.

White, M. and White, A. (2016). A greedy approach to adapting the trace parameter for temporal difference learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Wurman, P. R., Barrett, S., Kawamoto, K., MacGlashan, J., Subramanian, K., Walsh, T. J., Capobianco, R., Devlic, A., Eckert, F., Fuchs, F., et al. (2022). Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228.

Young, K. and Tian, T. (2019). MinAtar: An Atari-inspired testbed for thorough and reproducible reinforcement learning experiments. arXiv:1903.03176.

Yu, H., Mahmood, A. R., and Sutton, R. S. (2018). On generalized Bellman equations and temporal-difference learning. *Journal of Machine Learning Research*, 19(1):1864–1912.

Appendix A

Variance Reduction via Averaging n-step Returns

A.1 Variance Assumptions

Assumptions 3.1 and 3.2 are a significant relaxation and generalization of the assumptions made by the n-step variance model of Konidaris et al. (2011, Sec. 3). I show this by starting from their original assumptions and describing the steps taken to obtain my new assumptions.

Konidaris et al. begin by expanding the variance of an n-step return in the following way:

$$\begin{aligned}
&\mathrm{Var}[G_t^{(n)} \mid S_t] \\
&\quad = \mathrm{Var}[G_t^{(n-1)} + \gamma^{n-1}\delta_{t+n-1} \mid S_t] \\
&\quad = \mathrm{Var}[G_t^{(n-1)} \mid S_t] + \gamma^{2(n-1)}\mathrm{Var}[\delta_{t+n-1} \mid S_t] + 2\,\mathrm{Cov}[G_t^{(n-1)},\ \gamma^{n-1}\delta_{t+n-1} \mid S_t]\,. \qquad \text{(A.1)}
\end{aligned}$$

The covariance term is equivalent to

$$\begin{aligned}
\mathrm{Cov}[G_t^{(n-1)},\ \gamma^{n-1}\delta_{t+n-1} \mid S_t] &= \mathrm{Cov}[G_t^{(n-1)},\ G_t^{(n)} - G_t^{(n-1)} \mid S_t] \\
&= \mathrm{Cov}[G_t^{(n-1)},\ G_t^{(n)} \mid S_t] - \mathrm{Cov}[G_t^{(n-1)},\ G_t^{(n-1)} \mid S_t] \\
&= \mathrm{Cov}[G_t^{(n-1)},\ G_t^{(n)} \mid S_t] - \mathrm{Var}[G_t^{(n-1)} \mid S_t]\,. \qquad \text{(A.2)}
\end{aligned}$$

With the rationale that $G_t^{(n-1)}$ and $G_t^{(n)}$ are generated from the same trajectory and therefore highly correlated, Konidaris et al. assume that

$$\mathrm{Cov}[G_t^{(n-1)},\ G_t^{(n)} \mid S_t] \approx \mathrm{Var}[G_t^{(n-1)} \mid S_t]\,, \qquad \text{(A.3)}$$

which makes Eq. (A.2) approximately zero. Consequently, Eq. (A.1) becomes

$$\mathrm{Var}[G_t^{(n)} \mid S_t] \approx \mathrm{Var}[G_t^{(n-1)} \mid S_t] + \gamma^{2(n-1)}\mathrm{Var}[\delta_{t+n-1} \mid S_t]\,. \qquad \text{(A.4)}$$

The authors additionally assume that the variance of each TD error is the same, which is the same as Assumption 3.1.

Assumption 3.1 (Uniform TD-error variance)**.** $\text{Var}[\delta_{t+i} \mid S_t] \approx \kappa,\ \forall\ i \geq 0.$

Hence, Eq. (A.4) becomes $\text{Var}[G_t^{(n)} \mid S_t] \approx \text{Var}[G_t^{(n-1)} \mid S_t] + \gamma^{2(n-1)}\kappa\,.$ Since $\kappa = \text{Var}[\delta_t \mid S_t] = \text{Var}[G_t^{(1)} \mid S_t]$, unrolling the recursion gives the final n-step variance model:

$$\text{Var}[G_t^{(n)} \mid S_t] \approx \sum_{i=0}^{n-1} \gamma^{2i}\kappa\,. \tag{A.5}$$

This completes the derivation from Konidaris et al. (2011, Sec. 3), where the two major assumptions are Eq. (A.3) and Assumption 3.1.

Notice, however, that Eq. (A.5) could be obtained more simply by assuming that TD errors are uncorrelated—in fact, this is equivalent to assuming Eq. (A.3) holds. One way to see this is by decomposing the n-step return into a sum of TD errors and applying standard variance rules, assuming no correlations between the random variables:

$$\text{Var}[G_t^{(n)} \mid S_t] = \text{Var}[G_t^{(n)} - V(S_t) \mid S_t] = \text{Var}\left[\sum_{i=0}^{n-1} \gamma^i \delta_{t+i} \;\middle|\; S_t\right] = \sum_{i=0}^{n-1} \gamma^{2i} \text{Var}[\delta_{t+i} \mid S_t] = \sum_{i=0}^{n-1} \gamma^{2i}\kappa\,,$$

which is identical to Eq. (A.5). In my work, I directly make this uniform-correlation assumption, while generalizing it with an arbitrary correlation coefficient, $\rho \in [0, 1]$. This gives me Assumption 3.2. Note that $\rho \geq 0$ because an infinite number of TD errors cannot be negatively correlated simultaneously.

Assumption 3.2 (Uniform TD-error correlation)**.** $\text{Corr}[\delta_{t+i}, \delta_{t+j} \mid S_t] \approx \rho,\ \forall\ i \geq 0, \forall\ j \neq i.$

To summarize, my variance model makes Assumptions 3.1 and 3.2, which are equivalent to the assumptions made by Konidaris et al. (2011) when $\rho = 0$. Since $\text{Var}[\delta_{t+i} \mid S_t] = \text{Cov}[\delta_{t+i}, \delta_{t+i} \mid S_t]$, I am able to combine Assumptions 3.1 and 3.2 into a single, concise expression—recall Eq. (3.3).

Although the assumptions here may not always hold in practice, they would be difficult to improve without invoking information about the MDP's transition function or reward function. I show in Appendix A.2 that my derived variances based on these assumptions are nevertheless consistent with empirical data in real environments.

A.2 How Realistic Is the Proposed Variance Model?

My assumptions state that all TD errors have uniform variance and are equally correlated to one another. Since these assumptions may be violated in practice, it is informative to test how well my n-step variance model (Prop. 3.4) compares to the true n-step variances in several examples.

I consider three environments: the 19-state random walk from Section 3.4, a 4×3 gridworld (Russell and Norvig, 2009, Fig. 17.1), and a 10×8 gridworld (Sutton and Barto, 2018, Fig. 7.4). I choose these environments because they have known dynamics and are small enough to exactly calculate v_π with dynamic programming. The two gridworlds are made to be stochastic because each of the four moves (up, down, left, right) succeeds with only probability 80%; otherwise, the move is rotated by 90 degrees in either direction with probability 10% each. I let the agent execute a uniform-random behavior policy for all environments.

To make the results agnostic to any particular learning algorithm, I use v_π to compute the TD errors. I apply a discount factor of $\gamma = 0.99$ to the 10×8 gridworld (otherwise $v_\pi(s)$ would be constant for all $s \in \mathcal{S}$ due to the single nonzero reward) and leave the other two environments undiscounted. I then measure the variance of the n-step returns originating from the initial state of each environment, for $n \in \{1, \dots, 21\}$. Figure A.1 shows these variances plotted as a function of n and averaged over 10,000 episodes. The best-case (optimistic, $\rho = 0$) and worst-case (pessimistic, $\rho = 1$) variances predicted by the n-step model, assuming that $\kappa = \mathrm{Var}[\delta_0 \mid S_0]$, are also indicated by dashed lines.

For all of the environments, the measured n-step variances always remain within the lower and upper bounds predicted by Prop. 3.4. These results show that my n-step variance model can still make useful variance predictions even when my assumptions do not hold. The variances grow roughly linearly as a function of n, corresponding more closely to the linear behavior of the optimistic, uncorrelated case than the quadratic behavior of the pessimistic, maximally correlated case. This further suggests that the majority of TD-error pairs are weakly correlated in practice, which makes sense because temporally distant pairs are unlikely to be strongly related.

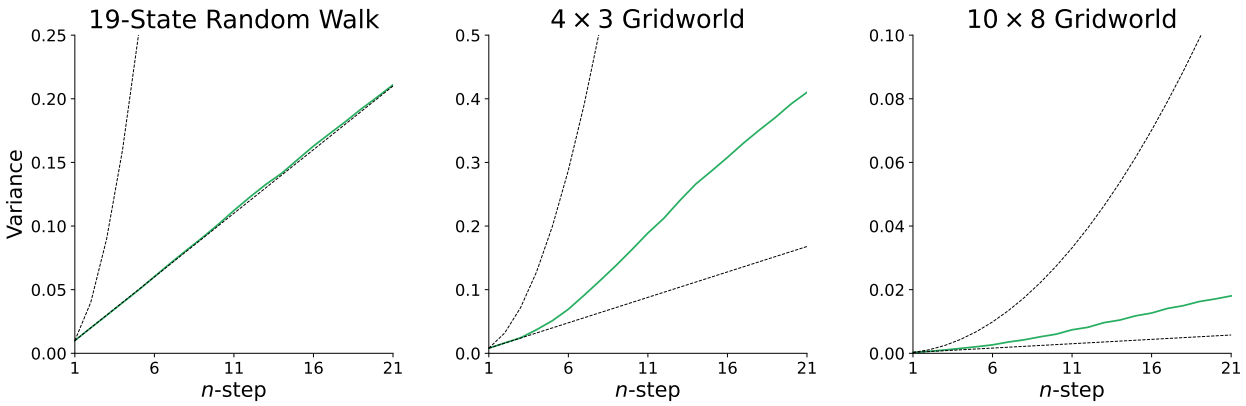


Figure A.1: Variances of the n-step returns originating from the initial state in three environments. The solid green line indicates the true variance while the dashed black lines indicate the lower and upper bounds predicted by my n-step variance model (Prop. 3.4).

A.3 Proofs

In this section, I include all proofs that were omitted from Chapter 3 for clarity.

A.3.1 Proof of Lemma 3.5

Lemma 3.5. *Any compound error can be written as a weighted sum of TD errors:*

$$G_t^{\boldsymbol{c}} - V(S_t) = \sum_{i=0}^{\infty} \gamma^i h_i \delta_{t+i}\,, \;\; \textit{where } h_i \coloneqq \sum_{n=i+1}^{\infty} c_n\,.$$

Proof. I decompose the compound error into a weighted average of n-step errors, and then decompose those n-step errors into weighted sums of TD errors using Eq. (2.6):

$$\begin{aligned}
G_t^{\boldsymbol{c}} - V(S_t) &= \left(\sum_{n=1}^{\infty} c_n G_t^{(n)}\right) - V(S_t) \\
&= \sum_{n=1}^{\infty} c_n \left(G_t^{(n)} - V(S_t)\right) \\
&= \sum_{n=1}^{\infty} \sum_{i=0}^{n-1} c_n \gamma^i \delta_{t+i} \\
&= \sum_{i=0}^{\infty} \sum_{n=i+1}^{\infty} c_n \gamma^i \delta_{t+i} \\
&= \sum_{i=0}^{\infty} \gamma^i h_i \delta_{t+i}\,,
\end{aligned}$$

which completes the lemma. □

A.3.2 Proof of Corollary 3.9

Corollary 3.9 (Variance reduction of λ-return)**.** *Under Assumptions 3.1 and 3.2, for all $\gamma \in [0, 1]$, the magnitude of variance reduction for a λ-return is bounded such that*

$$0 \leq \mathrm{Var}[G_t^{(n)} \mid S_t] - \mathrm{Var}[G_t^{\lambda} \mid S_t] \leq (1-\rho)\frac{\lambda}{1-\lambda^2}\kappa\,.$$

Proof. First, the left-hand side of the inequality follows immediately from Theorem 3.8. For the right-hand side, I make the observation that, for any compound return,

$$\sum_{i=0}^{\infty}\gamma^i h_i = \sum_{i=0}^{\infty}\sum_{n=i+1}^{\infty}\gamma^i c_n = \sum_{n=1}^{\infty}\sum_{i=0}^{n-1}\gamma^i c_n = \sum_{n=1}^{\infty} c_n \frac{1-\gamma^n}{1-\gamma} = \frac{1-\sum_{n=1}^{\infty} c_n\gamma^n}{1-\gamma} = \frac{1-\beta}{1-\gamma}\,.$$

This allows me to rewrite the compound variance model in Prop. 3.6 as

$$\mathrm{Var}[G_t^{\boldsymbol{c}} \mid S_t] = (1-\rho)\sum_{i=0}^{\infty}\gamma^{2i}h_i^2\kappa + \rho\sum_{i=0}^{\infty}\sum_{j=0}^{\infty}\gamma^{i+j}h_i h_j\kappa = (1-\rho)\sum_{i=0}^{\infty}\gamma^{2i}h_i^2\kappa + \rho\left(\frac{1-\beta}{1-\gamma}\right)^2\kappa\,.$$

The second term is identical for all returns with a fixed contraction modulus β, as is assumed in Theorem 3.8. Thus, by subtracting the variance of both returns, I get the following expression for variance reduction (positive means more variance reduction):

$$\mathrm{Var}[G_t^{(n)} \mid S_t] - \mathrm{Var}[G_t^{\boldsymbol{c}} \mid S_t] = (1-\rho)\left(\Gamma_2(n) - \sum_{i=0}^{\infty}\gamma^{2i}h_i^2\right)\kappa\,.$$

This reveals that the variance reduction of a compound return is proportional to $1-\rho$, affirming the result of Theorem 3.8 that variance reduction occurs whenever $\rho < 1$. Additionally, this variance reduction is proportional to the critical quantity $\Gamma_2(n) - \sum_{i=0}^{\infty}\gamma^{2i}h_i^2$, a function of the weights in the compound return. Substituting the weights for a λ-return, this critical quantity becomes

$$\Gamma_2(n) - \frac{1}{1-(\gamma\lambda)^2}\,.$$

This gap is monotonically increasing in γ and attains its maximum value as $\gamma \to 1$, i.e., the maximum-variance case of undiscounted rewards.[21] Taking the limit as $\gamma \to 1$ makes the possible variance reduction proportional to

$$n - \frac{1}{1-\lambda^2} = \frac{1}{1-\lambda} - \frac{1}{1-\lambda^2} = \frac{\lambda}{1-\lambda^2}\,,$$

[21] To see this, make the substitution $\lambda = (1-\gamma^{n-1})\,/\,(1-\gamma^n)$ from Prop. 3.12 (because the effective n-step is the same) and analyze the behavior as $\gamma \to 1$.

where the identity $n = 1/(1-\lambda)$ comes from Prop. 3.12, the effective n-step of the λ-return. This gives the final upper bound:

$$\mathrm{Var}[G_t^{(n)} \mid S_t] - \mathrm{Var}[G_t^{\lambda} \mid S_t] \leq (1-\rho)\frac{\lambda}{1-\lambda^2}\kappa\,,$$

which completes the inequality. □

A.3.3 Finite-Time Analysis (Proof of Theorem 3.10)

In this section, I prove Theorem 3.10 to establish a finite-time bound on the performance of multistep TD learning. I derive the bound in terms of the return's contraction modulus and variance, allowing me to invoke Theorem 3.8 and show an improved convergence rate.

At each iteration, compound TD learning updates the current parameters $\boldsymbol{w}_t \in \mathbb{R}^{N_{\boldsymbol{w}}}$ according to

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_t g_t^{\boldsymbol{c}}(\boldsymbol{w}_t)\,, \tag{A.6}$$

where $g_t^{\boldsymbol{c}}(\boldsymbol{w}) \coloneqq (G_t^{\boldsymbol{c}} - \boldsymbol{x}_t^\top \boldsymbol{w})\boldsymbol{x}_t$ and $G_t^{\boldsymbol{c}}$ is a compound return (or an n-step return). A value-function estimate for any state s is obtained by evaluating the dot product of the parameters and the state's corresponding feature vector. Following Bhandari et al. (2018), I assume that $\|\boldsymbol{X}(s)\|_2^2 \leq 1$, $\forall\ s \in \mathcal{S}$. This can be guaranteed in practice by normalizing the features, and is therefore not a strong assumption.

In the prediction setting, the agent's behavior policy is fixed such that the MDP can be cast as an MRP, where $r(s, s')$ denotes the expected reward earned when transitioning from state s to state s'. I adopt the i.i.d. state model from Bhandari et al. (2018, Sec. 3) and generalize it for multistep TD updates.

Assumption A.1 (i.i.d. state model)**.** *Assume the MRP under the policy π is ergodic. Let $\boldsymbol{d} \in \mathbb{R}^{|\mathcal{S}|}$ be the MRP's unique stationary distribution. Each iteration of Eq.* (A.6) *is calculated by first sampling a random initial state $S_{t,0} \sim d(\cdot)$ and then sampling a trajectory of subsequent states $S_{t,i+1} \sim \Pr(\cdot \mid S_{t,i})$, $\forall\ i \geq 0$.*

That is, a state $S_{t,0}$ sampled from the steady-state MRP forms the root node for the following trajectory, $S_{t,1}, S_{t,2}, \ldots$, that is generated according to the MRP transition function. Notably, this setting parallels the experience-replay setting utilized by deep RL agents.

To facilitate my analysis, I decompose the compound TD updates into weighted averages of n-step TD updates, where each n-step update has the form

$$g_t^n(\boldsymbol{w}) \coloneqq \left(G_t^{(n)} - \boldsymbol{x}_t^\top \boldsymbol{w}\right) \boldsymbol{x}_t \,.$$

This allows me to conveniently express a compound TD update as

$$g_t^{\boldsymbol{c}}(\boldsymbol{w}) \coloneqq \sum_{n=1}^{\infty} c_n g_t^n(\boldsymbol{w}) \,.$$

My proofs also make use of the *expected* n-step TD update:

$$\bar{g}^n(\boldsymbol{w}) \coloneqq \sum_{s_0 \in \mathcal{S}} \sum_{\tau \in \mathcal{S}^{n-1}} d(s_0) \Pr(\tau \mid s_0) \left(G^{(n)}(s_0, \tau, \boldsymbol{w}) - \boldsymbol{x}(s_0)^\top \boldsymbol{w}\right) \boldsymbol{x}(s_0)$$

where $(s_1, s_2, \dots) = \tau$ and $G^{(n)}(s_0, \tau, \boldsymbol{w}) \coloneqq r(s_0, s_1) + \cdots + \gamma^{n-1} r(s_{n-1}, s_n) + \gamma^n \boldsymbol{x}(s_n)^\top \boldsymbol{w}$ is the n-step return generated from (s_0, τ).

For brevity, let $R_i \coloneqq r(S_{t,i}\,, S_{t,i+1})$ and $\boldsymbol{x}_i \coloneqq \boldsymbol{x}(S_{t,i})$ be random variables sampled according to Assumption A.1. I more conveniently write the expected n-step TD update as

$$\bar{g}^n(\boldsymbol{w}) = \mathbb{E}\big[\boldsymbol{x}_0(R_0 + \gamma R_1 + \cdots + \gamma^{n-1} R_{n-1})\big] + \mathbb{E}\big[\boldsymbol{x}_0(\gamma^n \boldsymbol{x}_n - \boldsymbol{x}_0)^\top\big]\, \boldsymbol{w} \,. \tag{A.7}$$

The expected compound TD update easily follows as the weighted average

$$\bar{g}^{\boldsymbol{c}}(\boldsymbol{w}) \coloneqq \sum_{n=1}^{\infty} c_n \bar{g}^n(\boldsymbol{w}) \,.$$

Finally, let $\boldsymbol{w}^*$ be the fixed point of the compound TD update: i.e., $\bar{g}^{\boldsymbol{c}}(\boldsymbol{w}^*) = 0$. This fixed point always exists and is unique because the projected Bellman operator is a contraction mapping (Tsitsiklis and Van Roy, 1997), and therefore so is any weighted average of the n-iterated operators.

Before I prove Theorem 3.10, I must introduce two lemmas. The first establishes a lower bound on the angle between the expected TD update and the direction toward the fixed point.

Lemma A.2. *Define the diagonal matrix* $\boldsymbol{D} \coloneqq \operatorname{diag}(\boldsymbol{d})$. *For any* $\boldsymbol{w} \in \mathbb{R}^{N_{\boldsymbol{w}}}$,

$$(\boldsymbol{w}^* - \boldsymbol{w})^\top \bar{g}^{\boldsymbol{c}}(\boldsymbol{w}) \geq (1 - \beta) \|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\boldsymbol{w}}\|_{\boldsymbol{D}}^2 \,.$$

Proof. Let $\xi_i \coloneqq v_{\boldsymbol{w}^*}(S_{t,i}) - v_{\boldsymbol{w}}(S_{t,i}) = (\boldsymbol{w}^* - \boldsymbol{w})^\top \boldsymbol{x}_i$ for $i \geq 0$. By stationarity, each ξ_i is a correlated random variable with the same marginal distribution. Because $S_{t,0}$ is drawn from the stationary distribution, I have $\mathbb{E}[\xi_i^2] = \|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\boldsymbol{w}}\|_{\boldsymbol{D}}^2$.

From Eq. (A.7), I show

$$\bar{g}^{\boldsymbol{c}}(\boldsymbol{w}) = \bar{g}^{\boldsymbol{c}}(\boldsymbol{w}) - \bar{g}^{\boldsymbol{c}}(\boldsymbol{w}^*) = \sum_{n=1}^{\infty} c_n \mathbb{E}\big[\boldsymbol{x}_0(\gamma^n \boldsymbol{x}_n - \boldsymbol{x}_0)^\top (\boldsymbol{w} - \boldsymbol{w}^*)\big]$$
$$= \sum_{n=1}^{\infty} c_n \mathbb{E}[\boldsymbol{x}_0(\xi_0 - \gamma^n \xi_n)] \,.$$

It follows that

$$(\boldsymbol{w}^* - \boldsymbol{w})^\top \bar{g}^{\boldsymbol{c}}(\boldsymbol{w}) = \sum_{n=1}^{\infty} c_n \mathbb{E}[\xi_0(\xi_0 - \gamma^n \xi_n)]$$
$$= \mathbb{E}\big[\xi_0^2\big] - \sum_{n=1}^{\infty} c_n \gamma^n \mathbb{E}[\xi_0 \xi_n]$$
$$\geq \left(1 - \sum_{n=1}^{\infty} c_n \gamma^n\right) \mathbb{E}\big[\xi_0^2\big]$$
$$= (1-\beta)\|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\boldsymbol{w}}\|_{\boldsymbol{D}}^2 \,.$$

The inequality uses the Cauchy-Schwarz inequality along with the fact that every ξ_i has the same marginal distribution: thus, $\mathbb{E}[\xi_0 \xi_i] \leq \sqrt{\mathbb{E}[\xi_0^2]}\sqrt{\mathbb{E}[\xi_i^2]} = \mathbb{E}[\xi_0^2]$. □

The next lemma establishes a bound on the second moment of the squared norm of the TD update in terms of the contraction modulus β and the variance σ^2 of the compound return.

Lemma A.3. *Define* $\Delta^* \coloneqq \|\boldsymbol{r}\|_\infty + (1+\gamma)\|\boldsymbol{w}^*\|_\infty$ *and* $C \coloneqq \Delta^* / (1-\gamma)$. *For any* $\boldsymbol{w} \in \mathbb{R}^{N_{\boldsymbol{w}}}$,

$$\mathbb{E}[\|g_t(\boldsymbol{w})\|_2^2] \leq 2(1-\beta)^2 C^2 + 2\sigma^2 + 4(1+\beta)\|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\boldsymbol{w}}\|_{\boldsymbol{D}}^2 \,.$$

Proof. Let $\delta_{t,i}^* \coloneqq R_i + \gamma \boldsymbol{x}_{i+1}^\top \boldsymbol{w}^* - \boldsymbol{x}_i^\top \boldsymbol{w}^*$ and note that $\left|\delta_{t,i}^*\right| \leq \Delta^*$ for all $i \geq 0$ by the triangle inequality and the bounded-feature assumption. Denote the n-step and compound errors constructed from $\boldsymbol{w}^*$ by $\delta_t^{(n)} \coloneqq \sum_{i=0}^{n-1} \gamma^i \delta_{t,i}^*$ and $\delta_t^{\boldsymbol{c}} \coloneqq \sum_{n=1}^{\infty} c_n \delta_t^{(n)}$, respectively. I have

$$\mathbb{E}\big[\|g_t(\boldsymbol{w}^*)\|_2^2\big] = \mathbb{E}\big[\|\delta_t^{\boldsymbol{c}} \boldsymbol{x}_0\|_2^2\big] \leq \mathbb{E}\big[(\delta_t^{\boldsymbol{c}})^2\big] = \mathbb{E}[\delta_t^{\boldsymbol{c}}]^2 + \sigma^2 \,, \tag{A.8}$$

where the inequality follows from the assumption that $\|\boldsymbol{x}_0\|_2^2 \leq 1$. The absolute value of the expectation can be bounded using the triangle inequality:

$$\left|\mathbb{E}[\delta_t^{\boldsymbol{c}}]\right| = \left|\mathbb{E}\left[\sum_{n=1}^{\infty} c_n \delta_t^{(n)}\right]\right| \leq \sum_{n=1}^{\infty} c_n \, \Gamma_1(n) \, \Delta^* = \frac{1-\beta}{1-\gamma}\Delta^* = (1-\beta)C \,. \tag{A.9}$$

The identity $\sum_{n=1}^{\infty} c_n \Gamma_1(n) = (1-\beta) \,/\, (1-\gamma)$ comes from Eq. (3.8). Eqs. (A.8) and (A.9) imply

$$\mathbb{E}\big[\|g_t(\boldsymbol{w}^*)\|_2^2\big] \leq (1-\beta)^2 C^2 + \sigma^2 \,. \tag{A.10}$$

Recall that $\mathbb{E}[\xi_i^2] = \|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\boldsymbol{w}}\|_{\boldsymbol{D}}^2$ for all $i \geq 0$. Next, I show

$$\begin{aligned}
\mathbb{E}\big[\|g_t(\boldsymbol{w}) - g_t(\boldsymbol{w}^*)\|_2^2\big] &= \mathbb{E}\left[\left\|\sum_{n=1}^{\infty} c_n \boldsymbol{x}_0(\gamma^n \boldsymbol{x}_n - \boldsymbol{x}_0)^{\mathsf{T}}(\boldsymbol{w} - \boldsymbol{w}^*)\right\|_2^2\right] \\
&= \mathbb{E}\left[\left\|\sum_{n=1}^{\infty} c_n \boldsymbol{x}_0(\xi_0 - \gamma^n \xi_n)\right\|_2^2\right] \\
&\leq \sum_{n=1}^{\infty} c_n \mathbb{E}\big[\|\boldsymbol{x}_0(\xi_0 - \gamma^n \xi_n)\|_2^2\big] \\
&\leq \sum_{n=1}^{\infty} c_n \mathbb{E}\big[(\xi_0 - \gamma^n \xi_n)^2\big] \\
&\leq 2\sum_{n=1}^{\infty} c_n \left(\mathbb{E}\big[\xi_0^2\big] + \gamma^{2n}\mathbb{E}\big[\xi_n^2\big]\right) \\
&= 2\sum_{n=1}^{\infty} c_n (1 + \gamma^{2n}) \|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\boldsymbol{w}}\|_{\boldsymbol{D}}^2 \\
&\leq 2\sum_{n=1}^{\infty} c_n (1 + \gamma^{n}) \|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\boldsymbol{w}}\|_{\boldsymbol{D}}^2 \\
&= 2(1+\beta)\|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\boldsymbol{w}}\|_{\boldsymbol{D}}^2 \,.
\end{aligned} \tag{A.11}$$

The four inequalities respectively follow from Jensen's inequality, the bounded-feature assumption $\|\boldsymbol{x}_0\|_2^2 \leq 1$, the triangle inequality, and the fact that $\gamma^{2n} \leq \gamma^n$. The final equality comes from the definition of the contraction modulus, β. Combining Eqs. (A.10) and (A.11) gives the final result:

$$\begin{aligned}
\mathbb{E}\big[\|g_t(\boldsymbol{w})\|_2^2\big] &\leq \mathbb{E}\big[(\|g_t(\boldsymbol{w}^*)\|_2 + \|g_t(\boldsymbol{w}) - g_t(\boldsymbol{w}^*)\|_2)^2\big] \\
&\leq 2\,\mathbb{E}\big[\|g_t(\boldsymbol{w}^*)\|_2^2\big] + 2\,\mathbb{E}\big[\|g_t(\boldsymbol{w}) - g_t(\boldsymbol{w}^*)\|_2^2\big] \\
&\leq 2(1-\beta)^2 C^2 + 2\sigma^2 + 4(1+\beta)\|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\boldsymbol{w}}\|_{\boldsymbol{D}}^2 \,,
\end{aligned}$$

where the second inequality uses the algebraic identity $(x+y)^2 \leq 2x^2 + 2y^2$. □

I am now ready to derive the finite-time bound. I restate Theorem 3.10 and then provide the proof.

Theorem 3.10 (Finite-Time Analysis)**.** *Suppose TD learning with linear function approximation is applied under an i.i.d. state model (see Assumption A.1 in Appendix A.3.3) using compound return $G_t^{\boldsymbol{c}}$ as its target. Let β be the return's contraction modulus and let $\sigma^2 \geq 0$ be the return's variance. Assume that the features are normalized such that $\|\boldsymbol{X}(s)\|_2^2 \leq 1$, $\forall\ s \in \mathcal{S}$. Define $C \coloneqq (\|\boldsymbol{r}\|_\infty + (1+\gamma)\|\boldsymbol{w}^*\|_\infty) \,/\, (1-\gamma)$, where $\boldsymbol{w}^*$ is the minimizer of the projected Bellman error for $G_t^{\boldsymbol{c}}$. For any $T \geq (4/(1-\beta))^2$ and a constant step size $\alpha = 1/\sqrt{T}$,*

$$\mathbb{E}\big[\|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\bar{\boldsymbol{w}}_T}\|_{\boldsymbol{D}}^2\big] \leq \frac{\|\boldsymbol{w}^* - \boldsymbol{w}\|_2^2 + 2(1-\beta)^2 C^2 + 2\sigma^2}{(1-\beta)\sqrt{T}}\,,$$

where $\bar{\boldsymbol{w}}_T \coloneqq \frac{1}{T}\sum_{t=0}^{T-1} \boldsymbol{w}_t$.

Proof. TD learning updates the parameters according to Eq. (A.6). Therefore,

$$\begin{aligned}\|\boldsymbol{w}^* - \boldsymbol{w}_{t+1}\|_2^2 &= \|\boldsymbol{w}^* - \boldsymbol{w}_t - \alpha\, g_t(\boldsymbol{w}_t)\|_2^2 \\ &= \|\boldsymbol{w}^* - \boldsymbol{w}_t\|_2^2 - 2\alpha\, g_t(\boldsymbol{w}_t)^\top(\boldsymbol{w}^* - \boldsymbol{w}_t) + \alpha^2\|g_t(\boldsymbol{w})\|_2^2\,.\end{aligned}$$

Taking the expectation and then applying Lemmas A.2 and A.3 gives

$$\begin{aligned}&\mathbb{E}\big[\|\boldsymbol{w}^* - \boldsymbol{w}_{t+1}\|_2^2\big] \\ &\quad= \mathbb{E}\big[\|\boldsymbol{w}^* - \boldsymbol{w}_t\|_2^2\big] - 2\alpha\,\mathbb{E}\big[g_t(\boldsymbol{w}_t)^\top(\boldsymbol{w}^* - \boldsymbol{w}_t)\big] + \alpha^2\,\mathbb{E}\big[\|g_t(\boldsymbol{w})\|_2^2\big] \\ &\quad= \mathbb{E}\big[\|\boldsymbol{w}^* - \boldsymbol{w}_t\|_2^2\big] - 2\alpha\,\mathbb{E}\big[\mathbb{E}\big[g_t(\boldsymbol{w}_t)^\top(\boldsymbol{w}^* - \boldsymbol{w}_t)\big] \mid \boldsymbol{w}_t\big] + \alpha^2\,\mathbb{E}\big[\mathbb{E}\big[\|g_t(\boldsymbol{w})\|_2^2\big] \mid \boldsymbol{w}_t\big] \\ &\quad\leq \mathbb{E}\big[\|\boldsymbol{w}^* - \boldsymbol{w}_t\|_2^2\big] - \big(2\alpha(1-\beta) - 4\alpha^2(1+\beta)\big)\,\|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\boldsymbol{w}}\|_{\boldsymbol{D}}^2 + 2\alpha^2\left((1-\beta)^2C^2 + \sigma^2\right) \\ &\quad\leq \mathbb{E}\big[\|\boldsymbol{w}^* - \boldsymbol{w}_t\|_2^2\big] - \alpha(1-\beta)\|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\boldsymbol{w}}\|_{\boldsymbol{D}}^2 + 2\alpha^2\left((1-\beta)^2C^2 + \sigma^2\right)\,.\end{aligned}$$

The first inequality is due to Lemmas A.2 and A.3, which are applicable due to the i.i.d. setting (because the trajectory influencing g_t is independent of $\boldsymbol{w}_t$). The second inequality follows from the assumption that $\alpha \leq (1-\beta)\,/\,4$. Rearranging the above inequality gives

$$\mathbb{E}\big[\|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\boldsymbol{w}_t}\|_{\boldsymbol{D}}^2\big] \leq \frac{\|\boldsymbol{w}^* - \boldsymbol{w}_t\|_2^2 - \|\boldsymbol{w}^* - \boldsymbol{w}_{t+1}\|_2^2 + 2\alpha^2\left((1-\beta)^2C^2 + \sigma^2\right)}{\alpha(1-\beta)}\,.$$

Summing over T iterations and then invoking the assumption that $\alpha = 1\,/\,\sqrt{T}$,

$$\begin{aligned}\sum_{t=0}^{T-1}\mathbb{E}\big[\|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\boldsymbol{w}_t}\|_{\boldsymbol{D}}^2\big] &\leq \frac{\|\boldsymbol{w}^* - \boldsymbol{w}_0\|_2^2 - \|\boldsymbol{w}^* - \boldsymbol{w}_T\|_2^2 + 2\alpha^2\left((1-\beta)^2C^2 + \sigma^2\right)T}{\alpha(1-\beta)} \\ &\leq \frac{\|\boldsymbol{w}^* - \boldsymbol{w}_0\|_2^2 + 2\alpha^2\left((1-\beta)^2C^2 + \sigma^2\right)T}{\alpha(1-\beta)}\end{aligned}$$

$$= \frac{\|\boldsymbol{w}^* - \boldsymbol{w}_0\|_2^2 \sqrt{T} + 2\left((1-\beta)^2 C^2 + \sigma^2\right)\sqrt{T}}{1-\beta}\,.$$

I therefore conclude that

$$\mathbb{E}\big[\|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\bar{\boldsymbol{w}}_T}\|_{\boldsymbol{D}}^2\big] \leq \frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\big[\|\boldsymbol{v}_{\boldsymbol{w}^*} - \boldsymbol{v}_{\boldsymbol{w}_t}\|_{\boldsymbol{D}}^2\big] \leq \frac{\|\boldsymbol{w}^* - \boldsymbol{w}_0\|_2^2 + 2(1-\beta)^2 C^2 + 2\sigma^2}{(1-\beta)\sqrt{T}}\,,$$

which completes the bound. □

A.3.4 Proof of Prop. 3.11

Proposition 3.11 (TD Solution Quality)**.** *Let $\boldsymbol{w}^*$ be the minimizer of the projected Bellman error under linear function approximation for any n-step or compound return with contraction modulus β. The following bound always holds:*

$$\|\boldsymbol{X}\boldsymbol{w}^* - \boldsymbol{v}_\pi\|_{\boldsymbol{D}} \leq \frac{1}{1-\beta}\|\boldsymbol{\Pi}\boldsymbol{v}_\pi - \boldsymbol{v}_\pi\|_{\boldsymbol{D}}\,.$$

Proof. I generalize the bound for the λ-return operator given by Tsitsiklis and Van Roy (1997, Lemma 6). An n-step return or compound return corresponds to the operator $T_\pi^{\boldsymbol{c}}\colon \boldsymbol{v} \mapsto \sum_{n=1}^{\infty} c_n T_\pi^n \boldsymbol{v}$. This operator is a contraction mapping with the given modulus, β. The TD procedure in Eq. (A.6) converges to a fixed point, $\boldsymbol{w}^*$, which is the unique solution of the following projected Bellman equation that is analogous to Eq. (2.4):

$$\boldsymbol{\Pi} T_\pi^{\boldsymbol{c}}(\boldsymbol{X}\boldsymbol{w}^*) = \boldsymbol{X}\boldsymbol{w}^*\,,$$

where $\boldsymbol{\Pi}$ is the linear projection operator from Eq. (2.3). In their proof of Lemma 6, Tsitsiklis and Van Roy show that $\boldsymbol{\Pi}$ is nonexpansive with respect to the norm $\|\cdot\|_{\boldsymbol{D}}$. Therefore, for any compound return, it follows that

$$\begin{aligned}
\|\boldsymbol{X}\boldsymbol{w}^* - \boldsymbol{v}_\pi\|_{\boldsymbol{D}} &\leq \|\boldsymbol{X}\boldsymbol{w}^* - \boldsymbol{\Pi}\boldsymbol{v}_\pi\|_{\boldsymbol{D}} + \|\boldsymbol{\Pi}\boldsymbol{v}_\pi - \boldsymbol{v}_\pi\|_{\boldsymbol{D}} \\
&= \|\boldsymbol{\Pi} T_\pi^{\boldsymbol{c}}(\boldsymbol{X}\boldsymbol{w}^*) - \boldsymbol{\Pi}\boldsymbol{v}_\pi\|_{\boldsymbol{D}} + \|\boldsymbol{\Pi}\boldsymbol{v}_\pi - \boldsymbol{v}_\pi\|_{\boldsymbol{D}} \\
&\leq \|T_\pi^{\boldsymbol{c}}(\boldsymbol{X}\boldsymbol{w}^*) - \boldsymbol{v}_\pi\|_{\boldsymbol{D}} + \|\boldsymbol{\Pi}\boldsymbol{v}_\pi - \boldsymbol{v}_\pi\|_{\boldsymbol{D}} \\
&\leq \beta\|\boldsymbol{X}\boldsymbol{w}^* - \boldsymbol{v}_\pi\|_{\boldsymbol{D}} + \|\boldsymbol{\Pi}\boldsymbol{v}_\pi - \boldsymbol{v}_\pi\|_{\boldsymbol{D}}\,.
\end{aligned}$$

Solving the inequality for $\|\boldsymbol{X}\boldsymbol{w}^* - \boldsymbol{v}_\pi\|_{\boldsymbol{D}}$ gives the final bound. □

A.3.5 Proof of Prop. 3.12

Proposition 3.12 (Effective λ of n-step return)**.** *For any $n \geq 1$ and $\gamma < 1$, the λ-return with*

$$\lambda = \frac{1 - \gamma^{n-1}}{1 - \gamma^n}$$

has the same contraction modulus as the n-step return. For any $n \geq 1$ and $\gamma = 1$, the λ-return with

$$\lambda = \frac{n-1}{n}$$

has the same COM as the n-step return.

Proof. **Case $\gamma < 1$:** I substitute $c_n = (1-\lambda)\lambda^{n-1}$ into the weighted average in Eq. (3.5) to compute the contraction modulus of the λ-return:

$$\sum_{n=1}^{\infty} c_n \gamma^n = \sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1}\gamma^n = \gamma(1-\lambda)\sum_{n=1}^{\infty}(\gamma\lambda)^{n-1} = \frac{\gamma(1-\lambda)}{1-\gamma\lambda}\,.$$

I therefore seek λ such that

$$\frac{\gamma(1-\lambda)}{1-\gamma\lambda} = \gamma^n$$

in order to equate the λ-return's contraction modulus to that of the given n-step return. I multiply both sides of the equation by $1-\gamma\lambda$ and isolate λ to complete the case:

$$\begin{aligned}
\gamma(1-\lambda) &= \gamma^n(1-\gamma\lambda) \\
1-\lambda &= \gamma^{n-1}(1-\gamma\lambda) \\
1-\lambda &= \gamma^{n-1} - \gamma^n\lambda \\
\gamma^n\lambda - \lambda &= \gamma^{n-1} - 1 \\
\lambda(\gamma^n - 1) &= \gamma^{n-1} - 1 \\
\lambda &= (1-\gamma^{n-1}) \,/\, (1-\gamma^n)\,.
\end{aligned}$$

Case $\gamma = 1$: I use Prop. 3.7 with $c_n = (1-\lambda)\lambda^{n-1}$ to compute the effective n-step of the λ-return:

$$n = \sum_{k=1}^{\infty}(1-\lambda)\lambda^{k-1}k = (1-\lambda)\sum_{k=1}^{\infty}\lambda^{k-1}k = (1-\lambda)\frac{1}{(1-\lambda)^2} = \frac{1}{1-\lambda}\,.$$

Rearranging the equation $n = 1 \,/\, (1-\lambda)$ for λ gives the final result of $\lambda = (n-1) \,/\, n$. □

A.4 Pilar: Piecewise λ-Return

I present a basic search algorithm to find the corresponding Pilar for a given n-step return (see Algorithm 2). The algorithm accepts the desired effective n-step (which does not need to be an integer necessarily) as its only argument and returns the values (n_1, n_2, c) such that the two-bootstrap return $(1-c)G_t^{(n_1)} + c\,G_t^{(n_2)}$ minimizes the maximum absolute difference between its cumulative weights and those of the λ-return with the same effective n-step. The algorithm proceeds as follows. For each $n_1 \in \{1, \ldots, \lfloor n \rfloor\}$, scan through $n_2 \in \{n_1 + 1, n_1 + 2, \ldots, \}$ until the error stops decreasing. Every time a better (n_1, n_2)-pair is found, record the values, and return the last recorded values upon termination. The resulting Pilar has the same contraction modulus as the targeted n-step return; thus, their error-reduction properties are the same, but the Pilar's variance is lower by Theorem 3.8. To modify the search algorithm for undiscounted rewards, I just need to change λ and c such that they equate the COMs—rather than the contraction moduli—of the two returns. I also include this case in Algorithm 2.

Table A.1: n-step returns and Pilars with equal contraction moduli when $\gamma = 0.99$.

effective n-step	n_1	n_2	c
2	1	4	0.337
3	1	6	0.406
4	2	7	0.406
5	2	9	0.437
10	4	16	0.515
20	6	35	0.519
25	8	43	0.530
50	13	79	0.640
100	22	147	0.760

I populate Table A.1 with corresponding Pilar values for several common n-step returns when $\gamma = 0.99$. A discount factor of $\gamma = 0.99$ is extremely common in deep RL, and so it is hoped that this table serves as a convenient reference that helps practitioners avoid redundant searches with Algorithm 2.

Algorithm 2 Pilar(n)

require $n \geq 1,\ \gamma \in (0, 1)$
$\lambda = \begin{cases} (1 - \gamma^{n-1}) \,/\, (1 - \gamma^n) \text{ if } \gamma < 1 \\ (n-1) \,/\, n \text{ if } \gamma = 1 \end{cases}$
best_error $\leftarrow \infty$
for $n_1 = 1, \ldots, \lfloor n \rfloor$ **do**
    $n_2 \leftarrow \lfloor n \rfloor$
    error $\leftarrow \infty$
    repeat
        $n_2 \leftarrow n_2 + 1$
        $c \leftarrow \begin{cases} (\gamma^n - \gamma^{n_1}) \,/\, (\gamma^{n_2} - \gamma^{n_1}) \text{ if } \gamma < 1 \\ (n - n_1) \,/\, (n_2 - n_1) \text{ if } \gamma = 1 \end{cases}$
        prev_error $\leftarrow$ error
        error $\leftarrow$ ERROR(λ, n_1, n_2, c)
        if error $<$ best_error **then**
            values $\leftarrow (n_1, n_2, c)$
            best_error $\leftarrow$ error
        end if
    until error $\geq$ prev_error
end for
return values

function ERROR(λ, n_1, n_2, c)
    Let $h_i = \begin{cases} 1 & \text{if } i < n_1 \\ c & \text{else if } i < n_2 \\ 0 & \text{else} \end{cases}$
    return $\max_{i \geq 0} |\gamma^i h_i - (\gamma\lambda)^i|$
end function

Appendix B

Trajectory-Aware Off-Policy Corrections

B.1 Proofs

This section contains the proofs omitted from Chapter 6.

B.1.1 Proof of Lemma 6.2

Lemma 6.2. *$\boldsymbol{q}_\pi$ is a fixed point of $\mathcal{M}$. The difference between $\mathcal{M}\boldsymbol{q}$ and $\boldsymbol{q}_\pi$ is*

$$\mathcal{M}\boldsymbol{q} - \boldsymbol{q}_\pi = \boldsymbol{Z}(\boldsymbol{q} - \boldsymbol{q}_\pi)\,, \tag{6.6}$$

where $\boldsymbol{Z} \coloneqq \sum_{i=1}^{\infty} \gamma^i (\mathcal{H}_{i-1}\boldsymbol{P}_\pi - \mathcal{H}_i)$.

Proof. It is evident from Eq. (6.5) that $\boldsymbol{q}_\pi$ is a fixed point of $\mathcal{M}$ because $T_\pi \boldsymbol{q}_\pi - \boldsymbol{q}_\pi = 0$, and so $\mathcal{M}\boldsymbol{q}_\pi = \boldsymbol{q}_\pi$. Therefore,

$$\begin{aligned}
\mathcal{M}\boldsymbol{q} - \boldsymbol{q}_\pi &= \mathcal{M}\boldsymbol{q} - \mathcal{M}\boldsymbol{q}_\pi \\
&= \boldsymbol{q} + \sum_{i=0}^{\infty} \gamma^i \mathcal{H}_i (T_\pi \boldsymbol{q} - \boldsymbol{q}) - \boldsymbol{q}_\pi - \sum_{i=0}^{\infty} \gamma^i \mathcal{H}_i (T_\pi \boldsymbol{q}_\pi - \boldsymbol{q}_\pi) \\
&= \boldsymbol{q} - \boldsymbol{q}_\pi + \sum_{i=0}^{\infty} \gamma^i \mathcal{H}_i (T_\pi \boldsymbol{q} - T_\pi \boldsymbol{q}_\pi) - \sum_{i=0}^{\infty} \gamma^i \mathcal{H}_i (\boldsymbol{q} - \boldsymbol{q}_\pi) \\
&= \sum_{i=0}^{\infty} \gamma^i \mathcal{H}_i (T_\pi \boldsymbol{q} - T_\pi \boldsymbol{q}_\pi) - \sum_{i=1}^{\infty} \gamma^i \mathcal{H}_i (\boldsymbol{q} - \boldsymbol{q}_\pi) \\
&= \sum_{i=0}^{\infty} \gamma^{i+1} \mathcal{H}_i \boldsymbol{P}_\pi (\boldsymbol{q} - \boldsymbol{q}_\pi) - \sum_{i=1}^{\infty} \gamma^i \mathcal{H}_i (\boldsymbol{q} - \boldsymbol{q}_\pi)
\end{aligned}$$

$$
\begin{aligned}
&= \left(\sum_{i=0}^{\infty} \gamma^{i+1} \mathcal{H}_i \boldsymbol{P}_\pi - \sum_{i=1}^{\infty} \gamma^i \mathcal{H}_i \right) (\boldsymbol{q} - \boldsymbol{q}_\pi) \\
&= \left(\sum_{i=1}^{\infty} \gamma^i (\mathcal{H}_{i-1} \boldsymbol{P}_\pi - \mathcal{H}_i) \right) (\boldsymbol{q} - \boldsymbol{q}_\pi) \\
&= \boldsymbol{Z}(\boldsymbol{q} - \boldsymbol{q}_\pi) \,,
\end{aligned}
$$

which is the desired result. □

B.1.2 Proof of Lemma 6.3

Lemma 6.3. *If Condition 6.1 holds, then $\boldsymbol{Z}$ has nonnegative elements and its row sums obey $\boldsymbol{Z}\mathbf{1} \leq \gamma$.*

Proof. Define the linear operator $\boldsymbol{D}_i \coloneqq \mathcal{H}_{i-1}\boldsymbol{P}_\pi - \mathcal{H}_i$ and notice that $\boldsymbol{Z} = \sum_{i=1}^{\infty} \gamma^i \boldsymbol{D}_i$. I show that $\boldsymbol{D}_i$ comprises only nonnegative elements, and therefore so does $\boldsymbol{Z}$. For any $\boldsymbol{q} \in \mathbb{R}^{|\mathcal{S}\times\mathcal{A}|}$, observe that $\boldsymbol{D}_i\boldsymbol{q} = \mathcal{H}_{i-1}\boldsymbol{P}_\pi\boldsymbol{q} - \mathcal{H}_i\boldsymbol{q}$ and therefore

$$
\begin{aligned}
(D_i q)(s,a) &= \mathbb{E}_b \left[H_{t:t+i-1} \sum_{a' \in \mathcal{A}} \pi(a' \mid S_{t+i-1})\, q(S_{t+i-1}, a') - H_{t:t+i}\, q(S_{t+i}, A_{t+i}) \;\middle|\; (S_t, A_t) = (s,a) \right] \\
&= \mathbb{E}_b \left[\sum_{a' \in \mathcal{A}} \Big(\pi(a' \mid S_{t+i-1})\, H_{t:t+i-1} - b(a' \mid S_{t+i-1})\, H_{t:t+i} \Big)\, q(S_{t+i-1}, a') \;\middle|\; (S_t, A_t) = (s,a) \right] .
\end{aligned}
$$

Since I assumed that $H_{t:t+i} \leq H_{t:t+i-1} \cdot \rho_{t+i}$ holds for all transitions in Condition 6.1, I have $\pi(a' \mid S_{t+i-1})\, H_{t:t+i-1} - b(a' \mid S_{t+i-1})\, H_{t:t+i} \geq 0$, which implies that $\boldsymbol{D}_i \geq 0$. Furthermore, this holds for all $i \geq 1$, so $\boldsymbol{Z} \geq 0$ follows immediately.

To complete the proof, I show that the row sums of $\boldsymbol{Z}$ are bounded by γ. Recall that $\boldsymbol{P}_\pi\mathbf{1} = \mathbf{1}$. Hence,

$$
\begin{aligned}
\boldsymbol{Z}\mathbf{1} &= \sum_{i=1}^{\infty} \gamma^i (\mathcal{H}_{i-1}\boldsymbol{P}_\pi - \mathcal{H}_i)\mathbf{1} \\
&= \sum_{i=1}^{\infty} \gamma^i (\mathcal{H}_{i-1}\mathbf{1} - \mathcal{H}_i\mathbf{1}) \\
&= \sum_{i=0}^{\infty} \gamma^{i+1} \mathcal{H}_i\mathbf{1} - \sum_{i=1}^{\infty} \gamma^i \mathcal{H}_i\mathbf{1} \\
&= \gamma\mathbf{1} + \sum_{i=1}^{\infty} \gamma^{i+1} \mathcal{H}_i\mathbf{1} - \sum_{i=1}^{\infty} \gamma^i \mathcal{H}_i\mathbf{1}
\end{aligned}
$$

$$= \gamma \mathbf{1} - (1-\gamma) \sum_{i=1}^{\infty} \gamma^i \mathcal{H}_i \mathbf{1}$$
$$\leq \gamma \mathbf{1} \,,$$

because $\mathcal{H}_i \geq 0$, $\forall\ i \geq 1$. □

B.1.3 Proof of Theorem 6.5

Theorem 6.5. *Consider a sequence of target policies $(\pi_k)_{k\geq 0}$ and a sequence of arbitrary behavior policies $(b_k)_{k\geq 0}$. Let $\boldsymbol{q}_0$ be an arbitrary vector in $\mathbb{R}^{|\mathcal{S}\times\mathcal{A}|}$ and define the sequence $\boldsymbol{q}_{k+1} \coloneqq \mathcal{M}_k \boldsymbol{q}_k$, where $\mathcal{M}_k$ is the operator defined by Eq. (6.7). Assume that $(\pi_k)_{k\geq 0}$ is greedy in the limit, and let $\epsilon_k \geq 0$ be the smallest constant such that $T_{\pi_k}\boldsymbol{q}_k \geq T\boldsymbol{q}_k - \epsilon_k \|\boldsymbol{q}_k\| \mathbf{1}$. If Condition 6.1 holds for all k, then*

$$\|\mathcal{M}_k \boldsymbol{q}_k - \boldsymbol{q}_*\| \leq \gamma \|\boldsymbol{q}_k - \boldsymbol{q}_*\| + \frac{\epsilon_k}{1-\gamma} \|\boldsymbol{q}_k\| \,, \tag{6.8}$$

and, consequently, $\lim_{k\to\infty} \boldsymbol{q}_k = \boldsymbol{q}_*$.

Proof. I first derive the following upper bound:

$$T_{\pi_k}\boldsymbol{q}_k - T\boldsymbol{q}_* = \gamma \boldsymbol{P}_{\pi_k}\boldsymbol{q}_k - \gamma \boldsymbol{P} E \boldsymbol{q}_* \leq \gamma \boldsymbol{P}_{\pi_k}(\boldsymbol{q}_k - \boldsymbol{q}_*) \,.$$

From Eq. (6.7) and because $\boldsymbol{C}_k$ has nonnegative entries, I deduce that

$$\begin{aligned} \mathcal{M}_k \boldsymbol{q}_k - \boldsymbol{q}_* &= (\boldsymbol{I} - \boldsymbol{C}_k)(\boldsymbol{q}_k - \boldsymbol{q}_*) + \boldsymbol{C}_k(T_{\pi_k}\boldsymbol{q}_k - \boldsymbol{q}_*) && \text{(B.1)} \\ &= (\boldsymbol{I} - \boldsymbol{C}_k)(\boldsymbol{q}_k - \boldsymbol{q}_*) + \boldsymbol{C}_k(T_{\pi_k}\boldsymbol{q}_k - T\boldsymbol{q}_*) \\ &\leq (\boldsymbol{I} - \boldsymbol{C}_k)(\boldsymbol{q}_k - \boldsymbol{q}_*) + \gamma \boldsymbol{C}_k \boldsymbol{P}_{\pi_k}(\boldsymbol{q}_k - \boldsymbol{q}_*) \\ &= \boldsymbol{Z}_k(\boldsymbol{q}_k - \boldsymbol{q}_*) \,, && \text{(B.2)} \end{aligned}$$

where $\boldsymbol{Z}_k \coloneqq \boldsymbol{I} - \boldsymbol{C}_k(\boldsymbol{I} - \gamma \boldsymbol{P}_{\pi_k})$. Notice that $\boldsymbol{Z}_k$ is analogous to the matrix $\boldsymbol{Z}$ in Eq. (6.6) because, for policies π_k and b_k,

$$\begin{aligned} \boldsymbol{I} - \boldsymbol{C}_k(\boldsymbol{I} - \gamma \boldsymbol{P}_{\pi_k}) &= \boldsymbol{I} + \sum_{i=0}^{\infty} \gamma^i \mathcal{H}_i (\gamma \boldsymbol{P}_{\pi_k} - \boldsymbol{I}) \\ &= \boldsymbol{I} + \sum_{i=0}^{\infty} \gamma^{i+1} \mathcal{H}_i \boldsymbol{P}_{\pi_k} - \sum_{i=0}^{\infty} \gamma^i \mathcal{H}_i \\ &= \sum_{i=1}^{\infty} \gamma^i \mathcal{H}_{i-1} \boldsymbol{P}_{\pi_k} - \sum_{i=1}^{\infty} \gamma^i \mathcal{H}_i \end{aligned}$$

$$= \sum_{i=1}^{\infty} \gamma^i (\mathcal{H}_{i-1} \boldsymbol{P}_{\pi_k} - \mathcal{H}_i) \,.$$

Next, I derive the following lower bound:

$$T\boldsymbol{q}_k - T\boldsymbol{q}_* \geq T_{\pi_*}\boldsymbol{q}_k - T\boldsymbol{q}_* = \gamma \boldsymbol{P}_{\pi_*}(\boldsymbol{q}_k - \boldsymbol{q}_*) \,.$$

Additionally, for each policy π_k, there exists some $\epsilon_k \geq 0$ such that $T_{\pi_k}\boldsymbol{q}_k \geq T\boldsymbol{q}_k - \epsilon_k \|\boldsymbol{q}_k\| \mathbf{1}$ (recall that I defined ϵ_k to be as small as possible). Starting again from Eq. (B.1), and noting that the elements of $\boldsymbol{C}_k$ are nonnegative, I obtain

$$\begin{aligned}
\mathcal{M}_k \boldsymbol{q}_k - \boldsymbol{q}_* &\geq (\boldsymbol{I} - \boldsymbol{C}_k)(\boldsymbol{q}_k - \boldsymbol{q}_*) + \boldsymbol{C}_k(T\boldsymbol{q}_k - \boldsymbol{q}_*) - \epsilon_k \|\boldsymbol{q}_k\| \boldsymbol{C}_k \mathbf{1} \\
&= (\boldsymbol{I} - \boldsymbol{C}_k)(\boldsymbol{q}_k - \boldsymbol{q}_*) + \boldsymbol{C}_k(T\boldsymbol{q}_k - T\boldsymbol{q}_*) - \epsilon_k \|\boldsymbol{q}_k\| \boldsymbol{C}_k \mathbf{1} \\
&\geq (\boldsymbol{I} - \boldsymbol{C}_k)(\boldsymbol{q}_k - \boldsymbol{q}_*) + \gamma \boldsymbol{C}_k \boldsymbol{P}_{\pi_*}(\boldsymbol{q}_k - \boldsymbol{q}_*) - \epsilon_k \|\boldsymbol{q}_k\| \boldsymbol{C}_k \mathbf{1} \\
&= \boldsymbol{Z}_k^*(\boldsymbol{q}_k - \boldsymbol{q}_*) - \epsilon_k \|\boldsymbol{q}_k\| \boldsymbol{C}_k \mathbf{1} \,, \qquad \text{(B.3)}
\end{aligned}$$

where I have defined $\boldsymbol{Z}_k^* \coloneqq \boldsymbol{I} - \boldsymbol{C}_k(\boldsymbol{I} - \gamma \boldsymbol{P}_{\pi_*})$. By Lemma 6.3, since I assumed Condition 6.1 holds, both $\boldsymbol{Z}_k$ and $\boldsymbol{Z}_k^*$ have nonnegative elements and their row sums are bounded by γ. Therefore, when $\mathcal{M}_k \boldsymbol{q}_k - \boldsymbol{q}_* \geq 0$, Eq. (B.2) implies

$$\|\mathcal{M}_k \boldsymbol{q}_k - \boldsymbol{q}_*\| \leq \gamma \|\boldsymbol{q}_k - \boldsymbol{q}_*\| \,, \qquad \text{(B.4)}$$

because element-wise inequality for nonnegative matrices implies the inequality holds for their norms as well. When $\mathcal{M}_k \boldsymbol{q}_k - \boldsymbol{q}_* \leq 0$, I must use Eq. (B.3) and multiply both sides by -1 to get nonnegative matrices, giving

$$\begin{aligned}
\|\mathcal{M}_k \boldsymbol{q}_k - \boldsymbol{q}_*\| &\leq \gamma \|\boldsymbol{q}_k - \boldsymbol{q}_*\| + \epsilon_k \|\boldsymbol{q}_k\| \|\boldsymbol{C}_k\| \\
&\leq \gamma \|\boldsymbol{q}_k - \boldsymbol{q}_*\| + \frac{\epsilon_k}{1-\gamma} \|\boldsymbol{q}_k\| \,, \qquad \text{(B.5)}
\end{aligned}$$

because $\|\boldsymbol{C}_k\| \leq \sum_{i=0}^{\infty} \gamma^i \|\boldsymbol{P}_{\pi_k}\|^i = 1 \,/\, (1-\gamma)$. Since Eq. (B.5) is looser than Eq. (B.4), its bound holds in the worst case. It remains to show that this bound implies convergence to $\boldsymbol{q}_*$. Observe that

$$\begin{aligned}
\gamma \|\boldsymbol{q}_k - \boldsymbol{q}_*\| + \frac{\epsilon_k}{1-\gamma} \|\boldsymbol{q}_k\| &\leq \gamma \|\boldsymbol{q}_k - \boldsymbol{q}_*\| + \frac{\epsilon_k}{1-\gamma} (\|\boldsymbol{q}_k - \boldsymbol{q}_*\| + \|\boldsymbol{q}_*\|) \\
&= \left(\gamma + \frac{\epsilon_k}{1-\gamma} \right) \|\boldsymbol{q}_k - \boldsymbol{q}_*\| + \frac{\epsilon_k}{1-\gamma} \|\boldsymbol{q}_*\| \,.
\end{aligned}$$

My assumption of greedy-in-the-limit policies means that $\epsilon_k \to 0$ as $k \to \infty$; thus, there must exist some iteration k^* such that $\epsilon_k \leq \frac{1}{2}(1-\gamma)^2$, $\forall\ k \geq k^*$. Therefore, for $k \geq k^*$,

$$\|\mathcal{M}_k \boldsymbol{q}_k - \boldsymbol{q}_*\| \leq \frac{1+\gamma}{2}\|\boldsymbol{q}_k - \boldsymbol{q}_*\| + \frac{\epsilon_k}{1-\gamma}\|\boldsymbol{q}_*\|\,.$$

If $\gamma < 1$, then $\frac{1}{2}(1+\gamma) < 1$, and since $\|\boldsymbol{q}_*\|$ is finite, I conclude that $\|\boldsymbol{q}_k - \boldsymbol{q}_*\| \to 0$ as $k \to \infty$. □

B.2 Examples of Divergence

This section describes two counterexamples which show that violating Condition 6.1 could cause TD methods to diverge.

B.2.1 Counterexample 6.9: Off-Policy Truncated IS

Counterexample 6.9 (Off-Policy Truncated IS)**.** *Consider Truncated IS with $\lambda = 1$, so $H_{t:t+i} = \min(1, \Pi_{t+1:t+i})$. Assume the MDP has one state s and two actions $\{a_1, a_2\}$, the behavior policy is uniform random, and the target policy selects a_1 with probability $p \in (0,1)$ and selects a_2 otherwise. When $p = 0.6$ and $\gamma = 0.94$, then $\|\boldsymbol{Z}\| > 1$.*

The definitions of π and b imply

$$\boldsymbol{P}_\pi = \begin{bmatrix} p & 1-p \\ p & 1-p \end{bmatrix}, \qquad\qquad \boldsymbol{P}_b = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}.$$

Recall that I assumed $\lambda = 1$. I define the following constant, using the definition of $H_{t:t+i}$ for Truncated IS (let $\mathcal{F}_t \coloneqq \mathcal{F}_{0:t}$ for brevity):

$$\begin{aligned}
h_t^{(1)} &\coloneqq \mathbb{E}\big[H_{0:t} \;\big|\; (S_t, A_t) = (s, a_1)\big] \\
&= \sum_{\mathcal{F}_t} \Pr_b\big(\mathcal{F}_t \mid (S_t, A_t) = (s, a_1)\big) \cdot \min\left(1,\ \frac{\Pr_\pi(\mathcal{F}_t)}{\Pr_b(\mathcal{F}_{0:t})}\right) \\
&= \sum_{\mathcal{F}_{t-1}} \Pr_b(\mathcal{F}_{t-1}) \min\left(1,\ \frac{\Pr_\pi(\mathcal{F}_{t-1}) \cdot p}{\Pr_b(\mathcal{F}_{t-1}) \cdot \frac{1}{2}}\right) \qquad \text{(B.6)} \\
&= \sum_{\mathcal{F}_{t-1}} \min\Big(\Pr_b(\mathcal{F}_{t-1}),\ 2p \cdot \Pr_\pi(\mathcal{F}_{t-1})\Big) \\
&= \sum_{\mathcal{F}_{t-1}} \min\left(\frac{1}{2^{t-1}},\ 2p \cdot \Pr_\pi(\mathcal{F}_{t-1})\right).
\end{aligned}$$

Eq. (B.6) is justified because the conditional probability of a trajectory ending in action a_1 is just the probability of $\mathcal{F}_{t-1}$ under b, due to the 1-state (memoryless) MDP. I can simplify $h_t^{(1)}$ further by using the binomial theorem to calculate $\Pr_\pi(\mathcal{F}_{t-1}) = p^k(1-p)^{t-1-k}$, where $k \in \{0, \dots, t-1\}$ is the number of times a_1 is taken in $\mathcal{F}_{t-1}$. There are $\binom{t-1}{k}$ trajectories with this same probability. Therefore,

$$h_t^{(1)} = \sum_{\mathcal{F}_{t-1}} \min\left(\frac{1}{2^{t-1}},\ 2p \cdot \Pr_\pi(\mathcal{F}_{t-1})\right) = \sum_{k=0}^{t-1} \binom{t-1}{k} \min\left(\frac{1}{2^{t-1}},\ 2p \cdot p^k(1-p)^{t-1-k}\right) .$$

Likewise, I can compute $h_t^{(2)}$ by swapping p and $1-p$ above. Let $\odot$ denote element-wise multiplication. Using the fact that $\boldsymbol{P}_b^t = \boldsymbol{P}_b$, $\forall\ t \geq 1$, it follows that

$$\mathcal{H}_t = \boldsymbol{P}_b^t \odot \begin{bmatrix} h_t^{(1)} & h_t^{(2)} \\ h_t^{(1)} & h_t^{(2)} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} h_t^{(1)} & h_t^{(2)} \\ h_t^{(1)} & h_t^{(2)} \end{bmatrix} .$$

Using a computer program to calculate $\boldsymbol{Z}$, assuming that $p = 0.6$ and $\gamma = 0.94$, I obtain

$$\boldsymbol{Z} = \sum_{t=1}^{\infty} \gamma^t (\mathcal{H}_{t-1}\boldsymbol{P}_\pi - \mathcal{H}_t) \approx \begin{bmatrix} 0.704 & -0.436 \\ 0.704 & -0.436 \end{bmatrix} .$$

Therefore, $\|\boldsymbol{Z}\| \approx 1.14$, which is not a contraction, and the norm continues to increase for $p > 0.6$ or $\gamma > 0.94$.

B.2.2 Counterexample 6.10: On-Policy Binary Traces

Counterexample 6.10 (On-Policy Binary Traces)**.** *Assume the MDP has one state s and two actions $\{a_1, a_2\}$. Define a trajectory-aware method such that $H_{t:t+i} = 1$ if $A_{t+i} = a_1$ and $H_{t:t+i} = 0$ if $A_{t+i} = a_2$ (without loss of generality). Assume π and b are uniform random. When $\gamma \geq \frac{2}{3}$, then $\|\boldsymbol{Z}\| \geq 1$.*

The policy π is uniform random, so

$$\boldsymbol{P}_\pi = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} .$$

Let $\odot$ denote element-wise multiplication. Because $H_{t:t+i} = 1$ only when the trajectory $\mathcal{F}_{t:t+i}$ terminates in (s, a_1) and $H_{t:t+i} = 0$ otherwise, and since $\boldsymbol{P}_\pi^i = \boldsymbol{P}_\pi$, $\forall\ i \geq 1$, I also have

$$\mathcal{H}_i = P_\pi^i \odot \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} .$$

Using a computer program to calculate $\boldsymbol{Z}$, assuming that $\gamma = \frac{2}{3}$, I obtain

$$\boldsymbol{Z} = \sum_{i=1}^{\infty} \gamma^i (\mathcal{H}_{i-1} \boldsymbol{P}_\pi - \mathcal{H}_i) = \frac{1}{3} \begin{bmatrix} -1 & 2 \\ -1 & 2 \end{bmatrix} .$$

Therefore, $\|\boldsymbol{Z}\| = 1$, which is not a contraction, and the norm continues to increase for $\gamma > \frac{2}{3}$.

B.3 Implementation of Trajectory-Aware Eligibility Traces

The implementation of trajectory-aware methods is closely related to that of backward-view TD(λ) in the tabular setting (see, e.g., Sutton and Barto, 1998, Ch. 7.3). On each time step, an environment interaction is conducted according to the behavior policy b. Then, the eligibilities for previously visited state-action pairs are modified, the eligibility for the current state-action pair is incremented, and the current TD error is applied to all state-action pairs in proportion to their eligibilities. The only difference in the trajectory-aware case is that the eligibilities are not modified by simply multiplying a constant decay factor of $\gamma\lambda$.

Arbitrary, trajectory-dependent traces of the form $H_{t:t+i}$ can be complicated to implement. This stems from the fact that the index i in the $\mathcal{M}$ operator is defined *relative* to when the state-action pair was taken. In other words, each state-action pair (S_k, A_k) "disagrees" on the start of the current trajectory for reinforcement, generating its update from the unique sub-trajectory $\mathcal{F}_{k:t}$. Implementing coefficients of this form would theoretically be possible using the backward-view update

$$Q(S_k, A_k) \leftarrow Q(S_k, A_k) + \alpha_t \, \gamma^{t-k} \, H_{k:t} \, \delta_t^\pi \, , \quad \forall \; k \in \{0, 1, \ldots, t\} \, ,$$

but this would require repeatedly slicing the list of visited state-action pairs, $\mathcal{F}_{0:t}$. While this is technically feasible, it does not easily accommodate vectorization or parallelization.

Fortunately, this level of generality is rarely needed in practice, and specific optimizations can be made depending on the functional form of $H_{t:t+i}$. For example, Truncated IS defines $H_{t:t+i}$ to be a pure function of the IS estimate $\Pi_{t+1:t+i}$, which is useful because per-decision eligibility traces can be used to efficiently generate the IS estimates for every state-action pair visited during the episode. I demonstrate how this can be done in pseudocode (see Algorithm 3).

Recursive methods like Recursive Retrace and RBIS, where $H_{t:t+i}$ explicitly depends on $H_{t:t+i-1}$, require only two minor changes compared to Algorithm 3 for their implementations. These changes, which I highlight in blue for RBIS in Algorithm 4, correspond to the fact that the dynamic array Y is now used to store the previous trace $H_{k:t-1}$ rather than the previous IS estimate at each time step. The computational requirements for the methods remain nearly identical. The implementation for Recursive Retrace easily follows by changing line 10 of Algorithm 4 to

$$Y(k) \leftarrow \lambda \min(1,\ Y(k) \cdot \rho_t)\,.$$

B.4 Additional Experiment Details and Results

I conduct a grid search to find the best step size α for every λ-value for the four off-policy methods I evaluate in the Bifurcated Gridworld (Section 6.5). Using a training set of 1,000 trials, I search over $\lambda \in \{0, 0.1, \ldots, 1\}$ and $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, for a total of 55 hyperparameter combinations. At the start of each trial, the initial value function Q is sampled from a zero-mean Gaussian distribution with standard deviation $\sigma = 0.01$. I train each agent for 3,000 time steps, allowing extra time to complete the final episode. I then generate learning curves by plotting the 100-episode moving average of these returns as a function of the number of time steps and calculate their AUCs. In Table B.1, I report the step size α that led to the highest average AUC for each λ-value. Then, using a separate test set of 1,000 trials (to avoid bias in the search results), I use these α-values to generate the learning curves in Figure B.1. The AUCs for these learning curves are finally used in the creation of the λ-sweep plot (Figure 6.2).

In Figure B.2, I repeat this procedure for three additional, more complex gridworld topologies. Like the Bifurcated Gridworld, these environments feature one or more bifurcations that make fast credit assignment imperative, as well as additional challenges such as multiple goal cells. As before, the agent starts in S and receives +1 reward for taking any action in a goal, terminating the episode. The results are qualitatively similar to Figure 6.2; RBIS outperforms the other three methods by a significant margin over the left-hand portion of the λ-spectrum, and performs similarly to Retrace as $\lambda \to 1$.

Table B.1: The best step sizes found by the grid search in the Bifurcated Gridworld.

λ	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Retrace	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.7	0.7	0.5
Truncated IS	0.9	0.9	0.9	0.9	0.9	0.9	0.7	0.5	0.5	0.5	0.3
Recursive Retrace	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.7	0.5	0.5
RBIS	0.9	0.9	0.9	0.9	0.9	0.7	0.7	0.7	0.7	0.7	0.5

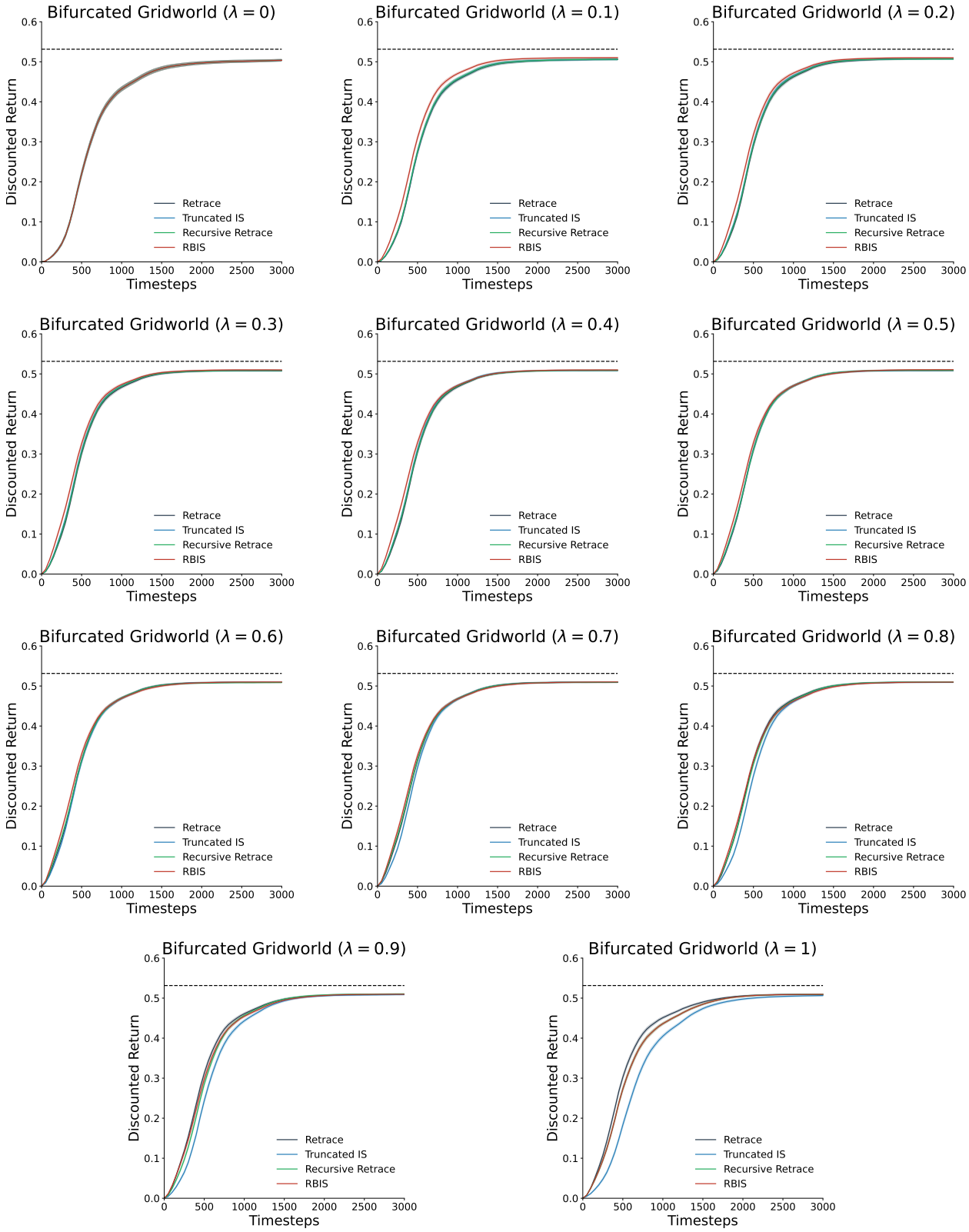


Figure B.1: Learning curves of off-policy methods for all λ-values tested in Bifurcated Gridworld. The dashed black line indicates the optimal discounted return for this problem.

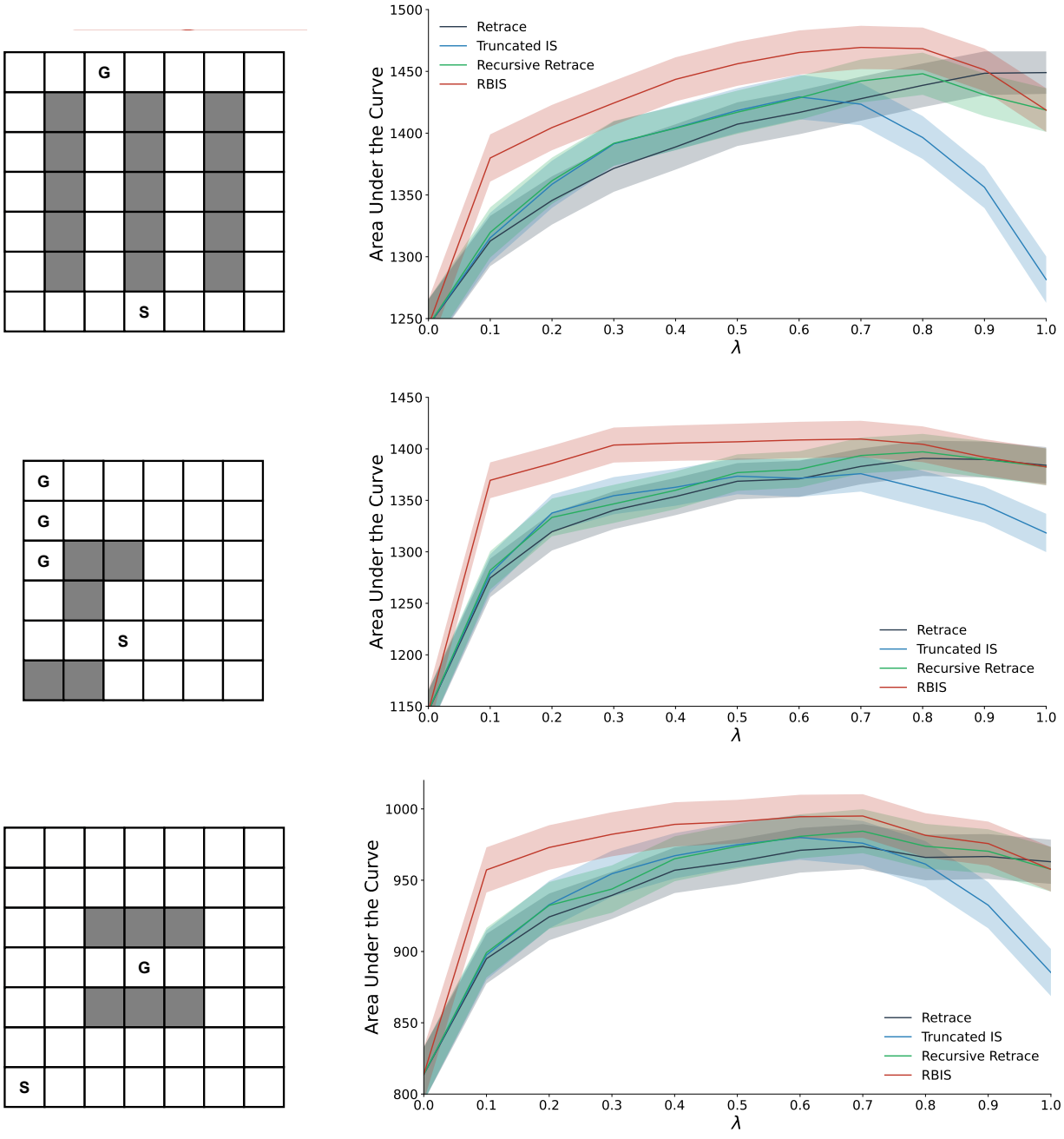


Figure B.2: λ-sweeps for off-policy methods in three additional gridworld topologies. The experiment procedure is identical to that used in the creation of Figure 6.2. The results are averaged over 1,000 trials with 95% confidence intervals shaded.

Algorithm 3 Truncated Importance Sampling

input value function Q, step size $\alpha \in (0, 1]$
for each episode **do**
    Reset environment and observe state S_0
    Reset dynamic array Y
    repeat for $t = 0, 1, 2, \ldots$
        Take action $A_t \sim b(\cdot \mid S_t)$, receive reward R_{t+1}, and observe next state S_{t+1}
        $\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
        $\delta_t^\pi = \begin{cases} R_{t+1} - Q(S_t, A_t) & \text{if } S_{t+1} \text{ is terminal} \\ R_{t+1} - Q(S_t, A_t) + \gamma \sum_{a' \in \mathcal{A}} \pi(a' \mid S_{t+1})\, Q(S_{t+1}, a') & \text{else} \end{cases}$
        for $k = 0, \ldots, t-1$ **do**
            $Y(k) \leftarrow Y(k) \cdot \rho_t$
        end for
        $Y(t) \leftarrow 1$
        for $k = 0, \ldots, t$ **do**
            $z \leftarrow (\gamma\lambda)^{t-k} \min(1,\ Y(k))$
            $Q(S_k, A_k) \leftarrow Q(S_k, A_k) + \alpha z \delta_t$
        end for
    until S_{t+1} is terminal
end for

Algorithm 4 Recency-Bounded Importance Sampling (RBIS)

input value function Q, step size $\alpha \in (0, 1]$
for each episode **do**
    Reset environment and observe state S_0
    Reset dynamic array Y
    repeat for $t = 0, 1, 2, \ldots$
        Take action $A_t \sim b(\cdot \mid S_t)$, receive reward R_{t+1}, and observe next state S_{t+1}
        $\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
        $\delta_t^\pi = \begin{cases} R_{t+1} - Q(S_t, A_t) & \text{if } S_{t+1} \text{ is terminal} \\ R_{t+1} - Q(S_t, A_t) + \gamma \sum_{a' \in \mathcal{A}} \pi(a' \mid S_{t+1})\, Q(S_{t+1}, a') & \text{else} \end{cases}$
        for $k = 0, \ldots, t-1$ **do**
            $Y(k) \leftarrow \min(\lambda^{t-k},\ Y(k) \cdot \rho_t)$
        end for
        $Y(t) \leftarrow 1$
        for $k = 0, \ldots, t$ **do**
            $z \leftarrow \gamma^{t-k} Y(k)$
            $Q(S_k, A_k) \leftarrow Q(S_k, A_k) + \alpha z \delta_t$
        end for
    until S_{t+1} is terminal
end for