Improving Sample Efficiency of Online Temporal Difference Learning

by

Yangchen Pan

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

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

A common scientific challenge for putting a reinforcement learning agent into practice is how to improve sample efficiency as much as possible with limited computational or memory resources. Such available physical resources may vary in different applications. My thesis introduces some approaches to flexibly balance sample efficiency and physical resource for prediction and control problems in an online reinforcement learning setting. Our methods can significantly improve sample efficiency with reasonable computational power and storage demand.

We draw on two key optimization strategies that are known to improve convergence rates: second-order optimizations and prioritized sampling of what data to update with. In this thesis, we mainly focus on the policy evaluation problem, though we also introduce effective sampling distribution for control tasks. Particularly, in policy evaluation problems, we develop an approximate second-order method to minimize Mean Squared Projected Bellman Error (MSPBE). Our method scales sub-quadratically with feature dimension in terms of computational and memory cost. We propose two techniques to efficiently and incrementally approximate the preconditioning matrix in the second-order updating rule: truncated singular value decomposition and sketching via random projection. We further introduce a simple regularization method to theoretically guarantee the unbiased convergence of our algorithm, under certain assumptions.

In control problems, we focus on studying effective sampling distribu-

tions to sample imagined experiences in model-based reinforcement learning (MBRL). Specifically, in a classic MBRL architecture called Dyna, we design novel search-control strategies, which refer to the mechanisms of generating states from which we query an environment model to acquire imagined experiences to improve the policy during the planning phase. We provide both theoretical and empirical evidence to verify that our methods improve sample efficiency.

Acknowledgements

I cannot express more of my thankfulness to my supervisors, Martha White and Amir-massoud Farahmand. They significantly positively changed my life.

I want to thank my Ph.D. examination committee members, Adam White, Bo Dai, Matthew Taylor, and Warren Powell, for their valuable time and feedback. Particularly, I would like to thank Bo Dai and Warren Powell for serving as the external examiners.

I want to thank my parents and relatives who always support me throughout my whole life.

Finally, I want to thank everyone in my life I ever met. They ultimately shape me to be a better person, indirectly or directly.

Contents

List of Figures

Chapter 1

Introduction

Reducing the number of training samples needed for Artificial Intelligence (AI) agents to interact with the real environment is one of the key challenges of putting reinforcement learning algorithms into practice. Such interactions can be extremely expensive in terms of time or even financial cost. The time cost can be extremely high: unlike using computer simulation to generate data, we have to wait for an agent to physically interact with the real environment to collect data samples. In some cases, the financial cost may also be non-trivial. For example, physical robotic control tasks may induce maintenance or expensive operating costs. An AI financial market trader may cause a lot of loss before starting to be profitable. As a result, given certain performance measure, we would like to design AI agents which can learn to achieve as good performance as possible by using as few real-world data samples as possible. That is, we want to improve the *sample efficiency* of an AI agent.

High sample efficiency should not be the only criterion for designing an ideal AI agent; computation or memory cost also matters. A high-performing agent may not be preferred in real-world applications, if it costs enormous amounts of physical resources, such as computational power or memory. Ideally, we would like to have an agent which can achieve optimal performance but uses little physical resources. Notwithstanding, such an ideal agent may not exist in practice. As a result, typically, some trade-off between sample efficiency and physical resource consumption needs to be made. **This motivates the research topic of this thesis**:

How can we improve the sample efficiency of a reinforcement learning algorithm with limited computational and memory resources?

We introduce some approaches from the mathematical optimization perspective, which can flexibly balance an agent's sample efficiency and the cost of physical resources in prediction and control problems in an online reinforcement learning setting. We put a slightly heavier emphasis on the prediction problem in this thesis. Prediction problem corresponds to policy evaluation, which concerns predicting state values under some policy, while the goal of control problems is to find a good policy for decision making. We consider prediction problems in a linear function approximation setting. We bring into policy evaluation algorithms quasi-second order optimization techniques, resulting in a new family of algorithms called Accelerated Gradient Temporal Difference (ATD) learning, which scale sub-quadratically to feature dimension in terms of computational and storage cost.

As for the control problem, we focus on a relatively under-studied area—designing effective sampling distribution in a model-based reinforcement learning (MBRL) setting, particularly in Dyna architecture (Sutton, 1991). Specifically, we discuss how to generate imagined experiences from an environment model. We use the term *imagined experiences* to make a distinction with real experiences: the former refers to those that do not correspond to the agent's physical interactions with the real environment. In contrast, the latter refers to those generated by physically interacting with the real environment. The phrase *imagined experience* is conceptually similar to what humans do when they think about future possibilities: they imagine how future scenarios would look like and make decisions. In some literature, the terms *hypothetical experience* or *simulated experience* are also used.

1.1 Motivation

In this section, we motivate the proposed approaches in the thesis. We focus on the following questions:

1. Why do we care about policy evaluation problems?

2. Why do we develop Accelerated gradient Temporal Difference (ATD) learning algorithms?

3. Why do we study sampling distributions in MBRL? Particularly, why do we study it within the Dyna formalism?

Motivation of Policy Evaluation Problems. Policy evaluation problems can be commonly seen in practical scenarios. It is sensible to evaluate a policy before deploying it—we would want to deploy a policy only when it is sufficiently high value. Studying policy evaluation algorithms is also naturally important for control problems. First, a policy evaluation algorithm may be easily adapted to be a control algorithm. For example, an agent may take actions greedily w.r.t. action values learned by a policy evaluation algorithm. Second, many well-known policy gradient methods involve learning a critic, which can be considered a policy evaluation process. Hence, improvement in policy evaluation algorithms potentially benefits algorithms for solving control problems.

Motivation of ATD Algorithms. Our motivation for developing ATD algorithms is to provide a more flexible balance between sample and computational efficiencies than the existing TD algorithms. These TD algorithms include computationally-frugal, linear, stochastic approximation methods to data efficient but quadratic Least Squares TD (LSTD) methods, with little in between. Stochastic approximation methods, such as TD learning (Sutton, 1988) and gradient TD methods, (Maei, 2011) require linear (in the number of features) computation per time step and linear memory. These linear TD-based algorithms are well suited to problems with high dimensional feature vectors—compared to available resources—and domains where agent interaction occurs at a high rate (Szepesvari, 2010). However, when sample efficiency is of primary concern, LSTD methods, which incur quadratic time and storage complexity, may be preferred. As a result, we would like to develop methods interpolating between linear TD methods and LSTD. We want algorithms that have similar sample efficiency with LSTD but incur less computational

and storage cost than LSTD while still converging to the same solution as LSTD does.

It is worth mentioning that some previous efforts attempt to reduce the computation and storage costs of LSTD methods to span the gap between TD and LSTD. However, there is still large room to make further improvements. For example, the iLSTD method (Geramifard & Bowling, 2006) achieves sub-quadratic computation per time step, but still requires memory that is quadratic in the size of the features. The tLSTD method (Gehring et al., 2016) uses an incremental singular value decomposition (SVD) to achieve sub-quadratic computation and storage. Though in practice, tLSTD achieves running time much closer to TD compared to iLSTD (Gehring et al., 2016), it finds a biased solution. Hence, there is a need to improve the sample efficiency of linear TD methods, while avoiding quadratic computation and storage and asymptotic bias. Our ATD algorithms are proposed to achieve this purpose.

Motivation of Studying Sampling Distribution in Model-based Reinforcement Learning. We separately discuss: 1) why we study the sampling distribution; 2) why we consider the MBRL setting; and 3) why we study Dyna.

The key reason for us to study the sampling distribution is simple: the sampling distribution—which, in a conventional supervised learning setting, is known to significantly affect the sample efficiency—is relatively under-explored in RL. Hence, designing the sampling distribution is a promising direction to improve the sample efficiency of RL algorithms.

The reason for concerning MBRL setting is that leveraging an environment model is a natural way to save physical interactions with the real world, because the agent can interact with the environment model to collect data, rather than interact with the real world. In MRBL research, most existing research focuses on learning an environment model. The problem of what kind of imagined experiences should be generated for improving the policy during the planning stage is relatively understudied, especially in relatively large, continuous state domains. There is some previous work indicating that different sampling mechanisms make a significant difference in sample efficiency (Moore

& Atkeson, 1993; Sutton et al., 2008; Gu et al., 2016; Pan et al., 2018).Much more insights and algorithms are needed for this important problem.

The reasons for us to study Dyna are twofold. First, Dyna provides a natural way to separate the model learning and model usage. The latter offers great flexibility in generating and using imagined experiences, as we detail in Chapter 6. Second, Dyna is directly linked with our ultimate objective—learning a good policy/value function. Dyna architecture (Sutton, 1991) is a planning paradigm that naturally interleaves learning and planning by simulating one-step experience to update the action-value function (i.e., how to use a model).

I have the belief that designing algorithms directly learning towards the ultimate goal is a promising and possibly superior research direction. In fact, methods along this routine have been being researched for a long time (Bengio, 1997; Tulabandhula & Rudin, 2013; Farahmand, 2018). Some other planning approaches that indirectly improve the policy through the imagined experiences should finally need some sort of compression for the policy, in the form of either a policy function or a value function. For example, the classic Monte Carlo Tree Search (MCTS) planning does not directly use the estimated return to improve the value/policy function; rather, it simply uses the estimated return of each action to make a decision. Hence, one would ultimately learn a policy/value function as it is too expensive to always do a tree search at each decision time. In contrast, Dyna directly improves the policy or value function—which is the ultimate purpose—by using a model, and only the policy/value function needs to be queried to make a decision.

1.2 Contributions

In this section, we briefly summarize the works which have been done in this thesis. Our contributions can be summarized as follows.

1. In policy evaluation problems, we introduce the approximate second-order method (i.e., Quasi-Newton method) to TD learning. It has a reduced computation and storage cost than least square TD algorithms

and improved sample efficiency than linear TD methods, balancing the two extremes. Newton's method was originally proposed in numerical analysis for root-finding problems. Hence it is naturally applied in optimization problems to find stationary points where the gradient is zero (Boyd & Vandenberghe, 2004). Newton's optimization is known to converge quadratically fast (Nocedal & Wright, 2006) but is expensive in terms of computation and storage costs as it requires inverting the full Hessian matrix at each iteration. The Quasi-Newton method (Broyden, 1972) reduces the computational cost and avoids directly computing and inverting the Hessian matrix.

To keep the sample efficiency of the second-order optimization method but lower its computation and memory cost, we propose to use an incremental matrix approximation method in the second-order TD algorithms. We derive the general form of the second-order TD updating rule, which we call Accelerated Gradient TD (ATD) learning (Pan et al., 2017b).

2. We propose two instances of the ATD algorithm by introducing two methods to incrementally approximate the matrix efficiently in terms of both computation and storage complexity (Pan et al., 2017b,a). Specifically, we bring in the incremental truncated Singular Value Decomposition (SVD) method and develop an incremental matrix sketching method by random projection.

3. We show that our algorithm's expected updating rule has theoretically guaranteed asymptotically unbiased convergent behavior under certain assumptions.

4. Experiments are conducted to show the properties and verify the improved sample efficiency of our algorithms. The experiments show that ATD algorithms: 1) achieve higher sample efficiency than linear TD algorithms and have lower computational cost than LSTD algorithms; 2) have reduced hyper-parameter sensitivity; 3) converge to the same

solution LSTD does.

5. In the classic MBRL Dyna architecture, we introduce the Stochastic Gradient Langevin Dynamics (SGLD) sampling (Welling & Teh, 2011) to design a new category of search-control methods, which we call gradient-based search-control. The search-control mechanism essentially decides the sampling distribution of experiences to improve the policy during the planning stage; hence we expect that a smart search-control mechanism should significantly improve the sample efficiency of the planning process.

6. Motivated by empirical observations or theoretical insights, we propose three sampling distributions, resulting in three search-control strategies. The first method is to climb on the value function to acquire high-value states for search-control. In the second method, we propose to get more samples from regions where the value function is more difficult to approximate. The approximation difficulty can be roughly measured by the gradient magnitude of the learned value function w.r.t. these states. Our third method overcomes the limitations of a TD error-based sampling in RL—prioritized experience replay method (Schaul et al., 2016). We leverage theoretical insights from error-based sampling in a conventional supervised learning setting and identify the limitations of applying such sampling method in RL algorithms: outdated priorities and insufficient sample space coverage issues. We then introduce the SGLD sampling method, which does not have such limitations. We empirically verify the efficacy of our proposed algorithms.

1.3 Thesis Organization

In addition to this introductory chapter, the thesis is structured first to provide some background of RL in a policy evaluation setting, followed by presenting our idea of developing the Accelerated gradient TD (ATD) learning algorithms. We then provide two concrete instances of our algorithm. After describing our development in the policy evaluation setting, we switch to the control setting

and present our gradient-based search-control mechanisms. We conclude the thesis by discussing the limitations of our method and potential future directions.

Chapter 2 Background in Reinforcement Learning

This chapter presents essential background in the Markov decision processes, temporal difference learning algorithms for policy evaluation problems in both the tabular case and linear function approximation setting, and relevant theoretical properties of these algorithms. Knowing these basic algorithms helps readers understand the motivation of our ATD algorithm. Their theoretical properties help readers understand the policy evaluation optimization objective introduced in the next chapter.

Chapter 3 Accelerated Gradient Temporal Difference Learning

This chapter presents our optimization objective function, based on which we derive the second-order updating rule of temporal difference learning. It presents empirical verification of this second-order optimization idea, followed by discussing the exposed challenges when using an approximate second-order updating rule, which involves a preconditioning matrix approximation. Then the generic form of our algorithm is introduced, based on which we develop concrete instances in the following two chapters that are able to tackle the challenges.

Chapter 4 Approximation by Incremental Truncated SVD

This chapter introduces the first instance of our ATD algorithm. It starts by describing using an incremental truncated SVD technique as a matrix approximation strategy. It then provides a theoretical and empirical analysis of applying such a strategy in the LSTD method, an essential baseline of our algorithm. It then points out the baseline's limitation and motivates our ATD algorithm with the SVD technique to approximate the preconditioning technique incrementally, which overcome the limitation. Both empirical evidence and convergence analysis are provided to show the efficacy of our algorithm.

Chapter 5 Approximation by Random Projection

This chapter introduces the second instance of our ATD algorithm. Its starts by describing using random projection as a matrix approximation strat-

egy. It includes an overview of the application of random projection in RL and discusses various ways of using such a technique in solving a linear system and its weaknesses. Then it presents our specific approach of using random projection in ATD, followed by showing the algorithm's theoretical properties. Empirical results are presented to show the algorithm's high performance and investigate the impact of feature properties on random projection in a linear value function approximation setting.

Chapter 6 Gradient-based Search-control in Dyna

This chapter switches to the control problem and presents the idea of using the Stochastic Gradient Langevin Dynamics (SGLD) method to flexibly design sampling distributions, which are implemented as the search-control mechanism in a classic MBRL framework called Dyna. The chapter starts with reviewing background in control problems and Dyna architecture and its search-control mechanism. Then it introduces three specific state sampling distributions to generate imagined experiences based on the SGLD sampling method, along with empirical or theoretical motivations or both. The three sampling distributions lead to three Dyna variants. This chapter concludes by showing empirical results to compare the sample efficiencies of those Dyna variants and model-free baselines on various benchmark domains.

Chapter 7 Discussion

This chapter summarizes the work that has been presented in this thesis. Additionally, it discusses the limitations of our methods and briefly introduces other efforts we have made but not detailed to address this thesis's topic question. It concludes by describing some future research directions.

Chapter 2

Background in Reinforcement Learning

This chapter introduces background in Reinforcement Learning (RL). It is mainly based on several classic publications in this area (Szepesvari, 2010; Sutton & Barto, 2018; Tsitsiklis & Van Roy, 1997). We first introduce background in the Markov decision process (MDP), which is the main theoretical framework for RL. We then sequentially introduce the following topics: the tabular Temporal Difference (TD) learning algorithm, TD under linear function approximation, and discussions about its convergent behaviors.

The reason for such organization of this chapter is as follows. Arguably, the most distinguished feature of RL algorithms lies in the class of TD learning methods, which leverages the temporal relation structure within a data stream. Unlike many other machine learning algorithms, the initial TD algorithms (Sutton, 1988) are not derived by directly optimizing some objective function. Instead, the theoretical understanding of TD's optimization objective in the linear function approximation setting comes later than the algorithm was proposed.

This chapter is organized based on this understanding. Hence, we start by introducing basic concepts in an MDP. Then we review the most basic TD algorithms and discuss their convergent behavior, which leads to the policy evaluation objective we will be using in the next chapter.

2.1 Markov Decision Processes

We model the interaction between an RL agent and its environment as a Markov decision process $(\mathcal{S}, \mathcal{A}, \mathrm{P}, r, \gamma)$ (Szepesvari, 2010), where $\mathcal{S}$ denotes the set of states, $\mathcal{A}$ denotes the set of actions, and $\mathrm{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, \infty)$ is the transition function which encodes the one-step state transition dynamics. On each discrete time step $t = 1, 2, 3, ...$, the agent selects an action according to its *behavior policy*, $A_t \sim \mu(S_t, \cdot)$, with $\mu : \mathcal{S} \times \mathcal{A} \to [0, \infty)$ and the environment responds by transitioning into a new state S_{t+1} according to P, that is $S_{t+1} \sim \mathrm{P}(\cdot|S_t = s_t, A_t = a_t)$, and emits a scalar reward $R_{t+1} \stackrel{\text{def}}{=} r(S_t, A_t, S_{t+1})$. The discount factor $\gamma : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to (0, 1]$ is used to compute the discounted return as follows. The *return*, denoted by $G_t \in \mathbb{R}$ is the discounted sum of future rewards given actions are selected according to π:

$$\begin{aligned} G_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1} R_{t+2} + \gamma_{t+1}\gamma_{t+2} R_{t+3} + ... \\ &= R_{t+1} + \gamma_{t+1} G_{t+1} \end{aligned} \tag{2.1}$$

where we denote $\gamma(S_t, A_t, S_{t+1}) \stackrel{\text{def}}{=} \gamma_{t+1}$ as a shorthand. Such generalization to transition-based discounting enables the unification of episodic and continuing tasks (White, 2017) and so we adopt it here for generality. In the standard continuing case, $\gamma_t = \gamma$ for some constant $\gamma < 1$ and for a standard episodic setting, $\gamma_t = 1$ until the end of an episode, at which point $\gamma_{t+1} = 0$, ending the infinite sum in the return. In practice, however, this may be a tunable parameter for optimizing algorithm performances. Readers can consider it as a constant in the rest of this thesis. In general, there are two types of problems in RL: the policy evaluation (also called prediction) problem and the control problem.

The objective under policy evaluation is to estimate the *value function*, $v_\pi : \mathcal{S} \to \mathbb{R}$, as the expected return from each state under some *target policy* $\pi : \mathcal{S} \times \mathcal{A} \to [0, \infty)$:

$$v^\pi(s) \stackrel{\text{def}}{=} \mathbb{E}[G_t | S_t = s, \pi],$$

so the expectation is defined over the future states encountered while selecting actions according to π.

The objective for control problems is to find an optimal policy. A policy π^* is an optimal policy if $v^{\pi^*}(s) \geq v^{\pi}(s), \forall s \in \mathcal{S}$ for all π. Under the function approximation setting, one may define the objective as the weighted sum of states' values. The weighting scheme may (Degris et al., 2012b; Imani et al., 2018; Zhang et al., 2019) or may not depend on a policy (Sutton et al., 1999; Silver et al., 2014).

We focus on the policy evaluation problem for now and defer the background review of control problems to Chapter 6. Readers can see that it is not difficult to turn a policy evaluation algorithm into a control algorithm.

In the most common *on-policy* evaluation setting $\pi = \mu$, otherwise $\pi \neq \mu$ and policy evaluation problem is said to be *off-policy*. In the rest of this chapter, we assume an *on-policy policy evaluation* setting by default.

2.2 Tabular Temporal Difference Learning

This section introduces Temporal Difference (TD) algorithms in tabular case followed by an extension to linear function approximation setting in the next section. For mathematical conciseness, unless otherwise specified, we assume the on-policy case. For notation consistency, we consider the feature vector of a state $S_t \in \mathcal{S}$ in the tabular case as a standard basis vector $\mathbf{x}(S_t)$ (i.e., an one-hot vector where only one entry is set to one to denote a particular state) and we write $\mathbf{x}_t$ for short. As a result, in tabular case, a state value can be defined as: $v^{\pi}(s) \stackrel{\text{def}}{=} \mathbf{x}(s)^{\top}\mathbf{w}$. In the continuous state space case, with linear function approximation, we consider $\mathbf{x} : \mathcal{S} \mapsto \mathbb{R}^d$ as some feature mapping, and a state value can still be defined the same as above.

2.2.1 TD(0) algorithm

For a fixed policy π, the value function $v^{\pi}(s)$ satisfy the Bellman equations

$$\forall s \in \mathcal{S}, v^{\pi}(s) = r^{\pi}(s) + \gamma \sum_{s'} \mathrm{P}^{\pi}(s'|s) v^{\pi}(s'),$$

where

$$\mathrm{P}^{\pi}(s'|s) \stackrel{\text{def}}{=} \sum_{a} \pi(a|s)\mathrm{P}(s'|s,a), r^{\pi}(s) \stackrel{\text{def}}{=} \sum_{a}\sum_{s'} \pi(a|s) P(s'|s,a) r(s,a,s')$$

is the expected one-step reward from state s. The Bellman equation can be proved from the definition of value function $v^\pi(s) \stackrel{\text{def}}{=} \mathbb{E}[G_t|S_t = s, \pi]$ (Sutton & Barto, 2018).

Under policy π, for any value function $v \in \mathbb{R}^{|\mathcal{S}|}$, one can define the Bellman operator as:

$$\forall s \in \mathcal{S}, T[v](s) \stackrel{\text{def}}{=} r^\pi(s) + \gamma \sum_{s'} \mathrm{P}^\pi(s'|s) v^\pi(s'). \tag{2.2}$$

The Bellman equation serves as the basis for TD learning. Consider that we initialize a value function $\hat{v}^\pi(\cdot)$, i.e., a table with $|\mathcal{S}|$ states recording the value of every state. Given a state value estimate $\hat{v}^\pi(s)$, while we do not have a target directly available from the data set as we do in a supervised learning setting, we can push our value estimate towards the target $r^\pi(s) + \gamma \sum_{s'} \mathrm{P}^\pi(s'|s)\hat{v}^\pi(s')$, which is known as the **Temporal Difference (TD) target**. This target can be stochastically sampled given a fixed policy once we have a sample in the form of state, next state, and reward s, s', r, resulting in the following updating rule known as TD learning (Sutton, 1988):

$$\hat{v}^\pi(s) \leftarrow \hat{v}^\pi(s) + \alpha(y^{TD} - \hat{v}^\pi(s))$$

where α is some learning rate and y^{TD} is estimated as $r + \gamma \hat{v}^\pi(s')$.

In a matrix format, one can write the Bellman operator as $T\mathbf{v} = \mathbf{r}^\pi + \gamma P^\pi \mathbf{v}$, where $\mathbf{r}, \mathbf{v}$ are vectors in $\mathbb{R}^{|\mathcal{S}|}$ so each index can retrieve the quantity of corresponding state. From this definition, it is easy to prove the operator is a maximum norm contraction when $\gamma \in (0, 1)$. By using the Bellman operator, the Bellman equation can be rewritten as $T\mathbf{v} = \mathbf{v}$. It is easy to see that given two value functions $\mathbf{v}_1, \mathbf{v}_2$, the operator T is γ-contraction: $||T\mathbf{v}_1 - T\mathbf{v}_2|| = ||\gamma P^\pi(\mathbf{v}_1 - \mathbf{v}_2)|| \leq \gamma||v_1 - v_2||$. This contraction property holds in both maximum norm (i.e., $||\cdot||_\infty$, see Szepesvari (2010, Page 79)) and stationary distribution weighted norm (see White (2017, Lemma 2.)) (i.e., $||\mathbf{v}||^2_{D^\pi} \stackrel{\text{def}}{=} \mathbf{v}^\top D^\pi \mathbf{v}$ and D^π is a diagonal matrix where the diagonal elements denote the stationary distribution induced by policy π). The contraction property of T can guarantee the sequence $\{\mathbf{v}_i\}$ generated by the Bellman operator is convergent to the unique fixed point of T (see Szepesvari (2010, Page 77,

Theorem 1) for technical details). The contraction property of the Bellman operator is the theoretical foundation for showing the convergence of TD algorithms.

2.2.2 TD(λ) algorithm

Implementing the above algorithm uses only one-step sampling to compute the TD target of a state value, which may introduce a large bias as the one-step sampling heavily relies on the estimate of next state's value. On the other extreme, one might think of using a Monte Carlo return, which is unbiased but potentially introduces large variance. As a result, it is reasonable to consider that using a weighted sum of different steps of reward samples may provide a better target by flexibly balancing bias-variance trade-off. This motivates the λ-weighted return:

$$G_t^\lambda \stackrel{\text{def}}{=} (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n}$$

and by considering post-termination terms separately, we have

$$G_t^\lambda \stackrel{\text{def}}{=} (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_{t:t+n} + \lambda^{T-t-1}G_t, \tag{2.3}$$

where

$$G_{t:t+n} \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + ... + \gamma_{n-1}R_{t+n} + \gamma^n v^\pi(S_{t+n}), 0 \le t \le T-n$$

is called n-step return. The λ parameter, which is typically called *bootstrap parameter*, assigns different weights to returns calculated by different steps.

This λ-weighted return induces another λ-weighted Bellman operator. One can define T^λ operator as following (assume every notation is under some fixed policy). First define an operator

$$T^{(n)}[v](s) \stackrel{\text{def}}{=} \mathbb{E}[\sum_{t=0}^{n}\gamma^t R_{t+1} + \gamma^{n+1}v(S_{n+1})|S_0 = s].$$

Then the operator $T^{(\lambda)}$ is defined as:

$$\begin{aligned} T^{(\lambda)}[v](s) &\stackrel{\text{def}}{=} (1-\lambda)\sum_{n=0}^{\infty}\lambda^n T^{(n)}[v](s) \\ &= (1-\lambda)\sum_{n=0}^{\infty}\lambda^n \mathbb{E}[\sum_{t=0}^{n}\gamma^t R_{t+1} + \gamma^{n+1}v(S_{n+1})|S_0 = s] \\ &= (1-\lambda)\sum_{n=0}^{\infty}\lambda^n (\sum_{t=0}^{n}\gamma^t \mathbb{E}[R_{t+1}|S_0 = s] + \gamma^{n+1}\mathbb{E}[v(S_{n+1})|S_0 = s]). \end{aligned}$$

The switch between summation and integration is done by the Tonelli–Fubini theorem (Tsitsiklis & Van Roy, 1997). By rewriting the above operator in matrix format, we have

$$T^{(\lambda)}\mathbf{v} = (1-\lambda)\sum_{n=0}^{\infty}\lambda^n(\sum_{t=0}^{n}(\gamma P^{\pi})^t\mathbf{r}^{\pi} + (\gamma P^{\pi})^{n+1}\mathbf{v}). \tag{2.4}$$

Again, this $T^{(\lambda)}$ operator is still a contraction mapping because

$$\begin{aligned} ||T^{(\lambda)}\mathbf{v}_1 - T^{(\lambda)}\mathbf{v}_2|| &= ||(1-\lambda)\sum_{n=0}^{\infty}\lambda^n(\gamma P^{\pi})^{n+1}(\mathbf{v}_1 - \mathbf{v}_2)|| \\ &\leq (1-\lambda)\sum_{n=0}^{\infty}\lambda^n\gamma^{n+1}||\mathbf{v}_1 - \mathbf{v}_2|| \\ &= \frac{\gamma(1-\lambda)}{1-\gamma\lambda}||\mathbf{v}_1 - \mathbf{v}_2||. \end{aligned}$$

As a result, it has a unique fixed point by Szepesvari (2010, Theorem 1, Page 77).

It is not difficult to see that, in practice, one has to wait for some time steps to acquire an estimate of the λ-return. Fortunately, a forward view can be derived to enable online update of value estimate in TD(λ) algorithm.

In order to introduce the vanilla TD(λ) algorithm, we slightly abuse the definition of λ return by the following:

$$G_t^{\lambda} \stackrel{\text{def}}{=} R_{t+1} + \gamma((1-\lambda)v^{\pi}(S_{t+1}) + \lambda G_{t+1}^{\lambda}). \tag{2.5}$$

This definition (Sutton et al., 2009; Maei, 2011) allows us to derive the backward view to estimate λ return in an online manner. By (2.5), we have

$$\delta_t^{\lambda} \stackrel{\text{def}}{=} G_t^{\lambda} - v^{\pi}(S_t) = \delta_t + \gamma\lambda\delta_{t+1}^{\lambda},$$

where

$$\delta_t \stackrel{\text{def}}{=} \delta(S_t, A_t, S_{t+1}, R_{t+1}, \gamma) \stackrel{\text{def}}{=} R_{t+1} + \gamma v^{\pi}_{\mathbf{w}}(S_{t+1}) - v^{\pi}_{\mathbf{w}}(S_t).$$

If we are able to estimate the δ_t^λ, then we can use the term $\delta_t^\lambda \mathbf{x}_t$ to update the weight vector. We take the expectation of $\delta_t^\lambda \mathbf{x}_t$ to get the backward view:

$$\begin{aligned}
\mathbb{E}[\delta_t^\lambda \mathbf{x}_t] &= \mathbb{E}[\delta_t \mathbf{x}_t] + \gamma\lambda \mathbb{E}[\delta_{t+1}^\lambda \mathbf{x}_t] && (2.6)\\
&= \mathbb{E}[\delta_t \mathbf{x}_t] + \gamma\lambda \mathbb{E}[\delta_t^\lambda \mathbf{x}_{t-1}] && (2.7)\\
&= \mathbb{E}[\delta_t \mathbf{x}_t] + \gamma\lambda (\mathbb{E}[\delta_t \mathbf{x}_{t-1}] + \gamma\lambda \mathbb{E}[\delta_{t+1}^\lambda \mathbf{x}_{t-1}]) && (2.8)\\
&= \mathbb{E}[\delta_t \mathbf{e}_t], \mathbf{e}_t \stackrel{\text{def}}{=} \mathbf{x}_t + \gamma\lambda \mathbf{e}_{t-1}, && (2.9)
\end{aligned}$$

where the last equality can be acquired by index switching an infinite number of times. $\mathbf{e}_t$ is called *accumulating* eligibility-trace vector, which reflects how much each state value should be adjusted once a new sample is acquired. This expectation allows us to get an estimate of $\mathbb{E}[\delta_t^\lambda \mathbf{x}_t]$ in an online manner (i.e. at each time step). Due to the consideration of practical robustness and convergence performance, *replacing* trace is often used by bounding the magnitude of entries of the trace vector no larger than one. That is, at current time step t, we down-weight all other states' trace value by $\gamma\lambda$ and set the trace value of the state S_t to one.

Note that though we still consider tabular case and $\mathbf{x}_t \in \mathbb{R}^{|\mathcal{S}|}$ is a standard basis vector, the value of a state can be written the same as we do in a linear function approximation setting. Hence this the above theoretical result can be extended to the linear function approximation setting in Section 2.3.

We would like to to provide a few more discussions about the trace vector, as it significantly affects the sample efficiency and this is an area being actively studied. The work by van Seijen & Sutton (2014) shows that the exact equivalence between the estimates of $\mathbb{E}[\delta_t^\lambda \mathbf{x}_t]$ and $\mathbb{E}[\delta_t \mathbf{e}_t]$ cannot be achieved when running the above TD algorithm unless there is no bootstrap (i.e. $\lambda = 0$). Matching the estimated weighted return with the Monte Carlo return truncated at the current time step motivates the *True-online* TD(λ) algorithm. The idea of true-online trace is to give up using the return estimated by infinite horizon; instead, we estimate all visited states' values by using a truncated

return up to the current time step t. Denote the weight vector $\mathbf{w}_k^t$ as the one calculated by using data up to time t for the kth (i.e. $k \leq t$) sample. At each time step t, we need to perform updating t times: $\mathbf{w}_1^t, \mathbf{w}_2^t, ..., \mathbf{w}_t^t$ by using the samples $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t$. True-online TD (van Seijen & Sutton, 2014) derives a backward view to allow a single update at each time step and the resulting estimated bootstrap return matches with the one from forward view. We refer readers to van Seijen et al. (2016) for a more detailed discussion about the true-online trace and to van Hasselt et al. (2021) for the most recent development in eligibility traces.

2.3 Linear Temporal Difference Learning

In this section, we discuss the TD learning algorithm with linear function approximation and its theoretical properties. The terminologies are closely following the work by Tsitsiklis & Van Roy (1997).

In order to handle large state space, some feature mapping can be used to convert the raw state variables to some feature space: $\mathbf{x} : \mathcal{S} \mapsto \mathbb{R}^d$ to enable generalization. We present the updating rules of $TD(\lambda)$ in linear function approximation setting as following:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t \delta_t \mathbf{e}_t, \tag{2.10}$$

$$\mathbf{e}_{t+1} = \gamma\lambda \mathbf{e}_t + \mathbf{x}_t. \tag{2.11}$$

We now focus on analyzing the convergent behavior of the algorithm, which leads to the discovery of the Mean Squared Projected Bellman Error (MSPBE) introduced in the next chapter. We first introduce some basic setup for analysis and then present the algorithm's deterministic version from the dynamic system perspective. Last, we discuss the convergent behavior of the stochastic version of the algorithm.

Basic Setup. Let $\mathbf{X} \subset \mathbb{R}^{|\mathcal{S}| \times d}$ be the feature matrix by applying $\mathbf{x}$ on each of the state in $\mathcal{S}$. Define the norm $||\cdot||_{D^\pi} \stackrel{\text{def}}{=} \sqrt{\langle \cdot, \cdot \rangle_{D^\pi}}$ and the space of finite value functions as $\mathcal{L}_2 \stackrel{\text{def}}{=} \{\mathbf{v} \in \mathbb{R}^{|\mathcal{S}|} | ||\mathbf{v}||_{D^\pi} < \infty\}$. Note that we assume that

the true value function under policy π is in this space: $\mathbf{v}^\pi \in \mathcal{L}_2$. We introduce the following basic setup.

1. Assume that the value function by linear approximation lies in $\mathcal{L}_2$, i.e. $col(\mathbf{X}) \stackrel{\text{def}}{=} \{\mathbf{X}\mathbf{w} | \mathbf{w} \in \mathbb{R}^d\} \subset \mathcal{L}_2$ and $\mathbf{X}$ has full column rank, i.e., the features are linearly independent.

2. $\forall \mathbf{v} \in \mathcal{L}_2$ the set $\text{argmin}_{\mathbf{v}' \in col(\mathbf{X})} ||\mathbf{v}' - \mathbf{v}||_{D^\pi}$ has a unique element. We call this the projection of $\mathbf{v}$ on $col(\mathbf{X})$ w.r.t. the norm $||\cdot||_{D^\pi}$ as detailed later.

3. The second-order moment of reward is finite. This ensures that the true value function under the target policy π lies in $\mathcal{L}_2$ and our space $\mathcal{L}_2$ is well defined.

4. The MDP is irreducible and aperiodic. This is a common assumption needed to characterize the limiting behavior of a policy and establish convergence results.

Deterministic Version of TD(λ). We provide simple theoretical analysis for linear TD algorithms by considering its deterministic version (i.e., expected updating rule). This analysis should equip readers with basic knowledge of TD algorithms' convergence mechanisms and a well-known policy evaluation objective—the mean squared projected Bellman error.

Denote the updating rule of the deterministic (i.e., expected) version as: $\bar{\mathbf{w}}_{t+1} = \bar{\mathbf{w}}_t + \alpha_t \mathbb{E}_\pi[\delta_t \mathbf{e}_t]$. Given the weight vector $\bar{\mathbf{w}}$, we can further write the expectation $\mathbb{E}_\pi[\delta_t \mathbf{e}_t]$ in a matrix format:

$$\begin{aligned}
\mathbb{E}_\pi[\delta_t(\bar{\mathbf{w}})\mathbf{e}_t] &= \mathbb{E}_\pi[(G_t^\lambda - \mathbf{x}_t^\top \bar{\mathbf{w}})\mathbf{x}_t] && (2.12)\\
&= \mathbb{E}_\pi[(T^{(\lambda)}[v^\pi](S_t) - v^\pi(S_t))\mathbf{x}_t] && (2.13)\\
&= \sum_s d^\pi(s)(T^{(\lambda)}[v^\pi](s) - v^\pi(s))\mathbf{x}(s) && (2.14)\\
&= \mathbf{X}^\top D^\pi (T^{(\lambda)}[\mathbf{X}\bar{\mathbf{w}}] - \mathbf{X}\bar{\mathbf{w}}). && (2.15)
\end{aligned}$$

The last equality is done by observing that the vector $\sum_s d^\pi(s)(T^{(\lambda)}[v^\pi](s) - v^\pi(s))\mathbf{x}(s)$ is generated by weighted sum of the feature vectors of all states

and the weight for state s is $d^\pi(s)(T^{(\lambda)}[v^\pi](s) - v^\pi(s))$. Hence we can take the transpose of the feature matrix and multiply it by a vector $D^\pi(T^{(\lambda)}[\mathbf{X}\bar{\mathbf{w}}] - \mathbf{X}\bar{\mathbf{w}})$ where each element is the weight corresponding to a state. We can alternatively write this expectation term as $\mathbf{b} - \mathbf{A}\bar{\mathbf{w}}$ by the following reasoning:

$$\begin{aligned}
\mathbb{E}_\pi[\delta_t(\bar{\mathbf{w}})\mathbf{e}_t] &= \mathbb{E}_\pi[(R_{t+1} + \gamma \mathbf{x}_{t+1}^\top \bar{\mathbf{w}} - \mathbf{x}_t^\top \bar{\mathbf{w}})\mathbf{e}_t] && (2.16)\\
&= \mathbb{E}_\pi[R_{t+1}\mathbf{e}_t] + \mathbb{E}_\pi[(\gamma \mathbf{x}_{t+1}^\top - \mathbf{x}_t^\top)\bar{\mathbf{w}}\mathbf{e}_t] && (2.17)\\
&= \mathbb{E}_\pi[R_{t+1}\mathbf{e}_t] - \mathbb{E}_\pi[\mathbf{e}_t(\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^\top]\bar{\mathbf{w}} && (2.18)\\
&= \mathbf{b} - \mathbf{A}\bar{\mathbf{w}}, && (2.19)
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{b} &\stackrel{\text{def}}{=} \mathbb{E}_\pi[R_{t+1}\mathbf{e}_t], && (2.20)\\
\mathbf{A} &\stackrel{\text{def}}{=} \mathbb{E}_\pi[\mathbf{e}_t(\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^\top]. && (2.21)
\end{aligned}$$

Notice that, the two ways of expressing the expectation term can help prove that $\mathbb{E}_\pi[\delta_t(\mathbf{w}^*)\mathbf{e}_t] = 0$ as we demonstrate later for the optimal weight vector $\mathbf{w}^*$ (i.e., the fixed point of the deterministic updating rule). As a result, we first characterize the optimal weight vector (i.e., find what $\mathbf{w}^*$ is). We show that this optimal weight vector is indeed the stationary point of the expected updating rule.

Projection operator. We need to introduce a projection operator before deriving the optimal weight vector. Fix an arbitrary vector $\mathbf{v} \in \mathcal{L}_2$. We want to project it onto $\{\mathbf{X}\mathbf{w} | \mathbf{w} \in \mathbb{R}^d\} \stackrel{\text{def}}{=} col(\mathbf{X})$. Let the projection operator be Π and $\mathbf{X}\mathbf{p}$ be the projection where $\mathbf{p} \in \mathbb{R}^d$. Then $\forall \mathbf{w} \in \mathbb{R}^d$, we have $\mathbf{X}\mathbf{w} \in col(\mathbf{X})$ and $\langle \mathbf{X}\mathbf{p} - \mathbf{v}, \mathbf{X}\mathbf{w} \rangle_{D^\pi} = 0$. The latter is by the definition of the projection: $\text{argmin}_{\mathbf{v}' \in col(\mathbf{X})} ||\mathbf{v}' - \mathbf{v}||_{D^\pi}$ since the D^π-orthogonal projection can minimize this norm. That is $\forall \mathbf{w} \in \mathbb{R}^d$,

$$\langle \mathbf{X}\mathbf{p} - \mathbf{v}, \mathbf{X}\mathbf{w} \rangle_{D^\pi} = (\mathbf{X}\mathbf{p} - \mathbf{v})^\top D^\pi \mathbf{X}\mathbf{w} = 0.$$

This indicates that the vector $\mathbf{X}^\top D^\pi(\mathbf{X}\mathbf{p} - \mathbf{v}) = \mathbf{0}$. Then

$$\begin{aligned}
\mathbf{X}^\top D^\pi \mathbf{X}\mathbf{p} &= \mathbf{X}^\top D^\pi \mathbf{v},\\
\mathbf{p} &= (\mathbf{X}^\top D^\pi \mathbf{X})^{-1} \mathbf{X}^\top D^\pi \mathbf{v}.
\end{aligned}$$

As a result, the projection is $\mathbf{Xp} = \mathbf{X}(\mathbf{X}^\top D^\pi \mathbf{X})^{-1}\mathbf{X}^\top D^\pi \mathbf{v}$. We call the matrix $\mathbf{X}(\mathbf{X}^\top D^\pi \mathbf{X})^{-1}\mathbf{X}^\top D^\pi$ the projection matrix as it projects the vector $\mathbf{v}$ onto $col(\mathbf{X})$. We use the symbol Π to denote it:

$$\Pi \stackrel{\text{def}}{=} \mathbf{X}(\mathbf{X}^\top D^\pi \mathbf{X})^{-1}\mathbf{X}^\top D^\pi.$$

Finding the fixed point to the deterministic version of TD updating rule. It is easy to show that the projection operator Π is non-expansive. Fix arbitrary $\mathbf{v}_1, \mathbf{v}_2 \in \mathcal{L}_2$, by Babylonian-Pythagorean theorem, we have

$$||\Pi\mathbf{v}_1 - \Pi\mathbf{v}_2||^2_{D^\pi} + ||\Pi\mathbf{v}_1 - \mathbf{v}_1||^2_{D^\pi} = ||\mathbf{v}_1 - \mathbf{v}_2||^2_{D^\pi},$$

hence,

$$||\Pi\mathbf{v}_1 - \Pi\mathbf{v}_2||^2_{D^\pi} \leq ||\mathbf{v}_1 - \mathbf{v}_2||^2_{D^\pi}.$$

This can be combined with the contraction property of $T^{(\lambda)}$ to show that the composite operator $\Pi T^{(\lambda)}$ is a contraction, hence a unique fixed point of this operator can be guaranteed. By the assumption that $\mathbf{X}$ has full column rank, there exists unique $\mathbf{w}^*, s.t. \Pi T^{(\lambda)}\mathbf{X}\mathbf{w}^* = \mathbf{X}\mathbf{w}^*$. This is also the rationality behind minimizing the Mean squared projected Bellman error introduced in the next chapter, i.e., $\min_{\mathbf{w}} ||\Pi T^{(\lambda)}\mathbf{X}\mathbf{w}^* - \mathbf{X}\mathbf{w}^*||^2_{D^\pi}$.

Notice that, this unique fixed point result, together with the two ways of expressing the expectation term as shown in (2.12) and (2.16) can help prove that $\mathbf{b} - \mathbf{A}\mathbf{w}^* = 0$ by the following simple reasoning. Multiplying both sides of $\Pi T^{(\lambda)}\mathbf{X}\mathbf{w}^* = \mathbf{X}\mathbf{w}^*$ by $\mathbf{X}^\top D^\pi$ gives us $\mathbf{X}^\top D^\pi T^{(\lambda)}\mathbf{X}\mathbf{w}^* = \mathbf{X}^\top D^\pi \mathbf{X}\mathbf{w}^*$. Hence $\mathbb{E}_\pi[\delta_t(\mathbf{w}^*)\mathbf{e}_t] = 0$ and hence $\mathbf{A}\mathbf{w}^* = \mathbf{b}$. One can further prove the matrix $\mathbf{A}$ is positive definite by checking certain properties of it (Sutton et al., 2016; Yu, 2015).

The concrete matrices format of $\mathbf{A}$ and $\mathbf{b}$ can be derived as follows. For $\mathbf{A} \stackrel{\text{def}}{=} \mathbb{E}_\pi[\mathbf{e}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^\top] = \mathbb{E}_\pi[\mathbf{e}_t\mathbf{x}_t^\top] - \mathbb{E}_\pi[\gamma\mathbf{e}_t\mathbf{x}_{t+1}^\top]$, we can separately analyze the two expectation terms. For $\mathbb{E}_\pi[\mathbf{e}_t\mathbf{x}_t^\top]$, observing that the expectation is

constant for any time step t, we can write it as[1]

$$\begin{aligned}\mathbb{E}_\pi[\mathbf{e}_t\mathbf{x}_t^\top] &= \mathbb{E}_\pi[\mathbf{e}_0\mathbf{x}_0^\top] \\ &= \mathbb{E}_\pi[\sum_{\tau=-\infty}^{0}(\gamma\lambda)^{-\tau}\mathbf{x}_\tau\mathbf{x}_0^\top] \\ &= \sum_{\tau=-\infty}^{0}(\gamma\lambda)^{-\tau}\mathbb{E}_\pi[\mathbf{x}_\tau\mathbf{x}_0^\top].\end{aligned}$$

Fix arbitrary time indexes t, τ such that $t \geq \tau$, we have

$$\begin{aligned}\mathbb{E}_\pi[\mathbf{x}_\tau\mathbf{x}_t^\top] &= \sum_s d^\pi(s)\mathbb{E}_\pi[\mathbf{x}(S_\tau)\mathbf{x}(S_t)^\top|S_\tau = s] \\ &= \sum_s d^\pi(s)\mathbf{x}(s)\mathbb{E}_\pi[\mathbf{x}(S_t)^\top|S_\tau = s] \\ &= \mathbf{X}^\top D^\pi(\mathbf{P}^\pi)^{t-\tau}\mathbf{X}.\end{aligned}$$

As a result,

$$\begin{aligned}\mathbb{E}_\pi[\mathbf{e}_t\mathbf{x}_t^\top] &= \sum_{\tau=-\infty}^{0}(\gamma\lambda)^{-\tau}\mathbb{E}_\pi[\mathbf{x}_\tau\mathbf{x}_0^\top] \\ &= \sum_{\tau=-\infty}^{0}(\gamma\lambda)^{-\tau}\mathbf{X}^\top D^\pi(\mathbf{P}^\pi)^{-\tau}\mathbf{X} \\ &= \sum_{\tau=0}^{\infty}(\gamma\lambda)^{\tau}\mathbf{X}^\top D^\pi(\mathbf{P}^\pi)^{\tau}\mathbf{X}.\end{aligned}$$

For the other term, following similar reasoning, we get

$$\mathbb{E}_\pi[\gamma\mathbf{e}_t\mathbf{x}_{t+1}^\top] = \gamma\sum_{\tau=0}^{\infty}(\gamma\lambda)^{\tau}\mathbf{X}^\top D^\pi(\mathbf{P}^\pi)^{\tau+1}\mathbf{X}.$$

[1] In the Emphatic TD paper (Sutton et al., 2016), the matrix format of the $\mathbf{A}$ can be derived by considering $\mathbb{E}_\pi[\mathbf{e}_t\mathbf{x}_t^\top] = \sum_s d^\pi(s)\mathbb{E}_\pi[\mathbf{e}_t|S_t = s]\mathbb{E}[\mathbf{x}_t|S_t = s]^\top$ because the two random variables are conditionally independent. That derivation gives the same result as the one presented here.

Then the matrix expression of $\mathbf{A}$ can be found as follows.

$$\begin{aligned}
\mathbf{A} &\stackrel{\text{def}}{=} \mathbb{E}_\pi[\mathbf{e}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^\top] \\
&= \mathbb{E}_\pi[\mathbf{e}_t\mathbf{x}_t^\top] - \mathbb{E}_\pi[\gamma\mathbf{e}_t\mathbf{x}_{t+1}^\top] \\
&= \sum_{\tau=0}^{\infty}(\gamma\lambda)^\tau\mathbf{X}^\top D^\pi(\mathbf{P}^\pi)^\tau\mathbf{X} - \gamma\sum_{\tau=0}^{\infty}(\gamma\lambda)^\tau\mathbf{X}^\top D^\pi(\mathbf{P}^\pi)^{\tau+1}\mathbf{X} \\
&= \mathbf{X}^\top D^\pi\Big(\sum_{\tau=0}^{\infty}(\gamma\lambda)^\tau(\mathbf{P}^\pi)^\tau - \gamma(\gamma\lambda)^\tau(\mathbf{P}^\pi)^{\tau+1}\Big)\mathbf{X} \\
&= \mathbf{X}^\top D^\pi\Big(\sum_{\tau=0}^{\infty}(\gamma\lambda)^\tau(\mathbf{P}^\pi)^\tau(\mathbf{I} - \gamma\mathbf{P}^\pi)\Big)\mathbf{X} \\
&= \mathbf{X}^\top D^\pi(\mathbf{I} - \gamma\lambda\mathbf{P}^\pi)^{-1}(\mathbf{I} - \gamma\mathbf{P}^\pi)\mathbf{X}.
\end{aligned}$$

As for the $\mathbf{b} \stackrel{\text{def}}{=} \mathbb{E}_\pi[R_{t+1}\mathbf{e}_t]$, we have:

$$\begin{aligned}
\mathbf{b} &= \sum_{\tau=-\infty}^{0}(\gamma\lambda)^{-\tau}\mathbb{E}_\pi[\mathbf{x}_\tau R_1], \\
&\quad \text{by following the same reasoning as above,} \\
\mathbf{b} &= \mathbf{X}^\top D^\pi(\mathbf{I} - \gamma\lambda\mathbf{P}^\pi)^{-1}\mathbf{r}^\pi.
\end{aligned}$$

In practice, the closed-form expressions of those matrices are rarely used, we typically directly estimate their expectation forms by samples: $\mathbb{E}_\pi[\mathbf{e}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^\top]$ and $\mathbb{E}_\pi[R_{t+1}\mathbf{e}_t]$. However, they are quite convenient when theoretically characterizing the convergence properties of TD algorithms.

Proof idea of TD(λ)'s convergence with linear function approximation. The proof is done by some theoretical results from stochastic approximation literature (Borkar & Mitter, 1999; Kushner & Yin, 2003). Typically the result states that an iterative stochastic updating rule in the form of

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha(\mathbf{A}(X_t)\mathbf{w}_t + \mathbf{b}(X_t)), X_t \stackrel{\text{def}}{=} (S_t, S_{t+1}, R_{t+1}, \gamma_{t+1})$$

converges to the solution of $\mathbb{E}_\pi[\mathbf{A}(X_t)]\mathbf{w} = \mathbb{E}_\pi[\mathbf{b}(X_t)]$ under some technical conditions. Those conditions typically include: 1) stepsize condition; 2) bounded noise, i.e., the difference between sampled term and the expected term; 3) positive definiteness of the matrix $\mathbf{A}$. For the purpose of satisfying

some conditions, basic property on the underlying MDP is required, such as ergodicity or irreducibility.

Error bound in terms of the difference between $d(v_w^\pi, v^*)$ and $d(\Pi v^*, v^*)$ (i.e., distance defined in terms of $||\cdot||_{D^\pi}$) can be established easily as an interesting supplementary result as follows.

$$\begin{aligned}
||\mathbf{X}\mathbf{w}^* - \mathbf{v}^\pi||_{D^\pi} &\leq ||\mathbf{X}\mathbf{w}^* - \Pi\mathbf{v}^\pi||_{D^\pi} + ||\Pi\mathbf{v}^\pi - \mathbf{v}^\pi||_{D^\pi} \\
&= ||\Pi T^{(\lambda)}\mathbf{X}\mathbf{w}^* - \Pi\mathbf{v}^\pi||_{D^\pi} + ||\Pi\mathbf{v}^\pi - \mathbf{v}^\pi||_{D^\pi} \\
&\leq ||T^{(\lambda)}\mathbf{X}\mathbf{w}^* - \mathbf{v}^\pi||_{D^\pi} + ||\Pi\mathbf{v}^\pi - \mathbf{v}^\pi||_{D^\pi} \\
&\leq \frac{\gamma(1-\lambda)}{1-\gamma\lambda}||\mathbf{X}\mathbf{w}^* - \mathbf{v}^\pi||_{D^\pi} + ||\Pi\mathbf{v}^\pi - \mathbf{v}^\pi||_{D^\pi},
\end{aligned}$$

where the second equality is done by the fixed point property, the third and the last inequalities are done by the non-expansive property of the projection operator and the contraction property of the $T^{(\lambda)}$ operator, respectively, as proved in the previous discussions. Thus, rearranging the terms in the above inequality can lead to the result:

$$||\mathbf{X}\mathbf{w}^* - \mathbf{v}^\pi||_{D^\pi} \leq \frac{(1-\lambda\gamma)}{1-\gamma}||\Pi\mathbf{v}^\pi - \mathbf{v}^\pi||_{D^\pi}.$$

It characterizes the quality of the solution (i.e., $\mathbf{w}^*$) by bounding its difference with the theoretically best solution in the space $col(\mathbf{X})$. We refer readers to the nice work by Tsitsiklis & Van Roy (1997) for detailed convergence analysis of linear TD algorithms.

Chapter 3

Accelerated Gradient Temporal Difference Learning

Second-order Newton methods are known to speed up gradient descent algorithms significantly (Boyd & Vandenberghe, 2004); in fact, it is not surprising to empirically observe that it can almost shoot the solution within a few iterations. However, it typically involves an expensive computation of some matrix (usually the inverse of a matrix), limiting its practical utility. The Quasi-Newton method leverages the idea of efficiently approximating the preconditioning matrix and has been extensively studied in optimization literature (Broyden, 1972; Nocedal & Wright, 2006). However, we did not see any work deriving an approximate second-order method by directly minimizing mean squared projected Bellman error (MSPBE) before our work. In this sense, our work is the first attempt to fill this gap. The resulting category of algorithm opens a new research line along the preconditioning TD algorithms.

In this chapter, we mainly introduce the idea of bringing in the second-order method to minimize MSPBE (Pan et al., 2017b) and present the generic form of our Accelerated Gradient Temporal Difference Learning (ATD) algorithm. It is known that the linear TD algorithm has a relatively low computational and storage cost comparing with the LSTD method. However, at the same time, it also has a much lower sample efficiency than the LSTD method. Our ATD algorithms interpolate between the two extremes, providing a flexible balance between sample and computation complexity.

This chapter is organized as follows. We firstly introduce the learning ob-

jective we attempt to optimize in Section 3.1. In Section 3.2, we introduce the key idea of adapting the second-order optimization method to TD algorithms to minimize MSPBE and present empirical results to validate this basic idea on benchmark domains. We also discuss using an approximate second-order method to improve computational and storage complexity, followed by a discussion about relevant challenges, motivating our ATD algorithms. In Section 3.3, we formally describe the general form of our ATD algorithms, which will be instantiated and discussed in detail in the next two chapters.

3.1 Mean Squared Projected Bellman Error

As we reviewed in the last chapter, in the case of linear function approximation, the state is represented by fixed length feature vectors $x : \mathcal{S} \to \mathbb{R}^d$, where $\mathbf{x}_t \stackrel{\text{def}}{=} x(S_t)$ and the approximation to the value function is formed as a linear combination of a learned weight vector, $\mathbf{w} \in \mathbb{R}^d$, and $x(S_t)$: $v_\pi(S_t) \approx \mathbf{w}^\top \mathbf{x}_t$. The goal of on-policy policy evaluation is to approximate state values under target policy π by learning $\mathbf{w}$ from samples generated by following behaviour policy $\mu = \pi$.

Recall that the linear TD(λ) algorithm for on-policy policy evaluation converges to the unique fixed point of $\Pi T^{(\lambda)}\mathbf{X}\mathbf{w} = \mathbf{X}\mathbf{w}$ under some mild conditions. This observation inspires the below objective called the Mean Squared Projected Bellman Error (MSPBE):

$$\min_{\mathbf{w}} ||\Pi T^{(\lambda)}\mathbf{X}\mathbf{w} - \mathbf{X}\mathbf{w}||^2_{D^\pi},$$

which is strongly convex under some mild conditions. To allow readers to conveniently compare this objective and the below one introduced for off-policy policy evaluation, we rewrite it into expectation form and matrices form by plugging in the expectation expressions and matrix expressions respectively as introduced in the previous chapter:

$$||\Pi T^{(\lambda)}\mathbf{X}\mathbf{w} - \mathbf{X}\mathbf{w}||^2_{D^\pi} = \mathbb{E}_\pi[\delta_t \mathbf{e}_t]^\top \mathbb{E}_\pi[\mathbf{x}_t \mathbf{x}_t^\top]^{-1} \mathbb{E}_\pi[\delta_t \mathbf{e}_t] \tag{3.1}$$

$$= (\mathbf{b} - \mathbf{A}\mathbf{w})^\top \mathbb{E}_\pi[\mathbf{x}_t \mathbf{x}_t^\top]^{-1} (\mathbf{b} - \mathbf{A}\mathbf{w}), \tag{3.2}$$

where the last equation is because of (2.20). Note that we use δ_t as a shorthand for the TD error $\delta_t(\mathbf{w}) = R_{t+1} + \gamma_{t+1}\mathbf{x}_{t+1}^\top \mathbf{w} - \mathbf{x}_t^\top \mathbf{w}$. In implementation, the middle matrix $\mathbb{E}_\pi[\mathbf{x}_t \mathbf{x}_t^\top]$ may be replaced by any positive definite matrix, because the unique optimum $\mathbf{w}^*$ of this objective function satisfies $\mathbf{b} - \mathbf{A}\mathbf{w}^* = \mathbf{0}$.

For formulation generality, we now introduce the MSPBE objective for off-policy policy evaluation problems, where the behavior policy μ is different from the target policy π. Similar to how we get the MSPBE objective, we can start from the linear TD method again. Recall the expected updating rule of linear TD method: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \mathbb{E}_\pi[\delta_t \mathbf{e}_t]$. Since we now only have access to samples of $\mathbb{E}_\mu[\delta_t \mathbf{e}_t]$, we use importance ratio to estimate the former expectation term:

$$\mathbb{E}_\pi[\delta_t \mathbf{e}_t] = \mathbb{E}_\mu[\rho_t \delta_t \mathbf{e}_t], \rho_t \stackrel{\text{def}}{=} \rho(s_t, a_t) \stackrel{\text{def}}{=} \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}.$$

We typically make a coverage assumption, which states that whenever the target policy has probability support (i.e., nonzero probability) on a state-action pair, the bebaviour policy should also have support on it. Hence, the denominator cannot be zero under such an assumption.

However, it turns out that such an algorithm does not guarantee convergence, mainly because the corresponding $\mathbf{A}$ matrix incurred by such an algorithm is no longer positive definite (Tsitsiklis & Van Roy, 1997; Sutton et al., 2016). And so TD(λ) can diverge when $\pi \neq \mu$ (off-policy). We refer readers to Sutton et al. (2016, Page 6, 8, 9) for a detailed discussion about the divergence (also called "instability") issue of off-policy linear TD.

To achieve stability (i.e., convergence), the emphatic TD (ETD) algorithm (Sutton et al., 2016) introduces *emphatic* weighting M_t. This weighting includes long-term information about π by further introducing a memory scalar called followon trace F_t (see [Pg. 16] (Sutton et al., 2016)). And the update rule of the two quantities works as follows.

$$M_t = \lambda + (1 - \lambda), F_t \qquad \triangleright \; F_t = \gamma \rho_{t-1} F_{t-1} + 1.$$

Now the eligibility trace vector $\mathbf{e}_{m,t}$ is updated by:

$$\mathbf{e}_{m,t} \stackrel{\text{def}}{=} \rho_t(\gamma \lambda \mathbf{e}_{m,t-1} + M_t \mathbf{x}_t).$$

An intuitive interpretation of this updating rule is that a state with larger emphatic weighting would affect more on weight vector update upon observing future states. Those weight units corresponding to a state's feature vector with large emphatic weighting would remain affected for a long time even after the time step observing that state feature. Note that we slightly abuse the notation by putting the importance ratio ρ_t inside the trace vector updating rule for conciseness. The ETD algorithm's updating rules are as follows.

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta_t\mathbf{e}_{m,t}, \tag{3.3}$$

$$F_t = \gamma\rho_{t-1}F_{t-1} + 1, \tag{3.4}$$

$$M_t = \lambda + (1-\lambda)F_t, \tag{3.5}$$

$$\mathbf{e}_{m,t} = \rho_t(\gamma\lambda\mathbf{e}_{m,t-1} + M_t\mathbf{x}_t). \tag{3.6}$$

By Yu (2015), under some mild technical conditions, this ETD algorithm converges to the minimizer of

$$\mathbb{E}_\mu[\delta_t\mathbf{e}_{m,t}]^\top\mathbb{E}_\mu[\mathbf{x}_t\mathbf{x}_t^\top]^{-1}\mathbb{E}_\mu[\delta_t\mathbf{e}_{m,t}].$$

Again, for the convenience of algorithm derivation, this objective can be further written in matrix format, leading to the following off-policy policy evaluation objective:

$$\text{MSPBE}(\mathbf{w}) = (\mathbf{b}_m - \mathbf{A}_m\mathbf{w})^\top\mathbf{C}^{-1}(\mathbf{b}_m - \mathbf{A}_m\mathbf{w}), \tag{3.7}$$

where the subscript m is used to differentiate those matrix notations from the ones in on-policy policy evaluation objective, and

$$\begin{aligned}
\mathbf{A}_m &\overset{\text{def}}{=} \mathbb{E}_\mu[\mathbf{e}_{m,t}(\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1})^\top], \\
\mathbf{b}_m &\overset{\text{def}}{=} \mathbb{E}_\mu[R_{t+1}\mathbf{e}_{m,t}], \\
\mathbf{C} &\overset{\text{def}}{=} \mathbb{E}_\mu[\mathbf{x}_t\mathbf{x}_t^\top],
\end{aligned}$$

with $\mathbf{b}_m - \mathbf{A}_m\mathbf{w} = \mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}]$. One can see that the key differences between this off-policy policy evaluation objective and the previous objective (3.1) exists in the trace vector and the expectation terms defined w.r.t. the behavior policy μ.

It should be noted that, in ETD updating rules 3.3, if we simply set $M_t = 1$ irrespective of the followon trace term F_t, then the algorithm reduces to off-policy TD(λ) and the convergence can no longer be guaranteed. As a result, the weighting scheme ((3.4),(3.5),(3.6)) defined by the followon trace term is the key to guarantee the stability of off-policy ETD algorithm (see Yu (2015, page 6,7) for details). If we further assume $\mu = \pi$, $\rho_t = 1$ for all t, the ETD algorithm reduces to the previously introduced on-policy temporal difference learning algorithm TD(λ).

Two common strategies to obtain the minimum $\mathbf{w}$ of this objective are stochastic temporal difference techniques, such as TD(λ) (Sutton, 1988), or directly approximating the linear system and solving for the weights, such as in LSTD(λ) (Boyan, 1999). The first class constitutes linear-complexity methods, both in computation and storage, including the family of gradient TD methods (Maei, 2011), True online TD methods (van Seijen & Sutton, 2014; van Hasselt et al., 2014) and several others (see Dann et al. (2014); White & White (2016) for a more complete summary).

On the other extreme, with quadratic computation and storage, one can approximate $\mathbf{A}_m$ and $\mathbf{b}_m$ incrementally and solve the system $\mathbf{A}_m\mathbf{w} = \mathbf{b}_m$. Given samples up to time step t: $\{(S_i, A_i, S_{i+1}, R_{i+1})\}_{i=1}^t$, one can estimate

$$\mathbf{A}_{m,t} \stackrel{\text{def}}{=} \frac{1}{t}\sum_{i=1}^{t} \mathbf{e}_{m,i}(\mathbf{x}_i - \gamma\mathbf{x}_{i+1})^\top,$$
$$\mathbf{b}_{m,t} \stackrel{\text{def}}{=} \frac{1}{t}\sum_{i=1}^{t} \mathbf{e}_{m,i}R_{i+1},$$

and then compute solution $\mathbf{w}$ such that $\mathbf{A}_{m,t}\mathbf{w} = \mathbf{b}_{m,t}$. Such least-squares TD (LSTD) methods are typically implemented incrementally using the Sherman-Morrison formula, requiring $\mathcal{O}(d^2)$ storage and computation per step. However, it is usually much more sample efficient than linear TD methods. Further, it is sound (convergent) even in the off-policy setting.

Our goal is to develop algorithms that interpolate between these two extremes. We want to have some algorithm with sample efficiency close to LSTD but has time and space complexity close to linear TD.

3.2 Second-order Optimization for MSPBE

Recall that our goal is to develop an algorithm interpolating between linear TD and LSTD. A natural idea to achieve this is to bring in second-order optimization method because 1) it has fast convergence rate—empirically it may converge within a few iterations and is close to solving a linear system directly; 2) many well-studied preconditioning matrix approximation techniques can be borrowed from math or conventional machine learning literature to improve computational and storage complexity. To our best knowledge, our work is the first second-order method derived by minimizing Mean Squared Projected Bellman Error (MSPBE).

Algorithm derivation. To derive the new algorithm, we first take the gradient of the MSPBE (in 3.7) to get

$$-\frac{1}{2}\nabla_{\mathbf{w}}\text{MSPBE}(\mathbf{w}) = \mathbf{A}_m^\top \mathbf{C}^{-1}\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}]. \tag{3.8}$$

Consider a second order update by computing the Hessian: $\mathbf{H} = \mathbf{A}_m^\top \mathbf{C}^{-1}\mathbf{A}_m^\top$.

For simplicity of notation, starting from now on, let $\mathbf{A} = \mathbf{A}_m$ and $\mathbf{b} = \mathbf{b}_m$. For invertible $\mathbf{A}$, the second-order update is

$$\begin{aligned}
\mathbf{w}_{t+1} &= \mathbf{w}_t - \tfrac{\alpha_t}{2}\mathbf{H}^{-1}\nabla_{\mathbf{w}}\text{MSPBE}(\mathbf{w}) \\
&= \mathbf{w}_t + \alpha_t(\mathbf{A}^\top\mathbf{C}^{-1}\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{C}^{-1}\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}] \\
&= \mathbf{w}_t + \alpha_t\mathbf{A}^{-1}\mathbf{C}\mathbf{A}^{-\top}\mathbf{A}^\top\mathbf{C}^{-1}\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}] \\
&= \mathbf{w}_t + \alpha_t\mathbf{A}^{-1}\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}].
\end{aligned}$$

In fact, for our quadratic loss, the optimal descent direction is

$$\mathbf{A}^{-1}\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}]$$

with $\alpha_t = 1$, in the sense that

$$\operatorname*{argmin}_{\Delta\mathbf{w}} \text{loss}(\mathbf{w}_t + \Delta\mathbf{w}) = \mathbf{A}^{-1}\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}].$$

It should be noted that, though we follow the conventional way of deriving the second-order updating rule (i.e., the Hessian inverse times the gradient

vector), the resulting updating rule neither has the Hessian inverse as the preconditioning matrix, nor has the gradient. Because in the above updating rule, $\mathbf{A}$ is not Hessian of MSPBE and $\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}]$ is not the gradient of MSPBE. For rigorousness, we would consistently call the matrix multiplied by $\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}]$ *preconditioning matrix*, rather than the Hessian inverse.

Computing and maintaining the matrix inversion and updating $\mathbf{w}$ requires quadratic computation and storage. Hence such methods do not provide additional advantages over LSTD method without using any efficient matrix approximations. We propose to bring in the Quasi-Newton approaches that are typically used in approximating the Hessian in the optimization literature. There have been recent insights that using approximate Hessians for stochastic gradient descent can in fact speed convergence (Schraudolph et al., 2007; Bordes et al., 2009; Mokhtari & Ribeiro, 2014).

Nonetheless, before we dive into the details of our ATD algorithm, as a sanity check, we first validate the idea of bringing in a second-order optimization method by empirically examining this second-order temporal difference learning updating rule. This further motivates our ATD algorithms formally introduced in the next Section 3.3.

Empirical Result of Second-order Temporal Difference Learning. We empirically examine the most basic form of the second-order TD learning rule without using bootstrapping parameter, emphasis term, and any low-rank matrix approximation technique.

In order to estimate the term $\mathbf{A}^{-1}\mathbb{E}_\pi[\delta_t(\mathbf{w})\mathbf{e}_{m,t}]$, we incrementally maintaining the full $\mathbf{A}$ by Sherman-Morrison formulae, which says upon adding the outer product of two column vectors $\mathbf{v}, \mathbf{u}$ to the $\mathbf{A}$ matrix, the inverse can be updated by

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^\top)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^\top\mathbf{A}^{-1}}{1 + \mathbf{v}^\top\mathbf{A}^{-1}\mathbf{u}}.$$

Typically the matrix $\mathbf{A}$ is initialized by some diagonal matrix (i.e., usually an identity matrix multiplied by some constant) to ensure invertibility at the beginning. Then we stochastic sampling the expectation term $\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}]$. That is, at time step t, given the training sample $\mathbf{x}_t, \mathbf{x}_{t+1}, r_{t+1}$, we run the

following updates:

$$\begin{aligned}
\mathbf{d}_t &= \mathbf{x}_t - \gamma \mathbf{x}_{t+1}, \\
\mathbf{A}_t &= (\frac{t-1}{t}\mathbf{A}_{t-1} + \frac{1}{t}\mathbf{x}_t \mathbf{d}_t^\top)^{-1}, \\
\delta_t &= r_{t+1} + \gamma \mathbf{x}_{t+1}^\top \mathbf{w}_t - \mathbf{x}_t^\top \mathbf{w}_t, \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha_t \mathbf{A}_t^{-1} \delta_t \mathbf{x}_t.
\end{aligned}$$

Our purpose of running this algorithm is to verify that 1) the second-order method does provide a significant improvement upon the linear TD method in terms of sample efficiency and 2) achieves performance very close to the LSTD method. We expect the latter because the second-order method mostly finds the solution within a few iterations in the supervised learning setting. We expect similar performance in the reinforcement learning setting where LSTD can be considered the closed-form solution at the current time step.

Figure 3.1 shows the learning curves in the form of Percentage Absolute Mean Error (PAME) as a function of time steps on the classic Boyan chain (Boyan & Moore, 1995b; Boyan, 1999) and Mountain Car domain (Sutton & Barto, 2018). Given n testing examples, the PAME is defined as

$$\frac{1}{n}\sum_{i=1}^{n} |\frac{y_i - \hat{y}_i}{y_i}|,$$

where the $y_i, \hat{y}_i$ denote the true value and predicted value at the ith testing example respectively.

On the Boyan chain domain, where the state space is finite and small and the environment dynamics is completely known (i.e., transition probability, reward, etc.), we show the result of ATD-TrueA—ATD with true preconditioning matrix (i.e. compute the true $\mathbf{A}$ as defined without estimation and invert it). It is a "cheating" version in that it assumes complete environment knowledge and helps us understand how the algorithm performs if the second order preconditioning knowledge is known. On the mountain car domain, we show ATD-FullA, where the $\mathbf{A}$ is computed by Sherman-Morrison update without using low-rank approximation. This is a realistic ATD version when computational and storage power are sufficient. Note that we use decaying

learning rate as $1/t$ for both algorithms to match the design of the two ATD variants introduced in the next two chapters. In fact, ATD-FullA can be much closer to LSTD if we use constant learning rate and optimize this parameter. We show such results in Figure 7.1, Chapter 7 when we discuss the limitations of ATD algorithms.

It is obvious that 1) ATD with second-order optimization learns much faster than linear TD algorithm, and 2) ATD with a full/true preconditioning matrix can find a similar solution to LSTD within the time horizon (i.e., the number samples processed) we tested on both domains. The two observations are aligned with our expectations and show that bringing a second-order method could be a promising direction to improve the sample efficiency of TD algorithms.

To this end, we validated the basic idea of applying second-order optimization method to minimize MSPBE. However, one can see that the implemented two ATD variants ATD-TrueA and ATD-FullA are not always feasible: ATD-TrueA can be done only when we have access to the full environment knowledge and the state space is small, while ATD-FullA is feasible only when the feature dimension (and hence the $\mathbf{A}$) dimension is low. In the next section, we present the general form of our ATD algorithm, which achieves better practical utility.

3.3 General Form of ATD Algorithms

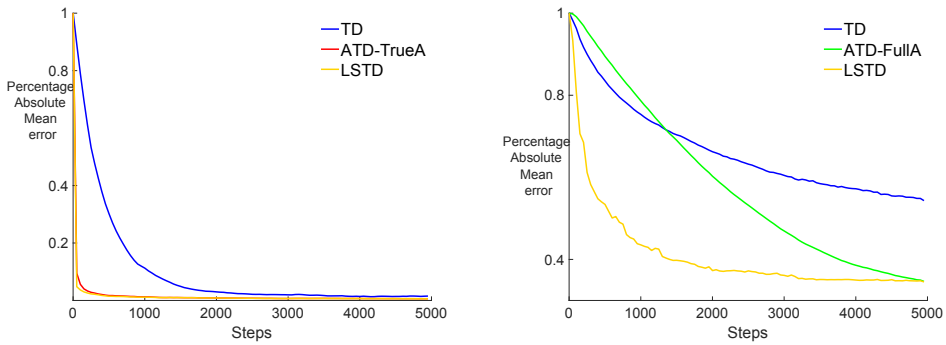
We now introduce our approach called Accelerated gradient TD (ATD), which approximates second-order gradient descent of the MSPBE as introduced in the previous section.

Recall from Section 3.2 that we already showed the below expected second-order updating rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \tfrac{\alpha_t}{2}\mathbf{H}^{-1}\nabla_{\mathbf{w}}\text{MSPBE}(\mathbf{w}) \tag{3.9}$$

$$= \mathbf{w}_t + \alpha_t \mathbf{A}^{-1}\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}]. \tag{3.10}$$

By observing the term $\mathbf{A}^{-1}\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}]$, we identify **two challenges**.



(a) Boyan chain (b) Mountain Car

Figure 3.1: Learning curves plotted as Percentage Absolute Mean Error (PAME) v.s. training steps on (a) Boyan chain and (b) Mountain Car, respectively. On the Boyan chain, a true $\mathbf{A}$ can be calculated, and we found ATD-FullA performs extremely similarly to LSTD and ATD-TrueA. Hence we only show ATD-TrueA in (a). All results are averaged over 50 runs, and the standard errors are small and are ignored.

1. First, we need an incremental and efficient way to approximate the preconditioning matrix $\mathbf{A}^{-1}$ that provides useful curvature information and that is also sub-quadratic in storage and computation.

2. Second, we need to ensure that after using the approximation, we do not find a biased solution.

To address the first challenge, we propose to approximate only $\mathbf{A}^{-1}$ and sample $\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}] = \mathbf{b} - \mathbf{A}\mathbf{w}$ using $\delta_t(\mathbf{w}_t)\mathbf{e}_t$ as an unbiased sample. The approximation technique should have theoretical guarantee for approximation quality and allows to compute the inverse of the approximated $\mathbf{A}$ in sub-quadratic computation and storage complexity under rank one update. As for the second challenge, we propose to use a regularization term $\eta\delta_t\mathbf{e}_t$. Intuitively, this regularization turns the potentially low-rank preconditioning matrix into full rank again. In Section 4.4, we theoretically show that the regularization helps ensure unbiased convergence.

General form of ATD. Finally, the general form of our proposed accelerated temporal difference learning update—which we call ATD(λ)—is

$$\mathbf{w}_{t+1} = \mathbf{w}_t + (\alpha_t \hat{\mathbf{A}}_t^\dagger + \eta \mathbf{I})\delta_t \mathbf{e}_t$$

with expected update

$$\mathbf{w}_{t+1} = \mathbf{w}_t + (\alpha_t \hat{\mathbf{A}}^\dagger + \eta \mathbf{I})\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}] \tag{3.11}$$

with regularization $\eta > 0$. We summarize the complete pseudo-code of the general form in Algorithm 1.

Algorithm 1 General form of Accelerated Temporal Difference Learning

▷ $\mathbf{e}_0 = \mathbf{0}$, initialized $\mathbf{w}_0 = \mathbf{0}$

$\mathbf{x}_0$: the feature vector of the initial state
η : the regularization weight/a small final stepsize value, e.g., $\eta = 0.00001$
α_t learning rate, e.g. $\alpha_t = 1/t$
for $t = 1, 2, ...$ **do**
In $\mathbf{x}_t$ select action $\sim \mu$, observe $\mathbf{x}_{t+1}$, reward r_{t+1}, discount γ_{t+1} (could be zero if terminal state)
$\delta_t = r_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t$
$\mathbf{e}_t =$ TRACE_UPDATE$(\mathbf{e}_{t-1}, \mathbf{x}_t, \gamma_t, \lambda_t)$ ▷ e.g., accumulative trace, replacing trace, emphatic trace, etc.
$\hat{\mathbf{A}}_t^\dagger =$ MATRIX_UPDATE$(\hat{\mathbf{A}}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t+1}, \gamma_t, \mathbf{e}_t, ...)$ ▷ The matrix may be maintained by using multiple matrices, such as SVD, etc. The input of this update function depends on the concrete approximation method. This part will be carefully studied in the next two chapters.
$\mathbf{w}_{t+1} = \mathbf{w}_t + (\alpha_t \hat{\mathbf{A}}_t^\dagger + \eta \mathbf{I})\delta_t \mathbf{e}_t$

If $\hat{\mathbf{A}}$ is a poor approximation of $\mathbf{A}$ or discards key information—as we will do with a low-rank approximation—then updating using only $\mathbf{b} - \hat{\mathbf{A}}\mathbf{w}$ will result in a biased solution, as is the case for tLSTD (see Section 4.1) as shown in Theorem 1. Instead, sampling $\mathbf{b} - \mathbf{A}\mathbf{w} = \mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}]$, as we show in Theorem 2, yields an unbiased solution, even with a poor approximation $\hat{\mathbf{A}}$.

Given the general form of ATD(λ), the next question is how to approximate $\mathbf{A}$. In the next two chapters, we theoretically and empirically study two ways to perform efficient preconditioning matrix approximations: incrementally truncated singular value decomposition (Chapter 4) and left-sided

matrix sketching via random projection (Chapter 5). We demonstrate the performance of ATD with the two variants versus many linear and subquadratic methods, indicating that ATD (1) can match the data efficiency of LSTD, with significantly less computation and storage; (2) is unbiased, unlike many of the alternative subquadratic methods; (3) significantly reduces parameter sensitivity for the stepsize, versus linear TD methods and (4) is significantly less sensitive to the choice of rank parameter than directly solving the linear system (i.e., tLSTD as detailed in the next chapter), enabling a smaller rank to be chosen and so providing a more efficient incremental algorithm. Overall, our results suggest that ATD may be the first practical subquadratic complexity TD method suitable for fully incremental policy evaluation.

In general, many other approximations to $\mathbf{A}$ could be used. We leave studying their theoretical and empirical properties as an important future direction for ATD.

Chapter 4

Approximation by Incremental Truncated SVD

When it comes to matrix approximation, a natural idea is to use a low-rank approximation. For a low-rank approximation $\hat{\mathbf{A}}$ with d-by-d dimensions, of rank k, represented with truncated singular value decomposition (SVD) $\hat{\mathbf{A}} = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top$, the storage requirement is $\mathcal{O}(dk)$ and the required matrix-vector multiplications are only $\mathcal{O}(dk)$ because for any vector $\mathbf{v}$, $\hat{\mathbf{A}}\mathbf{v} = \mathbf{U}_k \mathbf{\Sigma}_k (\mathbf{V}_k^\top \mathbf{v})$, is a sequence of $\mathcal{O}(dk)$ matrix-vector multiplications.

This chapter is mainly from Gehring et al. (2016); Pan et al. (2017b). It discusses an instance of the ATD algorithm by using incremental truncated SVD to approximate the preconditioning matrix in the updating rule. We choose this approximation because it is proved effective for incremental estimation in reinforcement learning, as reviewed in this chapter. The total computational complexity of the algorithm is $\mathcal{O}(dk + k^3)$ for the fully incremental update to $\hat{\mathbf{A}}$ and $\mathcal{O}(dk)$ for mini-batch updates of k samples. Notice that when $k = 0$, the algorithm reduces exactly to TD(λ). On the other extreme, where $k = d$, ATD is equivalent to an iterative form of LSTD(λ). Hence, choosing some k value between the extremes enables ATD to interpolate between linear TD and LSTD algorithms, which satisfies our original goal of developing ATD algorithms.

This chapter is organized as follows. We first present the algorithm of using incremental truncated SVD to approximate the matrix in the classic LSTD algorithm and study its limitations and relevant theoretical properties in Sec-

tion 4.1. Then we present algorithmic details and empirical results of comparing our ATD using such SVD approximation with existing well-known policy evaluation algorithms in Section 4.2. In Section 4.3, we conduct a further empirical study to compare our algorithm against several popular first-order stochastic optimization algorithms developed in recent years. We conclude this chapter by theoretically showing the unbiased convergence of ATD's expected updating rule in Section 4.4.

4.1 Incremental Truncated LSTD Learning

As we introduced, the TD updating rule can be written as $\mathbf{w} \leftarrow \mathbf{w} + \alpha(r_{t+1} + \gamma \mathbf{x}_{t+1}^\top \mathbf{w} - \mathbf{x}_t^\top \mathbf{w})\mathbf{z}_t$. The original LSTD algorithm (Bradtke & Barto, 1996) incrementally maintains $\mathbf{A}_t^{-1}$ using the Sherman-Morrison update so that on each step the new solution $\mathbf{w} = \mathbf{A}_t^{-1}\mathbf{b}_t$ can be compute.

We iteratively update and solve this system by maintaining a low-rank approximation to $\mathbf{A}_t$ directly. Any matrix $\mathbf{A} \in \mathbb{R}^{d\times d}$ has a singular value decomposition (SVD) $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, where $\mathbf{\Sigma} \in \mathbb{R}^{d\times d}$ is a diagonal matrix of the singular values of $\mathbf{A}$ and $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{d\times d}$ are orthonormal matrices: $\mathbf{U}^\top\mathbf{U} = \mathbf{I} = \mathbf{V}^\top\mathbf{V}$ and $\mathbf{U}\mathbf{U}^\top = \mathbf{I} = \mathbf{V}\mathbf{V}^\top$. With this decomposition, for full rank $\mathbf{A}$, the inverse of $\mathbf{A}$ is simply computed by inverting the singular values, to get $\mathbf{w} = \mathbf{A}^{-1}\mathbf{b} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^\top\mathbf{b}$. In many cases, however, the rank of $\mathbf{A}$ may be smaller than d, giving $d - \text{rank}(\mathbf{A})$ singular values that are zero. Further, we can approximate $\mathbf{A}$ by dropping (i.e., zeroing) some number of the smallest singular values, to obtain a rank k approximation. Correspondingly, rows of $\mathbf{U}$ and $\mathbf{V}$ are zeroed, reducing the size of these matrices to $d \times k$. The further we reduce the dimension, the more practical we can update the matrix efficiently; however there is clearly a trade-off in terms of accuracy of the solution. We first investigate the theoretical properties of using a low-rank approximation to $\mathbf{A}_t$ and then present the incremental t-LSTD algorithm.

4.1.1 Characterizing the Low-rank Approximation

Low-rank approximations provide an efficient approach to obtaining stable solutions for linear systems. The approach is particularly well motivated for our resource constrained setting, because of the classical Eckart-Young-Mirsky theorem (Eckart & Young, 1936; Mirsky, 1960), which states that the optimal rank k approximation to a matrix under any unitarily invariant norm (e.g., Frobenius norm, spectral norm, nuclear norm) is the truncated singular value decomposition. In addition to this nice property, which facilitates development of an efficient approximate LSTD algorithm, the truncated SVD can be viewed as a form of regularization (Hansen, 1986), improving the stability of the solution.

To see why truncated SVD regularizes the solution, consider the solution to the linear system

$$\mathbf{w} = \mathbf{A}^\dagger \mathbf{b} = \mathbf{V}\mathbf{\Sigma}^\dagger \mathbf{U}^\top \mathbf{b} = \sum_{i=1}^{\text{rank}(\mathbf{A})} \frac{\mathbf{v}_i \mathbf{u}_i^\top}{\sigma_i} \mathbf{b}$$

for ordered singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{\text{rank}(\mathbf{A})} > \sigma_{\text{rank}(\mathbf{A})+1} = 0, \ldots, \sigma_d = 0$. $\mathbf{A}^\dagger$ is the pseudo-inverse of $\mathbf{A}$, with $\mathbf{\Sigma}^\dagger = \text{diag}(\sigma_1^{-1}, ..., \sigma_{\text{rank}(\mathbf{A})}^{-1}, 0, ..., 0)$ composed of the inverses of the non-zero singular values. For very small, but still non-zero σ_i, the outer product $\mathbf{v}_i \mathbf{u}_i^\top$ will be scaled by a large number; this will often correspond to highly overfitting the observed samples and a high variance estimate. A common practice is to regularize $\mathbf{w}$ with $\eta \|\mathbf{w}\|_2$ for regularization weight $\eta > 0$, modifying the multiplier from σ_i^{-1} to $\sigma_i / (\sigma_i^2 + \eta)$ because

$$\mathbf{w} = (\mathbf{A}^\top \mathbf{A} + \eta \mathbf{I})^{-1} \mathbf{A}^\top \mathbf{b} = \mathbf{V}(\mathbf{\Sigma}^2 + \eta \mathbf{I})^{-1} \mathbf{\Sigma} \mathbf{U}^\top \mathbf{b}.$$

The regularization reduces variance but introduces bias controlled by η; for $\eta = 0$, we obtain the unbiased solution. Similarly, by thresholding the smallest singular values to retain only the top k singular values, we get

$$\mathbf{w} = \mathbf{A}_k^\dagger \mathbf{b} = \mathbf{V} \text{diag}(\sigma_1^{-1}, ..., \sigma_k^{-1}, 0, ..., 0) \mathbf{U}^\top \mathbf{b} = \sum_{i=1}^{k} \frac{\mathbf{v}_i \mathbf{u}_i^\top}{\sigma_i} \mathbf{b}, \tag{4.1}$$

and as a result, we introduce bias, but reduce variance; because the size of σ_k^{-1} can be controlled by the choice of $k < \text{rank}(\mathbf{A})$.

To characterize the bias-variance tradeoff, we bound the difference between the true solution, $\mathbf{w}^*$, and the approximate rank k solution at time t, $\mathbf{w}_{t,k}$. We use a similar analysis to the one used for regularized LSTD (Proposition 6.3.4, (Bertsekas, 2007)). This previous bound does not easily extend, because in regularized LSTD, the singular values are scaled up, maintaining the information in the singular vectors (i.e., no columns are dropped from $\mathbf{U}$ or $\mathbf{V}$). We bound the loss incurred by dropping singular vectors.

The following is a simple but realistic assumption for ill-posed systems to achieve such a bound (Hansen, 1990). The assumption states that $\mathbf{u}_i^\top \mathbf{b}$ shrinks faster than σ_i^p, where p specifies the smoothness of the solution $\mathbf{w}$ and is related to the smoothness parameter for the Hilbert space setting (Cor. 1.2.7, (Groetsch, 1984)).

Assumption 1: The linear system defined by $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ and $\mathbf{b}$ satisfy the *discrete Picard condition*: for some $p > 1$,

$$|\mathbf{u}_i^\top \mathbf{b}| \leq \sigma_i^p \qquad \text{for } i = 1, \ldots, \text{rank}(\mathbf{A})$$
$$|\mathbf{u}_i^\top \mathbf{b}| \leq \sigma_{\text{rank}(A)}^p \qquad \text{for } i = \text{rank}(\mathbf{A}) + 1, \ldots, d.$$

We write the SVD of $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ and $\mathbf{A}_t = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^\top$, where to avoid cluttered notation, we do not explicitly subscript with t. Further, though the singular values are unique, there is a space of equivalent singular vectors, up to sign changes and multiplication by rotation matrices. We assume that among the space of equivalent SVDs of $\mathbf{A}_t$, the most similar singular vectors for each singular value are chosen between $\mathbf{A}$ and $\mathbf{A}_t$. This avoids uniqueness issues without losing generality, because we only conceptually compare the SVDs of $\mathbf{A}$ and $\mathbf{A}_t$; the proof does not rely on practically obtaining this matching SVD.

Theorem 1 (Bias-variance trade-off of rank-k approximation (Gehring et al. (2016, Theorem 1))). *Let $\mathbf{A}_{t,k} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}_k\hat{\mathbf{V}}^\top$ be the approximated $\mathbf{A}$ after t samples, truncated to rank k, i.e., with the last $k+1, \ldots, d$ singular values zeroed. Let $\mathbf{w}^* = \mathbf{A}^\dagger\mathbf{b}$ and $\mathbf{w}_{t,k} = \mathbf{A}_{t,k}^\dagger\mathbf{b}_t$. Under Assumption 1 and 2, the relative error of the rank-k weights to the true weights $\mathbf{w}^*$ is bounded as follows:*

$$\|\mathbf{w}_{t,k} - \mathbf{w}^*\|_2 \leq \frac{1}{\hat{\sigma}_k}\|\mathbf{b}_t - \mathbf{A}_t\mathbf{w}^*\|_2 + (d-k)\epsilon(t) + \underbrace{(d-k)\sigma_k^{p-1}}_{bias}$$

for function $\epsilon : \mathbb{N} \to [0, \infty)$*, where* $\epsilon(t) \to 0$ *as* $t \to \infty$*:*

$$\epsilon(t) = \min\Big(rank(\mathbf{A})\sigma_1^{p-1}, \sum_{j=1}^{rank(\mathbf{A})} \left\|\mathbf{v}_j\sigma_j^{p-1} - \hat{\mathbf{v}}_j\hat{\sigma}_j^{p-1}\right\|_2 + \hat{\sigma}_k^{p-1} - \sigma_k^{p-1}\Big).$$

The key step is to split up the error into two terms: approximation error due to a finite number of samples t and bias due the choice of $k < d$. Then the second part is bounded using the discrete Picard condition to ensure that the magnitude of $\mathbf{u}_j^\top \mathbf{b}$ does not dominate the error, and by adding and subtracting terms to express the error in terms of differences between $\mathbf{A}$ and $\mathbf{A}_t$. Because $\mathbf{A}_t$ converges to $\mathbf{A}$ (Tsitsiklis & Van Roy, 1997), we can see that $\epsilon(t)$ converges to zero because the differences $\mathbf{v}_j\sigma_j^{p-1} - \hat{\mathbf{v}}_j\hat{\sigma}_j^{p-1}$ and $\hat{\sigma}_k^{p-1} - \sigma_k^{p-1}$ converge to zero.

Remark 1. *Notice that for no truncation, the bias term disappears and the first term could be very large because* $\hat{\sigma}_k = \hat{\sigma}_d$ *could be very small. In fact, previous work on finite sample analysis of LSTD uses an unbiased estimate and the bound suffers from an inverse relationship to the smallest eigenvalue of* $\mathbf{X}^\top\mathbf{X}$ *(Lazaric et al. (2010, Lemma 3), Ghavamzadeh et al. (2010); Tagorti & Scherrer (2015)). Here, we avoid such a potentially large constant in the bound at the expense of an additional bias term determined by the choice of* k*. Lasso-TD (Ghavamzadeh & Lazaric, 2011) similarly avoids such a dependence, using* ℓ_1 *regularization; to the best of our knowledge, however, there does not yet exist an efficient incremental Lasso-TD algorithm. A future goal is to use the above bound, to obtain a finite sample bound for t-LSTD(*λ*), using the most up-to-date analysis by Tagorti & Scherrer (2015) and more general techniques for linear system introduced by Avila Pires & Szepesvari (2012).*

4.1.2 Incremental Low-rank LSTD Algorithm

We have theoretically characterized the bias-variance trade-off by using a low-rank approximation to $\mathbf{A}_t$ for computing the solution to LSTD from t samples. However, the computational complexity of explicitly computing $\mathbf{A}_t$ from samples and then performing a SVD is $\mathcal{O}(d^3)$, which is not feasible for most settings. In this section, we propose an algorithm that incrementally computes a low-rank singular value decomposition of $\mathbf{A}_t$, from samples, with significantly improved storage $\mathcal{O}(dk)$ and computational complexity $\mathcal{O}(dk + k^3)$, which we can further be reduced to $\mathcal{O}(dk)$ using mini-batches of size k.

To maintain a low-rank approximation to $\mathbf{A}_t$ incrementally, we need to update the SVD with new samples. With each new $\mathbf{x}_t$, we add the rank-one matrix $\mathbf{e}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^\top$ to $\mathbf{A}_t$. Consequently, we can take advantage of recent advances for fast low-rank SVD updates (Brand, 2006), with some specialized computational improvements for our setting. Algorithm 4 summarizes the generic incremental update for t-LSTD, which can use mini-batches or update on each step, depending on the choice of the mini-batch size. We show the detailed incremental SVD per-time-step updating algorithm (i.e. mini-batch size is one) update in Algorithm 2 and the concrete way of computing the weight vector in Algorithm 3. The basics of the SVD update follow from previous work (Brand, 2006) but our implementation offers some optimizations specific for LSTD.

By maintaining the SVD incrementally, we do not need to explicitly maintain $\mathbf{A}_t$; therefore, storage is reduced to the size of the truncated singular vector matrices, which is $\mathcal{O}(dk)$. To maintain $\mathcal{O}(dk)$ computational complexity, matrix and vector multiplications need to be carefully ordered. For example, to compute $\mathbf{w}$, first $\tilde{\mathbf{b}} = \mathbf{U}^\top\mathbf{b}$ is computed in $\mathcal{O}(dk)$, then $\mathbf{\Sigma}_k\tilde{\mathbf{b}}$ is computed in $\mathcal{O}(k)$, and finally that is multiplied by $\mathbf{V}$ in $\mathcal{O}(dk)$. For $k = 1$ (update on each step), the $\mathcal{O}(k^3)$ computation arises from a re-diagonalization and the multiplication of the resulting orthonormal matrices. For mini-batches of size k, we can get further computational improvements by amortizing costs across k steps, to obtain a total amortized complexity $\mathcal{O}(dk)$, getting rid of the k^3

Algorithm 2 update-svd($\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}, \mathbf{L}, \mathbf{R}, \mathbf{e}, \mathbf{d}, k$) with one sample for incremental t-LSTD

$\mathbf{m} = \mathbf{L}^\top \mathbf{U}^\top \mathbf{e}$ // $\mathcal{O}(dk)$ time, as $\mathbf{v} = \mathbf{U}^\top \mathbf{e}$ is $\mathcal{O}(dk)$ and $\mathbf{L}^\top \mathbf{v}$ is $\mathcal{O}(k^2)$

$\mathbf{p} = \mathbf{e} - \mathbf{U}\mathbf{L}\mathbf{m}$ // $\mathcal{O}(dk)$ time

$\mathbf{n} = \mathbf{R}^\top \mathbf{V}^\top \mathbf{d}$ // $\mathcal{O}(dk)$ time

$\mathbf{q} = \mathbf{d} - \mathbf{V}\mathbf{R}\mathbf{n}$ // $\mathcal{O}(dk)$ time

$$\mathbf{K} \leftarrow \begin{bmatrix} \mathbf{\Sigma} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} + \begin{bmatrix} \mathbf{m} \\ \|\mathbf{p}\| \end{bmatrix} \begin{bmatrix} \mathbf{n} \\ \|\mathbf{q}\| \end{bmatrix}^\top$$

$[\tilde{\mathbf{L}}, \mathbf{\Sigma}, \tilde{\mathbf{R}}] \leftarrow \text{SVD}(\mathbf{K})$

$$\mathbf{L} \leftarrow \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \tilde{\mathbf{L}}$$ // $\mathcal{O}(k^3)$ time

$$\mathbf{R} \leftarrow \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \tilde{\mathbf{R}}$$ // $\mathcal{O}(k^3)$ time

if $\|\mathbf{p}\| \leq \epsilon$ **then** // $\epsilon = 0.00001$

    $\|\mathbf{p}\| \leftarrow \mathbf{0}$

else $\mathbf{p} \leftarrow \mathbf{p}/\|\mathbf{p}\|$ // normalize, update to $\mathbf{U}$

if $\|\mathbf{q}\| \leq \epsilon$ **then** // $\epsilon = 0.00001$

    $\|\mathbf{q}\| \leftarrow \mathbf{0}$

else $\mathbf{q} \leftarrow \mathbf{q}/\|\mathbf{q}\|$ // normalize, update to $\mathbf{V}$

$\mathbf{U} \leftarrow [\mathbf{U} \;\; \mathbf{p}]$ // Only allowed to grow to $2k$ columns

$\mathbf{V} \leftarrow [\mathbf{V} \;\; \mathbf{q}]$ // Only allowed to grow to $2k$ columns

// If reached size $2k$, reduce back to k by dropping smallest k singular values; $\mathcal{O}(dk)$ amortized complexity

if size($\mathbf{L}$) $\geq 2k$ **then**

    $\mathbf{\Sigma} \leftarrow \mathbf{\Sigma}(1:k, 1:k)$

    $\mathbf{U} \leftarrow \mathbf{U}\mathbf{L}$ // $\mathcal{O}(dk^2)$ time

    $\mathbf{U} \leftarrow \mathbf{U}(:, 1:k)$ // Concatenate back to k columns

    $\mathbf{V} \leftarrow \mathbf{V}\mathbf{R}$ // $\mathcal{O}(dk^2)$ time

    $\mathbf{V} \leftarrow \mathbf{V}(:, 1:k)$ // Concatenate back to k columns

    $\mathbf{L} = \mathbf{I}$, $\mathbf{R} = \mathbf{I}$ // Reintialize

return $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}, \mathbf{L}, \mathbf{R}$

Algorithm 3 compute-weights($\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}, \mathbf{L}, \mathbf{R}, \mathbf{b}$), $\mathcal{O}(dk)$

// Solve $\mathbf{A}^{-1}\mathbf{b}$ where $\mathbf{A} = \mathbf{U}\mathbf{L}\mathbf{\Sigma}\mathbf{R}^\top\mathbf{V}^\top$ and so $\mathbf{A}^{-1} = \mathbf{V}\mathbf{R}\mathbf{\Sigma}^{-1}\mathbf{L}^\top\mathbf{U}^\top$
// Does not invert any singular values that are below $0.01 * \hat{\sigma}_1$
$\tilde{\mathbf{b}} = \mathbf{L}^\top\mathbf{U}^\top\mathbf{b}$ // $\mathcal{O}(dk)$ time, implicit left singular vector is $\mathbf{U}\mathbf{L}$
$\hat{\sigma}_1 \leftarrow \mathbf{\Sigma}(1,1)$
$\mathbf{\Sigma}^\dagger \leftarrow \mathbf{0}$ // initialize as zero matrix
for $i \in \{1, \ldots, k\}$ **do**
    if $\mathbf{\Sigma}(i,i) > 0.01\hat{\sigma}_1$ **then**
        $\mathbf{\Sigma}^\dagger(i,i) \leftarrow \mathbf{\Sigma}(i,i)^{-1}$
    else
        break
$\mathbf{w} = \mathbf{V}\mathbf{R}\mathbf{\Sigma}^{-1}\tilde{\mathbf{b}}$ // $\mathcal{O}(dk)$ time

term.

As an additional benefit, unlike previous incremental LSTD algorithms, we maintain normalized $\mathbf{A}_t$ and $\mathbf{b}_t$, by incorporating the term β. On each step, we use

$$\mathbf{A}_{t+1} = \tfrac{1}{t+1}(t\mathbf{A}_t + \mathbf{e}_t\mathbf{d}_t^\top) = (1-\beta_t)\mathbf{A}_t + \beta_t\mathbf{e}_t\mathbf{d}_t^\top$$

for $\beta_t = \frac{1}{t+1}$. The multiplication of $\mathbf{A}_t$ by $1-\beta_t$ requires only $\mathcal{O}(k)$ computation because $(1-\beta_t)\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\top = \mathbf{U}_k(1-\beta_t)\mathbf{\Sigma}_k\mathbf{V}_k^\top$. Multiplying the full $\mathbf{A}$ matrix by $1-\beta_t$, on the other hand, would require $\mathcal{O}(d^2)$ computation, which is prohibitive. Further, β_t can be selected to obtain a running average, as in Algorithm 4, or more generally can be set to any $\beta_t \in (0,1)$. For example, to improve tracking, β can be chosen as a constant to weight more recent samples more highly in the value function estimate.

4.2 ATD with Incremental Truncated SVD

To get the ATD with incremental truncated SVD algorithm, we simply put the incremental SVD Algorithm 2 as the MATRIX_UPDATE function in Algorithm 1. For convenience, we summarize this algorithm in Algorithm 5. As we explained, the parameter β can be flexible to satisfy the demand of a special type of average when necessary. We opt to fix it as $1/t$ across all experiments, showing the generality of such stepsize choice.

Algorithm 4 t-LSTD(λ) using incremental SVD

// Input rank k, and mini-batch size k
// with differing update-svd for $k = 1$ and $k > 1$
$\mathbf{U} \leftarrow [], \mathbf{V} \leftarrow [], \mathbf{\Sigma} \leftarrow 0, \mathbf{b} \leftarrow \mathbf{0}, \mathbf{z} \leftarrow \mathbf{0}, i \leftarrow 0, t \leftarrow 1$
$\mathbf{x} \leftarrow$ the initial observation
repeat
  Take action according to π, observe $\mathbf{x}'$, reward r
  $\beta \leftarrow 1/(t+k)$
  $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \mathbf{x}$
  $\mathbf{d} \leftarrow \beta(\mathbf{x} - \gamma\mathbf{x}')$
  $\mathbf{Z}_{:,i} \leftarrow \mathbf{z}$
  $\mathbf{D}_{:,i} \leftarrow \mathbf{d}$
  $\mathbf{b} \leftarrow (1-\beta)\mathbf{b} + \beta\mathbf{z}r$
  $i \leftarrow i + 1$
  if $i \geq k$ **then**
    // Returns $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{d\times k}$, diagonal $\mathbf{\Sigma} \in \mathbb{R}^{k\times k}$
    $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V} \leftarrow$
    update-svd$(\mathbf{U}, (1-\beta)\mathbf{\Sigma}, \mathbf{V}, \sqrt{\beta}\mathbf{Z}, \sqrt{\beta}\mathbf{D}, k)$
    $\mathbf{Z} \leftarrow 0^{d\times k}, \mathbf{D} \leftarrow 0^{d\times k}, i \leftarrow 0, t \leftarrow t + k$
  $\mathbf{w} \leftarrow \mathbf{V}\mathbf{\Sigma}^{\dagger}\mathbf{U}^{\top}\mathbf{b}$ // $\mathcal{O}(dk)$ time
until agent done interaction with environment

4.2.1 Empirical Results

All the following experiments investigate the on-policy setting, and thus we make use of the standard version of ATD for simplicity. The purpose of our experiments is to answer the following questions: 1) does our method really interpolate between LSTD and TD across domains? 2) does our method, as a second-order method, have a lower sensitivity to its hyper-parameter than other baselines? 3) can our singular value truncation method effectively get rid of useless features? The results presented in this section were generated over 756 **thousand individual experiments** run on three different domains. Detailed descriptions of each domain, error calculation, and all other parameter settings are discussed in detail in the appendix. We included a wide variety of baselines in our experiments, additional related baselines excluded from our study are also discussed in the Appendix A.1.4.

Our first batch of experiments were conducted on Boyan's chain—a domain known to elicit the strong advantages of LSTD(λ) over TD(λ). In

Algorithm 5 Accelerated Temporal Difference Learning

▷ where $\mathbf{U}_0 = [], \mathbf{V}_0 = [], \mathbf{\Sigma}_0 = [], \mathbf{b}_0 = \mathbf{0}, \mathbf{e}_0 = \mathbf{0}$, initialized $\mathbf{w}_0$ arbitrarily

function ATD($k, \eta, \epsilon, \lambda$)
$\mathbf{x}_0 =$ first observation
$\eta =$ a small final stepsize value, e.g., $\eta = 10e-4$
for $t = 1, 2, ...$ **do**
At state $\mathbf{x}_t$, select action $\sim \pi$, observe $\mathbf{x}_{t+1}$, reward r_{t+1}, discount γ_{t+1} (could be zero if terminal state)
$\beta = 1/t$
$\delta_t = r_{t+1} + \gamma_{t+1}\mathbf{w}^\top\mathbf{x}_{t+1} - \mathbf{w}^\top\mathbf{x}_t$
$\mathbf{e}_t =$ TRACE_UPDATE$(\mathbf{e}_{t-1}, \mathbf{x}_t, \gamma_t, \lambda_t)$ ▷ or call EMPAHTIC_TRACE_UPDATE to use emphatic weighting
$\mathbf{b}_t = (1-\beta)\mathbf{b}_{t-1} + \beta\mathbf{e}_t r_{t+1}$
▷ $\mathbf{U}_t\mathbf{\Sigma}_t\mathbf{V}_t^\top = (1-\beta)\mathbf{U}_{t-1}\mathbf{\Sigma}_{t-1}\mathbf{V}_{t-1}^\top + \beta\mathbf{e}_t(\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1})^\top$
$[\mathbf{U}_t, \mathbf{\Sigma}_t, \mathbf{V}_t] =$ SVD-UPDATE$(\mathbf{U}_{t-1}, (1-\beta)\mathbf{\Sigma}_{t-1}, \mathbf{V}_{t-1}, \beta\mathbf{e}_t, (\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1}), k)$
▷ Ordering of matrix operations important, first multiply $\mathbf{U}_t^\top(\epsilon\mathbf{b}_t + \delta_t\mathbf{e})$ in $\mathcal{O}(dk)$ time
▷ to get a new vector, then by $\mathbf{\Sigma}_t^\dagger$ and $\mathbf{V}_t$ to maintain only matrix-vector multiplications
$\mathbf{w}_{t+1} = \mathbf{w}_t + (\frac{1}{t}\mathbf{V}_t\mathbf{\Sigma}_t^\dagger\mathbf{U}_t^\top + \eta\mathbf{I})(\epsilon\mathbf{b}_t + \delta_t\mathbf{e}_t)$ ▷ where $\mathbf{\Sigma}_t^\dagger = \text{diag}(\hat{\sigma}_1^{-1}, \ldots, \hat{\sigma}_k^{-1}, \mathbf{0})$

Boyan's chain the agent's objective is to estimate the value function based on a low-dimensional, dense representation of the underlying state (perfect representation of the value function is possible). The ambition of this experiment was to investigate the performance of ATD in a domain where the preconditioner matrix is full rank; no rank truncation is applied. We compared five linear-complexity methods (TD(0), TD(λ), true online TD(λ), ETD(λ), true online ETD(λ)), against LSTD(λ) and ATD, reporting the percentage error relative to the true value function over the first 1000 steps, averaged over 200 independent runs. We swept a large range of stepsize parameters, trace decay rates, and regularization parameters, and tested both fixed and decaying stepsize schedules. Figure 4.1 summarizes the results.

Both LSTD(λ) and ATD achieve lower error compared to all the linear baselines—even thought each linear method was tuned using 864 combinations of stepsizes and λ. In terms of sensitivity, the choice of stepsize for TD(0) and

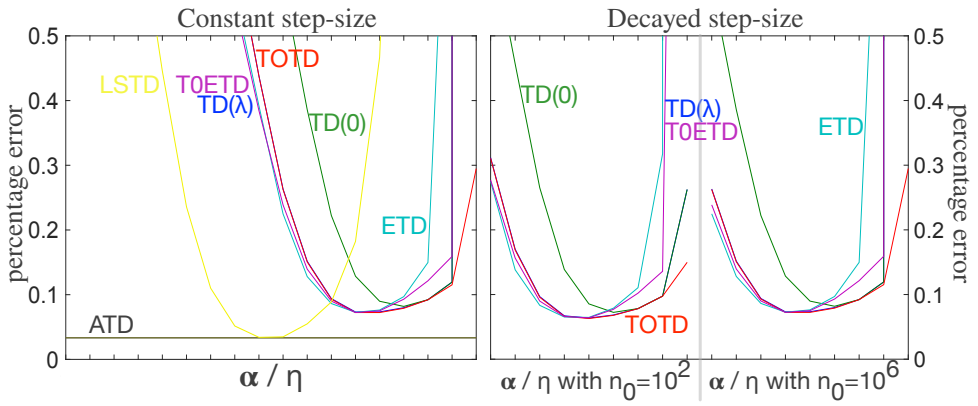


Figure 4.1: Parameter sensitivity in Boyan's chain with constant stepsize (LHS) and decayed stepsizes (RHS). In the plots above, each point summarizes the mean performance (over 1000 time steps) of an algorithm for one setting of α for linear methods, or initialization parameter η for LSTD, and $\alpha/100$ regularizer for ATD, using percentage error compared to the true value function. In the decayed stepsize case, where $\alpha_t = \alpha_0 \frac{n_0+1}{n_0+\text{episode\#}}$, 18 values of α_0 and two values of n_0 were tested—corresponding to the two sides of the RHS graph. The LSTD algorithm (in yellow) has no parameters to decay. Our ATD algorithm (in black) achieves the lowest error in this domain, and exhibits little sensitivity to it's regularization parameter (with stepsize as $\alpha_t = \frac{1}{t}$ across all experiments).

ETD exhibit large effect on performance (indicated by sharp valleys), whereas true-online TD(λ) is the least sensitive to learning rate. LSTD(λ) using the Sherman-Morrison update (used in many prior empirical studies) is sensitive to the regularization parameter; the parameter free nature of LSTD may be slightly overstated in the literature.[1]

Our second batch of experiments investigated characteristics of ATD in a classic benchmark domain with a sparse high-dimensional feature representation where perfect approximation of the value function is not possible—Mountain car with tile coding. The policy to be evaluated stochastically takes the action in the direction of the sign of the velocity, with performance measured by computing a truncated Monte Carlo estimate of the return from states sampled from the stationary distribution (detailed in the appendix).

[1]We are not the first to observe this. Sutton & Barto (2018) note that η plays a role similar to the stepsize for LSTD.

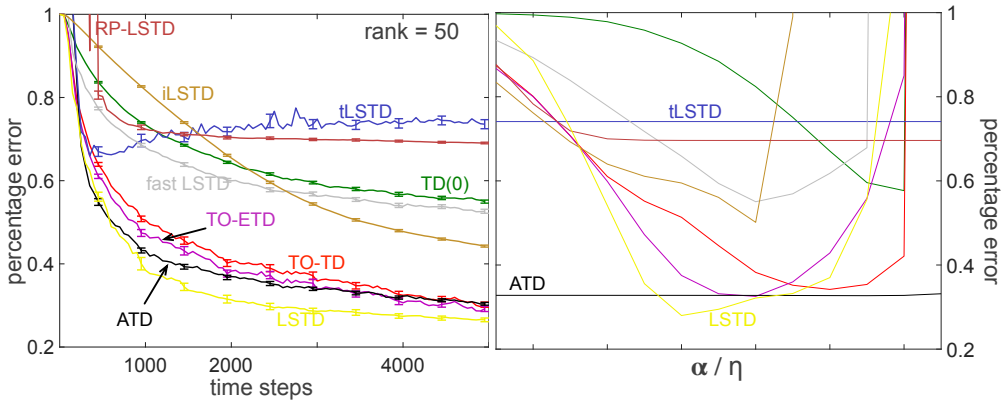


Figure 4.2: The learning curves (LHS) are percentage absolute mean error (i.e., we write it as percentage error in the figure label) versus time steps averaged over 100 runs of ATD with rank 50, LSTD and several baselines described in text. The sensitivity plot (RHS) is with respect to the learning rate of the linear methods, and regularization parameter of the matrix methods. The tLSTD algorithm has no parameter besides rank, while ATD has little sensitivity to it's regularization parameter.

We used a fine grain tile coding of the the 2D state, resulting in a 1024 dimensional feature representation with exactly 10 units active on every time step. We tested TD(0), true online TD(λ), true online ETD(λ), and sub-quadratic methods, including iLSTD (Geramifard et al., 2007), tLSTD (Gehring et al., 2016), random projection LSTD (Ghavamzadeh et al., 2010), and fast LSTD (Prashanth et al., 2013). As before a wide range of parameters (α, λ, η) were swept over a large set. Performance was averaged over 100 independent runs. A fixed stepsize schedule was used for the linear TD baselines, because that achieved the best performance. The results are summarized in figure 4.2.

LSTD and ATD exhibit faster initial learning compared to all other methods. This is particularly impressive since k is less than 5% of the size of $\mathbf{A}$. Both fast LSTD and projected LSTD perform considerably worse than the linear TD-methods, while iLSTD exhibits high parameter sensitivity. tLSTD has no tunable parameter besides k, but performs poorly due to the high stochasticity in the policy—additional experiments with randomness in action selection of 0% and 10% yielded better performance for tLSTD, but never equal to ATD. The true online linear methods perform very well compared to

ATD, but this required sweeping hundreds of combinations of α and λ, whereas ATD exhibited little sensitivity to its regularization parameter (see Figure 4.2 RHS); ATD achieved excellent performance with the same parameter setting as we used in Boyan's chain.[2]

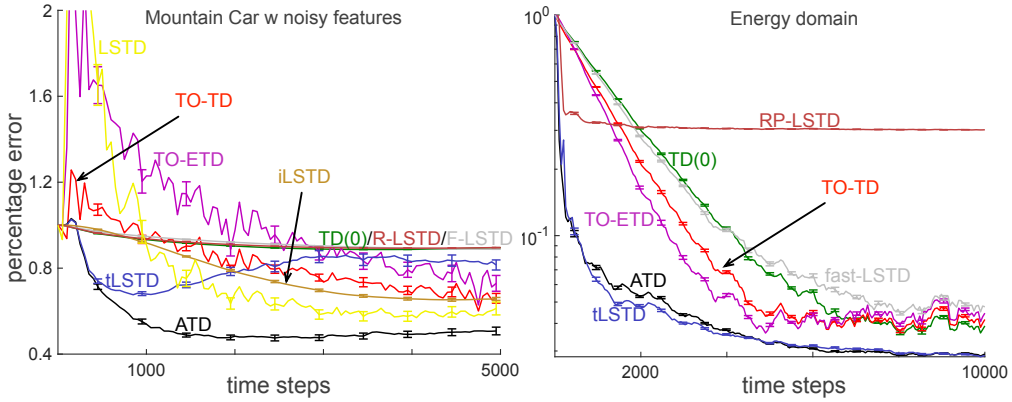


Figure 4.3: Learning curves on Mountain Car with noisy features (LHS) and on Energy allocation (RHS), and the latter's y-axis is in log scale. We show percentage absolute mean error v.s. number of training time steps.

We ran an additional experiment in Mountain Car to more clearly exhibit the benefit of ATD over existing methods. We used the same setting as above, except that 100 additional features were added to each feature vector, with 50 of them randomly set to one and the rest zero. This noisy feature vector is meant to emulate a situation such as a robot that has a sensor that becomes unreliable, generating noisy data, but the remaining sensors are still useful for the task at hand. The results are summarized in Figure 4.3. Naturally all methods are adversely effected by this change, however ATD's low rank approximation enables the agent to ignore the unreliable feature information and learn efficiently. tLSTD, as suggested by our previous experiments, does not seem to cope well with the increase in stochasticity.

Our final experiment compares the performance of several sub-quadratic complexity policy evaluation methods in an industrial energy allocation simulator (Salas & Powell, 2013; Jiang et al., 2014) with much larger feature

[2]For the remaining experiments in the paper, we excluded the TD methods without true online traces because they perform worse than their true online counterparts in all our experiments. This result matches the results in van Seijen et al. (2016).

dimension (see Figure 4.3). The problem was originally modeled as a finite horizon undiscounted task (Salas & Powell, 2013), with four state variables at each time step: the amount of energy in the storage device R_t, the net amount of wind energy E_t, time aggregate demand D_t, and price of electricity P_t in the spot market. The reward function encodes the revenue earned by the agent's energy allocation strategy as a real value number. The original goal of the control problem is to maximize revenue by regulating the energy flow among the following four components: 1) the wind farms, which generate energy; 2) the storage device, which stores energy; 3) the grid, which transports energy; 4) and the market end, which consumes energy. We concern about the policy evaluation problem, and the policy to be evaluated was produced by an approximate dynamic programming algorithm from the literature(Salas & Powell, 2013). The simulation program is from *Energy storage datasets II* from *http://castlelab.princeton.edu*. We refer to Appendix A.1.4 for any missing details.

Again, we used tile coding to convert the state variable into high-dimensional binary feature vectors, similar to how the Acrobot domain was encoded (Sutton, 1996). We tile coded all 3-wise combinations, all pair-wise combinations, and each of the five state variables independently (sometimes called stripped tilings). More specifically we used:

- all five one-wise tilings of 5 state variables, with gridsize = 4, numtilings = 32 (memory = $5 \times 4 \times 32$)
- all ten two-wise tilings of 5 state variables, with gridsize = 4, numtilings = 32 (memory = $10 \times 4^2 \times 32$)
- all ten three-wise tilings of 5 state variables, with gridsize = 2, numtilings = 32 (memory = $10 \times 2^3 \times 32$)

This resulted in a binary feature vector of length 8320, which we hashed down to $8192 = 2^{13}$. Training data and evaluation were conducted in the exact same manner as the Mountain car experiment. We refer to Appendix A.1.4 for hyper-parameter details.

As before, we report PAME and the true targets are computed from Monte Carlo rollouts, averaging performance over 50 independent runs and selecting and testing parameters from an extensive set (detailed in the Appendix A.1.4). The policy was optimized ahead of time and fixed, and the feature vectors were produced via tile coding, resulting in an 8192 dimensional feature vector with 800 units active on each step. Although the feature dimension here is still relatively small, a quadratic method like LSTD nonetheless would require over 67 million operations per time step, and thus methods that can exploit low rank approximations are of particular interest. The results indicate that both ATD and tLSTD achieve the fastest learning, as expected. The intrinsic rank in this domain appears to be small compared to the feature dimension—which is exploited by ATD and tLSTD with $k = 40$—while the performance of tLSTD indicates that the domain exhibits little stochasticity. The appendix contains additional results for this domain—in the small rank setting ATD significantly outperforms tLSTD.

4.3 Comparison of ATD with Popular First-order Stochastic Optimization Methods

There have been many developments in first-order stochastic gradient descent (SGD) optimization methods to accelerate learning. These methods are typically used in a conventional machine learning setting where a gradient vector can be calculated. We apply those methods to linear TD algorithms by considering the TD's updating direction $\delta_t \mathbf{e}_t$ as the gradient, though it is theoretically not a gradient of any function (see Maei (2011, Page 19)). Hence it is theoretically unclear where these TD variants would converge and what objective function they are optimizing.

We compare our method to several well-known and widely-used first-order stochastic gradient methods, including **AdaGrad** by Duchi et al. (2011), **RMSProp** by Tieleman & Hinton (2012), **Adam** optimization method by Kingma & Ba (2014), and **AMSGrad** by Reddi et al. (2018). We review those algorithms and their underlying ideas. We then introduce how one can adapt

them to TD learning and present empirical study to compare these methods and our ATD method. We conclude this section by discussing the similarities and differences between these TD variants and ATD methods.

Notations. We introduce a few general notations for the convenience of describing the algorithms below. We denote the gradient descent direction of the loss function being minimized as $-\mathbf{g}(\mathbf{w})$, where $\mathbf{w}$ is the parameter vector we attempt to learn. Vector notations with subscript t indicates the variable at the tth iteration. $diag(\cdot)$ operation extracts the diagonal elements of the input matrix. α is the learning rate. Whenever mathematical operations operate on a vector, the computation is element-wise. The initialization of vectors is zero vectors unless otherwise specified. Other notations will be introduced inline whenever necessary.

Adagrad. Define

$$\mathbf{v}_t = \mathbf{v}_{t-1} + \mathbf{g}_t^2,$$

Adagrad uses the update

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \frac{1}{\sqrt{\mathbf{v}_t} + \epsilon_s} \mathbf{g}_t,$$

where $\epsilon_s > 0$ is some small smoothing parameter added to the denominator to avoid zero values. The element-wise vector inverse takes linear time in terms of weight dimension. Adagrad is an extension of SGD, which improves the convergence rate. The intuitive understanding of this updating rule is to scale down the effect of those highly active weight units (i.e., those with large derivative magnitude) to not overly dominate the learning process.

RMSProp. RMSProp replaces the updating rule for $\mathbf{v}_t$ by using an exponential moving average: $\mathbf{v}_t = (1 - \beta)\mathbf{v}_{t-1} + \beta \mathbf{g}_t^2$, resulting in one more hyper-parameter β. This modification avoids the problem of extremely small learning rate when the magnitude of $\mathbf{v}_t$ grows too large.

Adam. Adam is probably the most popular optimization algorithm at the time the thesis is written. Comparing with RMSProp, it has two main modifications: 1) introducing the first moment estimation and 2) introducing bias correction for the first and the second moment estimation. The updating rules are as follows.

$$\begin{aligned}
\mathbf{m}_t &= (1-\beta_1)\mathbf{m}_{t-1} + \beta_1 \mathbf{g}_t, \\
\mathbf{v}_t &= (1-\beta_2)\mathbf{v}_{t-1} + \beta_2 \mathbf{g}_t^2, \\
\hat{\mathbf{m}}_t &= \mathbf{m}_t / (1-\beta_1^t), \\
\hat{\mathbf{v}}_t &= \mathbf{v}_t / (1-\beta_2^t), \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon_s},
\end{aligned}$$

where β_1, β_2 are hyper-parameters deciding the averaging rate of the first and second moment estimation.

AMSGrad. AMSGrad is proposed to resolve an issue in the convergence proof of Adam optimization method. The only modification comparing with Adam is $\hat{\mathbf{v}}_t = \max(\hat{\mathbf{v}}_{t-1}, \mathbf{v}_t)$.

Adapting to linear TD. There is no straightforward way to apply the above methods developed in a conventional machine learning setting to the linear TD algorithm in RL. It is because the linear TD method is not derived by taking the gradient of any objective function, and the term $\delta_t \mathbf{x}_t$ is not a gradient of any function. This can be proved by checking the symmetric property of the second-order derivative.[3] As a result, the gradient vector $\mathbf{g}$ is not available in linear TD. However, an intuitive implementation is to still treat $-\delta_t \mathbf{x}_t$ as the gradient vector $\mathbf{g}_t$ and then apply the above updating rules. The resulting TD variants are named as **TD-[Adam/AdaGrad/AMSGrad/RMSProp]**. Though they lack theoretical support, they are empirically effective, as shown below.

[3] TD is sometimes called semi/pseudo-gradient method (Sutton & Barto, 2018).

Empirical results. This experiment aims to investigate if there is still an advantage when comparing our ATD(-SVD) with the above TD variants on a policy evaluation task. Hence, we still evaluate algorithms by examining how close the learned states' values are to true values. It should be noted that ATD is convergent to the minimizer of the MSPBE objective; however, the convergent behavior of those TD variants is unknown.

We consider the Mountain Car domain with the same setting as described in the previous section. For bootstrap parameter λ, we sweep over the range

$$\lambda \in \{0.7, 0.9, 0.93, 0.95, 0.97, 0.99, 1.0\}.$$

For first and second moment estimates forgetting rates β_1, β_2, we sweep over

$$\{0.01, 0.1, 0.4, 0.9, 0.99, 0.999\}$$

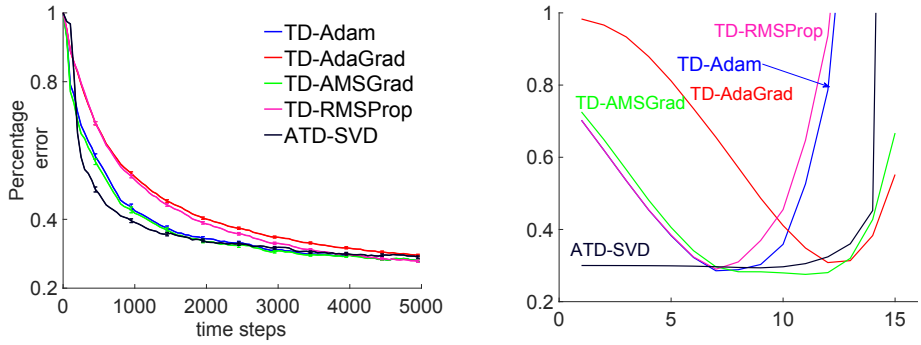
whenever applicable. For learning rate α, we sweep over

$$\{2^{i-10} | i \in \{0, 1, ..., 13, 14\}\},$$

and the regularization weight for ATD is swept over the same range as learning rate but scaled by 0.01 (i.e., $0.01 \times \alpha$). Note that this range of regularization weights includes much larger values than we used in the previous section, which uses $\{0.001 \times 2^{i-7} | i \in \{0, 1, ..., 12\}\}$. Such details are in Appendix A.1.4.

Figure 4.4(a) shows the learning curves. It can be seen that our algorithm still performs better than those TD variants during the early learning stage. Among these TD variants, TD-Adam and TD-AMSGrad seem to be better than other variants. Figure 4.4(b) shows the sensitivity curves, which strongly support our claimed benefit of ATD—insensitivity to the hyper-parameter η. Note that ATD becomes worse only when choosing an extraordinarily large regularization weight. All first-order TD variants are much more sensitive to α in that only a small range of it leads to low error. An additional interesting observation is that AMSGrad is less sensitive to the learning rate than Adam. This seems the first time to show AMSGrad's advantage in sensitivity. We believe this result makes intuitive sense: AMSGrad was proposed to fix an issue in the convergence of Adam, whose direct consequence avoids an abrupt

change in learning rate due to the moment estimations. As a result, we expect AMSGrad to be more robust to different learning rates than Adam.



(a) Learning curve

(b) Sensitivity curve

Figure 4.4: Comparing TD variants with ATD $k = 50$ on Mountain car domain. (a) shows the percentage absolute mean error (PAME) v.s. the number of training time steps. (b) shows the sensitivity curve. The x-axis means 15 regularization weights (ηs) for ATD and learning rates αs for TD variants. For each hyper-parameter on x-axis, we report the error by optimizing over other hyper-parameters for TD variants. The results are averaged over 100 random seeds.

4.4 Convergence of ATD with SVD

We now present the convergence result of our ATD's expected updating rule. As with previous convergence results for TD learning algorithms, the first key step is to prove that the expected update converges to the TD fixed point. Unlike previous proofs of convergence in expectation, we do not require the true $\mathbf{A}$ to be full rank. This generalization is important, because as shown previously, $\mathbf{A}$ is often low-rank, even if features are linearly independent (Bertsekas, 2007; Gehring et al., 2016). Further, ATD should be more effective if $\mathbf{A}$ is low-rank, and so requiring a full-rank $\mathbf{A}$ would limit the typical use-cases for ATD.

To get across the main idea, we first prove convergence of ATD with weightings that give positive semi-definite $\mathbf{A}$, which holds under mild technical conditions according to Yu (2015, Proposition C.1). Note that in this section, we do not use the emphatic weighting subscript m for the matrix notations

$\mathbf{A}, \mathbf{b}$ for conciseness. A more general proof for other weightings is in the Appendix A.1.1.

Assumption 1. $\mathbf{A}$ *is diagonalizable, that is, there exists invertible* $\mathbf{Q} \in \mathbb{R}^{d\times d}$ *with normalized columns (eigenvectors) and diagonal* $\mathbf{\Lambda} \in \mathbb{R}^{d\times d}$*, where the diagonal elements are the real eigenvalues:* $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_d)$*, such that* $\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}$*. Assume the ordering* $\lambda_1 \geq \ldots \geq \lambda_d$*.*

Assumption 2. $\alpha \in (0, 2)$ *and* $0 < \eta \leq \lambda_1^{-1} \max(2 - \alpha, \alpha)$*.*

Finally, we introduce an assumption that is only used to characterize the convergence rate. This condition has been previously used (Hansen, 1990; Gehring et al., 2016) to enforce a level of smoothness on the system.

Assumption 3. *The linear system defined by* $\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}$ *and* $\mathbf{b}$ *satisfy the discrete Picard condition: for some* $p > 1$*,* $|(\mathbf{Q}^{-1}\mathbf{b})_j| \leq \lambda_j^p$ *for all* $j = 1, \ldots, \textit{rank}(\mathbf{A})$*.*

Theorem 2. *Under Assumptions 1 and 2, for any* $k \geq 0$*, let* $\hat{\mathbf{A}}$ *be the rank-k approximation* $\hat{\mathbf{A}} = \mathbf{Q}\mathbf{\Lambda}_k\mathbf{Q}^{-1}$ *of* $\mathbf{A}$*, where* $\mathbf{\Lambda}_k \in \mathbb{R}^{d\times d}$ *with* $\mathbf{\Lambda}_k(j, j) = \lambda_j$ *for* $j = 1, \ldots, k$ *and zero otherwise. The expected updating rule in (3.11):*

$$\mathbf{w}_{t+1} = \mathbf{w}_t + (\alpha_t \hat{\mathbf{A}}^\dagger + \eta\mathbf{I})\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}]$$

converges to the fixed-point $\mathbf{w}^\star = \mathbf{A}^\dagger\mathbf{b}$*.*
Further, if Assumption 3 is satisfied and $\mathbf{w}$ *is initialized as a zero vector (i.e.,* $\mathbf{w}_0 = \mathbf{0}$*), the convergence rate of* $\mathbf{w}_t \mapsto \mathbf{w}^\star$ *is*

$$\|\mathbf{w}_t - \mathbf{w}^\star\| \leq \max\Big(\max_{j\in\{1,\ldots,k\}} |1 - \alpha - \eta\lambda_j|^t \lambda_j^{p-1}, \max_{j\in\{k+1,\ldots,\textit{rank}(\mathbf{A})\}} |1 - \eta\lambda_j|^t \lambda_j^{p-1}\Big)$$

Proof. We use a general result about stationary iterative methods which is applicable to the case where $\mathbf{A}$ is not full column rank. Shi et al. (2011, Theorem 1.1) states that given a singular and consistent linear system $\mathbf{A}\mathbf{w} = \mathbf{b}$[4] where $\mathbf{b}$ is in the range of $\mathbf{A}$, the stationary iteration with $\mathbf{w}_0 \in \mathbb{R}^d$ for

[4] If a system has at least one solution, it is said to be consistent.

$t = 1, 2, \ldots$

$$\mathbf{w}_t = (\mathbf{I} - \mathbf{BA})\mathbf{w}_{t-1} + \mathbf{Bb} \tag{4.2}$$

converges to the solution $\mathbf{w} = \mathbf{A}^\dagger \mathbf{b}$ if and only if the following three conditions are satisfied.

Condition **I**: the matrix $\mathbf{I} - \mathbf{BA}$ has absolute eigenvalues equal to 1 or strictly less than 1.

Condition **II**: $\text{rank}(\mathbf{BA}) = \text{rank}[(\mathbf{BA})^2]$.

Condition **III**: $\text{nullspace}(\mathbf{BA}) = \text{nullspace}(\mathbf{A})$.

We verify these conditions to prove the result. First, by (Yu, 2015), there exists at least one $\mathbf{w}$ s.t. $\mathbf{w} = \mathbf{A}^\dagger \mathbf{b}$.

To rewrite our updating rule (3.11) to be expressible in terms of (4.2), let $\mathbf{B} = \alpha \hat{\mathbf{A}}^\dagger + \eta \mathbf{I}$, giving

$$\begin{aligned}
\mathbf{BA} &= \alpha \hat{\mathbf{A}}^\dagger \mathbf{A} + \eta \mathbf{A} = \alpha \mathbf{Q} \mathbf{\Lambda}_k^\dagger \mathbf{Q}^{-1} \mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^{-1} + \eta \mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^{-1} \\
&= \alpha \mathbf{Q} \mathbf{I}_k \mathbf{Q}^{-1} + \eta \mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^{-1} \\
&= \mathbf{Q}(\alpha \mathbf{I}_k + \eta \mathbf{\Lambda}) \mathbf{Q}^{-1}
\end{aligned} \tag{4.3}$$

where $\mathbf{I}_k$ is a diagonal matrix with the indices $1, \ldots, k$ set to 1, and the rest zero.

Proof for condition I. Using (4.3), $\mathbf{I} - \mathbf{BA} = \mathbf{Q}(\mathbf{I} - \alpha \mathbf{I}_k - \eta \mathbf{\Lambda}) \mathbf{Q}^{-1}$. To bound the maximum absolute value in the diagonal matrix $\mathbf{I} - \alpha \mathbf{I}_k - \eta \mathbf{\Lambda}$, we consider eigenvalue λ_j in $\mathbf{\Lambda}$, and address two cases. Because $\mathbf{A}_m$ is positive semi-definite for the assumed m (Sutton et al., 2016), $\lambda_j \geq 0$ for all $j = 1, \ldots, d$.

Case 1: $j \leq k$.

$$\begin{aligned}
&|1 - \alpha - \eta \lambda_j| && \triangleright \text{ for } 0 < \eta < \max\left(\frac{2-\alpha}{\lambda_1}, \frac{\alpha}{\lambda_1}\right) \\
&< \max(|1 - \alpha|, |1 - \alpha - (2 - \alpha)|, |1 - \alpha - \alpha|) \\
&= \max(|1 - \alpha|, 1, 1) < 1 && \triangleright \text{ because } \alpha \in (0, 2).
\end{aligned}$$

Case 2: $j > k$. $\quad |1 - \eta\lambda_j| < 1 \quad$ if $0 < \eta < 2/\lambda_j$ which is true for $\eta = \lambda_1^{-1}\max(2-\alpha, \alpha)$ for any $\alpha \in (0, 2)$.

Proof for condition II. $(\mathbf{BA})^2$ does not change the number of positive eigenvalues, so the rank is unchanged.

Proof for condition III. To show the nullspaces of $\mathbf{BA}$ and $\mathbf{A}$ are equal, it is sufficient to prove $\mathbf{BAw} = \mathbf{0}$ if and only if $\mathbf{Aw} = \mathbf{0}$ for $\mathbf{w} \neq \mathbf{0}$. It is easy to see that the matrix $\mathbf{B} = \mathbf{Q}(\alpha\mathbf{\Lambda}_k + \eta\mathbf{I})\mathbf{Q}^{-1}$, is invertible because 0 is not one of its eigenvalues, since $\eta > 0$. This reasoning is supported by the invertible matrix theorem (Stover, 2006). For any $\mathbf{w} \in \text{nullspace}(\mathbf{A})$, we get $\mathbf{BAw} = \mathbf{B0} = \mathbf{0}$, and so $\mathbf{w} \in \text{nullspace}(\mathbf{BA})$. For any $\mathbf{w} \in \text{nullspace}(\mathbf{BA})$, $\mathbf{BAw} = \mathbf{0} \implies \mathbf{Aw} = \mathbf{B}^{-1}\mathbf{0} = \mathbf{0}$, and so $\mathbf{w} \in \text{nullspace}(\mathbf{A})$.

Convergence rate. On each step, we update with $\mathbf{w}_{t+1} = (\mathbf{I} - \mathbf{BA})\mathbf{w}_t + \mathbf{Bb}$. After inductively computation, we can acquire

$$\mathbf{w}_t = \sum_{i=0}^{t-1}(\mathbf{I} - \mathbf{BA})^i\mathbf{Bb} + (\mathbf{I} - \mathbf{BA})^t\mathbf{w}_0 = \sum_{i=0}^{t-1}(\mathbf{I} - \mathbf{BA})^i\mathbf{Bb}$$

For $\bar{\mathbf{\Lambda}} = \mathbf{I} - \alpha\mathbf{I}_k - \eta\mathbf{\Lambda}$, because $(\mathbf{I} - \mathbf{BA})^i = \mathbf{Q}\bar{\mathbf{\Lambda}}^i\mathbf{Q}^{-1}$,

$$\begin{aligned}
\mathbf{w}_t &= \mathbf{Q}\left(\sum_{i=0}^{t-1}\bar{\mathbf{\Lambda}}^i\right)\mathbf{Q}^{-1}\mathbf{Q}(\alpha\mathbf{\Lambda}_k^\dagger + \eta\mathbf{I})\mathbf{Q}^{-1}\mathbf{b} \\
&= \mathbf{Q}\left(\sum_{i=0}^{t-1}\bar{\mathbf{\Lambda}}^i\right)(\alpha\mathbf{\Lambda}_k^\dagger + \eta\mathbf{I})\mathbf{Q}^{-1}\mathbf{b}
\end{aligned}$$

and because $\mathbf{w}_t \to \mathbf{w}^\star$,

$$\begin{aligned}
\|\mathbf{w}_t - \mathbf{w}^\star\| &= \|\mathbf{Q}\left(\sum_{i=0}^{\infty}\bar{\mathbf{\Lambda}}^i - \sum_{i=0}^{t-1}\bar{\mathbf{\Lambda}}^i\right)(\alpha\mathbf{\Lambda}_k^\dagger + \eta\mathbf{I})\mathbf{Q}^{-1}\mathbf{b}\| \\
&\leq \|\mathbf{Q}\bar{\mathbf{\Lambda}}_t(\alpha\mathbf{\Lambda}_k^\dagger + \eta\mathbf{I})\mathbf{Q}^{-1}\mathbf{b}\| \qquad \triangleright\ \bar{\mathbf{\Lambda}}_t(j,j) \stackrel{\text{def}}{=} \frac{\bar{\lambda}_j^t}{1 - \bar{\lambda}_j} \\
&\leq \|\mathbf{Q}\|\|\bar{\mathbf{\Lambda}}_t(\alpha\mathbf{\Lambda}_k^\dagger + \eta\mathbf{I})\mathbf{Q}^{-1}\mathbf{b}\| \\
&\leq \|\bar{\mathbf{\Lambda}}_t(\alpha\mathbf{\Lambda}_k^\dagger + \eta\mathbf{I})\|\|\mathbf{Q}^{-1}\mathbf{b}\|.
\end{aligned}$$

The last inequality holds because $\mathbf{Q}$ has normalized columns and $\|\mathbf{Q}\| \leq 1$.

For $j = 1, \ldots, k$, we have that the magnitude of the values in $\bar{\mathbf{\Lambda}}_t(\alpha\mathbf{\Lambda}_k^\dagger + \eta\mathbf{I})$

are

$$|\frac{(1-\alpha-\eta\lambda_j)^t}{\alpha+\eta\lambda_j}(\alpha\lambda_j^{-1}+\eta)| = |\frac{(1-\alpha-\eta\lambda_j)^t}{\lambda_j}| \leq \frac{|1-\alpha-\eta\lambda_j|^t}{\lambda_j}.$$

For $j = k, \ldots, \text{rank}(\mathbf{A})$, we get the magnitude as $|\frac{(1-\eta\lambda_j)^t}{\lambda_j}| \leq \frac{|1-\eta\lambda_j|^t}{\lambda_j}$.

Under the discrete Picard condition $|(\mathbf{Q}^{-1}\mathbf{b})_j| \leq \lambda_j^p$, the denominator λ_j cancels, giving the desired result. □

Remark 2. *By simple algebraic computation, using a customized initialization of the weight vector (i.e.,* $\mathbf{w}_0$*) would add an additional term* $|1-\alpha-\eta\lambda_j|^t \cdot \|\mathbf{Q}^{-1}\mathbf{w}_0\|$ *to the two terms corresponding to* $j \leq k$ *and* $j > k$*. Since using nonzero weight vector potentially indicates special exploration strategy, which complicates the interpretation of this bound, we focus on the simple case when the weight is initialized as a zero vector in this thesis. It could be an interesting future direction to study how to initialize the weight vector to make the constant term* $\|\mathbf{Q}^{-1}\mathbf{w}_0\|$ *small, improving the convergence rate.*

Theorem 2 gives insight into the utility of ATD for speeding up convergence, as well as the effect of k. Consider TD(λ), which has positive definite $\mathbf{A}$ in on-policy learning (Sutton (1988, Theorem 2)). The theorem guarantees that ATD convergences to the TD fixed-point, for any k. For $k = 0$, the expected ATD update is exactly the expected TD update. Now, we can compare the convergence rate of TD and ATD, using the above convergence rate.

Take for instance the setting $\alpha = 1$ for ATD, which is common for second-order methods and let $p = 2$. The rate of convergence reduces to the maximum of $\max_{j\in\{1,\ldots,k\}} \eta^t\lambda_j^{t+1}$ and $\max_{j\in\{k+1,\ldots,\text{rank}(\mathbf{A})\}} |1-\eta\lambda_j|^t\lambda_j$. In early learning, the convergence rate for TD is dominated by $|1-\eta\lambda_1|^t\lambda_1$, because λ_j is larger relative to $|1-\eta\lambda_j|^t$ for small t. ATD, on the other hand, for a larger k, can pick a smaller η and so has a much smaller value for $j = 1$, i.e., $\eta^t\lambda_1^{t+1}$, and $|1-\eta\lambda_j|^t\lambda_j$ is small because λ_j is small for $j > k$. As k gets smaller, $|1-\eta\lambda_{k+1}|^t\lambda_{k+1}$ becomes larger, slowing convergence. For low-rank domains, however, k could be quite small and the preconditioner could still improve the convergence rate in early learning.

Hence, ATD, when combined with a low-rank approximation, converges in expectation to the TD fixed-point, with convergence rate dependent on the choice of rank. Unlike previous subquadratic methods, consistency is guaranteed for ATD even when the rank is chosen to be one. The regularization $\eta > 0$ is key to ensure this consistency, by providing a full rank preconditioner $\alpha_t \hat{\mathbf{A}}_t^\dagger + \eta \mathbf{I}$.

ATD is a quasi-second order method, meaning sensitivity to parameters should be reduced and thus it should be simpler to set the parameters. The convergence rate provides intuition that, for reasonably chosen k, the regularizer η should be small—smaller than a typical stepsize for TD. Additionally, because ATD is a stochastic update, not the expected update, we make use of typical conventions from stochastic gradient descent to set our parameters. We set $\alpha_t = \frac{\alpha_0}{t}$, as in previous stochastic second-order methods (Schraudolph et al., 2007), where we choose $\alpha_0 = 1$ and set η to a small fixed value. Our choice for η represents a small final stepsize, as well as matching the convergence rate intuition.

On the bias of subquadratic methods. The ATD(λ) update was derived to ensure convergence to the minimum of the MSPBE, either for the on-policy or off-policy setting. Our algorithm summarizes past information, in $\hat{\mathbf{A}}$, to improve the convergence rate, without requiring quadratic computation and storage. Prior work aspired to the same goal, however, the resultant algorithms are biased. The iLSTD algorithm can be shown to converge for a specific class of feature selection mechanisms (Geramifard et al. (2007, Theorem 2)); this class, however, does not include the greedy mechanism that is used in iLSTD algorithm to select a descent direction. The random projections variant of LSTD (Ghavamzadeh et al., 2010) can significantly reduce the computational complexity compared with conventional LSTD, with projections down to size k, but the reduction comes at a cost of an increase in the approximation error (Ghavamzadeh et al., 2010). Fast LSTD (Prashanth et al., 2013) does randomized TD updates on a batch of data, which could be run incrementally with $O(dk)$ by using mini-batches of size k. Though it has a nice theoretical

characterization, this algorithm is restricted to $\lambda = 0$. Finally, the most related algorithm is tLSTD as we discussed in Section 4.1, which also uses a low-rank approximation to $\mathbf{A}$.

In ATD, $\hat{\mathbf{A}}_t$ is used very differently from how $\hat{\mathbf{A}}_t$ is used in tLSTD. The tLSTD algorithm uses a similar approximation $\hat{\mathbf{A}}_t$ as ATD, but tLSTD uses it to compute a closed form solution $\mathbf{w}_t = \hat{\mathbf{A}}_t^{\dagger}\mathbf{b}_t$, and thus is biased (Theorem 1). In fact, the bias grows with decreasing k, proportionally to the magnitude of the kth largest singular value of $\mathbf{A}$. In ATD, the choice of k is decoupled from the fixed point, and so can be set to balance learning speed and computation with no fear of asymptotic bias.

Chapter 5

Approximation by Random Projection

Random projection (Johnson & Lindenstrauss, 1984) is a popular matrix sketching technique, which has been broadly applied in various practical machine learning settings due to its low computational cost and simplicity (Bingham & Mannila, 2001; Achlioptas, 2003; Freund et al., 2008). Particularly, matrix sketching usually needs to be pass-efficient. That is, a data point is read at most a constant time or even one time. Such demand makes random projection an attractive option in many practical applications whenever we expect an algorithm to utilize a data point immediately by efficient computation once it becomes available and then discard it. A fully incremental, online reinforcement learning is a natural example of such a setting.

In RL, however, before our work (Pan et al., 2017a), algorithms with matrix sketching via random projection were mainly studied theoretically, and the biased solution issue induced by such a method is not yet well resolved. Furthermore, they lack empirical investigations. This chapter focuses on a detailed study of ATD with the matrix sketching technique. For conciseness, in this thesis, whenever we say *sketching*, we mean sketching by random projection.

This chapter is mainly based on Pan et al. (2017a). We introduce the variant of ATD using matrix sketching to incrementally approximate the $\mathbf{A}$ matrix, which further reduces the computational and storage complexity comparing with the incremental truncated SVD approach presented in the previous

chapter.

The chapter is organized as follows. In Section 5.1, we briefly review relevant literature and discuss issues about sketching in RL. Based on these issues, Section 5.2 motivates to sketch the linear system for policy evaluation problems rather than the features. Then we introduce our main approach: ATD with left-sided sketching of $\mathbf{A}$ in Section 5.3. We then present empirical results to verify the efficacy of our ATD instance and to investigate the relationship between feature properties and learning performance in Section 5.4. The convergence property is studied in Section 5.5. We conclude this chapter by discussing the difference between the usage of random projection in ATD and that in a standard supervised learning setting in Section 5.6.

5.1 A Review of Sketching in RL

Sketching has been extensively used for efficient communication and solving large linear systems, with a solid theoretical foundation and a variety of different sketches (Woodruff, 2014). We investigate the utility of sketching for improving policy evaluation within reinforcement learning. Sketching has been previously used in RL, specifically to reduce the dimension of the features. Bellemare et al. (2012) replaced the standard biased hashing function used for tile coding (Sutton, 1996), instead using count-sketch.[1] Ghavamzadeh et al. (2010) investigated sketching features to reduce the dimensionality and make it feasible to run least-squares TD learning (LSTD) for policy evaluation. In LSTD, the value function is estimated by incrementally computing a $d \times d$ matrix $\mathbf{A}$, where d is the same as feature dimension, and an d-dimensional vector $\mathbf{b}$, where the parameters are estimated as the solution to this linear system.

One approach to make LSTD more feasible is to project—sketch—the features. Sketching involves sampling a random matrix $\mathbf{S} : \mathbb{R}^{k \times d}$ from a family of matrices $\mathcal{S}$, to project a given d-dimensional vector $\mathbf{x}$ to a (much smaller)

[1] They called the sketch the tug-of-war sketch, but it is more standard to call it count-sketch.

k-dimensional vector $\mathbf{Sx}$. The goal in defining this class of sketching matrices is to maintain certain properties of the original vector. The following is a standard definition for such a family.

Definition 1 (Sketching)**.** *Let d and k be positive integers, $\delta \in (0, 1)$, and $\epsilon \in \mathbb{R}^+$. Then, $\mathcal{S} \subset \mathbb{R}^{k \times d}$ is called a family of sketching matrices with parameters (ϵ, δ), if for a random matrix, $\mathbf{S}$, chosen uniformly at random from this family, we have that $\forall \mathbf{x} \in \mathbb{R}^d$*

$$\mathbb{P}\Big[(1-\epsilon)\|\mathbf{x}\|_2^2 \leq \|\mathbf{Sx}\|_2^2 \leq (1+\epsilon)\|\mathbf{x}\|_2^2\Big] \geq 1-\delta,$$

where the probability is w.r.t. to the distribution over $\mathbf{S}$.

We will explore the utility of sketching the features with several common sketches. These sketches all require $k = \Omega(\epsilon^{-2} \ln(1/\delta) \ln d)$.

Gaussian random projections, also known as the JL-Transform (Johnson & Lindenstrauss, 1984), has each entry in $\mathbf{S}$ i.i.d. sampled from a Gaussian, $\mathcal{N}(0, \frac{1}{k})$.

Count sketch selects exactly one uniformly picked non-zero entry in each column, and sets that entry to either 1 or -1 with equal probability (Charikar et al., 2002; Gilbert & Indyk, 2010). The Tug-of-War sketch (Alon et al., 1996) performs very similarly to Count sketch in our experiments, so we omit it.

Combined sketch is the product of a count sketch matrix and a Gaussian projection matrix (Wang, 2015; Boutsidis & Woodruff, 2015).

Hadamard sketch—the Subsampled Randomized Hadamard Transform—is computed as $\mathbf{S} = \frac{1}{\sqrt{kd}}\mathbf{DH}_d\mathbf{P}$, where $\mathbf{D} \in \mathbb{R}^{d \times d}$ is a diagonal matrix with each diagonal element uniformly sampled from $\{1, -1\}$, $\mathbf{H}_d \in \mathbb{R}^{d \times d}$ is a Hadamard matrix and $\mathbf{P} \in \mathbb{R}^{d \times k}$ is a column sampling matrix (Ailon & Chazelle, 2006).

With sketching, the norm distance between the recovery $\mathbf{S}^\top\mathbf{Sx}$ and the original $\mathbf{x}$ is low, with high probability. For the above families, the entries in $\mathbf{S}$ are zero-mean i.i.d., giving $\mathbb{E}[\mathbf{S}^\top\mathbf{S}] = \mathbf{I}$ over all possible $\mathbf{S}$. Consequently, in expectation, the recovery $\mathbf{S}^\top\mathbf{Sx}$ is equal to $\mathbf{x}$. For a stronger result, a Chernoff bound can be used to bound the deviation of $\mathbf{S}^\top\mathbf{S}$ from this expected value:

for the parameters (ϵ, δ) of the matrix family, we get that $\mathbb{P}\Big[(1-\epsilon)I \prec \mathbf{S}^\top\mathbf{S} \prec (1+\epsilon)I\Big] \geq 1-\delta$.

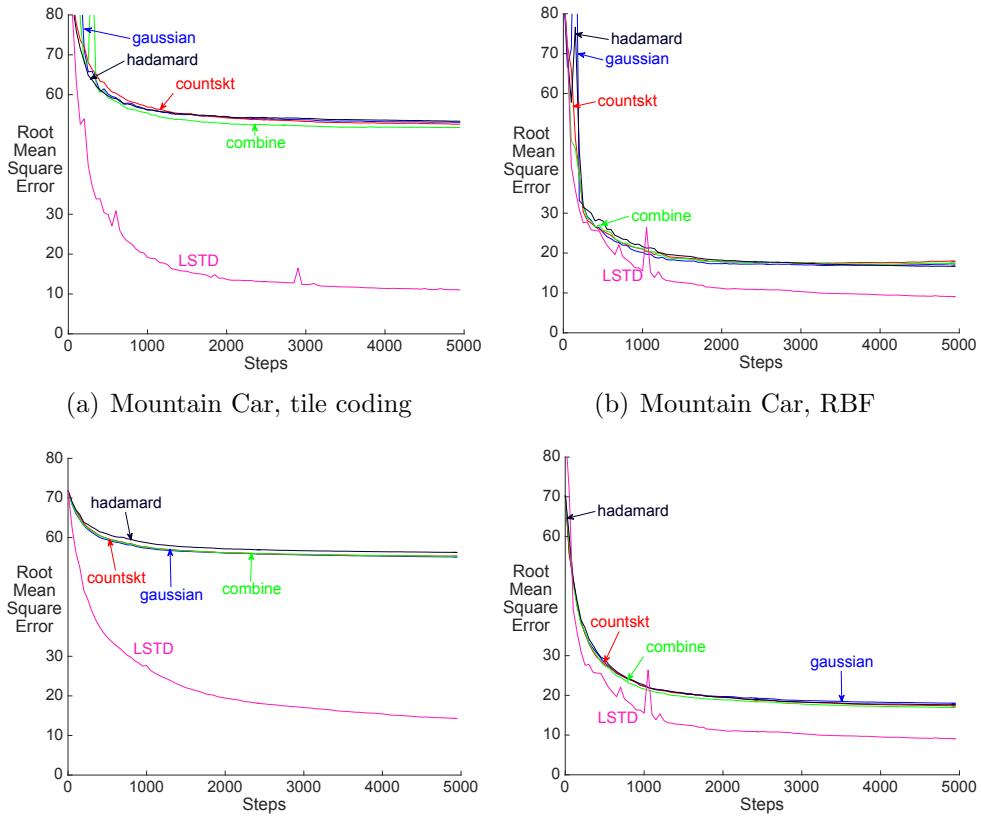
These properties suggest that using sketching for the feature vectors should provide effective approximations. Bellemare et al. (2012) showed that they could use these projections for tile coding, rather than the biased hashing function that is typically used, to improve learning performance for the control setting. The efficacy, however, of sketching given features versus using the unsketched features is not well-understood.

We investigate the properties of sketching the features, shown in Figure 5.1 with a variety of sketches in two benchmark domains for RBF and tile-coding representations for an overview of these representations. For both domains, the observations space is 2-dimensional, with expansion to $d = 1024$ and $k = 50$. The results are averaged over 50 runs, with ξ, λ swept over 13 values, with ranges listed in Appendix A.2.3. We see that sketching the features can incur significant bias, particularly for tile coding, even with a reasonably large $k = 50$ to give $\mathcal{O}(dk)$ runtimes. This bias reduces with k but remains quite high and is likely too unreliable for practical use. As a result, a natural question is if we can benefit from sketching, with minimal bias or without incurring any bias at all.

5.2 Sketching the LSTD Linear System

All of the work on sketching within reinforcement learning has investigated sketching the features; however, we can instead consider sketching the linear system, $\mathbf{A}\mathbf{w} = \mathbf{b}$. For such a setting, we can sketch the left and right subspaces of $\mathbf{A}$ with different sketching matrices, $\mathbf{S}_L \in \mathbb{R}^{k_L \times d}$ and $\mathbf{S}_R \in \mathbb{R}^{k_R \times d}$. Depending on the choices of k_L and k_R, we can then solve the smaller system $\mathbf{S}_L\mathbf{A}\mathbf{S}_R^\top\mathbf{S}_R\mathbf{w} = \mathbf{S}_L\mathbf{b}$ efficiently.

One natural improvement should be in one-sided sketching. By only sketching from the left, for example, and setting $\mathbf{S}_R = \mathbf{I}$, we do not project $\mathbf{w}$. Rather, we only project the constraints to the linear system $\mathbf{A}\mathbf{w} = \mathbf{b}$. Importantly, this does not introduce bias: the original solution $\mathbf{w}$ to $\mathbf{A}\mathbf{w} = \mathbf{b}$



(a) Mountain Car, tile coding

(b) Mountain Car, RBF

(c) Puddle World, tile coding

(d) Puddle World, RBF

Figure 5.1: Efficacy of different sketches for sketching the features for LSTD, with $k = 50$. The RMSE is w.r.t. the optimal value function, computed using rollouts. LSTD(λ) is included as the baseline, with $\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}$, with the other curves corresponding to different sketches of the features, to give $\mathbf{w} = (\mathbf{SAS}^\top)^{-1}\mathbf{Sb}$ as used for the random projections LSTD algorithm. The RBF width in Mountain Car is $\sigma = 0.12$ times the range of the state space and in Puddle World is $\sigma = \sqrt{0.0072}$. The 1024 centers for RBFs are chosen to uniformly cover the 2-d space in a grid. For tile coding, we discretize each dimension by 10, giving 10×10 grids, use 10 tilings, and set the memory size as 1024. The bias is high for tile coding features and much better for the RBF features, though still quite large. The different sketches perform similarly.

is also a solution to $\mathbf{SAw} = \mathbf{Sb}$ for any sketching matrix $\mathbf{S}$. The projection, however, removes uniqueness in terms of the solutions $\mathbf{w}$, since the system is under-constrained. Conversely, by only sketching from the right, and setting $\mathbf{S}_L = \mathbf{I}$, we constrain the space of solutions to a unique set, and do not remove any constraints. For this setting, however, it is unlikely that $\mathbf{w}$ with $\mathbf{Aw} = \mathbf{b}$ satisfies $\mathbf{AS}^\top\mathbf{w} = \mathbf{b}$.

The conclusion from many initial experiments is that the key benefit from asymmetric sketching is when only sketching from the left. We experimented with all pairwise combinations of Gaussian random projections, Count sketch, Tug-of-War sketch, and Hadamard sketch for $\mathbf{S}_L$ and $\mathbf{S}_R$. We additionally experimented with only sketching from the right, setting $\mathbf{S}_L = \mathbf{I}$. In all of these experiments, we found asymmetric sketching provided little to no benefit over using $\mathbf{S}_L = \mathbf{S}_R$ and that sketching only from the right also performed similarly to using $\mathbf{S}_L = \mathbf{S}_R$. We further investigated column and row selection sketches (see Wang (2015) for a thorough overview), but also found these to be ineffective. We, therefore, proceed with an investigation into effectively using left-side sketching. In the next section, we provide an efficient $\mathcal{O}(dk)$ algorithm to compute $(\mathbf{S}_L\mathbf{A})^\dagger$, to enable computation of $\mathbf{w} = (\mathbf{S}_L\mathbf{A})^\dagger\mathbf{S}_L\mathbf{b}$ and for use within an unbiased quasi-Newton algorithm.

We conclude this section with an interesting connection to a data-dependent projection method that has been used for policy evaluation that further motivates the utility of sketching only from the left.

This algorithm—called truncated LSTD (tLSTD)—was described in the last Chapter. It incrementally maintains a rank k approximation of $\mathbf{A}$ matrix, using an incremental SVD. We show below that this approach is projecting $\mathbf{A}$ from the left with the top k left singular vectors. This is called a data-dependent projection because the projection depends on the observed data, as opposed to the data-independent projection—the sketching matrices—which is randomly sampled independently of the data.

Proposition 1. *Let* $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ *be singular value decomposition of the true* $\mathbf{A}$. *Assume the singular values are in decreasing order and let* $\mathbf{\Sigma}_k$ *be the top*

k singular values, with corresponding k left singular vectors $\mathbf{U}_k$ *and k right singular vectors* $\mathbf{V}_k$*. Then the solution* $\mathbf{w} = \mathbf{V}_k \mathbf{\Sigma}_k^\dagger \mathbf{U}_k^\top \mathbf{b}$ *(used for tLSTD) corresponds to LSTD using asymmetric sketching with* $\mathbf{S}_L = \mathbf{U}_k^\top$ *and* $\mathbf{S}_R = \mathbf{I}$*.*

Proof. We know $\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_d]$ for singular vectors $\mathbf{u}_i \in \mathbb{R}^d$ with $\mathbf{u}_i^\top \mathbf{u}_i = 1$ and $\mathbf{u}_i^\top \mathbf{u}_j = 0$ for $i \neq j$. Since $\mathbf{U}_k = [\mathbf{u}_1, \dots, \mathbf{u}_k]$, we get that $\mathbf{U}_k^\top \mathbf{U} = [\mathbf{I}_k \;\; \mathbf{0}_{d-k}] \in \mathbb{R}^{k \times d}$ for k-dimensional identity matrix $\mathbf{I}_k$ and zero matrix $\mathbf{0}_{d-k} \in \mathbb{R}^{k \times (d-k)}$. Then we get that $\mathbf{S}_L \mathbf{b} = \mathbf{U}_k^\top \mathbf{b}$ and $\mathbf{S}_L \mathbf{A} = [\mathbf{I}_k \;\; \mathbf{0}_{d-k}] \mathbf{\Sigma} \mathbf{V}^\top = \mathbf{\Sigma}_k \mathbf{V}^\top = \mathbf{\Sigma}_k \mathbf{V}_k^\top$. Therefore, $\mathbf{w} = (\mathbf{S}_L \mathbf{A})^\dagger \mathbf{S}_L \mathbf{b} = \mathbf{V}_k \mathbf{\Sigma}_k^\dagger \mathbf{U}_k^\top \mathbf{b}$. □

5.3 Our Approach: Left-sided Projection

In this section, we develop an efficient approach to use the smaller, sketched matrix $\mathbf{SA}$ for incremental policy evaluation. We propose to sketch the linear system in LSTD instead. The key idea is only to sketch the constraints of the system (the left-side of $\mathbf{A}$) rather than the variables (the right-side of $\mathbf{A}$). Sketching features, on the other hand, by design, sketches both constraints, and variables. We show that even with a straightforward linear system solution, the left-sided sketch can significantly reduce bias. We further show how to use this left-sided sketch within a quasi-Newton algorithm, providing an unbiased policy evaluation algorithm that can still benefit from the computational improvements of sketching.

The key novelty is designing such system-sketching algorithms when also incrementally computing the linear system solution. There is a wealth of literature on sketching linear systems to reduce computation. In general, however, many sketching approaches cannot be applied to the incremental policy evaluation problem because the approaches are designed for a static linear system. For example, Gower & Richtárik (2015) provide a host of possible solutions for solving large linear systems. However, they assume access to $\mathbf{A}$ upfront, so in memory and computation, the algorithm design is not suitable for the incremental setting. Some popular sketching approaches, such as Frequent Directions (Ghashami et al., 2014), has been successfully used for the online setting, for quasi-Newton algorithms (Luo et al., 2016); however, they sketch

symmetric matrices that are growing with the number of samples.

The most straightforward way to use $\mathbf{SA}$ is to incrementally compute $\mathbf{SA}$, and periodically solve $\mathbf{w} = (\mathbf{SA})^\dagger \mathbf{Sb}$. This costs $O(dk)$ per step, and $O(d^2k)$ every time the solution is recomputed. To maintain $O(dk)$ computation per-step, this full solution could only be computed every d steps, which is too infrequent to provide a practical incremental policy evaluation approach. Further, because it is an underconstrained system, there are likely to be infinitely many solutions to $\mathbf{SAw} = \mathbf{Sb}$; amongst those solutions, we would like to sub-select amongst the unbiased solutions to $\mathbf{Aw} = \mathbf{b}$.

We first discuss how to efficiently maintain $(\mathbf{SA})^\dagger$, and then describe how to use that matrix to obtain an unbiased algorithm. Let $\tilde{\mathbf{A}} \stackrel{\text{def}}{=} \mathbf{SA} \in \mathbb{R}^{k \times d}$. For this underconstrained system with $\tilde{\mathbf{b}} \stackrel{\text{def}}{=} \mathbf{Sb}$, the minimum norm solution to $\tilde{\mathbf{A}}\mathbf{w} = \tilde{\mathbf{b}}$ is $\mathbf{w} = \tilde{\mathbf{A}}^\top (\tilde{\mathbf{A}}\tilde{\mathbf{A}}^\top)^{-1}\tilde{\mathbf{b}}$ and $\tilde{\mathbf{A}}^\dagger = \tilde{\mathbf{A}}^\top (\tilde{\mathbf{A}}\tilde{\mathbf{A}}^\top)^{-1} \in \mathbb{R}^{d \times k}$. To maintain $\tilde{\mathbf{A}}_t^\dagger$ incrementally, therefore, we simply need to maintain $\tilde{\mathbf{A}}_t$ incrementally and the $k \times k$-matrix $(\tilde{\mathbf{A}}_t \tilde{\mathbf{A}}_t^\top)^{-1}$ incrementally.

Let $\tilde{\mathbf{e}}_t \stackrel{\text{def}}{=} \mathbf{S}\mathbf{e}_t$, $\mathbf{d}_t \stackrel{\text{def}}{=} \mathbf{d}_t$ and $\mathbf{h}_t \stackrel{\text{def}}{=} \tilde{\mathbf{A}}_t \mathbf{d}_t$. We can update the sketched system in $O(dk)$ time and space

$$\begin{aligned}
\tilde{\mathbf{A}}_{i+1} &= \tilde{\mathbf{A}}_i + \tfrac{1}{i+1}\left(\tilde{\mathbf{e}}_i \mathbf{d}_i^\top - \tilde{\mathbf{A}}_i\right) \\
\tilde{\mathbf{b}}_{i+1} &= \tilde{\mathbf{b}}_i + \tfrac{1}{i+1}\left(\tilde{\mathbf{e}}_i R_{i+1} - \tilde{\mathbf{b}}_i\right)
\end{aligned}$$

To maintain $(\tilde{\mathbf{A}}_t \tilde{\mathbf{A}}_t^\top)^{-1}$ incrementally, notice that the unnormalized update is

$$\begin{aligned}
\tilde{\mathbf{A}}_{t+1}\tilde{\mathbf{A}}_{t+1}^\top &= (\tilde{\mathbf{A}}_t + \tilde{\mathbf{e}}_t \mathbf{d}_t^\top)(\tilde{\mathbf{A}}_t + \tilde{\mathbf{e}}_t \mathbf{d}_t^\top) \\
&= \tilde{\mathbf{A}}_t \tilde{\mathbf{A}}_t^\top + \tilde{\mathbf{e}}_t \mathbf{h}_t^\top + \mathbf{h}_t \tilde{\mathbf{e}}_t^\top + ||\mathbf{d}_t||_2^2 \|\tilde{\mathbf{e}}_t \tilde{\mathbf{e}}_t^\top.
\end{aligned}$$

Hence, $(\tilde{\mathbf{A}}_{t+1}\tilde{\mathbf{A}}_{t+1}^\top)^{-1}$ can be updated from $(\tilde{\mathbf{A}}_t \tilde{\mathbf{A}}_t^\top)^{-1}$ by applying the Sherman-Morrison update three times. For a normalized update, based on samples, the update is

$$\begin{aligned}
\tilde{\mathbf{A}}_{t+1}\tilde{\mathbf{A}}_{t+1}^\top = \left(\tfrac{t}{t+1}\right)^2 \tilde{\mathbf{A}}_t \tilde{\mathbf{A}}_t^\top + \tfrac{t}{(t+1)^2}\left(\tilde{\mathbf{e}}_t \mathbf{h}_t^\top + \mathbf{h}_t \tilde{\mathbf{e}}_t^\top\right) \\
+ \tfrac{1}{(t+1)^2} ||\mathbf{d}_t||_2^2 \|\tilde{\mathbf{e}}_t \tilde{\mathbf{e}}_t^\top
\end{aligned}$$

We can then compute $\mathbf{w}_t = \tilde{\mathbf{A}}_t (\tilde{\mathbf{A}}_t \tilde{\mathbf{A}}_t^\top)^\dagger \tilde{\mathbf{b}}_t$ on each step.

This solution, however, provides the minimum norm solution rather than the unbiased solution, even though the unbiased solution is feasible for the underconstrained system. To achieve this unbiased solution, we use our stochastic approximation algorithm—ATD. This method is a quasi-second order method, that relies on a low-rank approximation $\hat{\mathbf{A}}_t$ to $\mathbf{A}_t$; using this approximation, the update is $\mathbf{w}_{t+1} = \mathbf{w}_t + (\alpha_t \hat{\mathbf{A}}_t^\dagger + \eta \mathbf{I})\delta_t \mathbf{e}_t$. Instead of being used to explicitly solve for $\mathbf{w}$, the approximation matrix is used to provide curvature information. The inclusion of η constitutes a small regularization component, that pushes the solution towards the unbiased solution.

5.4 Empirical Results

This section focuses on studying three problems: 1) how does ATD with sketching work as we increase rank, and how sensitive is hyper-parameters? 2) does ATD with sketching worth on high-dimensional domain in terms of computational cost? 3) how do feature properties affect the efficacy of sketching techniques?

5.4.1 Overall Performance of ATD with Sketching

We now conduct experiments to address the first two questions. We set $k = 50$, unless otherwise specified, to average all results over 50 runs and sweep the same number of hyper-parameters for each algorithm whenever applicable. Detailed experimental settings, such as parameter ranges, are in Appendix A.2.3. To distinguish projections, we add -P for the two-sided projection (e.g., ATD-P, LSTD-P) and -L for the left-sided projection (e.g., ATD-L, LSTD-L) to the algorithm name. Note that since we already show that sketching seems to be more effective with RBF features, we mainly focus on investigating the performance with RBF features in this section. Additional results with tile coding features are presented in the Appendix A.2.4.

Performance and parameter sensitivity for RBFs. We first compare the algorithms in Puddle World, in Figures 5.2 and 5.3. We provide additional

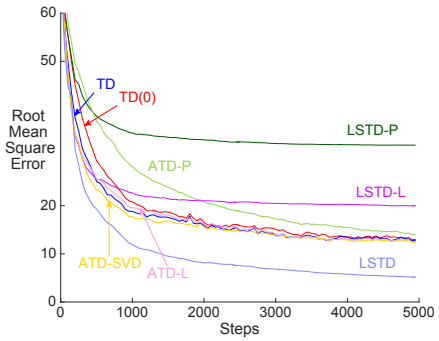
results in Mountain Car in Figure A.2 in the Appendix A.2.4.

We summarize our important observations as follows: 1) ATD-L's performance sits between TD and LSTD method: it achieves closer sample efficiency to LSTD than TD, but it incurs lower computational cost than LSTD; 2) ATD with sketching decreases bias relative to its LSTD variant; 3) ATD with left-sided sketching typically performs as well as ATD-SVD, but is significantly faster; 4) two-sided projection—projecting the features—generally does much worse than only projecting the left-side of $\mathbf{A}$; 5) the LSTD algorithms with random projection are less sensitive to initialization than the vanilla LSTD. We hypothesize that the reason for the insensitivity is that with matrix sketching, LSTD only has to initialize a smaller $k \times k$ symmetric matrix, $(\mathbf{SA}(\mathbf{SA})^\top)^{-1} = \eta\mathbf{I}$, and so is much more robust to this initialization. In fact, across settings, we found initializing to $\mathbf{I}$ was effective. Similarly, ATD-L benefits from this robustness since it needs to initialize the same matrix and then further overcomes bias using the approximation to $\mathbf{A}$ only for curvature information.

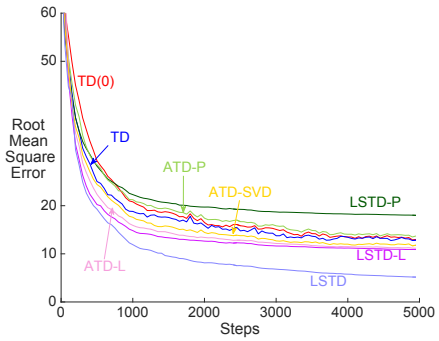
Experiments on high dimensional domains. We apply our sketching techniques on a higher dimensional domain—Acrobot with more than 10k basis functions—to illustrate practical usability. The Acrobot domain (Sutton & Barto, 2018) is a four-dimensional episodic task, where the goal is to raise an arm to a certain height, and an episode ends when the height level is reached. We used $14,400$ uniformly-spaced centers within the state space, resulting in $14,400$ dimensional features. We summarize the results in the caption of Figure 5.4, with the overall conclusion that ATD-L provides an attractive way to reduce parameter sensitivity of TD and benefit from random projection to reduce computation.

5.4.2 The Impact of Feature Properties

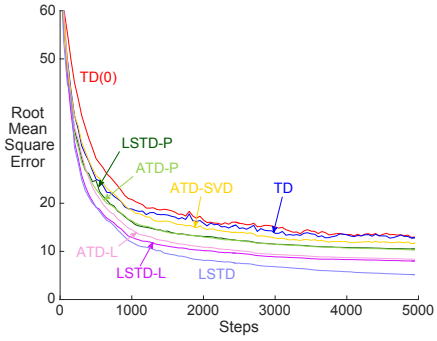
To investigate the properties of these sketching approaches, we need to understand when we expect sketching to have the most benefit. Despite the wealth of literature on sketching and solid theoretical results, there seem to be fewer



(a) RBF, $k = 25$



(b) RBF, $k = 50$

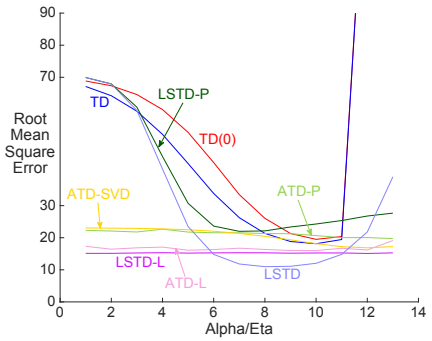


(c) RBF, $k = 75$

Figure 5.2: Change in performance when increasing k, from 25 to 75 on Puddle World. Two-sided projection (i.e., projecting the features) significantly improves with larger k but is strictly dominated by left-side projection. At $k = 50$, the left-side projection methods are outperforming TD and are less variant. ATD-SVD seems to gain less with increasing k, though we generally found ATD-SVD to perform more poorly than ATD-P, particularly for RBF representations.

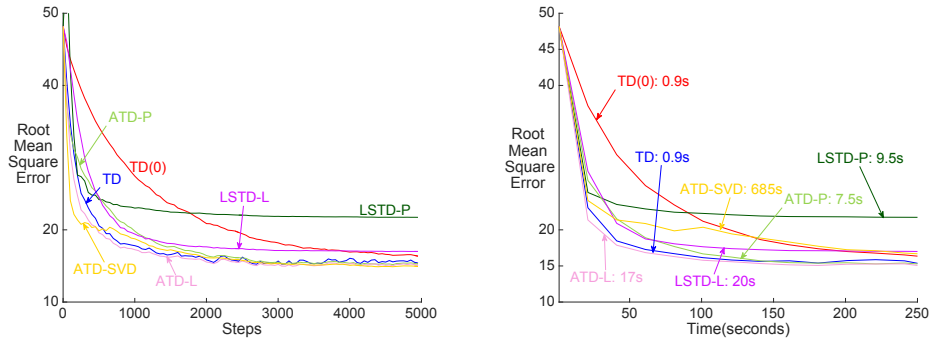
empirical investigations into when sketching brings in benefit. This section elucidates some hypotheses about when sketching should be most effective, which we then explore in our experiments.

Recall Figure 5.1 in Section 5.1; it was clear that sketching the RBF features was much more effective than sketching the tile coding features. Therefore, a natural investigation is into the properties of representations that are more amenable to sketching (as we show in the next section). The key differences between these two representations are in smoothness, density, and overlap. The tile coding representation has non-smooth $0, 1$ features, which



(a) Puddle World, RBF, $k = 50$

Figure 5.3: Sensitivity figure on Puddle World domain using projected dimension $k = 50$. This corresponds to the learning curve from Figure 5.2(b). Note that we sweep initialization for LSTD-P but keep the initialization parameter fixed across all other settings. The one-side projection is almost insensitive to initialization, and the corresponding ATD version is insensitive to regularization. Though ATD-SVD also shows insensitivity, the performance of ATD-SVD is much worse than sketching methods for the RBF representation.



(a) Acrobot, RBF

(b) Acrobot, RMSE vs Time

Figure 5.4: Results in domains with high-dimensional features, using $k = 50$ and with results averaged over 30 runs. For Acrobot, the (left-side) sketching methods perform well and are much less sensitive to parameters than TD. We show RMSE versus time for runtime comparison, allowing the algorithms to process up to 25 samples per second to simulate a real-time setting learning; slow algorithms cannot process all 25 within a second. With computation taken into account, ATD-L has a significant win over ATD-SVD, and does not lose relative to TD. Total runtime in seconds for one run for each algorithm is labeled in the plot. ATD-SVD is much slower because of the incremental SVD.

do not overlap in each grid. Instead, the overlap for tile coding results from overlapping tilings. This differs from RBF overlap, where centers are arranged in a grid, and only edges of the RBF features overlap. Theoretical work in sketching for regression (Maillard & Munos, 2012), however, does not require features to be smooth.

Some theoretical results suggest sketching could be more amenable for more distinct features—less overlap or potentially fewer tilings. Balcan et al. (2006) shows a worst-case setting where data-independent sketching results in poor performance. They propose a two-stage projection to maintain separability in classification. The first stage uses a data-dependent projection to ensure features are not highly correlated. The second uses a data-independent projection (a sketch) to further reduce the dimension after the orthogonal projection. The implied conclusion from this result is that if the features are not highly correlated, then the first step can be avoided, and the data-independent sketch should similarly maintain classification accuracy. This result suggests that sketching for feature expansions with less redundancy should perform better.

We might also expect sketching to be more robust to the condition number of the matrix. For sketching in regression, Fard et al. (2012) found a bias-variance trade-off when increasing k, where for large k, estimation error from a larger number of parameters became a factor. The smallest eigenvalue of the sketched matrix should be larger than that of the original matrix; this improvement in condition number compensates for the loss in information. Similarly, we might expect that maintaining an incremental singular value decomposition for ATD could be less robust than ATD with left-side sketching.

We explore how the feature properties—smoothness, density, overlap and redundancy—can change the performance of sketching in linear value function approximation, as shown in Figure 5.5. There are following important observations and conclusions.

First, the smooth feature seems more suitable for sketching as the algorithms with RBF feature are better in most settings than tile coding and spline features, which are not smooth. Second, increasing density—which can be done by increasing overlap or tilings (i.e., redundancy)—should also benefit

sketching methods. This can be seen by observing: 1) tile coding with increasing tilings (i.e., nonzero entries) tends to perform better; 2) RBF with more tilings tends to perform better; 3) spline features with more overlap tends to perform better. Note that Figure 5.5 (a)(d) does not weaken this conclusion, though the RMSE corresponding to several rightmost points seems to go up. It is likely due to more inferior feature quality because those algorithms without using random projection also perform worse. Third, random projection seems to be less sensitive to feature quality. Since LSTD can be thought of as a closed-form solution, its performance can be an indicator of the feature quality. One can see that random projection methods are less negatively impacted than TD and ATD-SVD methods when the feature quality becomes worse across figures.

5.5 Convergence of ATD with Sketching

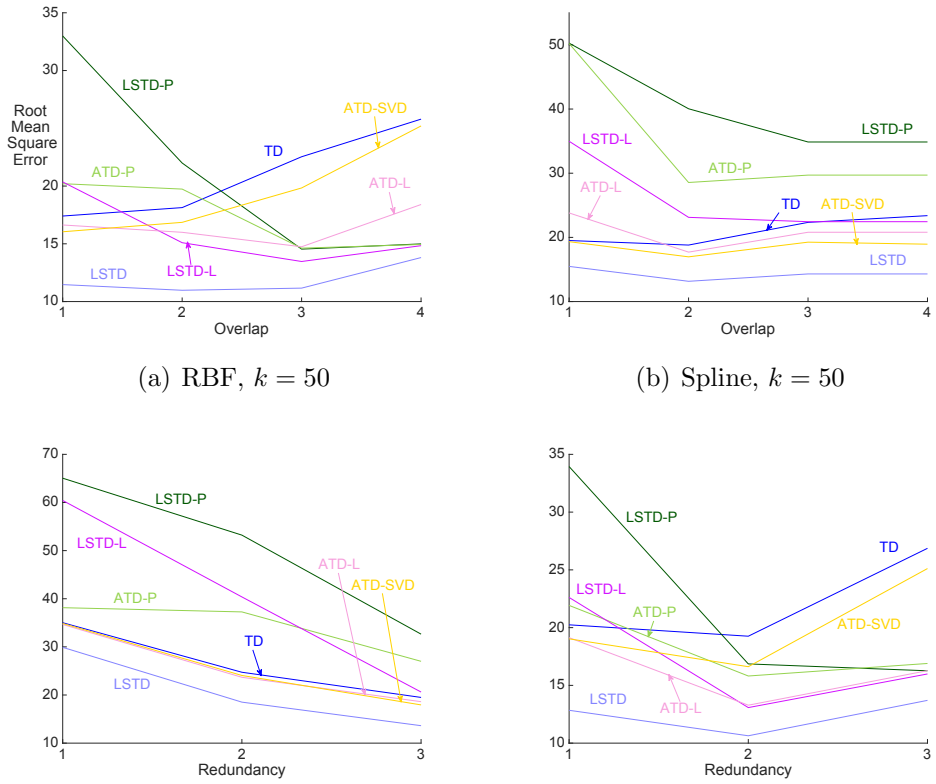
We now show that for our alternative approximation, we still obtain unbiased solutions. We use results for iterative methods for singular linear systems (Shi et al., 2011; Wang & Bertsekas, 2013), since $\mathbf{A}$ may be singular. $\mathbf{A}$ has been shown to be positive semi-definite under standard assumptions on the MDP (Yu, 2015). Note that, again, we assume $\mathbf{A}$ is positive semi-definite, instead of providing these MDP assumptions. This holds under mild conditions according to Yu (2015)

Assumption 4. *For* $\mathbf{S} \in \mathbb{R}^{k\times d}$ *and* $\mathbf{B} = \alpha(\mathbf{SA})^\dagger\mathbf{S} + \eta\mathbf{I}$ *with* $\mathbf{B} \in \mathbb{R}^{d\times d}$*, the matrix* $\mathbf{BA}$ *is diagonalizable.*

Assumption 5. $\mathbf{A}$ *is positive semi-definite.*

Assumption 6. $\alpha \in (0, \frac{1}{2})$ *and* $0 < \eta \leq \frac{1}{2\lambda_{max}(\mathbf{A})}$ *where* $\lambda_{max}(\mathbf{A})$ *is the maximum eigenvalue of* $\mathbf{A}$*.*

Theorem 3. *Under Assumptions 4-6, the expected updating rule* $\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbb{E}_\pi[\mathbf{B}\delta_t\mathbf{e}_t]$ *converges to a fixed-point* $\mathbf{w}^\star = \mathbf{A}^\dagger\mathbf{b}$*.*



(a) RBF, $k = 50$

(b) Spline, $k = 50$

(c) Tile coding, $k = 50$

(d) RBFs with tilings, $k = 50$

Figure 5.5: The effect of varying the representation properties in Puddle World with $d = 1024$. In **(a)** and **(b)**, we examine the impact of varying the overlap, for both smooth features (RBFs) and 0-1 features (Spline). For spline, the feature is 1 if $||\mathbf{x} - \mathbf{c}_i|| < \sigma$ and otherwise 0. The spline feature represents a bin, like for tile coding, but here we adjust the widths of the bins so that they can overlap and do not use tilings. The x-axis has four width values to give a corresponding feature vector norm of about $20, 40, 80, 120$. In **(c)** and **(d)**, we vary the redundancy, where the number of tilings is increased, and the total number of features is kept constant. We generate tilings for RBFs like tile coding, but each grid cell uses an RBF similarity rather than a spline similarity. We used $4 \times 16 \times 16, 16 \times 8 \times 8$ and $64 \times 4 \times 4$.

Proof. The expected updating rule is $\mathbb{E}_\pi[\mathbf{B}\delta_t\mathbf{e}_t] = \mathbf{B}(\mathbf{b} - \mathbf{A}\mathbf{w}_t)$. As in the proof of convergence for Theorem 2 in with SVD (Section 4.4), we similarly verify the conditions from Theorem 1.1 in Shi et al. (2011).

Notice first that $\mathbf{BA} = \alpha(\mathbf{SA})^\dagger\mathbf{SA} + \eta\mathbf{A}$.

For singular value decomposition, $\mathbf{SA} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, we have that $(\mathbf{SA})^\dagger\mathbf{SA} = \mathbf{V}\mathbf{\Sigma}^\dagger\mathbf{U}^\top\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \mathbf{V}[\mathbf{I}_{\tilde{k}}\ \mathbf{0}_{d-k}]\mathbf{V}^\top$, where $\tilde{k} \leq k$ is the rank of $\mathbf{SA}$. The maximum eigenvalue of $(\mathbf{SA})^\dagger\mathbf{SA}$ is therefore 1.

Because $(\mathbf{SA})^\dagger\mathbf{SA}$ and $\mathbf{A}$ are both positive semidefinite, $\mathbf{BA}$ is positive semi-definite. By Weyl's inequalities,

$$\lambda_{\max}(\mathbf{BA}) \leq \alpha\lambda_{\max}((\mathbf{SA})^\dagger\mathbf{SA}) + \eta\lambda_{\max}(\mathbf{A}).$$

Therefore, the eigenvalues of $\mathbf{I} - \mathbf{BA}$ have absolute value strictly less than 1, because $\eta \leq (2\lambda_{\max}(\mathbf{A}))^{-1}$ and $\alpha < 1/2 = (2\lambda_{\max}((\mathbf{SA})^\dagger\mathbf{SA}))^{-1}$ by assumption.

For the second condition, since $\mathbf{BA}$ is PSD and diagonalizable, we can write $\mathbf{BA} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}$ for some matrices $\mathbf{Q}$ and diagonal matrix $\mathbf{\Lambda}$ with eigenvalues greater than or equal to zero. Then $(\mathbf{BA})^2 = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1} = \mathbf{Q}\mathbf{\Lambda}^2\mathbf{Q}^{-1}$ has the same rank.

For the third condition, because $\mathbf{BA}$ is the sum of two positive semi-definite matrices, the nullspace of $\mathbf{BA}$ is a subset of the nullspace of each of those matrices individually:

$$\text{nullspace}(\mathbf{BA}) = \text{nullspace}(\alpha(\mathbf{SA})^\dagger\mathbf{SA} + \eta\mathbf{A}) \subseteq \text{nullspace}(\eta\mathbf{A}) = \text{nullspace}(\mathbf{A}).$$

In the other direction, for all $\mathbf{w}$ such that $\mathbf{A}\mathbf{w} = \mathbf{0}$, its clear that $\mathbf{BA}\mathbf{w} = \mathbf{0}$, and so

$$\text{nullspace}(\mathbf{A}) \subseteq \text{nullspace}(\mathbf{BA}).$$

Hence, $\text{nullspace}(\mathbf{A}) = \text{nullspace}(\mathbf{BA})$. □

5.6 Discussions about Sketching

Sketching has been used for quasi-Newton updates in online learning; a natural question is if those methods are applicable for policy evaluation. Luo et al.

(2016) consider sketching approaches for an online Newton-update, for general functions rather than just the linear function approximation case we consider here. They similarly have to consider updates amenable to incrementally approximating a matrix (a Hessian in their case). In general, however, porting these quasi-Newton updates to policy evaluation for reinforcement learning is problematic for two reasons. First, the objective function for temporal difference learning is the MSPBE, which is the product of three expectations. It is not straightforward to obtain an unbiased sample of this gradient. Consequently, it is not straightforward to apply quasi-Newton online algorithms that assume access to unbiased gradients. Second, the Hessian can be nicely approximated in terms of gradients, and is symmetric; both are exploited when deriving the sketched online Newton-update (Luo et al., 2016). We, on the other hand, have an asymmetric matrix $\mathbf{A}$.

In the other direction, we could consider if our approach could be beneficial for the online regression setting. For linear regression, with $\gamma = 0$, the matrix $\mathbf{A}$ actually corresponds to the Hessian. In contrast to previous approaches that sketched the features (Maillard & Munos, 2012; Fard et al., 2012; Luo et al., 2016), therefore, one could instead sketch the system and maintain $(\mathbf{SA})^\dagger$. Since the second-order update is $\mathbf{A}^{-1}\mathbf{g}_t$ for gradient $\mathbf{g}_t$ on iteration t, an approximate second-order update could be computed as $((\mathbf{SA})^\dagger\mathbf{S} + \eta\mathbf{I})\mathbf{g}_t$.

In our experiments, we found sketching both sides of $\mathbf{A}$ to be less effective and found little benefit from modifying the chosen sketch; however, these empirical conclusions warrant further investigation. With more understanding into the properties of $\mathbf{A}$, it could be possible to benefit from this variety. For example, sketching the left-side of $\mathbf{A}$ could be seen as sketching the eligibility trace, and the right-side as sketching the difference between successive features. For some settings, there could be properties of either of these vectors that are particularly suited to a certain sketch. As another example, the key benefit of many of the sketches over Gaussian random projections is in enabling the dimension k to be larger, by using (sparse) sketching matrices where dot product are efficient. We could not easily benefit from these properties, because $\mathbf{SA}$ could be dense and computing matrix-vector products and incre-

mental inverses would be expensive for larger k. For sparse $\mathbf{A}$, or when $\mathbf{SA}$ has specialized properties, it could be more possible to benefit from different sketches.

Finally, the idea of sketching fits well into a larger theme of random representations within reinforcement learning. A seminal paper on random representations (Sutton & Whitehead, 1993) demonstrates the utility of random threshold units, as opposed to more carefully learned units. Though end-to-end training has become more popular in recent years, there is evidence that random representations can be quite powerful (Aubry & Jaffard, 2002; Rahimi & Recht, 2007, 2008; Maillard & Munos, 2012), or even combined with descent strategies (Mahmood & Sutton, 2013). For reinforcement learning, this learning paradigm is particularly suitable, because data cannot be observed upfront. Data-independent representations, such as random representations and sketching approaches, are therefore particularly appealing and warrant further investigation for the incremental and online reinforcement learning setting.

Chapter 6

Gradient-based Search-control in Dyna

Stochastic gradient optimization algorithms typically sample a mini-batch of training examples to approximate the expected gradient. Sampling distribution of the training examples significantly impacts the sample efficiency of these stochastic optimization algorithms and this problem has been actively explored in a conventional machine learning setting (Needell et al., 2014; Zhao & Zhang, 2015; Wang et al., 2017). We study what type of sampling distribution one should use to accelerate reinforcement learning algorithms in this chapter. We consider the corresponding concept of sampling distribution in an RL context as *search-control*—the mechanism deciding what kind of imagined experiences to use by generating state or state-action pairs from which we query a model to get the next state and reward. It is a crucial component in Dyna (Sutton, 1991), which is a classic model-based reinforcement learning (MBRL) architecture.

Our key idea for sampling distribution design is to bring in the Stochastic Gradient Langevin Dynamics (SGLD) method, which has been broadly used in the conventional machine learning, such as large-scale Bayesian learning (Welling & Teh, 2011; Sato & Nakagawa, 2014), global convergence analysis in optimization (Xu et al., 2018), generative modeling (Song & Ermon, 2019, 2020) or energy-based models (Du & Mordatch, 2019), etc. Note that, for a multivariate random variable (i.e., a state $s \in \mathcal{S}$), specifying a probability distribution and sampling from that specified distribution may be quite

challenging tasks. SGLD enables to do both easily. SGLD sampling then simplifies the sampling distribution design problem to the problem of finding some score/energy function that measures the importance of states. The name of our method, *gradient-based search-control*, comes from the fact that SGLD sampling relies on a modified gradient updating rule, as we detail in this chapter.

We organize this chapter as follows. We firstly review basic background in search-control within the Dyna MBRL architecture in Section 6.1, followed by a detailed discussion about problem formulation, SGLD method, and how it is used in Dyna in Section 6.2. Then we present three search-control methods originating from RL and regular supervised learning perspectives in Section 6.3, 6.4, and 6.5. Specifically, we propose to search states 1) that have high values (Pan et al., 2019); 2) whose values are considered as difficult to estimate (Pan et al., 2020b); or 3) whose absolute TD errors are large (Mei et al., 2020). By leveraging the generalization power of the learned value function, we search those states of interest via gradient ascent. We provide theoretical insight and empirical evidence to show the efficacy of our proposed search-control mechanisms. Note that this chapter's contents are mainly from the papers by Pan et al. (2019, 2020b) and Mei et al. (2020).

6.1 Background in Dyna and Search-control

This section briefly introduces control problems in RL and then we focus on reviewing the key concepts about Dyna and search-control.

Control problems. Recall that Chapter 2 discusses the basic building framework of RL: Markov decision processes and then describes the *policy evaluation* problem, which does not directly concern searching a high-performing policy; rather, it focuses on evaluating a policy. Another category of research problems is called *control*, where we aim at finding a policy that can maximize certain performance measure. A commonly used measure is the expected return starting from some state (Sutton & Barto, 2018). That is, we would like

to solve the optimization problem $\max_\pi v^\pi(s_0)$ for $s_0 \in \mathcal{S}$. This objective can be easily extended to maximizing the sum of weighted returns over a set of states (Degris et al., 2012b; Imani et al., 2018; Maei, 2018; Zhang et al., 2019).

The policy is either directly parameterized and learned, as in policy gradient methods (Williams, 1992; Sutton & Barto, 2018), or the action-values are learned and the policy inferred by acting greedily with respect to the action-values, as in Q-learning (Watkins & Dayan, 1992). Methods interpolating between value-based and policy-based methods—typically those actor-critic algorithms (Degris et al., 2012a; Lillicrap et al., 2016; Schulman et al., 2015, 2017; Haarnoja et al., 2018)—are quite popular due to their superior empirical performances.

In either setting, it is common to parameterize the policy/value function by a neural network (NN). For example, Deep Q Networks (DQN) (Riedmiller, 2005; Mnih et al., 2015) parameterize the action-value function $Q_\theta : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ by a NN. The bootstrap target for updating a state-action value is computed by using a separate target NN: $Q_{\theta^-} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ parameterized by θ^-: $y_t = r_{t+1} + \gamma \max_{a'} Q_{\theta^-}(s_{t+1}, a')$. The target NN parameter θ^- is updated by copying from θ every certain number of time steps. Online RL control problems with nonlinear function approximation can be highly unstable (Dai et al., 2018; Liu et al., 2019), and the target network technique can be used to mitigate this issue. It effectively stabilizes deep RL algorithms according to various empirical (Mnih et al., 2015; Lillicrap et al., 2016; Liu et al., 2019) and theoretical evidence (Fan et al., 2020; Zhang et al., 2021). In this thesis, we focus on a generic MBRL framework rather than designing a specific control algorithm. Hence we do not provide additional discussions about different control algorithms. We mainly use DQN as the basic algorithm in our framework for empirical demonstrations.

In fact, the control problems have been studied a long time ago outside of the context of modern computational RL. Attempts have been made to unify control algorithms for sequential decision problems studied within different contexts. We refer readers to the work by Powell (2021, Chapter 1,2) and Powell (2007) for alternative formulation for control problems and specific

categorization of control algorithms.

Model-based Reinforcement Learning (MBRL) and Dyna. MBRL for control are critical to obtain sample efficient learning (Sutton, 1991; Daw, 2012; Sutton & Barto, 2018). They have been successfully applied to many benchmark domains (Gu et al., 2016; Ha, David and Schmidhuber, Jürgen, 2018; Kaiser et al., 2020), and some well-known challenging games such as Atari, Go, chess and shogi (Schrittwieser et al., 2020). The Dyna architecture, introduced by Sutton (1991), is one of the classical MBRL architectures, which integrates model-free and model-based policy updates in an online RL setting (Algorithm 6). At each time step, a Dyna agent uses the real experience to learn a model and to perform model-free policy update, and during the ***planning*** stage, simulated experiences are acquired from the model to further improve the policy. We would like to clarify that, the term *planning* refers to any computational process that takes a model as input and produces or improves a policy for interacting with the modeled environment, as described in Chapter 8 in the book by Sutton & Barto (2018).

A closely related method to Dyna in model-free learning setting is experience replay (ER) (Lin, 1992; Adam et al., 2012), which utilizes a buffer to store experiences. An agent using the ER buffer randomly samples the recorded experiences at each time step to update the policy. Though ER can be thought of as a simplified form of MBRL (van Seijen & Sutton, 2015), a model provides more flexibility in acquiring simulated experiences.

Search-control. A crucial aspect of Dyna is the ***search-control*** mechanism. It is the mechanism for selecting states or state-action pairs to query the model in order to generate simulated experiences (Sutton & Barto (2018, Section 8.2)). We call the corresponding data structure for storing those states or state-action pairs the *search-control queue*. Search-control is of vital importance in Dyna, as it can significantly affect the model-based agent's sample efficiency. The search-control strategy in the vanilla Dyna Algorithm 6 is to sample visited states or state-action pairs, i.e., use the initial state-action pairs

Algorithm 6 Generic Dyna Architecture: Tabular Setting

Initialize $Q(s,a)$ and model $\mathcal{M}(s,a)$, $\forall(s,a) \in \mathcal{S} \times \mathcal{A}$
while true **do**
  observe s, take action a by ϵ-greedy w.r.t action values
  execute a, observe reward R and next state s'
  Q-learning update for $Q(s,a)$
  update model $\mathcal{M}(s,a)$ (i.e. by counting)
  store (s,a) into search-control queue
  for i=1:d **do**
    sample $(\tilde{s},\tilde{a})$ from search-control queue
    $(\tilde{s}',\tilde{R}) \leftarrow \mathcal{M}(\tilde{s},\tilde{a})$ // simulated transition
    Q-learning update for $Q(\tilde{s},\tilde{a})$ // planning update

stored in the ER buffer as the search-control queue. This approach, however, does not lead to an agent that outperforms a model-free agent that uses ER. To see this, consider a deterministic environment, and assume that we have the exact model. If we simply sample visited state-action pairs for search-control, the next-state and reward would be the same as those in the ER buffer. In practice, we have model errors too, which causes some performance deterioration (Talvitie, 2014, 2017). Without an elegant search-control mechanism, we are not likely to benefit from the flexibility given by a model.

Several search-control mechanisms have already been explored. Prioritized sweeping (Moore & Atkeson, 1993) is one such method that is designed to speed up the value iteration process: the simulated transitions are updated based on the absolute temporal difference error. It has been adopted to continuous domains with function approximation too (Sutton et al., 2008; Pan et al., 2018; Corneil et al., 2018; Wan et al., 2019). Gu et al. (2016) utilizes local linear models to generate optimal trajectories/experiences through iLQR (Li & Todorov, 2004) and use those experiences for Dyna-style planning.

Observing that Dyna's search-control mechanisms are not widely explored on continuous state domains, in this chapter, we would like to pursue a probabilistic perspective for search-control mechanism design in continuous state domains. We attempt to answer the question:

what type of state sampling distribution for search-control can help improve sample efficiency during the planning stage?

Given this topic question, we would like to emphasize that this chapter focuses on improving sample efficiency (by designing sampling distributions) rather than saving computation or memory costs.

Note that, even if we have a heuristic to assign importance to different states, it is typically challenging to specify the corresponding probability distribution and sample from it. In the next section, one will see that the SGLD sampling only requires to specify some score function $h(\cdot)$ to do both.

6.2 Stochastic Gradient Langevin Dynamics for Search-control

This section firstly reviews relevant knowledge background in the Langevin dynamics, followed by introducing the SGLD sampling method. With the SGLD sampling method, designing a sampling distribution is reduced to the design of the score function $h(\cdot)$. We then describe using the SGLD sampling method to sample states to fill the search-control queue in Dyna and present the corresponding Hill-climbing(HC) Dyna architecture.

Langevin dynamics background. Langevin dynamics is used as a tool to analyze optimization algorithms (Xu et al., 2018) or to acquire an estimate of the expected parameter values w.r.t. some posterior distribution in Bayesian learning (Welling & Teh, 2011; Sato & Nakagawa, 2014). The overdamped Langevin dynamics can be described by a stochastic differential equation (SDE)

$$\mathrm{d}W(t) = \nabla U(W_t)\mathrm{d}t + \sqrt{2}\mathrm{d}B_t,$$

where $B_t \in \mathbb{R}^d$ is a d-dimensional Brownian motion and U is a continuous differentiable function. Under some conditions, it turns out that the Langevin diffusion $(W_t)_{t\geq 0}$ converges to a unique invariant distribution $p(x) \propto \exp(U(x))$ (Chiang et al., 1987).

By applying the Euler-Maruyama discretization scheme to the SDE, we acquire the discretized version

$$Y_{k+1} = Y_k + \alpha_{k+1}\nabla U(Y_k) + \sqrt{2\alpha_{k+1}}Z_{k+1},$$

where $(Z_k)_{k\geq 1}$ is an i.i.d. sequence of standard d-dimensional Gaussian random vectors and $(\alpha_k)_{k\geq 1}$ is a sequence of step sizes. This discretization scheme was used to acquire samples from the original invariant distribution $p(x) \propto \exp(U(x))$ through the Markov chain $(Y_k)_{k\geq 1}$ when it converges to the chain's stationary distribution (Roberts, 1996). The distance between the limiting distribution of $(Y_k)_{k\geq 1}$ and the invariant distribution of the underlying SDE (i.e., the convergence mode) has been characterized through various bounds depending on different technical conditions (Roberts, 1996; Teh et al., 2016; Durmus & Moulines, 2017).

Therefore, one can flexibly design sampling distribution by choosing the function U in the SDE above. We now use the notations from RL setting to clarify how the sampling can be done.

SGLD sampling method. Let $h(\cdot) : \mathcal{S} \mapsto \mathbb{R}$ be some differentiable function w.r.t. the input $s \in \mathcal{S}$. Given some initial state $s_0 \in \mathcal{S}$, let the state sequence $\{s_i\}$ be the one generated by updating rule

$$s_{i+1} \leftarrow s_i + \alpha_h \nabla_s h(s_i) + \sqrt{2\alpha_h/\xi} X_i,$$

where α_h is a sufficiently small stepsize, ξ is the temperature parameter and $X_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is a Gaussian random variable. Then the sequence $\{s_i\}$ asymptotically converges to the distribution $p(s) \propto exp(\xi h(s))$ as $i \to \infty$. In implementation, one may treat the Gaussian variance as a hyper-parameter and ignore the temperature ξ, as the latter can be adjusted by adjusting the variance according to practical performance.

With this sampling method, designing a state sampling distribution and sampling from the designed distribution is simplified to design the score function $h(\cdot)$. This chapter will present several instances of $h(\cdot)$ motivated by either theoretical or empirical evidence.

Dyna with Gradient-based Search-control. We now describe how the SGLD sampling method is applied in the Dyna architecture. We introduce our generic Dyna architecture with SGLD sampling Algorithm 7, which we call Hill Climbing (HC)-Dyna.

Algorithm 7 HC-Dyna: Generic framework

Input: Hill Climbing (HC) criterion function $h : \mathcal{S} \mapsto \mathbb{R}$; Initialize empty search-control queue B_s; empty ER buffer B_{er}; initialize policy; HC stepsize α_h and gradient noise standard deviation σ; mini-batch size b; environment $\mathcal{P}$; mixing rate ρ decides the proportion of imagined experiences in a mini-batch.

for $t = 1, 2, \ldots$ **do**
  Add $(s_t, a_t, s_{t+1}, r_{t+1})$ to B_{er}
  while within some budget time steps **do**
    $s \leftarrow s + \alpha_h \nabla_s h(s) + X$, $X \sim \mathcal{N}(\mathbf{0}, \sigma\mathbf{I})$ // HC for search-control
    Add s into B_s
  // n planning updates/steps
  for n times **do**
    $B \leftarrow \emptyset$ // initialize an empty mini-batch B
    for $b\rho$ times **do**
      Sample $s \sim B_s$, on-policy action a
      Sample $s', r \sim \mathcal{P}(s, a)$
      $B \leftarrow (s, a, s', r)$
    Sample $b(1-\rho)$ experiences from B_{er}, add to B
    Update policy/value on mixed mini-batch B

We consider a one-step model which maps a state-action pair to its possible next state and reward: $\mathcal{P} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S} \times \mathbb{R}$. Our HC-Dyna is building upon the tabular Dyna formalism 6 for MBRL. HC-Dyna provides a special approach to *Search-Control* (SC), which generates states by Hill Climbing (HC, i.e., the updating rule in SGLD) on some criterion/importance function $h(\cdot)$. According to SGLD sampling, such generative method ensures the sampled states follow certain desired sampling distribution as we introduce in the last section.

The algorithmic framework maintains two buffers: the conventional ER buffer storing experiences (an experience has the form of $(s_t, a_t, s_{t+1}, r_{t+1})$) and a *search-control queue* storing the states acquired by search-control mechanisms. At each time step t, a real experience[1] $(s_t, a_t, s_{t+1}, r_{t+1})$ is collected and stored into ER buffer. Then the HC search-control process starts to collect states and store them into the search-control queue. A imagined experience is obtained by first selecting a state s from the search-control queue, then select-

[1]Sometimes an experience is also called a transition.

ing an action a according to the current policy, and then querying the model to get the next state s' and reward r to form an experience (s, a, s', r). These imagined transitions are combined with real experiences into a single mini-batch to update the training parameters. The n updates, performed before taking the next action, are called *planning updates* (Sutton & Barto, 2018), as they improve the action-value estimates—and so the policy—using a model. The choice of pairing states with on-policy actions to form imagined experiences has been reported to be beneficial (Gu et al., 2016; Pan et al., 2018; Janner et al., 2019).

We want to mention that our SGLD sampling method for search-control has connections to exploration methods. Exploration methods typically add bonuses to action values when taking an action (Lattimore & Szepesvari, 2020) or to the reward function when learning the action values (Stadie et al., 2015; Bellemare et al., 2016; Pathak et al., 2017; Linke et al., 2019). Similar to SGLD, those exploration methods encourage visiting unseen states and potentially introduce state distribution bias. However, the main difference is that exploration encourages an agent to visit those unseen states physically. In contrast, our SGLD methods acquire unseen states by gradient ascent on some learned function $h(\cdot)$. Hence, some states generated by our mechanism may never be physically visited by the agent in the real environment.

By borrowing insights from both the reinforcement learning and the conventional supervised learning perspectives, the rest of this chapter will motivate and study three instances for $h(\cdot)$: 1) the value function $v(s)$ from Pan et al. (2019); 2) the gradient magnitude $||\nabla_s v(s)||$ from Pan et al. (2020b) and 3) the TD error magnitude from Mei et al. (2020).

6.3 Hill Climbing on Value Estimates

In this section, we motivate and present our search-control strategy based on hill climbing on value function estimate. We refer readers to Appendix B.1.1 for any missing experimental details.

6.3.1 A Motivating Example

We provide an example of how the value function surface changes during learning on a simple continuous-state GridWorld domain, which is a variant of the one introduced by (Peng & Williams, 1993). This provides an intuition on why it is useful to populate the search-control queue with states obtained by hill climbing on the estimated value function, as proposed in the next section.

Consider the GridWorld in Figure 6.1(a). In each episode, the agent starts from a uniformly sampled point from the area $[0, 0.05]^2$ and terminates when it reaches the goal area $[0.95, 1.0]^2$. There are four actions {UP, DOWN, LEFT, RIGHT}; each leads to a 0.05 unit move towards the corresponding direction. As a cost-to-goal problem, the reward is -1 per step. In Figure 6.1, we plot the value function surface after 0, 14k, and 20k mini-batch updates to DQN. We visualize the gradient ascent trajectories with 100 gradient steps starting from five states $(0.1, 0.1)$, $(0.9, 0.9)$, $(0.1, 0.9)$, $(0.9, 0.1)$, and $(0.3, 0.4)$. The gradient of the value function used in the gradient ascent is

$$\nabla_s V(s) = \nabla_s \max_a Q_\theta(s, a). \tag{6.1}$$

At the beginning, with a randomly initialized NN, the gradient with respect to state is almost zero, as seen in Figure 6.1(b). As the DQN agent updates its parameters, the gradient ascent generates trajectories directed towards the goal, though after only 14k steps, these are not yet contiguous, as seen Figure 6.1(c). After 20k steps, as in Figure 6.1(d), even though the value function is still inaccurate, the gradient ascent trajectories take all initial states to the goal area. This suggests that as long as the estimated value function roughly reflects the shape of the optimal value function, the trajectories provide a demonstration of how to reach the goal—or high-value regions—and speed up learning by focusing updates on these relevant regions.

More generally, by focusing planning on regions the agent *thinks* are high-value, it can quickly correct value function estimates before visiting those regions, and so avoid unnecessary interaction. We demonstrate this in Figure 6.1(e), where the agent obtains gains in performance by updating from high-value states, even when its value estimates have the wrong shape. After

20k learning steps, the values are flipped by negating the sign of the parameters in the output layer of the NN. Dyna-Value, introduced in Section 6.3.2, quickly recovers compared to DQN and on-policy updates from the ER buffer. Because the gradient ascent process can quickly generate states around the incorrectly high value regions, and then the planning steps help pushing down these erroneously high-values, and the agent can recover much more quickly.

6.3.2 Value-based Search-control

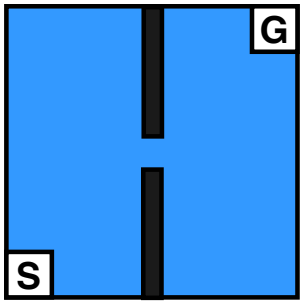
In this section, we present the algorithmic details of our Dyna variant, called Dyna-Value.[2] Note that the algorithm closely follows our generic Algorithm 7. We refer readers to the Appendix B.1.1 for any missing details.

The key component of our algorithm is to use the hill climbing procedure developed in the previous section, to generate states for SC. To generate states for search control, we need an algorithm that can climb on the estimated value function surface. For NNs, this can be difficult. The value function surface can be very flat or very rugged, causing the gradient ascent to get stuck in local optima and hence interrupt the gradient traveling process. Further, the state variables may have very different numerical scales. When using a regular gradient ascent method, it is likely for the state variables with a smaller numerical scale to immediately go out of the state space.

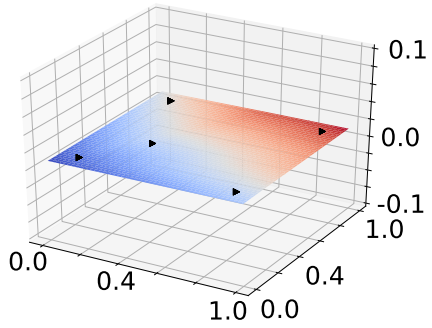
SGLD naturally address the first issue, of flat or rugged function surfaces by adding Gaussian noise on each gradient ascent step. Intuitively, this provides robustness to flat regions and avoids getting stuck in local maxima on the function surface, by diffusing across the surface to high-value regions.

To address the second issue of vastly different numerical scales among state variables, we use a standard strategy to be invariant to scale: natural gradient ascent. A popular choice of natural gradient is derived by defining the metric tensor as the Fisher information matrix (Amari & Douglas, 1998; Amari, 1998; Thomas et al., 2016). We introduce a simple and computationally efficient metric tensor: the inverse of covariance matrix of the states $\mathbf{\Sigma}_s^{-1}$. This choice

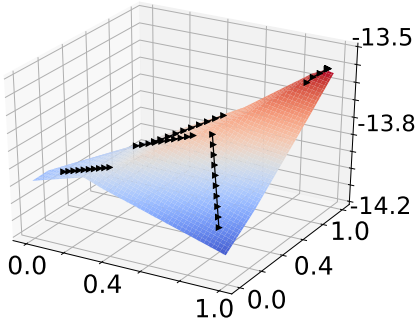
[2]Note that HC-Dyna is the name of our generic Dyna framework with SGLD search-control, while Dyna-Value is the name of a concrete instance of HC-Dyna.



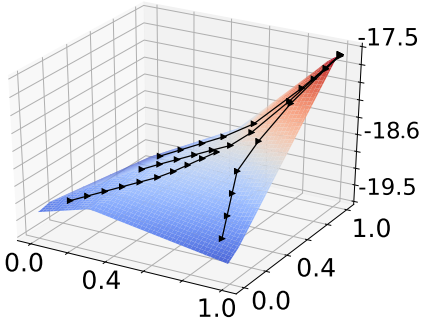
(a) GridWorld domain



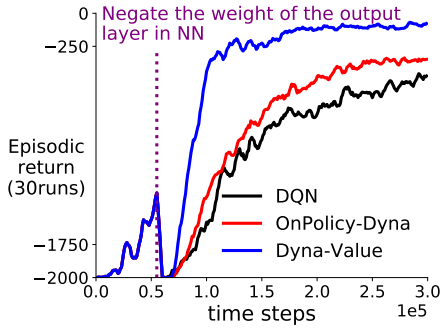
(b) Before update



(c) Update 14k times



(d) Update 20k times



(e) Negation of NN

Figure 6.1: (b-d) The value function on the GridWorld domain with gradient ascent trajectories. (e) shows learning curves (sum of rewards per episode v.s. time steps) where each algorithm needs to recover from a bad NN initialization (i.e. the value function looks like the reverse of (d)).

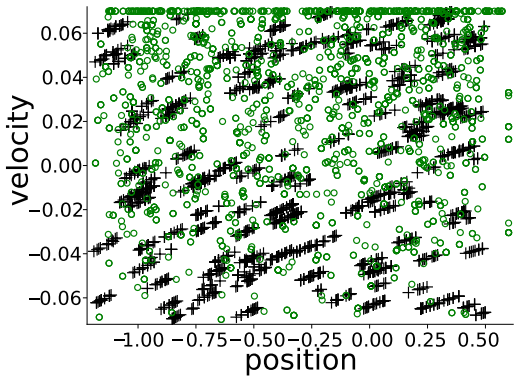


Figure 6.2: The search-control queue filled by using + or not using **o** natural gradient on MountainCar-v0.

is simple, because the covariance matrix can easily be estimated online. We can define the following inner product:

$$\langle s, s' \rangle = s^\top \mathbf{\Sigma}_s^{-1} s', \forall s, s' \in \mathcal{S},$$

which induces a vector space—the Riemannian manifold—where we can compute the distance of two points s and $s + \Delta$ that are close to each other by $d(s, s + \Delta) \stackrel{\text{def}}{=} \Delta^\top \mathbf{\Sigma}_s^{-1} \Delta$. The steepest ascent updating rule based on this distance metric becomes $s \leftarrow s + \alpha \mathbf{\Sigma}_s \mathbf{g}$, where $\mathbf{g}$ is the gradient of the value function.

We demonstrate the utility of using the natural gradient scaling. Figure 6.2 shows the states from the search-control queue filled by hill climbing in early stages of learning (after 8000 steps) on Mountain Car. The domain has two state variables with very different numerical scale: position $\in [-1.2, 0.6]$ and velocity $\in [-0.07, 0.07]$. Using a regular gradient update, the queue shows a state distribution with many states concentrated near the top since it is very easy for the velocity variable to go out of boundary. In contrast, the one with natural gradient, shows clear trajectories with an obvious tendency to the right top area (position ≥ 0.5), which is the goal area.

In addition to using this new method for search-control, we also found it beneficial to include updates on the experiences generated in the real world. The mini-batch sampled for training has ρ proportion of transitions generated by states from the SC queue, and $1 - \rho$ from the ER buffer. For example, for

$\rho = 0.75$ with a mini-batch size of 32, the updates consists of $24 (= 32 \times 0.75)$ transitions generated from states in the SC queue and 6 transitions from the ER buffer. Previous work using Dyna for learning NN value functions also used such mixed mini-batches (Holland et al., 2018).

One potential reason this mixing design is beneficial is that it alleviates issues with heavily skewing the sampling distribution to be off-policy. Tabular Q-learning is an off-policy learning algorithm, which has strong convergence guarantees under mild assumptions (Tsitsiklis, 1994). When moving to function approximation, empirically, previous prioritized ER work pointed out that skewing the sampling distribution from the ER buffer can lead to a biased solution (Schaul et al., 2016). Though the ER buffer is not on-policy, because the policy is continually changing, the distribution of states is closer to the states that would be sampled by the current policy than those in SC. Using mixed states from the ER buffer, and those generated by Hill Climbing, could alleviate some of the issues with this skewness. Another possible reason that such mixed sampling could be necessary is due to model error. The use of real experience could mitigate issues with such error.

We provide a small experiment in the continuous state GridWorld, depicted in Figure 6.1. The continuous-state setting uses NNs—as described more fully in Appendix B.1.1—with a mini-batch size of 32. Figure 6.3 shows the performance of Dyna-Value as the mixing proportion increases from 0 (ER only) to 1.0 (SC only). A mixing rate around $\rho = 0.5$ provides the best results. Generally, using too few search-control samples do not improve performance; focusing too many updates on search-control samples seems to slightly speed up early learning, but then later learning suffers.

To gain an intuition for why our algorithm with an appropriate mixing rate achieves superior performance than DQN, we visualize the states in the search-control queue for Dyna-Value with mixing rate 0.5 in the GridWorld domain (Figure 6.4). We also show the states in the ER buffer at the same time step for both Dyna-Value and DQN to contrast. There are two interesting outcomes from this visualization. First, the modification to search-control significantly changes where the agent explores, as evidenced by the ER buffer distribution.

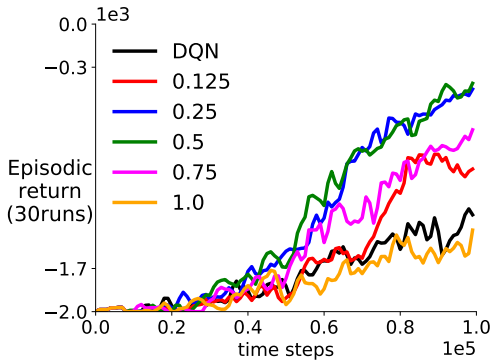
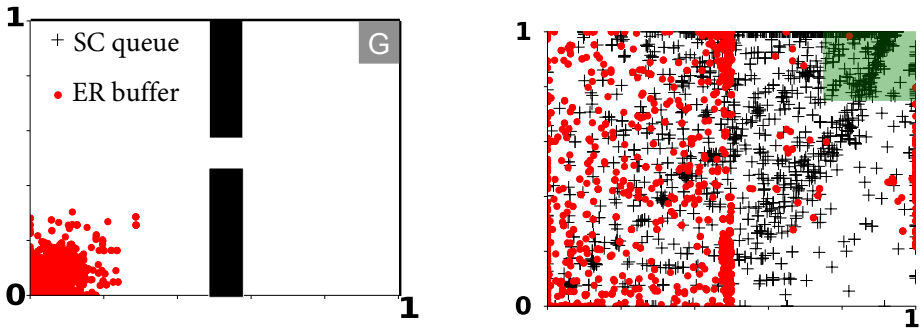


Figure 6.3: The effect of mixing rate on learning performance on (Continuous state) GridWorld. The numerical label means Dyna-Value with a certain mixing rate. The results are averaged over 30 runs (i.e., random seeds). The standard errors are small and sufficient to distinguish the learning curves using mixing rates $0.25, 0.5$ from the rest.

Second, Dyna-Value has many states in the SC queue that are near the goal region even when its ER buffer samples concentrate on the left part on the square. The agent can still update around the goal region even when it is physically in the left part of the domain.



(a) DQN (b) Dyna-Value

Figure 6.4: Figure (a)(b) show buffer(red ·)/queue(black +) distribution on GridWorld ($\mathbf{s} \in [0, 1]^2$) by uniformly sampling 2k states. (a) is showing ER buffer when running DQN; hence there is no "+" in it. (b) shows 0.2% of the ER samples fall in the green shadow (i.e., high-value region), while 27.8% samples from the SC queue are there.

6.4 Hill Climbing on Local Frequency

This section introduces another variant of gradient-based search-control strategy: we can get more samples from the regions of state space where the value function is more difficult to estimate. We firstly review some concepts in signal processing and conduct experiments in the supervised learning setting to show that a high frequency function is more difficult to approximate (Section 6.4.1). In order to quantify the difficulty of estimation, we borrow a crucial idea from the signal processing literature: a signal with higher frequency terms requires more samples for accurate reconstruction. We then propose a method to locally measure the frequency of a point in a function's domain and provide a theoretical justification for our method (Theorem 4 in Section 6.4.2). We use the hill climbing approach as discussed above to adapt our method to design a search-control mechanism for the Dyna architecture (Section 6.4.3).

6.4.1 Understanding the Difficulty of Function Approximation

In a regular regression setting, we illustrate that high frequency regions of a function is difficult to approximate. We show that by assigning more training data to those regions, the learning performance considerably improves. To make this insight practically useful, we employ the sum of gradient and Hessian norms of a function as a measure of the local frequency of a function. We establish a theoretical connection between our proposed criterion and the local frequency of a function. This would be the foundation of our frequency-based search-control method in Section 6.4.3.

Consider the standard regression problem with the mean square loss. Given a training set $D = \{(x_i, y_i)\}_{i=1:n}$, our goal is to learn an unknown target function $f^*(x) = \mathbb{E}[Y|X = x]$ by empirical risk minimization. Formally, we aim to solve

$$f = \operatorname*{argmin}_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} (f(x_i) - y_i)^2,$$

where $\mathcal{H}$ is some hypothesis space. Suppose that we can choose the distributions of samples $\{x_i\}$. How should we select them in order to improve the

quality of the learned function? One intuitive heuristic is that if we know the regions in the domain of f^* that are more difficult to approximate, we can assign more training data there in order to help the learning process. The important question is how to quantify the difficulty of approximating a function. We borrow an idea from the field of signal processing to suggest a method.

The Nyquist-Shannon sampling theorem in signal processing states that given a band-limited function (or signal) $f : \mathbb{R} \mapsto \mathbb{R}$ with the highest frequency (in the Fourier domain) of $\omega_{\text{bandwidth}}$, we can perfectly reconstruct it based on regular samples (in the time domain) obtained at the sampling rate of $2\omega_{\text{bandwidth}}$ (Zayed, 1993).[3] Therefore, if the Fourier transform of a function has high frequency terms, more samples are required to reconstruct it accurately. We note that the sampling theory has been applied in the sample complexity analysis of machine learning algorithms (Smale & Zhou, 2004, 2005; Jiang, 2019). Although the problem setting in machine learning is somewhat different from this result in signal processing, it still provides a high-level intuition for us: regions with higher frequency signal require more learning data.

To make this high-level intuition concrete, we consider the following function:

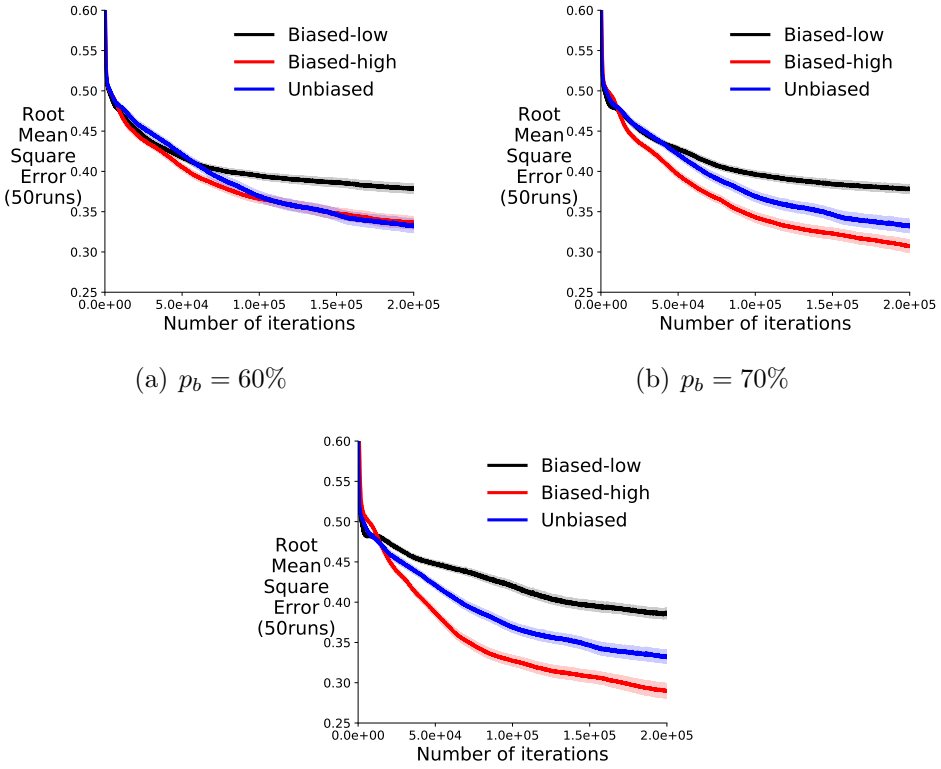
$$f_{\sin}(x) = \begin{cases} \sin(8\pi x) & x \in [-2, 0), \\ \sin(\pi x) & x \in [0, 2]. \end{cases} \tag{6.2}$$

It is easy to check that the regions $[-2, 0)$ and $[0, 2]$ contain signals with frequency ratio $8 : 1$. Based on the intuition from the sampling theorem, the $[-2, 0)$ interval requires more training data than the $[0, 2]$ interval. Given the same amount of training data, and the same learning algorithm, we would expect that assigning more fraction of the training data on $[-2, 0)$ to perform better than distributing them uniformly or assigning more samples to the $[0, 2]$ interval.

An illustrative experiment. To empirically verify the intuition, we conduct a simple regression task, with $f_{\sin}$ as the target function. The training set

[3] Sampling rate refers to number of samples per second used to reconstruct continuous signals.

$\mathcal{T} = \{(x_i, y_i)\}_{i=1:n}$ is generated by sampling $x \in [-2, 2]$, and adding Gaussian noise $\mathcal{N}(0, \sigma^2)$ on Eq. (6.2), where the standard deviation is set to be $\sigma = 0.1$. We present the ℓ_2 regression learning curves of training datasets with different biased sampling ratios $p_b \in \{60\%, 70\%, 80\%\}$, as shown in Figure 6.5 (a)-(c). We observe that biased training data sampling ratios towards high frequency region clearly speeds up learning. This is consistent with the intuitive insight and suggests that our heuristic to assign more data to high frequency regions leads to faster learning (Pan et al., 2020a).



(a) $p_b = 60\%$ (b) $p_b = 70\%$

(c) $p_b = 80\%$

Figure 6.5: Testing root mean squared error as a function of number of mini-batch updates. The naming rule of the learning curves is intuitive. For example, $p_b = 60\%$ means 60% of the training data are from the high frequency region $[-2, 0)$ and is labeled as **Biased-high**. We include unbiased training dataset as a reference (**Unbiased**). The total numbers of training data are the same across all experiments. The testing set is unbiased and the results are averaged over 50 random seeds with the shade showing standard error.

6.4.2 Identifying High Frequency Regions of a Function

Identifying the high-frequency region of $f_{\sin}$ in the previous toy problem was easy, as each region contained a signal with a constant known frequency. In practice, we face two main difficulties in identifying the high-frequency regions of a function. The first is that we do not have access to the underlying target function but only to an approximate function estimated using data, e.g., a trained neural network. The second is that frequency is a global property rather than a local one.

To make the high-frequency heuristic practically useful, we need a simple criterion that (a) uses function approximation, (b) characterizes local frequency information, and (c) can be efficiently calculated. Inspired by the function $f_{\sin}$ in Eq. (6.2), a natural idea is to calculate the first order $f'(x) \stackrel{\text{def}}{=} \frac{df(x)}{dx}$ or second order derivative $f''(x) \stackrel{\text{def}}{=} \frac{d^2 f(x)}{dx^2}$ because they both satisfy (a) and (c). To understand property (b), consider the following examples.

Example 1. *For $f_{\sin}$ defined in Eq. (6.2), calculate the integrals of squared first order derivative $f'_{\sin}$ on high frequency region $[-2, 0)$ and low frequency region $[0, 2]$, respectively:*

$$\int_{-2}^{0} |f'_{\sin}(x)|^2 \, dx = 64\pi^2, \quad \int_{0}^{2} |f'_{\sin}(x)|^2 \, dx = \pi^2.$$

Example 2. *Let $f : [-\pi, \pi] \to \mathbb{R}$ be a band-limited real valued function defined as*

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(nx) + b_n \sin(nx),$$

where $a_0, a_n, b_n \in \mathbb{R}$, $n = 1, 2, \ldots, N$ are Fourier coefficients of frequency $\frac{n}{2\pi}$. Then,

$$\int_{-\pi}^{\pi} |f'(x)|^2 \, dx = \pi \cdot \sum_{n=1}^{N} n^2 \left(a_n^2 + b_n^2\right), \quad \int_{-\pi}^{\pi} |f''(x)|^2 \, dx = \pi \cdot \sum_{n=1}^{N} n^4 \left(a_n^2 + b_n^2\right).$$

Example 1 shows that the integral of squared first-order derivative ratio is $64 : 1$ (the frequency ratio is $8 : 1$), and the region with large derivative magnitude is indeed the high-frequency region. Moreover, Example 2 indicates that for one-dimensional real-valued functions over a bounded domain, the integral

of a derivative magnitude is closely related to the frequency information. For the squared derivative, the integral is the same as weighting the frequency terms a_n and b_n proportional to n^2. For the squared second-order derivative, the integral is the same as weighting the frequency terms proportional to n^4. The weighting schemes n^2 or n^4 emphasize the higher frequency terms.

Given a function $f(\cdot)$ and a point x in its domain, we propose to measure the frequency of f around a small neighborhood of x (we call this *local frequency*) using the following function:

$$g(x) \stackrel{\text{def}}{=} \|\nabla_x f(x)\|^2 + \|H_f(x)\|_F^2, \tag{6.3}$$

where $\|\nabla_x f(x)\|$ is the ℓ_2-norm of the gradient at x, and $H_f(x)_F$ is the Frobenius norm of the Hessian matrix of f at x. We claim that the local frequency of f around x is proportional to $g(x)$. We theoretically justify this claim. For real-valued functions in the Euclidean space, our theory connects local gradient and Hessian norms to local function energy [4], and local frequency distribution. The proof of our theorem is in Appendix B.2.1.

Theorem 4. *Given any function $f : \mathbb{R}^n \to \mathbb{R}$, for any frequency vector $k \in \mathbb{R}^n$, define its local Fourier transform as*

$$\hat{f}(k) \stackrel{\text{def}}{=} \int_{y \in B(x,1)} f(y) \exp\left\{-2\pi i \cdot y^\top k\right\} dy,$$

for local function $f(y)$ defined around x, i.e., $y \in B(x,1) \stackrel{\text{def}}{=} \{y : \|y - x\| < 1\}$. Assume the local function "energy" is finite,

$$\int_{y \in B(x,1)} [f(y)]^2 \, dy = \int_{\mathbb{R}^n} \|\hat{f}(k)\|^2 dk < \infty, \quad \forall x \in \mathbb{R}^n. \tag{6.4}$$

Define "local frequency distribution" of $f(x)$ as:

$$\pi_{\hat{f}}(k) \stackrel{\text{def}}{=} \frac{\|\hat{f}(k)\|^2}{\int_{\mathbb{R}^n} \|\hat{f}(\tilde{k})\|^2 d\tilde{k}}, \quad \forall k \in \mathbb{R}^n. \tag{6.5}$$

[4] We consider the notion of energy in signal processing terminology: the energy of a continuous-time signal $x(t)$ is defined as $\int x(t)^2 dt$. In our theory, the function f is the signal.

Then, for any $x \in \mathbb{R}^n$, we have:

1) The first order connection:

$$\int_{y \in B(x,1)} \|\nabla f(y)\|^2 \, dy = 4\pi^2 \cdot \left[\int_{y \in B(x,1)} [f(y)]^2 \, dy \right] \cdot \left[\int_{\mathbb{R}^n} \pi_{\hat{f}}(k) \cdot \|k\|^2 \, dk \right], \tag{6.6}$$

2) The second order connection:

$$\int_{y \in B(x,1)} ||H_f(y)||_F^2 dy = 16\pi^4 \left[\int_{y \in B(x,1)} [f(y)]^2 \, dy \right] \cdot \left[\int_{\mathbb{R}^n} \pi_{\hat{f}}(k) \cdot \|k\|^4 \, dk \right]. \tag{6.7}$$

Remark 3. *Note that $\pi_{\hat{f}}$ defined in Eq. (6.5) is a probability distribution over $\mathbb{R}^n$ as:*

$$\int_{k \in \mathbb{R}^n} \pi_{\hat{f}}(k) dk = 1, \text{ and } \pi_{\hat{f}}(k) \geq 0, \quad \forall k \in \mathbb{R}^n.$$

We use such a distribution to characterize local frequency because it is more natural to use a distribution to describe the frequency of a function/signal since a signal usually has a range of frequencies. Introducing a distribution helps to turn the range of frequencies into an expectation, which is a scalar.

6.4.3 Frequency-based Search-control

In this section, we explain the Dyna architecture with the frequency-based search-control (Dyna-Frequency), which, again, closely follow the generic Algorithm 7. To find states in high-frequency regions, we combine sampling from high-frequency regions and high-value regions of the state space. We refer readers to Algorithm 11 in the appendix for implementation details.

Our goal is to query the model more often from the states in high-frequency regions of the value function. The intuition behind this search-control mechanism is that those regions correspond to where learning the (value) function is more difficult. Hence more states from the region might be helpful. To populate the search-control queue with states from those regions, we can do hill climbing on $g(s) = \|\nabla_s V(s)\|^2 + \|H_v(s)\|_F^2$. Theorem 4, however, suggests that states with large gradient norm can either have large absolute value, or high local frequency, or both. We want to avoid many samples from regions

with large negative value states, as those states may be rarely visited under the optimal policy. A sensible strategy to get around this problem is to combine the proposed hill climbing method with the previous strategy of hill climbing on the value function, as the latter tends to generate high-value states.

We propose the following method for combining those approaches. At each environment time step, with a certain probability p, we perform hill climbing by either

$$s \leftarrow s + \alpha \begin{cases} \nabla_s g(s) & \text{with probability of } p \quad (6.8a) \\ \nabla_s V(s) & \text{with probability of } 1-p \quad (6.8b) \end{cases}$$

and store states along the gradient trajectory in the search-control queue. Note that in this case, the criterion function $h(\cdot)$ introduced in the generic framework 7 is stochastically sampled.

When hill climbing on the value function (6.8b), we sample the initial state from the ER buffer. This populates the search-control queue with states from the high-value regions of the state space. When hill climbing on $g(s)$ (6.8a), however, we sample the initial state from the search-control queue itself (instead of the ER buffer). This way ensures that the initial state for searching high-frequency region has a relatively high value. Hill climbing on $g(s)$ from an initial state with a high value populates the search-control queue with high-frequency samples around high-value regions of the state space.

Similar to hill climbing on the value function, we obtain the state-value function in both (6.8a) and (6.8b) by taking the maximum of the estimated action-value, i.e. $V(s) = \max_a Q(s,a) \approx \max_a Q_\theta(s,a)$ where θ is the parameter of the Q-network.

As one can see, there are several limitations of hill climbing on value function or local frequency. First, they do not provide any theoretical justification about why using the stochastic gradient ascent trajectories for search-control can improve sample efficiency. Second, HC on gradient norm and Hessian norm of the learned value function suffer from significant computation cost and zero or explosive gradient due to the high order differentiation (i.e., $\nabla_s ||\nabla_s v(s)||$) as suggested by the authors. When using ReLu as activation functions, such high order differentiation almost results in zero gradients. We empirically observed

this phenomenon. This drawback can also be reasoned by intuition from the work by Goodfellow et al. (2015), which suggests that ReLU neural networks are locally almost linear. Then it is not surprising to have zero higher order derivatives. Third, the two methods are prone to find sub-optimal policies. Consider that the value function is relatively well-learned and fixed. Then value/frequency-based search-control would still find those high-value states even though they might already have low TD error. In the next section, we propose a method with a solid theoretical intuition and does not have the limitations of the two previously discussed methods.

6.5 Hill Climbing on TD Error Magnitude

In this section, we propose a new search-control strategy motivated by overcoming the limitation of prioritized sampling distribution that has been deployed in the prioritized ER method (Schaul et al., 2016). We firstly provide a theoretical insight into the prioritized ER's advantage and point out its two drawbacks: outdated priorities and insufficient sample space coverage, which may significantly weaken its efficacy. To mitigate the two issues, we apply the SGLD sampling method to acquire states and leverage an environment model to acquire imagined experiences by simulating priorities. We demonstrate that the samples generated by our method are distributed closer to the ideal TD error-based sampling distribution (i.e., the one does not suffer from the two drawbacks).

6.5.1 Theoretical Insight into Prioritized Sampling

In the l_2 regression, we minimize the mean squared error $\min_\theta \frac{1}{2n}\sum_{i=1}^{n}(f_\theta(x_i)-y_i)^2$, for training set $\mathcal{T}=\{(x_i,y_i)\}_{i=1}^{n}$ and function approximator f_θ, such as a neural network. In error-based prioritized sampling, we define the priority of a sample $(x,y)\in\mathcal{T}$ as $|f_\theta(x)-y|$; the probability of drawing a sample $(x,y)\in\mathcal{T}$ is typically $q(x,y;\theta)\propto|f_\theta(x)-y|$. For a point $(x,y)\in\mathcal{T}$, we employ the following form to compute the probabilities:

$$q(x,y;\theta)\stackrel{\text{def}}{=}\frac{|f_\theta(x)-y|}{\sum_{i=1}^{n}|f_\theta(x_i)-y_i|} \tag{6.9}$$

We can show an equivalence between the gradients of the squared objective with this prioritization and the cubic power objective $\frac{1}{3n}\sum_{i=1}^{n}|f_\theta(x_i)-y_i|^3$ in the Theorem 5 below. The empirical demonstration of this equivalence is in Appendix B.3.3.

Theorem 5. *For a constant c determined by θ and $\mathcal{T}$, we have*

$$c\mathbb{E}_{(x,y)\sim q(x,y;\theta)}[\nabla_\theta \frac{1}{2}(f_\theta(x)-y)^2] = \mathbb{E}_{(x,y)\sim uniform(\mathcal{T})}[\nabla_\theta \frac{1}{3}|f_\theta(x)-y|^3].$$

Proof. The R.H.S. is:

$$\begin{aligned}\mathbb{E}_{(x,y)\sim uniform(\mathcal{T})}[\nabla_\theta \frac{1}{3}|f_\theta(x)-y|^3] &= \mathbb{E}_{(x,y)\sim uniform(\mathcal{T})}[\frac{1}{3}\nabla_\theta(|f_\theta(x)-y|^2)^{\frac{3}{2}}] \\ &= \frac{1}{3n}\sum_{i=1}^{n}\nabla_\theta(|f_\theta(x_i)-y_i|^2)^{\frac{3}{2}}] \\ &= \frac{1}{2n}\sum_{i=1}^{n}|f_\theta(x_i)-y_i|\nabla_\theta(f_\theta(x_i)-y_i)^2\end{aligned}$$

And the L.H.S. is

$$\begin{aligned}&\mathbb{E}_{(x,y)\sim q(x,y;\theta)}[\nabla_\theta \frac{1}{2}(f_\theta(x)-y)^2] = \sum_{i=1}^{n}\frac{1}{2}q(x_i,y_i;\theta)\nabla_\theta(f_\theta(x_i)-y_i)^2 \\ &= \frac{1}{2\sum_{j=1}^{n}|f_\theta(x_j)-y_j|}\sum_{i=1}^{n}|f_\theta(x_i)-y_i|\nabla_\theta(f_\theta(x_i)-y_i)^2 \\ &= \frac{n}{\sum_{i=1}^{n}|f_\theta(x_i)-y_i|}\mathbb{E}_{(x,y)\sim uniform(\mathcal{T})}[\nabla_\theta \frac{1}{3}|f_\theta(x)-y|^3]\end{aligned}$$

Setting $c = \frac{\sum_{i=1}^{n}|f_\theta(x_i)-y_i|}{n}$ completes the proof. □

This simple theorem provides an intuitive reason for why prioritized sampling can help improve sample efficiency: the gradient direction of the cubic function is sharper than that of the square function when the error is relatively large (Figure B.5). We refer readers to the work by Fujimoto et al. (2020) regarding more discussions about the equivalence between prioritized sampling and uniform sampling.

Below, theorem 6 further characterizes the difference between the convergence rates by using gradient descent to optimize the mean squared error and

the cubic power objective, providing a solid motivation for using error-based prioritized sampling. Please see Appendix B.3.1 for the proof and its corresponding hitting time simulations.

Theorem 6 (Fast early learning)**.** *Consider the following two objectives:*

$$\ell_2(x,y) \stackrel{\text{def}}{=} \frac{1}{2}(x-y)^2, \ell_3(x,y) \stackrel{\text{def}}{=} \frac{1}{3}|x-y|^3.$$

Define the functional gradient flow updates on these two objectives:

$$\frac{dx_t}{dt} = -\eta \frac{d\{\frac{1}{2}(x_t-y)^2\}}{dx_t}, \frac{d\tilde{x}_t}{dt} = -\eta \frac{d\{\frac{1}{3}|\tilde{x}_t-y|^3\}}{d\tilde{x}_t}.$$

Define $\delta_t \stackrel{\text{def}}{=} |x_t - y|, \tilde{\delta}_t \stackrel{\text{def}}{=} |\tilde{x}_t - y|$*. Given error threshold* $\epsilon \geq 0$*, define the hitting time* $t_\epsilon \stackrel{\text{def}}{=} \min_t\{t : \delta_t \leq \epsilon\}$ *and* $\tilde{t}_\epsilon \stackrel{\text{def}}{=} \min_t\{t : \tilde{\delta}_t \leq \epsilon\}$*. For any initial function value* x_0 *s.t.* $\delta_0 > 1$*,* $\exists \epsilon_0 \in (0,1)$ *such that* $\forall \epsilon > \epsilon_0, t_\epsilon \geq \tilde{t}_\epsilon$*.*

This theorem says that it is faster to get to a certain low loss point with the cubic objective when the initial loss is relatively large. Though it is not our focus here to investigate the practical utility of the high power objectives, we include some empirical results and discuss the reasons why such objectives should not be preferred in general problems in Appendix B.3.2.

6.5.2 Limitations of the Prioritized ER

Inspired by the above theorems, we now discuss two drawbacks of prioritized sampling: **outdated priorities** and **insufficient sample space coverage**. We empirically examine their importance and effects in Appendix B.3.3.

The above two theorems show that prioritized sampling benefits from the faster convergence rate of the cubic power objective. The equivalence in Theorem 5 requires to update the priorities of *all training samples* by using the *updated training parameters* θ at each time step. In an online RL setting, however, at the current time step t, the original prioritized ER method only updates the priorities of those experiences from the sampled mini-batch, leaving the priorities of the rest of experiences unchanged (Schaul et al., 2016). We call this limitation of prioritized ER **outdated priorities**. It is typically not feasible to update the priorities of all visited experiences at each time step.

In fact, in an online RL setting, "*all training samples*" in RL are restricted to those visited experiences in the ER buffer, which may only contain a small subset of the whole state space, making the estimate of the prioritized sampling distribution inaccurate. There can be many reasons for the small coverage: the exploration is difficult, the state space is too large, or the memory resource of the buffer is quite limited. We call this issue **insufficient sample space coverage**, which is also noted by Fedus et al. (2020).

Note that the issue of *insufficient sample space coverage* should not be considered equivalent to the off-policy distribution issue in ER methods. The latter refers to some old experiences in the ER buffer that may be unlikely to appear under the current policy (Novati & Koumoutsakos, 2019; Zha et al., 2019; Sun et al., 2020; Oh et al., 2021). In contrast, the issue of insufficient sample space coverage can rise naturally. For example, the state space is large, and an agent can only visit a small subset of the state space during the early learning stage. Then those states in the buffer have a small coverage of the state space. We visualize the state space coverage issue on an RL domain in Section 6.5.3.

6.5.3 TD Error-based Search-control

We now introduce our TD-error-based prioritized sampling for search-control, which enables us to acquire states *1) whose absolute TD errors are estimated by using current parameter θ_t and 2) that are not restricted to those visited ones*. As a result, we overcome the limitations discussed in the above section.

Let $v^\pi(\cdot;\theta_t) : \mathcal{S} \mapsto \mathbb{R}$ be a differentiable value function under policy π parameterized by θ_t. For $s \in \mathcal{S}$, define $y(s) \stackrel{\text{def}}{=} \mathbb{E}_{r,s' \sim \mathcal{P}^\pi(s',r|s)}[r + \gamma v^\pi(s';\theta_t)]$, and denote the TD error as $\delta(s,y;\theta_t) \stackrel{\text{def}}{=} y(s) - v(s;\theta_t)$. Then we simply set the criterion function based on which we do SGLD as $h(s_i) \stackrel{\text{def}}{=} \log|\delta(s_i, y(s_i);\theta_t)|$. This would give us the state samples whose distribution is approximately $p(s) \propto |\delta(s, y(s), \theta_t)|$, according to the SGLD sampling theory reviewed in Section 6.2. We can simply plug this SGLD updating rule into Algorithm 7 to get states for search-control. We call this algorithm Dyna-TD.

Implementation. In practice, we can compute the state value estimate by $v(s) = \max_a Q(s, a; \theta_t)$ as suggested before. In the case that a true environment model is not available, we compute an estimate $\hat{y}(s)$ of $y(s)$ by a learned model. Then at each time step t, states approximately following the distribution $p(s) \propto |\delta(s, y(s))|$ can be generated by

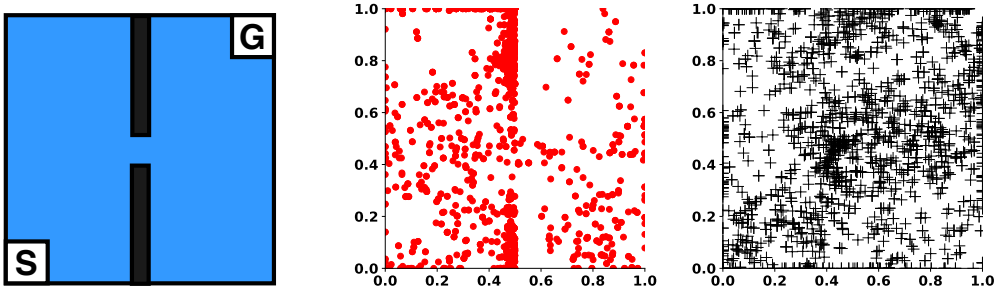
$$s \leftarrow s + \alpha_h \nabla_s \log |\hat{y}(s) - \max_a Q(s, a; \theta_t)| + X, \tag{6.10}$$

where X is a Gaussian random variable with zero-mean and some small variance. In the implementation, observing that α_h is small, we consider $\hat{y}(s)$ as a constant given a state s without backpropagating of θ through it.

Empirical verification of the SGLD sampling method. We visualize the distribution of the sampled states by our sampling method and those from the buffer of the prioritized ER, verifying that our sampled states have more extensive coverage of the state space. We then empirically verify that our sampling distribution is closer to a brute-force calculated (as detailed soon) prioritized sampling distribution—which does not suffer from the two limitations—than the prioritized ER method does. Finally, we discuss concerns regarding computational cost. Please see Appendix B.3.4 for any missing details.

Large sample space coverage. During early learning (at the 16kth environment time step), we visualize 2k states sampled from 1) DQN's buffer trained by prioritized ER and 2) our algorithm Dyna-TD's Search-Control (SC) queue on GridWorld (Figure 6.6(a)). Figure 6.6 (b) shows that DQN's ER buffer does not cover sufficiently the top-left part and the right half part. In contrast, Figure 6.6 (c) shows that states from our SC queue are distributed almost everywhere on the square. These visualizations verify that our sampled states cover broader sample space than the prioritized ER does.

Sampling distribution is close to the ideal one. We denote our sampling distribution as $p_1(\cdot)$, the one acquired by conventional prioritized ER as $p_2(\cdot)$, and the one computed by thorough priority updating of enumerating all states

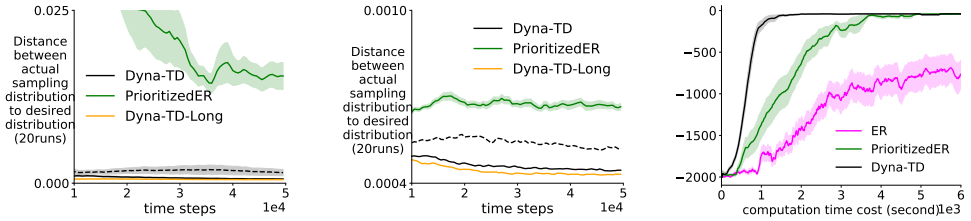


(a) GridWorld (b) PrioritizedER buffer (c) Dyna-TD SC queue

Figure 6.6: (a) shows the GridWorld. The state space is $\mathcal{S} = [0, 1]^2$, and the agent starts from the left bottom and should learn to take action from $\mathcal{A} = \{up, down, right, left\}$ to reach the right top within as few steps as possible. (b) shows the states sampled from the ER buffer of prioritized ER. (c) shows the SC queue state distribution of our Dyna-TD.

in the state space as $p^*(\cdot)$ (this one should be unrealistic in practice and we call it the ideal distribution as it does not suffer from the two limitations we discussed). We visualize how well $p_1(\cdot)$ and $p_2(\cdot)$ can approximate $p^*(\cdot)$ on the GridWorld domain, where the state distributions can be conveniently estimated by discretizing the continuous state GridWorld to a 50×50 one. We compute the distances of p_1, p_2 to p^* by two sensible weighting schemes: 1) on-policy weighting: $\sum_{j=1}^{2500} d^\pi(s_j)|p_i(s_j) - p^*(s_j)|, i \in \{1, 2\}$, where d^π is approximated by uniformly sample 3k states from a recency buffer; 2) uniform weighting: $\frac{1}{2500}\sum_{j=1}^{2500} |p_i(s_j) - p^*(s_j)|, i \in \{1, 2\}$.

We plot the distances change when we train our algorithm and the prioritized ER in Figure 6.7(a)(b). They show that the Hill Climbing (HC) procedure in our algorithm Dyna-TD, either with a true or an online learned model, produces a state distribution with significantly closer distance to the desired sampling distribution p^* than PrioritizedER under both weighting schemes. In contrast, the state distribution acquired from PrioritizedER, which suffers from the two limitations, is far away from p^*. It should also be noted that we include Dyna-TD-Long, which runs a large number of HC steps. Its corresponding sampling distribution should be closer to the stationary distribution. However, there is only a tiny difference between the regular Dyna-TD and



(a) on-policy weighting (b) uniform weighting (c) time cost v.s. episodic return

Figure 6.7: (a)(b) show the distance change as a function of environment time steps for **Dyna-TD (black)**, **PrioritizedER (forest green)**, and **Dyna-TD-Long (orange)**, with different weighting schemes. The **dashed** line corresponds to our algorithm with an online learned model. The corresponding evaluation learning curve is in the Figure 6.8(c). (d) shows the policy evaluation performance as a function of running time (in seconds) with **ER(magenta)**. All results are averaged over 20 random seeds. The shade indicates standard error.

Dyna-TD-Long, implying that one can save computational cost by running fewer HC steps.

Computational cost. It is known that the Langevin dynamics sampling method requires considerable computation power. Note that our HC rule 6.10 is doing gradient w.r.t. a single state, rather than a mini-batch, hence its computational cost is reasonable. Let the mini-batch size be b, and the number of HC steps be k_{HC}. If we assume one mini-batch update takes $\mathcal{O}(c)$, then the time cost of our sampling is $\mathcal{O}(ck_{HC}/b)$. On the GridWorld, Figure 6.7(c) shows that given the same time budget, our algorithm achieves better performance, even though DQN and PrioritizedER can process many more samples per second. Hence the additional time spent on search-control is worth it.

6.6 Experiments

In this section, we design experiments to answer the following questions. (1) Can Dyna variants outperform those ER-based model free baselines? (2) By mitigating the limitations of the conventional prioritized ER method, can Dyna-TD outperform the prioritized ER under various planning budgets in

different environments? (3) Can Dyna-TD outperforms the previous Dyna variants (i.e., Dyna-TD, Dyna-Frequency)? (4) What could be the practical implications of using Dyna-TD in a real-world application?

Baselines. **ER** is DQN with a regular ER buffer without prioritized sampling. **PrioritizedER** is the one by Schaul et al. (2016), which has the drawbacks as discussed in Section 6.5.2. **Dyna-Value** is the Dyna variant that performs HC on the learned value function to acquire states to populate the SC queue (see Section 6.3). **Dyna-Frequency** is the Dyna variant that performs HC on the norm of the gradient of the value function to acquire states to populate the SC queue (see Section 6.4). For a fair comparison, we stochastically sample the same number of mini-batches at each environment time step to train those model-free baselines as the number of planning updates in Dyna variants. We are able to fix the same HC hyper-parameter setting across all environments. Please see Appendix B.3.4 for any missing details.

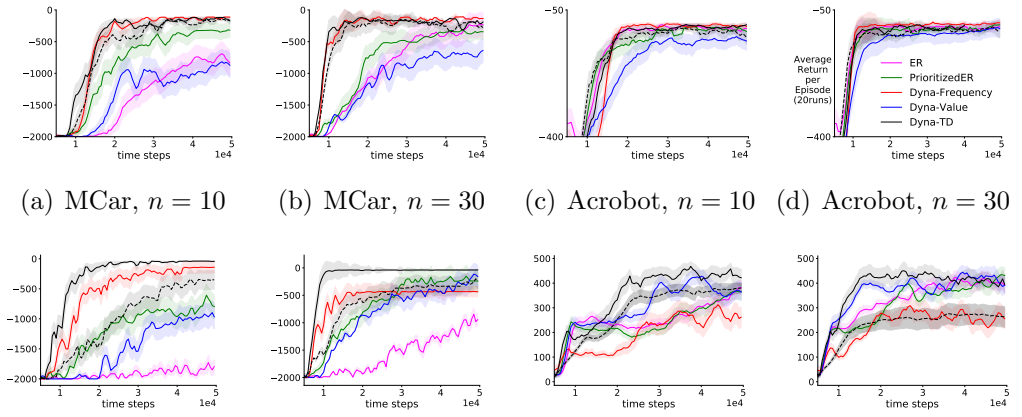
Since the main focus of this chapter is to study search-control, we isolate the model error effect by assuming an accurate environment model is available in our experiments. For curiosity, we test using a learned model for the Dyna-TD method, which consistently performs the best in our experience. Whenever using an online learned model, the model is learned by uniformly sampling a mini-batch of experiences from the ER buffer at each environment time step. It is reasonable to believe that some special model learning techniques may benefit a particular search-control strategy. We leave such techniques as a future direction.

Overall Performance. Figure 6.8 shows the performances of different algorithms on MountainCar, Acrobot, GridWorld (Figure 6.6(a)), and CartPole. First, our Dyna variants consistently outperform model-free baselines, including ER and PrioritizedER across domains and planning update settings. Particularly, as occurred in the supervised learning experiment, the PrioritizedER may not even outperform regular ER due to the limitations we discussed.

Second, Dyna-TD's performance significantly improves and even outper-

forms other Dyna variants when increasing the planning budget (i.e., planning updates n) from 10 to 30. This superiority validates the utility of those additional imagined experiences acquired by our sampling method. In contrast, both ER and PrioritizedER show limited gain when increasing the number of mini-batch updates, implying the drawback of only using those visited experiences.

Third, Dyna-Value/Frequency frequently converges to a sub-optimal policy when using a large number of planning updates, while Dyna-TD always finds a better one. It may be that the two Dyna variants frequently generate high-value/frequency states whose TD errors are low, which wastes samples. Furthermore, the severe sampling distribution bias can hurt the performance (Schaul et al., 2016). Dyna-Frequency also suffers from explosive or zero gradients and is sensitive to hyper-parameters, which may explain its inconsistent performance.



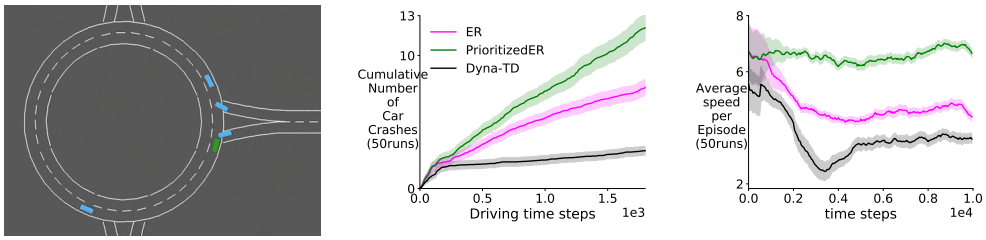
(a) MCar, $n = 10$ (b) MCar, $n = 30$ (c) Acrobot, $n = 10$ (d) Acrobot, $n = 30$

(e) GWorld, $n = 10$ (f) GWorld, $n = 30$ (g) CartPole, $n = 10$ (h) CartPole, $n = 30$

Figure 6.8: Episodic return v.s. environment time steps. We show evaluation learning curves of **Dyna-TD (black)**, **Dyna-Frequency (red)**, **Dyna-Value (blue)**, **PrioritizedER (forest green)**, and **ER(magenta)** with planning updates $n = 10, 30$ on Mountain Car (MCar), Acrobot, GridWorld(GWorld) and CartPole domains. The **dashed** line denotes Dyna-TD with an online learned model. All results are averaged over 20 random seeds after smoothing over a window of size 30. The shade indicates standard error.

Autonomous driving application. We study the practical utility of our method in a relatively large autonomous driving application (Leurent, 2018) with an online learned model. We use the roundabout-v0 domain (Figure 6.9 (a)) whose state space is $\mathcal{S} \subset \mathbb{R}^{90}$. The agent should learn to go through a roundabout by lane change and longitude control. The reward is designed such that the car should go through the roundabout as fast as possible without collision.

We observe that all algorithms perform similarly when evaluating algorithms by episodic return (i.e. sum of rewards per episode). However, there is a significantly lower number of car crashes with the policy learned by our algorithm, as shown in Figure 6.9(b). Figure 6.9 (c) suggests that ER and PrioritizedER gain reward mainly due to fast speed, which potentially incurs many car crashes. Though car crashes incur high TD error, the conventional prioritized ER method still incurs many crashes, which may indicate its prioritized sampling distribution does not provide enough crash experiences to learn. By actively searching for such experiences, our agent gets sufficient training during the planning stage and can reduce crashes.



(a) roundabout (b) Num of car crashes (c) Avg. speed

Figure 6.9: (a) shows the roundabout domain. (b) shows crashes v.s. total driving time steps during policy evaluation. (c) shows the average speed per evaluation episode v.s. environment time steps. We show **Dyna-TD (black)** with an online learned model, **PrioritizedER (forest green)**, and **ER (magenta)**. Results are averaged over 50 random seeds after smoothing over a window of size 30. The shade indicates standard error.

Chapter 7

Discussion

This thesis attempted to answer the question:

how can we improve the sample efficiency of a reinforcement learning algorithm with limited computational and memory resources?

We provided some answers for this question in both prediction and control settings. In prediction, we propose a new category of gradient temporal different learning algorithms, called Accelerated Temporal Difference (ATD) learning. We derive an approximate second-order optimization method by bringing in efficient, incremental matrix approximation techniques: SVD and random projection. We prove that the expected update of ATD is convergent to the unbiased solution under certain assumptions. We empirically demonstrate its improved sample efficiency and parameter insensitivity, even with significant approximations in the preconditioning matrix. The family of ATD algorithms provides a promising avenue for using stochastic gradient descent results to improve sample complexity with feasible computational complexity.

In the control setting, we investigate designing efficient sampling distribution by stochastic gradient Langevin dynamics for search-control mechanism in Dyna architecture. We propose such mechanisms based on ideas originating from both reinforcement learning and regular supervised learning settings. Our mechanisms leverage the generalization power of some learned function approximator (e.g., value function) to perform hill-climbing to find states that have high value, are in the high-frequency region, or have high TD error magnitude. We present a new Dyna algorithm, called HC-Dyna, which generates states for

search-control by SGLD methods. We demonstrate that our algorithms can significantly improve sample efficiency in several benchmark domains.

This discussion chapter is organized as follows. We discuss some limitations of our approach, which seem not to have an immediate or straightforward solution. We then conclude this chapter by discussing some potential future works.

7.1 Limitations

This section discusses limitations of our ATD algorithms and then limitations of our gradient-based search-control methods.

7.1.1 ATD Algorithms

First, in practice, our ATD algorithms require a decaying learning rate to show high sample efficiency, which puts an obstacle in a continual learning setting.[1] We illustrate this limitation by empirically comparing ATD with the full preconditioning matrix and the incrementally approximate one on the mountain car domain. We set the bootstrap parameter $\lambda = 0$. Other settings are the same as previous experiments whenever applicable.

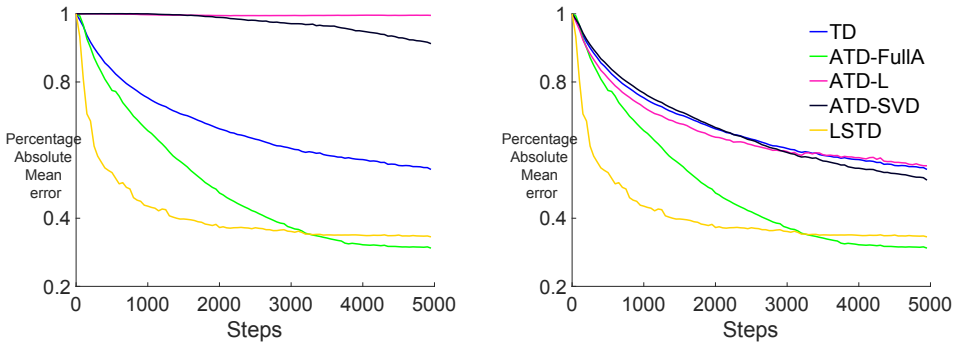
We conduct two sets of experiments for our ATD with SVD approximation (labeled as **ATD-SVD**) and with random projection approximation (labeled as **ATD-L**) by using a constant step size. The first set is in Figure 7.1(a). It shows ATD algorithms with a fixed, very small regularizer η as we did in the earlier experiments in this thesis. We only sweep over the constant learning rate in this set of experiments. The purpose is to directly contrast with our previous results of ATD with decaying learning rate. The second set of experiments is sweeping over *both* learning rate and regularizer for ATD algorithms. The purpose is to verify that even with thorough hyper-parameter optimization, ATD algorithms with constant learning rates can still not work well. As we expected, they choose a small learning rate but a relatively large

[1] Though it is quite common to require a decaying learning rate in theory to guarantee convergence, linear TD methods are usually able to work reasonably well with a constant learning rate in implementation.

regularization parameter, resulting in a similar performance to the linear TD algorithm.

In summary, Figure 7.1 shows that: 1) with the full $\mathbf{A}$ matrix (without the truncation effect), ATD learns faster with a constant learning rate and can even find a better solution than LSTD within 5k time steps. 2) in contrast, ATD-SVD and ATD-L learn much slower with a constant learning rate as it has to choose a small learning rate to avoid divergence. The two crucial observations illuminate the negative effect of the matrix approximation when incrementally maintaining the preconditioning matrix.

It should be noted that the unknown truncation effect of the SVD should not be the reason for the decaying learning rate, as the random projection method does not have such an effect, but it still needs a decaying learning rate. An intuitive explanation is that the incremental approximation of the preconditioning matrix can augment the error between the actual updating rule and the expected updating rule. Hence one must reduce the preconditioning part to offset the error. We leave it as future work to rigorously understand the reason behind the demand of using a decaying learning rate.



(a) Fixed regularizer (b) Optimized regularizer

Figure 7.1: Learning curves in terms of Percentage absolute mean error by using (a) a fixed regularization parameter and (b) an optimized regularization parameter for ATD variants. The labels are the same in both figures. The results are averaged over 50 random seeds.

Second, it remains to understand the convergence property of our ATD's stochastic updating rule. It should be noted that we only showed the conver-

gence of the expected updating rule in Section 4.4. One difficulty exists in the truncation error effect, which is still an open problem. The significance of solving such a problem may lead to a general theoretical framework to study the convergence of a broader class of preconditioning temporal difference learning algorithms.

Third, ATD with matrix sketching via random projection seems to be sensitive to feature type. One can observe this from the results in Section 5.4. It is unclear if there is a rule-of-thumb or some theory to help decide if our matrix sketching technique should be adopted given a particular feature type.

7.1.2 Gradient-based Search-control

First, we do not directly establish a connection between the convergence rate of RL algorithms and sampling distribution. Our methods currently only borrow theoretical insights from simplified supervised learning.

Second, we do not rigorously address biased sampling distribution issue; instead, we only propose a mixed mini-batch technique to mitigate such an issue. The consequence is that it would lead to a biased solution if we keep using the imagined experiences.

Third, the gradient-based search-control can provide some unrealistic states, which possibly negatively affect the performance when most of such states are unrealistic. Though some projections may be used to project those states back into the state space, it could be unrealistic to know how this projection should be implemented.

Last, the stochastic gradient Langevin dynamics method has high computational complexity as a classic Monte Carlo sampling method. However, this limitation could be advantageous in the long term as we usually observe that increasing computational power enables better performance when using such a sampling method.

7.2 Future Research Directions

Among the many possible future directions, we prioritize pursuing the following directions.

First, it is natural to investigate the utility of ATD in an off-policy policy evaluation setting. Our empirical study only focuses on on-policy policy evaluation settings without using the emphatic trace, although we provide a generic off-policy policy evaluation objective and notations in Section 3.2. One particular challenge of the off-policy policy evaluation problem is the variance of the importance ratio (Precup et al., 2000; Liu et al., 2018). Hence, a potentially interesting motivation is to study if ATD with low-rank approximation can help reduce variance resulting from the importance ratio, which would be needed to estimate the preconditioning matrix.

Second, applying the ATD algorithm in a deep learning setting may be of great interest as linear function is unlikely to be sufficiently expressive for complex problems. A tentative way to do so is to bring in reservoir computing approaches (Tanaka et al., 2019). We can consider that the feature vector comes from a computing reservoir (e.g. echo state network), and then the corresponding state value is linear in this feature. Then the existed empirical insights and theoretical development naturally apply.

Third, there is significance to bridge the online and offline (i.e., batch mode) learning algorithms. The latter is one type of the most actively studied and applied policy evaluation algorithms (Thomas et al., 2015b,a; Jiang & Li, 2016; Thomas & Brunskill, 2016; Dai et al., 2020). These methods learn from a batch of data rather than a single data point at each time step. They hence can learn knowledge from large amounts of data, resulting in more stable performance. Furthermore, due to the availability of a batch of data, these algorithms may further benefit from better-designed reweighting schemes (Jiang & Li, 2016; Thomas & Brunskill, 2016; Zhang et al., 2020; Wen et al., 2020; Hanna et al., 2019) or even learning objectives (Thomas et al., 2015a; Liu et al., 2018). On the other hand, fully online algorithms are capable of learning from a data point immediately whenever it becomes available and then discard it, which

seems to do things that a really "intelligent" agent can do. However, a bad single online update may impede the performance. Bridging the two types of algorithms should aim at 1) stabilizing the learning algorithm's performance by reducing estimation variance; 2) reducing the amount of data needed to be stored; 3) improving performance immediately whenever a data point becomes available.

Last, it is interesting to develop and test ATD's control version. It is intuitive to extend a policy evaluation algorithm to a control one: we can simply evaluate/learn the action-value instead of the state value and then take the action greedily w.r.t. to the action values. The challenging part should exist in the theoretical analysis of the convergence property. A recent work by (Devraj & Meyn, 2017) may provide nice suggestions regarding theory.

Along the research line of our gradient-based search-control strategy, there are several promising future directions. First, a natural follow-up question is how a model should be learned to benefit our sampling method. This work mostly focuses on sampling imagined experiences rather than model learning algorithms. Existing results show that learning a model while taking into account how to use it should make the learning performance robust to model errors (Farahmand et al., 2017; Farahmand, 2018). Second, one may apply our approach with a model in some latent space (Hamilton et al., 2014; Wahlström et al., 2015; Ha, David and Schmidhuber, Jürgen, 2018; Hafner et al., 2019; Schrittwieser et al., 2020), which enables our method to scale to large domains. Third, since there are existing works examining how ER is affected by bootstrap return (Daley & Amato, 2019), by buffer or mini-batch size (Zhang & Sutton, 2017; Liu & Zou, 2017), and by number of environment steps taken per gradient step (Fu et al., 2019; van Hasselt et al., 2018; Fedus et al., 2020). It is worth studying the theoretical implications of those design choices and their effects on prioritized ER's efficacy, which may inspire improved sampling distributions.

Last, future efforts should also be made to theoretically interpret other sampling distributions induced by prioritizing samples, as our cubic objective explains only one version of the error-based prioritization. There are other

prioritization strategies, such as distribution location-based or reward-based prioritization (Lambert et al., 2020). It is interesting to explore whether these alternatives can also be formulated as surrogate objectives. Furthermore, the recent work by Fujimoto et al. (2020) establishes an equivalence between various prioritized sampling distributions and uniform sampling for different loss functions, which bears similarities to our Theorem 5. It is interesting to study if those general loss functions enjoy a similar benefit in terms of convergence rate as shown in our Theorem 6.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., and et al. TensorFlow: Large-scale machine learning on heterogeneous systems. *Software available from tensorflow.org*, 2015.

Achlioptas, D. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, pp. 671–687, 2003.

Adam, S., Busoniu, L., and Babuska, R. Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics*, pp. 201–212, 2012.

Ailon, N. and Chazelle, B. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. *ACM Symposium on Theory of Computing*, 2006.

Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. *ACM Symposium on Theory of Computing*, 1996.

Amari, S.-I. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

Amari, S.-I. and Douglas, S. C. Why natural gradient? *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1213–1216, 1998.

Anschel, O., Baram, N., and Shimkin, N. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. *International Conference on Machine Learning*, pp. 176–185, 2017.

Aubry, J.-M. and Jaffard, S. Random Wavelet Series. *Communications in Mathematical Physics*, 2002.

Avila Pires, B. and Szepesvari, C. Statistical linear estimation with penalized estimators: an application to reinforcement learning. *International Conference on Machine Learning*, 2012.

Balcan, M.-F., Blum, A., and Vempala, S. Kernels as features: On kernels, margins, and low-dimensional mappings. *Machine Learning*, 2006.

Bellemare, M., Veness, J., and Bowling, M. Sketch-Based Linear Value Function Approximation. *Advances in Neural Information Processing Systems*, 2012.

Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. *International Conference on Neural Information Processing Systems*, pp. 1479–1487, 2016.

Bengio, Y. Using a financial training criterion rather than a prediction criterion. *International Journal of Neural Systems*, pp. 433–443, 1997.

Bertsekas, D. *Dynamic Programming and Optimal Control*. Athena Scientific Press, 2007.

Bingham, E. and Mannila, H. Random projection in dimensionality reduction: Applications to image and text data. *International Conference on Knowledge Discovery and Data Mining*, pp. 245–250, 2001.

Bordes, A., Bottou, L., and Gallinari, P. SGD-QN: Careful quasi-Newton stochastic gradient descent. *Journal of Machine Learning Research*, 2009.

Borkar, V. S. and Mitter, S. K. A strong approximation theorem for stochastic recursive algorithms. *Journal of Optimization Theory and Applications*, 1999.

Boutsidis, C. and Woodruff, D. P. Communication-optimal distributed principal component analysis in the column-partition model. *arXiv:1504.06729*, 2015.

Boyan, J. and Moore, A. W. Generalization in Reinforcement Learning: Safely Approximating the Value Function. *Advances in Neural Information Processing Systems*, 1995a.

Boyan, J. A. Least-squares temporal difference learning. *International Conference on Machine Learning*, 1999.

Boyan, J. A. and Moore, A. W. Generalization in reinforcement learning: Safely approximating the value function. *Advances in Neural Information Processing Systems*, 1995b.

Boyd, S. and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, 2004.

Bradtke, S. J. and Barto, A. G. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 1996.

Brand, M. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and its Applications*, 2006.

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

Broyden, C. G. *Quasi-Newton Methods*. Numerical Methods for Unconstrained Optimization. Academic Press, London, 1972.

Chandak, Y., Theocharous, G., Kostas, J., Jordan, S., and Thomas, P. Learning action representations for reinforcement learning. *International Conference on Machine Learning*, pp. 941–950, 2019.

Charikar, M., Chen, K., and Farach-Colton, M. Finding frequent items in data streams. *Theoretical Computer Science*, 2002.

Chiang, T.-S., Hwang, C.-R., and Sheu, S. J. Diffusion for global optimization in $\mathbb{R}^n$. *SIAM Journal on Control and Optimization*, pp. 737–753, 1987.

Corneil, D. S., Gerstner, W., and Brea, J. Efficient model-based deep reinforcement learning with variational state tabulation. *International Conference on Machine Learning*, pp. 1049–1058, 2018.

Dabney, W. and Thomas, P. S. Natural Temporal Difference Learning. *AAAI Conference on Artificial Intelligence*, 2014.

Dai, B., Shaw, A., Li, L., Xiao, L., He, N., Liu, Z., Chen, J., and Song, L. SBEED: Convergent reinforcement learning with nonlinear function approximation. *International Conference on Machine Learning*, pp. 1125–1134, 2018.

Dai, B., Nachum, O., Chow, Y., Li, L., Szepesvari, C., and Schuurmans, D. Coindice: Off-policy confidence interval estimation. *Advances in Neural Information Processing Systems*, pp. 9398–9411, 2020.

Daley, B. and Amato, C. Reconciling lambda-returns with experience replay. *Advances in Neural Information Processing Systems*, pp. 1133–1142, 2019.

Dann, C., Neumann, G., and Peters, J. Policy evaluation with temporal differences: a survey and comparison. *Journal of Machine Learning Research*, 2014.

Daw, N. D. Model-based reinforcement learning as cognitive search: Neurocomputational theories. *Cognitive search: Evolution, algorithms and the brain*, 2012.

Degris, T., Pilarski, P. M., and Sutton, R. S. Model-free reinforcement learning with continuous action in practice. *American Control Conference*, 2012a.

Degris, T., White, M., and Sutton, R. S. Off-policy actor-critic. *International Conference on Machine Learning*, pp. 179–186, 2012b.

Devraj, A. M. and Meyn, S. Zap q-learning. *Advances in Neural Information Processing Systems*, 2017.

Du, Y. and Mordatch, I. Implicit generation and modeling with energy based models. *Advances in Neural Information Processing Systems*, 2019.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, pp. 2121–2159, 2011.

Durmus, A. and Moulines, E. Nonasymptotic convergence analysis for the unadjusted Langevin algorithm. *The Annals of Applied Probability*, pp. 1551–1587, 2017.

Eckart, C. and Young, G. The approximation of one matrix by another of lower rank. *Psychometrika*, 1936.

Fan, J., Wang, Z., Xie, Y., and Yang, Z. A theoretical analysis of deep q-learning. *Conference on Learning for Dynamics and Control*, pp. 486–489, 2020.

Farahmand, A.-m. Iterative value-aware model learning. *Advances in Neural Information Processing Systems*, pp. 9072–9083, 2018.

Farahmand, A.-M., Barreto, A., and Nikovski, D. Value-aware loss function for model-based reinforcement learning. *International Conference on Artificial Intelligence and Statistics*, pp. 1486–1494, 2017.

Fard, M. M., Grinberg, Y., Pineau, J., and Precup, D. Compressed least-squares regression on sparse spaces. *AAAI Conference on Artificial Intelligence*, 2012.

Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., and Dabney, W. Revisiting fundamentals of experience replay. *International Conference on Machine Learning*, pp. 3061–3071, 2020.

French, R. M. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, pp. 128–135, 1999.

Freund, Y., Dasgupta, S., Kabra, M., and Verma, N. Learning the structure of manifolds using random projections. *Advances in Neural Information Processing Systems*, 2008.

Fu, J., Kumar, A., Soh, M., and Levine, S. Diagnosing bottlenecks in deep q-learning algorithms. *International Conference on Machine Learning*, pp. 2021–2030, 2019.

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

Fujimoto, S., Meger, D., and Precup, D. An equivalence between loss functions and non-uniform sampling in experience replay. *Advances in Neural Information Processing Systems*, 2020.

Gehring, C., Pan, Y., and White, M. Incremental Truncated LSTD. *International Joint Conference on Artificial Intelligence*, 2016.

Geramifard, A. and Bowling, M. Incremental least-squares temporal difference learning. *AAAI Conference on Artificial Intelligence*, 2006.

Geramifard, A., Bowling, M., and Zinkevich, M. iLSTD: Eligibility traces and convergence analysis. *Advances in Neural Information Processing Systems*, 2007.

Ghashami, M., Desai, A., and Phillips, J. M. Improved practical matrix sketching with guarantees. *European Symposium on Algorithms*, 2014.

Ghavamzadeh, M. and Lazaric, A. Finite-sample analysis of Lasso-TD. *International Conference on Machine Learning*, 2011.

Ghavamzadeh, M., Lazaric, A., Maillard, O. A., and Munos, R. LSTD with random projections. *Advances in Neural Information Processing Systems*, 2010.

Gilbert, A. and Indyk, P. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 2010.

Givchi, A. and Palhang, M. Quasi newton temporal difference learning. *Asian Conference on Machine Learning*, 2014.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. *International Conference on Artificial Intelligence and Statistics*, 2010.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *International Conference on Learning Representations*, 2015.

Gower, R. M. and Richtárik, P. Randomized Iterative Methods for Linear Systems. *arXiv:1506.03296*, 2015.

Groetsch, C. W. *The Theory of Tikhonov Regularization for Fredholm Equations of the First Kind*. Pitman Advanced Publishing Program, 1984.

Gu, S., Lillicrap, T. P., Sutskever, I., and Levine, S. Continuous Deep Q-Learning with Model-based Acceleration. *International Conference on Machine Learning*, pp. 2829–2838, 2016.

Ha, David and Schmidhuber, Jürgen. Recurrent world models facilitate policy evolution. *Advances in Neural Information Processing Systems*, pp. 2450–2462, 2018.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning*, pp. 1861–1870, 2018.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. *International Conference on Machine Learning*, pp. 2555–2565, 2019.

Hamilton, W. L., Fard, M. M., and Pineau, J. Efficient learning and planning with compressed predictive states. *Journal of Machine Learning Research*, 2014.

Hanna, J., Niekum, S., and Stone, P. Importance sampling policy evaluation with an estimated behavior policy. *International Conference on Machine Learning*, pp. 2605–2613, 2019.

Hansen, P. C. The truncated SVD as a method for regularization. *BIT Numerical Mathematics*, 1986.

Hansen, P. C. The discrete picard condition for discrete ill-posed problems. *BIT Numerical Mathematics*, 1990.

Hasselt, H. Double q-learning. *Advances in Neural Information Processing Systems*, 2010.

Holland, G. Z., Talvitie, E., and Bowling, M. The effect of planning shape on dyna-style planning in high-dimensional state spaces. *CoRR*, abs/1806.01825, 2018.

Huber, P. J. Robust estimation of a location parameter. *Annals of Mathematical Statistics*, pp. 73–101, 1964.

Imani, E., Graves, E., and White, M. An off-policy policy gradient theorem using emphatic weightings. *International Conference on Neural Information Processing Systems*, pp. 96–106, 2018.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, pp. 12519–12530, 2019.

Jiang, D. R., Pham, T. V., Powell, W. B., Salas, D. F., and Scott, W. R. A comparison of approximate dynamic programming techniques on benchmark energy storage problems: Does anything work? *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2014.

Jiang, H. A new perspective on machine learning: How to do perfect supervised learning. *CoRR:abs/1901.02046*, 2019.

Jiang, N. and Li, L. Doubly robust off-policy value evaluation for reinforcement learning. *International Conference on Machine Learning*, pp. 652–661, 2016.

Johnson, W. B. and Lindenstrauss, J. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics*, 1984.

Kaiser, Ł., Babaeizadeh, M., Miłos, P., Osiński, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Mohiuddin, A., Sepassi, R., Tucker, G., and Michalewski, H. Model based reinforcement learning for atari. *International Conference on Learning Representations*, 2020.

Kemker, R., McClure, M., Abitino, A., Hayes, T. L., and Kanan, C. Measuring catastrophic forgetting in neural networks. *AAAI conference on Artificial Intelligence*, 2018.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, pp. 1179–1191, 2020.

Kushner, H. and Yin, G. *Stochastic Approximation Algorithms and Recursive Algorithms and Applications*. Springer, 2003.

Lambert, N., Amos, B., Yadan, O., and Calandra, R. Objective mismatch in model-based reinforcement learning. *CoRR abs/2002.04523*, 2020.

Lan, Q., Pan, Y., Fyshe, A., and White, M. Maxmin Q-learning: controlling the estimation bias of Q-learning. *International Conference on Learning Representations*, 2020.

Lattimore, T. and Szepesvari, C. *Bandit Algorithms*. Cambridge University Press, 2020.

Lazaric, A., Ghavamzadeh, M., and Munos, R. Finite sample analysis of LSTD. *International Conference on Machine Learning*, 2010.

Le, L., Kumaraswamy, R., and White, M. Learning sparse representations in reinforcement learning with sparse coding. *International Joint Conference on Artificial Intelligence*, pp. 2067–2073, 2017.

Lee, D., Defourny, B., and Powell, W. B. Bias-corrected q-learning to control max-operator bias in q-learning. *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pp. 93–99, 2013.

Leurent, E. An environment for autonomous driving decision-making. https://github.com/eleurent/highway-env, 2018.

Li, W. and Todorov, E. Iterative linear quadratic regulator design for nonlinear biological movement systems. *International Conference on Informatics in Control, Automatin and Robotics*, pp. 222–229, 2004.

Liang, Y., Machado, M. C., Talvitie, E., and Bowling, M. State of the art control of atari games using shallow reinforcement learning. *International Conference on Autonomous Agents & Multiagent Systems*, pp. 485–493, 2016.

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

Lim, S., Joseph, A., Le, L., Pan, Y., and White, M. Actor-expert: A framework for using action-value methods in continuous action spaces. *CoRR abs/1810.09103*, 2018.

Lin, L.-J. Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, 1992.

Linke, C., Ady, N. M., White, M., Degris, T., and White, A. Adapting behaviour via intrinsic reward: A survey and empirical study. *Journal of Artificial Intelligence Research*, 2019.

Liu, Q., Li, L., Tang, Z., and Zhou, D. Breaking the curse of horizon: Infinite-horizon off-policy estimation. *International Conference on Neural Information Processing Systems*, pp. 5361–5371, 2018.

Liu, R. and Zou, J. The effects of memory replay in reinforcement learning. *Conference on Communication, Control, and Computing*, 2017.

Liu, V., Kumaraswamy, R., Le, L., and White, M. The utility of sparse representations for control in reinforcement learning. *AAAI Conference on Artificial Intelligence*, pp. 4384–4391, 2019.

Luo, H., Agarwal, A., Cesa-Bianchi, N., and Langford, J. Efficient Second Order Online Learning by Sketching. *Advances in Neural Information Processing Systems*, 2016.

Maei, H. *Gradient Temporal-Difference Learning Algorithms*. PhD thesis, University of Alberta, 2011.

Maei, H. R. Convergent actor-critic algorithms under off-policy training and function approximation. *CoRR abs/1802.07842*, 2018.

Mahadevan, S., Liu, B., Thomas, P. S., Dabney, W., Giguere, S., Jacek, N., Gemp, I., and Liu, J. Proximal reinforcement learning: A new theory of sequential decision making in primal-dual spaces. *CoRR abs/1405.6757*, 2014.

Mahmood, A. R. and Sutton, R. Representation search through generate and test. *AAAI Workshop on Learning Rich Representations from Low-Level Sensors*, 2013.

Maillard, O.-A. and Munos, R. Linear Regression With Random Projections. *Journal of Machine Learning Research*, 2012.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation*, pp. 109–165, 1989.

Mei, J., Pan, Y., White, M., Farahmand, A., and Yao, H. Beyond prioritized replay: Sampling states in model-based RL via simulated priorities. *CoRR abs/2007.09569*, 2020.

Meyer, D., Degenne, R., Omrane, A., and Shen, H. Accelerated gradient temporal difference learning algorithms. *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2014.

Mirsky, L. Symmetric gauge functions and unitarily invariant norms. *Quartely Journal Of Mathetmatics*, 1960.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 2015.

Mokhtari, A. and Ribeiro, A. RES: Regularized stochastic BFGS algorithm. *IEEE Transactions on Signal Processing*, 2014.

Moore, A. W. and Atkeson, C. G. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, pp. 103–130, 1993.

Needell, D., Ward, R., and Srebro, N. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. *Advances in Neural Information Processing Systems*, 2014.

Nocedal, J. and Wright, S. J. *Numerical Optimization*. Springer, second edition, 2006.

Novati, G. and Koumoutsakos, P. Remember and forget for experience replay. *International Conference on Machine Learning*, pp. 4851–4860, 2019.

Oh, Y., Lee, K., Shin, J., Yang, E., and Hwang, S. J. Learning to sample with local and global contexts in experience replay buffer. *International Conference on Learning Representations*, 2021.

Pan, Y., Azer, E. S., and White, M. Effective sketching methods for value function approximation. *Uncertainty in Artificial Intelligence*, 2017a.

Pan, Y., White, A., and White, M. Accelerated Gradient Temporal Difference Learning. *AAAI Conference on Artifical Intelligence*, 2017b.

Pan, Y., Farahmand, A.-m., White, M., Nabi, S., Grover, P., and Nikovski, D. Reinforcement learning with function-valued action spaces for partial differential equation control. *International Conference on Machine Learning*, 2018.

Pan, Y., Zaheer, M., White, A., Patterson, A., and White, M. Organizing experience: a deeper look at replay mechanisms for sample-based planning in continuous state domains. *International Joint Conference on Artificial Intelligence*, pp. 4794–4800, 2018.

Pan, Y., Yao, H., Farahmand, A.-m., and White, M. Hill climbing on value estimates for search-control in Dyna. *International Joint Conference on Artificial Intelligence*, 2019.

Pan, Y., Imani, E., Farahmand, A.-m., and White, M. An implicit function learning approach for parametric modal regression. *Advances in Neural Information Processing Systems*, pp. 11442–11452, 2020a.

Pan, Y., Mei, J., and Farahmand, A.-m. Frequency-based search-control in Dyna. *International Conference on Learning Representations*, 2020b.

Pan, Y., Banman, K., and White, M. Fuzzy tiling activations: A simple approach to learning sparse representations online. *International Conference on Learning Representations*, 2021.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. *International Conference on Machine Learning*, pp. 2778–2787, 2017.

Peng, J. and Williams, R. J. Efficient learning and planning within the Dyna framework. *Adaptive behavior*, 1993.

Powell, W. B. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley-Interscience, 2007.

Powell, W. B. Reinforcement learning and stochastic optimization: A unified framework for sequential decisions. *Wiley*, 2021.

Prashanth, L. A., Korda, N., and Munos, R. Fast LSTD using stochastic approximation: Finite time analysis and application to traffic control. *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2013.

Precup, D., Sutton, R. S., and Singh, S. P. Eligibility traces for off-policy policy evaluation. *International Conference on Machine Learning*, pp. 759–766, 2000.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems*, 2007.

Rahimi, A. and Recht, B. Uniform approximation of functions with random bases. *Annual Allerton Conference on Communication, Control, and Computing*, 2008.

Reddi, S. J., Kale, S., and Kumar, S. On the convergence of Adam and beyond. *International Conference on Learning Representations*, 2018.

Riedmiller, M. Neural fitted Q iteration – first experiences with a data efficient neural reinforcement learning method. *Machine Learning*, pp. 317–328, 2005.

Roberts, Gareth O.and Tweedie, R. L. Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli*, pp. 341–363, 1996.

Salas, D. F. and Powell, W. B. Benchmarking a Scalable Approximate Dynamic Programming Algorithm for Stochastic Control of Multidimensional Energy Storage Problems. *INFORMS Journal on Computing*, 2013.

Sato, I. and Nakagawa, H. Approximation analysis of stochastic gradient Langevin dynamics by using Fokker-Planck equation and Ito process. *International Conference on Machine Learning*, 2014.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized Experience Replay. *International Conference on Learning Representations*, 2016.

Schlegel, M., Pan, Y., Chen, J., and White, M. Adapting kernel representations online using submodular maximization. *International Conference on Machine Learning*, pp. 3037–3046, 2017.

Schraudolph, N., Yu, J., and Günter, S. A stochastic quasi-Newton method for online convex optimization. *International Conference on Artificial Intelligence and Statistics*, 2007.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, pp. 604–609, 2020.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. *International Conference on Machine Learning*, pp. 1889–1897, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR abs/1707.06347*, 2017.

Shi, X., Wei, Y., and Zhang, W. Convergence of general nonstationary iterative methods for solving singular linear equations. *SIAM Journal on Matrix Analysis and Applications*, 2011.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. *International Conference on Machine Learning*, pp. I–387–I–395, 2014.

Smale, S. and Zhou, D.-X. Shannon sampling and function reconstruction from point values. *Bulletin of the American Mathematical Society*, 41:279–305, 2004.

Smale, S. and Zhou, D.-X. Shannon sampling II: Connections to learning theory. *Applied and Computational Harmonic Analysis*, 19(3):285–302, 2005.

Song, Y. and Ermon, S. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 2019.

Song, Y. and Ermon, S. Improved techniques for training score-based generative models. *Advances in Neural Information Processing Systems*, 2020.

Stadie, B. C., Levine, S., and Abbeel, P. Incentivizing exploration in reinforcement learning with deep predictive models. *ArXiv*, abs/1507.00814, 2015.

Stover, C. Invertible matrix theorem. *MathWorld–A Wolfram Web Resource*, 2006.

Strehl, A. L., Li, L., and Littman, M. L. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, pp. 2413–2444, 2009.

Sun, P., Zhou, W., and Li, H. Attentive experience replay. *AAAI Conference on Artificial Intelligence*, pp. 5900–5907, 2020.

Sutton, R., Maei, H., Precup, D., and Bhatnagar, S. Fast gradient-descent methods for temporal-difference learning with linear function approximation. *International Conference on Machine Learning*, 2009.

Sutton, R. S. Learning to predict by the methods of temporal differences. *Machine Learning*, pp. 9–44, 1988.

Sutton, R. S. Integrated modeling and control based on reinforcement learning and dynamic programming. *Advances in Neural Information Processing Systems*, 1991.

Sutton, R. S. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, 1996.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

Sutton, R. S. and Whitehead, S. Online learning with random representations. *International Conference on Machine Learning*, 1993.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *International Conference on Neural Information Processing Systems*, pp. 1057–1063, 1999.

Sutton, R. S., Szepesvári, C., Geramifard, A., and Bowling, M. Dyna-style planning with linear function approximation and prioritized sweeping. *Conference on Uncertainty in Artificial Intelligence*, pp. 528–536, 2008.

Sutton, R. S., Mahmood, A. R., and White, M. An emphatic approach to the problem of off-policy temporal-difference learning. *Journal of Machine Learning Research*, 2016.

Szepesvari, C. *Algorithms for Reinforcement Learning*. Morgan & Claypool Publishers, 2010.

Szita, I. and Lőrincz, A. The many faces of optimism: A unifying approach. *International Conference on Machine learning*, pp. 1048–1055, 2008.

Tagorti, M. and Scherrer, B. On the Rate of Convergence and Error Bounds for LSTD(λ). *International Conference on Machine Learning*, 2015.

Talvitie, E. Model regularization for stable sample rollouts. *Uncertainty in Artificial Intelligence*, 2014.

Talvitie, E. Self-Correcting Models for Model-Based Reinforcement Learning. *AAAI Conference on Artificial Intelligence*, 2017.

Tanaka, G., Yamane, T., Héroux, J. B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., and Hirose, A. Recent advances in physical reservoir computing: A review. *Neural Networks*, pp. 100–123, 2019.

Teh, Y., Thiery, A. H., and Vollmer, S. J. Consistency and fluctuations for stochastic gradient langevin dynamics. *Journal of Machine Learning Research*, pp. 7:1–7:33, 2016.

Thomas, P. S. and Brunskill, E. Data-efficient off-policy policy evaluation for reinforcement learning. *International Conference on Machine Learning*, pp. 2139–2148, 2016.

Thomas, P. S., Niekum, S., Theocharous, G., and Konidaris, G. Policy evaluation using the omega-return. *Advances in Neural Information Processing Systems*, 2015a.

Thomas, P. S., Theocharous, G., and Ghavamzadeh, M. High confidence off-policy evaluation. *AAAI Conference on Artificial Intelligence*, pp. 3000–3006, 2015b.

Thomas, P. S., Silva, B. C., Dann, C., and Brunskill, E. Energetic natural gradient descent. *International Conference on Machine Learning*, pp. 2887–2895, 2016.

Thrun, S. and Schwartz, A. Issues in Using Function Approximation for Reinforcement Learning. *Fourth Connectionist Models Summer School*, 1993.

Tieleman, T. and Hinton, G. Rmsprop: divide the gradient by a running average of its recent magnitude. *Coursera: Neural Networks for Machine Learning*, 2012.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. *International Conference on Intelligent Robots and Systems*, 2012.

Tsitsiklis, J. and Van Roy, B. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 1997.

Tsitsiklis, J. N. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, pp. 185–202, 1994.

Tulabandhula, T. and Rudin, C. Machine learning with operational costs. *Journal of Machine Learning Research*, pp. 1989–2028, 2013.

van Hasselt, H., Mahmood, A. R., and Sutton, R. Off-policy TD (λ) with a true online equivalence. *Conference on Uncertainty in Artificial Intelligence*, 2014.

van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., and Modayil, J. Deep reinforcement learning and the deadly triad. *Deep Reinforcement Learning Workshop at Advances in Neural Information Processing Systems*, 2018.

van Hasselt, H., Madjiheurem, S., Hessel, M., Silver, D., Barreto, A., and Borsa, D. Expected eligibility traces. *AAAI Conference on Artificial Intelligence*, pp. 9997–10005, 2021.

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

van Seijen, H. and Sutton, R. A deeper look at planning as learning from replay. *International Conference on Machine Learning*, 2015.

van Seijen, H., Mahmood, R. A., Pilarski, P. M., Machado, M. C., and Sutton, R. S. True Online Temporal-Difference Learning. *Journal of Machine Learning Research*, 2016.

Wahlström, N., Schön, T. B., and Deisenroth, M. P. From pixels to torques: Policy learning with deep dynamical models. *Deep Learning Workshop at International Conference on Machine Learning*, 2015.

Wan, Y., Zaheer, M., White, A., White, M., and Sutton, R. S. Planning with expectation models. *International Joint Conference on Artificial Intelligence*, pp. 3649–3655, 2019.

Wang, L., Yang, Y., Min, M. R., and Chakradhar, S. Accelerating deep neural network training with inconsistent stochastic gradient descent. *Neural networks : the official journal of the International Neural Network Society*, 2017.

Wang, M. and Bertsekas, D. P. On the convergence of simulation-based iterative methods for solving singular linear systems. *Stochastic Systems*, 2013.

Wang, S. A Practical Guide to Randomized Matrix Computations with MATLAB Implementations. *arXiv:1505.07570*, 2015.

Watkins, C. J. C. H. and Dayan, P. Q-learning. *Machine Learning*, pp. 279–292, 1992.

Welling, M. and Teh, Y. W. Bayesian learning via stochastic gradient Langevin dynamics. *International Conference on Machine Learning*, pp. 681–688, 2011.

Wen, J., Dai, B., Li, L., and Schuurmans, D. Batch stationary distribution estimation. *International Conference on Machine Learning*, pp. 10203–10213, 2020.

White, A. M. and White, M. Investigating practical, linear temporal difference learning. *International Conference on Autonomous Agents and Multiagent Systems*, 2016.

White, M. Unifying task specification in reinforcement learning. *International Conference on Machine Learning*, 2017.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, pp. 229–256, 1992.

Woodruff, D. P. Sketching as a tool for numerical linear algebra. *arXiv:1411.4357*, 2014.

Xu, P., Chen, J., Zou, D., and Gu, Q. Global convergence of Langevin dynamics based algorithms for nonconvex optimization. *Advances in Neural Information Processing Systems*, 2018.

Yu, H. On convergence of emphatic temporal-difference learning. *Annual Conference on Learning Theory*, 2015.

Zayed, A. *Advances in Shannon's Sampling Theory*. Taylor & Francis, 1993.

Zha, D., Lai, K.-H., Zhou, K., and Hu, X. Experience replay optimization. *International Joint Conference on Artificial Intelligence*, pp. 4243–4249, 2019.

Zhang, R., Dai, B., Li, L., and Schuurmans, D. Gendice: Generalized offline estimation of stationary values. *International Conference on Learning Representations*, 2020.

Zhang, S. and Sutton, R. S. A Deeper Look at Experience Replay. *Deep Reinforcement Learning Symposium at Advances in Neural Information Processing Systems*, 2017.

Zhang, S., Boehmer, W., and Whiteson, S. Generalized off-policy actor-critic. *Advances in Neural Information Processing Systems*, 2019.

Zhang, S., Yao, H., and Whiteson, S. Breaking the deadly triad with a target network. *International Conference on Machine Learning*, 2021.

Zhang, Z., Pan, Z., and Kochenderfer, M. J. Weighted Double Q-learning. *International Joint Conference on Artificial Intelligence*, pp. 3455–3461, 2017.

Zhao, P. and Zhang, T. Stochastic optimization with importance sampling for regularized loss minimization. *International Conference on Machine Learning*, pp. 1–9, 2015.

Appendix A

ATD Algorithms

A.1 ATD with SVD

This section includes relevant content for the ATD algorithm with incremental truncated singular value decomposition. First, we include a general convergence proof in Section A.1.1. Second, we present algorithmic details for implementing our algorithm in Section A.1.2. Third, in Section A.1.3, we discuss additional ways to interpolate between the linear TD updating rule and quadratic LSTD updating rule. Finally, we summarize all implementation details, including testing domains, parameter sweeps, algorithm evaluation methods, and present additional experimental results in Section A.1.4.

A.1.1 Convergence Proof

For the more general setting, where m can also equal $\mathbf{D}_\mu$, we redefine the rank-k approximation. We say the rank-k approximation $\hat{\mathbf{A}}$ to $\mathbf{A}$ is composed of eigenvalues $\{\lambda_{i_1}, \ldots, \lambda_{i_k}\} \subseteq \{\lambda_1, \ldots, \lambda_d\}$ if $\hat{\mathbf{A}} = \mathbf{Q}\mathbf{\Lambda}_k\mathbf{Q}^{-1}$ for diagonal $\mathbf{\Lambda}_k \in \mathbb{R}^{d\times d}$, $\mathbf{\Lambda}(i_j, i_j) = \lambda_{i_j}$ for $j = 1, \ldots, k$, and zero otherwise.

Theorem 7. *Under Assumptions 1 and 2, let $\hat{\mathbf{A}}$ be the rank-k approximation composed of eigenvalues $\{\lambda_{i_1}, \ldots, \lambda_{i_k}\} \subseteq \{\lambda_1, \ldots, \lambda_d\}$. If $\lambda_d \geq 0$ or $\{\lambda_{i_1}, \ldots, \lambda_{i_k}\}$ contains all the negative eigenvalues in $\{\lambda_1, \ldots, \lambda_d\}$, then the expected updating rule in (3.11) converges to the fixed-point $\mathbf{w}^\star = \mathbf{A}^\dagger\mathbf{b}$.*

Proof. We use a general result about stationary iterative methods (Shi et al., 2011), which applies to the case where $\mathbf{A}$ is not full rank. Shi et al. (2011,

Theorem 1.1) states that given a singular and consistent linear system $\mathbf{A}\mathbf{w} = \mathbf{b}$ where $\mathbf{b}$ is in the range of $\mathbf{A}$, the stationary iteration with $\mathbf{w}_0 \in \mathbb{R}^d$ for $t = 1, 2, \ldots$

$$\mathbf{w}_i = (\mathbf{I} - \mathbf{B}\mathbf{A})\mathbf{w}_{t-1} + \mathbf{B}\mathbf{b} \tag{4.2}$$

converges to the solution $\mathbf{w} = \mathbf{A}^\dagger\mathbf{b}$ if and only if the following three conditions are satisfied.

Condition **I**: the eigenvalues of $\mathbf{I} - \mathbf{B}\mathbf{A}$ are equal to 1 or have absolute value strictly less than 1.

Condition **II**: $\text{rank}(\mathbf{B}\mathbf{A}) = \text{rank}[(\mathbf{B}\mathbf{A})^2]$.

Condition **III**: the null space $\mathcal{N}(\mathbf{B}\mathbf{A}) = \mathcal{N}(\mathbf{A})$.

We verify these conditions to prove the result. First, because we are using the projected Bellman error, we know that $\mathbf{b}$ is in the range of $\mathbf{A}$ and the system is consistent: there exists $\mathbf{w}$ s.t. $\mathbf{A}\mathbf{w} = \mathbf{b}$.

To rewrite our updating rule (3.11) to be expressible in terms of (4.2), let $\mathbf{B} = \alpha\hat{\mathbf{A}}^\dagger + \eta\mathbf{I}$, giving

$$\begin{aligned}
\mathbf{B}\mathbf{A} &= \alpha\hat{\mathbf{A}}^\dagger\mathbf{A} + \eta\mathbf{A} \\
&= \alpha\mathbf{Q}\mathbf{\Lambda}_k^\dagger\mathbf{Q}^{-1}\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1} + \eta\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1} \\
&= \alpha\mathbf{Q}\mathbf{I}_k\mathbf{Q}^{-1} + \eta\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1} \\
&= \mathbf{Q}(\alpha\mathbf{I}_k + \eta\mathbf{\Lambda})\mathbf{Q}^{-1}
\end{aligned} \tag{4.3}$$

where $\mathbf{I}_k$ is a diagonal matrix with the indices $i_1, \ldots, i_k$ set to 1, and the rest zero.

Proof for condition I. Using (4.3), $\mathbf{I} - \mathbf{B}\mathbf{A} = \mathbf{Q}(\mathbf{I} - \alpha\mathbf{I}_k - \eta\mathbf{\Lambda})\mathbf{Q}^{-1}$. To bound the maximum absolute value in the diagonal matrix $\mathbf{I} - \alpha\mathbf{I}_k - \eta\mathbf{\Lambda}$, we consider eigenvalue λ_j in $\mathbf{\Lambda}$, and address three cases.

Case 1: $j \in \{i_1, \ldots, i_k\}$, $\lambda_j \geq 0$:

$$\begin{aligned}
&|1-\alpha-\eta\lambda_j| && \triangleright \text{ for } 0 < \eta < \max\left(\frac{2-\alpha}{\lambda_1}, \frac{\alpha}{\lambda_1}\right) \\
&< \max(|1-\alpha|, |1-\alpha-(2-\alpha)|, |1-\alpha-\alpha|) \\
&= \max(|1-\alpha|, 1, 1) < 1 && \triangleright \text{ because } \alpha \in (0,2).
\end{aligned}$$

Case 2: $j \in \{i_1, \ldots, i_k\}$, $\lambda_j < 0$: $|1-\alpha-\eta\lambda_i| = |1-\alpha+\eta|\lambda_i|| < 1$ if $0 \leq 1-\alpha+\eta|\lambda_i| < 1 \implies \eta < \alpha/|\lambda_i|$.

Case 3: $j \notin \{i_1, \ldots, i_k\}$. For this case, $\lambda_j \geq 0$, by assumption, as $\{i_1, \ldots, i_k\}$ contains the indices for all negative eigenvalues of $\mathbf{A}$. So $|1-\eta\lambda_i| < 1$ if $0 < \eta < 2/\lambda_i$.

All three cases are satisfied by the assumed $\alpha \in (0,2)$ and $\eta \leq \lambda_{\max}^{-1} \max(2-\alpha, \alpha)$. Therefore, the absolute value of the eigenvalues of $\mathbf{I} - \mathbf{BA}$ are all less than 1, so the first condition holds.

Proof for condition II. $(\mathbf{BA})^2$ does not change the number of positive eigenvalues, so the rank is unchanged.

$$\begin{aligned}
\mathbf{BA} &= \mathbf{Q}(\alpha\mathbf{I}_k + \eta\mathbf{\Lambda})\mathbf{Q}^{-1}, \\
(\mathbf{BA})^2 &= \mathbf{Q}(\alpha\mathbf{I}_k + \eta\mathbf{\Lambda})\mathbf{Q}^{-1}\mathbf{Q}(\alpha\mathbf{I}_k + \eta\mathbf{\Lambda})\mathbf{Q}^{-1}, \\
&= \mathbf{Q}(\alpha\mathbf{I}_k + \eta\mathbf{\Lambda})^2\mathbf{Q}^{-1}.
\end{aligned}$$

Proof for condition III. To show that the nullspaces of $\mathbf{BA}$ and $\mathbf{A}$ are equal, it is sufficient to prove $\mathbf{BAw} = \mathbf{0}$ if and only if $\mathbf{Aw} = \mathbf{0}$. Because $\mathbf{B} = \mathbf{Q}(\alpha\mathbf{\Lambda}_k + \eta\mathbf{I})\mathbf{Q}^{-1}$, we know that $\mathbf{B}$ is invertible as long as $\alpha \neq -\eta\lambda_j$. Because $\eta > 0$, this is clearly true for $\lambda_j \geq 0$ and also true for $\lambda_j < 0$ because η is strictly less than $\alpha/|\lambda_j|$. For any $\mathbf{w} \in \text{nullspace}(\mathbf{A})$, we get $\mathbf{BAw} = \mathbf{B0} = \mathbf{0}$, and so $\mathbf{w} \in \text{nullspace}(\mathbf{BA})$. For any $\mathbf{w} \in \text{nullspace}(\mathbf{BA})$, we get $\mathbf{BAw} = \mathbf{0} \implies \mathbf{Aw} = \mathbf{B}^{-1}\mathbf{0} = \mathbf{0}$, and so $\mathbf{w} \in \text{nullspace}(\mathbf{A})$, completing the proof. □

With $k = d$, the update is a gradient descent update on the MSPBE, and so will converge even under off-policy sampling. As $k \ll d$, the gradient is

only approximate, and theoretical results about (stochastic) gradient descent no longer apply. For this reason, we use the iterative update analysis above to understand convergence properties. Iterative updates for the full expected update, with preconditioners, have been studied in reinforcement learning (c.f. (Wang & Bertsekas, 2013)); however, they typically analyzed different preconditioners, as they had no requirements for reducing computation below quadratic computation. For example, they consider a regularized preconditioner $\mathbf{B} = (\mathbf{A} + \eta\mathbf{I})^{-1}$, which is not compatible with an incremental singular value decomposition and to the best of our knowledge, current iterative eigenvalue decompositions require symmetric matrices.

The theorem is agnostic to what components of $\mathbf{A}$ are approximated by the rank-k matrix $\hat{\mathbf{A}}$. In general, a natural choice, particularly in on-policy learning or more generally with a positive definite $\mathbf{A}$, is to select the largest magnitude eigenvalues of $\mathbf{A}$, which contain the most significant information about the system and so are likely to give the most useful curvature information. However, $\hat{\mathbf{A}}$ could also potentially be chosen to obtain convergence for off-policy learning with $m = d_\mu$, where $\mathbf{A}$ is not necessarily positive semi-definite. This theorem indicates that if the rank k approximation $\hat{\mathbf{A}}$ contains the negative eigenvalues of $\mathbf{A}$, even if it does not contain the remaining information in $\mathbf{A}$. Then we obtain convergence under off-policy sampling. We can, of course, use the emphatic weighting more easily for off-policy learning, but if the weighting $m = d_\mu$ is desired rather than m_{ETD}, then carefully selecting $\hat{\mathbf{A}}$ for ATD enables that choice.

A.1.2 Algorithmic Details

In this section, we outline the full ATD(λ) algorithm. The algorithm here contains a slight generalization, with a specified ϵ. Using ϵ provides further improvement in sample complexity, without incurring any significant computational overhead. Instead of fulling stochastically sampling $\delta_t(\mathbf{w}_t)\mathbf{e}_t = r_{t+1}\mathbf{e}_t + (\gamma_{t+1}\mathbf{x}_{t+1}^\top\mathbf{w}_t - \mathbf{x}_t^\top\mathbf{w}_t)\mathbf{e}_t$, we could maintain an estimate of the expected value $\mathbf{b}_t = \mathbb{E}[r_{t+1}\mathbf{e}_t]$ and stochastically sample $(\gamma_{t+1}\mathbf{x}_{t+1}^\top\mathbf{w}_t - \mathbf{x}_t^\top\mathbf{w}_t)\mathbf{e}_t$. We cannot maintain a sample average of the second component efficiently (because

then we would have to store $\mathbf{A}_t$), but we can maintain the sample average $\mathbf{b}_t$ efficiently. In an effort to focus exploration on the basic ATD approach, we tested $\epsilon = 0$ in this work. Nevertheless, we include the more general algorithm to give more insight into the interpolation between TD and LSTD, discussed in the next section.

Putting it all together, we arrive at our proposed update scheme. Let $\hat{\mathbf{A}}$ be a rank k approximation to the true $\mathbf{A}$, with $\hat{\mathbf{A}}^\dagger$ the Moore-Penrose pseudo-inverse of $\hat{\mathbf{A}}$. Let $\eta > 0$ be a regularization parameter and $\epsilon > 0$ be an averaging parameter. Then the update is

$$\begin{aligned}&\mathbf{w}_{t+1} = \mathbf{w}_t + \\ &(\alpha_t \hat{\mathbf{A}}^\dagger + \eta \mathbf{I})(\epsilon \mathbf{b}_t + ((1-\epsilon) r_{t+1} + \mathbf{w}^\top \mathbf{x}_{t+1} - \mathbf{w}^\top \mathbf{x}_t)\mathbf{e}_t)\end{aligned}$$

with expected update

$$\mathbf{w}_{t+1} = \mathbf{w}_t + (\alpha_t \hat{\mathbf{A}}^\dagger + \eta \mathbf{I})(\mathbf{b} - \mathbf{A}\mathbf{w}_t) \tag{3.11}$$

for all $0 \leq \epsilon \leq 1$.

Algorithm 8 Conventional trace update for ATD

function TRACE_UPDATE($\mathbf{e}_{t-1}, \mathbf{x}_t, \gamma_t, \lambda_t$)
  return $\gamma_t \lambda_t \mathbf{e}_{t-1} + \mathbf{x}_t$

Algorithm 9 Emphatic trace update for ATD

▷ where $F_0 \leftarrow 0$, $M_0 \leftarrow 0$ is initialized globally, before executing the for loop in ATD(λ)

function EMPHATIC_TRACE_UPDATE($\mathbf{e}_{t-1}, \mathbf{x}_t, \gamma_t, \lambda_t$)
  $\rho_t \leftarrow \frac{\pi(s_t, a_t)}{\mu(s_t, a_t)}$ ▷ Where $\rho_t = 1$ in the on-policy case
  $F_t \leftarrow \rho_{t-1}\gamma_t F_{t-1} + i_t$ ▷ For uniform interest, $i_t = 1$
  $M_t \leftarrow \lambda_t i_t + (1 - \lambda_t) F_t$

  return $\rho_t(\gamma_t \lambda_t \mathbf{e}_{t-1} + M_t \mathbf{x}_t)$

A.1.3 Smoothly Interpolating between TD and LSTD

The choice of k allows ATD(λ) to smoothly vary between TD(λ) and an iterative form of LSTD(λ). One side of this equivalence is straightforward. When

$k = 0$, and $\epsilon = 0$, ATD(λ) reduces to the conventional TD(λ) algorithm because $\hat{\mathbf{A}}^\dagger = \mathbf{0}$ and so $\mathbf{w} \leftarrow \mathbf{w} + \eta\delta\mathbf{e}$, where $\eta = \alpha$ because $\lambda_1 = 0$. On the other end of the spectrum, the equivalence of ATD(λ) and LSTD(λ) is more involved. Consider an iterative version of the LSTD(λ) that uses a exponentially weighted moving average to update $\mathbf{w}$ towards $\mathbf{A}_t^\dagger\mathbf{b}$ on each step:

$$\mathbf{w}_{t+1} = (1 - \alpha_t)\mathbf{w}_t + \alpha_t\mathbf{A}_t^\dagger\mathbf{b}. \tag{A.1}$$

Now, when $k = d, \epsilon = 1$, and $\eta = 0$, ATD(λ) can be shown to perform approximately the same updates as *exponentially-weighted LSTD(λ)*. For $k < d$, however, this averaging would introduce bias because $\hat{\mathbf{A}} \neq \mathbf{A}_t$. For $\alpha_t = 1.0$, this results in the standard LSTD(λ) update, where selecting $\alpha_t < 1.0$ results in $\mathbf{w}_t$ changing more smoothly. For $k = d, \epsilon = 1, \eta = 0$, the ATD($\lambda$) update becomes

$$\begin{aligned}
\mathbf{w}_{t+1} &= \mathbf{w}_t + (\alpha_t\hat{\mathbf{A}}_t^\dagger + \eta\mathbf{I})(\epsilon\mathbf{b}_t + \delta_\epsilon\mathbf{e}_t) \\
&= \mathbf{w}_t + \alpha_t\mathbf{A}_t^\dagger\mathbf{b}_t - \alpha_t\mathbf{A}_t^\dagger\mathbf{e}_t(\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1})^\top\mathbf{w} \\
&= (\mathbf{I} - \tilde{\alpha}_t)\mathbf{w}_t + \alpha_t\mathbf{A}_t^\dagger\mathbf{b}_t,
\end{aligned}$$

where $\tilde{\alpha}_t = \alpha_t\mathbf{A}_t^\dagger\mathbf{e}_t(\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1})^\top$. For $t = 1$,

$$\tilde{\alpha}_1 = \alpha_1\mathbf{A}_1^\dagger\mathbf{e}_1(\mathbf{x}_1 - \gamma_2\mathbf{x}_2)^\top = \alpha_1\mathbf{I}$$

and so $(\mathbf{I} - \tilde{\alpha}_1)\mathbf{w}_1 = (1 - \alpha_1)\mathbf{w}_1$, giving the same update as exponentially-weighted LSTD(λ). The next step introduces some approximation, because $\tilde{\alpha}_2 = \alpha_2\mathbf{A}_2^\dagger\mathbf{e}_2(\mathbf{x}_2 - \gamma_3\mathbf{x}_3)^\top$ is close to α_2, but no longer exact. Despite the modification in the exponential averaging from the stochastic sampling, the same information is available in $\hat{\mathbf{A}}_t$ and $\mathbf{b}_t$ and ATD(λ) with these parameter settings is likely to behave similarly to exponentially-weighted LSTD(λ).

A.1.4 Detailed Experimental Specification

In both mountain car and energy storage domains, we do not have access to the parameters of the underlying MDPs (as we do in Boyan's chain) and thus must turn to Monte Carlo rollouts to estimate v_π in order to evaluate

our value function approximation methods. In both domains, we followed the same strategy. Our two ATD instances and all compared baselines can be found at https://github.com/yannickycpan/atd_tlstd.

To generate training data, we generated 100 trajectories of rewards and observations under the target policy, starting randomly from a small area near a start state. Each trajectory is composed of a fixed number of steps, either 5000 or 10000, and may contain many episodes in the case of episodic tasks like mountain car. The start states for each trajectory were sampled uniformly from (1) near the bottom of the hill with zero velocity for mountain car, (2) a small set of valid start states specified by the energy storage domains (Salas & Powell, 2013). Each trajectory represents one independent run of the domain.

The testing data was sampled according to the on-policy distribution induced by the target policy. For both domains we generated a single long trajectory selecting actions according to π. Then we randomly sampled 2000 states from this one trajectory. In mountain car domain, we ran 500 Monte Carlo rollouts to compute undiscounted sum of future rewards until termination, and take the average as an estimate true value. In the energy allocation domain, we ran 300 Monte Carlo rollouts for each evaluation state, each with length 1000 steps[1], averaging over 300 trajectories from each of the evaluation states. We evaluated the algorithms' performances by comparing the agent's prediction value with the estimated value of the 2000 evaluation states, at every 50 steps during training. We measured the percentage absolution mean error:

$$\text{error}(\mathbf{w}) = \frac{1}{2000}\sum_{i=1}^{2000}\frac{|\mathbf{w}^T\mathbf{x}(s_i) - \hat{v}_\pi(s_i)|}{|\hat{v}_\pi(s_i)|},$$

where $\hat{v}_\pi(s_i) \in \mathbb{R}$ denotes the Monte Carlo estimate of the value of evaluation state s_i.

Algorithms. The algorithms included in the experiments constitute a wide range of stochastic approximation algorithms and matrix-based (subquadratic)

[1] After 1000 steps the reward ($\sup reward = 1850$) can be scaled by γ^{1000}, which is smaller than 10^{-5} given our $\gamma = 0.99$.

algorithms. However, there are a few related algorithms that we chose not to include; for completeness, we explain our decision-making here.

There have been some accelerations proposed to gradient TD algorithms (Mahadevan et al., 2014; Meyer et al., 2014; Dabney & Thomas, 2014). However, they have either been shown to perform poorly in practice (White & White, 2016), or were based on applying accelerations outside their intended use (Meyer et al., 2014; Dabney & Thomas, 2014). Dabney & Thomas (2014) explored a similar update to ATD, but for the control setting and with an incremental update to the Fisher information matrix rather than $\mathbf{A}$ used here. As they acknowledge, this approach for TD methods is somewhat ad-hoc, as the typical update is not a gradient. Rather, their method is better suited for the policy gradient algorithms explored in that paper. Meyer et al. (2014) applied an accelerated Nesterov technique, called SAGE, to the two timescale gradient algorithms. Their approach does not take advantage of the simpler quadratic form of the MSPBE and only uses an approximate Lipschitz constant to improve the stepsize selection. Diagonal approximations to $\mathbf{A}$ constitute a strictly more informative stepsize approach, and we found these to be inferior to our low-rank strategy. The results by Meyer et al. (2014) using SAGE for GTD similarly indicated little to no gain. Finally, Givchi & Palhang (2014) investigated using diagonal approximations to $\mathbf{A}$ for TD, using ideas from the stochastic gradient descent literature. However, that paper does not justify their strategy. Throughout, the TD update is called the gradient and $\mathbf{A}$ the Hessian, the secant update for the diagonal of $\mathbf{A}^{-1}$ does not guarantee positive stepsizes, and so an ad-hoc rule is introduced, and the convergence result is incomplete in that the cited theorem does not permit a vector of stepsizes.

On the other hand, the true-online methods have consistently been demonstrated to have surprisingly strong performance (White & White, 2016), and so we opt instead for these practical competitors. Iterate or Polyak averaging can be applied to all the methods tested in our experiments. We achieve similar variance reduction benefits via averaging over many independent runs of all algorithms; we excluded this enhancement from our comparisons.

Boyan's Chain. This domain was implemented exactly as describe in Boyan's paper (Boyan, 1999). The task is episodic and the true value function is known, and thus we did not need to compute rollouts. Otherwise evaluation was performed exactly as described above. We tested the following parameter settings:

- $\alpha_0 \in \{0.1 \times 2.0^j | j = -12, -11, -10, ..., 4, 5\}$, 18 values in total
- $n_0 \in \{10^2, 10^6\}$
- $\lambda \in \{0.0, 0.1, ..., 0.9, 0.91, 0.93, 0.95, 0.97, 0.99, 1.0\}$, 16 values in total
- $\eta \in \{10^j | j = -4, -3.5, -3, ..., 3.5, 4, 4.5\}$, 18 values in total

The linear methods, (e.g., TD(0) true online ETD(λ)), made use of α_0, n_0, and λ, whereas the LSTD made use of η to initialize the incremental approximation of $\mathbf{A}$ inverse and λ. For the linear methods we also tested decaying step size schedule as originally investigated by Boyan

$$\alpha_t = \alpha_0 \frac{n_0 + 1}{n_0 + \#\text{terminations}}.$$

We also tested constant step-sizes where $\alpha_t = \alpha_0$. The ATD algorithm, as proposed, was tested with one fixed parameter setting.

Mountain Car. Our second batch of experiments was conducted on the classic RL benchmark domain Mountain Car. We used the specification from Sutton & Barto (2018) of the domain, where the agent's objective is to select one of three discrete actions (reverse, coast, forward), based on the continuous position and velocity of an underpowered car to drive it out of a valley, at which time the episode terminates. This is an undiscounted task. Each episode begins at the standard initial location–randomly near the hill's bottom– with zero velocity. Actions were selected according to a stochastic *Bang-bang* policy, where the reverse is selected if the velocity is negative and forward is selected if the velocity is positive and occasionally a random action is selected—we tested randomness in action selection of 0%, 10%, and 20%.

We used tile coding to convert the continuous state variable into high-dimensional binary feature vectors. The position and velocity we tile coded

jointly with ten tilings, each forming a two-dimensional uniform grid partitioned by 10 tiles in each dimension. This resulted in a binary feature vector of length 1000, with exactly ten components equal to one and the remaining equal to zero. We requested a 1024 memory size to guarantee the performance of the tile coder, which finally resulted in 1024 features. We used a standard freely available implementation of tile coding [2], which is described in detail in Sutton & Barto (2018).

We tested the following parameter settings for Mountain Car:

- $\alpha_0 \in \{0.1 \times 2.0^j | j = -7, -6, ..., 4, 5\}$ divided by number of tilings, 13 values in total
- $\lambda \in \{0.0, 0.1, ..., 0.9, 0.93, 0.95, 0.97, 0.99, 1.0\}$, 15 values in total
- $\eta \in \{10^j | j = -4, -3.25, -2.5, ..., 3.5, 4.25, 5.0\}$, 13 values in total

The linear methods (e.g., TD(0)), iLSTD, and fast LSTD made use of α_0 as stepsize, ATD uses $\alpha_0/100$ as a regularizer, whereas the LSTD and random projection LSTD made use of the η as regularization for Sherman-Morrison matrix initialization. All methods except fast LSTD and TD(0) made use of the λ parameter. iLSTD used decaying step-sizes with $n_0 = 10^2$. In addition we fixed the number of descent dimensions for iLSTD to one (recommended by previous studies (Geramifard & Bowling, 2006; Geramifard et al., 2007)). We found that the linear methods, on the other hand, performed worse in this domain with decayed stepsizes, so we only reported the performance for the constant step size setting. In this domain, we tested several settings for the regularization parameter for ATD. However, as the results demonstrate, ATD is insensitive to this parameter. Therefore we present results with the same fixed parameter setting for ATD as used in Boyan's chain. The low-rank matrix methods—including ATD—were tested with rank equal to $20, 30, 40, 50$, and 100. With rank $20, 30, 40$, we observed that ATD can still do reasonably well but converges slower. However, the rank = 100 setting does not show obvious

[2] https://webdocs.cs.ualberta.ca/ sutton/tiles2.html

strength, likely due to the fact that the threshold for inverting $\mathbf{A}$ remains unchanged.

Energy Allocation. The general description is in Section 4.2.1. There are originally four state variables on this domain: the amount of energy in the storage device R_t, the net amount of wind energy E_t, time aggregate demand D_t, and price of electricity P_t in the spot market. We made several minor modifications to the simulator to allow generating training or testing data for policy evaluation. First, we modified the original policy by setting the input time index as (#timeindex mod 24) so that we can remove the restriction that the time index must be no greater than 24, though the policy should be no longer optimal. Second, the original demand variable D_t is a function of time. We were able to remove this dependency by setting one more variable D_{t-1} at time step t. As a result, the state is encoded as five variables. Third, we considered the problem a continuous task by setting a discount rate ($\gamma = 0.99$) when estimating the values of states.

Here we describe the environment dynamics by showing how the state variables change at each time step. We first introduce a few notations. The stochastic processes associated with P_t, E_t are the jump process and uniform process, respectively. The demand process is deterministic. The ranges of R_t, E_t, P_t, D_t are: $[0, 30], [1, 7], [30, 70], [0, 7]$. When generating the training trajectories, we randomly choose the initial values of state variables R_t, E_t, P_t, D_t from the ranges: $[0, 10], [1, 5], [30, 50], [0, 7]$, respectively.

Define the vector $\phi \stackrel{\text{def}}{=} [0, 0, -1, 1, 1, -1]$, $T = 25$, and let $\mathbf{a}$ be an input action (i.e., a 6 dimensional vector). Let $\mathcal{PN}(\mu, \sigma, a, b, \Delta)$ be a discrete pseudonormal distribution with five parameters $\mu, \sigma, a, b, \Delta$. The distribution can be used to sample discrete values range from a to b with discretization level Δ. We refer to Salas & Powell (2013, Page 16) for concrete description of such a distribution. Let

$$\epsilon_t^P \sim \mathcal{PN}(0, 2.5, 30, 70, 1),$$

and

$$\epsilon_t^J \sim \mathcal{PN}(0, 50, 30, 70, 1).$$

Let $u_t \sim$Uniform$(0,1), \epsilon_t^E \sim$Uniform$(-1,1)$. Then the above variables evolve according to:

$$\begin{aligned}
R_t &= R_{t-1} + \mathbf{a}^\top \phi, \\
D_t &= \left\lfloor \max\{0, 3 - 4sin(\frac{2\pi t}{T})\} \right\rfloor, \\
P_t &= \min\{\max\{P_{t-1} + \epsilon_t^P + \mathbf{I}(u_t \leq p)\epsilon_t^J, P_{min}\}, P_{max}\}, p = 0.031, \\
E_t &= \min\{\max\{E_{t-1} + \epsilon_t^E, E_{min}\}, E_{max}\},
\end{aligned}$$

where $P_{min} = 30, P_{max} = 70, E_{min} = 1, E_{max} = 7$ correspond to the ranges for the two variables P_t, E_t we described above. $\mathbf{I}(\cdot)$ is an indicator function. It outputs 1 if the input is true; otherwise, 0.

To optimize the algorithms, we tested a similar set of parameters as before:

- $\alpha_0 \in \{0.1 \times 2.0^j | j = -7, -6, ..., 4, 5\}$ divided by number of tilings, 13 values in total
- $\lambda \in \{0.0, 0.1, ..., 0.9, 1.0\}$, 10 values in total
- $\eta \in \{10^j | j = -4, -3.25, -2.5, ..., 3.5, 4.25, 5.0\}$, 13 values in total.

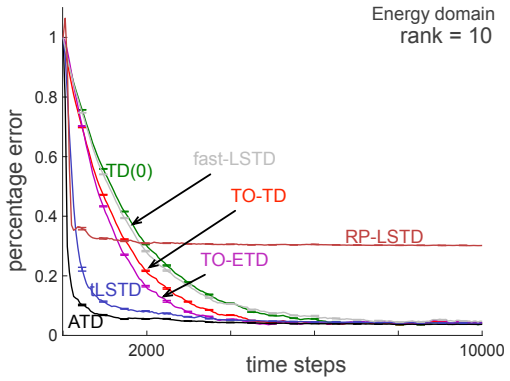


Figure A.1: Learning curves on energy allocation domain with rank equal to 10. Here we see the clear difference in the effect of rank on these two methods. ATD is only using the curvature information in $\hat{\mathbf{A}}$ to speed learning, whereas tLSTD uses $\hat{\mathbf{A}}$ in a closed-form solution.

Due to the size of the feature vector, we excluded LSTD from the results. The iLSTD was also excluded due to its slow runtime and poor performance in Mountain Car. Note that though the iLSTD avoids $\mathcal{O}(d^2)$ computation per step for sparse features, it still needs to store and update an $\mathcal{O}(d^2)$ matrix, and so does not scale as well as the other sub-quadratic methods.

A.2 ATD with Random Projection

This section includes relevant content for the ATD algorithm with random projection. First, we discuss low-rank properties of left-side sketching in Section A.2.1. Second, we introduce several other intuitive sketch-based iterative ATD updating rules in Section A.2.2. Last, in Section A.2.3, we include all experimental details and additional empirical results related to our ATD-sketch algorithm.

A.2.1 Row-rank Properties of SA

To ensure the right pseudo-inverse is well-defined, we show that the projected matrix $\mathbf{SA}$ is full row-rank with high probability, if $\mathbf{A}$ has a sufficiently high rank. We know that the probability measure of row-rank deficient matrices for $\mathbf{S}$ has zero mass. However, in the following, we prove a stronger and practically more useful claim that $\mathbf{SA}$ is far from being row-rank deficient. Formally, we define a matrix to be δ-*full row-rank* if there is no row that can be replaced by another row with distance at most δ to make that matrix row-rank deficient.

Proposition 2. *Let* $\mathbf{S} \in \mathbb{R}^{k\times d}$ *be any Gaussian matrix with* 0 *mean and unit variance. For* $r_A = rank(\mathbf{A})$ *and for any* $\delta > 0$, $\mathbf{SA}$ *is* δ*-full row-rank with probability at least* $1 - \exp(-2\frac{(r_A(1-0.8\delta)-k)^2}{r_A})$.

Proof. Let $\mathbf{A} = \mathbf{U\Sigma V}^\top$ be the SVD for $\mathbf{A}$. Since $\mathbf{U}$ is an orthonormal matrix, $\mathbf{S}' = \mathbf{SU}$ has the same distribution as $\mathbf{S}$ and the rank of $\mathbf{SA}$ is the same as $\mathbf{S}'\mathbf{\Sigma}$. Moreover notice that the last $d - r_A$ columns of $\mathbf{S}'$ get multiplied by all-zero rows of $\mathbf{\Sigma}$. Therefore, in what follows, we assume we draw a random matrix $\mathbf{S}' \in \mathbb{R}^{k\times r_A}$(similar to how $\mathbf{S}$ is drawn), and that $\mathbf{\Sigma} \in \mathbb{R}^{r_A\times r_A}$ is a full rank diagonal matrix. We study the rank of $\mathbf{S}'\mathbf{\Sigma}$.

Consider iterating over the rows of $\mathbf{S}'$, the probability that any new row is δ-far from being a linear combination of the previous ones is at least $1 - 0.8\delta$. To see why, assume that you currently have i rows and sample another vector $\mathbf{v}$ with entries sampled i.i.d. from a standard Gaussian as the candidate for the next row in $\mathbf{S}'$. The length corresponding to the projection of any row $\mathbf{S}'_{j:}$ onto $\mathbf{v}$, i.e., $\mathbf{S}'_{j:}\mathbf{v} \in \mathbb{R}$, is a Gaussian random variable. Thus, the probability of the $\mathbf{S}'_{j:}\mathbf{v}$ being within δ is at most 0.8δ. This follows from the fact that the area under probability density function of a standard Gaussian random variable over $[0, x]$ is at most $0.4x$, for any $x > 0$.

This stochastic process is a Bernoulli trial with a success probability of at least $1 - 0.8\delta$. The trial stops when there are k successes or when the number of iterations reaches r_A. The Hoeffding inequality bounds the probability of failure by $\exp(-2\frac{(r_A(1-0.8\delta)-k)^2}{r_A})$. □

A.2.2 Alternative Iterative Updates

In addition to the proposed iterative algorithm using a left-sided sketch of $\mathbf{A}$, we experimented with a variety of alternative updates that proved ineffective. We list them here for completeness.

We experimented with a variety of iterative updates. For a linear system, $\mathbf{Aw} = \mathbf{b}$, one can iteratively update using $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t)$ and $\mathbf{w}_t$ will converge to a solution of the system (under some conditions). We tested the following ways to use sketched linear systems.

First, for the two-sided sketched $\mathbf{A}$, we want to solve for $\mathbf{S}_L\mathbf{A}\mathbf{S}_R^\top\mathbf{w} = \mathbf{S}_L\mathbf{b}$. If $\tilde{\mathbf{A}}_t = \mathbf{S}_L\mathbf{A}_t\mathbf{S}_R^\top$ is square, we can use the iterative update

$$\begin{aligned}
\tilde{\mathbf{A}}_{t+1} &= \tilde{\mathbf{A}}_t + \frac{1}{t+1}\left(\mathbf{S}_L\mathbf{e}_t(\mathbf{S}_R\mathbf{d}_t)^\top - \tilde{\mathbf{A}}_t\right), \\
\tilde{\mathbf{b}}_{t+1} &= \tilde{\mathbf{b}}_t + \frac{1}{t+1}\left(r_{t+1}\mathbf{S}_L\mathbf{e}_t - \tilde{\mathbf{b}}_t\right), \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha_t(\tilde{\mathbf{b}}_{t+1} - \tilde{\mathbf{A}}_{t+1}\mathbf{w}_t), \\
&= \mathbf{w}_t + \alpha_t(\mathbf{S}_L\mathbf{b}_{t+1} - \mathbf{S}_L\mathbf{A}_{t+1}\mathbf{S}_R^\top\mathbf{w}_t)
\end{aligned}$$

and use $\mathbf{w}$ for prediction on the sketched features. Another option is to main-

tain the inverse incrementally, using Sherman-Morrison

$$\begin{aligned}
\mathbf{a}_d &= \mathbf{d}_t^\top \mathbf{S}_R^\top \tilde{\mathbf{A}}_t^{-1}, \\
\mathbf{a}_u &= \tilde{\mathbf{A}}_t^{-1} \mathbf{S}_L \mathbf{e}_t, \\
\tilde{\mathbf{A}}_t^{-1} &= \tilde{\mathbf{A}}_t^{-1} - \frac{\mathbf{a}_u \mathbf{a}_d}{1 + \mathbf{d}_t^\top \mathbf{a}_u}, \\
\tilde{\mathbf{b}}_t &= \tilde{\mathbf{b}}_t + \frac{r_{t+1} \mathbf{S}_L \mathbf{e}_t - \tilde{\mathbf{b}}_t}{t}, \\
\mathbf{w} &= \tilde{\mathbf{A}}_t^{-1} \tilde{\mathbf{b}}_t.
\end{aligned}$$

If $\mathbf{S}_L \mathbf{A} \mathbf{S}_R^\top$ is not square (e.g., $\mathbf{S}_R = \mathbf{I}$), we instead solve for

$$\mathbf{S}_L^\top \mathbf{S}_L \mathbf{A} \mathbf{S}_R^\top \mathbf{S}_R \mathbf{w} = \mathbf{S}_L^\top \mathbf{S}_L \mathbf{b},$$

where applying $\mathbf{S}_L^\top$ provides the recovery from the left and $\mathbf{S}_R$ the recovery from the right.

Second, with the same sketching, we also experimented with $\mathbf{S}_L^\dagger$, instead of $\mathbf{S}_L^\top$ for the recovery, and similarly for $\mathbf{S}_R$, but this provided no improvement.

For this square system, the iterative update is

$$\begin{aligned}
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha_t \mathbf{S}_L^\dagger (\tilde{\mathbf{b}}_{t+1} - \tilde{\mathbf{A}}_{t+1} \mathbf{S}_R \mathbf{w}_t) \\
&= \mathbf{w}_{t+1} + \alpha_t \mathbf{S}_L^\dagger (\mathbf{S}_L \mathbf{b}_{t+1} - \mathbf{S}_L \mathbf{A}_{t+1} \mathbf{S}_R^\top \mathbf{S}_R \mathbf{w}_t)
\end{aligned}$$

for the same $\tilde{\mathbf{b}}_t$ and $\tilde{\mathbf{A}}_t$ which can be efficiently kept incrementally, while the pseudoinverse of $\mathbf{S}_L$ only needs to be computed once at the beginning.

Third, we tried to solve the system $\mathbf{S}_L^\top \mathbf{S}_L \mathbf{A} \mathbf{w} = \mathbf{b}$, using the updating rule $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t (\mathbf{b}_{t+1} - \mathbf{S}_L^\top \mathbf{S}_L \mathbf{A}_{t+1} \mathbf{w}_t)$, where the matrix $\mathbf{S}_L \mathbf{A}_{t+1}$ can be incrementally maintained at each step by using a simple rank-one update.

Fourth, we tried to explicitly regularize these iterative updates by adding a small step in the direction of $\delta_t \mathbf{e}_t$.

In general, none of these iterative methods performed well. We hypothesize this may be due to difficulties in choosing stepsize parameters. Ultimately, we found the sketched updated within ATD to be the most effective.

A.2.3 Experimental Details

Mountain Car. The setting is the same as introduced in Section A.1.4. We estimate the true value—the expected return—by computing the average

over 1000 returns, generated by rollouts. The policy for Mountain Car is the energy pumping policy with 20% randomness starting from slightly random initial states. The discount rate is 1.0, and is 0 at the end of the episode, and the reward is always -1.

Puddle world. Puddle World (Boyan & Moore, 1995a) is an episodic task, where the goal is for a robot in a continuous gridworld to reach a goal state within as fewest steps as possible. The state is 2-dimensional, consisting of (x, y) positions. We use the same setting as described in Sutton & Barto (2018), with a discount of 1.0 and -1 per step, except when going through a puddle that gives a higher magnitude negative reward. We compute the true values from 2000 states in the same way as Mountain Car. A simple heuristic policy choosing the action leading to the shortest Euclidean distance with 10% randomness is used.

Acrobot. Acrobot is a four-dimensional episodic task, where the goal is to raise an arm to a certain level. The reward is -1 for non-terminal states and 0 for goal state, again with the discount rate set to 1.0. We use the same tile coding as described in (Sutton & Barto, 2018), except that we use memory size $2^{15} = 32,768$. To get a reasonable policy, we used true-online Sarsa(λ) to go through 15000 episodes with stepsize $\alpha = 0.1/48$ and bootstrap parameter $\lambda = 0.9$. Each episode starts with slight randomness. The policy is $\epsilon-$greedy with respect to state value and $\epsilon = 0.05$. The way we compute true values and generate training trajectories are the same as we described for the above two domains.

Energy allocation. Energy allocation (Salas & Powell, 2013) is a continuing task with a five-dimensional state, where we use the same settings as detailed before. The matrix $\mathbf{A}$ was shown to have a low-rank structure and hence matrix approximation methods are expected to perform well.

RBF features. For radial basis functions, we used format

$$k(\mathbf{x}, \mathbf{c}) = \exp(-\frac{||\mathbf{x} - \mathbf{c}||_2^2}{2\sigma^2}),$$

where σ is called RBF width and $\mathbf{c}$ is a feature. On Mountain Car, because the position and velocity have different ranges, we set the bandwidth separately for each feature using

$$k(\mathbf{x}, \mathbf{c}) = \exp(-((\frac{\mathbf{x}_1 - \mathbf{c}_1}{0.12r_1})^2 + (\frac{\mathbf{x}_2 - \mathbf{c}_2}{0.12r_2})^2)),$$

where r_1 is the range of the first state variable and r_2 is the range of second state variable.

In Figure 5.5, we used a relatively rarely used representation which we call spline feature. For sample $\mathbf{x}$, the ith spline feature is set to 1 if $||\mathbf{x} - \mathbf{c}_i|| < \delta$ and otherwise set as 0. The centers are selected in exactly the same way as for the RBFs.

Parameter optimization. We swept the following ranges for stepsize (α), bootstrap parameter (λ), regularization parameter (η), and initialization parameter ξ for all domains:

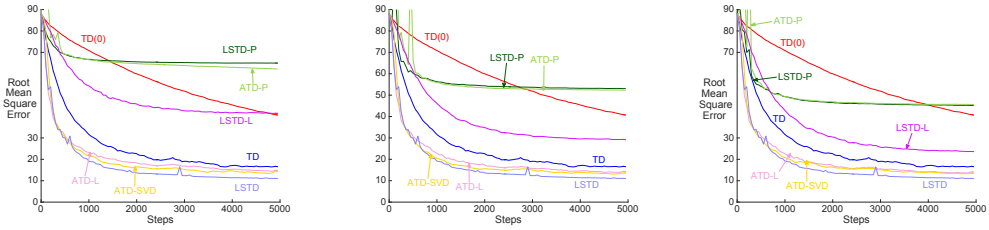
1. $\alpha \in \{0.1 \times 2.0^j | j = -7, -6, ..., 4, 5\}$ divided by l_1 norm of feature representation, 13 values in total.
2. $\lambda \in \{0.0, 0.1, ..., 0.9, 0.93, 0.95, 0.97, 0.99, 1.0\}$, 15 values in total.
3. $\eta \in \{0.01 \times 2.0^j | j = -7, -6, ..., 4, 5\}$ divided by l_1 norm of feature representation, 13 values in total.
4. $\xi \in \{10^j | j = -5, -4.25, -3.5, ..., 2.5, 3.25, 4.0\}$, 13 values in total.

To choose the best parameter setting for each algorithm, we used the sum of RMSE across all steps for all the domains Energy allocation. For this domain, optimizing based on the whole range causes TD to pick an aggressive stepsize to improve early learning at the expense of later learning. Therefore, for Energy allocation, we instead select the best parameters based on the sum of the RMSE for the second half of the steps.

For the ATD algorithms, as described previously, we set $\alpha_t = \frac{1}{t}$ and only swept the regularization parameter η, which is set to 0.1 times the range of α for other TD baselines.

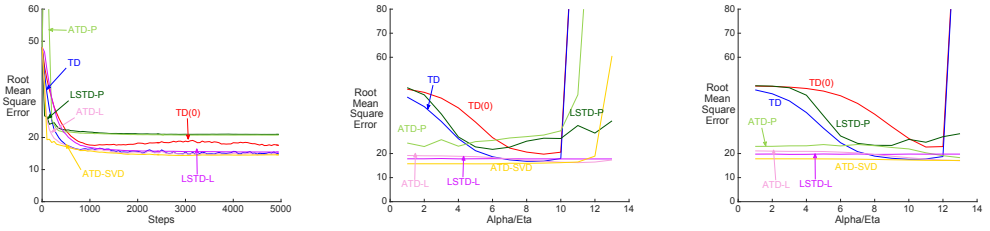
A.2.4 Additional Experimental Results

We include additional results to validate the utility of our algorithm further. First, we show the learning curves and parameter sensitivity in Mountain Car in Figure A.4, for RBFs and tile coding. Similarly, we only showed Acrobot with RBFs in the main text and have results with tile coding here.



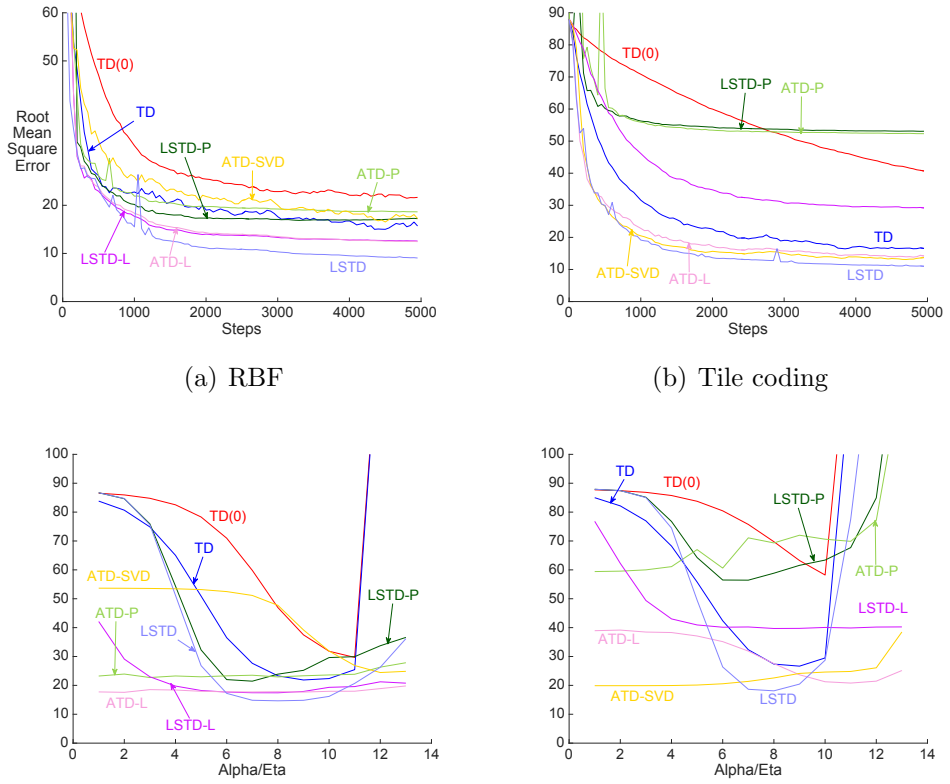
(a) Tile coding, $k = 25$ (b) Tile coding, $k = 50$ (c) Tile coding, $k = 75$

Figure A.2: Change in performance when increasing k, from 25 to 75 on Mountain Car domain. We can draw similar conclusions to the same experiments in Puddle World in the main text. Here, the unbiased of ATD-L is even more evident; even with as low a dimension as 25, it performs similarly to LSTD.



(a) tile coding, $k = 50$ (b) tile coding, $k = 50$ (c) RBF, $k = 75$

Figure A.3: Additional experiments in Acrobot, for tile coding with $k = 50$ and for RBFs with $k = 75$.



(a) RBF

(b) Tile coding

(c) RBF, Sensitivity

(d) Tile coding, Sensitivity

Figure A.4: **(a)** and **(b)** are learning curves on Mountain Car with $k = 50$, and **(c)** and **(d)** are their corresponding parameter-sensitivity plots. The sensitivity plots report average RMSE over the entire learning curve for the best λ for each parameter. The stepsize α is reported for TD, the initialization parameter ξ for the LSTD methods and the regularization parameter η for the ATD methods. The initialization for the matrices in the ATD methods is fixed to the identity. The range for the regularization term η is 0.1 times the range for α. As before, the sketching approaches with RBFs perform better than with tile coding. The sensitivity of the left-side projection methods is significantly lower than the TD methods. ATD-L also seems to be less sensitive than ATD-SVD, and incurs less bias than LSTD-L.

Appendix B

Search-control Methods

B.1 Value-based Search-control

B.1.1 Experimental Details

Algorithmic details. We summarize our algorithm in Algorithm 10. The algorithm closely follows our generic Algorithm 7, but it includes implementation details. To ensure some separation between states in the SC queue, we use a threshold ϵ_a to decide whether or not to add a state into the queue. We use a simple heuristic to set this threshold on each step, as the following sample average: $\epsilon_a \approx \epsilon_a^{(T)} = \sum_{t=1}^{T} \frac{||s_{t+1} - s_t||_2/\sqrt{d}}{T}$. We divide by $\sqrt{d}$ to make the threshold less sensitive to state dimensions. The start state for the gradient ascent is randomly sampled from the ER buffer.

Implementation details of common settings. The continuous GridWorld domain is written by ourselves. Deep learning implementation is based on TensorFlow with version 1.1.0 (Abadi et al., 2015). For DQN, we use Adam optimizer (Kingma & Ba, 2014), Xavier initializer (Glorot & Bengio, 2010), set mini-batch size $b = 32$, buffer size 100k. All activation functions are ReLU except the output layer of the Q-value is linear. The output layer parameters were initialized from a uniform distribution $[-0.003, 0.003]$.

Experimental details of TabularGridWorld domain. Our TabularGridWorld is similar to the continuous state domain introduced in 6.1(a) except that we do not have a wall, and we introduce stochasticity to make it more

Algorithm 10 Dyna-Value

Input: budget k for the number of gradient ascent steps (e.g., $k = 100$), n the number planning steps/updates, stochasticity η for gradient ascent (e.g., $\eta = 0.1$), ρ percentage of updates from SC queue (e.g., $\rho = 0.5$), d the number of state variables, i.e. $\mathcal{S} \subset \mathbb{R}^d$.
Initialize empty SC queue S_c and ER buffer B_{er}
$\hat{\mathbf{\Sigma}}_s \leftarrow \mathbf{I}$ (empirical covariance matrix)
$\mu_{ss} \leftarrow \mathbf{0} \in \mathbb{R}^{d \times d}, \mu_s \leftarrow \mathbf{0} \in \mathbb{R}^d$ (auxiliary variables for computing empirical covariance matrix, sample average will be maintained for μ_{ss}, μ_s)
$\epsilon_a \leftarrow 0$ (threshold for accepting a state)
for $t = 1, 2, \ldots$ **do**
  Observe (s, a, s', r) and add it to B_{er}
  $\mu_{ss} \leftarrow \frac{(t-1)\mu_{ss} + ss^\top}{t}, \mu_s \leftarrow \frac{(t-1)\mu_s + s}{t}$
  $\hat{\mathbf{\Sigma}}_s \leftarrow \mu_{ss} - \mu_s \mu_s^\top$
  $\epsilon_a \leftarrow (1-\beta)\epsilon_a + \beta ||s' - s||_2 / \sqrt{d}$ for $\beta = 0.001$
  Sample s_0 from B_{er}, $\tilde{s} \leftarrow \infty$
  for $i = 0, \ldots, k$ **do**
    $g_{s_i} \leftarrow \nabla_s V(s_i) = \nabla_s \max_a Q_\theta(s_i, a)$
    $s_{i+1} \leftarrow s_i + \frac{0.1}{||\hat{\mathbf{\Sigma}}_s g_{s_i}||} \hat{\mathbf{\Sigma}}_s g_{s_i} + X_i, X_i \sim \mathcal{N}(0, \eta \hat{\mathbf{\Sigma}}_\mathbf{s})$
    if distance$(\tilde{s}, s_{i+1}) \geq \epsilon_a$ **then**
      Add s_{i+1} into B_s, $\tilde{s} \leftarrow s_{i+1}$
  for n times **do**
    Sample a mixed mini-batch b, with proportion ρ from B_s and $1 - \rho$ from B_{er}
    Update parameters θ (i.e. DQN update) with b

representative. Four actions are available and can take the agent to the next $\{up, down, left, right\}$ grid, respectively. An action can be executed successfully with a probability of 0.8; otherwise, random action is taken.The TabularGridWorld size is 20×20 and each episode start from left-bottom grid and would terminate if reached the right-top grid or 1k time steps. The return will not be truncated unless the right-top grid is reached. The discount rate is $\gamma = 1.0$. For all algorithms, we fixed the exploration noise as $\epsilon = 0.2$ and sweep over learning rate $\{2^0, 2^{-0.25}, 2^{-0.5}, 2^{-0.75}, 2^{-1}, 2^{-1.5}, 2^{-2.0}, 2^{-2.5}\}$.We fix using exploration noise $\epsilon = 0.2$.

Experimental details of continuous GridWorld. All continuous state domain, we set discount rate $\gamma = 0.99$. We set the episode length limit as

2000 for GridWorld, while keeping other domains as the default setting. We use warmup steps 5000 for all algorithms to fill the ER buffer before learning begins.

For all Q networks, we consistently use a neural network with two 32 units hidden ReLU layers. We use target network moving frequency $\tau = 1000$ and sweep learning rate $\{0.001, 0.0001, 0.00001\}$ for vanilla DQN with ER with planning step 5. For our particular parameters, we fixed the same setting across all domains: ϵ_a is sample average, the number of gradient steps $k = 100$ with gradient ascent step size 0.1 and queue size $1e6$. We incrementally update the empirical covariance matrix. When evaluating each algorithm, we keep a small noise $\epsilon = 0.05$ when taking action and evaluate one episode every 1000 environment time steps for each run.

B.2 Frequency-based Search-control

In Section B.2.1, we provide calculations for Example 1, Example 2 and theoretical proof of Theorem 4. Additional experiments regarding separate criterion of hill climbing, and on continuous control problems are shown in Section B.2.2. Experimental details for reproducing our empirical results are in Section B.2.3.

B.2.1 Theoretical Proofs

This section includes calculations for Example 1, Example 2, and theoretical proof of Theorem 4.

Calculations for Example 1 and Example 2

Example 1. For $f_{\sin}$ defined in Eq. (6.2), calculate the integrals of squared first order derivative $f'_{\sin}$ on high frequency region $[-2, 0)$ and low frequency region $[0, 2]$, respectively:

$$\int_{-2}^{0} [f'_{\sin}(x)]^2 \, dx = 64\pi^2, \quad \int_{0}^{2} [f'_{\sin}(x)]^2 \, dx = \pi^2.$$

Proof. Taking derivative and integral,

$$\int_{-2}^{0} [f'(x)]^2 \, dx = 64\pi^2 \int_{-2}^{0} [\cos(8\pi x)]^2 \, dx = 64\pi^2,$$
$$\int_{0}^{2} [f'(x)]^2 \, dx = \pi^2 \int_{0}^{2} [\cos(\pi x)]^2 \, dx = \pi^2.$$ □

Example 2. Let $f : [-\pi, \pi] \to \mathbb{R}$ be a band-limited real valued function defined as

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(nx) + b_n \sin(nx),$$

where $a_0, a_n, b_n \in \mathbb{R}$, $n = 1, 2, \ldots, N$ are Fourier coefficients of frequency $\frac{n}{2\pi}$. Then,

$$\int_{-\pi}^{\pi} |f'(x)|^2 \, dx = \pi \cdot \sum_{n=1}^{N} n^2 \left(a_n^2 + b_n^2\right), \quad \int_{-\pi}^{\pi} |f''(x)|^2 \, dx = \pi \cdot \sum_{n=1}^{N} n^4 \left(a_n^2 + b_n^2\right).$$

Proof. Taking derivative of f,

$$f'(x) = \sum_{n=1}^{N} [-n a_n \sin(nx)] + \sum_{n=1}^{N} [n b_n \cos(nx)].$$

Taking square of f',

$$\begin{aligned}
[f'(x)]^2 = &\sum_{n=1}^{N}\sum_{m=1}^{N} [nm a_n a_m \sin(nx)\sin(mx)] \\
&- \sum_{n=1}^{N}\sum_{m=1}^{N} [nm a_n b_m \sin(nx)\cos(mx)] \\
&- \sum_{n=1}^{N}\sum_{m=1}^{N} [mn a_m b_n \sin(mx)\cos(nx)] \\
&+ \sum_{n=1}^{N}\sum_{m=1}^{N} [nm b_n b_m \cos(nx)\cos(mx)].
\end{aligned}$$

Taking integral,

$$\begin{aligned}
\int_{-\pi}^{\pi} [f'(x)]^2 \, dx &= \int_{-\pi}^{\pi} \sum_{n=1}^{N} \sum_{m=1}^{N} [nma_n a_m \sin(nx) \sin(mx)] dx \\
&\quad - \int_{-\pi}^{\pi} \sum_{n=1}^{N} \sum_{m=1}^{N} [nma_n b_m \sin(nx) \cos(mx)] dx \\
&\quad - \int_{-\pi}^{\pi} \sum_{n=1}^{N} \sum_{m=1}^{N} [mna_m b_n \sin(mx) \cos(nx)] dx \\
&\quad + \int_{-\pi}^{\pi} \sum_{n=1}^{N} \sum_{m=1}^{N} [nmb_n b_m \cos(nx) \cos(mx)] dx \\
&= \sum_{n=1}^{N} \sum_{m=1}^{N} [nma_n a_m \pi \delta_{n,m} - 0 - 0 + nmb_n b_m \pi \delta_{n,m}] \\
&= \pi \cdot \sum_{n=1}^{N} n^2 \left(a_n^2 + b_n^2\right),
\end{aligned}$$

where

$$\delta_{n,m} \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } n = m, \\ 0, & \text{otherwise.} \end{cases}$$

Using similar arguments, taking derivative of $f'(x)$,

$$f''(x) = \sum_{n=1}^{N} \left[-n^2 a_n \cos(nx)\right] + \sum_{n=1}^{N} \left[-n^2 b_n \sin(nx)\right].$$

Taking integral,

$$\int_{-\pi}^{\pi} [f''(x)]^2 \, dx = \pi \cdot \sum_{n=1}^{N} n^4 \left(a_n^2 + b_n^2\right). \qquad \square$$

Proof for Theorem 4

Notations. For any vector norm $||\cdot||$, we mean l_2 norm and we ignore the subscript unless clarification is needed. We use Frobenius norm $||\cdot||_F$ for matrix. We use subscript y_l to denote the lth element in vector y. Let $H_f(y)$ be the Hessian matrix of $f(y)$. We write H for short unless clarification is needed. Let $H_{l,:}$ be the lth row of the Hessian matrix.

Proof description. We establish the connection between local gradient norm, Hessian norm and local frequency. To build such connection, we introduce a definition of $\pi_{\hat{f}}$ as shown below and we call it "local frequency distribution" of $f(x)$. $\pi_{\hat{f}}$ is a probability distribution over $\mathbb{R}^n$, i.e., $\int_{k\in\mathbb{R}^n} \pi_{\hat{f}}(k)dk = 1$, and $\pi_{\hat{f}}(k) \geq 0, \quad \forall k \in \mathbb{R}^n$. Within an open subset of domain (an unit ball), this distribution characterizes the proportion of a particular frequency component occupies. The proof can be described by three key steps:1) We use a local Fourier transform to express a function locally (i.e. within an unit ball). 2) we calculate the gradient/Hessian norm based on this local Fourier transform; 3) we take integration over the unit ball of the gradient/Hessian norm to build the connection with the local frequency distribution $\pi_{\hat{f}}$ and function energy.

Theorem 4. Given any function $f : \mathbb{R}^n \to \mathbb{R}$, for any frequency vector $k \in \mathbb{R}^n$, define its local Fourier transform as

$$\hat{f}(k) \stackrel{\text{def}}{=} \int_{y\in B(x,1)} f(y) \exp\left\{-2\pi i \cdot y^\top k\right\} dy,$$

for local function $f(y)$ defined around x, i.e., $y \in B(x,1) \stackrel{\text{def}}{=} \{y : \|y - x\| < 1\}$. Assume the local function "energy" is finite,

$$\int_{y\in B(x,1)} [f(y)]^2 \, dy = \int_{\mathbb{R}^n} \|\hat{f}(k)\|^2 dk < \infty, \quad \forall x \in \mathbb{R}^n.$$

Define "local frequency distribution" of $f(x)$ as:

$$\pi_{\hat{f}}(k) \stackrel{\text{def}}{=} \frac{\|\hat{f}(k)\|^2}{\int_{\mathbb{R}^n} \|\hat{f}(\tilde{k})\|^2 d\tilde{k}}, \quad \forall k \in \mathbb{R}^n.$$

Then, $\forall x \in \mathbb{R}^n$, we have:

1) the first order connection:

$$\int_{y\in B(x,1)} \|\nabla f(y)\|^2 \, dy = 4\pi^2 \cdot \left[\int_{y\in B(x,1)} [f(y)]^2 \, dy\right] \cdot \left[\int_{\mathbb{R}^n} \pi_{\hat{f}}(k) \cdot \|k\|^2 \, dk\right],$$

2) the second order connection:

$$\int_{y\in B(x,1)} ||H(y)||_F^2 dy = 16\pi^4 \left[\int_{y\in B(x,1)} [f(y)]^2 \, dy\right] \cdot \left[\int_{\mathbb{R}^n} \pi_{\hat{f}}(k) \cdot \|k\|^4 \, dk\right]$$

Proof. 1) We first prove the first order connection.

Consider the following function defined locally around x,

$$f_x(y) \stackrel{\text{def}}{=} \begin{cases} f(y), & \text{if } y \in B(x,1), \\ 0, & \text{otherwise.} \end{cases}$$

By definition, the Fourier transform of f_x is

$$\begin{aligned}\hat{f}(k) &= \int_{y\in B(x,1)} f(y)\exp\left\{-2\pi i\cdot y^\top k\right\} dy \\ &= \int_{\mathbb{R}^n} f_x(y)\exp\left\{-2\pi i\cdot y^\top k\right\} dy.\end{aligned}$$

And the inverse Fourier transform of $f_x(y)$, $\forall y\in B(x,1)$ is,

$$f_x(y) = \int_{\mathbb{R}^n} \hat{f}(k)\exp\left\{2\pi i\cdot y^\top k\right\} dk,$$

and then the gradient $\forall y\in B(x,1)$ is

$$\nabla f(y) = \nabla f_x(y) = \int_{\mathbb{R}^n} \hat{f}(k)\exp\left\{2\pi i\cdot y^\top k\right\}(2\pi i\cdot k)\, dk. \tag{B.1}$$

To calculate gradient norm, we use complex conjugate,

$$\nabla f^*(y) = \int_{\mathbb{R}^n} \hat{f}^*(k')\exp\left\{-2\pi i\cdot y^\top k'\right\}(-2\pi i\cdot k')\, dk',$$

where

$$\hat{f}^*(k') = \int_{\mathbb{R}^n} f_x(y')\exp\left\{2\pi i\cdot y'^\top k'\right\} dy'$$

is the complex conjugate of $\hat{f}(k)$. Therefore,

$$\begin{aligned}\|\nabla f(y)\|^2 &= \langle \nabla f(y), \nabla f^*(y)\rangle \\ &= \int_{\mathbb{R}^n}\int_{\mathbb{R}^n} \hat{f}(k)\hat{f}^*(k')\exp\left\{2\pi i\cdot y^\top(k-k')\right\}\left(4\pi^2 k^\top k'\right) dk dk'.\end{aligned} \tag{B.2}$$

Taking integral of $\|\nabla f(y)\|^2$ within the unit ball centered at x,

$$\int_{y\in B(x,1)} \|\nabla f(y)\|^2\, dy = \int_{\mathbb{R}^n} \|\nabla f_x(y)\|^2\, dy, \text{ by function definition} \tag{B.3a}$$

$$= \int_{\mathbb{R}^n}\int_{\mathbb{R}^n} \hat{f}(k)\hat{f}^*(k')\left[\int_{\mathbb{R}^n}\exp\left\{2\pi i\cdot y^\top(k-k')\right\} dy\right]\left(4\pi^2 k^\top k'\right) dk dk' \tag{B.3b}$$

$$= \int_{\mathbb{R}^n}\int_{\mathbb{R}^n} \hat{f}(k)\hat{f}^*(k')\delta_{k-k',\mathbf{0}}\left(4\pi^2 k^\top k'\right) dk dk' \tag{B.3c}$$

$$= 4\pi^2\int_{\mathbb{R}^n} \|\hat{f}(k)\|^2\cdot\|k\|^2\, dk. \tag{B.3d}$$

Recall the definition of local function "energy" around x,

$$\begin{aligned}\int_{\mathbb{R}^n} \|\hat{f}(k)\|^2 dk &= \int_{\mathbb{R}^n} \left\langle \hat{f}(k), \hat{f}^*(k) \right\rangle dk \\ &= \int_{y\in\mathbb{R}^n} \int_{y\in\mathbb{R}^n} f_x(y) f_x(y') \left[\int_{\mathbb{R}^n} \exp\left\{2\pi i k^\top (y'-y)\right\} dk \right] dy dy' \\ &= \int_{y\in\mathbb{R}^n} \int_{y\in\mathbb{R}^n} f_x(y) f_x(y') \delta_{y'-y,\mathbf{0}} dy dy' \\ &= \int_{y\in\mathbb{R}^n} f_x^2(y) dy \\ &= \int_{y\in B(x,1)} f^2(y) dy < \infty.\end{aligned}$$

The last line is done by definition of $f_x(y)$ and finite energy assumption. For $y \in B(x,1)$, the local gradient information is related to local energy and frequency distribution,

$$\begin{aligned}\int_{y\in B(x,1)} \|\nabla f(y)\|^2 dy &= 4\pi^2 \int_{\mathbb{R}^n} \|\hat{f}(k)\|^2 \cdot \|k\|^2 \frac{\int_{\mathbb{R}^n} \|\hat{f}(\tilde{k})\|^2 d\tilde{k}}{\int_{\mathbb{R}^n} \|\hat{f}(\tilde{k})\|^2 d\tilde{k}} dk \\ &= 4\pi^2 \int_{\mathbb{R}^n} \pi_{\hat{f}}(k) \|k\|^2 \int_{\mathbb{R}^n} \|\hat{f}(\tilde{k})\|^2 d\tilde{k} dk \\ &= 4\pi^2 \cdot \left[\int_{y\in B(x,1)} f^2(y) dy \right] \cdot \left[\int_{\mathbb{R}^n} \pi_{\hat{f}}(k) \cdot \|k\|^2 dk \right],\end{aligned}$$

where the last equality follows by $\int_{\mathbb{R}^n} \|\hat{f}(\tilde{k})\|^2 d\tilde{k} = \int_{y\in B(x,1)} f^2(y) dy$ which is established in the derivation (B.4).

2) Now we prove the second order connection.

To show the second order connection, we start from Eq. (B.1). Then the lth row of the Hessian matrix $H_{l,:}$ can be written as:

$$H_{l,:} = \frac{\partial \nabla f(y)}{\partial y_l}^\top$$

where we use the notation $\frac{\partial \nabla f(y)}{\partial y_l}$ to denote the vector formed by taking partial derivative of each element in the gradient vector $\nabla f(y)$ w.r.t. y_l. Then,

$$\frac{\partial \nabla f(y)}{\partial y_l} = \int_{\mathbb{R}^n} \hat{f}(k) \exp\left\{2\pi i \cdot y^\top k\right\} \left(4\pi^2 i^2 (e_l^\top k) k\right) dk,$$

where e_l is standard basis vector where the lth element is one. To calculate the norm of the vector $H_{l,:} = \frac{\partial \nabla f(y)}{\partial y_l}^\top$, we use complex conjugate again and

follow the similar derivation as done in Eq. (B.2):

$$
\begin{aligned}
&||H_{l,:}||_2^2 = \langle H_{l,:}, H_{l,:} \rangle \\
&= \int_{\mathbb{R}^n} \int_{\mathbb{R}^n} \hat{f}(k)\hat{f}^*(k') \exp\left\{2\pi i \cdot y^\top (k-k')\right\} \left(16\pi^4 i^4 (e_l^\top k)(e_l^\top k') k^\top k'\right) dk dk'
\end{aligned}
$$

Note that the square of Frobenius norm of the Hessian matrix can be written as $||H||_F^2 = \sum_{i,j} H_{i,j}^2 = \sum_{l=1}^n ||H_{l,:}||_2^2$. Then,

$$
\begin{aligned}
&||H||_F^2 = \sum_{l=1}^n ||H_{l,:}||_2^2 \\
&= \int_{\mathbb{R}^n} \int_{\mathbb{R}^n} \hat{f}(k)\hat{f}^*(k') \exp\left\{2\pi i y^\top (k-k')\right\} \left(16\pi^4 i^4 \sum_{l=1}^n (e_l^\top k)(e_l^\top k') k^\top k'\right) dk dk' \\
&= 16\pi^4 \int_{\mathbb{R}^n} \int_{\mathbb{R}^n} \hat{f}(k)\hat{f}^*(k') \exp\left\{2\pi i y^\top (k-k')\right\} (k^\top k')^2 dk dk'
\end{aligned}
$$

Taking the integration of $||H||_F^2$ over y variable within a ball with center x and unit radius, we acquire:

$$
\begin{aligned}
\int_{y \in B(x,1)} ||H(y)||_F^2 dy &= 16\pi^4 \int_{\mathbb{R}^n} ||\hat{f}(k)||^2 ||k||^4 dk \\
&= 16\pi^4 \left[\int_{y \in B(x,1)} [f(y)]^2 \, dy\right] \cdot \left[\int_{\mathbb{R}^n} \pi_{\hat{f}}(k) \cdot \|k\|^4 \, dk\right]
\end{aligned}
$$

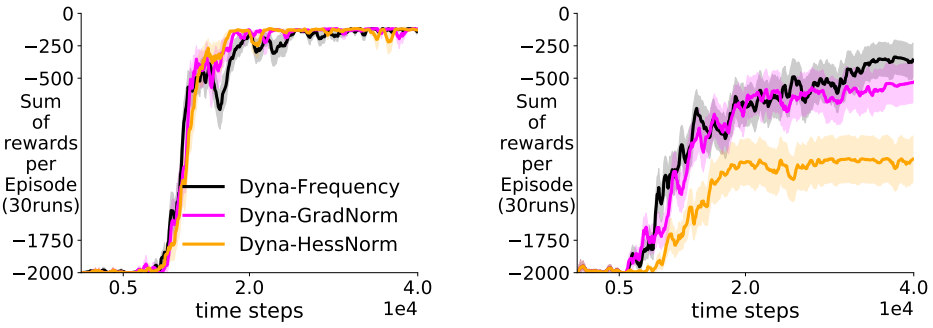
where the derivation process for the first equation is a simple modification from the derivation (B.3) and the second equation follows the same derivation (B.5).

□

B.2.2 Additional Experiments

In this section, we briefly study the effect of doing hill climbing on only gradient norm or Hessian norm. Then we demonstrate that our search-control strategy can be also used for continuous control algorithms.

Hill climbing on only gradient norm or Hessian norm. We use the form of $g(s) = \|\nabla_s V(s)\|^2 + \|H_v(s)\|_F^2$ to search states from high (local) frequency region of the value function. Besides the theoretical reason, there is a practical demand of such design. On value function surface, regions which have low (or even zero) gradient magnitude may have high Hessian magnitude, and



(a) plan steps 10, MountainCar

(b) plan steps 10, GridWorld

Figure B.1: Evaluation curves (sum of episodic reward v.s. environment time steps) of hill climbing on gradient norm (Dyna-GradNorm) and Hessian norm (Dyna-HessNorm) on MountainCar and GridWorld with 10 planning updates. All results are averaged over 30 random seeds.

vice versa. Hence, it can help move along the gradient trajectory in case that one of the term vanished at some point. Such cases can be a result of function approximation (smoothness/differentiability), or of the nature of the task, or both. In Fig. B.1, we show the results of using only either gradient norm or Hessian norm. The reason we choose MountainCar and GridWorld is that, the former has a value function surface with lots of variations; while the latter's value function increases smoothly from the initial state to the goal state, which indicates a small magnitude second-order derivative. Indeed, we empirically observe that the term $\nabla_s \|H_v(s)\|_F^2$ frequently gives a zero vector on GridWorld. This explains the bad performance of Dyna-HessNorm in Fig. B.1(b). In contrast, Fig. B.1(a) shows slightly better performance of Dyna-HessNorm and Dyna-GradNorm. Notice that, an intuitive and more general form of $g(x)$ can be

$$g(s) = \eta_1 \|\nabla_s V(s)\|^2 + \eta_2 \|H_v(s)\|_F^2,$$

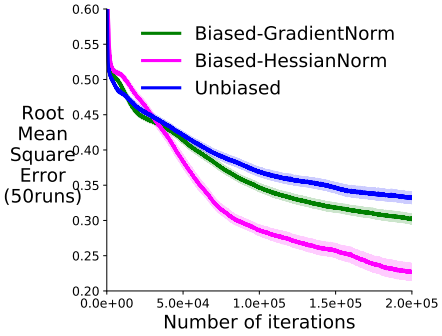
at the cost that additional meta-parameters are introduced.

In fact, there are many different ways to combine hill climbing strategies. Here are some unsuccessful trials. For example, climbing on direct combinations of $V(s)$ (value function) and $g(s)$ (frequency criterion), such as $V(s) + g(s)$, or $V(s)g(s)$, did not work well. The reasons are as follow-

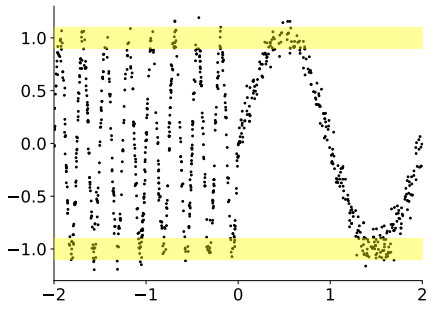
ing. First, such combination can lead to unpredictable gradient behaviour. It can alter the trajectory solely based on either $g(s)$ or $V(s)$, and the effect is unclear. It may lead to states with neither high value or high frequency. Last, and probably the most important, hill climbing on $V(s)$ and on $g(s)$ have fundamentally different insights. The former is based on the intuition that the value information should be propagated from the high value region to low value region; as a result, it requires to store states along the whole trajectory, including those in low value region. However, the latter is based on the insight that the function value in high frequency region is more difficult to approximate and needs more samples, while there is no obvious reason to propagate those information back to low frequency region. As a result, this approach does not emphasize on recording states throughout the whole hill climbing trajectory.

Empirical demonstration of sampling based on Hessian norm or gradient norm in a supervised learning setting. Our calculation in the examples 1, 2 implies that regions with large gradient and Hessian norm correspond to high frequency regions. We empirically verify this insight. Our expectation is that biasing training dataset towards high gradient norm and Hessian norm would achieve better learning results. In Fig. B.2(a), **Biased-GradientNorm** corresponds to uniformly sampling $x \in [-2, 2]$ for 60% of training data and sampling proportional to gradient norm (i.e., $p(x) \propto |f'_{\sin}(x)|$) for the remaining 40%; while **Biased-HessianNorm** corresponds to sampling proportional to Hessian norm (i.e., $p(x) \propto |f''_{\sin}(x)|$) for the remaining 40% of training data. In Fig. B.2(b)(c), we visualize the two types of biased training points. Sampling according to the gradient norm or the Hessian norm leads to denser point distribution in the high frequency region $[-2, 0)$: there are $65.35\%, 68.97\%$ of training points fall in $[-2, 0]$ in Fig. B.2(b), (c) respectively. An important difference between Fig. B.2(b) and (c) is that, sampling according to Hessian norm leads to denser points around spikes: there are 18.17% points fall in the yellow area in (b) and 27.45% such points in (c). Those areas around spikes should be more difficult to approximate as the underlying func-

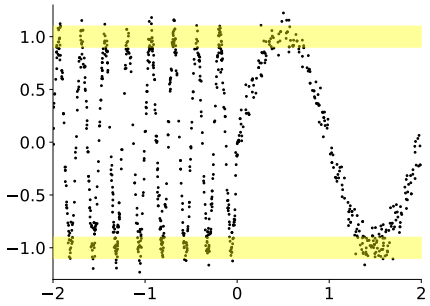
tion changes sharply, which explains the superior performance on the data set biased by Hessian norm. Fig. B.2(a) shows that such biased training datasets provide fast learning, similar to the high frequency biased training datasets in Fig. 6.5.



(a) RMSE vs. number of iterations



(b) biased training points - Gradient



(c) biased training points - Hessian

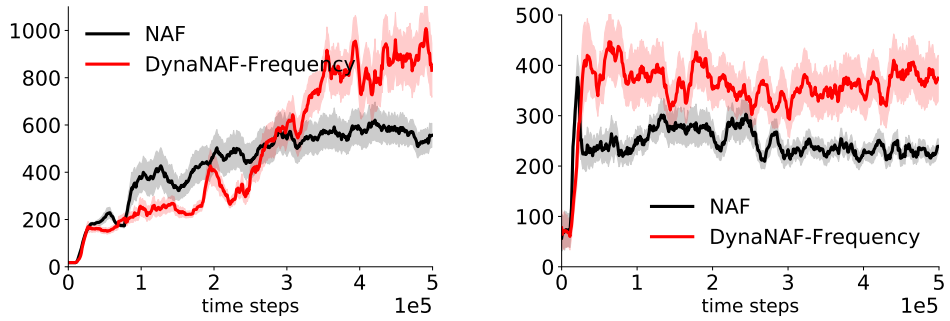
Figure B.2: We show the learning curve of the l_2 regression on three training datasets in Figure (a) and show 1k points uniformly sampled from the two biased training data sets in (b)(c), respectively. The total number of training data points is the same across all experiments. The yellow area includes all the spikes and is defined by restricting $||y| - 1.0| < 0.1$. The testing set is unbiased, and the results are averaged over 50 random seeds with the shade indicating the standard error.

Continuous Control. We show a simple demonstration where our method is adapted to two continuous control tasks: Hopper-v2 and Walker2d-v2 from Mujoco (Todorov et al., 2012) by using a continuous Q learning algorithm called NAF (Normalized Advantage Function) (Gu et al., 2016). The algorithm

parameterizes the action value function as

$$Q(s,a) = V(s) - (a - \mu(s))^T P(a - \mu(s)),$$

where P is a positive semi-definite matrix and hence the action with maximum value can be easily found: $\text{argmax}_a\, Q(s,a) = \mu(s)$. Our search-control strategy naturally applies here by utilizing the value function $V(s)$. From Fig. B.3, one can see that our algorithm (**DynaNAF-Frequency**) finds a better policy comparing with the model-free NAF. We leave the empirical study of applying our strategy to other continuous control algorithms (Lillicrap et al., 2016; Schulman et al., 2017; Fujimoto et al., 2018; Haarnoja et al., 2018; Lim et al., 2018) as a future work.



(a) Hopper-v2 (b) Walker2d-v2

Figure B.3: The learning curves showing sum of rewards per episode as a function of environment time steps. We use 5 planning steps for both algorithm. The results are averaged over 10 random seeds.

B.2.3 Experimental Details

All of our implementations are based on tensorflow with version 1.13.0 (Abadi et al., 2015). For the supervised learning experiment shown in Section 6.4.1, we use mini-batch size $b = 128$, 16×16 tanh units neural network, with learning rate 0.001 for all algorithms. The learning curve is plotted by computing the testing error every 20 iterations. When generating Fig. B.2, in order to sample points according to $p(x) \propto |f'(x)|$ or $p(x) \propto |f''(x)|$, we use $10,000$ even spaced points on the domain $[-2, 2]$ and the probabilities are computed by normalization across the 10k points.

We provide the pseudo-code in Algorithm 11 with sufficient details to recreate our experimental results. Define

$$v_s \stackrel{\text{def}}{=} \nabla_s \max_a Q(s,a),$$

then

$$g_s \stackrel{\text{def}}{=} \nabla_s g(s) = \nabla_s(||\nabla_s \max_a Q(s,a)||_2^2 + ||H_v(s)||_F^2) = \nabla_s(||v_s||_2^2 + ||\nabla_s v_s||_F^2).$$

Note that we use a squared norm to ensure numerical stability when taking gradient. Then for value-based search-control, we use

$$s \leftarrow s + \frac{\alpha}{||\hat{\Sigma}_s v_s||}\hat{\Sigma}_s v_s + X_i, X_i \sim N(0, \eta\hat{\Sigma}_s) \tag{B.6}$$

and for frequency-based search-control, we use

$$s \leftarrow s + \frac{\alpha}{||\hat{\Sigma}_s g_s||}\hat{\Sigma}_s g_s + X_i, X_i \sim N(0, \eta\hat{\Sigma}_s) \tag{B.7}$$

where $\hat{\Sigma}_s$ is empirical covariance matrix estimated from visited states, and we set $\eta = 0.01, \alpha = 0.01$ across all experiments. Notice that comparing with the previous work, we omitted the projection step as we found it is unnecessary in our experiments.

B.3 TD Error-based Search-control

The appendix includes the following contents:

1. Section B.3.1 provides the full proof of Theorem 6 and its simulations.
2. Section B.3.2: a discussion and some empirical study of high power objectives.
3. Section B.3.3: supplementary experimental results: training error results to check the negative effects of the limitations of prioritized sampling; results to verify the equivalence between prioritized sampling and cubic power; results on MazeGridWorld from (Pan et al., 2020b).
4. Section B.3.4: details for reproducible research.

Algorithm 11 Dyna architecture with Frequency-based search-control with additional details

B_s: search-control queue, B: the experience replay buffer
$\mathcal{M} : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \times \mathbb{R}$, the environment model
m: number of search-control samples to fetch at each step
p: probability of choosing value-based hill climbing rule (we set $p = 0.5$ for all experiments)
$\beta \in [0, 1]$: mixing factor in a mini-batch, i.e. βb samples in a mini-batch are simulated from model
n: number of state variables, i.e. $\mathcal{S} \subset \mathbb{R}^n$
ϵ_a: empirically learned threshold as sample average of $||s_{t+1} - s_t||_2/\sqrt{n}$
d: number of planning steps
Q, Q': current and target Q networks, respectively
b: the mini-batch size
τ: update target network Q' every τ updates to Q
$t \leftarrow 0$ is the time step
$n_\tau \leftarrow 0$ is the number of parameter updates
// Gradient ascent hill climbing
With probability $p, 1-p$, choose hill climbing Eq. (B.6) o Eq. (B.7) respectively;
sample s from B_s if choose rule Eq. (B.6), or from B otherwise; set $c \leftarrow 0, \tilde{s} \leftarrow s$
while $c < m$ **do**
    update s by executing the chosen hill climbing rule
    if s is out of state space **then**: // resample the initial state and hill climbing rule
        With probability $p, 1-p$, choose hill climbing rule Eq. (B.6) or Eq. (B.7) respectively;
        sample s from B_s if choose Eq. (6.8), or from B otherwise; set $c \leftarrow 0, \tilde{s} \leftarrow s$
        continue
    if $||s - \tilde{s}||_2/\sqrt{n} > \epsilon_a$ **then**:
        add s to B_s, $\tilde{s} \leftarrow s, c \leftarrow c + 1$
// d planning updates: sample d mini-batches
for d times **do** // d planning updates
    sample βb states from B_s and pair them with on-policy actions, and query $\mathcal{M}$ to get next states and rewards
    sample $b(1-\beta)$ transitions from B an stack these with the simulated transitions
    use the mixed mini-batch for parameter (i.e. DQN) update
    $n_\tau \leftarrow n_\tau + 1$
    if $mod(n_\tau, \tau) == 0$ **then**:
        $Q' \leftarrow Q$
$t \leftarrow t + 1$

B.3.1 Theoretical Proofs

Proofs for Theorem 6

Theorem 6. Consider the following two objectives:

$$\ell_2(x,y) \stackrel{\text{def}}{=} \frac{1}{2}(x-y)^2, \ell_3(x,y) \stackrel{\text{def}}{=} \frac{1}{3}|x-y|^3.$$

Define the functional gradient flow updates on these two objectives:

$$\frac{dx_t}{dt} = -\eta \frac{d\{\frac{1}{2}(x_t-y)^2\}}{dx_t}, \frac{d\tilde{x}_t}{dt} = -\eta \frac{d\{\frac{1}{3}|\tilde{x}_t-y|^3\}}{d\tilde{x}_t}.$$

Define $\delta_t \stackrel{\text{def}}{=} |x_t - y|, \tilde{\delta}_t \stackrel{\text{def}}{=} |\tilde{x}_t - y|$. Given error threshold $\epsilon \geq 0$, define the hitting time $t_\epsilon \stackrel{\text{def}}{=} \min_t\{t : \delta_t \leq \epsilon\}$ and $\tilde{t}_\epsilon \stackrel{\text{def}}{=} \min_t\{t : \tilde{\delta}_t \leq \epsilon\}$. For any initial function value x_0 s.t. $\delta_0 > 1$, $\exists \epsilon_0 \in (0,1)$ such that $\forall \epsilon > \epsilon_0, t_\epsilon \geq \tilde{t}_\epsilon$.

Proof. Basic idea: given the same ϵ and the same initial value of x, first we derive $t_\epsilon = \frac{1}{\eta} \cdot \ln\left\{\frac{\delta_0}{\epsilon}\right\}, \tilde{t}_\epsilon = \frac{1}{\eta} \cdot \left(\frac{1}{\epsilon} - \frac{1}{\delta_0}\right)$. Then we analyze the condition on ϵ to see when $t_\epsilon \geq \tilde{t}_\epsilon$, i.e. minimizing the square error is slower than minimizing the cubic error. The concrete proof is as follows.

For the gradient flow update on the ℓ_2 objective, we have,

$$\begin{aligned}
\frac{d\ell_2(x_t,y)}{dt} &= \frac{d\ell_2(x_t,y)}{d\delta_t} \cdot \frac{d\delta_t}{dx_t} \cdot \frac{dx_t}{dt} \\
&= \delta_t \cdot \text{sign}(x_t - y) \cdot [-\eta \cdot (x_t - y)] \\
&= \delta_t \cdot \text{sign}(x_t - y) \cdot [-\eta \cdot \text{sign}(x_t - y) \cdot \delta_t] \\
&= -\eta \cdot \delta_t^2 = -2 \cdot \eta \cdot \ell_2(x_t,y).
\end{aligned}$$

which implies,

$$\frac{d\{\ln \ell_2(x_t,y)\}}{dt} = \frac{1}{\ell_2(x_t,y)} \cdot \frac{d\ell_2(x_t,y)}{dt} = -2 \cdot \eta.$$

Taking integral, we have,

$$\ln \ell_2(x_t,y) - \ln \ell_2(x_0,y) = -2 \cdot \eta \cdot t,$$

which is equivalent to (letting $\delta_t = \epsilon$),

$$t_\epsilon \stackrel{\text{def}}{=} \frac{1}{2\eta} \cdot \ln\left\{\frac{\ell_2(x_0,y)}{\ell_2(x_t,y)}\right\} = \frac{1}{\eta} \cdot \ln\left\{\frac{\delta_0}{\delta_t}\right\} = \frac{1}{\eta} \cdot \ln\left\{\frac{\delta_0}{\epsilon}\right\}.$$

On the other hand, for the gradient flow update on the ℓ_3 objective, we have,

$$\begin{aligned}\frac{d\ell_3(\tilde{x}_t, y)}{dt} &= \frac{d\ell_3(\tilde{x}_t, y)}{d\tilde{\delta}_t} \cdot \frac{d\tilde{\delta}_t}{d\tilde{x}_t} \cdot \frac{d\tilde{x}_t}{dt} \\ &= \tilde{\delta}_t^2 \cdot \operatorname{sign}(\tilde{x}_t - y) \cdot \left[-\eta \cdot \tilde{\delta}_t^2 \cdot \operatorname{sign}(\tilde{x}_t - y)\right] \\ &= -\eta \cdot \tilde{\delta}_t^4 = -3^{\frac{4}{3}} \cdot \eta \cdot (\ell_3(\tilde{x}_t, y))^{\frac{4}{3}},\end{aligned}$$

which implies,

$$\frac{d\{(\ell_3(\tilde{x}_t, y))^{-\frac{1}{3}}\}}{dt} = -\frac{1}{3} \cdot (\ell_3(\tilde{x}_t, y))^{-\frac{4}{3}} \cdot \frac{d\ell_3(\tilde{x}_t, y)}{dt} = 3^{\frac{1}{3}} \cdot \eta.$$

Taking integral, we have,

$$(\ell_3(\tilde{x}_t, y))^{-\frac{1}{3}} - (\ell_3(\tilde{x}_0, y))^{-\frac{1}{3}} = 3^{\frac{1}{3}} \cdot \eta \cdot t,$$

which is equivalent to (letting $\tilde{\delta}_t = \epsilon$),

$$\tilde{t}_\epsilon \stackrel{\text{def}}{=} \frac{1}{3^{\frac{1}{3}} \cdot \eta} \cdot \left[(\ell_3(\tilde{x}_t, y))^{-\frac{1}{3}} - (\ell_3(\tilde{x}_0, y))^{-\frac{1}{3}}\right] = \frac{1}{\eta} \cdot \left(\frac{1}{\tilde{\delta}_t} - \frac{1}{\delta_0}\right) = \frac{1}{\eta} \cdot \left(\frac{1}{\epsilon} - \frac{1}{\delta_0}\right).$$

Then we have,

$$\begin{aligned}t_\epsilon - \tilde{t}_\epsilon &= \frac{1}{\eta} \cdot \ln\left\{\frac{\delta_0}{\epsilon}\right\} - \frac{1}{\eta} \cdot \left(\frac{1}{\epsilon} - \frac{1}{\delta_0}\right) \\ &= \frac{1}{\eta} \cdot \left[\left(\ln\frac{1}{\epsilon} - \frac{1}{\epsilon}\right) - \left(\ln\frac{1}{\delta_0} - \frac{1}{\delta_0}\right)\right].\end{aligned}$$

Define the function $f(x) = \ln\frac{1}{x} - \frac{1}{x}, x > 0$ is continuous and $\max_{x>0} f(x) = f(1) = -1$. We have $\lim_{x\to 0} f(x) = \lim_{x\to\infty} f(x) = -\infty$, and $f(\cdot)$ is monotonically increasing for $x \in (0, 1]$ and monotonically decreasing for $x \in (1, \infty)$.

Given $\delta_0 > 1$, we have $f(\delta_0) < f(1) = -1$. Using the intermediate value theorem for $f(\cdot)$ on $(0, 1]$, we have $\exists \epsilon_0 < 1$, such that $f(\epsilon_0) = f(\delta_0)$. Since $f(\cdot)$ is monotonically increasing on $(0, 1]$ and monotonically decreasing on $(1, \infty)$, for any $\epsilon \in [\epsilon_0, \delta_0]$, we have $f(\epsilon) \geq f(\delta_0)$.[1] Hence we have,

$$t_\epsilon - \tilde{t}_\epsilon = \frac{1}{\eta} \cdot [f(\epsilon) - f(\delta_0)] \geq 0. \qquad \square$$

[1]Note that $\epsilon < \delta_0$ by the design of using gradient descent updating rule. If the two are equal, $t_\epsilon = \tilde{t}_\epsilon = 0$ holds trivially.

Remark 4. *Figure B.4 shows the function* $f(x) = \ln \frac{1}{x} - \frac{1}{x}, x > 0$. *Fix arbitrary* $x' > 1$, *there will be another root* $\epsilon_0 < 1$ *s.t.* $f(\epsilon_0) = f(x')$. *However, there is no real-valued solution for* ϵ_0. *The solution in* $\mathbb{C}$ *is* $\epsilon_0 = -\frac{1}{W(\log 1/\delta_0 - 1/\delta_0 - \pi i)}$, *where* $W(\cdot)$ *is a Wright Omega function. Hence, finding the exact value of* ϵ_0 *would require a definition of ordering on complex plane. Our current theorem statement is sufficient for the purpose of characterizing convergence rate. The theorem states that there always exists some desired low error level* < 1, *minimizing the square loss converges slower than the cubic loss.*

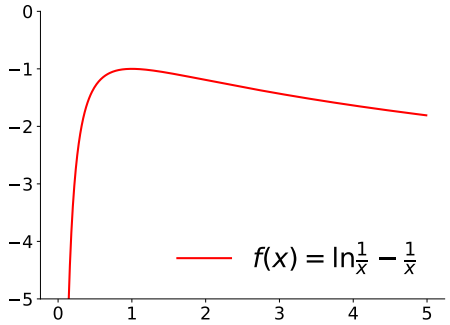
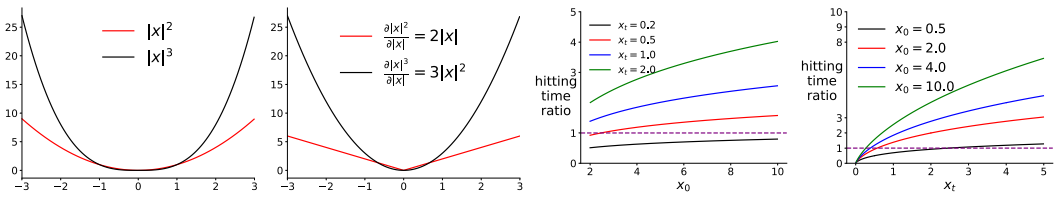


Figure B.4: The function $f(x) = \ln \frac{1}{x} - \frac{1}{x}, x > 0$. The function reaches maximum at $x = 1$.

Simulations. The theorem says that if we want to minimize our loss function to certain small nonzero error level, the cubic loss function offers faster convergence rate. Intuitively, cubic loss provides sharper gradient information when the loss is large as shown in Figure B.5(a)(b). Here we provides a simulation. Consider the following minimization problems: $\min_{x \geq 0} x^2$ and $\min_{x \geq 0} x^3$. We use the hitting time formulae $t_\epsilon = \frac{1}{\eta} \cdot \ln \left\{ \frac{\delta_0}{\epsilon} \right\}, \tilde{t}_\epsilon = \frac{1}{\eta} \cdot \left(\frac{1}{\epsilon} - \frac{1}{\delta_0} \right)$ derived in the proof, to compute the hitting time ratio $\frac{t_\epsilon}{\tilde{t}_\epsilon}$ under different initial values x_0 and final error value ϵ. In Figure B.5(c)(d), we can see that it usually takes a significantly shorter time for the cubic loss to reach a certain x_t with various initial x_0 values.

B.3.2 High Power Loss Functions

We would like to point out that directly using a high power objective in general problems is unlikely to have an advantage.



(a) cubic v.s. square (b) |derivative| (c) x_0 v.s. hitting time (d) x_t v.s. hitting time

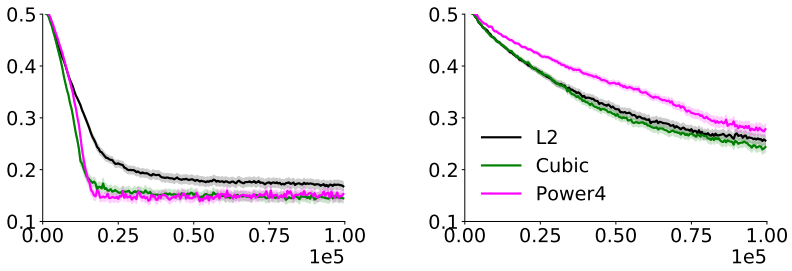
Figure B.5: (a) show cubic v.s. square function. (b) shows their absolute derivatives. (c) shows the hitting time ratio v.s. initial value x_0 under different target value x_t. (d) shows the ratio v.s. the target x_t to reach under different x_0. Note that a ratio larger than 1 indicates a longer time to reach the given x_t for the square loss.

First, notice that our convergence rate is characterized w.r.t. to the expected updating rule, not stochastic gradient updating rule. When using a stochastic sample to estimate the gradient, high power objectives are sensitive to the outliers as they augment the effect of noise. Robustness to outliers is also the motivation behind the Huber loss (Huber, 1964) which, in fact, uses low power error in most places so it can be less sensitive to outliers.

We conduct experiments to examine the effect of noise on using high power objectives. We use the same dataset as described in Section B.3.3. We use a training set with 4k training examples. The naming rules are as follows. **Cubic** is minimizing the cubic objective (i.e. $\min_\theta \frac{1}{n}\sum_{i=1}^n |f_\theta(x_i) - y_i|^3$) by uniformly sampling, and **Power4** is $\min_\theta \frac{1}{n}\sum_{i=1}^n (f_\theta(x_i) - y_i)^4$ by uniformly sampling.

Figure B.6 (a)(b) shows the learning curves of uniformly sampling for Cubic and for Power4 trained by adding noises with standard deviation $\sigma = 0.1, 0.5$ respectively to the training targets. It is not surprising that all algorithms learn slower when we increase the noise variance added to the target variables. However, one can see that *high power objectives is more sensitive to noise variance added to the targets than the regular L2*: when $\sigma = 0.1$, the higher power objectives perform better than the regular L2; after increasing σ to 0.5, Cubic becomes almost the same as L2, while Power4 becomes worse than L2.

Second, it should be noted that in our theorem, we do not characterize the convergence rate to the minimum; instead, we show the convergence rate to a certain low error solution, corresponding to early learning performance. In



(a) $\sigma = 0.1$ (b) $\sigma = 0.5$

Figure B.6: Figure(a)(b) show the testing RMSE as a function of number of mini-batch updates with increasing noise standard deviation σ added to the training targets. We compare the performances of **Power4(magenta)**, **L2 (black)**, **Cubic (forest green)**. The results are averaged over 50 random seeds. The shade indicates standard error. Note that the testing set is not noise-contaminated.

optimization literature, it is known that cubic power would converge slower to the minimizer as it has a relatively flat bottom. However, it may be an interesting future direction to study how to combine objectives with different powers so that optimizing the hybrid objective leads to a faster convergence rate to the optimum and is robust to outliers.

B.3.3 Additional Experiments

In this section, we include the following additional experimental results:

1. Experiments showing the effect of limitations of outdated priorities and insufficient sample space coverage.

2. Supplementary to Figure B.7: the learning performance measured by training errors to show the negative effects of the two limitations.

3. Empirical verification of Theorem 5 (prioritized sampling and uniform sampling on cubic power equivalence).

Negative Effects of the Limitations

We now empirically show that the outdated priorities and insufficient sample space coverage significantly weaken the advantage of the prioritized sampling

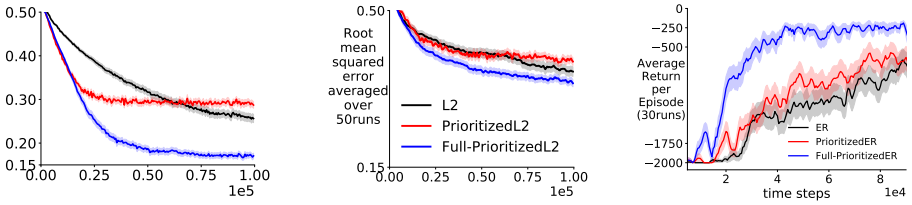
method.

Experiment setup. We conduct experiments on a supervised learning task. We generate a training set $\mathcal{T}$ by uniformly sampling $x \in [-2, 2]$ and adding zero-mean Gaussian noise with standard deviation $\sigma = 0.5$ to the target $f_{\sin}(x)$ values to ensure the learning is reasonably difficult. Define $f_{\sin}(x) \stackrel{\text{def}}{=} \sin(8\pi x)$ if $x \in [-2, 0)$ and $f_{\sin}(x) = \sin(\pi x)$ if $x \in [0, 2]$. The testing set contains 1k samples where the targets are not noise-contaminated. The high frequency region $[-2, 0]$ should have relatively large prediction error and usually takes long time to learn. Hence we expect prioritized sampling to make a clear difference in terms of sample efficiency on this dataset. We use 32×32 tanh layers neural network for all algorithms. We will design experiments to examine the performances of the algorithms with and without the two limitations respectively.

Naming of algorithms. **L2**: the l_2 regression with uniformly sampling from $\mathcal{T}$. **Full-PrioritizedL2**: the l_2 regression with prioritized sampling according to the distribution defined in (6.9), the priorities of *all samples* in the training set are updated after each mini-batch update. **PrioritizedL2**: the only difference with **Full-PrioritizedL2** is that *only* the priorities of those training examples sampled in the mini-batch are updated at each iteration, the rest of the training samples use the original priorities. This resembles the approach taken by the prioritized ER in RL (Schaul et al., 2016). We show the learning curves in Figure B.7.

Outdated priorities. Figure B.7 (a) shows that PrioritizedL2 without updating all priorities can be significantly worse than Full-PrioritizedL2. Correspondingly, we further verify this phenomenon on the classical Mountain Car domain (Brockman et al., 2016). Figure B.7(c) shows the evaluation learning curves of different variants of DQN corresponding to the supervised learning algorithms. We use a small 16×16 ReLu NN as the Q-function, highlighting the issue of priority updating: every mini-batch update potentially perturbs

the values of many other states. Hence many experiences in the ER buffer have the wrong priorities. We do find Full-PrioritizedER performs significantly better.



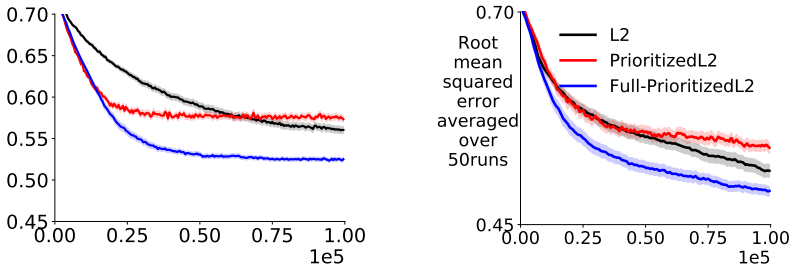
(a) $|\mathcal{T}| = 4000$ (b) $|\mathcal{T}| = 400$ (c) Mountain Car.

Figure B.7: Learning curves showing testing RMSE v.s. number of mini-batch updates. We compare **L2 (black)**, **PrioritizedL2 (red)**, and **Full-PrioritizedL2 (blue)**. (a)(b) show the learning performances trained on a large and small training set respectively. (c) shows the result of a corresponding RL experiment on mountain car domain. We compare episodic return v.s. environment time steps for **ER (black)**, **PrioritizedER (red)**, and **Full-PrioritizedER (blue)**. Results are averaged over 50 random seeds on (a), (b) and 30 on (c). The shade indicates standard error.

Sample space coverage. To check the effect of insufficient sample space coverage, we examine how the relative performances of L2 and Full-PrioritizedL2 change when we train them on a smaller training dataset with only 400 examples as shown in Figure B.7(b). The small training set has a small coverage of the sample space. Unsurprisingly, using a small training set makes all algorithms perform worse than they do on the larger one; however,*it significantly narrows the gap between Full-PrioritizedL2 and L2.* Such narrowed gap indicates that prioritized sampling needs sufficient samples across the sample space to estimate the prioritized sampling distribution.

Training Error Corresponding to Figure B.7

Note that our Theorem 5 and 6 characterize the expected gradient calculated on the training set; hence it is sufficient to examine the learning performances measured by training errors. However, the testing error is usually the primary concern, so we put the testing error in the main body. As a sanity check, we



(a) $|\mathcal{T}| = 4000$ (b) $|\mathcal{T}| = 400$

Figure B.8: Figure (a)(b) show the training RMSE as a function of number of mini-batch updates with a training set containing 4k examples and another containing 400 examples respectively. We compare the performances of **Full-PrioritizedL2 (blue)**, **L2 (black)**, and **PrioritizedL2 (red)**. The results are averaged over 50 random seeds. The shade indicates standard error.

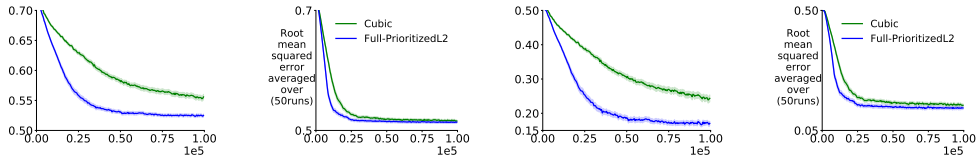
also investigate the learning performances measured by training error and find that those algorithms behave similarly as shown in Figure B.8 where the algorithms are trained by using training sets with decreasing training examples from (a) to (b). As we reduce the training set size, Full-PrioritizedL2 is closer to L2. Furthermore, PrioritizedL2 is always worse than Full-PrioritizedL2. These observations show the negative effects resulting from the issues of outdated priorities and insufficient sample space coverage.

Empirical verification of Theorem 5

Theorem 5 states that the expected gradient of doing prioritized sampling on mean squared error is equal to the gradient of doing uniformly sampling on cubic power loss. As a result, we expect that the learning performance on the training set (note that we calculate gradient by using training examples) should be similar when we use a large mini-batch update as the estimate of the expectation terms become close.

We use the same dataset as described in Section B.3.3 and keep using training size 4k. Figure B.9(a)(b) shows that when we increase the mini-batch size, *the two algorithms Full-PrioritizedL2 and Cubic are becoming very close to each other*, verifying our theorem.

Note that our theorem characterizes the expected gradient calculated on



(a) b=128, $\sigma = 0.5$ (b) b=512, $\sigma = 0.5$ (c) b=128, $\sigma = 0.5$ (d) b=512, $\sigma = 0.5$

Figure B.9: Figure(a)(b) show the training RMSE as a function of number of mini-batch updates with increasing mini-batch size b. Figure (c)(d) show the testing RMSE. We compare the performances of **Full-PrioritizedL2 (blue)**, **Cubic (forest green)**. As we increase the mini-batch size, the two performs more similar to each other. The results are averaged over 50 random seeds. The shade indicates standard error.

the training set; hence it is sufficient to examine the learning performances measured by training errors. However, usually, the testing error is the primary concern. For completeness, we also investigate the learning performances measured by testing error and find that the tested algorithms behave similarly as shown in Figure B.9(c)(d).

B.3.4 Reproducible Research

Our implementations are based on tensorflow with version 1.13.0 (Abadi et al., 2015). We use Adam optimizer (Kingma & Ba, 2014) for all experiments.

Reproduce experiments in Section 6.5

Supervised learning experiment. For the supervised learning experiment shown in Section 6.5.1, we use 32×32 tanh units neural network, with learning rate swept from $\{0.01, 0.001, 0.0001, 0.00001\}$ for all algorithms. We compute the constant c as specified in the Theorem 5 at each time step for Cubic loss. We compute the testing error every 500 iterations/mini-batch updates and our evaluation learning curves are plotted by averaging 50 random seeds. For each random seed, we randomly split the dataset to testing set and training set and the testing set has 1k data points. Note that the testing set is not noise-contaminated.

Reinforcement Learning experiments in Section 6.5.1. We use a particularly small neural network 16×16 to highlight the issue of incomplete priority updating. Intuitively, a large neural network may be able to memorize each state's value and thus updating one state's value is less likely to affect others. We choose a small neural network, in which case a complete priority updating for all states should be very important. We set the maximum ER buffer size as 10k and mini-batch size as 32. The learning rate is 0.001 and the target network is updated every 1k steps.

Distribution distance computation in Section 6.5.3. We now introduce the implementation details for Figure 6.7. The distance is estimated by the following steps. First, in order to compute the desired sampling distribution, we discretize the domain into 50×50 grids and calculate the absolute TD error of each grid (represented by the left bottom vertex coordinates) by using the true environment model and the current learned Q function. We then normalize these priorities to get probability distribution p^*. Note that this distribution is considered as the desired one since we have access to all states across the state space with priorities computed by current Q-function at each time step.

Second, we estimate our sampling distribution by randomly sampling 3k states from search-control queue and count the number of states falling into each discretized grid and normalize these counts to get p_1.

Third, for comparison, we estimate the sampling distribution of the conventional prioritized ER (Schaul et al., 2016) by sampling 3k states from the prioritized ER buffer and count the states falling into each grid and compute its corresponding distribution p_2 by normalizing the counts. Then we compute the distances of p_1, p_2 to p^* by two weighting schemes: 1) on-policy weighting: $\sum_{j=1}^{2500} d^\pi(s_j)|p_i(s_j) - p^*(s_j)|, i \in \{1, 2\}$, where d^π is approximated by uniformly sample 3k states from a recency buffer and normalizing their visitation counts on the discretized GridWorld; 2) uniform weighting: $\frac{1}{2500}\sum_{j=1}^{2500} |p_i(s_j) - p^*(s_j)|, i \in \{1, 2\}$. We examine the two weighting schemes because of two considerations: for the on-policy weighting, we concern about

the asymptotic convergent behavior and want to down-weight those states with relatively high TD error but get rarely visited as the policy gets close to optimal; uniform weighting makes more sense during early learning stage, where we consider all states are equally important and want the agents to sufficiently explore the whole state space.

Computational cost v.s. performance in Section 6.5.3. The setting is the same as we used for Section 6.6. We use plan step/updates=10 to generate that learning curve.

Reproduce experiments in Section 6.6

Common settings. For all discrete control domains other than roundabout-v0, we use 32×32 neural network with ReLu hidden units except the Dyna-Frequency which uses tanh units. This is one of its disadvantages: the search-control of Dyna-Frequency requires the computation of Hessian-gradient product and it is empirically observed that the Hessian is frequently zero when using ReLu as hidden units. Except the output layer parameters which were initialized from a uniform distribution $[-0.003, 0.003]$, all other parameters are initialized using Xavier initialization (Glorot & Bengio, 2010). We use mini-batch size $b = 32$ and maximum ER buffer size 50k. All algorithms use target network moving frequency 1000 and we sweep learning rate from $\{0.001, 0.0001\}$. We use warm up steps 5k (i.e. random action is taken in the first 5k time steps) to populate the ER buffer before learning starts. We keep exploration noise as 0.1 without decaying.

Termination condition on OpenAI environments. On OpenAI, each environment has a time limit and the termination flag will be true if either the time limit reached or the actual termination condition is satisfied. However, theoretically, we should truncate the return if and only if the actual termination condition is satisfied. All of our experiments are conducted by setting discount rate $\gamma = 0.0$ if and only if the actual termination condition is satisfied. For example, on the mountain car, *done = true if and only if* the

position≥ 0.5.

Hyper-parameter settings. Across RL experiments including both discrete and continuous control tasks, we are able to fix the same parameters for our hill climbing updating rule 6.10: $s \leftarrow s + \alpha_h \nabla_s \log |\hat{y}(s) - \max_a Q(s, a; \theta_t)| + X$, where we fix $\alpha_h = 0.1, X \sim N(0, 0.01)$.

For our algorithm Dyna-TD, we are able to keep the same parameter setting across all discrete domains: $c = 20$ and learning rate 0.001. For all Dyna variants, we fetch the same number of states ($m = 20$) from hill climbing (i.e. search-control process) as Dyna-TD does, and use $\epsilon_{accept} = 0.1$ and set the maximum number of gradient step as $k = 100$ unless otherwise specified.

Our Prioritized ER is implemented as the proportional version with sum tree data structure. To ensure fair comparison, since all model-based methods are using mixed mini-batch of samples, we use prioritized ER without importance ratio but half of mini-batch samples are uniformly sampled from the ER buffer as a strategy for bias correction. For Dyna-Value and Dyna-Frequency, we use the setting as described by the previous sections.

For the purpose of learning an environment model on those discrete control domains, we use a 64×64 ReLu units neural network to predict $s' - s$ and reward given a state-action pair s, a; and we use mini-batch size 128 and learning rate 0.0001 to minimize the mean squared error objective for training the environment model.

Environment-specific settings. All of the environments are from OpenAI (Brockman et al., 2016) except that the GridWorld is designed by ourselves. For all OpenAI environments, we use the default setting except on Mountain Car where we set the episodic length limit to 2k. The GridWorld has state space $\mathcal{S} = [0, 1]^2$ and each episode starts from the left bottom and the goal area is at the top right $[0.95, 1.0]^2$. There is a wall in the middle with a hole to allow the agent to pass.

On roundabout-v0 domain, we use 64×64 ReLu units for all algorithms and set mini-batch size as 64. The environment model is learned by using a $200 \times$

200 ReLu neural network trained by the same way mentioned above. For Dyna-TD, we start using the model after 5k steps and set $m = 100, k = 500$ and we do search-control every 50 environment time steps to reduce computational cost. To alleviate the effect of model error, we use only 16 out of 64 samples from the search-control queue in a mini-batch.

On Mujoco domains Hopper and Walker2d, we use 200×100 ReLu units for all algorithms and set mini-batch size as 64. The environment model is learned by using a 200×200 ReLu neural network trained by the same way mentioned above. For Dyna-TD, we start using the model after 10k steps and set $m = 100, k = 500$ and we do search-control every 50 environment time steps to reduce computational cost. To alleviate the effect of model error, we use only 16 out of 64 samples from the search-control queue in a mini-batch.

Appendix C

Other Efforts to Improve Sample Efficiency

This section introduces other efforts that have been made but are not detailed in this thesis to improve sample efficiency. Improving sample efficiency has been one of the central research topics in machine learning. This thesis only explores a small corner in this big research thrust. Below includes some discussions about our efforts towards improving sample efficiency from several other perspectives: 1) reducing representation interference (also called catastrophic forgetting); 2) bias/variance control; 3) semi-parametric method for model learning; 4) regularization.

First, representation learning in online learning systems can strongly impact learning efficiency, both positively due to generalization but also negatively due to interference (Liang et al., 2016; Le et al., 2017; Liu et al., 2019; Chandak et al., 2019). Neural networks particularly suffer from interference—where updates for some inputs degrade accuracy for others—when training on temporally correlated data (McCloskey & Cohen, 1989; French, 1999; Kemker et al., 2018). To mitigate the catastrophic forgetting, we propose a novel sparse representation learning technique (Pan et al., 2021). Recent work has shown that sparse representations-where only a small percentage of active units- can significantly reduce interference. Those works, however, relied on relatively complex regularization or meta-learning approaches that have only been used offline in a pre-training phase. We pursue a direction that achieves sparsity by design rather than by learning. Specifically, we design an activation function

that produces sparse representations deterministically by construction and so is more amenable to online training. The idea relies on the simple binning approach. However, it overcomes the two key limitations of binning: zero gradients for the flat regions almost everywhere and lost precision—reduced discrimination—due to coarse aggregation. We introduce a Fuzzy Tiling Activation that provides non-negligible gradients and produces overlap between bins that improves discrimination.

Second, we propose a novel Maxmin Q-learning to resolve the overestimation issue of value-based control algorithm (i.e., Q-learning) (Lan et al., 2020), which is known to impede the quality of the learned policy (Thrun & Schwartz, 1993; Szita & Lőrincz, 2008; Strehl et al., 2009; Hasselt, 2010; Kumar et al., 2020). Although algorithms have been proposed to reduce overestimation bias (Hasselt, 2010; Lee et al., 2013; Anschel et al., 2017; Zhang et al., 2017; Fujimoto et al., 2018), we lack an understanding of how bias interacts with performance and the extent to which existing algorithms mitigate bias. We 1) highlight that the effect of overestimation bias on learning efficiency is environment-dependent; 2) propose a generalization of Q-learning, called *Maxmin Q-learning*, which provides a parameter to control bias flexibly; 3) show theoretically that there exists a parameter choice for Maxmin Q-learning that leads to unbiased estimation with a lower approximation variance than Q-learning; and 4) prove the convergence of our algorithm in the tabular case, as well as the convergence of several previous Q-learning variants, using a novel Generalized Q-learning framework. We empirically verify that our algorithm better controls estimation bias in toy environments and achieves superior performance on several benchmark problems.

Third, in the MBRL setting, we develop a more efficient model learning technique by semi-parametric method (Pan et al., 2018; Schlegel et al., 2017), called Reweighted Experience Models (REMs), that makes it simple to sample next states or predecessors. We demonstrate that Dyna with such a model exhibits advantages over replay-based methods in learning in continuous state problems. The performance gap grows when moving to larger stochastic domains of increasing size.

Last, we attempt to obtain sample efficient learning by explicitly capturing regularities in the action space for a special class of partial differential equation control problems, where there can be arbitrarily high-dimensional continuous action space (Pan et al., 2018). In particular, we propose the concept of action descriptors, which encode regularities among spatially extended action dimensions and enable the agent to control high-dimensional action PDEs. We provide theoretical evidence suggesting that this approach can be more sample efficient than a conventional approach that treats each action dimension separately and does not explicitly exploit the spatial regularity of the action space. The action descriptor approach is then used within the deep deterministic policy gradient algorithm. Experiments on two PDE control problems, with up to 256-dimensional continuous actions, show the advantage of the proposed approach over the conventional one.