VALUE LEARNING THROUGH BELLMAN RESIDUALS AND NEURAL FUNCTION APPROXIMATIONS IN DETERMINISTIC SYSTEMS

BY

EHSAN SALEH

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2021

Urbana, Illinois

Adviser:

Professor Timothy M. Bretl

ABSTRACT

Deterministic Bellman residual minimization is the problem of learning values in a Markov Decision Process (MDP) by optimizing for the Bellman residuals directly without using any heuristic surrogates. Compared to the traditional Approximate Dynamic Programming (ADP) methods, this approach can have both advantages and disadvantages. One of the advantages of such an approach is that the underlying optimization problem can provably converge to the desired solution with better theoretical sample efficiency guarantees, while ADP heuristics are prone to divergence and have worse theoretical sample efficiency. On the other hand, the disadvantages of Bellman residual minimization is the requirement of two independent future samples, a.k.a. the *double sampling* requirement, in order to form an unbiased estimate of the Bellman residual. Despite that some versions of BRM have superior theoretical properties, the superiority comes from the double sampling trick, limiting their applicability to simulator environments with state resetting functionality. Hence the algorithms cannot be applied to non-simulator environments where state resetting is not available. This requirement can trivially be waived for deterministic dynamics, since any combination of observations and actions are guaranteed to generate identical future observations. For this *double sampling* requirement, Bellman residual minimization methods tended to be overlooked in the literature, and this work tries to evaluate the efficacy of this approach compared to the more common ADP heuristics.

In this work, we make a simple observation that BRM can be applied without state resetting if the environment is deterministic, which Baird [2] has hinted in his original paper. The resulting algorithm is simple, convergent, and works well in benchmark control problems. Also, its implementation could be as simple as changing a line of code in their ADP counterparts. We compare Q-learning to its DBRM version theoretically, confirm the theoretical predictions from experiments, and also discover some surprising empirical behaviors and provide explanations.

"To my beloved parents."

ACKNOWLEDGMENTS

I would like to thank Professor Nan Jiang, whose help and supervision was key in this project and without his expertise in all the steps, this work would not have been possible. This work was published in the Optimization Foundations for Reinforcement Learning Workshop of the Neural Information Processing Systems Conference [26], and utilized resources supported by the National Science Foundation's Major Research Instrumentation (MRI) program, grant Number 1725729.

TABLE OF CONTENTS

Chapter 1: INTRODUCTION

In reinforcement learning (RL) with function approximation, there are two general strategies to addressing the *temporal credit assignment* problem and learning (optimal) value functions: approximate dynamic programming (ADP [6]; e.g., Q-learning and TD), which learns from bootstrapped targets, and Bellman residual minimization (BRM; e.g., residual gradient [2]), which minimizes the Bellman residual directly.

A crucial distinction between the two approaches is that BRM methods require the *double sampling* trick to form an unbiased estimate of the Bellman residual,[1] that is, these algorithms require two i.i.d. samples of the next-state from the same state-action pair. While BRM with double sampling enjoys some superior properties compared to DP (see Section 3), the double sampling operation requires state resetting and is only available in relatively simple simulated environments. When state resetting is not available, can we still run BRM algorithms?

While this is difficult in general, we note that in *deterministic* environments, double sampling can be trivially implemented without state resetting as a second i.i.d. sample would be identical to the first one. This gives rise to very simple and convergent algorithm, which we call *Deterministic Bellman Residual Minimization* (DBRM) and should work well in (nearly) deterministic environments. Given that many empirical RL benchmarks are deterministic or only mildly stochastic [9], it is surprising that such a method has not received any attention besides being lightly mentioned in Baird [2]'s original paper (to the best of our knowledge).

We fill this hole in literature by providing a study of the DBRM algorithm. In particular, we consider Q-learning as a representative ADP algorithm, and compare it to its DBRM counterpart both theoretically and empirically. Below is a summary of our results:

1. We review and compare the theoretical properties of ADP and DBRM, showing that DBRM has superior approximation guarantees and can also handle mild stochasticity (Section 3).

2. On benchmark control problems with deterministic dynamics, we show that on-policy

[1] There are BRM algorithms that do not require double sampling, such as modified BRM [1] and SBEED [11], but they are closer in relation to ADP than to BRM with double sampling in terms of the approximation guarantees. See Section 3.2 for further discussions.

DBRM simply works as expected,[2] sometimes outperforming Q-learning (Section 4.1). As a side-product, our results also shed light on the approximation guarantees of Q-learning.

3. We inject different levels of stochasticity into the environment and observe DBRM to gradually break down, again as expected from theory. As another side-product, we argue that determinism of dynamics is ill-defined in the usual sense, and DBRM provides a way to test such determinism in a more canonical manner (Section 3.2 and 6).

4. We also compare Q-learning and DBRM on off-policy data (Section 4.3). Perhaps surprisingly, DBRM completely brakes down in Acrobot while the environment is fully deterministic. By further investigation, we find that DBRM is more sensitive to non-exploratory data than ADP methods, and we provide a concrete tabular MDP example to illustrate the difference in their behaviors. We also propose a preliminary solution to the issue by promoting a residual minimization strategy that is more robust to distribution mismatch (Section 4.3.2).

[2] Q-learning and its DBRM counterpart are both off-policy algorithms. See the meaning of "on-policy" in Section 4.

Chapter 2: BACKGROUND

2.1 MARKOV DECISION PROCESS (MDP) PRELIMINARIES

An infinite-horizon discounted Markov Decision Process (MDP) is often specified by $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function, $R : \mathcal{S} \times \mathcal{A} \to [0, R_{\max}]$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor (where $\Delta(\mathcal{F})$ denotes the set of all probability distributions over $\mathcal{F}$, otherwise known as the Credal set of $\mathcal{F}$). A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ specifies a distribution over actions for each state. We call an MDP finite, if both its state and action spaces are finite sets. A random trajectory $\tau = (s_1, a_1, s_2, a_2, \cdots)$ can be induced from a policy π given a starting state s as described in Algorithm 2.1.

The expected sum of discounted rewards as a function of starting state s is called the value function of π, that is,

$$V^\pi(s) := \mathbb{E}[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid s_1 = s; \pi]. \tag{2.1}$$

We can define $Q^\pi(s, a)$ similarly by conditioning on $s_1 = s$ and $a_1 = a$:

$$Q^\pi(s, a) := \mathbb{E}[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid s_1 = s, a_1 = a; \pi]. \tag{2.2}$$

In an infinite-horizon discounted MDP, there always exists an optimal policy $\pi^\star$ that maximizes V^π and Q^π for all starting states (and actions) simultaneously, and as a shorthand we let $V^\star := V^{\pi^\star}$, $Q^\star := Q^{\pi^\star}$. In particular, $Q^\star$ satisfies the Bellman optimality equation $Q^\star = \mathcal{T} Q^\star$, where $\mathcal{T} : \mathbb{R}^{\mathcal{S} \times \mathcal{A}} \to \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$ is defined as

$$(\mathcal{T} f)(s, a) := R(s, a) + \gamma \mathbb{E}_{s' \sim P(s,a)}[\max_{a' \in \mathcal{A}} f(s', a')]. \tag{2.3}$$

It is known that $\mathcal{T}$ is a γ-contraction under ℓ_∞, hence repeatedly applying $\mathcal{T}$ to any function will eventually converge to $Q^\star$, known as the value iteration algorithm [25].

Algorithm 2.1: Sampling Trajectories in an MDP

Require: The initial state s.
Require: The policy π.
1: Initialize the state $s_1 = s$.
2: **for** $i = 1, 2, \ldots$ **do**
3: Sample the current action from the policy π given the current state s: $a_i \sim \pi(s_i)$.
4: Take a step in the transition dynamics to generate the next state: $s_{i+1} \sim P(s_i, a_i)$.
5: **end for**
6: **return** The trajectory $\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \cdots)$.

2.2 THE METHOD OF LEARNING FROM TEMPORAL DIFFERENCES

The Temporal-Differences (TD) learning procedure of Sutton [30] for predicting state values is formally described in Algorithm A.2 of Appendix A. This method builds upon the Least Mean Squared (LMS) adaptive filter of Widrow and Stearns [36], which is shown in Figure A.1 and is formally defined in Algorithm A.1 of the appendix. The discussion into the connection between the LMS filter and the TD-learning method, and their commonly used *delta update rule* is left to Section A.1 of the appendix as well. In short, the TD-learning method can be tuned to either (1) act as a Monte-Carlo value learner on one extreme, or (2) perform value iteration by bootstrapping to approximate the Bellman operator on the other extreme. This can be done using a continuous control parameter λ, hence justifying the TD(λ) naming of the algorithm. Figure A.2 of the appendix visualizes the effect of the λ parameter on the resulting algorithmic behavior. The rest of our discussion focuses on the bootstrapping extreme of the TD-learning method, which is commonly known as the TD(0) variant. This algorithm is the backbone of the Q-learning policy training method, which we will analyze and compare against our DBRM method.

2.3 Q-LEARNING AND FUNCTION APPROXIMATION

When the state of an MDP is high-dimensional and/or continuous, function approximation is often deployed to generalize over the state space. Let f_θ denote a parameterized Q-value function, and the goal of the algorithm is to find θ such that $f_\theta \approx Q^\star$. Let Θ be the set of possible parameter values, and $\mathcal{F} := \{f_\theta : \theta \in \Theta\}$ be the corresponding function class. We

say $\mathcal{F}$ is *realizable* if it captures the $Q^\star$ of the MDP, that is, $Q^\star \in \mathcal{F}$.

Given a dataset of transition tuples $D = \{(s, a, r, s')\}$, define

$$\mathcal{L}_D(f; f') := \frac{1}{|D|} \sum_{(s,a,r,s') \in D} \left(f(s, a) - r - \gamma \max_{a' \in \mathcal{A}} f(s', a')\right)^2 . \tag{2.4}$$

For the ease of exposition we will assume (s, a) pairs are drawn i.i.d. from some fixed data distribution in our theoretical arguments, although the real data is more complicated and the (s, a) pairs can be correlated and nonstationary. $\mathcal{L}_D(f; f')$ is the loss of f as a predictor for a regression problem, where the input is (s, a) and the expected value of the label is $(\mathcal{T} f)(s, a)$. This motivates Fitted Q-Iteration [15],

$$f_k \leftarrow \arg\min_{f \in \mathcal{F}} \mathcal{L}_D(f; f_{k-1}), \tag{2.5}$$

which solves the regression problem over $\mathcal{F}$ to approximate a value iteration step. Q-learning can be viewed as the stochastic approximation of FQI, that is,

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}_D(f_\theta; f_\theta^-), \tag{2.6}$$

where f_θ^- is the same as f_θ except that the former is treated as a constant function in gradient calculation, and η is the learning rate. Also note that any gradient term of the form $\nabla_\theta \mathcal{L}(\cdot)$, can always be replaced by its stochastic gradient on a single (s, a, r, s') tuple or a minibatch.

Chapter 3: DETERMINISTIC BELLMAN RESIDUAL MINIMIZATION (DBRM)

3.1 THE ALGORITHM

The DBRM version of Q-learning is simply

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}_D(f_\theta; f_\theta). \tag{3.1}$$

Comparing to Eq.(2.4), the only difference is that we now pass gradient into the second f_θ. This essentially corresponds to applying gradient descent to the optimization problem

$$\arg\min_{f \in \mathcal{F}} \mathcal{L}_D(f; f). \tag{3.2}$$

Most ADP algorithms (including FQI and Q-learning) can diverge under function approximation [18, 34, 35], as they are fundamentally iterative algorithms (Eq.(2.5)) and lack a globally consistent objective. In contrast, DBRM is obviously convergent as it simply minimizes the objective in Eq.(3.2). So why don't we use it?

3.2 THEORETICAL ANALYSES AND COMPARISONS

To find $f_\theta \approx Q^\star$ we need to minimize the Bellman residual $\|f - \mathcal{T}f\|$. The problem with DBRM in stochastic environments is that Eq.(3.2) is in general a biased estimation of the Bellman residual. To see why, let $\mathcal{L}(f; f) := \mathbb{E}_D[\mathcal{L}_D(f; f)]$, and we have

$$\mathcal{L}(f; f) = \mathbb{E}_{(s,a,r,s')}[(f(s,a) - (\mathcal{T}f)(s,a))^2] + \gamma \mathbb{E}_{(s,a)}[\mathbb{V}_{s' \sim P(s,a)}[\max_{a' \in \mathcal{A}} f(s', a')]], \tag{3.3}$$

The first term on the RHS is the Bellman residual under the marginal distribution of (s, a) in the data (see our i.i.d. assumption of (s, a) in Section 2), which is the desired quantity. The trouble is in the second term, which inappropriately penalizes f with large variance under the transition distributions.

Indeed, prior works have tried to overcome this difficulty, by either approximating the

second term and subtracting it [1, 11, 23], or avoiding it to appear in the first place with double sampling [2, 21]. None of these are necessary in deterministic environments where $P(s,a)$ is a point mass and $\mathbb{V}_{s'\sim P(s,a)}[\cdot] = 0$, and DBRM can be viewed as a special case of BRM with double sampling in this case.

In fact, we show that DBRM works under more general conditions: As long as $\mathbb{V}_{s'\sim P(s,a)}[V^\star(s')]$ is small, DBRM can provably find a good approximation to $Q^\star$, as shown next:

Proposition 3.1. Let $\epsilon_{\text{dtmn}} := \gamma\mathbb{E}_{(s,a)}[\mathbb{V}_{s'\sim P(s,a)}[V^\star(s')]]$. Fixing any $\hat{f} \in \mathcal{F}$, define $\epsilon_{\text{estm \& opt}} := \mathcal{L}(\hat{f};\hat{f}) - \min_{f\in\mathcal{F}} \mathcal{L}(f;f)$. Then if $Q^\star \in \mathcal{F}$, we have

$$\|\hat{f} - \mathcal{T}\hat{f}\| := \mathbb{E}_{(s,a,r,s')}[(\hat{f}(s,a) - (\mathcal{T}\hat{f})(s,a))^2] = \epsilon_{\text{dtmn}} + \epsilon_{\text{estm \& opt}}.$$

Proof. LHS $\leq \mathcal{L}(\hat{f};\hat{f}) \leq \mathcal{L}(Q^\star;Q^\star) + \epsilon_{\text{estm \& opt}}$

$$= \|Q^\star - \mathcal{T}Q^\star\| + \gamma\mathbb{E}_{(s,a)}[\mathbb{V}_{s'\sim P(s,a)}[\max_{a'\in\mathcal{A}} V^\star(s')]] + \epsilon_{\text{estm \& opt}} \leq \epsilon_{\text{dtmn}} + \epsilon_{\text{estm \& opt}}. \qquad \text{QED.}$$

In the proposition we use $\epsilon_{\text{estm \& opt}}$ to summarize the error due to the finite sample effect and imperfect optimization, in the sense that with infinite data we would have $\mathcal{L} = \mathcal{L}_D$ (assuming uniform convergence of $\mathcal{F}$); plus the exact optimization of Eq.(3.2), we would have $\epsilon_{\text{estm \& opt}} = 0$. The proposition shows that $\hat{f}$ has small Bellman error on the data distribution if the environment is mildly stochastic and estimation & optimization errors are small. Note that small Bellman error of $\hat{f}$ can be easily translated into the performance guarantees of its greedy policy, which we do not detail here [1, 10].

One important implication of Proposition 3.1 is in the way we measure stochasticity, as requiring ϵ_{dtmn} to be small is much more lenient than requiring deterministic dynamics. For example, if the stochastic factors of the environment do not interfere with the agent's decision-making process (e.g., ignoring them results in approximate bisimulations [20]), Proposition 3.1 still guarantees that Eq.(3.2) is a sensible objective for learning. Moreover, as we will argue in Section 6, determinism of dynamics is an ill-defined notion in the first place, and running DBRM may be a better way of testing it.

Another important property, which is also shared by the versions that require double

sampling, is that *DBRM only requires realizability* as the representation assumption on $\mathcal{F}$, while ADP methods are known to diverge or suffer from exponential sample complexity under realizability [12]. Indeed, finite sample analyses of ADP algorithms often characterize the approximation error of $\mathcal{F}$ as $\inf_{f \in \mathcal{F}} \sup_{f' \in \mathcal{F}} \|f - \mathcal{T} f'\|$, known as the inherent Bellman error [23]. When assumed to be 0, this corresponds to requiring $\mathcal{F}$ to be closed under Bellman update ($\forall f \in \mathcal{F}, \mathcal{T} f \in \mathcal{F}$), a condition far stronger than realizability [10]. We will see empirical results consistent with these theoretical predictions later in Section 4.1, where Q-learning finds value functions that are significantly further away from $Q^\star$ than DBRM.

Given the superior approximation guarantees of DBRM, one would naturally wonder: does it actually work in practice?

Chapter 4: EMPIRICAL EVALUATIONS AND ANALYSIS

In this chapter, we verify the performance of DBRM and compare it to Q-learning empirically on standard control benchmarks. As we will see, DBRM (the on-policy version; see definition below) works very well and compares favorably to Q-learning, and sometimes achieves significantly better results thanks to its stable optimization and superior approximation power.

4.1 ON-POLICY EXPERIMENTS IN DETERMINISTIC ENVIRONMENTS

4.1.1 Environments

We conduct all our experiments in this work on 3 classical control domains: Mountain-Car, Cart-Pole, and Acrobot from the OpenAI Gym collection [9]. These domains have continuous state space of dimensions 2, 4, and 6, and discrete action spaces of cardinalities 3, 2, and 3, respectively.

4.1.2 Function Approximation

For both DBRM and Q-learning[22], we use the same neural network architecture. The neural net consists of two hidden layers, each with 64 neurons. The input and output dimensions are determined by the state space dimension and the action space size of each individual domain. Tanh is used as the activation function.

4.1.3 Exploration Scheme

In this section we compare the "on-policy" versions of Q-learning and DBRM;[1] the comparison of their off-policy versions will be done in Section 4.3.2. On-policy agent uses the

[1]Both algorithms are off-policy algorithms. By "on-policy", we mean that the data collection policy is determined by the algorithm itself, so different algorithms will collect different datasets. In contrast, we will compare the algorithms in the off-policy mode in Section 4.3.2, where all algorithms use exactly the same dataset.

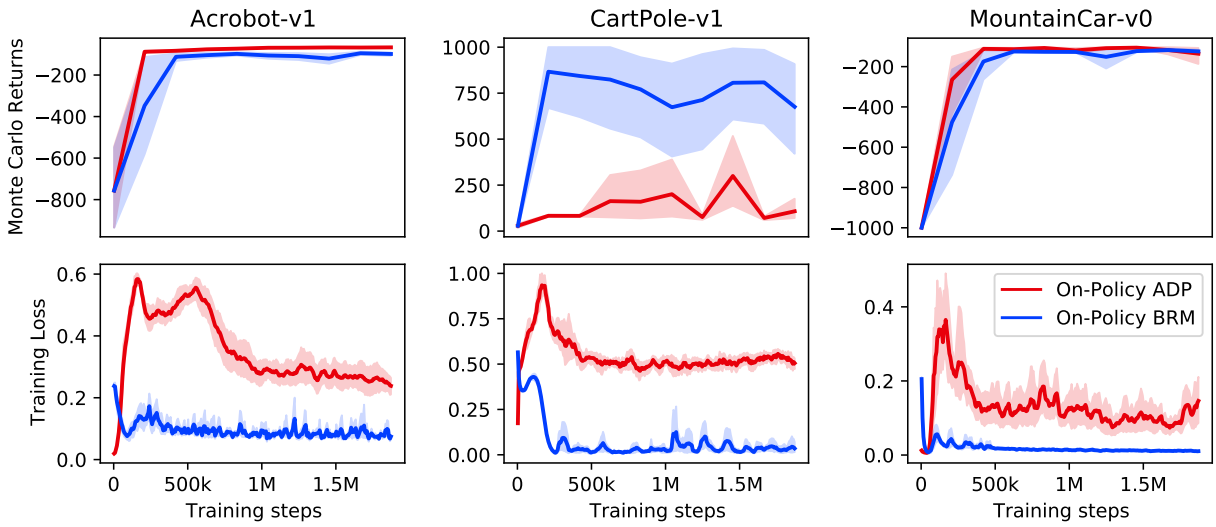


Figure 4.1: Comparison between Q-learning and DBRM (on-policy). The top row shows the Monte-Carlo evaluation results, and the bottom row shows the training loss. All error bars in this work show a 95% confidence interval for the mean using 1000 bootstrap samples.

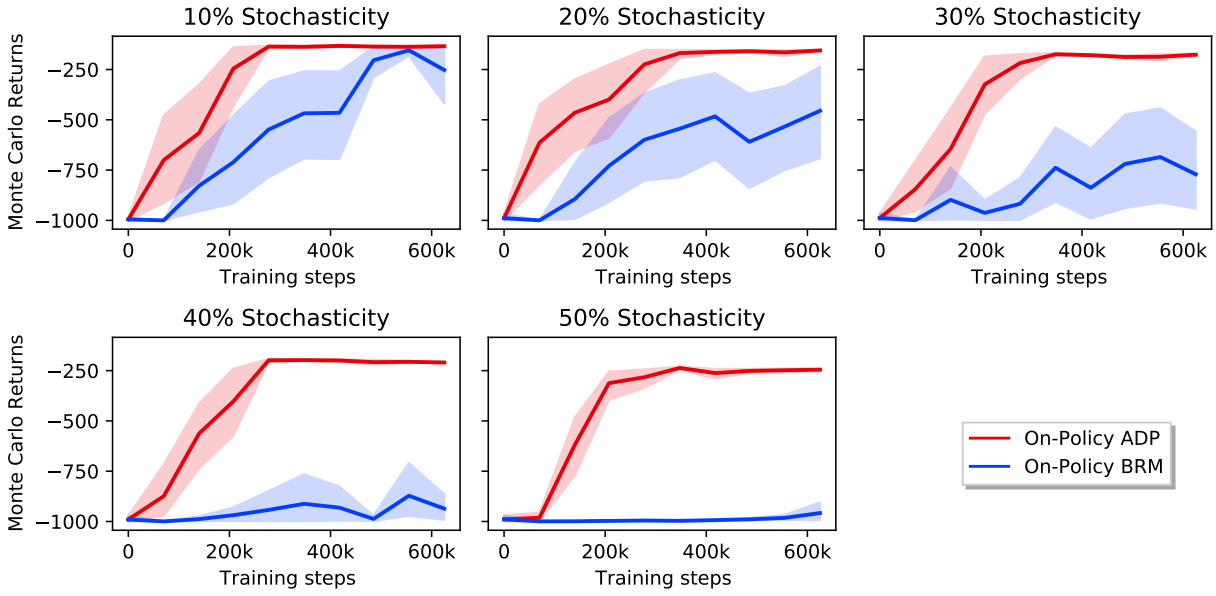


Figure 4.2: Injecting different levels of stochasticity to the Mountain-Car environment, and observing the performance degradation trend of on-policy BRM, while on-policy TD maintains a clear lead.

standard ϵ-greedy strategy for exploration, that is, taking uniformly random actions with probability ϵ. ϵ starts with 0.1, ends with 0.02, and is linearly interpolated during training.

4.1.4 Algorithm Details

Our on-policy agents maintain a replay buffer consisting of $50,000$ most recent samples collected. The mini-batch size and the learning rate are fixed at 32 and 5×10^{-4}, respectively, with the Adam optimizer used in all experiments. The discount factor was set at 0.99. In the on-policy settings, learning did not start until after 1000 samples were collected. Q-learning needs a target network for bootstrap sampling, which is updated every 500 training steps. (BRM does not need a target network.) Similar to popular implementations, we use huber loss instead of squared loss for Q-learning. For BRM we use the squared loss just to avoid undesired disconnections from the theory; we did experiment with the huber loss and the behavior of the algorithms was qualitatively similar.

4.1.5 Evaluation

During training, around every 3000 training steps (or 6000 steps, depending on the total duration) we freeze training and perform Monte-Carlo policy evaluation on the current policy (with ϵ=0.02). The evaluation is conducted for a total of 11 times during the training period, including the beginning and the end. During the evaluation, there is no discounting, the episode length is capped at 1000, and the total rewards are averaged across 100 trajectories with randomly generated initial states. Besides MC return, we also record the training losses $(f(s,a) - r - \max_{a' \in \mathcal{A}} f(s', a'))^2$ and smooth the curves with a window size of 6250.

4.1.6 Results

We plot the MC returns and the training losses averaged over 10 runs in Figure 4.1. As we can see, DBRM's performance compares favorably to Q-learning in Acrobot and MountainCar, and significantly outperforms the latter in CartPole.

Another interesting observation comes from the Acrobot and MountainCar results, where

Q-learning and DBRM achieve similar performance via very different solutions. In particular, DBRM finds value-functions with very low squared losses $\mathcal{L}_D(f; f)$ as fully expected. Q-learning finds functions with much higher losses, which is also expected from the theory (Section 3.2). However, while the functions learned by Q-learning are further away from $Q^\star$ than DBRM, the resulting policies are equally performing. We will leave the investigation of this intriguing phenomenon to future work.

4.2 EXPERIMENTS IN STOCHASTIC ENVIRONMENTS

In this subsection we test the robustness of DBRM against varying extents of violation to the deterministic assumption. To have a set of comparable and well controlled experiments, we convert the environments used in Section 4 into stochastic environments by redefining the action space to contain inherent randomness. In particular, the environments are manipulated to take a uniformly random action with a certain probability regardless of the action commanded by the agent. For instance, when the stochasticity level is 0.4, it meant that 60% of the actions taken are exactly the ones commanded to the environment, and the rest 40% taken actions were uniformly random. Note that this is different from the random actions in ϵ-greedy exploration, as the agent cannot observe the actual action taken, and only has access to the redefined actions, encompassing the randomness as part of the environment inherently.

4.2.1 Results

We show the results for the MountainCar environment in Figure 4.2. As we increase the level of stochasticity, Q-learning's performance degenerates slightly as the task becomes genuinely more difficult to control. In contrast, the performance of DBRM drops gradually but also significantly as stochasticitiy increases, which are consistent with the theoretical predictions. As the stochasticity increases, the natural double-sampling trick within deterministic environments (which only required a single sample in practice) gradually becomes ineffective. Since BRM relies on the double-sampling trick to form unbiased estimates of the Bellman residual, DBRM's performance starts to deteriorate.

4.3 ON THE SENSITIVITY OF DBRM TO OFF-POLICYNESS AND DISTRIBUTION MISMATCH

While the results in Section 4 are informative, the comparisons are not fully apple-to-apple: The "on-policy" experiments introduce a distracting factor, that the two algorithms collect their own datasets. Since we are mostly interested in the capability of Q-learning and DBRM in solving the temporal credit assignment problem, such a distracting factor makes it difficult to determine if the differences in performance should be attributed to the algorithms or the datasets. Therefore, it is important that we compare their performance in a completely off-policy manner.

To compare the two algorithms off-policy, in each trial of the experiment we first run Q-learning with ϵ-greedy exploration for 1.875 million steps to collect a dataset, and feed this dataset to two agents: one running Q-learning and one running DBRM, both in a completely off-policy manner. The rest of the settings are exactly the same as in Section 4.1. Note that the off-policy Q-learning agent would not exactly reproduce the behavior of the on-policy agent that generated data, as the on-policy agent generates the dataset in a rather non-stationary fashion (later trajectories are generally closer to optimal), while the off-policy agent has access to the entire dataset from the very beginning.

4.3.1 Results

The results are shown in Figure 4.3. The curves for Q-learning look similar to their on-policy counterparts in Figure 4.1, showing that Q-learning is not sensitive to off-policy data and is consistent with that reported by Fu et al. [16]. On the other hand, we see that DBRM's performance degrades significantly in CartPole and MountainCar, and the agent completely breaks down in Acrobot.

Comparing this surprising negative result to the success of DBRM in Section 4.1, we speculate that it is likely due to the sensitivity of DBRM to off-policy data and distribution mismatch—that the data distribution is different from that induced by the agent's policy. While distribution mismatch is in general an issue to most RL algorithms, it has been reported that Q-learning is not very sensitive to off-policyness of the data Fu et al. [16]. In fact, we

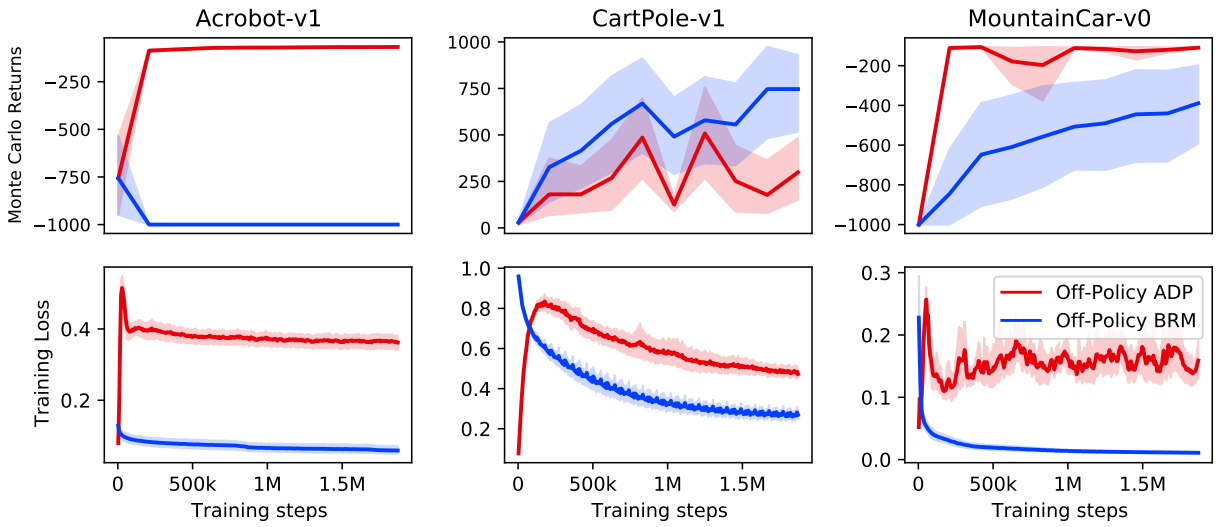


Figure 4.3: The off-policy ADP and BRM methods performance comparison. The top row shows the Monte-Carlo evaluation during 10 points of training. The bottom row shows the training loss used in each optimization method.

are able to construct an example, shown in Figure 4.4, where distribution mismatch is in its most severe form—non-exploratory data.

While all algorithms will generally fail under non-exploratory data, Q-learning may still give sensible solutions sometimes, especially when the values of state-action pairs with missing data are initialized in a pessimistic manner [17]. Indeed, in the specific example considered in Figure 4.4, we only have data about taking action "R" in all states and know nothing about the consequences of taking "L". With appropriately initialized values, Q-learning still converges to a sensible estimation and yields a policy that takes actions "R", as the influence of the pessimistic values of "L" is blocked by the max operator and never propagates through the Bellman updates. With the same initialization and the same dataset, however, DBRM learns a value function far off the track, and the values of "L" do influence the learning process and cause wide estimates.

4.3.2 Preliminary Solution: Objectives with Higher Norms

We propose a preliminary idea to mitigate the sensitivity of DBRM to distribution mismatch and provide primary results validating the idea. Since the distribution mismatch is often

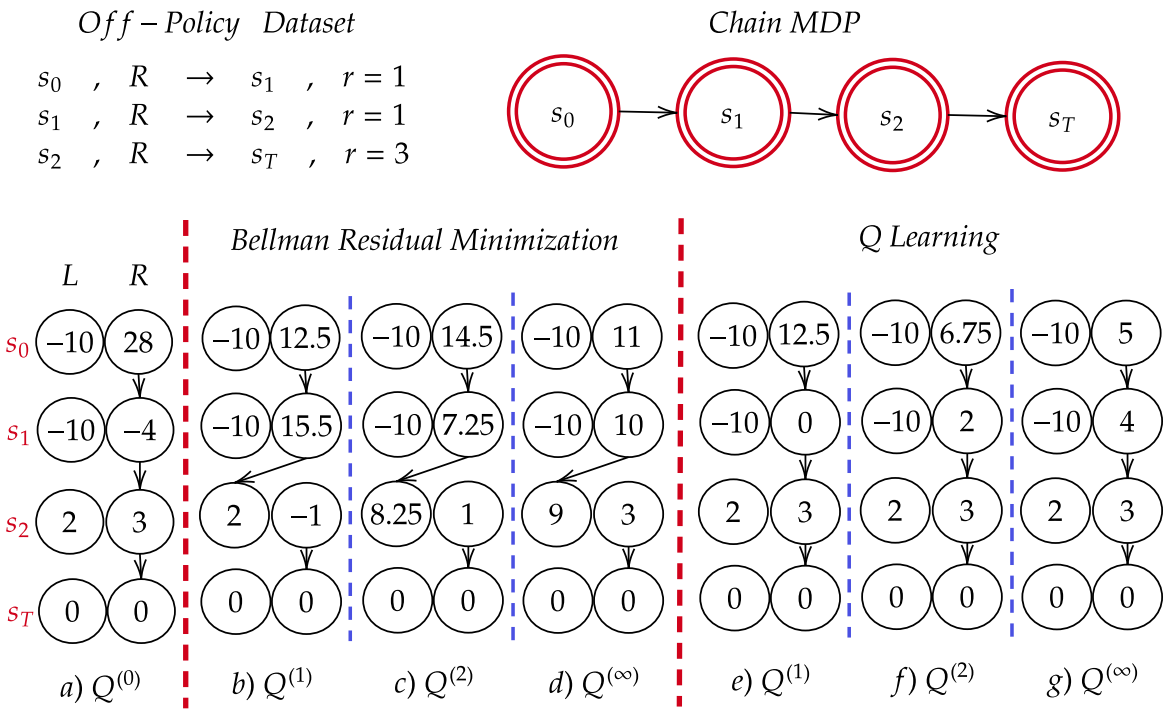


Figure 4.4: We show an example of chain MDP (with a tabular representation of value functions) where Q-learning and DBRM behave very differently under non-exploratory data. The top-right corner shows the MDP, which has two actions "L" and "R" with the same deterministic state transitions. They only differ in rewards and "L" gives very negative rewards and is suboptimal. The top left table shows the off-policy data-set which is non-exploratory ("L" never taken). Sub-figure (a) shows an initial Q-function, which will be updated using DBRM (b–d) and Q-learning (e–g) respectively on its right, with a learning rate of 0.25. The dashed arrows denote $s, a \to s', a'$ where $a' = \arg\max_{a''} Q(s', a'')$. In other words, a state-action pair without an arrow pointing to it has its influence blocked out in that iteration of learning. These arrows remain on the optimal policy using Q-learning. However, DBRM suffers from an early drift, and due to the lack of exploratory data it never recovers from it.

reflected as important states and actions appearing in the data less frequently than they should, a natural idea is to suppress it by minimizing the ℓ_∞ norm of Bellman error instead of ℓ_2 norm.[2]

Unfortunately, ℓ_∞ norm is too conservative and not amendable to optimization. Also, using the ℓ_∞ norm may result in the reduction of the effective sample size, which may therefore cause the algorithm to require much larger sample sets for producing optimal policies. As an

[2]In fact, ℓ_∞ norm is commonly used in the theoretical analyses of tabular RL to handle distribution mismatch.

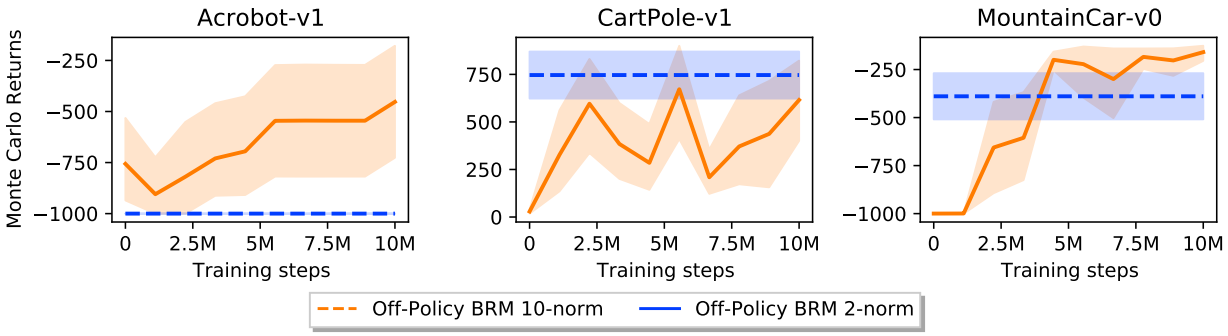


Figure 4.5: Training off-policy 10-norm objective for a large number of steps. The data collected here were the same 2.5 million samples used in the last experiment, but the optimization lasted for 10 million iterations. Horizontal lines represent the performance of the DBRM agent in Figure 4.3 at the end of training.

intermediate solution, we propose changing the objective of DBRM to

$$\frac{1}{|D|}\sum_{(s,a,r,s')\in D}\left|f(s,a)-r-\gamma\max_{a'\in\mathcal{A}}f(s',a')\right|^p, \tag{4.1}$$

for $p \geq 2$. DBRM's objective is the special case of $p = 2$. As p increases, optimization becomes more difficult and more samples may be needed.[3] We test this modification with $p = 6$ and $p = 10$ under the same setup as the off-policy experiments described earlier in this section, and indeed the progress of optimization is slower than $p = 2$ (see Figure B.1 in the appendix). However, given enough optimization steps, we see that the agent with $p = 10$ is able to learn non-trivial policies in Acrobot, where it completely fails previously with $p = 2$, and still maintain reasonable performance in the other two domains.

[3]From a theoretical point of view, loss of sample efficiency is due to the fact that the concentration behavior of $(\cdot)^p$ will get worse as p increases.

Chapter 5: RELATED WORK

The convergence of TD-learning and ADP is a complex analytical issue even when considering perfect state representations. To name a few, Dayan [13], Dayan and Sejnowski [14], Sutton [31], and Jaakkola et al. [19] provide some of the fundamental analytical results for the convergence of the TD-learning in the absence of function approximation. Section A.3 of Appendix A offers a more detailed account of these results. In short, there are strong theoretical guarantees about the convergence of the TD-learning method to the optimal value function on finite MDPs with reasonably practical assumptions.

On the other hand, the convergence of the TD-learning method under function approximation is a much more complicated matter. There are many theoretical and practical scenarios where TD-learning under function approximation is proven to fail or diverge (e.g., see Baird [3], Boyan and Moore [7], Gordon [18, 18], Tsitsiklis and Van Roy [33, 33], and Bertsekas [5]), whereas other analyses could prove its convergence under certain assumptions (e.g., see [18, 29], and Tsitsiklis and Van Roy [34]). A detailed explanation of these results and their distinctions is provided in Section A.4 of Appendix A. In short, (1) off-policy sampling of states and actions and (2) utilizing non-linear function approximation classes are two of the dominant factors causing the TD-learning method to provably diverge or fail to learn any optimal value predictors.

Tsitsiklis and Van Roy [34] proved the convergence of ADP under linear function approximation. Baird [2] showed divergence example for linear value approximation using ADP for off-policy training, or on-policy training with environment restart. Bradtke [8] proposed BRM with quadratic function approximation with convergence guarantees. Williams and Baird [37] proved tight bounds for relating the Bellman residuals to the policy performance. Baird [2], Sutton et al. [32] proved BRM methods converge to a local optimum using any function approximation. Dayan [13] reported some conditions where ADP diverges. Baird [2], Tsitsiklis and Van Roy [34] also reinforced this matter with either on-policy or off-policy examples of divergence. As for the speed of convergence, Schoknecht and Merke [28] used spectral analysis to prove the faster nature of ADP, but did not generalize to function approximation. To address the issue of slower convergence for residual gradients relative to

ADP, Baird [2] proposed a super set of algorithms called residual algorithms.

Scherrer [27] studied the stability versus faster convergence trade-off between BRM and ADP with linear function approximation, and concluded ADP methods yield marginally better performance, but their inherent numerical instability makes them overall less appealing than BRM. Recently, Zhang et al. [38] explored utilizing BRM as a replacement to ADP methods in the Deep Deterministic Policy Gradient (DDPG) algorithm, and found that BRM shows superior empirical results. We also refer the readers to the references in Zhang et al. [38] for a more comprehensive summary of previous understanding on ADP vs BRM. Despite that the relationship between ADP and BRM has been extensively studied in literature, our focus on deterministic environments where BRM can be run without state resetting is novel to the best of our knowledge.

Chapter 6: CONCLUSION

In this work we have examined the simple idea of running Bellman Residual Minimization algorithms on deterministic environments without state resetting or double sampling, which has not received its deserved attention in the literature. By comparing Q-learning to its DBRM counterpart empirically, we confirm a number of theoretical predictions, and also observe interesting behaviors of DBRM, for example, that it is sensitive to distribution mismatch.

While DBRM shows promising results on simple control benchmarks, and our Proposition 3.1 shows that it requires a weaker condition than deterministic dynamics, it is still unclear if such performance will scale to complex environments even if they are deterministic. In fact, the determinism of an environment in the function approximation setting is an ill-defined notion, as all RL environments can be viewed as deterministic (with random initial states) [24]. In other words, *misspecified function approximation and stochasticity are inherently indistinguishable*, and it does not make sense to talk about deterministic dynamics without reference to the function approximation in use. This hints at an interesting use of DBRM: We can use it to test the true stochasticity of an environment (just as in Figure 4.2), as the success of the algorithm indicates that the environment is, in a sense, deterministic under the particular choice of function approximation.

Appendix A: LEARNING VALUES BY TEMPORAL DIFFERENCES

The method of Dynamic Programming was formally defined by Bellman and Dreyfus [4], which could be applied to the problem of learning values in a finite MDP with known transition dynamics. However, such assumptions are too strong for most reinforcement learning applications. This motivated the work on solving the temporal credit assignment problem [30].

Before jumping to introduce the Temporal Difference (TD) algorithms, we would like to take a step back and present the *delta update rule*. Earlier than Sutton [30], a lot of research was conducted on developing adaptive filters, which had many applications in the areas of pattern recognition, signal processing, and adaptive control. One of such adaptive learning strategies was the Least Mean-Squared (LMS) filter, which utilized the delta update rule and will be briefly presented next.

A.1 THE LEAST MEAN SQUARED FILTER AND THE DELTA UPDATE RULE

The adaptive filtering problem was to predict the next observation given a fixed-length history of observations as shown in Figure A.1. One approach to solve such a problem was to construct a data-set of observations for supervised learning, and solve the task using empirical risk minimization techniques for regression. However, this does not comply with the adaptive nature of the problem, where learning parameters and collecting samples must be performed simultaneously. Instead, the LMS filter [36] allows for making predictions and utilizing them while running the trajectory as described in Algorithm A.1.

The Stochastic Approximation (SA) used within the LMS filter can be interpreted as objectively minimizing the least mean squared error $\delta^2 := \mathbb{E}_{t\sim U}[\delta_t^2]$ where U is the nearly uniform distribution over the entire trajectory's time steps. Trying to directly compute the gradients for δ^2 requires obtaining δ_t^2 values in the first place, which is a luxury that cannot be afforded in online learning. Instead, the LMS filter uses SA and treats δ_t^2 as an unbiased estimate for δ^2 at each time step t. The particular update rule used within the LMS filter $\theta_i \leftarrow \theta_i - \eta_t \cdot \delta_t \cdot \frac{\partial \hat{v}_t}{\partial \theta_i}$ is otherwise known as the *delta update rule*, and would turn out to inspire

Algorithm A.1: The Least Mean Squared (LMS) Adaptive Filter

Require: The initial state s_1 and the policy π.
Require: The prediction target estimator v_{Target}.
Require: The learning rate parameter η_t.
Require: The adaptive filter length k.
Require: The initial k prediction parameters $\theta_1, \theta_2, \cdots, \theta_k$.
1: Run the policy π for k initial steps to obtain $(s_1, s_2, \cdots, s_k)$.
2: **for** $t = 1, 2, \ldots$ **do**
3: Set $T \leftarrow t + k - 1$.
4: Sample the current action from the policy π given the current state s_T: $a_T \sim \pi(s_T)$.
5: Take a step in the transition dynamics to generate the next state: $s_{T+1} \sim P(s_T, a_T)$.
6: Construct an estimate of the prediction target v^t_{Target} as a function of the most recent partial trajectory $s_t, s_{t+1}, \cdots, s_T$.
7: Construct a linear prediction of target using the current parameters and the recent history of observations $\hat{v}_t = \sum_{i=t}^{T} s_i \theta_{i-t+1}$.
8: Find the delta $\delta_t = v^t_{\text{Target}} - \hat{v}_t$.
9: Make a delta update for each θ_i:

$$\begin{aligned}\theta_i \leftarrow & \theta_i - \eta_t \cdot \frac{\partial}{\partial \theta_i}\left[\frac{1}{2}\delta_t^2\right] \\ = & \theta_i - \eta_t \cdot \delta_t \cdot \frac{\partial \hat{v}_t}{\partial \theta_i}.\end{aligned}$$

10: **end for**
11: **return** The adaptive filter parameters $\theta = (\theta_1, \theta_2, \cdots, \theta_k)$.

the TD-learning algorithm.

A.2 THE TEMPORAL-DIFFERENCE LEARNING ALGORITHM TD(λ)

The Temporal Differences (TD) learning's task is to predict the values for each state in a MDP, and is formally defined in Algorithm A.2. In short, the procedure is quite similar to the LMS filter with a few subtle distinctions:

- First, the target value is not only a function of the observations, actions, rewards, and the transition dynamics, but also uses the current set of parameters θ to construct an estimate of the target value.
- Second, the predicted value function is assumed to only be a function of the current state $\hat{v}_t := V(s_t; \theta)$.

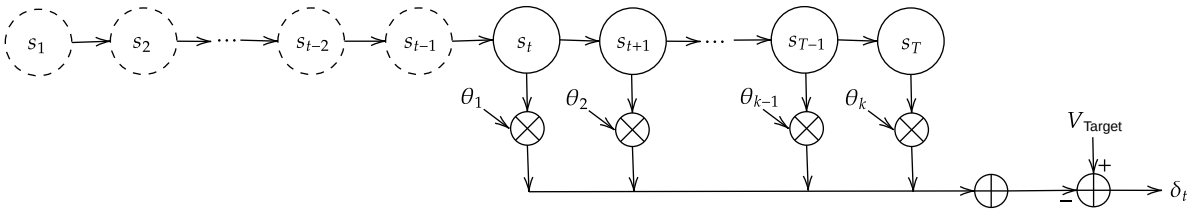


Figure A.1: The Least Mean-Squared (LMS) prediction filter. The filter length is supposed to be k, and $\theta_1, \theta_2, \cdots, \theta_k$ are the filter's adaptive (i.e., learned) parameters. the target for prediction is denoted by v_{Target}, which was originally intended to be the next state s_{t+1}. However, this target can be substitute by the Q-values for each state, as would be done later in the TD(1) algorithm.

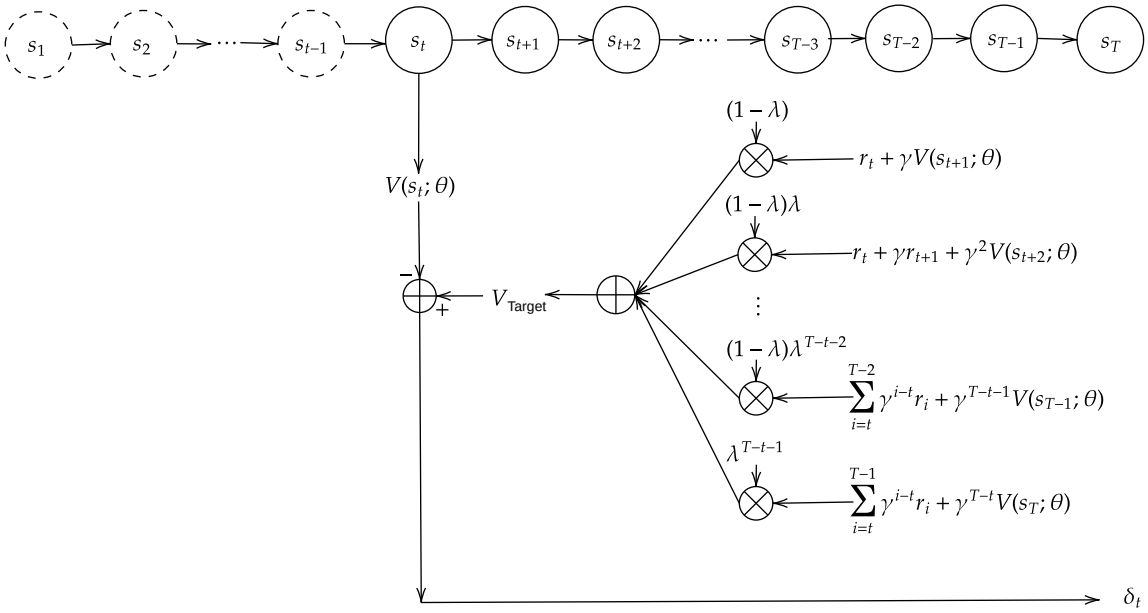
- Finally, the λ parameter provides a control for the importance of future-looking target predictors.

Each of these distinctions cause algorithmic consequences that will be discussed next.

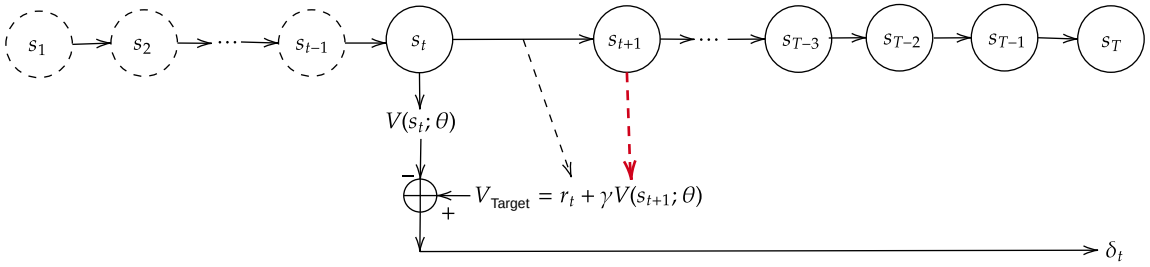
The TD-learning algorithm uses the current parameters to estimate the next state's value, as if the current parameters provide "ground truth" estimates independently. This assumption is obviously incorrect, especially at the early stages of the learning process. However, this practice can form a powerful learning approach, and is commonly referred to as *bootstrapping* the current estimator for the purposes of gradient construction. This causes certain benefits to the optimization process of the algorithm, which were discussed in Section 4.3. This practice is key in studying the convergence of the algorithm and its sample-efficiency bounds.

It is worth noting that the relationship $\theta_{\text{new}} - \theta_{\text{old}} \propto \nabla_\theta \delta_t^2$ is held for the LMS adaptive filter, as the v_{Target}^t values are independent of the parameters θ in the adaptive filter's problem formulation. However, v_{Target}^t is defined as a function of θ in the TD-learning algorithm, which makes $\theta_{\text{new}} - \theta_{\text{old}} \not\propto \nabla_\theta \delta_t^2$ in this case. This is, in an essence, where the Bellman Residual Minimization (BRM) algorithms are different from the TD-Learning methods; instead of computing the TD-learning semi-gradient, BRM algorithms try to exactly construct the $\nabla_\theta \delta_t^2$ gradient and use it for updating the parameters.

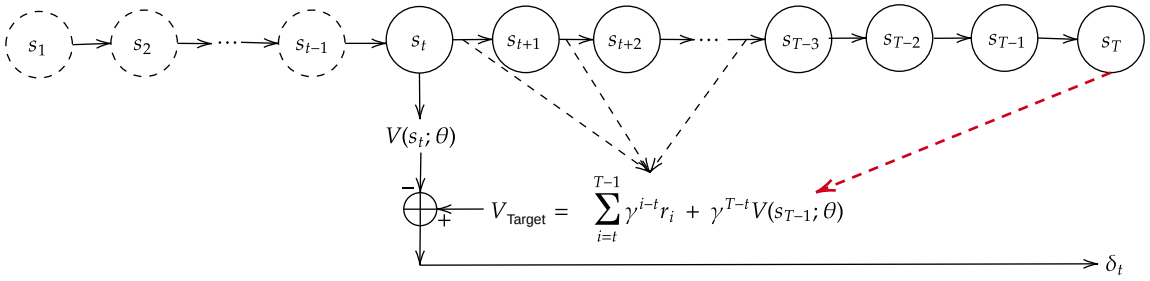
To understand the role of the λ parameter, it is useful to simplify the resulting algorithm for both $\lambda = 0$ and $\lambda = 1$ extremes. Figure A.2 shows the flow in which the target value is computed given an arbitrary λ. It also visualizes the simplified algorithm in the cases of $\lambda = 0$ and $\lambda = 1$. When $\lambda = 0$, the target value is built only as a function of the next state



(a) The TD(λ) algorithm



(b) The TD(0) algorithm



(c) The TD(1) algorithm

Figure A.2: (a) The construction of the delta for the TD(λ) algorithm with $0 < \lambda < 1$. (b) The simplification of the TD(λ) flow for the limit $\lambda = 0$ case. The dashed red arrow and the dashed black arrow indicates that only s_{t+1} and r_t are necessary for computing the target value V_{Target}, respectively. (c) The simplification of the TD(λ) flow for the limit $\lambda = 1$ case. As can be seen, the much later observation s_T and all the rewards in between (i.e., $r_t, r_{t+1}, \cdots, r_{T-1}$) are required for computing V_{Target}.

Algorithm A.2: The Temporal Difference-Learning TD(λ) Algorithm

Require: The initial state s_1, the policy π, and the discount factor γ.
Require: The λ parameter.
Require: The learning rate parameter η_t.
Require: The history length k (where we should ideally have $k \simeq O((1-\lambda)^{-1})$).
Require: The initial prediction parameters θ.

Run the policy π for k initial steps to obtain $(s_1, r_1, s_2, r_2 \cdots, s_{k-1}, r_{k-1}, s_k)$.
for $t = 1, 2, \ldots$ **do**
  Set $T \leftarrow t + k - 1$.
  Sample the current action from the policy π given s_T: $a_T \sim \pi(s_T)$.
  Take a step in the transition dynamics to get the next state: $s_{T+1} \sim P(s_T, a_T)$.
  Construct an estimate of the bootstrapped prediction target:

$$v^t_{\text{Target}} := (1-\lambda) \sum_{j=t}^{T-2} \lambda^{j-t+1} (\sum_{i=1}^{j} \gamma^{i-t} r_i + \gamma^{j-t+1} V(s_{j+1}; \theta)).$$

  Construct a value prediction of s_t using the current parameters $\hat{v}_t := V(s_t; \theta)$.
  Find the delta $\delta_t = v^t_{\text{Target}} - \hat{v}_t$.
  Make a delta update for θ:

$$\theta \leftarrow \theta - \eta_t \cdot \delta_t \cdot \nabla_\theta \hat{v}_t.$$

end for
return The value prediction parameters θ.

s_{t+1}. This limit conserves the online-learning characteristics of the TD-learning algorithm, and stresses the bootstrapping approximations the most. On the other hand, when $\lambda = 1$ and T is set to be much larger than t (i.e., $k \to \infty$ in Algorithm A.2), the target value would mostly consist of a Monte-Carlo estimate of the payoff, and the bootstrapping approximations vanish.

A.3 CONVERGENCE OF TD(λ) WITH PERFECT REPRESENTATION

The method of learning through temporal differences was formally introduced in Sutton [31]. This work proposed two theoretical results about the introduced method. First, the TD-learning Algorithm A.2 degenerates to the LMS filter, which was given in Algorithm A.1, in the special $\lambda = 1$ case. The intuition for this proposition can be deduced by visually

comparing the LMS filter of Figure A.1, and the TD(1) simplification shown in Figure A.2; if k is asymptotically large and linear function approximation is used, TD(1) becomes a special case of the LMS adaptive filter. This tied the new method of TD-learning to an abundance of existing literature and convergence guarantees derived for adaptive filters and helped it gain some theoretical footing [31, 36]. On the other hand, the TD(0) extreme was less studied, and the idea of being able to learn target values through mere single step differences was novel at the time.

Initially, Sutton [31] proved that for a finite MDP with an underlying absorbing Markov chain and no inaccessible states, the TD(0) algorithm's predictions converge in expectation to the ideal values if linear value predictors and a small enough learning rate η were used. Although this facilitated a proper introduction of the TD-learning method, this particular guarantee was rather weak since it only provided insurance about convergence *of* the mean rather than convergence *in* the mean; having

$$\lim_{t\to\infty} |\mathbb{E}[\theta_t] - \theta^*| = 0 \tag{A.1}$$

is much weaker than having

$$\lim_{t\to\infty} \mathbb{E}[|\theta_t - \theta^*|] = 0. \tag{A.2}$$

Also, the derivation assumed tabular settings, a specific λ value, and having a fixed learning rate η. Most importantly, only linear function approximation was considered in this result.

Dayan [13] proved a similar result for the general case of $\lambda \in (0, 1)$. Specifically, it proved convergence *of* the mean for general λ values. Later, Dayan and Sejnowski [14] provided the stronger theoretical guarantee of convergence *in* the mean for $\lambda \in (0, 1)$ with probability 1. All such efforts were pursued within the domain of dynamic programming, whereas the TD-learning algorithm can also be viewed as an instance of Stochastic Approximation (SA) methods through the realm of optimization. In fact, Jaakkola et al. [19] used the latter approach, and provided similar theoretical guarantees from a more general SA perspective for both TD and Q-learning methods while assuming typical SA requirements for convergence (e.g., having the learning rate sequence η_t converge neither *too fast* nor *too slow* to zero).

A.4 CONVERGENCE OF TD(λ) WITH FUNCTION APPROXIMATION

None of the results described so far touched on the issue of function approximation in detail, and assumed tabular settings and perfect state representations, where the number of trainable parameters equals the cardinality of the state space $|\mathcal{S}|$. This is obviously impractical with continuous or exponentially large state spaces, and calls upon function approximation to be involved. Adding function approximation to the mix introduces a few issues:

1. In the presence of large state spaces, computing the Bellman operator $\mathcal{T}(f)$ is practically impossible as it would require intractable amounts of samples.

2. The Bellman operator's projection to the function approximation class may introduce substantial errors to the process.

Furthermore, even in the absence of these complications, using function approximation and tying different states predicted values by shared parameters could cause divergence itself. For instance, consider the following example:

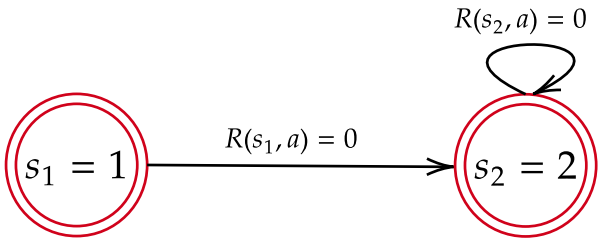


Figure A.3: The MDP used in Example A.1 to illustrate the possibility of divergence in the presence of function approximation even with finite MDPs. This example assumes access to all transition samples and uses the least squares update rule for value iteration.

Example A.1. Least-squares value iteration's counter example of Tsitsiklis and Van Roy [33] Consider the MDP shown in Figure A.3 with the state space $\mathcal{S} = \{1, 2\}$, the action space $\mathcal{A} = \{0\}$, the reward function $R(s, a) = 0$, and the value function class $\mathcal{F} = \{f_\theta | f_\theta(s) = \theta s, \theta \in \mathbb{R}\}$. Using the least squares update rule

$$\theta_{k+1} \leftarrow \arg\min_{\theta} \sum_{s \in \mathcal{S}} (f_\theta(s) - \mathcal{T}(f_{\theta_k})(s))^2 \tag{A.3}$$

with $\gamma > 5/6$ will lead to divergence.

It is worth noting that all states in Example A.1 have a value of zero, leading the optimal value function to be realizable (i.e., $V^* \in \mathcal{F}$). This goes to show the complications induced by incorporating function approximation in value learning even under ideal conditions.

To tackle the first issue, learning compact features for states were pursued as a solution, so that TD-learning would be tractable in the compact feature space. In other words, these efforts tried to outsource the problem of large state spaces to representation learning and properly aggregating states together. Under this state aggregation strategy, Tsitsiklis and Van Roy [33] upper bounded the resulting TD-learned feature-based predictor's error linearly by the maximal state-aggregation error (and similarly for the corresponding greedy policy). Singh et al. [29] proposed the *soft state aggregation* model, where each state was *softly* represented by a probabilistic mixture model. Under such a fixed representation, Singh et al. [29] showed that TD-learning would provably converge to a solution where the Bellman equations would be held in the cluster space. Since even the best solution for the Bellman equations in the cluster space may be sub-optimal if such a representation was weak, the paper also proposed adaptively tuning the representations while applying TD-learning, but no proof of convergence was given for such an approximation strategy.

As for the second issue, Gordon [18] tried to incorporate function approximation into a common contraction-map analysis by treating the projection to function class step as an operator attached to the Bellman operator. Showing that the Bellman operator was a contraction map was a typical way of analyzing value and policy iteration methods without function approximation. Gordon [18] showed that if the projection step was a non-expansion w.r.t. the l_∞ norm, then applying TD-learning would converge to a unique solution. However, that unique solution may not be optimal unless the optimal value function was closed to a fixed point solution of the projection operator. Although such an analysis guaranteed convergence for a broad class of approximators such as the k-nearest neighbor class, it could not be applied to many popular function classes, such as linear regression models and neural networks, as their induced projection operators could be expansion maps.

Since the introduction of the TD-learning algorithm, many examples were published where the TD-learning method was shown to fail under function approximation [3, 5, 7, 18, 33]. However, one of the common threads amongst all of them is the independence of the sampling process from the underlying MDP's transition model. Tsitsiklis and Van Roy [34] applying linear approximators for TD-learning on a single endless trajectory of an irreducible aperiodic Markov chain, would have guaranteed convergence with a probability of one. This is one of the strongest theoretical results for TD-learning's convergence, and it shows that convergence for linear approximators is guaranteed when the states are sampled according to the MDP's steady-state distribution. However, it also shows an example where even though the states are sampled from the steady-state distribution, the presence of non-linear function approximation for TD-learning causes the learned parameters to diverge.

Appendix B: EXPERIMENT DETAILS AND ADDITIONAL RESULTS

B.1 FURTHER DETAILS ON THE EXPERIMENTS WITH HIGH-NORMS

For the off-policy version where we used a higher p-norm, since we need exploratory data, we do not use the replay buffer created when training the on-policy agent. Instead, we take the final policy learned by the on-policy agent and sample trajectories with a 0.6 chance of exploration, ans use this data to train the off-policy agents with higher-norm objectives.

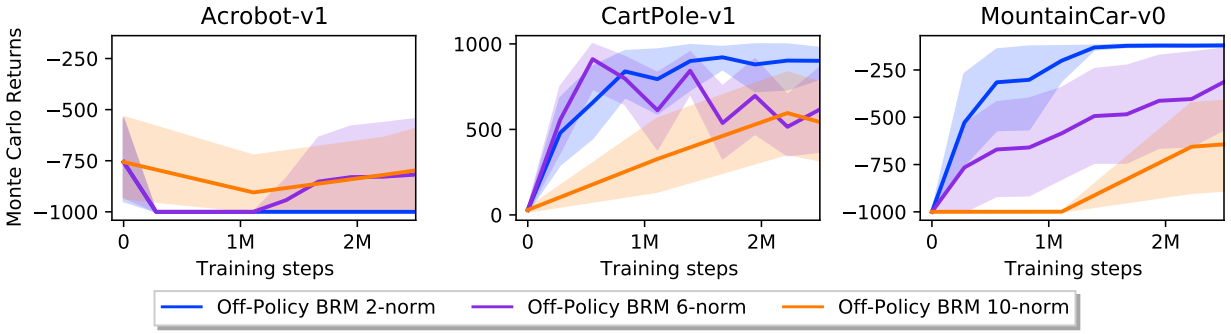


Figure B.1: The effect of higher p-norms on the DBRM off-policy training process. The exploration buffer was used in this experiment instead of the replay buffer. The training here used 2.5 million steps. The 10-norm was chosen for further training, since it seemed to be performing better on the problematic acrobot environment. The top row shows the Monte-Carlo evaluation during 10 points of training. The bottom row shows the training loss used in each optimization method.

B.2 FURTHER DETAILS ON THE RESIDUAL DRIFT ISSUE IN PRACTICE

The residual drifting issue in the Acrobot domain is better illustrated in Figure B.2. The experiments were conducted with 20 restarts, and 10 different independent off-policy and on-policy trails. The trajectories were cut off after 200 time steps. The fitted on-policy and off-policy Q functions are denoted as $Q_{\texttt{on}}$ and $Q_{\texttt{off}}$, respectively. Their corresponding greedy policies are denoted as $\pi_{\texttt{on}}$ and $\pi_{\texttt{off}}$, respectively. The residuals shown in each subplot are defined as below

$$\delta_{(a)}(t) = Q_{\texttt{off}}(s_t, \pi_{\texttt{on}}(s_t)) - R(s_t, \pi_{\texttt{on}}(s_t)) - \gamma \max_a Q_{\texttt{off}}(s_{t+1}, a) \tag{B.1}$$

$$\delta_{(b)}(t) = Q_{\texttt{on}}(s_t, \pi_{\texttt{on}}(s_t)) - R(s_t, \pi_{\texttt{on}}(s_t)) - \gamma \max_a Q_{\texttt{on}}(s_{t+1}, a) \tag{B.2}$$

$$\delta_{(c)}(t) = Q_{\texttt{off}}(s_t, \pi_{\texttt{on}}(s_t)) - R(s_t, \pi_{\texttt{on}}(s_t)) - \gamma \quad Q_{\texttt{off}}(s_{t+1}, \pi_{\texttt{on}}(s_{t+1})) \tag{B.3}$$

$$\delta_{(d)}(t) = Q_{\texttt{off}}(s_t, \pi_{\texttt{off}}(s_t)) - R(s_t, \pi_{\texttt{off}}(s_t)) - \gamma \max_a Q_{\texttt{off}}(s_{t+1}, a) \tag{B.4}$$

Figures B.2 (a) and (b) show that the off-policy optimization is successful, and the objective is much smaller for the off-policy agent. However, comparing (a) and (c) suggests that the off-policy agent is utilizing the residual drift phenomenon to achieve this level of small residuals.

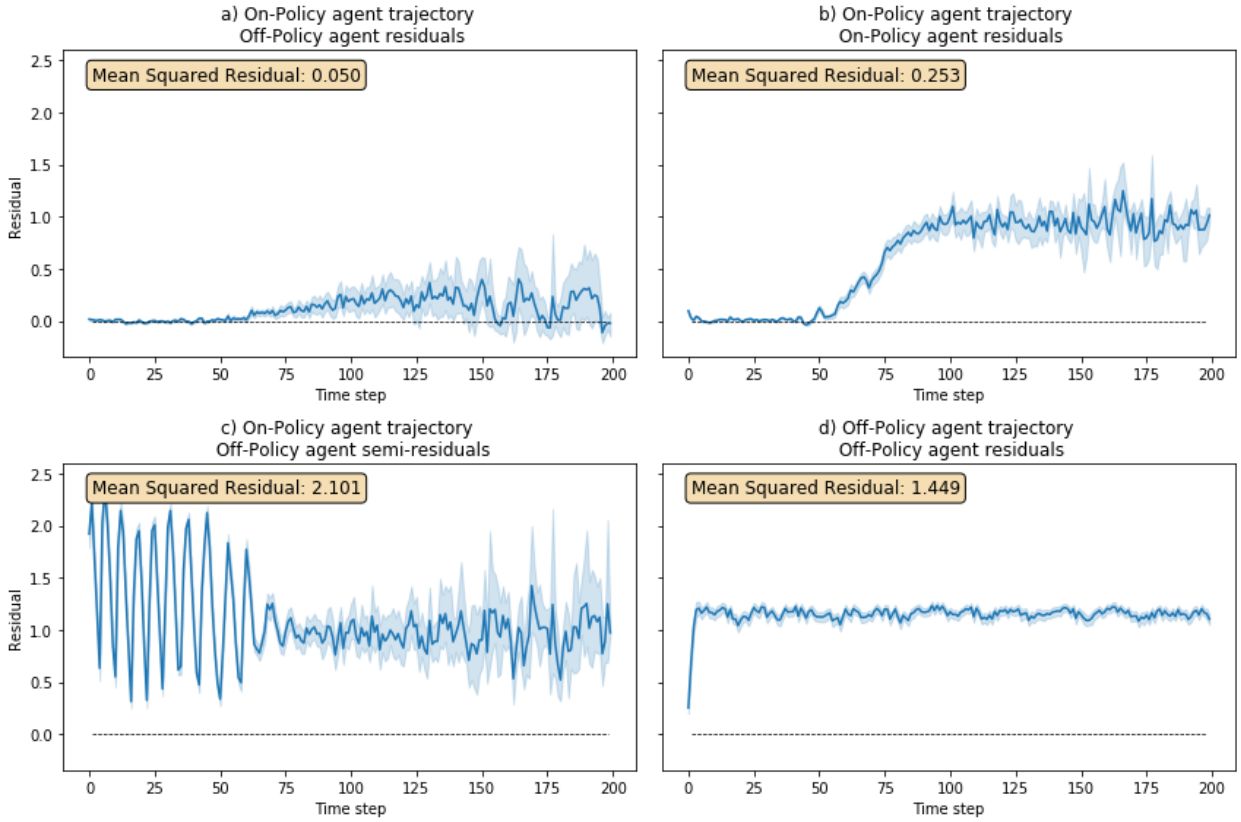


Figure B.2: Residual drift suggestions in the Acrobot domain. The horizontal axis represents time steps during the simulated trajectories, and not training progress (i.e. other words, the plots were created with fully trained off-policy and on-policy agents). See above for clarification on each type of residuals plotted.

REFERENCES

[1] András Antos, Csaba Szepesvári, and Rémi Munos. Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129, 2008.

[2] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

[3] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

[4] Richard E Bellman and Stuart E Dreyfus. *Applied dynamic programming*, volume 2050. Princeton university press, 2015.

[5] Dimitri P Bertsekas. A counterexample to temporal differences learning. *Neural computation*, 7(2):270–279, 1995.

[6] Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[7] Justin Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing systems*, pages 369–376, 1995.

[8] Steven J Bradtke. Reinforcement learning applied to linear quadratic regulation. In *Advances in neural information processing systems*, pages 295–302, 1993.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[10] Jinglin Chen and Nan Jiang. Information-theoretic considerations in batch reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 1042–1051, 2019.

[11] Bo Dai, Albert Shaw, Lihong Li, Lin Xiao, Niao He, Zhen Liu, Jianshu Chen, and Le Song. Sbeed: Convergent reinforcement learning with nonlinear function approximation. In *International Conference on Machine Learning*, pages 1133–1142, 2018.

[12] Christoph Dann, Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, John Langford, and Robert E Schapire. On Oracle-Efficient PAC RL with Rich Observations. In *Advances in Neural Information Processing Systems*, pages 1429–1439, 2018.

[13] Peter Dayan. The convergence of td (λ) for general λ. *Machine learning*, 8(3-4):341–362, 1992.

[14] Peter Dayan and Terrence J Sejnowski. Td (λ) converges with probability 1. *Machine Learning*, 14(3):295–301, 1994.

[15] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

[16] Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. Diagnosing bottlenecks in deep q-learning algorithms. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2021–2030, 2019.

[17] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2052–2062, 2019.

[18] Geoffrey J Gordon. Stable function approximation in dynamic programming. In *Proceedings of the twelfth international conference on machine learning*, pages 261–268, 1995.

[19] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, 1994. doi: 10.1162/neco.1994.6.6.1185.

[20] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for MDPs. In *Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics*, pages 531–539, 2006.

[21] Odalric-Ambrym Maillard, Rémi Munos, Alessandro Lazaric, and Mohammad Ghavamzadeh. Finite-sample analysis of bellman residual minimization. In *Proceedings of 2nd Asian Conference on Machine Learning*, pages 299–314, 2010.

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[23] Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(May):815–857, 2008.

[24] Andrew Y Ng and Michael Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 406–415. Morgan Kaufmann Publishers Inc., 2000.

[25] ML Puterman. Markov Decision Processes. *Jhon Wiley & Sons, New Jersey*, 1994.

[26] Ehsan Saleh and Nan Jiang. Deterministic bellman residual minimization. 2019.

[27] Bruno Scherrer. Should one compute the temporal difference fix point or minimize the bellman residual? the unified oblique projection view. *arXiv preprint arXiv:1011.4362*, 2010.

[28] Ralf Schoknecht and Artur Merke. Td (0) converges provably faster than the residual gradient algorithm. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 680–687, 2003.

[29] Satinder P Singh, Tommi Jaakkola, and Jordan Michael I. Reinforcement learning with soft state aggregation. *Advances in neural information processing systems 7*, 7:361, 1995.

[30] Richard S Sutton. Temporal credit assignment in reinforcement learning. 1985.

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

[32] Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 993–1000. ACM, 2009.

[33] John N Tsitsiklis and Benjamin Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1):59–94, 1996.

[34] John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, 42(5), 1997.

[35] Benjamin Van Roy. *Feature-based methods for large scale dynamic programming*. PhD thesis, Massachusetts Institute of Technology, 1994.

[36] Bernard Widrow and Samuel D Stearns. Adaptive signal processing prentice-hall. *Englewood Cliffs, NJ*, 1985.

[37] Ronald J Williams and Leemon C. Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical report, Citeseer, 1993.

[38] Shangtong Zhang, Wendelin Boehmer, and Shimon Whiteson. Deep residual reinforcement learning, 2019.

Appendix C: LIST OF ABBREVIATIONS AND NOMENCLATURE

C.1 LIST OF ABBREVIATIONS

ADP	Approximate Dynamic Programming
BRM	Bellman Residual Minimization
DBRM	Deterministic Bellman Residual Minimization
DDPG	Deep Deterministic Policy Gradient
DP	Dynamic Programming
FQI	Fitted Q-Iteration
LMS	Least Mean Squares
MDP	Markov Decision Processes
MRI	Major Research Instrumentation
RL	Reinforcement Learning
SA	Stochastic Approximation
TD	Temporal Differences

C.2 LIST OF NOMENCLATURE

Markov Decision Processes Elements

$\mathcal{A}$	Action Space
$\mathcal{S}$	State Space
γ	Discount Factor
R	Reward Function
$R_{\max}$	Maximal Reward Value
P	Probabilistic Transition Function
π	Policy
τ	Trajectory
D	Dataset of Transition Tuples

Value Functions and Operators

V^{π}	Value Function of Policy π
Q^{π}	Q-Value Function of Policy π
$\mathcal{T}$	Bellman Operator

Mathematical Notations and Operators

Δ	Credal Set Operator
l_p	Minkowski's p Norm

Function Approximation Elements

$\mathcal{F}$	Function Approximation Family
f	Value Function Approximator Instance
θ	Approximation Parameters to be Learned
η_t	Learning Rate Sequence
λ	The Temporal Difference Learning Parameter