Deterministic Bellman Residual Minimization

Ehsan Saleh
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
ehsans2@illinois.edu

Nan Jiang
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
nanjiang@illinois.edu

Abstract

Bellman Residual Minimization (BRM) algorithms are alternatives to Approximate Dynamic Programming for learning value functions in reinforcement learning. 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.

In this paper, we make a simple observation that BRM can be applied without state resetting if the environment is deterministic, which Baird [1995] has hinted in his original paper. The resulting algorithm is simple, convergent, and works well in benchmark control problems. 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.

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 [Bertsekas and Tsitsiklis, 1996]; e.g., Q-learning and TD), which learns from bootstrapped targets, and Bellman residual minimization (BRM; e.g., residual gradient [Baird, 1995]), 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 simple and convergent *Deterministic Bellman Residual Minimization* (DBRM) algorithm which should work well in (nearly) deterministic environments. Given that many empirical RL benchmarks are deterministic or only mildly stochastic [Brockman et al., 2016], it is surprising that such a method has eluded the literature (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] There are BRM algorithms that do not require double sampling, such as modified BRM [Antos et al., 2008] and SBEED [Dai et al., 2018], 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.

Optimization Foundations for Reinforcement Learning Workshop at NeurIPS 2019, Vancouver, Canada.

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 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. 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 7).
4. We also compare Q-learning and DBRM on off-policy data (Section 5). 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 5).

2 Background

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. A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ specifies a distribution over actions for each state, and induces random trajectories from a given starting state s as follows: $s_1 = s$, $a_1 \sim \pi(s_1)$, $r_1 = R(s_1, a_1)$, $s_2 \sim P(s_2, a_2)$, $a_2 \sim \pi(s_2)$, ... Value function of π is defined as $V^\pi(s) := \mathbb{E}[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid s_1 = s; \pi]$, and we define $Q^\pi(s, a)$ by conditioning on $s_1 = s$ and $a_1 = a$. In an infinite-horizon discounted MDP, there always exists an optimal policy $\pi^\star$ that maximizes V^π and Q^π, which we denote by $V^\star := V^{\pi^\star}$, $Q^\star := Q^{\pi^\star}$. 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

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

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 [Puterman, 1994].

Function Approximation and Q-Learning Let f_θ denote a parameterized Q-value function, and the goal 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 function class. We say $\mathcal{F}$ is *realizable* if $Q^\star \in \mathcal{F}$.

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

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

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 [Ernst et al., 2005],

$$f_k \leftarrow \arg\min_{f \in \mathcal{F}} \mathcal{L}_D(f; f_{k-1}), \tag{2}$$

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 $\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}_D(f_\theta; f_\theta^-)$. where f_θ^- is the gradient disconnected version of f_θ and η is the learning rate. Notationally, a gradient term $\nabla_\theta \mathcal{L}(\cdot)$, can be replaced by its stochastic estimate.

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

3 Deterministic Bellman Residual Minimization (DBRM)

3.1 The Algorithm

The DBRM version of Q-learning is simply

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}_D(f_\theta; f_\theta). \tag{3}$$

Comparing to Eq.(1), 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

$$\arg\min_{f \in \mathcal{F}} \mathcal{L}_D(f; f). \tag{4}$$

Most ADP algorithms (including FQI and Q-learning) can diverge under function approximation [Van Roy, 1994, Gordon, 1995, Tsitsiklis and Van Roy, 1997], as they are fundamentally iterative algorithms (Eq.(2)) and lack a globally consistent objective. In contrast, DBRM is obviously convergent as it simply minimizes the objective in Eq.(4). 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.(4) 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

$$\mathcal{L}(f; f) = \mathbb{E}_{(s,a,r,s')}[(f(s,a) - (\mathcal{T}f)(s,a))^2] + \gamma \mathbb{E}_{(s,a)}[\mathbb{V}_{s' \sim P(s,a)}[\max_{a' \in \mathcal{A}} f(s', a')]], \tag{5}$$

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. Prior works proposed either approximating the second term [Munos and Szepesvári, 2008, Antos et al., 2008, Dai et al., 2018], or avoiding it with double sampling [Baird, 1995, Maillard et al., 2010]. None of these are necessary in deterministic environments where $P(s, a)$ is a point mass and $\mathbb{V}_{s' \sim P(s,a)}[\cdot] = 0$.

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 in the following result:

Proposition 1. *Let* $\epsilon_{dtmn} := \gamma \mathbb{E}_{(s,a)}[\mathbb{V}_{s' \sim P(s,a)}[V^\star(s')]]$. *Fixing any* $\hat{f} \in \mathcal{F}$, *define* $\epsilon_{estm\ \&\ opt} := \mathcal{L}(\hat{f}; \hat{f}) - \min_{f \in \mathcal{F}} \mathcal{L}(f; f)$. *Then if* $Q^\star \in \mathcal{F}$, *we have*

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

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

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

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 [Antos et al., 2008, Chen and Jiang, 2019].

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 [Dann et al., 2018]. 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 [Munos and Szepesvári, 2008]. 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 [Chen and Jiang, 2019]. 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?

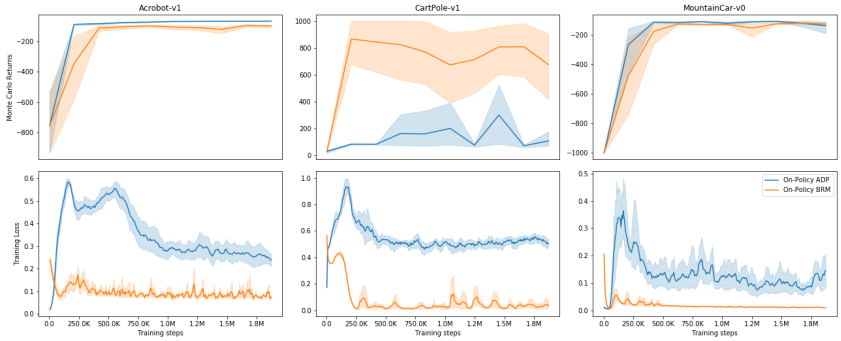

Figure 1: Comparison between Q-learning and DBRM (on-policy). The top row shows the Monte-Carlo evaluation results, and the bottom row shows the training loss. All error bars in this paper show a 95% confidence interval for the mean using 1000 bootstrap samples.

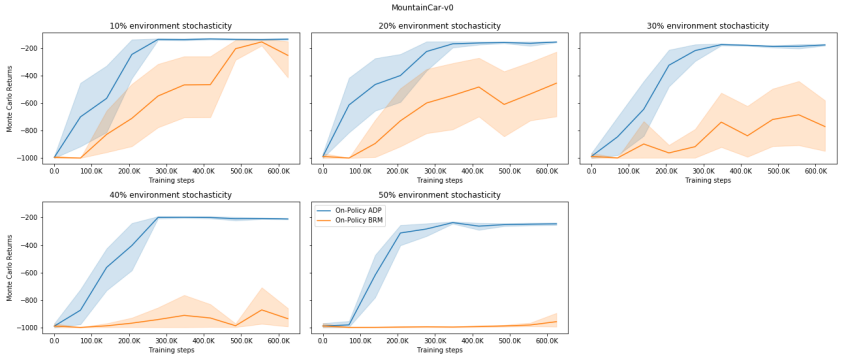

Figure 2: Injecting stochasticity levels to the Mountain-Car environment. The performance degradation of on-policy BRM is vs the clear lead of on-policy TD is observable. An stochasticity level of 40%, ensures 60% of actions be exactly as commanded by the agent and the rest are random.

4 DBRM: Does It Work?

4.1 On-Policy Experiments in Deterministic Environments

Mountain-Car, Cart-Pole, and Acrobot from the OpenAI Gym collection [Brockman et al., 2016] were used as experimental. A 3-layer MLP was used for function approximation with two hidden 64-unit layers and Tanh activation. Further details about the exploration scheme, algorithmic details, and evaluation are in the supplementary material due to space limitation, and we highly encourage the reader to go over them.

Results We plot the MC returns and the training losses averaged over 10 runs in Figure 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. DBRM finds value-functions with low squared losses $\mathcal{L}_D(f; f)$ and Q-learning having higher losses, which is theoretically expected (Section 3.2). However, 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. For this, we convert the environments used in Section 4 into stochastic environments by manipulating the deterministic environments to take a uniformly random action

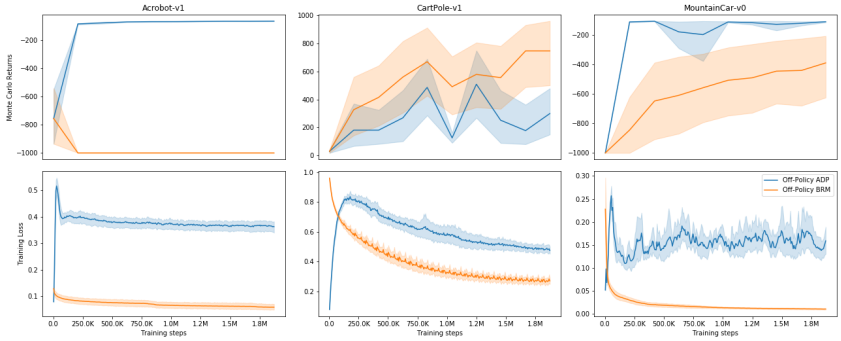

Figure 3: The off-policy ADF and BRM methods performance comparison. The top row shows the Monte-Carlo evaluation during 10 points of training. The bottom row shows the training loss used in each optimization method.

with a certain probability regardless of the action commanded by the agent. Note that this is different from the random actions in ϵ-greedy exploration.

Results We show the results for MountainCar in Figure 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.

5 On the Sensitivity of DBRM on 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 require the two algorithms to collect their own datasets. Since we are mostly interested in the capability of Q-learning and DBRM in solving the temporal credit assignment problem, it is important that we compare their performance in a completely off-policy manner. Experimental description is left to the supplementary material due to lack of space.

Results The results are shown in Figure 3. The curves for Q-learning look similar to their on-policy counterparts in Figure 1, showing that Q-learning is not sensitive to off-policy data and is consistent with that reported by Fu et al. [2019]. 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 DBRM's success 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. Q-learning is reported to be less sensitive to off-policyness of data Fu et al. [2019]. Figure 4, shows a severe distribution mismatch example due to non-exploratory data where Q-learning can still give sensible solutions [Fujimoto et al., 2019].

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 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.[3] Unfortunately, ℓ_∞ norm is too conservative and not amendable to optimization, and may result in reduction of effective sample size. As an intermediate solution, we propose changing the objective of DBRM to $\frac{1}{|D|}\sum |f(s,a) - r - \gamma \max_{a' \in \mathcal{A}} f(s',a')|^p$, 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.[4] Although higher norms require more 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] In fact, ℓ_∞ norm is commonly used in the theoretical analyses of tabular RL to handle distribution mismatch.

[4] 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.

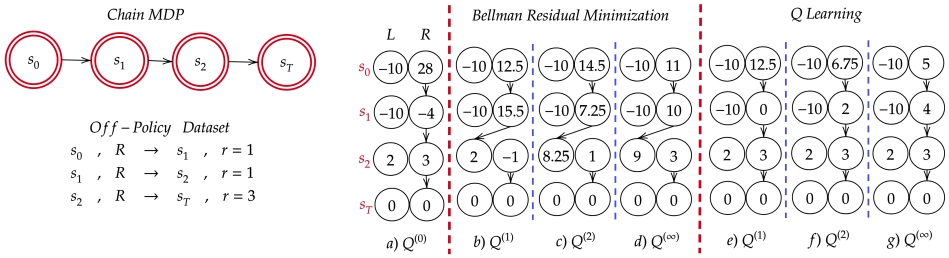

Figure 4: A chain MDP when Q-learning and DBRM behave differently under non-exploratory data. The top-lef shows the MDP with "L" and "R" actions equal in transition and different in rewards. The buttom left table shows the off-policy data-set where "L" is never taken. (a) shows an initial Q-function for both DBRM (b–d) and Q-learning (e–g) (learning rate is set to 0.25). The dashed arrows denote $s, a \to s', a'$ transitions where $a' = \arg\max_{a''} Q(s', a'')$. A state-action pair without an arrow has its influence blocked out. These arrows remain still using Q-learning, but, DBRM suffers from an early drift and due to lack of exploration never recovers.

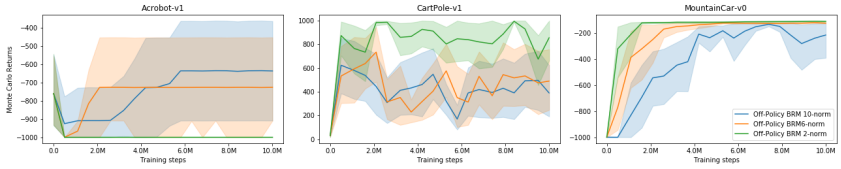

Figure 5: Training off-policy p-norm objectives for 10M steps. The samples were the same 2.5M ones used in last experiment ($p \in \{2, 6, 10\}$).

6 Related Work

Bradtke [1993] proposed BRM with quadratic function approximation with convergence guarantees. Williams and Baird [1993] proved tight bounds for relating the Bellman residuals to the policy performance. Baird [1995], Sutton et al. [2009] proved BRM methods converge to a local optimum using any function approximation. Dayan [1992] reported some conditions where ADP diverges with Tsitsiklis and Van Roy [1997], Baird [1995] reinforcing this matter. Schoknecht and Merke [2003] used spectral analysis to prove ADP being faster with no function approximation.

Scherrer [2010] studied the stability vs faster convergence trade-off with linear approximation, and found ADP to perform marginally better only when numerically stable. Zhang et al. [2019] explored BRM in DDPG and found superior empirical results. Despite the extensively studied relationship of ADP and BRM, our focus on deterministic environments is novel to the best of our knowledge.

7 Conclusion

In this paper 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 benchmarks, and proposition 1 requiring weaker conditions than deterministic dynamics, it is unclear if such performance will scale to complex (deterministic) environments. In fact, environmental determinism in function approximation settings is ill-defined, as all RL environments can be viewed as deterministic (with random initial states) [Ng and Jordan, 2000]. In other words, *misspecified function approximation and stochasticity are inherently indistinguishable*. This hints at an interesting use of DBRM for testing the stochasticity of an environment (just as in Figure 2).

References

A. Antos, C. Szepesvári, and R. 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.

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

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

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

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

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

B. Dai, A. Shaw, L. Li, L. Xiao, N. He, Z. Liu, J. Chen, and L. Song. Sbeed: Convergent reinforcement learning with nonlinear function approximation. In *International Conference on Machine Learning*, pages 1133–1142, 2018.

C. Dann, N. Jiang, A. Krishnamurthy, A. Agarwal, J. Langford, and R. E. Schapire. On Oracle-Efficient PAC RL with Rich Observations. In *Advances in Neural Information Processing Systems*, pages 1429–1439, 2018.

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

D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

J. Fu, A. Kumar, M. Soh, and S. Levine. Diagnosing bottlenecks in deep q-learning algorithms. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2021–2030, 2019.

S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2052–2062, 2019.

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

O.-A. Maillard, R. Munos, A. Lazaric, and M. Ghavamzadeh. Finite-sample analysis of bellman residual minimization. In *Proceedings of 2nd Asian Conference on Machine Learning*, pages 299–314, 2010.

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

A. Y. Ng and M. 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.

M. Puterman. Markov Decision Processes. *Jhon Wiley & Sons, New Jersey*, 1994.

B. 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.

R. Schoknecht and A. 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.

R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, and E. 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.

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

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

R. J. Williams and L. C. Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical report, Citeseer, 1993.

S. Zhang, W. Boehmer, and S. Whiteson. Deep residual reinforcement learning, 2019.

A On-Policy Experimental details in Deterministic Environments

Environments We conduct all our experiments in this paper on 3 classical control domains: Mountain-Car, Cart-Pole, and Acrobot from the OpenAI Gym collection [Brockman et al., 2016]. These domains have continuous state space of dimensions 2, 4, and 6, and discrete action spaces of cardinalities 3, 2, and 3, respectively.

Function Approximation For both DBRM and Q-learning, 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.

Exploration Scheme In this section we compare the "on-policy" versions of Q-learning and DBRM;[5] the comparison of their off-policy versions will be done in Section 5. On-policy agent uses the 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.

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.

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.

B Experimental Details of the sensitivity to Off-policyness and distribution mismatch

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.

C Additional Results

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.

Further Details on the Residual Drift issue in practice The residual drifting issue in the Acro-bot domain is better illustrated in figure 7. The experiments were conducted with 20 restarts, and ten different independent off-policy and on-policy trails. The trajectories were cut off after 200 time

[5] 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 5, where all algorithms use exactly the same dataset.

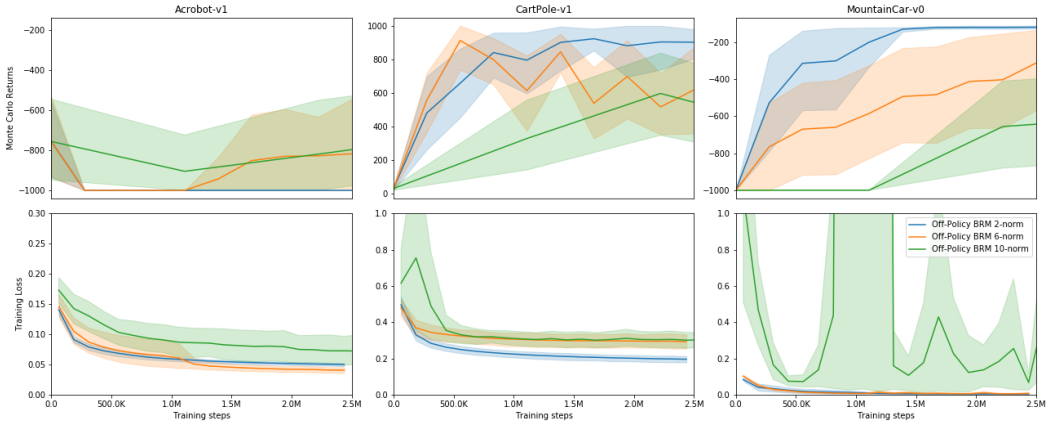

Figure 6: The effect of higher p-norms on the DBRM off-policy training process. The exploration buffer was used in this experiment instead of the replay buffer. The training here used 2.5 million steps. The 10-norm was chosen for further training, since it seemed to be performing better on the problematic 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.

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

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

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

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

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

Figure 7(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.

Trial plots More revealing swarm plots showing trial performances one by one are given below.

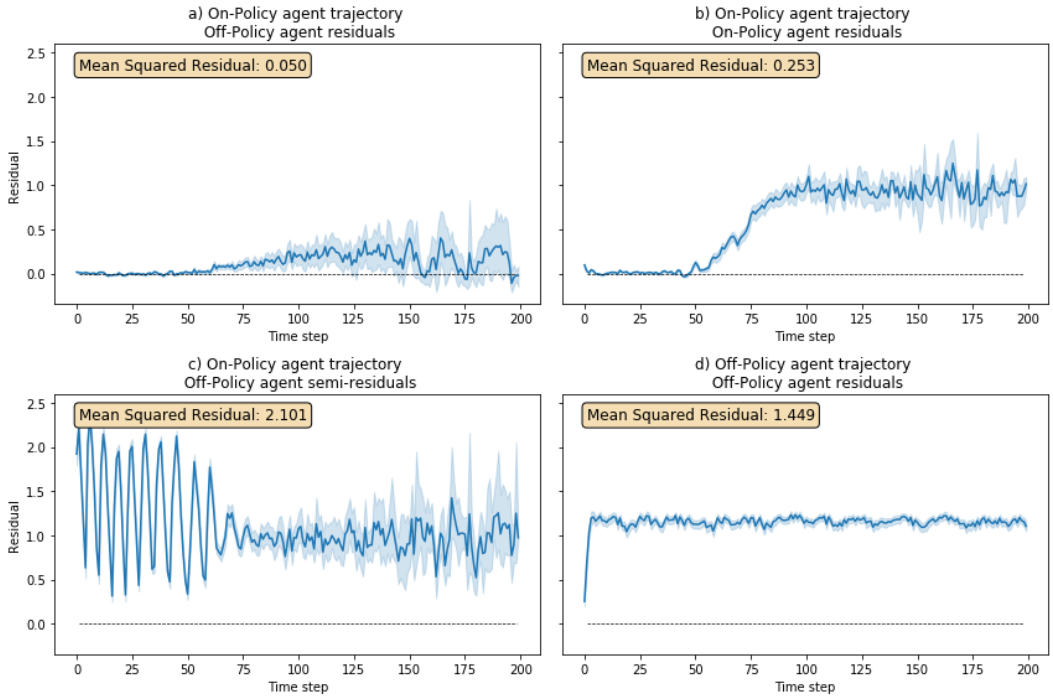

Figure 7: Residual drift suggestions in Acrobot domain. The horizontal axis represents time steps during the simulated trajectories, and not training progress (i.e. other words, the plots were created with fully trained off-policy and on-policy agents). See above for clarification on each type of residuals plotted.

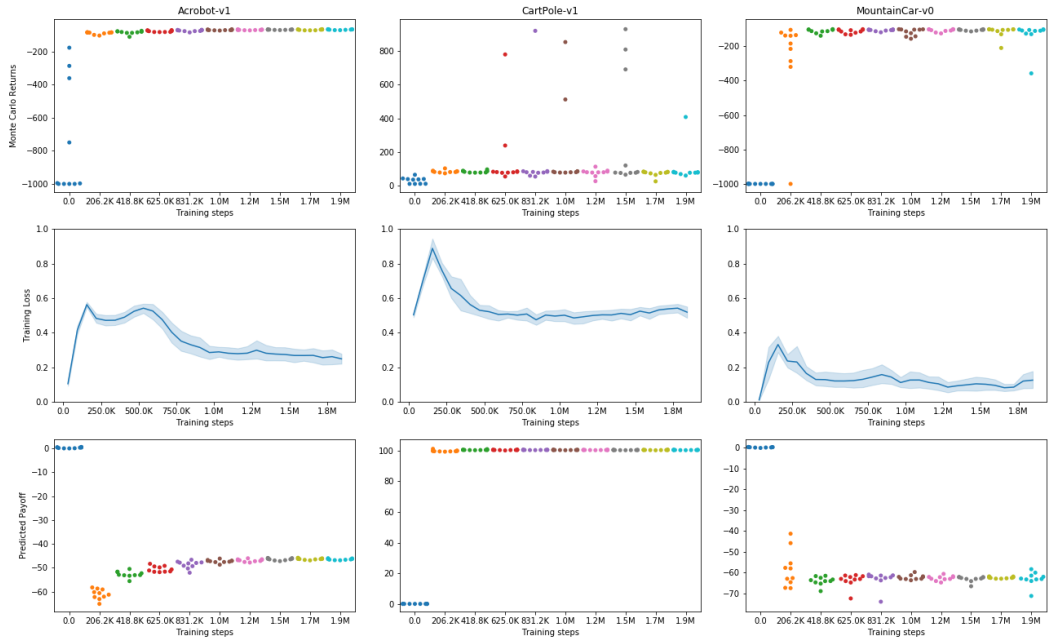

Figure 8: The swarm plots of trial-by-trial performance for on-policy ADP. The predict payoff shows the average value of the Q-function learned on the initial states.

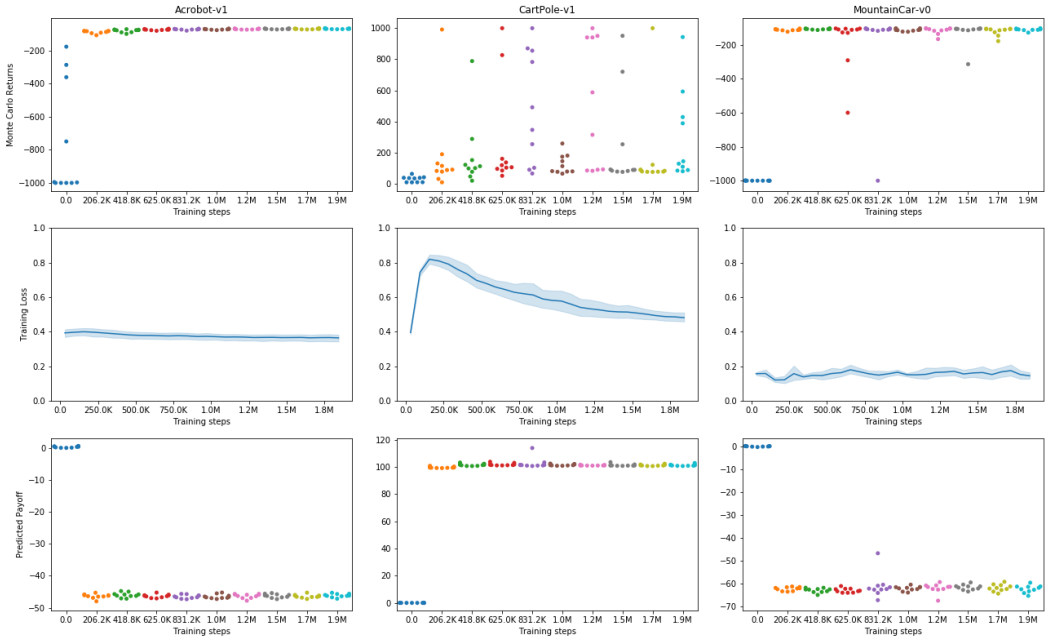

Figure 9: The swarm plots of trial-by-trial performance for off-policy ADP. The predict payoff shows the average value of the Q-function learned on the initial states.

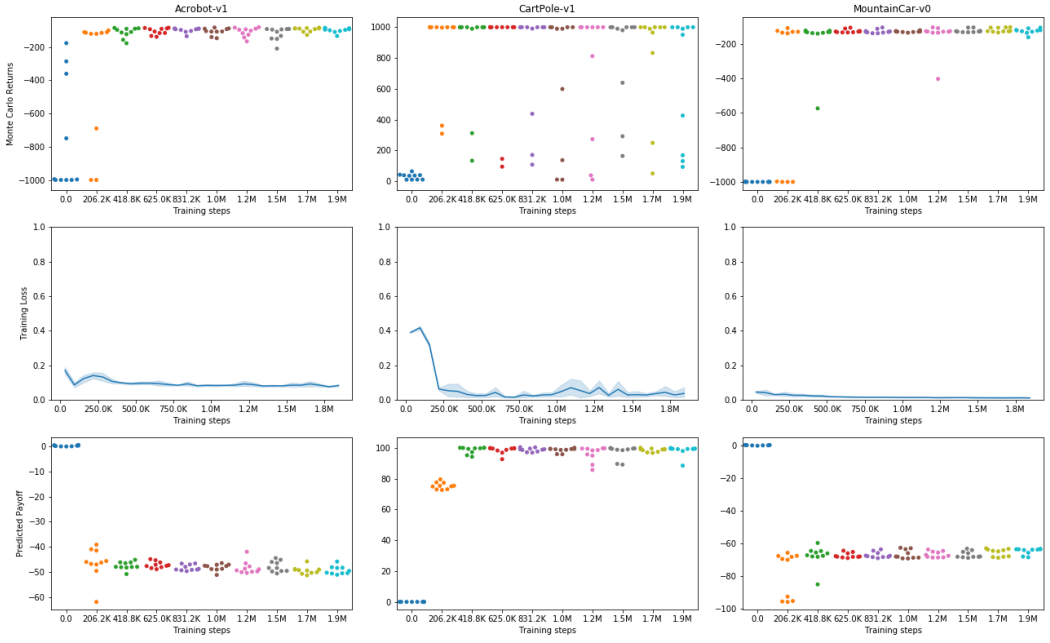

Figure 10: The swarm plots of trial-by-trial performance for on-policy DBRM. The predict payoff shows the average value of the Q-function learned on the initial states.

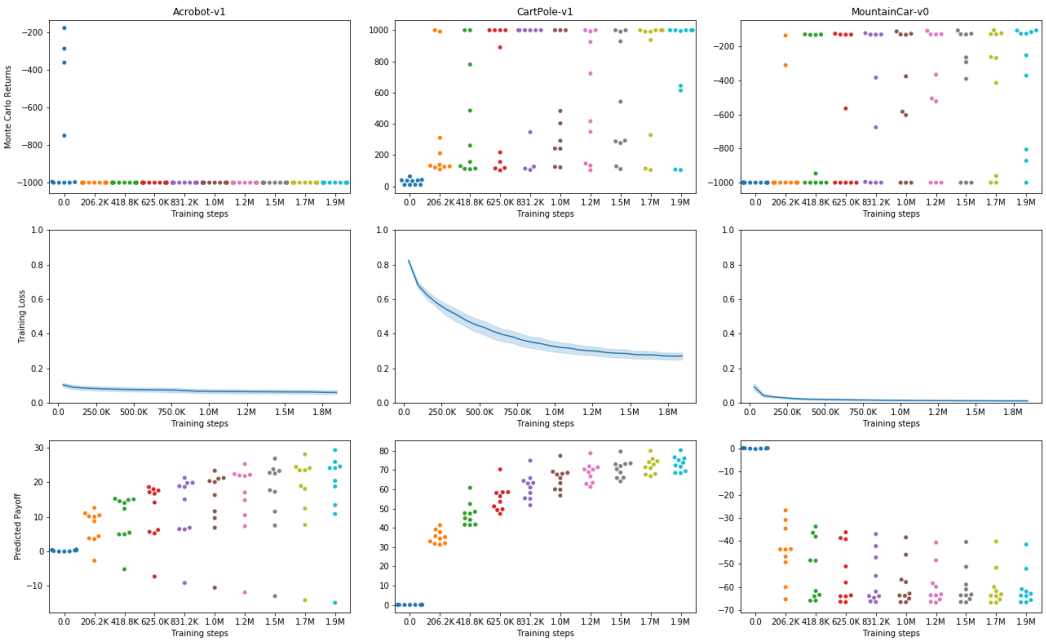

Figure 11: The swarm plots of trial-by-trial performance for off-policy DBRM. The predict payoff shows the average value of the Q-function learned on the initial states.