DisCor: Corrective Feedback in Reinforcement Learning via Distribution Correction

Aviral Kumar [1] **Abhishek Gupta** [1] **Sergey Levine** [1]

Abstract

In this paper, we study how deep RL methods based on bootstrapping-based Q-learning can suffer from a pathological interaction between function approximation and the data distribution used to train the Q-function: with standard supervised learning, online data collection should induce corrective feedback, where new data corrects mistakes in old predictions. With dynamic programming methods like Q-learning, such feedback may be absent. This can lead to potential instability, sub-optimal convergence, and poor results when learning from noisy, sparse or delayed rewards. Based on these observations, we propose a new algorithm, DisCor, which explicitly optimizes for data distributions that can correct for accumulated errors in the value function. Using a tractable approximation of this distribution for training, DisCor results in substantial improvements in a range of challenging RL settings, such as multi-task learning and learning from noisy reward signals.

1. Introduction

Reinforcement learning (RL) algorithms, when combined with high-capacity deep neural net function approximators, have shown promise in domains ranging from robotic manipulation (Kalashnikov et al., 2018) to recommender systems (Shani et al., 2005). However, current deep RL methods can be difficult to use, due to sensitivity with respect to hyperparameters and inconsistent and unstable convergence. We hypothesize that one source of instability in reinforcement learning with function approximation and value function estimation, such as Q-learning (Watkins and Dayan, 1992; Riedmiller, 2005; Mnih et al., 2015) and actor-critic algorithms (Haarnoja et al., 2017; Konda and Tsitsiklis, 2002), is a pathological interaction between the data distribution induced by the latest policy, and the errors induced in the learned approximate value function as a consequence of training on this distribution.

While a number of prior works (Achiam et al., 2019; Fu et al., 2019; Liu and Zou, 2018) have provided a theoretical examination of various approximate dynamic programming (ADP) methods, prior work has not extensively studied the relationship between the data distribution induced by the latest value function and the *errors* in the future value functions obtained by training on this data. When using supervised learning to train contextual bandits or dynamics models, online data collection results in a kind of "hard negative" mining: the model collects transitions that lead to good outcomes according to the model (potentially erroneously). This results in collecting precisely the data needed to *correct* errors and improve. On the contrary, ADP algorithms that use bootstrapped targets rather than ground-truth target values may not enjoy such corrective feedback with online data collection in the presence of function approximation.

Since function approximation couples Q-values at different states, the data distribution under which ADP updates are performed directly affects the learned solution. As we will argue in Section 2, online data collection may give rise to distributions that fail to correct errors in Q-values at states that are used as bootstrapping targets due to this coupling effect. If the bootstrapping targets in ADP updates are themselves are erroneous, then any form of Bellman error minimization using these targets may not result in the correction of errors in the Q-function, leading to poor performance. In this work, we show that we can explicitly address this by modifying the ADP training routine to re-weight the data buffer to a distribution that explicitly optimizes for corrective feedback, giving rise to our proposed method, **DisCor**. With DisCor, transitions sampled from the data buffer are reweighted with weights that are inversely proportional to the estimated errors in their target values. Thus, transitions with erroneous targets are down-weighted. We show that DisCor can be derived from a principled objective that results in a simple algorithm that reweights the training distribution based on estimated target value error, so as to mitigate error accumulation. In practice, DisCor can be used in conjunction with several modern deep RL algorithms (DQN or SAC), and as we show, it substantially improves performance in challenging RL settings, such as multi-task RL and complex manipulation tasks.

[1]UC Berkeley. Correspondence to: Aviral Kumar <aviralk@berkeley.edu>.

Accepted at Workshop on Inductive Biases, Invariances and Generalization, International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

Background. An MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. $\mathcal{S}, \mathcal{A}$ represent state and action spaces, $P(s'|s,a)$ and $r(s,a)$ represent the dynamics and reward function, and $\gamma \in (0,1)$ represents the discount factor. The discounted marginal state (and state-action) distribution of the policy $\pi(a|s)$ is denoted as $d^\pi(s)$ (and $d^\pi(s,a) = d^\pi(s)\pi(a|s)$). We define P^π, the state-action transition matrix under a policy π as $P^\pi Q(s,a) := \mathbb{E}_{s' \sim P(\cdot|s,a), a' \sim \pi(a'|s')}[Q(s',a')]$. V^* and Q^* denote the optimal state-action value function obtained by recursively iterating the Bellman optimality operator, $\mathcal{B}^*$. With function approximation, these algorithms project the values of the Bellman optimality operator $\mathcal{B}^*$ onto a family of Q-function approximators, under a sampling or data distribution μ. Q-function fitting is usually interleaved with additional data collection, which typically uses a policy derived from the latest value function, augmented with either ϵ-greedy (Watkins, 1989; Mnih et al., 2015) or Boltzmann-style (Haarnoja et al., 2018; Sutton et al.) exploration. For commonly used ADP methods, μ simply corresponds to the on-policy state-action marginal, $\mu_k = d^{\pi_k}$ (at iteration k) or else a "replay buffer" (Mnih et al., 2015; Lillicrap et al., 2015; Lin, 1992) formed as a mixture distribution over all past policies, such that $\mu_k = 1/k \sum_{i=1}^{k} d^{\pi_i}$.

2. Corrective Feedback in Q-Learning

When learning with supervised regression onto the true value function (e.g., in a bandit setting), active data collection methods will visit precisely those state-action tuples that have erroneously optimistic values, observe their true values, and correct the errors, by fitting these true values. However, ADP methods that use bootstrapped target values may not be able to correct errors this way, and online data collection may not reduce the error between the current Q-function and Q^*. This is because function approximation error can result in erroneous bootstrap target values at some state-action tuples. Visiting these tuples more often will simply cause the function approximator to more accurately fit these *incorrect* target values, rather than correcting the target values themselves.

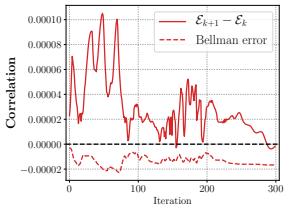
Didactic example. To build intuition, consider tree-structured MDP example in Figure 1. We illustrate a potential run of Q-learning (Alg. 2) with on-policy data collection. Q-values at different states are updated to match their (potentially incorrect) bootstrap target values under a distribution, $\mu(s,a)$, which, in this case is dictated by the visitation frequency under the current policy. The choice of $\mu(s,a)$, does not affect the resulting Q-function when function approximation is *not* used, as long as μ is full-support. However, with function approximation, updates across state-action pairs affect each other. Erroneous updates higher up in the tree, trying to match incorrect target values, may prevent error correction at leaf nodes if the states have similar representations under function approximation (i.e., if they are partially *aliased*). States closer to the root have higher frequencies (because there are fewer of them) than the leaves, exacerbating this problem. This issue can compound: the resulting erroneous leaf values are again used as targets for other nodes, which may have higher frequencies, further preventing the leaves from learning correct values.

If we can instead train with $\mu(s,a)$ that puts higher probability on nodes with correct target values, we can alleviate this issue. We would expect that such a method would first fit the most accurate target values (at the leaves), and only then update the nodes higher up, as shown in Figure 1 (right). Our proposed algorithm, DisCor, shows how to construct such a distribution in Section 3.

Value error in ADP. To more formally quantify, and devise solutions to this issue, we first define our notion of error correction in ADP in terms of *value error*:

Definition 2.1. *The value error is defined as the error of the current Q-function, Q_k to the optimal Q^* averaged under the on-policy marginal, $d^{\pi_k}(s,a)$: $\mathcal{E}_k = \mathbb{E}_{d^{\pi_k}}[|Q_k - Q^*|]$.*

A smooth decrease in value error $\mathcal{E}_k$ indicates that effective error correction in the Q-function. If $\mathcal{E}_k$ fluctuates or increases, the algorithm is making poor learning progress. When the value error $\mathcal{E}_k$ is roughly stagnant at a non-zero value, this indicates premature convergence.



To analyze this phenomenon computationally, we use the gridworld MDPs from Fu et al. (2019) and visualize the **correlations** between policy visitations $d^{\pi_k}(s,a)$ and the value of Bellman error after the ADP update, i.e. $|Q_{k+1} - \mathcal{B}^* Q_k|(s,a)$ (*dashed*), as well as the correlation between visitations and the *difference* in value errors after and before the update, $\mathcal{E}_{k+1}(s,a) - \mathcal{E}_k(s,a)$ (*solid*). As we see on the left, as expected, Bellman error correlates *negatively* with visitation frequency (dashed line), suggesting that visiting a state more often decreases its Bellman error. However, the change in value error $\mathcal{E}_{k+1} - \mathcal{E}_k$ in general *does not* correlate negatively with visitation. Value error often increases in states that are visited more frequently, suggesting that corrective feedback is often lacking. We discuss several consequences (suboptimal/premature convergence, inability to learn from sparse rewards) of this inability to correct value errors with on-policy distributions and function approximation in Appendix E due to lack of space.

In fact, we can construct a family of MDPs generalizing our didactic tree example where training with on-policy or replay buffer distributions theoretically requires at least exponentially many iterations to converge to Q^*, if at all convergence to Q^* happens. Proof is in Appendix D.

Theorem 2.1 (Exponential lower bound for on-policy and

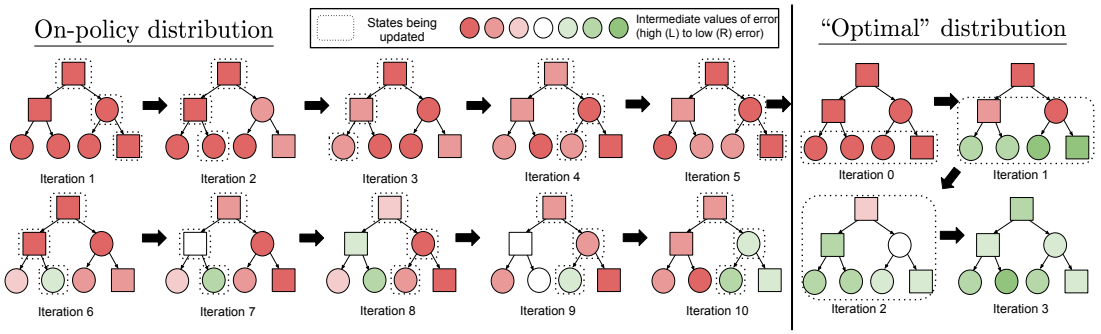


Figure 1. **Left:** Depiction of possible Q-learning iterations on a tree-structured MDP with on-policy sampling. The trajectory sampled at each iteration is shown with dotted boundaries. Function approximation results in aliasing (coupling) of the box-shaped and circle-shaped nodes (i.e., instances of each shape has similar features values). Updating the values at one circle node affects all other circles, likewise for boxes. Regressing to erroneous targets at one circle node may induce errors at another circle node, *even if the other node has a correct target*, simply because the other node is visited less often. **Right:** If a distribution that puts higher probability on nodes with correct target values, i.e. which moves from leaves to nodes higher up, is chosen, then, the effects of function approximation aliasing are reduced, and correct Q-values can be obtained.

replay buffer distributions). *There exists a family of MDPs parameterized by $H > 0$, with $|\mathcal{S}| = 2^H$, $|\mathcal{A}| = 2$ and state features Φ, such that on-policy or replay-buffer Q-learning requires $\Omega\left(\gamma^{-H}\right)$ exact Bellman projection steps for convergence to Q^*, if at all convergence happens. This happens even with features, Φ that can represent the optimal Q-function near-perfectly, i.e., $||Q^* - \Phi w||_\infty \leq \varepsilon$.*

3. Distributions for Value Error Reduction

We discussed how, with function approximation and on-policy or replay-buffer training distributions, the value error $\mathcal{E}_k$ may not decrease over the course of training. What if we instead directly optimize the data distribution at each iteration so as to minimize value error? To do so, we derive a functional form for this "optimal" distribution by formulating an optimization problem that directly optimizes the training distribution $p_k(s,a)$ at each iteration k, thus greedily minimizing $\mathcal{E}_k$, that results at the end of that iteration k. Note that $p_k(s,a)$ is now distinct from the on-policy or buffer data distribution denoted by $\mu(s,a)$. We will then show how to approximately solve for $p_k(s,a)$, yielding a simple practical algorithm in the next section. All proofs are in Appendix A. We can write the optimal $p_k(s,a)$ as the solution to the following optimization problem:

$$\min_{p_k} \ \mathbb{E}_{d^{\pi_k}}\left[|Q_k - Q^*|\right] \ \text{s.t.} \tag{1}$$

$$Q_k = \arg\min_{Q} \mathbb{E}_{p_k}\left[(Q - \mathcal{B}^* Q_{k-1})^2\right]. \tag{2}$$

Theorem 3.1. *The solution $p_k(s,a)$ to the optimization in Equation 1 satisfies*

$$p_k(s,a) \propto \exp\left(-|Q_k - Q^*|(s,a)\right) \frac{|Q_k - \mathcal{B}^* Q_{k-1}|(s,a)}{\lambda^*}, \tag{3}$$

where $\lambda^* \in \mathbb{R}^+$ is the magnitude of Lagrange multiplier for $\sum_{s,a} p_k(s,a) = 1$ in Problem 1.

Intuitively, the optimal p_k in Equation 3 assigns higher probability to state-action tuples with high Bellman error $|Q_k - \mathcal{B}^* Q_{k-1}|$, but only when the resulting Q-value Q_k is close to Q^*. However, this expression contains terms that depend on Q^* and Q_k, namely $|Q_k - Q^*|$ and $|Q_k - \mathcal{B}^* Q_{k-1}|$, which are observed only *after* p_k is chosen. As we will show next, we need to estimate these quantities using surrogates, that only depend upon the past Q-function iterates in order to use p_k in a practical algorithm. Intuitively, these surrogates exploit the rich structure in Bellman iterations: the Bellman error at each iteration contributes to the error against Q^*. We present these approximations below, and then combine then to derive our proposed algorithm, DisCor.

Surrogate for $|Q_k - Q^*|$. For approximating the error against Q^*, we show that the cumulative sum of discounted and propagated Bellman errors over the past iterations of training, denoted as Δ_k and shown in Equation 4, are equivalent to an upper bound on $|Q_k - Q^*|$ in Theorem 3.2. We define Δ_k as:

$$\Delta_k = \sum_{i=1}^{k} \gamma^{k-i} \left(\prod_{j=i}^{k-1} P^{\pi_j} \right) |Q_i - (\mathcal{B}^* Q_{i-1})|. \tag{4}$$

We can then use Δ_k to define an upper bound on the value error $|Q_k - Q^*|$, as follows:

Theorem 3.2. *There exists a $k_0 \in \mathbb{N}$, such that $\forall\ k \geq k_0$ and Δ_k from Equation 4, Δ_k satisfies the following inequality, pointwise, for each s, a, as well as, $\Delta_k \to |Q_k - Q^*|$ as $\pi_k \to \pi^*$.*

$$\Delta_k(s,a) + \sum_{i=1}^{k} \gamma^{k-i} \alpha_i \geq |Q_k - Q^*|(s,a),$$

$$\alpha_i = \frac{2R_{\max}}{1-\gamma} \mathrm{D_{TV}}(\pi_i(\cdot|s), \pi^*(\cdot|s)).$$

A proof and intermediate steps of simplification can be found in Appendix B. Using an upper bound of this form in Equation 3 may downweight more transitions, but will never upweight a transition that should not be upweighted.

Estimating $|Q_k - \mathcal{B}^*Q_{k-1}|$. The Bellman error multiplier term $|Q_k - \mathcal{B}^*Q_{k-1}|$ in Equation 3 is also not known in advance. A viable approximation is to bound $|Q_k - \mathcal{B}^*Q_{k-1}|$ between the minimum and maximum Bellman errors obtained at the previous iteration, $c_1 = \min_{s,a} |Q_{k-1} - \mathcal{B}^*Q_{k-2}|$ and $c_2 = \max_{s,a} |Q_{k-1} - \mathcal{B}^*Q_{k-2}|$. This approximation can be shown to be optimal in a minimax sense, and can be applied with Equation 3 to replace the Bellman error multiplier $|Q_k - \mathcal{B}^*Q_{k-1}|$, giving us a lower-bound on $p_k(s,a)$ in terms of c_1 and c_2.

Re-weighting the replay buffer μ. Since it is hard to directly obtain samples from p_k via online interaction, a practically viable alternative is to use the samples from a standard replay buffer distribution, denoted μ, but reweight these samples using importance weights $w_k = p_k(s,a)/\mu(s,a)$. Instead of directly re-weighting to p_k (which suffers from high variance importance weights), we re-weight samples from μ to a projection of p_k, denoted as q_k, that is still close to μ under the KL-divergence metric, such that $q_k = \arg\min_q \mathbb{E}_{q(s,a)}[\log p_k(s,a)] + \tau \mathrm{D}_{\mathrm{KL}}(q||\mu)$, where $\tau > 0$. The weights w_k are given by (App. B):

$$w_k(s,a) \propto \exp\left(\frac{-|Q_k - Q^*|(s,a)}{\tau}\right) \frac{|Q_k - \mathcal{B}^*Q_{k-1}|(s,a)}{\lambda^*} \tag{5}$$

Putting it all together. We have noted all practical, tractable approximations to the expression for optimal p_k (Equation 3), including estimating surrogates for Q_k and Q^*, and the usage of importance weights to simply *re-weighting transitions* in the replay buffer, rather than altering the exploration strategy. We now put these together to obtain a tractable expression for weights in our method, shown in Equation 6. Due to space limitations, we only provide the final expression of weights (which can be obtained as a lower bound on Equation 5) and discuss the derivation in Appendix C.

$$w_k(s,a) \propto \exp\left(-\frac{\gamma\left[P^{\pi_{k-1}}\Delta_{k-1}\right](s,a)}{\tau}\right). \tag{6}$$

Practical Algorithm. Our practical algorithm, **DisCor** (**Dis**tribution **Cor**rection), is shown in Algorithm 1, with the main differences from standard ADP methods highlighted in red. DisCor trains a parametric model, Δ_ϕ, to estimate $\Delta_k(s,a)$ at each state-action pair using the recursion in Equation 4 via an ADP update (Line 8). Δ_ϕ is required to compute the weights described in Equation 6. DisCor also introduces a weighted Q-function backup with weights $w_k(s,a)$ (Line 7). Since DisCor simply adds a change to the training distribution, this change can be applied to popular ADP algorithms such as DQN or SAC, as shown in Algorithm 3, Appendix H.

Intuitively, $(P^{\pi_{k-1}}\Delta_{k-1})(s,a)$ corresponds to the estimated upper bound on the error of the target values for the current transition, due to the backup operator $P^{\pi_{k-1}}$, as described in Equation 6. This *downweights* those transitions for which the bootstrapped *target* Q-value estimate has a high estimated error to Q^*, effectively focusing the learning on samples where the supervision (target value) is estimated to be accurate, which are precisely the samples that we expect maximally improve accuracy of the Q function.

Algorithm 1 DisCor (Distribution Correction)

1: Initialize Q-values $Q_\theta(s,a)$, initial distribution $p_0(s,a)$, a replay buffer μ, and an error model $\Delta_\phi(s,a)$.
2: **for** step k in $\{1, \ldots, \mathrm{N}\}$ **do**
3: Collect M samples using π_k, add them to replay buffer μ, sample $\{(s_i, a_i)\}_{i=1}^N \sim \mu$
4: Evaluate $Q_\theta(s,a)$ and $\Delta_\phi(s,a)$ on samples (s_i, a_i).
5: Compute target values for Q and Δ on samples:
$y_i = r_i + \gamma \max_{a'} Q_{k-1}(s_i', a')$
$\hat{a}_i = \arg\max_a Q_{k-1}(s_i', a)$
$\hat{\Delta}_i = |Q_\theta(s,a) - y_i| + \gamma \Delta_{k-1}(s_i', \hat{a}_i)$
6: Compute w_k using Equation 6.
7: Minimize Bellman error for Q_θ weighted by w_k.
$\theta_{k+1} \leftarrow \operatorname{argmin}_\theta \frac{1}{N}\sum_i^N w_k(s_i,a_i)(Q_\theta(s_i,a_i) - y_i)^2$
8: Minimize ADP error for training ϕ.
$\phi_{k+1} \leftarrow \operatorname{argmin}_\phi \frac{1}{N}\sum_{i=1}^N (\Delta_\phi(s_i,a_i) - \hat{\Delta}_i)^2$
9: **end for**

4. Experimental Evaluation of DisCor

In our experiments, we study the following questions: **(1)** Does DisCor lead to a decrease in value error, mitigating the issues raised in Section 2?, **(2)** How do approximations from Section 3 affect the efficacy of DisCor in ensuring value error reduction? **(3)** How does DisCor compare to prior methods, including those that also reweight the data in various ways?, **(4)** Can DisCor attain good performance in challenging settings, such as multi-task RL, robotic manipulation or Atari games? Due to space limitations, we only describe the key-takeaways from our empirical results, and refer the reader to experiments in App. F.

We first analyze DisCor and an oracle version of DisCor, that uses exact Q^* to compute $|Q_k - Q^*|$, on gridworld MDPs from (Fu et al., 2019), and find that empirically in Figure 4 that, **(a)** DisCor (oracle) is somewhat better than DisCor, both in terms of the ability to reduce value errors, and in terms of final policy performance, and **(b)** DisCor (and oracle) both outperform other sampling schemes, including on-policy, replay buffer, prioritization based on Bellman error similar to PER (Schaul et al., 2015).

DisCor can be applied in conjunction with deep RL methods such as DQN and SAC. As shown in Appendix F, DisCor outperforms standard unweighted SAC by **1.2x** in terms of success rate on robotic manipulation tasks from Meta-world (Yu et al., 2019). DisCor attains **50%** more than the success rate of SAC in challenging multi-task learning settings on the MT10 benchmark (Yu et al., 2019). On image-based Atari games, DisCor outperforms standard DQN, on all three games tested on: Pong, Breakout and Asterix.

Discussion. We showed that deep RL algorithms are unable to correct errors in the value function in scenarios with naïve online data collection, and we proposed an algorithm, DisCor, that re-weights the data distribution to induce maximal error correction. DisCor outperforms unweighted RL on a wide range of tasks, sometimes by **50%**.

References

Joshua Achiam, Ethan Knight, and Pieter Abbeel. Towards characterizing divergence in deep q-learning. *ArXiv*, abs/1903.08894, 2019.

Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. Understanding the impact of entropy on policy optimization. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2019. URL http://proceedings.mlr.press/v97/ahmed19a.html.

Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G. Bellemare. Dopamine: A Research Framework for Deep Reinforcement Learning. 2018. URL http://arxiv.org/abs/1812.06110.

Adithya M. Devraj and Sean P. Meyn. Zap q-learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 2232–2241, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Simon Du, Yuping Luo, Ruosong Wang, and Hanrui Zhang. Provably efficient q-learning with function approximation via distribution shift error checking oracle. In *NeurIPS*, 06 2019.

Simon S. Du, Sham M. Kakade, Ruosong Wang, and Lin F. Yang. Is a good representation sufficient for sample efficient reinforcement learning? In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1genAVKPB.

Amir-massoud Farahmand, Csaba Szepesvári, and Rémi Munos. Error propagation for approximate policy and value iteration. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.

D. Farias and B. Roy. On the existence of fixed points for approximate value iteration and temporal-difference learning. *Journal of Optimization Theory and Applications*, 105:589–608, 06 2000. doi: 10.1023/A:1004641123405.

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*. PMLR, 2019. URL http://proceedings.mlr.press/v97/fu19a.html.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, pages 1587–1596, 2018.

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

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning (ICML)*, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL http://arxiv.org/abs/1801.01290.

Elad Hazan, Sham Kakade, Karan Singh, and Abby Van Soest. Provably efficient maximum entropy exploration. 2019. URL https://arxiv.org/pdf/1812.02690.pdf.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. *CoRR*, abs/1803.00933, 2018. URL http://arxiv.org/abs/1803.00933.

Yuenan Hou, Lifeng Liu, Qing Wei, Xudong Xu, and Chunlin Chen. A novel ddpg method with prioritized experience replay. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 316–321. IEEE, 2017.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems 31*. 2018.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *CoRL*, 2018.

Vijaymohan Konda and John N. Tsitsiklis. *Actor-Critic Algorithms*. PhD thesis, USA, 2002. AAI0804543.

Steven G. Krantz and Harold R. Parks. The implicit function theorem: History, theory, and applications. 2002.

Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. 2019. URL http://arxiv.org/abs/1906.00949.

Boris Lesner and Bruno Scherrer. Tight performance bounds for approximate modified policy iteration with non-stationary policies. *ArXiv*, abs/1304.5610, 2013.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *International Conference on Learning Representations (ICLR)*, 2015.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Ruishan Liu and James Zou. The effects of memory replay in reinforcement learning. In *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2018.

Vincent Liu, Raksha Kumaraswamy, Lei Le, and Martha White. The utility of sparse representations for control in reinforcement learning. *CoRR*, abs/1811.06626, 2018. URL http://arxiv.org/abs/1811.06626.

Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *J. Artif. Int. Res.*, 61(1):523–562, January 2018. ISSN 1076-9757.

Hamid R. Maei, Csaba Szepesvári, Shalabh Bhatnagar, Doina Precup, David Silver, and Richard S. Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, 2009.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, feb 2015. ISSN 0028-0836.

Rémi Munos. Error bounds for approximate policy iteration. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 560–567. AAAI Press, 2003. ISBN 1577351894.

Rémi Munos. Error bounds for approximate value iteration. In *AAAI Conference on Artificial intelligence (AAAI)*, pages 1006–1011. AAAI Press, 2005.

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

Theodore J. Perkins and Doina Precup. A convergent form of approximate policy iteration. NIPS'02, 2002.

Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

R. Tyrrell Rockafellar. *Convex analysis*. Princeton Mathematical Series. Princeton University Press, Princeton, N. J., 1970.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *International Conference on Learning Representations (ICLR)*, 2015.

Tom Schaul, Diana Borsa, Joseph Modayil, and Razvan Pascanu. Ray interference: a source of plateaus in deep reinforcement learning. *CoRR*, abs/1904.11455, 2019. URL http://arxiv.org/abs/1904.11455.

Bruno Scherrer. Approximate policy iteration schemes: A comparison. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page II–1314–II–1322. JMLR.org, 2014.

Bruno Scherrer, Mohammad Ghavamzadeh, Victor Gabillon, Boris Lesner, and Matthieu Geist. Approximate modified policy iteration and its application to the game of tetris. *Journal of Machine Learning Research*, 16 (49):1629–1676, 2015. URL http://jmlr.org/papers/v16/scherrer15a.html.

Guy Shani, David Heckerman, and Ronen I Brafman. An mdp-based recommender system. *Journal of Machine Learning Research*, 6(Sep):1265–1295, 2005.

Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*.

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 *International Conference on Machine Learning (ICML)*, 2009a.

Richard S. Sutton, Hamid Reza Maei, and Csaba Szepesvári. A convergent o(n) temporal-difference algorithm for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2009b.

Sunil Thulasidasan, Tanmoy Bhattacharya, Jeff Bilmes, Gopinath Chennupati, and Jamal Mohd-Yusof. Combating label noise in deep learning using abstention. In *Proceedings of 35th International Conference on Machine Learning*, 05 2019.

John N. Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine Learning*, 16(3):185–202, Sep 1994. ISSN 1573-0565. doi: 10.1007/BF00993306. URL https://doi.org/10.1007/BF00993306.

John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. Technical report, IEEE Transactions on Automatic Control, 1997.

Kristian Hartikainen George Tucker Sehoon Ha Jie Tan Vikash Kumar Henry Zhu Abhishek Gupta Pieter Abbeel Tuomas Haarnoja, Aurick Zhou and Sergey Levine. Soft actor-critic algorithms and applications. Technical report, 2018.

Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *ArXiv*, abs/1812.02648, 2018.

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

Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989. URL http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019. URL https://arxiv.org/abs/1910.10897.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020.

Appendices

A. Detailed Proof of Theorem 3.1 (Section 3)

In this appendix, we present a detailed proofs for the theoretical derivation of DisCor outlined in Section 3. To get started, we mention the optimization problem being solved for convenience.

$$\begin{aligned} \min_{p_k} \;& \mathbb{E}_{d^{\pi_k}}\left[|Q_k - Q^*|\right] \\ \text{s.t. } & Q_k = \arg\min_{Q} \mathbb{E}_{p_k}\left[\left(Q - \mathcal{B}^* Q_{k-1}\right)^2\right]. \end{aligned} \tag{7}$$

We break down this derivation in steps marked as relevant paragraphs. The first step is to decompose the objective into a more tractable one via an application of the Fenchel-Young inequality (Rockafellar, 1970).

Step 1: Fenchel-Young Inequality. The optimization objective in Problem 7 is the inner product of $d^{\pi_{k-1}}$ and $|Q_k - Q^*|$. We can decompose this objective by applying the Fenchel-Young inequality (Rockafellar, 1970). For any two vectors, $x, y \in \mathbb{R}^d$, and any convex function f and its Fenchel conjugate f^*, we have that, $x^T y \leq f(x) + f^*(y)$. We therefore have:

$$\mathbb{E}_{d^{\pi_k}}\left[|Q_k - Q^*|\right] \leq f\left(|Q_k - Q^*|\right) + f^*\left(d^{\pi_k}\right). \tag{8}$$

Since minimizing an upper bound leads to minimization of the original objective, we can replace the objective in Problem 7 with the upper bound in Equation 8. As we will see below, a convenient choice of f is the *soft-min* function:

$$f(x) = -\log\left(\sum_i e^{-x_i}\right), \quad f^*(y) = \mathcal{H}(y). \tag{9}$$

f^* in this case is given by the Shannon entropy, which is defined as $\mathcal{H}(y) = -\sum_j y_j \log y_j$. Plugging this back in problem 7, we obtain an objective that dictates minimization of the marginal state-action entropy of the policy π.

In order to make this objective even more convenient and tractable, we upper bound the Shannon entropy, $\mathcal{H}(y)$ by the entropy of the uniform distribution over states and actions, $\mathcal{H}(\mathcal{U})$. This step ensures that the entropy of the state-action marginal d^π is not reduced drastically due to the choice of p. We can now minimize this upper bound, since minimizing an upper bound, leads to a minimization of the original problem, and therefore, we obtain the following new optimization problem shown in Equation 10 is:

$$\begin{aligned} \min_{p_k} \;& -\log\left(\sum_{s,a} \exp(-|Q_k - Q^*|(s,a))\right) \\ \text{s.t. } & Q_k = \arg\min_{Q} \mathbb{E}_{p_k}\left[\left(Q - \mathcal{B}^* Q_{k-1}\right)^2\right]. \end{aligned} \tag{10}$$

Another way to interpret this step is to modify the objective in Problem 7 to maximize entropy-augmented V_{CF}: $\mathrm{V}_{\mathrm{CF}}(k) + \mathcal{H}(d^{\pi_k})$ as is common in a number of prior works, albeit with entropy over different distributions such as (Hazan et al., 2019; Haarnoja et al., 2018). This also increases the smoothness of the loss landscape, which is crucial for performance of RL algorithms (Ahmed et al., 2019).

Step 2: Computing the Lagrangian. In order to solve optimization problem Problem 10, we follow standard procedures for finding solutions to constrained optimization problems. We first write the Lagrangian for this problem, which includes additional constraints to ensure that p_k is a valid distribution:

$$\mathcal{L}(p_k; \lambda, \mu) = -\log\left(\sum_{s,a} \exp(-|Q_k - Q^*|(s,a))\right) + \lambda\left(\sum_{s,a} p_k(s,a) - 1\right) - \mu^T p_k. \tag{11}$$

with constraints $\sum_{s,a} p_k(s,a) = 1$ and $p_k(s,a) \geq 0$ $(\forall s, a)$ and their corresponding Lagrange multipliers, λ and μ, respectively, that ensure p_k is a valid distribution. An optimal p_k is obtained by setting the gradient of the Lagrangian with respect to p_k to 0. This requires computing the gradient of Q_k, resulting from Bellman error minimization, i.e. computing the derivative through the solution of another optimization problem, with respect to the distribution p_k. We use the implicit function theorem (IFT) (Krantz and Parks, 2002) to compute this gradient. We next present an application of IFT in our scenario.

Step 3: IFT gradient used in the Lagrangian. We derive an expression for $\frac{\partial Q_k}{\partial p_k}$ which will be used while computing the gradient of the Lagrangian Equation 11 which involves an application of the implicit function theorem. The IFT gradient is given by:

$$\left.\frac{\partial Q_k}{\partial p_k}\right|_{Q_k, p_k} = -\left[\text{Diag}(p_k)\right]^{-1}\left[\text{Diag}(Q_k - \mathcal{B}^* Q_{k-1})\right]. \tag{12}$$

To get started towards showing Equation 12, we consider a non-parametric representation for Q_k (a table), so that we can compute a tractable term without going onto the specific calculations for Jacobian or inverse-Hessian vector products for different parametric models. In this case, the Hessians in the expression for IFT and hence, the implicit gradient are given by:

$$H_Q = 2\,\text{Diag}(p_k) \qquad H_{Q,p_k} = 2\,\text{Diag}(Q_k - \mathcal{B}^* Q_{k-1})$$
$$\frac{\partial Q_k}{\partial p_k} = -\left[H_Q\right]^{-1} H_{Q,p_k} = -\text{Diag}\left(\frac{Q_k - \mathcal{B}^* Q_{k-1}}{p_k}\right) \tag{13}$$

provided $p_k \geq 0$ (which is true, since we operate in a full coverage regime, as there is no exploration bottleneck when all transitions are provided). This quantity is 0 only if the Bellman residuals $Q_k - \mathcal{B}^* Q_{k-1}$ are 0, however, that is rarely the case, hence this gradient is non-zero, and intuitively quantifies the right relationship: Bellman residual errors $Q_k - \mathcal{B}^* Q_{k-1}$ should be higher at state-action pairs with low density p_k, indicating a roughly inverse relationship between the two terms – which is captured by the IFT term.

Step 4: Computing optimal p_k. Now that we have the equation for IFT (Equation 12) and an expression for the Lagrangian (Equation 11), we are ready to compute the optimal p_k via an application of the KKT conditions. At an optimal p_k, we have,

$$\frac{\partial \mathcal{L}(p_k; \lambda, \mu)}{\partial p_k} = 0 \implies \frac{\text{sgn}(Q_k - Q^*)\exp(-|Q_k - Q^*|(s,a))}{\sum_{s',a'}\exp(-|Q_k - Q^*|(s',a'))} \cdot \frac{\partial Q_k}{\partial p_k} + \lambda - \mu_{s,a} = 0. \tag{14}$$

Now, re-arranging Equation 14 and plugging in the expression for $\frac{\partial Q_k}{\partial p_k}$ from Equation 12 in this Equation to obtain an expression for $p_k(s,a)$, we get:

$$p_k(s,a) \propto \exp\left(-|Q_k - Q^*|(s,a)\right)\frac{|Q_k - \mathcal{B}^* Q_{k-1}|(s,a)}{\lambda^*}. \tag{15}$$

Provided, each component of p is positive, i.e. $p_k(s,a) \geq 0$ for all s, a, the optimal dual variable $\mu^*_{s,a} = 0$, satisfies $\mu^*(s,a)p_k(s,a) = 0$ by KKT-conditions, and $\mu^* \geq 0$ (since it is a Lagrange dual variable), thus implying that $\mu^* = \mathbf{0}$.

Intuitively, the expression in Equation 15 assigns higher probability to state-action tuples with high Bellman error $|Q_k - \mathcal{B}^* Q_{k-1}|$, but only when the *post-update* Q-value Q_k is close to Q^*. Hence we obtain the required theorem.

Summary of the derivation. To summarize, our derivation for the optimal p_k consists of the following key steps:

- Use the Fenchel-Young inequality to get a convenient form for the objective.
- Compute the Lagrangian, and use the implicit function theorem to compute gradients of the Q-function Q_k with respect to the distribution p_k.
- Compute the expression for optimal p_k by setting the Lagrangian gradient to 0.

B. Proofs for Tractable Approximations in Section 3

Here we present the proofs for the arguments behind each of the approximations described in Section 3.

Computing weights w_k for re-weighting the buffer distribution, μ. Since sampling directly from p_k may not be easy, we instead choose to re-weight samples transitions drawn from a replay buffer μ, using weights w_k to make it as close to p_k. How do we obtain the exact expression for $w_k(s,a)$? One option is to apply importance sampling: choose w_k as the importance ratio, $w_k(s,a) = \frac{p_k(s,a)}{\mu(s,a)}$, however this suffers from two problems – (1) importance weights tend to exhibit high

variance, which can be detrimental for stochastic gradient methods; and (2) densities $\mu(s,a)$, needed to compute w_k are unknown.

In order to circumvent these problems, we solve a different optimization problem, shown in Problem 16 to find the optimal *surrogate* projection distribution q_k, which is closest to p_k, under the expected likelihood metric, $\mathbb{E}_{q_k}[\log p_k]$, and at the same time closest to μ as well under the KL-divergence metric, trading off these quantities by a factor τ.

$$q_k^* = \arg\min_{q_k} \mathbb{E}_{q_k}\left[\log p_k\right] + (\tau)\mathrm{D}_{\mathrm{KL}}(q_k||\mu) \tag{16}$$

where λ is a temperature hyperparameter that trades of bias and variance. The solution to the above optimization is shown in Equation 17, where the second statement follows by using a tractable approximation of setting $\mu^{1-\frac{1}{\tau}}$ to be equal to μ, which can be ignored if τ is large, hence $1-\frac{1}{\tau} \approx 1$. The third statement follows by an application of Equation 15 and the fourth statement denotes the importance ratio, $\frac{q_k(s,a)}{\mu_k(s,a)}$, as the weights w_k.

$$\begin{aligned} q_k^*(s,a) &\propto (\mu_k) \cdot \exp\left(\frac{\log p_k(s,a)}{\tau}\right) \\ \therefore \frac{q_k^*}{\mu_k} &\propto \exp\left(\frac{-|Q_k - Q^*|(s,a)}{\tau}\right) \frac{|Q_k - \mathcal{B}^* Q_{k-1}|(s,a)}{\lambda^*} \end{aligned} \tag{17}$$

Our next proof justifies the usage of the estimate Δ_k, which is a worst-case upper bound on $|Q_k - Q^*|$ in Equation 17.

Proof of Theorem 3.2. We now present a Lemma B.0.1 which proves a recursive inequality for $|Q_k - Q^*|$, then show that the corresponding recursive estimator upper bounds $|Q_k - Q^*|$ pointwise in Lemma B.0.2, and then finally show that our chosen estimator Δ_k is equivalent to this recursive estimator in Theorem B.1 therefore proving Theorem 3.2.

Lemma B.0.1. *For any $k \in \mathbb{N}$, $|Q_k - Q^*|$ satisfies the following recursive inequality, pointwise for each s, a:*

$$|Q_k - Q^*| \leq |Q_k - \mathcal{B}^* Q_{k-1}| + \gamma P^{\pi_{k-1}}|Q_{k-1} - Q^*| + \frac{2R_{max}}{1-\gamma} \max_s \mathrm{D}_{\mathrm{TV}}(\pi_{k-1}, \pi^*).$$

Proof. Our proof relies on a worst-case expansion of the quantity $|Q_k - Q^*|$. The proof follows the following steps. The first few steps follow common expansions/inequalities operated upon in the work on error propagation in Q-learning (Munos, 2005).

$$\begin{aligned} |Q_k - Q^*| &\overset{(a)}{=} |Q_k - \mathcal{B}^* Q_{k-1} + \mathcal{B}^* Q_{k-1} - Q^*| \\ &\overset{(b)}{\leq} |Q_k - \mathcal{B}^* Q_{k-1}| + |\mathcal{B}^* Q_{k-1} - \mathcal{B}^* Q^*| \\ &\overset{(c)}{=} |Q_k - \mathcal{B}^* Q_{k-1}| + |R + \gamma P^{\pi_{k-1}} Q_{k-1} - R - \gamma P^{\pi^*} Q^*| \\ &\overset{(d)}{=} |Q_k - \mathcal{B}^* Q_{k-1}| + \gamma |P^{\pi_{k-1}} Q_{k-1} - P^{\pi_{k-1}} Q^* + P^{\pi_{k-1}} Q^* - P^{\pi^*} Q^*| \\ &\overset{(e)}{\leq} |Q_k - \mathcal{B}^* Q_{k-1}| + \gamma P^{\pi_{k-1}} |Q_{k-1} - Q^*| + \gamma |P^{\pi_{k-1}} - P^{\pi^*}||Q^*| \\ &\overset{(f)}{\leq} |Q_k - \mathcal{B}^* Q_{k-1}| + \gamma P^{\pi_{k-1}} |Q_{k-1} - Q^*| + \frac{2R_{\max}}{1-\gamma} \max_s \mathrm{D}_{\mathrm{TV}}(\pi_{k-1}, \pi^*) \end{aligned}$$

where (a) follows from adding and subtracting $\mathcal{B}^* Q_{k-1}$, (b) follows from an application of triangle inequality, (c) follows from the definition of $\mathcal{B}^*$ applied to two different Q-functions, (d) follows from algebraic manipulation, (e) follows from an application of the triangle inequality, and (f) follows from bounding the maximum difference in transition matrices $|P^{\pi_{k-1}} - P^*|$ by maximum total variation divergence between policy π_{k-1} and π^*, and bounding the maximum possible value of Q^* by $\frac{R_{\max}}{1-\gamma}$.

□

We next show that an estimator that satisfies the recursive equality corresponding to Lemma B.0.1 is a pointwise upper bound on $|Q_k - Q^*|$.

Lemma B.0.2. *For any $k \in \mathbb{N}$, an vector Δ'_k satisfying*

$$\Delta'_k := |Q_k - \mathcal{B}^* Q_{k-1}| + \gamma P^{\pi_{k-1}} \Delta'_{k-1}. \tag{18}$$

with $\alpha_k = \frac{2R_{\max}}{1-\gamma} \max_s \mathrm{D_{TV}}(\pi_k, \pi^*)$, and an initialization $\Delta'_0 := |Q_0 - Q^*|$, pointwise upper bounds $|Q_k - Q^*|$ with an offset depending on α_i, i.e. $\Delta'_k + \sum_i \alpha_i \gamma^{k-i} \geq |Q_k - Q^*|$.

Proof. Let Δ'_k be an estimator satisfying Equation 18. In order to show that $\Delta'_k + \sum_i \gamma^{k-i}\alpha_i \geq |Q_k - Q^*|$, we use the principle of mathematical induction. The base case, $k = 0$ is satisfied, since $\Delta'_0 + \alpha_0 \geq |Q_0 - Q^*|$. Now, let us assume that for a given $k = m$, $\Delta'_m + \sum_i \gamma^{m-i}\alpha_i \geq |Q_m - Q^*|$ pointwise for each (s, a). Now, we need to show that a similar relation holds for $k = m + 1$, and then we can appeal to the principle of mathematical induction to complete the argument. In order to show this, we note that,

$$\begin{aligned}
\Delta'_{m+1} =& |Q_{m+1} - \mathcal{B}^* Q_m| + \gamma P^{\pi_m} \Delta'_m + \sum_i^{m+1} \gamma^{m+1-i}\alpha_i && (19)\\
=& |Q_{m+1} - \mathcal{B}^* Q_m| + \gamma P^{\pi_m} (\Delta'_m + \sum_{i=0}^{m} \gamma^{m-i}\alpha_i) + \alpha_{m+1} && (20)\\
\geq& |Q_{m+1} - \mathcal{B}^* Q_m| + \gamma P^{\pi_m} |Q_m - Q^*| + \alpha_m && (21)\\
\geq& |Q_{m+1} - Q^*| && (22)
\end{aligned}$$

where (19) follows from the definition of Δ'_k, (20) follows by rearranging the recursive sum containing α_i, for $i \leq m$ alongside Δ_m, (21) follows from the inductive hypothesis at $k = m$, and (22) follows from Lemma B.0.1.

Thus, by using the principle of mathematical induction, we have shown that $\Delta'_k + \sum_i \gamma^{k-i}\alpha_i \geq |Q_k - Q^*|$ pointwise for each s, a, for every $k \in \mathbb{N}$. □

The final piece in this argument is to show, that the estimator Δ_k used by the DisCor algorithm (Algorithm 1), which is initialized randomly, i.e. not initialized to $\Delta_0 = |Q_0 - Q^*|$, still satisfies the property from Lemma B.0.2, possibly for certain $k \in \mathbb{N}$.

Therefore, we now show why: $\Delta_k + \sum_{i=1}^{k} \alpha_i \gamma^{k-i} \geq |Q_k - Q^*|$ point-wise for a sufficiently large k. We restate a slightly modified version of Theorem 3.2 for convenience.

Theorem B.1 (Formal version of Theorem 3.2)**.** *For a sufficiently large $k \geq k_0 = \frac{\log(1-\gamma)}{\log \gamma}$, the error estimator Δ_k pointwise satisfies:*

$$\Delta_k + \sum_{i=0}^{k} \gamma^i \alpha_{k-i} \geq |Q_k - Q^*|$$

where α_i's are scalar constants independent of any state-action pair. (Note that Theorem 3.2 has a typo γ^i instead of γ^{k-i}, this theorem presents the correct inequality.)

Proof. *Main Idea/ Sketch:* As shown in Algorithm 1, the estimator Δ_k is initialized randomly, without taking into account $|Q_0 - Q^*|$. Therefore, in this theorem, we want to show that *irrespective* of the initialization of Q_0, a randomly initialized Δ_k eventually satisfies the inequality shown in Theorem 3.2. Now, we present the formal proof.

Consider k_0 to be the smallest k, such that the following inequality is satisfied:

$$\gamma^k \max_{Q_0, Q^*} |Q_0 - Q^*| \leq 1 \tag{23}$$

Thus, $k_0 \geq \frac{\log(1-\gamma)}{\log \gamma}$, assuming $R_{\max} = 1$ without loss of generality. For a different reward scaling, the bound can be scaled appropriately. To see this, we substitute $|Q_0 - Q^*|$ as an upper-bound $R_{\max}/(1-\gamma)$, and bound $R_{\max}$ by 1.

Let Δ'_k correspond to the upper-bound estimator as derived in Lemma B.0.2. For each $k \geq k_0$, the contribution of the initial error $|Q_0 - Q^*|$ in $|Q_k - Q^*|$ is upper-bounded by 1, and gradually decreases with a rate γ as more backups are

performed, i.e., as k increases. Thus we can construct another sequence $\Delta_1, \cdot, \Delta_k, \cdots$ which chooses to ignore $|Q_0 - Q^*|$, and initializes $\Delta_0 = 0$ (or randomly) and the sequences Δ and Δ'_k satisfy:

$$|\Delta'_k - \Delta_k| < 1, \forall k \geq k_0 \tag{24}$$

Furthermore, the difference $|\Delta'_k - \Delta_k|$ steadily shrinks to 0, with a linear rate γ, so the overall contribution of the initialization sub-optimality $|Q_0 - Q^*|$ drops linearly with a rate of γ. Hence, Δ' and Δ converge to the same sequence beyond a fixed $k = k_0$. Since Δ'_k is computed using the RHS of Lemma B.0.1, it is guaranteed to be an upper bound on $|Q_k - Q^*|$:

$$\left| \left(\Delta_k + \sum_{i=1}^{k} \gamma^{k-i} \alpha_i \right) - \left(\Delta'_k + \sum_{i=1} \gamma^{k-i} \alpha_i \right) \right| \leq 1. \tag{25}$$

Since, $\Delta'_k + \sum_i \gamma^{k-i} \alpha_i \geq |Q_k - Q^*|$, we get $\forall\ k \geq k_0$, using 25, that

$$\Delta_k + \sum_{i=1}^{k} \gamma^{k-i} \alpha_i \geq |Q_k - Q^*| - \gamma^{k-k_0}. \tag{26}$$

Hence, $\Delta_k + \sum_{i=1}^{k} \gamma^{k-i} \alpha_i \geq |Q_k - Q^*|$ for large k.

A note on the value of k_0. For a discounting of $\gamma = 0.95$, we get that $k_0 \approx 59$ and for $\gamma = 0.99$, $k_0 \approx 460$. In practical instances, an RL algorithm takes a minimum of about $\geq$ 1M gradient steps, so this value of k_0 is easy achieved. Even in the gridworld experiments presented in Section F.1, $\gamma = 0.95$, hence, the effects of initialization stayed significant only until about 59 iterations during training, out of a total of 300 or 500 performed, which is a small enough percentage.

Summary of Proof for Theorem 3.2. Δ_k in DisCor is given by the quantity $\Delta_k = |Q_k - \mathcal{B}^* Q_{k-1}| + \gamma P^{\pi_{k-1}} \Delta_{k-1}$, is an upper bound for the error $|Q_k - Q^*|$, and we can safely initialize the parametric function Δ_ϕ using standard neural network initialization, since the value of initial error will matter *only* infinitesimally after a large enough k.

As $k \to \infty$, the following is true:

$$\lim_{k \to \infty} \Big| |\Delta_k - |Q_k - Q^*| \Big| \leq \lim_{k \to \infty} \sum_{i=1}^{k} \gamma^{k-i} \alpha^i \tag{27}$$

$$= \lim_{k \to \infty} \sum_{i=1}^{k} \gamma^{k-i} \mathrm{D}_{\mathrm{TV}}(\pi_i, \pi^*) \tag{28}$$

Also, note that if π_k is improving, i.e. $\pi_k \to \pi^*$, then, we have that $\mathrm{D}_{\mathrm{TV}}(\pi_k, \pi^*) \to 0$, and since limit of a sum is equal to the sum of the limit, and $\gamma < 1$, therefore, the final inequality in Equation 28 tends to 0 as $k \to \infty$.

C. Tractable Approximations for a Practical Algorithm

We now discuss how to put together all tractable approximations discussed in Section 3 to go from the optimal distribution p_k (or w_k) to a tractable expression for weights, that downweight the states with high estimated target value error.

We have noted all practical approximations to the expression for optimal p_k (Equation 3), including estimating surrogates for Q_k and Q^*, and the usage of importance weights to develop a method that can achieve the benefits of the optimal distribution, simply by *re-weighting transitions* in the replay buffer, *rather than altering the exploration strategy*. We also discussed a technique to reduce the variance of weights used for this reweighting. We now put these techniques together to obtain the final, practically tractable expression for the weights used for our practical approach.

We note that the term $|Q_k - Q^*|$, appearing inside the exponent in the expression for w_k in Equation 5 can be approximated by the tractable upper bound Δ_k. However, computing Δ_k requires the quantity $|Q_k - \mathcal{B}^* Q_{k-1}|$ which also is unknown when w_k is being chosen. Combining the upper bound on $|Q_k - \mathcal{B}^* Q_{k-1}| \leq c_2$, Theorem 3.2 and Equation 4, we obtain the following bound:

$$|Q_k - Q^*| \leq \gamma P^{\pi_{k-1}} \Delta_{k-1} + c_2 + \sum_i \gamma^i \alpha_i \tag{29}$$

Using this bound in the expression for w_k, along with the lower bound, $|Q_k - \mathcal{B}^* Q_{k-1}| \geq c_1$, we obtain the following lower bound on weights w_k:

$$w_k \propto \exp\left(\frac{-c_2 - \gamma\left[P^{\pi_{k-1}}\Delta_{k-1}\right](s,a)}{\tau}\right)\frac{c_1}{\lambda^*} \tag{30}$$

Finally, we note that using a worst-case lower bound for w_k (Equation 30) will down-weight some additional transitions which in reality lead to low error accumulation, but this scheme will never up-weight a transition with high error, thus providing for a "conservative" distribution. A less conservative expression for getting these weights is a subject of future work. Simplifying the constants c_1, c_2 and λ^*, the final expression for the practical choice of w_k is:

$$w_k(s,a) \propto \exp\left(-\frac{\gamma\left[P^{\pi_{k-1}}\Delta_{k-1}\right](s,a)}{\tau}\right). \tag{31}$$

D. Proof From Section 2

In this section, we provide the omitted proof from Section 2 of this paper. Before going into the proofs, we first describe notation and prove some lemmas that will be useful later in the proofs.

We also describe the underlying ADP algorithm we use as an ideal algorithm for the proofs below.

Algorithm 2 Generic ADP algorithm

1: Initialize Q-values Q_0.
2: **for** step t in $\{1, \ldots, \text{N}\}$ **do**
3: Collect trajectories using π_t
4: Choose distribution D_t for projection.
5: $Q_{t+1} \leftarrow \prod_{D_t} \mathcal{B}^* Q_t$
$\prod_{D_t} \mathcal{B}^* = \arg\min_Q \mathbb{E}_{D_t}[(Q(s,a) - \mathcal{B}^* Q_{t-1}(s,a))^2]$
6: **end for**

Assumptions. The assumptions used in the proofs are as follows:

- Q-function is linearly represented, i.e. given a set of features, $\phi(s,a) \in \mathbb{R}^d$ for each state and action, concisely represented as the matrix $\Phi \in \mathbb{R}^{|S||A| \times d}$, Q-learning aims to learn a d-dimensional feature vector w, such that $Q(s,a) = w^T \phi(s,a)$. Linear function approximation is not a limiting factor in this case as we will argue in Assumption D.1, for problems with sufficiently large $|S|$ and $|A|$.

D.1. Suboptimal Convergence of On-policy Q-learning

We first discuss a prior result from Farias and Roy (2000) that describes how Q-learning can converge sub-optimally when performed with on-policy distributions, thus justifying our empirical observation of suboptimal convergence with on-policy distributions.

Theorem D.1 ((Farias and Roy, 2000))**.** *Projected Bellman optimality operator under the on-policy distribution $\mathcal{H} = \prod_{D_\pi} \mathcal{B}^*$ with a Boltzmann policy, $\pi \propto \exp(Q/\tau)$, where $0 < \tau$ always has one or more fixed points.*

Proof. This statement was proven to be true in (Farias and Roy, 2000), where it was shown the projection operator $\mathcal{H}$ has the same fixed points as another operator, F_α given by:

$$F_\alpha(x) := x + \alpha \Phi^T D_\pi \left(\mathcal{B}^* \Phi x - \Phi x\right) \tag{32}$$

where $\alpha \in (0,1)$ is a constant. They showed that the operator F_α is a contraction for small-enough α and used a compact set argument to generalize it to other positive values of α. We refer the reader to (Farias and Roy, 2000) for further reference.

They then showed a 2-state MDP example (Example 6.1, (Farias and Roy, 2000)) such that the Bellman operator $\mathcal{H}$ has **2** fixed points, thereby showing the existence of one or more fixed points for the on-policy Bellman backup operator.

This provides some theoretical evidence behind our observation in Figure 3(left), where we observed that learning to a suboptimal policy and the error plateaued, and this result provides some theoretical justification behind this empirical observation. □

D.2. Proof of Theorem 2.1

We now provide an existence proof which highlights the difference in the speeds of learning accurate Q-values from online or on-policy and replay buffer distributions versus a scheme like DisCor. We first state an assumption (Assumption D.1) on the linear features parameterization used for the Q-function. This assumption ensures that the optimal Q-function exists in the function class (i.e. linear function class) used to model the Q-function. This assumption has also been used in a number of recent works including (Du et al., 2020). Analogous to (Du et al., 2020), in our proof, we show that this assumption is indeed satisfied for features lying in a space that is logarithmic in the size of the state-space. For this theorem, we present an episodic example, and operate in a finite horizon setting with discounting γ and H denotes the horizon length. An episode terminates deterministically as soon as the run reaches a terminal node – in our case a leaf node of the tree MDP, i.e. a node at level $H-1$ – as we will see next.

Assumption D.1. *There exists $\delta \geq 0$, and $w \in \mathbb{R}^d$, such that for any $(s,a) \in \mathcal{S} \times \mathcal{A}$, the optimal Q-function satisfies: $|Q^*(s,a) - w^T\phi(s,a)| \leq \delta$.*

We first prove an intermediate property of Φ satisfying the above assumption that will be crucial for the lower bound argument for on-policy distributions.

Corollary D.1.1. *There exists a set of features $\Phi \in \mathbb{R}^{2^H \times O(H^2/\varepsilon^2)}$ satisfying assumption D.1, such that the following holds: $||I_{2^H} - \Phi\Phi^T||_\infty \leq \epsilon$.*

Proof. This proof builds on the existence argument presented in (Du et al., 2020). Using the ε-rank property of the identity matrix, one can show that there exists a feature set $\Phi \in \mathbb{R}^{2^H \times O(H/\varepsilon^2)}$ such that $||I_{2^H} - \Phi\Phi^T||_\infty \leq \epsilon$. Thus, we can choose any such Φ, for a sufficiently low threshold ε. In order to assign features Φ to a state, we can simply perform an enumeration of nodes in the tree via a standard graph search procedure such as depth first search and assign a node (s,a) a feature vector $\phi(s,a)$. To begin with, let's show how we can satisfy assumption D.1 by choosing a different weight vector w_h for each level h, such that we obtain $|Q_h(s,a) - w_h^T\phi(s,a)| \leq \epsilon$. Since for each level h exactly one state satisfies $Q^*(s_j,a_j) = \gamma^{H-j+1}$, so we can just let $w_j = \gamma^{H-j+1}\phi(s_j,a_j)$ and thus we are able to satisfy Assumption D.1. This is the extent of the argument used in (Du et al., 2020).

Now we generalize this argument to find a single $w \in \mathbb{R}^d$, unlike different weights w_h for different levels h. In order to do this, we create a new Φ', of size $\Phi' \in \mathbb{R}^{2^H \times O(H^2/\epsilon^2)}$ (note H^2 versus H dimensions for Φ' and Φ) given any Φ satisfying the argument in the above paragraph, such that

$$\Phi'(s,a) = \left[\underbrace{0,...,0}_{h \times \dim(\phi(s,a))}, \underbrace{\Phi(s,a)}_{\dim(\phi(s,a))}, 0,...,0 \right] \tag{33}$$

Essentially, we pad Φ with zeros, such that for (s,a) belonging to a level h, Φ' is equal to Φ in the $h-$th, $\dim(\phi(s,a))$-sized block.

A choice of a single $w \in \mathbb{R}^{\dim(\Phi'(s,a))}$ for Φ' is given by simply concatenating $w_1, \cdots, w_h$ found earlier for Φ.

$$w = [w_1, w_2, \cdots, w_H] \tag{34}$$

It is easy to see that $w^T\Phi'$ satisfies assumption D.1. A fact that will be used in the proof for Theorem D.2, is that this construction of Φ' also satisfies: $||I_{2^H} - \Phi'\Phi'^T||_\infty \leq \epsilon$. □

We now restate the theorem from Section 2 and provide a proof below.

Theorem D.2 (Exponential lower bound for on-policy distributions)**.** *There exists a family of MDPs parameterized by $H > 0$, with $|\mathcal{S}| = 2^H$, $|\mathcal{A}| = 2$ and a set of features satisfying Assumption D.1, such that on-policy sampling distribution, i.e. $D_k = d^{\pi_k}$, requires $\Omega\left(\gamma^{-H}\right)$ exact fixed-point iteration steps in the generic algorithm (Algorithm 2) for convergence, if at all, the algorithm converges to an $\varepsilon-$accurate Q-function.*

Proof of Theorem D.2. *Tree Construction.* Consider the family of tree MDPs like the one shown in Figure 2. Both the transition function T and the reward function r are deterministic, and there are two actions at each state: a_1 and a_2. There are H level of states, thereby forming a full binary tree of depth H. Executing action a_1 transitions the state to its left child

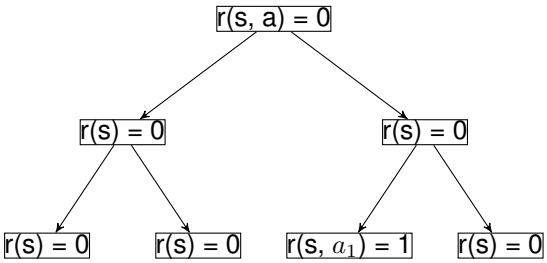


Figure 2. Example element of the tree family of MDPs used to prove the lower bound in Theorem D.2. Here, the depth of the tree $H = 2$. $r(s) = 0$ implies that executing any action a_1 or a_2, a reward of 0 is obtained as state s. (s^*, a^*) is given by state marked r(s, a_1) = 1.

int he tree and executing action a_2 transitions the state to its right child. There are 2^h states in level h. Among the 2^{H-1} states in level $H-1$, there is one state, s^*, such that action a^* at this state yields a reward of $r(s^*, a^*) = 1$. For other states of the MDP, $r(s, a) = 0$. This is a typical example of a sparse reward problem, generally used for studying exploration (Du et al., 2020), however, we re-iterate that in this case, we are primarily interested in the number of iterations needed to learn, and thereby assume that the algorithm is given infinite access to the MDP, and all transitions are observed, and the algorithm just picks a distribution D_k, in this case, the on-policy state-action marginal for performing backups.

Main Argument. Now, we are equipped with a family of the described tree MDPs and a corresponding set of features Φ which can represent an $\varepsilon-$accurate Q-function. Our aim is to show that on-policy Q-learning takes steps, exponential in the horizon for solving this task.

For any stochastic policy $\pi(a|s)$, and $\bar{p}$ defined as $\bar{p} = \min_{s \in \mathcal{S}, a \in \mathcal{A}} \pi(a|s)$, $0 < \bar{p} < 0.5$, the marginal state-action distribution satisfies:

$$d^{\pi}(s^*, a^*) \leq \gamma^H \cdot (1 - \bar{p})^{H+1} \tag{35}$$

Since d^{π} is a discounted state-action marginal distribution, another property that it satisfies is that:

$$c \leq ||d^{\pi}||_2 \leq \frac{1}{1-\gamma} 2^H \tag{36}$$

where c is a constant $c > 0$. The above is true, since, there are 2^H states in this MDP, and the maximum values of any entry in d^{π} can be $\frac{1}{1-\gamma}$ since, $1-\gamma$ is the least eigenvalue of $(I - \gamma P^{\pi})$ for any policy π, since $||P^{\pi}||_2 = 1$.

Now, under an on-policy sampling scheme and a linear representation of the Q-function as assumed, the updates on the weights for each iteration of Algorithm 2 are given by (D_{π_k} represents $\text{Diag}(d^{\pi_k})$):

$$w_{k+1} = \left(\Phi^T D_{\pi_k} \Phi\right)^{-1} \Phi^T D_{\pi_k} \left(r + \gamma P^{\pi_k} \Phi w_k\right) \tag{37}$$

Now, $||D_{\pi_k} r|| \leq \gamma^H (1-\bar{p})^{H+1} ||\phi(s^*, a^*)||$ from the property Equation 35. Hence, the maximum 2-norm of the updated w_{k+1} is given by:

$$\begin{aligned} ||w_{k+1}||_2 \leq & \; || \left(\Phi^T D_{\pi_k} \Phi\right)^{-1} \Phi^T D_{\pi_k} R||_2 + \gamma || \left(\Phi^T D_{\pi_k} \Phi\right)^{-1} \Phi^T D_{\pi_k} P^{\pi_k} \Phi w_k||_2 \\ \leq & \; \frac{\gamma^H (1-\bar{p})^{H+1}}{||D_{\pi_k}||_F \cdot (1-\varepsilon) \cdot 2^{H-1}} + \gamma ||w_k||_2 \\ \leq & \; \frac{\gamma^H (1-\bar{p})^{H+1} c}{(1-\varepsilon) \cdot 2^{H-1}} + ||w_k||_2 \\ = & \; (\gamma)^H \cdot c \cdot \frac{(1)}{(1-\varepsilon) \cdot 2^{H-1}} + ||w_k||_2. \end{aligned} \tag{38}$$

where the first inequality follows by an application of the triangle inequality, the second inequality follows by using the minimum value of the Frobenius norm of the matrix Φ to be $(1-\varepsilon) \cdot 2^{H-1}$ (using the $\varepsilon-$rank lemma used to satisfy Assumption D.1) in the denominator of the first term, bounding $||D_{\pi_k} r||$ by Equation 35, and finally bounding the second term by $\gamma ||w_k||_2$, since the maximum eigenvalue of the entire matrix in front of w_k is ≤ 1, as it is a projection matrix with a discount γ valued scalar multiplier. The third inequality follows from lower bounding D_{π_k} by c using Equation 36.

The optimal w^* is given by the fixed point of the Bellman optimality operator, and in this case satisfies the following via Cauchy-Schwartz inequality,

$$\begin{aligned} & (I - \gamma P^*)\Phi w^* = r \\ \implies & ||\Phi||_F \cdot ||w^*||_2 \geq ||(I - \gamma P^*)^{-1} r|| \geq \frac{1}{1+\gamma}||r||_2 \\ \implies & (1+\varepsilon) \cdot 2^{H-1} \cdot ||w^*||_2 \geq \frac{1}{1+\gamma} \\ \implies & ||w^*||_2 \geq \frac{1}{1+\gamma} \cdot 2^{-H+1} \cdot (1+\varepsilon)^{-1} \end{aligned} \tag{39}$$

Thus, in order for w_k to be equal to w^*, it must satisfy the above condition (Equation 39). If we choose an initialization $w_0 = \mathbf{0}$ (or a vector sufficiently close to 0), we can compute the minimum number of steps it will take for on-policy ADP to converge in this setting by using 38 and 39:

$$\begin{aligned} k &\geq \frac{(1+\gamma)^{-1} \cdot 2^{-H+1} \cdot (1+\varepsilon)^{-1}}{(\gamma)^H \cdot (1-\bar{p})^H \cdot \frac{(c)}{(1-\varepsilon)\cdot 2^{H-1}}} \\ \implies k &\approx \Omega\left(\gamma^{-H}\right) \end{aligned} \tag{40}$$

for sufficiently small ε. Hence, the bound follows.

A note on the bound. Since typically RL problems usually assume discount factors γ close to 1, one might wonder the relevance is this bound in practice. We show via an example that this is indeed relevant. In particular, we compute the value of this bound for commonly used $\gamma, \bar{p}$ and H. For a discount $\gamma = 0.99$, and a minimum probability of $\bar{p} = 0.01$ (as it is common to use entropy bonuses that induce a minimum probability of taking each action), this bound is of the order of

$$(\gamma \cdot (1-\bar{p}))^H \approx 10^9 \quad \text{for H = 1000} \tag{41}$$

for commonly used horizon lengths of 1000 (example, on the gym benchmarks).

Corollary D.2.1 (Extension to replay buffers)**.** *There exists a family of MDPs parameterized by $H > 0$, with $|\mathcal{S}| = 2^H$, $|\mathcal{A}| = 2$ and a set of features Φ satisfying assumption D.1, such that ADP with replay buffer distribution takes $\Omega(\gamma^{-H})$ many steps of exact fixed-point iteration for convergence of ADP, if at all convergence happens to an $\epsilon-$accurate Q-function.*

Proof of Corollary D.2.1. For replay buffers, we can prove a similar statement as previously. The steps in this proof follow exactly the steps in the proof for the previous theorem.

With replay buffers, the distribution for the projection at iteration k is given by:

$$d_k(s,a) = \frac{1}{k}\sum_{i=1}^{k} d_{\pi_k}(s,a) \tag{42}$$

Therefore, we can bound the probability of observing any state-action pair similar to Equation 35 as:

$$d_k(s^*,a^*) \leq \frac{1}{k}\sum_{i=1}^{k} \gamma^H \cdot (1-\bar{p})^{H+1} \tag{43}$$

with $\bar{p}$ as defined previously. Note that this inequality is the same as the previous proof, and doesn't change. We next bound the 2-norm of the state-visitation distribution, in this case, the state-distribution in the buffer.

$$c \leq ||d_k||_2 \leq \frac{1}{1-\gamma} \cdot 2^H \tag{44}$$

where $c > 0$. The two main inequalities used are thus the same as the previous proof. Now, we can simply follow the previous proof to prove the result.

Practical Implications. In this example, both on-policy and replay buffer Q-learning suffer from the problem of exponentially many samples need to reach the optimal Q-function. Even in our experiments in Section 2, we find that on-policy distributions tend to reduce errors very slowly, at a rate that is very small. The above bound extends this result to replay buffers as well.

In our next result, however, we show that an optimal choice of distribution, including DisCor, can avoid the large iteration complexity in this family of MDPs. Specifically, using the errors against Q^*, i.e. $|Q_k - Q^*|$ can help provide a signal to improve the Q-function such that this optimal distribution / DisCor will take only $\text{poly}(H)$ many iterations for convergence.

Theorem D.3 (Optimal distributions / DisCor)**.** *In the tree MDP family considered in Theorem 2.1, with linear function approximation for the Q-function, and with Assumption D.1 for the features Φ, DisCor takes $\text{poly}(H)$ many exact iterations for $\varepsilon-$accurate convergence to the optimal Q-function.*

Proof. We finally show that the DisCor algorithm, which prioritizes states based on the error in target values, will take $\text{poly}(H)$ many steps for convergence. Assume that Q-values are initialized randomly, for example via a normal random variable with standard deviation σ, i.e., $Q_0(s,a) \sim \mathcal{N}(0, \sigma^2)$, however, σ is very small, but is more than 0 ($\sigma > 0$) (this proof is still comparable to the proof for on-policy distributions, since Q-values can also be initialized very close to 0 even in that case, and the proof of Theorem D.2 still remains valid.).

Now we reason about a run of DisCor in this case.

Iteration 1. In the first iteration, among all nodes in the MDP, the leaf nodes (depth H-1) have 0 error at the corresponding target values, since an episode terminates once a rollout reaches a leaf node. Hence, the algorithm will assign equal mass to all leaf node states, and exactly update the Q-values for nodes in this level (upto ε-accuracy).

Iteration 2. In the second iteration, the leaf nodes at level $H-1$ have accurate Q-values, therefore, the algorithm will pick nodes at the level $H-2$, for which the target values, i.e. Q-values for nodes at level $H-1$, have 0 error. The algorithm will update Q-values at these nodes at level $H-2$, while ensuring that the incurred error at the nodes at level $H-1$ isn't beyond ε. Since, the optimal value function Q^* can be represented upto $\varepsilon-$accuracy, we can satisfy this criterion.

Iteration k. In iteration k, the algorithm updates Q-values for nodes at level $H-k$, while also ensuring Q-values for all nodes at a level higher than $H-k$ are estimated within the range of $\varepsilon-$allowable error. This is feasible since, Q^* is expressible with $\varepsilon-$accuracy within the linear function class chosen.

This iteration process continues, and progress level by level, from the leaves (level $H-1$) to the root (level 0). At each iteration Q-values for all states at the same level, and below are learned together. Since learning progresses in a "one level at-a-time" fashion, with guaranteed correct target values (i.e. target values are equal to the optimal Q-function Q^*) for any update that the algorithm performs, it would take at most $\text{poly}(H)$ many iterations (for example, multiple passes through the depth of the tree) for ε-accurate convergence to the optimal Q-function. □

E. Consequences of the Inability to Correct Value Errors

In Figure 3, we plot value error $\mathcal{E}_k$ over the course of Q-learning with on-policy and replay buffer distributions. The plots show prolonged periods where $\mathcal{E}_k$ is increasing or fluctuating. When this happens, the policy has poor performance, with returns that are unstable or stagnating (Fig. 3). To study the effects of function approximation and distributions on this issue, we can control for both of these factors. When a uniform distribution $\text{Unif}(s,a)$ is used instead of the on-policy distribution, as shown in Fig. 3 (red), or when using a tabular representation without function approximation, but with the on-policy distribution, as shown in with Fig. 3 (brown), we see that $\mathcal{E}_k$ decreases smoothly, suggesting that the combination of function approximation and naïve distributions can result in challenges in value error reduction.

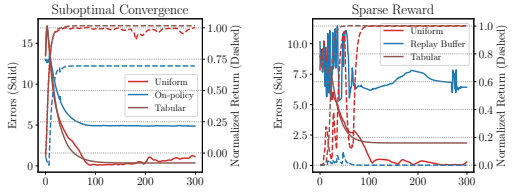


Figure 3. Value error ($\mathcal{E}_k$) and policy performance (normalized return) for **Left**: sub-optimal convergence with on-policy distributions, **Right**: instabilities in learning progress with replay buffers. Note that an oracle re-weighting to a uniform data distribution or complete removal of function approximation, gives rise to decreasing $\mathcal{E}_k$ curve and better policy performance.

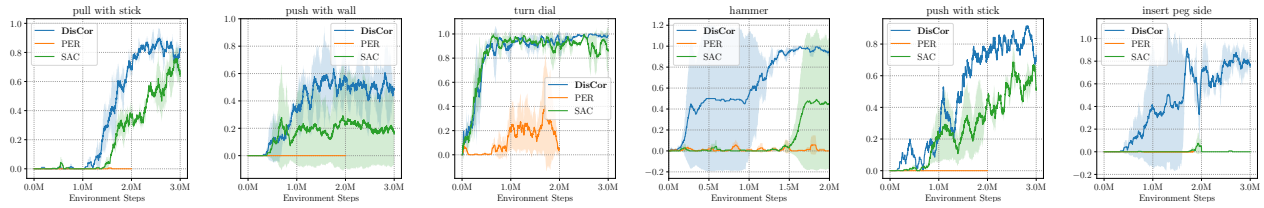


Figure 6. Evaluation success of DisCor, unweighted SAC and PER on six MetaWorld tasks. From left to right: pull stick, push with wall, push stick, turn dial, hammer and insert peg side. Note that DisCor achieves better final success rates or learns faster on most of the tasks and is the only method that learns on one task.

F. Expanded Experimental Section

F.1. Analysis of DisCor on Tabular Environments

We first use the tabular domains from Section 2, described in detail in Appendix I.1, to analyze error correction induced by DisCor and evaluate the effect of the approximations used in our method, such as the upper bound estimator Δ_k, both when the algorithm is provided with Fig. 10). all transitions in the replay buffer and simply chooses a weighting on them (no sampling error) and when the algorithm collects its own transitions via exploration. In both settings, in Figure 4, value error $\mathcal{E}_k$ decreases smoothly with DisCor. An oracle version of the algorithm (DisCor (oracle); Equation 5), which uses the true error $|Q_k - Q^*|$ in place of Δ_k, is somewhat better than DisCor (Fig. 5, red vs blue), but DisCor still outperforms on-policy and replay buffer schemes (green and pink), which often fail to reduce $\mathcal{E}_k$ as shown in Section 2. While DisCor (oracle) consistently performs better than DisCor, as we would expect, the approximate DisCor algorithm still attains better performance than naïve uniform weighting and prioritization similar to PER. This shows that the principle behind DisCor is effective when applied exactly, and that even the approximation that we use in practice improves performance.

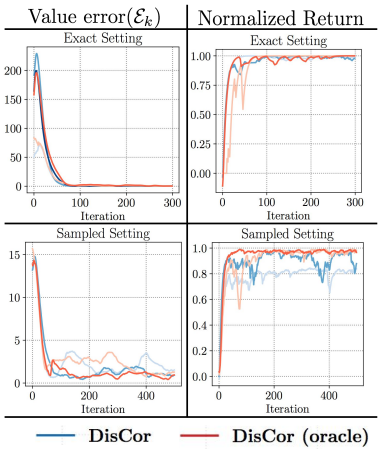


Figure 4. Value Error $\mathcal{E}_k$/ return for *two* runs of DisCor (blue) and DisCor (oracle) (red) in exact (top) and sampled (bottom) settings. Note (i) DisCor achieves similar performance as DisCor (oracle), (ii) $\mathcal{E}_k$ generally decreases with both methods.

F.2. Continuous Control Experiments

We next perform a comparative evaluation of DisCor on several continuous control tasks, using six robotic manipulation tasks from the Meta-World suite (pull stick, hammer, insert peg side, push stick, push with wall and turn dial) (these are shown in Figure 13 in Appendix G). These domains were chosen because they are challenging for state-of-the-art RL methods, such as SAC (Haarnoja et al., 2018). We applied DisCor to these tasks by modifying the weighting of samples in SAC. DisCor does not alter any hyperparameter from SAC, and requires minimal tuning. There is only one additional temperature hyperparameter, which is also automatically chosen. More details are presented in App. H.2.

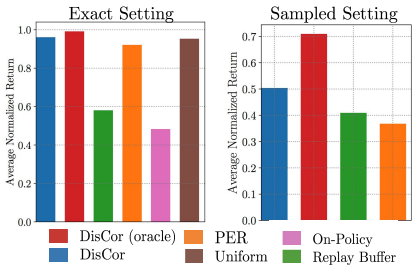


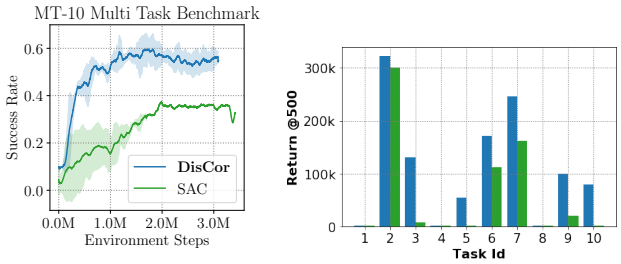
Figure 5. Performance of **DisCor**, **DisCor (oracle)** and other distributions averaged across tabular domains with and without sampling error.. DisCor is generally comparable to DisCor (oracle), and both of them generally outperform all other distributions.

We compare DisCor to standard SAC without weighting, as well as prioritized experience replay (PER) (Schaul et al., 2015), which uses weights based on the last Bellman error. The results in Figure 6 show that DisCor outperforms prior methods on these tasks. DisCor learns substantially faster on most of the tasks.

We also performed comparisons on the more conventional gym benchmarks, where we see a small but consistent benefit from DisCor reweighting. Since prior methods, such as SAC already solves these tasks easily, and have been tuned well for them, the room for improvement is very small. We include these results in Appendix I.3 for completeness. We also evaluate on a stochastic reward variant of gym benchmarks, where we observe an improvement trend. However, on tasks that have not been tuned as extensively or exhibit challenging properties, such as multi-task learning or complex manipulation tasks, current RL methods can perform poorly.

F.3. Multi-Task Reinforcement Learning

Another challenging setting for current RL methods is the multi-task RL setting. This is known to be difficult, to the point that often times learning completely separate policies for each of the tasks is actually faster, and results in better performance, than learning the tasks together (Yu et al., 2019; Schaul et al., 2019). We evaluate on the MT10 MetaWorld benchmark (Yu et al., 2019), which consists of ten robotic manipulation tasks to be learned jointly. We follow the protocol from (Yu et al., 2019), and append task ID to the state. As shown in Figure 7(a), DisCor outperforms SAC by a large margin, achieving **50%** higher success rates compared to SAC, and a high overall return (Fig 16). Figure 7(b) shows that DisCor makes progress on **7/10** tasks, as compared to **3/10** for SAC. We further evaluate DisCor and SAC on the more challenging MT50 benchmark (Yu et al., 2019), shown in Figure 17, and observe a similar benefit as compared to MT10, where the baseline SAC algorithms tends to plateau at a suboptimal success rate for about 4M environment steps, whereas DisCor keeps learning, and achieves asymptotic performance faster.



(a) Success rate (b) Per-task return

Figure 7. Performance of DisCor (blue) and unweighted SAC (green) on the MT10 benchmark. We observe that: (1) DisCor outperforms unweighted SAC by a factor of **1.5** in terms success rate; (2) DisCor achieves a non-trivial return on **7/10** tasks after 500k environment steps, as compared to **3/10** for unweighted SAC.

F.4. Arcade Learning Environment

Our final experiments were aimed at testing the efficacy of DisCor on stochastic, discrete-action, image-observation environments. To this end, we evaluated DisCor on three commonly reported tasks from the Atari suite – Pong, Breakout and Asterix. We compare to the baseline DQN (Mnih et al., 2015), all our implementations are built off of Dopamine (Castro et al., 2018), and use the evaluation protocol with sticky actions (Machado et al., 2018). We build DisCor on top of DQN by simply replacing the standard replay buffer sampling scheme in DQN with the DisCor weighted update. We show in Figure 8 that DisCor usually outperforms unweighted DQN in learning speed and performance.

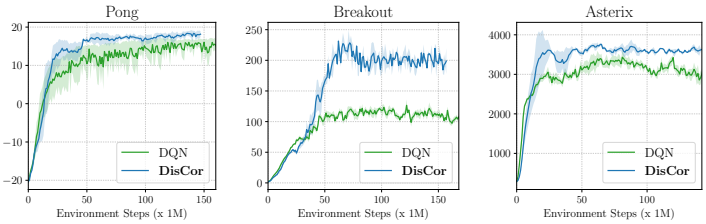


Figure 8. DQN vs DisCor on Atari. Note that DisCor generally improves learning speed and asymptotic performance.

G. Related Work

G.1. Summary of Related Work

Prior work has pointed out a number of issues arising when dynamic programming is used with function approximation. (Munos, 2005; Munos and Szepesvári, 2008; Farahmand et al., 2010; Scherrer et al., 2015; Lesner and Scherrer, 2013; Scherrer, 2014) focused on analysing error induced in Bellman projections, under the assumption of an abstract error model. Convergent backups (Sutton et al., 2009b;a; Maei et al., 2009) were developed. However, divergence is rarely observed to be an issue with deep Q-learning methods (Fu et al., 2019; van Hasselt et al., 2018). In contrast to these works, which mostly focus on convergence of the Bellman backup, we focus on the interaction between the ADP update and the data distribution μ. Prior work on Q-learning and stochastic approximation analyzes time-varying μ, but either without function approximation (Watkins and Dayan, 1992; Tsitsiklis, 1994; Devraj and Meyn, 2017), or when fully online (Tsitsiklis and Roy, 1997), unlike our setting, that uses replay buffer data.

While generalization effects of deep neural nets with ADP updates have been studied (Achiam et al., 2019; Fu et al., 2019; Liu et al., 2018; Kumar et al., 2019), often under standard NTK (Jacot et al., 2018) assumptions (Achiam et al., 2019), the high-level idea in these prior works has been to suppress any coupling effects of the function approximator, effectively obtaining tabular behavior. In contrast, DisCor solves an optimization problem for the distribution p_k that maximally reduces value error, and does not explicitly suppress coupling effects, as these can be important for generalization in high dimensions. (Schaul et al., 2019) studies the effect of data distribution on multi-objective policy gradient methods and reports a pathological interaction between the data distribution and optimization. (Farias and Roy, 2000) shows the existence

of suboptimal fixed points with on-policy TD learning as we observed empirically in Figure 3 (left). DisCor re-weights the transition in the buffer based on an estimate of their error to the true optimal value function. This scheme resembles learning with noisy labels via "abstention" from training on labels that are likely to be inaccurate (Thulasidasan et al., 2019). Prioritized sampling has been used previously in ADP methods to instead prioritize transitions with higher Bellman error (Schaul et al., 2015; Hessel et al., 2018; Hou et al., 2017; Horgan et al., 2018). We show in Section 4 that this approach is less effective than DisCor experimentally. Recent work (Du et al., 2019) has attempted to use a distribution-checking oracle to control the amount of exploration performed. DisCor, instead, re-weights the data distribution without requiring any oracles.

G.2. Expanded Related Work

Error propagation in ADP. A number of prior works have analysed error propagation in ADP methods. Most work in this area has been devoted to analysing how errors in Bellman error minimization propagate through the learning process of the ADP algorithm, typically focusing on methods such as fitted Q-iteration (FQI) (Riedmiller, 2005) or approximate policy iteration (Perkins and Precup, 2002). Prior works in this area assume an abstract error model, and analyze how errors propagate. Typically these prior works only limitedly explore reasons for error propagation or present methods to curb error propagation. (Munos, 2003) analyze error propagation in approximate policy iteration methods using quadratic norms. (Munos, 2005) analyze the propagation of error across iterations of approximate value iteration (AVI) for L_p-norm $p = (1, 2)$. (Munos and Szepesvári, 2008) provide finite sample guarantees of AVI using error propagation analysis. Similar ideas have been used to provide error bounds for a number of different methods – (Farahmand et al., 2010; Scherrer et al., 2015; Lesner and Scherrer, 2013; Scherrer, 2014) and many more. In this work, we show that ADP algorithms suffer from an absence of corrective feedback, which arises because the data distribution collected by an agent is insufficient to ensure that error propagation is eventually corrected for. We further propose an approach, DisCor, which can be used in conjunction with modern deep RL methods.

Offline / Batch Reinforcement Learning. Our work bears some similarity to the recent body of literature on batch, or offline reinforcement learning (Kumar et al., 2019; Fujimoto et al., 2019; Wu et al., 2019), where the goal is to learn an effective policy, using access to only a finite, off-policy dataset collected previously. All of these works augment ADP methods with additional constraints on the policy to be close to the data-collection policy, under some closeness metric. While (Kumar et al., 2019) show that this choice can be motivated from the perspective of error propagation, we note that there are clear differences between our work and such prior works in offline RL. First, the problem statement of offline RL requires learning from completely offline experience, however, our method learns online, via on-policy interaction and a replay buffer. While error propagation due to unobserved state-action pairs (Kumar et al., 2019; Fujimoto et al., 2019) is the primary problem behind incorrect Q-functions in offline RL, in this paper, firstly, we show that such error accumulation also happens in online reinforcement learning, which results in a lack of corrective feedback, and secondly, the primary reason behind such error propagation is an interaction between data distribution and function approximation.

Generalization effects in deep Q-learning. There are a number of recent works that theoretically analyze and empirically demonstrate that certain design decisions for neural net architectures used for Q-learning, or ADP objectives can prove to be significant in deep Q-learning. For instance, (Liu et al., 2018) point out that sparse representations may help Q-learning algorithms, which links back to prior literature on state-aliasing and destructive interference. (Achiam et al., 2019) uses an objective inspired from the neural tangent kernel (NTK) (Jacot et al., 2018) to "cancel" generalization effects in the Q-function induced across state-action pairs to mimic tabular and online Q-learning. Our approach, DisCor, can be interpreted as *only* indirectly affecting generalization via the target Q-values for state-action pairs that will be used as bootstrap targets for the Bellman backup, which are expected to be accurate with DisCor, and this can aid generalization, similar to how generalization can be achieved via abstention from training on noisy labels in supervised learning (Thulasidasan et al., 2019).

H. Experimental and Implementation Details

In this section, we provide experimental details, such as the DisCor algorithm in practice (Section H.1), and the hyperparameter choices (Section H.2).

Algorithm 3 DisCor: Deep RL Version

1: Initialize online Q-network $Q_\theta(s,a)$, target Q-network, $Q_{\bar{\theta}}(s,a)$, error network $\Delta_\phi(s,a)$, target error network $\Delta_{\bar{\phi}}$, initial distribution $p_0(s,a)$, a replay buffer β and a policy $\pi_\psi(a|s)$, number of gradient steps G, target network update rate η, initial temperature for computing weights w_k, τ_0.
2: **for** step k in $\{1, \ldots, \}$ **do**
3: Collect M samples using $\pi_\psi(a|s)$, add them to replay buffer β, sample $\{(s_i, a_i)\}_{i=1}^N \sim \beta$
4: Evaluate $Q_\theta(s,a)$ and $\Delta_\phi(s,a)$ on samples (s_i, a_i).
5: Compute target values for Q and Δ on samples:

$$y_i = r_i + \gamma \mathbb{E}_{a' \sim \pi_\psi(a'|s')}[Q_{\bar{\theta}}(s'_i, a')]$$

$$\hat{\Delta}_i = |Q_\theta(s,a) - y_i| + \gamma \mathbb{E}_{\hat{a}_i \sim \pi(a_i|s')}[\Delta_{\bar{\phi}}(s'_i, \hat{a}_i)]$$

6: Compute w_k using Equation 6 with temperature τ_k
7: Take G gradient steps on the Bellman error for training Q_θ weighted by w_k.

$$\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{N} \sum_{i=1}^{N} w_k(s_i, a_i) \cdot (Q_\theta(s_i, a_i) - y_i)^2$$

8: Tale G gradient steps to minimize unweighted (regular) Bellman error for training ϕ.

$$\phi \leftarrow \phi - \alpha \nabla_\phi \frac{1}{N} \sum_{i=1}^{N} (\Delta_\theta(s_i, a_i) - \hat{\Delta}_i)^2$$

9: Update the policy π_ψ if it is explicitly modeled.

$$\psi \leftarrow \psi + \alpha \nabla_\psi \mathbb{E}_{s \sim \beta, a \sim \pi_\psi(a|s)}[Q_\theta(s,a)]$$

10: Update target networks using soft updates (SAC), hard updates (DQN)

$$\bar{\theta} \leftarrow (1-\eta)\bar{\theta} + \eta\theta$$

$$\bar{\phi} \leftarrow (1-\eta)\bar{\phi} + \eta\phi$$

11: Update temperature hyperparameter for DisCor:

$$\tau_{k+1} \leftarrow (1-\eta)\tau_k + \eta \ \text{BATCH-MEAN}(\Delta_\phi(s_i, a_i))$$

12: **end for**

H.1. DisCor in Practice

In this section, we provide details on the experimental setup and present the pseudo-code for the practical instantiation of our algorithm, DisCor.

The pseudocode for the practical algorithm is provided in Algorithm 3. Like any other ADP algorithm, such as DQN or SAC, our algorithm maintains a pair of Q-functions – the online Q-network Q_θ and a target network $Q_{\bar{\theta}}$. For continuous control domains, we use the clipped double Q-learning trick (Fujimoto et al., 2018), which is also referred to as the "twin-Q" trick, and it further parametrizes another pair of online and target Q-functions, and uses the minimum Q-value for backup computation. In addition to Q-functions, in a continuous control domain, we parametrize a separate policy network π_ψ similar to SAC. In a discrete action domain, the policy is just given by a greedy maximization of the online Q-network.

DisCor further maintains a model for accumulating errors Δ_ϕ parameterized by ϕ and the corresponding target error network $\Delta_{\bar{\phi}}$. In the setting with two Q-functions, DisCor models two networks, one for modelling error in each Q-function. At every step, a few (depending upon the algorithm) gradient steps are performed on Q and Δ, and π – if it is explicitly modeled, for instance in continuous control domains. This is a modification of generalized ADP Algorithm 2 and the corresponding DisCor version (Algorithm 1), customized to modern deep RL methods.

H.2. Experimental Hyperparameter Choices

We finally specify the hyperparameters we used for our experiments. These are as follows:

- *Temperature τ:* DisCor mainly introduces one hyperparameter, the temperature τ used to compute the weights w_k in Equation 6. As shown in Line 11 of Algorithm 3, DisCor maintains a moving average of the temperatures and uses this average to perform the weighting. This removes the requirement for tuning the temperature values at all. For initialization, we chose $\tau_0 = 10.0$ for all our experiments, irrespective of the domain or task.
  In our preliminary experiments, we tried experimenting with a fixed temperature, which did not yield good results, and suffered from either too large importance weights, giving rise to high variance in the learning curve or wasn't effective beyond SAC at all.
- *Architecture for Δ_ϕ:* For the design of the error network, Δ_ϕ, we utilize a network with 1 extra hidden layer than the corresponding Q-network. For instance, in metaworld domains, the standard Q-network used was [256, 256, 256] in size, and thus we used an error network of size: [256, 256, 256, 256], and for MT10 tasks we used [160, 160, 160, 160, 160, 160] sized Q-networks (Yu et al., 2020) and 1-extra layer error networks Δ_ϕ.
- *Target net updates:* We performed target net updates for $\Delta_{\bar{\phi}}$ in the same manner as standard Q-functions, in all domains. For instance, in MetaWorld, we update the target network $\Delta_{\bar{\phi}}$ with a soft update rate of 0.005 at each environment step, as is standard with SAC (Tuomas Haarnoja and Levine, 2018), whereas in DQN (Mnih et al., 2015), we use hard target resets.
- *Learning rates for Δ_ϕ:* These were chosen to be the same as the corresponding learning rate for the Q-function, which is $3e-4$ for SAC and 0.0025 for DQN. We also searched over the space of three learning rates: $[3e-4, 1e-4, 5e-4]$, and did not find a huge difference across these. The only somewhat visible difference suggested that a learning rate of $3e-4$ or $5e-4$ worked better than 1e-4..
- *Official Implementation repositories used for our work:*
  1. Soft-Actor-Critic (Haarnoja et al., 2018): `https://github.com/rail-berkeley/softlearning/`
  2. Dopamine (Castro et al., 2018): Offical DQN implementation `https://github.com/google/dopamine`, and the baseline DQN numbers were reported from the logs available at: `https://github.com/google/dopamine/tree/master/baselines`
  3. Gridworlds (Fu et al., 2019): `https://github.com/justinjfu/diagnosing_qlearning`
- We perform self-normalized importance sampling across a batch, instead of regular importance sampling, since that gives rise to more stable training, and suffers less from the curse of variance in importance sampling.
- *Seeds*: In all our experiments, we implemented our methods on top of the official repositories, ran each experiment for 4 randomly chosen seeds from the interval, $[10, 10000]$, in Meta-World, OpenAI gym and tabular environments. For DQNs on atari, we were only able to run 3 seeds for each game for our method, however, we found similar performances, and less variance across seeds, as is evident from the variance bands in the corresponding results. For baseline DQN, we just used the log files provided by the dopamine repository for our results.

I. Additional Experiments

We now present some additional experimental results which we could not present de to shortage of space in Section 4.

I.1. Tabular Environment Analysis

Environment Setup. We used the suite of tabular environments from from Fu et al. (2019), which provides a suite of 8 tabular environments and several different plug-and-play options for choices of input feature space, reward functions, etc. a suite of algorithms based on fitted Q-iteration (Riedmiller, 2005), which forms the basis of modern deep RL algorithms that use ADP. We evaluated performance on different variants of the $(16, 16)$ gridworld provided, with different reward styles (sparse, dense), different observation functions (one-hot, random features, locally smooth observations), and different amounts of entropy coefficients (0.01, 0.1). We evaluated on five different kinds of environments: grid16randomobs, grid16onehot, grid16smoothobs, grid16smoothsparse, grid16randomsparse – which cover a wide variety of combinations of feature and reward types. We also evaluated on CliffWalk, Sparsegraph and MountainCar MDPs in Figures 9 and 10.

Sampling Modes. We evaluated in two modes – (1) exact mode, in the absence of sampling error, where an algorithm is provided with all transitions in the MDP and simply chooses a weighting over the states rather than sampling transitions from the environment, and (2) sampled mode, which is the conventional RL paradigm, where the algorithm performs online

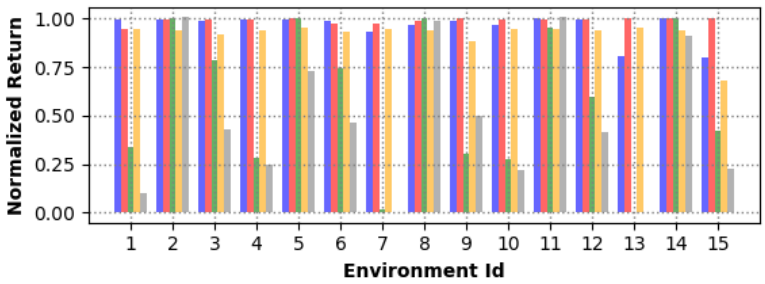


Figure 9. Performance of different methods: DisCor (blue), DisCor (oracle) (red), Replay buffer Q-learning (green, on-policy (grey) and prioritized updates (orange), across different environments measured in terms of smooth normalized returns in the **exact** setting with all transitions. Note that DisCor and DisCor (oracle) generally tend to perform better.

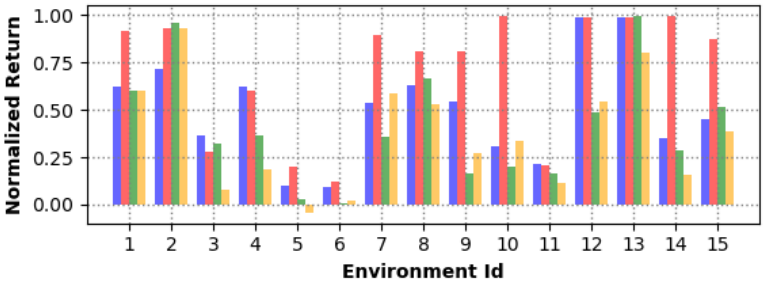


Figure 10. Performance of different methods: DisCor (blue), DisCor (oracle) (red), Replay buffer Q-learning (green) and prioritized updates (orange). across different environments measured in terms of smooth normalized return with **sampled** transitions. Note that DisCor and DisCor (oracle) generally tend to perform better.

data collection to collect its own data.

Setup for Figure 3. For Figure 3, we used the grid16randomobs MDP (which is a 16×16 gridworld with randomly initialized vectors as observations), with an entropy penalty of 0.01 to the policy.

Results. We provide some individual environment performance curves showing the smoothed normalized return achieved at the end of 300 steps of training in both exact (Figure 9) and sampled (Figure 10) settings. We also present some individual-environment learning curves for these environments comparing different methods in both exact (Figure 11) and sampled (Figure 12).

I.2. MetaWorld Tasks

In this section, we first provide a pictorial description of the six hard tasks we tested on from meta-world, where SAC usually does not perform very well. Figure 13 shows these tasks. We provide the trends for average return achieved during evaluation (not the success rate as shown in Figure 6 in Section 4) for each of the six tasks. Note that DisCor clearly

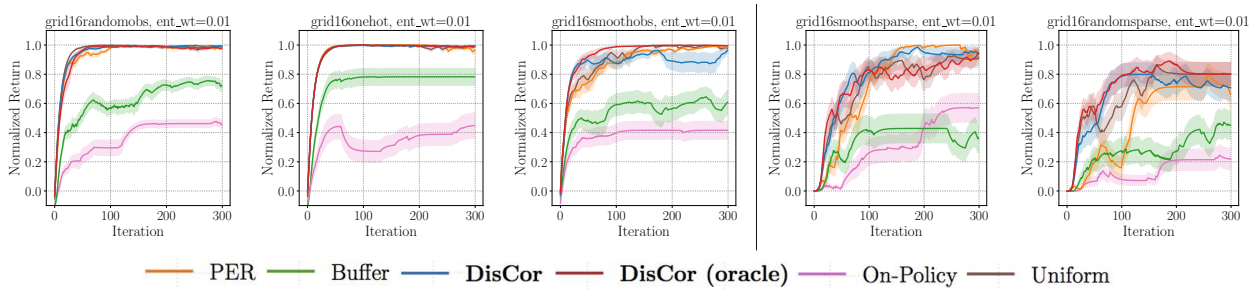


Figure 11. Learning curves for different algorithms in the **exact** setting. Note that DisCor (blue) and DisCor (oracle) (red) are generally the best algorithms in these settings. Replay Buffers (green) help over on-policy (pink) distributions. Prioritizing transitions based on high Bellman error (orange) is performant in some cases, but hurts in the other cases – it is especially slow in cases with sparse rewards, note the speed of learning on grid16randomsparse and grid16smoothsparse (**right** of the vertical line) environments.

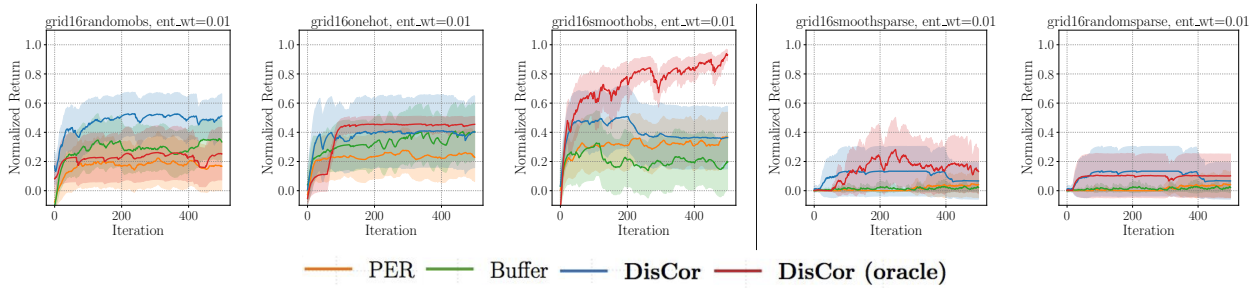


Figure 12. Learning curves for different algorithms in the **sampled** setting. Note that DisCor and DisCor (oralce) anre generally the best algorithms in these settings. Replay Buffers (green) help over on-policy (gray) distributions, but may the algorithm may still fail to reach optimal return. Prioritizing for high Bellman error (PER) may fail to learn in sparse-reward tasks as is evident from the curves for sparse reward environments (**right** of the vertical line).

outperforms both the baseline SAC and the prior method PER in all six cases, achieving nearly **50%** more than the returns achieved by SAC.

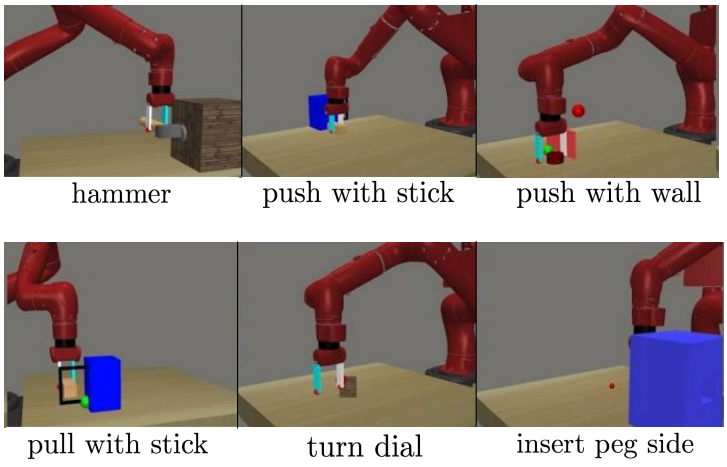


Figure 13. Visual description of the six MetaWorld tasks used in our experiments in Section 4. Figures taken from (Yu et al., 2019).

I.3. OpenAI Gym Benchmarks

Here we present an evaluation on the standard OpenAI continuous control gym benchmark environments. Modern ADP algorithms such as SAC can already solve these tasks easily, without any issues, since these algorithms have been tuned on these tasks. A comparison of the three algorithms DisCor, SAC and PER, on three of these benchmark tasks is shown in Figure 15 (top). We note that in this case, all the algorithms are roughly comparable to each other. For instance, DisCor performs better than SAC and PER on Walker2d, however, is outperformed by SAC on Ant.

Stochastic reward signals. That said, we also performed an experiment to verify the impact of stochasticity, such as noise in the reward signal, on the DisCor algorithm as compared to other baseline algorithms like SAC and PER. Analogous the diagnostic tabular experiments on low signal-to-noise ratio environments, such as those with sparse reward, we would expect a baseline ADP method to be impacted more due to an absence of corrective feedback in tasks with stochastic reward noise, since a noisy reward effectively reduces the signal-to-noise ratio. We would also expect a method that ensures corrective feedback to perform better.

In order to test this hypothesis, we created stochastic reward tasks out of the OpenAI gym benchmarks. We modified the reward function $r(s, a)$ in these gym tasks to be equal to:

$$r'(s, a) = r(s, a) + z, \quad z \sim \mathcal{N}(0, 1) \tag{45}$$

and the agent is only provided these noisy rewards during training. However, we only report the deterministic ground-truth

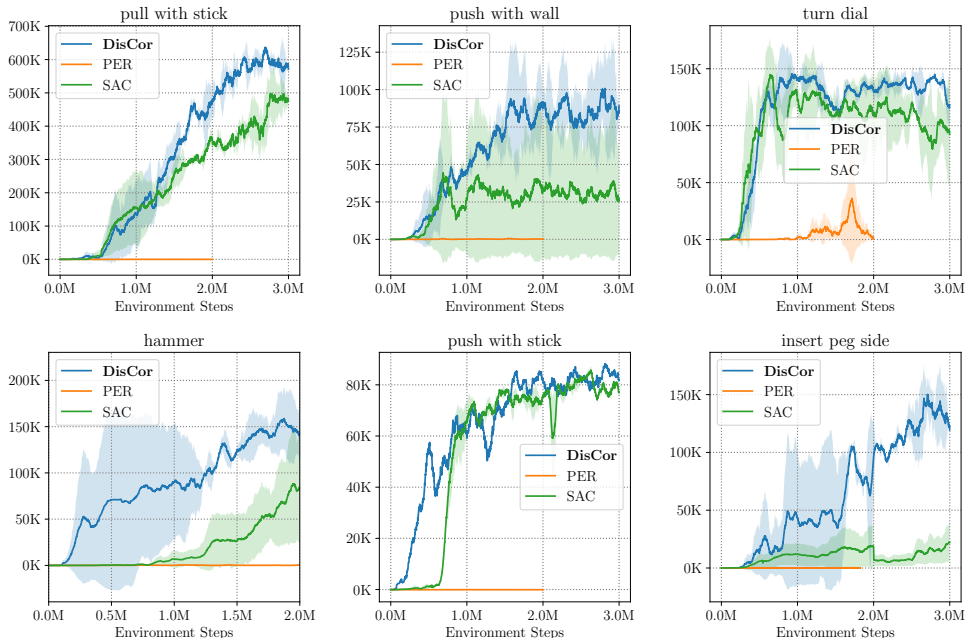


Figure 14. Evaluation average return achieved by DisCor (blue), SAC (green) and PER (orange) on six Metaworld benchmarks. From left to right: pull stick, push with wall, push with stick, turn dial, hammer and insert peg side tasks. Note that DisCor clearly achieves better returns or learns faster in most of the tasks.

reward during evaluation. We present the results in Figure 15 (bottom). Observe that in this scenario, DisCor emerges as the best performing algorithm on these tasks, and outperforms other baselines SAC and PER both in terms of asymptotic performance (example, HalfCheetah) and sample efficiency (example, Ant).

We also compare DisCor to AFM (Fu et al., 2019), a prior method similar to prioritized experience replay on the MuJoCo gym benchmarks. We find that DisCor clearly outperforms AFM in these scenarios. We present these results in Figure 15 (bottom) where the top row presents results in the case of regular gym benchmarks, and the bottom row presents results in the case of gym benchmarks with stochastic reward noise.

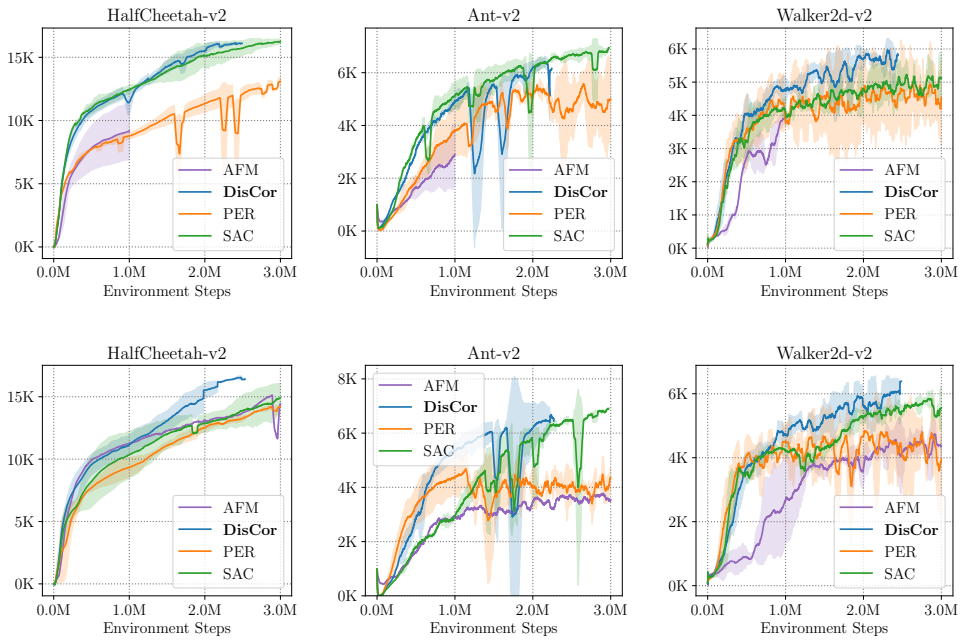
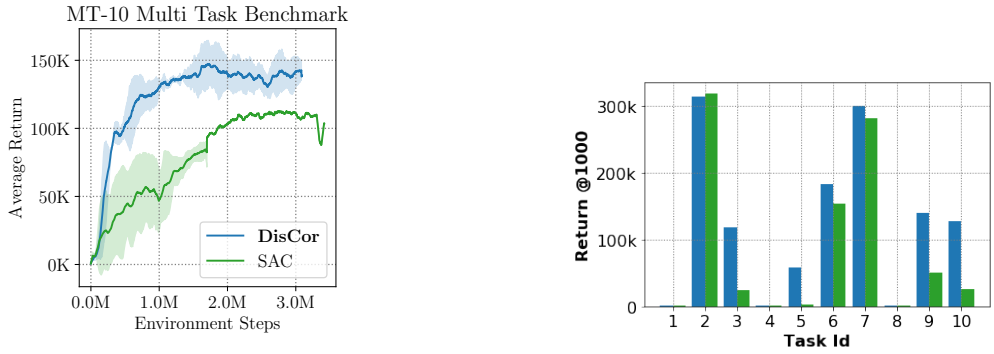


Figure 15. Performance of DisCor, SAC, PER and AFM on continuous control gym benchmarks (top row) and gym benchmarks *with stochastic reward noise* (bottom row). Observe that DisCor learns slightly faster and performs better than SAC and PER on these stochastic problems and that DisCor clearly out-performs AFM in both scenarios, on all three benchmarks tested on.

I.4. MT10 Multi-Task Experiments

In this section, we present the trend of returns, as a learning curve and as a comparative histogram (at 1M environment steps of training) for the multi-task MT10 benchmark, extending the results shown in Section F.3, Figure 7. These plots are shown in Figure 16. Observe that DisCor achieves more than **30%** of the return of SAC, and obtains an individually higher value of return on more tasks.



(a) Average task return

(b) Per-task return at 1M steps

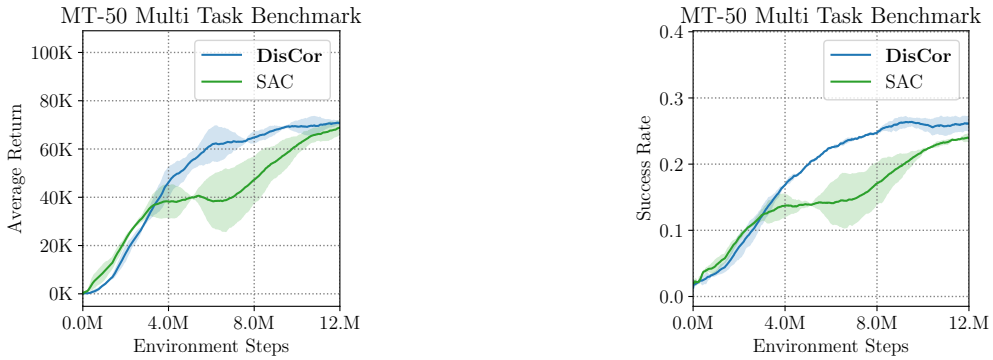
Figure 16. Performance of DisCor and SAC on the MT10 benchmark. Returns for DisCor are higher than SAC by around **30%**; (2) DisCor achieves a non-trivial return on **7/10** tasks after 1000k steps, as compared to **3/10** for unweighted SAC, similar to the trend at 500k steps shown in Figure 7.

I.5. MT50 Multi-Task Experiments

We further evaluated the performance of DisCor on the multi-task MT50 benchmark (Yu et al., 2019). This is an extremely challenging benchmark where the task is to learn a single policy that can solve 50 tasks together, with the same evaluation protocol as previously used in the MT10 experiments (Section F.3 and Appendix I.4). We present the results (average task return and average success rate) in Figures 17. Note that while SAC tends to saturate/plateau in between 4M - 8M steps, accounting for corrective feedback via the DisCor algorithm makes the algorithm continue learning in that scenario too.

Game	n-step DQN ($n = 3$)	DisCor (Regular)	n-step DisCor ($n = 3$)
Pong	17	17	**19**
Breakout	37	**175**	47

Table 1. Average Performance of DQN + 3-step returns, DisCor and Discor + 3-step returns on Pong and Breakout at 60M steps into training, rounded off to the nearest integer. Note that DisCor clearly outperforms DQN with multi-step returns. We also find that adding n-step returns to DisCor can hurt, for instance, on Breakout, where the same hurts with DQN as well (for comparison, see Figure 8 in the main paper), however, we still observe that DisCor, when applied with multi-step returns performs better than DQN with multi-step returns as well, indicating the benefits of DisCor even when methods such as multi-step returns are used.



(a) Average task return

(b) Average success rate

Figure 17. Performance of DisCor and SAC on the MT50 benchmark. Note that, DisCor clearly keeps learning unlike SAC which tends to plateau for about 3M steps in the middle (the stretch between 4M and 7M steps on the x-axis, where SAC exhibits a small gradient in the learning progress, whereas DisCor continuously keeps learning).

I.6. DQN with multi-step returns

N-step returns with DQN are hypothesized to stabilize learning since updates to the Q-function now depends on reward values spanning multiple steps, and the coefficient of the bootstrapped Q-value is γ^T, which is exponentially smaller than γ used conventionally in Bellman backups, implying that the error accumulation process due to incorrect targets is reduced. Thus, we perform a comparison of DisCor and DQN with n-step backups, where n was chosen to be 3, $n = 3$, in accordance with commonly used multi-step return settings for Atari games. We present the average return obtained by DisCor and DQN (+n-step), with sticky actions, in Table 1. We clearly observe that DisCor outperforms DQN with 3-step returns in all three games evaluated on. We also observe that n-step returns applied with DisCor also outperform n-step returns applied with DQN, indicating the benefits of using DisCor even when other techniques, such as n-step returns are used.

I.7. Code for the Method

The code is shown in Figure 18. It is a simplified version of the code from our implementation of DisCor on top of the official SAC repository (Tuomas Haarnoja and Levine, 2018).

```
    def _init_critic_update_with_dist(self):
        """Update critic with distribution weighting,
           and update \delta_\phi using recursive update. """
        next_actions = self._policy.actions([self._next_observations_ph])

        ## Compute errors at next state, and an action from the policy
        qf_pred_errs = self._error_fns([self._next_observations_ph, next_actions])

        ## error_model_tau_ph: moving mean of the error values over batches
        err_logits = -tf.stop_gradient(
            self._discount * qf_pred_errs / self._error_model_tau_ph)

        Q_target = tf.stop_gradient(self._get_Q_target())
        Q_values = self._Q([self._observations_ph, self._actions_ph])

        ## Compute importance sampled loss, also perform self-normalized sampling
        loss, weights = importance_sampled_loss(
            labels=Q_target, predictions=Q_values,
            weights=err_logits, weight_options='self_normalized')

        ## Train Q-function
        Q_training_ops = tf.contrib.layers.optimize_loss(loss, learning_rate=self._Q_lr,
            optimizer=self._Q_optimizer, variables=self._Q.trainable_variables)
        training_ops.update({'Q': tf.group(Q_training_ops)})

        ## Training the error function
        err_values = self._error_fns([self._observations_ph, self._actions_ph])

        ## Mean Bellman error used to compute target values for error
        bellman_errors = tf.abs(Q_values - Q_target)
        err_targets = tf.stop_gradient(self._get_error_target(bellman_errors))

        ## This is used to update the moving mean, self._error_model_tau_ph
        self._mean_error_values = tf.reduce_mean(err_values)

        ## Simple mean squared error loss for \delta_\phi
        err_losses =  tf.losses.mean_squared_error(
            labels=err_targets, predictions=err_values, weights=0.5)

        ## Update error function: \delta_\phi
        err_training_ops = tf.contrib.layers.optimize_loss(err_losses,
            learning_rate=self._dist_lr,
            optimizer=self._err_optimizer, variables=self._error_fns.trainable_variables)
        training_ops.update({'Error': tf.group(err_training_ops)})

```

Figure 18. Code for training the error function Δ_ϕ, and modified training for the Q-function $Q(s, a)$ using Δ_ϕ to get weights $w(s, a)$ for training. Code written in convention with regular Tensorflow guidelines, in the same style as the official SAC implementation (Tuomas Haarnoja and Levine, 2018).