Non-delusional Q-learning and value iteration

Tyler Lu
Google AI
tylerlu@google.com

Dale Schuurmans
Google AI
schuurmans@google.com

Craig Boutilier
Google AI
cboutilier@google.com

Abstract

We identify a fundamental source of error in Q-learning and other forms of dynamic programming with function approximation. *Delusional bias* arises when the approximation architecture limits the class of expressible greedy policies. Since standard Q-updates make globally uncoordinated action choices with respect to the expressible policy class, inconsistent or even conflicting Q-value estimates can result, leading to pathological behaviour such as over/under-estimation, instability and even divergence. To solve this problem, we introduce a new notion of *policy consistency* and define a local backup process that ensures global consistency through the use of *information sets*—sets that record constraints on policies consistent with backed-up Q-values. We prove that both the model-based and model-free algorithms using this backup remove delusional bias, yielding the first known algorithms that guarantee optimal results under general conditions. These algorithms furthermore only require polynomially many information sets (from a potentially exponential support). Finally, we suggest other practical heuristics for value-iteration and Q-learning that attempt to reduce delusional bias.

1 Introduction

Q-learning is a foundational algorithm in reinforcement learning (RL) [34, 26]. Although Q-learning is guaranteed to converge to an optimal state-action value function (or Q-function) when state-action pairs are explicitly enumerated [34], it is potentially unstable when combined with function approximation (even simple linear approximation) [1, 8, 29, 26]. Numerous modifications of the basic update, restrictions on approximators, and training regimes have been proposed to ensure convergence or improve approximation error [12, 13, 27, 18, 17, 21]. Unfortunately, simple modifications are unlikely to ensure near-optimal performance, since it is NP-complete to determine whether even a linear approximator can achieve small worst-case Bellman error [23]. Developing variants of Q-learning with good worst-case behaviour for standard function approximators has remained elusive.

Despite these challenges, Q-learning remains a workhorse of applied RL. The recent success of *deep* Q-learning, and its role in high-profile achievements [19], seems to obviate concerns about the algorithm's performance: the use of deep neural networks (DNNs), together with various augmentations (such as experience replay, hyperparameter tuning, etc.) can reduce instability and poor approximation. However, deep Q-learning is far from robust, and can rarely be applied successfully by inexperienced users. Modifications to mitigate systematic risks in Q-learning include double Q-learning [30], distributional Q-learning [4], and dueling network architectures [32]. A study of these and other variations reveals surprising results regarding the relative benefits of each under ablation [14]. Still, the full range of risks of approximation in Q-learning has yet to be delineated.

In this paper, we identify a fundamental problem with Q-learning (and other forms of dynamic programming) with function approximation, distinct from those previously discussed in the literature. Specifically, we show that approximate Q-learning suffers from *delusional bias*, in which updates are based on mutually inconsistent values. This inconsistency arises because the Q-update for a state-action pair, (s, a), is based on the *maximum* value estimate over all actions at the next state, which

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

ignores the fact that the actions so-considered (including the choice of a at s) might not be *jointly realizable* given the set of admissible policies derived from the approximator. These "unconstrained" updates induce errors in the target values, and cause a distinct source of value estimation error: Q-learning readily backs up values based on action choices that the greedy policy class cannot realize.

Our first contribution is the identification and precise definition of delusional bias, and a demonstration of its detrimental consequences. From this new perspective, we are able to identify anomalies in the behaviour of Q-learning and value iteration (VI) under function approximation, and provide new explanations for previously puzzling phenomena. We emphasize that delusion is an inherent problem affecting the interaction of Q-updates with constrained policy classes—more expressive approximators, larger training sets and increased computation do not resolve the issue.

Our second contribution is the development of a new *policy-consistent backup* operator that fully resolves the problem of delusion. Our notion of consistency is in the same spirit as, but extends, other recent notions of temporal consistency [5, 22]. This new operator does not simply backup a single future value at each state-action pair, but instead backs up a *set* of candidate values, each with the associated set of policy commitments that justify it. We develop a model-based value iteration algorithm and a model-free Q-learning algorithm using this backup that carefully integrate value- and policy-based reasoning. These methods complement the value-based nature of value iteration and Q-learning with explicit constraints on the policies consistent with generated values, and use the values to select policies from the admissible policy class. We show that in the tabular case with policy constraints—isolating delusion-error from approximation error—the algorithms converge to an optimal policy in the admissible policy class. We also show that the number of information sets is bounded polynomially when the greedy policy class has finite VC-dimension; hence, the algorithms have polynomial-time iteration complexity in the tabular case.

Finally, we suggest several heuristic methods for imposing policy consistency in batch Q-learning for larger problems. Since consistent backups can cause information sets to proliferate, we suggest search heuristics that focus attention on promising information sets, as well as methods that impose (or approximate) policy consistency within batches of training data, in an effort to drive the approximator toward better estimates.

2 Preliminaries

A *Markov decision process (MDP)* is defined by a tuple $\mathbf{M} = (S, A, p, p_0, R, \gamma)$ specifying a set of *states* S and *actions* A; a transition kernel p; an initial state distribution p_0; a reward function R; and a discount factor $\gamma \in [0, 1]$. A (stationary, deterministic) *policy* $\pi : S \to A$ specifies the agent's action at every state s. The state-value function for π is given by $V^\pi(s) = \mathbb{E}[\sum_{t\geq 0} \gamma^t R(s_t, \pi(s_t))]$ while the state-action value (or *Q-function*) is $Q^\pi(s, a) = R(s, a) + \gamma\, \mathbb{E}_{p(s'|s,a)}\, V^\pi(s')$, where expectations are taken over random transitions and rewards. Given any Q-function, the policy "Greedy" is defined by selecting an action a at state s that maximizes $Q(s, a)$. If $Q = Q^*$, then Greedy is optimal.

When p is unknown, Q-learning can be used to acquire the optimal Q^* by observing trajectories generated by some (sufficiently exploratory) behavior policy. In domains where tabular Q-learning is impractical, *function approximation* is typically used [33, 28, 26]. With function approximation, Q-values are approximated by some function from a class parameterized by Θ (e.g., the weights of a linear function or neural network). We let $\mathcal{F} = \{f_\theta : S \times A \to \mathbb{R} \mid \theta \in \Theta\}$ denote the set of expressible value function approximators, and denote the class of admissible greedy policies by

$$G(\Theta) = \Big\{\pi_\theta \;\Big|\; \pi_\theta(s) = \operatorname*{argmax}_{a\in A} f_\theta(s, a), \theta \in \Theta\Big\}. \tag{1}$$

In such cases, online *Q-learning* at transition s, a, r, s' (action a is taken at state s, leading to reward r and next state s') uses the following update given a previously estimated Q-function $Q_\theta \in \mathcal{F}$,

$$\theta \leftarrow \theta + \alpha\Big(r + \gamma \max_{a'\in A} Q_\theta(s', a') - Q_\theta(s, a)\Big)\nabla_\theta Q_\theta(s, a). \tag{2}$$

Batch versions of Q-learning (e.g., fitted Q-iteration, batch experience replay) are similar, but fit a regressor repeatedly to batches of training examples (and are usually more data efficient and stable).

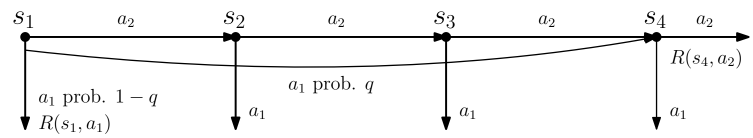

Figure 1: A simple MDP that illustrates delusional bias (see text for details).

3 Delusional bias and its consequences

The problem of delusion can be given a precise statement (which is articulated mathematically in Section 4): *delusional bias occurs whenever a backed-up value estimate is derived from action choices that are not realizable in the underlying policy class*. A Q-update backs up values for each state-action pair (s, a) by *independently* choosing actions at the corresponding next states s' via the max operator; this process implicitly assumes that $\max_{a' \in A} Q_\theta(s', a')$ is achievable. However, the update can become inconsistent under function approximation: if no policy in the admissible class can jointly express all past (implicit) action selections, backed-up values do not correspond to Q-values that can be achieved by any expressible policy. (We note that the source of this potential estimation error is quite different than the optimistic bias of maximizing over noisy Q-values addressed by double Q-learning; see Appendix A.5.) Although the consequences of such delusional bias might appear subtle, we demonstrate how delusion can profoundly affect both Q-learning and value iteration. Moreover, these detrimental effects manifest themselves in diverse ways that appear disconnected, but are symptoms of the same underlying cause. To make these points, we provide a series of concrete counter-examples. Although we use linear approximation for clarity, the conclusions apply to any approximator class with finite capacity (e.g., DNNs with fixed architectures), since there will always be a set of $d+1$ state-action choices that are jointly infeasible given a function approximation architecture with VC-dimension $d < \infty$ [31] (see Theorem 1 for the precise statement).

3.1 A concrete demonstration

We begin with a simple illustration. Consider the undiscounted MDP in Fig. 1, where episodes start at s_1, and there are two actions: a_1 causes termination, except at s_1 where it can move to s_4 with probability q; a_2 moves deterministically to the next state in the sequence s_1 to s_4 (with termination when a_2 taken at s_4). All rewards are 0 except for $R(s_1, a_1)$ and $R(s_4, a_2)$. For concreteness, let $q = 0.1$, $R(s_1, a_1) = 0.3$ and $R(s_4, a_2) = 2$. Now consider a linear approximator $f_\theta(\phi(s, a))$ with two state-action features: $\phi(s_1, a_1) = \phi(s_4, a_1) = (0, 1)$; $\phi(s_1, a_2) = \phi(s_2, a_2) = (0.8, 0)$; $\phi(s_3, a_2) = \phi(s_4, a_2) = (-1, 0)$; and $\phi(s_2, a_1) = \phi(s_3, a_1) = (0, 0)$. Observe that no $\pi \in G(\Theta)$ can satisfy both $\pi(s_2) = a_2$ and $\pi(s_3) = a_2$, hence the optimal unconstrained policy (take a_2 everywhere, with expected value 2) is not realizable. Q-updating can therefore never converge to the unconstrained optimal policy. Instead, the optimal *achievable* policy in $G(\Theta)$ takes a_1 at s_1 and a_2 at s_4 (achieving a value of 0.5, realizable with $\theta^* = (-2, 0.5)$).

Unfortunately, Q-updating is unable to find the optimal admissible policy π_{θ^*} in this example. How this inability materializes depends on the update regime, so consider online Q-learning (Eq. 2) with data generated using an εGreedy behavior policy ($\varepsilon = 0.5$). In this case, it is not hard to show that Q-learning must converge to a fixed point $\hat{\theta} = (\hat{\theta}_1, \hat{\theta}_2)$ where $-\hat{\theta}_1 \leq \hat{\theta}_2$, implying that $\pi_{\hat{\theta}}(s_2) \neq a_2$, i.e., $\pi_{\hat{\theta}} \neq \pi_{\theta^*}$ (we also show this for any $\varepsilon \in [0, 1/2]$ when $R(s_1, a_1) = R(s_4, a_2) = 1$; see derivations in Appendix A.1). Instead, Q-learning converges to a fixed point that gives a "compromised" admissible policy which takes a_1 at both s_1 and s_4 (with a value of 0.3; $\hat{\theta} \approx (-0.235, 0.279)$).

This example shows how delusional bias prevents Q-learning from reaching a reasonable fixed-point. Consider the backups at (s_2, a_2) and (s_3, a_2). Suppose $\hat{\theta}$ assigns a "high" value to (s_3, a_2) (i.e., so that $Q_{\hat{\theta}}(s_3, a_2) > Q_{\hat{\theta}}(s_3, a_1)$) as required by π_{θ^*}; intuitively, this requires that $\hat{\theta}_1 < 0$, and generates a "high" bootstrapped value for (s_2, a_2). But any update to $\hat{\theta}$ that tries to fit this value (i.e., makes $Q_{\hat{\theta}}(s_2, a_2) > Q_{\hat{\theta}}(s_2, a_1)$) forces $\hat{\theta}_1 > 0$, which is inconsistent with the assumption, $\hat{\theta}_1 < 0$, needed to generate the high bootstrapped value. In other words, any update that moves (s_2, a_2) higher *undercuts the justification* for it to be higher. The result is that the Q-updates compete with each other, with $Q_{\hat{\theta}}(s_2, a_2)$ converging to a compromise value that is not realizable by any policy in $G(\Theta)$.

This induces an inferior policy with lower expected value than π_{θ^*}. We show in Appendix A.1 that avoiding any backup of these inconsistent edges results in Q-learning converging to the optimal expressible policy. Critically, this outcome is not due to approximation error itself, but the inability of Q-learning to find the value of the optimal *representable* policy.

3.2 Consequences of delusion

There are several additional manifestations of delusional bias that cause detrimental outcomes under Q-updating. Concrete examples are provided to illustrate each, but we relegate details to the appendix.

Divergence: Delusional bias can cause Q-updating to diverge. We provide a detailed example of divergence in Appendix A.2 using a simple linear approximator. While divergence is typically attributed to the interaction of the approximator with Bellman or Q-backups, the example shows that if we correct for delusional bias, convergent behavior is restored. Lack of convergence due to cyclic behavior (with a lower-bound on learning rates) can also be caused by delusion: see Appendix A.3.

The Discounting Paradox: Another phenomenon induced by delusional bias is the *discounting paradox*: given an MDP with a specific discount factor γ_{eval}, Q-learning with a *different* discount γ_{train} results in a Q-function whose greedy policy has better performance, relative to the target γ_{eval}, than when trained with γ_{eval}. In Appendix A.4, we provide an example where the paradox is extreme: a policy trained with $\gamma = 1$ is provably *worse* than one trained myopically with $\gamma = 0$, even when evaluated using $\gamma = 1$. We also provide an example where the gap can be made arbitrarily large. These results suggest that treating the discount as hyperparameter might yield systematic training benefits; we demonstrate that this is indeed the case on some benchmark (Atari) tasks in Appendix A.10.

Approximate Dynamic Programming: Delusional bias arises not only in Q-learning, but also in approximate dynamic programming (ADP) (e.g., [6, 9]), such as approximate value iteration (VI). With value function approximation, VI performs full state Bellman backups (as opposed to sampled backups as in Q-learning), but, like Q-learning, applies the max operator independently at successor states when computing expected next state values. When these choices fall outside the greedy policy class admitted by the function approximator, delusional bias can arise. Delusion can also occur with other forms of policy constraints (without requiring the value function itself to be approximated).

Batch Q-learning: In the example above, we saw that delusional bias can cause convergence to Q-functions that induce poor (greedy) policies in standard online Q-learning. The precise behavior depends on the training regime, but poor behavior can emerge in batch methods as well. For instance, batch Q-learning with experience replay and replay buffer shuffling will induce the same tension between the conflicting updates. Specific (nonrandom) batching schemes can cause even greater degrees of delusion; for example, training in a sequence of batches that run through a batch of transitions at s_4, followed by batches at s_3, then s_2, then s_1 will induce a Q-function that deludes itself into estimating the value of (s_1, a_2) to be that of the optimal unconstrained policy.

4 Non-delusional Q-learning and dynamic programming

We now develop a provably correct solution that directly tackles the source of the problem: the potential inconsistency of the set of Q-values used to generate a Bellman or Q-backup. Our approach avoids delusion by using *information sets* to track the "dependencies" contained in all Q-values, i.e., the *policy assumptions* required to justify any such Q-value. Backups then prune infeasible values whose information sets are not policy-class consistent. Since backed-up values might be designated inconsistent when new dependencies are added, this *policy-consistent backup* must maintain *alternative* information sets and their corresponding Q-values, allowing the (implicit) backtracking of prior decisions (i.e., max Q-value choices). Such a policy-consistent backup can be viewed as unifying both value- and policy-based RL methods, a perspective we detail in Sec. 4.3.

We develop policy consistent backups in the tabular case while allowing for an arbitrary policy class (or arbitrary policy constraints)—the case of greedy policies with respect to some approximation architecture f_θ is simply a special case. This allows the method to focus on delusion, without making any assumptions about the specific value approximation. Because delusion is a general phenomenon, we first develop a model-based consistent backup, which gives rise to non-delusional *policy-class value iteration*, and then describe the sample-backup version, *policy-class Q-learning*. Our main theorem establishes the convergence, correctness and optimality of the algorithm (including

Algorithm 1 Policy-Class Value Iteration (PCVI)

Input: $S, A, p(s' \mid s, a), R, \gamma, \Theta$, initial state s_0

$\mathtt{Q}[sa] \leftarrow$ initialize to mapping $\Theta \mapsto 0$ for all s, a
$\mathtt{ConQ}[sa] \leftarrow$ initialize to mapping $[s \mapsto a] \mapsto 0$ for all s, a
Update $\mathtt{ConQ}[s]$ for all s (i.e., combine all table entries in $\mathtt{ConQ}[sa_1], \ldots, \mathtt{ConQ}[sa_m]$)
repeat
  for all s, a **do**
    $\mathtt{Q}[sa] \leftarrow R_{sa} + \gamma \bigoplus_{s'} p(s' \mid s, a)\mathtt{ConQ}[s']$
    $\mathtt{ConQ}[sa](Z) \leftarrow \mathtt{Q}[sa](X)$ for all X such that $Z = X \cap [s \mapsto a]$ is non-empty
    Update $\mathtt{ConQ}[s]$ by combining table entries of $\mathtt{ConQ}[sa']$ for all a'
  end for
until $\mathtt{Q}$ converges: $\mathsf{dom}(\mathtt{Q}(sa))$ and $\mathtt{Q}(sa)(X)$ does not change for all s, a, X
`/* Then recover an optimal policy */`
$X^* \leftarrow \operatorname{argmax}_X \mathtt{ConQ}[s_0](X)$
$q^* \leftarrow \mathtt{ConQ}[s_0](X^*)$
$\theta^* \leftarrow \mathsf{Witness}(X^*)$
return π_{θ^*} and q^*.

the complete removal of delusional bias), and computational tractability (subject to a tractable consistency oracle).

4.1 Policy-class value iteration

We begin by defining *policy-class value iteration (PCVI)*, a new VI method that operates on collections of information sets to guarantee discovery of the optimal policy in a given class. For concreteness, we specify a policy class using Q-function parameters, which determines the class of realizable greedy policies (just as in classical VI). Proofs and more formal definitions can be found in Appendix A.6. We provide a detailed illustration of the PCVI algorithm in Appendix A.7, walking through the steps of PCVI on the example MDP in Fig. 1.

Assume an MDP with n states $S = \{s_1, \ldots, s_n\}$ and m actions $A = \{a_1, \ldots, a_m\}$. Let Θ be the parameter class defining Q-functions. Let $\mathcal{F}$ and $G(\Theta)$, as above, denote the class of expressible value functions and admissible greedy policies respectively. (We assume ties are broken in some canonical fashion.) Define $[s \mapsto a] = \{\theta \in \Theta \mid \pi_\theta(s) = a\}$. An *information set* $X \subseteq \Theta$ is a set of parameters (more generally, policy constraints) that justify assigning a particular Q-value q to some (s, a) pair. Below we use the term "information set" to refer both to X and (X, q) as needed.

Information sets will be organized into finite *partitions* of Θ, i.e., a set of non-empty subsets $P = \{X_1, \ldots, X_k\}$ such that $X_1 \cup \cdots \cup X_k = \Theta$ and $X_i \cap X_j = \emptyset$, for all $i \neq j$. We abstractly refer to the elements of P as *cells*. A partition P' is a *refinement* of P if for all $X' \in P'$ there exists an $X \in P$ such that $X' \subseteq X$. Let $\mathcal{P}(\Theta)$ be the set of all finite partitions of Θ. A *partition function* $h : P \to \mathbb{R}$ associates values (e.g., Q-values) with all cells (e.g., information sets). Let $\mathcal{H} = \{h : P \to \mathbb{R} \mid P \in \mathcal{P}(\Theta)\}$ denote the set of all such partition functions. Define the *intersection sum* for $h_1, h_2 \in \mathcal{H}$ to be:

$$(h_1 \oplus h_2)(X_1 \cap X_2) = h_1(X_1) + h_2(X_2), \quad \forall X_1 \in \mathsf{dom}(h_1), X_2 \in \mathsf{dom}(h_2), X_1 \cap X_2 \neq \emptyset.$$

Note that the intersection sum incurs at most a quadratic blowup: $|\mathsf{dom}(h)| \leq |\mathsf{dom}(h_1)| \cdot |\mathsf{dom}(h_2)|$.

The methods below require an *oracle* to check whether a policy π_θ is consistent with a set of state-to-action constraints: i.e., given $\{(s, a)\} \subseteq S \times A$, whether there exists a $\theta \in \Theta$ such that $\pi_\theta(s) = a$ for all pairs. We assume access to such an oracle, "Witness". For *linear* Q-function parameterizations, Witness can be implemented in polynomial time by checking the consistency of a system of linear inequalities.

PCVI, shown in Alg. 1, computes the optimal policy $\pi_{\theta^*} \in G(\Theta)$ by using information sets and their associated Q-values organized into partitions (i.e., partition functions over Θ). We represent Q-functions using a table $\mathtt{Q}$ with one entry $\mathtt{Q}[sa]$ for each (s, a) pair. Each such $\mathtt{Q}[sa]$ is a partition function over $\mathsf{dom}(\mathtt{Q}[sa]) \in \mathcal{P}(\Theta)$. For each $X_i \in \mathsf{dom}(\mathtt{Q}[sa])$ (i.e., for each information set $X_i \subseteq \Theta$ associated with (s, a)), we assign a unique Q-value $\mathtt{Q}[sa](X_i)$. Intuitively, the Q-value $\mathtt{Q}[sa](X_i)$ is *justified* only if we limit attention to policies $\{\pi_\theta : \theta \in X_i\}$. Since $\mathsf{dom}(\mathtt{Q}[sa])$ is a partition, we have a Q-value for *any realizable policy*. (The partitions $\mathsf{dom}(\mathtt{Q}[sa])$ for each (s, a) generally differ.)

$\mathtt{ConQ}[sa]$ is a restriction of $\mathtt{Q}[sa]$ obtained by intersecting each cell in its domain, $\mathsf{dom}(\mathtt{Q}[sa])$, with $[s \mapsto a]$. In other words, $\mathtt{ConQ}[sa]$ is a partition function defined on some partition of $[s \mapsto a]$ (rather than all of Θ), and represents Q-values of cells that are consistent with $[s \mapsto a]$. Thus, if $X_i \cap [s \mapsto a] = \emptyset$ for some $X_i \in \mathsf{dom}(\mathtt{Q}[sa])$, the corresponding Q-value disappears in $\mathtt{ConQ}[sa]$. Finally, $\mathtt{ConQ}[s] = \cup_a \mathtt{ConQ}[sa]$ is the partition function over Θ obtained by collecting all the "restricted" action value functions. Since $\cup_a [s \mapsto a]$ is a partition of Θ, so is $\mathtt{ConQ}[s]$.

The key update in Alg. 1 is Line 6, which jointly updates all Q-values of the relevant sets of policies in $G(\Theta)$. Notice that the maximization typically found in VI is *not present*—this is because the operation computes and *records* Q-values for *all choices of actions at the successor state s'*. This is the key to allowing VI to maintain consistency: if a future Bellman backup is inconsistent with some previous max-choice at a reachable state, the corresponding cell will be pruned and an alternative maximum will take its place. Pruning of cells, using the oracle Witness, is implicit in Line 6 (pruning of $\oplus$) and Line 7 (where non-emptiness is tested).[1] Convergence of PCVI requires that each $\mathtt{Q}[sa]$ table—both its partition and associated Q-value—converge to a fixed point.

Theorem 1 (PCVI Theorem)**.** *PCVI (Alg. 1) has the following guarantees:*

(a) (Convergence and correctness) The function $\mathtt{Q}$ *converges and, for each $s \in S, a \in A$, and any $\theta \in \Theta$: there is a unique $X \in \mathsf{dom}(\mathtt{Q}[sa])$ such that $\theta \in X$ and $Q^{\pi_\theta}(s, a) = \mathtt{Q}[sa](X)$.*

(b) (Optimality and non-delusion) π_{θ^*} is an optimal policy within $G(\Theta)$ and q^* is its value.

(c) (Runtime bound) Assume $\oplus$ and non-emptiness checks (lines 6 and 7) have access to Witness*. Let $\mathcal{G} = \{g_\theta(s, a, a') := \mathbf{1}[f_\theta(s, a) - f_\theta(s, a') > 0], \forall s, a \neq a' \mid \theta \in \Theta\}$. Each iteration of Alg. 1 runs in time $O(nm \cdot [\binom{m}{2}n]^{2\,\mathsf{VCDim}(\mathcal{G})}(m-1)w)$ where $\mathsf{VCDim}(\cdot)$ denotes the VC-dimension of a set of boolean-valued functions, and w is the worst-case running time of* Witness *(with at most nm state-action constraints). Combined with Part (a), if $\mathsf{VCDim}(\mathcal{G})$ is finite then* $\mathtt{Q}$ *converges in time polynomial in n, m and w.*

Corollary 2. *Alg. 1 runs in polynomial time for linear greedy policies. It runs in polynomial time in the presence of a polynomial time* Witness *for deep Q-network (DQN) greedy policies.*

(A more complete statement of the Cor. 2 is found in Appendix A.6.) The number of cells in a partition may be significantly less than suggested by the bounds, as it depends on the reachability structure of the MDP. For example, in an MDP with only self-transitions, the partition for each state has a single cell. We note that Witness is tractable for linear approximators, but is NP-hard for DNNs [7]. The poly-time result in Cor. 2 does not contradict the NP-hardness of finding a linear approximator with small worst-case Bellman error [23], since nothing is asserted about the Bellman error and we are treating the approximator's VC-dimension as a constant.

Demonstrating PCVI: We illustrate PCVI with a simple example that shows how poorly classical approaches can perform with function approximation, even in "easy" environments. Consider a simple deterministic grid world with the 4 standard actions and rewards of 0, except 1 at the top-right, 2 at the bottom-left, and 10 at the bottom-right corners; the discount is $\gamma = 0.95$. The agent starts at the top-left. The optimal policy is to move down the left side to the left-bottom corner, then along the bottom to the right bottom corner, then staying. To illustrate the effects of function approximation, we considered linear approximators defined over *random* feature representations: feature vectors were produced for each state-action pair by drawing independent standard normal values.

Fig. 2 shows the *estimated* maximum value achievable from the start state produced by each method (dark lines), along with the actual expected value achieved by the greedy policies produced by each method (light lines). The left figure shows results for a 4×4 grid with 4 random features, and the right for a 5×5 grid with 5 random features. Results are averaged over 10 runs with different random feature sets (shared by the algorithms). Surprisingly, even when the linear approximator can support near-optimal policies, classical methods can utterly fail to realize this possibility: in 9 of 10 trials (4×4) and 10 of 10 trials (5×5) the classical methods produce greedy policies with an expected value of *zero*, while PCVI produces policies with value *comparable to the global optimum*.

[1]If arbitrary policy constraints are allowed, there may be no feasible policies, in which case Witness will prune each cell immediately, leaving no Q-functions, as desired.

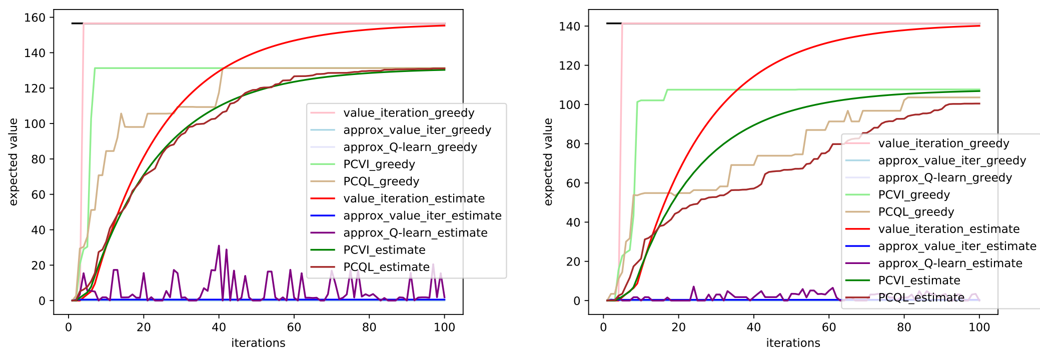

Figure 2: Planning and learning in a grid world with random feature representations. (Left: 4×4 grid using 4 features; Right: 5×5 grid using 5 features.) Here "iterations" means a full sweep over state-action pairs, except for Q-learning and PCQL, where an iteration is an episode of length $3/(1-\gamma) = 60$ using εGreedy exploration with $\varepsilon = 0.7$. Dark lines: estimated maximum achievable expected value. Light lines: actual expected value achieved by greedy policy.

4.2 Policy-class Q-learning

A tabular version of Q-learning using the same partition-function representation of Q-values as in PCVI yields *policy-class Q-learning PCQL*, shown in Alg. 2.[2] The key difference with PCVI is simply that we use *sample* backups in Line 4 instead of full Bellman backups as in PCVI.

Algorithm 2 Policy-Class Q-learning (PCQL)

Input: Batch $B = \{(s_t, a_t, r_t, s'_t)\}_{t=1}^{T}$, γ, Θ, scalars α_t^{sa}.
1: **for** $(s, a, r, s') \in B$, t is iteration counter **do**
2: For all a', if $s'a' \notin \texttt{ConQ}$ then initialize $\texttt{ConQ}[s'a'] \leftarrow ([s' \mapsto a'] \mapsto 0)$.
3: Update $\texttt{ConQ}[s']$ by combining $\texttt{ConQ}[s'a'](X)$, for all a', $X \in \mathsf{dom}(\texttt{ConQ}[s'a'])$
4: $\texttt{Q}[sa] \leftarrow (1-\alpha_t^{sa})\texttt{Q}[sa] \oplus \alpha_t^{sa}(r + \gamma\texttt{ConQ}[s'])$
5: $\texttt{ConQ}[sa](Z) \leftarrow \texttt{Q}[sa](X)$ for all X such that $Z = X \cap [s \mapsto a]$ is non-empty
6: **end for**
7: Return ConQ, Q

The method converges under the usual assumptions for Q-learning: a straightforward extension of the proof for PCVI, replacing full VI backups with Q-learning-style sample backups, yields the following:

Theorem 3. *The (a) convergence and correctness properties and (b) optimality and non-delusion properties associated with the PCVI Theorem 1 hold for PCQL, assuming the usual sampling requirements, the Robbins-Monro stochastic convergence conditions on learning rates α_t^{sa} and access to the* Witness *oracle.*

Demonstrating PCQL: We illustrate PCQL in the same grid world tasks as before, again using random features. Figure 2 shows that PCQL achieves comparable performance to PCVI, but with lighter time and space requirements, and is still significantly better than classical methods.

We also applied PCQL to the initial illustrative example in Fig. 1 with $R(s_1, a_1) = 0.3$, $R(s_4, a_2) = 2$ and uniform random exploration as the behaviour policy, adding the use of a value approximator (a linear regressor). We use a heuristic that maintains a global partition of Θ with each cell X holding a regressor $\texttt{ConQ}(s, a; w_X)$, for $w_X \in \Theta$ predicting the consistent Q-value at s, a (see details in Sec. 5 and Appendix A.8). The method converges with eight cells corresponding to the realizable policies. The policy (equivalence class) is $\texttt{ConQ}(s_1, \pi_X(s_1); w_X)$ where $\pi_X(s_1)$ is the cell's action

[2] PCQL uses the same type of initialization and optimal policy extraction as PCVI; details are omitted.

at s_1; the value is $w_X \cdot \phi(s, \pi_X(s_1))$. The cell X^* with the largest such value at s_1 is indeed the optimal realizable policy: it takes a_1 at s_1 and s_2, and a_2 elsewhere. The regressor $w_{X^*} \approx (-2, 0.5)$ fits the consistent Q-values perfectly, yielding optimal (policy-consistent) Q-values, because `ConQ` need not make tradeoffs to fit inconsistent values.

4.3 Unification of value- and policy-based RL

We can in some sense interpret PCQL (and, from the perspective of model-based approximate dynamic programming, PCVI) as unifying value- and policy-based RL. One prevalent view of value-based RL methods with function approximation, such Q-learning, is to find an approximate value function or Q-function (VF/QF) with low *Bellman error* (BE), i.e., where the (pointwise) difference between the approximate VF/QF and its Bellman backup is small. In approximate dynamic programming one often tries to minimize this directly, while in Q-learning, one usually fits a regression model to minimize the mean-squared temporal difference (a sampled form of Bellman error minimization) over a training data set. One reason for this emphasis on small BE is that the max norm of BE can be used to directly bound the (max norm) loss of the value of *greedy policy* induced by the approximate VF/QF and the value of the *optimal policy*. It is this difference in performance that is of primary interest.

Unfortunately, the bounds on induced policy quality using the BE approximation are quite loose, typically $2||\mathrm{BE}||_\infty/(1-\gamma)$ (see [6], bounds with ℓ_p norm are similar [20]). As such, minimizing BE does not generally provide policy guarantees of practical import (see, e.g., [11]). As we see in the cases above (and also in the appendix) that involve delusional bias, a small BE can in fact be rather misleading with respect to the induced policy quality. For example, Q-learning, using least squares to minimize the TD-error as a proxy for BE, often produces policies of poor quality.

PCQL and PCVI take a different perspective, embracing the fact that the VF/QF approximator strictly limits that class of greedy policies that can be realized. In these algorithms, no Bellman backup or Q-update ever involves values that cannot be realized by an admissible policy. This will often result in VFs/QFs with greater BE than their classical counterparts. But, in the exact tabular case, we derive the true value of the induced (approximator-constrained) policy and guarantee that it is optimal. In the regression case (see Sec. 5), we might view this as attempting to minimize BE *within the class of admissible policies*, since we only regress toward policy-consistent values.

The use of information sets and consistent cells effectively means that PCQL and PCVI are engaging in policy search—indeed, in the algorithms presented here, they can be viewed as enumerating all consistent policies (in the case of Q-learning, distinguishing only those that might differ on sampled data). In contrast to other policy-search methods (e.g., policy gradient), both PCQL and PCVI use (sampled or full) Bellman backups to direct the search through policy space, while simultaneously using policy constraints to limit the Bellman backups that are actually realized. They also use these values to select an optimal policy from the feasible policies generated within each cell.

5 Toward practical non-delusional Q-learning

The PCVI and PCQL algorithms can be viewed as constructs that demonstrate how delusion arises and how it can be eliminated in Q-learning and approximate dynamic programming by preventing inadmissible policy choices from influencing Q-values. However, the algorithms maintain information sets and partition functions, which is impractical with massive state and action sets. In this section, we suggest several heuristic methods that allow the propagation of some dependency information in practical Q-learning to mitigate the effects of delusional bias.

Multiple regressors: With multiple information sets (or cells), we no longer have a unique set of labels with which to fit an approximate Q-function regressor (e.g., DNN or linear approximator). Instead, each cell has its own set of labels. Thus, if we maintain a global collection of cells, each with its own Q-regressor, we have a set of approximate Q-functions that give both a compact representation and the ability to generalize across state-action pairs for any set of policy consistent assumptions. This works in both batch and pure online Q-learning (see Appendix A.8 for details.)

The main challenge is the proliferation of information sets. One obvious way to address this is to simply limit the total number of cells and regressors: given the current set of regressors, at any update, we first create the (larger number of) new cells needed for the new examples, fit the regressor for each new *consistent* cell, then prune cells according to some criterion to keep the total number of regressors manageable. This is effectively a search through the space of information sets and can be managed using a variety of methods (branch-and-bound, beam search, etc.). Criteria for generating, sampling and/or pruning cells can involve: (a) the magnitude to the Q-labels (higher expected values are better); (b) the constraints imposed by the cell (less restrictive is better, since it minimizes future inconsistency); the diversity of the cell assignments (since the search frontier is used to manage "backtracking").

If cell search maintains a restricted frontier, our cells may no longer cover all of policy space (i.e, `Q` is no longer a partition of Θ). This runs the risk that some future Q-updates may not be consistent with *any* cell. If we simply ignore such updates, the approach is *hyper-vigilant*, guaranteeing policy-class consistency at the expense of losing training data. An alternative relaxed approach is to merge cells to maintain a full partition of policy space (or prune cells and in some other fashion relax the constraints of the remaining cells to recover a partition). This *relaxed* approach ensures that all training data is used, but risks allowing some delusion to creep into values by not strictly enforcing all Q-value dependencies.

Q-learning with locally consistent data: An alternative approach is to simply maintain a single regressor, but ensure that any batch of Q-labels is self-consistent before updating the regressor. Specifically, given a batch of training data and the current regressor, we first create a single set of consistent labels for each example (see below), then update the regressor using these labels. With no information sets, the dependencies that justified the previous regressor are not accounted for when constructing the new labels. This may allow delusion to creep in; but the aim is that this heuristic approach may mitigate its effects since each new regressor is at least "locally" consistent with respect to its own updates. Ideally, this will keep the sequence of approximations in a region of θ-space where delusional bias is less severe. Apart from the use of a consistent labeling procedure, this approach incurs no extra overhead relative to Q-learning.

Oracles and consistent labeling: The first approach above requires an oracle, Witness, to test consistency of policy choices, which is tractable for linear approximators (linear feasibility test), but requires solving an integer-quadratic program when using DQN (e.g., a ReLU network). The second approach needs some means for generating consistent labels. Given a batch of examples $B = \{(s_t, a_t, r_t, s'_t)\}_{t=1}^{T}$, and a current regressor $\widetilde{Q}$, labels are generated by selecting an a'_t for each s'_t as the max. The selection should satisfy: (a) $\cap_t [s'_t \mapsto a'_t] \neq \emptyset$ (i.e., selected max actions are mutually consistent); and (b) $[s_t \mapsto a_t] \cap [s'_t \mapsto a'_t] \neq \emptyset$, for all t (i.e., choice at s'_t is consistent with taking a_t at s_t). We can find a consistent labeling maximizing some objective (e.g., sum of resulting labels), subject to these constraints. For a linear approximator, the problem can be formulated as a (linear) mixed integer program (MIP); and is amenable to several heuristics (see Appendix A.9).

6 Conclusion

We have identified *delusional bias*, a fundamental problem in Q-learning and approximate dynamic programming with function approximation or other policy constraints. Delusion manifests itself in different ways that lead to poor approximation quality or divergence for reasons quite independent of approximation error itself. Delusional bias thus becomes an important entry in the catalog of risks that emerge in the deployment of Q-learning. We have developed and analyzed a new policy-class consistent backup operator, and the corresponding model-based PCVI and model-free PCQL algorithms, that fully remove delusional bias. We also suggested several practical heuristics for large-scale RL problems to mitigate the effect of delusional bias.

A number of important direction remain. The further development and testing of practical heuristics for policy-class consistent updates, as well as large-scale experiments on well-known benchmarks, is critical. This is also important for identifying the prevalence of delusional bias in practice. Further development of practical consistency oracles for DNNs and consistent label generation is also of interest. We are also engaged in a more systematic study of the discounting paradox and the use of the discount factor as a hyper-parameter.

References

[1] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the International Conference on Machine Learning (ICML-95)*, pages 30–37, 1995.

[2] Peter L. Bartlett, Nick Harvey, Chris Liaw, and Abbas Mehrabian. Nearly-tight VC-dimension and pseudodimension bounds for piecewise linear neural networks. `arXiv:1703.02930`, 2017.

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013.

[4] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML-17)*, 2017.

[5] Marc G. Bellemare, Georg Ostrovski, Arthur Guez, Philip S. Thomas, and Rémi Munos. Increasing the action gap: New operators for reinforcement learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, pages 1476–1483, Québec City, QC, 2016.

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

[7] Avrim Blum and Ronald L. Rivest. Training a 3-node neural network is NP-complete. In *COLT*, pages 9–18, 1988.

[8] Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7 (NIPS-94)*, pages 369–376. MIT Press, Cambridge, 1995.

[9] Daniela Pucci de Farias and Benjamin Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865, 2003.

[10] Rick Durrett. *Probability: Theory and Examples*. Cambridge University Press, 2013.

[11] Matthieu Geist, Bilal Piot, and Olivier Pietquin. Is the Bellman residual a bad proxy? In *Advances in Neural Information Processing Systems 30 (NIPS-17)*, pages 3208–3217, Long Beach, CA, 2017.

[12] Geoffrey J. Gordon. Stable function approximation in dynamic programming. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML-95)*, pages 261–268, Lake Tahoe, 1995.

[13] Geoffrey J. Gordon. *Approximation Solutions to Markov Decision Problems*. PhD thesis, Carnegie Mellon University, 1999.

[14] 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. `arXiv:1710.02298`, 2017.

[15] Nan Jiang, Alex Kulesza, Satinder Singh, and Richard Lewis. The dependence of effective planning horizon on model accuracy. In *Proceedings of the Thirteenth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-14)*, pages 1181–1189, Istanbul, Turkey, 2015.

[16] Lucas Lehnert, Romain Laroche, and Harm van Seijen. On value function representation of long horizon problems. In *Proceedings of the Thirty-second AAAI Conference on Artificial Intelligence (AAAI-18)*, 2018.

[17] Hamid Maei, Csaba Szepesvári, Shalabh Bhatnagar, and Richard S. Sutton. Toward off-policy learning control with function approximation. In *International Conference on Machine Learning*, pages 719–726, Haifa, Israel, 2010.

[18] Francisco Melo and M. Isabel Ribeiro. Q-learning with linear function approximation. In *Proceedings of the International Conference on Computational Learning Theory (COLT)*, pages 308–322, 2007.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas 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. *Science*, 518:529–533, 2015.

[20] Rémi Munos. Performance bounds in lp-norm for approximate value iteration. *SIAM Journal on Control and Optimization*, 46(2):541–561, 2007.

[21] Rémi Munos, Thomas Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems 29 (NIPS-16)*, pages 1046–1054, Barcelona, 2016.

[22] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems 30 (NIPS-17)*, pages 1476–1483, Long Beach, CA, 2017.

[23] Marek Petrik. *Optimization-based Approximate Dynamic Programming*. PhD thesis, University of Massachusetts, 2010.

[24] Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13:145–147, July 1972.

[25] Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41:247–261, April 1972.

[26] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2018.

[27] Csaba Szepesvári and William Smart. Interpolation-based Q-learning. In *Proceedings of the International Conference on Machine Learning (ICML-04)*, 2004.

[28] Gerald Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3):257–277, May 1992.

[29] John H. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42:674–690, 1996.

[30] Hado van Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems 23 (NIPS-10)*, pages 2613–2621, Vancouver, BC, 2010.

[31] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, September 1998.

[32] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML-16)*, 2016.

[33] Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989.

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

A Appendix: supplementary material

A.1 Example 1: Q-learning fixed point derivation

We first characterize the set of fixed points, $\hat{\theta} = (\hat{\theta}_1, \hat{\theta}_2)$, produced by online Q-learning (or QL). (Note that the initial parameters $\theta^{(0)}$ do not influence this analysis.) Using the ε-greedy behaviour policy $\pi_b = \varepsilon\mathsf{Greedy}$, any fixed point must satisfy two conditions:
(1) The behaviour policy must not change with each update; If $\pi_{\hat{\theta}}(s) = a$ then $\varepsilon\mathsf{Greedy}$ takes a with probability $1 - \varepsilon$ and any other action $a' \neq a$ with uniform probability.
(2) The expected update values for θ summed across all state-action pairs must be zero under the stationary visitation frequencies $\mu(s, a)$ of the behaviour policy.

The second condition imposes the following set of constraints,

$$\sum_{s,a} \mu(s,a)\phi(s,a)\Big(R(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q_{\hat{\theta}}(s',a') - Q_{\hat{\theta}}(s,a)\Big) = \mathbf{0}\ . \tag{3}$$

In Fig. 1, we first check if there exist a fixed point that corresponds to the optimal policy. The optimal policy takes a_1 in s_1 and a_2 in s_4 deriving expected value $R(s_1, a_1) + qR(s_4, a_2)$ if s_1 is the initial state. For linear function approximators, this implies $\theta_2 \geq 0.8\theta_1$ (choose a_1 over a_2 in s_1) and $-\theta_1 > \theta_2$ (choose a_2 over a_1 in s_4), which then implies $\theta_1 < 0$ (hence the policy takes a_1 in s_2 and a_2 in s_3). Under $\varepsilon\mathsf{Greedy}$, the stationary visitation frequencies μ is given in Table 1.

Table 1: $\mu(s, a)$ is the expected number of times action a is taken at state s in an episode under $\varepsilon\mathsf{Greedy}$ for the optimal (greedy) policy.

s, a	$\mu(s,a)$	s, a	$\mu(s,a)$
s_1, a_1	$1-\varepsilon$	s_3, a_1	ε^3
s_1, a_2	ε	s_3, a_2	$\varepsilon^2(1-\varepsilon)$
s_2, a_1	$\varepsilon(1-\varepsilon)$	s_4, a_1	$\varepsilon^3(1-\varepsilon) + \varepsilon(1-\varepsilon)q$
s_2, a_2	ε^2	s_4, a_2	$\varepsilon^2(1-\varepsilon)^2 + (1-\varepsilon)^2 q$

We can calculate the expected update at each state-action pair, i.e.

$$\Delta\theta(s,a) = \mu(s,a)\phi(s,a)\Big(R(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q_\theta(s',a') - Q_\theta(s,a)\Big)\ .$$

For example, at (s_1, a_1) we can transition to either a terminal state (with probability $1-q$) or s_4 (with probability q and after which the best action is $a' = a_2$ with bootstrapped value $-\theta_1$). Therefore at steady state the expected update at (s_1, a_1) is

$$[(1-\varepsilon)(1-q)(R(s_1,a_1) - \theta_2) + (1-\varepsilon)q(R(s_1,a_1) - \theta_1 - \theta_2)]\ \phi(s_1,a_1)\ .$$

Table 2 lists the expected updates for each state-action pair. If we sum the expected updates and set to $(0, 0)$ we get two equations with two unknowns, from which we solve for θ. The equation from the first component implies

$$\theta_1 = \frac{-(1-\varepsilon)^2(\varepsilon^2+q)R(s_4,a_2)}{(1-\varepsilon)^2(\varepsilon^2+q) + 0.64\varepsilon + 1.44\varepsilon^2}\ .$$

The equation from the second component implies

$$\theta_2 = \frac{(1-q)R(s_1,a_1) + qR(s_1,a_1) - q\theta_1}{1 + \varepsilon(\varepsilon^2+q)}\ .$$

If we let $R(s_1, a_1) = 0.3$, $R(s_4, a_2) = 2$ and $\varepsilon = 1/2$ then $\hat{\theta} \approx (-0.114, 0.861)$, contradicting the constraint that $\theta_2 < -\theta_1$ (i.e., a_2 is taken at s_4). Fig. 3 shows that for $R(s_1, a_1) = R(s_4, a_2) = 1$, $\theta_2 \not< -\theta_1$ for all $\varepsilon \in [0, 1/2]$ and again violating the assumed constraints. Therefore we conclude QL does not converge to the optimal greedy policy for these parameter configurations.

Now consider if QL may converge to the second best policy where $\pi_\theta(s_1) = a_1$ and $\pi_\theta(s_4) = a_1$ deriving expected value of $R(s_1, a_1)$ starting at state s_1. Say $\theta_1 < 0$. The corresponding state-action visitation frequencies at convergence is identical to Table 1 except $\mu(s_4, a_1) = (\varepsilon^2 + q)(1-\varepsilon)^2$ and $\mu(s_4, a_2) = \varepsilon(1-\varepsilon)(\varepsilon^2 + q)$. The expected updates are identical to Table 2 except at (s_1, a_1) it is $(0, (1-\varepsilon)(1-q)(R(s_1, a_1) - \theta_2) + (1-\varepsilon)qR(s_1, a_1))$, at (s_3, a_2) it is $(-\varepsilon^2(1-\varepsilon)(\theta_1 + \theta_2), 0)$, at (s_4, a_1) it is $(0, -(1-\varepsilon)^2(\varepsilon^2 + q)\theta_2)$ and at (s_4, a_2) it is $(-\varepsilon(1-\varepsilon)(\varepsilon^2 + q)(R(s_4, a_2) + \theta_1), 0)$. Again, we can solve these equations and get

$$\begin{aligned} \theta_2 &= \frac{-R(s_1, a_1)}{\varepsilon q + \varepsilon^3 - \varepsilon^2 - 1}, \\ \theta_1 &= \frac{-\varepsilon^2(1-\varepsilon)\theta_2 - (1-\varepsilon)^2(\varepsilon^2 + q)R(s_4, a_2)}{-(1-\varepsilon)^2(\varepsilon^2 + q) - \varepsilon^2(1-\varepsilon) - 1.44\varepsilon^2 - 0.64\varepsilon}. \end{aligned}$$

Plugging in $R(s_1, a_1) = 0.3$, $R(s_4, a_2) = 2$ and $\varepsilon = 1/2$ we have $\hat{\theta} \approx (-0.235, 0.279)$ which is a feasible solution. In particular, we empirically verified that starting with initial $\theta^{(0)} = (0, 0)$, QL converges to this solution. This second best policy is also a fixed point for $R(s_1, a_1) = R(s_4, a_2) = 1$ with any $\varepsilon \in [0, 1/2]$.

The delusion is caused by the backup at (s_2, a_2). Since we assume $\theta_1 < 0$ the bootstrapped future Q-value is $Q_\theta(s_3, a_2)$. But this is inconsistent: there is no θ taking a_2 at s_2 and a_2 at s_3. Such a backup increases θ_2 in order to propagate a higher value along two edges that are inconsistent (these edges can never belong to the same policy, and can pollute the value of the Q-learned policy). In fact, if we enforce consistency by backing up (s_3, a_1), the expected update at (s_2, a_2) reduces from $-1.44\varepsilon^2\theta_1$ to $-0.64\varepsilon^2\theta_1$. The consistent backup reduces two opposing effects: backup at (s_2, a_2) wants to increase θ_1 while backup at (s_3, a_2) wants to decrease θ_1—resulting in a compromise that corresponds to an inferior policy. When $R(s_1, a_1) = 0.3$, $R(s_4, a_2) = 2$, and $\varepsilon = 1/2$ this reduction in the expected update quantity implies QL (with a consistent backup) converges to the optimal policy with $\hat{\theta} \approx (-0.308, 0.282)$; whereas with an inconsistent backup it converges to the second best policy (and cannot converge to the optimal policy for any initial condition).

Table 2: Expected update $\Delta\theta(s, a)$ for each state-action pair under the optimal greedy policy.

s, a	Expected update $\Delta\theta(s, a)$
s_1, a_1	$(0, (1-\varepsilon)(1-q)(R(s_1, a_1) - \theta_2) + (1-\varepsilon)q(R(s_1, a_1) - \theta_1 - \theta_2))$
s_1, a_2	$(-0.64\varepsilon\theta_1, 0)$
s_2, a_1	$(0, 0)$
s_2, a_2	$(-1.44\varepsilon^2\theta_1, 0)$
s_3, a_1	$(0, 0)$
s_3, a_2	$(0, 0)$
s_4, a_1	$(0, -\varepsilon(1-\varepsilon)(\varepsilon^2 + q)\theta_2)$
s_4, a_2	$(-(1-\varepsilon)^2(\varepsilon^2 + q)(R(s_4, a_2) + \theta_1), 0)$

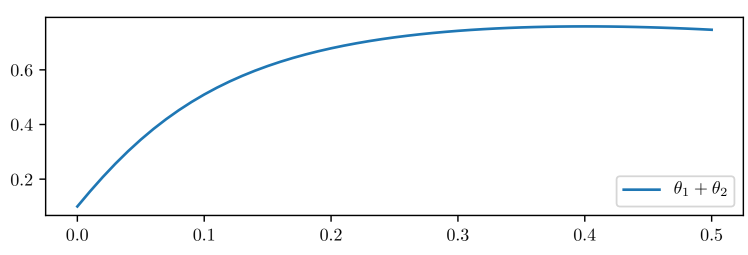

Figure 3: This shows $\theta_1 + \theta_2 > 0$, implying QL cannot converge to the optimal greedy policy for various ε exploration probabilities. Here, $R(s_1, a_1) = R(s_4, a_2) = 1$ and $q = 0.1$.

A.2 Divergence due to delusional bias

We show that delusional bias can actually lead to *divergence* of Q-learning with function approximation. In fact, a trivial example suffices. Consider a deterministic MDP with states $S = \{s_1, s_2\}$ and actions $A = \{a_1, a_2\}$, where from any state, action a_1 always transitions to s_1 and action a_2 always transitions to state s_2. Rewards are always 0.

Define a linear approximator for the Q-function by a single basis feature ϕ; i.e. we approximate $Q(s, a)$ by $\phi(s, a)\theta$ for some scalar θ. Let $Z = \sqrt{12 + 2\eta + \eta^2}$ for $\eta > 0$ arbitrarily close to 0 (η is used merely for tie breaking). Then define $\phi(s_1, a_1) = (1 + \eta)/Z$; $\phi(s_1, a_2) = 1/Z$; $\phi(s_2, a_1) = 1/Z$; and $\phi(s_2, a_2) = 3/Z$, which ensures $\|\phi\|_2 = 1$. Clearly, this Q-approximation severely restricts the set of expressible greedy policies: for $\theta > 0$ the greedy policy always stays at the current state; otherwise, for $\theta < 0$, the greedy policy always chooses to switch states. Interestingly, as in [1], this limited basis is still sufficient to express the optimal Q-function via $\theta = 0$ (there are no rewards).

We will show that the behaviour of approximate Q-learning with ε-greedy exploration can still diverge due to delusional bias; in particular that, after initializing to $\theta_0 = 1$, θ grows positively without bound. To do so, we examine the expected behaviour of the approximate Q-learning update under the stationary visitation frequencies. Note that, since $\theta > 0$ will hold throughout the analysis, the ε-greedy policy chooses to stay at the same state with probability $1 - \varepsilon$ and switches states with probability ε. Therefore, the stationary visitation frequencies, $\mu(s, a)$, is given by $\mu(s_1, a_1) = (1 - \varepsilon)/2$; $\mu(s_1, a_2) = \varepsilon/2$; $\mu(s_2, a_1) = \varepsilon/2$; and $\mu(s_2, a_2) = (1 - \varepsilon)/2$.

Consider the learning update, $\theta \leftarrow \theta + \alpha\Delta\theta$, where the *expected* update is given by

$$\mathbb{E}[\Delta\theta] = \sum_{s,a} \mu(s, a)\phi(s, a)\delta(s, a),$$

using the Q-learning temporal difference error, δ, given by

$$\delta(s, a) = \gamma \sum_{s'} p(s'|s, a) \max_{a'} \phi(s', a')\theta - \phi(s, a)\theta$$

(recall the rewards are 0). In this example, the temporal differences are $\delta(s_1, a_1) = -(1 - \gamma)\theta/Z$; $\delta(s_1, a_2) = (3\gamma - 1)\theta/Z$; $\delta(s_2, a_1) = -(1 - \gamma)\theta/Z$; and $\delta(s_2, a_2) = -3(1 - \gamma)\theta/Z$, in the limit when $\eta \to 0$. Observe that $\delta(s_1, a_2)$ demonstrates a large delusional bias in this case (whenever $\theta > 0$). In particular, the other δ values are small negative numbers (assuming $\gamma \approx 1$), while $\delta(s_1, a_2)$ is close to $2/Z$. The delusion occurs because the update through (s_1, a_2) thinks that a large future value can be obtained by switching to s_2, but the greedy policy allows no such switch. In particular, $\mathbb{E}[\Delta\theta]$ can be explicitly computed to be

$$\mathbb{E}[\Delta\theta] = \left(\frac{(5 - 3\varepsilon)\theta}{Z}\right)\gamma - \left(\frac{(5 - 4\varepsilon)\theta}{Z}\right),$$

which is bounded above zero for all $\theta \geq 1$ whenever $\delta > (5 - 4\varepsilon)/(5 - 3\varepsilon)$. Note that, since $\phi > 0$, δ is positive homogeneous in θ. Therefore, the expected value of θ_k is

$$\begin{aligned}
\mathbb{E}[\theta_k] &= \mathbb{E}[\theta_{k-1} + \alpha\, \mathbb{E}[\Delta\theta_{k-1}]] \\
&= \mathbb{E}[\theta_{k-1} + \alpha\theta_{k-1}\, \mathbb{E}[\Delta\theta_0]] \\
&= (1 + \alpha\, \mathbb{E}[\Delta\theta_0])\, \mathbb{E}[\theta_{k-1}],
\end{aligned}$$

leading to divergence w.p. 1 (see, e.g., [10, Chap.4]) since we have established $\mathbb{E}[\Delta\theta_0] > 0$ for some $\gamma < 1$.

A.3 Q-learning cyclic behaviour due to delusion

We show that delusion caused by the online Q-learning backup along an infeasible path leads to cycling of solutions and hence does not converge when the learning rate α_t^{sa} is lower bounded by $\alpha > 0$ (i.e. some schedule where α is the smallest learning rate). Consider Fig. 4. The path (s_1, a_2), (s_2, a_2) is infeasible: as it requires that $\theta_2 > 0$ (choosing a_2 over a_1 at s_1) and $\theta_2 < 0$ (taking a_2 at s_2). We assume $\gamma = 1$ for this episodic MDP.

There are four potential updates, each at a different (s, a)-pair. Any backup at (s_1, a_1) or (s_2, a_1) does not change the weight vector. Consider the first update at (s_2, a_2) at iteration k such that $\alpha_k = \alpha$. Assume $\theta^{(k-1)} = (b, c)$. A reward of $R(s_2, a_2) = 1/\sqrt{\alpha}$ is obtained, hence we have the update

$$\theta^{(k)} = (b, c) + \alpha \cdot \left(0, -\frac{1}{\sqrt{\alpha}}\right) \cdot \left(\frac{1}{\sqrt{\alpha}} + \gamma \cdot 0 - \left(-\frac{c}{\sqrt{\alpha}}\right)\right) = (b, -1) \ .$$

The next backup with non-zero updates occurs at (s_1, a_2) at step k' (since an update at (s_2, a_2) would not change θ and updates at (s_1, a_1) and (s_2, a_1) gets multiplied by all zero features), $\theta^{(k'-1)} = (b, -1)$ and

$$\theta^{(k')} = (b, -1) + \alpha \cdot \left(0, \frac{1}{\sqrt{\alpha}}\right) \cdot \left(0 + \gamma \cdot \frac{1}{\sqrt{\alpha}} - \left(-\frac{1}{\sqrt{\alpha}}\right)\right) = (b, 1) \ .$$

This shows the cyclic behaviour of Q-learning when the two jointly infeasible state-action pairs "undo" each others' updates. We can easily extend this small example to larger feature spaces with larger collections of infeasible state-action pairs.

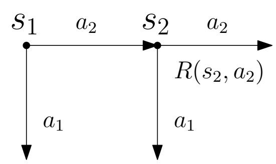

Figure 4: This two state MDP has $\phi(s_1, a_2) = (0, 1/\sqrt{\alpha})$, $\phi(s_2, a_2) = (0, -1/\sqrt{\alpha})$, $\phi(s_1, a_1) = \phi(s_2, a_1) = (0, 0)$. There is a single non-zero reward $R(s_2, a_2) = 1/\sqrt{\alpha}$. All transitions are deterministic and only one is non-terminal: $s_1, a_2 \to s_2$.

A.4 The discounting paradox

We first illustrate the discounting paradox with the simple MDP in Fig. 5. Delusional bias causes approximate QL to behave paradoxically: a policy trained with $\gamma = 1$ will be *worse* than a policy trained with $\gamma = 0$, even when evaluated non-myopically using $\gamma = 1$.[3]

We use a linear approximator

$$Q_\theta(s, a) = \theta_1 \phi(s) + \theta_2 \phi(a) + \theta_3, \tag{4}$$

with the feature embeddings $\phi(s_1) = 2$; $\phi(s_2) = 1$; $\phi(s_2') = 0$; $\phi(a_1) = -1$; and $\phi(a_2) = 1$. The two discounts $\gamma = 0$ and $\gamma = 1$ each give rise to different optimal parameters, inducing the corresponding greedy policies:

$$\hat{\theta}^{\gamma=0} = \mathsf{QL}(\gamma = 0, \theta^{(0)}, \varepsilon\mathsf{Greedy}), \qquad \hat{\pi}_0 = \mathsf{Greedy}(Q_{\hat{\theta}^{\gamma=0}}),$$
$$\hat{\theta}^{\gamma=1} = \mathsf{QL}(\gamma = 1, \theta'^{(0)}, \varepsilon\mathsf{Greedy}), \qquad \hat{\pi}_1 = \mathsf{Greedy}(Q_{\hat{\theta}^{\gamma=1}}).$$

For all $\varepsilon \leq 1/2$ we will show that $V^{\hat{\pi}_0}_{\gamma=1} > V^{\hat{\pi}_1}_{\gamma=1}$.

The (greedy) policy class representable by the linear approximator, $G(\Theta) = \{\mathsf{Greedy}(Q_\theta) : \theta \in \mathbb{R}^3\} = \{\pi_{a_1}, \pi_{a_2}\}$, is extremely limited: if $\theta_2 < 0$, the greedy policy always takes a_1 (policy π_{a_1}), while if $\theta_2 > 0$, it always takes a_2 (policy π_{a_2}). When evaluating these policies using $\gamma = 1$, we find that $V^{\pi_{a_2}}_{\gamma=1} = 2 > 2 - \delta = V^{\pi_{a_1}}_{\gamma=1}$, hence the optimal policy in $G(\Theta)$ is π_{a_2}. By contrast, the *unconstrained* optimal policy π^* takes a_1 in s_1 and a_2 in s_2. The paradox arises because, as we will see, the myopic learner ($\gamma = 0$) converges to the best representable policy $\hat{\pi}_0 = \pi_{a_2}$, whereas the non-myopic learner ($\gamma = 1$) converges to the worse policy $\hat{\pi}_1 = \pi_{a_1}$.

To prove this result, we characterize the fixed points, $\hat{\theta}^{\gamma=0}$ and $\hat{\theta}^{\gamma=1}$, produced by QL with $\gamma = 0$ and $\gamma = 1$, respectively. (Note that the initial parameters $\theta^{(0)}, \theta'^{(0)}$ do not influence this analysis.) Using

[3] Note that we use these discount rates of 0 and 1 to illustrate how extreme the paradox can be—there is nothing intrinsic to the paradox that depends on the use of the purely myopic variant versus the infinite-horizon variant.

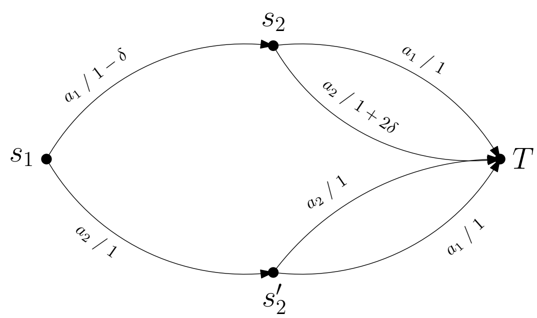

Figure 5: A deterministic MDP starting at state s_1 and terminating at T, $Q_\theta(T, a) = 0$. Directed edges are transitions of the form a/r where a is the action taken and r the reward. Parameter $\delta > 0$.

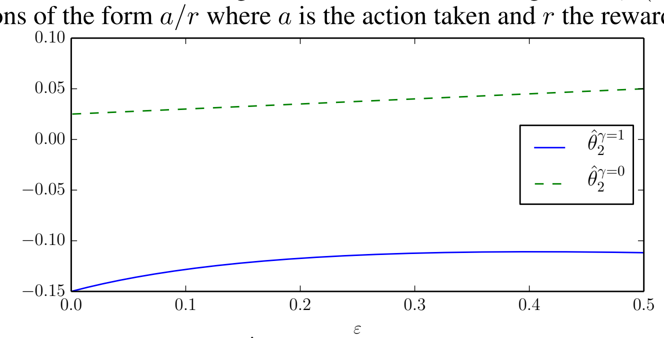

Figure 6: Fixed points $\hat{\theta}_2^\gamma$ when training with $\gamma = 0, 1$; $\delta = 0.1$.

the behaviour policy $\pi_b = \varepsilon\mathsf{Greedy}$, any fixed point $\hat{\theta}$ of QL must satisfy two conditions (see also Appendix A.1):

(1) The behaviour policy must not change with each update; if $\mathrm{sgn}(\hat{\theta}_2) = -1$ (resp. $+1$) then $\varepsilon\mathsf{Greedy}$ takes a_1 with probability $1 - \varepsilon$ (resp. a_2).

(2) The expected update values for θ must be zero under the stationary visitation frequencies $\mu(s, a)$ of the behaviour policy.[4]

The set of fixed points is entirely characterized by the sign of $\hat{\theta}_2$. For fixed points where $\hat{\theta}_2 < 0$ the second condition imposes the following set of constraints,

$$\sum_{s,a} \mu_{a_1}(s,a)\phi(s,a)\Big(R(s,a) + (\gamma s' - s)\hat{\theta}_1 - (\gamma + a)\hat{\theta}_2 + (\gamma - 1)\hat{\theta}_3\Big) = \mathbf{0}\ ; \quad \hat{\theta}_2 < 0\ , \tag{5}$$

where s' follows s, a. Similarly, when $\hat{\theta}_2 > 0$, we have

$$\sum_{s,a} \mu_{a_2}(s,a)\phi(s,a)\Big(R(s,a) + (\gamma s' - s)\hat{\theta}_1 + (\gamma - a)\hat{\theta}_2 + (\gamma - 1)\hat{\theta}_3\Big) = \mathbf{0}\ ; \quad \hat{\theta}_2 > 0\ . \tag{6}$$

Note that there is no fixed point where $\hat{\theta}_2 = 0$.

If we set $\delta = 0.1$, $\varepsilon = 0.1$ and $\gamma = 1$ and solve for (5), we obtain $\hat{\theta}^{\gamma=1} \approx (0.793, -0.128, 0.265)$, whereas there is no solution for (6). Alternatively, by setting $\gamma = 0$, there is a solution to (6) given by $\hat{\theta}^{\gamma=0} \approx (-0.005, 0.03, 0.986)$, but there is no solution to (5). (Fig. 6 shows the entire family of fixed points as ε varies.) Thus, $\mathsf{Greedy}(Q_{\hat{\theta}^{\gamma=1}})$ (the non-myopically trained policy) always takes action a_1, whereas $\mathsf{Greedy}(Q_{\hat{\theta}^{\gamma=0}})$ (the myopically trained policy) always takes action a_2. When evaluating these policies under $\gamma = 1$, the myopic policy has value 2, while paradoxically the non-myopic policy has lower value $2 - \delta$.

This discounting paradox arises precisely because Q-learning fails to account for the fact that the class of (greedy) policies admitted by the simple linear approximator is extremely limited. In particular, under non-myopic ($\gamma = 1$) training, QL "believes" that taking a_1 in s_1 results in a better long-term reward, which is in fact true if we can execute the *unconstrained* optimal policy: take a_1, then a_2 (i.e., the top path), receiving a total reward of $2 + \delta$. However, this policy is infeasible, since the

[4] If $\hat{\theta}_2 < 0$, the greedy policy is π_{a_1}, which induces the stationary distribution $\mu_{a_1}(s_1, a_1) = 1 - \varepsilon$; $\mu_{a_1}(s_1, a_2) = \varepsilon$; $\mu_{a_1}(s_2, a_1) = (1-\varepsilon)^2$; $\mu_{a_1}(s_2, a_2) = \varepsilon(1-\varepsilon)$; $\mu_{a_1}(s_2', a_1) = \varepsilon(1-\varepsilon)$; and $\mu_{a_1}(s_2', a_2) = \varepsilon^2$. When $\hat{\theta}_2 > 0$, the greedy policy π_{a_2} induces a similar distribution, but with ε and $1 - \varepsilon$ switched.

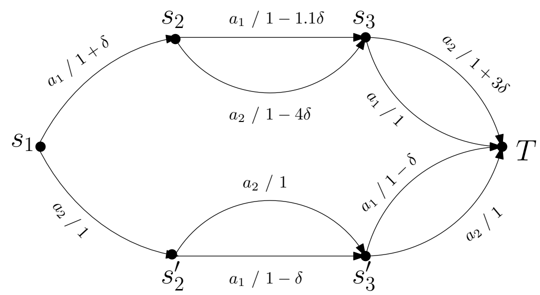

Figure 7: An MDP similar to Fig. 5; non-myopic is much worse.

policy class $G(\Theta)$ consists of greedy policies that can only take a_1 or a_2 at all states: if a_1 is taken at s_1, it must also be taken at s_2.

More specifically, the paradox emerges because, in the temporal difference, the $\gamma \max_{a' \in A} Q(s', a')$ term maximizes over all possible actions, without regard to actions that may have been taken to reach s'. If reaching s' requires taking some prior action that renders a specific a' infeasible at s' w.r.t. $G(\Theta)$, then Q-learning "deludes" itself, believing it can achieve future values that are, in fact, infeasible under the constrained policy class $G(\Theta)$.

The example above illustrates one instance of the discounting paradox, showing that training using the larger correct discount factor can give rise to a slightly worse policy than training using an "incorrect" smaller discount. We now consider how bad the performance loss can be and also show that the paradox can occur in the opposite direction (smaller target discount outperformed by larger incorrect discount).

The two induced greedy policies above, π_{a_1} and π_{a_2}, have a difference in value δ (see Fig. 5) when evaluated with $\gamma = 1$; δ must be relatively small for the paradox to arise (e.g., if $\varepsilon = 0.1$, $\delta \lessapprox 0.23$). A different MDP (see Fig. 7) induces a delusional bias in which the non-myopic Q-learned policy can be *arbitrarily worse* (in the competitive-ratio sense) than the myopic Q-learned policy. To show this we use the same linear approximator (4) and action embeddings, but new feature embeddings $\phi(s_1) = 3$, $\phi(s_2) = 2$, $\phi(s'_2) = 1.9999$, $\phi(s_3) = 1.1$, and $\phi(s'_3) = 1.0999$. When evaluating with $\gamma = 1$, the policy π_{a_2} (with value 3) has an advantage of 0.1δ over π_{a_1} (with value $3 - 0.1\delta$). Assume the behaviour policy is εGreedy with $\varepsilon = 0.2$ (other ε also work), whose distribution $\mu(s, a)$ can be computed as in Footnote 4.

For Q-learning with $\gamma = 1$, we solve the constraints in (5) to find fixed points where $\hat{\theta}_2 < 0$. The system of equations has the form $\mathbf{A}\hat{\theta} = \mathbf{b}$; solving for $\hat{\theta}$ gives:

$$\hat{\theta}^{\gamma=1} = \mathbf{A}^{-1}\mathbf{b} \approx \begin{bmatrix} 1.03320 + 0.09723\delta \\ -0.00002 - 0.08667\delta \\ -0.09960 - 0.65835\delta \end{bmatrix} .$$

This implies, for any $\delta \geq 0$, the Q-learned greedy policy is π_{a_1}. Similarly, fixed points with $\gamma = 0$, where $\hat{\theta}_2 > 0$, we have $\hat{\theta}_2^{\gamma=0} \approx 0.10333\delta$. This implies the Q-learned greedy policy is π_{a_2} for any $\delta > 0$. In particular, we have that

$$\lim_{\delta \to 30^-} \frac{V_{\gamma=1}^{\mathsf{Greedy}(\hat{\theta}^{\gamma=1})}}{V_{\gamma=1}^{\mathsf{Greedy}(\hat{\theta}^{\gamma=0})}} = \frac{3 - 0.1(30)}{3} = 0 .$$

Therefore the non-myopic policy may be arbitrarily worse than the myopic policy. (Other fixed points, where $\hat{\theta}_2^{\gamma=1} > 0, \hat{\theta}_2^{\gamma=0} < 0$ may be reached depending on the initial $\theta^{(0)}$.)

The paradox can also arise in the "opposite" direction, when evaluating policies purely myopically with $\gamma = 0$: the myopic Q-learned policy can be arbitrarily worse than a non-myopic policy. As shown above, for any $\delta > 0$, the non-myopic policy chooses a_1 at s_1, thus its value is $1 + \delta$ (since subsequent rewards are fully discounted) whereas the myopic policy chooses a_2 receiving a value of 1. Thus, the non-myopic policy has a value advantage of δ, which translates into an arbitrarily large improvement ratio over the myopic policy: $\lim_{\delta \to \infty}(1 + \delta)/1 = \infty$.

A.5 Comparisons to double Q-learning

The maximization bias in Q-learning [30] is an over-estimation bias of the bootstrap term $\max_{a' \in A} \hat{Q}(s', a')$ in Q-updates. If the action space is large, and there are not enough transition examples to confidently estimate the true $Q(s', a')$, then the variance of each $\hat{Q}(s, a)$ for any a is large. Taking the max of random variables with high variance will over-estimate, even if $\max_{a' \in A} \mathbb{E}[\hat{Q}(s, a)] = \max_{a' \in A} Q(s, a)$.

Double Q-learning aim to fix the maximization bias by randomly choosing to update one of two Q-functions, Q_1 and Q_2. For any given transition (s, a, r, s), if Q_1 is chosen to be updated with probability $1/2$, then the update becomes

$$Q_1(s, a) \leftarrow Q_1(s, a) + \alpha(r + \gamma Q_2(\operatorname*{argmax}_{a' \in A} Q_1(s', a') - Q_1(s, a))\nabla Q_1(s, a)$$

If Q_2 is randomly chosen, a similar update is applied by switching the roles of Q_1 and Q_2. A common behaviour policy is $\varepsilon\mathsf{Greedy}(Q_1 + Q_2)$. The idea behind this update is to reduce bias by having Q_2, trained with different transitions, evaluate the max action of Q_1.

This type of maximization bias is due to lack of samples, which causes large variance that biases the max value. The MDP counter-example in Fig. 5 is deterministic, both in the rewards and transitions, so in fact there is no maximization bias. To make this concrete, we can derive the double Q-learning fixed points Q_θ and $Q_{\theta'}$ of Fig. 5. Using similar reasoning as in Eq. 5, we can write the constraints for fixed points where $\theta_2 + \theta_2' < 0$ (resp. > 0)—i.e. Greedy always selects a_1 (resp. always a_2).

$$\sum_{s,a} \mu(s, a)\phi(s, a)(R(s, a) + \gamma(\theta_1 s' + \theta_2 a' + \theta_3) - \theta_1' s - \theta_2' a - \theta_3') = \mathbf{0} \tag{7}$$

$$\sum_{s,a} \mu(s, a)\phi(s, a)(R(s, a) + \gamma(\theta_1' s' + \theta_2' a'' + \theta_3') - \theta_1 s - \theta_2 a - \theta_3) = \mathbf{0} \tag{8}$$

The weighting $\mu(s, a) = \frac{1}{2}\mu_b(s, a)$ where $\mu_b(s, a)$ corresponds to the behaviour policy. If $\theta_2 + \theta_2' < 0$, $\varepsilon\mathsf{Greedy}$ prefers a_1, and there are one of three possibilities. First, $a' = a'' = a_1$ implying $\theta_2, \theta_2' < 0$; second, $(a', a'') = (a_1, a_2)$ implying $\theta_2 > 0, \theta_2' < 0$; third, $(a', a'') = (a_2, a_1)$ implying $\theta_2 < 0, \theta_2' > 0$. We can substitute all three cases into Eqs. 7, 8, solve the system of equations, and check if θ, θ' satisfy the constraints. One solution is a fixed point of regular Q-learning where $Q_\theta = Q_\theta'$. For both $\gamma = 0, 1$, we check for other possible solutions with the MDP of Fig 5, they do not exist. Fixed points when $\theta_2 + \theta_2' > 0$ can be found similarly. Thus double Q-learning does not mitigate the delusional bias issue.

A.6 Concepts and proofs for PCVI and PCQL

In this section, we elaborate on various definitions and provide proofs of the results in Section 4.

We formally define functions and binary operators acting on partitions of Θ.

Definition 4. *Let $\mathcal{X}$ be a set, a* finite partition *of $\mathcal{X}$ is any set of non-empty subsets $P = \{X_1, \dots, X_k\}$ such that $X_1 \cup \cdots \cup X_k = \mathcal{X}$ and $X_i \cap X_j = \emptyset$, for all $i \neq j$. We call any $X_i \in P$ a* cell. *A partition P' of $\mathcal{X}$ is a* refinement *of P if for all $X' \in P'$ there exists a $X \in P$ such that $X' \subseteq X$. Let $\mathcal{P}(\mathcal{X})$ denote the set of all finite partitions of $\mathcal{X}$.*

Definition 5. *Let $P \in \mathcal{P}(\mathcal{X})$. A mapping $h : P \to \mathbb{R}$ is called a* function of partition *P. Let $\mathcal{H} = \{h : P \to \mathbb{R} \mid P \in \mathcal{P}(\mathcal{X})\}$ be the set of all such functions of partitions. Let $h_1, h_2 \in \mathcal{H}$, an* intersection sum *is a binary operator $h = h_1 \oplus h_2$ defined by*

$$h(X_1 \cap X_2) = h_1(X_1) + h_2(X_2), \quad \forall X_1 \in \mathsf{dom}(h_1), X_2 \in \mathsf{dom}(h_2), X_1 \cap X_2 \neq \emptyset$$

where $\mathsf{dom}(\cdot)$ is the domain of a function (in this case a partition of $\mathcal{X}$). We say h_1 is a refinement *of h_2 if partition $\mathsf{dom}(h_1)$ is a refinement of $\mathsf{dom}(h_2)$.*

Note there is at most a quadratic blowup: $|\mathsf{dom}(h)| \leq |\mathsf{dom}(h_1)| \cdot |\mathsf{dom}(h_2)|$. The tuple $(\mathcal{H}, \oplus)$ is almost an abelian group, but without the inverse element property.

Proposition 6. *The following properties hold for the intersection sum.*

- *(Identity) Let $e = X \mapsto 0$, then $h \oplus e = e \oplus h = h$ for all $h \in \mathcal{H}$.*

- *(Refinement) For all $h_1, h_2 \in \mathcal{H}$, we have: (i) $h_1 \oplus h_2$ is a refinement of h_1 and of h_2. (ii) if h_1' and h_2' is a refinement of h_1 and h_2, respectively, then $h_1' \oplus h_2'$ is a refinement of $h_1 \oplus h_2$.*
- *(Closure) $h_1 \oplus h_2 \in \mathcal{H}$, for all $h_1, h_2 \in \mathcal{H}$*
- *(Commutative) $h_1 \oplus h_2 = h_2 \oplus h_1$, for all $h_1, h_2 \in \mathcal{H}$*
- *(Associative) $(h_1 \oplus h_2) \oplus h_3 = h_1 \oplus (h_2 \oplus h_3)$, for all $h_1, h_2, h_3 \in \mathcal{H}$*

Proof. The identity element e property follows trivially. Refinement (i) follows from the fact that for any two partitions $P_1, P_2 \in \mathcal{P}(X)$, say $P_1 = \mathsf{dom}(h_1)$ and $P_2 = \mathsf{dom}(h_2)$, the set $P = \mathsf{dom}(h_1 \oplus h_2) = \{X_1 \cap X_2 \mid X_1 \in P_1, X_2 \in P_2, X_1 \cap X_2 \neq \emptyset\}$ is also a partition where $X_1 \cap X_2 \in P$ is a subset of both $X_1 \in P_1$ and $X_2 \in P_2$. Refinement (ii) is straightforward, let $X' \in \mathsf{dom}(h_1')$ and $Y' \in \mathsf{dom}(h_2')$ where there is a non-empty intersection. By definition, $X' \subseteq X \in \mathsf{dom}(h_1)$ and $Y' \subseteq Y \in \mathsf{dom}(h_2)$. We have $X' \cap Y' \in \mathsf{dom}(h_1' \oplus h_2')$ and $(X' \cap Y') \subseteq (X \cap Y)$, so refinement (ii) holds. Commutative and associative properties essentially follow from that of the corresponding properties of set intersection and the addition operators. Closure follows from the refinement property since $h_1 \oplus h_2$ is a mapping whose domain is the refined (finite) partition. □

Assumption 7. *We have access to an oracle* Witness *where for any $X \subseteq \Theta$ defined by a conjunction of a collection of state to action constraints, outputs $\emptyset$ if X is an inconsistent set of constraints, and outputs any witness $\theta \in X$ otherwise.*

Theorem 1. *PCVI (Alg. 1) has the following guarantees:*

(a) (Convergence and correctness) The Q *function converges and, for each $s \in S, a \in A$, and any $\theta \in \Theta$: there is a unique $X \in \mathsf{dom}(\mathtt{Q}[sa])$ s.t. $\theta \in X$ and*

$$Q^{\pi_\theta}(s,a) = \mathtt{Q}[sa](X). \tag{9}$$

(b) (Optimality and Non-delusion) Given initial state s_0, π_{θ^*} is an optimal policy within $G(\Theta)$ and q^* is its value.

(c) (Runtime bound) Assume $\oplus$ and non-emptiness checks (lines 6 and 7) have access to Witness*. Let*

$$\mathcal{G} = \{g_\theta(s,a,a') := \mathbf{1}[f_\theta(s,a) - f_\theta(s,a') > 0], \forall s, a \neq a' \mid \theta \in \Theta\}, \tag{10}$$

where $\mathbf{1}[\cdot]$ is the indicator function. Then each iteration of Alg. 1 runs in time $O(nm \cdot [\binom{m}{2} n]^{2\,\mathsf{VCDim}(\mathcal{G})}(m-1)w)$ where $\mathsf{VCDim}(\cdot)$ is the VC-dimension [31] of a set of boolean-valued functions, and w is the worst-case running time of the oracle called on at most nm state-action constraints. Combined with Part (a), if $\mathsf{VCDim}(\mathcal{G})$ is finite then Q *converges in time polynomial in n, m, w.*

Corollary 8. *Let $\phi : S \times A \to \mathbb{R}^d$ be a vector representation of state-action pairs. Define*

- *$\mathcal{F}_{\text{linear}} = \{f_\theta(s,a) = \theta^T \phi(s,a) + \theta_0 \mid \theta \in \mathbb{R}^d, \theta_0 \in \mathbb{R}\}$ and*
- *$\mathcal{F}_{\text{DNN}}$ the class of real-valued ReLU neural networks with input $\phi(s,a)$, identity output activation, W the number of weight parameters, and L the number of layers.*

Let $\mathcal{G}_{\text{linear}}$ and $\mathcal{G}_{\text{DNN}}$ be the corresponding boolean-valued function class as in Eqn. 10. Then

- *$\mathsf{VCDim}(\mathcal{G}_{\text{linear}}) = d$ and*
- *$\mathsf{VCDim}(\mathcal{G}_{\text{DNN}}) = O(WL \log W)$.*

Furthermore, the Witness *oracle can be implemented in polynomial time for $\mathcal{F}_{\text{linear}}$ but is NP-hard for neural networks. Therefore Alg. 1 runs in polynomial time for linear greedy policies and runs in polynomial time as a function of the number of oracle calls for deep Q-network (DQN) greedy policies.*

Proof. Let $g_\theta \in \mathcal{G}_{\text{linear}}$ then $g_\theta(s,a,a') = \mathbf{1}[\theta^T(\phi(s,a) - \phi(s,a')) > 0]$, these linear functions have VC-dimension d. For any $f_\theta \in \mathcal{F}_{\text{DNN}}$ we construct a ReLU network g_θ as follows. The first input $\phi(s,a)$ goes through the network f_θ with identity output activation and the second input $\phi(s,a')$ goes through the same network, they are then combined with the difference $f_\theta(s,a) - f_\theta(s,a')$ before being passed through the $\mathbf{1}[\cdot]$ output activation function. Such as network has $2W$ number of weight parameters (with redundancy) and the same L number of layers. Then the VC-dimension of this network follows from bounds given in [2].

The oracle can be implemented for linear policies by formulating it as set of linear inequalities and solving with linear programming methods. For neural networks, the problem of deciding if a training set can be correctly classified is a classical NP-hardness result [7] and can be reduced to this oracle problem by converting the training data into state to action constraints. $\square$

Before proving the main theorem, we first show that the number of unique policies is polynomial in n when the VC-dimension is finite. This implies a bound on the blowup in the number of cells under the $\oplus$ operator.

Proposition 9. *Let $\mathcal{G}$ be defined as in Eqn. 10, then we have*

$$|G(\Theta)| \leq \sum_{i=0}^{\mathsf{VCDim}(\mathcal{G})} \binom{\binom{m}{2}n}{i} = O\left(\left[\binom{m}{2}n\right]^{\mathsf{VCDim}(\mathcal{G})}\right).$$

Proof. We construct a one-to-one mapping from functions in $G(\Theta)$ to functions in $\mathcal{G}$. Let $\pi_\theta \in G(\Theta)$. Suppose $\pi_\theta(s) = a$ for some state s. This implies $f_\theta(s,a) > f_\theta(s,a')$ for all $a' \neq a$, or equivalently $\mathbf{1}[f_\theta(s,a) - f_\theta(s,a') > 0] = 1$ for all $a' \neq a$. The converse is also true by definition. Thus the mapping $\pi_\theta \mapsto g_\theta$ is one-to-one, but not necessarily onto since g_θ is sensitive to all pairwise comparisons of a and a'. For ties we can assume both f and g selects the action with smallest index in A). The Sauer-Shelah Lemma [24, 25] gives us the bound $|\mathcal{G}| \leq \sum_{i=0}^{\mathsf{VCDim}(\mathcal{G})} \binom{|\mathsf{dom}(\mathcal{G})|}{i}$ where $|\mathsf{dom}(\mathcal{G})| = |S \times A \times A - \{(a,a) \mid a \in A\}| = \binom{m}{2}n.$ $\square$

Proof of Theorem 1. **Part (a):** first, let us argue that for any state-action pair s, a, these two conditions hold when executing the algorithm,

(1) $\mathsf{dom}(\mathtt{Q}[sa])$ is always a partition of Θ and

(2) $\mathsf{dom}(\mathtt{ConQ}[sa])$ is always a partition of $[s \mapsto a]$.

Note that the initialization in lines 1 and 2 satisfy these two conditions. Consider any iteration of the algorithm. We have that $\mathsf{dom}(\mathtt{ConQ}[s])$ is a partition of Θ since $\mathsf{dom}(\mathtt{ConQ}[s]) = \dot{\bigcup}_{a\in A}\mathsf{dom}(\mathtt{ConQ}[sa])$ and each $\mathsf{dom}(\mathtt{ConQ}[sa])$ is a partition of $[s \mapsto a]$ (this is the invariant condition in the loop). Hence $\mathtt{ConQ}[s]$ is a function of a partition of Θ. Line 6 is an update that is well-defined and results in $\mathsf{dom}(\mathtt{Q}[sa]) \in \mathcal{P}(\Theta)$, this is due to the commutative, associative and closure properties of the intersection sum (see Prop. 6). Thus condition (1) is satisfied. The update in line (7) ensures that $\mathsf{dom}(\mathtt{ConQ}[sa])$ has no empty sets and that each set Z in its domain comes from $\mathsf{dom}(\mathtt{Q}[sa])$ but with the constraint that $\pi_\theta(s) = a$. Thus condition (2) is satisfied.

To show convergence of $\mathtt{Q}$, we first show the partitions $\mathsf{dom}(\mathtt{ConQ}[s]) \in \mathcal{P}(\Theta)$ eventually converges—i.e. they do not change. Since partitions in $\mathtt{Q}$ is derived from partitions in $\mathtt{ConQ}$, then $\mathtt{Q}$ would also converge. Once partitions converge, we show that the backups in line 6 contain value iteration updates for policies within each cell. An application of standard convergence rates for value iteration gives us the desired results.

We first show that after each iteration of the inner loop, $\mathtt{ConQ}[s]$ is a refinement of the old $\mathtt{ConQ}[s]$ of a previous iteration. Let $\mathtt{Q}^{(i)}$, $\mathtt{ConQ}^{(i)}$, denote the tables at iteration i of the outer loop—that is, after executing i full passes of the inner loop. We claim that $\mathtt{ConQ}^{(i)}[s]$ is a refinement of $\mathtt{ConQ}^{(i-1)}[s]$ for all s. We prove this by induction on i. Base case: $i = 1$; $\mathtt{ConQ}^{(0)}[s]$ are partitions of the form $\{[s \mapsto a_1], \ldots, [s \mapsto a_m]\}$. Line 7 ensures $\mathtt{ConQ}^{(1)}[sa]$ is a refinement of $[s \mapsto a]$ since $\mathtt{Q}[sa]$ is a partition of Θ (shown above). Since this is true for any a then $\mathtt{ConQ}^{(1)}[s]$ is a refinement of $\mathtt{ConQ}^{(0)}[s]$. For $i > 1$, line 6 ensures that $\mathsf{dom}(\mathtt{Q}^{(i)}[sa]) = \mathsf{dom}(\bigoplus_{s'} \mathtt{ConQ}^{(i-1)}[s'])$ and $\mathsf{dom}(\mathtt{Q}^{(i-1)}[sa]) = \mathsf{dom}(\bigoplus_{s'} \mathtt{ConQ}^{(i-2)}[s'])$. By inductive hypothesis $\mathtt{ConQ}^{(i-1)}[s']$ is a refinement of $\mathtt{ConQ}^{(i-2)}[s']$. Therefore by Prop. 6's refinement property, $\mathtt{Q}^{(i)}[sa]$ is a refinement of $\mathtt{Q}^{(i-1)}[sa]$. By line 7 it follows that $\mathtt{ConQ}^{(i)}[sa]$ is a refinement of $\mathtt{ConQ}^{(i-1)}[sa]$. Finally $\mathtt{ConQ}^{(i)}[s]$ is the combination of all such relevant refinements, resulting in a refinement of $\mathtt{ConQ}^{(i-1)}[s]$. This ends the induction proof.

Thus a full pass of the inner loop may result in new cells for $\texttt{ConQ}^{(i)}[s]$ (part of a refinement) but never reduce the number of cells (i.e. never be an "anti-refinement"). If no new cells in $\texttt{ConQ}^{(i)}[s]$ are introduced in a full pass of the inner loop, i.e. $\mathsf{dom}(\texttt{ConQ}^{(i)}[s]) = \mathsf{dom}(\texttt{ConQ}^{(i-1)}[s])$ for all s, then $\mathsf{dom}(\texttt{Q})$ (and hence $\mathsf{dom}(\texttt{ConQ}[s])$) has converged. To see this, note that $\mathsf{dom}(\texttt{Q}^{(i+1)}[sa]) = \mathsf{dom}(\bigoplus_{s'} \texttt{ConQ}^{(i)}[s']) = \mathsf{dom}(\bigoplus_{s'} \texttt{ConQ}^{(i-1)}[s']) = \mathsf{dom}(\texttt{ConQ}^{(i)}[s])$. This implies $\mathsf{dom}(\texttt{ConQ}^{(i+1)}[s]) = \mathsf{dom}(\texttt{ConQ}^{(i)}[s])$ for all s. Hence in all subsequent iterations the partitions in Q and ConQ does not change, i.e. it converges. The maximum number of full passes of the inner loop before no new cells are introduced in a full inner pass is the maximum total number of cells in $\texttt{ConQ}[s]$ summed across all states (in the worst case when each full pass of inner loop results in only one new cell), since this maximum total is bounded, Q converges eventually.

Assume that cells in Q and ConQ has converged. Let $\theta \in \Theta$, now we will show the update in line 6 contains the update

$$Q^{\pi_\theta}(s,a) \leftarrow R_{sa} + \gamma \sum_{s' \in S} p(s' \mid s,a) Q^{\pi_\theta}(s', \pi(s')).$$

This is the on-policy value iteration update, which converges to the Q-function of π_θ with enough updates, thus Eqn. (9) would hold. Fix s, a. For any next state s', there exists a $X_{s'} \in \mathsf{dom}(\texttt{ConQ}[s'])$ such that $\theta \in X_{s'}$ (as we've established above that $\mathsf{dom}(\texttt{ConQ}[s']) \in \mathcal{P}(\Theta)$). Moreover, let $Y = \bigcap_{s' \in S} X_{s'}$, then by definition $\theta \in Y \in \mathsf{dom}(\texttt{Q}[sa])$. Thus the update in line 6 has the intersection sum where sets $X_{s'}$, for all s', are combined through an intersection, that is, it contains the update

$$\texttt{Q}[sa](Y) \leftarrow R_{sa} + \gamma \sum_{s' \in S} p(s' \mid s,a) \texttt{ConQ}[s'](X_{s'}),$$

where $\texttt{ConQ}[s'](X_{s'})$ stores the backed-up Q-value of θ in state s' performing action $\pi_\theta(s')$, the consistent action for θ. Line 7 ensures that only $\texttt{ConQ}[s\ \pi_\theta(s)]$ contains the cell $Z' = Y' \cap [s \mapsto \pi_\theta(s)]$ containing parameter θ (all other $\texttt{ConQ}[sa']$ does not contain a cell that contains θ, for $a' \neq \pi_\theta(s)$). Thus, only $\texttt{ConQ}[s\ \pi_\theta(s)](Z')$ is updated with $\texttt{Q}[s\ \pi_\theta(s)](Y')$ which is consistent with θ's action in state s. Hence, we've established that for any π_θ its corresponding Q-values is updated at every iteration and stored in Q while only the consistent Q-values are stored in ConQ.

Part (b): by construction and from above results, $\mathsf{dom}(\texttt{ConQ}[s_0]) \in \mathcal{P}(\Theta)$. By Part (a), for any policy π_θ, there exists a X_θ such that the policy has value $\texttt{Q}[s_0\ \pi_\theta(s_0)](X_\theta) = \texttt{ConQ}[s_0\ \pi_\theta(s_0)](X_\theta \cap [s_0 \mapsto \pi_\theta(s_0)]) = \texttt{ConQ}[s_0](X_\theta \cap [s_0 \mapsto \pi_\theta(s_0)])$ when starting from s_0. Thus $q^* = \texttt{ConQ}[s_0][X^*] \geq \texttt{ConQ}[s_0][X_\theta]$ is the largest Q-value of any policy in $G(\Theta)$, upon convergence of Q. The oracle returns a particular witness in X^* whose greedy policy attains value q^*.

Part (c): line 6 dominates the running time. There are $m-1$ applications of $\oplus$ operator, which is implemented by intersecting all pairs of cells from two partitions. There is a call to Witness to determine if the intersection of any pair of cells results in an inconsistent set of constraints, taking time at most w. Prop. 9 upper bounds the number of cells in a partition, thus we require $O([\binom{m}{2}n]^{\mathsf{VCDim}\,\mathcal{G}}(m-1)w)$ time for line 6, which is executed nm times within the inner loop. The number of iterations until $\mathsf{dom}(\texttt{Q}[s])$ converges is the maximum number of cells in $\texttt{ConQ}[s]$ summed across all states s (shown in proof of Part (a)). For finite VC-dimension, this summed total is polynomial. Once the partitions of $\texttt{Q}[s]$ converges, standard results concerning polynomial time convergence of Q-values within each cell apply. □

A.7 PCVI example

We walk through the steps of the PCVI Algorithm for the example MDP in Fig. 1 and show how it computes the optimal admissible policy. We assume the same feature representation as in the discussion of Sec. 3.1; see Table 3 for the features and rewards.

Recall that the optimal admissible policy has parameters $\theta^* = (-2, 0.5)$, and its greedy policy π_{θ^*} selects a_1 at s_1 and a_2 at s_4, giving expected value of 0.5 at initial state s_1. We walk through each step of PCVL below. Table 4 describes the critical data structures used during PCVL updates.

Table 3: Features and rewards for MDP in Fig. 1. Transitions are as in Fig. 1 with a stochastic transition at s_1, a_1 which goes to s_4 with probability 0.1 and terminates with probability 0.9. Initial state is s_1 and $\gamma = 1$.

s, a	$\phi(s, a)$	$R(s, a)$	s, a	$\phi(s, a)$	$R(s, a)$
s_1, a_1	$(0, 1)$	0.3	s_3, a_1	$(0, 0)$	0
s_1, a_2	$(0.8, 0)$	0	s_3, a_2	$(-1, 0)$	0
s_2, a_1	$(0, 0)$	0	s_4, a_1	$(0, 1)$	0
s_2, a_2	$(0.8, 0)$	0	s_4, a_2	$(-1, 0)$	2

Table 4: Explanation of critical data structures used in PCVL algorithm.

Data structure	**Description**
$\texttt{Q}[sa]$	A table mapping an element (set) X of a partition of Θ to $\mathbb{R}$. $\texttt{Q}[sa](X)$ is the Q-value of taking action a at s and then following a greedy policy parameterized by $\theta \in X$. It *may be* that $\pi_\theta(s) \neq a$.
$\texttt{ConQ}[sa]$	A table mapping an element (set) X of a partition of $[s \to a] = \{\theta \in \Theta : \pi_\theta(s) = a\}$ to $\mathbb{R}$. $\texttt{ConQ}[sa](X)$ is the Q-value of taking action a at s and then following a greedy policy parameterized by $\theta \in X$. It *must be* that $\pi_\theta(s) = a$. $\texttt{ConQ}$ is short for consistent Q-values.
$\texttt{ConQ}[s]$	A table formed from concatenating tables $\texttt{ConQ}[sa]$ for all a at s. The domain of $\texttt{ConQ}[s]$ is a partition of Θ.

Initialization. First initialize $\texttt{Q}$ and $\texttt{ConQ}$ tables for all $i \in \{1, 2, 3, 4\}$ and $j \in \{1, 2\}$.

$$\texttt{Q}[s_i a_j] = \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline \Theta & 0 \end{array}\right]$$

$$\texttt{ConQ}[s_i a_j] = \left[\begin{array}{c|c} \text{Partition of } [s_i \mapsto a_j] & \text{Q-values} \\ \hline [s_i \mapsto a_j] & 0 \end{array}\right]$$

$$\texttt{ConQ}[s_i] = \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline \Theta & 0 \end{array}\right]$$

Let $\perp$ be the terminal state and let $\texttt{ConQ}[\perp]$ be initialized as above, in the same way as $\texttt{ConQ}[s_i]$.

$\texttt{ConQ}[s_2]$	
Partition of Θ	Q-val
$[s_1 \to a_2][s_2 \to a_2][s_3 \to a_1]$	0
$[s_1 \to a_2][s_2 \to a_1]$	0
$[s_1 \to a_1][s_4 \to a_1]$	0.3
$[s_1 \to a_1][s_4 \to a_2]$	0.5

$\texttt{ConQ}[s_2]$	
Partition of Θ	Q-val
$[s_2 \to a_2][s_3 \to a_1]$	0
$[s_2 \to a_1]$	0

$\texttt{ConQ}[s_3]$	
Partition of Θ	Q-val
$[s_3 \to a_2][s_4 \to a_2]$	2
$[s_3 \to a_2][s_4 \to a_1]$	0
$[s_3 \to a_1]$	0

$\texttt{ConQ}[s_4]$	
Partition of Θ	Q-val
$[s_4 \to a_2]$	2
$[s_4 \to a_1]$	0

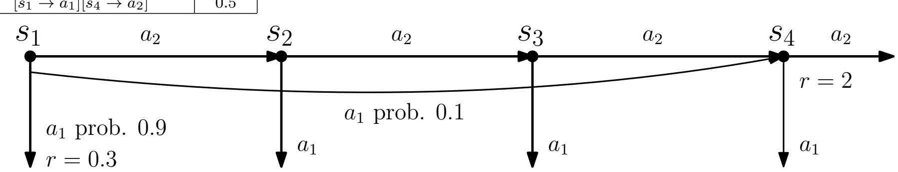

Figure 8: Example run of PCVI on MDP in Fig. 1. The blue, green, red and gray arrows indicate the direction of backups for constructing the tables $\texttt{ConQ}[s_i]$ for $i = 1..4$ in that order.

Iteration #1. We now go through PCVL's updates within the for loop (lines 5-8). We run the updates backwards, starting with s_4, a_2. The updates are:

$$\begin{aligned}
\texttt{Q}[s_4 a_2] &= R(s_4, a_2) + \gamma[p(\perp | s_4, a_2)\texttt{ConQ}[\perp]] \\
&= \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline \Theta & 2 \end{array}\right] \\
\texttt{ConQ}[s_4 a_2] &= \left[\begin{array}{c|c} \text{Partition of } [s_4 \mapsto a_2] & \text{Q-values} \\ \hline \Theta \cap [s_4 \mapsto a_2] = [s_4 \mapsto a_2] & 2 \end{array}\right] \\
\texttt{ConQ}[s_4] &= \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline [s_4 \mapsto a_1] & 0 \\ [s_4 \mapsto a_2] & 2 \end{array}\right]
\end{aligned}$$

Iteration #2. Update s_4, a_1.

$$\begin{aligned}
\texttt{Q}[s_4 a_1] &= R(s_4, a_1) + \gamma[p(\perp | s_4, a_1)\texttt{ConQ}[\perp]] \\
&= \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline \Theta & 0 \end{array}\right] \\
\texttt{ConQ}[s_4 a_2] &= \left[\begin{array}{c|c} \text{Partition of } [s_4 \mapsto a_1] & \text{Q-values} \\ \hline \Theta \cap [s_4 \mapsto a_1] = [s_4 \mapsto a_1] & 0 \end{array}\right] \\
\texttt{ConQ}[s_4] &= \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline [s_4 \mapsto a_1] & 0 \\ [s_4 \mapsto a_2] & 2 \end{array}\right]
\end{aligned}$$

Iteration #3. Update s_3, a_2.

$$\begin{aligned}
\texttt{Q}[s_3 a_2] &= R(s_3, a_2) + \gamma[p(s_4 | s_3, a_2)\texttt{ConQ}[s_4]] \\
&= \texttt{ConQ}[s_4] \\
&= \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline [s_4 \mapsto a_1] & 0 \\ [s_4 \mapsto a_2] & 2 \end{array}\right] \\
\texttt{ConQ}[s_3 a_2] &= \left[\begin{array}{c|c} \text{Partition of } [s_3 \mapsto a_2] & \text{Q-values} \\ \hline [s_4 \mapsto a_1] \cap [s_3 \mapsto a_2] = [s_4 \mapsto a_1][s_3 \mapsto a_2] & 0 \\ [s_4 \mapsto a_2] \cap [s_3 \mapsto a_2] = [s_4 \mapsto a_2][s_3 \mapsto a_2] & 2 \end{array}\right] \\
\texttt{ConQ}[s_3] &= \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline [s_3 \mapsto a_1] & 0 \\ [s_4 \mapsto a_1][s_3 \mapsto a_2] & 0 \\ [s_4 \mapsto a_2][s_3 \mapsto a_2] & 2 \end{array}\right]
\end{aligned}$$

Iteration #4. Update s_3, a_1.

$$\begin{aligned}
\texttt{Q}[s_3 a_1] &= R(s_3, a_1) + \gamma[p(\perp | s_3, a_1)\texttt{ConQ}[\perp]] \\
&= \texttt{ConQ}[\perp] \\
&= \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline \Theta & 0 \end{array}\right] \\
\texttt{ConQ}[s_3 a_1] &= \left[\begin{array}{c|c} \text{Partition of } [s_3 \mapsto a_1] & \text{Q-values} \\ \hline \Theta \cap [s_3 \mapsto a_1] = [s_3 \mapsto a_1] & 0 \end{array}\right] \\
\texttt{ConQ}[s_3] &= \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline [s_3 \mapsto a_1] & 0 \\ [s_4 \mapsto a_1][s_3 \mapsto a_2] & 0 \\ [s_4 \mapsto a_2][s_3 \mapsto a_2] & 2 \end{array}\right]
\end{aligned}$$

Iteration #5. Update s_2, a_2.

$$\begin{aligned}
\texttt{Q}[s_2a_2] &= R(s_2,a_2) + \gamma[p(s_3|s_2,a_2)\texttt{ConQ}[s_3]] \\
&= \texttt{ConQ}[s_3] \\
&= \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline [s_3 \mapsto a_1] & 0 \\ [s_4 \mapsto a_1][s_3 \mapsto a_2] & 0 \\ [s_4 \mapsto a_2][s_3 \mapsto a_2] & 2 \end{array}\right] \\
\texttt{ConQ}[s_2a_2] &= \left[\begin{array}{c|c} \text{Partition of } [s_2 \mapsto a_2] & \text{Q-values} \\ \hline [s_3 \mapsto a_1][s_2 \mapsto a_2] & 0 \\ [s_4 \mapsto a_1][s_3 \mapsto a_2] \cap [s_2 \mapsto a_2] = \emptyset & - \\ [s_4 \mapsto a_2][s_3 \mapsto a_2] \cap [s_2 \mapsto a_2] = \emptyset & - \end{array}\right] \\
&= \left[\begin{array}{c|c} \text{Partition of } [s_2 \mapsto a_2] & \text{Q-values} \\ \hline [s_3 \mapsto a_1][s_2 \mapsto a_2] & 0 \end{array}\right]
\end{aligned}$$

The witness oracle checks feasibility of $[s_3 \mapsto a_2][s_2 \mapsto a_2]$ by solving a system of linear inequalities. In this case,

$$\begin{aligned}
[s_2 \mapsto a_2] &\Longrightarrow \theta \cdot \phi(s_2,a_2) > \theta \cdot \phi(s_2,a_1) \Longrightarrow \theta_1 > 0, \\
[s_3 \mapsto a_2] &\Longrightarrow \theta \cdot \phi(s_3,a_2) > \theta \cdot \phi(s_3,a_1) \Longrightarrow \theta_1 < 0.
\end{aligned}$$

Hence the assignment of these two policy actions to these two states is infeasible and PCVI eliminates those two entries in the $\texttt{ConQ}[s_2a_2]$ table.

$$\texttt{ConQ}[s_2] = \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline [s_2 \mapsto a_1] & 0 \\ [s_3 \mapsto a_1][s_2 \mapsto a_2] & 0 \end{array}\right]$$

Iteration #6. Update s_2, a_1.

$$\begin{aligned}
\texttt{Q}[s_2a_1] &= R(s_2,a_1) + \gamma[p(\perp|s_2,a_1)\texttt{ConQ}[\perp]] \\
&= \texttt{ConQ}[\perp] \\
&= \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline \Theta & 0 \end{array}\right] \\
\texttt{ConQ}[s_2a_1] &= \left[\begin{array}{c|c} \text{Partition of } [s_2 \mapsto a_1] & \text{Q-values} \\ \hline \Theta \cap [s_2 \mapsto a_1] = [s_2 \mapsto a_1] & 0 \end{array}\right] \\
\texttt{ConQ}[s_2] &= \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline [s_2 \mapsto a_1] & 0 \\ [s_3 \mapsto a_1][s_2 \mapsto a_2] & 0 \end{array}\right]
\end{aligned}$$

Iteration #7. Update s_1, a_2.

$$\begin{aligned}
\texttt{Q}[s_1a_2] &= R(s_1,a_2) + \gamma[p(s_2|s_1,a_2)\texttt{ConQ}[s_2]] \\
&= \texttt{ConQ}[s_2] \\
&= \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline [s_2 \mapsto a_1] & 0 \\ [s_3 \mapsto a_1][s_2 \mapsto a_2] & 0 \end{array}\right] \\
\texttt{ConQ}[s_1a_2] &= \left[\begin{array}{c|c} \text{Partition of } [s_1 \mapsto a_2] & \text{Q-values} \\ \hline [s_2 \mapsto a_1][s_1 \mapsto a_2] & 0 \\ [s_3 \mapsto a_1][s_2 \mapsto a_2][s_1 \mapsto a_2] & 0 \end{array}\right] \\
\texttt{ConQ}[s_1] &= \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline [s_1 \mapsto a_1] & 0 \\ [s_2 \mapsto a_1][s_1 \mapsto a_2] & 0 \\ [s_3 \mapsto a_1][s_2 \mapsto a_2][s_1 \mapsto a_2] & 0 \end{array}\right]
\end{aligned}$$

Iteration #8. Update s_1, a_1.

$$\begin{aligned}
\texttt{Q}[s_1 a_1] &= R(s_1, a_1) + \gamma[p(\perp|s_1, a_1)\texttt{ConQ}[\perp] \oplus p(s_4|s_1, a_1)\texttt{ConQ}[s_4]] \\
&= 0.3 + 0.9\texttt{ConQ}[\perp] \oplus 0.1\texttt{ConQ}[s_4] \\
&= \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline \Theta \cap [s_4 \mapsto a_1] & 0.3 + 0.9\texttt{ConQ}[\perp](\Theta) + 0.1\texttt{ConQ}[s_4]([s_4 \mapsto a_1]) = 0.3 \\ \Theta \cap [s_4 \mapsto a_2] & 0.3 + 0.9\texttt{ConQ}[\perp](\Theta) + 0.1\texttt{ConQ}[s_4]([s_4 \mapsto a_2]) = 0.5 \end{array}\right] \\
\texttt{ConQ}[s_1 a_1] &= \left[\begin{array}{c|c} \text{Partition of } [s_1 \mapsto a_1] & \text{Q-values} \\ \hline [s_4 \mapsto a_1][s_1 \mapsto a_1] & 0.3 \\ [s_4 \mapsto a_2][s_1 \mapsto a_1] & 0.5 \end{array}\right] \\
\texttt{ConQ}[s_1] &= \left[\begin{array}{c|c} \text{Partition of } \Theta & \text{Q-values} \\ \hline [s_2 \mapsto a_1][s_1 \mapsto a_2] & 0 \\ [s_3 \mapsto a_1][s_2 \mapsto a_2][s_1 \mapsto a_2] & 0 \\ [s_4 \mapsto a_1][s_1 \mapsto a_1] & 0.3 \\ [s_4 \mapsto a_2][s_1 \mapsto a_1] & 0.5 \end{array}\right]
\end{aligned}$$

In iteration #9, we would update s_4, a_2 but none of the Q or ConQ data structures change. Likewise, subsequent iteration updates to any other s, a does not change these tables. Thus we have convergence and can recover the optimal admissible policy as follows. For initial state s_1 we look up table $\texttt{ConQ}[s_1]$ and find that the feasible set with highest value is $X^* = [s_4 \mapsto a_2][s_1 \mapsto a_1]$ with value 0.5. A feasible parameter $\theta^* = (-2, 0.5)$ can also be found via, for example, solving a system of linear inequalities by linear programming. Fig. 8 summarizes the backups performed by PCVL and the resulting ConQ tables.

A.8 PCQL with ConQ regression

While Policy-class Q-Learning allows one to generalize in the sense that a parameterized policy is returned, it does not explicitly model a corresponding Q-function. Such a Q-function allows one to predict and generalize state-action values of unobserved states. For example, when an initial state is never observed in the training data, the Q-function can be used to assess how good a particular cell or equivalence class of policies are.

This is the general idea behind our heuristic. It keeps a global collection of information sets that are gradually refined based on training data. The information set contains both the constraints defining a cell and a regressor that predicts *consistent* Q-values for that cell's policies. When there are too many information sets, one can use a pruning heuristic, such as removing information sets with low Q-values. The backup operation must respect the consistency requirement: a cell's Q-regressor gets updated only if $[s \mapsto a]$ is consistent with its constraints. As discussed earlier, pruning feasible information sets may result in no updates to any cell given a sample transition (since no cell may be consistent). But this can be avoided if our heuristic merges instead of deleting cells.

The algorithm pseudo-code is given in Algorithm 3. Every information set is a pair (X, w) where X is a set of parameters consistent with some set of constraints and w are the weights of a Q-regressor. In general w may be from a different function class than Θ. The Q-labels that w learns from is generated in the consistent manner stated above, that is, it predicts the consistent Q-value $\texttt{ConQ}_w(s, a)$ for any state-action pair. While it produces values for any (s, a) that is inconsistent with X, such values are not used for backups or finding an optimal policy.

A.9 Constructing consistent labels

We formulate the problem of consistent labeling as an mixed integer program (MIP).

Assume a batch of training examples $B = \{(s_t, a_t, r_t, s'_t)\}_{t=1}^{T}$, and a current regressor $\widetilde{Q}$ used to create "bootstrapped" labels. The nominal Q-update generates labels of the following form for each pair (s_t, a_t): $q_t = r_t + \gamma \max_{a'_t} \widetilde{Q}(s'_t, a'_t)$. The updated regressor is trained in supervised fashion with inputs (s_t, a_t) and label q_t.

To ensure policy class consistency, we must restrict the selection of the maximizing action a'_t so that:

Algorithm 3 Policy-Class Q-Learning with Regression

Input: Batch $B = \{(s_t, a_t, r_t, s'_t)\}_{t=1}^{T}$, γ, Θ, scalars α_t^{sa}, initial state s_0.

Initialize information sets $I \leftarrow \{(\Theta, w_\Theta)\}$.
$visited \leftarrow \emptyset$
for $(s, a, r, s') \in B$, t is iteration counter **do**
  If $s \notin visited$ then Refine(s)
  If $s' \notin visited$ then Refine(s')
  for $(X, w) \in I$ **do**
    if $X \cap [s \mapsto a] \neq \emptyset$ **then**
      $w \leftarrow w + \alpha_t^{sa}(r + \gamma \mathtt{ConQ}_w(s', \pi_X(s')) - \mathtt{ConQ}_w(s, a))\nabla_w \mathtt{ConQ}_w(s, a)$
    end if
    Prune I if too many information sets
  end for
end for

`/* Then recover an optimal policy */`
If $s_0 \notin visited$ then Refine(s_0)
$(X^*, w^*) \leftarrow \mathrm{argmax}_{(X,w)} \mathtt{ConQ}_w(s_0, \pi_X(s_0))$
Select some witness $\theta^* \in X^*$ then return π_{θ^*}

Procedure Refine(s)
  $I_{new} \leftarrow \emptyset$
  for $(X, w) \leftarrow I$.pop() **do**
    $X_i \leftarrow X \cap [s \mapsto a_i]$ for all a_i
    If $X_i \neq \emptyset$ then I_{new}.add$((X_i, w))$
  end for
  $I \leftarrow I_{new}$
  $visited$.add(s)

- $\cap_t [s'_t \mapsto a'_t] \neq \emptyset$ (i.e., selected maximizing actions are mutually consistent); and
- $[s_t \mapsto a_t] \cap [s'_t \mapsto a'_t] \neq \emptyset$, for all t (i.e., choice at s'_t is consistent with taking a_t at s_t).

We construct a consistent labeling by finding an assignment $\sigma : s'_t \rightarrow A(s'_t)$, assuming some reasonable maximization objective that satisfies these constraints. We illustrate this using the sum of the resulting labels as the optimization objective, though other objectives are certainly possible.

With a linear approximator, the problem can be formulated as a (linear) mixed integer program (MIP). For any parameter vector $\theta \in \Theta$, we write $\theta(s, a)$ to denote the linear expression of $Q(s, a; \theta)$. To meet the first requirement, we assume a single (global) parameter vector θ_g. For the second, we have a separate parameter vector θ_t for each training example (s_t, a_t, r_t, s'_t). We have variables q_t representing the bootstrapped (partial) label for each training example. Finally, we have an indicator variable $\mathbb{I}^t_{a'}$ for each $t \leq T$, $a' \in A(s')$: this denotes the selection of a' as the maximizing action for s'_t. We assume rewards are non-negative for ease of exposition. The following IP

over the interval becomes linear:

$$\max_{\mathbb{I}^t_{a'}, q_t, \theta_g, \theta_t} \quad \sum_{t \leq T} q_t \tag{11}$$

$$\text{s.t.} \quad \theta_t(s_t, a_t) \geq \theta_t(s_t, b) \forall b \in A(s_t), \forall t \tag{12}$$

$$\theta_t(s'_t, a'_t) \geq \mathbb{I}^t_{a'} \theta_t(s'_t, b') \forall b' \in A(s'_t), \forall t \tag{13}$$

$$\theta_g(s'_t, a'_t) \geq \mathbb{I}^t_{a'} \theta_g(s'_t, b') \forall b' \in A(s'_t), \forall t \tag{14}$$

$$q_t = r_t + \gamma \sum_{a' \in A(s'_t)} \mathbb{I}^t_{a'} \theta_g(s'_t, a'_t) \forall t \tag{15}$$

$$\sum_{a' \in A(s'_t)} \mathbb{I}^t_{a'} = 1, \forall t \tag{16}$$

The nonlinear term $\mathbb{I}^t_{a'}\theta_g(s'_t, a'_t)$ in the fourth constraint can be linearized trivially using a standard transformation since it is the product of a real-valued and a binary (0-1) indicator variable. This IP can be tackled heuristically using various greedy heuristics as well.

A.10 Discounting Hyperparameter Experiments

Our experiments use the Atari learning environment [3] to demonstrate the impact of changing the training discount factor, γ_ℓ, used in Q-learning on the resulting greedy policy. The delusional bias is particularly pronounced as γ_ℓ becomes close to 1, since incorrect temporal difference terms become magnified in the updates to the function approximator. As shown above, the delusional bias problem cannot be solved solely by choosing the right γ_ℓ; instead we show here that sweeping a range of γ_ℓ values can reveal the discounting paradoxes in realistic environments, while also showing how γ_ℓ tuning might mitigate some of the ill effects. Recall that it is possible for a larger γ_ℓ might lead to a better policy when evaluated on a smaller γ_e—this is in contrast to [15, 16] which suggest using a smaller $\gamma_\ell \leq \gamma_e$. In fact we see both phenomena.

We use a state-of-the-art implementation of Deep Q-Networks (DQN) [19], where we trained $Q_{\hat{\theta}^{\gamma_\ell}}$ by varying γ_ℓ used in the Q-updates while holding other hyperparameters constant. We use εGreedy with $\varepsilon = 0.07$ as the behaviour policy, and run training iterations until the training scores converge. Each iteration consists of at most 2.5×10^5 training steps (i.e., transitions), with max steps per episode set to 27×10^3. We use experience replay with a buffer size of 10^6 and a mini-batch size of 32. We evaluate converged Q-functions using the corresponding greedy policy[5] over episodes of 1.2×10^6 steps. We do 5 training restarts for each game tested.

We vary γ_ℓ to reflect varying effective horizon lengths, $h = 1/(1-\gamma_\ell)$, of $h \in \{100 - 10n : n = 0, 1, \ldots, 8\}$. ($\gamma_\ell > 0.99$ tends to cause divergence. For evaluation, we use the same values for γ_e, plus $\gamma_e \in \{0.995, 1\}$—the latter is commonly reported as total undiscounted return.

Figs. 9 and 10 show (normalized) returns across four benchmark Atari games, averaged across the 5 training restarts. The implementation we use is known to have low variability in performance/value in Atari once DQN has converged. Indeed, training scores converged to very similar values across restarts (i.e., the return of εGreedy($\varepsilon = 0.07$) with $\gamma_e = 1$); however, the resulting greedy policies can still exhibit differences (e.g., SpaceInvaders, $\gamma_\ell = 0.9833$ vs. $\gamma_\ell = 0.9857$ rows).

These heatmaps reveal several insights. Training with smaller γ_ℓ usually results in better policies, even when evaluated at maximum horizons, e.g., if we compare the first two rows to the last two across all four games. This is particularly true in Seaquest, where the two smallest γ_ℓ values give the best policies across a range of γ_e. The last row of Seaquest shows the most non-myopic policy performs worst.

Most critically, the heatmaps are clearly not diagonally dominant. One might expect each column to be single-peaked at the "correct" discount factor $\gamma_\ell = \gamma_e$, which should exhibit the highest (normalized) discounted return, with returns are monotonically non-decreasing when $\gamma_\ell < \gamma_e$ and monotonically non-increasing when $\gamma_\ell > \gamma_e$. But this does not occur in any domain. Qbert is perhaps the closest to being diagonally dominant, where smaller (resp. larger) γ_ℓ tend to perform better for smaller (resp. larger) γ_e.

In SpaceInvaders, a myopic policy π_1 (trained with $\gamma_\ell = 0.9667$) is generally better, than a more non-myopic policy π_2 (e.g. $\gamma_\ell = 0.9889$). Policy π_1 consistently achieves the best average evaluation score for for a variety of γ_e. about 29.7% better than π_2 for $\gamma_e = 1$. Figs. 11 and 12 show additional heatmaps demonstrating a different view of relative performance. Our results show that the "opposite" counter-example (Sec. 3.2) can also arise in practice. See Fig. 11, Qbert heatmap—in particular policy trained with $\gamma_\ell = 0.975$ generally performs better than policy trained with $\gamma_\ell = 0.99$. Prior work has only uncovered outperformance of myopically trained policies (though not with Q-learning), while we show that the opposite can also occur.

Figs. 11 and 12 have been normalized across each column so that average discounted sum of rewards in each column are in the unit interval. This allows for easier comparison across the different policies learned from varying γ_ℓ. Note that this also exacerbates potentially small differences in the values (e.g. when original values in columns lie in a small range).

[5] We use εGreedy with $\varepsilon = 0.005$ to prevent in-game loops.

In Seaquest, one can clearly see the relative outperformance of models trained with $\gamma_\ell = 0.95, 0.9667$ over more non-myopic models (e.g. last three rows). For Qbert, the best policy for $\gamma_e < 0.995$ is the policy trained with $\gamma_\ell = 0.975$ while the best policy for $\gamma_e = 1$ is trained with $\gamma_\ell = 0.9857$. For Pong, at first place, the performance of model trained with $\gamma_\ell = 0.95$ gradually becomes worse. But this in fact is due to scaling by ever larger discount factors—if the undiscounted scores are negative then larger discount scaling only decreases its performance.

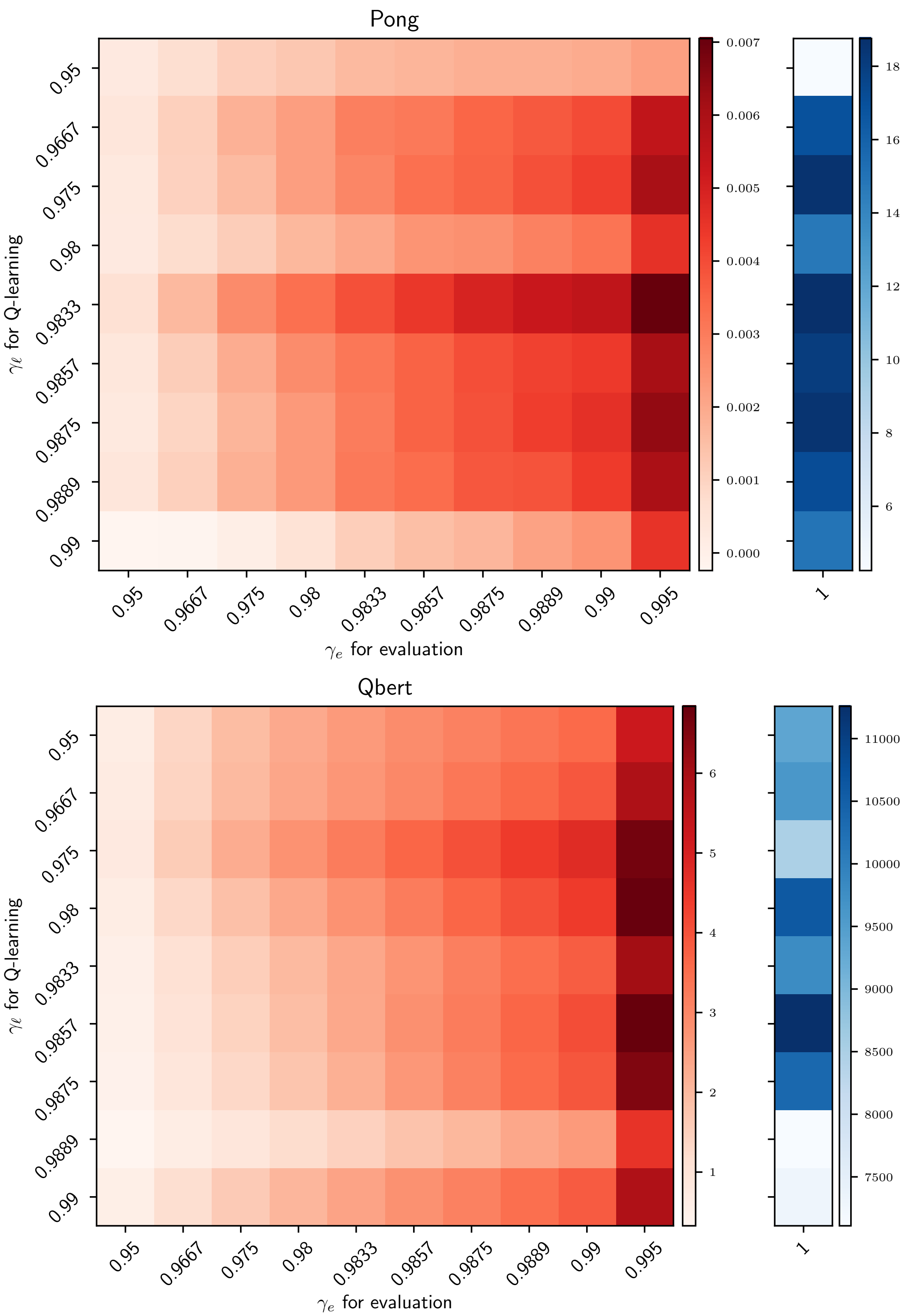

Figure 9: Each entry s_{ij} of left (red) heatmap shows the normalized scores for evaluating εGreedy$(Q_{\hat{\theta}^{\gamma_\ell}}, \varepsilon = 0.005)$ (row i) using γ_e (column j). The normalization is across each column: $s_{ij} \leftarrow (1 - \gamma_e)s_{ij}$. The right heatmaps (blue) show avg. unnormalized undiscounted total returns. Results averaged over 5 training restarts.

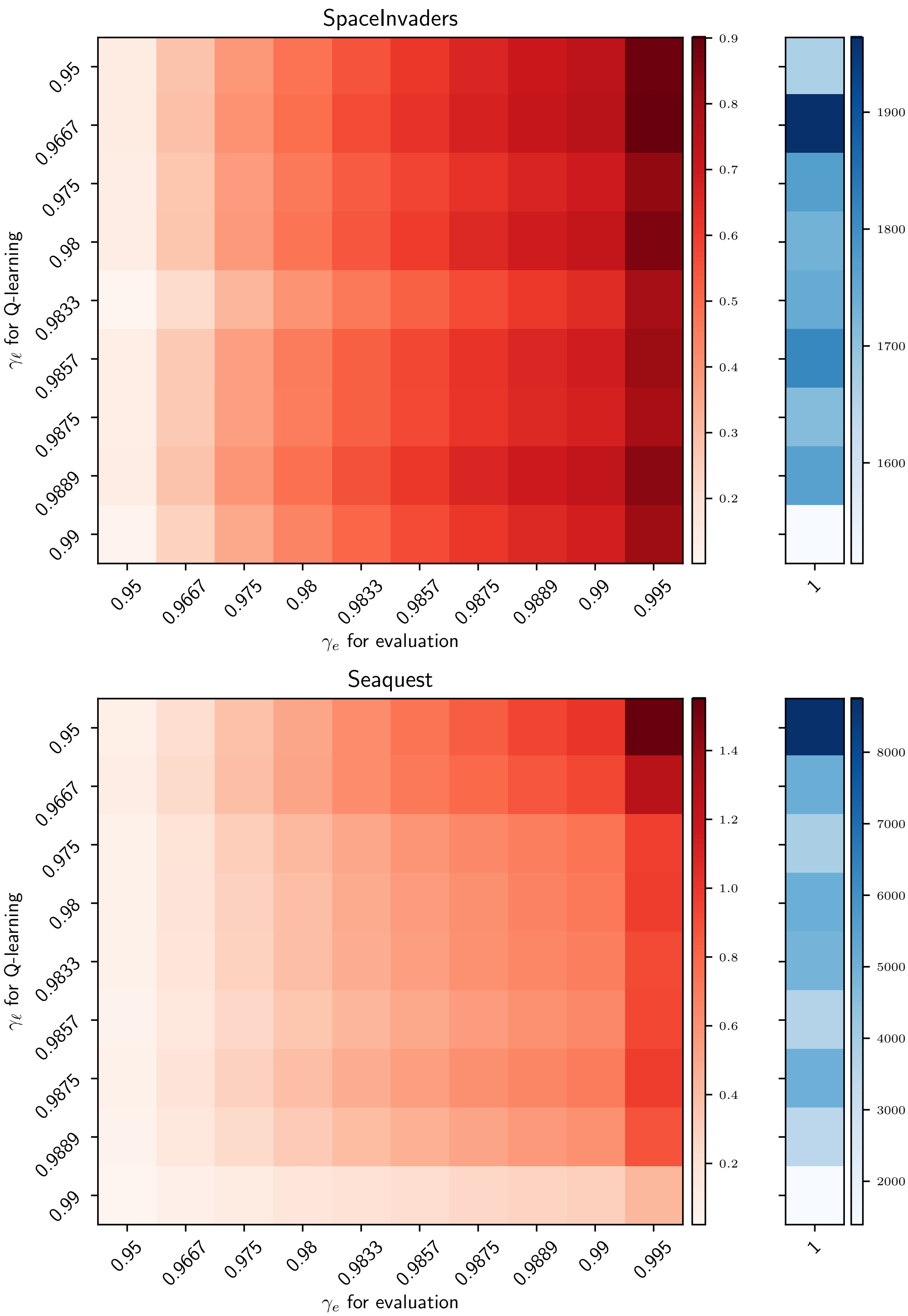

Figure 10: Similar to Fig. 9 with heatmaps for SpaceInvaders and Seaquest.

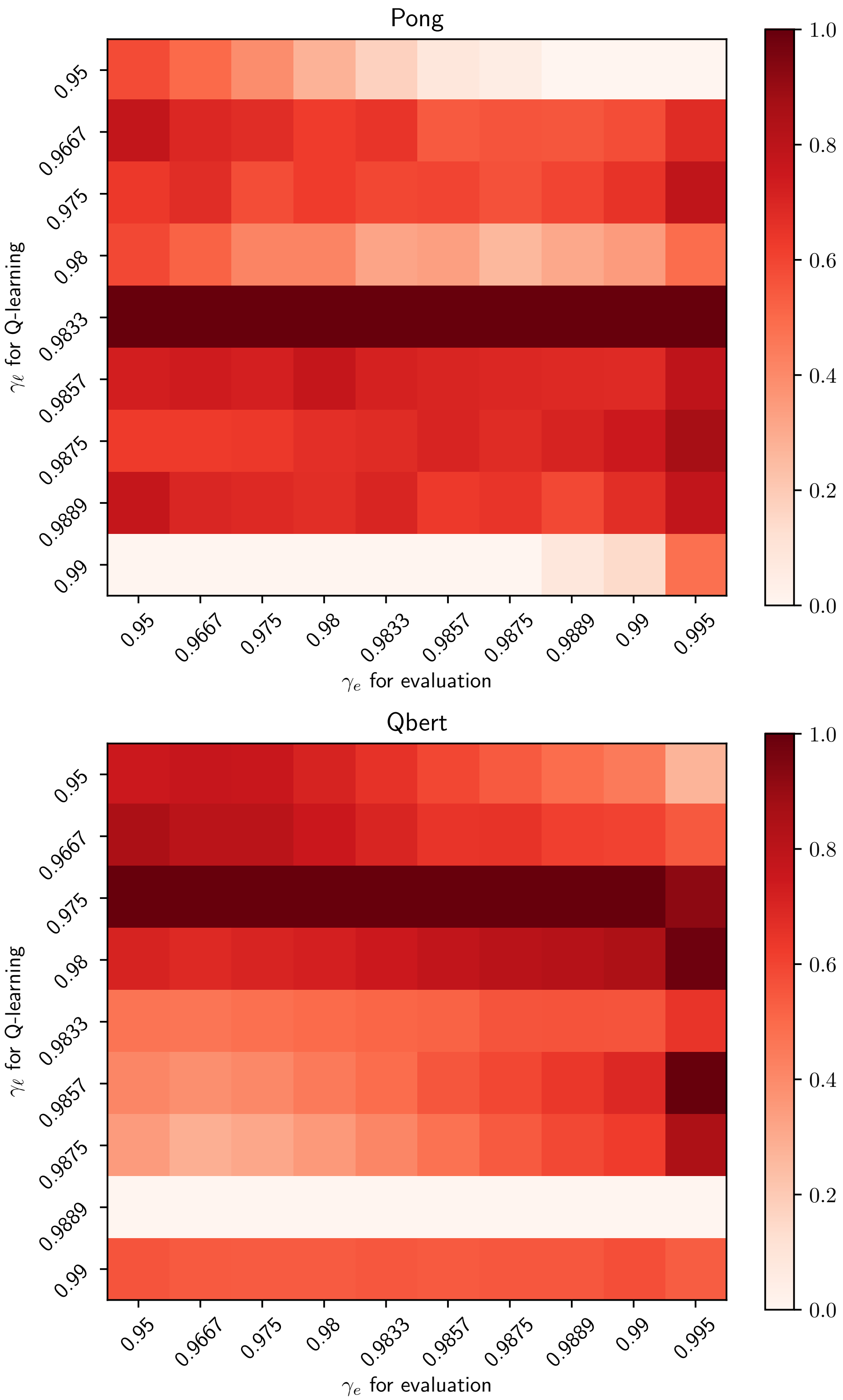

Figure 11: Each entry s_{ij} of a heatmap shows the scores normalized to the unit interval for evaluating $\varepsilon\mathsf{Greedy}(Q_{\hat{\theta}^{\gamma_\ell}}, \varepsilon = 0.005)$ (row i) using γ_e (column j). The normalization is across each column: $s_{ij} \leftarrow (s_{ij} - \min_k s_{kj})/(\max_k s_{kj} - \min_k s_{kj})$. Results averaged over 5 training restarts.

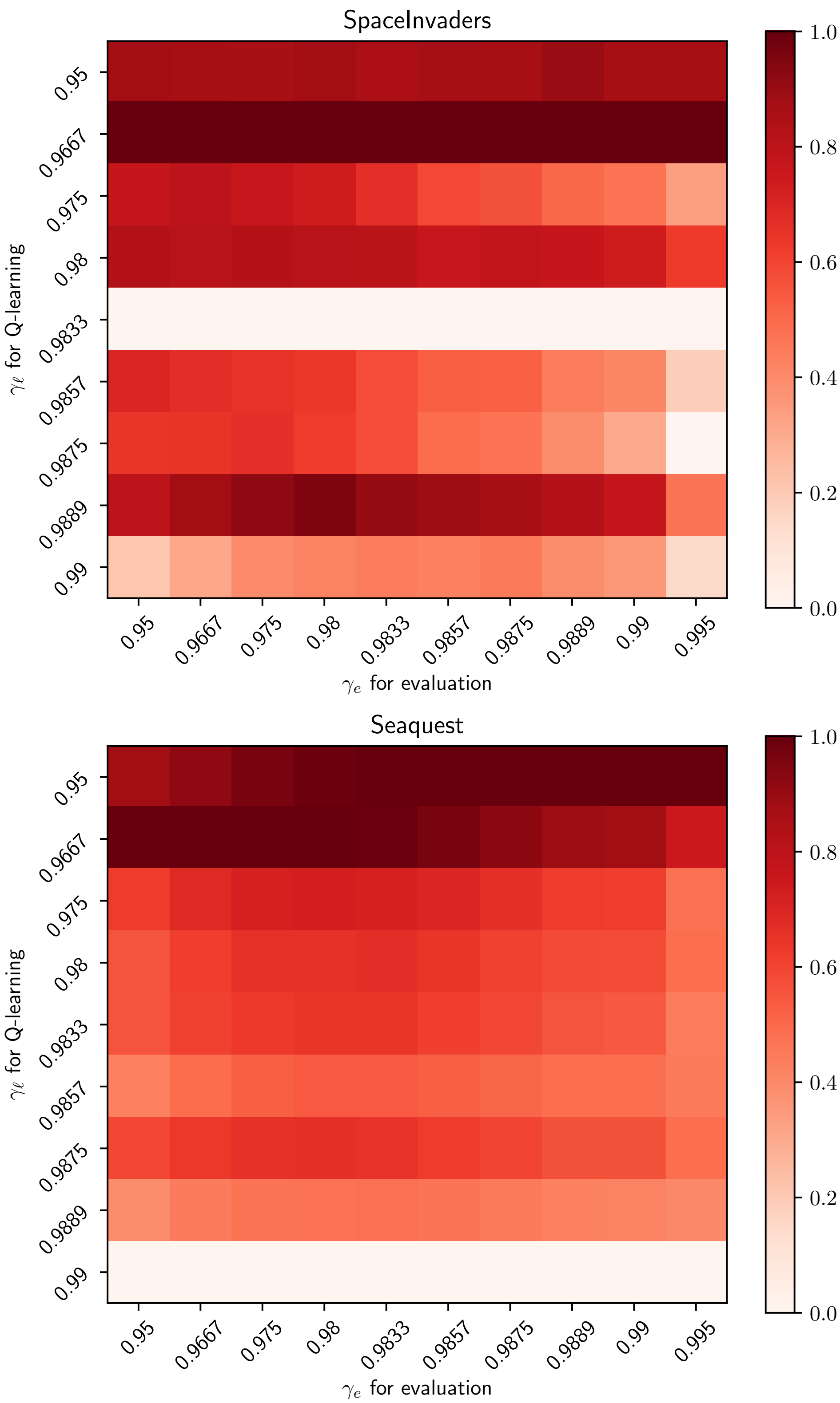

Figure 12: Similar to Fig. 11 with heatmaps for SpaceInvaders and Seaquest.