Detecting Spiky Corruption in Markov Decision Processes

Jason Mancuso[1*], **Tomasz Kisielewski**[2*], **David Lindner**[3*] and **Alok Singh**[4*]
[1]Dropout Labs
[2]Independent Researcher
[3]ETH Zürich
[4]Terrafuse
jason@manc.us, tymorl@gmail.com, lindnerd@ethz.ch, alok.singh@berkeley.edu

Abstract

Current reinforcement learning methods fail if the reward function is imperfect, i.e. if the agent observes reward different from what it actually receives. We study this problem within the formalism of Corrupt Reward Markov Decision Processes (CRMDPs). We show that if the reward corruption in a CRMDP is sufficiently "spiky", the environment is solvable. We fully characterize the regret bound of a Spiky CRMDP, and introduce an algorithm that is able to detect its corrupt states. We show that this algorithm can be used to learn the optimal policy with any common reinforcement learning algorithm. Finally, we investigate our algorithm in a pair of simple gridworld environments, finding that our algorithm can detect the corrupt states and learn the optimal policy despite the corruption.

1 Introduction

The reward function distinguishes reinforcement learning (RL) from other forms of learning. If the reward function is misspecified, the agent can learn undesired behavior [Everitt *et al.*, 2017]. It is an open problem in RL to detect or avoid misspecified rewards [Amodei *et al.*, 2016].

[Everitt *et al.*, 2017] formalize this problem by introducing *Corrupt Reward Markov Decision Processes* (CRMDPs). Informally, a CRMDP is a MDP in which the agent receives a corrupted reward signal. A *No Free Lunch Theorem* for CRMDPs states that they are unlearnable in general; however, additional assumptions on the problem can lead to learnable subclasses. In particular, [Everitt *et al.*, 2017] introduce a set of strong assumptions that allow for a quantilizing agent to achieve sublinear regret. Other assumptions, however, may lead to distinct learnable subclasses that are also useful in practice.

In this work, we propose a set of assumptions to create such a subclass. Intuitively, our assumptions capture the notion of the corruption being "spiky" with respect to a distance measure on the state space.

[*]equal contribution

1.1 Problem motivation

CRMDPs naturally capture different notions of reward misspecification such as wireheading, side effects, avoiding supervision, and sensory malfunction [Everitt *et al.*, 2017; Amodei *et al.*, 2016]. This makes them a useful framework for developing RL agents robust to reward corruption.

Additionally, different approaches to learning reward functions can be interpreted in the CRMDP framework, such as semi-supervised RL [Finn *et al.*, 2016], cooperative inverse RL [Hadfield-Menell *et al.*, 2016], learning from human preferences [Christiano *et al.*, 2017], and learning values from stories [Riedl and Harrison, 2016]. Approaches to learn a reward function from expert input such as inverse reinforcement learning (IRL) [Ng *et al.*, 2000] can yield corrupt reward functions when learning from an expert of bounded rationality or from sub-optimal demonstrations. CRMDPs may be able to provide new theoretical guarantees and insights for IRL methods, which are often limited by the assumption that the expert acts nearly optimal.

1.2 Solution motivation

Our approach is inspired by connections to work on robustness to noise and fairness in supervised learning. The connection between supervised learning and RL has been discussed before (e.g. see [Barto and Dietterich, 2004]); here, we only use it in an informal way to motivate our approach.

In particular, one way to view supervised learning is as a special case of RL. In this interpretation, the RL policy corresponds to the supervised learning model, the actions are to pick a specific label for an input, and the reward is an indicator function that is 1 if the true label matches our pick, and 0 otherwise. The reward is provided by an oracle that can provide the true labels for a fixed training set of samples.

In noisy supervised learning, this oracle can be fallible, meaning the true label of an instance may not match the label the oracle provides. In this setting, the goal is to learn the true reward function despite only having access to a corrupt reward function. It has been observed that deep neural networks can learn in the presence of certain kinds of noise [Rolnick *et al.*, 2017; Drory *et al.*, 2018], which suggests that some classes of CRMDPs beyond those investigated in [Everitt *et al.*, 2017] can be solved.

For further inspiration, we turn to the field of fairness in supervised classification. [Dwork *et al.*, 2012] provide a nat-

ural definition of individual fairness using distance metrics on the input and output spaces and a corresponding Lipschitz conditions. Intuitively, a classifier is considered fair if it provides similar labels for similar input samples. Our approach to solving CRMDPs is similar. However, we apply Lipschitz conditions to the reward function rather than the classifier. A simple derivation shows that these interpretations are equivalent when the likelihood of the classifier is used to define the reward function.

1.3 Related Work

We aim to detect reward corruption by assuming the true reward to be "smooth" and the corruption to be "spiky". Smoothness of the reward function with respect to a distance in state space is a classic notion in machine learning [Santamaría *et al.*, 1997]. Assumptions about the smoothness of the reward function have also been used to ensure safety properties for RL algorithms, for example by [Turchetta *et al.*, 2016] or [García and Fernández, 2012].

Both define smoothness with respect to a distance metric in the state space which is similar to our approach. However, they tackle the problem of safely exploring an MDP, i.e. without visiting dangerous states, and do not consider reward corruption. Another key difference are the assumptions on the distance functions, which are a subset of metrics that are connected to the (known) transition function in [Turchetta *et al.*, 2016] or are just the Euclidean distance in [García and Fernández, 2012]. In contrast, we allow any metric.

There also exist approaches for automatically learning distance functions on the state space [Taylor *et al.*, 2011; Globerson and T. Roweis, 2005; Davis *et al.*, 2007; Jain *et al.*, 2009]. Such methods might be used in future work that remove the need for explicitly providing a distance function.

2 Problem statement

Let us recall the definition of CRMDPs from [Everitt *et al.*, 2017].

Definition 1 (CRMDP)**.** *A* Corrupt Reward MDP *(CRMDP) is a finite-state MDP $\langle \mathcal{S}, \mathcal{A}, T, R\rangle$ with an additional* corrupt reward function $C\colon \mathcal{S} \to \mathbb{R}$*. We call $\langle \mathcal{S}, \mathcal{A}, T, R\rangle$ the* underlying MDP*, $\mathcal{S}_n = \{x \in \mathcal{S} \mid R(x) = C(x)\}$ the set of* non-corrupt states*, and its complement, $\mathcal{S}_c = \mathcal{S} \setminus \mathcal{S}_n$, the set of* corrupt states*.*

Note that C represents the reward observed by an agent and may be equal to the real reward R in some, or even all, of the states. Our aim is to identify the corrupt states in $\mathcal{S}_c$ and learn an optimal policy with respect to R while only observing C. In general, this is impossible according to the CRMDP No Free Lunch Theorem. However special classes of CRMDPs may not have this limitation [Everitt *et al.*, 2017]. Therefore, we consider CRMDPs with a specific form of R and C.

Definition 2 (Spiky CRMDP)**.** *Let M be a CRMDP with two additional functions, d and* LV*. $d\colon \mathcal{S} \times \mathcal{S} \to \mathbb{R}^+$ is a metric on the state space, and* LV$\colon \mathbf{Powerset}(\mathcal{S}) \times \mathcal{S} \to \mathbb{R}^+$ *is non-decreasing with respect to set inclusion. We call M a* Spiky CRMDP *if the following assumptions are satisfied:*

1. *$\mathcal{S}_n$ is nonempty*
2. *$\forall x, y \in \mathcal{S},\ |R(x) - R(y)| \leq d(x, y)$,*
3. *$\forall x \in \mathcal{S}_c,\ \mathrm{LV}_{\mathcal{S}_n}(x) > \sup_{y \in \mathcal{S}_n} \mathrm{LV}_{\mathcal{S}}(y)$.*

We call d the distance *between the states, and* LV *the* Lipschitz violation measure*.*

Intuitively, the distance d should capture some notion of smoothness of the true reward function in each state. The goal is to construct this distance such that the reward in corrupt states is much less smooth (hence, "spiky"). The assumptions 2 and 3 formalize this intuition.

Note the strong relationship between the distance and reward functions. For a given Spiky CRMDP one cannot be modified independently of the other without breaking the assumptions. In particular, any linear transformation applied to the reward, such as e.g. scaling, has to be also performed on the distance function.

The LV function is meant to be a measure of Lipschitz violations of a state – "how much" does a given state violate (2) with respect to some set of states $\mathcal{S}_n$ when one substitutes C for R. We propose two ways of measuring this, which further refine the class of Spiky CRMDPs. Unless otherwise noted, all our examples satisfy the conditions in definition 2 with either of these functions used as LV.

Definition 3 (NLV)**.** *The* Number of Lipschitz Violations *of $x \in \mathcal{S}$ with respect to $A \subseteq \mathcal{S}$ is*

$$\mathrm{NLV}_A(x) := \mu(\{y \in A \mid |C(x) - C(y)| > d(x, y)\}),$$

where μ is a measure on $\mathcal{S}$.

Definition 4 (TLV)**.** *The* Total Lipschitz Violation *of $x \in \mathcal{S}$ with respect to $A \subseteq \mathcal{S}$ is*

$$\mathrm{TLV}_A(x) = \int_{y \in A} \min\{0, |C(x) - C(y)| - d(x, y)\}\ d\mu(y),$$

where μ is a measure on $\mathcal{S}$.

Note that both variants require a measure on the state space. In general there might not be a natural choice, but for finite state spaces, which we will consider, the counting measure is often reasonable. In this case, NLV counts the number of states which violate the Lipschitz condition with the given state, while TLV sums up the magnitudes of the violations.

3 Theoretical Results

3.1 Corruption identification

With this setup in place, we can now introduce an algorithm to detect corrupt states in finite Spiky CRMDPs, which is shown in algorithm 1. The core idea is simple: We maintain a set of corrupt states, initially empty, and sort all states descending by their Lipschitz violation with respect to all states. Then for each state we check its Lipschitz violation with respect to all states that we have not identified as corrupt yet. If it is positive, we mark the state as corrupt. As soon as we encounter a state with zero Lipschitz violation we are done.

This algorithm makes use of assumptions 2 and 3 in definition 2. Assumption 3 makes sure that by sorting the states in descending order we consider all corrupt states first. Assumption 2 then provides us with a simple stopping condition, namely no further violations of the Lipschitz condition.

Algorithm 1 An algorithm for identifying corrupt states in Spiky CRMDPs.

function IDENTIFYCORRUPTSTATES($\mathcal{S}, \mathrm{LV}) \to \hat{\mathcal{S}}_c$
  $\hat{\mathcal{S}}_c \leftarrow \emptyset$
  Sort $x \in \mathcal{S}$ by $\mathrm{LV}_{\mathcal{S}}(x)$ decreasing
  for $x \in \mathcal{S}$ **do**
    if $\mathrm{LV}_{\mathcal{S}\setminus\hat{\mathcal{S}}_c}(x) = 0$ **then**
      return $\hat{\mathcal{S}}_c$
    Add x to $\hat{\mathcal{S}}_c$

Proposition 1. *Let M be a spiky CRMDP. Then algorithm 1 returns $\mathcal{S}_c$ when given $\mathcal{S}$ and* LV *as input.*

This proposition simply states that the detection algorithm is able to correctly detect all corrupt states. The proof of this and all following statements is included in the appendix.

3.2 A posteriori bounds on regret

We now turn to the problem of learning an optimal policy despite the corruption in a CRMDP. Our first approach is to learn from an optimistic estimate of the true reward based on the Lipschitz condition on the rewards. To this end we first define such upper and lower bounds on the rewards.

Definition 5. *Let $M = \langle \mathcal{S}, \mathcal{A}, T, R, C, d, \mathrm{LV}\rangle$ be a spiky CRMDP. Then for a state x we define the* reward lower Lipschitz bound *to be*

$$\mathrm{lb}(x) = \max_{y \in \mathcal{S}_n} R(y) - d(x, y).$$

We call any state that introduces this bound and is closest to x the lower Lipschitz bounding state*. Symmetrically we define the upper bounds and upper bounding states.*

$$\mathrm{ub}(x) = \min_{y \in \mathcal{S}_n} R(y) + d(x, y).$$

For a non-corrupt state, both bounds just equal the true reward, so it's its own bounding state. We also get $\mathrm{lb}(x) \leq R(x)$ and $\mathrm{ub}(x) \geq R(x)$, because the distance function is positive definite.

Note that the reward lower (or upper) Lipschitz bound and bounding state can be computed by the agent after identifying the corrupt states, because it only requires access to the distance function and real reward function for non-corrupt states, which is equal to the corrupt reward there.

After computing these bounds, the agent can compute upper bounds on the regret it is experiencing. Finding a policy with respect to this optimistic estimate of the true reward function of corrupt states gives us a way to bound the expected regret using the Lipschitz bounds.

Proposition 2. *The expected regret with respect to R of a policy π' optimal with respect to* ub *(the reward* upper *Lipschitz bound) is bounded from above by*

$$\sup_{\pi \text{ ub-}optimal} \mathbb{E}_{\tau\sim\pi} \sum_{x\in\tau} \mathrm{ub}(x) - \mathrm{lb}(x). \tag{1}$$

Algorithm 2 An example of using algorithm 1 online when 3' is satisfied.

function LEARNONLINE(π, LV)
  $\hat{\mathcal{S}}_c \leftarrow \emptyset$
  for $\tau \sim \pi$ **do**
    $X \leftarrow IdentifyCorruptStates(\tau, \mathrm{LV})$
    $\hat{\mathcal{S}}_c \leftarrow \hat{\mathcal{S}}_c \cup X$
    $\hat{\mathcal{S}}_n \leftarrow \hat{\mathcal{S}}_n \cup (\tau \setminus \hat{\mathcal{S}}_c)$
    Train RL agent using reward signal $\mathrm{lb}_{\hat{\mathcal{S}}_n}$

This bound is not particularly useful as a theoretical result, because it is fairly easy to contrive CRMDPs where it becomes as large as the difference between the least and greatest possible cumulative rewards.

However, it might be useful to increase sample efficiency in an active reward learning setting. Say we have a supervisor that we can ask to provide us with the real reward of a given state, but such a question is expensive. We therefore want to maximize the information we get from a single question. The way we compute the bound (1) allows us to pick a question such that the upper bound on regret improves the most, which is a good criterion for question quality. We do not investigate this further, but suggest it as useful future work.

3.3 Optimality with corruption avoidance

To be able to guarantee an optimal policy despite corruption, we have to make an additional assumption about the environment. In particular, we will assume that the underlying MDP has at least one optimal policy which avoids all corrupt states. This essentially means that identifying and then avoiding the corrupt states is enough to solve the environment.

Proposition 3. *Let $M = \langle \mathcal{S}, \mathcal{A}, T, R, C, d, \mathrm{LV}\rangle$ be a spiky reward CRMDP and $\dot{M} = \langle \mathcal{S}, \mathcal{A}, T, R\rangle$ its underlying MDP. Then if there exists a $\dot{M}$-optimal policy π^* generating a trajectory $\tau \sim \pi^*$ that does not contain any corrupt states, then any policy optimal with respect to M using* lb *as a reward function will also be $\dot{M}$-optimal.*

The assumption of an optimal policy that always avoids corrupt states is very strong, especially in stochastic environments. However, this is to be expected since our result allows for solutions without any regret.

It is also worth noting that the assumption might be slightly weakened in practice, as we discuss in our experimental results.

4 Practical considerations

Algorithm 1 sorts over an entire state space, which requires (1) complete knowledge of the state space and (2) computational resources to perform a sorting operation.

4.1 Modification for Online Learning

We would like our algorithm to work online, i.e. at most considering a small batch of trajectories at once. Our current assumptions are not enough for such an algorithm to be correct. However, it is possible by strengthening assumption 3 in definition 2:

3' $\forall \pi, \tau \sim \pi \, \forall x \in \tau_c, \ \mathrm{LV}_{\tau_n}(x) > \sup_{y \in \tau_n} \mathrm{LV}_{\tau}(y)$,

where $\tau \sim \pi$ is a trajectory sampled from policy π. We treat the trajectory τ as a sequence of states, τ_c are the corrupt states in this sequence, and $\tau_n = \tau \setminus \tau_c$.

This is the same condition as before, except that we require it to hold over all possible trajectories through the MDP. Functionally speaking, this allows us to be sure that we'll be able to iteratively perform corruption identification without misidentifying any states. This assumption is much stronger than the previous version and restricts the class of environments satisfying it considerably. In practice, we believe that many interesting environments will still satisfy it, and many of those that do not may still be learnable in a similar manner.

With assumption 3' satisfied, we can use algorithm 1 for online reinforcement learning, as shown in algorithm 2. To do this, we sample trajectories from the current policy of the agent, apply the algorithm 1 on the individual trajectories and then update the policy using the reward lower Lipschitz bound.

Note that $\mathrm{lb}_{\hat{\mathcal{S}}_n}(x) = \max_{y \in \hat{\mathcal{S}}_n} R(y) - d(x, y)$. A straightforward application of proposition 1 shows that $\hat{\mathcal{S}}_c \to \mathcal{S}_c$ from below and $\hat{\mathcal{S}}_n \to \mathcal{S}_n$ from above when using algorithm 2.

In order to actually use the identified states, we also need the ability to compute the lb and ub functions. This once again would normally require access to the whole state space with corrupt states identified. However, we can approximate lb and ub by slowly building up the known state space and corrupt state space. This is what is reflected in algorithm 2, specifically when using $\mathrm{lb}_{\hat{\mathcal{S}}_n} \approx \mathrm{lb}$ as a reward signal. This approximation is pessimistic for corrupt states but converges as $\hat{\mathcal{S}}_n \to \mathcal{S}_n$.

4.2 Memory complexity

Memory complexity is an additional challenge for our algorithm. Keeping the whole state space in memory is usually not feasible, and even keeping only the encountered states can quickly result in performance problems. We implemented some optimizations to reduce the memory consumption of our approach. While they had no effect in the small toy environments we consider in this paper, for completeness and future reference we include a description of these optimizations in appendix B.

5 Experiments

We ran experiments on gridworld environments, detailed below. On each environment, we trained three different agents using an implementation of PPO [Schulman *et al.*, 2017].

The first agent just uses PPO with access to the corrupt reward, without any consideration for corruption.

The second agent uses PPO with access to the hidden reward, with the corruption removed. These are two baselines – how well an algorithm performs on the corrupt state and how well it could perform on the environment if it was not corrupted.

The third agent uses algorithm 2 during the rollout phase of the PPO algorithm. In particular, it identifies corrupt states in the rollouts and replaces their rewards by the Lipschitz bounds. The full code used, logs and commands to reproduce the experiments can be found at: https://github.com/jvmancuso/safe-grid-agents.

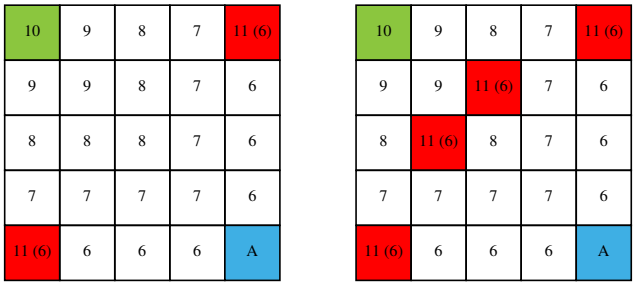

Figure 1: Toy Spiky CRMDP envorinments *Corners* on the left, *On-TheWay* on the right. The blue cell in the lower right hand corner is the starting position of the agent and the green cell in the upper left hand corner the goal. The true reward collected in each state is determined by the max-distance to the goal and shown here by the numbers in each cell. The reward in the red cells is corrupted and the underlying true reward is shown in parentheses.

To evaluate the quality of our algorithm for each experiment we calculate the average corrupt reward, average hidden reward, the sample complexity needed to achieve this result, and the ratio of this complexity to the complexity needed in the non-corrupt baseline. The results are summarized in Table 1 and we proceed by discussing them in detail.

5.1 Toy Spiky-CRMDP

Similar to [Everitt *et al.*, 2017], we construct a toy example under which our learnability guarantee from proposition 3 is satisfied. We also slightly tweak this toy example to break the requirements for proposition 3, with the hope of demonstrating that the theorem's requirements can be relaxed to some extent without harming learnability. The figure shows the toy example and its modification.

The gridworlds are shown in 1. The agent starts on the blue field and, in typical gridworld fashion, can move up/down/left/right as its action. The red cells are corrupted states and give the agent an unusually high reward, thereby satisfying our conditions about the "spikyness" of the corruption. We use the Manhattan metric as our distance measure. Note that in the environment on the right, the optimal policy must encounter corrupt states, violating the assumption of proposition 3. However, because the optimal policy remains optimal after substituting lb for the reward we should still expect good performance from our algorithm.

5.2 Results

The baseline results for both environments are as expected. PPO with access to the corrupt reward very quickly learns the corrupt-optimal policy, going straight to the bottom left or top right corner. PPO with access to the hidden reward needs significantly more data to learn the optimal policy, because the problem is more complicated and cannot be solved with a constant policy.

Environment	Reward	Agent	Avg. Corrupt Reward	Avg. True Reward	Sample Complexity	SC ratio
Corners	Corrupt	Baseline	73	48	5421	0.17
	Uncorrupt	Baseline	64	64	31410	1.00
	Uncorrupt	CRMDP	64	64	32250	1.03
	Corrupt	CRMDP	64	64	57000	1.81
OnTheWay	Corrupt	Baseline	73	48	4194	0.09
	Uncorrupt	Baseline	64	64	45310	1.00
	Uncorrupt	CRMDP	64	64	51750	1.14
	Corrupt	CRMDP	64	64	94380	2.08

Table 1: Results of the gridworld experiments described in section 5. In addition to the final corrupted and hidden true reward we report the *sample complexity* of each, which is defined as the number of episodes required for a moving average (momentum=0.9) of observed return to reach its optimal value. The *SC ratio* is the ratio of the sample complexity compared to the baseline model that has access to the hidden true reward function. Note that sample complexity measures are generally susceptible to noise, and should be interpreted with caution.

As a sanity check we ran an additional baseline test for these toy environments – our algorithm with access to the hidden reward. As it does not encounter any states it could perceive as corrupt it performs comparably to the baseline.

Finally we ran our algorithm without access to the hidden reward. In both cases it learned the optimal policy, requiring about two times as much data as the agent with access to the hidden reward. It is worth pointing out that because of the way the environments are constructed, this additional data is most likely *not* used for learning the bounds on the true reward. These bounds will almost always be as good as they can as soon as the agent identifies the corruption. Rather, the increased difficulty probably stems from the lower differences between optimal and sub-optimal policy payoffs.

6 Conclusion

The class of Spiky CRMDPs resolves several limitations in the class of previously known, solvable CRMDPs. In particular, this class of MDPs need not have finite diameter (state spaces symmetric in time), we demonstrate that they can be solved in the usual MDP formalism without recourse to the decoupled RL of [Everitt *et al.*, 2017].

Despite the the experimental support for our algorithm's success in toy gridworld environments, there are several limitations of our solution. Even though we can minimize the regret, it required quite a few assumptions to do so. Our results for the *OnTheWay* environment suggest that these assumptions can be weakened further, and this could be useful future work. However, we believe the regret bound is most intriguing, as it can be used for accelerating exploration in decoupled RL schemes. This is most apparent for semi-supervised RL, but also applies to other settings in which reward information can be inferred from external channels or actors. For this bound to become practically useful, future work should prioritize learning the Spiky CRMDP distance metric d from trajectory data in an online or active reward learning setting.

More generally, Lipschitz reward functions and spiky corruption can be seen as a particularly strong prior in the Bayesian RL setting. Our theorems and experiments demonstrate that this can be used to encode useful inductive biases in relevant environments. While we demonstrate a single instance of its usefulness in learning within misspecified environments, these priors can be very useful in a wide variety of practical settings in which Bayesian RL has traditionally fallen short.

7 Acknowledgements

Thanks to Victoria Krakovna and Tom Everitt for generous feedback and insightful discussions about this work and to Rohin Shah, Jan Leike, and Geoffrey Irving for providing valuable feedback on early proposals of our approach.

Furthermore, thanks to the organizers of the 2nd AI Safety Camp during which a significant portion of this work was completed.

References

[Amodei *et al.*, 2016] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul F. Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *CoRR*, abs/1606.06565, 2016.

[Barto and Dietterich, 2004] A. G. Barto and T.G. Dietterich. *Reinforcement learning and its relationship to supervised learning*. John Wiley and Sons, Inc, 2004.

[Christiano *et al.*, 2017] Paul F. Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, 2017.

[Davis *et al.*, 2007] Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S. Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 209–216, New York, NY, USA, 2007. ACM.

[Drory *et al.*, 2018] Amnon Drory, Shai Avidan, and Raja Giryes. On the resistance of neural nets to label noise. *CoRR*, abs/1803.11410, 2018.

[Dwork *et al.*, 2012] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Innovations in Theoretical Computer Science Conference*, 2012.

[Everitt *et al.*, 2017] Tom Everitt, Victoria Krakovna, Laurent Orseau, and Shane Legg. Reinforcement learning with a corrupted reward channel. In *International Joint Conference on Artificial Intelligence*, 2017.

[Finn *et al.*, 2016] Chelsea Finn, Tianhe Yu, Justin Fu, Pieter Abbeel, and Sergey Levine. Generalizing skills with semi-supervised reinforcement learning. *arXiv preprint arXiv:1612.00429*, 2016.

[García and Fernández, 2012] Javier García and Fernando Fernández. Safe exploration of state and action spaces in reinforcement learning. *J. Artif. Int. Res.*, 45(1):515–564, September 2012.

[Globerson and T. Roweis, 2005] Amir Globerson and Sam T. Roweis. Metric learning by collapsing classes. In *Advances in Neural Information Processing Systems*, volume 18, 01 2005.

[Hadfield-Menell *et al.*, 2016] Dylan Hadfield-Menell, Anca D. Dragan, Pieter Abbeel, and Stuart J. Russell. Cooperative inverse reinforcement learning. *CoRR*, abs/1606.03137, 2016.

[Jain *et al.*, 2009] Prateek Jain, Brian Kulis, Inderjit S. Dhillon, and Kristen Grauman. Online metric learning and fast similarity search. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 761–768. Curran Associates, Inc., 2009.

[Ng *et al.*, 2000] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[Riedl and Harrison, 2016] Mark O. Riedl and Brent Harrison. Using stories to teach human values to artificial agents. In *AI, Ethics, and Society, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 13, 2016.*, 2016.

[Rolnick *et al.*, 2017] David Rolnick, Andreas Veit, Serge J. Belongie, and Nir Shavit. Deep learning is robust to massive label noise. *CoRR*, abs/1705.10694, 2017.

[Santamaría *et al.*, 1997] Juan Carlos Santamaría, Richard S. Sutton, and Ashwin Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behaviour*, 6:163–217, 1997.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Taylor *et al.*, 2011] Matthew E. Taylor, Brian Kulis, and Fei Sha. Metric learning for reinforcement learning agents. In *AAMAS*, 2011.

[Turchetta *et al.*, 2016] Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration in finite markov decision processes with gaussian processes. In *Neural Information Processing Systems (NIPS)*, pages 4305–4313, December 2016.

A Proofs of theoretical results

Proof of proposition 1:

Proof. First note that

$$\forall_{x\in\mathcal{S}_c}\,\mathrm{LV}_{\mathcal{S}}(x) \geq \mathrm{LV}_{\mathcal{S}_n}(x) > \sup_{y\in\mathcal{S}_n}\mathrm{LV}_{\mathcal{S}}(y),$$

because in general LV is non-decreasing with respect to inclusion, that is $\mathrm{LV}_A \geq \mathrm{LV}_B$ if $A \subseteq B$. This immediately tells us that all corrupt states $x \in \mathcal{S}_c$ are processed by Algorithm 1 before all non-corrupt states $y \in \mathcal{S}_n$.

We will show that all states added to the $\hat{\mathcal{S}}_c$ set are actually corrupt. For a state $x \in \mathcal{S}$ to be added to that set it has to satisfy

$$\mathrm{LV}_{\mathcal{S}\setminus\hat{\mathcal{S}}_c}(x) > 0.$$

This condition can only be satisfied only if $\mathcal{S}_c \setminus \hat{\mathcal{S}}_c \neq \emptyset$, because otherwise $\mathcal{S} \setminus \hat{\mathcal{S}}_c$ would contain no corrupt states and no non-corrupt states can violate the Lipschitz condition. But Algorithm 1 processes all corrupt states before any non-corrupt states, so x has to be among the corrupt ones.

The algorithm returns as soon as it finds a state x satisfying $\mathrm{LV}_{\mathcal{S}\setminus\hat{\mathcal{S}}_c}(x) = 0$. Because of the processing order it is sufficient to prove that such a state is a non-corrupt state. Note that

$$\mathrm{LV}_{\mathcal{S}\setminus\hat{\mathcal{S}}_c}(x) \geq \mathrm{LV}_{\mathcal{S}_n}(x),$$

because, as we have already shown, $\hat{\mathcal{S}}_c \subseteq \mathcal{S}_c$. Since the left hand side of this expression is zero and the right one is non-negative by definition, it also has to be zero. But zero cannot be strictly greater than a nonempty supremum of nonnegative values, so x cannot be corrupt. □

Proof of proposition 2:

Proof. First note that any policy π has average cumulative reward with respect to ub greater or equal than as with respect to R

$$\mathop{\mathbb{E}}_{\tau\sim\pi}\sum_{x\in\tau}R(x) \leq \mathop{\mathbb{E}}_{\tau\sim\pi}\sum_{x\in\tau}\mathrm{ub}(x), \tag{2}$$

because in general $\mathrm{ub}(x) \geq R(x)$ as the only difference between these functions is a substitution of upper bounds for some, possibly lower, values of R. In particular, since this is also true for R-optimal policies, this means that any ub-optimal policy π' will have average cumulative reward with respect to ub greater or equal than the R-optimal policy π^* has with respect to R:

$$\mathop{\mathbb{E}}_{\tau\sim\pi^*}\sum_{x\in\tau}R(x) \leq \mathop{\mathbb{E}}_{\tau\sim\pi^*}\sum_{x\in\tau}\mathrm{ub}(x) \leq \mathop{\mathbb{E}}_{\tau\sim\pi'}\sum_{x\in\tau}\mathrm{ub}(x). \tag{3}$$

Now, by (2), we get another bound on average cumulative reward of any policy π

$$\begin{aligned}\mathop{\mathbb{E}}_{\tau\sim\pi}\sum_{x\in\tau}\mathrm{ub}(x) + (\mathrm{lb}(x) - \mathrm{ub}(x)) &= \mathop{\mathbb{E}}_{\tau\sim\pi}\sum_{x\in\tau}\mathrm{lb}(x)\\ &\leq \mathop{\mathbb{E}}_{\tau\sim\pi}\sum_{x\in\tau}R(x).\end{aligned}$$

By moving the part in parentheses to the right side of the inequality we get

$$\mathop{\mathbb{E}}_{\tau\sim\pi}\sum_{x\in\tau}\mathrm{ub}(x) \leq \mathop{\mathbb{E}}_{\tau\sim\pi}\sum_{x\in\tau}R(x) + (\mathrm{ub}(x) - \mathrm{lb}(x)). \tag{4}$$

Using (3) and (4) for ub-optimal policies π' we get

$$\begin{aligned}\mathop{\mathbb{E}}_{\tau\sim\pi^*}\sum_{x\in\tau}R(x) &\leq \mathop{\mathbb{E}}_{\tau\sim\pi'}\sum_{x\in\tau}\mathrm{ub}(x)\\ &\leq \mathop{\mathbb{E}}_{\tau\sim\pi'}\sum_{x\in\tau}R(x) + (\mathrm{ub}(x) - \mathrm{lb}(x)).\end{aligned}$$

We get the final bound by moving the reward to the left hand side and taking a supremum over ub-optimal policies of both sides.

$$\sup_{\pi' \text{ ub-optimal}} \mathop{\mathbb{E}}_{\tau\sim\pi^*} \sum_{x\in\tau} R(x) - \mathop{\mathbb{E}}_{\tau\sim\pi'} \sum_{x\in\tau} R(x) \leq \sup_{\pi' \text{ ub-optimal}} \mathop{\mathbb{E}}_{\tau\sim\pi'} \sum_{x\in\tau} (\mathrm{ub}(x) - \mathrm{lb}(x)).$$

□

Proof of proposition 3:

Proof. Recall the inequality in (2), that is

$$\mathop{\mathbb{E}}_{\tau\sim\pi} \sum_{x\in\tau} \mathrm{lb}(x) \leq \mathop{\mathbb{E}}_{\tau\sim\pi} \sum_{x\in\tau} R(x).$$

This means that π^* is lb-optimal, because its average cumulative reward does not change, so it is still the greatest possible. Any other lb-optimal policy has to get at least as much average cumulative reward as π^* with respect to lb. Since it would get at least as much reward with respect to R, it is also R-optimal. □

B Reducing memory consumption

We cannot avoid to store all the corrupt states, as we need to substitute our approximations for the rewards received when encountering them. However, keeping all the non-corrupt states in memory is not strictly necessary, because they are only used to improve our approximation of lb. To reduce memory consumption, we can instead keep only a small set of them and use it to update the cached lb of all known corrupt states. We add newly encountered non-corrupt states to this set, but if it gets too large, we remove some states at random.

Since we always update our approximation of lb using newly encountered non-corrupt states, it converges to the correct value as long as a state giving the best bound is not encountered earlier than the corrupt state. Because of the stronger version of assumption 3, we can even expect the state giving the best bound to be in the very trajectory identifying the corrupt state. Because of this we do not expect any practical problems with this optimization. All of our experiments use this modification of algorithm 2; however, in the toy environments presented the size of the cached set was bigger than the state space, thus there was no practical effect.

Future work could further reduce memory consumption by keeping the information about the state space in different ways, for example using neural networks to approximate the required functions.