Risk Averse Robust Adversarial Reinforcement Learning

Xinlei Pan
Daniel Seita
Yang Gao
John Canny

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-164
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-164.html

December 1, 2019

Acknowledgement

I would like to sincerely thank Professor John Canny for providing the opportunity to work with him on my thesis and research projects. His kind guidance and patience provide me a great experience to develop my research skills in machine learning.

I would also like to thank Daniel Seita and Yang Gao for helping with my research and providing feedbacks on this work. Thanks for Professor Ronald Fearing for providing further feedbacks on the thesis as the second reader. Thank you all the lab members in the Canny group for the great time that we had together. Also thanks for all my friends at Berkeley for their support. Finally, I would like to thank my family for their support and love throughout the years when I study at Berkeley.

Risk Averse Robust Adversarial Reinforcement Learning

by Xinlei Pan

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

[illegible]

Professor John F. Canny
Research Advisor

5/14/19

(Date)

* * * * * * *

[illegible]

Professor Ronald Fearing
Second Reader

5/16/2019

(Date)

Abstract

Deep reinforcement learning has recently made significant progress in solving computer games and robotic control tasks. A known problem, though, is that policies overfit to the training environment and may not avoid rare, catastrophic events such as automotive accidents. A classical technique for improving the robustness of reinforcement learning algorithms is to train on a set of randomized environments, but this approach only guards against common situations. Recently, robust adversarial reinforcement learning (RARL) was developed, which allows efficient applications of random and systematic perturbations by a trained adversary. A limitation of RARL is that only the expected control objective is optimized; there is no explicit modeling or optimization of risk. Thus the agents do not consider the probability of catastrophic events (i.e., those inducing abnormally large negative reward), except through their effect on the expected objective. In this paper we introduce risk-averse robust adversarial reinforcement learning (RARARL), using a risk-averse protagonist and a risk-seeking adversary. We test our approach on a self-driving vehicle controller. We use an ensemble of policy networks to model risk as the variance of value functions. We show through experiments that a risk-averse agent is better equipped to handle a risk-seeking adversary, and experiences substantially fewer crashes compared to agents trained without an adversary.

Contents

1 Introduction

Reinforcement learning has demonstrated remarkable performance on a variety of sequential decision making tasks such as Go [36], Atari games [20], autonomous driving [35, 40], and continuous robotic control [9, 17]. Reinforcement learning (RL) methods fall under two broad categories: model-free and model-based. In model-free RL, the environment's physics are not modeled, and such methods require substantial environment interaction and can have prohibitive sample complexity [34]. In contrast, model-based methods allow for systematic analysis of environment physics, and in principle should lead to better sample complexity and more robust policies. These methods, however, have to date been challenging to integrate with deep neural networks and to generalize across multiple environment dimensions [22, 41], or in truly novel scenarios, which are expected in unrestricted real-world applications such as driving.

In this work, we focus on model-free methods, but include *explicit modeling of risk*. We additionally focus on a framework that includes an *adversary* in addition to the main (i.e., protagonist) agent. By modeling risk, we can train stronger adversaries and through competition, more robust policies for the protagonist (see Figure 1.1 for an overview). We envision this as enabling training of more robust agents in simulation and then using sim-to-real techniques [33] to generalize to real world applications, such as house-hold robots or autonomous driving, with high reliability and safety requirements. A recent algorithm combining robustness in reinforcement learning and the adversarial framework is robust adversarial reinforcement learning (RARL) [30], which trained a robust protagonist agent by having an adversary providing random and systematic attacks on input states and dynamics. The adversary is itself trained using reinforcement learning, and tries to minimize the long term expected reward while the protagonist tries to *maximize* it. As the adversary gets stronger, the protagonist experiences harder challenges.

RARL, along with similar methods [19], is able to achieve some robustness, but the level of variation seen during training may not be diverse enough to resemble the variety encountered in the real-world. Specifically, the adversary does not actively seek catastrophic outcomes as does the agent constructed in this paper. Without such experiences, the protagonist agent will not learn to guard against them. Consider autonomous driving: a car controlled by the protagonist may suddenly be hit by another car. We call this and other similar events *catastrophic* since they present

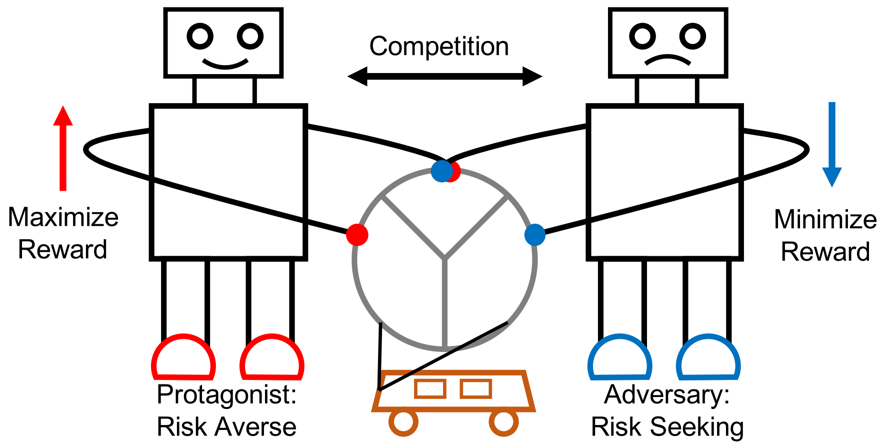

Figure 1.1: Risk averse robust adversarial reinforcement learning diagram: an autonomous driving example. Our framework includes two competing agents acting against each other, trying to drive a car (protagonist), or trying to slow or crash the car (adversary). We include a notion of risk modeling in policy learning. The risk-averse protagonist and risk-seeking adversarial agents learn policies to maximize or minimize reward, respectively. The use of the adversary helps the protagonist to effectively explore risky states.

extremely negative rewards to the protagonist, and should not occur under a reasonable policy. Such catastrophic events are highly unlikely to be encountered if an adversary only randomly perturbs the environment parameters or dynamics, or if the adversary only tries to minimize total reward. In this paper, we propose risk averse robust adversarial reinforcement learning (RARARL) for training risk averse policies that are simultaneously robust to dynamics changes. Inspired by [37], we model risk as the variance of value functions. To emphasize that the protagonist be averse to catastrophes, we design an asymmetric reward function (see Section 4.1): successful behavior receives a small positive reward, whereas catastrophes receive a very negative reward.

A robust policy should not only maximize long term expected reward, but should also select actions with low variance of that expected reward. Maximizing the expectation of the value function only maximizes the point estimate of that function without giving a guarantee on the variance. While [37] proposed a method to estimate that variance, it assumes that the number of states is limited, while we don't assume limited number of states and that assumption makes it impractical to apply it to real world settings where the number of possible states could be infinitely large. Here, we use an ensemble of Q-value networks to estimate variance. A similar technique was proposed in Bootstrapped DQNs [24] to assist exploration, though in our case, the primary purpose of the ensemble is to estimate variance.

We consider a two-agent reinforcement learning scenario (formalized in Section 3). Unlike in [30], where the agents performed actions simultaneously, here they take turns executing actions, so that one agent may take multiple steps to bring the environment in a more challenging state for the other. We seek to enable the adversarial agent to actively explore the parameter variation space, so that the perturbations are generated more efficiently. We use a discrete control task, autonomous driving with the TORCS [39] simulator, to demonstrate the benefits of RARARL.

2 Related Work

Reinforcement Learning with Adversaries. A recent technique in reinforcement learning involves introducing adversaries and other agents that can adjust the environment difficulty for a main agent. This has been used for robust grasping [29], simulated fighting [5], and RARL [30], the most relevant prior work to ours. RARL trains an adversary to appropriately perturb the environment for a main agent. The perturbations, though, were limited to a few parameters such as mass or friction, and the trained protagonist may be vulnerable to other variations.

The works of [19] and [26] proposed to add noise to state observations to provide adversarial perturbations, with the noise generated using fast gradient sign method [14]. However, they did not consider training an adversary or training risk averse policies. The work of [27] proposed to introduce Bayesian optimization to actively select environment variables that may induce catastrophes, so that models trained can be robust to these environment dynamics. However, they did not systematically explore dynamics variations and therefore the model may be vulnerable to changing dynamics even if it is robust to a handful of rare events.

Robustness and Safety in RL. More generally, robustness and safety have long been explored in reinforcement learning [7, 8, 23]. Chow et al. [8] proposed to model risk via constraint or chance constraint on the conditional value at risk (CVaR). This paper provided strong convergence guarantees but made strong assumptions: value and constrained value functions are assumed to be known exactly and to be differentiable and smooth. Risk is estimated by simply sampling trajectories which may never encounter adverse outcomes, whereas with sparse risks (as is the case here) adversarial sampling provides more accurate estimates of the probability of a catastrophe.

A popular ingredient is to enforce *constraints* on an agent during exploration [21] and policy updates [1, 15]. Alternative techniques include random noise injection during various stages of training [12, 31], injecting noise to the transition dynamics during training [32], learning when to reset [11] and even physically crashing as needed [13]. However, Rajeswaran et al. [32] requires training on a target domain and experienced performance degradation when the target domain has a different model parameter distribution from the source. We also note that in control theory, [3, 4] have provided theoretical analysis for robust control, though their focus lies in model based RL instead of model free RL. These prior techniques are orthogonal to our contribution, which relies on *model ensembles* to estimate variance.

Uncertainty-Driven Exploration. Prior work on exploration includes [25], which measures novelty of states using state prediction error, and [6], which uses pseudo counts to explore novel states. In our work, we seek to measure the risk of a state by the variance of value functions. The adversarial agent explores states with high variance so that it can create appropriate challenges for the protagonist.

Simulation to Real Transfer. Running reinforcement learning on physical hardware can be dangerous due to exploration and slow due to high sample complexity. One approach to deploying RL-trained agents safely in the real world is to experience enough environment variation during training in simulation so that the real-world environment looks just like another variation. These simulation-to-real techniques have grown popular, including domain randomization [33, 38] and dynamics randomization [28]. However, their focus is on transferring policies to the real world rather than training robust and risk averse policies.

3 Risk Averse Robust Adversarial RL

In this section, we formalize our risk averse robust adversarial reinforcement learning (RARARL) framework.

3.1 Two Players Reinforcement Learning

We consider the environment as a Markov Decision Process (MDP) $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma\}$, where $\mathcal{S}$ defines the state space, $\mathcal{A}$ defines the action space, $\mathcal{R}(s, a)$ is the reward function, $\mathcal{P}(s'|s, a)$ is the state transition model, and γ is the reward discount rate. There are two agents: the *protagonist P* and the *adversary A*.

Definition. Protagonist Agent. A protagonist P learns a policy π_P to maximize discounted expected reward $\mathbb{E}_{\pi_P}[\sum \gamma^t r_t]$. The protagonist should be risk averse, so we define the value of action a at state s to be

$$\hat{Q}_P(s, a) = Q_P(s, a) - \lambda_P Var_k[Q_P^k(s, a)], \tag{3.1}$$

where $\hat{Q}_P(s, a)$ is the modified Q function, $Q_P(s, a)$ is the original Q function, and $Var_k[Q_P^k(s, a)]$ is the variance of the Q function across k different models, and λ_P is a constant; The term $-\lambda_P Var_k[Q_P^k(s, a)]$ is called the *risk-averse* term thereafter, and encourages the protagonist to seek lower variance actions. The reward for P is the environment reward r_t at time t.

Definition. Adversarial Agent. An adversary A learns a policy π_A to *minimize* long term expected reward, or to maximize the negative discounted reward $\mathbb{E}_{\pi_A}[\sum -\gamma^t r_t]$. To encourage the adversary to systematically seek adverse outcomes, its modified value function for action selection is

$$\hat{Q}_A(s, a) = Q_A(s, a) + \lambda_A Var_k[Q_A^k(s, a)], \tag{3.2}$$

where $\hat{Q}_A(s, a)$ is the modified Q function, $Q_A(s, a)$ is the original Q function, $Var_k[Q_A^k(s, a)]$ is the variance of the Q function across k different models, and λ_A is a constant; the interaction between agents becomes a zero-sum game by setting $\lambda_A = \lambda_P$. The term $\lambda_A Var_k[Q_A^k(s, a)]$ is called the *risk-seeking* term thereafter. The reward of A is the negative of the environment reward $-r_t$, and its action space is the same as for the protagonist.

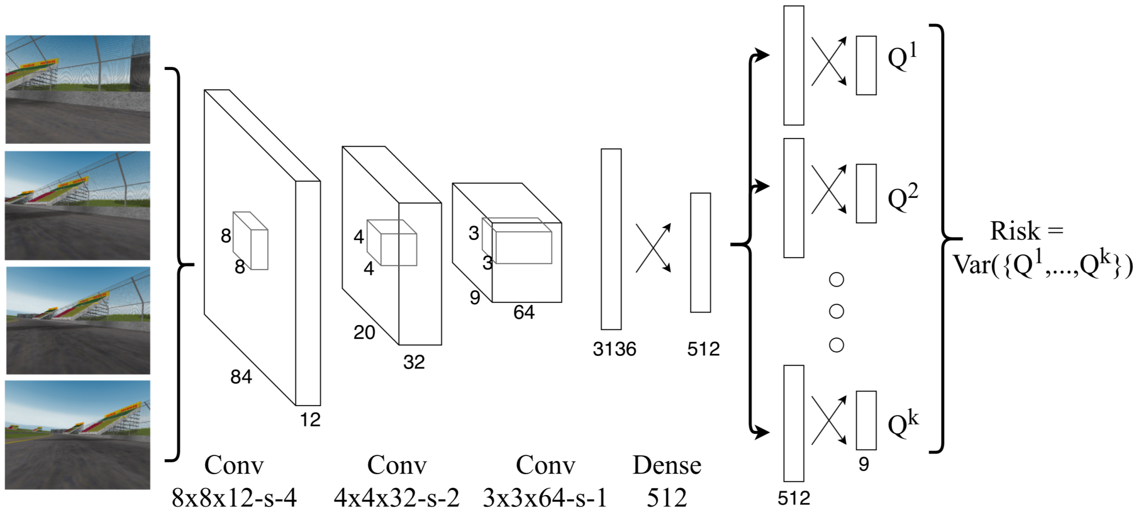

Figure 3.1: Our neural network design. (Notation: "s" indicates stride for the convolutional weight kernel, and two crossing arrows indicate dense layers.) The input is a sequence of four stacked observations to form an $(84 \times 84 \times 12)$-dimensional input. It is passed through three convolutional layers to obtain a 3136-dimensional vector, which is then processed through a dense layer. (All activations are ReLus.) The resulting 512-dimensional vector is copied and passed to k branches, which each process it through dense layers to obtain a state value vector $Q^i(s, \cdot)$. We apply the ensemble DQN framework for estimating the value function variance.

The necessity of having two agents working separately instead of jointly is to provide the adversary more power to create challenges for the protagonist. For example, in autonomous driving, a single risky action may not put the vehicle in a dangerous condition. In order to create a catastrophic event (e.g., a traffic accident) the adversary needs to be stronger. In our experiments (a vehicle controller with discrete control), the protagonist and adversary alternate full control of a vehicle, though our methods also apply to settings in [30], where the action applied to the environment is a sum of contributions from the protagonist and the adversary.

3.2 Reward Design and Risk Modeling

To train risk averse agents, we propose an asymmetric reward function design such that good behavior receives small positive rewards and risky behavior receives *very* negative rewards. See Section 4.1 and Equation 4.1 for details.

The risk of an action can be modeled by estimating the variance of the value function across different models trained on different sets of data. Inspired by [24], we estimate the variance of Q value functions by training multiple Q value networks in parallel. Hereafter, we use Q to denote the entire Q value network, and use Q^i to denote the i-th head of the multi-heads Q value network.[1] As shown in Figure 3.1, the network takes in input s, which consists of stacked frames of consecutive observations. It passes s through three shared convolutional layers, followed by one (shared) dense layer. After this, the input is passed to k different heads which perform one dense layer to obtain k action-value outputs: $\{Q^1(s,\cdot),\dots,Q^k(s,\cdot)\}$. Defining the mean as $\tilde{Q}(s,a) = \frac{1}{k}\sum_{i=1}^{k} Q^i(s,a)$, the variance of a single action a is,

$$Var_k(Q(s,a)) = \frac{1}{k}\sum_{i=1}^{k}(Q^i(s,a) - \tilde{Q}(s,a))^2, \tag{3.3}$$

where we use the k subscripts to indicate variance over k models, as in Equations 3.1 and 3.2. The variance in Equation 3.3 measures risk, and our goal is for the protagonist and adversarial agents to select actions with low and high variance, respectively.

At training time, when we sample one action using the Q values, we randomly choose one of k heads from Q^1 to Q^k, and use this head throughout one episode to choose the action that will be applied by the agent. When updating Q functions, our algorithm (like DQN [20]) samples a batch of data of size B from the replay buffer $\{(s,a,s',r,done)_t\}_{t=1}^{B}$ which, for each data point, includes the state, action, next state, reward, and task completion signal. Then we sample a k-sized mask. Each mask value is sampled using a Poisson distribution (modeling a true Bootstrap sample with replacement) instead of the Bernoulli distribution in [24] (sample without replacement). At test time, the mean value $\tilde{Q}(s,a)$ is used for selecting actions.

3.3 Risk Averse RARL

In our two-player framework, the agents take actions sequentially, not simultaneously: the protagonist takes m steps, the adversary takes n steps, and the cycle repeats. The experience of each agent is only visible to itself, which means each agent changes the environment transition dynamics for another agent. The Q learning Bellman equation is modified to be compatible with this case. Let the current and target value functions be Q_P and Q_P^* for the protagonist, and (respectively) Q_A and Q_A^* for the adversary. Given the current state and action pair (s_t, a_t), we denote actions executed by the protagonist as a_t^P and actions taken by the adversary as a_t^A. The target value functions are $Q_P(s_t^P, a_t^P) = r(s_t^P, a_t^P) + \sum_{i=1}^{n}\gamma^i r(s_{t+i}^A, a_{t+i}^A) + \gamma^{n+1}\max_a Q^*(s_{t+n+1}^P, a)$, and, similarly,$Q_A(s_t^A, a_t^A) = r(s_t^A, a_t^A) + \sum_{i=1}^{m}\gamma^i r(s_{t+i}^P, a_{t+i}^P) + \gamma^{m+1}\max_a Q^*(s_{t+m+1}^A, a)$. To increase train-

[1]We use Q and Q^i to represent functions that could apply to either the protagonist or adversary. If it is necessary to distinguish among the two agents, we add the appropriate subscript of P or A.

ing stability for the protagonist, we designed a *training schedule* Ξ of the adversarial agent. For the first ξ steps, only the protagonist agent takes actions. After that, for every m steps taken by the protagonist, the adversary takes n steps. The reason for this training schedule design is that we observed if the adversarial agent is added too early (e.g., right at the start), the protagonist is unable to attain any rewards. Thus, we let the protagonist undergo a sufficient amount of training steps to learn basic skills. The use of masks in updating Q value functions is similar to [24], where the mask is a integer vector of size equal to batch size times number of ensemble Q networks, and is used to determine which model is to be updated with the sample batch. Algorithm 1 describes our training algorithm.

Algorithm 1: Risk Averse RARL Training Algorithm

```
Result: Protagonist Value Function $Q_P$; Adversarial Value Function $Q_A$.
Input: Training steps T; Environment $env$; Adversarial Action Schedule $\Xi$; Exploration rate $\epsilon$;
  Number of models $k$.
Initialize: $Q_P^i$, $Q_A^i$ $(i = 1, \cdots, k)$; Replay Buffer $RB_P$, $RB_A$; Action choosing head $H_P$,
  $H_A \in [1, k]$; t = 0; Training frequency $f$; Poisson sample rate $q$;
while $t < T$ do
    Choose Agent $g$ from $\{A(Adversarial\ agent), P(Protagonist\ agent)\}$ according to $\Xi$ ;
    Compute $\hat{Q}_g(s, a)$ according to (3.1) and (3.2) ;
    Select action according to $\hat{Q}_g(s, a)$ by applying $\epsilon$-greedy strategy ;
    Excute action and get $obs, reward, done$;
    $RB_g = RB_g \cup \{(obs, reward, done)\}$;
    if $t \% f = 0$ then
        Generate mask $M \in \mathbb{R}^k \sim Poisson(q)$;
        Update $Q_P^i$ with $RB_P$ and $M_i$, $i = 1, 2, ..., k$;
        Update $Q_A^i$ with $RB_A$ and $M_i$, $i = 1, 2, ..., k$;
    if $done$ then
        update $H_p$ and $H_a$ by randomly sampling integers from 1 to $k$ ;
        reset $env$;
    t = t + 1;
```

4 Experiments

We evaluated models trained by RARARL on an autonomous driving environment, TORCS [39]. Autonomous driving has been explored in recent contexts for policy learning and safety [2, 10, 18] and is a good testbed for risk-averse reinforcement learning since it involves events (particularly crashes) that qualify as catastrophes.

4.1 Simulation Environment

For experiments, we use the Michigan Speedway environment in TORCS [39], which is a round way racing track; see Figure 5.4 for sample observations. The states are $(84 \times 84 \times 3)$-dimensional RGB images. The vehicle can execute nine actions: (1) move left and accelerate, (2) move ahead and accelerate, (3) move right and accelerate, (4) move left, (5) do nothing, (6) move right, (7) move left and decelerate, (8) move ahead and decelerate, (9) move right and decelerate.

We next define our asymmetric reward function. Let v be the magnitude of the speed, α be the angle between the speed and road direction, p be the distance of the vehicle to the center of the road, and w be the road width. We additionally define two binary flags: $\mathbf{1}_{\text{st}}$ and $\mathbf{1}_{\text{da}}$, with $\mathbf{1}_{\text{st}} = 1$ if the vehicle is stuck (and 0 otherwise) and $\mathbf{1}_{\text{da}} = 1$ if the vehicle is damaged (and 0 otherwise). Letting $C = \left\lceil \frac{\mathbf{1}_{\text{st}} + \mathbf{1}_{\text{da}}}{2} \right\rceil$, the reward function is defined as:

$$r = \beta v \left(\cos(\alpha) - |\sin(\alpha)| - \frac{2p}{w} \right) (1 - \mathbf{1}_{\text{st}})(1 - \mathbf{1}_{\text{da}}) + r_{\text{cat}} \cdot C \tag{4.1}$$

with the intuition being that $\cos(\alpha)$ encourages speed direction along the road direction, $|\sin(\alpha)|$ penalizes moving across the road, and $\frac{2p}{w}$ penalizes driving on the side of the road. We set the *catastrophe reward* as $r_{\text{cat}} = -2.5$ and set $\beta = 0.025$ as a tunable constant which ensures that the magnitude of the non-catastrophe reward is significantly less than that of the catastrophe reward. The catastrophe reward measures collisions, which are highly undesirable events to be avoided. We note that constants $\lambda_P = \lambda_A$ used to blend reward and variance terms in the risk-augmented Q-functions in Equations 3.1 and 3.2 were set to 0.1.

We consider two additional reward functions to investigate in our experiments. The *total progress reward* excludes the catastrophe reward:

$$r = \beta v \left(\cos(\alpha) - |\sin(\alpha)| - \frac{2p}{w}\right)(1 - \mathbf{1}_{\text{st}})(1 - \mathbf{1}_{\text{da}}), \tag{4.2}$$

and the *pure progress reward* is defined as

$$r = \beta v \Big(\cos(\alpha) - |\sin(\alpha)|\Big)(1 - \mathbf{1}_{\text{st}})(1 - \mathbf{1}_{\text{da}}). \tag{4.3}$$

The total progress reward considers both moving along the road and across the road, and penalizes large distances to the center of the road, while the pure progress only measures the distance traveled by the vehicle, regardless of the vehicle's location. The latter can be a more realistic measure since vehicles do not always need to be at the center of the road.

4.2 Baselines and Our Method

All baselines are optimized using Adam [16] with learning rate 0.0001 and batch size 32. In all our ensemble DQN models, we trained with 10 heads since empirically that provided a reasonable balance between having enough models for variance estimation but not so much that training time would be overbearing. For each update, we sampled 5 models using Poisson sampling with $q = 0.03$ to generate the mask for updating Q value functions. We set the training frequency as 4, the target update frequency as 1000, and the replay buffer size as 100,000. For training DQN with an epsilon-greedy strategy, the ϵ decreased linearly from 1 to 0.02 from step 10,000 to step 500,000. The time point to add in perturbations is $\xi = 550,000$ steps, and for every $m = 10$ steps taken by protagonist agent, the random agent or adversary agent will take $n = 1$ step.

Vanilla DQN. The purpose of comparing with vanilla DQN is to show that models trained in one environment may overfit to specific dynamics and fail to transfer to other environments, particularly those that involve random perturbations. We denote this as **`dqn`**.

Ensemble DQN. Ensemble DQN tends to be more robust than vanilla DQN. However, without being trained on different dynamics, even Ensemble DQN may not work well when there are adversarial attacks or simple random changes in the dynamics. We denote this as **`bsdqn`**.

Ensemble DQN with Random Perturbations Without Risk Averse Term. We train the protagonist and provide random perturbations according to the schedule Ξ. We do not include the variance guided exploration term here, so only the Q value function is used for choosing actions. The schedule Ξ is the same as in our method. We denote this as **`bsdqnrand`**.

Ensemble DQN with Random Perturbations With the Risk Averse Term. We only train the protagonist agent and provide random perturbations according to the adversarial training schedule Ξ. The protagonist selects action based on its Q value function and the risk averse term. We denote this as **`bsdqnrandriskaverse`**.

Ensemble DQN with Adversarial Perturbation. This is to compare our model with [30]. For a fair comparison, we also use Ensemble DQN to train the policy while the variance term is not used as either risk-averse or risk-seeking term in either agents. We denote this as **`bsdqnadv`**.

Our method. In our method, we train both the protagonist and the adversary with Ensemble DQN. We include here the variance guided exploration term, so the Q function and its variance across different models will be used for action selection. The adversarial perturbation is provided according to the adversarial training schedule Ξ. We denote this as **`bsdqnadvriskaverse`**.

5 Evaluation

To evaluate robustness of our trained models, we use the same trained models under different testing conditions, and evaluate using the previously-defined reward classes of total progress (Equation 4.2), pure progress (Equation 4.3), and additionally consider the reward of catastrophes. We present three broad sets of results: (1) **No perturbations**. (Figure 5.1) We tested all trained models from Section 4.2 without perturbations. (2) **Random perturbations**. (Figure 5.2) To evaluate the robustness of trained models in the presence of random environment perturbations, we benchmarked all trained models using random perturbations. For every 10 actions taken by the main agent, 1 was taken at random. (3) **Adversarial Perturbations**. (Figure 5.3) To test the ability of our models to avoid catastrophes, which normally require deliberate, non-random perturbations, we test with a trained adversarial agent which took 1 action for every 10 taken by the protagonist.

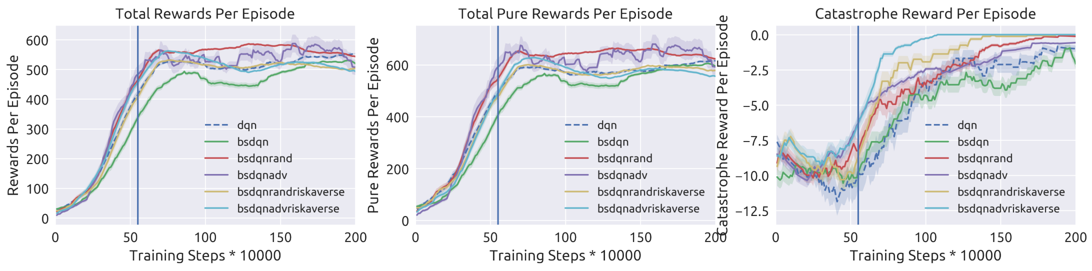

Figure 5.1: Testing all models without attacks or perturbations. The reward is divided into distance related reward (left subplot), progress related reward (middle subplot). We also present results for catastrophe reward *per episode* (right subplot). The blue vertical line indicates the beginning of adding perturbations during training. All legends follow the naming convention described in Section 4.2.

All subplots in Figures 5.1, 5.2, and 5.3 include a vertical blue line at 0.55 million steps indicating when perturbations were first applied during training (if any). Before 0.55 million steps, we allow enough time for protagonist agents to be able to drive normally. We choose 0.55 million steps because the exploration rate decreases to 0.02 at 0.50 million steps, and we allow additional 50000 steps for learning to stabilize.

Does adding adversarial agent's perturbation affect the robustness? In Table 5.1, we compare the robustness of all models by their catastrophe rewards. The results indicate that adding

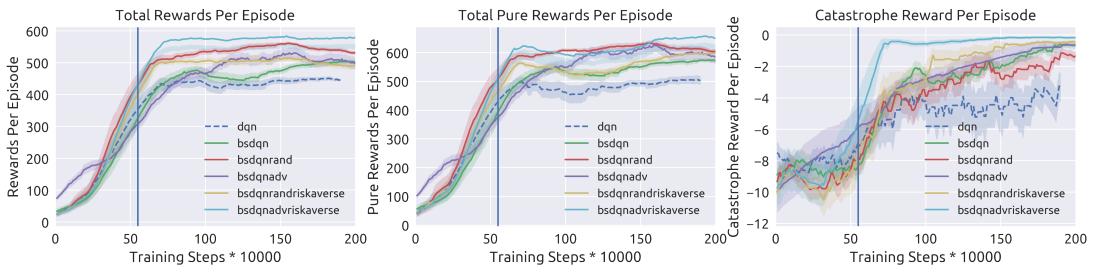

Figure 5.2: Testing all models with random attacks. The three subplots follow the same convention as in Figure 5.1.

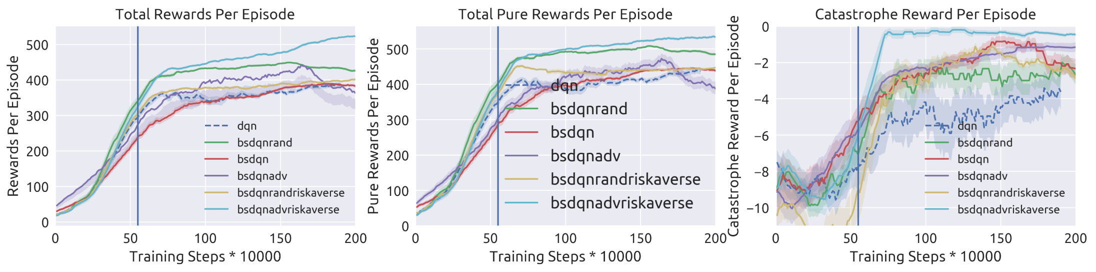

Figure 5.3: Testing all models with adversarial attack. The three subplots follow the same convention as in Figure 5.1.

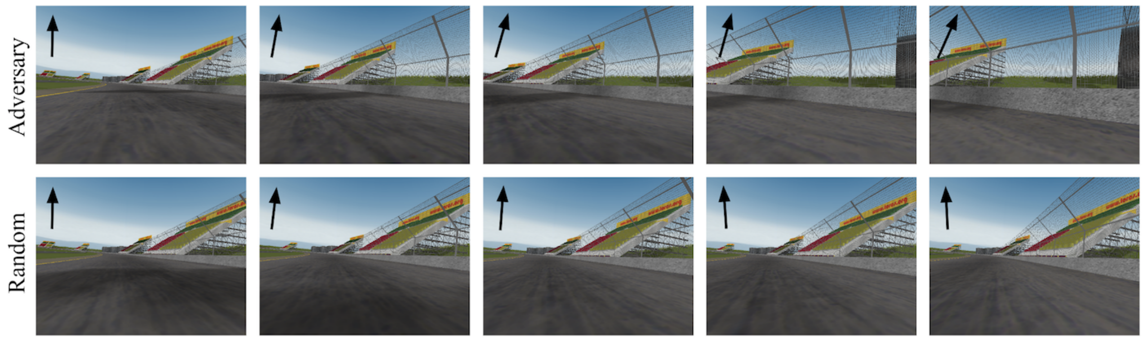

Figure 5.4: Two representative (subsampled) sequences of states in TORCS for a trained protagonist, with either a trained adversary (top row) or random perturbations (bottom row) affecting the trajectory. The overlaid arrows in the upper left corners indicate the direction of the vehicle. The top row indicates that the trained adversary is able to force the protagonist to drive towards the right and into the wall (i.e., a catastrophe). Random perturbations cannot affect the protagonist's trajectory to the same extent because many steps of deliberate actions in one direction are needed to force a crash.

perturbations improves a model's robustness, especially to adversarial attacks. DQN trained with random perturbations is not as robust as models trained with adversarial perturbations, since random perturbations are weaker than adversarial perturbations.

Table 5.1: Robustness of Models Measured by Average Best Catastrophe Reward Per Episode (Higher is better)

Exp	Normal	Random Perturb	Adv. Perturb
dqn	-0.80	-3.0	-4.0
bsdqn	-0.90	-1.1	-2.5
bsdqnrand	-0.10	-1.0	-2.1
bsdqnadv	-0.30	-0.5	-1.0
bsdqnrandriskaverse	-0.09	-0.4	-2.0
bsdqnadvriskaverse	**-0.08**	**-0.1**	**-0.1**

How does the risk term affect the robustness of the trained models? As shown in Figures 5.2 and 5.3, models trained with the risk term achieved better robustness under both random and adversarial perturbations. We attribute this to the risk term encouraging the adversary to aggressively explore regions with high risk while encouraging the opposite for the protagonist.

How do adversarial perturbations compare to random perturbations? A trained adversarial agent can enforce stronger perturbations than random perturbations. By comparing Figure 5.2 and Figure 5.3, we see that the adversarial perturbation provides stronger attacks, which causes the reward to be lower than with random perturbations.

We also visualize an example of the differences between a trained adversary and random perturbations in Figure 5.4, which shows that a trained adversary can force the protagonist (a vanilla DQN model) to drive into a wall and crash.

6 Conclusion

We show that by introducing a notion of risk averse behavior, a protagonist agent trained with a learned adversary experiences substantially fewer catastrophic events during test-time rollouts as compared to agents trained without an adversary. Furthermore, a trained adversarial agent is able to provide stronger perturbations than random perturbations and can provide a better training signal for the protagonist as compared to providing random perturbations. In future work, we will apply RARARL in other safety-critical domains, such as in surgical robotics.

Acknowledgments

Xinlei Pan is supported by Berkeley Deep Drive. I would like to thank Daniel Seita and Yang Gao for helping me with this project.

Bibliography

1. J. Achiam, D. Held, A. Tamar, and P. Abbeel. “Constrained Policy Optimization”. In: *International Conference on Machine Learning (ICML)*. 2017.
2. A. Amini, L. Paull, T. Balch, S. Karaman, and D. Rus. “Learning Steering Bounds for Parallel Autonomous Systems”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.
3. A. Aswani, P. Bouffard, and C. Tomlin. “Extensions of learning-based model predictive control for real-time application to a quadrotor helicopter”. In: *American Control Conference (ACC), 2012*. IEEE. 2012, pp. 4661–4666.
4. A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin. “Provably safe and robust learning-based model predictive control”. In: *Automatica* 49:5, 2013, pp. 1216–1226.
5. T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch. “Emergent Complexity via Multi-Agent Competition”. In: *International Conference on Learning Representations (ICLR)*. 2018.
6. M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. “Unifying count-based exploration and intrinsic motivation”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 1471–1479.
7. S. Carpin, Y.-L. Chow, and M. Pavone. “Risk Aversion in Finite Markov Decision Processes Using Total Cost Criteria and Average Value at Risk”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2016.
8. Y. Chow, M. Ghavamzadeh, L. Janson, and M. Pavone. “Risk-Constrained Reinforcement Learning with Percentile Risk Criteria”. In: *Journal of Machine Learning Research*, 2018.
9. Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. “Benchmarking Deep Reinforcement Learning for Continuous Control”. In: *International Conference on Machine Learning (ICML)*. 2016.
10. S. Ebrahimi, A. Rohrbach, and T. Darrell. “Gradient-free Policy Architecture Search and Adaptation”. In: *Conference on Robot Learning (CoRL)*. 2017.
11. B. Eysenbach, S. Gu, J. Ibarz, and S. Levine. “Leave no Trace: Learning to Reset for Safe and Autonomous Reinforcement Learning”. In: *International Conference on Learning Representations (ICLR)*. 2018.
12. M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg. “Noisy Networks for Exploration”. In: *International Conference on Learning Representations (ICLR)*. 2018.

13. D. Gandhi, L. Pinto, and A. Gupta. “Learning to Fly by Crashing”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017.

14. I. J. Goodfellow, J. Shlens, and C. Szegedy. “Explaining and harnessing adversarial examples”. In: *International Conference on Learning Representations (ICLR)*. 2015.

15. D. Held, Z. McCarthy, M. Zhang, F. Shentu, and P. Abbeel. “Probabilistically Safe Policy Transfer”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

16. D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. In: *International Conference on Learning Representations (ICLR)*. 2015.

17. T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. “Continuous control with deep reinforcement learning”. In: *International Conference on Learning Representations (ICLR)*. 2016.

18. G.-H. Liu, A. Siravuru, S. Prabhakar, M. Veloso, and G. Kantor. “Learning End-to-End Multimodal Sensor Policies for Autonomous Navigation”. In: *Conference on Robot Learning (CoRL)*. 2017.

19. A. Mandlekar, Y. Zhu, A. Garg, L. Fei-Fei, and S. Savarese. “Adversarially Robust Policy Learning: Active Construction of Physically-Plausible Perturbations”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017.

20. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518:7540, 2015, pp. 529–533.

21. T. M. Moldovan and P. Abbeel. “Safe Exploration in Markov Decision Processes”. In: *International Conference on Machine Learning (ICML)*. 2012.

22. A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. “Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

23. R. Neuneier and O. Mihatsch. “Risk Sensitive Reinforcement Learning”. In: *Neural Information Processing Systems (NIPS)*. 1998.

24. I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. “Deep exploration via bootstrapped DQN”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 4026–4034.

25. D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. “Curiosity-driven exploration by self-supervised prediction”. In: *International Conference on Machine Learning (ICML)*. Vol. 2017. 2017.

26. A. Pattanaik, Z. Tang, S. Liu, G. Bommannan, and G. Chowdhary. “Robust Deep Reinforcement Learning with Adversarial Attacks”. In: *arXiv preprint arXiv:1712.03632*, 2017.

27. S. Paul, K. Chatzilygeroudis, K. Ciosek, J.-B. Mouret, M. A. Osborne, and S. Whiteson. “Alternating Optimisation and Quadrature for Robust Control”. In: *AAAI 2018-The Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

28. X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

29. L. Pinto, J. Davidson, and A. Gupta. "Supervision via Competition: Robot Adversaries for Learning Tasks". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

30. L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta. "Robust Adversarial Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*, 2017.

31. M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz. "Parameter Space Noise for Exploration". In: *International Conference on Learning Representations (ICLR)*. 2018.

32. A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine. "Epopt: Learning robust neural network policies using model ensembles". In: *International Conference on Learning Representations (ICLR)*. 2017.

33. F. Sadeghi and S. Levine. "CAD2RL: Real Single-Image Flight Without a Single Real Image". In: *Robotics: Science and Systems*. 2017.

34. J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. "Trust Region Policy Optimization". In: *International Conference on Machine Learning (ICML)*. 2015.

35. S. Shalev-Shwartz, S. Shammah, and A. Shashua. "Safe, multi-agent, reinforcement learning for autonomous driving". In: *arXiv preprint arXiv:1610.03295*, 2016.

36. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529:7587, 2016, pp. 484–489.

37. A. Tamar, D. Di Castro, and S. Mannor. "Learning the variance of the reward-to-go". In: *Journal of Machine Learning Research* 17:13, 2016, pp. 1–36.

38. J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017.

39. B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner. "Torcs, the open racing car simulator". In: *Software available at http://torcs. sourceforge. net*, 2000.

40. Y. You, X. Pan, Z. Wang, and C. Lu. "Virtual to Real Reinforcement Learning for Autonomous Driving". In: *British Machine Vision Conference*, 2017.

41. Y. Zhu, Z. Wang, J. Merel, A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramar, R. Hadsell, N. de Freitas, and N. Heess. "Reinforcement and Imitation Learning for Diverse Visuomotor Skills". In: *Robotics: Science and Systems (RSS)*. 2018.