Risk Averse Robust Adversarial Reinforcement Learning

Extended Abstract

Xinlei Pan
UC Berkeley
Berkeley, CA
xinleipan@berkeley.edu

John Canny
UC Berkeley
Berkeley, CA
jfc@cs.berkeley.edu

ABSTRACT

Robot controllers need to deal with a variety of uncertainty and still perform well. Reinforcement learning and deep networks often suffer from brittleness and e.g. difficulty on transfer from simulation to real environments. Recently, robust adversarial reinforcement learning (RARL) was developed which allows application of random and systematic perturbations by an adversary. A limitation of previous work is that only the expected control objective is optimized. i.e. there is no explicit modeling or optimization of risk. Thus it is not possible to control the probability of catastrophic events. In this paper we introduce risk-averse robust adversarial RL, using a risk-averse main agent and a *risk-seeking* adversary. We show through experiments in vehicle control that a risk-averse agent is better equipped to handle a risk-seeking adversary, and allows fewer catastrophic outcomes.

KEYWORDS

Risk averse reinforcement learning, Adversarial learning, Robust reinforcement learning

ACM Reference Format:
Xinlei Pan and John Canny. 2018. Risk Averse Robust Adversarial Reinforcement Learning: Extended Abstract. In *Proceedings of ACM Computer Science in Cars Symposium (CSCS'18).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Reinforcement learning has demonstrated remarkable performance on a variety of sequential decision making tasks such as Go [6] and Atari games [2]. In order to use reinforcement learning trained agents in real world applications such as house-hold robots or autonomous driving vehicle, the agents need a very high level of robustness and safety. Previous work on robust RL has used random perturbation of a limited number of parameters to change the dynamics [5]. In [1], gradient-based adversarial perturbations were introduced during training. In [4] an adversary agent was itself trained using reinforcement learning. This approach is more general than [1], and includes both random and systematic adversaries that directly provide adversarial attack on the input states and dynamics. However, these methods only achieve limited diversity of dynamics, which may not be diverse enough to resemble the real world variety. These methods also mainly consider the robustness of models instead of considering the robustness and risk averse ability of the trained model. In this paper, we consider the task of training risk averse robust reinforcement policies. We model the risk as variance of value functions, inspired by [7]. In order to emphasize that the agent should be averse to extreme catastrophe, we design the reward function to be asymmetric. A robust policy should not only maximize long term expected cumulative reward, but should also select actions with small variance of that expectation. Here, we use emsembles of Q value network to estimate variance of Q values. The work in [3] proposes a similar technique to help exploration. We consider a scenario where there are two agents working together and combating each other. We then propose to train risk averse robust adversarial reinforcement learning policies (RARARL).

CSCS'18, September 2018, Munich, Germany

2 RISK AVERSE ROBUST ADVERSARIAL REINFORCEMENT LEARNING

Two Player Reinforcement Learning. We consider the environment as a Markov Decision Process (MDP) $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma\}$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is 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 in this game, one agent is called the *protagonist* agent (denoted by P) which strives to learn a policy π_P to maximize cumulative expected reward $\mathbb{E}_{\pi_P}[\sum \gamma^t r_t]$. The other agent is called the *adversarial* agent (denoted by A), which strives to learn a policy π_A to pit the protagonist agent and minimize the cumulative expected reward. In our implementation, the adversarial agent gets the negative of the environmental reward $-r$. The protagonist agent should be risk averse, where we model the risk as variance of the value function (introduced later on), so the value of action a at

state s is modified as, $\hat{Q}_P(s,a) = Q_P(s,a) - \lambda Var_m[Q_P^m(s,a)]$, where $\hat{Q}_P(s,a)$ is the modified Q function, and $Q_P(s,a)$ is the original Q function, and $Var_m[Q_P^m(s,a)]$ is the variance of Q function across m different models, and λ is a multiplication constant, and we choose its value to be 0.1. The term $-\lambda Var_m[Q_P^m(s,a)]$ is called the risk-averse term thereafter. In order for the adversarial agent to be risk seeking so as the provide more challenges for the protagonist agent, the modified value function for action selection is $\hat{Q}_A(s,a) = Q_A(s,a) + \lambda Var_m[Q_A^m(s,a)]$ where λ is a multiplication constant, and the value is 1. The term $\lambda Var_m[Q_P^m(s,a)]$ is called the risk-seeking term thereafter. The action space of this agent is the same as protagonist agent. The two agents take actions in turn during training. **Asymmetric Reward Design**. In order to train risk averse agents, we propose to design asymmetric reward function. Namely, good behavior will receive only small positive reward, while risk behavior will receive very negative reward, so that the trained agent can be risk averse and robust. **Risk Modeling using Value Variance**. The risk of a certain action can be modeled by estimating the variance of value function across different models trained on different sets of data. Inspired by [3], we estimate variance of Q value functions by training multiple Q value network together, and take their Q value variance as risk measurement. The training process for a single agent is the same as in [3].

3 EXPERIMENTS

Simulation Environments. For discrete control, we used the car racing TORCS environment [8]. We use the Michigan Speedway environment in TORCS. The vehicle takes actions in a discrete space consisting of 9 actions, which are combinations of move ahead, accelerate, and turn left/right. The reward function is defined to proportional to the speed along the road direction and also penalize collision as catastrophe. The catastrophe reward is much more negative than normal reward.

Baselines and Our Method. **Vanilla DQN**. The purpose of comparing with vanilla DQN is to show that models trained in one environment may overfit to that specific dynamics and fail to transfer to other dynamics. We denote this experiment as **`dqn`**. **Ensemble DQN**. Ensemble DQN tends to be more robust than vanilla DQN since it gets the vote from all ensemble policies. However, without being trained on a different dynamics, even Ensemble DQN may not work well when there is adversarial attacks or simple random dynamics change. We denote this experiment as **`bsdqn`**. **Ensemble DQN with Random Perturbations Without Risk Averse Term**. We train the protagonist agent and provide random perturbation. However, for comparison reasons, we didn't include here variance guided exploration term. Namely, only the Q value function will be used for choosing actions. We denote this experiment as **`bsdqnrand`**. **Ensemble DQN with Random Perturbations With the Risk Averse Term**. We only train the protagonist agent and provide random perturbations. The Protagonist agent will select action based on its Q value function and the risk averse term. We denote this experiment as **`bsdqnrandriskaverse`**. **Ensemble DQN with Adversarial Perturbation**. This is to compare our model with [4]. To make 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 agent. We denote this experiment as **`bsdqnadv`**. **Our method**. In our method, we train both protagonist agent and adversarial agent with Ensemble DQN. We will include here the variance guided exploration term. Therefore, the Q function and its variance across different models will be used for action selection. We denote this experiment as **`bsdqnadvriskaverse`**. **Evaluation**. First, testing without perturbations. We tested all trained models without perturbations. Second, testing with random perturbations. Specifically, we tested these models by randomly taking 1 action for every 10 actions taken by the main agent. Third, testing with Adversarial Perturbations. Random perturbations may not result in catastrophe for the vehicle. Therefore, we also tested all models with our trained adversarial agent. Specifically, we tested these models by taking 1 action decided by the adversarial agent for every 10 actions taken by the main agent. The results are shown in figure 1.

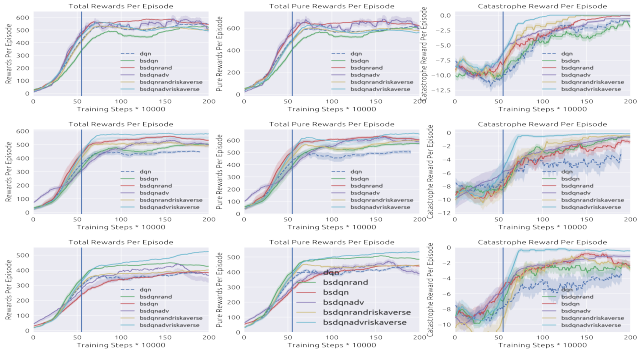

Figure 1: Testing all models without attack, with random attack, and with adversarial attack (from top to bottom). The reward is divided into distance related reward (left), progress related reward (middle). The result for catastrophe reward per episode is present in the last image.

For all results curve, there is a vertical blue line corresponding to 0.55 million steps showing the beginning of the added perturbations during training. Either the random perturbation or adversarial perturbation is added.

4 CONCLUSION

We show that by introducing a notion of risk averse behavior, training agents with adversarial agent achieves significantly less catastrophe than agents trained without adversarial attack. Trained adversarial agent is also able to provide perturbations stronger than random perturbations.

REFERENCES

[1] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Li Fei-Fei, and Silvio Savarese. 2017. Adversarially Robust Policy Learning: Active Construction of Physically-Plausible Perturbations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[3] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. 2016. Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems*. 4026–4034.

[4] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. 2017. Robust Adversarial Reinforcement Learning. *International Conference on Machine Learning (ICML)* (2017).

[5] Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. 2017. Epopt: Learning robust neural network policies using model ensembles. In *International Conference on Learning Representations (ICLR)*.

[6] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[7] Aviv Tamar, Dotan Di Castro, and Shie Mannor. 2016. Learning the variance of the reward-to-go. *Journal of Machine Learning Research* 17, 13 (2016), 1–36.

[8] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. 2000. Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net* (2000).