Policy Learning under Uncertainty and Risk

by

Yudong Luo

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2024

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Mohammad Ghavamzadeh Senior Principal Scientist Amazon
Supervisor(s):	Pascal Poupart Professor, School of Computer Science University of Waterloo
Internal Member:	Yuying Li Professor, School of Computer Science University of Waterloo
	Hongyang Zhang Assistant Professor, School of Computer Science University of Waterloo
Internal-External Member:	Yash Vardhan Pant Assistant Professor, Dept. of Electrical and Computer Engineering University of Waterloo

Author's Declaration

Statement of Contributions

This thesis consists of three of the author's research works during his PhD at the University of Waterloo, including Luo et al. (2022), Luo et al. (2023), and Luo et al. (2024). The first work was published at ICLR 2022. The second work was published at NeurIPS 2023. The third work was published at RLC 2024. Yudong Luo is the first author of these three papers, who proposed the ideas, conducted the experiments and wrote the papers.

Abstract

Recent years have seen a rapid growth of reinforcement learning (RL) research. In year 2015, deep RL achieved superhuman performance in Atari video games (Mnih et al., 2015). In year 2016, the Alpha Go developed by Google DeepMind beat Lee Sedol, one of the top Go players in South Korea. In year 2022, OpenAI released ChatGPT 3.5, a powerful large language model, which is fine-tuned by RL algorithms. Traditional RL considers the problem that an agent interacts with an environment to acquire a good policy. The performance of the policy is usually evaluated by the expected value of total discounted rewards (or called return) collected in the environment. However, the mostly studied domains (including the three mentioned above) are usually deterministic or contain less randomness. In many real world applications, the domains are highly stochastic, thus agents need to perform decision making under uncertainty. Due to the randomness of the environment, another natural consideration is to minimize the risk, since only maximizing the expected return may not be sufficient. For instance, we want to avoid huge financial loss in portfolio management, which motivates the mean variance trade off.

In this thesis, we focus on the problem of policy learning under uncertainty and risk. This requires the agent to quantify the intrinsic uncertainty of the environment and be risk-averse in specific cases, instead of only caring for the mean of the return.

To quantify the intrinsic uncertainty, in this thesis, we stick to the distributional RL method. Due to the stochasticity of the environment dynamic and also stochastic polices, the future return that an agent can get at a state is naturally a random variable. Distributional RL aims to learn the full value distribution of this random variable. Usually, the value distribution is represented by its quantile function. However, the quantile functions learned by existing algorithms suffer from limited representation ability or quantile crossing issue, which is shown to hinder policy learning and exploration. We propose a new learning algorithm to directly learn a monotonic, smooth, and continuous quantile representation, which provides much flexibility for value distribution learning in distributional RL.

For risk-averse policy learning, we study two common types of risk measure, i.e., measure of variability, e.g., variance, and tail risk measure, e.g., conditional value at risk (CVaR). 1) Mean variance trade off is a classic yet popular problem in RL. Traditional methods directly restrict the total return variance. Recent methods restrict the per-step reward variance as a proxy. We thoroughly examine the limitations of these variance-based methods in the policy gradient approach, and propose to use an alternative measure of variability, Gini deviation, as a substitute. We study various properties of this new risk measure and derive a policy gradient algorithm to minimize it. 2) CVaR is another popular risk measure for risk-averse RL. However, RL algorithms utilizing policy gradients to optimize CVaR face significant challenges with sample inefficiency, hindering their practical applications. This inefficiency stems from two main facts: a focus on tail-end performance that overlooks many sampled trajectories, and the potential of gradient vanishing when the lower tail of the return distribution is overly flat. To address these challenges, we start from an insight that in many scenarios, the risk-averse behavior is only required in a subset of states, and propose a simple mixture policy parameterization. This method integrates a risk-neutral policy with an adjustable policy to form a risk-averse policy. By employing this strategy, all collected trajectories can be utilized for policy updating, and the issue of vanishing gradients is counteracted by stimulating higher returns through the risk-neutral component, thus the sample efficiency is significantly improved.

Acknowledgments

First and foremost, I would like to thank my supervisor Pascal Poupart for his support and encouragement during my PhD. I still remember the time I met him in person in the AAAI 2020 conference in New York, where I expressed my interests in reinforcement learning and the willing of pursuing a PhD with him. Though it was a hard time to start the PhD during the pandemic, he still offered a lot of guidance on my research via online meetings. He taught me how to write a good academic paper, how to give a clear presentation, and how to properly defend my ideas in the rebuttal. In short, his mentorship has not only shaped my academic development but also provided me with mental resilience to overcome challenges. I would also like to thank Professor Yuying Li and Hongyang Zhang for serving in my supervisory committee and providing feedback of my works. I greatly appreciate Dr. Mohammad Ghavamzadeh and Professor Yash Vardhan Pant for being my thesis examiners.

My research works during PhD may not be done without the help of my collaborators, Guiliang, Han, Jiaqi, Yangchen, Ziyuan. Especially, Yangchen shared a lot of experience on doing research during his PhD, which inspired me a lot. I am also grateful for Professor Ruodu Wang for his kind help. He is always willing to answer my questions on risk-averse problems with patience.

Four years have passed in what feels like both a long and short journey. I am greatly thankful for the friends I have met here—Adam, Guiwen, Haolin, Luwei, Minghao, Shuhui, Yanting, and Yimiao. We traveled together, celebrated moments together, making life in Waterloo far more vibrant than I could have imagined. Specially, I am immensely grateful for Susu and Xiangyu, who despite being thousands of kilometers away, consistently offer support whenever I find myself feeling lost, upset, frustrated, or struggling. I am also grateful for my old friends in Vancouver, Heng, and Mohan, with whom every visit yields cherished memories.

I would also like to extend my thanks to the swimming pool and gym facilities at PAC. They have been instrumental in refreshing my mind and strengthening my body.

Finally, thanks to my parents for giving me life and raising me up. Distance may separate us physically, but our hearts remain connected.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Reinforcement learning (RL) considers the problem that an agent interacts with an environment to maximize its expected cumulative rewards (or called return) (Sutton & Barto, 2018). This interaction is usually modeled as a Markov decision process (MDP) as shown in Fig. 1.1, but the agent is initially uncertain about the true dynamics and rewards of the MDP. Each time, the agent performs an action, receives a reward, and moves to the next state. With this data collected, the agent will be more and more certain of the environment and learn which action leads to higher return. Usually, the expected future return the agent can get by taking an action at a state is called the Q value.

RL algorithms have shown super human performance in several domains, e.g., Atari video games (Mnih et al., 2015), and Go. We should notice that these domains are deterministic, and the algorithms used to master these domains are risk-neutral, i.e., the goal is to maximize the **expectation** of the return. In real world applications, the environ-

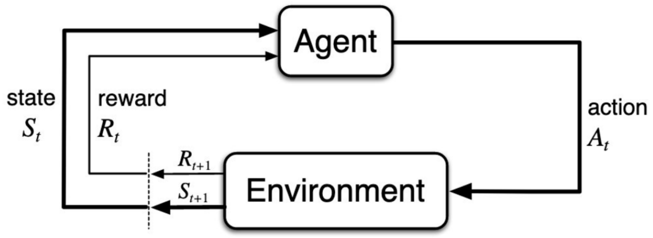


Figure 1.1: Markov Decision Process, figure from Sutton & Barto (2018). Agent interacts with the environment by taking action A_t. Environment transits agent to S_{t+1} and provides a reward R_{t+1}.

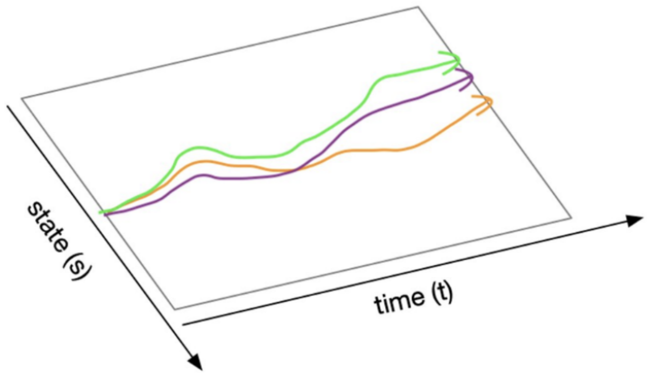


Figure 1.2: Random trajectories. Even though the agent starts from the same initial state, it may end up with different trajectories due to the intrinsic uncertainty.

ment dynamic is usually stochastic, thus the uncertainty of the environment need to be considered, and the risk need to be avoided. For instance, we want to avoid huge financial losses in portfolio management, which motivates mean-variance portfolio management research (Markowitz & Todd, 2000).

1.1 Uncertainty in RL and Distributional RL

Generally, the uncertainty in RL stems from two sources:

Aleatoric uncertainty due to the intrinsic stochasticity of the environment such as stochastic rewards, transitions, and the intrinsic stochasticity of agent's policy.

Epistemic uncertainty due to limited data samples caused by insufficient exploration. While the epistemic uncertainty can be eliminated if the agent is equipped with a perfect exploration strategy, the aleatoric uncertainty always remains during the learning process.

Due to the aleatoric uncertainty, even though the agent starts from the same initial state and takes the same initial action, it may end up with different trajectories and therefore different returns, as shown in Fig. 1.2. This naturally suggests the return at each state is a random variable. Estimating the uncertainty of the return variable has been widely used in RL for guiding exploration and stabilizing policies. O'Donoghue et al. (2018) designed an uncertainty Bellman equation that estimates the variance of the Q value posterior distribution. Distributional RL methods (Bellemare et al., 2017; Dabney et al., 2018b,a; Zhou et al., 2020; Luo et al., 2022) directly model the distribution of return variables.

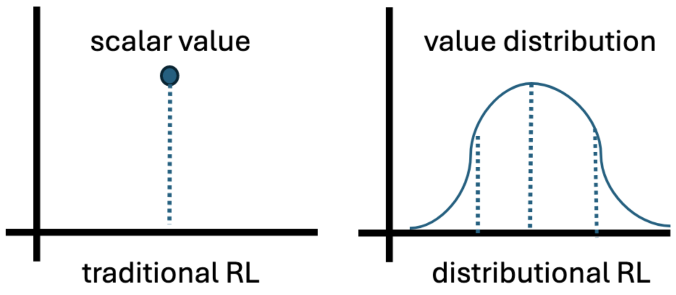


Figure 1.3: Traditional RL V.S. distributional RL. Traditional RL aims to learn a point estimation for the action value. Distributional RL instead aims to recover the full action value distribution.

Bootstrapped methods (Osband et al., 2016; Da Silva et al., 2020) learn ensembles of Q values to capture uncertainty.

In this thesis, we focus on the distributional RL methods. The difference between traditional RL and distributional RL is briefly depicted in Fig. 1.3. In traditional RL, the algorithms only estimate a scalar value, i.e., Q value (or expected return) to quantify the preference of an action. In distributional RL, the whole distribution of the return is modeled, which provides a richer understanding of uncertainty in the environment and allows agents to make more informed decisions. For example, agents can choose actions with the highest expected value (i.e., Q value) as in risk-neutral setting, or choose actions according to risk values extracted from the value distribution (Dabney et al., 2018a). Distributional RL is shown to outperform traditional RL in several domains, e.g., Atari video games (Bellemare et al., 2017).

In distributional RL, how to parameterize the return distribution is crucial. The main stream methods represent the distribution by its quantile function, also known as inverse cumulative distribution function. Existing distributional RL methods use step functions (Dabney et al., 2018b,a; Zhou et al., 2020) or piece-wise linear functions (Zhou et al., 2021) to represent the quantile, which have limited representation ability. Some of them also suffer from quantile crossing issue, which will be discussed later in Chapter 3.

1.2 Risk Measures and Risk-Averse RL

The demand for avoiding risk in decision making motivates risk-averse RL. For example, we want to avoid collisions in autonomous driving (Naghshvar et al., 2018), or avoid huge financial losses in portfolio management (Björk et al., 2014). The classical or risk-neutral RL methods (Sutton & Barto, 2018) only care about the expectation (or mean) of the return, where the higher order moments or other statistics of the return are neglected. In contrast, risk-averse RL optimizes some risk measures of the return instead of optimizing the mean only.

In this thesis, we focus on two commonly used risk classes, i.e., measure of variability and tail risk measure. Measure of variability reflects the dispersion of a random variable, thus it considers the *information of the whole distribution* (we do not consider semi-deviation in this paper). For example, variance and standard deviation are well known and common choices for measures of variability. Tail risk measure only cares about the value of a distribution in its left tail and reflects the *worst case values of a distribution*, thus only partial information of the distribution is considered. Value at risk (VaR) and conditional value at risk (CVaR) are two common examples of tail risk measures. For readers not familiar with VaR and CVaR, we give a visualization in Fig. 1.4. VaR is the smallest value if the probability of a random variable larger than this value is no less than a given threshold (also called a risk level). CVaR is the expected value smaller than VaR. In RL, a policy's return with low variance indicates a stable performance. While a policy's return with high CVaR indicates a good performance guarantee even in worst case scenarios.

Usually, users may expect some properties of risk measures, and risk measures satisfying some of them are called coherent (Furman et al., 2017). Let $\mathcal{M}$ denote the set of real random variables. Consider a measure $\rho : \mathcal{M} \to (-\infty, \infty]$ along with the following properties

- (A) *Law-invariance*: if $X, Y \in \mathcal{M}$ have the same distributions, then $\rho(X) = \rho(Y)$
- (A1) *Positive homogeneity*: $\rho(\lambda X) = \lambda\rho(X)$ for all $\lambda > 0$, $X \in \mathcal{M}$
- (A2) *Sub-additivity*: $\rho(X + Y) \leq \rho(X) + \rho(Y)$ for $X, Y \in \mathcal{M}$
- (B1) *Monotonicity*: $\rho(X) \leq \rho(Y)$ when $X \leq Y$, $X, Y \in \mathcal{M}$
- (B2) *Translation invariance*: $\rho(X - m) = \rho(X) - m$ for all $m \in \mathbb{R}$, $X \in \mathcal{M}$
- (C1) *Standardization*: $\rho(m) = 0$ for all $m \in \mathbb{R}$

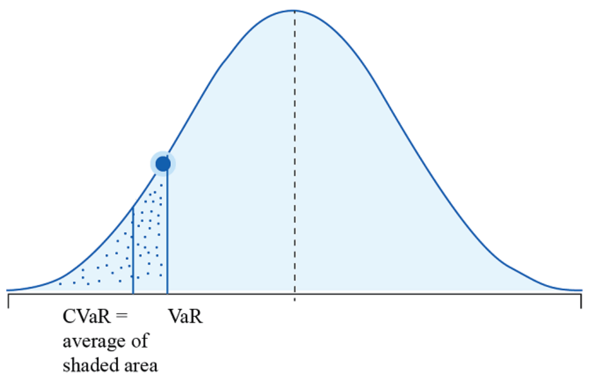


Figure 1.4: Visualization of VaR and CVaR of a distribution. By choosing a VaR value at the tail of a distribution, CVaR is the average value of the distribution no larger than VaR.

- (C2) *Location invariance*: $\rho(X - m) = \rho(X)$ for all $m \in \mathbb{R}$, $X \in \mathcal{M}$

Coherent measure of variability: a measure of variability is *coherent* if it satisfies properties (A), (A1), (A2), (C1) and (C2) (Furman et al., 2017). Standard deviation is coherent while variance is not since variance does not satisfy (A1). However, variance is still a common choice in RL given its easy interpretability and computation, which leads to the field of mean-variance RL (see Chapter 4).

Coherent risk measure: a risk measure is *coherent* if it satisfies properties (A1), (A2), (B1) and (B2). CVaR is coherent while VaR is not (whose definitions will be given in Chapter 5). CVaR is often preferred to VaR in RL since CVaR considers the expectation of the tail while VaR is like a chance constraint.

Optimizing risk of the total return, e.g., variance and CVaR, in RL is not easy due to the lack of time consistency, i.e., optimizing risk at each step is not consistent with optimizing risk of the total return. As a result, the well developed value-based risk-neutral RL algorithms can not be directly applied, e.g., Bellman equation (Eq. 2.6). The most straightforward way to optimize variance or CVaR is still policy gradient. The idea of policy gradient is similar as doing gradient descent to minimize some loss function in supervised learning. The policy is usually parameterized by a function with some parameters. The gradient of the objective with respect to policy parameters is computed and the policy parameters are updated via gradient ascent. For measure of variability, e.g., variance, the objective is usually maximizing the mean of the total return while minimizing the return variance. For tail risk measure, e.g., CVaR, the objective is usually maximizing the CVaR of total return given a risk level.

Tough the policy gradient for variance is easy to compute, we give a thorough analysis in Chapter 4 to reveal the issue caused by using variance which leads to unstable policy update. For tail risk measure, e.g., CVaR, by definition, it only uses the tail data of the return distribution (usually a small portion), thus the sample efficiency is extremely low with the majority data being discarded after the policy update.

1.3 Contribution

Based on the brief analysis above, in this thesis, we focus on the following questions:

1. What is an efficient way to learn quantile functions of value distributions in distributional RL?

2. How to stabilize policy gradient for measure of variability?

3. How to improve sample efficiency of CVaR policy gradient?

To give answers to these questions, our contributions can be summarized as follows.

1. For distributional RL, we propose a new learning framework to learn quantile functions for value distributions. Specifically, our quantile parameterization, inspired by monotonic splines, directly learns a smooth continuous and monotonic quantile function, which offers more flexibility compared to previous methods. Also, the quantile crossing issue is avoided thanks to the monotonicity guarantee of our method. We compare our method with all existing distributional RL methods by the time when this work was published in several stochastic environments. Particularly, one method named NDQFN (Zhou et al., 2021) also learns continuous and monotonic quantile function, we compare with this method under different quantile sampling regimes. Results show that our estimation for quantile functions enhances distributional RL in terms of faster empirical convergence and higher rewards in most cases.

2. To optimize measure of variability, we show the commonly used term variance (defined on both total return and per-step reward) leads to unstable policy updates, and may fail to learn a reasonable risk-averse policy. This optimization issue is caused by the square function used by variance. We thoroughly examine the limitations of these variance-based methods, such as sensitivity to numerical scale and hindering of policy learning, and propose to use an alternative measure of variability, Gini deviation, as a substitute. Different

from variance, Gini deviation computes the absolute value of the difference between samples, and Yitzhaki et al. (2003) showed that Gini deviation is superior to variance if the underlying distribution is far from Gaussian. The corresponding policy gradient algorithm is derived for optimizing Gini deviation in RL. Since standard deviation (STD) gets rid of the square term in variance by taking the square root of variance, we also give a discussion on the policy gradient of STD and a comparison between Gini deviation and STD in the experiments.

3. To optimize CVaR, generally, the sample efficiency of CVaR policy gradient is low and improving sample efficiency of CVaR optimization is hard due to the following two facts: 1) according to CVaR gradient formula, the majority of samples do not contribute to the gradient calculation; 2) the gradient may potentially vanish if the left tail of the distribution's quantile function is overly flat. We give a concrete example to reveal this gradient vanishing issue. To address these challenges, we provide insight that in many domains the risk-averse behavior is only required in a subset of states and agents can behave akin to a risk neutral agent in the remaining states. We provide a concrete example to validate this idea. Based on this insight, we propose a simple mixture policy parameterization. This method integrates a risk-neutral policy with an adjustable policy to form a risk-averse policy. By employing this strategy, all collected trajectories can be utilized for policy updating which improves sample efficiency, and the issue of vanishing gradients is counteracted by stimulating higher returns through the risk-neutral component, thus lifting the tail and preventing flatness. Our empirical study reveals that this mixture parameterization is uniquely effective across a variety of benchmark domains. Specifically, it excels in identifying risk-averse CVaR policies in some Mujoco environments where the traditional CVaR-PG fails to learn a reasonable policy.

Chapter 2

Reinforcement Learning Background

This chapter introduces some basic background in RL. It is mainly based on the commonly referred book by Sutton & Barto (2018). We first introduce the background of the Markov decision process (MDP) (Bellman, 1957), which is the main theoretical framework for RL. We then introduce the value based algorithm and the policy gradient (PG) algorithm, which are two main stream algorithms in RL. Finally, we introduce the concept of offline RL.

2.1 Markov Decision Process

The interaction between agent and environment is modeled as a MDP (Fig. 1.1), represented by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma, \mu_0)$. $\mathcal{S}$ and $\mathcal{A}$ denote state and action spaces. $P(\cdot|s, a)$ defines the state transition dynamics, with $P : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \to [0, \infty)$. R is the state and action dependent reward. $\gamma \in [0, 1]$ is a discount factor, which controls the importance of rewards that are closer in time to the agent, and μ_0 is the distribution of the initial state. The agent starts from the initial state $S_0 \sim \mu_0$. At any time point t, the agent is in a state S_t and has to take an action A_t according to its policy π, i.e., $A_t \sim \pi(\cdot|S_t)$, with $\pi : \mathcal{S} \times \mathcal{A} \to [0, \infty)$. Upon taking this action, the agent transits to next state $S_{t+1} \sim P(\cdot|S_t = s_t, A_t = a_t)$ and receives a scalar reward r sampled from $R_{t+1} \overset{\text{def}}{=} R(S_t, A_t)$. We will overload the notation of R to represent the reward random variable when states and actions are given. For example, $R(s, a)$ is the reward random variable when agent takes action a in a particular state s, and the scalar value $r(s, a)$ sampled from $R(s, a)$ is what the agent actually gets when it visits state s and takes action a (some work may treat $R(s, a)$ as a scalar, while we treat it as a random variable as a general case). We may also overload the notation of P to represent the probability of other quantities when the context is clear.

The *return* random variable at time t, denoted by G_t^π is the discounted sum of future rewards given that actions are selected according to π (we may write G_t for simplicity).

$$\begin{aligned} G_t \overset{\text{def}}{=} & R_{t+1} + \gamma R_{t+1} + \gamma^2 R_{t+3} + ... \\ = & R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{2.1}$$

Thus, the random variable G_0 indicates the return starting from the initial state following π. The goal of the agent (in a risk-neutral setting) is to maximize the expected value of G_0, i.e.,

$$\max_\pi \mathbb{E}[G_0] \tag{2.2}$$

Note that the choice of γ and whether the summation in Eq. 2.1 goes to infinity will lead to different considerations for the problem defined in Eq. 2.2. Usually, $\gamma = 0$ is not particularly studied since the agent is myopic in being concerned only with maximizing immediate rewards. When $\gamma > 0$, we can write Eq. 2.1 in a more unified way as

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{2.3}$$

When $T \neq \infty$, γ can equal to 1. This is often called an episodic task. When $T = \infty$, γ is set to be smaller than 1 to ensure G_t is bounded. This is often called a discounted continuing task. The undiscounted continuing task, also known as the average reward setting, i.e., $T = \infty$ and $\gamma = 1$, is also possible, but requires different techniques to solve, which is outside the scope of this thesis.

In this thesis, we focus on the discounted continuing task, i.e., $T = \infty$ and $\gamma \in (0, 1)$. For on-policy policy gradient methods, sometimes we are unable to sample a trajectory with $T = \infty$. In this case, it is a discounted episodic task, and we can regard it as a special case of the discounted continuing task by treating the reward to be 0 when time $t > T$.

It is usually convenient to define the value function to represent the expected return the agent can get when it is in some status. The state value function is defined as

$$V^\pi(s) \overset{\text{def}}{=} \mathbb{E}[G_t | S_t = s, \pi] \tag{2.4}$$

The state-action value function is defined as

$$Q^\pi(s, a) \overset{\text{def}}{=} \mathbb{E}[G_t | S_t = s, A_t = a, \pi] \tag{2.5}$$

2.2 Value-based Method

The value-based method usually refers to the approaches that only estimate the value functions in Eq. 2.4 or 2.5, and extract a policy from the value function accordingly, e.g. taking argmax over the action space (when the action space is discrete). For simplicity of expression, we consider the state and action spaces as discrete. The basic idea is that, the optimal policy, i.e., the solution to Eq. 2.2, should achieve the maximum Q value for each state-action pair. We can consider this through counter examples. Suppose a trajectory the agent can get under π is $(s_0, a_0, s_1, a_1, ...)$, if the value of $Q^\pi(s_t, a_t)$ is not the maximum and can be increased, it will in turn increase the value of $Q^\pi(s_0, a_0)$ (we may think $\mathbb{E}[G_0]$ is estimated by $Q^\pi(s_0, a_0)$), which means π is not the optimal policy.

Define the optimal state-action value function as $Q^*(s,a) \overset{\text{def}}{=} \arg\max_\pi Q^\pi(s,a)$. $Q^*(s,a)$ satisfies the Bellman optimality equation as (Bellman, 1966)

$$Q^*(s,a) = \mathbb{E}[R(s,a)] + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a') \tag{2.6}$$

Based on this equation, one can define the Bellman optimality operator as

$$\forall s \in \mathcal{S}, a \in \mathcal{A} \quad \mathcal{T}Q(s,a) = \mathbb{E}[R(s,a)] + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a') \tag{2.7}$$

The Bellman optimality operator in Eq. 2.7 is known as a contraction mapping in the sense that

$$\|\mathcal{T}Q_1 - \mathcal{T}Q_2\|_\infty \le \gamma \|Q_1 - Q_2\|_\infty \tag{2.8}$$

for any Q function Q_1 and Q_2. This contraction property can be proved based on Banach fixed-point theorem (Banach, 1922).

To interpret the Bellman optimality operator, consider that we initialize a state-action value table with $|\mathcal{S}| \times |\mathcal{A}|$ entries. To update the value for every $Q(s,a)$, while we do not have a direct target as in supervised learning, we can push our value estimate towards the target given by $\mathbb{E}[R(s,a)] + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$, which is known as the **temporal difference (TD) target**. This procedure is guaranteed to recover the optimal $Q^*(s,a)$ for each (s,a) when it converges. To extract the optimal policy, one can simply do $\pi^*(s) = \arg\max_a Q^*(s,a)$.

2.2.1 Q Learning

The above example to recover Q^* is based on the Bellman optimality operator, which requires the full knowledge of the reward and transition function of the MDP, and works on the whole state-action space at a time. In real RL problems, agents only get point samples, which are $(s_t, a_t, r(s_t, a_t), s_{t+1})$ tuples during the interaction. Thus, in practice agents can only update Q functions based on these point samples, instead of the whole state-action space at a time as done in Eq. 2.7.

To update with point samples, usually the learning process is that we choose a learning rate α_t and update the Q value after observing some transition data, i.e.,

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha_t(s_t, a_t)\Big[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)\Big] \tag{2.9}$$

where $0 \leq \alpha_t(s_t, a_t) \leq 1$ is the step size. Eq. 2.9 is also known as Q learning. The following theorem shows that Q learning converges to the optimal value Q^* under some conditions.

Theorem 1 *Given a finite MDP, the Q learning algorithm, with the update rule in Equation 2.9 converges with probability 1 to the optimal Q function as long as*

$$\sum_t \alpha_t(s, a) = \infty \qquad \sum_t \alpha_t^2(s, a) < \infty \tag{2.10}$$

for all $(s, a) \in \mathcal{S} \times \mathcal{A}$.

Proof. See the proof by Jaakkola et al. (1993).

2.2.2 Deep Q Learning

To leverage the advancement of deep learning, deep Q learning or deep Q network (DQN) (Mnih et al., 2013) uses a deep neutral network as function approximator for the Q function. When interacting with the environment, the transition samples are stored in a buffer $\mathcal{B}$. To update the Q function, a mini batch of $\{(s, a, r(s, a), s')\}$ is sampled from the buffer, and the parameters of the Q function is updated to minimize the squared TD error

$$\mathcal{L}^2(\phi) = \mathbb{E}_{(s,a,r(s,a),s')\sim\mathcal{B}}\Big[r(s, a) + \gamma \max_{a'} Q_{\phi^-}(s', a') - Q_\phi(s, a)\Big]^2 \tag{2.11}$$

where ϕ is the training parameter of the Q function, ϕ^- is the target network parameter which is updated periodically with the most recent ϕ.

DQN has shown super human performance in video games (Mnih et al., 2013). There are several follow up works of DQN, e.g., dueling DQN (Wang et al., 2016), double DQN (Van Hasselt et al., 2016) to stabilize and improve the performance of DQN.

2.3 Policy Gradient Method

Recall that the goal is to solve the maximization problem in Eq. 2.2. A straightforward way is doing gradient ascent to maximize $\mathbb{E}[G_0]$ with respect to the parameter of policy π. This is similar as doing gradient descent to minimize a loss function in supervised learning.

Denote the policy by $\pi_\theta(a|s)$, i.e., the probability of choosing action a in state s, where θ denotes the policy parameters. Here we are interested in computing $\nabla_\theta \mathbb{E}[G_0]$. This gradient is given in Chapter 13 of the book by Sutton & Barto (2018) by calculating $\nabla_\theta V^\pi(s_0)$ (since $V^\pi(s_0)$ estimates $\mathbb{E}[G_0]$). We give another method based on Monte Carlo sampling, i.e., sampling trajectories from the environment.

Given a policy π_θ, by executing this policy, we get a trajectory $\tau = (s_0, a_0, r_1, s_1, a_1, ...r_T, s_T)$, with the return of this trajectory $R_\tau = r_1 + \gamma r_2 + ... + \gamma^{T-1} r_T$. Note that $\mathbb{E}[G_0] = \mathbb{E}_\tau[R_\tau]$, thus $\nabla_\theta \mathbb{E}[G_0] = \nabla_\theta \mathbb{E}_\tau[R_\tau]$

$$\begin{aligned}\nabla_\theta \mathbb{E}_\tau[R_\tau] &= \nabla_\theta \sum_\tau P(\tau;\theta) R_\tau = \sum_\tau R_\tau P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} \\ &= \sum_\tau R_\tau P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) = \mathbb{E}_\tau[R_\tau \nabla_\theta \log P(\tau;\theta)]\end{aligned} \tag{2.12}$$

where we applied the equality that $\nabla_\theta \log(x) = \frac{1}{x}\nabla_\theta x$. At this point, we still need to compute the log probability of the trajectory, i.e., $\log P(\tau;\theta)$. The probability of a trajectory is defined as

$$P(\tau;\theta) = \mu(s_0) \prod_{t=0}^{T-1} \Big[\pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t)\Big] \tag{2.13}$$

Thus, the logarithm is

$$\log P(\tau;\theta) = \log \mu(s_0) + \sum_{t=0}^{T-1} \Big[\log \pi_\theta(a_t|s_t) + \log P(s_{t+1}|s_t, a_t)\Big] \tag{2.14}$$

whose gradient is

$$\nabla_\theta \log P(\tau;\theta) = \nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) \tag{2.15}$$

Combining Eq. 2.12 and 2.15, the policy gradient is

$$\nabla_\theta \mathbb{E}_\tau[R_\tau] = \mathbb{E}_\tau\Big[R_\tau \nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t)\Big] \tag{2.16}$$

The gradient in Eq. 2.16 has high variance. The variance can be reduced by the following two commonly used techniques. First, there exists cross time terms in Eq. 2.16. Recall that $R_\tau = r_1 + \gamma r_2 + ... + \gamma^{T-1} r_T$. Thus there exist terms of $r_i \nabla_\theta \log \pi_\theta(a_j|s_j)$ with $i < j$. Those terms can be removed. One simple explanation is that actions executed at later time steps will not effect the rewards received earlier. In practice, the expectation of those terms is zero, thus removing them can reduce variance. Denote $R_{\tau,t} \overset{\text{def}}{=} \sum_{t'=t+1}^{T} \gamma^{t'-t-1} r_t$, i.e., the return starting from time t of trajectory τ. Removing those cross time terms yields

$$\nabla_\theta \mathbb{E}_\tau[R_\tau] = \mathbb{E}_\tau\Big[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \gamma^t R_{\tau,t}\Big] \tag{2.17}$$

Second, the term $R_{\tau,t}$ in Eq. 2.17 can be regarded as the Monte Carlo estimation of $Q^\pi(s_t, a_t)$. Based on the concept of control variate, we can subtract $V^\pi(s_t)$ from it to reduce variance, since $V^\pi(s_t) = \mathbb{E}_{a_t} Q^\pi(s_t, a_t)$, which results in

$$\nabla_\theta \mathbb{E}_\tau[R_\tau] = \mathbb{E}_\tau\Big[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \gamma^t \big(R_{\tau,t} - V^\pi(s_t)\big)\Big] \tag{2.18}$$

Eq. 2.18 is more often known as REINFORCE with baseline, where $V^\pi(s_t)$ is the baseline. In practice, we need another function to estimate this baseline.

Remark. The algorithm REINFORCE with baseline in Eq. 2.18 is also derived in Chapter 13.4 of Sutton & Barto (2018). However, the readers who are familiar with RL may find there is a γ^t term that does not appear in commonly used forms of policy gradient. To understand this, define the unnormalized discounted state visitation distribution as

$$\rho_\pi(s) \overset{\text{def}}{=} \sum_{t=0}^{T-1} \gamma^t \mathrm{Pr}(S_t = s|\pi, P) \tag{2.19}$$

By replacing $R_{\tau,t}$ with $Q^{\pi}(s_t, a_t)$, the gradient in Eq. 2.18 equals to

$$\mathbb{E}_{s_t \sim \rho_\pi(\cdot), a_t \sim \pi_\theta(\cdot|s_t)} \Big[\nabla_\theta \log \pi_\theta(a_t|s_t)(Q^\pi(s_t, a_t) - V^\pi(s_t)) \Big] \tag{2.20}$$

which reveals the commonly used form of policy gradient in the literature. In practice, most policy gradient methods effectively use undiscounted state visitation distributions, i.e., $\gamma = 1$ for ρ_π (Gu et al., 2017).

2.3.1 Deterministic Policy Gradient

In the above policy gradient, the policy function π_θ is always modeled as a probability distribution over actions. Deterministic policy gradient (DPG) instead defines the policy as a deterministic function, i.e., $a = \pi_\theta(s)$, and aims to generalize the policy gradient from a stochastic policy to a deterministic one. The gradient is given by (Silver et al., 2014)

$$\nabla_\theta \mathbb{E}[G_0] = \mathbb{E}_{s \sim \rho_\pi(\cdot)} \Big[\nabla_\theta \pi_\theta(s) \nabla_a Q^\pi(s, a)|_{a=\pi_\theta(s)} \Big] \tag{2.21}$$

Readers may refer to Silver et al. (2014) for the proof. For an intuitive interpretation of this gradient, we can think of it as differentiating the Q-function (using the chain rule)

$$\nabla_\theta \mathbb{E}_{s \sim \rho_\pi(\cdot)} \Big[Q^\pi(s, \pi_\theta(s)) \Big] \tag{2.22}$$

Thus, doing gradient ascent maximizes $\mathbb{E}_{s \sim \rho_\pi(\cdot)}[Q^\pi(s, \pi_\theta(s))]$. Recall that in value-based approaches, the optimal action is a greedy maximization of the state-action value, i.e., $\pi(s) = \arg\max_a Q(s, a)$. When greedy action is hard to compute, e.g., action space is continuous, a simple and computationally attractive alternative is to move the policy in the direction of the gradient of Q, rather than globally maximizing Q.

2.3.2 Off-Policy Policy Gradient

Both policy gradients described in Eq. 2.18 and 2.21 are on-policy policy gradients, i.e., the state is sampled from $\rho_\pi(\cdot)$ induced by the current policy π. It is often the case that we have trajectories generated by another behavior policy $\breve{\pi}$ and try to optimize π with the data we have. Under the off-policy setting, the objective to maximize is slightly different from the on-policy setting, which is given by (Degris et al., 2012)

$$J(\theta) = \sum_s \rho_{\breve{\pi}}(s) \sum_a \pi_\theta(a|s) Q^{\pi_\theta}(s, a) \tag{2.23}$$

The gradient is

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{s\sim\rho_{\breve{\pi}}(\cdot)}\Big[\sum_a Q^{\pi_\theta}(s,a)\pi_\theta(a|s)\Big] \\ &\approx \mathbb{E}_{s\sim\rho_{\breve{\pi}}(\cdot)}\Big[\sum_a Q^{\pi_\theta}(s,a)\nabla_\theta\pi_\theta(a|s)\Big] \\ &= \mathbb{E}_{s\sim\rho_{\breve{\pi}}(\cdot),a\sim\breve{\pi}(\cdot|s)}\Big[\frac{\pi_\theta(a|s)}{\breve{\pi}(a|s)}Q^{\pi_\theta}(s,a)\nabla_\theta\log\pi_\theta(a|s)\Big]\end{aligned} \tag{2.24}$$

The approximation in the second line of Eq. 2.24 ignores the term $\pi_\theta(a|s)\nabla_\theta Q^{\pi_\theta}(s,a)$ since $\nabla_\theta Q^{\pi_\theta}(s,a)$ is hard to compute in reality. However, Eq. 2.24 still guarantees a policy improvement, which is justified by Degris et al. (2012).

When a policy is deterministic, the objective function is thus

$$J(\theta) = \sum_s \rho_{\breve{\pi}}(s)Q^{\pi_\theta}(s,\pi_\theta(s)) \tag{2.25}$$

whose gradient is (Silver et al., 2014)

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{s\sim\rho_{\breve{\pi}}(\cdot)}\Big[\nabla_\theta\pi_\theta(s)\nabla_a Q^{\pi_\theta}(s,a)|_{a=\pi_\theta(s)}\Big] \tag{2.26}$$

Again, the term that depends on $\nabla_\theta Q^{\pi_\theta}(s,a)$ is ignored.

2.3.3 Policy Gradient with Deep Learning

In deep RL, the policy can be modeled by a deep neutral network. For a stochastic policy, when the action space is discrete, the policy is usually a categorical distribution; when the action space is continuous, a Gaussian distribution is a common choice, whose mean and standard deviation are the outputs of a neutral network. For a deterministic policy, the policy is just a one-to-one mapping from states to actions.

Here we briefly introduce some well-known deep PG algorithms.

On-Policy. REINFORCE with a baseline is definitely one of the most important algorithms, which is introduced in Eq. 2.18. By representing policy and baseline functions with deep neutral networks, it is then incorporated with the deep learning framework. Trust region policy optimization (TRPO) (Schulman et al., 2015) sets a constraint that ensures the updated policy stays close to the previous policy to ensure stability. Proximal policy optimization (PPO) (Schulman et al., 2017) uses a simple clip ratio function to approximately achieve the constraint in TRPO.

Off-Policy. Deep deterministic policy gradient (DDPG) (Lillicrap et al., 2016b) is the deep learning version of the off-policy DPG in Eq. 2.26. With the help of the deep learning framework, it is convenient to compute the gradient of the Q function with respect to actions. TD3 (Fujimoto et al., 2018) enhanced DDPG by learning two Q functions and taking the minimum value of them to address function approximation errors in DDPG. Soft actor critic (SAC) (Haarnoja et al., 2018) is based on maximum entropy RL, and is mathematically different from the policy gradients introduced in this chapter. Interested readers may refer to Haarnoja et al. (2018) for details.

2.4 Offline Reinforcement Learning

The methods discussed above are all online RL methods, i.e., there is an online environment that agents can interact with. In practice, the environment is usually provided as a simulator, i.e., Atari games (Bellemare et al., 2013), Mujoco (Todorov et al., 2012), or real world simulators like Commonroad (Wang et al., 2021). In many real world applications, directly interacting with the environment is expensive or impossible. For example, it is dangerous to directly learn autonomous driving policies using real cars on the highway. Though using simulator for policy learning is safe, developing accurate simulators for every real world domains is challenging.

Compared with learning policies from scratch via trials in interactive domains, usually, the data generated by other users is relatively easy to collect. For example, it is easy to collect the driving behavior of taxi drivers in a city by installing some tracking system on the vehicles. Thus, we can consider learning autonomous driving policies from the collected data instead of interacting with the environment.

Learning a policy directly from a fixed dataset without interacting with the environment is called offline RL. The challenge of learning a policy from a dataset only is that the dataset may not have full coverage of all actions. Thus, greedy decisions based on the learned Q values as in Eq. 2.9 is problematic especially when the Q value is an overestimate for out-of-distribution actions (Fujimoto et al., 2019).

Offline RL methods can be broadly categorized into value-based, policy-based, and model-based approaches. Value-based methods, such as conservative Q learning (Kumar et al., 2020) and its variants, learn a value function from the data and use it to derive policies. To avoid selecting out-of-distribution actions, the Q values of out-of-distribution actions are underestimated. Policy-based methods, including behavioral cloning and imitation learning (Fujimoto & Gu, 2021), directly optimize policies based on the observed behavior. Model-based methods leverage the data to learn a model of the environment dynamics, which can be used for planning or policy improvement (Yu et al., 2020). Since offline RL is not the main focus of this thesis, and it only contributes to a component of the algorithm proposed in Chapter 5, we only provide a brief review in this section.

Chapter 3

Distributional Reinforcement Learning with Monotonic Splines

3.1 Introduction

As introduced in Chapter 2, a fundamental problem in traditional value-based RL is to estimate the expectation of future returns (Mnih et al., 2015; Van Hasselt et al., 2016). However, due to the stochastic environment transition dynamics and stochastic policies, the future return is naturally a random variable. Distributional RL differs from traditional RL by also taking into account the intrinsic randomness of returns (Morimura et al., 2010; Bellemare et al., 2017). To do so, distributional RL algorithms estimate the underlying distribution of the total return random variable. In contrast, traditional value-based RL algorithms focus only on the mean of the random variable.

Distributional RL offers several advantages over traditional value-based RL that computes only expected returns. The distribution of returns enables risk-sensitive RL by facilitating the optimization of other statistics than just the mean of the returns (Dabney et al., 2018a; Martin et al., 2020). Even when we stick to maximizing the mean of the returns, the distribution offers a more reliable and robust way of computing the expectation, which has led to a series of records on the Atari benchmark among value-based non-distributed RL techniques (Bellemare et al., 2017; Dabney et al., 2018a,b; Hessel et al., 2018; Yang et al., 2019; Zhou et al., 2020; Nguyen et al., 2021). Intuitively, while it is sufficient to represent an expected return by a single mean value, errors due to finite samples and function approximations can be reduced by "canceling" each other when multiple sample returns or quantile values are used. This is similar to the benefits of ensemble learning techniques although, strictly speaking, distributional RL is not an ensemble RL technique. In fact, distributional RL has been combined with ensemble learning and truncated critic predictions to mitigate the overestimation bias in continuous control (Kuznetsov et al., 2020).

One key aspect of distributional RL algorithms is the parameterization of return distributions. In Categorical DQN (C51) (Bellemare et al., 2017), the return distributions are limited to categorical distributions over a fixed set of discrete values. It is also shown that the distributional Bellman operator is a contraction under the maximal form of the Wasserstein metric, but in practice, C51 optimizes the cross-entropy loss with a Cramér-minimizing projection (Rowland et al., 2018). To bridge the gap between theoretical analysis and algorithmic implementation, quantile regression (QR)-based distributional RL algorithms (Dabney et al., 2018a,b; Yang et al., 2019; Zhou et al., 2020) estimate a finite number of quantile values instead of the distribution of returns since quantile regression can easily use the Wasserstein metric as the objective. In fact, the Wasserstein metric is approximately minimized by optimizing the quantile Huber loss (Huber, 1992) between the Bellman updated distribution and the current return distribution.

Although with an infinite number of quantiles, the step quantile function in those quantile regression based methods will approximate the full quantile function arbitrarily closely, in practice, it is infeasible to have infinite quantiles in most existing architectures. In addition, the quantile crossing issue, recently pointed out and solved by (Zhou et al., 2020), was ignored by previous distributional RL techniques. The issue is that if no global constraint is applied, the quantile values estimated by a neural network at different quantile levels are not guaranteed to satisfy monotonicity, which can distort policy search and affect exploration during training (Zhou et al., 2020).

In this chapter, we propose to learn a continuous representation for quantile functions based on monotonic rational-quadratic splines (Gregory & Delbourgo, 1982). The monotonic property of these splines naturally solves the quantile crossing issue described above. Furthermore, unlike step functions or piecewise linear functions that provide a crude approximation in each bin, monotonic splines provide a more flexible and smooth approximation. With sufficiently many knots, splines can approximate any quantile function arbitrarily closely. We compare empirically our spline-based technique with other quantile-based methods in stochastic environments. We demonstrate that our method offers greater accuracy in terms of quantile approximation, faster convergence during training and higher rewards at test time.

Remark. Tough distributional RL captures the intrinsic uncertainty of the environment, in this chapter, we consider policy learning in risk-neutral setting, i.e., maximizing the expected return (Eq. 2.2).

3.2 Background: Distributional RL

For a policy π, the discounted sum of returns starting from state s by taking action a is denoted as a random variable $Z^\pi(s,a) \overset{\text{def}}{=} \sum_{t=0}^{\infty} \gamma^t R(S_t, A_t)$, where $S_0 = s$, $A_0 = a$, $S_{t+1} \sim P(\cdot|S_t, A_t)$, and $A_t \sim \pi(\cdot|S_t)$. The Q value defined in Chapter 2 is $Q^\pi(s,a) = \mathbb{E}[Z^\pi(s,a)]$. In most deep RL studies, Q is approximated by a neural network. To update Q, Q learning trains the network iteratively to minimize the squared temporal difference (TD) error in Eq. 2.11.

Instead of learning the scalar $Q(s,a)$, distributional RL considers the distribution over returns (the law of Z) to capture the aleatoric uncertainty. A similar distributional Bellman operator for Z can be derived as (Bellemare et al., 2017)

$$\mathcal{T}^\pi Z^\pi(s,a) \overset{D}{=} R(s,a) + \gamma Z^\pi(S', A'), \tag{3.1}$$

with $S' \sim P(\cdot|s,a)$, $A' \sim \pi(\cdot|S')$, and $X \overset{D}{=} Y$ indicates that random variables X and Y follow the same distribution. In theory, Bellemare et al. (2017) proved the distributional Bellman operator is a contraction in the maximal of p-Wasserstein metric (see Lemma 3 of Bellemare et al. (2017))

$$W_p(X,Y) = \Big(\int_0^1 |F_X^{-1}(\omega) - F_Y^{-1}(\omega)|^p d\omega \Big)^{1/p}, \tag{3.2}$$

where F^{-1} is the quantile function (inverse cumulative distribution function). Following this theory, a series of distributional RL algorithms have been proposed based on quantile regression to estimate F^{-1} at precisely chosen quantile levels, such that the Wasserstein metric is approximately minimized.

For a random variable Z with cumulative distribution function $F_Z(z) = \mathcal{P}(Z \leq z)$, the ζ-level quantile of Z is $\min\{z | F_Z(z) \geq \zeta\}, \zeta \in (0,1]$. For a given quantile level $\breve{\zeta}$ and value v, the quantile regression loss is expressed as

$$\mathbb{E}_Z \Big[\big(\breve{\zeta} \mathbb{I}_{\{Z>v\}} + (1-\breve{\zeta}) \mathbb{I}_{\{Z<v\}} \big) |Z - v| \Big] \tag{3.3}$$

The gradient is

$$\mathbb{E}_Z \Big[\breve{\zeta} \mathbb{I}_{\{Z>v\}} - (1-\breve{\zeta}) \mathbb{I}_{\{Z<v\}} \Big] \tag{3.4}$$

Similar to the Q learning algorithm in Eq. 2.9, we can only use point samples to do quantile regression. Suppose the return variable $Z(s_t, a_t)$ is represented by a set of quantiles $\{\Theta_1(s_t, a_t), ..., \Theta_N(s_t, a_t)\}$, corresponding to quantile levels $\{\zeta_1, \zeta_2, ..., \zeta_N\}$. The return variable for the next state-action $Z(s_{t+1}, a_{t+1})$ is represented by a set of quantiles $\{\Theta_1(s_{t+1}, a_{t+1}), ..., \Theta_N(s_{t+1}, a_{t+1})\}$. The gradient in Eq. 3.4 leads to the update rule

$$\Theta_i^{t+1}(s_t, a_t) \leftarrow \Theta_i^t(s_t, a_t) + \frac{\alpha_t(s_t, a_t)}{N} \sum_{j=1}^{N} (\zeta_i - \mathbb{I}\{r(s_t, a_t) + \gamma \Theta_j^t(s_{t+1}, a_{t+1}) < \Theta_i^t(s_t, a_t)\}) \tag{3.5}$$

Theorem 2 (Theorem 5.1 in Rowland et al. (2023)) *Given a finite MDP, consider the quantile update rule in Eq. 3.5 with non-negative step sizes satisfying*

$$\sum_{t=0}^{\infty} \alpha_t(s_t, a_t) = \infty, \quad \alpha_t = o(1/\log t) \tag{3.6}$$

Then Θ^k converges almost surely to the set of fixed points with probability 1.

Remark. The distributional Bellman operator in Eq. 3.1 is in the policy evaluation setting, i.e., the policy π is given and fixed. It is different from the Bellman optimality operator in Eq. 2.7 where there is a maximum for the next Q value. The contraction property for the random variable of Eq. 3.1 is thus in the policy evaluation setting. If we apply a similar greedy action selection for S' for Eq. 3.1, i.e.,

$$\mathcal{T} Z(s, a) \overset{D}{=} R(s, a) + \gamma Z(S', A'), \quad A' = \arg\max_{a'} \mathbb{E}[Z(S', a')], \tag{3.7}$$

then we can only expect a contraction for $\mathbb{E}[Z]$, e.g., see Lemma 4 of Bellemare et al. (2017).

In this chapter, we focus on the control problem in risk neutral setting, i.e., Eq. 3.7.

In addition, users can only perform quantile regression with limited quantile points in practice. In this case, we may think the random variable Z is projected to a quantile distribution space. The projected distributional Bellman operator is still a contraction mapping (see Proposition 2).

3.2.1 Quantile Regression based Methods

Different distributional RL methods consider different parameterization of quantiles. In QR-DQN (Dabney et al., 2018b), the random return is approximated by a uniform mixture of N Diracs

$$Z_\Theta(s,a) = \frac{1}{N}\sum_{i=1}^{N} \delta_{\Theta_i(s,a)}, \tag{3.8}$$

with each Θ_i set to a fixed quantile level, $\hat{\zeta}_i = \frac{\zeta_{i-1}+\zeta_i}{2}$ for $1 \le i \le N$, and $\zeta_i = i/N$. The quantile estimation is performed by minimizing the quantile Huber loss (Huber loss is for the purpose of making the loss function smooth), with threshold η

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} \rho^{\eta}_{\hat{\zeta}_i}(\delta_{ij}) \tag{3.9}$$

on the pairwise TD error $\delta_{ij} = r + \gamma\Theta_j(s', a') - \Theta_i(s,a)$, where

$$\begin{gathered} \rho^{\eta}_{\zeta}(\delta) = |\zeta - \mathbb{I}_{\delta<0}|\mathcal{L}_\eta(\delta), \text{ with} \\ \mathcal{L}_\eta(\delta) = \begin{cases} \frac{1}{2}\delta^2, & |\delta| \le \eta \\ \eta(|\delta| - \frac{1}{2}\eta), & \text{otherwise.} \end{cases} \end{gathered} \tag{3.10}$$

Based on QR-DQN, Dabney et al. (2018a) proposed to sample quantile levels from a base distribution, e.g. $\zeta \in U([0,1])$ rather than fixing them. They built an implicit quantile network (IQN) to learn mappings from sampled probability embeddings to corresponding quantile values. FQF (Yang et al., 2019) further improves IQN by learning a function to propose τ's. However, the quantile values generated by neural networks may not satisfy the non-decreasing property of F^{-1} (known as the quantile crossing issue). This was recently solved by NC-QR-DQN (Zhou et al., 2020), by applying a softmax, followed by a cumulative sum of the output logits of the neural network Ω, and then rescaling by multiplying a non-negative factor $\alpha(s,a)$ and adding an offset $\beta(s,a)$:

$$\begin{gathered} \Theta_i(s,a) = \alpha(s,a) \times \iota_{i,a} + \beta(s,a), \text{ with} \\ \iota_{i,a} = \sum_{j=0}^{i} \chi_{j,a}, \text{ and } \chi_{j,a} = \text{softmax}(\Omega(s))_{j,a} \end{gathered} \tag{3.11}$$

One recent method NDQFN (Zhou et al., 2021) further combines the ideas of NC-QR-DQN and IQN to learn a monotonic function for F^{-1} by connecting the neighboring two

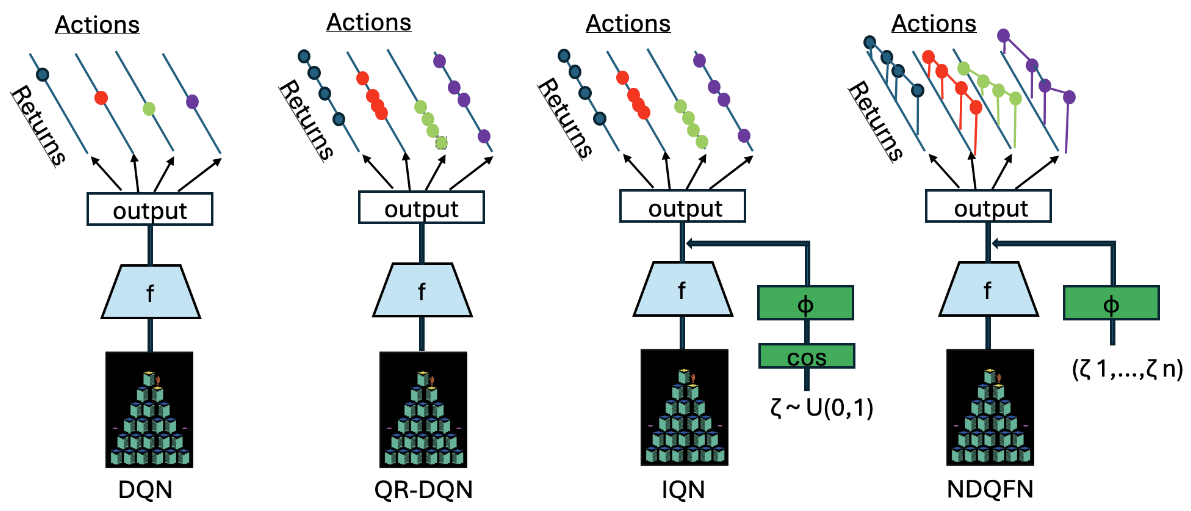


Figure 3.1: The architecture of DQN, QR-DQN, IQN, and NDQFN. In this figure, we assume the size of the action space is 4. The colored dots represent the output value of the network. DQN only outputs a single value for an action. Distributioanl RL methods generally output multiple (quantile) values for an action.

monotonic quantile data points with line segments. Different from NC-QR-DQN, NDQFN generates monotonic quantile values by first learning a baseline value and then adding non-negative increments.

We briefly show the architecture of different quantile based distributional RL methods in Fig. 3.1. Suppose the size of the action space is 4. DQN (Mnih et al., 2013) outputs 4 scalar values, one for each action to represent the Q value, i.e., $Q(s, a)$. QR-DQN (Dabney et al., 2018b) outputs N values for each action dimension, to represent the quantile values of $Z(s, a)$. QR-DQN assumes these N quantile values are uniformly spaced. IQN (Dabney et al., 2018a) instead samples the quantile level from uniform(0,1), and embeds the quantile level as input to output corresponding quantile values. NDQFN (Zhou et al., 2021) assumes a fixed quantile level set $(\zeta_1, ...\zeta_n)$ and outputs the corresponding quantile values. Then piece-wise linear interpolation is applied. The output layer of NC-QR-DQN (Zhou et al., 2020) is the same as QR-DQN, and the architecture of FQF (Yang et al., 2019) is similar to IQN, which are not shown in the figure.

3.2.2 Other Distributional Methods

Other recent methods investigate different metrics for the distributional Bellman operator. Moment matching, generally parameterized as the maximum mean discrepancy (MMD) between two sample sets in a reproducing Hilbert kernel space (Gretton et al., 2012), is adopted by Nguyen et al. (2021) to propose moment matching DQN (MM-DQN). The MMD loss with kernel κ is derived as:

$$d_\kappa^2(\{\upsilon_i\}, \{\psi_i\}) = \frac{1}{N^2}\sum_{i,j}\kappa(\upsilon_i, \upsilon_j) + \frac{1}{M^2}\sum_{i,j}\kappa(\psi_i, \psi_j) - \frac{2}{NM}\sum_{i,j}\kappa(\upsilon_i, \psi_j), \tag{3.12}$$

where $\{\upsilon_i\}_{i=1}^N \sim Z(s,a)$ and $\{\psi_i\}_{i=1}^M \sim R(s,a) + \gamma Z(S', A')$.

It is worth noting that the theoretical analysis by Nguyen et al. (2021) shows the distributional Bellman operator under MMD is not a contraction with commonly used Gaussian kernels or exp-prod kernels. It is a contraction only when the kernel function is shift invariant and scale sensitive.

Categorical distributional RL was also combined with policy gradient to obtain the Reactor algorithm (Gruslys et al., 2018) for discrete control and the Distributed Distributional Deep Deterministic Policy Gradient (D4PG) algorithm (Barth-Maron et al., 2018) for continuous control. Subsequently, Singh et al. (2020) replaced categorical return distributions by samples in Sample-based Distributional Policy Gradient (SDPG), yielding improved sample efficiency. The return distribution can also be represented by a generative network trained by adversarial training (in the same way as GANs) to minimize temporal differences between sampled returns (Doan et al., 2018; Freirich et al., 2019). While most distributional RL techniques compute state-action return distributions, Li & Faisal (2021) proposed the Bayesian Distributional Policy Gradient (BDPG) algorithm that computes state return distributions and uses inference to derive a curiosity bonus. In another line of work, Tessler et al. (2019) introduced the Distributional Policy Optimization (DPO) framework in which an agent's policy evolves towards a distribution over improving actions.

3.3 Learning a Return Distribution via Monotonic Splines

Our method is originally motivated by NC-QR-DQN, where a special architecture is designed for the last layer of the neural network to satisfy the monotonicity of F^{-1}. The output represents the estimated quantile values at chosen quantile levels. One drawback of discretization is that a precise approximation for F^{-1} may need infinite levels. But in practice, one can only use finite quantile levels to estimate quantile values for decision making. In this work, we propose to learn a dense approximation for F^{-1} using monotonic rational-quadratic splines (Gregory & Delbourgo, 1982) as a building block.

3.3.1 Quantile Approximation with Monotonic Rational Quadratic Splines

Monotonic splines produce a monotonic interpolation to a set of monotonic data points (called knots) $\{(x_k, y_k)\}_{k=0}^{K}$, which has been recently used as a transformation function in normalizing flows (Durkan et al., 2019; Dolatabadi et al., 2020). Furthermore, denote $\{d_k\}_{k=0}^{K}$, a set of positive numbers, as the derivative of each knot. The monotonic rational-quadratic splines aim to find rational-quadratic functions with form $f_k(x) = \frac{O_k(x)}{P_k(x)}$ to fit the points and derivatives in each interval (called bin) $[x_k, x_{k+1}]$, where O_k and P_k are quadratic functions (with form $ax^2 + bx + c$).

Gregory & Delbourgo (1982) suggested to construct O_k and P_k as follows. Denote $g_k = (y_{k+1} - y_k)/(x_{k+1} - x_k)$ and $h_k(x) = (x - x_k)/(x_{k+1} - x_k)$ for $x \in [x_k, x_{k+1}]$. The expressions for the quadratic $O_k(h_k(x))$ and $P_k(h_k(x))$ for the k^{th} bin is defined by (use h_k for short of $h_k(x)$):

$$\begin{aligned} O_k(h_k) &= g_k y_{k+1} h_k^2 + (y_k d_{k+1} + y_{k+1}) h_k (1 - h_k) + g_k y_k (1 - h_k)^2 \\ P_k(h_k) &= g_k + (d_{k+1} + d_k - 2g_k) h_k (1 - h_k) \end{aligned} \tag{3.13}$$

Then, the rational-quadratic function for the k^{th} bin is computed by the quotient of O_k and P_k

$$f_k(h_k) = \frac{O_k(h_k)}{P_k(h_k)} = y_k + \frac{(y_{k+1} - y_k)[g_k h_k^2 + d_k h_k (1 - h_k)]}{g_k + (d_{k+1} + d_k - 2g_k) h_k (1 - h_k)}. \tag{3.14}$$

Equation 3.14 is proven to be monotonic and continuously differentiable, while passing through the knots and satisfying the given derivatives at the knots (Gregory & Delbourgo, 1982).

Proposition 1 (Gregory & Delbourgo (1982)) *The function f whose formulation in k-th bin given by Eq. 3.14 is monotonic on $[x_0, x_K]$*

Proof. Here we assume the derivatives satisfy the necessary conditions for monotonicity, i.e.,

$$\mathrm{sgn}(d_k) = \mathrm{sgn}(d_{k+1}) = \mathrm{sgn}(g_k) \tag{3.15}$$

For $x \in [x_k, x_{k+1}]$,

$$f'_k(x) = \frac{P'_k(h_k)Q_k(h_k) - P_k(h_k)Q'_k(h_k)}{(x_{k+1} - x_k)Q_k(h_k)^2} \tag{3.16}$$

The numerator is

$$P'_k(h_k)Q_k(h_k) - P_k(h_k)Q'_k(h_k) = (x_{k+1} - x_k)g_k^2[d_{k+1}h_k^2 + 2g_k h_k(1-h_k) + d_k(1-h_k)^2] \tag{3.17}$$

Since $h_k = \frac{x-x_k}{x_{k+1}-x_k}$, $h_k \in [0,1]$. Thus $h_k(1-h_k) \geq 0$. Using the necessary condition in Eq. 3.15, we have

$$\mathrm{sgn}\Big(d_{k+1}h_k^2 + 2g_k h_k(1-h_k) + d_k(1-h_k)^2\Big) = \mathrm{sgn}(g_k) \tag{3.18}$$

which shows the spline function is monotonic.

The fact that spline functions satisfy the end point values (y_k, y_{k+1}) and derivatives (d_k, d_{k+1}) is easy to check by setting $x = x_k$ and $x = x_{k+1}$.

The monotonicity of the above splines fits the non-decreasing property of F^{-1}. Let $F^{-1}_{Z(s,a)}(\zeta)$ be the quantile function for the random variable of the discounted total return $Z(s,a)$ with $\zeta \in [0,1]$. Given the number of bins K, the aims of the spline approximator for $F^{-1}_{Z(s,a)}$ are threefold. First, propose a partition for the domain of definition $[0,1]$ with $\zeta_0 < ... < \zeta_k < ... < \zeta_K$. Here $\zeta_0 = 0$ and $\zeta_K = 1$. Second, estimate the corresponding quantile values $q_0 < ... < q_k < ... < q_K$. Third, assess the derivatives at those points with $d_0, ..., d_k, ..., d_K$. We give a small fixed positive value for d_0 and d_K as they are assigned with endpoints. After the generation of these three sets of statistics, the monotonic spline of each bin is given by Equation 3.14 (by replacing x_k by ζ_k and y_k by q_k).

3.3.2 Model Implementation

We now show how to learn the monotonic splines for quantile functions in distributional RL by neural networks, and we name the technique *spline DQN* (SPL-DQN). As shown in Figure 3.2, the SPL-DQN consists of three major components, including a *Feature Extractor* which extracts latent features from a state, a *Knots Logit Network* which, for each action, generates the logits of the widths and heights for K bins, and derivatives for $K-1$ inner knots, and a *Bin Scale Network* which recovers the heights in $[0,1]$ to the original quantile

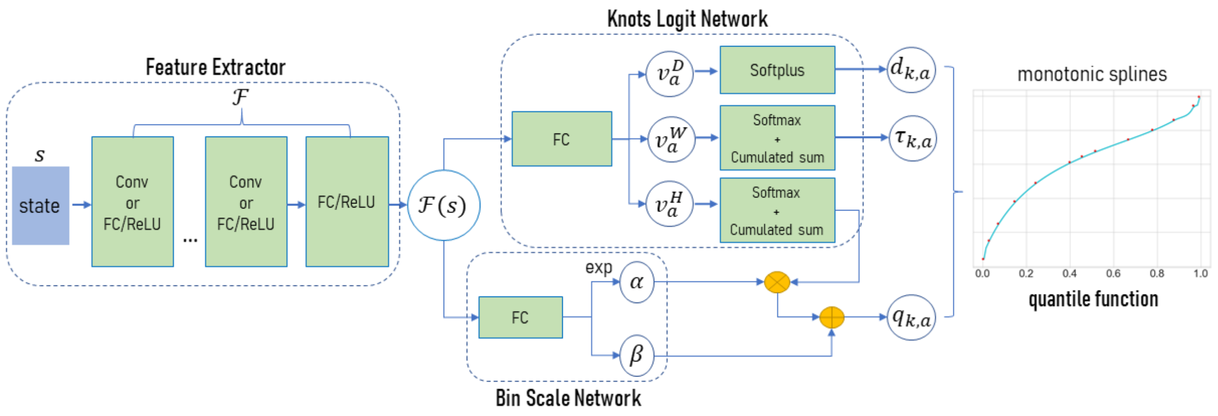


Figure 3.2: System Flow of SPL-DQN architecture

range. Here we describe the model for a discrete action space of size $|\mathcal{A}|$. To use monotonic splines in continuous control, the model can be modified by taking state-action pairs as input and only producing knots for that state-action pair.

The *Feature Extractor* $\mathcal{F}$ is usually made up of multiple convolutional layers with subsequent fully-connected layers for image-like inputs or stacked fully-connected layers for non-image inputs. It produces the feature embedding $\mathcal{F}(s) \in \mathbb{R}^d$ of state s. Then the *Knots Logit Network* $\mathcal{W}$ maps $\mathcal{F}(s)$ to unconstrained logits v with dimension $|\mathcal{A}| \times (3K-1)$ using a fully-connected layer. The vector v_a for each action a is partitioned as $v_a = [v_a^W, v_a^H, v_a^D]$, where v_a^W and v_a^H have length K, and v_a^D has length $K-1$. Instead of directly learning $\zeta_{k,a}$ and $q_{k,a}$ associated with each monotonic knot, we propose to learn the normalized width and height of each bin. Here, vectors v_a^W and v_a^H are each passed through a softmax function and are interpreted as the normalized widths and heights. Vector v_a^D is regarded as the derivatives, and is passed through a softplus function to satisfy monotonicity.

With the width and height of each bin, $\zeta_{k,a}$ and $q_{k,a}$ of each knot can be easily calculated by a cumulative sum. Since the values of v_a^W and v_a^H fall into $[0,1]$, each $\zeta_{k,a}$ $(k>0)$ is computed by

$$\zeta_{k,a} = \sum_{i=1}^{k} v_{i,a}^W, \;\; k = 1, ..., K; \;\; a = 1, ..., |\mathcal{A}| \tag{3.19}$$

without rescaling as the domain of a quantile function is $[0,1]$ ($\zeta_{0,a} = 0$ by definition). To compute each $q_{k,a}$, another transformation is required to rescale v_a^H to a range corresponding to the true quantile values. Inspired by NC-QR-DQN, we introduce the *Bin Scale Network* to generate two adaptive scale factors α and β by applying a fully connected layer

$\mathcal{C}$: $\mathbb{R}^d \to \mathbb{R}^{|\mathcal{A}|\times 2}$ to the state embedding $\mathcal{F}(s)$. We compute the exponential of α to ensure the total bin height is positive. Then $q_{0,a} = \beta_a$ and for $k > 0$, $q_{k,a}$ is computed by

$$q_{k,a} = \exp(\alpha_a) \times \sum_{i=1}^{k} v_{i,a}^H + \beta_a,\ \ k = 1, ..., K;\ \ a = 1, ..., |\mathcal{A}| \tag{3.20}$$

3.3.3 Approximate Wasserstein Metric Minimization

When using continuous approximations of the quantile functions, there are several choices to compute the integral of the Wasserstein metric between two quantile functions. We can try to calculate the integral directly, but this is not straightforward for rational-quadratic functions since the integral rarely has a closed form. Alternatively, we can calculate the Riemann integral, but this leads to a loss function analogous to the L1-norm, which may cause instability in training. Thus, in this work, we perform quantile regression (Koenker & Hallock, 2001) in a projected space to approximately minimize the Wasserstein metric.

Let $\tilde{\zeta} = (\tilde{\zeta}_0, ..., \tilde{\zeta}_N)$ be a fixed sequence of non-decreasing quantile levels (note that the set $\tilde{\zeta}$ is different from the x-values of knots in Section 3.3.1 to partition the $[0, 1]$ domain, which are learned by the neural network. In our experiments, we let $\tilde{\zeta}$ be uniformly fixed), we project the monotonic spline quantile function f to a quantile distribution space $\mathcal{Z}_Q$ by computing

$$Z_q(s, a) = \sum_{i=1}^{N} (\tilde{\zeta}_i - \tilde{\zeta}_{i-1})\delta_{\hat{q}_i(s,a)}, \tag{3.21}$$

where each $\hat{q}_i(s, a)$ is the corresponding quantile value at the quantile level $\hat{\zeta}_i = \frac{\tilde{\zeta}_{i-1}+\tilde{\zeta}_i}{2}$ given by $f(\hat{\zeta}_i)$ with $1 \leq i \leq N$. To compute $f(\hat{\zeta}_i)$, we first search which bin $\hat{\zeta}_i$ lies in. Then the value is returned by the corresponding spline function given $h_k(\hat{\zeta}_i)$ as input. In this case, the optimal value distribution Z is achieved by minimizing the 1-Wasserstein metric with Z_q

$$W_1(Z(s, a), Z_q(s, a)) = \sum_{i=1}^{N} \int_{\tilde{\zeta}_{i-1}}^{\tilde{\zeta}_i} |F^{-1}_{Z(s,a)}(\omega) - \hat{q}_i(s, a)| d\omega, \tag{3.22}$$

which is equivalent to finding a projection operator Π_{W_1} such that

$$\Pi_{W_1} Z := \arg \min_{Z_q \in \mathcal{Z}_Q} (Z, Z_q). \tag{3.23}$$

Furthermore, Dabney et al. (2018b) shows that the unique minimizer of this operator is given by

$$F^{-1}_{Z(s,a)}(\hat{\zeta}_i) = \hat{q}_i(s,a),\ \hat{\zeta}_i = \frac{\tilde{\zeta}_{i-1} + \tilde{\zeta}_i}{2} \tag{3.24}$$

Proposition 2 (Proposition 2 in Dabney et al. (2018b)) *Let Π_{W_1} be the quantile projector defined above. When applied to value distributions, it gives a projection for each state-value distribution. For any two value distributions $Z_1, Z_2 \in \mathcal{Z}$ for an MDP with countable state and action spaces,*

$$\overline{d}_\infty(\Pi_{W_1}\mathcal{T}^\pi Z_1, \Pi_{W_1}\mathcal{T}^\pi Z_2) \leq \gamma \overline{d}_\infty(Z_1, Z_2), \tag{3.25}$$

where $\overline{d}_p(Z_1, Z_2) = \sup_{s,a} W_p(Z_1(s,a), Z_2(s,a))$ and $\mathcal{Z}$ is the space of action-value distributions with finite moments.

Proposition 2 suggests that after projecting f to Z_q, the operator $\Pi_{W_1}\mathcal{T}^\pi$ is a γ-contraction under the measure $\overline{d}_\infty$ and the repetition of this operator converges to a fixed point in space $\mathcal{Z}_Q$.

Based on Proposition 2, the ultimate goal is to estimate quantile values in Equation 3.24 for $F^{-1}_{Z(s,a)}$ using quantile regression in each training batch. In our implementation, we uniformly fix $\tilde{\zeta} = (\tilde{\zeta}_0, ..., \tilde{\zeta}_N)$ to be consistent with QR-DQN and NC-QR-DQN, which leads to the same quantile Huber loss as shown in Equations 3.9 and 3.10. However, the advantage of our method over QR-DQN and NC-QR-DQN is that we can freely enrich the density of $\tilde{\tau}$ to get a better estimation of the quantile function without increasing the size of the model architecture, while QR-DQN and NC-QR-DQN must enlarge the output dimension of their models to get more quantile estimates. Since we can freely query quantile values at any quantile level, quantile level embedding as done in IQN and FQF is no longer necessary in our method.

Remark: Although one recent method, NDQFN, also learns continuous monotonic quantile functions, our method is different from NDQFN in three aspects. First, the x-values of those monotonic knots, i.e., $\zeta_0, ..., \zeta_K$, are uniformly fixed in NDQFN, while they are trainable in our method. Second, by also learning the derivatives at each knot, we get a smooth interpolation over the entire domain, while NDQFN connects those knots with line segments, which has limited approximation ability. Third, to get the increments of y-values of those knots, i.e., $q_0, ..., q_K$, NDQFN learns a function taking the quantile level embeddings, i.e., the embeddings of corresponding ζs, as input, while we do not calculate increments but use a scale network as discussed in Section 3.3.2.

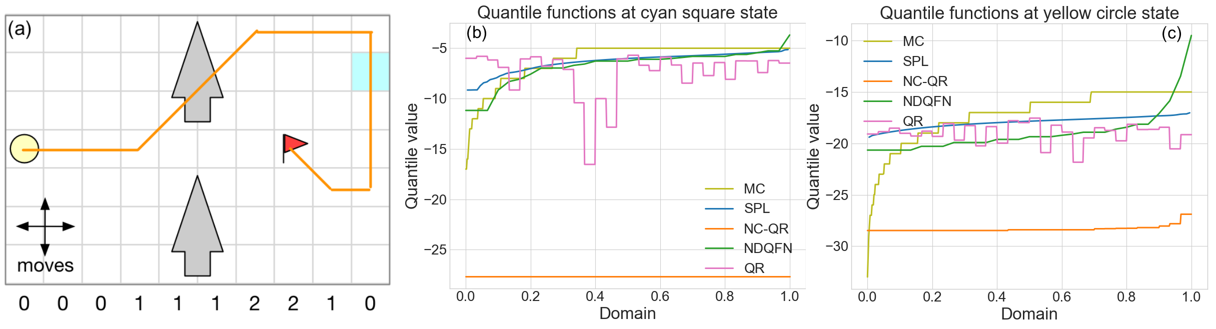


Figure 3.3: (a) Windy Gridworld, with wind strength shown along bottom row. The detial of the domain is described in text. (b) & (c) The quantile functions for value distribution of the cyan square state and yellow circle state by MC, SPL-DQN (SPL), NC-QR-DQN (NC-QR), NDQFN, and QR-DQN (QR).

To demonstrate the monotonicity and approximation strength of our method in stochastic environments, we plot the quantile functions learned by SPL-DQN, NC-QR-DQN, NDQFN, and QR-DQN in a variant of the classic Windy Gridworld domain (Sutton & Barto, 2018). In Figure 3.3a, the agent starts at the yellow circle state and makes standard moves in a gridworld to reach the red flag. A reward of -1 is earned at each step. Some columns are affected by some wind blowing from bottom to top. The orange line shows the optimal trajectory without stochasticity. We set each state transition to have probability 0.1 of moving in a random direction without any wind effect, otherwise the transition is affected by the wind, which pushes the agent northward. All methods here use the same training settings and similar network architectures as discussed in Appendix A.1.1. We compute the ground truth value distribution for an optimal policy (learned by policy iteration) at each state by performing one thousand Monte-Carlo (MC) rollouts and recording the observed returns as an empirical distribution. Then we transform the empirical distribution to the quantile function as the baseline. Here we show case the learned quantile functions at cyan square state and yellow circle state (start state) as shown in Figures 3.3b and 3.3c.

All these four methods eventually learn the optimal policy, however their quantile approximations are quite different. Without constraints, quantile functions given by QR-DQN clearly violate the monotonic property, which is known as the quantile crossing issue (Zhou et al., 2020). Although NC-QR-DQN applies monotonic constraints, the estimated quantile range is biased towards smaller values according to the quantile functions given by MC, and we observe that the quantile functions learned by NC-QR-DQN are straight lines for some states, e.g. cyan square state, which means that it fails to learn the value distribution

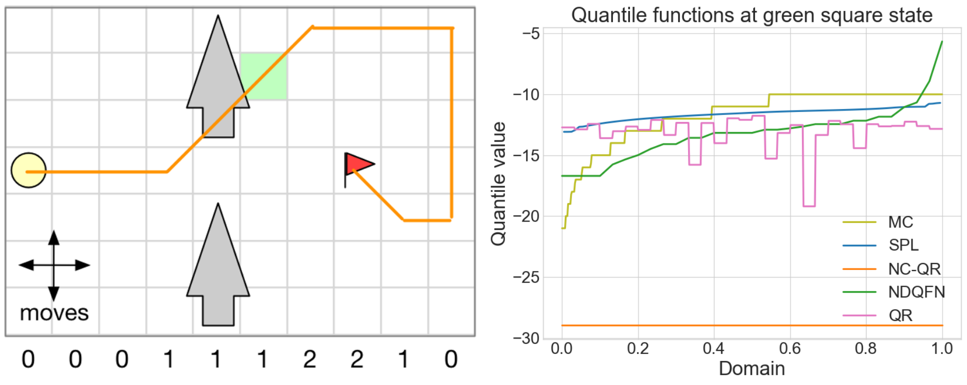


Figure 3.4: The learned quantile functions at the green square state. The detail of the Windy Gridworld is described in text.

in those states, and in turn this leads to a biased estimation for the start state. The reason for this biased estimation is that in NC-QR-DQN, when rescaling the quantile range in Equation 3.11, a ReLU function is imposed to the coefficient α to ensure it is non-negative. However, this often sets α to zero and the quantile distribution will only depend on the shift parameter β (which leads to a straight line). In this case, the value distribution cannot be precisely captured. For NDQFN, its quantile approximation at the goal nearing state (cyan square state) is close to SPL-DQN , but it overestimates the quantile range at the start state. We also observe the overestimation issue of NDQFN at another state in the middle of the orange line trajectory as shown in Fig. 3.4. Though still exhibiting estimation errors, the quantile functions learned by SPL-DQN are often the closest to the ground truth.

3.4 Experiments

While most previous distributional RL algorithms were evaluated with Atari games from the Arcade Learning Environment (ALE), it was noted that the ALE is deterministic (Bellemare et al., 2017) and therefore questionable as a benchmark to evaluate distributional algorithms that are designed to capture environment stochasticity when there is none. However, we note that sticky actions can be used in Atari games to introduce stochasticity in policies (Machado et al., 2018) and this regime was used to evaluate IQN and FQF (Yang et al., 2019). When the environment is deterministic, value distributions still arise due to stochastic policies, stochastic approximations and random parameter initialization, but the resulting value distributions tend to be simple and close to deterministic.

It is also well-known that deterministic environments possess optimal policies that are open-loop and therefore ignore observations (Machado et al., 2018; Koul et al., 2019). In practice, it is often desirable to train controllers with simulators in which noise is injected to increase the robustness of the learned policies in case of discrepancies between the simulator and the real world. Hence, in this work, we modify several robotics environments by adding stochasticity, including one discrete environment from OpenAI Gym (Brockman et al., 2016b) and nine continuous environments from PyBulletGym (Ellenberger, 2018–2019). We compare our method with QR-DQN, IQN, FQF, NC-QR-DQN, MM-DQN, and NDQFN. For MM-DQN, we used the unrectified Kernel $\kappa_\alpha(x, y) = -||x - y||^\alpha$ with $\alpha = 1$ (parameter taken from Nguyen et al. (2021)) instead of the Gaussian kernel recommended by the authors when they tested on Atari games since the unrectified kernel gave better results in the robotics benchmarks used in this thesis. For a fair comparison, we made sure the same *Feature Extractor* architecture was used in different models. To simplify acronyms, we omit -DQN when referring to a method in what follows.

3.4.1 Computing QR Loss in Different Methods

We first briefly summarize how different QR-based distributional RL methods sample quantile values when computing the QR loss. Since the QR loss is computed in a time difference manner, we will need N current quantile samples $\{q_i^1\}, i = 1, ..., N$ and N' target quantile samples $\{q_i^2\}, i = 1, ..., N'$ corresponding to two quantile level sets $\{\zeta_i^1\}, i = 1, ..., N$ and $\{\zeta_i^2\}, i = 1, ..., N'$. Without loss of generalization, we consider $N = N'$. Here we discuss the case with discrete actions, and denote the action space by $|\mathcal{A}|$.

For discrete quantile approximations, including QR-DQN, NC-QR-DQN, IQN, and FQF, in order to get N quantile samples for each action, the output dimension of the model is $|\mathcal{A}| \times N$ for an input state. For QR-DQN and NC-QR-DQN, $\{\tau_i^1\}$ and $\{\tau_i^2\}$ are assumed to be uniformly spaced. For IQN, $\{\zeta_i^1\}$ and $\{\zeta_i^2\}$ are independently drawn from a uniform distribution $U([0, 1])$. For FQF, $\{\zeta_i^1\}$ and $\{\zeta_i^2\}$ are proposed by a quantile fraction network.

For methods that learn a continuous approximation of the quantile function, including SPL-DQN and NDQFN, the output (for an input state) consists of knots with shape $|\mathcal{A}| \times (K + 1)$ when the domain is divided into K bins. For SPL-DQN, it learns the x, y values, and derivatives of those knots. A smooth continuous function with closed form is obtained in each bin. When sampling quantile values to compute the QR loss, SPL-DQN uniformly fixes $\{\zeta_i^1\}$ and $\{\zeta_i^2\}$, and $\{q_i^1\}$ and $\{q_i^2\}$ are obtained by querying the closed form with $\{\zeta_i^1\}$ and $\{\zeta_i^2\}$ as inputs. For NDQFN, it only learns the y-values of those knots, and

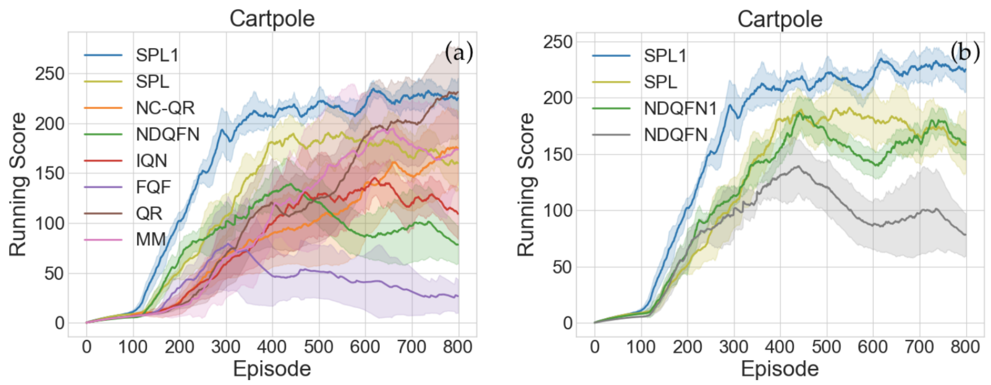


Figure 3.5: Performance comparison in stochastic Cartpole. Each curve is averaged over 5 seeds with shaded area indicating standard error.

the x-values of the knots are uniformly spaced. The continuous function is constructed by connecting neighboring knots with linear functions. When sampling quantile values to compute the QR loss, NDQFN draws $\{\tau_i^1\}$ and $\{\tau_i^2\}$ from a uniform distribution $U([0,1])$ independently, and $\{q_i^1\}$ and $\{q_i^2\}$ are obtained by querying the linear functions in each bin.

3.4.2 Discrete control in Cartpole

We begin our experimental results in a stochastic environment with a discrete action space modified from Cartpole (Florian, 2007). The system is controlled by a force of $+1$ or -1 applied to the cart. A reward of $+1$ is returned if the pole remains upright. We set each state transition to have probability 0.05 of moving to a neighboring state to make the environment stochastic. The QR-based methods use $N = 8$ quantiles to compute the QR loss. MM-DQN uses $N = M = 8$ samples. More training details are provided in Appendix A.1.2.

As the episode rewards may vary significantly due to stochasticity, to better reflect the training process, we define the running score as a soft update of episode rewards:

$$running\_score = 0.99 \times running\_score + 0.01 \times episode\_rewards \tag{3.26}$$

Figure 3.5 shows the running score curves for stochastic Cartpole. In general, SPL learns much faster (faster empirical convergence) than its counterparts. As discussed before, SPL can freely increase the number of quantiles when performing quantile regression without enlarging the output dimension of the model. We further increase the number of quantiles to 24 to compute the QR loss while keeping the number of bins unchanged ($K = 8$ and $N = 24$), yielding the curve labeled 'SPL1' in Figure 3.5. This curve shows that approximately minimizing the Wasserstein metric with more quantiles leads to better quantile approximations and increases the learning speed and performance of SPL. As NDQFN also learns continuous quantile functions, we do the same experiment ($K = 8$ and $N = 24$) for NDQFN, whose training curve is labeled by 'NDQFN1' in Figure 3.5(b) (to make the comparison clear, we show this in another figure). Although its training performance improves, SPL with $N = 24$ is still better.

Table 3.1: Noise settings for different environments in PyBulletGym

Environments	Noise
InvertedPendulum	$\mathcal{N}(0, 0.02)$
InvertedDoublePendulum	$\mathcal{N}(0, 0.01)$
InvertedPendulumSwingup	$\mathcal{N}(0, 0.05)$
Reacher	$\mathcal{N}(0, 0.01)$
Walker2D	$\mathcal{N}(0, 0.005)$
HalfCheetah	$\mathcal{N}(0, 0.005)$
HalfCheetah1	$\mathcal{N}(0, 0.008)$
HalfCheetah2	$\mathcal{N}(0, 0.01)$
Ant	$\mathcal{N}(0, 0.01)$
Hopper	$\mathcal{N}(0, 0.003)$
Humanoid	$\mathcal{N}(0, 0.003)$

3.4.3 Continuous control in PyBulletGym

PyBulletGym provides RoboSchool[1], which is a free port of MuJoCo[2]. The state of these environments contains joint information of a robot and an action is a multi-dimensional continuous vector. We take nine environments from RoBoSchool and make them stochastic by introducing Gaussian noise $\mathcal{N}(\mu, \sigma)$ to both the location and velocity of each part of the robot, with $\mu = 0$ and σ varying in different environments. We choose a reasonable σ for each environment such that robots won't exhibit unrealistic motion. That is, for noise sensitive environments, such as Walker2D and Humanoid, we use a smaller σ, and for

[1]https://openai.com/blog/roboschool/
[2]http://www.mujoco.org

relatively easy tasks, like InvertedPendulumSwingup, we choose a bigger one. The noise setting for different environments is shown in Table 3.1 in the appendix.

To evaluate on continuous control tasks, we combine distributional RL with DDPG (Lillicrap et al., 2016a) by modifying the critic, as done by Zhang & Yao (2019). Instead of learning Q, the critic learns the distribution Z directly. To handle continuous actions, the critic takes state-action pairs as input. As an exception, for the Humanoid environment, we combine distributional RL with SAC (Haarnoja et al., 2018) due to the fact that DDPG is not as good as SAC for this environment. To update the actor in DDPG and SAC, the expectation of Q values is computed as the expectation of quantile samples given by the distributional critic. We refer to the original papers for hyperparameter settings, which are discussed in Appendix A.1.3. We also include raw DDPG and SAC as baselines.

Figure 3.6 shows the running score curves given by Equation 3.26 for these stochastic environments. Generally, the training performance varies among different approaches in different environments, however, in most cases, the quantile regression based methods who learn monotonic quantile representations are better than those whose quantile representations have no monotonicity guarantee, which clarifies that the quantile crossing issue can distort policy learning as pointed out by Zhou et al. (2020). Especially, for SPL, apart from Reacher and InvertedPendulumSwingup, it always converges faster and performs better during training. For InvertedPendulumSwingup, SPL performs comparably to NC-QR. Although NDQFN also learns continuous monotonic quantile functions, its performance is even worse than NC-QR in most cases, because NDQFN queries linear functions for quantile samples when computing QR loss, but the approximation ability of piecewise linear function is very limited. For methods with no monotonic quantile guarantee, we notice that although IQN is the best in Reacher, it performs worse in InvertedPendulum and Humanoid during training.

To further demonstrate the ability of our method to handle uncertainty of the environment, we slightly increase the noise in HalfCheetah to $\mathcal{N}(0, 0.008)$ (labeled by HalfCheetah1) and $\mathcal{N}(0, 0.01)$ (labeled by HalfCheetah2). The training curves in these two environments are shown in Figure 3.7. On average, QR and MM for DDPG behave poorly in these three HalfCheetah variants. The enhanced randomness of environments degrades the training performance of SPL, but SPL is generally faster and better than NC-QR and IQN, thanks to more precise quantile approximations.

During training, Ornstein-Uhlenheck noise $\mathcal{OU}(\mu', \sigma')$ (Uhlenbeck & Ornstein, 1930) is utilized when selecting actions to induce exploration in DDPG. At the evaluation stage, the methods are executed with only exploitation (without action noise). We test the best models we get after training for each method, and the testing score across different

Table 3.2: Scaled testing scores across different stochastic environments. Scores are averaged over 4 seeds.

Environments	MM	QR	FQF	IQN	NDQFN	NC-QR	SPL
InvertedPendulum	0.911	0.940	0.953	0.970	0.992	0.969	**0.999**
InvertedDoublePendulum	0.814	0.978	0.975	0.967	0.990	0.993	**1.019**
InvertedPendulumSwingup	0.461	0.945	0.223	0.944	1.091	1.145	**1.179**
Reacher	-1.501	0.412	-10.546	**4.269**	3.416	4.241	2.972
Walker2D	0.503	0.661	0.585	1.375	0.776	1.732	**3.142**
HalfCheetah	0.731	1.084	0.809	2.122	1.039	2.932	**3.004**
HalfCheetah1	0.763	0.897	0.859	1.764	1.156	2.158	**2.633**
HalfCheetah2	0.834	0.855	0.773	1.741	1.231	1.812	**2.21**
Ant	0.871	2.283	0.345	2.403	2.388	3.045	**3.321**
Hopper	0.689	0.868	0.671	0.960	0.893	1.405	**1.609**
Humanoid	1.077	1.409	0.035	0.044	1.108	1.558	**1.640**

environments are shown in Table 3.2. We test all DDPG based agents without Ornstein-Uhlenheck noise for 0.125 million frames, and SAC based agents for 2.5 thousand episodes. We treat DDPG and SAC scores as baselines and scale other methods' scores by them, i.e.

$$method\_scaled\_test\_score = \frac{method\_raw\_test\_score}{DDPG/SAC\_raw\_test\_score} \tag{3.27}$$

Apart from Reacher, SPL outperforms its counterparts in all other domains. For the first two environments, although the training performances vary significantly among different methods, the testing scores of their best models are close to each other. For most remaining environments, the testing scores of SPL and NC-QR are significantly better than other methods.

Different sampling regimes of SPL and NDQFN

Since both SPL and NDQFN learn continuous quantile function, different sampling strategies can be applied to sample quantiles for QR loss. As discussed in Sec. 3.4.1, in the original paper of NDQFN, it samples quantile levels $\{\zeta_i\}$ from $U(0, 1)$ as done in IQN. While our method SPL fixes quantile level $\{\zeta_i\}$ as uniformly spaced as done in QR-DQN. However, NDQFN can also use uniformly spaced quantile levels, and SPL can also use quantile levels sampled from $U(0, 1)$.

In Fig. 3.8, we label the learning curve by SPL-rnd to denote the SPL with quantile levels sampled from $U(0, 1)$, and label the curve by NDQFN-uni to denote the NDQFN with uniformly spaced quantile levels. In general, there is no clear advantage of on sampling strategy over the other, e.g., in Reacher, SPL-rnd is better than SPL (which is actually SPL-uni), but in Ant, SPL (SPL-uni) is better than SPL-rnd. But SPL outperforms NDQFN in both training regimes. Note that though original SPL does not work well in the Reacher domain, SPL-rnd instead, achieves a comparable result as the best method, i.e., IQN, in this domain.

3.5 Summary

Based on previous works in distributional RL, in this chapter, we propose a more general and precise approximation for quantile functions using monotonic rational-quadratic splines. With a monotonic continuous representation of the quantile function, the quantile value at every quantile level is accessible during training, yielding greater accuracy. In the windy gridworld domain, the learned value distribution is closer to the ground truth compared with other distributional RL methods. In stochastic robotics domain, our method leads to higher expected returns. Particularly, comparing with piece-wise linear function interpolation in NDQFN, our rational quadratic spline parameterization performs better in most cases.

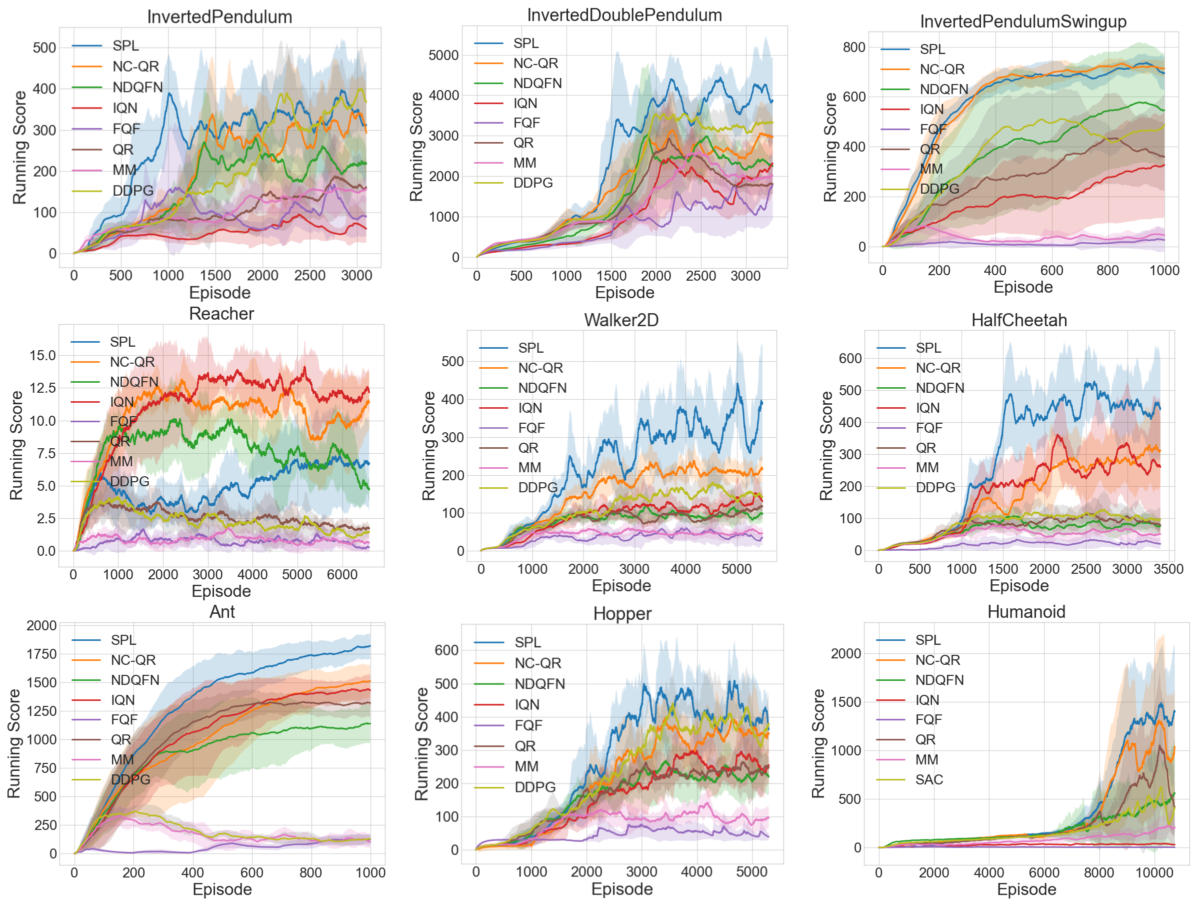


Figure 3.6: Performance comparison in stochastic RoboSchool. Each curve is averaged by 7 seeds. The first eight environments are solved with DDPG. The last one is assigned to SAC.

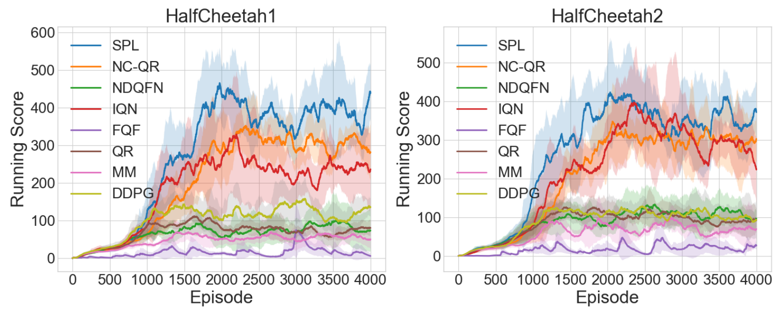


Figure 3.7: Performance comparison in two stochastic HalfCheetahs with enhanced randomness.

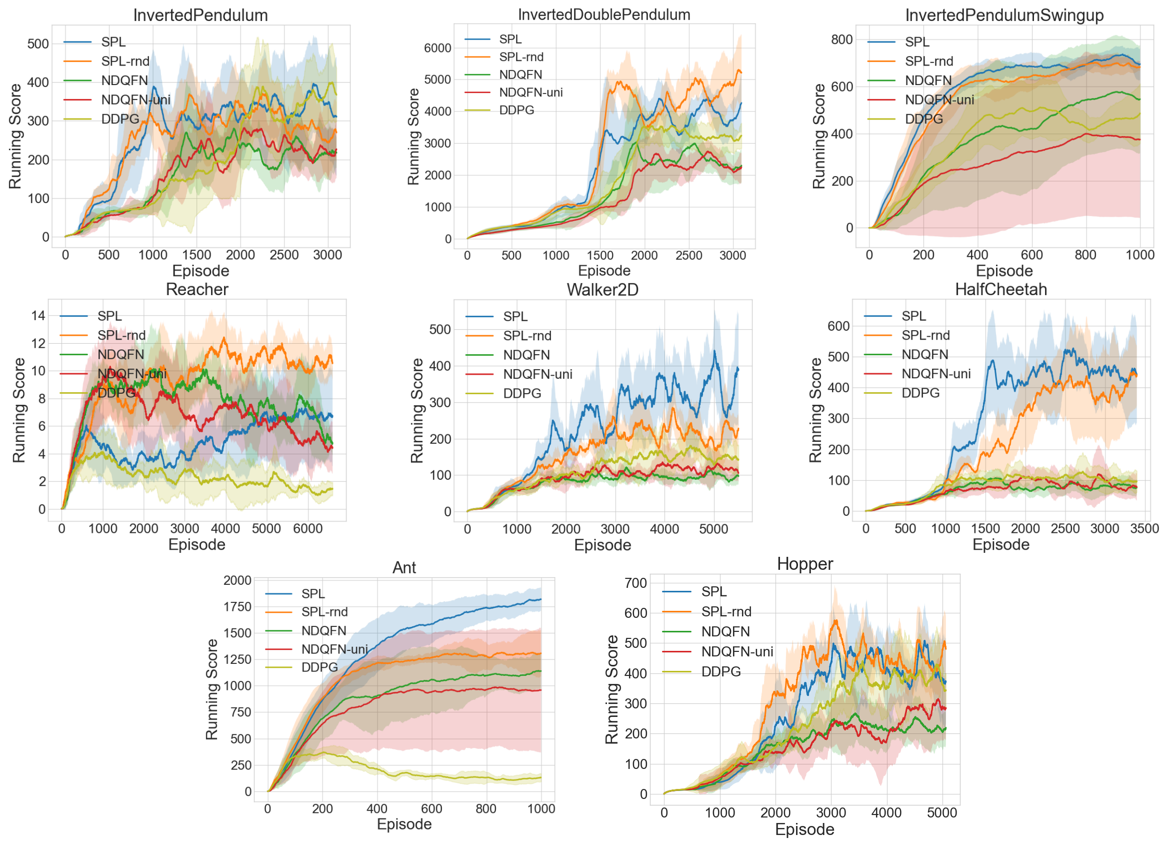


Figure 3.8: Performance comparison of SPL and NDQFN when trained with uniformly spaced quantile fractions or random quantile fractions sampled from $U([0,1])$ in eight environments with DDPG as the baseline

Chapter 4

Optimizing Measure of Variability in RL: Gini Deviation as an Alternative to Variance

4.1 Introduction

Although learning a value distribution is discussed in Chapter 3, it remains in the risk-neutral setting. The demand for avoiding risks in practical applications has inspired risk-averse reinforcement learning (RARL). For example, we want to avoid collisions in autonomous driving (Naghshvar et al., 2018), or avoid huge financial losses in portfolio management (Björk et al., 2014). In these cases, we would like to optimize a risk measure of the total return instead of maximizing the expectation only.

Many risk measures have been studied for RARL, for instance, exponential utility functions (Borkar, 2002), value at risk (VaR) (Chow et al., 2017), conditional value at risk (CVaR) (Chow & Ghavamzadeh, 2014; Greenberg et al., 2022), and variance (Tamar et al., 2012; La & Ghavamzadeh, 2013). In this chapter, we mainly focus on **measures of variability**, where variance is a popular choice, as variance has advantages in interpretability and computation (Markowitz & Todd, 2000; Li & Ng, 2000). Such a paradigm is referred to as mean-variance RL. Traditional mean-variance RL methods consider the variance of the total return random variable. Usually, the total return variance is treated as a constraint to the RL problem, i.e., it is lower than some threshold (Tamar et al., 2012; La & Ghavamzadeh, 2013; Xie et al., 2018). Recently, Bisi et al. (2020) proposed a reward-volatility risk measure, which considers the variance of the per-step reward random variable. Bisi et al. (2020) shows that the per-step reward variance is an upper bound of the total return variance and can better capture the short-term risk. Zhang et al. (2021) further simplifies Bisi et al. (2020)'s method by introducing Fenchel duality.

Directly optimizing total return variance is challenging. It either necessitates double sampling (Tamar et al., 2012) or calls for other techniques to avoid double sampling for faster learning (Tamar et al., 2012; La & Ghavamzadeh, 2013; Xie et al., 2018). As for the reward-volatility risk measure, Bisi et al. (2020) uses a complicated trust region optimization due to the modified reward's policy-dependent issue. Zhang et al. (2021) overcomes this issue by modifying the reward according to Fenchel duality. However, this reward modification strategy can possibly hinder policy learning by changing a "good" reward to a "bad" one, which we discuss in detail in this work.

To overcome the limitations of variance-based risk measures, we propose to use a new measure of variability: Gini deviation (GD). We first review the background of mean-variance RL. Particularly, we explain the limitations of both total return variance and per-step reward variance risk measures. We then introduce GD as a dispersion measure for random variables and highlight its properties for utilizing it as a measure of variability in policy gradient methods. Since computing the gradient using the original definition of GD is challenging, we derive the policy gradient algorithm from its quantile representation to minimize it. To demonstrate the effectiveness of our method in overcoming the limitations of variance-based risk measures, we modify several domains (Guarded Maze (Greenberg et al., 2022), Lunar Lander (Brockman et al., 2016a), Mujoco (Todorov et al., 2012)) where risk-aversion can be clearly verified. We show that our method can learn risk-averse policy with high return and low risk in terms of variance and GD, when others fail to learn a reasonable policy.

4.2 Background: Mean-Variance RL

Mean-variance RL aims to maximize $\mathbb{E}[G_0]$ and additionally minimize its variance $\mathbb{V}[G_0]$(Tamar et al., 2012; La & Ghavamzadeh, 2013; Xie et al., 2018). Generally, there are two ways to define a variance-based risk. The first one defines the variance based on the Monte Carlo **total return** G_0. The second defines the variance on the **per-step reward** R. We review these methods and their limitations in the following subsections. We will refer to π, π_θ and θ interchangeably throughout this chapter when the context is clear.

4.2.1 Total Return Variance

Methods proposed by Tamar et al. (2012); La & Ghavamzadeh (2013); Xie et al. (2018) consider the problem

$$\max_{\pi} \mathbb{E}[G_0], \quad \text{s.t. } \mathbb{V}[G_0] \leq \xi \tag{4.1}$$

where ξ indicates the user's tolerance of the variance. Using the Lagrangian relaxation procedure (Bertsekas, 1997), we can transform it to the following unconstrained optimization problem: $\max_\pi \mathbb{E}[G_0] - \lambda \mathbb{V}[G_0]$, where λ is a trade-off hyper-parameter. Note that the mean-variance objective is in general NP-hard (Mannor & Tsitsiklis, 2011) to optimize. The main reason is that although variance satisfies a Bellman equation, it lacks the monotonicity of dynamic programming (Sobel, 1982).

Double Sampling in total return variance. We first show how to solve unconstrained mean-variance RL via vanilla stochastic gradient. Suppose the policy is parameterized by θ, define $J(\theta) = \mathbb{E}_\pi[G_0]$ and $M(\theta) := \mathbb{E}_\pi\big[(\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t))^2\big]$, then $\mathbb{V}[G_0] = M(\theta) - J^2(\theta)$. The unconstrained mean-variance objective is equivalent to $J_\lambda(\theta) = J(\theta) - \lambda\big(M(\theta) - J^2(\theta)\big)$, whose gradient is

$$\nabla_\theta J_\lambda(\theta_t) = \nabla_\theta J(\theta_t) - \lambda \nabla_\theta (M(\theta) - J^2(\theta)) \tag{4.2}$$

$$= \nabla_\theta J(\theta_t) - \lambda\big(\nabla_\theta M(\theta) - 2J(\theta)\nabla_\theta J(\theta)\big) \tag{4.3}$$

The unbiased estimates for $\nabla_\theta J(\theta)$ and $\nabla_\theta M(\theta)$ can be estimated by approximating the expectations over trajectories by using a single set of trajectories as discussed in Section 2.3, i.e., $\nabla_\theta J(\theta) = \mathbb{E}_\tau[R_\tau \omega_\tau(\theta)]$ and $\nabla_\theta M(\theta) = \mathbb{E}_\tau[R_\tau^2 \omega_\tau(\theta)]$, where R_τ is the return of trajectory τ and $\omega_\tau(\theta) = \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)$. In contrast, computing an unbiased estimate for $J(\theta)\nabla_\theta J(\theta)$ requires two distinct sets of trajectories to estimate $J(\theta)$ and $\nabla_\theta J(\theta)$ separately, which is known as double sampling.

Remark. Some work claims that double sampling cannot be implemented without having access to a generative model of the environment that allows users to sample at least two next states (Xie et al., 2018). This is, however, not an issue in our setting where we allow sampling multiple trajectories. As long as we get enough trajectories, estimating $J(\theta)\nabla_\theta J(\theta)$ is possible.

Still, different methods were proposed to avoid this double sampling for faster learning. Specifically, Tamar et al. (2012) considers the setting $\gamma = 1$ and considers an unconstrained problem:

$$\max_\theta L_1(\theta) = \mathbb{E}[G_0] - \lambda g\big(\mathbb{V}[G_0] - \xi\big) \tag{4.4}$$

where $\lambda > 0$ is a tunable hyper-parameter, and penalty function $g(x) = (\max\{0, x\})^2$. This method produces faster estimates for $\mathbb{E}[G_0]$ and $\mathbb{V}[G_0]$ and a slower updating for θ at each episode, which yields a two-time scale algorithm. La & Ghavamzadeh (2013) considers the setting $\gamma < 1$ and converts Formula 4.1 into an unconstrained saddle-point problem:

$$\max_{\lambda} \min_{\theta} L_2(\theta, \lambda) = -\mathbb{E}[G_0] + \lambda\big(\mathbb{V}[G_0] - \xi\big) \tag{4.5}$$

where λ is the dual variable. This approach uses a perturbation method and a smoothed function method to compute the gradient of value functions with respect to policy parameters. Xie et al. (2018) considers the setting $\gamma = 1$, and introduces Fenchel duality $x^2 = \max_y(2xy - y^2)$ to avoid the term $J(\theta)\nabla_\theta J(\theta)$ in the gradient. The original problem is then transformed into

$$\max_{\theta, y} L_3(\theta, y) = 2y\big(\mathbb{E}[G_0] + \frac{1}{2\lambda}\big) - y^2 - \mathbb{E}[G_0^2] \tag{4.6}$$

where y is the dual variable.

Limitations of Total Return Variance. The presence of the square term R_τ^2 in the mean-variance gradient $\nabla_\theta M(\theta) = \mathbb{E}_\tau[R_\tau^2 \omega_\tau(\theta)]$(Equation 4.2) makes the gradient estimate sensitive to the numerical scale of the return, as empirically verified later. This issue is inherent in all methods that require computing $\nabla_\theta \mathbb{E}[G_0^2]$. Users can not simply scale the reward by a small factor to reduce the magnitude of R_τ^2, since when scaling reward by a factor c, $\mathbb{E}[G_0]$ is scaled by c but $\mathbb{V}[G_0]$ is scaled by c^2. Consequently, scaling the reward may lead to different optimal policies being obtained.

4.2.2 Per-step Reward Variance

A recent perspective uses per-step reward variance $\mathbb{V}[R]$ as a proxy for $\mathbb{V}[G_0]$. The probability mass function of R is

$$\Pr(R = x) = \sum_{s,a} d_\pi(s, a)\mathbb{I}_{r(s,a)=x} \tag{4.7}$$

where $\mathbb{I}$ is the indicator function, and

$$d_\pi(s, a) = (1 - \gamma)\sum_{t=0}^{\infty} \gamma^t \Pr(S_t = s, A_t = a|\pi, P) \tag{4.8}$$

is the normalized discounted state-action distribution. Then we have $\mathbb{E}[R] = (1-\gamma)\mathbb{E}[G_0]$ and $\mathbb{V}[G_0] \leq \frac{\mathbb{V}[R]}{(1-\gamma)^2}$ (see Lemma 1 of Bisi et al. (2020)). Thus, Bisi et al. (2020) considers the following objective

$$\hat{J}_\lambda(\pi) = \mathbb{E}[R] - \lambda\mathbb{V}[R] = \mathbb{E}[R - \lambda(R - \mathbb{E}[R])^2] \tag{4.9}$$

This objective can be cast as a risk-neutral problem in the original MDP, but with a new reward function $\hat{r}(s,a) = r(s,a) - \lambda\big(r(s,a) - (1-\gamma)\mathbb{E}[G_0]\big)^2$. However, this $\hat{r}(s,a)$ is non-stationary (policy-dependent) due to the occurrence of $\mathbb{E}[G_0]$, so standard risk-neutral RL algorithms cannot be directly applied. Instead, this method uses trust region optimization (Schulman et al., 2015) to solve.

Zhang et al. (2021) introduces Fenchel duality to Equation 4.9. The transformed objective is

$$\hat{J}_\lambda(\pi) = \mathbb{E}[R] - \lambda\mathbb{E}[R^2] + \lambda \max_y(2\mathbb{E}[R]y - y^2) \tag{4.10}$$

which equals to

$$\max_{\pi,y} J_\lambda(\pi, y) = \sum_{s,a} d_\pi(s,a)\big(r(s,a) - \lambda r(s,a)^2 + 2\lambda r(s,a)y\big) - \lambda y^2 \tag{4.11}$$

The dual variable y and policy π are updated iteratively. In each inner loop k, y has analytical solution $y_{k+1} = \sum_{s,a} d_{\pi_k}(s,a)r(s,a) = (1-\gamma)\mathbb{E}_{\pi_k}[G_0]$ since it is quadratic for y. After y is updated, learning π is a risk-neutral problem in the original MDP, but with a new modified reward

$$\hat{r}(s,a) = r(s,a) - \lambda r(s,a)^2 + 2\lambda r(s,a)y_{k+1} \tag{4.12}$$

Since $\hat{r}(s,a)$ is now stationary, any risk-neutral RL algorithms can be applied for policy updating.

Limitations of Per-step Reward Variance. 1) $\mathbb{V}[R]$ is not an appropriate surrogate for $\mathbb{V}[G_0]$ due to fundamentally different implications. Consider a simple example. Suppose the policy, the transition dynamics and the rewards are all deterministic, then $\mathbb{V}[G_0] = 0$ while $\mathbb{V}[R]$ is usually nonzero unless all the per-step rewards are equal. In this case, shifting a specific step reward by a constant will not affect $\mathbb{V}[G_0]$ and should not alter the optimal risk-averse policy. However, such shift can lead to a big difference for $\mathbb{V}[R]$ and may result in an invalid policy as we demonstrated in later example. **2) Reward modification hinders policy learning.** Since the reward modifications in Bisi et al. (2020) (Equation 4.9) and Zhang et al. (2021) (Equation 4.12) share the same issue, here we take Equation 4.12 as an example. This modification is likely to convert a positive reward to a much smaller or even negative value due to the square term, i.e. $-\lambda r(s,a)^2$. In addition, at the beginning of the learning phase, when the policy performance is not good, y is likely to be negative in some environments (since y relates to $\mathbb{E}[G_0]$). Thus, the third term $2\lambda r(s,a)y$ decreases the reward value even more. This prevents the agent to visit the good (i.e., rewarding) state even if that state does not contribute any risk. These two limitations raise a great challenge to subtly choose the value for λ and design the reward for the environment.

Empirical demonstration of the limitations. Consider a maze problem (a modified version of Guarded Maze (Greenberg et al., 2022)) in Figure 4.1. Starting from the bottom left corner, the agent aims to reach the green goal state. The gray color corresponds to walls. The rewards for all states are deterministic (i.e., -1) except for the red state whose reward is a categorical distribution with mean -1. The reward for visiting the goal is a positive constant value. To reach the goal, a risk-neutral agent prefers the path at the bottom that goes through the red state, but $\mathbb{V}[G_0]$ will be nonzero. A risk-averse agent prefers the white path in the figure even though $\mathbb{E}[G_0]$ is slightly lower, but $\mathbb{V}[G_0] = 0$. Per-step reward variance methods aim to use $\mathbb{V}[R]$ as a proxy of $\mathbb{V}[G_0]$. For the risk-averse policy leading to the white path, ideally, increasing the goal reward by a constant will not effect $\mathbb{V}[G_0]$, but will make a big difference to $\mathbb{V}[R]$. For instance, when the goal reward is 10, $\mathbb{V}[R] = 10$. When goal reward is 20, $\mathbb{V}[R] \approx 36.4$, which is much more risk-averse. Next, consider the reward modification (Equation 4.12) for the goal reward when it is 20. The square term in Equation 4.12 is -400λ. It is very easy to make the goal reward negative even for small λ, e.g., 0.1. We do find this reward modification prevents the agent from reaching the goal in our experiments.

4.3 Gini Deviation as an Alternative of Variance

To avoid the limitations of $\mathbb{V}[G_0]$ and $\mathbb{V}[R]$ we have discussed, in this chapter, we propose to use Gini deviation as an alternative of variance. Also, since GD has a similar definition and similar properties as variance, it serves as a more reasonable proxy of $\mathbb{V}[G_0]$ compared to $\mathbb{V}[R]$.

4.3.1 Gini Deviation: Definition and Properties

GD (Gini, 1912), also known as Gini mean difference or mean absolute difference, is defined as follows. For a random variable X, let X_1 and X_2 be two i.i.d. copies of X, i.e., X_1 and

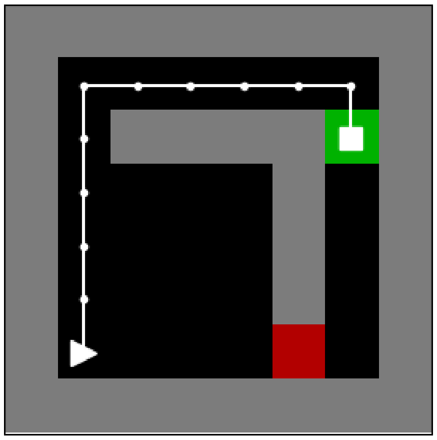


Figure 4.1: A modified Guarded Maze (Greenberg et al., 2022). Red state returns an uncertain reward (details in text).

X_2 are independent and follow the same distribution as X. Then GD is given by

$$\mathbb{D}[X] = \frac{1}{2}\mathbb{E}[|X_1 - X_2|] \tag{4.13}$$

Variance can be defined in a similar way as $\mathbb{V}[X] = \frac{1}{2}\mathbb{E}[(X_1 - X_2)^2]$.

Given samples $\{x_i^1\}_{i=1}^n$ from X_1 and $\{x_j^2\}_{j=1}^n$ from X_2. The unbiased empirical estimations for GD and variance are

$$\begin{aligned} \hat{\mathbb{D}}[X] &= \frac{1}{2n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}|x_i^1 - x_j^2| \\ \hat{\mathbb{V}}[X] &= \frac{1}{2n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i^1 - x_j^2)^2 \end{aligned} \tag{4.14}$$

Both risk profiles aim to measure the variability of a random variable and share similar properties (Yitzhaki et al., 2003). For example, they are both location invariant, and can be presented as a weighted sum of order statistics. Yitzhaki et al. (2003) argues that the GD is superior to the variance as a measure of variability for distributions far from Gaussian. We refer readers to this paper for a full overview. Here we highlight two properties of $\mathbb{D}[X]$ to help interpret it. Let $\mathcal{M}$ denote the set of real random variables and let $\mathcal{M}^p$, $p \in [1, \infty)$ denote the set of random variables whose probability measures have finite p-th moment, then

- $\mathbb{V}[X] \geq \sqrt{3}\,\mathbb{D}[X]$ for all $X \in \mathcal{M}^2$.
- $\mathbb{D}[cX] = c\mathbb{D}[X]$ for all $c > 0$ and $X \in \mathcal{M}$.

The first property is known as Glasser's inequality Glasser (1962), which shows $\mathbb{D}[X]$ is a lower bound of $\mathbb{V}[X]$ if X has finite second moment. The second one is known as positive homogeneity in coherent measures of variability Furman et al. (2017), and is also clear from the definition of GD in Equation 4.13. In RL, considering X is the return variable, this means GD is less sensitive to the reward scale compared to variance, i.e., scaling the return will scale $\mathbb{D}[X]$ linearly, but quadratically for $\mathbb{V}[X]$. We also provide an intuition of the relation between GD and variance from the perspective of convex order, as shown in Appendix B.1. Note also that while variance and GD are both measures of variability, GD is a *coherent* measure of variability (Furman et al., 2017). Sec. 1.2 provides a discussion of the properties of coherent measures of *variability*, while explaining the differences with coherent measures of *risk* such as conditional value at risk (CVaR).

4.3.2 Signed Choquet Integral for Gini Deviation

This section introduces the concept of signed Choquet integral, which provides an alternative definition of GD and makes gradient-based optimization convenient. Note that with the original definition (Equation 4.13), it can be intractable to compute the gradient w.r.t. the parameters of a random variable's density function through its GD.

The Choquet integral (Choquet, 1954) was first used in statistical mechanics and potential theory and was later applied to decision making as a way of measuring the expected utility (Grabisch, 1996). The signed Choquet integral belongs to the Choquet integral family and is defined as:

Definition 1 (Wang et al. (2020), Equation 1) *A signed Choquet integral $\Phi_h : X \to \mathbb{R}, X \in \mathcal{L}^\infty$ is defined as*

$$\Phi_h(X) = \int_{-\infty}^{0} \Big(h\big(\Pr(X \geq x)\big) - h(1)\Big)dx + \int_{0}^{\infty} h\big(\Pr(X \geq x)\big)dx \tag{4.15}$$

where $\mathcal{L}^\infty$ is the set of bounded random variables in a probability space, h is the distortion function and $h \in \mathcal{H}$ such that $\mathcal{H} = \{h : [0,1] \to \mathbb{R}, h(0) = 0, h \text{ is of bounded variation}\}$.

This integral has become the building block of law-invariant risk measures [1] after the work of Kusuoka (2001); Grechuk et al. (2009). One reason for why signed Choquet integral is of interest to the risk research community is that it is not necessarily monotone. Since most practical measures of variability are not monotone, e.g., variance, standard deviation, or deviation measures in Rockafellar et al. (2006), it is possible to represent these measures in terms of Φ_h by choosing a specific distortion function h.

Lemma 1 (Wang et al. (2020), Section 2.6) *Gini deviation is a signed Choquet integral with a concave h given by $h(\alpha) = -\alpha^2 + \alpha, \alpha \in [0, 1]$.*

This Lemma provides an alternative definition for GD, i.e., $\mathbb{D}[X] = \int_{-\infty}^{\infty} h\big(\Pr(X \geq x)\big)dx, h(\alpha) = -\alpha^2 + \alpha$. However, this integral is still not easy to compute. Here we turn to its quantile representation for easy calculation.

Lemma 2 (Wang et al. (2020), Lemma 3) *$\Phi_h(X)$ has a quantile representation. If F_X^{-1} is continuous, then $\Phi_h(X) = \int_1^0 F_X^{-1}(1-\alpha)dh(\alpha)$, where F_X^{-1} is the quantile function (inverse CDF) of X.*

Combining Lemma 1 and 2, $\mathbb{D}[X]$ can be computed alternatively as

$$\mathbb{D}[X] = \Phi_h(X) = \int_0^1 F_X^{-1}(1-\alpha)dh(\alpha) = \int_0^1 F_X^{-1}(\alpha)(2\alpha - 1)d\alpha \tag{4.16}$$

With this quantile representation of GD, we can derive a policy gradient method for our new learning problem in the next section. It should be noted that variance cannot be directly defined by a Φ_h-like quantile representation, but as a complicated related representation: $\mathbb{V}[X] = \sup_{h \in \mathcal{H}} \big\{\Phi_h(X) - \frac{1}{4}\|h'\|_2^2\big\}$, where $\|h'\|_2^2 = \int_0^1 (h'(p))^2 dp$ if h is continuous, and $\|h'\|_2^2 := \infty$ if it is not continuous (see Example 2.2 of Liu et al. (2020)). Hence, such representation of the conventional variance measure is not readily usable for optimization.

4.3.3 Discussion on Variance, Standard Deviation, and Gini Deviation

Variance, standard deviation and Gini deviation all describes the variability of a distribution. Mathematically, they all correspond to the mean of some difference between every

[1] Law-invariant property is one of the popular "financially reasonable" axioms. If a functional returns the same value for two random variables with the same distribution, then the functional is called law-invariant.

pair of points. The main difference is how this difference is computed. Gini deviation uses L1 distance, standard deviation uses L2 distance and variance uses squared L2 distance. As a result, Gini deviation is less sensitive to outliers because L1 distance is less sensitive than L2 distance to outliers. As reveled in Sec. 4.2, using variance (squared L2 distance) causes the stability issue.

To intuitively interpret the limitations of variance and standard deviation, we give some examples. When a random variable follows a Gaussian distribution, variance or standard deviation is the best choice to describe the variability or dispersion of the random variable, since variance is a parameter of the Gaussian distribution. However, when the underlying distribution is far from Gaussian, e.g., Pareto distribution, multimodal distribution, then variance or standard deviation potentially leads to misleading interpretations of the data's variability. For example, variance or standard deviation is sensitive to the extreme values in Pareto distribution due to L2 distance. For multimodal distribution, variance is not sufficient since data are concentrated around several peaks. In the context of RL, the total return distribution is generally not Gaussian and very likely to contain extreme values or being multimodal, thus Gini deviation is preferred over variance and standard deviation.

Here we give an analysis of using standard deviation for policy gradients. By taking the square root of variance, standard deviation gets rid of the square and is positive homogeneous. Readers may wonder if using standard deviation may stabilize the policy gradient. First, similar to Gini deviation, standard deviation can also be defined using signed Choquet integral

$$\mathrm{STD}[X] = \sup_{h\in\mathcal{H}'} \int_0^1 F_X^{-1}(1-\alpha)dh(\alpha) \tag{4.17}$$

where $\mathcal{H}' = \{h \in \mathcal{H}, h(1) = 0, \int_0^1 (h'(t))^2 dt < 1, h \text{ is concave}\}$. It is not convenient to take gradient due to the supreme over a function space. Second, directly taking gradient for standard deviation is still possible by using the gradient of variance (since $\mathrm{STD}[X] = \sqrt{\mathbb{V}[X]}$)

$$\nabla\mathrm{STD}[X] = \frac{1}{2\sqrt{\mathbb{V}[X]}}\nabla\mathbb{V}[X] \tag{4.18}$$

Using the notation in Sec. 4.2.1, the gradient of $\mathrm{STD}[G_0]$ is (via Monte Carlo sampling)

$$\mathbb{E}_\tau[\frac{R_\tau^2\omega(\theta)}{2\sqrt{\mathbb{V}[R_\tau]}}] - \mathbb{E}_\tau[\frac{R_\tau}{\sqrt{\mathbb{V}[R_\tau]}}] \cdot \mathbb{E}_\tau[R_\tau\omega(\theta)] \tag{4.19}$$

where $\omega(\theta)$ includes the gradient of sum of $\log\pi$. Note that this gradient can be potentially unbounded since $\sqrt{\mathbb{V}[R_\tau]}$ can be very small or be zero. For example, in the maze domain in Sec. 4.5.1, the return of the risk-averse path has zero variance, then the gradient of standard deviation causes a division by zero error in this domain.

We give a comparison of Gini deviation and standard deviation in the lunarlander domain in Fig. 4.7.

4.4 Policy Gradient for Mean-Gini Deviation

In this section, we consider a new learning problem by replacing the variance with GD. Specifically, we consider the following objective

$$\max_{\pi} \mathbb{E}[G_0] - \lambda \mathbb{D}[G_0] \tag{4.20}$$

where λ is the trade-off parameter. To maximize this objective, we may update the policy towards the gradient ascent direction. Computing the gradient for the first term has been widely studied in risk-neutral RL Sutton & Barto (2018). Computing the gradient for the second term may be difficult at the first glance from its original definition, however, it becomes possible via its quantile representation (Equation 4.16).

4.4.1 Gini Deviation Gradient Formula

We first give a general gradient calculation in Proposition 3 for GD of a random variable Z, whose distribution function is parameterized by θ. This is the main contribution of this chapter. In RL, we can interpret θ as the policy parameters, and Z as the return under that policy, i.e., G_0. Denote the Probability Density Function (PDF) of Z as $f_Z(z;\theta)$. Given a confidence level $\alpha \in (0,1)$, the α-level quantile of Z is denoted as $q_\alpha(Z;\theta)$, and given by

$$q_\alpha(Z;\theta) = F^{-1}_{Z_\theta}(\alpha) = \inf\{z : \Pr(Z_\theta \leq z) \geq \alpha\} \tag{4.21}$$

For technical convenience, we make the following assumptions, which are also realistic in RL.

Assumption 1 *Z is a continuous random variable, and bounded in range $[-b, b]$ for all θ.*

Assumption 2 *$\frac{\partial}{\partial \theta_i} q_\alpha(Z;\theta)$ exists and is bounded for all θ, where θ_i is the i-th element of θ.*

Assumption 3 *$\frac{\partial f_Z(z;\theta)}{\partial \theta_i} / f_Z(z;\theta)$ exists and is bounded for all θ, z. θ_i is the i-th element of θ.*

Since Z is continuous, the second assumption is satisfied whenever $\frac{\partial}{\partial \theta_i} f_Z(z;\theta)$ is bounded. These assumptions are common in likelihood-ratio methods, e.g., see Tamar et al. (2015). Relaxing these assumptions is possible but would complicate the presentation.

Proposition 3 *Let Assumptions 1, 2, 3 hold. Then*

$$\nabla_\theta \mathbb{D}[Z_\theta] = -\mathbb{E}_{z\sim Z_\theta}\left[\nabla_\theta \log f_Z(z;\theta) \int_z^b \big(2F_{Z_\theta}(t) - 1\big) dt\right] \tag{4.22}$$

Proof. By Equation 4.16, the gradient of $\mathbb{D}[Z_\theta] = \Phi_h(Z_\theta)$ $(h(\alpha) = -\alpha^2 + \alpha, \alpha \in [0,1])$ is

$$\nabla_\theta \mathbb{D}[Z_\theta] = \nabla_\theta \Phi_h(Z_\theta) = \int_0^1 (2\alpha - 1)\nabla_\theta F_{Z_\theta}^{-1}(\alpha) d\alpha = \int_0^1 (2\alpha - 1)\nabla_\theta q_\alpha(Z;\theta) d\alpha. \tag{4.23}$$

This requires to calculate the gradient for any α-level quantile of Z_θ, i.e., $\nabla_\theta q_\alpha(Z;\theta)$. Based on the assumptions and the definition of the α-level quantile, we have $\int_{-b}^{q_\alpha(Z;\theta)} f_Z(z;\theta)dz = \alpha$. Taking a derivative and using the Leibniz rule we obtain

$$0 = \nabla_\theta \int_{-b}^{q_\alpha(Z;\theta)} f_Z(z;\theta)dz = \int_{-b}^{q_\alpha(Z;\theta)} \nabla_\theta f_Z(z;\theta)dz + \nabla_\theta q_\alpha(Z;\theta) f_Z\big(q_\alpha(Z;\theta);\theta\big) \tag{4.24}$$

Rearranging the term, we get

$$\nabla_\theta q_\alpha(Z;\theta) = -\int_{-b}^{q_\alpha(Z;\theta)} \nabla_\theta f_Z(z;\theta)dz \cdot \left[f_Z\big(q_\alpha(Z;\theta);\theta\big)\right]^{-1} \tag{4.25}$$

Plugging back to Equation 4.23 gives us an intermediate version of $\nabla_\theta \mathbb{D}[Z_\theta]$.

$$\nabla_\theta \mathbb{D}[Z_\theta] = -\int_0^1 (2\alpha - 1) \int_{-b}^{q_\alpha(Z;\theta)} \nabla_\theta f_Z(z;\theta)dz \cdot \left[f_Z\big(q_\alpha(Z;\theta);\theta\big)\right]^{-1} d\alpha \tag{4.26}$$

To make the integral over α clearer, we rewrite $q_\alpha(Z;\theta)$ as $F_{Z_\theta}^{-1}(\alpha)$, where F_{Z_θ} is the CDF.

$$\nabla_\theta \mathbb{D}[Z_\theta] = -\int_0^1 (2\alpha - 1) \int_{-b}^{F_{Z_\theta}^{-1}(\alpha)} \nabla_\theta f_Z(z;\theta)dz \frac{1}{f_Z(F_{Z_\theta}^{-1}(\alpha);\theta)} d\alpha$$

Switching the integral order, we get

$$
\begin{aligned}
\nabla_\theta \mathbb{D}[Z_\theta] &= -\int_{-b}^{b}\int_{F_{Z_\theta}(z)}^{1} (2\alpha-1)\nabla_\theta f_Z(z;\theta)\frac{1}{f_Z(F^{-1}_{Z_\theta}(\alpha);\theta)} d\alpha dz \\
&= -\int_{-b}^{b} \nabla_\theta f_Z(z;\theta)\int_{F_{Z_\theta}(z)}^{1} (2\alpha-1)\frac{1}{f_Z(F^{-1}_{Z_\theta}(\alpha);\theta)} d\alpha dz
\end{aligned}
\tag{4.27}
$$

Denote $t = F^{-1}_{Z_\theta}(\alpha)$, then $\alpha = F_{Z_\theta}(t)$. Here, we further change the inner integral from $d\alpha$ to $dF_{Z_\theta}(t)$, i.e., $d\alpha = dF_{Z_\theta}(t) = f_Z(t;\theta)dt$. The integral range for t is now from $F^{-1}_{Z_\theta}(F_{Z_\theta}(z)) = z$ to $F^{-1}_{Z_\theta}(1) = b$.

$$
\begin{aligned}
\nabla_\theta \mathbb{D}[Z_\theta] &= -\int_{-b}^{b} \nabla_\theta f_Z(z;\theta)\int_{z}^{b} \big(2F_{Z_\theta}(t)-1\big)\frac{1}{f_Z(t;\theta)} dF_{Z_\theta}(t)\ dz \\
&= -\int_{-b}^{b} \nabla_\theta f_Z(z;\theta)\int_{z}^{b} \big(2F_{Z_\theta}(t)-1\big) dt\ dz
\end{aligned}
\tag{4.28}
$$

Applying $\nabla_\theta \log(x) = \frac{1}{x}\nabla_\theta x$ to $\nabla_\theta f_Z(z;\theta)$, we have

$$
\begin{aligned}
\nabla_\theta \mathbb{D}[Z_\theta] &= -\int_{-b}^{b} f_Z(z;\theta)\nabla_\theta \log f_Z(z;\theta)\int_{z}^{b} \big(2F_{Z_\theta}(t)-1\big) dt\ dz \\
&= -\mathbb{E}_{z\sim Z_\theta}\Big[\nabla_\theta \log f_Z(z;\theta)\int_{z}^{b} \big(2F_{Z_\theta}(t)-1\big) dt\Big]
\end{aligned}
\tag{4.29}
$$

4.4.2 Gini Deviation Policy Gradient via Sampling

In a typical application, Z in Section 4.4.1 would correspond to the performance of a system, e.g., the total return G_0 in RL. Note that in order to compute Equation 4.22, one needs access to $\nabla_\theta \log f_Z(z;\theta)$: the sensitivity of the system performance to the parameters θ. Usually, the system performance is a complicated function and calculating its probability distribution is intractable. However, in RL, the performance is a function of trajectories. The sensitivity of the trajectory distribution is often easy to compute. This naturally suggests a sampling based algorithm for gradient estimation.

Now consider Equation 4.22 in the context of RL, i.e., $Z = G_0$ and θ is the policy parameter.

$$
\nabla_\theta \mathbb{D}[G_0] = -\mathbb{E}_{g\sim G_0}\big[\nabla_\theta \log f_{G_0}(g;\theta)\int_{g}^{b} \big(2F_{G_0}(t)-1\big) dt\big] \tag{4.30}
$$

To sample from the total return variable G_0, we need to sample a trajectory τ from the environment by executing π_θ and then compute its corresponding return $R_\tau := r_1 + \gamma r_2 + ... + \gamma^{T-1} r_T$, where r_t is the per-step reward at time t, and T is the trajectory length. The probability of the sampled return can be calculated as

$$f_{G_0}(R_\tau; \theta) = \mu_0(s_0) \prod_{t=0}^{T-1} [\pi_\theta(a_t|s_t) p(r_{t+1}|s_t, a_t)] \tag{4.31}$$

The gradient of its log-likelihood is the same as that of

$$P(\tau|\theta) = \mu_0(s_0) \prod_{t=0}^{T-1} [\pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)] \tag{4.32}$$

since the difference in transition probability does not alter the policy gradient. It is well known that $\nabla_\theta \log P(\tau|\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)$.

For the integral part of Equation 4.30, it requires the knowledge of the CDF of G_0. In practice, this means we should obtain the full value distribution of G_0, which is usually not easy. One common approach to acquire an empirical CDF or quantile function (inverse CDF) is to get the quantile samples of a distribution and then apply some reparameterization mechanism. For instance, reparameterization is widely used in distributional RL for quantile function estimation. The quantile function has been parameterized as a step function (Dabney et al., 2018b,a), a piece-wise linear function (Zhou et al., 2021), or other higher order spline functions (Luo et al., 2022). In this chapter, we use the step function parameterization given its simplicity. To do so, suppose we have n trajectory samples $\{\tau_i\}_{i=1}^n$ from the environment and their corresponding returns $\{R_{\tau_i}\}_{i=1}^n$, the returns are sorted in ascending order such that $R_{\tau_1} \leq R_{\tau_2} \leq ... \leq R_{\tau_n}$, then each R_{τ_i} is regarded as a quantile value of G_0 corresponding to the quantile level $\zeta_i = \frac{1}{2}(\frac{i-1}{n} + \frac{i}{n})$, i.e., we assume $q_{\zeta_i}(G_0; \theta) = R_{\tau_i}$. This strategy is also common in distributional RL, e.g., see Dabney et al. (2018b); Yue et al. (2020). The largest return R_{τ_n} is regarded as the upper bound b in Equation 4.30.

Thus, given ordered trajectory samples $\{\tau_i\}_{i=1}^n$, an empirical estimation for GD policy gradient is

$$-\frac{1}{n-1} \sum_{i=1}^{n-1} \eta_i \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}), \text{ where } \eta_i = \sum_{j=i}^{n-1} \frac{2j}{n} \big(R_{\tau_{j+1}} - R_{\tau_j}\big) - \big(R_{\tau_n} - R_{\tau_i}\big) \tag{4.33}$$

We give an example here to show how to estimate the integral of CDF in Eq. 4.30. The CDF function F_{G_0} is parameterized by a step function given its quantiles $\{R_{\tau_i}\}_{i=1}^n$, which

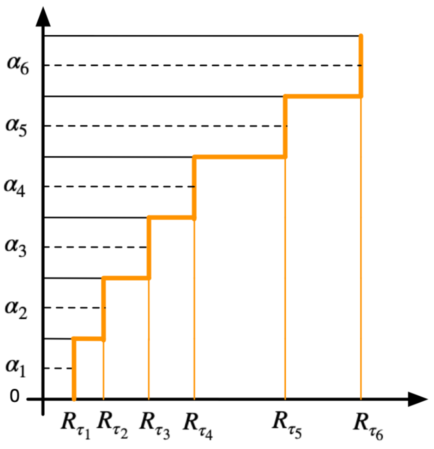


Figure 4.2: An example of parameterizing (inverse) CDF given six quantiles. The function is highlighted in the bold line of orange color.

satisfy $R_{\tau_1} \leq R_{\tau_2} \leq ... \leq R_{\tau_n}$. An example of the step function is shown in Figure 4.2. With this parameterization, the integral over CDF can be regarded as the area below the step function. Thus, for each τ_i, the integral over CDF is approximated as (R_{τ_n} is treated as b)

$$\int_{R_{\tau_i}}^{R_{\tau_n}} 2F_{G_0}(t)dt \approx \sum_{j=i}^{n-1} 2 \times \frac{j}{n}\left(R_{\tau_{j+1}} - R_{\tau_j}\right) \tag{4.34}$$

which yields the estimation in Eq. 4.33.

The sampled trajectories can be used to estimate the gradient for $\mathbb{E}[G_0]$ in the meantime, e.g., the well known vanilla policy gradient (VPG), which is more often used as REINFORCE with baseline as in Eq. 2.18. Apart from VPG, another choice to maximize $\mathbb{E}[G_0]$ is using PPO (Schulman et al., 2017).

4.4.3 Incorporating Importance Sampling

For on-policy policy gradient, samples are abandoned once the policy is updated, which is expensive for our gradient calculation since we are required to sample n trajectories each time. To improve the sample efficiency to a certain degree, we incorporate importance sampling (IS) to reuse samples for multiple updates in each loop. For each τ_i, the IS ratio is $\rho_i = \prod_{t=0}^{T-1} \pi_\theta(a_{i,t}|s_{i,t})/\pi_{\hat{\theta}}(a_{i,t}|s_{i,t})$, where $\hat{\theta}$ is the old policy parameter when $\{\tau_i\}_{i=1}^n$

are sampled. Suppose the policy gradient for maximizing $\mathbb{E}[G_0]$ is REINFORCE baseline. With IS, the empirical mean-GD policy gradient is

$$\frac{1}{n}\sum_{i=1}^{n}\rho_i\sum_{t=0}^{T-1}\nabla_\theta \log\pi_\theta(a_{i,t}|s_{i,t})(g_{i,t}-V(s_{i,t}))+\frac{\lambda}{n-1}\sum_{i=1}^{n-1}\rho_i\eta_i\sum_{t=0}^{T-1}\nabla_\theta\log\pi_\theta(a_{i,t}|s_{i,t}) \quad (4.35)$$

where $g_{i,t}$ is the sum of rewards-to-go as defined above. $V(s_{i,t})$ is the value function. The first part can also be replaced by PPO-Clip policy gradient. Then we have

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{t=0}^{T-1}\nabla_\theta \min\big(\frac{\pi_\theta(a_{i,t}|s_{i,t})}{\pi_{\hat{\theta}}(a_{i,t}|s_{i,t})}A_{i,t}, f(\epsilon, A_{i,t})\big)+\frac{\lambda}{n-1}\sum_{i=1}^{n-1}\rho_i\eta_i\sum_{t=0}^{T-1}\nabla_\theta\log\pi_\theta(a_{i,t}|s_{i,t}) \quad (4.36)$$

where $A_{i,t}$ is the advantage estimate, and $f()$ is the clip function in PPO with ϵ being the clip range, i.e. $f(\epsilon, A_{i,t}) = \text{clip}(\frac{\pi_\theta(a_{i,t}|s_{i,t})}{\pi_{\hat{\theta}}(a_{i,t}|s_{i,t})}, 1-\epsilon, 1+\epsilon)A_{i,t}$.

The extreme IS values ρ_i will introduce high variance to the policy gradient. To stabilize learning, one strategy is that in each training loop, we only select τ_i whose ρ_i lies in $[1-\delta, 1+\delta]$, where δ controls the range. The updating is terminated if the chosen sample size is lower than some threshold, e.g., $\beta\cdot n, \beta\in(0,1)$. Another strategy is to directly clip ρ_i by a constant value ζ, i.e., $\rho_i=\min(\rho_i,\zeta)$, e.g., see Bottou et al. (2013). In our experiments, we use the first strategy for Equation 4.35, and the second for Equation 4.36. We leave other techniques for variance reduction of IS for future study. The full algorithm that combines GD with REINFORCE and PPO is summaried in Algo 1 and 2.

4.5 Experiments

Our experiments were designed to serve two main purposes. First, we investigate whether the GD policy gradient approach could successfully discover risk-averse policies in scenarios where variance-based methods tend to fail. To accomplish this, we manipulated reward choices to assess the ability of the GD policy gradient to navigate risk-averse behavior. Second, we sought to verify the effectiveness of our algorithm in identifying risk-averse policies that have practical significance in both discrete and continuous domains. We aimed to demonstrate its ability to generate meaningful risk-averse policies that are applicable and valuable in practical settings.

Baselines. We compare our method with the original mean-variance policy gradient (Equation 4.2, denoted as MVO), Tamar's method (Tamar et al., 2012) (denoted as Tamar), MVP (Xie et al., 2018), and MVPI (Zhang et al., 2021). Specifically, MVO requires multiple trajectories to compute $J(\theta)\nabla_\theta J(\theta)$. We use $\frac{n}{2}$ trajectories to estimate $J(\theta)$ and another $\frac{n}{2}$ to estimate $\nabla_\theta J(\theta)$, where n is the sample size. MVPI is a general framework for policy iteration whose inner risk-neutral RL solver is not specified. For the environment with discrete actions, we build MVPI on top of Q-Learning or DQN (Mnih et al., 2015). For continuous action environments, MVPI is built on top of TD3 (Fujimoto et al., 2018) as in Zhang et al. (2021). We use REINFORCE to represent the REINFORCE with baseline method. We use MG as a shorthand of mean-GD to represent our method. In each domain, we ensure each method's policy or value nets have the same neural network architecture.

For policy updating, MVO and MG collect n episodes before updating the policy. In contrast, Tamar and MVP update the policy after each episode. Non-tabular MVPI updates the policy at each environment step. In hyperparameter search, we use the parameter search range in MVPI (Zhang et al., 2021) as a reference, making reasonable refinements to find an optimal parameter setting. Please refer to Appendix B.2 for any missing implementation details.

4.5.1 Tabular case: Maze Problem

This domain is a modified Guarded Maze (Greenberg et al., 2022) that was previously described in Section 4.2.2. The original Guarded Maze is asymmetric with two openings to reach the top path (in contrast to a single opening for the bottom path). In addition, paths via the top tend to be longer than paths via the bottom. We modified the maze to be more symmetric in order to reduce preferences arising from certain exploration strategies that might be biased towards shorter paths or greater openings, which may confound risk aversion. Every movement before reaching the goal receives a reward of -1 except moving to the red state, where the reward is sampled from $\{-15, -1, 13\}$ with probability $\{0.4, 0.2, 0.4\}$ (mean is -1) respectively. The maximum episode length is 100. MVO and MG collect $n = 50$ episodes before updating the policy. Agents are tested for 10 episodes per evaluation.

The failure of variance-based baselines under simple reward manipulation. We first set the goal reward to 20. Here, we report the optimal risk-aversion rate achieved during training. Specifically, we measure the percentage of episodes that obtained the optimal risk-averse path, represented by the white color path in Figure 4.1, out of all completed episodes up to the current stage of training.

Notice that MVO performs well in this domain when using double sampling to estimate its gradient. Then we increase the goal reward to 40. This manipulation does not affect the

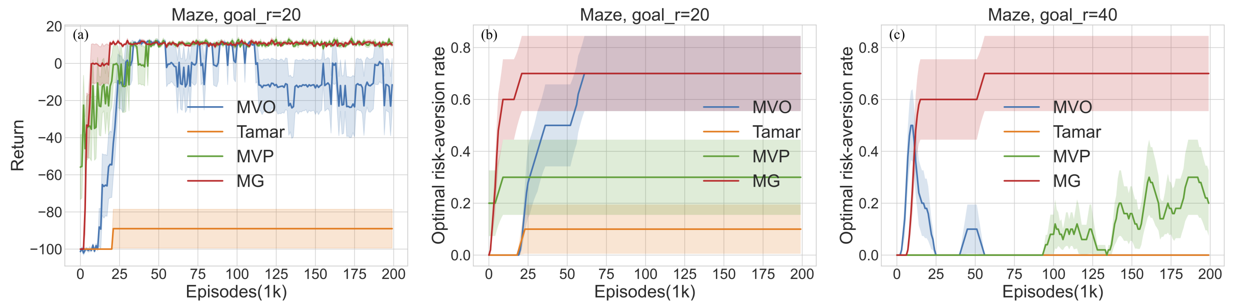


Figure 4.3: (a) Policy evaluation return and (b,c) optimal risk-aversion rate v.s. training episodes in Maze. Curves are averaged over 10 seeds with shaded regions indicating standard errors. For optimal risk-aversion rate, higher is better.

return variance of the optimal risk-averse policy, since the reward is deterministic. However, the performances of MVO, Tamar, MVP all decrease, since they are more sensitive to the numerical scale of the return (due to the $\mathbb{E}[G_0^2]$ term introduced by variance). MVPI is a policy iteration method in this problem, whose learning curve is not intuitive to show. It finds the optimal risk-averse path when the goal reward is 20, but it fails when the goal reward is 40. An analysis for MVPI is given in Sec. 4.5.1. We compare the sensitivity of different methods with respect to λ in Fig. 4.4.

Analysis for MVPI in Maze (MVPI-Q-Learning)

MVPI-Q-Learning finds the optimal risk-averse path when goal reward is 20 but fails when goal reward is 40. Since it is not intuitive to report the learning curve for a policy iteration method where its reward is modified in each iteration, we give an analysis here.

The value of dual variable y in Equation 4.12 is $(1-\gamma)\mathbb{E}[G_0]$ given the current policy. Recall that the maximum episode length is 100. At the beginning, when the Q function is randomly initialized (i.e., it is a random policy), $\mathbb{E}[G_0] = \sum_{t=0}^{99} 0.999^t(-1) \approx -95.2$. Thus $y = (1-0.999)\times(-95.2) = -0.0952$, the goal reward after modification is $r_{\text{goal}} = 20 - 0.2 \times 20^2 + 2 \times 0.2 \times 20 \times y \approx -60.7$. For the red state, its original reward is sampled from $\{-15, -1, 13\}$. After the reward modification, it becomes sampling from $\{-59.4, -1.16, -21.2\}$. Thus the expected reward of the red state is now $r_{\text{red}} = 0.4 \times (-59.4) + 0.2 \times (-1.16) + 0.4 \times (-21.2) = -32.472$. Given the maximum episode length is 100, the optimal policy is still the white path in Figure 4.1. (Because the expected return for the white path is $\sum_{t=0}^{9} 0.999^t(-1) + 0.999^{10}(-60.7) \approx -70$. The expected return for a random walk is $\sum_{t=0}^{99} 0.999^t(-1) \approx -95.2$. The expected return for the shortest path

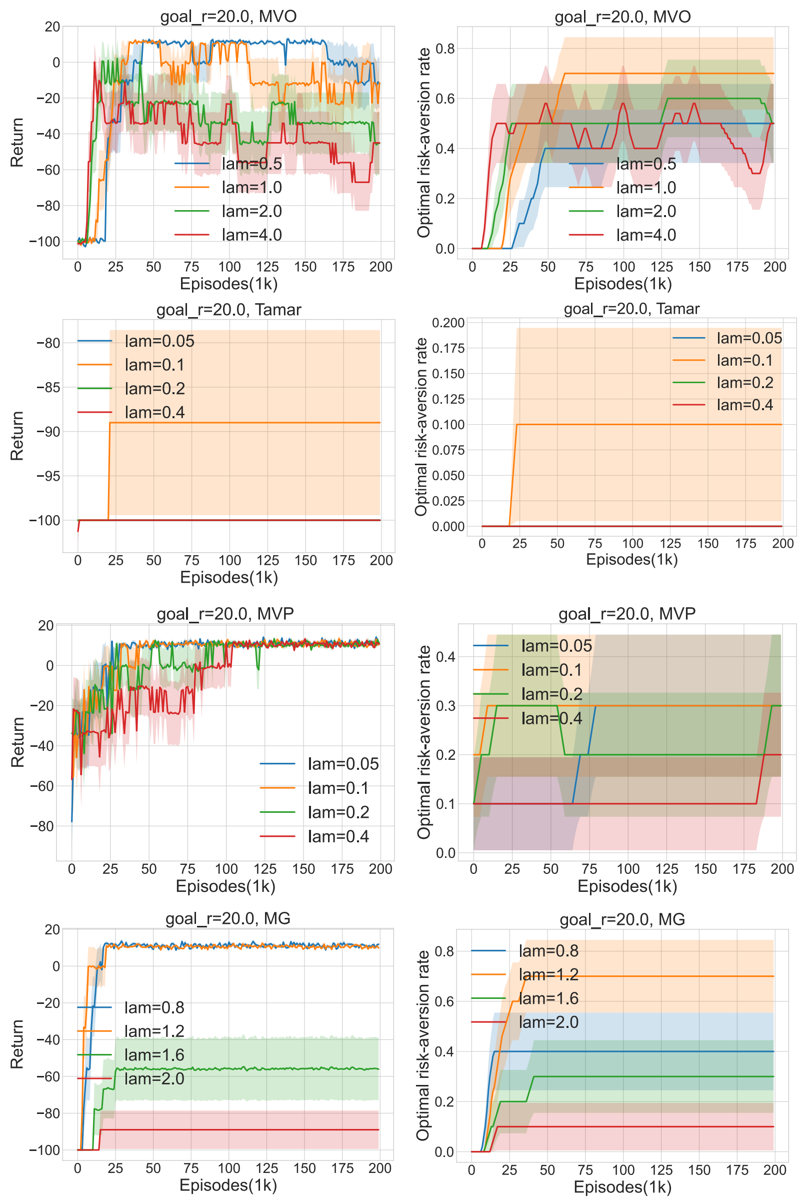


Figure 4.4: Policy evaluation return and optimal risk-aversion rate v.s. training episodes in Maze (goal reward is 20) for MVO, Tamar, MVP, and MG with different λ. The reasonable λ range varies in different methods. Curves are averaged over 10 seeds with shaded regions indicating standard errors.

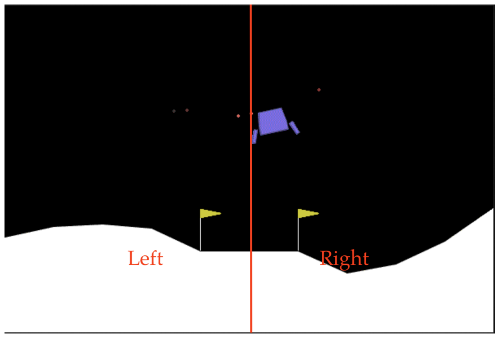


Figure 4.5: Divide the ground of LunarLander into left and right parts by the middle (red) line. If landing in the right area, an additional noisy reward is given.

going through the red state is even lower than the white path since the reward of the red state after modification is pretty negative: -32.472.)

However, when goal reward is 40, after modification, the goal reward becomes $r_{\text{goal}} = 40 - 0.2 \times 40^2 + 2 \times 0.2 \times 40 \times y \approx -281.5$. In this case, the optimal policy has to avoid the goal state since it leads to a even lower return.

Remark. Scaling rewards by a small factor is not an appropriate approach to make algorithms less sensitive to the numerical scale for both total return variance and per-step reward variance, since it changes the original mean-variance objective in both cases.

4.5.2 Discrete control: LunarLander

This domain is taken from OpenAI Gym Box2D environments Brockman et al. (2016a). We refer readers to its official documents for the full description. Originally, the agent is awarded 100 if it comes to rest. We divide the ground into two parts by the middle line of the landing pad, as shown in Figure 4.5 in Appendix. If the agent lands in the right area, an additional noisy reward sampled from $\mathcal{N}(0, 1)$ times 90 is given. A risk-averse agent should learn to land at the left side as much as possible. We include REINFORCE as a baseline to demonstrate the risk-aversion of our algorithm. REINFORCE, MVO and MG collect $n = 30$ episodes before updating their policies. Agents are tested for 10 episodes per evaluation.

We report the rate at which different methods land on the left in Figure 4.6(b) (we omit the failed methods), i.e, the percentage of episodes successfully landing on the left

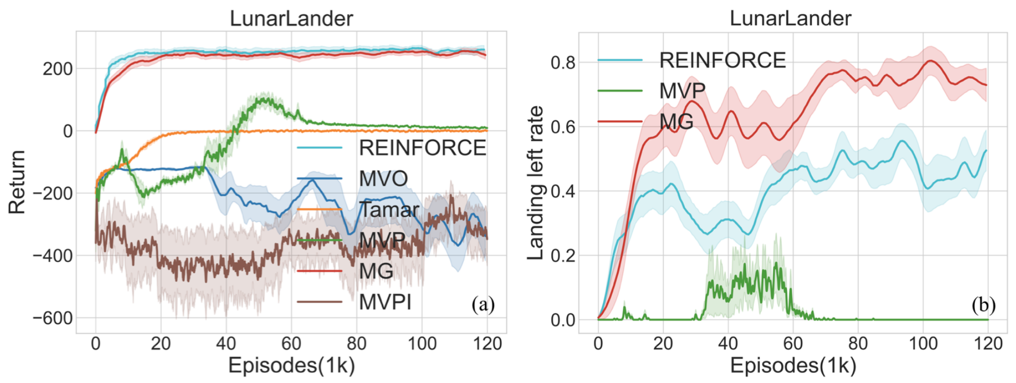


Figure 4.6: (a) Policy evaluation return and (b) left-landing rate (i.e., risk-averse landing rate) v.s. training episodes in LunarLander. Curves are averaged over 10 seeds with shaded regions indicating standard errors. For landing left rate, higher is better.

per evaluation. MVO, Tamar, and MVP do not learn reasonable policies in this domain according to their performances in Figure 4.6(a). MVP learns to land in the middle of the learning phase, but soon after fails to land. Since successfully landing results in a large return (success reward is 100), the return square term ($\mathbb{E}[G_0^2]$) introduced by variance makes MVP unstable. MVPI also fails to land since $\mathbb{V}[R]$ is sensitive to the numerical scale of rewards. In this domain, the success reward is much larger than other reward values. Furthermore, reward modification in MVPI turns large success rewards into negative values, which prevents the agent from landing on the ground. MG achieves a comparable return with REINFORCE, but clearly learns a risk-averse policy by landing more on the left.

The additional results on replacing variance by standard deviation is shown in Fig. 4.7. where the mean-standard deviation objective is labeled by MSTD. Regarding the performance (i.e., evaluation return), MSTD is much better than MVO in Fig. 4.6. However, MSTD still fails to learn a risk averse behavior in this domain.

4.5.3 Continuous control: Mujoco

Mujoco Todorov et al. (2012) is a collection of robotics environments with continuous states and actions in OpenAI Gym Brockman et al. (2016a). Here, we selected three domains (InvertedPendulum, HalfCheetah, and Swimmer) that are conveniently modifiable, where we are free to modify the rewards to construct risky regions in the environment (Through empirical testing, risk-neutral learning failed when similar noise was introduced to other

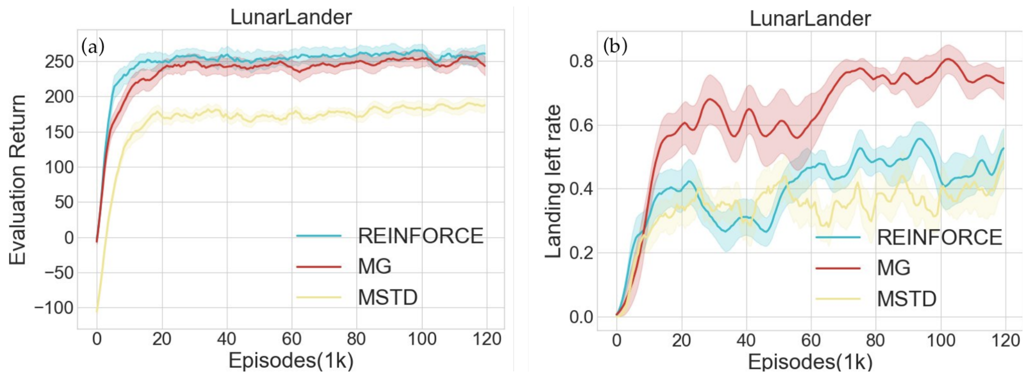


Figure 4.7: (a) Policy evaluation return and (b) left-landing rate (i.e., risk-averse landing rate) v.s. training episodes in LunarLander. MSTD means mean-standard deviation objective. Curves are averaged over 10 seeds with shaded regions indicating standard errors. For landing left rate, higher is better.

Mujoco domains. Consequently, identifying the cause for the failure of risk-averse algorithms on other domains became challenging). Motivated by and following Malik et al. (2021); Liu et al. (2023), we define a risky region based on the X-position. For instance, if X-position > 0.01 in InvertedPendulum, X-position < -3 in HalfCheetah, and X-position > 0.5 in Swimmer, an additional noisy reward sampled from $\mathcal{N}(0, 1)$ times 10 is given. Location information (i.e., X-position) is appended to the agent's observation. A risk-averse agent should reduce the time it visits the noisy region in an episode. To ensure that agents move both forward and backward with equal preference in terms of expected reward in the environments, we define the distance-based reward as the difference in distance between the current and previous states from the origin, regardless of the sign of the X-position. We also include the risk-neutral algorithms as baselines to highlight the risk-aversion degree of different methods.

All the risk-averse policy gradient algorithms still use VPG to maximize the expected return in InvertedPendlulum (thus the risk-neutral baseline is REINFORCE). Using VPG is also how these methods are originally derived. However, VPG is not good at more complex Mujoco domains, e.g., see OpenAI's benchmark [2]. In HalfCheetah and Swimmer, we combine those algorithms with PPO-style policy gradient to maximize the expected return. Minimizing the risk term remains the same as their original forms. MVPI is an off-policy time-difference method in Mujoco. We train it with 1e6 steps instead of as many

[2]https://spinningup.openai.com/en/latest/spinningup/bench.html

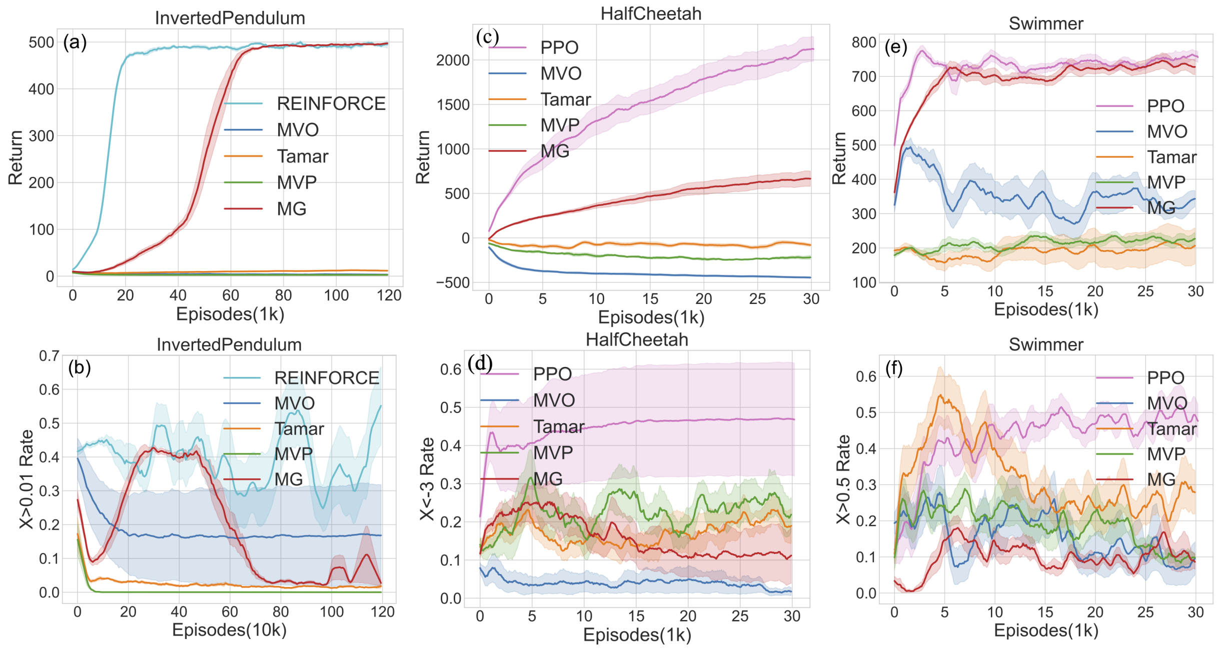


Figure 4.8: (a,c,e) Policy evaluation return and (b,d,f) location visiting rate v.s. training episodes in Mujoco of episode-based methods. Curves are averaged over 10 seeds with shaded regions indicating standard errors. For location visiting rate, lower is better.

episodes as other methods. MVO and MG sample $n = 30$ episodes in InvertedPendulum and $n = 10$ in HalfCheetah and Swimmer before updating policies. Agents are tested for 20 episodes per evaluation. The percentage of time steps visiting the noisy region in an episode is shown in Figure 4.8(b,d,f). Compared with other return variance methods, MG achieves a higher return while maintaining a lower visiting rate. Comparing MVPI and TD3 against episode-based algorithms like MG is not straightforward within the same figure due to the difference in parameter update frequency. MVPI and TD3 update parameters at each environment time step. We shown their learning curves in Figure 4.9. MVPI also learns risk-averse policies in all three domains according to its learning curves.

We further design two domains using HalfCheetah and Swimmer (marked as HalfCheetah1 and Swimmer1 in the figure's caption). The randomness of the noisy reward linearly decreases when agent's forward distance grows. To encourage the agent to move forward, only the forward reward is positive. The additional noisy reward is sampled from $\mathcal{N}(0, 1)$ times 10 times $1 - \frac{X}{20}$ if X-position > 0. To maximize the expected return and minimize risk, the agent has to move forward as far as possible. The results are shown in Figures 4.10,4.11. In these two cases, only MG shows a clear tendency of moving forward,

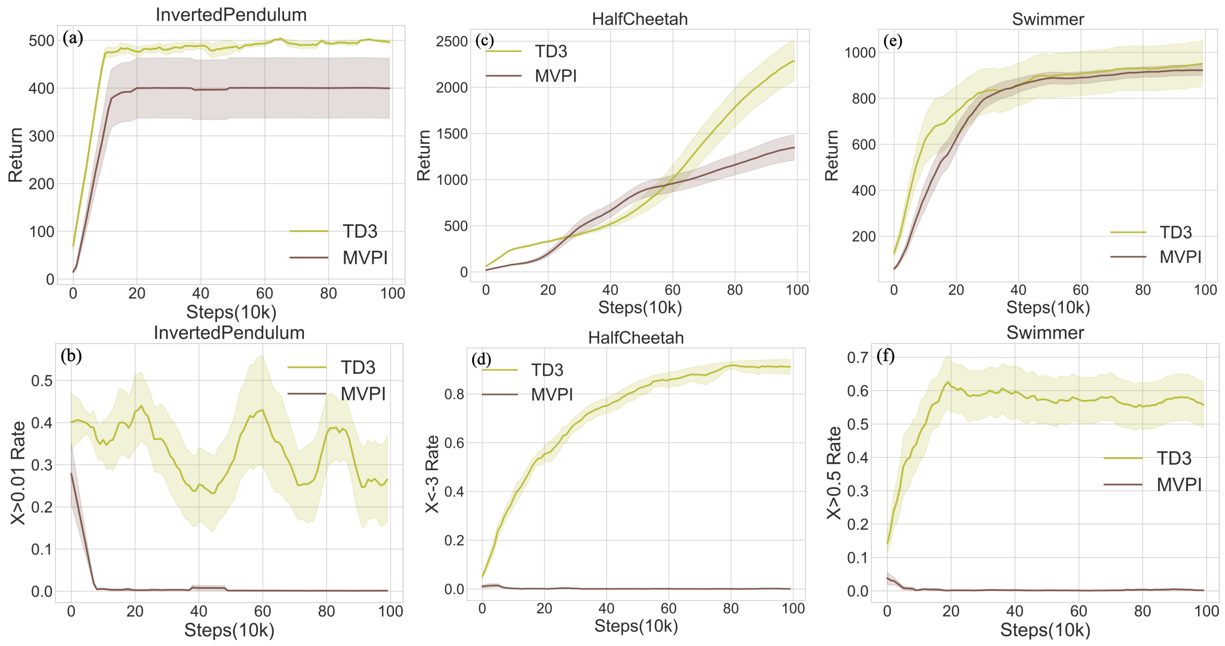


Figure 4.9: (a,c,e) Policy evaluation return and (b,d,f) location visiting rate v.s. training episodes in Mujoco of TD3 and MVPI. Curves are averaged over 10 seeds with shaded regions indicating standard errors. For location visiting rate, lower is better.

which suggests our method is less sensitive to reward choices compared with methods using $\mathbb{V}[R]$.

The return variance and GD during learning in the above environments are also reported in Appendix B.2. In general, when other return variance based methods can find the risk-averse policy, MG maintains a lower or comparable return randomness when measured by both variance and GD. When other methods fail to learn a reasonably good risk-averse policy, MG consistently finds a notably higher return and lower risk policy compared with risk-neutral methods. MVPI has the advantage to achieve low return randomness in location based risky domains, since minimizing $\mathbb{V}[R]$ naturally avoids the agent from visiting the noisy region. But it fails in distance-based risky domains.

4.6 Summary

This chapter proposes to use a new risk measure, Gini deviation, as a substitute for variance in mean-variance RL. It is motivated to overcome the limitations of the existing total

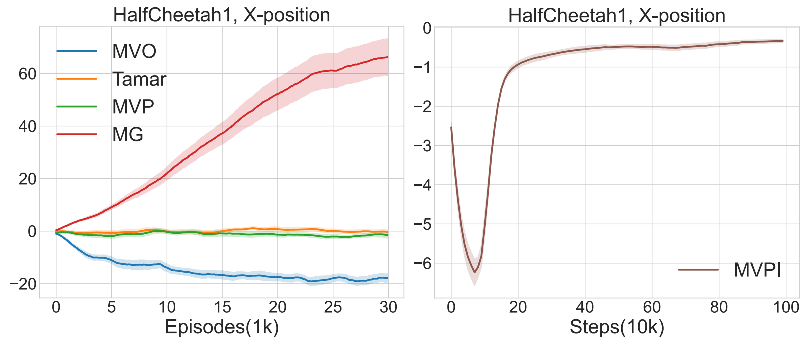


Figure 4.10: The distance agents covered in HalfCheetah1. Curves are averaged over 10 seeds with shaded regions indicating standard errors.

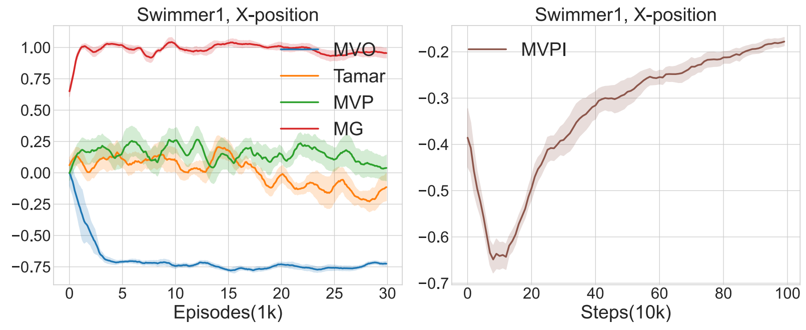


Figure 4.11: The distance agents covered in Swimmer1. Curves are averaged over 10 seeds with shaded regions indicating standard errors.

return variance and per-step reward variance methods, e.g., sensitivity to numerical scale and hindering of policy learning. A gradient formula is presented and a sampling-based policy gradient estimator is proposed to minimize such risk. We empirically show that our method can succeed when the variance-based methods will fail to learn a risk-averse or a reasonable policy. This new risk measure may inspire a new line of research in RARL. First, one may study the practical impact of using GD and variance risk measures. Second, hybrid risk measures may be adopted in real-world applications to leverage the advantages of various risk measures.

Algorithm 1 Mean-Gini Deviation Policy Gradient (with REINFORCE baseline)

```
**Input:** Iterations number $K$, sample size $n$, inner update number $M$, policy learning rate $\alpha_\theta$, value learning rate $\alpha_\phi$, importance sampling range $\delta$, inner termination parameter $\beta$, trade-off parameter $\lambda$.
Initialize policy $\pi_\theta$ parameter $\theta$, value $V_\phi$ parameter $\phi$.
for $k = 1$ to $K$ do
    Sample $n$ trajectories $\{\tau_i\}_{i=1}^n$ by $\pi_\theta$, compute return $\{R(\tau_i)\}_{i=1}^n$
    Compute rewards-to-go for each state in $\tau_i$: $g_{i,t}$
    for $m = 1$ to $M$ do
        Compute importance sampling ratio for each trajectory $\{\rho_i\}_{i=1}^n$
        Select $\mathcal{D} = \{\tau_s\}$ whose $\rho_s \in [1-\delta, 1+\delta]$
        Sort trajectories such that $R(\tau_1) \leq ... \leq R(\tau_{|\mathcal{D}|})$
        if $|\mathcal{D}| < n \cdot \beta$ then
            break
        end if
        mean_grad = 0, gini_grad = 0
        for $i = 1$ to $|\mathcal{D}|$ do
            mean_grad += $\rho_i \cdot \sum_0^{T-1} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})(g_{i,t} - V\phi(s_{i,t}))$
            Update $V_\phi$ by mean-squared error $\frac{1}{T}\sum_{t=0}^{T-1}(V_\phi(s_{i,t}) - g_{i,t})^2$ with learning rate $\alpha_\phi$
        end for
        for $i = 1$ to $|\mathcal{D}| - 1$ do
            gini_grad += $-\rho_i \cdot \eta_i \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})$, where
            $\eta_i = \sum_{j=i}^{|\mathcal{D}|-1} \frac{2j}{|\mathcal{D}|}(R_{j+1} - R_j) - (R_{|\mathcal{D}|} - R_i)$
        end for
        Update $\pi_\theta$ by $\left(\frac{1}{|\mathcal{D}|}\text{ mean\_grad - } \frac{\lambda}{|\mathcal{D}|-1}\text{ gini\_grad}\right)$ with learning rate $\alpha_\theta$ (Equation 4.35)
    end for
end for
```

Algorithm 2 Mean-Gini Deviation Policy Gradient (with PPO)

```
Input: Iterations number $K$, sample size $n$, inner update number $M$, policy learning rate $\alpha_\theta$, value learning rate $\alpha_\phi$, importance sampling clip bound $\zeta$, trade-off parameter $\lambda$.
Initialize policy $\pi_\theta$ parameter $\theta$, value $V_\phi$ parameter $\phi$.
for $k = 1$ to $K$ do
    Sample $n$ trajectories $\{\tau_i\}_{i=1}^n$ by $\pi_\theta$, compute return $\{R(\tau_i)\}_{i=1}^n$
    Compute rewards-to-go for each state in $\tau_i$: $g_{i,t}$
    Compute advantages for each state-action in $\tau_i$: $A(s_{i,t}, a_{i,t})$ based on current $V_\phi$
    for $m = 1$ to $M$ do
        Compute importance sampling ratio for each trajectory $\{\rho_i\}_{i=1}^n$, and $\rho_i = \min(\rho_i, b)$

        Sort trajectories such that $R(\tau_1) \leq ... \leq R(\tau_n)$
        mean_grad = 0, gini_grad = 0
        for $i = 1$ to $n$ do
            mean_grad += PPO-Clip actor grad
            Update $V_\phi$ by mean-squared error $\frac{1}{T}\sum_{t=0}^{T-1}(V_\phi(s_{i,t}) - g_{i,t})^2$ with learning rate $\alpha_\phi$
        end for
        for $i = 1$ to $n - 1$ do
            gini_grad += $-\rho_i \cdot \eta_i \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})$, where
            $\eta_i = \sum_{j=i}^{n-1} \frac{2j}{n}(R_{j+1} - R_j) - (R_n - R_i)$
        end for
        Update $\pi_\theta$ by $\left(\frac{1}{nT}\right.$ mean_grad - $\frac{\lambda}{(n-1)T}$ gini_grad$\left.\right)$ with learning rate $\alpha_\theta$ (Equation 4.36)
    end for
end for
```

Chapter 5

Optimizing CVaR in RL: A Simple Mixture Policy Parameterization for Improving Sample Efficiency

5.1 Introduction

We discussed how to optimize a measure of variability in Chapter 4. In this chapter, we consider another type of risk measure called tail risk measures (Liu & Wang, 2021). Intuitively, tail risk measures only consider the tail of a distribution, and the well known tail risk measures include Value at Risk (VaR) (Chow et al., 2018; Jung et al., 2022), and Conditional VaR (CVaR) (Tamar et al., 2015; Lim & Malik, 2022). CVaR is more often preferred than VaR because it is coherent (Delbaen & Biagini, 2000) and it considers the expectation of the tail. Thus in this chapter, we focus on optimizing CVaR in RL.

In the context of RL, CVaR emphasizes the worst case outcome of a policy's return. Intuitively, CVaR measures the expected return below a specific quantile level α, termed the risk level. Among the existing CVaR algorithms in RL (Tamar et al., 2015; Chow et al., 2018; Tang et al., 2019; Yang et al., 2021; Ying et al., 2022), policy gradient (PG) is a common choice. CVaR-PG samples a batch of N trajectories and maximizes the mean return of the αN trajectories with worst returns (Tamar et al., 2015). This approach suffers from sample inefficiency due to two major facts (Greenberg et al., 2022): 1) $1-\alpha$ portion of sampled trajectories are discarded; 2) gradients vanish when the tail of the return quantile function is overly flat, which is discussed later in Sec. 5.3.1. Another line of research on optimizing CVaR is based on distributional RL (Bellemare et al., 2017), e.g., Dabney et al.

(2018a); Tang et al. (2019); Keramati et al. (2020). However, due to the time-inconsistency of the risk, the objectives of some approaches differ from maximizing the α-CVaR of the total return, while the behavior of some others are not well-understood yet (Lim & Malik, 2022).

In this chapter, we focus on the policy gradient approach and propose a simple mixture policy parameterization to improve sample efficiency. Our key insight is that in many real-world risk-sensitive domains, the agent may only need to perform risk-averse actions in a subset of states, e.g., related to risky regions, and behave akin to a risk-neutral agent in other states. We give an example in Sec. 5.3.3. This motivates representing a risk-averse policy via integrating a risk-neutral policy and an adjustable component. With this parameterization, all collected trajectories can be used to update the policy under the mixture framework, and gradient vanishing is counteracted by stimulating higher returns with the help of its risk-neutral component, thus lifting the tail and preventing flatness of the quantile function. To demonstrate the effectiveness of our method in learning risk-averse policies, we modify several domains (Maze (Greenberg et al., 2022), Lunar Lander(Brockman et al., 2016a), Mujoco (Todorov et al., 2012)) where risk-aversion can be clearly verified. We empirically show that our method can learn a risk-averse policy when others fail to learn a reasonable policy.

To the best of our knowledge, a generally applicable approach to improve the sample efficiency of CVaR-PG algorithms remains unclear. The most recent work to improve sample effiency of CVaR policy gradient is by Greenberg et al. (2022). Their idea is to modify the environment dynamics such that the sample trajectories correspond to the tail return trajectories in the original unmodified environment. In this case, more trajectories can be used for policy update and thus sample efficiency is improved. However, this method cannot be applied easily to every domain since modifying the environment dynamics is usually domain specific. We compare with this method in one domain from their paper in Sec. 5.4.4. In summary, our work in this chapter provides 1) insights into a novel perspective in scenarios where risk-averse behaviors are required only in a subset of states; 2) a simple mixture policy parameterization to address sample inefficiency. Notably, our algorithm, in certain Mujoco domains, advances the state-of-the-art in CVaR optimization.

5.2 Background: CVaR Optimization in RL

5.2.1 Problem Formulation

Let Z be a bounded random variable with cumulative distribution function $F_Z(z) = \mathcal{P}(Z \leq z)$. Denote the α-quantile as $q_\alpha(Z) = \min\{z|F_Z(z) \geq \alpha\}, \alpha \in (0, 1]$. The CVaR at confidence level α is given by (Rockafellar et al., 2000)

$$\text{CVaR}_\alpha(Z) = \frac{1}{\alpha} \int_0^\alpha q_\beta(Z) d\beta \tag{5.1}$$

When $\alpha \to 1$, $\text{CVaR}_\alpha(Z)$ becomes $\mathbb{E}[Z]$. If Z has a continuous distribution, $\text{CVaR}_\alpha(Z)$ is more intuitively expressed as $\text{CVaR}_\alpha(Z) = \mathbb{E}[Z|Z \leq q_\alpha(Z)]$. Thus, $\text{CVaR}_\alpha(Z)$ can be interpreted as the expected value of the α-portion of the left tail of the distribution of Z. Another way to define $\text{CVaR}_\alpha(Z)$ is (Rockafellar et al., 2000)

$$\text{CVaR}_\alpha(Z) = \max_{k \in \mathbb{R}} k - \frac{1}{\alpha} \mathbb{E}[(k - Z)^+] \tag{5.2}$$

where $(x)^+ = \max\{x, 0\}$, and the maximum is always attained at $k = q_\alpha(Z)$ as a by product.

In this chapter, we consider the problem of maximizing the CVaR of total return G_0^π given a confidence level α (we consider small α in practice) (Tamar et al., 2015), i.e.,

$$\max_\pi \text{CVaR}_\alpha(G_0^\pi) \tag{5.3}$$

Remark. Some works optimize the CVaR term plus the mean term or treat CVaR as a constraint (Chow et al., 2018; Yang et al., 2021; Ying et al., 2022), which differ from the problem in Eq. 5.3. In addition, the risk defined on the total return (Eq. 5.3) is known as the static risk. Another line of research on CVaR works on dynamic risk (Ruszczyński, 2010; Huang et al., 2021; Du et al., 2023), where risk is recursively computed at each time step. This chapter focuses on the static risk. The comparison between static and dynamic CVaR is discussed, e.g., in Lim & Malik (2022).

5.2.2 CVaR Policy Gradient (CVaR-PG)

The most straightfoward way to solve Eq. 5.3 is policy gradient. Let π be parameterized by θ. Under some mild assumptions, the gradient of Eq. 5.3 w.r.t. θ can be estimated by sampling trajectories $\{\tau_i\}_{i=1}^N$ from the environment using π_θ (Tamar et al., 2015).

$$\nabla_\theta \text{CVaR}_\alpha(G_0^{\pi_\theta}) \simeq \frac{1}{\alpha N}\sum_{i=1}^{N} \mathbb{I}_{\{R_{\tau_i}\leq \hat{q}_\alpha\}}(R_{\tau_i}-\hat{q}_\alpha)\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \tag{5.4}$$

where R_τ represents the total return of trajectory τ, $\hat{q}_\alpha$ is the empirical α-quantile estimated from $\{R_{\tau_i}\}_{i=1}^N$, and T is the maximum trajectory length. This gradient is derived from Eq. 5.1, and we briefly show how it is derived.

We briefly show how this gradient is derived. Following the same notations and assumptions in Sec. 4.4.1, we consider computing a general case for $\nabla_\theta \text{CVaR}_\alpha(Z)$ where the density function of Z is determined by parameter θ denoted by $f_Z(z;\theta)$. Z is continuous and bounded in $[-b,b]$. $q_\alpha(Z;\theta)$ is the α-level quantile. By definition

$$\text{CVaR}_\alpha(Z_\theta) = \frac{1}{\alpha}\int_{-b}^{q_\alpha(Z;\theta)} f_Z(z;\theta) z dz \tag{5.5}$$

Taking a derivative and using the Leibniz rule

$$\nabla_\theta \text{CVaR}_\alpha(Z_\theta) = \frac{1}{\alpha}\int_{-b}^{q_\alpha(Z;\theta)} \nabla_\theta f_Z(z;\theta) z dz + \frac{1}{\alpha} f_Z\big(q_\alpha(Z;\theta);\theta\big) q_\alpha(Z;\theta) \tag{5.6}$$

Plugging the gradient for quantile in Eq. 4.24 to Eq. 5.6, we obtain

$$\nabla_\theta \text{CVaR}_\alpha(Z_\theta) = \frac{1}{\alpha}\int_{-b}^{q_\alpha(Z;\theta)} \nabla_\theta f_Z(z;\theta)\big(z - q_\alpha(Z;\theta)\big) dz \tag{5.7}$$

Now consider the case in RL, i.e, Z_θ is G_0. Using Monte Carlo sampling, sampling return from G_0 corresponds to sampling trajectories $\{\tau_i\}_{i=1}^N$ from the environment using π_θ and computing the trajectories' return $\{R_{\tau_i}\}_{i=1}^N$. As analyzed in Sec. 4.4.2, $\nabla_\theta f_{G_0}(R_{\tau_i};\theta)$ is estimated by $\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})$ of τ_i. Under the assumption of continuous random, the number of trajectories whose return is smaller than the α quantile is αN. Combining the calculation together yields the estimation in Eq. 5.4.

Note that computing policy gradient from Eq. 5.2 is also feasible and results in a similar update as Eq. 5.4, e.g., see Algo. 1 in Chow et al. (2018). Here the variable k and the density parameter θ are updated separately. When k is fixed, the gradient for updating θ is similar to Eq. 5.4.

5.2.3 Distributional RL with CVaR

Distributional RL (Bellemare et al., 2017) is recently used for CVaR optimization. Since it directly learns a value distribution, the risk metric is easy to compute. Denote the return random variable at the state-action pair (s, a) as $Z^\pi(s, a) = \sum_{t=0}^{\infty} \gamma^t R(S_t, A_t)$, where $S_0 = s$, $A_0 = a$, $S_{t+1} \sim P(\cdot|S_t, A_t)$, and $A_t \sim \pi(\cdot|S_t)$. Then the distributional Bellman equation is given by $Z^\pi(s, a) \overset{D}{=} R + \gamma Z^\pi(S', A')$, with $S' \sim P(\cdot|s, a)$, $A' \sim \pi(\cdot|S')$, and $X \overset{D}{=} Y$ indicates that random variables X and Y follow the same distribution. The well known Q-value can be extracted by $Q^\pi(s, a) = \mathbb{E}[Z^\pi(s, a)]$.

Dabney et al. (2018a); Keramati et al. (2020) propose to select actions according to

$$Z^\pi(s, a) \overset{D}{=} R + \gamma Z^\pi(S', A'), \qquad A' = \arg\max_{a'} \mathrm{CVaR}_\alpha(Z^\pi(S', a')) \tag{5.8}$$

This strategy is simple and intuitive. It always selects actions leading to the largest α-CVaR at the current step and is termed as "Markov action selection strategy" by Lim & Malik (2022). Within the framework of actor critic, a similar way is applied by updating the actor towards the α-CVaR of the critic, e.g., see Tang et al. (2019). However, Lim & Malik (2022) showed this strategy converges to neither static nor dynamic optimal CVaR policies by counterexamples, e.g., see Proposition 1 in Lim & Malik (2022). This is also expected in general since risk measures generally can not be optimized via dynamic programming. Thus, it is not consistent with the problem in Eq. 5.3.

Bäuerle & Ott (2011) simplified Eq. 5.2 by avoiding optimizing k and fixing it to some constant k_0, resulting in the problem $\max_\pi -\mathbb{E}[(k_0 - G_0^\pi)^+]$. This problem can be modeled by an augmented MDP with new state $\tilde{s} = (s, k) \in \mathcal{S} \times \mathbb{R}$, where k is a moving variable keeping track of the accumulated rewards so far. The new state transition is $\hat{P}(s', k'|s, k, a) = P(s'|s, a)\mathbb{I}\{k' = \frac{k - r(s,a)}{\gamma}\}$. There is no reward in the new MDP unless the terminal state, given by $-k^+$. Lim & Malik (2022) incorporated this perspective with distributional RL by introducing the tracking variable, and proposed a new action selection strategy as

$$Z^\pi(s, a) \overset{D}{=} R + \gamma Z^\pi(S', A'), \qquad A' = \arg\max_{a'} \mathbb{E}[-(\frac{k - R}{\gamma} - Z^\pi(S', a'))^+] \tag{5.9}$$

where k is the tracking variable at (s, a), and is set to α-CVaR for the initial state. Lim & Malik (2022) showed that the optimal CVaR policy is a fixed point of Eq. 5.9 if it exists and it is unique. However, when π is not CVaR optimal, its behavior is generally unknown.

5.2.4 Other CVaR RL Algorithms

There are several other CVaR algorithms in the context of MDPs, where full knowledge of the MDP is required. Thus, they are not applicable to RL problems where transition dynamics are unknown. These works are less relevant to ours and hence we only provide a brief review here. Based on the theory of CVaR decomposition (Pflug & Pichler, 2016), a dynamic programming approach is developed by decomposing the CVaR via its risk envelope (Chow et al., 2015). This approach returns the optimal α-CVaR value for any $\alpha \in (0, 1]$. Recently, Hau et al. (2023) pointed out this method has some flaws in the control setting, and provided counter examples.

5.3 Mixture Parameterization Policies

In this section, we examine the difficulties inherent in classical CVaR-PG methods. This examination sets the stage for our proposed solution: a mixture parameterization approach.

5.3.1 Challenges of CVar-PG: low-efficiency gradient estimation

The classical CVaR-PG (Eq. 5.4) faces two significant challenges that undermine its sample efficiency and practical applicability.

Firstly, to emphasize the tail outcomes, a small value of α is chosen. Consequently, only an α-fraction of the trajectories contribute to the gradient estimation in Eq. 5.4, leading to the discarding of the majority of trajectories and resulting in low sample efficiency.

Secondly, as identified by Greenberg et al. (2022), a small α also introduces a gradient vanishing issue. This occurs because the term $\mathbb{I}_{\{R(\tau_i)\leq\hat{q}_\alpha\}}(R(\tau_i)-\hat{q}_\alpha)$ can equal zero for any τ_i satisfying $R(\tau_i) \leq \hat{q}_\alpha$, i.e., $R(\tau_i) = \hat{q}_\alpha$ for those trajectories τ_i selected by the indicator function. This issue arises when the left tail of the quantile function is notably flat, meaning that all quantile values below the α-quantile are identical. Such a scenario is particularly likely in environments with a discrete rewards distribution, a fact that is often overlooked when assuming continuous rewards. For illustration, we present the empirical quantile function of G_0^π obtained through Monte Carlo sampling in Fig. 5.1(c), during the initial training phase with a random policy in a maze environment (detailed in Sec. 5.3.3 and shown in Fig. 5.1(a)). In this scenario, if the agent neither reaches the goal nor enters the red state, the resulting trajectories will yield identical low returns, leading to a markedly flat left tail of the quantile function for G_0^π.

To tackle gradient vanishing, Greenberg et al. (2022) proposed curriculum learning by starting from an α close to 1 (risk-neutral) and gradually decreasing α to its target value. To further improve sample efficiency, Greenberg et al. (2022) proposed a sampling method based on cross-entropy to sample high-risk scenarios from the environment. The algorithm is then focused on learning high-risk parts of the environment and thus improving sample efficiency. However, this sampling strategy requires knowledge of the environment dynamics and the ability to control the parameters of the dynamics in ways that are domain specific, which is not realistic for many RL domains.

5.3.2 Mixture with Risk-neutral Policy

To address the aforementioned challenges, our key observation is that many real-world risk-sensitive applications exhibit a pattern wherein only a subset of states requires risk-averse behavior. In the remaining portion of the state space, the agent can behave akin to a risk-neutral agent. For example, in scenarios with minimal or no other cars on a highway, a driver may simply need to follow the road without slowing down or braking, as long as the vehicle remains under the speed limit. This observation leads us to propose representing the policy as a mixture of a risk-neutral policy and an adjustable component, i.e.,

$$\pi(a|s) = w(s)\pi'(a|s) + (1 - w(s))\pi^n(a|s) \tag{5.10}$$

where $w(s) \in [0, 1]$ is the mixture component weight. π^n is the risk neutral policy, and π' is the adjustable policy. At different phases of a task, the agent self-selects the most suitable policies to execute to ensure the overall policy π is risk averse.

It is evident that the proposed parameterization effectively addresses the challenges outlined earlier. Firstly, it allows for the use of all trajectories collected so far to update the risk-neutral policy within the mixture framework. Secondly, the risk-neutral component encourages the agent to venture into areas of high reward, potentially avoiding the flat tail of the return distribution, and hence mitigates the issue of vanishing gradients. We illustrate the advantages in the following example.

5.3.3 A Motivating Maze Example

Consider a maze domain in Fig. 5.1(a), which is originally from Greenberg et al. (2022) and slightly modified by Luo et al. (2023). Starting from the bottom left corner, the goal of the agent is to reach the green goal state. The gray color marks the walls. The per-step reward is deterministic (i.e., -1) except for the red state, whose reward distribution

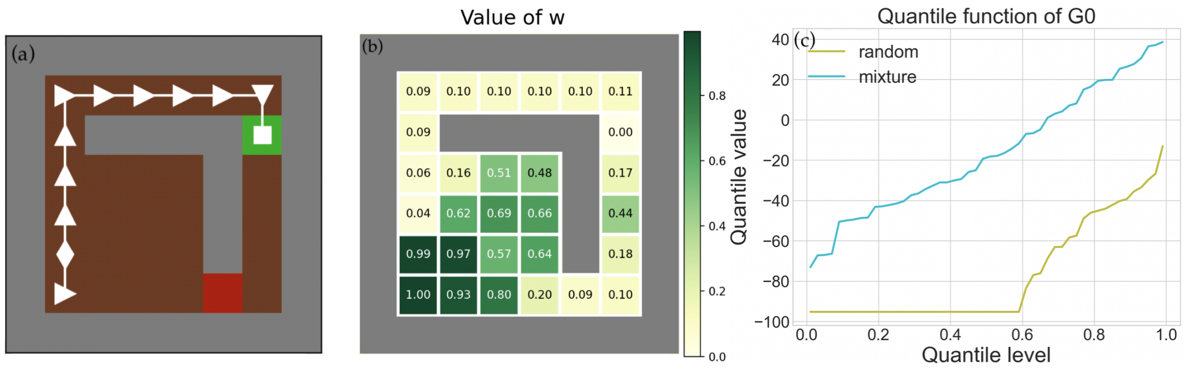


Figure 5.1: (a) A maze domain with green goal state. The red state returns an uncertain reward (details in Sec. 5.3.3). Triangle pointers indicate the risk-neutral actions (not unique for the second state). (b) Value of w of Eq. 5.10 for each state after the mixture policy is updated by CVaR-PG. (c) The empirical quantile function of the total return in maze at an early training stage, if the initial policy is a random and mixture policy.

is $-1 + \mathcal{N}(0,1) \times 30$. The reward for visiting the goal is a positive constant value (i.e., 10). Thus, the shortest path going through the red state towards the goal is the optimal risk-neutral path, while the longer path (shown in white color) is α-CVaR optimal if α is small, though its expected return is slightly lower.

In this domain, suppose we are given the optimal risk neutral policy for each state (which is actually easy to get, e.g., via Q-learning or value iteration (Sutton & Barto, 2018), or even by observing the shortest path), it is easy to see most actions along the white (i.e., risk-averse) path are the same as the risk-neutral policy except the initial state. This means the risk-averse agent only needs to adjust the actions at that state and then follow the optimal risk-neutral policy afterwards. We validate this idea by visualizing the value of $w(s)$ of Eq. 5.10 in Fig. 5.1(b), after the mixture policy is trained by CVaR-PG. The value of $w(s)$ represents the probability of choosing π' at each state. Here the risk neutral policy π^n is pre-computed and provided as the softmax of the optimal Q-values (we use temperatures to make the entropy of π^n small). Thus, π' and w are the components that need to be learned by CVaR-PG. As shown in the figure, the probability of choosing π' is only high in the surroundings of the starting state, and the probability of choosing π^n significantly increases after the agent moves far away from the beginning. Also, the empirical quantile function of G_0^π obtained by this mixture policy at the initial training phase is shown in Fig. 5.1(c). Compared with the randomly initialized policy, the flat tail is eliminated, thereby preventing gradient vanishing.

Remark. This concept, where risk-averse behavior is required only in a subset of states, extends to various fields. For example, in portfolio management, such behavior is crucial only in particular market trends (Ji et al., 2019; Yu et al., 2023), and in healthcare, it is essential only with specific health indicator warnings (Mulligan et al., 2023).

Policy gradient of the mixture policy in tabular case. We show how to compute $\nabla \log \pi(a|s)$ in the above maze case. For our mixture policy, the policy parameter θ consists of two parts $\theta = (\theta_1, \theta_2)$, where θ_1 is for the adjustive policy π'_{θ_1}, and θ_2 is for the weight w. In tabular case, π'_{θ_1} is usually represented by a softmax over state-action feature $\varrho(s,a)$ times policy parameter, i.e.,

$$\pi'_{\theta_1}(a|s) = \frac{\exp\big(\varrho(s,a)\cdot\theta_1\big)}{\sum_b \exp\big(\varrho(s,b)\cdot\theta_1\big)} \tag{5.11}$$

with

$$\nabla_{\theta_1} \log \pi_\theta(a|s) = \varrho(s,a) - \mathbb{E}_{b\sim\pi_\theta(\cdot|s)}\varrho(s,b) \tag{5.12}$$

We represent $w(s) = \sigma(\varrho(s,a)\cdot\theta_2)$, where $\sigma(\cdot)$ is the sigmoid function. Thus

$$\pi_\theta(a|s) = \sigma(\varrho(s,a)\cdot\theta_2)\pi'_{\theta_1}(a|s) + \Big(1-\sigma(\varrho(s,a)\cdot\theta_2)\Big)\pi^n(a|s) \tag{5.13}$$

The derivative of the logarithm is

$$\nabla_{\theta_1} \log \pi_\theta(a|s) = \frac{1}{\pi_\theta(a|s)}\sigma(\varrho(s,a)\cdot\theta_2)\pi'_{\theta_1}(a|s)\nabla_{\theta_1}\log\pi'_{\theta_1}(a|s) \tag{5.14}$$

$$\nabla_{\theta_2} \log \pi_\theta(a|s) = \frac{1}{\pi_\theta(a|s)}(\pi'_{\theta_1}(a|s) - \pi^n(a|s))\sigma(\varrho(s,a)\cdot\theta_2)\Big(1-\sigma(\varrho(s,a)\cdot\theta_2)\Big)\varrho(s,a) \tag{5.15}$$

5.3.4 Offline RL Risk Neutral Learning

This section explores the process of acquiring a risk-neutral policy under the function approximation setting. A natural question is that if we can pre-train a risk neutral policy for mixture as done in the above maze (Sec. 5.3.3), since risk neutral learning is in general faster and more sample efficient than risk averse learning. However, we discovered that incorporating a pre-trained deep risk-neutral policy into the mixture policy frequently leads to a suboptimal risk-averse policy. We elaborate this finding using the LunarLander domain which is previously described in Sec. 4.5.2. In this domain, an additional noisy

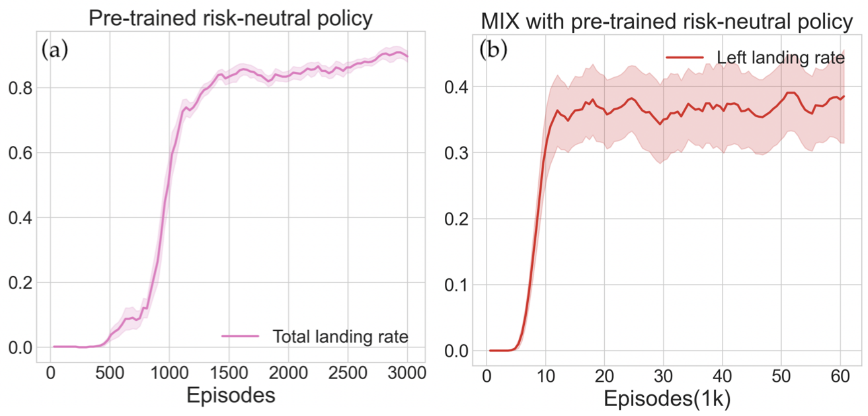


Figure 5.2: (a) The total successful landing rate (y-axsis) of pre-trained risk-neutral policy. (b) The left landing (i.e., risk-averse) (y-axsis) rate of Mix by incorporating this pre-trained risk-neutral policy. Curves are averaged over 10 seeds with shaded regions indicating standard errors.

reward is provided if agent lands on the right ground. Thus, a risk averse agent should land on the left ground as much as possible. Similar as the risk-neutral policy in maze, we represent the risk-neutral policy by the softmax of Q-values with temperature. The Q-values are learned by DQN (Mnih et al., 2013). To validate the pre-trained risk-neutral policy performs well, we show its total successful landing rate in Fig. 5.2(a). The whole training process for mixture policy is as follows. The first 3k episodes are used to update the risk-neutral policy only (i.e., update DQN), with the remaining part of the mixture policy unchanged. After the first 3k episodes, the risk-neutral policy is fixed, and the remaining part of the mixture policy begins to update. However, as indicated by the left landing rate in Fig. 5.2(b), mixture policy leads to a suboptimal risk-averse policy. One possible reason may be when training the deep risk-neutral RL algorithm, the data distribution tends to concentrate on those states in the optimal (or near optimal) trajectories. Thus, the learned value or policy function approximator may not generalize well around the risk-averse path.

To update the risk neutral policy along with the risk averse learning, observing that the update of CVaR-PG typically involves collecting substantial trajectories, these trajectories naturally constitute an empirical MDP to which an offline RL algorithm can be applied to extract a risk-neutral policy. The field of offline RL has seen rapid advancements in recent years, offering promising solutions for solving the empirical MDP formed from the collected trajectories.

Offline RL attempts to learn an optimal policy from a pre-gathered offline dataset $\mathcal{D} = \{(s, a, s', r)\}_{i=1}^n$, where the learning algorithm is restricted to learning from the samples contained within $\mathcal{D}$ without any additional interaction with the real environment. One key challenge in offline RL is to not overestimate the action values outside of the dataset (Fujimoto et al., 2019). To address this challenge, there are generally two strategies. The first approach aims to keep the learned policy closely aligned with the dataset's policy by applying some KL constraint, ensuring the learned policy remains within the dataset's support (Peng et al., 2020; Brandfonbrener et al., 2021; Fujimoto & Gu, 2021). The second strategy involves directly optimizing the policy using the samples available in the dataset (Fujimoto et al., 2019; Kostrikov et al., 2022; Xiao et al., 2023).

In our research, we utilize Implicit Q-Learning (IQL) (Kostrikov et al., 2022) for learning risk-neutral policies, chosen for its proven reliability and empirical validation.

IQL possesses a Q estimator $Q_\phi(s, a)$, a value estimator $V_\psi(s)$, and a policy $\pi_\vartheta^n(a|s)$. Q-function is updated via minimizing

$$L_Q(\phi) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}[(r(s, a) + \gamma V_\psi(s') - Q_\phi(s, a))^2] \tag{5.16}$$

Value function is updated via expectile regression to avoid overestimation ($Q_{\hat{\phi}}$ is the target function)

$$L_V(\psi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}[L_2^\eta(Q_{\hat{\phi}}(s, a) - V_\psi(s))], \quad L_2^\eta(u) = |\eta - \mathbb{I}_{\{u<0\}}|u^2 \tag{5.17}$$

Policy is updated by advantage-weighted regression (Peters & Schaal, 2007) with temperature β

$$L_{\pi^n}(\vartheta) = \mathbb{E}_{(s,a)\sim\mathcal{D}}[\exp(\beta(Q_\phi(s, a) - V_\psi(s))) \log \pi_\vartheta^n(a|s)] \tag{5.18}$$

All the trajectories are stored in a replay buffer for IQL update to learn π^n. In practice, we can perform this update after enough transition data are collected. The overall process of training the mixture policy is described in Algo. 3.

Algorithm 3 Mixture policy for CVaR-PG

```
**Input:** risk level $\alpha$, trajectories sampled per batch $N$, training steps $M$, IQL update frequency $C$
**Initialize:** policy $\pi_\theta = w_{\theta_2}\pi'_{\theta_1} + (1 - w_{\theta_2})\pi^n_\vartheta$ where $\theta = (\theta_1, \theta_2)$, buffer $\mathcal{B}$, Q-function $Q_\phi$ (target $Q_{\hat{\phi}}$), value function $V_\psi$
for $m$ in $1 : M$ do
    $\{\tau_i\}_{i=1}^N \leftarrow$ run_episodes$(\pi_\theta, N)$ Sample trajectories
    Store $\{\tau_i\}_{i=1}^N$ to $\mathcal{B}$
    Update $\theta$ via CVaR-PG$(\pi_\theta, \{\tau_i\}_{i=1}^N, \alpha)$ CVaR PG, i.e., Eq. 5.4
    if m % C == 0 then
        Sample $\mathcal{D} = \{(s, a, r, s')\} \sim \mathcal{B}$ Risk-neutral, e.g., IQL updates
        Update $Q_\phi$ via Eq. 5.16
        Update $V_\psi$ via Eq. 5.17
        Update $\pi^n_\vartheta$ via Eq. 5.18
    end if
end for
```

5.3.5 Related Work: Mixture Policy

Due to its two-layered structure, a mixture policy is also called a hierarchical policy (Daniel et al., 2012). Though the idea of mixture policy is not new, it is mainly applied in risk-neutral settings. Osa et al. (2023) constructed a mixture of deterministic policies for offline RL tasks and showed it can mitigate the issue of critic error accumulation in offline RL. Wulfmeier et al. (2020) and Seyde et al. (2022) utilized mixture policy to capture the diverse motivations of the robots such that the skill learned by each sub-policy can be transferred. Akrour et al. (2021) adopted a mixture policy to enhance the interpretability of decision making. A mixture policy is also used for option discovery (Zhang & Whiteson, 2019; Wulfmeier et al., 2021). The similar mixture structure also appears in value (critic) function learning, for instance, mixture critic is utilized for distributional RL (Choi et al., 2019; Kuznetsov et al., 2020).

5.4 Experiments

We modify several domains such that the risk-averse behavior is clear to identify to evaluate the algorithms. We include REINFORCE with baseline method, as a risk-neutral baseline. In more complex domains, we use SAC (Haarnoja et al., 2018) instead.

Baselines. We compare our method with CVaR-PG in Eq. 5.4 (Tamar et al., 2015), distributional RL with Markov action selection strategy in Eq. 5.8 (denoted as DRL-mkv), and Lim's action selection strategy in Eq. 5.9 (Lim & Malik, 2022) (denoted as DRL-lim). In continuous action domains, we adapt DRL-mkv and DRL-lim by DPG (Silver et al., 2014) as done in Tang et al. (2019), we give a summary for this adaption below. We use MIX to represent our method. Pre-computed π^n in maze is provided to MIX as described in Sec 5.3.3 since it is easy to get. In other domains, π^n is learned by IQL during training. Please refer to Appendix C.1 for any missing implementation details.

DRL-mkv and DRL-lim in continuous domain. The idea is to replace the Q value in Eq. 2.20 (i.e., the risk neutral policy gradient) by the quantities we aim to optimize, e.g., CVaR.

For DRL-mkv, the actor is updated via

$$\nabla_\theta J_\alpha(\theta) = \mathbb{E}_{s,a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s)\mathrm{CVaR}_\alpha(Z^\pi(s,a))] \tag{5.19}$$

for DRL-lim, the actor is updated via

$$\nabla_\theta J_\alpha(\theta) = \mathbb{E}_{s,k,a\sim\pi_\theta}\left[\nabla_\theta - \log \pi(a|s)\mathbb{E}[(k - Z^\pi(s,a))^+]\right] \tag{5.20}$$

where k is the tracking variable at state s.

Quantile regression in DRL-mkv and DRL-lim. Both DRL-mkv and DRL-lim are built on top of distributional RL (Bellemare et al., 2017). The most commonly used approach to update distributional value function (critic) is quantile regression (Dabney et al., 2018b,a; Zhou et al., 2020, 2021; Luo et al., 2022). We also adopt quantile regression in our implementation. As discussed in Chapter 3, Zhou et al. (2020) pointed out some previous quantile regression based work, e.g., QR-DQN (Dabney et al., 2018b), IQN (Dabney et al., 2018a) suffered from the quantile crossing issue, i.e., the predicted quantile values do not satisfy the monotonicity of the quantile function. This is shown to hinder policy learning and exploration (Zhou et al., 2020). The monotonicity of the quantile is also important in DRL-mkv and DRL-lim to make sure the estimated quantities, e.g. α-CVaR, are correct. We follow the approach in Yue et al. (2020) by sorting the predicted quantile values to make them non decreasing.

Remark. The method in Greenberg et al. (2022) is CVaR-PG with curriculum learning and a special trajectory sampling strategy, which is orthogonal to our approach. The idea of this sampling strategy is that by changing some parameters of the environment, the sampled trajectories correspond to the tail return trajectories in the original environment. The parameters are updated based on the idea of cross entropy (De Boer et al.,

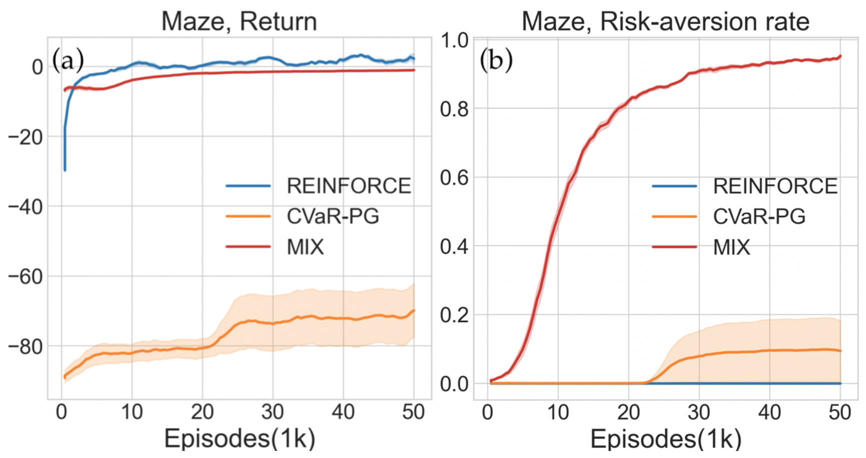


Figure 5.3: (a) Policy return (y-axsis) and (b) Risk-aversion (long path) rate (y-axsis) v.s. training episodes in Maze. Curves are averaged over 10 seeds with shaded regions indicating standard errors..

2005). Thus this method requires to control the environment dynamics, and may not be straightforwardly applicable to most domains discussed here. We compare with it in one domain from Greenberg et al. (2022) in Sec. 5.4.4.

5.4.1 Tabular case: Maze Problem

This domain is modified from Greenberg et al. (2022) that was previously described in Sec. 5.3.3. The maximum episode length is 100. CVaR $\alpha = 0.1$. REINFORCE, CVaR-PG, and MIX collect $N = 50$ episodes before updating the policy. Here we report the rate of choosing the long path during training in Fig. 5.3(b). Since the policy is non-deterministic, the length of the sampled risk-averse path may not be exactly 11 (the length of the white path in Fig. 5.1(a)). Here we treat a path as risk-averse if it goes towards the top, reaches the goal, and the path length does not exceed 14.

CVaR-PG fails to learn a reasonable policy even in this simple domain due to gradient vanishing as discussed in Sec. 5.3.1. By initializing MIX with a risk neutral policy, it achieves a relatively high return at the early learning phase, thus potentially avoids gradient vanishing.

5.4.2 Discrete control: LunarLander

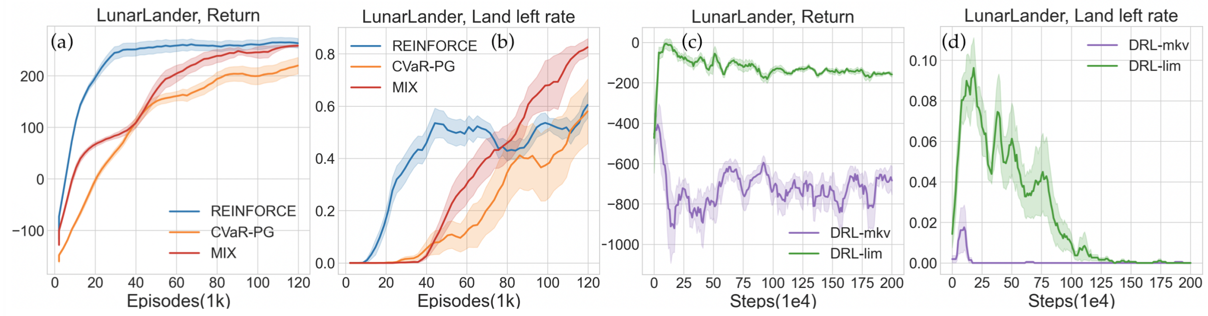


Figure 5.4: (a,c) Policy return (y-axis), and (b,d) Left-landing rate (i.e., risk-averse landing rate) (y-axis) v.s. training episodes or steps in LunarLander. Curves are averaged over 10 seeds with shaded regions indicating standard errors. For the landing left rate, higher is better.

This domain is taken from OpenAI Gym (Brockman et al., 2016a). We refer readers to its official documents for a full description. The goal of the agent is to land on the ground without crashing. We split the ground into left and right parts by the middle line of the landing pad, as shown in Fig. 4.5 in Chapter 4. If landing on the right, an additional noisy reward sampled from $\mathcal{N}(0, 1)$ times 100 is given. A risk-averse agent should learn to land on the left as much as possible. We set CVaR $\alpha = 0.1$. REINFORCE, CVaR-PG, and MIX collect $N = 30$ episodes before updating the policy. DRL-mkv and DRL-lim are off-policy methods and update policies at each environment step. We train them for 2e6 steps instead of as many episodes as other methods.

We report the left-landing rate of different methods in Fig. 5.4(b) and (d). Comparing DRL-mkv and DRL-lim against episode-based algorithms is not straightforward within the same figure due to the difference in parameter update frequency. Thus we show them separately. MIX achieves a comparable return with REINFORCE at the end, but shows a clear risk-aversion by landing more on the left. DRL-mkv and DRL-lim can not learn a reasonable policy given the small CVaR α. As mentioned in Section 5.2.3, they optimize a different objective than CVaR that is not well understood.

Investigating the IQL behavior. The parameterization of mixtures naturally raises the question whether IQL alone is sufficient to identify a risk-averse policy. As a result, we also report the landing rate of the risk neural component π^n of MIX learned by IQL

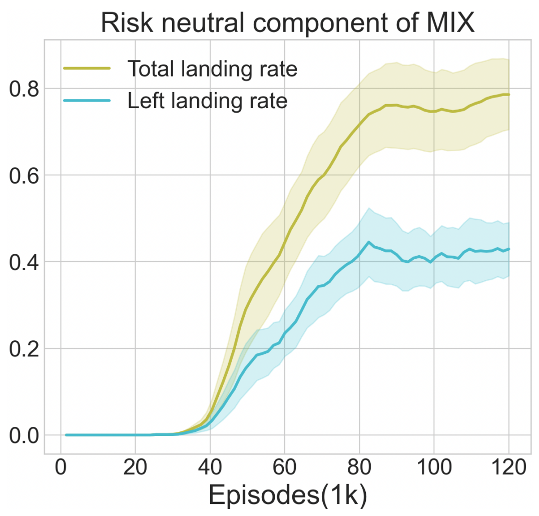


Figure 5.5: Total landing rate and left landing rate of IQL (y-axsis) in MIX during training. Curves are averaged over 10 seeds with shaded regions indicating standard errors.

during training in Fig. 5.5 to indicate IQL along does not achieve risk-aversion. Also, the left landing rate (i.e., risk averse rate) is roughly the half of the total landing rate, which means IQL does not show a clear preference of laning at left or right in this domain.

5.4.3 Continuous control: Mujoco

Mujoco (Todorov et al., 2012) is a collection of robotics environments with continuous states and actions in OpenAI Gym (Brockman et al., 2016a). Here, we select three domains, namely InvertedPendulum, HalfCheetah, and Ant. Inspired by Malik et al. (2021); Liu et al. (2023), we define the risky region based on the X-position. Specifically, if X-position > 0.04 in InvertedPendulum, X-position < -3 in HalfCheetah and Ant, a zero-mean Gaussian noise is added to the reward ($\mathcal{N}(0, 1) \times 10$ in InvertedPendulum, $\mathcal{N}(0, 1) \times 50$ in HalfCheetah and Ant). To further ensure that agents move both forward and backward with equal preference in terms of expected reward in HalfCheetah and Ant, we define the distance-based reward as the difference in distance between the current and previous states from the origin, regardless of the sign of the X-position. This means the distance reward is positive only when agents are moving away from the origin.

Consequently, in InvertedPendulum, a risk-averse agent aims to keep the pendulum balanced while staying out of the noisy region. In HalfCheetah and Ant, a risk-averse

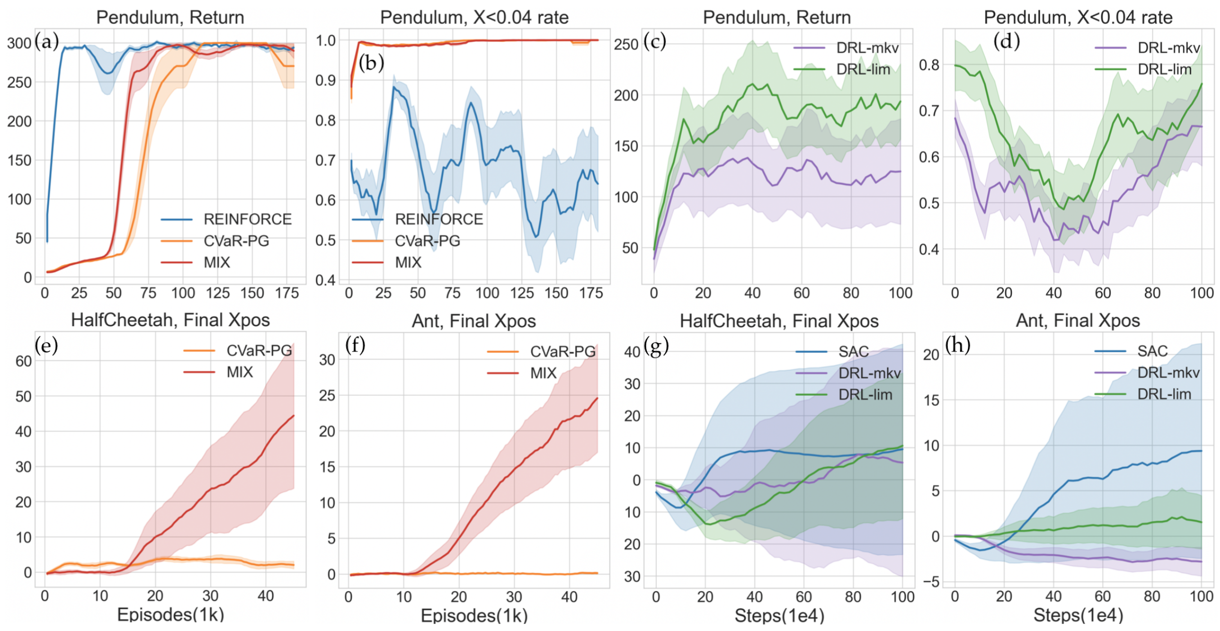


Figure 5.6: (a, c) Policy return (y-axsis) in InvertedPendulum, (b, d) visiting non-noisy region rate (y-axis) in InvertedPendulum, (e, g) Final X-position (y-axsis) in HalfCheetah, (f, h) Final X-position in Ant (y-axsis) v.s. training episodes for steps in Mujoco. Curves are averaged over 10 seeds with shaded regions indicating standard errors. For the location visiting rate, higher is better.

agent should learn to move toward the opposite direction of the noisy region. We optimize CVaR $\alpha = 0.2$. REINFORCE still serves as the risk neutral baseline in InvertedPendulum. In HalfCheetah and Ant, we use SAC (Haarnoja et al., 2018) instead, since the vanilla policy gradient is not good at more complex domains. REINFORCE, CVaR-PG, and MIX collect $N = 30$ episodes before updating the policy in InvertedPendulum, and $N = 15$ in HalfCheetah and Ant. DRL-mkv, DRL-lim, and SAC are trained for 1e6 steps.

We report the total return and X$<$ 0.04 rate in InvertedPendulum, which are sufficient to reflect the risk-averse behavior of the agent, since the reward is 1 as long as the pendulum is balanced. In HalfCheetah and Ant, we report the final X-position in Fig. 5.6, as the return can not reflect which direction the agent is moving in the two domains. CVaR-PG achieves a risk-averse policy in InvertedPendulum, i.e., high return and high rate of staying in the non-noisy region. But it fails to learn a reasonable policy in HalfCheetah and Ant, i.e., the final X-position is always close to the origin. MIX learns risk-averse policy by moving away from the noisy region in all three domains. DRL-mkv and DRL-lim generally do not work well in all three domains since they optimize a different objective than CVaR that is not well understood (see Sec 5.2.3).

5.4.4 Driving Game

Since the method by Greenberg et al. (2022) modifies the environment dynamic and it is unclear how it can be applied to the domains described above, we choose a domain from Greenberg et al. (2022) for comparison. In this driving game, there are two cars. The agent's car has to follow the ego car for 30 seconds as closely as possible without colliding. Every 1.5 seconds, the leader chooses a random action: drive straight, accelerate, decelerate, change lane, or brake hard, with probabilities $p_0 = (0.35, 0.3, 0.248, 0.1, 0.002)$. We refer reader to Sec. 5.2 of Greenberg et al. (2022) for more details. The state dimension is 5. The action dimension is 5.

The method proposed in Greenberg et al. (2022) is named CeSoR, which includes a curriculum learning scheduler to adjust CVaR α during learning, i.e., starting from a large value for α and gradually decreasing to its target value; and a trajectory generator which controls the environment dynamic. In this domain, it controls the behavior of the leader car, i.e., can modify the action probability p_0.

We directly use the code provided by Greenberg et al. (2022) to produce the results for CeSoR. CeSoR is orthogonal to MIX, and these two can be combined. We combine MIX with curriculum learning (denoted as MIX+SoR, SoR means soft risk to represent

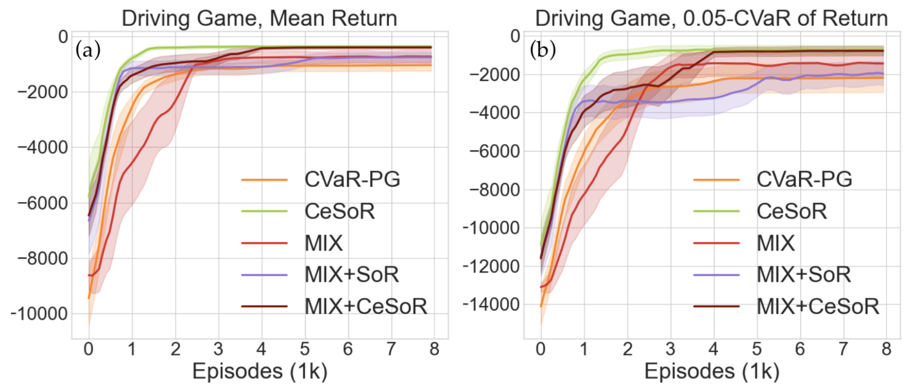


Figure 5.7: (a) The expected return (y-axsis), and (b) the 0.05-CVaR of the return (y-axsis) achieved by CVaR-PG, CeSoR, MIX, MIX+SoR, and MIX+CeSoR in driving game. Curves are averaged over 10 seeds with shaded regions indicating standard errors.

curriculum learning in Greenberg et al. (2022)), and combine MIX with CeSoR (denoted as MIX+CeSoR) in this domain.

We report the mean return and the 0.05-CVaR of the return in Fig. 5.7. For both the mean and tail of the return, CeSoR, MIX, and MIX variants are better than CVaR-PG. CeSoR is slightly better than MIX, since CeSoR possesses the environment dynamic information while MIX does not. MIX with curriculum learning, i.e., MIX+SoR, learns faster than MIX at the early training stage than MIX, though the final mean return is the same as MIX. MIX+CeSoR is better than MIX and MIX+SoR with respect to the mean and tail of the return, and is comparable to CeSoR. CeSoR learns faster and achieves the highest risk averse rewards among all techniques, however it requires access to the environment dynamics and the ability to change the parameters of the dynamics in a way that is domain specific. In contrast, MIX and MIX+SoR do not need dynamics information and therefore can be applied directly to any domain.

Remark. Our mixture policy method differs from the curriculum learning idea in Greenberg et al. (2022). Tough the CVaR α starts from a large value in curriculum learning, where the objective is close to a risk-neutral problem, it is an on-policy policy gradient method, i.e., the trajectories used to update the policy is generated by the current policy (if no importance sampling is assumed). In contrast, the risk-neutral component of our mixture policy is trained by an off-policy (offline) algorithm, in this case, all the encountered trajectories can be stored in the replay buffer for policy update.

5.5 Summary

This chapter proposes a mixture policy framework for CVaR-PG. Though the idea of mixture policy is not new, it is mainly applied in risk-neutral settings. In our approach, the policy is a mixture of a risk neutral policy with an adjustable policy. It is motivated to overcome the sample inefficiency of the original CVaR-PG, caused by the waste of most sampled trajectories and gradient vanishing in some domains. The method is based on the insight that in many applications the risk averse behavior is only required in a subset of state space. This perspective covers many real world fields, e.g., driving cars, healthcare, fiance. We empirically show that our method can succeed when others fail to learn a risk-averse or a reasonable policy by mitigating the sample efficiency issue.

Chapter 6

Conclusion

The thesis studies the problem of learning policies under uncertainty and risk, e.g., how to efficiently learn a quantile function for a return distribution; how to stabilize policy gradient for measures of variability; how to improve the sample efficiency of the CVaR policy gradient.

We provided some answers for these questions. For distributional RL, we propose a new learning framework, monotonic spline DQN, to learn quantile functions for value distributions. The quantile function is parameterized by monotonic splines which provide a continuous smooth monotonic interpolation. This method offers much more flexibility in terms of quantile learning and sampling compared with existing methods. For optimizing measures of variability, we provide a thorough analysis to reveal the potential issues of using variance or standard deviation as the measure function. These issues are not discussed before in the literature. We propose to use another measure of variability, Gini deviation, as an alternative to stabilize the policy gradient. For optimizing CVaR, the sample efficiency of the original policy gradient is low due to its mathematical nature and the potential of gradient vanishing. We propose a simple mixture policy approach by mixing a risk neutral component to improve the sample efficiency and tackle the gradient vanishing problem.

6.1 Limitations

6.1.1 Monotonic spline DQN

First, similar to previous work in distributional RL, the output dimension of the neural network is much larger than the original DQN (since N quantile values are associated with an action), thus it may not scale as well as non-distributional RL techniques.

Second, there is a computational cost associated with representing the value distribution. Since our approach requires to estimate the derivatives of the knots, it generally requires more parameters compared with NDQFN (piece-wise linear) if the number of bins are the same. It also requires more parameters comparing with QR-DQN and NC-QR-DQN where the quantile levels are fixed and only quantile values need to be learned. More training parameters usually require more training time. Thus there is a trade off between the learning cost and the performance.

Third, this method is value-based, thus it is limited to RL techniques that include a value function. When a value function is hard to incorporate, e.g., Gini deviation policy gradient (Chapter 4), CVaR policy gradient (Chapter 5), distributional RL is not straightfoward to apply.

6.1.2 Mean-Gini deviation policy gradient

First, similar to all on-policy policy gradient algorithms, new trajectories need to be sampled under the current policy every time the policy is updated. Thus the sample efficiency is not high. One way to improve sample efficiency is by incorporating importance sampling for off-policy updates as done in Sec. 4.4.3. However, this may introduce high variance to the policy gradient, and the proposed methods, e.g., selecting small importance sampling ratio or clipping importance sampling may introduce bias.

Second, our approach requires to sample multiple trajectories for one update. The update frequency is lower than those mean-variance methods that can do per-episode update, and the off-policy reward based mean-variance methods that can do per environment step update.

6.1.3 Mixture policy for CVaR optimization

First, we pinpointed a class of risk-averse RL problems characterized by requiring risk-averse behavior in a subset of states, suitable for our mixture approach. Though this category intuitively covers a broad range of scenarios, situations that do not fit this framework remain unexplored and may not be suitable for our approach.

Second, our method can be potentially integrated with other techniques aimed at enhancing sample efficiency of CVaR policy gradient, e.g., Greenberg et al. (2022), given its versatile nature. However, exploring such hybrid methodologies falls outside the scope of our current research.

6.2 Discussion of Dynamic Risk Measure

Distributional RL learns the full value distribution. Thus, the risk profile is easy to be extracted, which seems to indicate distributional RL is a good candidate to accelerate risk-sensitive RL. Generally, distributional RL methods are value-based and heavily rely on the distributional Bellman equation. The value distribution estimates are updated via a time difference manner. The methods discussed in Chapter 3 are all risk-neutral methods, i.e., the goal is to maximize the expected total return. Tough some methods explored the possibility to combine distributional RL with risk-sensitive RL by updating a policy towards some risk measures of the value distribution (see Sec. 5.2.4), those approaches are not consistent with the objective of optimizing the risk measures of the total return. Those approaches may empirically achieve risk-sensitive behaviors in some cases but the performance depends on carefully setting the risk preference (Dabney et al., 2018a). The main barrier of combining distributional RL and risk-sensitive RL together lies in the fact that the risk measures defined on total return rarely have a Bellman equation in control settings in contrast to risk-neutral RL. There are Bellman equations of risk measures for policy evaluation, e.g., variance (Tamar et al., 2012), CVaR (Chow et al., 2015), but those equations can not be applied to control settings since dynamic programming cannot be used.

To obtain a Bellman equation for risk measures in control settings, the dynamic risk measures come to rescue. Dynamic risk measures, as briefly mentioned in Sec. 5.2.1, are defined recursively at each time step. For a MDP M and a static risk measure ρ, the dynamic risk measure $\rho^{\pi}_{\infty}(M)$ is defined as

$$\rho^{\pi}_{\infty}(M) = R(s_0, \pi(s_0)) + \rho\Big(\gamma R(s_1, \pi(s_1)) + \rho\big(\gamma^2 R(s_2, \pi(s_2)) + ...\big)\Big) \tag{6.1}$$

where $(s_0, s_1, ...)$ indicates random trajectories drawn from the MDP M by policy π. Define the risk-sensitive value function under policy π as $V^{\pi}(s) = \rho^{\pi}_{\infty}(M|s_0 = s)$, the risk-sensitive Bellman equation (Ruszczyński, 2010) is

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \min_{\xi \in \mathcal{U}_{\rho}(P(\cdot|s,a))} \sum_{s'} P(s'|s, \pi(a))\xi(s')V^{\pi}(s') \tag{6.2}$$

where $\mathcal{U}_{\rho}(P(\cdot|s, a))$ is the risk envelope of risk measure ρ. When $\rho = \text{CVaR}_{\alpha}$, the risk envelope is given by

$$\mathcal{U}_{\text{CVaR}_{\alpha}}(P(\cdot|s, a)) = \{\xi : \xi(\omega) \in [0, \frac{1}{\alpha}], \sum_{\omega} \xi(\omega)P(\omega|s, a) = 1\} \tag{6.3}$$

Likewise, the optimal risk-sensitive value $V^*(s) = \max_\pi \rho^\pi_\infty(M|s_0 = s)$ is the unique solution to the risk-sensitive Bellman optimality equation

$$V^*(s) = \max_a \Big\{ R(s,a) + \gamma \min_{\xi \in \mathcal{U}_\rho(P(\cdot|s,a))} \sum_{s'} P(s'|s,a)\xi(s')V^*(s') \Big\} \tag{6.4}$$

We can view the dynamic risk measure as the expected value in a MDP where the transition dynamics at each step are modified within the risk-envelope.

Note that the above Bellman equation requires the environment transition dynamics to be known to users and the risk measure ρ can be expressed via its risk envelope. The benefit of using dynamic risk measures is that we can use Bellman's equation to do temporal difference updates instead of on-policy policy gradient, which improves sample efficiency. The drawback is that the dynamic risk measure is hard to interpret. When risk is recursively defined at each step, the overall meaning of the risk is unclear. This may be problematic in real world applications when the meaning of the risk should be clear.

6.3 Future Research Directions

Among the many possible future directions, we prioritize pursuing the following directions.

First, it is natural to investigate how to combine distributional RL with risk-sensitive RL if we wish the optimization procedure to be consistent with the risk-sensitive objective. As discussed before, when the static risk is defined on the total return G_0, there is no Bellman optimality equation for static risk measures in general, thus combining distributional RL with static risk is not straightfoward. When the risk measure is dynamic, though a Bellman optimality equation exists, the value function for dynamic risk can not be directly extracted from the value distribution in distributional RL. These two facts make this problem challenging but also meaningful.

Second, though new algorithms are developed for optimizing static measures of variability, e.g., Gini deviation, and static tail risk measure, e.g., CVaR, and achieved better empirical performance than existing methods, the sample efficiency is still not high especially compared with off-policy risk-neutral methods. As a result, one potential avenue for future work is to further enhance the sample efficiency of our algorithms. This can be achieved by more effectively utilizing off-policy data or by adapting the algorithm to be compatible with online, incremental learning.

Third, there is significance to study the connection between different risk measures and the effect of hybrid risk measures in RL. Based on the literature review, different risk measures are studied separately in different RL papers, and therefore it is not clear what is the relation among these risk measures nor when should we prefer one risk measure over another are unclear. Also, hybrid risk measures, i.e., combining different risk measures, may be adopted in real-world applications to leverage the advantages of individual risk measures.

Last, offline risk-averse RL is an interesting and challenging future direction. In real world applications, it is usually easy to collect data from the real domains and the offline RL techniques can be applied. Different from online RL, offline RL leans a policy only from a fixed dataset, which makes Monte Carlo based risk-averse policy gradient hard to be directly applied. There exists some research work on this topic when the risk measure is CVaR, e.g., Urpí et al. (2021); Rigter et al. (2023). Urpí et al. (2021) updates a policy according to Eq. 5.19, which is not consistent with optimizing either static or dynamic risk. Rigter et al. (2023) uses dynamic CVaR (Eq. 6.1) and is model-based. How to optimize static risk or risk measures other than CVaR are still open questions.

References

Riad Akrour, Davide Tateo, and Jan Peters. Continuous action reinforcement learning from a mixture of interpretable experts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):6795–6806, 2021.

Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. Coherent measures of risk. *Mathematical finance*, 9(3):203–228, 1999.

N. N. Author. Suppressed for anonymity, 2021.

Stefan Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta mathematicae*, 3(1):133–181, 1922.

Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, TB Dhruva, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

Nicole Bäuerle and Jonathan Ott. Markov decision processes with average-value-at-risk criteria. *Mathematical Methods of Operations Research*, 74:361–379, 2011.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 449–458. PMLR, 2017.

Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pp. 679–684, 1957.

Richard Bellman. Dynamic programming. *science*, 153(3731):34–37, 1966.

Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997.

Lorenzo Bisi, Luca Sabbioni, Edoardo Vittori, Matteo Papini, and Marcello Restelli. Risk-averse trust region optimization for reward-volatility reduction. *International Joint Conference on Artificial Intelligence*, 2020.

Tomas Björk, Agatha Murgoci, and Xun Yu Zhou. Mean–variance portfolio optimization with state-dependent risk aversion. *Mathematical Finance: An International Journal of Mathematics, Statistics and Financial Economics*, 24(1):1–24, 2014.

Vivek S Borkar. Q-learning for risk-sensitive control. *Mathematics of operations research*, 27(2):294–311, 2002.

Léon Bottou, Jonas Peters, Joaquin Quiñonero-Candela, Denis X Charles, D Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard, and Ed Snelson. Counterfactual reasoning and learning systems: The example of computational advertising. *Journal of Machine Learning Research*, 14(11), 2013.

David Brandfonbrener, William F Whitney, Rajesh Ranganath, and Joan Bruna. Offline RL without off-policy evaluation. In *Advances in Neural Information Processing Systems*, 2021.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016a.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016b.

Yunho Choi, Kyungjae Lee, and Songhwai Oh. Distributional deep reinforcement learning with a mixture of gaussians. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9791–9797. IEEE, 2019.

Gustave Choquet. Theory of capacities. In *Annales de l'institut Fourier*, volume 5, pp. 131–295, 1954.

Yinlam Chow and Mohammad Ghavamzadeh. Algorithms for cvar optimization in mdps. *Advances in neural information processing systems*, 27, 2014.

Yinlam Chow, Aviv Tamar, Shie Mannor, and Marco Pavone. Risk-sensitive and robust decision-making: a cvar optimization approach. *Advances in neural information processing systems*, 28, 2015.

Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *The Journal of Machine Learning Research*, 18(1):6070–6120, 2017.

Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *Journal of Machine Learning Research*, 18(167):1–51, 2018.

Felipe Leno Da Silva, Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. Uncertainty-aware action advising for deep reinforcement learning agents. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 5792–5799, 2020.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *Proceedings of the International conference on machine learning (ICML)*, pp. 1096–1105. PMLR, 2018a.

Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 32, 2018b.

Christian Daniel, Gerhard Neumann, and Jan Peters. Hierarchical relative entropy policy search. In *Artificial Intelligence and Statistics*, pp. 273–281. PMLR, 2012.

Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134:19–67, 2005.

Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *International conference on machine learning*, 2012.

Freddy Delbaen and Sara Biagini. *Coherent risk measures.* Springer, 2000.

Dieter Denneberg. Premium calculation: Why standard deviation should be replaced by absolute deviation1. *ASTIN Bulletin: The Journal of the IAA*, 20(2):181–190, 1990.

Thang Doan, Bogdan Mazoure, and Clare Lyle. Gan q-learning. *arXiv preprint arXiv:1805.04874*, 2018.

Hadi Mohaghegh Dolatabadi, Sarah Erfani, and Christopher Leckie. Invertible generative modeling using linear rational splines. In *International Conference on Artificial Intelligence and Statistics (AISTAT)*, pp. 4236–4246. PMLR, 2020.

Yihan Du, Siwei Wang, and Longbo Huang. Provably efficient risk-sensitive reinforcement learning: Iterated cvar and worst path. In *Proceedings of the Eleventh International Conference on Learning Representations, ICLR-23*, 2023.

R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* John Wiley and Sons, 2nd edition, 2000.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Benjamin Ellenberger. Pybullet gymperium. https://github.com/benelot/pybullet-gym, 2018–2019.

Yingjie Fei, Zhuoran Yang, and Zhaoran Wang. Risk-sensitive reinforcement learning with function approximation: A debiasing approach. In *International Conference on Machine Learning*, pp. 3198–3207. PMLR, 2021.

Razvan V Florian. Correct equations for the dynamics of the cart-pole system. *Center for Cognitive and Neural Studies (Coneural), Romania*, 2007.

Dror Freirich, Tzahi Shimkin, Ron Meir, and Aviv Tamar. Distributional multivariate policy evaluation and exploration with the bellman gan. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1983–1992. PMLR, 2019.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. *International Conference on Machine Learning*, pp. 2052–2062, 2019.

Edward Furman, Ruodu Wang, and Ričardas Zitikis. Gini-type measures of risk and variability: Gini shortfall, capital allocations, and heavy-tailed risks. *Journal of Banking & Finance*, 83:70–84, 2017.

Corrado Gini. *Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche.[Fasc. I.].* Tipogr. di P. Cuppini, 1912.

Gerald J Glasser. Variance formulas for the mean difference and coefficient of concentration. *Journal of the American Statistical Association*, 57(299):648–654, 1962.

Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion.* Addison-Wesley, Reading, Massachusetts, 1994.

Michel Grabisch. The application of fuzzy integrals in multicriteria decision making. *European journal of operational research*, 89(3):445–456, 1996.

Bogdan Grechuk, Anton Molyboha, and Michael Zabarankin. Maximum entropy principle with general deviation measures. *Mathematics of Operations Research*, 34(2):445–467, 2009.

Ido Greenberg, Yinlam Chow, Mohammad Ghavamzadeh, and Shie Mannor. Efficient risk-averse reinforcement learning. *Advances in Neural Information Processing Systems*, 35: 32639–32652, 2022.

JA Gregory and R Delbourgo. Piecewise rational quadratic interpolation to monotonic data. *IMA Journal of Numerical Analysis*, 2(2):123–130, 1982.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1): 723–773, 2012.

Audrunas Gruslys, Will Dabney, Mohammad Gheshlaghi Azar, Bilal Piot, Marc Bellemare, and Remi Munos. The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. *International Conference on Learning Representations*, 2017.

Arjun K Gupta and Mohammad AS Aziz. Convex ordering of random variables and its applications in econometrics and actuarial science. *European Journal of Pure and Applied Mathematics*, 3(5):779–785, 2010.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Mary R Hardy and Julia L Wirch. The iterated cte: a dynamic risk measure. *North American Actuarial Journal*, 8(4):62–75, 2004.

Jia Lin Hau, Erick Delage, Mohammad Ghavamzadeh, and Marek Petrik. On dynamic programming decompositions of static risk measures in markov decision processes. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 32, 2018.

Takuya Hiraoka, Takahisa Imagawa, Tatsuya Mori, Takashi Onishi, and Yoshimasa Tsuruoka. Learning robust options by conditional value at risk optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

Audrey Huang, Liu Leqi, Zachary C Lipton, and Kamyar Azizzadenesheli. On the convergence and optimality of policy gradient for markov coherent risk. *arXiv preprint arXiv:2103.02827*, 2021.

Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, pp. 492–518. Springer, 1992.

Tommi Jaakkola, Michael Jordan, and Satinder Singh. Convergence of stochastic iterative dynamic programming algorithms. *Advances in neural information processing systems*, 6, 1993.

Ran Ji, KC Chang, and Zhenlong Jiang. Risk-aversion adjusted portfolio optimization with predictive modeling. In *2019 22th International Conference on Information Fusion (FUSION)*, pp. 1–8. IEEE, 2019.

Whiyoung Jung, Myungsik Cho, Jongeui Park, and Youngchul Sung. Quantile constrained reinforcement learning: A reinforcement learning framework constraining outage probability. *Advances in Neural Information Processing Systems*, 35:6437–6449, 2022.

M. J. Kearns. *Computational Complexity of Machine Learning.* PhD thesis, Department of Computer Science, Harvard University, 1989.

Ramtin Keramati, Christoph Dann, Alex Tamkin, and Emma Brunskill. Being optimistic to be conservative: Quickly learning a cvar policy. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 4436–4443, 2020.

Donald Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1986.

Roger Koenker and Kevin F Hallock. Quantile regression. *Journal of economic perspectives*, 15(4):143–156, 2001.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022.

Anurag Koul, Sam Greydanus, and Alan Fern. Learning finite state representations of recurrent policy networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Shigeo Kusuoka. On law invariant coherent risk measures. In *Advances in mathematical economics*, pp. 83–95. Springer, 2001.

Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 5556–5566. PMLR, 2020.

Prashanth La and Mohammad Ghavamzadeh. Actor-critic algorithms for risk-sensitive mdps. *Advances in neural information processing systems*, 26, 2013.

Leslie Lamport. *LATEX — A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

P. Langley. Crafting papers on machine learning. In Pat Langley (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Duan Li and Wan-Lung Ng. Optimal dynamic portfolio selection: Multiperiod mean-variance formulation. *Mathematical finance*, 10(3):387–406, 2000.

Luchen Li and A Aldo Faisal. Bayesian distributional policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016a.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *International Conference on Learning Representations*, 2016b.

Shiau Hong Lim and Ilyas Malik. Distributional reinforcement learning for risk-sensitive policies. *Advances in Neural Information Processing Systems*, 35:30977–30989, 2022.

Fangda Liu and Ruodu Wang. A theory for measures of tail risk. *Mathematics of Operations Research*, 46(3):1109–1128, 2021.

Fangda Liu, Jun Cai, Christiane Lemieux, and Ruodu Wang. Convex risk functionals: Representation and applications. *Insurance: Mathematics and Economics*, 90:66–79, 2020.

Guiliang Liu, Yudong Luo, Ashish Gaurav, Kasra Rezaee, and Pascal Poupart. Benchmarking constraint inference in inverse reinforcement learning. In *International Conference on Learning Representations*, 2023.

Yudong Luo, Guiliang Liu, Haonan Duan, Oliver Schulte, and Pascal Poupart. Distributional reinforcement learning with monotonic splines. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=C8Ltz08PtBp.

Yudong Luo, Guiliang Liu, Pascal Poupart, and Yangchen Pan. An alternative to variance: Gini deviation for risk-averse policy gradient. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=B7QRV4XXiK.

Yudong Luo, Yangchen Pan, Han Wang, Philip Torr, and Pascal Poupart. A simple mixture policy parameterization for improving sample efficiency of cvar optimization. *arXiv preprint arXiv:2403.11062*, 2024.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61: 523–562, 2018.

Shehryar Malik, Usman Anwar, Alireza Aghasi, and Ali Ahmed. Inverse constrained reinforcement learning. In *International Conference on Machine Learning*, pp. 7390–7399. PMLR, 2021.

Shie Mannor and John Tsitsiklis. Mean-variance optimization in markov decision processes. *International Conference on Machine Learning*, 2011.

Harry M Markowitz and G Peter Todd. *Mean-variance analysis in portfolio choice and capital markets*, volume 66. John Wiley & Sons, 2000.

John Martin, Michal Lyskawinski, Xiaohu Li, and Brendan Englot. Stochastically dominant distributional reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 6745–6754. PMLR, 2020.

R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (eds.). *Machine Learning: An Artificial Intelligence Approach, Vol. I.* Tioga, Palo Alto, CA, 1983.

T. M. Mitchell. The need for biases in learning generalizations. Technical report, Computer Science Department, Rutgers University, New Brunswick, MA, 1980.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Philippe Mongin. Expected utility theory. 1998.

Tetsuro Morimura, Masashi Sugiyama, Hisashi Kashima, Hirotaka Hachiya, and Toshiyuki Tanaka. Parametric return density estimation for reinforcement learning. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 368–375, 2010.

Karen Mulligan, Drishti Baid, Jason N Doctor, Charles E Phelps, and Darius N Lakdawalla. Risk preferences over health: Empirical estimates and implications for healthcare decision-making. Technical report, National Bureau of Economic Research, 2023.

Mohammad Naghshvar, Ahmed K Sadek, and Auke J Wiggers. Risk-averse behavior planning for autonomous driving under uncertainty. *arXiv preprint arXiv:1812.01254*, 2018.

A. Newell and P. S. Rosenbloom. Mechanisms of skill acquisition and the law of practice. In J. R. Anderson (ed.), *Cognitive Skills and Their Acquisition*, chapter 1, pp. 1–51. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1981.

Thanh Tang Nguyen, Sunil Gupta, and Svetha Venkatesh. Distributional reinforcement learning via moment matching. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

Takayuki Osa, Akinobu Hayashi, Pranav Deo, Naoki Morihira, and Takahide Yoshiike. Offline reinforcement learning with mixture of deterministic policies. *Transactions on Machine Learning Research*, 2023.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29, 2016.

Brendan O'Donoghue, Ian Osband, Remi Munos, and Volodymyr Mnih. The uncertainty bellman equation and exploration. In *International conference on machine learning*, pp. 3836–3845, 2018.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2020.

Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th international conference on Machine learning*, pp. 745–750, 2007.

Georg Ch Pflug and Alois Pichler. Time-consistent decisions and temporal decomposition of coherent risk functionals. *Mathematics of Operations Research*, 41(2):682–699, 2016.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, 2014.

Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. Epopt: Learning robust neural network policies using model ensembles. In *International Conference on Learning Representations*, 2016.

Marc Rigter, Bruno Lacerda, and Nick Hawes. One risk to rule them all: A risk-sensitive perspective on model-based offline reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2023.

R Tyrrell Rockafellar, Stanislav Uryasev, et al. Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42, 2000.

R Tyrrell Rockafellar, Stan Uryasev, and Michael Zabarankin. Generalized deviations in risk analysis. *Finance and Stochastics*, 10(1):51–74, 2006.

Elvezio M Ronchetti and Peter J Huber. *Robust statistics*. John Wiley & Sons, 2009.

Michael Rothschild and Joseph E Stiglitz. Increasing risk: I. a definition. In *Uncertainty in Economics*, pp. 99–121. Elsevier, 1978.

Mark Rowland, Marc Bellemare, Will Dabney, Rémi Munos, and Yee Whye Teh. An analysis of categorical distributional reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 29–37. PMLR, 2018.

Mark Rowland, Rémi Munos, Mohammad Gheshlaghi Azar, Yunhao Tang, Georg Ostrovski, Anna Harutyunyan, Karl Tuyls, Marc G Bellemare, and Will Dabney. An analysis of quantile temporal-difference learning. *arXiv preprint arXiv:2301.04462*, 2023.

Andrzej Ruszczyński. Risk-averse dynamic programming for markov decision processes. *Mathematical programming*, 125:235–261, 2010.

A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):211–229, 1959.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

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

Tim Seyde, Wilko Schwarting, Igor Gilitschenski, Markus Wulfmeier, and Daniela Rus. Strength through diversity: Robust behavior learning via mixture policies. In *Conference on Robot Learning*, pp. 1144–1155. PMLR, 2022.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pp. 387–395. Pmlr, 2014.

Rahul Singh, Keuntaek Lee, and Yongxin Chen. Sample-based distributional policy gradient. *arXiv preprint arXiv:2001.02652*, 2020.

Matthew J Sobel. The variance of discounted markov decision processes. *Journal of Applied Probability*, 19(4):794–802, 1982.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Aviv Tamar, Dotan Di Castro, and Shie Mannor. Policy gradients with variance related risk criteria. *International Conference on Machine Learning*, 2012.

Aviv Tamar, Yonatan Glassner, and Shie Mannor. Optimizing the cvar via sampling. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Yichuan Charlie Tang, Jian Zhang, and Ruslan Salakhutdinov. Worst cases policy gradients. *arXiv preprint arXiv:1911.03618*, 2019.

Chen Tessler, Guy Tennenholtz, and Shie Mannor. Distributional policy optimization: An alternative approach for continuous control. *Advances in Neural Information Processing Systems (NeurIPS)*, 32:1352–1362, 2019.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

Núria Armengol Urpí, Sebastian Curi, and Andreas Krause. Risk-averse offline reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=TBIzh9b5eaz.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Ruodu Wang, Yunran Wei, and Gordon E Willmot. Characterization, robustness, and aggregation of signed choquet integrals. *Mathematics of Operations Research*, 45(3): 993–1015, 2020.

Xiao Wang, Hanna Krasowski, and Matthias Althoff. Commonroad-rl: A configurable reinforcement learning environment for motion planning of autonomous vehicles. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pp. 466–472. IEEE, 2021.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pp. 1995–2003. PMLR, 2016.

Jialong Wu, Haixu Wu, Zihan Qiu, Jianmin Wang, and Mingsheng Long. Supported policy optimization for offline reinforcement learning. *arXiv preprint arXiv:2202.06239*, 2022.

Markus Wulfmeier, Abbas Abdolmaleki, Roland Hafner, Jost Tobias Springenberg, Michael Neunert, Tim Hertweck, Thomas Lampe, Noah Siegel, Nicolas Heess, and Martin Riedmiller. Compositional transfer in hierarchical reinforcement learning. *Robotics: Science and Systems*, 2020.

Markus Wulfmeier, Dushyant Rao, Roland Hafner, Thomas Lampe, Abbas Abdolmaleki, Tim Hertweck, Michael Neunert, Dhruva Tirumala, Noah Siegel, Nicolas Heess, et al. Data-efficient hindsight off-policy option learning. In *International Conference on Machine Learning*, pp. 11340–11350. PMLR, 2021.

Chenjun Xiao, Han Wang, Yangchen Pan, Adam White, and Martha White. The in-sample softmax for offline reinforcement learning. *International Conference on Learning Representations*, 2023.

Tengyang Xie, Bo Liu, Yangyang Xu, Mohammad Ghavamzadeh, Yinlam Chow, Daoming Lyu, and Daesub Yoon. A block coordinate ascent algorithm for mean-variance optimization. *Advances in Neural Information Processing Systems*, 31, 2018.

Menahem E Yaari. The dual theory of choice under risk. *Econometrica: Journal of the Econometric Society*, pp. 95–115, 1987.

Derek Yang, Li Zhao, Zichuan Lin, Tao Qin, Jiang Bian, and Tieyan Liu. Fully parameterized quantile function for distributional reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 6190–6199, 2019.

Qisong Yang, Thiago D Simão, Simon H Tindemans, and Matthijs TJ Spaan. Wcsac: Worst-case soft actor critic for safety-constrained reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 10639–10646, 2021.

ChengYang Ying, Xinning Zhou, Hang Su, Dong Yan, Ning Chen, and Jun Zhu. Towards safe reinforcement learning via constraining conditional value-at-risk. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 3673–3680, 7 2022.

Shlomo Yitzhaki et al. Gini's mean difference: A superior measure of variability for non-normal distributions. *Metron*, 61(2):285–316, 2003.

Shi Yu, Haoran Wang, and Chaosheng Dong. Learning risk preferences from investment portfolios using inverse optimization. *Research in International Business and Finance*, 64:101879, 2023.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33:14129–14142, 2020.

Yuguang Yue, Zhendong Wang, and Mingyuan Zhou. Implicit distributional reinforcement learning. *Advances in Neural Information Processing Systems*, 33:7135–7147, 2020.

Shangtong Zhang and Shimon Whiteson. Dac: The double actor-critic architecture for learning options. *Advances in Neural Information Processing Systems*, 32, 2019.

Shangtong Zhang and Hengshuai Yao. Quota: The quantile option architecture for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pp. 5797–5804, 2019.

Shangtong Zhang, Bo Liu, and Shimon Whiteson. Mean-variance policy iteration for risk-averse reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 10905–10913, 2021.

Fan Zhou, Jianing Wang, and Xingdong Feng. Non-crossing quantile regression for distributional reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 15909–15919. Curran Associates, Inc., 2020.

Fan Zhou, Zhoufan Zhu, Qi Kuang, and Liwen Zhang. Non-decreasing quantile function network with efficient exploration for distributional reinforcement learning. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3455–3461. International Joint Conferences on Artificial Intelligence Organization, 2021.

APPENDICES

Appendix A

Supplementary for Chapter 3

A.1 Experiment Details

A.1.1 Windy Gridworld

Training hyperparameters We train SPL-DQN with NC-QR-DQN, NDQFN, and QR-DQN in the stochastic Windy Gridworld. The common parameter settings are summarized in Table A.1. The model inputs are the coordinates of the state. The ϵ-greedy parameter decreases by a half every two thousand episodes. To make the results comparable, we use the same Feature Extractor for the methods. For SPL-DQN and NDQFN, the number of bins is $K = 30$.

Additional results To further demonstrate the approximation ability of SPL compared with NDQFN, we train SPL and NDQFN with 10 random seeds in windy gridworld, and check the quantile functions learnt by these two methods for the states on the orange line trajectory (apart from the goal state). There are 15 states on the trajectory times 10 seeds, which yield 150 quantile functions. For each state, we can use the shortest path to determine an upper bound on the return for that state. Whenever a quantile function goes above that upper bound, we have a clear overestimation. For NDQFN, 36.7% (55/150) of the quantile functions yield an overestimation. In contrast, for SPL, only 4% (6/150) of the quantile functions yield an overestimation.

Table A.1: Common hyperparameters for SPL-DQN, NC-QR-DQN, NDQFN, and QR-DQN.

Hyperparameter	Value
Optimizer	Adam
Learning rate	0.001
Batch size	50
Discount factor (γ)	1
Initial ϵ-greedy	0.3
Minimal ϵ-greedy	0.01
Training episodes	30000
Sampling quantiles number	30
Feature Extractor hidden size	[20, 40, 80]

A.1.2 Cartpole

Training hyperparameters We train SPL-DQN with QR-DQN, IQN, FQF, NC-QR-DQN, MM-DQN, and NDQFN in the stochastic Cartpole. The common parameter settings are summarized in Table A.2. The model input is a vector of length 4, which contains the cart position, the cart velocity, the pole angle, and the pole angular velocity. The ϵ-greedy parameter decreases by 0.00005 every time step. To make the results comparable, we use the same Feature Extractor for the methods.

Specifically, for SPL-DQN and NDQFN, the number of bins is $K = 8$. For FQF, it contains a quantile fraction proposal network, whose learning rate is $2.5e^{-9}$, and the optimizer is RMSProp (Yang et al., 2019).

A.1.3 PyBulletGym

Training hyperparameters Hyperparameters for DDPG and DDPG based models are summarized in Table A.3. The critic also uses an L_2 weight decay of 10^{-2}. The soft target update coefficient is 0.001. Ornstein-Uhlenheck noise ($\mathcal{OU}(\mu', \sigma')$) (Uhlenbeck & Ornstein, 1930) is combined with actions for exploration in DDPG, where we use $\mu' = 0$ and $\sigma' = 0.1$.

Hyperparameters for SAC and SAC based models are summarized in Table A.4. The soft target update coefficient is 0.005.

For the critic implemented by SPL-DQN and NDQFN, the number of bins is $K = 32$. For the critic implemented by FQF, the learning rate for quantile fraction network is $2.5e^{-9}$,

Table A.2: Common hyperparameters across SPL-DQN, QR-DQN, IQN, FQF, NC-QR-DQN, MM-DQN, and NDQFN

Hyperparameter	Value
Optimizer	Adam
Learning rate	0.001
Batch size	32
Discount factor (γ)	0.99
Initial ϵ-greedy	0.3
Minimal ϵ-greedy	0.1
Training episodes	800
Sampling quantiles number (QR based methods)	8
Samples number (MM-DQN)	8
Feature Extractor hidden size	[128, 128]

and the corresponding optimizer is RMSProp.

A.2 DDPG and SAC with distributional critic

We summarize the algorithm when using distributional critic for DDPG and SAC in this section. For FQF critic, it has to update the quantile fraction proposal network separately, so it is individually described. Similarly, for MM critic, it uses moment matching instead of quantile regression, and it is individually described as well. For SAC based methods, readers can refer to the original paper for how to compute gradient for policy and state value.

Table A.3: Hyperparameters for DDPG and DDPG based methods

Hyperparameter	Value
Optimizer	Adam
Actor learning rate	10^{-4}
Critic learning rate	10^{-3}
Batch size	64
Discount factor (γ)	0.99
Training frames	one million
Sampling quantiles number	32
Actor hidden size	[400, 300]
Critic's Feature Extractor hidden size	[400, 300]

Table A.4: Hyperparameters for SAC and SAC based methods

Hyperparameter	Value
Optimizer	Adam
Actor learning rate	3×10^{-3}
Critic learning rate	3×10^{-3}
Entropy learning rate	3×10^{-3}
Batch size	64
Discount factor (γ)	0.99
Training frames	three million
Sampling quantiles number	32
Actor hidden size	[256, 256]
Critic's Feature Extractor hidden size	[256, 256]

Algorithm 4 DDPG with QR-based distributional critic (apart from FQF)

```
Require: Initialize critic network $Z(s, a|\psi)$ with weights $\psi$; Initialize actor network $\mu(s|\theta)$
    with weights $\theta$; Initialize target network $Z'$ and $\mu'$ with weights $\psi' \leftarrow \psi$, $\theta' \leftarrow \theta$;
    Initialize reply buffer $\mathcal{D}$
 1: for each episode do
 2:     Initialize random process $\mathcal{OU}$ for action exploration
 3:     for each time step t do
 4:         $a_t \sim \mu(s_t|\theta) + \mathcal{OU}_t$
 5:         $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
 6:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
 7:         Sample minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $\mathcal{D}$
 8:         Choose current quantile levels $\{\zeta\}$ according to critic's strategy
 9:         Compute corresponding current quantiles $\{q_i\} \leftarrow Z(s_i, a_i|\psi)$
10:         Choose target quantile levels $\{\bar{\zeta}\}$ according to critic's strategy
11:         Compute corresponding target quantiles $\{\bar{q}_{i+1}\} \leftarrow r_i + \gamma Z'(s_{i+1}, \mu'(s_{i+1}|\theta')|\psi')$
12:         Update critic by minimizing QR loss
13:         Compute expectation of quantiles $Q_{s,a} \leftarrow \mathbb{E}[Z(s, a|\psi)]$
14:         Update actor by
15:             $\nabla_\theta J \approx \frac{1}{N} \sum_i \nabla_a Q_{s,a}|_{s=s_i, a=\mu(s_i)} \nabla_\theta \mu(s|\mu)|_{s_i}$
16:         Update target networks
17:             $\psi' \leftarrow \sigma\psi + (1 - \sigma)\psi'$
18:             $\theta' \leftarrow \sigma\theta + (1 - \sigma)\theta'$
19:     end for
20: end for
```

Algorithm 5 DDPG with QR-based distributional critic (FQF)

```
Require: Initialize critic value network $Z(s,a|\psi)$ with weights $\psi$; Initialize actor network
    $\mu(s|\theta)$ with weights $\theta$; Initialize target network $Z'$ and $\mu'$ with weights $\psi' \leftarrow \psi$, $\theta' \leftarrow \theta$;
    Initialize reply buffer $\mathcal{D}$
    Initialize critic fraction proposal network $P(s,a|\omega)$ with weights $\omega$ and its target
    $P'(s,a|\omega')$ with weights $\omega'$
 1: for each episode do
 2:     Initialize random process $\mathcal{OU}$ for action exploration
 3:     for each time step t do
 4:         $a_t \sim \mu(s_t|\theta) + \mathcal{OU}_t$
 5:         $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
 6:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
 7:         Sample minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $\mathcal{D}$
 8:         Compute current quantile levels $\{\zeta\} \leftarrow P(s_i, a_i|\omega)$
 9:         Compute corresponding current quantiles $\{q_i\} \leftarrow Z(s_i, a_i|\psi)$
10:         Choose target quantile levels $\{\bar{\zeta}\} \leftarrow P(s_{i+1}, \mu'(s_{i+1}|\theta')|\omega)$
11:         Compute corresponding target quantiles $\{\bar{q}_{i+1}\} \leftarrow r_i + \gamma Z'(s_{i+1}, \mu'(s_{i+1}|\theta')|\psi')$
12:         Update critic fraction proposal network by
13:                 $\frac{\partial \omega}{\partial \tau_i} = 2Z[\tau_i] - Z[\hat{\tau}_i] - Z[\hat{\tau}_{i-1}]$, $\hat{\tau} = \frac{\tau_i + \tau_{i+1}}{2}$
14:         Update critic value network by minimizing QR loss
15:         Compute expectation of quantiles $Q_{s,a} \leftarrow \mathbb{E}[Z(s,a|\psi)]$
16:         Update actor by
17:                 $\nabla_\theta J \approx \frac{1}{N}\sum_i \nabla_a Q_{s,a}|_{s=s_i, a=\mu(s_i)} \nabla_\theta \mu(s|\mu)|_{s_i}$
18:         Update target networks
19:                 $\psi' \leftarrow \sigma\psi + (1-\sigma)\psi'$
20:                 $\theta' \leftarrow \sigma\theta + (1-\sigma)\theta'$
21:                 $\omega' \leftarrow \sigma\omega + (1-\sigma)\omega'$
22:     end for
23: end for
```

Algorithm 6 DDPG with MM distributional critic

```
Require: Initialize critic network $Z(s,a|\psi)$ with weights $\psi$; Initialize actor network $\mu(s|\theta)$
    with weights $\theta$; Initialize target network $Z'$ and $\mu'$ with weights $\psi' \leftarrow \psi$, $\theta' \leftarrow \theta$;
    Initialize reply buffer $\mathcal{D}$
 1: for each episode do
 2:   Initialize random process $\mathcal{OU}$ for action exploration
 3:   for each time step t do
 4:     $a_t \sim \mu(s_t|\theta) + \mathcal{OU}_t$
 5:     $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
 6:     $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
 7:     Sample minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $\mathcal{D}$
 8:     Compute current Q samples $\{q_i\} \leftarrow Z(s_i, a_i|\psi)$
 9:     Compute target Q samples $\{\bar{q}_{i+1}\} \leftarrow r_i + \gamma Z'(s_{i+1}, \mu'(s_{i+1}|\theta')|\psi')$
10:     Update critic by minimizing MMD loss
11:     Compute expectation of Q values $Q_{s,a} \leftarrow \mathbb{E}[Z(s,a|\psi)]$
12:     Update actor by
13:         $\nabla_\theta J \approx \frac{1}{N} \sum_i \nabla_a Q_{s,a}|_{s=s_i, a=\mu(s_i)} \nabla_\theta \mu(s|\mu)|_{s_i}$
14:     Update target networks
15:         $\psi' \leftarrow \sigma\psi + (1-\sigma)\psi'$
16:         $\theta' \leftarrow \sigma\theta + (1-\sigma)\theta'$
17:   end for
18: end for
```

Algorithm 7 SAC with QR-based distributional critic (apart from FQF)

```
Require: The learning rates $\lambda_\pi$, $\lambda_Z$, and $\lambda_V$ for functions $\pi_\theta$, $Z_w$, and $V_\psi$; Initialize
    parameters $\theta$, $w$, $\psi$, $\bar{\psi}$; Initialize reply buffer $\mathcal{D}$
 1: for each iteration do
 2:     for each time step t do
 3:         $a_t \sim \pi_\theta(s_t)$
 4:         $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
 5:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
 6:     end for
 7:     for each gradient update step do
 8:         Choose current and target quantile fractions according to critic's strategy
 9:         Compute current and target quantile values (for computing $J_Z$)
10:         $\psi \leftarrow \psi - \lambda_V \nabla_\psi J_V(\psi)$
11:         $w_i \leftarrow w_i - \lambda_Z \nabla_{w_i} J_Z(w_i)$ for $i \in \{1, 2\}$ ($J_Z$ is QR loss)
12:         $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta J_\pi(\theta)$
13:         $\bar{\psi} \leftarrow \sigma\psi + (1 - \sigma)\bar{\psi}$
14:     end for
15: end for
```

Algorithm 8 SAC with QR-based distributional critic (FQF)

```
Require: The learning rates $\lambda_\pi$, $\lambda_Z$, and $\lambda_V$ for functions $\pi_\theta$, $Z_w$, and $V_\psi$; Initialize
    parameters $\theta$, $w$, $\psi$, $\bar{\psi}$; Initialize reply buffer $\mathcal{D}$; The learning rate $\lambda_P$ for fraction
    proposal network $P_\phi$; Initialize parameter $\phi$
 1: for each iteration do
 2:     for each time step t do
 3:         $a_t \sim \pi_\theta(s_t)$
 4:         $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
 5:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
 6:     end for
 7:     for each gradient update step do
 8:         Compute current and target quantile fractions using $P_\phi$
 9:         Compute current and target quantile values (for computing $J_Z$)
10:         $\psi \leftarrow \psi - \lambda_V \nabla_\psi J_V(\psi)$
11:         $\phi \leftarrow \phi - \lambda_\phi \nabla_\phi J_P(\phi)$ ($J_P$ is quantile fraction loss)
12:         $w_i \leftarrow w_i - \lambda_Z \nabla_{w_i} J_Z(w_i)$ for $i \in \{1, 2\}$ ($J_Z$ is QR loss)
13:         $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta J_\pi(\theta)$
14:         $\bar{\psi} \leftarrow \sigma\psi + (1 - \sigma)\bar{\psi}$
15:     end for
16: end for
```

Algorithm 9 SAC with QR-based distributional critic (MM)

```
Require: The learning rates $\lambda_\pi$, $\lambda_Z$, and $\lambda_V$ for functions $\pi_\theta$, $Z_w$, and $V_\psi$; Initialize
    parameters $\theta$, $w$, $\psi$, $\bar{\psi}$; Initialize reply buffer $\mathcal{D}$
 1: for each iteration do
 2:     for each time step t do
 3:         $a_t \sim \pi_\theta(s_t)$
 4:         $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
 5:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
 6:     end for
 7:     for each gradient update step do
 8:         Compute current and target Q value samples (for computing $J_Z$)
 9:         $\psi \leftarrow \psi - \lambda_V \nabla_\psi J_V(\psi)$
10:         $w_i \leftarrow w_i - \lambda_Z \nabla_{w_i} J_Z(w_i)$ for $i \in \{1, 2\}$ ($J_Z$ is MMD loss)
11:         $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta J_\pi(\theta)$
12:         $\bar{\psi} \leftarrow \sigma\psi + (1 - \sigma)\bar{\psi}$
13:     end for
14: end for
```

Appendix B

Supplementary for Chapter 4

B.1 Convex Order, Gini Deviation, and Variance

Convex order describes dominance in terms of variability and is widely used in actuarial science.

Definition 2 (Gupta & Aziz (2010)) *Consider two random variables X and Y, X is called convex order smaller than Y, succinctly $X \leq_{\text{cx}} Y$, if $\mathbb{E}[\psi(X)] \leq \mathbb{E}[\psi(Y)]$, for all convex function $\psi()$, assuming that both expectations exist.*

In convex order $X \leq_{\text{cx}} Y$, Y is also called a mean-preserving spread of X (Rothschild & Stiglitz, 1978), which intuitively means that Y is more spread-out (and hence more random) than X. Thus, it is often desirable for a measure of variability to be monotone with respect to convex order (Furman et al., 2017). Both variance and GD, as a measure of variability, are consistent with convex order, i.e.,

- If $X \leq_{\text{cx}} Y$, then $\mathbb{V}[X] \leq \mathbb{V}[Y]$ for all $X, Y \in \mathcal{M}$
- If $X \leq_{\text{cx}} Y$, then $\mathbb{D}[X] \leq \mathbb{D}[Y]$ for all $X, Y \in \mathcal{M}$

Proof. It is immediate that $X \leq_{\text{cx}} Y$ implies $\mathbb{E}[X] = \mathbb{E}[Y]$. If we take convex function $\psi(x) = x^2$, we can get the order of variance $\mathbb{V}[X] \leq \mathbb{V}[Y]$. For the proof of GD, please refer to the following Lemma. Recall that GD can be expressed in the form of signed Choquet integral with a concave function h.

Lemma 3 (Wang et al. (2020),Theorem 2) *Convex order consistency of a signed Choquet integral is equivalent to its distortion function h being concave, i.e., $X \leq_{\text{cx}} Y$ if and only if the signed Choquet integral $\Phi_h(X) \leq \Phi_h(Y)$ for all concave functions $h \in \mathcal{H}$.*

B.2 Experiments Details

B.2.1 General Descriptions of Different Methods

Among the methods compared in this thesis, Tamar (Tamar et al., 2012) and MVP (Xie et al., 2018) are on-policy policy gradient methods. MVO and MG are policy gradient methods, but sample n trajectories and use IS to update. Non-tabular MVPI (Zhang et al., 2021) is an off-policy time-difference method.

Policy gradeint methods. MVO, Tamar, MVP, are originally derived based on VPG. Since VPG is known to have a poor performance in Mujoco, we also combined these mean-variance methods with PPO. Thus these mean-variance methods and our mean-GD method have different instantiation to maximize the expected return in different domains:

- With VPG: in Maze, LunarLander, InvertedPendulum.
- With PPO: in HalfCheetah, Swimmer.

When the risk-neutral policy gradient is VPG, for MVO and MG, it is REINFORCE with baseline; for Tamar and MVP, we strictly follow their papers to implement the algorithm, where no value function is used. MVO and MG collect n trajectories and use the IS strategy in Algorithm 1 to update policies. Tamar and MVP do not need IS, and update policies at the end of each episode.

When the risk-neutral policy gradient is PPO, we augment PPO with the variance or GD policy gradient from the original methods. MVO and MG collect n trajectories and use the IS strategy in Algorithm 2 to compute the gradient for the risk term. Tamar and MVP still update policies once at the end of each episode.

MVPI. We implemented three versions of MVPI in different domains:

- Tabular: in Maze, MVPI is a policy iteration method (Algorithm 1 in Zhang et al. (2021)).
- With DQN: in LunarLander, since this environment has discrete action space.

Table B.1: Model components in different methods

	Policy func	Value func	Additional training variables
MVO-VPG	√	√	
MVO-PPO	√	√	
Tamar-VPG	√	×	J,V (mean,variance)
Tamar-PPO	√	√	J,V (mean,variance)
MVP-VPG	√	×	y (dual variable)
MVP-PPO	√	√	y (dual variable)
MG-VPG	√	√	
MG-PPO	√	√	
MVPI-Q-Learning	×	√	
MVPI-DQN	×	√	
MVPI-TD3	√	√	

- With TD3: in InvertedPendulum, HalfCheetah, Swimmer, since these environments have continuous action space.

We summarize the components required in different methods in Table B.1.

B.2.2 Modified Guarded Maze Problem

The maze consists of a 6×6 grid. The agent can visit every free cell without a wall. The agent can take four actions (up, down, left, right). The maximum episode length is 100.

Policy function. For methods requiring a policy function, i.e., MVO, Tamar, MVP, MG, the policy is represented as

$$\pi_\theta(a|s) = \frac{e^{\varrho(s,a)\cdot\theta}}{\sum_b e^{\varrho(s,b)\cdot\theta}} \tag{B.1}$$

where $\phi(s,a)$ is the state-action feature vector. Here we use one-hot encoding to represent $\phi(s,a)$. Thus, the dimension of $\phi(s,a)$ is $6 \times 6 \times 4$. The derivative of the logarithm is

$$\nabla_\theta \log \pi_\theta(a|s) = \varrho(s,a) - \mathbb{E}_{b\sim\pi_\theta(\cdot|s)}\varrho(s,b) \tag{B.2}$$

Value function. For methods requiring a value function, i.e., REINFORCE baseline used in MVO and MG, and Q-learning in MVPI, the value function is represented as $V_\omega(s) = \varrho(s)\cdot\omega$ or $Q_\omega(s,a) = \varrho(s,a)\cdot\omega$. Similarly, $\varrho(s)$ is a one-hot encoding.

Optimizer. The policy and value loss are optimized by stochastic gradient descent (SGD).

Learning Parameters

We set discount factor $\gamma = 0.999$.

MVO: policy learning rate is 1e-5 $\in$\{5e-5, 1e-5, 5e-6\}, value function learning rate is 100 times policy learning rate. $\lambda = 1.0 \in$\{0.6, 0.8, 1.0, 1.2\}. Sample size $n = 50$. Maximum inner update number $M = 10$. IS ratio range $\delta = 0.5$. Inner termination ratio $\beta = 0.6$.

Tamar: policy learning rate is 1e-5 $\in$\{5e-5, 1e-5, 5e-6\}, J, V learning rate is 100 times policy learning rate. Threshold $b = 50 \in$\{10, 50, 100\}, $\lambda = 0.1 \in$\{0.1, 0.2, 0.4\}.

MVP: policy learning rate is 1e-5$\in$\{5e-5, 1e-5, 5e-6\}, y learning rate is the same. $\lambda = 0.1 \in$\{0.1, 0.2, 0.4\}.

MG: policy learning rate is 1e-4$\in$\{5e-4, 1e-4, 5e-5\}, value function learning rate is 100 times policy learning rate. $\lambda = 1.2 \in$\{0.8, 1.0, 1.2\}. Sample size $n = 50$. Maximum inner update number $M = 10$. IS ratio range $\delta = 0.5$. Inner termination ratio $\beta = 0.6$.

MVPI: Q function learning rate 5e-3$\in$\{5e-3, 1e-3, 5e-4\}, $\lambda = 0.2 \in$\{0.2, 0.4, 0.6\}.

Return Variance and Gini Deviation in Maze

We report the return's variance and GD during learning for different methods, as shown in Figure B.1 and B.2. Tamar Tamar et al. (2012) is unable to reach the goal in both settings. MVO fails to reach the goal when the return magnitude increases. MVP Xie et al. (2018)'s optimal risk-aversion rate is much lower than MG. MG can learn a risk averse policy in both settings with lower variance and GD, which suggests it is less sensitive to the return numerical scale.

B.2.3 LunarLander Discrete

The agent's goal is to land the lander on the ground without crashing. The state dimension is 8. The action dimension is 4. The detailed reward information is available at this webpage [1]. We divide the whole ground into left and right parts by the middle line of the

[1]https://www.gymlibrary.dev/environments/box2d/lunar_lander/

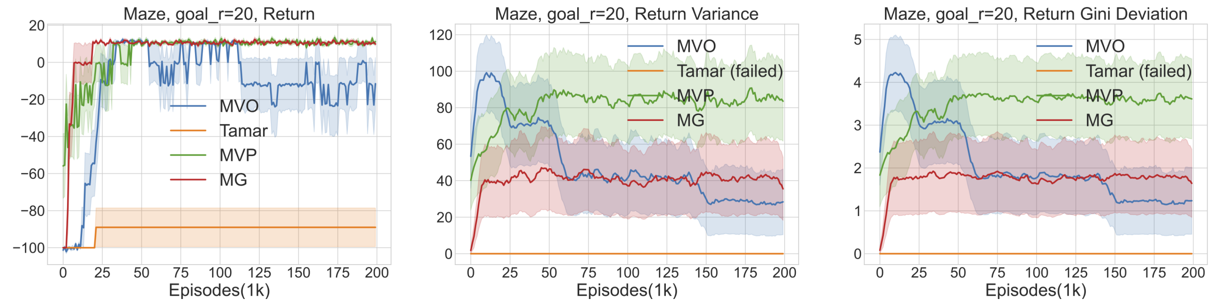


Figure B.1: Expected return, return variance and Gini deviation of different methods in Maze when goal reward is 20. Curves are averaged over 10 seeds with shaded regions indicating standard errors.

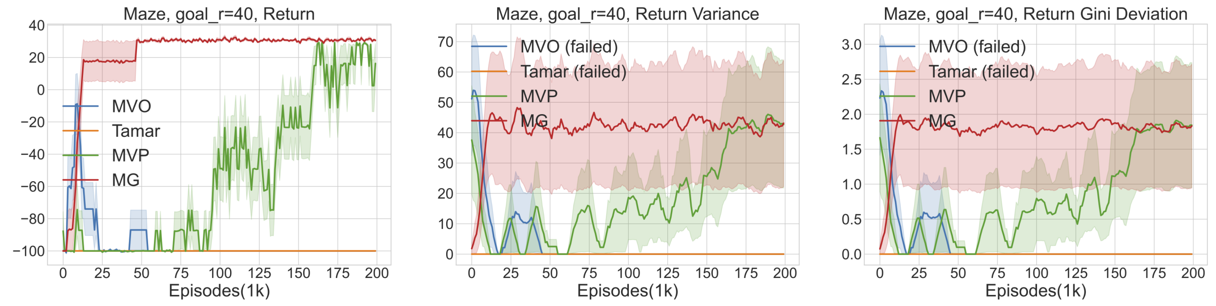


Figure B.2: Expected return, return variance and Gini deviation of different methods in Maze when goal reward is 40. Curves are averaged over 10 seeds with shaded regions indicating standard errors.

landing pad as shown in Figure 4.5. If the agent lands in the right part, an additional noisy reward signal sampled from $\mathcal{N}(0,1)$ times 90 is given. We set the maximum episode length to 1000. Note that the original reward for successfully landing is 100, thus the numerical scale of both return and reward is relatively large in this domain.

Policy function. The policy is a categorical distribution in REINFORCE, MVO, Tamar, MVP and MG, modeled as a neural network with two hidden layers. The hidden size is 128. Activation is ReLU. Softmax function is applied to the output to generate categorical probabilities.

Value function. The value function in REINFORCE, MVO, MG, and Q function in MVPI-DQN is a neural network with two hidden layers. The hidden size is 128. Activation is ReLU.

Optimizer. The optimizer for policy and value functions is Adam.

Learning Parameters

Discount factor is $\gamma = 0.999$

REINFORCE (with baseline): the policy learning rate is 7e-4 $\in$ {7e-4, 3e-4, 7e-5}, value function learning rate is 10 times policy learning rate.

MVO: policy learning rate is 7e-5 $\in$ {7e-4, 3e-4, 7e-5}, value function learning rate is 10 times policy learning rate. $\lambda = 0.4 \in \{0.4, 0.6, 0.8\}$. Sample size $n = 30$. Maximum inner update number $M = 10$. IS ratio range $\delta = 0.5$. Inner termination ratio $\beta = 0.6$.

Tamar: policy learning rate is 7e-5 $\in$ {7e-4, 3e-4, 7e-5}. J, V learning rate is 100 times the policy learning rate. Threshold $b = 50 \in \{10,50,100\}$. $\lambda = 0.2 \in \{0.2, 0.4, 0.6\}$.

MVP: policy learning rate is 7e-5 $\in$ {7e-4, 3e-4, 7e-5}. y learning rate is the same. $\lambda = 0.2 \in \{0.2, 0.4, 0.6\}$.

MG: policy learning rate is 7e-4 $\in$ {7e-4, 3e-4, 7e-5}, value function learning rate is 10 times policy learning rate. $\lambda = 0.6 \in \{0.4, 0.6, 0.8\}$. Sample size $n = 30$. Maximum inner update number $M = 10$. IS ratio range $\delta = 0.5$. Inner termination ratio $\beta = 0.6$.

MVPI: Q function learning rate is 7e-4 $\in$ {7e-4, 3e-4, 7e-5}, $\lambda = 0.2 \in \{0.2, 0.4, 0.6\}$. Batch size is 64.

Return Variance and Gini Deviation in LunarLander

The return's variance and GD of different methods during training is shown in Figure B.3. All the risk-averse methods, apart from ours, fail to learn a reasonable policy in this domain.

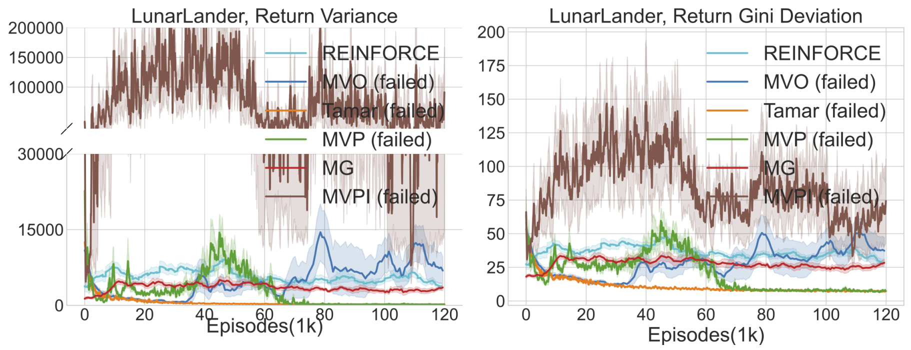


Figure B.3: Return variance and Gini deviation of different methods in LunarLander. Curves are averaged over 10 seeds with shaded regions indicating standard errors.

Our method achieves a comparable return, but with lower variance and GD compared with risk-neutral method.

B.2.4 InvertedPendulum

(The description of the Mujoco environments can be found at this webpage [2].)

The agent's goal is to balance a pole on a cart. The state dimension is 4 (X-position is already contained). The action dimension is 1. At each step, the environment provides a reward of 1. If the agent reaches the region X-coordinate > 0.01, an additional noisy reward signal sampled from $\mathcal{N}(0, 1)$ times 10 is given. To avoid the initial random speed forcing the agent to the X-coordinate > 0.01 region, we decrease the initial randomness for the speed from $U(-0.01, 0.01)$ to $U(-0.0005, 0.0005)$, where $U()$ represents the uniform distribution. The game ends if angle between the pole and the cart is greater than 0.2 radian or a maximum episode length 500 is reached.

Policy function. The policy is a normal distribution in REINFORCE, and VPG based methods (MVO, Tamar, MVP, and MG), modeled as a neural network with two hidden layers. The hidden size is 128. Activation is ReLU. Tanh is applied to the output to scale it to $(-1, 1)$. The output times the maximum absolute value of the action serves as the mean of the normal distribution. The logarithm of standard deviation is an independent trainable parameter.

The policy is a deterministic function in TD3 and MVPI, modeled as a neural network with two hidden layers. The hidden size is 128. Activation is ReLU. Tanh is applied to the

[2]https://www.gymlibrary.dev/environments/mujoco/

output to scale it to $(-1,1)$. The output times the maximum absolute value of the action is the true action executed in the environment.

Value function. The value function in REINFORCE, VPG based MVO, VPG based MG, TD3, and MVPI is a neural network with two hidden layers. The hidden size is 128. Activation is ReLU.

Optimizer. Optimizer for both policy and value function is Adam.

Learning Parameters

Discount factor $\gamma = 0.999$.

REINFORCE (with baseline): policy learning rate is 1e-4 $\in$ {1e-4, 5e-5, 5e-4}, value function learning rate is 10 times policy learning rate.

MVO: policy learning rate is 1e-5 $\in$ {1e-4, 5e-5, 1e-5}, value function learning rate is 10 times policy learning rate. $\lambda = 0.6 \in$ {0.2, 0.4, 0.6}. Sample size $n = 30$. Maximum inner update number $M = 10$. IS ratio range $\delta = 0.5$. Inner termination ratio $\beta = 0.6$.

Tamar: policy learning rate is 1e-5 $\in$ {1e-4, 5e-5, 1e-5}. J, V learning rate is 100 times policy learning rate. Threshold $b = 50 \in$ {10,50,100}. $\lambda = 0.2 \in$ {0.2, 0.4, 0.6}.

MVP: policy learning rate is 1e-5 $\in$ {1e-4, 5e-4, 1e-5}. y learning rate is the same. $\lambda = 0.2 \in$ {0.2, 0.4, 0.6}.

MG: policy learning rate is 1e-4 $\in$ {1e-4, 5e-5, 1e-4}, value function learning rate is 10 times policy learning rate. $\lambda = 1.0 \in$ {0.6, 1.0, 1.4}. Sample size $n = 30$. Maximum inner update number $M = 10$. IS ratio range $\delta = 0.5$. Inner termination ratio $\beta = 0.6$.

MVPI: Policy and value function learning rate 3e-4 $\in$ {3e-4, 7e-5, 1e-5}, $\lambda = 0.2 \in$ {0.2, 0.4, 0.6}. Batch size is 256.

TD3: Policy and value function learning rate 3e-4 $\in$ {3e-4, 7e-5, 1e-5}. Batch size is 256.

Return Variance and Gini Deviation in InvertedPendulum

The return variance and GD of different methods are shown in Figure B.4 and B.5.

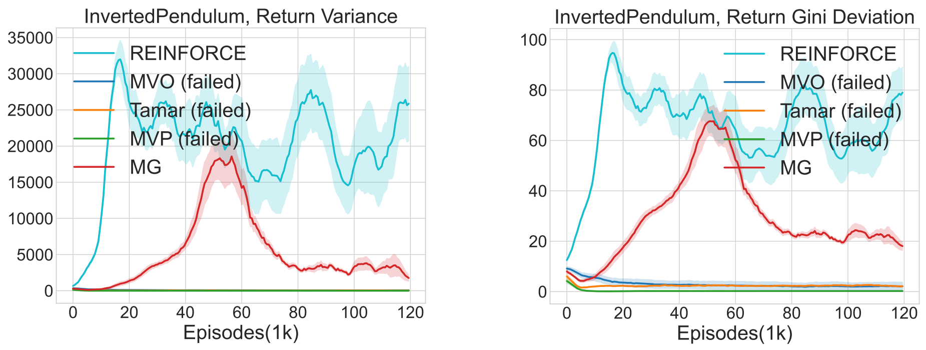


Figure B.4: Return variance and Gini deviation of policy gradient methods in InvertedPendulum. Curves are averaged over 10 seeds with shaded regions indicating standard errors.

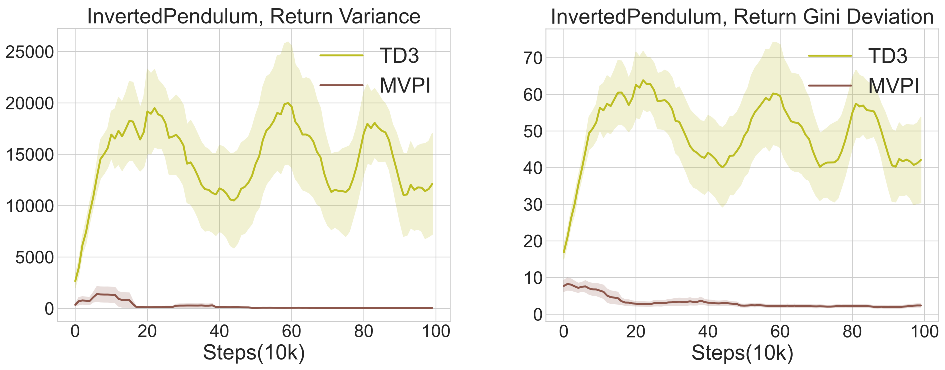


Figure B.5: Return variance and Gini deviation of TD3 and MVPI in InvertedPendulum. Curves are averaged over 10 seeds with shaded regions indicating standard errors.

B.2.5 HalfCheetah

The agent controls a robot with two legs. The state dimension is 18 (add X-position). The action dimension is 6. The reward is determined by the speed between the current and the previous time step and a penalty over the magnitude of the input action (Originally, only speed toward right is positive, we make the speed positive in both direction so that agent is free to move left or right). If the agent reaches the region X-coordinate < -3, an additional noisy reward signal sampled from $\mathcal{N}(0,1)$ times 10 is given. The game ends when a maximum episode length 500 is reached.

Policy function. The policy is a normal distribution in PPO, and PPO based methods (MVO, Tamar, MVP, and MG). The architecture is the same as in InvertedPendulum. Hidden size is 256.

The policy of TD3 and MVPI is the same as in InvertedPendulum. Hidden size is 256.

Value function. The value function in PPO, PPO based methods (MVO, Tamar, MVP, and MG), TD3 and MVPI is a neural network with two hidden layers. The hidden size is 256. Activation is ReLU.

Optimizer. Optimizer for policy and value is Adam.

Learning Parameters

Discount factor is $\gamma = 0.99$.

Common parameters of PPO and PPO based methods. GAE parameter: 0.95, Entropy coef: 0.01, Critic coef: 0.5, Clip ϵ: 0.2, Grad norm: 0.5.

PPO. policy learning rate 3e-4 $\in$ {3e-4, 7e-5, 1e-5}, value function learning rate is the same. Inner update number $M = 5$.

MVO. policy learning rate 7e-5 $\in$ {3e-4, 7e-5, 1e-5}, value function learning rate is the same. Sample size $n = 10$. Inner update number 5.

Tamar. policy learning rate 7e-5 $\in$ {3e-4, 7e-5, 1e-5}, value function learning rate is the same. J, V learning rate is 100 times policy learning rate. Threshold $b = 50 \in$ {10,50,100}. $\lambda = 0.2 \in$ {0.2, 0.4, 0.6}.

MVP. policy learning rate 7e-5 $\in$ {3e-4, 7e-5, 1e-5}, value function and y learning rate is the same. $\lambda = 0.2 \in$ {0.2, 0.4, 0.6}.

MG. policy learning rate 3e-4 $\in$ {3e-4, 7e-5, 1e-5}, value function learning rate is the same. Sample size $n = 10$. Inner update number $M = 5$.

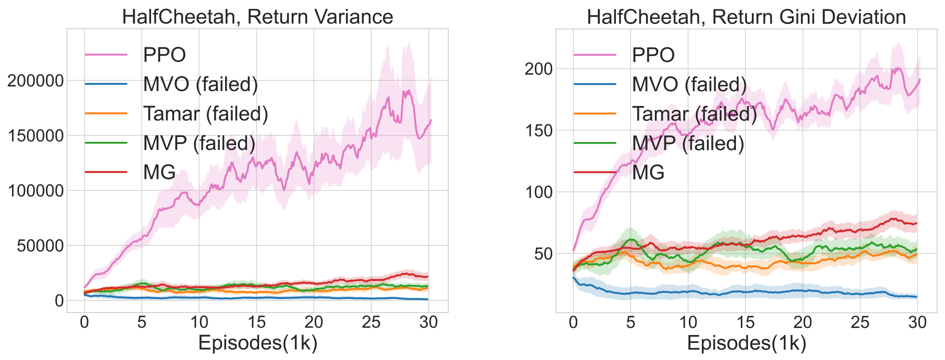


Figure B.6: Return variance and Gini deviation of on-policy methods in HalfCheetah. Curves are averaged over 10 seeds with shaded regions indicating standard errors

TD3. policy learning rate 3e-4 $\in$ {3e-4, 7e-5, 1e-5}, value function learning rate is the same. Batch size is 256.

MVPI. policy learning rate 3e-4 $\in$ {3e-4, 7e-5, 1e-5}, value function learning rate is the same. $\lambda = 0.2 \in \{0.2, 0.4, 0.6\}$. Batch size is 256.

Return Variance and Gini Deviation in HalfCheetah

See Figures B.6 and B.7.

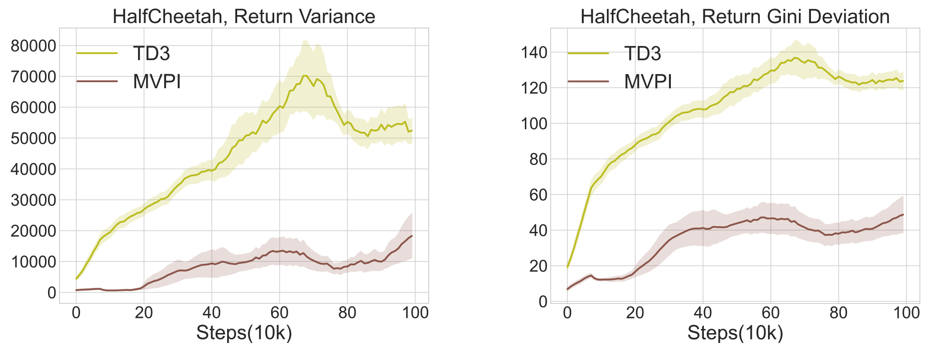


Figure B.7: Return variance and Gini deviation of off-policy methods in HalfCheetah. Curves are averaged over 10 seeds with shaded regions indicating standard errors

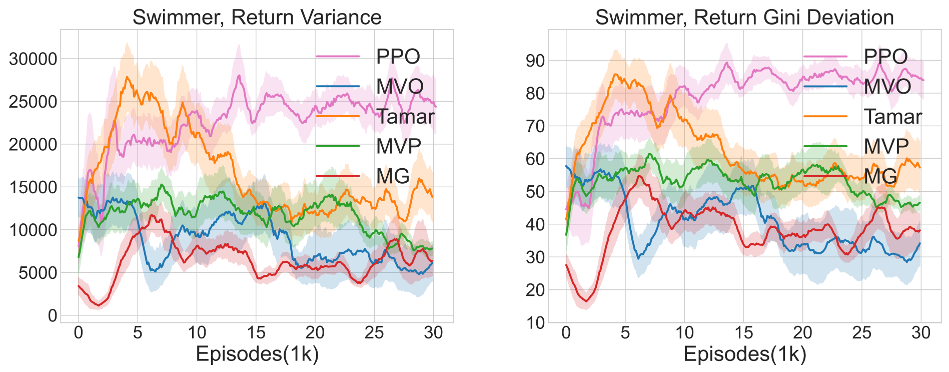


Figure B.8: Return variance and Gini deviation of on-policy methods in Swimmer. Curves are averaged over 10 seeds with shaded regions indicating standard errors

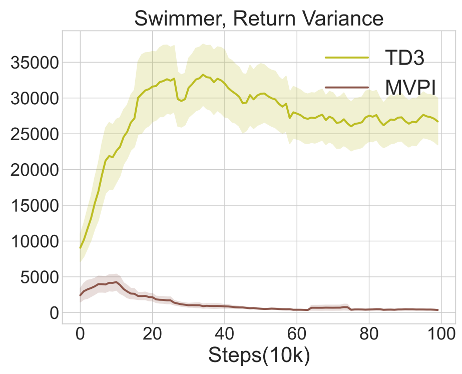
B.2.6 Swimmer

The agent controls a robot with two rotors (connecting three segments) and learns how to move. The state dimension is 10 (add XY-positions). The action dimension is 2. The reward is determined by the speed between the current and the previous time step and a penalty over the magnitude of the input action (Originally, only speed toward right is positive, we make the speed positive in both direction so that agent is free to move left or right). If agent reaches the region X-coordinate > 0.5, an additional noisy reward signal sampled from $\mathcal{N}(0, 1)$ times 10 is given. The game ends when a maximum episode length 500 is reached.

The neural network architectures and learning parameters are the same as in HalfCheetah.

Return Variance and Gini Deviation in Swimmer

See Figures B.8 and B.9.



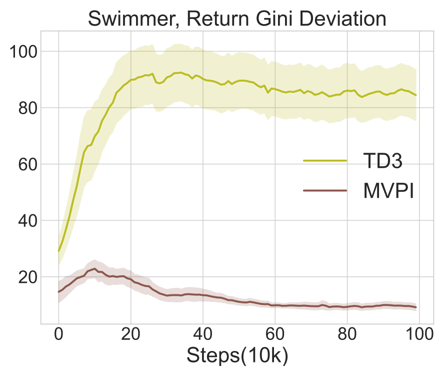


Figure B.9: Return variance and Gini deviation of off-policy methods in Swimmer. Curves are averaged over 10 seeds with shaded regions indicating standard errors

Appendix C

Supplementary for Chapter 5

C.1 Experiments Details

C.1.1 The Maze Problem

The maze consists of a 6×6 grid. The initial state of the agent is fixed at the bottom left corner. The action space is four (up, down, left, right). The maximum episode length is 100.

Policy function. For CVaR-PG, the policy is represented as

$$\pi_\theta(a|s) = \frac{e^{\varrho(s,a)\cdot\theta}}{\sum_b e^{\varrho(s,b)\cdot\theta}} \tag{C.1}$$

where $\varrho(s,a)$ is the state-action feature vector, basically a one-hot encoding in our implementation. Thus, the dimension of $\varrho(s,a)$ is $6 \times 6 \times 4$. The derivative of the logarithm is

$$\nabla_\theta \log \pi_\theta(a|s) = \varrho(s,a) - \mathbb{E}_{b\sim\pi_\theta(\cdot|s)}\varrho(s,b) \tag{C.2}$$

For our mixture policy, the policy parameter θ consists of two parts $\theta = (\theta_1, \theta_2)$, where θ_1 is for the adjustive policy π'_{θ_1}, and θ_2 is for the weight w.

$$\pi_\theta(a|s) = \sigma(\zeta(s,a)\cdot\theta_2)\pi'_{\theta_1}(a|s) + \Big(1 - \sigma(\zeta(s,a)\cdot\theta_2)\Big)\pi^n(a|s) \tag{C.3}$$

where $\sigma(\cdot)$ is the sigmoid function. The derivative of the logarithm is

$$\nabla_{\theta_1} \log \pi_\theta(a|s) = \frac{1}{\pi_\theta(a|s)} \sigma(\zeta(s,a) \cdot \theta_2) \pi'_{\theta_1}(a|s) \nabla_{\theta_1} \log \pi'_{\theta_1}(a|s) \tag{C.4}$$

$$\nabla_{\theta_2} \log \pi_\theta(a|s) = \frac{1}{\pi_\theta(a|s)} (\pi'_{\theta_1}(a|s) - \pi^n(a|s)) \sigma(\zeta(s,a) \cdot \theta_2) \Big(1 - \sigma(\zeta(s,a) \cdot \theta_2)\Big) \zeta(s,a) \tag{C.5}$$

Value function. The value function of REINFORCE baseline is represented as $V_v(s) = \zeta(s) \cdot v$. Similarly, $\zeta(s)$ is a one-hot encoding.

Learning Parameters

Discount factor $\gamma = 0.999$. Optimizer is stochastic gradient descent (SGD).

REINFORCE: Policy learning rate is 1e-2$\in$ {1e-2, 1e-3, 1e-4}. Value learning rate is 10 times policy learning rate.

CVaR-PG: Learning rate is 1e-2$\in$ {1e-1, 1e-2, 1e-3, 1e-4}.

MIX: Learning rate is 1e-2$\in$ {1e-2, 1e-3, 1e-4}.

C.1.2 LunarLander Discrete

The goal is to land the agent on the ground without crashing. The state dimension is 8. The action dimension is 4. The detailed reward information is available at this webpage [1]. Here, we split the ground into left and right parts by the middle line of the landing pad as shown in Figure 4.5. If the agent lands on the right part of the ground, an additional noisy reward signal $\mathcal{N}(0, 1) \times 100$ is given. The maximum episode length is 500.

Policy function. The policy is a categorical distribution in REINFORCE and CVaR-PG, modeled as a neutral network. Hidden layer: 2. Hidden size: 128. Activation: ReLU. Softmax function is applied to the output to generate categorical probabilities.

The policy of MIX is a weighted summation of π' and π^n with weight w. π' and w are modeled as a neutral network with two output heads. π^n is a separate neutral network. Both of them have: Hidden layer: 2. Hidden size: 128. Activation: ReLU.

[1]https://www.gymlibrary.dev/environments/box2d/lunar_lander/

Value function. For value function in REINFORCE baseline, Q and V function in IQL of MIX. Hidden layer: 2. Hidden size: 128. Activation: ReLU.

For distributional value function in DRL-mkv and DRL-lim. Hidden layer: 2. Hidden size: 128. Activation: ReLU. Quantile size (i.e., final layer output size): 80.

Learning Parameters

Discount factor $\gamma = 0.999$. Optimizer is Adam.

REINFORCE: Policy learning rate is 7e-4$\in$ {1e-3, 7e-4, 3e-4, 1e-4}. Value learning rate is 10 times policy learning rate.

CVaR-PG: Learning rate is 7e-4$\in$ {1e-3, 7e-4, 3e-4, 1e-4}.

MIX: Learning rate for π' and w is 7e-4$\in$ {1e-3, 7e-4, 3e-4, 1e-4}. Learning rate for IQL part (including policy and value functions) is 1e-4$\in$ {3e-4, 1e-4}. IQL update frequency $C = 50$, by sampling 2e5 transitions from buffer. $\eta = 0.8$ in Eq. 5.17. $\beta = 1$ in Eq. 5.18.

DRL-mkv: Learning rate is 7e-4$\in$ {1e-3, 7e-4, 3e-4, 1e-4, 7e-5}.

DRL-lim: Learning rate is 1e-4$\in$ {1e-3, 7e-4, 3e-4, 1e-4, 7e-5}.

C.1.3 InvertedPendulum

The description of the Mujoco environments can be found at this webpage [2].

The goal is to balance a inverted pendulum on a cart. The state dimension is 4 (X-position is already contained in the observation). The action dimension is 1. Per step reward is 1. If the agent reaches the region X-position ¿ 0.04, and additional noisy reward sampled from $\mathcal{N}(0, 1)$ times 10 is given. The game ends if angle between the pendulum and the cart is greater than 0.2 radian or a maximum episode length 300 is reached.

Policy function. The policy is a normal distribution in REINFORCE and CVaR-PG, modeled as a neutral network. Hidden layer: 2. Hidden size: 128. Activation: ReLU. Tanh is applied to the last layer. The logarithm of standard deviation is an independent trainable parameter.

For MIX, π' and w is a neutral network with two output heads. One for the mean of the normal distribution π', one for w. The logarithm of standard deviation is an independent

[2]https://www.gymlibrary.dev/environments/mujoco/

trainable parameter. π^n is a separate neutral network as above. Both of them have: Hidden layer: 2. Hidden size: 128. Activation: ReLU. Tanh is applied to the output of the distribution layer.

Value function. For value function in REINFORCE baseline, Q and V function in IQL of MIX. Hidden layer: 2. Hidden size: 128. Activation: ReLU.

For distributional value function in DRL-mkv and DRL-lim. Hidden layer: 2. Hidden size: 128. Activation: ReLU. Quantile size (i.e., final layer output size): 80.

Learning Parameters

Discount factor $\gamma = 0.999$. Optimizer is Adam.

REINFORCE: Policy learning rate is 3e-4$\in$ {7e-4, 3e-4, 1e-4}. Value learning rate is 10 times policy learning rate.

CVaR-PG: Learning rate is 3e-4$\in$ {7e-4, 3e-4, 1e-4}.

MIX: Learning rate for π' and w is 3e-4$\in$ {7e-4, 3e-4, 1e-4}. Learning rate for IQL part (including policy and value functions) is 1e-4$\in$ {3e-4, 1e-4}. IQL update frequency $C = 50$, by sampling 1e5 transitions from buffer. $\eta = 0.9$ in Eq. 5.17. $\beta = 2$ in Eq. 5.18.

DRL-mkv: Learning rate is 7e-4$\in$ {1e-3, 7e-4, 3e-4, 1e-4, 7e-5}.

DRL-lim: Learning rate is 1e-3$\in$ {1e-3, 7e-4, 3e-4, 1e-4, 7e-5}.

C.1.4 HalfCheetah

The agent controls a robot with two legs. The state dimension is 18 (add X-position). The action dimension is 6. One part of the reward is determined by the distance covered between the current and the previous time step. Originally, it is positive only when the agent moves toward the forward (right) direction. To encourage the agent to freely move forward (left) and backward (right), we modify this part of the reward to make it positive as long as the agent is moving far from the origin. If the agent reaches the region X-position <-3, an additional noisy reward sampled from $\mathcal{N}(0, 1)$ times 50 is given. The game ends when a maximum episode length 500 is reached.

Policy function. Hidden size: 256. Others are the same as the case in InvertedPendulum.

Value function. Hidden size: 256. Others are the same as the case in InvertedPendulum.

Learning Parameters

Discount factor $\gamma = 0.99$. Optimizer is Adam.

SAC: Learning rate is 3e-4$\in$ {7e-4, 3e-4, 1e-4}.

CVaR-PG: Learning rate is 3e-4$\in$ {7e-4, 3e-4, 1e-4}.

MIX: Learning rate for π' and w is 3e-4$\in$ {7e-4, 3e-4, 1e-4}. Learning rate for IQL part (including policy and value functions) is the same. IQL update frequency $C = 30$, by sampling 2e5 transitions from buffer. $\eta = 0.8$ in Eq. 5.17. $\beta = 2$ in Eq. 5.18.

DRL-mkv: Learning rate is 3e-4$\in$ {7e-4, 3e-4, 1e-4, 7e-5}.

DRL-lim: Learning rate is 1e-4$\in$ {7e-4, 3e-4, 1e-4, 7e-5}.

C.1.5 Ant

The agent controls a robot with four legs attached to it with each leg having two links. The state dimension is 113 (add X-position). The action dimension is 8. Similar to HalfCheetah, we modify the reward to make the distance based reward positive as long as the agent is moving far from the origin. If the agent reaches the region X-position <-3, an additional noisy reward sampled from $\mathcal{N}(0, 1)$ times 50 is given. The game ends when a maximum episode length 500 is reached.

Policy function. Hidden size: 256. Others are the same as the case in InvertedPendulum.

Value function. Hidden size: 256. Others are the same as the case in InvertedPendulum.

Learning Parameters

Discount factor $\gamma = 0.99$. Optimizer is Adam.

SAC: Learning rate is 3e-4$\in$ {7e-4, 3e-4, 1e-4}.

CVaR-PG: Learning rate is 3e-4$\in$ {7e-4, 3e-4, 1e-4}.

MIX: Learning rate for π' and w is 3e-4$\in$ {7e-4, 3e-4, 1e-4}. Learning rate for IQL part (including policy and value functions) is the same. IQL update frequency $C = 30$, by sampling 2e5 transitions from buffer. $\eta = 0.8$ in Eq. 5.17. $\beta = 2$ in Eq. 5.18.

DRL-mkv: Learning rate is 7e-5$\in$ {7e-4, 3e-4, 1e-4, 7e-5}.

DRL-lim: Learning rate is 7e-5$\in$ {7e-4, 3e-4, 1e-4, 7e-5}.

C.1.6 Driving Game

The goal of this game is to control the agent's car to follow the leader car without colliding. The state dimension is 5. The action dimension is 5. We refer reader to Sec. 5.2 of (Greenberg et al., 2022) for more details.

Policy function. Policy of CeSoR and CVaR-PG: Hidden size: 32. Hidden layer: 2. Activation: Tanh.

Policy of MIX: Hidden size: 32. Activation: Tanh. Others are the same as the case in InvertedPendulum.

Value function. Q and V of IQL: Hidden size: 32. Others are the same as the case in InvertedPendulum.

Learning Parameters

CVaR $\alpha = 0.05$. Update policy after gathering $N = 80$ trajectories. The starting value for CVaR α is 0.8.

CVaR-PG: Learning rate 1e-2$\in$ {2e-2, 1e-2, 5e-3}.

CeSoR: Learning rate 1e-2$\in$ {2e-2, 1e-2, 5e-3}.

MIX: Learning rate for π' and w is 1e-2$\in$ {2e-2, 1e-2, 5e-3}. Learning rate for IQL part (including policy and value function) is 5e-3. IQL update frequency $C = 50$, by sample 2e4 transitions from buffer. $\eta = 0.8$ in Eq. 5.17. $\beta = 2$ in Eq. 5.18.

MIX+SoR: Learning parameters are the same as MIX.

MIX+CeSoR: Learning parameters are the same as MIX.