Mean-Variance Policy Iteration for Risk-Averse Reinforcement Learning

Shangtong Zhang, Bo Liu, Shimon Whiteson

1 University of Oxford
2 Auburn University

shangtong.zhang@cs.ox.ac.uk, bo.liu@auburn.edu, shimon.whiteson@cs.ox.ac.uk

Abstract

We present a mean-variance policy iteration (MVPI) framework for risk-averse control in a discounted infinite horizon MDP optimizing the variance of a per-step reward random variable. MVPI enjoys great flexibility in that any policy evaluation method and risk-neutral control method can be dropped in for risk-averse control off the shelf, in both on- and off-policy settings. This flexibility reduces the gap between risk-neutral control and risk-averse control and is achieved by working on a novel augmented MDP directly. We propose risk-averse TD3 as an example instantiating MVPI, which outperforms vanilla TD3 and many previous risk-averse control methods in challenging Mujoco robot simulation tasks under a risk-aware performance metric. This risk-averse TD3 is the first to introduce deterministic policies and off-policy learning into risk-averse reinforcement learning, both of which are key to the performance boost we show in Mujoco domains.

Introduction

One fundamental task in reinforcement learning (RL, Sutton and Barto 2018) is control, in which we seek a policy that maximizes certain performance metrics. In risk-neutral RL, the performance metric is usually the expectation of some random variable, for example, the expected total (discounted or undiscounted) reward (Puterman 2014; Sutton and Barto 2018). We, however, sometimes want to minimize certain risk measures of that random variable while maximizing its expectation. For example, a portfolio manager usually wants to reduce the risk of a portfolio while maximizing its return. Risk-averse RL is a framework for studying such problems.

Although many real-world applications can potentially benefit from risk-averse RL, e.g., pricing (Wang 2000), healthcare (Parker 2009), portfolio management (Lai et al. 2011), autonomous driving (Maurer et al. 2016), and robotics (Majumdar and Pavone 2020), the development of risk-averse RL largely falls behind risk-neutral RL. Risk-neutral RL methods have enjoyed superhuman performance in many domains, e.g., Go (Silver et al. 2016), protein design (Senior, Jumper, and Hassabis 2018), and StarCraft II (Vinyals et al. 2019), while no human-level performance has been reported for risk-averse RL methods in real-world applications. Risk-neutral RL methods have enjoyed stable off-policy learning (Watkins and Dayan 1992; Mnih et al. 2015), while state-of-the-art risk-averse RL methods, e.g., Xie et al. (2018), still require on-policy samples. Risk-neutral RL methods have exploited deep neural network function approximators and distributed training (Mnih et al. 2016; Espeholt et al. 2018), while tabular and linear methods still dominate the experiments of risk-averse RL literature (Tamar, Di Castro, and Mannor 2012; Prashanth and Ghavamzadeh 2013; Xie et al. 2018). Such a big gap between risk-averse RL and risk-neutral RL gives rise to a natural question: can we design a meta algorithm that can easily leverage recent advances in risk-neutral RL for risk-averse RL? In this paper, we give an affirmative answer via the mean-variance policy iteration (MVPI) framework.

Although many risk measures have been used in risk-averse RL, in this paper, we mainly focus on variance (Sobel 1982; Mannor and Tsitsiklis 2011; Tamar, Di Castro, and Mannor 2012; Prashanth and Ghavamzadeh 2013; Xie et al. 2018) given its advantages in interpretability and computation (Markowitz and Todd 2000; Li and Ng 2000). Such an RL paradigm is usually referred to as mean-variance RL, and previous mean-variance RL methods usually consider the variance of the total reward random variable (Tamar, Di Castro, and Mannor 2012; Prashanth and Ghavamzadeh 2013; Xie et al. 2018). Recently, Bisi et al. (2020) propose a reward-volatility risk measure that considers the variance of a per-step reward random variable. Bisi et al. (2020) show that the variance of the per-step reward can better capture the short-term risk than the variance of the total reward and usually leads to smoother trajectories.

In complicated environments with function approximation, pursuing the exact policy that minimizes the variance of the total reward is usually intractable. In practice, all we can hope is to lower the variance of the total reward to a certain level. As the variance of the per-step reward bounds the variance of the total reward from above (Bisi et al. 2020), in this paper, we propose to optimize the variance of the per-step reward as a proxy for optimizing the variance of the total reward. Though the policy minimizing the variance of the per-step reward does not necessarily minimize the variance of the total reward, in this paper, we show, via MVPI, that...
optimizing the variance of the per-step reward as a proxy is more efficient and scalable than optimizing the variance of the total reward directly.

MVPI enjoys great flexibility in that any policy evaluation method and risk-neutral control method can be dropped in for risk-averse control off the shelf, in both on- and off-policy settings. Key to the flexibility of MVPI is that it works on an augmented MDP directly, which we make possible by introducing the Fenchel duality and block cyclic coordinate ascent to solve a policy-dependent reward issue (Bisi et al. 2020). This issue refers to a requirement to solve an MDP whose reward function depends on the policy being followed, i.e., the reward function of this MDP is nonstationary. Consequently, standard tools from the MDP literature are not applicable. We propose risk-averse TD3 as an example instantiating MVPI, which outperforms vanilla TD3 (Fujimoto, van Hoof, and Meger 2018) and many previous mean-variance RL methods (Tamar, Di Castro, and Mannor 2012; Prashanth and Ghavamzadeh 2013; Xie et al. 2018; Bisi et al. 2020) in challenging Mujoco robot simulation tasks in terms of a risk-aware performance metric. To the best of our knowledge, we are the first to benchmark mean-variance RL methods in Mujoco domains, a widely used benchmark for robotic-oriented RL research, and the first to bring off-policy learning and deterministic policies into mean-variance RL.

**Mean-Variance RL**

We consider an infinite horizon MDP with a state space $S$, an action space $A$, a bounded reward function $r : S \times A \rightarrow \mathbb{R}$, a transition kernel $p : S \times S \times A \rightarrow [0, 1]$, an initial distribution $\mu_0 : S \rightarrow [0, 1]$, and a discount factor $\gamma \in (0, 1]$. The initial state $S_0$ is sampled from $\mu_0$. At time step $t$, an agent takes an action $A_t$ according to $\pi(\cdot|S_t)$, where $\pi : A \times S \rightarrow [0, 1]$ is the policy followed by the agent. The agent then gets a reward $R_{t+1} = r(S_t, A_t)$ and proceeds to the next state $S_{t+1}$ according to $p(\cdot|S_t, A_t)$. In this paper, we consider a deterministic reward setting for the ease of presentation, following Chow (2017); Xie et al. (2018). The return at time step $t$ is defined as $G_t = \sum_{i=t}^{\infty} \gamma^i r(S_{i+t}, A_{i+t})$. When $\gamma < 1$, $G_t$ is always well defined. When $\gamma = 1$, to ensure $G_t$ remains well defined, it is usually assumed that all polices are proper (Bertsekas and Tsitsiklis 1996), i.e., for any policy $\pi$, the chain induced by $\pi$ has some absorbing states, one of which the agent will eventually go to with probability 1. Furthermore, the rewards are always 0 thereafter. For any $\gamma \in (0, 1]$, $G_0$ is the random variable indicating the total reward, and we use its expectation $J(\pi) = \mathbb{E}_{\mu_0, \pi, \gamma}[G_0]$, as our primary performance metric. In particular, when $\gamma = 1$, we can express $G_0$ as $G_0 = \sum_{t=0}^{T-1} r(S_t, A_t)$, where $T$ is a random variable indicating the first time the agent goes to an absorbing state. For any $\gamma \in [0, 1]$, the state value function and the state-action value function are defined as $v_\pi(s) = \mathbb{E}[G_t|S_t = s]$ and $q_\pi(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a]$ respectively.

**Total Reward Perspective.** Previous mean-variance RL methods (Prashanth and Ghavamzadeh 2013; Tamar, Di Castro, and Mannor 2012; Xie et al. 2018) usually consider the variance of the total reward. Namely, they consider the following problem:

$$
\max_\theta \mathbb{E}[G_0] \quad \text{subject to} \quad \mathbb{V}(G_0) \leq \xi, \quad (1)
$$

where $\mathbb{V}(\cdot)$ indicates the variance of a random variable, $\xi$ indicates the user’s tolerance for variance, and $\pi$ is parameterized by $\theta$. In particular, Prashanth and Ghavamzadeh (2013) consider the setting $\gamma < 1$ and convert (1) into an unconstrained saddle-point problem: $\max_\theta \min_\lambda \mathbb{E}[G_0] + \lambda(\mathbb{V}(G_0) - \xi)$, where $\lambda$ is the dual variable. (Prashanth and Ghavamzadeh 2013) use stochastic gradient descent to find the saddle-point of $L_1(\theta, \lambda)$. To estimate $\nabla_{\theta, \lambda} L_1(\theta, \lambda)$, they propose two simultaneous perturbation methods: simultaneous perturbation stochastic approximation and smoothed functional (Bhatnagar, Prasad, and Prashanth 2013), yielding a three-timescale algorithm. Empirical success is observed in a simple traffic control MDP.

Tamar, Di Castro, and Mannor (2012) consider the setting $\gamma = 1$. Instead of using the saddle-point formulation in Prashanth and Ghavamzadeh (2013), they consider the following unconstrained problem: $\max_\theta \mathbb{E}[G_0] - \lambda(\mathbb{V}(G_0) - \xi)$, where $\lambda > 0$ is a hyperparameter to be tuned and $g(\cdot)$ is a penalty function, which they define as $g(x) = (\max(0, x))^2$. The analytical expression of $\nabla_{\theta} L_2(\theta)$ they provide involves a term $\mathbb{E}[G_0] \nabla_{\theta} \mathbb{E}[G_0]$. To estimate this term, Tamar, Di Castro, and Mannor (2012) consider a two-timescale algorithm and keep running estimates for $\mathbb{E}[G_0]$ and $\mathbb{V}(G_0)$ in a faster timescale, yielding an episodic algorithm. Empirical success is observed in a simple portfolio management MDP.

Xie et al. (2018) consider the setting $\gamma = 1$ and set the penalty function $g(\cdot)$ in Tamar, Di Castro, and Mannor (2012) to the identity function. With the Fenchel duality $x^* = \max_y (2xy - y^2)$, they transform the original problem into $\max_{\theta, \lambda} L_3(\theta, \lambda) = 2g(\mathbb{E}[G_0] + \lambda^2) - y^2 - \mathbb{E}[G_0]^2$, where $y$ is the dual variable. Xie et al. (2018) then propose a solver based on stochastic coordinate ascent, yielding an episodic algorithm.

**Per-Step Reward Perspective.** Recently Bisi et al. (2020) propose a reward-volatility risk measure, which is the variance of a per-step reward random variable $R$. In the setting $\gamma < 1$, it is well known that the expected total discounted reward can be expressed as

$$
J(\pi) = \frac{1}{1-\gamma} \sum_{s,a} d_\pi(s,a) r(s,a),
$$

where $d_\pi(s,a)$ is the normalized discounted state-action distribution:

$$
d_\pi(s,a) = (1-\gamma) \sum_{t=0}^{\infty} \gamma^t \Pr(S_t = s, A_t = a|\mu_0, \pi, p).
$$

We now define the per-step reward random variable $R$, a discrete random variable taking values in the image of $r$, by defining its probability mass function as $p(R = x) = \sum_{s,a} d_\pi(s,a) I_r(s,a)=x$, where $I$ is the indicator function. It follows that $\mathbb{E}[R] = (1-\gamma)J(\pi)$. Bisi et al. (2020) argue that $\mathbb{V}(R)$ can better capture short-term risk than $\mathbb{V}(G_0)$ and
optimizing $V(R)$ usually leads to smoother trajectories than optimizing $V(G_0)$, among other advantages of this risk measure. Bisi et al. (2020), therefore, consider the following objective:

$$J_\lambda(\pi) \equiv \mathbb{E}[R] - \lambda V(R).$$  \hfill (2)

Bisi et al. (2020) show that $J_\lambda(\pi) = \mathbb{E}[R - \lambda(R - \mathbb{E}[R])^2]$, i.e., to optimize the risk-aware objective $J_\lambda(\pi)$ is to optimize the canonical risk-neutral objective of a new MDP, which is the same as the original MDP except that the new reward function is

$$r'(s, a) \equiv r(s, a) - \lambda (r(s, a) - (1 - \gamma)J(\pi))^2.$$  

Unfortunately, this new reward function $r'$ depends on the policy $\pi$ due to the occurrence of $J(\pi)$, implying the reward function is actually nonstationary. By contrast, we refer to this problem as the policy-dependent-reward issue. Due to this issue, the rich classical MDP toolbox cannot be applied to this new MDP easily, and the approach of Bisi et al. (2020) does not and cannot work on this new MDP directly.

Bisi et al. (2020) instead work on the objective Eq (2) directly without resorting to the augmented MDP. They propose to optimize a performance lower bound of $J_\lambda(\pi)$ by extending the performance difference theorem (Theorem 1 in Sch alman et al. (2015)) from the risk-neutral objective $J(\pi)$ to the risk-aware objective $J_\lambda(\pi)$, yielding the Trust Region Volatility Optimization (TRVO) algorithm, which is similar to Trust Region Policy Optimization (Sch alman et al. 2015).

Importantly, Bisi et al. (2020) show that $V(G_0) \leq \frac{V(R)}{1 - \gamma}$, indicating that minimizing the variance of $R$ implicitly minimizes the variance of $G_0$. We, therefore, can optimize $V(R)$ as a proxy (upper bound) for optimizing $V(G_0)$. In this paper, we argue that $V(R)$ is easier to optimize than $V(G_0)$. The methods of Tamar, Di Castro, and Mannor (2012), Xie et al. (2018) optimizing $V(G_0)$ involve terms like $E[G_0]^2$ and $E[G_0]$, which lead to terms like $G_0^2 \sum_{t=0}^{T-1} \nabla \log \pi(A_t | S_t)$ in their update rules, yielding large variance. In particular, it is computationally prohibitive to further expand $G_0^2$ explicitly to apply variance reduction techniques like baselines (Williams 1992). By contrast, we show in the next section that by considering $V(R)$, MVPI involves only $r(s, a)^2$, which is easier to deal with than $G_0^2$.

**Mean-Variance Policy Iteration**

Although in many problems our goal is to maximize the expected total undiscounted reward, practitioners often find that optimizing the discounted objective ($\gamma < 1$) as a proxy for the undiscounted objective ($\gamma = 1$) is better than optimizing the undiscounted objective directly, especially when deep neural networks are used as function approximators (Minh et al. 2015; Lillicrap et al. 2015; Espeholt et al. 2018; Xu, van Hasselt, and Silver 2018; Van Seijen, Fatemi, and Tavakoli 2019). We, therefore, focus on the discounted problem, which allows us to consider optimizing the variance of the per-step reward as a proxy (upper bound) for optimizing the variance of the total reward.

To address the policy-dependent reward issue, we use the Fenchel duality to rewrite $J_\lambda(\pi)$ as

$$J_\lambda(\pi) = \mathbb{E}[R] - \lambda \mathbb{E}[R^2] + \lambda \mathbb{E}[R]$$ \hfill (3)

yielding the following problem:

$$\max_{\pi, y} J_\lambda(\pi, y) \equiv \sum_{s,a} d_\pi(s, a) (r(s, a) - \lambda r(s, a)^2 + 2\lambda r(s, a)y) - \lambda y^2.$$  

We then propose a block cyclic coordinate ascent (BCCA, Luenerber and Ye [1984] Tseng 2001; Saha and Tewari 2010, 2013; Wright 2015) framework to solve (4), which updates $y$ and $\pi$ alternatively as shown in Algorithm 1.

**Algorithm 1:** Mean-Variance Policy Iteration (MVPI)

for $k = 1, \ldots, do$

Step 1: $y_{k+1} \doteq (1 - \gamma)J(\pi_k)$ // The exact solution for $\arg \max_y J_\lambda(\pi_k, y)$

Step 2: $\pi_{k+1} \doteq \arg \max_\pi \left( \sum_{s,a} d_\pi(s, a) \left(r(s, a) - \lambda r(s, a)^2 + 2\lambda r(s, a)y_{k+1}\right) \right)$

end

At the $k$-th iteration, we first fix $\pi_k$ and update $y_{k+1}$ (Step 1). As $J_\lambda(\pi_k, y)$ is quadratic in $y$, $y_{k+1}$ can be computed analytically as $y_{k+1} = \sum_{s,a} d_\pi(s, a)r(s, a) = (1 - \gamma)J(\pi_k)$, i.e., all we need in this step is $J(\pi_k)$, which is exactly the performance metric of the policy $\pi_k$. We, therefore, refer to Step 1 as policy evaluation. We then fix $y_{k+1}$ and update $\pi_{k+1}$ (Step 2). Remarkably, Step 2 can be reduced to the following problem:

$$\pi_{k+1} = \arg \max_\pi \sum_{s,a} d_\pi(s, a) \hat{r}(s, a; y_{k+1}),$$

where $\hat{r}(s, a; y) \doteq r(s, a) - \lambda r(s, a)^2 + 2\lambda r(s, a)y$. In other words, to compute $\pi_{k+1}$, we need to solve a new MDP, which is the same as the original MDP except that the reward function is $\hat{r}$ instead of $r$. This new reward function $\hat{r}$ does not depend on the policy $\pi$, avoiding the policy-dependent-reward issue of Bisi et al. (2020). In this step, a new policy $\pi_{k+1}$ is computed. An intuitive conjecture is that this step is a policy improvement step, and we confirm this with the following proposition:

**Proposition 1.** (Monotonic Policy Improvement)

$\forall k$, $J_\lambda(\pi_{k+1}) \geq J_\lambda(\pi_k)$.

Though the monotonic improvement w.r.t. the objective $J_\lambda(\pi, y)$ in Eq (4) follows directly from standard BCCA theories, Theorem 1 provides the monotonic improvement w.r.t. the objective $J_\lambda(\pi)$ in Eq (3). The proof is provided
in the appendix. Given Theorem 1, we can now consider the whole BCCA framework in Algorithm 1 as a policy iteration framework, which we call mean-variance policy iteration (MVPI). Let \( \{\pi_\theta : \theta \in \Theta\} \) be the function class for policy optimization, we have

**Assumption 1.** \( \{\theta \in \Theta, y \in \mathbb{R} \mid J_\lambda(\theta, y) \geq J_\lambda(\theta_0)\} \) is compact, where \( \theta_0 \) is the initial parameters.

**Assumption 2.** \( \sup_{\theta \in \Theta} \left| \frac{\partial \log \pi(a|s)}{\partial y} \right| < \infty, \sup_{\theta \in \Theta} \left| \sum \log \pi_\theta(a|s) \right| < \infty, \Theta \) is open and bounded.

**Proposition 2.** (Convergence of MVPI with function approximation) Under Assumptions 1 & 2 let

\[
y_{k+1} = \arg \max_y J_\lambda(\theta_k, y), \\
\theta_{k+1} = \arg \max_{\theta \in \Theta} J_\lambda(\theta, y_{k+1}), \quad k = 0, 1, \ldots
\]

then \( J_\lambda(\theta_{k+1}) \geq J_\lambda(\theta_k) \). \( \{J_\lambda(\theta_k)\}_{k=1,...} \) converges, and \( \lim_{k \to \infty} \left| \sum \log \pi_\theta(a|s) \right| = 0 \).

**Remark 1.** Assumption 1 is standard in BCCA literature (e.g., Theorem 4.1 in Tseng (2001)). Assumption 2 is standard in policy optimization literature (e.g., Assumption 4.1 in Papini et al. (2018)). Convergence in the form of \( \lim \inf \) also appears in other literature (e.g., Luenberger and Ye (1984); Tseng (2001); Konda (2002); Zhang et al. (2020b)).

The proof is provided in the appendix. MVPI enjoys great flexibility in that any policy evaluation method and risk-neutral control method can be dropped in off the shelf, which makes it possible to leverage all the advances in risk-neutral RL. MVPI differs from the standard policy iteration (PI, e.g., see Bertsekas and Tsitsiklis (1996); Puterman (2014); Sutton and Barto (2018)) in two key ways: (1) policy evaluation in MVPI requires only a scalar performance metric, while standard policy evaluation involves computing the value of all states. (2) policy improvement in MVPI considers an augmented reward \( \tilde{r} \), which is different at each iteration, while standard policy improvement always considers the original reward. Standard PI can be used to solve the policy improvement step in MVPI.

### Average Reward Setting

So far we have considered the total reward as the primary performance metric for mean-variance RL. We now show that MVPI can also be used when we consider the average reward as the primary performance metric. Assuming the chain induced by \( \pi \) is ergodic and letting \( \overline{d}_s(s) \) be its stationary distribution, Filar, Kallenberg, and Lee (1989); Prashanth and Ghavamzadeh (2013) consider the long-run variance risk measure

\[
\bar{\Lambda}(\pi) = \sum_{s,a} \overline{d}_s(s, a)(r(s, a) - \bar{J}(\pi))^2
\]

for the average reward setting, where \( \overline{d}_s(s, a) \equiv \overline{d}_s(s) \pi(a|s) \) and \( \bar{J}(\pi) = \sum_{s,a} \overline{d}_s(s, a)r(s, a) \) is the average reward. We now define a risk-aware objective

\[
\overline{J}_\lambda(\pi) = \bar{J}(\pi) - \lambda \bar{\Lambda}(\pi)
\]

where we have used the Fenchel duality and BCCA can take over to derive MVPI for the average reward setting as Algorithm 1. It is not a coincidence that the only difference between (4) and (5) is the difference between \( \tilde{d}_s(s, a) \) and \( \overline{d}_s(s, a) \). The root cause is that the total discounted reward of an MDP is always equivalent to the average reward of an artificial MDP (up to a constant multiplier), whose transition kernel is \( \bar{p}(s'|s, a) = \gamma p(s'|s, a) + (1 - \gamma)\mu_0(s') \) (e.g., see Section 2.4 in Konda (2002) for details).

### Off-Policy Learning

Off-policy learning has played a key role in improving data efficiency (Lin (1992); Mnih et al. (2015) and exploration (Osband et al. 2016; Osband, Aslanides, and Cassirer 2018) in risk-neutral control algorithms. Previous mean-variance RL methods, however, consider only the on-policy setting and cannot be easily made off-policy. For example, it is not clear whether perturbation methods for estimating gradients (Prashanth and Ghavamzadeh 2013) can be used off-policy. To reweight terms like \( G_s(\pi(\cdot|s)) \) from Tamar, Di Castro, and Mannor (2012), Xie et al. (2018) in the off-policy setting, we would need to compute the product of importance sampling ratios \( \Pi_{i=k}^{T-1} \pi(a_i|s_i) / \mu(a_i|s_i) \), where \( \mu \) is the behavior policy. This product usually suffers from high variance (Precup, Sutton, and Dasgupta 2001; Liu et al. 2018) and requires knowing the behavior policy \( \mu \), both of which are practical obstacles in real applications. By contrast, as MVPI works on an augmented MDP directly, any risk-neutral off-policy learning technique can be used for risk-averse off-policy control directly. In this paper, we consider MVPI in both on-line and off-line off-policy settings.

### On-line setting

In the on-line off-policy setting, an agent interacts with the environment following a behavior policy \( \mu \) to collect transitions, which are stored into a replay buffer (Lin (1992) for future reuse. Mujoco robot simulation tasks (Brockman et al. 2016) are common benchmarks for this paradigm (Lillicrap et al. 2015; Haarnoja et al. 2018), and TD3 is a leading algorithm in Mujoco tasks. TD3 is a risk-neutral control algorithm, reducing the over-estimation bias (Hasselt 2010) of DDPG (Lillicrap et al. 2015), which is a neural network implementation of the deterministic policy gradient theorem (Silver et al. 2014). Given the empirical success of TD3, we propose MVPI-TD3 for risk-averse control in this setting. In the policy evaluation step of MVPI-TD3, we set \( y_{k+1} \) to the average of the recent \( K \) rewards, where \( K \) is a hyperparameter to be tuned and we have assumed the policy changes slowly. Theoretically, we should use a weighted average as \( \overline{d}_s(s, a) \) is a discounted distribution. Though implementing this weighted average is straightforward, practitioners usually ignore discounting for state visitation in policy gradient methods to improve sample efficiency (Mnih et al. 2016; Schulman et al. 2015; 2017; Bacon, Harb, and Precup 2017). Hence, we do not use the weighted average in MVPI-TD3. In the policy improvement step of MVPI-TD3, we sample a mini-batch of transitions from the replay buffer and perform one TD3 gradient update. The pseudocode of MVPI-TD3 is provided in the appendix.

### Off-line setting

In the off-line off-policy setting, we are
presented with a batch of transitions \{s_i, a_i, r_i, s'_i\}_{i=1,...,K} and want to learn a good target policy \(\pi\) for control solely from this batch of transitions. Sometimes those transitions are generated by following a known behavior policy \(\mu\). But more commonly, those transitions are generated from multiple unknown behavior policies, which we refer to as the behavior-agnostic off-policy setting (Nachum et al. 2019a). Namely, the state-action pairs \(\{s_i, a_i\}\) are distributed according to some unknown distribution \(d\), which may result from multiple unknown behavior policies. The successor state \(s'_i\) is distributed according to \(\rho^*(\cdot|s_i, a_i)\) and \(r_i = r(s, a)\). The degree of off-policyness in this setting is usually larger than the on-line off-policy setting.

In the off-line off-policy setting, the policy evaluation step in MVPI becomes the standard off-policy evaluation problem (OPE). Thomas, Theocharous, and Ghavamzadeh (2015), Thomas and Brunskill (2016), Jiang and Li (2016), Liu et al. (2018), where we want to estimate a scalar performance metric of a policy with off-line samples. One promising approach to OPE is density ratio learning, where we use function approximation to learn the density ratio \(d_{\pi_\theta|\pi_\theta}(s, a)\) directly, which we then use to reweight \(r(s, a)\). All off-policy evaluation algorithms can be integrated into MVPI in a plug-and-play manner. In the off-line off-policy setting, the policy improvement step in MVPI becomes the standard off-policy policy optimization problem, where we can reweight the canonical on-policy actor-critic (Sutton et al. 2000; Konda 2002) with the density ratio as in Liu et al. (2019) to achieve off-policy policy optimization. Algorithm 2 provides an example of Off-line MVPI.

In the on-line off-policy learning setting, the behavior policy and the target policy are usually closely correlated (e.g., in MVPI-TD3), we, therefore, do not need to learn the density ratio. In the off-line off-policy learning setting, the dataset may come from behavior policies that are arbitrarily different from the target policy. We, therefore, resort to density ratios for account to this discrepancy. Density ratio learning itself is an active research area and is out of scope of this paper. See Hallak and Mannor (2017), Liu et al. (2018), Gelada and Bellemare (2019), Nachum et al. (2019a), Zhang et al. (2020a), Zhang, Liu, and Whiteson (2020), Mousavi et al. (2020) for more details about density ratio learning.

**Experiments**

All curves in this section are averaged over 10 independent runs with shaded regions indicate standard errors. All implementations are publicly available\footnote{https://github.com/ShangtongZhang/DeepRL}.

**On-line learning setting.** In many real-world robot applications, e.g., in a warehouse, it is crucial that the robots’ performance be consistent. In such cases, risk-averse RL is an appealing option to train robots. Motivated by this, we benchmark MVPI-TD3 on eight Mujoco robot manipulation tasks from OpenAI gym. Though Mujoco tasks have a stochastic initial state distribution, they are usually equipped with a deterministic transition kernel. To make them more suitable for investigating risk-averse control algorithms, we add a Gaussian noise \(N(0, 0.1^2)\) to every action. As we are not aware of any other off-policy mean-variance RL method, we use several recent on-policy mean-variance RL method as baselines, namely, the methods of Tamar, Di Castro, and Mannor (2012); Prashanth and Ghavamzadeh (2013), MVP (Xie et al. 2018), and TRVO (Bisi et al. 2020). The methods of Tamar, Di Castro, and Mannor (2012); Prashanth and Ghavamzadeh (2013) and MVP are not designed for deep RL settings. To make the comparison fair, we improve those baselines with parallelized actors to stabilize the training of neural networks as in Mnih et al. (2016)\footnote{They are on-policy algorithms so we cannot use experience replay.}.

**Algorithm 2: Off-line MVPI**

**Input:** A batch of transitions \(\{s_i, a_i, r_i, s'_i\}_{i=1,...,K}\), a learning rate \(\alpha\), and the number of policy updates \(T\)

**while True do**

- Learn the density ratio \(\rho_\pi(s, a) \equiv \frac{d_\pi(s, a)}{d_\pi(s', a)}\) with \(\{s_i, a_i, r_i, s'_i\}_{i=1,...,K}\) // For example, use GradientDICE
  - \(\text{Zhang, Liu, and Whiteson (2020)}\)
  - \(y \leftarrow \frac{1}{K} \sum_{i=1}^{K} \rho_\pi(s_i, a_i) r_i\)
  - for \(i = 1, \ldots, K\) do
    - \(r_i \leftarrow r_i - \lambda \tau^2 + 2\lambda r_i y\)
    - \(a'_i \sim \pi(\cdot|s'_i)\)
  - end
  - for \(j = 1, \ldots, T\) do
    - Learn \(q_\pi(s, a)\) with \(\{s_i, a_i, r_i, s'_i\}_{i=1,...,K}\)
    - // For example, use TD(0)
      - \(\text{Sutton (1998)}\) in \(S \times A\)
      - Learn \(\rho_\pi(s, a)\) with \(\{s_i, a_i, r_i, s'_i\}_{i=1,...,K}\)
    - \(\theta \leftarrow \theta + \alpha \rho_\pi(s_i, a_i) \nabla q \log \pi(a_i|s_i) q_{\pi}(s_i, a_i)\)
      - where \(i\) is randomly selected
  - end

end
eter tuning is conducted, including $\lambda$ and $\xi$. This means they fail to achieve the risk-performance trade-off in our tested domains, indicating that those methods are not able to scale up to Mujoco domains with neural network function approximation. Those flat curves suggest that perturbation-based gradient estimation in [Prashanth and Ghavamzadeh (2013)] may not work well with neural networks, and the $G_0^T \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a|s) \text{ term in Tamar, Di Castro, and Mannor (2012)}$ and MVP may suffer from high variance, yielding instability. By contrast, the two algorithms from the per-step reward perspective (MVPI-TD3 and TRVO) do learn a reasonable policy, which experimentally supports our argument that optimizing the variance of the per-step reward is more efficient and scalable than optimizing the variance of the total reward.

As shown in Figure 1 the vanilla risk-neutral TD3 outperforms all risk-averse algorithms (in terms of episodic return). This is expected as it is in general hard for a risk-averse algorithm to outperform its risk-neutral counterpart in terms of a risk-neutral performance metric. We now compare TD3, MVPI-TD3 and TRVO in terms of a risk-aware performance metric. To this end, we test the agent at the end of training for an extra 100 episodes to compute a risk-aware performance metric. We report the normalized statistics in Table 1. The results show that MVPI-TD3 outperforms TD3 in 6 out of 8 tasks in terms of the risk-aware performance metric. Moreover, MVPI-TD3 outperforms TRVO in 7 out of 8 tasks. We also compare the algorithms in terms of the sharp ratio (SR, [Sharpe 1966]). Although none of the algorithms optimizes SR directly, MVPI-TD3 outperforms TD3 and TRVO in 6 and 7 tasks respectively in terms of SR. This performance boost of MVPI-TD3 over TRVO indeed results from the performance boost of TD3 over PPO, and it is the flexibility of MVPI that makes this off-the-shelf application TD3 in risk-averse RL possible. We also provide versions of Figure 1 and Table 1 with $\lambda = 0.5$ and $\lambda = 2$ in the appendix. The relative performance is the same as $\lambda = 1$. Though we propose to optimize the variance of the per-step reward as a proxy for optimizing the variance of the total reward, we also provide in the appendix curves comparing MVPI-TD3 and TRVO w.r.t. risk measures consisting of both mean and variance of the per-step reward. With those risk measures, MVPI-TD3 generally outperforms TRVO as well.

| Algorithm   | $J_{\text{TRVO}}$ | $J_{\text{TRVO}}$ | $J_{\text{TRVO}}$ | $J_{\text{TRVO}}$ | $J_{\text{TRVO}}$ | $J_{\text{TRVO}}$ | $J_{\text{TRVO}}$ |
|-------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| InvertedP   | -321%             | -1%               | 3 · 10%           | -100%             | 0%                | 0%                | 0%                |
| InvertedD.P.| -367%             | -17%              | 365%              | -62%              | 42%               | 1%                | -41%              |
| HalfCheetah | 92%               | -87%              | -92%              | -54%              | 99%               | -44%              | -98%              |
| Walker2d    | -1%               | -61%              | 0%                | -61%              | 88%               | -55%              | -88%              |
| Swimmer     | -10%              | -7%               | 18%               | -14%              | 4%                | -2%               | -53%              |
| Hopper      | 57%               | -25%              | -57%              | 13%               | 72%               | -3%               | -71%              |
| Reacher     | -48%              | -12%              | 102%              | 21%               | -1%               | -1%               | 0%                |
| Ant         | 97%               | -85%              | -97%              | -18%              | 96%               | -53%              | -96%              | |

| Algorithm   | $J_{\text{MVPI}}$ | $J_{\text{MVPI}}$ | $J_{\text{MVPI}}$ | $J_{\text{MVPI}}$ | $J_{\text{MVPI}}$ | $J_{\text{MVPI}}$ |
|-------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| InvertedP   | -321%             | -1%               | 3 · 10%           | -100%             | 0%                | 0%                |
| InvertedD.P.| -367%             | -17%              | 365%              | -62%              | 42%               | 1%                |
| HalfCheetah | 92%               | -87%              | -92%              | -54%              | 99%               | -44%              |
| Walker2d    | -1%               | -61%              | 0%                | -61%              | 88%               | -55%              |
| Swimmer     | -10%              | -7%               | 18%               | -14%              | 4%                | -2%               |
| Hopper      | 57%               | -25%              | -57%              | 13%               | 72%               | -3%               |
| Reacher     | -48%              | -12%              | 102%              | 21%               | -1%               | -1%               |
| Ant         | 97%               | -85%              | -97%              | -18%              | 96%               | -53%              | |

Table 1: Normalized statistics of TRVO and MVPI-TD3. MVPI is shorthand for MVPI-TD3 in this table. For algo $\in \{\text{MVPI-TD3, TRVO, TD3}\}$, we compute the risk-aware performance metric as $J_{\text{algo}} = \text{mean}_{\text{algo}} - \lambda \text{variance}_{\text{algo}}$ with $\lambda = 1$, where $\text{mean}_{\text{algo}}$ and $\text{variance}_{\text{algo}}$ are mean and variance of the 100 evaluation episodic returns. Then we compute the normalized statistics as $\Delta_{\text{algo}} = \frac{J_{\text{algo}} - J_{\text{TD3}}}{J_{\text{TD3}}}$, $\Delta_{\text{mean}} = \frac{\text{mean}_{\text{algo}} - \text{mean}_{\text{TD3}}}{\text{mean}_{\text{TD3}}}$, $\Delta_{\text{variance}} = \frac{\text{variance}_{\text{algo}} - \text{variance}_{\text{TD3}}}{\text{variance}_{\text{TD3}}}$, $\Delta_{\text{SR}} = \frac{\text{SR}_{\text{algo}} - \text{SR}_{\text{TD3}}}{\text{SR}_{\text{TD3}}}$.

Both MVPI-TD3 and TRVO are trained with $\lambda = 1$. All $J_{\text{algo}}$ are averaged over 10 independent runs.

**Related Work**

Both MVPI and [Bisi et al. 2020] consider the per-step reward perspective for mean-variance RL. In this work, we mainly use the variance of the per-step reward as a proxy (upper bound) for optimizing the variance of the total reward. Though TRVO in [Bisi et al. 2020] is the same as instantiating MVPI with TRPO, the derivation is dramatically different. In particular, it is not clear whether the performance-lower-bound-based derivation for TRVO can be adopted to deterministic policies, off-policy learning, or other policy optimization paradigms, and this is not explored in [Bisi et al. 2020]. By contrast, MVPI is compatible with
any existing risk-neutral policy optimization technique. Furthermore, MVPI works for both the total discounted reward setting and the average reward setting. It is not clear how the performance lower bound in Bisi et al. (2020), which plays a central role in TRVO, can be adapted to the average reward setting. All the advantages of MVPI over TRVO result from addressing the policy-dependent-reward issue in Bisi et al. (2020).

While the application of Fenchel duality in RL is not new, previously it has been used only to address double sampling issues (e.g., Liu et al. (2015); Dai et al. (2017); Xie et al. (2018); Nachum et al. (2019a)). By contrast, we use Fenchel duality together with BCAA to address the policy-dependent-reward issue in Bisi et al. (2020) and derive a policy iteration framework that appears to be novel to the RL community.

Besides variance, value at risk (VaR, Chow et al. 2018), conditional value at risk (CVaR, Chow and Ghavamzadeh 2014; Tamar, Glassner, and Mannor 2015; Chow et al. 2018), sharp ratio (Tamar, Di Castro, and Mannor 2012), and exponential utility (Howard and Matheson 1972; Borkar 2002) are also used for risk-averse RL. In particular, it is straightforward to consider exponential utility for the per-step reward, which, however, suffers from the same problems as the exponential utility for the total reward, e.g., it overflows easily (Gosavi, Das, and Murray 2014).

**Conclusion**

In this paper, we propose MVPI for risk-averse RL. MVPI enjoys great flexibility such that any policy evaluation method and risk-neutral control method can be dropped in for risk-averse control off the shelf, in both on- and off-policy settings. This flexibility dramatically reduces the gap between risk-neutral control and risk-averse control. To the best of our knowledge, MVPI is the first empirical success of risk-averse RL in Mujoco robot simulation domains, and is also the first success of off-policy risk-averse RL and risk-averse RL with deterministic policies. Deterministic policies play an important role in reducing the variance of a policy (Silver et al. 2014). Off-policy learning is important for improving data efficiency (Mnih et al. 2015) and exploration (Osband, Aslanides, and Cassirer 2018). Incorporating those two elements in risk-averse RL appears novel and is key to the observed performance improvement.

Possibilities for future work include considering other risk measures (e.g., VaR and CVaR) of the per-step reward random variable, integrating more advanced off-policy policy optimization techniques (e.g., Nachum et al. 2019b) in off-
policy MVPI, optimizing $\lambda$ with meta-gradients (Xu, van Hasselt, and Silver 2018), analyzing the sample complexity of MVPI, and developing theory for approximate MVPI.

References

Bacon, P.-L.; Harb, J.; and Precup, D. 2017. The option-critic architecture. In Proceedings of the 31st AAAI Conference on Artificial Intelligence.

Bertsekas, D. 1995. Nonlinear Programming. Athena Scientific.

Bertsekas, D. P.; and Tsitsiklis, J. N. 1996. Neuro-Dynamic Programming. Athena Scientific Belmont, MA.

Bhatnagar, S.; Prasad, H.; and Prashanth, L. 2013. Stochastic Recursive Algorithms for Optimization. Springer London.

Bisio, L.; Sabbioni, L.; Vittori, E.; Papini, M.; and Restelli, M. 2020. Risk-Averse Trust Region Optimization for Reward-Volatility Reduction. In Proceedings of the 29th International Joint Conference on Artificial Intelligence. Special Track on AI in FinTech.

Borkar, V. S. 2002. Q-learning for risk-sensitive control. Mathematics of Operations Research.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI gym. arXiv preprint arXiv:1606.01540.

Chow, Y. 2017. Risk-Sensitive and Data-Driven Sequential Decision Making. Ph.D. thesis, Stanford University.

Chow, Y.; and Ghavamzadeh, M. 2014. Algorithms for CVaR optimization in MDPs. In Advances in Neural Information Processing Systems.

Chow, Y.; Ghavamzadeh, M.; Janson, L.; and Pavone, M. 2018. Risk-constrained reinforcement learning with percentile risk criteria. The Journal of Machine Learning Research.

Dai, B.; Shaw, A.; Li, L.; Xiao, L.; He, N.; Liu, Z.; Chen, J.; and Song, L. 2017. SBEED: Convergent reinforcement learning with nonlinear function approximation. arXiv preprint arXiv:1712.10285.

Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; Wu, Y.; and Zhokhov, P. 2017. OpenAI Baselines. https://github.com/openai/baselines.

Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; et al. 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. arXiv preprint arXiv:1802.01561.

Filar, J. A.; Kallenberg, L. C.; and Lee, H.-M. 1989. Variance-penalized Markov decision processes. Mathematics of Operations Research.

Fujimoto, S.; van Hoof, H.; and Meger, D. 2018. Addressing function approximation error in actor-critic methods. arXiv preprint arXiv:1802.09477.

Gelada, C.; and Bellemare, M. G. 2019. Off-Policy Deep Reinforcement Learning by Bootstrapping the Covariate Shift. In Proceedings of the 33rd AAAI Conference on Artificial Intelligence.

Gosavi, A. A.; Das, S. K.; and Murray, S. L. 2014. Beyond exponential utility functions: A variance-adjusted approach for risk-averse reinforcement learning. In 2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint arXiv:1801.01290.

Hallak, A.; and Mannor, S. 2017. Consistent on-line off-policy evaluation. In Proceedings of the 34th International Conference on Machine Learning.

Hasselt, H. V. 2010. Double Q-learning. In Advances in neural information processing systems.

Howard, R. A.; and Matheson, J. E. 1972. Risk-sensitive Markov decision processes. Management Science.

Jiang, N.; and Li, L. 2016. Doubly robust off-policy value evaluation for reinforcement learning. In International Conference on Machine Learning.

Konda, V. R. 2002. Actor-critic algorithms. Ph.D. thesis, Massachusetts Institute of Technology.

Lai, T. L.; Xing, H.; Chen, Z.; et al. 2011. Mean-variance portfolio optimization when means and covariances are unknown. The Annals of Applied Statistics.

Li, D.; and Ng, W.-L. 2000. Optimal dynamic portfolio selection: Multi-period mean-variance formulation. Mathematical finance.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

Lin, L.-J. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine Learning.

Liu, B.; Liu, J.; Ghavamzadeh, M.; Mahadevan, S.; and Petrik, M. 2015. Finite-Sample Analysis of Proximal Gradient TD Algorithms. In Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence.

Liu, Q.; Li, L.; Tang, Z.; and Zhou, D. 2018. Breaking the curse of horizon: Infinite-horizon off-policy estimation. In Advances in Neural Information Processing Systems.

Liu, Y.; Swaminathan, A.; Agarwal, A.; and Brunskill, E. 2019. Off-Policy Policy Gradient with State Distribution Correction. arXiv preprint arXiv:1904.08473.

Luersenberger, D. G.; and Ye, Y. 1984. Linear and nonlinear programming (3rd Edition). Springer.

Maei, H. R. 2011. Gradient temporal-difference learning algorithms. Ph.D. thesis, University of Alberta.

Majumdar, A.; and Pavone, M. 2020. How should a robot assess risk? Towards an axiomatic theory of risk in robotics. In Robotics Research. Springer.

Mannor, S.; and Tsitsiklis, J. 2011. Mean-variance optimization in Markov decision processes. arXiv preprint arXiv:1104.5601.

Markowitz, H. M.; and Todd, G. P. 2000. Mean-variance analysis in portfolio choice and capital markets. John Wiley & Sons.

Maurer, M.; Christian Gerdes, J.; Lenz, B.; and Winner, H. 2016. Autonomous driving: technical, legal and social aspects. Springer Nature.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In Proceedings of the 33rd International Conference on Machine Learning.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. Nature.

Mousavi, A.; Li, L.; Liu, Q.; and Zhou, D. 2020. Black-box Off-policy Estimation for Infinite-Horizon Reinforcement Learning. In International Conference on Learning Representations.
Proofs

Proof of Proposition 1

\[ J_λ(\pi_{k+1}) = \sum_{s,a} d_{π_{k+1}}(s,a) (r(s,a) - λr(s,a))^2 + λ\max_y (2 \sum_{s,a} d_{π_{k+1}}(s,a) r(s,a)y - y^2) \]
\[ \geq \sum_{s,a} d_{π_{k+1}}(s,a) (r(s,a) - λr(s,a))^2 + λ(2 \sum_{s,a} d_{π_{k+1}}(s,a) r(s,a)y_{k+1} - y_{k+1}^2) \]
\[ = \sum_{s,a} d_{π_{k+1}}(s,a) (r(s,a) - λr(s,a))^2 + λ(2 \sum_{s,a} d_{π_{k+1}}(s,a) r(s,a)y) - y^2) \]
\[ \geq \sum_{s,a} d_{π_k}(s,a) (r(s,a) - λr(s,a))^2 + 2λr(s,a)y_{k+1}) - λy_{k+1}^2 \]

(By definition, \(π_{k+1}\) is the maximizer.)

\[ = J_λ(π_k) \]

Proof of Proposition 2

Lemma 1. Under Assumption 2, \(\nabla_θ J_λ(θ)\) is Lipschitz continuous in \(θ\).

Proof. By definition,

\[ \nabla J_λ(θ) = \nabla E[R] - \nabla λE[R^2] + 2λE[R]\nabla E[R]. \]

The policy gradient theorem (Sutton et al. 2000) and the boundedness of \(\nabla \log π_θ(a|s)\) imply that \(\nabla E[R]\) is bounded. So \(E[R]\) is Lipschitz continuous. Lemma B.2 in Papini et al. (2018) shows that the Hessian of \(E[R]\) is bounded. So \(\nabla E[R]\) is Lipschitz continuous. So does \(\nabla E[R^2]\). Together with the boundedness of \(E[R]\), it is easy to see \(\nabla J_λ(θ)\) is Lipschitz continuous.

We now prove Proposition 2.

Proof. Under Assumption 1, Theorem 4.1(c) in Tseng (2001) shows that the limit of any convergent subsequence \(\{(θ_k, y_k)\}_{k∈K}\) referred to as \(θ_K, y_K\), satisfies \(\nabla_θ J_λ(θ_K, y_K) = 0\) and \(\nabla y J_λ(θ_K, y_K) = 0\). In particular, that Theorem 4.1(c) is developed for general block coordinate ascent algorithms with \(M\) blocks. Our MVPI is a special case with two blocks (i.e., \(θ\) and \(y\)). With only two blocks, the conclusion of Theorem 4.1(c) follows immediately from Eq (7) and Eq (8) in Tseng (2001), without involving the assumption that the maximizers of the \(M - 2\) blocks are unique.

As \(J_λ(θ, y)\) is quadratic in \(y\), \(\nabla y J_λ(θ_K, y_K) = 0\) implies \(y_K = \arg\max_y J_λ(θ_K, y) = (1 - γ)J(θ_K)\). Recall the Fenchel duality \(x^2 = \max_z f(x, z)\), where \(f(x, z) = 2xz - z^2\). Applying Danskin’s theorem (Proposition B.25 in Bertsekas 1995) to Fenchel duality yields

\[ \frac{∂x^2}{∂x} = \arg\max_z f(x, z) \cdot \frac{∂x}{∂x}. \]  

Note Danskin’s theorem shows that we can treat \(\arg\max_z f(x, z)\) as a constant independent of \(x\) when computing the gradients in the RHS of Eq (6). Applying Danskin’s theorem in the Fenchel duality used in Eq (6) yields

\[ \nabla_θ J_λ(θ_K) = \nabla_θ J_λ(θ_K, y_K) = 0. \]

Eq (7) can also be easily verified without invoking Danskin’s theorem by expanding the gradients explicitly. Eq (7) indicates that the subsequence \(\{θ_k\}_{k∈K}\) converges to a stationary point of \(J_λ(θ)\).
Theorem 1 establishes the monotonic policy improvement when we search over all possible policies (The $\arg\max$ of Step 2 in Algorithm 1 is taken over all possible policies). Fortunately, the proof of Theorem 1 can also be used (up to a change of notation) to establish that

$$J_\lambda(\theta_{k+1}) \geq J_\lambda(\theta_k).$$

In other words, the monotonic policy improvement also holds when we search over all possible policies (The $\arg\max$ of Step 2 in Algorithm 1 is taken over all possible policies). Consequently, $\xi$ does not matter. We tune $\lambda$ from $\{0.5, 1, 2\}$. We set the perturbation $\beta$ in Prashanth and Ghavamzadeh (2013) to $10^{-4}$. We use 16 parallelized actors. The initial learning rate of the RMSprop optimizer is $7 \times 10^{-5}$.

Experiment Details

The pseudocode of MVPI-TD3 and our TRVO (MVPI-PPO) are provide in Algorithms 3 and 4 respectively.

Algorithm 3: MVPI-TD3

Input:

$\theta, \psi$: parameters for the deterministic policy $\pi$ and the value function $q$

$K$: number of recent rewards for estimating the policy performance

$\lambda$: weight of the variance penalty

Initialize the replay buffer $\mathcal{M}$

Initialize $S_0$

for $t = 0, \ldots$ do

\[ A_t \leftarrow \pi(S_t) + \mathcal{N}(0, \sigma^2) \]

Execute $A_t$, get $R_{t+1}, S_{t+1}$

Store $(S_t, A_t, R_{t+1}, S_{t+1})$ into $\mathcal{M}$

\[ y \leftarrow \frac{1}{K} \sum_{i=t-K+2}^{t+1} R_i \]

Sample a mini-batch $\{s_i, a_i, r_i, s_i'\}_{i=1,\ldots,N}$ from $\mathcal{M}$

for $i = 1, \ldots, N$ do

\[ \hat{r}_i \leftarrow r_i - \lambda \sigma_i^2 + 2\lambda r_i y \]

end

Use TD3 with $\{s_i, a_i, \hat{r}_i, s_i'\}_{i=1,\ldots,N}$ to optimize $\theta$ and $\psi$

$t \leftarrow t + 1$

end

Task Selection: We use eight Mujoco tasks from Open AI gym (Brockman et al. 2016) and implement the tabular MDP in Figure 3a by ourselves.

Function Parameterization: For MVPI-TD3 and TD3, we use the same network architecture as Fujimoto, van Hoof, and Meger (2018). For TRVO (MVPI-PPO), the methods of Tamar, Di Castro, and Mannor (2012), Prashanth and Ghavamzadeh (2013), and MVP, we use the same network architecture as Schulman et al. (2017).

Hyperparameter Tuning: For MVPI-TD3 and TD3, we use the same hyperparameters as Fujimoto, van Hoof, and Meger (2018). In particular, for MVPI-TD3, we set $K = 10^4$. For TRVO (MVPI-PPO), we use the same hyperparameters as Schulman et al. (2017). We implement the methods of Prashanth and Ghavamzadeh (2013), Tamar, Di Castro, and Mannor (2012) and MVP with multiple parallelized actors like A2C in Dharwal et al. (2017) and inherit the common hyperparameters from Dharwal et al. (2017).

Hyperparameters of Prashanth and Ghavamzadeh (2013): To increase stability, we treat $\lambda$ as a hyperparameter instead of a variable. Consequently, $\xi$ does not matter. We tune $\lambda$ from $\{0.5, 1, 2\}$. We set the perturbation $\beta$ in Prashanth and Ghavamzadeh (2013) to $10^{-4}$. We use 16 parallelized actors. The initial learning rate of the RMSprop optimizer is $7 \times 10^{-5}$.
We report the empirical results with \(\lambda\) policy with tabular logits. Results comparing MVPI-TD3 and TRVO w.r.t. \(J_{\text{reward}}\) in Figures 4, 5, and 6. For an algorithm, \(J_{\text{reward}}\) is defined as

\[
J_{\text{reward}} = \text{mean}(\{r_i\}) - \lambda \text{variance}(\{r_i\}),
\]

where \(\{r_i\}\) are the rewards during the evaluation episodes. We do not use the \(\gamma\)-discounted reward in computing \(J_{\text{reward}}\) because we use the direct average for the policy evaluation step in both MVPI-TD3 and TRVO.

| Algorithm 4: MVPI-PPO | \(\Delta J_{\text{MVPI}}\) | \(\Delta J_{\text{MVPI mean}}\) | \(\Delta J_{\text{MVPI variance}}\) | \(\Delta J_{\text{SR}}\) | \(\Delta J_{\text{SR mean}}\) | \(\Delta J_{\text{SR variance}}\) |
|-----------------------|----------------|----------------|----------------|---|----------------|----------------|
| InvertedP.            | -51%          | -2%           | 107%          | -100%  | 0%             | 0%             |
| InvertedD.P.          | -269%         | -31%          | 266%          | -64%  | 4%             | 9%             | 4%             | 11%             |
| HalfCheetah            | 81%           | -82%          | -81%          | 59%   | 97%            | -40%           | -96%           | 188%            |
| Walker2d              | -17%          | -49%          | 15%           | -52%  | 81%            | -37%           | -80%           | 41%             |
| Swimmer               | -2%           | 8%            | 165%          | -34%  | 1%             | -3%            | -69%           | 76%             |
| Hopper                | -9%           | -17%          | 9%            | -20%  | 94%            | 27%            | -93%           | 364%            |
| Reacher               | -34%          | -13%          | 98%           | 20%   | -12%           | -5%            | 35%            | 10%             |
| Ant                   | 94%           | -80%          | -94%          | -19%  | 80%            | -41%           | -80%           | 32%             |

Table 2: Table I with \(\lambda = 0.5\).

Hyperparameters of Tamar, Di Castro, and Mannor (2012): We tune \(\lambda\) from \{0.5, 1, 2\}. We use \(\xi = 50\), tuned from \{1, 10, 50, 100\}. We set the initial learning rate of the RMSprop optimizer to \(7 \times 10^{-4}\), tuned from \{\(7 \times 10^{-5}\), \(7 \times 10^{-4}\), \(7 \times 10^{-3}\)\}. We also test the Adam optimizer, which performs the same as the RMSprop optimizer. The learning rates for the running estimates of \(E[G_0]\) and \(V(G_0)\) is 100 times of the initial learning rate of the RMSprop optimizer. We use 16 parallelized actors. We use policy entropy as a regularization term, whose weight is 0.01. We clip the gradient by norm with a threshold 0.5.

Hyperparameters of Xie et al. (2018): We tune \(\lambda\) from \{0.5, 1, 2\}. We set the initial learning rate of the RMSprop optimizer to \(7 \times 10^{-4}\), tuned from \{\(7 \times 10^{-5}\), \(7 \times 10^{-4}\), \(7 \times 10^{-3}\)\}. We also test the Adam optimizer, which performs the same as the RMSprop optimizer. We use 16 parallelized actors. We use policy entropy as a regularization term, whose weight is 0.01. We clip the gradient by norm with a threshold 0.5.

Computing Infrastructure: We conduct our experiments on an Nvidia DGX-1 with PyTorch, though no GPU is used. In our off-line off-policy experiments, we set \(\bar{K}\) to \(10^3\) and use tabular representation for \(\rho_\pi, q_\pi\). For \(\pi\), we use a softmax policy with tabular logits.

Other Experimental Results

We report the empirical results with \(\lambda = 0.5\) and \(\lambda = 2\) in Figure 4, Table 2, Figure 5, and Table 3.
Figure 4: Figure 1 with $\lambda = 0.5$.

Figure 5: Figure 1 with $\lambda = 2$.

| Environment    | $\Delta$ TRVO | $\Delta$ mean | $\Delta$ variance | $\Delta$ TRVO | $\Delta$ MVC | $\Delta$ MVC | $\Delta$ MVC | $\Delta$ MVC |
|----------------|---------------|---------------|-------------------|---------------|--------------|--------------|--------------|--------------|
| InvertedPendulum-v2 | -4130%         | -5%           | 2·10^{8}          | -100%         | 0%           | 0%           | 0%           | 0%           |
| InvertedDoublePendulum-v2 | -39%          | -13%          | 398%              | -61%          | 77%          | -31%         | -77%         | 44%           |
| Ant-v2         | 97%           | -94%          | -97%              | -65%          | 97%          | -63%         | -97%         | 101%          |
| HalfCheetah-v2 | 98%           | -84%          | -98%              | 0%            | 96%          | -66%         | -95%         | 60%           |
| Swimmer-v2     | -210%         | -23%          | 619%              | -71%          | -6%          | -7%          | -13%         | -1%           |
| Walker2d-v2    | 64%           | -31%          | -64%              | 15%           | 90%          | -18%         | -89%         | 152%          |
| Reacher-v2     | -43%          | -2%           | 74%               | 23%           | 0%           | 1%           | 1%           | 1%            |

Table 3: Table 1 with $\lambda = 2$. 
Figure 6: Comparing MVPI-TD3 and TRVO in terms of $J_{\text{reward}}$ with $\lambda = 0.5$

Figure 7: Comparing MVPI-TD3 and TRVO in terms of $J_{\text{reward}}$ with $\lambda = 1$

Figure 8: Comparing MVPI-TD3 and TRVO in terms of $J_{\text{reward}}$ with $\lambda = 2$