Train Hard, Fight Easy: Robust Meta Reinforcement Learning

Ido Greenberg
Technion, Nvidia Research
gido@campus.technion.ac.il

Shie Mannor
Technion, Nvidia Research
shie@ee.technion.ac.il

Gal Chechik
Bar Ilan University, Nvidia Research
gchechik@nvidia.com

Eli Meirom
Nvidia Research
emeirom@nvidia.com

Abstract

A major challenge of reinforcement learning (RL) in real-world applications is the variation between environments, tasks or clients. Meta-RL (MRL) addresses this issue by learning a meta-policy that adapts to new tasks. Standard MRL methods optimize the average return over tasks, but often suffer from poor results in tasks of high risk or difficulty. This limits system reliability since test tasks are not known in advance. In this work, we define a robust MRL objective with a controlled robustness level. Optimization of analogous robust objectives in RL is known to lead to both **biased gradients** and **data inefficiency**. We prove that the gradient bias disappears in our proposed MRL framework. The data inefficiency is addressed via the novel Robust Meta RL algorithm (***RoML***). RoML is a meta-algorithm that generates a robust version of any given MRL algorithm, by identifying and over-sampling harder tasks throughout training. We demonstrate that RoML achieves robust returns on multiple navigation and continuous control benchmarks.

1 Introduction

Reinforcement learning (RL) has achieved impressive results in a variety of applications in recent years, including cooling systems control [Luo et al., 2022] and conversational chatbots [Cohen et al., 2022]. A significant challenge in extending this success to mass production is the variation between instances of the problem, e.g., different cooling systems or different chatbot end-users. Meta-RL (MRL) addresses this challenge by learning a "meta-policy" that quickly adapts to new tasks [Thrun and Pratt, 1998, Finn et al., 2017]. In the examples above, MRL would maximize the average return of the adapted policy for a new cooling system or a new end-user.

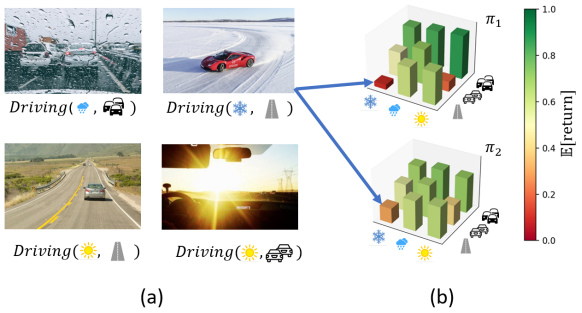


Figure 1: (a) An illustration of driving tasks, characterized by various weather conditions and traffic density. (b) The returns of two meta-policies π_1, π_2 on these tasks. π_1 has a higher average return, but π_2 is more robust to high-risk tasks. The task space is discretized only for illustration purposes.

However, optimizing the return of the average client might not suffice, as certain clients may still experience low or even negative returns. If 10% of the clients report poor performance, it may deter potential clients from adopting the new technology – even if its average return is high. This highlights

16th European Workshop on Reinforcement Learning (EWRL 2023).

the need for MRL systems that provide robust returns across tasks. Robustness is further motivated by risk sensitivity in many natural RL applications, such as medical treatment [Yu et al., 2021] and driving [Shalev-Shwartz et al., 2016]. For example, as illustrated in Fig. 1, an agent should drive safely at *any* road profile – even if at some roads the driving would be more cautious than necessary.

A common approach for risk-averse optimization is the max-min objective [Collins et al., 2020]; in MRL, that would mean searching for a meta-policy with the highest expected-return in the worst possible task. This expresses the most extreme risk aversion, which only attends to the one worst case out of all the possible outcomes. Furthermore, in certain problems, worst cases are inevitable (e.g., in certain medical treatments, a fatal outcome cannot be avoided), thus optimizing the minimum return might not provide any meaningful result. A more general objective is the average return over the worst α quantiles ($0 \leq \alpha \leq 1$), also known as the Conditional Value-at-Risk (CVaR). Notice that the CVaR is a generalization of both the mean (for $\alpha = 1$) and the minimum (for $\alpha = 0$).

CVaR is a coherent risk measure used for risk management in various fields [Filippi et al., 2020], e.g., banking regulation [Acerbi and Szekely, 2014]. CVaR optimization has long been researched in RL [Tamar et al., 2015, Hiraoka et al., 2019], where it is equivalent to robust optimization under uncertain perturbations [Chow et al., 2015]. In this work, we extend CVaR optimization to MRL, by replacing the standard MRL objective

$$\operatorname*{argmax}_{\pi} J^{\theta}(R), \quad J^{\theta}(R) = \mathbb{E}_{\tau,R}[R], \tag{1}$$

with a CVaR objective, which aims for a policy that is robust to high-risk tasks:

$$\operatorname*{argmax}_{\pi} J^{\theta}_{\alpha}(R), \quad J^{\theta}_{\alpha}(R) = \texttt{CVaR}^{\alpha}_{\tau}\left[\mathbb{E}_{R}[R]\right]. \tag{2}$$

In both equations, τ is a random task and R is the random return of policy π_θ in τ.

In Section 4, we follow the standard approach of policy gradient (PG) for $\texttt{CVaR}_\alpha$ optimization in RL, and apply it to MRL. That is, for every batch of N trajectories, we apply the learning step to the αN trajectories with the lowest returns. This standard approach, CVaR-PG, is known to suffer from a major limitation: in an actor-critic framework, the critic leads to a biased gradient estimator – to the extent that it may point to the opposite direction [Tamar et al., 2015]. This limitation is quite severe: many CVaR-PG implementations [Tamar et al., 2015, Greenberg et al., 2022] rely on vanilla PG without a critic (REINFORCE, Williams [1992]); others pay the price of gradient bias – in favor of advanced actor-critic methods that reduce the gradient variance [Rajeswaran et al., 2017].

This limitation is particularly concerning in *meta* RL, where high complexity and noise require more sophisticated algorithms than REINFORCE. Fortunately, **Section 4 eliminates this concern: in MRL, in contrast to RL, the CVaR policy gradient is proven to remain unbiased regardless of the choice of critic.** Hence, our proposed method – CVaR Meta Learning (***CVaR-ML***) – can be safely applied on top of any MRL algorithm. This makes CVaR-ML a *meta-algorithm*: given an arbitrary MRL algorithm, CVaR-ML generates a robust version of it.

Nevertheless, in CVaR optimization methods, another source of gradients variance and sample inefficiency is the large proportion of data not being utilized. Every iteration, we rollout trajectories for N tasks, but only use αN of them for training. To mitigate this effect, we introduce in Section 5 the Robust Meta RL algorithm (***RoML***). RoML assumes that tasks can be selected during training. It learns to identify tasks with lower returns and over-samples them. By training on high-risk tasks, the meta-agent learns policies that are robust to them without discarding data. Hence, **RoML increases the sample efficiency by a factor of up to $\boldsymbol{\alpha^{-1}}$**. Unlike common adversarial methods, which search for the worst-case sample (task) that minimizes the return [Collins et al., 2020], RoML lets the user specify the desired level of robustness α, and addresses the entire α-tail of the return distribution.

We test our algorithms on several domains. Section 6.1 considers a navigation problem, where both CVaR-ML and RoML obtain better CVaR returns than their risk-neutral baseline. Furthermore, they learn substantially different navigation policies. Section 6.2 considers several continuous control environments with varying tasks. These environments are challenging for CVaR-ML, which entirely fails to learn. Yet, RoML preserves its effectiveness and consistently improves the robustness of the returns. In addition, Appendix E demonstrates that under certain conditions, RoML can be applied to supervised settings as well – providing robust supervised meta-learning.

As a meta-algorithm, in each experiment RoML improves the robustness of its baseline algorithm – using the same hyper-parameters as the baseline. The *average* return is also improved in certain experiments, indicating that even the risk-neutral objective of Eq. (1) may benefit from robustness.

Contribution: (a) We propose a principled CVaR optimization framework for robust meta-RL. While the analogous problem in standard RL suffers from biased gradients and data inefficiency, we (b) prove theoretically that MRL is immune to the former, and (c) address the latter via the novel Robust Meta RL algorithm (RoML). Finally, (d) we demonstrate the robustness of RoML experimentally.

2 Related Work

Meta-RL for the **average task** is widely researched, including methods based on gradients [Finn et al., 2017, Gupta et al., 2018], latent memory [Zintgraf et al., 2019, Rakelly et al., 2019] and offline meta learning [Dorfman et al., 2020, Pong et al., 2022]. It is used for applications ranging from robotics [Nagabandi et al., 2018] to education [Wu et al., 2021]. Adversarial meta learning was studied for minimax optimization of the **lowest-return task**, in supervised meta learning [Collins et al., 2020, Goldblum et al., 2020] and MRL [Lin et al., 2020]. Other works studied the robustness of MRL to distributional shifts [Mendonca et al., 2020, Ajay et al., 2022]. However, the **CVaR task** objective has not been addressed yet in the framework of MRL.

Risk-averse RL. In *standard* RL, risk awareness is widely studied for both safety [García and Fernández, 2015, Greenberg and Mannor, 2021] and robustness [Derman et al., 2020]. CVaR specifically was studied using PG [Tamar et al., 2015, Rajeswaran et al., 2017, Hiraoka et al., 2019, Huang et al., 2021a], value iteration [Chow et al., 2015] and distributional RL [Dabney et al., 2018, Schubert et al., 2021, Lim and Malik, 2022]. CVaR optimization was also shown equivalent to mean optimization under robustness [Chow et al., 2015], motivating robust-RL methods [Pinto et al., 2017, Godbout et al., 2021]. In this work, we propose a *meta-learning* framework and algorithms for CVaR optimization, and point to both similarities and differences from the standard RL setting.

Sampling. In Section 5, we use the cross-entropy method [de Boer et al., 2005] to sample high-risk tasks for training. The cross-entropy method has been studied in standard RL for both optimization [Mannor et al., 2003, Huang et al., 2021b] and sampling [Greenberg et al., 2022]. Sampling in RL was also studied for regret minimization in the framework of Unsupervised Environment Design [Dennis et al., 2020, Jiang et al., 2021]; and for accelerated curriculum learning in Contextual RL [Klink et al., 2020, Eimer et al., 2021]. By contrast, we address MRL (where the current task is not known to the agent, unlike Contextual RL), and optimize the CVaR risk measure instead of the mean.

3 Preliminaries

MRL. Consider a set of Markov Decision Processes (MDPs) $\{(S, A, \tau, \mathcal{P}_\tau, \mathcal{P}_{0,\tau}, \gamma)\}_{\tau\in\Omega}$, where the distribution of transitions and rewards $\mathcal{P}_\tau$ and the initial state distribution $\mathcal{P}_{0,\tau}$ both depend on task $\tau \in \Omega$. The task itself is drawn from a distribution $\tau \sim D$ over a general space Ω, and is not known to the agent. The agent can form a belief regarding the current τ based on the task history h, which consists of repeating triplets of states, actions and rewards [Zintgraf et al., 2019]. Thus, the meta-policy $\pi_\theta(a; s, h)$ ($\theta \in \Theta$) maps the current state $s \in S$ and the history $h \in \prod(S \times A \times \mathbb{R})$ (consisting of state-action-reward triplets) to a probability distribution over actions.

A meta-rollout is defined as a sequence of $K \geq 1$ episodes of length $T \in \mathbb{N}$ over a single task τ: $\Lambda = \{\{(s_{k,t},\, a_{k,t},\, r_{k,t})\}_{t=1}^{T}\}_{k=1}^{K}$. For example, in a driving problem, τ might be a geographic area or type of roads, and Λ a sequence of drives on these roads. The return of the agent over a meta-rollout is defined as $R(\Lambda) = \frac{1}{K}\sum_{k=1}^{K}\sum_{t=0}^{T}\gamma^t r_{k,t}$, where $r_{k,t}$ is the (random variable) reward at step t in episode k. Given a task τ and a meta-policy π_θ, we denote by $P_\tau^\theta(x)$ the conditional PDF of the return R. With a slight abuse of notation, we shall use $P_\tau^\theta(\Lambda)$ to also denote the PDF of the meta-rollout itself. The standard MRL objective is to maximize the expected return $J^\theta(R) = \mathbb{E}_{\tau,R}[R]$.

While the meta policy $\pi_\theta(s, a; h)$ is history-dependent, it can still be learned using standard policy gradient (PG) approaches, by considering h as part of an extended state space $\tilde{s} = (s, h) \in \tilde{S}$. Then, the policy gradient can be derived directly:

$$\nabla_\theta J^\theta(R) = \int_\Omega D(z) \int_{-\infty}^{\infty} (x - b)\nabla_\theta P_z^\theta(x) \cdot dx \cdot dz, \tag{3}$$

where D is a probability measure over the task space Ω, $P_z^\theta(x)$ is the PDF of R (conditioned on π_θ and $\tau = z$), and b is any arbitrary baseline that is independent of θ [Agrawal, 2019]. While

a direct gradient estimation via Monte Carlo sampling is often noisy, its variance can be reduced by an educated choice of baseline b. In the common actor-critic framework [Mnih et al., 2016], a learned value function $b = V(s; h)$ is used. This approach is used in many SOTA algorithms in deep RL, e.g., PPO [Schulman et al., 2017]; and by proxy, in MRL algorithms that rely on them, e.g., VariBAD [Zintgraf et al., 2019].

A major challenge in MRL is the extended state space $\tilde{S}$, which now includes the whole task history. Common algorithms handle the task history via a low-dimensional embedding that captures transitions and reward function [Zintgraf et al., 2019]; or using additional optimization steps w.r.t. task history [Finn et al., 2017]. Our work does not compete with such methods, but rather builds upon them: our methods operate as meta-algorithms that run on top of existing MRL baselines.

CVaR-PG. Before moving on to CVaR optimization in MRL, we first recap the common PG approach for standard (non-meta) RL. For a random variable X and α-quantile $q_\alpha(X)$, the CVaR is defined as $\texttt{CVaR}_\alpha(X) = \mathbb{E}\left[X \mid X \leq q_\alpha(X)\right]$. For an MDP (S, A, P, P_0, γ), the CVaR-return objective is $\tilde{J}^\theta_\alpha(R) = \texttt{CVaR}^\alpha_{R\sim P^\theta}[R] = \int_{-\infty}^{q^\theta_\alpha(R)} x \cdot P^\theta(x) \cdot dx$, whose corresponding policy gradient is [Tamar et al., 2015]:

$$\nabla_\theta \tilde{J}^\theta_\alpha(R) = \int_{-\infty}^{q^\theta_\alpha(R)} (x - q^\theta_\alpha(R)) \cdot \nabla_\theta P^\theta(x) \cdot dx. \tag{4}$$

Given a sample of N trajectories $\{\{(s_{i,t},\, a_{i,t})\}_{t=1}^T\}_{i=1}^N$ with returns $\{R_i\}_{i=1}^N$, the policy gradient can be estimated by [Tamar et al., 2015, Rajeswaran et al., 2017]:

$$\nabla_\theta \tilde{J}^\theta_\alpha(R) \approx \frac{1}{\alpha N} \sum_{i=1}^N \mathbf{1}_{R_i \leq \hat{q}^\theta_\alpha} \cdot (R_i - \hat{q}^\theta_\alpha) \cdot \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{i,t}; s_{i,t}), \tag{5}$$

where $\hat{q}^\theta_\alpha$ is an estimator of the current return quantile.

Notice that in contrast to mean-PG, in CVaR optimization the baseline *cannot* follow an arbitrary critic, but should approximate the total return quantile $q^\theta_\alpha(R)$. Tamar et al. [2015] showed that any baseline $b \neq q^\theta_\alpha(R)$ inserts bias to the CVaR gradient estimator, potentially to the level of pointing to the opposite direction (as discussed in Appendix A.1 and Fig. 5). As a result, CVaR-PG methods in RL either are limited to basic REINFORCE with a constant baseline [Greenberg et al., 2022], or use a critic for variance reduction at the cost of biased gradients [Rajeswaran et al., 2017].

Another major source of gradient-variance in CVaR-PG is its reduced sample efficiency: notice that Eq. (5) only exploits $\approx \alpha N$ trajectories out of each batch of N trajectories (due to the term $\mathbf{1}_{R_i \leq \hat{q}^\theta_\alpha}$), hence results in estimation variance larger by a factor of α^{-1}.

4 CVaR Optimization in Meta-Learning

In this section, we show that **unlike standard RL, CVaR-PG in MRL permits a flexible baseline *without* presenting biased gradients**. Hence, policy gradients for *CVaR* objective in *MRL* is substantially different from both *mean*-PG in MRL (Eq. (3)) and CVaR-PG in *RL* (Eq. (4)).

To derive the policy gradient, we first define the policy value per task and the tail of tasks.

Definition 1. The value of policy π_θ in task τ is denoted by $V^\theta_\tau = \mathbb{E}_{R\sim P^\theta_\tau}[R]$. Notice that V^θ_τ depends on the random variable τ. We define the α-tail of tasks w.r.t. π_θ as the tasks with the lowest values: $\Omega^\theta_\alpha = \{z \in \Omega \mid V^\theta_z \leq q_\alpha(V^\theta_\tau)\}$.

Assumption 1. To simplify integral calculations, we assume that for any $z \in \Omega$ and $\theta \in \Theta$, R is a continuous random variable (i.e., its conditional PDF $P^\theta_z(x)$ has no atoms). We also assume that $v(z) = V^\theta_z$ is a continuous function for any $\theta \in \Theta$.

Theorem 1 (Meta Policy Gradient for CVaR)**.** Under Assumption 1, the policy gradient of the CVaR objective in Eq. (2) is

$$\nabla_\theta J^\theta_\alpha(R) = \int_{\Omega^\theta_\alpha} D(z) \int_{-\infty}^{\infty} (x - b) \nabla_\theta P^\theta_z(x) \cdot dx \cdot dz, \tag{6}$$

where b is *any* arbitrary baseline independent of θ.

Proof intuition (the formal proof is in Appendix A). In RL, the CVaR-PG focuses on the α low-return trajectories and overlooks high-return trajectories. Hence, if the relative return $(R - b)$ were positive in the α-tail, the gradient would increase the tail probability, i.e., the probability of unsuccessful actions. Thus, the baseline must guarantee that $(R - b)$ is negative on the tail, as visualized in Fig. 5 in the appendix. In MRL, on the other hand, the focus is on the subset Ω_α^θ of *tasks*. The gradient expresses the entire return distribution for these tasks, hence actions that led to high returns are always preferred over ones that led to low returns (Fig. 6). The task distribution D itself does not depend on θ, so there cannot be a side effect of increasing the task-tail probability. □

According to Theorem 1, the CVaR PG in MRL permits *any* baseline b. As discussed in Section 3, this flexibility is necessary, for example, in any actor-critic framework.

To estimate the gradient from meta-rollouts of the tail tasks, we transform the integration of Eq. (6) into an expectation:

Corollary 1. Eq. (6) can be written as

$$\nabla_\theta J_\alpha^\theta(R) = \mathbb{E}_{\tau\sim D}\left[\mathbb{E}_{\Lambda\sim P_\tau^\theta}\left[g(\Lambda)\right] \;\middle|\; V_\tau^\theta \le q_\alpha(V_\tau^\theta)\right], \tag{7}$$

where $g(\Lambda) = (R(\Lambda) - b)\sum_{\substack{1\le k\le K,\\ 1\le t\le T}} \nabla_\theta \log \pi_\theta(a_{k,t};\, \tilde{s}_{k,t})$; and $\tilde{s}_{k,t} = (s_{k,t}, h_{k,t})$ is the extended state (that includes all the task history $h_{k,t}$ until trajectory k, step t).

Proof. We apply the standard log trick $\nabla_\theta P_z^\theta = P_z^\theta \cdot \nabla_\theta \log P_z^\theta$ to Eq. (6), after substituting the meta-rollout PDF: $P_z^\theta(\Lambda) = \prod_{k=1}^K \left[P_{0,z}(s_{k,0}) \cdot \prod_{t=1}^T P_z(s_{k,t+1}, r_{k,t} \mid s_{k,t}, a_{k,t})\pi_\theta(a_{k,t};\, \tilde{s}_{k,t})\right]$. □

For a practical Monte-Carlo estimation of Eq. (7), given a task z_i, we need to estimate whether $V_{z_i}^\theta \le q_\alpha(V_\tau^\theta)$. To estimate $V_{z_i}^\theta$, we can generate M i.i.d meta-rollouts with returns $\{R_{i,m}\}_{m=1}^M$, and calculate their average return $\hat{V}_{z_i}^\theta = R_i = \sum_{m=1}^M R_{i,m}/M$. Then, the quantile $q_\alpha(V_\tau^\theta)$ can be estimated over a batch of tasks $\hat{q}_\alpha = q_\alpha(\{R_i\}_{i=1}^N)$. If $\hat{V}_{z_i}^\theta \le \hat{q}_\alpha$, we use *all* the meta-rollouts of z_i for the gradient calculation (including meta-rollouts that by themselves have a higher return $R_{i,m} > \hat{q}_\alpha$). Notice that we use M i.i.d meta-rollouts, each consisting of K episodes (the episodes within a meta-rollout are *not* independent, due to agent memory).

Putting it together, we obtain the sample-based gradient estimator of Eq. (7):

$$\begin{aligned} \nabla_\theta J_\alpha^\theta(R) &\approx \frac{1}{\alpha N}\sum_{i=1}^N \mathbf{1}_{R_i \le \hat{q}_\alpha^\theta} \sum_{m=1}^M g_{i,m}, \\ g_{i,m} &\coloneqq (R_{i,m} - b)\sum_{k=1}^K\sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{i,m,k,t};\, \tilde{s}_{i,m,k,t}), \end{aligned} \tag{8}$$

where $a_{i,m,k,t}$, $\tilde{s}_{i,m,k,t}$ are the action and the state-and-history at task i, meta-rollout m, trajectory k and step t.

The procedure described above follows the principles of CVaR-PG in (non-meta) RL, as the learning rule is only applied to the tail of the sampled batch. However, in MRL we consider a batch of tasks rather than a batch of trajectories. As discussed in Theorem 1 and its proof, this distinction has a substantial impact on the gradient and the resulting algorithm. Specifically, Eq. (8) allows for greater flexibility than Eq. (4), as it permits any baseline b that does not depend on θ. This allows gradient calculation using any PG algorithm, including SOTA methods such as PPO [Schulman et al., 2017] (which are already used in MRL methods such as VariBAD Zintgraf et al. [2019]). Therefore, in contrast to standard RL, CVaR optimization is not restricted to basic REINFORCE.

Our CVaR Meta Learning method (**CVaR-ML**, Algorithm 1) leverages this property to operate as a *meta-algorithm*, providing a robust version for any given baseline algorithm, such as Finn et al. [2017], Zintgraf et al. [2019]:

Definition 2. A *baseline MRL algorithm* learns a meta-policy π_θ using a training step `ML`. Given a batch of meta-rollouts $\{\Lambda_i\}$, $\texttt{ML}(\{\Lambda_i\})$ updates π_θ.

Algorithm 1: CVaR Meta Learning (CVaR-ML)

1 **Input**: Meta-learning algorithm (Definition 2); robustness level $\alpha \in (0, 1]$; task distribution D; N tasks per batch; M meta-rollouts per task

2 **while** not finished training **do**

```
// Sample tasks
```

3 Sample $\{z_i\}_{i=1}^N \sim D$

```
// Run meta-rollouts
```

4 $\{\{\Lambda_{i,m}\}_{m=1}^M\}_{i=1}^N \leftarrow \text{rollouts}(\{z_i\}_{i=1}^N,\ M)$

5 $R_{i,m} \leftarrow \text{return}(\Lambda_{i,m}), \quad \forall i,\ m$

```
// Compute sample quantile
```

6 $R_i \leftarrow \text{mean}(\{R_{i,m}\}_{m=1}^M), \quad \forall i$

7 $\hat{q}_\alpha \leftarrow \text{quantile}(\{R_i\}_{i=1}^N,\ \alpha)$

```
// Meta-learning algorithm train step
```

8 $\texttt{ML}\big(\ \{\Lambda_{i,m} \mid R_i \le \hat{q}_\alpha,\ 1 \le m \le M\}\ \big)$

Notice that CVaR-ML only handles task filtering, and uses the baseline training step ML as a black box (Line 8). Hence, it can be used with *any* MRL baseline – not just PG methods. In fact, by using a supervised meta-learning baseline, CVaR-ML can be applied to the supervised setting as well with minimal modifications, namely, replacing meta-rollouts with examples and returns with losses.

5 Efficient CVaR-ML

Theorem 1 guarantees unbiased gradients when using Algorithm 1; however, it does not bound their variance. In particular, Line 8 applies the learning step to a subset of only αNM meta-rollouts out of NM, which increases the estimator variance by a factor of α^{-1} compared to mean optimization. This could be prevented if we knew the set Ω_α^θ of tail-tasks (for the current π_θ), and sampled only these tasks, using the distribution $D_\alpha^\theta(z) = \alpha^{-1}\mathbf{1}_{V_z^\theta \le q_\alpha(V_\tau^\theta)} D(z)$. Proposition 1 shows that this would indeed recover the sample efficiency.

Proposition 1 (Variance reduction)**.** Denote by G the estimator of $\nabla_\theta J_\alpha^\theta(R)$ in Eq. (8), assume there is no quantile error ($\hat{q}_\alpha^\theta = q_\alpha^\theta$), and denote $\mathbb{E}_D[\cdot] = \mathbb{E}_{z_i \sim D,\, R_{i,m} \sim P_{z_i}^\theta}[\cdot]$. Then, switching the task sample distribution to D_α^θ leads to a variance reduction of factor α:

$$\mathbb{E}_{D_\alpha^\theta}[\alpha G] = \mathbb{E}_D[G], \quad \text{Var}_{D_\alpha^\theta}(\alpha G) \le \alpha \cdot \text{Var}_D(G).$$

Proof sketch (the complete proof is in Appendix B). We calculate the expectation and variance directly. G is proportional to $\mathbf{1}_{R_i \le q_\alpha^\theta}$ (Eq. (8)). The condition $R_i \le q_\alpha^\theta$ leads to multiplication by the probability α when sampled from D (where it corresponds to the α-tail); but not when sampled from D_α^θ (where it is satisfied w.p. 1). This factor α cancels out the ratio between the expectations of G and αG (thus the expectations are identical) – but not the ratio α^2 between their variances. □

Following the motivation of Proposition 1, we wish to increase the number of train tasks that come from the tail distribution Ω_α^θ. To that end, we assume to have certain control over the sampling of training tasks. This assumption is satisfied in most simulated environments, as well as many real-world scenarios. For example, when training a driver, we choose the tasks, roads and times of driving throughout training. In this section, we propose a method to make these choices.

We begin with parameterizing the task distribution D: we consider a parametric family D_ϕ such that $D = D_{\phi_0}$. Then, we wish to modify the parameter ϕ so that D_ϕ aligns with D_α^θ as closely as

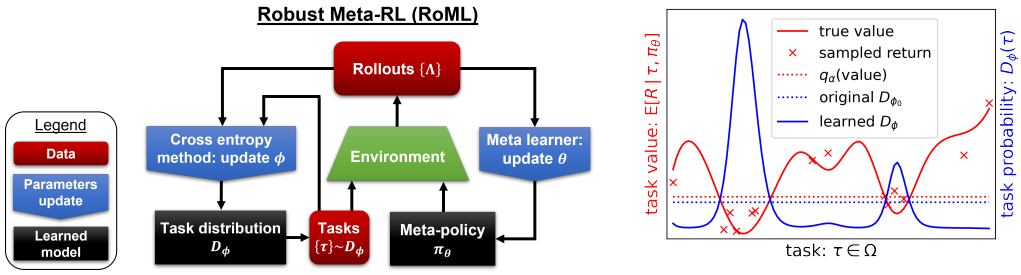


Figure 2: **Left**: RoML uses the cross entropy method to modify the task distribution D_ϕ, which is used to generate the next meta-rollouts. **Right**: illustration of an arbitrary point of time in training: the task distribution D_ϕ (blue) is learned according to the task values of the current meta-policy π_θ (red). Since low-return tasks are over-sampled, the learned meta-policy is more robust to the selection of task.

possible. To that end, we use the Cross Entropy Method (CEM, de Boer et al. [2005]), which searches for ϕ^* that minimizes the KL-divergence (i.e., cross entropy) between the two:

$$\begin{aligned} \phi^* \in \operatorname{argmin}_{\phi'} D_{KL}\left(D_\alpha^\theta \,||\, D_{\phi'}\right) &= \operatorname{argmax}_{\phi'} \mathbb{E}_{z\sim D_{\phi_0}}\left[\mathbf{1}_{V_z^\theta \le q_\alpha(V_\tau^\theta)} \log D_{\phi'}(z)\right] \\ &= \operatorname{argmax}_{\phi'} \mathbb{E}_{z\sim D_{\phi}}\left[w(z)\, \mathbf{1}_{V_z^\theta \le q_\alpha(V_\tau^\theta)} \log D_{\phi'}(z)\right], \end{aligned} \tag{9}$$

where $w(z) = \frac{D_{\phi_0}(z)}{D_\phi(z)}$ is the importance sampling weight corresponding to $z \sim D_\phi$. Note that Eq. (9) has a closed-form solution for most of the standard families of distributions [de Boer et al., 2005].

Algorithm 2: Robust Meta RL (RoML)

Input: Meta-learning algorithm (Definition 2); robustness level $\alpha \in (0, 1]$; parametric task distribution D_ϕ; original parameter ϕ_0; N tasks per batch, $\nu \in [0, 1)$ of them sampled from the original D_{ϕ_0}; CEM quantile $\beta \in (0, 1)$

Initialize:

$\phi \leftarrow \phi_0, \quad N_o \leftarrow \lfloor \nu N \rfloor, \quad N_s \leftarrow \lceil (1-\nu) N \rceil$

while not finished training **do**

// Sample tasks

Sample $\{z_{o,i}\}_{i=1}^{N_o} \sim D_{\phi_0}, \quad \{z_{\phi,i}\}_{i=1}^{N_s} \sim D_\phi$

$z \leftarrow (z_{o,1}, \ldots, z_{o,N_o}, z_{\phi,1}, \ldots, z_{\phi,N_s})$

// Rollouts and meta-learning step

$\{\Lambda_i\}_{i=1}^N \leftarrow \text{rollout}(\{z_i\}_{i=1}^N)$

$R_i \leftarrow \text{return}(\Lambda_i), \quad \forall i \in \{1, \ldots, N\}$

$\texttt{ML}(\{\Lambda_i\}_{i=1}^N)$

// Estimate reference quantile

$w_{o,i} \leftarrow 1, \qquad \forall i \in \{1, \ldots, N_o\}$

$w_{\phi,i} \leftarrow \frac{D_{\phi_0}(z_{\phi,i})}{D_\phi(z_{\phi,i})}, \quad \forall i \in \{1, \ldots, N_s\}$

$w \leftarrow (w_{o,1}, \ldots, w_{o,N_o}, w_{\phi,1}, \ldots, w_{\phi,N_s})$

$\hat{q}_\alpha \leftarrow \text{weighted_quantile}(\{R_i\}, w, \alpha)$

// Compute sample quantile

$q_\beta \leftarrow \text{quantile}(\{R_i\}, \beta)$

// Update sampler

$q \leftarrow \max(\hat{q}_\alpha, q_\beta)$

$\phi \leftarrow \operatorname{argmax}_{\phi'} \sum_{i \le N} w_i \, \mathbf{1}_{R_i \le q} \log D_{\phi'}(z_i)$

For a batch of N tasks sampled from $D = D_{\phi_0}$, Eq. (9) essentially chooses the αN tasks with the lowest returns, and updates ϕ to focus on these tasks. This may be noisy unless $\alpha N \gg 1$. Instead, the CEM chooses a larger number of tasks $\beta > \alpha$ for the update, where β is a hyper-parameter. ϕ is updated according to these βN lowest-return tasks, and the next batch is sampled from $D_\phi \neq D_{\phi_0}$. This repeats iteratively: every batch is sampled from D_ϕ, where ϕ is updated according to the βN lowest-return tasks of the former batch. Each task return is also compared to the α-quantile of the *original* distribution D_{ϕ_0}. If more than βN tasks yield lower returns, the CEM permits more samples for the update step. The return quantile over D_{ϕ_0} can be estimated from D_ϕ at any point using importance sampling weights. See more details about the CEM in Appendix C.1.

In our problem, the target distribution is the tail of D_{ϕ_0}. Since the tail is defined by the agent returns in these tasks, it varies with the agent and is non-stationary throughout training. Thus, we use the dynamic-target CEM of Greenberg [2022]. To smooth the changes in the sampled tasks, the sampler is also regularized to always provide certain exposure to all the tasks: we force ν percent of every batch to be sampled from the original distribution $D = D_{\phi_0}$, and only $1 - \nu$ percent from D_ϕ.

Putting this together, we obtain the Robust Meta RL algorithm (**RoML**), summarized in Algorithm 2 and Fig. 2. RoML does not require multiple meta-rollouts per update (parameter M in Algorithm 1), since it directly models high-risk tasks. Similarly to CVaR-ML, RoML is a meta-algorithm and can operate on top of any meta-learning baseline (Definition 2). Given the baseline implementation, only the tasks sampling procedure needs to be modified, which makes RoML easy to implement.

6 Experiments

We implement **RoML** and **CVaR-ML** on top of two different risk-neutral MRL baselines – **VariBAD** [Zintgraf et al., 2019] and **PEARL** [Rakelly et al., 2019]. As a risk-averse reference for comparison, we use **CeSoR** [Greenberg et al., 2022], an efficient sampling-based method for CVaR optimization in RL, implemented on top of PPO. As another reference, we use the Unsupervised Environment Design algorithm **PAIRED** [Dennis et al., 2020], which uses regret minimization to learn robust policies on a diverse set of tasks.

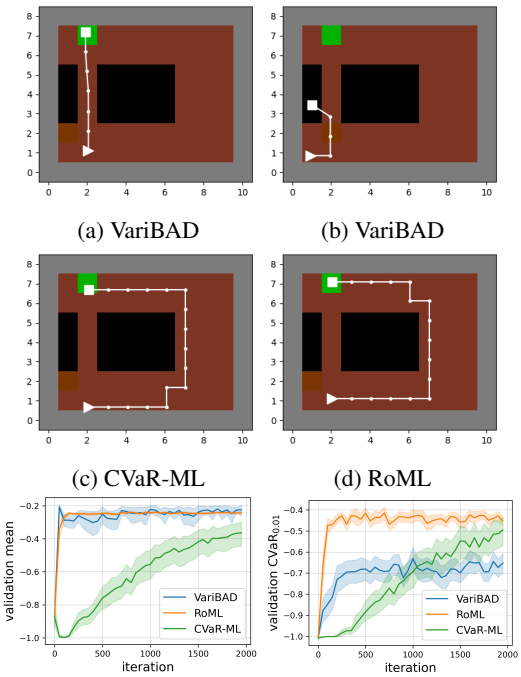
Section 6.1 demonstrates the mean/CVaR tradeoff, as our methods learn substantially different meta-policies from their baseline. Section 6.2 demonstrates the difficulty of the naive CVaR-ML in more challenging control benchmarks, and the efficacy of RoML in them. Appendix E demonstrates an

implementation of CVaR-ML and RoML on top of **MAML** [Finn et al., 2017] for *supervised* meta-learning. Each experiment is repeated for 30 seeds, using the same hyper-parameters and architectures for RoML, CVaR-ML and their baseline. In particular, we follow the original hyper-parameters of Zintgraf et al. [2019], Rakelly et al. [2019], Dennis et al. [2020], Greenberg et al. [2022] (Section 6.2) and Finn et al. [2017] (Appendix E). In Algorithm 1 we use $M = 1$ meta-rollout per task. The running times of RoML and CVaR-ML are indistinguishable from their baselines (RoML's CEM computations are negligible). See more details in Appendix D, and the code in our repositories: VariBAD, PEARL, CeSoR, PAIRED and MAML.

6.1 Khazad Dum

We demonstrate the tradeoff between mean and CVaR optimization in the Khazad Dum benchmark, visualized in Fig. 3. The agent begins at a random point in the bottom-left part of the map, and has to reach the green target as quickly as possible, without falling into the black abyss. The bridge is not covered and thus is exposed to wind and rain, rendering its floor slippery and creating an additive action noise (Fig. 3b) – to a level that varies with the weather. Each task is characterized by the rain intensity, which is exponentially distributed. The CEM in RoML is allowed to modify the parameter of this exponential distribution. Note that the agent is not aware of the current task (i.e, the weather), but may infer it from observations. We set the target risk level to $\alpha = 0.01$, and train each meta-agent for a total of $5 \cdot 10^6$ frames. See complete details in Appendix D.1.

We implement CVaR-ML and RoML on top of VariBAD. As shown in Fig. 3, VariBAD learns to try the short path – at the risk of rare falls into the abyss. By contrast, our CVaR-optimizing methods take the longer path and avoid risk. This policy increases the cumulative cost of time-steps, but leads to higher CVaR returns, as shown in Fig. 3f. In addition to superior CVaR, RoML also provides competitive *average* returns in this example (Fig. 3e). Finally, in accordance with Proposition 1, RoML learns significantly faster than CVaR-ML.



(a) VariBAD (b) VariBAD (c) CVaR-ML (d) RoML (e) Mean (f) CVaR

Figure 3: Khazad-Dum: (a-d) sample episodes; (e-f) test return vs. training iteration, with 95% confidence intervals over 30 seeds.

6.2 Continuous Control

We rely on standard continuous control problems from the MuJoCo framework [Todorov et al., 2012]: training a cheetah to run (**HalfCheetah**), and training a **Humanoid** and an **Ant** to walk. For each of the 3 environments, we create 3 meta-learning versions: (1) *Goal* or *Vel* [Finn et al., 2017], where each task corresponds to a different location or velocity objective, respectively; (2) *Mass*, where each task corresponds to a different body mass; and (3) *Body*, where each task corresponds to different mass, head size and physical damping level (similarly to Wang and Van Hoof [2022]). In addition, to experiment with high-dimensional task spaces, we randomly draw 10 numeric variables from *env.model* in HalfCheetah, and let them vary between tasks. We define 3 such environments with different random sets of task variables (**HalfCheetah 10D-task a,b,c**). For each of the 12 environments above, we set a target risk level of $\alpha = 0.05$ and optimize for $K = 2$ episodes per task. Additional implementation details are specified in Appendix D.2.

Interestingly, the naive approach of **CVaR-ML** consistently fails to meta-learn in all the cheetah environments. It remains unsuccessful even after large number of steps, indicating a difficulty beyond sample inefficiency. A possible explanation is the effectively decreased batch size of CVaR-ML. **PAIRED** and **CeSoR** also fail to adjust to the MRL environments, and obtain poor CVaR returns.

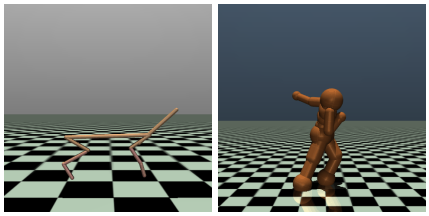
Table 1: $\text{CVaR}_{0.05}$ return over 1000 test tasks, for different models and MuJoCo environments. Standard deviation is presented over 30 seeds. Mean returns are displayed in Table 2.

	HalfCheetah			HalfCheetah 10D-task		
	Vel	Mass	Body	(a)	(b)	(c)
CeSoR	-2606 ± 25	902 ± 36	478 ± 27	637 ± 26	981 ± 31	664 ± 26
PAIRED	-725 ± 65	438 ± 37	218 ± 51	229 ± 59	354 ± 53	81 ± 65
CVaR-ML	-897 ± 23	38 ± 6	76 ± 5	120 ± 11	141 ± 11	81 ± 4
PEARL	-1156 ± 23	1115 ± 19	800 ± 5	1140 ± 33	1623 ± 23	$\mathbf{1016 \pm 5}$
VariBAD	-202 ± 6	1072 ± 16	835 ± 30	1126 ± 6	1536 ± 39	988 ± 13
RoML (VariBAD)	$\mathbf{-184 \pm 4}$	$\mathbf{1259 \pm 19}$	$\mathbf{935 \pm 17}$	$\mathbf{1227 \pm 13}$	$\mathbf{1697 \pm 24}$	999 ± 20
RoML (PEARL)	-1089 ± 31	1186 ± 34	808 ± 6	1141 ± 27	$\mathbf{1657 \pm 18}$	$\mathbf{1024 \pm 6}$

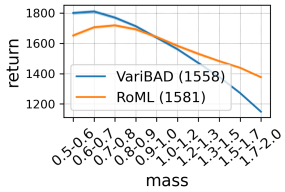
	Humanoid			Ant		
	Vel	Mass	Body	Goal	Mass	Body
VariBAD	801 ± 10	1283 ± 18	1290 ± 19	-500 ± 9	1370 ± 6	$\mathbf{1365 \pm 4}$
RoML (VariBAD)	$\mathbf{833 \pm 4}$	$\mathbf{1378 \pm 20}$	$\mathbf{1365 \pm 21}$	$\mathbf{-454 \pm 8}$	$\mathbf{1385 \pm 3}$	$\mathbf{1368 \pm 4}$

RoML, on the other hand, consistently improves the CVaR returns (Table 1) compared to its baseline (VariBAD or PEARL), while using the same hyper-parameters as the baseline. The VariBAD baseline presents better returns and running times than PEARL on HalfCheetah, and thus is used for the 6 extended environments (Humanoid and Ant). RoML improves the CVaR return in comparison to the baseline algorithm in all the 18 experiments (6 with PEARL and 12 with VariBAD).

In 5 out of 18 experiments, RoML slightly improves the *average* return compared to its baseline, and not only the CVaR (Table 2 in the appendix). The robustness of RoML to the selected task is also demonstrated in Fig. 4c. In multi-dimensional task spaces, RoML learns to focus the sampling modification on the high-impact variables, as demonstrated in Fig. 10 in the appendix. Finally, qualitative inspection shows that RoML learns to handle larger masses, for example, by leaning forward and letting gravity pull the agent forward (see animations on GitHub).



(a) HalfCheetah-Body (b) Humanoid-Mass



(c) HalfCheetah-Mass

Figure 4: MuJoCo: (a-b) sample frames, where tasks vary in mass, head size and damping level; (c) average return per range of tasks in HalfCheetah-Mass. RoML learns to act robustly: it is less sensitive to the task, and in particular performs better on high-risk tasks.

7 Summary and Future Work

We defined a robust MRL objective and derived the CVaR-ML algorithm to optimize it. In contrast to its analogous algorithm in standard RL, we proved that CVaR-ML does not present biased gradients, yet it does inherit the same data inefficiency. To address the latter, we introduced RoML and demonstrated its advantage in sample efficiency and CVaR return.

The main limitation of RoML is the need for at least partial control over the selection of training tasks. This assumption is common in other RL frameworks [Dennis et al., 2020, Jiang et al., 2021], and often holds in both simulations and the real world (e.g., deciding where and when to train driving). Yet, future research may increase the data efficiency of CVaR-ML *without* relying on such control.

Our meta-algorithm can also be applied to other scenarios in which a natural task structure is present, and leverage the structure to improve task robustness. Future applications may include supervised learning and coordinate-based regression (e.g., Tancik et al. [2020]).

Finally, RoML is easy to implement, operates agnostically as a meta-algorithm on top of existing MRL methods, and can be set to any desired robustness level. We believe that these properties, along with our empirical results, make RoML a promising candidate for MRL in risk-sensitive applications.

References

Carlo Acerbi and Balazs Szekely. Back-testing expected shortfall. Risk, 27(11):76–81, 2014.

Shipra Agrawal. Reinforcement learning: Policy gradient methods, 2019. URL https://ieor8100.github.io/rl/docs/Lecture%205%20-%20policy%20gradients.pdf.

Anurag Ajay, Abhishek Gupta, Dibya Ghosh, Sergey Levine, and Pulkit Agrawal. Distributionally adaptive meta reinforcement learning. arXiv preprint arXiv:2210.03104, 2022.

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.

Deborah Cohen, Moonkyung Ryu, Yinlam Chow, Orgad Keller, Ido Greenberg, Avinatan Hassidim, Michael Fink, Yossi Matias, Idan Szpektor, Craig Boutilier, and Gal Elidan. Dynamic planning in open-ended dialogue using reinforcement learning, 2022. URL https://arxiv.org/abs/2208.02294.

Liam Collins, Aryan Mokhtari, and Sanjay Shakkottai. Task-robust model-agnostic meta-learning. Advances in Neural Information Processing Systems, 33:18860–18871, 2020.

Will Dabney, Georg Ostrovski, David Silver, and Remi Munos. Implicit quantile networks for distributional reinforcement learning. In Proceedings of the 35th International Conference on Machine Learning, pages 1096–1105, 2018.

P. T. 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.

Tito Homem de Mello and Reuven Y. Rubinstein. Rare event estimation for static models via cross-entropy and importance sampling, 2003.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. Advances in neural information processing systems, 33:13049–13061, 2020.

Esther Derman, Daniel Mankowitz, Timothy Mann, and Shie Mannor. A bayesian approach to robust reinforcement learning. In Uncertainty in Artificial Intelligence, pages 648–658. PMLR, 2020.

Ron Dorfman, Idan Shenfeld, and Aviv Tamar. Offline meta learning of exploration. arXiv preprint arXiv:2008.02598, 2020.

Theresa Eimer, André Biedenkapp, Frank Hutter, and Marius Lindauer. Self-paced context evaluation for contextual reinforcement learning. In International Conference on Machine Learning, pages 2948–2958. PMLR, 2021.

Carlo Filippi, Gianfranco Guastaroba, and Maria Grazia Speranza. Conditional value-at-risk beyond finance: a survey. International Transactions in Operational Research, 27(3):1277–1319, 2020.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In International conference on machine learning, pages 1126–1135. PMLR, 2017.

Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. J. Mach. Learn. Res., 16:1437–1480, 2015.

Mathieu Godbout, Maxime Heuillet, Sharath Chandra, Rupali Bhati, and Audrey Durand. CARL: Conditional-value-at-risk adversarial reinforcement learning. arXiv preprint arXiv:2109.09470, 2021.

Micah Goldblum, Liam Fowl, and Tom Goldstein. Adversarially robust few-shot learning: A meta-learning approach. Advances in Neural Information Processing Systems, 33:17886–17895, 2020.

Ido Greenberg. Cross entropy method with non-stationary score function. https://pypi.org/project/cross-entropy-method/, 2022.

Ido Greenberg and Shie Mannor. Detecting rewards deterioration in episodic reinforcement learning. In Proceedings of the 38th International Conference on Machine Learning, volume 139, pages 3842–3853. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/greenberg21a.html.

Ido Greenberg, Yinlam Chow, Mohammad Ghavamzadeh, and Shie Mannor. Efficient risk-averse reinforcement learning. Advances in Neural Information Processing Systems, 2022. URL https://arxiv.org/abs/2205.05138.

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. Advances in neural information processing systems, 31, 2018.

Takuya Hiraoka, Takahisa Imagawa, Tatsuya Mori, Takashi Onishi, and Yoshimasa Tsuruoka. Learning robust options by conditional value at risk optimization. NeurIPS, 05 2019.

Audrey Huang, Liu Leqi, Zachary C. Lipton, and Kamyar Azizzadenesheli. On the convergence and optimality of policy gradient for markov coherent risk, 2021a. URL https://arxiv.org/abs/2103.02827.

Kevin Huang, Sahin Lale, Ugo Rosolia, Yuanyuan Shi, and Anima Anandkumar. CEM-GD: Cross-entropy method with gradient descent planner for model-based reinforcement learning, 2021b.

Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. Advances in Neural Information Processing Systems, 34:1884–1897, 2021.

Pascal Klink, Carlo D'Eramo, Jan R Peters, and Joni Pajarinen. Self-paced deep reinforcement learning. Advances in Neural Information Processing Systems, 33:9216–9227, 2020.

Shiau Hong Lim and Ilyas Malik. Distributional reinforcement learning for risk-sensitive policies. Advances in Neural Information Processing Systems, 2022.

Zichuan Lin, Garrett Thomas, Guangwen Yang, and Tengyu Ma. Model-based adversarial meta-reinforcement learning. Advances in Neural Information Processing Systems, 33:10161–10173, 2020.

Jerry Luo, Cosmin Paduraru, Octavian Voicu, Yuri Chervonyi, Scott Munns, Jerry Li, Crystal Qian, Praneet Dutta, Jared Quincy Davis, Ningjia Wu, Xingwei Yang, Chu-Ming Chang, Ted Li, Rob Rose, Mingyan Fan, Hootan Nakhost, Tinglin Liu, Brian Kirkman, Frank Altamura, Lee Cline, Patrick Tonker, Joel Gouker, Dave Uden, Warren Buddy Bryan, Jason Law, Deeni Fatiha, Neil Satra, Juliet Rothenberg, Mandeep Waraich, Molly Carlin, Satish Tallapaka, Sims Witherspoon, David Parish, Peter Dolan, Chenyu Zhao, and Daniel J. Mankowitz. Controlling commercial cooling systems using reinforcement learning, 2022. URL https://arxiv.org/abs/2211.07357.

Shie Mannor, Reuven Rubinstein, and Yohai Gat. The cross entropy method for fast policy search. Proceedings, Twentieth International Conference on Machine Learning, 2, 07 2003.

Russell Mendonca, Xinyang Geng, Chelsea Finn, and Sergey Levine. Meta-reinforcement learning robust to distributional shift via model identification and experience relabeling. arXiv preprint arXiv:2006.07178, 2020.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In International conference on machine learning, pages 1928–1937. PMLR, 2016.

Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. arXiv preprint arXiv:1803.11347, 2018.

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, page 2817–2826. JMLR.org, 2017.

Vitchyr H Pong, Ashvin V Nair, Laura M Smith, Catherine Huang, and Sergey Levine. Offline meta-reinforcement learning with online self-supervision. In International Conference on Machine Learning, pages 17811–17829. PMLR, 2022.

Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. Epopt: Learning robust neural network policies using model ensembles. ICLR, 2017.

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In International conference on machine learning, pages 5331–5340. PMLR, 2019.

Frederik Schubert, Theresa Eimer, Bodo Rosenhahn, and Marius Lindauer. Automatic risk adaptation in distributional reinforcement learning. arXiv preprint arXiv:2106.06317, 2021.

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

Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. arXiv preprint arXiv:1610.03295, 2016.

Aviv Tamar, Yonatan Glassner, and Shie Mannor. Optimizing the CVaR via sampling. AAAI'15, page 2993–2999, 2015.

Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P. Srinivasan, Jonathan T. Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations, 2020. URL `https://arxiv.org/abs/2012.02189`.

Sebastian Thrun and Lorien Pratt. Learning to learn. Springer Science & Business Media, 1998.

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, pages 5026–5033. IEEE, 2012.

J. R. R. Tolkien. The Lord of the Rings: The Fellowship of the Ring. George Allen and Unwin, 1954.

Anthony Tromba and Jerrold E Marsden. Vector calculus. Freeman, 1996.

Qi Wang and Herke Van Hoof. Model-based meta reinforcement learning using graph structured surrogate models and amortized policy search. In International Conference on Machine Learning, pages 23055–23077. PMLR, 2022.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8(3):229–256, 1992.

Mike Wu, Noah Goodman, Chris Piech, and Chelsea Finn. Prototransformer: A meta-learning approach to providing student feedback. arXiv preprint arXiv:2107.14035, 2021.

Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. Reinforcement learning in healthcare: A survey. ACM Computing Surveys (CSUR), 55(1):1–36, 2021.

Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. In International Conference on Learning Representations, 2019.

Contents

1 Introduction **1**

2 Related Work **3**

3 Preliminaries **3**

4 CVaR Optimization in Meta-Learning **4**

5 Efficient CVaR-ML **6**

6 Experiments **7**

7 Summary and Future Work **9**

A Policy Gradient for CVaR Optimization in Meta RL **14**

B Proof of Proposition 1 **17**

C The Cross Entropy Method **18**

D Experiments: Detailed Settings and Results **18**

E Supervised Meta-Learning **22**

A Policy Gradient for CVaR Optimization in Meta RL

In this section we provide the complete proof for Theorem 1. For completeness, Appendix A.1 recaps of the analogous proof in standard (non-meta) RL, before moving on to the proof in Appendix A.2. This allows us to highlight the differences between the two.

The substantial difference is that in RL, the CVaR is defined directly over the low-return trajectories, and the policy parameter θ affects the probability of each trajectory in the tail (Eq. (10)). In MRL (Eq. (2)), on the other hand, the CVaR is defined over the low-return tasks, whose probability is not affected directly by θ (Eq. (12)). This allows a decoupling between θ and τ, which results in Theorem 1.

Another high-level intuition is as follows. In RL, the CVaR-PG gradient is invariant to successful strategies, hence must be explicitly negative for the unsuccessful ones (in order not to encourage them, see Fig. 5). In MRL, within the tasks of interest, the gradient always encourages successful strategies on account of the unsuccessful ones (Fig. 6).

A.1 Recap: PG for CVaR in (non-meta) RL

We briefly recap the calculation of Propositions 1 and 2 in Tamar et al. [2015] for CVaR policy gradient under the standard RL settings.

Definition 3 (CVaR return in (non-meta) RL)**.** Consider an MDP (S, A, P, γ, P_0) with the cumulative reward (i.e., return) $R \sim P^\theta$, whose α-quantile is $q_\alpha^\theta(R)$. Recall the CVaR objective defined in Section 3:

$$\tilde{J}_\alpha^\theta(R) = \texttt{CVaR}_{R\sim P^\theta}^{\alpha}[R] = \int_{-\infty}^{q_\alpha^\theta(R)} x \cdot P^\theta(x) \cdot dx$$

To calculate the policy gradient of $\tilde{J}_\alpha^\theta(R)$, we begin with the conservation of probability mass below the quantile q_α^θ:

$$\int_{-\infty}^{q_\alpha^\theta(R)} P^\theta(x)dx \equiv \alpha.$$

Then, using the Leibniz integral rule, we have

$$0 = \nabla_\theta \int_{-\infty}^{q_\alpha^\theta(R)} P^\theta(x)dx = \left[\int_{-\infty}^{q_\alpha^\theta(R)} \nabla_\theta P^\theta(x)dx\right] + \left[P^\theta(q_\alpha^\theta(R)) \cdot \nabla_\theta q_\alpha^\theta(R)\right]. \tag{10}$$

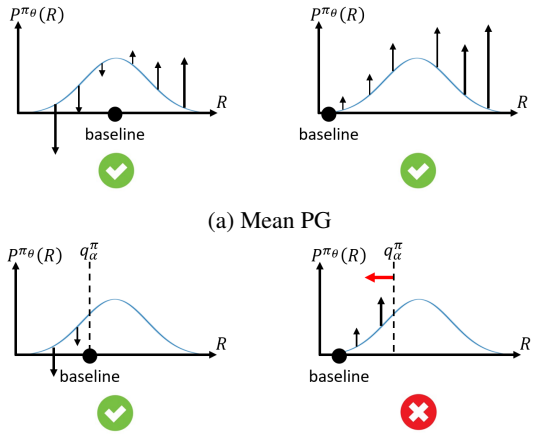
Notice that as a particular consequence of the conservation rule Eq. (10), positive gradients of $P^\theta(x)$ cause the quantile $q_\alpha^\theta(R)$ to decrease. This phenomenon makes the CVaR policy gradient sensitive to the selection of baseline, as visualized in Fig. 5b. In fact, the quantile $q_\alpha^\theta(R)$ itself is the only baseline that permits unbiased gradients:

$$\begin{aligned}
\nabla_\theta \tilde{J}_\alpha^\theta(R) = \nabla_\theta \int_{-\infty}^{q_\alpha^\theta(R)} x \cdot P^\theta(x) \cdot dx = \\
\left[\int_{-\infty}^{q_\alpha^\theta(R)} x \cdot \nabla_\theta P^\theta(x)dx\right] + \left[q_\alpha^\theta(R) \cdot P^\theta(q_\alpha^\theta(R)) \cdot \nabla_\theta q_\alpha^\theta(R)\right] \overbrace{=}^{Eq.\ (10)} \\
\left[\int_{-\infty}^{q_\alpha^\theta(R)} x \cdot \nabla_\theta P^\theta(x)dx\right] - \left[q_\alpha^\theta(R) \cdot \int_{-\infty}^{q_\alpha^\theta(R)} \nabla_\theta P^\theta(x)dx\right] = \\
\int_{-\infty}^{q_\alpha^\theta(R)} (x - q_\alpha^\theta(R)) \cdot \nabla_\theta P^\theta(x)dx,
\end{aligned} \tag{11}$$

which gives us Eq. (4).

A.2 PG for CVaR in Meta RL

We now turn to PG for CVaR optimization in *MRL*. We rely on the definitions and notations of Section 3, as well as Definition 1 and Assumption 1 in Section 4. Notice that $J_\alpha^\theta(R)$ of Eq. (2) can be



(a) Mean PG

(b) CVaR PG, Eq. (4)

Figure 5: Illustration of the policy gradient estimation in standard RL. (a) In Mean PG, the expected gradient is independent of the baseline: even if most of the distribution P^{π_θ} seems to be "pushed upwards", its normalization to a total probability of 1 forces the probability of low returns to decrease for that of high returns will increase. (b) In CVaR PG, due to the effect of Eq. (10), any baseline except for $q_\alpha^\theta(R)$ leads to biased gradients.

written in integral form as

$$J_\alpha^\theta(R) = \texttt{CVaR}_{\tau\sim D}^{\alpha}\left[\mathbb{E}_{R\sim P_\tau^\theta}[R]\right] = \int_{\Omega_\alpha^\theta} D(z) \int_{-\infty}^{\infty} x \cdot P_z^\theta(x) \cdot dx \cdot dz.$$

In Eq. (11) above, the boundary of the integral over the α-tail is simply the scalar q_α^θ. In MRL, this is replaced by the boundary of the set Ω_α^θ, defined in a general topological space. Thus, we begin by characterizing this boundary.

Lemma 1 (The boundary of Ω_α^θ)**.** Under Assumption 1, $\forall z \in \partial\Omega_\alpha^\theta : \int_{-\infty}^{\infty} x \cdot P_z^\theta(x) \cdot dx = q_\alpha(V_\tau^\theta)$.

Proof. Since $v(z) = V_z^\theta$ is a continuous function between topological spaces, by denoting $B = (-\infty,\, q_\alpha(V_\tau^\theta)]$ we have

$$\partial\Omega_\alpha^\theta = \partial v^{-1}(B) \overbrace{\subseteq}^{\text{continuous } v} v^{-1}(\partial B) = v^{-1}(\{q_\alpha(V_\tau^\theta)\}) = \{z \in \Omega \,|\, V_z^\theta = q_\alpha(V_\tau^\theta)\},$$

hence $\forall z \in \partial\Omega_\alpha^\theta : V_z^\theta = q_\alpha(V_\tau^\theta)$. Notice that $V_z^\theta = \int_{-\infty}^{\infty} x P_z^\theta(x) \cdot dx$. □

Finally, we can prove Theorem 1.

Proof of Theorem 1. First, we consider the conservation of probability:

$$\int_{\Omega_\alpha^\theta} D(z) dz \equiv \alpha$$

The gradient of this integral can be calculated using a high-dimensional generalization of the Leibniz integral rule, named Reynolds Transport Theorem (RTT, Tromba and Marsden [1996]):

$$\begin{aligned} 0 \overbrace{=}^{\substack{\text{derivative of}\\ \text{a constant}}} \nabla_\theta \int_{\Omega_\alpha^\theta} D(z) dz \overbrace{=}^{\text{RTT}} \int_{\Omega_\alpha^\theta} \nabla_\theta D(z) dz + \int_{\partial\Omega_\alpha^\theta} D(z) \cdot (v_b \cdot n) \cdot dA \\ \overbrace{=}^{\nabla_\theta D(z) \equiv 0} \int_{\partial\Omega_\alpha^\theta} D(z) \cdot (v_b \cdot n) \cdot dA, \end{aligned} \tag{12}$$

where $n(z, \theta)$ is the outward-pointing unit vector that is normal to the surface $\partial\Omega_\alpha^\theta$, and $v_b(z, \theta)$ is the velocity of the area element on the surface with area dA. Notice that $\nabla_\theta D(z) \equiv 0$ in the last

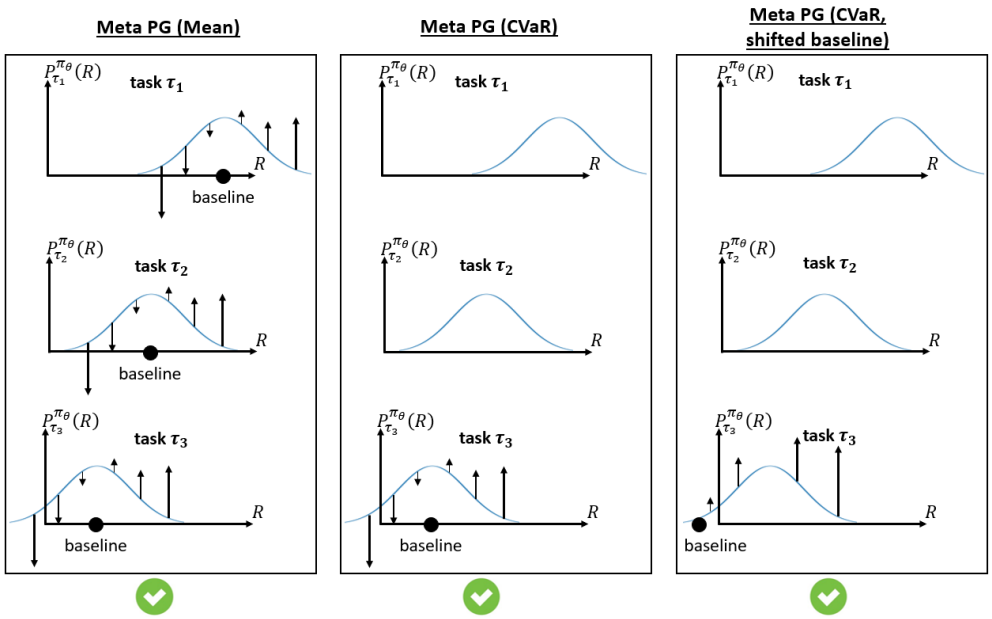


Figure 6: Illustration of the meta policy gradient estimation. In Mean meta-PG (left, Eq. (3)), the PG is estimated for all tasks. In CVaR meta-PG (center, Eq. (6)), it is only calculated over the low-return tasks. Since the returns distribution is decoupled from the tasks distribution, shifting the baseline (right) does not insert bias to the PG, in accordance with Theorem 1.

equality expresses a substantial difference from the standard RL settings in Eq. (10): there, we have $\nabla_\theta P^\theta(x)$, which does not necessarily vanish.

Next, we turn to the meta policy gradient itself, again using Reynolds Transport Theorem:

$$\begin{aligned}
\nabla_\theta J_\alpha^\theta(R) = \nabla_\theta \int_{\Omega_\alpha^\theta} D(z) \int_{-\infty}^{\infty} x P_z^\theta(x) \cdot dx \cdot dz \overbrace{=}^{\text{RTT}} \\
\left[\int_{\Omega_\alpha^\theta} D(z) \int_{-\infty}^{\infty} x \nabla_\theta P_z^\theta(x) \cdot dx \cdot dz\right] + \left[\int_{\partial\Omega_\alpha^\theta} D(z) \left(\int_{-\infty}^{\infty} x P_z^\theta(x) \cdot dx\right) \cdot (v_b \cdot n) \cdot dA\right] \overbrace{=}^{Lemma\ 1} \\
\left[\int_{\Omega_\alpha^\theta} D(z) \int_{-\infty}^{\infty} x \nabla_\theta P_z^\theta(x) \cdot dx \cdot dz\right] + \left[q_\alpha(V_\tau^\theta) \int_{\partial\Omega_\alpha^\theta} D(z) \cdot (v_b \cdot n) \cdot dA\right] \overbrace{=}^{Eq.\ (12)} \\
\int_{\Omega_\alpha^\theta} D(z) \int_{-\infty}^{\infty} x \nabla_\theta P_z^\theta(x) \cdot dx \cdot dz.
\end{aligned} \tag{13}$$

Finally, we show that any θ-independent additive baseline (highlighted in the equation) does not change the gradient calculation:

$$\begin{aligned}
&\int_{\Omega_\alpha^\theta} D(z) \int_{-\infty}^{\infty} (x - b) \nabla_\theta P_z^\theta(x) \cdot dx \cdot dz = \\
&\left[\int_{\Omega_\alpha^\theta} D(z) \int_{-\infty}^{\infty} x \nabla_\theta P_z^\theta(x) \cdot dx \cdot dz\right] - \left[\int_{\Omega_\alpha^\theta} D(z) \int_{-\infty}^{\infty} b \nabla_\theta P_z^\theta(x) \cdot dx \cdot dz\right] = \\
&\left[\int_{\Omega_\alpha^\theta} D(z) \int_{-\infty}^{\infty} x \nabla_\theta P_z^\theta(x) \cdot dx \cdot dz\right] - \left[\int_{\Omega_\alpha^\theta} D(z) \cdot b \cdot \nabla_\theta \left(\int_{-\infty}^{\infty} P_z^\theta(x) \cdot dx\right) \cdot dz\right] \overbrace{=}^{\int_{-\infty}^{\infty} P_z^\theta(x) dx \equiv 1} \\
&\left[\int_{\Omega_\alpha^\theta} D(z) \int_{-\infty}^{\infty} x \nabla_\theta P_z^\theta(x) \cdot dx \cdot dz\right] - 0 = \nabla_\theta J_\alpha^\theta(R),
\end{aligned} \tag{14}$$

which completes the proof. Notice that we used the identity $\nabla_\theta \int_{-\infty}^{\infty} P_z^\theta(x) dx = \nabla_\theta 1 = 0$; this does not hold for the analogous term in the standard RL settings in Eq. (10), $\nabla_\theta \int_{-\infty}^{q_\alpha^\theta(x)} P^\theta(x) dx$, whose gradient depends on $\nabla_\theta q_\alpha^\theta(x)$ according to the Leibniz integral rule.

□

B Proof of Proposition 1

Proof. Recall that by Eq. (8), $G = \frac{1}{\alpha N} \sum_{i=1}^{N} \mathbf{1}_{R_i \le \hat{q}_\alpha^\theta} \sum_{m=1}^{M} g_{i,m}$. Denoting $G_i = \sum_{m=1}^{M} g_{i,m}$ and substituting $\hat{q}_\alpha^\theta = q_\alpha^\theta$, we have

$$G = \frac{1}{N} \sum_{i=1}^{N} \alpha^{-1} \mathbf{1}_{R_i \le q_\alpha^\theta} G_i.$$

Expectation: Since $\{G_i\}$ are i.i.d, and using the law of total probability, we obtain

$$\begin{aligned}
\mathbb{E}_D[\alpha^{-1} \mathbf{1}_{R_i \le q_\alpha^\theta} G_i] =& \alpha \cdot \left(\alpha^{-1} \cdot 1 \cdot \mathbb{E}_D[G_1 \,|\, R_1 \le q_\alpha^\theta]\right) + (1-\alpha) \cdot \left(\alpha^{-1} \cdot 0 \cdot \mathbb{E}_D[G_1 \,|\, R_1 > q_\alpha^\theta]\right) \\
=& \mathbb{E}_D[G_1 \,|\, R_1 \le q_\alpha^\theta].
\end{aligned}$$

By switching the task sample distribution to D_α^θ, and using the definition of D_α^θ, we simply have

$$\mathbb{E}_{D_\alpha^\theta}[\alpha^{-1} \mathbf{1}_{R_i \le q_\alpha^\theta} G_i] = \alpha^{-1} \mathbb{E}_{D_\alpha^\theta}[G_1] = \alpha^{-1} \mathbb{E}_D[G_1 \,|\, R_1 \le q_\alpha^\theta].$$

Together, we obtain $\mathbb{E}_{D_\alpha^\theta}[\alpha G] = \mathbb{E}_D[G]$ as required.

Variance: For the original distribution, since $\{G_i\}$ are i.i.d, we have

$$\begin{aligned}
N \cdot \mathrm{Var}_D(G) &= \mathrm{Var}_D(\alpha^{-1} \mathbf{1}_{R_1 \le q_\alpha^\theta} G_1) \\
&= \mathbb{E}_D[\alpha^{-2} \mathbf{1}_{R_1 \le q_\alpha^\theta} G_1^2] - \mathbb{E}_D[\alpha^{-1} \mathbf{1}_{R_1 \le q_\alpha^\theta} G_1]^2 \\
&= \mathbb{E}_D[\alpha \alpha^{-2} G_1^2 \,|\, R_1 \le q_\alpha^\theta] - \mathbb{E}_D[\alpha \alpha^{-1} G_1 \,|\, R_1 \le q_\alpha^\theta]^2 \\
&= \alpha^{-1} \mathbb{E}_{D_\alpha^\theta}[G_1^2] - \mathbb{E}_{D_\alpha^\theta}[G_1]^2 \\
&\ge \alpha^{-1} (\mathbb{E}_{D_\alpha^\theta}[G_1^2] - \mathbb{E}_{D_\alpha^\theta}[G_1]^2) \\
&= \alpha^{-1} \mathrm{Var}_{D_\alpha^\theta}(G_1).
\end{aligned}$$

For the tail distribution D_α^θ, however,

$$N \cdot \mathrm{Var}_{D_\alpha^\theta}(\alpha G) = \alpha^2 \mathrm{Var}_{D_\alpha^\theta}(\alpha^{-1} \mathbf{1}_{R_1 \le q_\alpha^\theta} G_1) = \mathrm{Var}_{D_\alpha^\theta}(G_1),$$

which completes the proof.

□

C The Cross Entropy Method

C.1 Background

The Cross Entropy Method (CEM, de Boer et al. [2005]) is a general approach to rare-event sampling and optimization. In this work, we use its sampling version to sample high-risk tasks from the tail of D. As described in Section 5, the CEM repeatedly samples from the parameterized distribution D_ϕ, and updates ϕ according to the β-tail of the sampled batch. Since every iteration focuses on the tail of its former, we intuitively expect exponential convergence to the tail of the original distribution. While theoretical convergence analysis does not guarantee the exponential rate [de Mello and Rubinstein, 2003], practically, the CEM often converges within several iterations. For clarity, we provide the pseudo-code for the basic CEM in Algorithm 3. In this version, the CEM repeatedly generates samples from the tail of the given distribution D_{ϕ_0}.

Algorithm 3: The Cross Entropy Method (CEM)

Input: distribution D_{ϕ_0}; score function R; target level q; batch size N; CEM quantile β.

$\phi \leftarrow \phi_0$
while true **do**
    // Sample
    Sample $z \sim D_\phi^N$
    $w_i \leftarrow D_{\phi_0}(z_i)/D_\phi(z_i) \quad (1 \leq i \leq N)$
    Print z
    // Update
    $q' \leftarrow \max\left(q,\ q_\beta\left(\{R(z_i)\}_{i=1}^N\right)\right)$
    $\phi \leftarrow \text{argmax}_{\phi'} \sum_{i=1}^N w_i \mathbf{1}_{R(z_i)\leq q'} \log D_{\phi'}(z_i)$

C.2 Discussion

The CEM is the key to the flexible robustness level of RoML (Algorithm 2): it can learn to sample not only the single worst-case task, but all the α tasks with the lowest returns.

The CEM searches for a task distribution within a parametric family of distributions. This approach can handle infinite task spaces, and learn the difficulty of tasks in the entire task space from a mere finite sample of tasks. For example, assume that the tasks correspond to environment parameters that take continuous values within some bounded box (as in Section 6.2 and Appendix E). The CEM can fit a distribution over an entire subset of the box – from a mere finite batch of tasks. This property lets the CEM learn the high-risk tasks quickly and accelerates the meta-training, as demonstrated in Section 6 and Appendix D.3.

On the other hand, this approach relies on the structure in the task space. If the tasks do not have a natural structure like the ones in the bounded box, it is not trivial to define the parametric family of distributions. This is the case in certain supervised meta learning problems. For example, in the common meta-classification problem [Finn et al., 2017], the task space consists of subsets of classes to classify. This is a discrete space without a trivial metric between tasks. Hence, it is difficult for the CEM to learn the α lowest-return tasks from a finite sample. Thus, while RoML is applicable to supervised meta learning as well as MRL, certain task spaces require further adjustment, such as a meaningful embedding of the task space. This challenge is left for future work.

D Experiments: Detailed Settings and Results

D.1 Khazad Dum

> *At the end of the hall the floor vanished and fell to an unknown depth. The outer door could only be reached by a slender bridge of stone, without kerb or rail, that spanned the chasm with one curving spring of fifty feet. It was an ancient defence of the Dwarves against any enemy that might capture the First Hall and the outer passages. They could only pass across it in single file.* [Tolkien, 1954]

The detailed settings of the Khazad Dum environment presented in Section 6.1 are as follows. Every task is carried for $K = 4$ episodes of $T = 32$ times steps. The return corresponds to the undiscounted sum of the rewards ($\gamma = 1$). Every time step has a cost of $1/T$ points if the L1-distance of the agent from the target is larger than 5; and for distances between 0 and 5, the cost varies linearly between 0 and $1/T$. By reaching the destination, the agent obtains a reward of $5/T$, and has no more costs for the rest of the episode. By falling to the abyss, the agent can no longer reach the goal and is bound to suffer a cost of $1/T$ for every step until the end of the episode. Every step, the agent observes its location (represented using a soft one-hot encoding, similarly to Greenberg et al. [2022]) and chooses whether to move left, right, up or down. If the agent attempts to move into a wall, it remains in place.

The tasks are characterized by the rain intensity, distributed $\tau \sim Exp(0.1)$. The rain only affects the episode when the agent crosses the bridge: then, the agent suffers from an additive normally-distributed action noise $\mathcal{N}(0, \tau^2)$, in addition to a direct damage translated into a cost of $3 \cdot \tau$. For RoML, the CEM is implemented over the exponential family of distributions $Exp(\phi)$ with $\phi_0 = 0.1$ and $\beta = 0.05$. In this toy benchmark we use no regularization ($\nu = 0$).

In addition to the test returns throughout meta-training shown in Fig. 3, Fig. 7 displays the final test returns at the end of the meta-training, over 30 seeds and 3000 test tasks per seed.

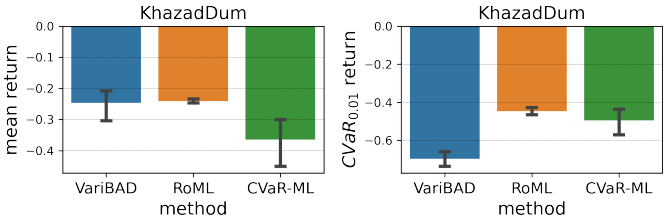


Figure 7: Khazad-Dum: Mean and CVaR returns over 30 seeds and 3000 test tasks.

D.2 Continuous Control

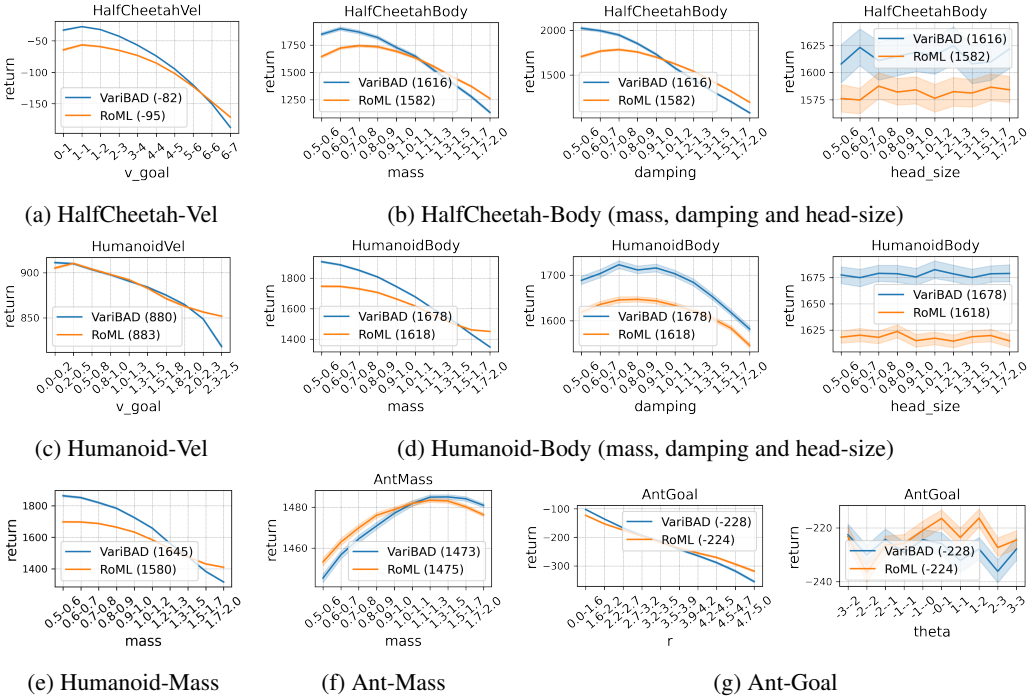
Below we describe the task distributions in the MuJoCo benchmarks introduced in Section 6.2, as well as their parameterization for the CEM. In each benchmark, we used CEM quantile $\beta = 0.2$ and regularization of $\nu = 0.2$ samples per batch.

- **HalfCheetah-Vel**: The original task distribution is uniform $\tau \sim U([0, 7])$ in HalfCheetah (the task space $[0, 7]$ was extended in comparison to Zintgraf et al. [2019], to create a more significant tradeoff between tasks). We rewrite it as $\tau = 7\tilde{\tau}$, $\tilde{\tau} \sim Beta(2\phi, 2 - 2\phi)$ with $\phi_0 = 0.5$ (leading to $Beta(1, 1)$, which is indeed the uniform distribution). The CEM learns to modify ϕ. Notice that this parameterization satisfies $\mathbb{E}_{D_\phi}[\tilde{\tau}] = \phi$.
- **Humanoid-Vel**: Same as HalfCheetah-Vel, with task space $[0, 2.5]$ instead of $[0, 7]$.
- **Ant-Goal**: The target location is random within a circle of radius 5. We represent the target location in polar coordinates, and write $r \sim Beta(2\phi_1, 2-2\phi_1)$ and $\theta \sim Beta(2\phi_2, 2-2\phi_2)$ (up to multiplicative factors 5 and 2π). The original distribution parameter is $\phi_0 = (0.5, 0.5)$, and the CEM learns to modify it.
- **HalfCheetah-Mass, Humanoid-Mass, Ant-Mass**: The task $\tau \in [0.5, 2]$ corresponds to the multiplicative factor of the body mass (e.g., $\tau = 2$ is a doubled mass). The original task distribution is uniform over the log factor, i.e., $\log_2 \tau \sim U([-1, 1])$. Again, we re-parameterize the uniform distribution as $Beta$, and learn to modify its parameter.
- **HalfCheetah-Body, Humanoid-Body, Ant-Body**: The 3 components of the task correspond to multiplicative factors of different physical properties, and each of them is distributed independently and uniformly in log, i.e., $\forall 1 \leq j \leq 3 : \log_2 \tau_j \sim U([-1, 1])$. We re-parameterize this as 3 independent $Beta$ distributions with parameters $\phi = (\phi_1, \phi_2, \phi_3)$.
- **HalfCheetah 10D-task**: Again, the task components correspond to multiplicative factors of different properties of the model. This time, there are 10 different properties (i.e., the task space is 10-dimensional), but each of them varies in a smaller range: $\log_2 \tau_j \sim U([-0.5, 0.5])$. Each such property is a vector, and is multiplied by τ_j when executing

Table 2: Mean return over 1000 test tasks, for different models and MuJoCo environments. Standard deviation is presented over 30 seeds. CVaR returns are displayed in Table 1.

	HalfCheetah			HalfCheetah 10D-task		
	Vel	Mass	Body	(a)	(b)	(c)
CeSoR	-1316 ± 18	1398 ± 31	1008 ± 34	1222 ± 23	1388 ± 20	1274 ± 32
PAIRED	-545 ± 55	662 ± 30	492 ± 51	551 ± 53	706 ± 36	561 ± 65
CVaR-ML	-574 ± 22	113 ± 8	193 ± 6	263 ± 15	250 ± 11	192 ± 5
PEARL	-534 ± 15	$\mathbf{1726 \pm 13}$	$\mathbf{1655 \pm 6}$	1843 ± 9	1866 ± 13	1425 ± 6
VariBAD	$\mathbf{-82 \pm 2}$	1558 ± 32	1616 ± 28	$\mathbf{1893 \pm 6}$	$\mathbf{1984 \pm 67}$	$\mathbf{1617 \pm 12}$
RoML (VariBAD)	-95 ± 3	1581 ± 32	1582 ± 21	1819 ± 8	$\mathbf{1950 \pm 20}$	$\mathbf{1616 \pm 13}$
RoML (PEARL)	-519 ± 15	1553 ± 18	1437 ± 8	1783 ± 7	1859 ± 10	1399 ± 8

	Humanoid			Ant		
	Vel	Mass	Body	Goal	Mass	Body
VariBAD	$\mathbf{880 \pm 4}$	$\mathbf{1645 \pm 22}$	$\mathbf{1678 \pm 17}$	$\mathbf{-229 \pm 3}$	$\mathbf{1473 \pm 3}$	$\mathbf{1476 \pm 1}$
RoML (VariBAD)	$\mathbf{883 \pm 4}$	1580 ± 17	1618 ± 18	$\mathbf{-224 \pm 3}$	$\mathbf{1475 \pm 2}$	1472 ± 1



(a) HalfCheetah-Vel (b) HalfCheetah-Body (mass, damping and head-size)

(c) Humanoid-Vel (d) Humanoid-Body (mass, damping and head-size)

(e) Humanoid-Mass (f) Ant-Mass (g) Ant-Goal

Figure 8: Average return per range of tasks in the various MuJoCo environments (global average in parentheses). HalfCheetah-Mass is displayed in Fig. 4c.

the task τ. The 10 properties are selected randomly, among all the variables of type *float ndarray* in *env.model*. We generate 3 such MRL environments – with 3 different random sets of 10 task-variables each. Some examples for properties are inertia, friction and mass.

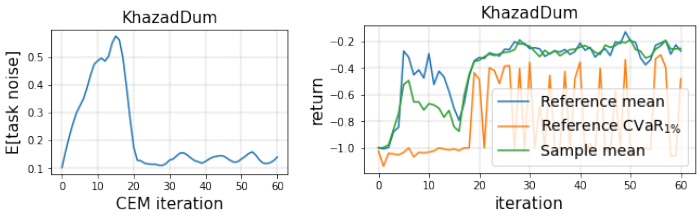
In the experiments, we rely on the official implementations of VariBAD [Zintgraf et al., 2019] and PEARL [Rakelly et al., 2019], both published under the MIT license. CVaR-ML and RoML are implemented on top of these baseline, and their running times are indistinguishable from the baselines. All experiments were performed on machines with Intel Xeon 2.2 GHZ CPU and NVIDIA's V100 GPU. Each experiment (meta-training and testing) required 12-72 hours, depending on the environment and the baseline algorithm.

Table 2 and Fig. 8 present detailed results for our MuJoCo experiments, in addition to the results presented in Section 6.2.

D.3 The Cross Entropy Method

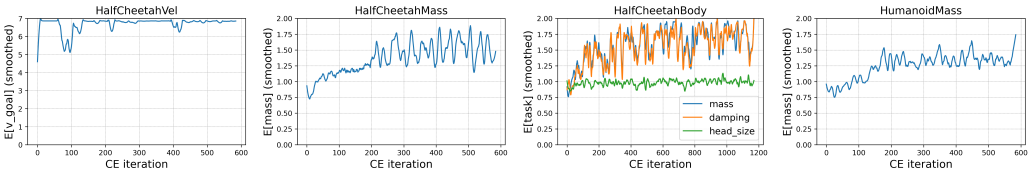
We implemented RoML using the Dynamic Cross Entropy Method implementation of Greenberg [2022]. Below we concentrate results related to the CEM functionality in all the experiments:

- **Learned sample distribution**: One set of figures corresponds to the learned sample distribution, as measured via ϕ throughout the meta-training.
- **Sample returns**: A second set of figures corresponds to the returns over the sampled tasks (corresponding to D_ϕ): ideally, we would like them to align with the α-tail of the reference returns (corresponding to D_{ϕ_0}). Thus, the figures present the mean sample return along with the mean and CVaR reference returns (the references are estimated from the sample returns using Importance Sampling weights). In most figures, we see that the sample returns (in green) shift significantly from the mean reference return (in blue) towards the CVaR reference return (orange), at least for part of the training. Note that in certain environments, the distinction between difficulty of tasks can only be made after the agent has already learned a basic meaningful policy, hence the sample returns do not immediately deviate from the mean reference.



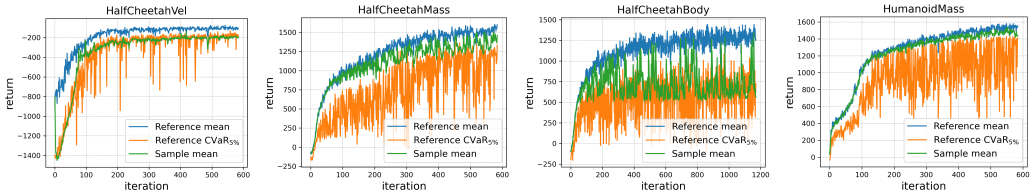
(a) Learned sample distribution (b) Sample returns

Figure 9: The CEM in Khazad-Dum. Note that the effect of the CEM concentrates at the first half of the meta-training; once the meta-policy learns to focus on the long path, the agent becomes invariant to the sampled tasks, and the sampler gradually returns to the original task distribution $\phi \approx \phi_0$.



(a) HalfCheetah-Vel (b) HalfCheetah-Mass (c) HalfCheetah-Body (d) Humanoid-Mass

Figure 10: Learned sample distribution in MuJoCo benchmarks. Notice that for HalfCheetah-Body, the CEM has to control 3 different task parameters simultaneously.



(a) HalfCheetah-Vel (b) HalfCheetah-Mass (c) HalfCheetah-Body (d) Humanoid-Mass

Figure 11: Sample returns in MuJoCo benchmarks.

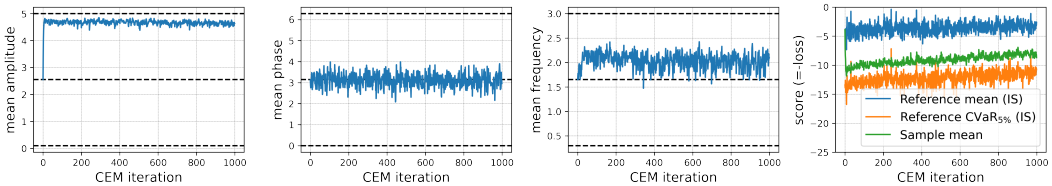


Figure 12: Sine Regression: The 3 left figures show the learned sample distribution, corresponding to average amplitudes, phases and frequencies. The CEM immediately learns that the amplitude has the strongest effect on test loss, whereas frequency has a moderate effect and phase has none. The right figure shows in green the mean sample scores (in supervised learning these are the negative losses instead of the returns).

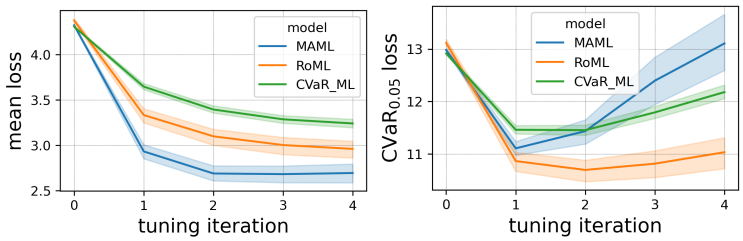
E Supervised Meta-Learning

Our work focuses on robustness to difficult tasks in MRL. However, the concept of training on harder data to improve robustness, as embodied in RoML, is applicable beyond the scope of RL. As a preliminary proof-of-concept, we apply RoML to a toy supervised meta-learning problem of sine regression, based on Finn et al. [2017].

In this problem, the input is $x \in [0, 2\pi)$, the desired output is $y = A\sin(\omega x + b)$, and the task is defined by the parameters $\tau = (A, b, \omega)$, distributed uniformly over $\Omega = [0.1, 5] \times [0, 2\pi] \times [0.3, 3]$. Similarly to Finn et al. [2017], the model is fine-tuned for each task via a gradient-descent optimization step over 10 samples $\{(x_i, y_i)\}_{i=1}^{10}$, and is tested on another set of 10 samples. The goal is to find model weights that adapt quickly to new task data – in expectation over tasks.

We implement CVaR-ML and RoML on top of the MAML algorithm [Finn et al., 2017], with $\alpha = 0.05$. For the CEM of RoML, we re-parameterize the uniform task distribution using Beta distributions (see details below). As shown in Fig. 12 in the appendix, RoML learns to focus on tasks (sine functions) with high amplitudes and slightly increased frequencies, without changing the phase distribution. Fig. 13 displays the test losses over 30 seeds, after meta-training for 10000 tasks. Again, RoML achieves better CVaR losses than both CVaR-ML and the baseline.

In this problem, the task space Ω induces a natural structure on the tasks, as we can tell which tasks are "close" to each other. The CEM takes advantage of this structure to characterize high-risk tasks – even though it can never observe the infinitely many possible tasks. The applicability of CEM to supervised learning in general is discussed in Appendix C.2.



(a) Mean (b) $\texttt{CVaR}_{0.05}$

Figure 13: Sine Regression: Mean and CVaR losses over 10000 test tasks, against the number of tuning gradient-steps at test time. The 95% confidence intervals are calculated over 30 seeds.

Additional implementation details: In comparison to Finn et al. [2017], we added the sine frequency as a third parameter in the task space, since the original problem was to simplistic to pose a mean/CVaR tradeoff. The tasks are distributed uniformly, and we reparameterize them for the CEM using the Beta distribution, similarly to Appendix D.2: $\tau = \{\tau_j\}_{j=1}^{3}$, $\tau_j \sim Beta(2\phi_j, 2 - 2\phi_j)$. On top of this, we add a linear transformation from the Beta distribution domain $[0, 1]$ to the actual task range ($[0.1, 5]$ for amplitude, $[0, 2\pi]$ for phase and $[0.3, 3]$ for frequency). Note that the original uniform distribution is recovered by $\phi_0 = (0.5, 0.5, 0.5)$. The parameter $\phi_j \in [0, 1]$, which is controlled by the CEM, equals the expected task $\mathbb{E}_{\tau \sim D_\phi}[\tau_j]$. The other hyper-parameters of RoML are set to $\beta = 0.2$ (CEM quantile) and $\nu = 0$ (no regularization).