In Search for Architectures and Loss Functions in Multi-Objective Reinforcement Learning

Mikhail Terekhov
CLAIRE, EPFL
mikhail.terekhov@epfl.ch

Caglar Gulcehre
CLAIRE, EPFL
caglar.gulcehre@epfl.ch

Abstract

Multi-objective reinforcement learning (MORL) is essential for addressing the intricacies of real-world RL problems, which often require trade-offs between multiple utility functions. However, MORL is challenging due to unstable learning dynamics with deep learning-based function approximators. The research path most taken has been to explore different **value**-based loss functions for MORL to overcome this issue. Our work empirically explores model-free **policy** learning loss functions and the impact of different architectural choices. We introduce two different approaches: *Multi-objective Proximal Policy Optimization* (MOPPO), which extends PPO to MORL, and *Multi-objective Advantage Actor Critic* (MOA2C), which acts as a simple baseline in our ablations. Our proposed approach is straightforward to implement, requiring only small modifications at the level of function approximator. We conduct comprehensive evaluations on the MORL *Deep Sea Treasure*, *Minecart*, and *Reacher* environments and show that MOPPO effectively captures the Pareto front. Our extensive ablation studies and empirical analyses reveal the impact of different architectural choices, underscoring the robustness and versatility of MOPPO compared to popular MORL approaches like Pareto Conditioned Networks (PCN) and Envelope Q-learning in terms of MORL metrics, including hypervolume and expected utility.

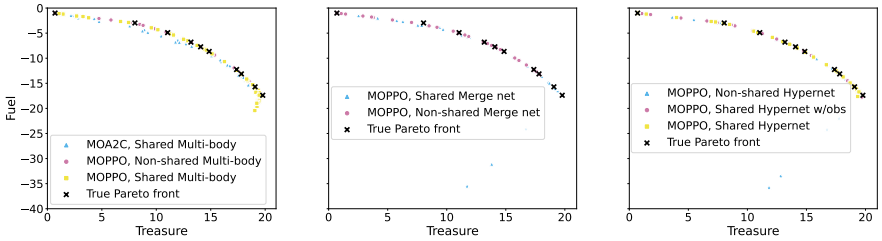
1 Introduction

Many optimization problems in the real world require consideration of multiple conflicting objectives. Liu and Vicente [2022] provide examples of accuracy versus fairness trade-offs in credit scoring and criminal justice, and Vamplew et al. [2021] show how to trade performance for safety in intelligent agents. *Multi-objective optimization* is the field that studies these problems formally. It is known as *multi-objective reinforcement learning (MORL)* in the sequential decision-making setting. In MORL, we seek *policies* that maximize the respective objectives. A single policy mapping states to actions is insufficient to satisfy all possible trade-offs between objectives; hence, in MORL, we usually discuss sets of policies covering these trade-offs.

The performance of modern MORL approaches is often measured on toy grid-world or 2D locomotion problems, such as those in MO-Gym by Alegre et al. [2022]. At the same time, single-objective RL is already used in many practical applications, such as language modeling [Ouyang et al., 2022], real-world robot locomotion [Fu et al., 2023], and control of scientific equipment [Degrave et al., 2022]. One plausible explanation of this gap is that MORL approaches often explicitly store Pareto-optimal policies or rely on Q-learning. For toy problems, this provides optimal coverage, but this is not scalable, and we noticed that the training can suffer from unstable learning dynamics, especially when different rewards interfere with each other.

Rather than maintaining a set of weights for each trade-off, we implicitly model the optimal set of policies by conditioning the learned policy on each objective's vector of *relative weights*. We call this

38th Workshop on Aligning Reinforcement Learning Experimentalists and Theorists (ARLET 2024).



(a) Multi-body architectures (b) Merge net architectures (c) Hypernet architectures

Figure 1: **Pareto fronts on *Deep Sea Treasure*:** Performance of a selection of our methods on the Deep Sea Treasure environment, split by the actor-critic architecture and the learning algorithm. The details of the architectures and algorithms are described in Sections 4 and 5, respectively. In this simple gridworld, the agent's task is to find the biggest treasure, but big treasures require it to spend more fuel. The tension between the two objectives is formalized as a *Pareto front* of the problem. Our proposed approaches effectively cover the true Pareto front. Some of the methods produce a few outliers because the policy struggles to learn near the boundary of the simplex Δ_K of reward weights, where one of the rewards (in this case, fuel) is completely discarded.

approach *Dynamic MORL (DMORL)*. This allows us to learn a **single model** *that encompasses all possible solutions on the Pareto front*. Our approach relies on *linear scalarization* to model trade-offs. Thus, the policy learns to optimize a convex combination of the objectives and is conditioned on the coefficients of this combination. Despite the theoretical limitations of linear scalarization in MORL [Vamplew et al., 2008], we find that DMORL when paired with a sufficiently expressive neural network, can generate a continuous parameterization of the entire Pareto front. In this paper, we investigate two components needed to scale MORL for modern reinforcement learning tasks: the learning algorithm and the architecture. For the DMORL framework in this paper, we mainly focus on PPO [Schulman et al., 2017] and generalize it to the multi-objective case. As a baseline, we introduce a multi-objective version of A2C [Mnih et al., 2016]. We investigate multiple actor-critic architectures, including a multi-body network, merge networks for relative weights, and hypernetworks. We also normalize the rewards using the PopArt scheme [Hessel et al., 2019] and propose a novel method to control the entropy during training. An example Pareto front produced by our methods on a simple test environment is shown in Figure 1. To summarize, our main contributions are:

- We propose a scalable family of algorithms for multi-objective on-policy RL.
- We propose and evaluate different actor-critic architectures for multi-objective RL.
- We describe a method to control the policy's entropy during training for MORL dynamically and show that it improves learning stability.
- We demonstrate the effectiveness of our proposed approaches to DMORL on both deterministic and stochastic MORL benchmark environments.

2 Related work

Multi-objective RL has been primarily studied from an off-policy perspective. Multiple MORL approaches have been developed as extensions of Q-learning [Abels et al., 2019, Lu et al., 2022], some of which condition the Q-network on the scalarization weights as we do with policies. Among these methods, we use Envelope Q-learning [Yang et al., 2019] as a baseline, since its implementation is publicly available and includes the case of discrete actions, which is the focus of our work. Another notable example is the recent work by Hung et al. [2022], which also proposes a way to perform policy updates for policies conditioned on relative weights along with learning a Q-function, but the implementation only considers continuous control. Although Q-learning thrives in toy problems and is efficient in some more complex domains [Mnih et al., 2013], it has many failure modes, including the so-called "deadly triad" [Van Hasselt et al., 2018]. Off-policy methods also struggle with capacity loss [Lyle et al., 2022] and, more generally, with generalizable feature learning [Lan et al., 2022]. Our work instead focuses on on-policy methods.

Alegre et al. [2023] also propose a sample-efficient method for MORL, but they employ model-based learning while we focus on the model-free setup. Roijers et al. [2018], Reymond et al. [2023] directly optimize the policy in the model-free setting (the latter work also includes a critic), but target a single non-linear utility function. Xu et al. [2020], similarly to us, apply PPO to the multi-objective case, but they do not condition the policies on the utility function, instead maintaining a set of policies explicitly. MOMPO by Abdolmaleki et al. [2020] is an approach to MORL using techniques from distributional RL [Levine, 2018]. In contrast to our method, MOMPO requires multiple training runs to cover the Pareto front. One of the most relevant approaches to ours is called *Pareto Conditioned Networks (PCN)* by Reymond et al. [2022]. It operates in the DMORL setup and learns a policy conditioned on the desired value for all objectives. This allows for PCN to generalize to non-convex PF. This work draws from reward-conditioned policies (RCP) by Kumar et al. [2019], a recent supervised learning algorithm for RL. However, due to the nature of its replay buffer, PCN is not adapted to stochastic environments.

3 Dynamic multi-objective reinforcement learning

Multi-objective reinforcement learning is the search for optimal policies for multi-objective Markov decision processes (MOMDP). A MOMDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathbf{r}, P, \kappa, \gamma)$, where sets $\mathcal{S}$ and $\mathcal{A}$ are state and action spaces respectively, $\mathbf{r} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^K$ is a K-dimensional reward function, and $P(s' \mid s, a)$ is the transition probability. Finally, κ is the distribution of initial states, and $\gamma \in [0, 1)$ is the discount factor. The only difference between MOMDP and MDP is the range of the reward — $\mathbb{R}^K$ instead of $\mathbb{R}$. In this work, we optimize so-called linear scalarizations with relative weights given by $\boldsymbol{\alpha} \in \Delta_K$ from the $(K-1)$-dimensional simplex

$$\Delta_K = \left\{ \boldsymbol{\alpha} \in \mathbb{R}^K \mid \sum_{i=1}^{K} \alpha_i = 1,\ \alpha_i \geq 0\ \forall i \right\}. \tag{1}$$

For $\boldsymbol{\alpha} \in \Delta_K$, a *scalarized reward* is $r(s, a, \boldsymbol{\alpha}) = \boldsymbol{\alpha}^\top \mathbf{r}(s, a)$. Since we would like to cover all scalarizations with a single model, we have to generalize our policy definition so that it is also conditioned on $\boldsymbol{\alpha}$. Hence, our parameterized policies are of the form $\pi(a \mid s, \boldsymbol{\alpha})$. For a policy π, the vector-value function also has to depend on $\boldsymbol{\alpha}$:

$$\mathbf{V}^\pi(s, \boldsymbol{\alpha}) = \mathbb{E}_{\tau \sim p^\pi(\tau \mid \boldsymbol{\alpha})} \left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}(s_t, a_t) \middle| s_0 = s \right] \in R^K. \tag{2}$$

Here, $\tau = (s_0, a_0, s_1, \ldots)$ is a trajectory sampled from the transition dynamics P and the policy $\pi(a \mid s, \boldsymbol{\alpha})$. We can also define the Q-function and the advantage in the usual way:

$$\mathbf{Q}^\pi(s, a, \boldsymbol{\alpha}) = \mathbf{r}(s, a) + \gamma \mathbb{E}_{s' \sim P(s' \mid s, a)} \left[\mathbf{V}^\pi(s', \boldsymbol{\alpha}) \right], \tag{3}$$

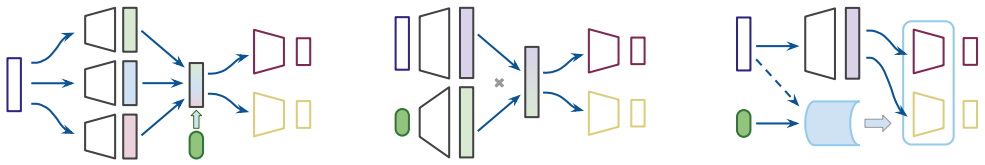
$$\mathbf{A}^\pi(s, a, \boldsymbol{\alpha}) = \mathbf{Q}^\pi(s, a, \boldsymbol{\alpha}) - \mathbf{V}^\pi(s, \boldsymbol{\alpha}). \tag{4}$$

The vector-objective associated with π is then given by

$$\mathbf{J}(\pi, \boldsymbol{\alpha}) = \mathbb{E}_{s \sim \kappa} \left[\mathbf{V}^\pi(s, \boldsymbol{\alpha}) \right] \in \mathbb{R}^K. \tag{5}$$

In practice, our policies will be represented by a neural architecture with parameters $\theta \in \mathbb{R}^{D_a}$. In this case, we write the policy, a.k.a. the *actor*, as $\pi_\theta(a|s, \boldsymbol{\alpha})$ and use shortcuts $\mathbf{V}^\theta = \mathbf{V}^{\pi_\theta}$ and $\mathbf{J}(\theta, \boldsymbol{\alpha}) = \mathbf{J}(\pi_\theta, \boldsymbol{\alpha})$. MOPPO and MOA2C will also require a *critic*, i.e., a neural approximation to the value function. It is parameterized by $\psi \in \mathbb{R}^{D_c}$, and we denote it as $\widetilde{\mathbf{V}}_\psi(s, \boldsymbol{\alpha})$. Some of the parameters might be shared between the actor and the critic. Our architecture's overall set of parameters will be called ν.

Pareto Front To define a Pareto Front, we need the notion of strict dominance. We say that a vector $\mathbf{p} \in \mathbb{R}^K$ *strictly dominates* $\mathbf{q} \in \mathbb{R}^K$ (denoted as $\mathbf{p} \succ \mathbf{q}$) iff $p_i > q_i$ for all $i \in \{1, \ldots, K\}$. Next, individual parameters $\boldsymbol{\alpha}$ give rise to partially specified policies $\pi(\cdot \mid \cdot, \boldsymbol{\alpha})$. We say that a policy $\pi(\cdot \mid \cdot, \boldsymbol{\alpha})$ strictly dominates $\pi(\cdot \mid \cdot, \boldsymbol{\alpha}')$ iff $\mathbb{E}_{s \sim \kappa}\big[\mathbf{V}^\pi(s, \boldsymbol{\alpha})\big] \succ \mathbb{E}_{s \sim \kappa}\big[\mathbf{V}^\pi(s, \boldsymbol{\alpha}')\big]$. A partially specified policy $\pi(\cdot \mid \cdot, \boldsymbol{\alpha})$ is said to be *weakly Pareto-optimal* if no other policy dominates it. The set of all such policies is called a *weak Pareto set*, and the image of the Pareto set in the objective space is called a *weak Pareto front* (PF).



(a) **Multi-body**. State s_t is passed through individual embeddings, outputs are interpolated using $\boldsymbol{\alpha}$. Then, the interpolated outputs are passed to the actor and critic heads.

(b) **Merge net**. State s_t and $\boldsymbol{\alpha}$ are passed through embeddings and multiplied. This is then passed to the actor and critic.

(c) **Hypernetwork**. State s_t is embedded, and $\boldsymbol{\alpha}$ is passed through a hypernetwork producing the actor and critic heads.

Figure 2: **Actor-critic architectures with shared trunks:** Non-shared versions are organized similarly. The dashed line in the hypernetwork chart is optional: s_t can be passed into the hypernetwork, where we get the architecture's "hypernet w/obs" variant.

In a nutshell, to identify the points on the PF, we are optimizing the *expected utility metric*, introduced by Zintgraf et al. [2015] as

$$J(\theta) = \mathbb{E}_{\boldsymbol{\alpha}\sim\mathcal{D}(\boldsymbol{\alpha})}\left[\boldsymbol{\alpha}^\top \mathbf{J}(\theta, \boldsymbol{\alpha})\right] \tag{6}$$

for some distribution $\mathcal{D}$ over Δ_K with full support. Currently, we always set $\mathcal{D}$ to be uniform, but other distributions can also be relevant. For a convex PF, if the neural architecture is expressive enough so that the entire PF is covered by a set of weights θ^*, it is easy to see that θ^* will also be a maximizer of (6) for any distribution $\mathcal{D}$. The theoretical situation is more involved in the misspecified case (when the optimal policies are not represented by any θ) since the maximizer depends on the exact distribution $\mathcal{D}$. Still, we will not study our approach from the theoretical perspective.

We evaluate our models with two of the most common metrics in the MORL literature: expected utility (6) and hypervolume.

Hypervolume Given a Pareto front $\mathcal{P}$ and a reference point $\mathbf{p}_0$, the hypervolume $HV(\mathcal{P})$ indicates the volume of the polytope in the region of the objective space dominated by $\mathcal{P}$ and bounded from below by the reference point. Formally, in K-dimensional objective space, it can be defined as:

$$HV(\mathcal{P}) = \int_{\mathbb{R}^K} \mathbb{I}_{H(\mathcal{P})}(\mathbf{z}) d\mathbf{z} \tag{7}$$

where $H(\mathcal{P}) = \{\mathbf{z} \in \mathcal{Z} \mid \exists \mathbf{p} \in \mathcal{P} : \mathbf{p}_0 \preceq \mathbf{z} \preceq \mathbf{p}\}$. Here, $\preceq$ is the relation operator that indicates the Pareto-dominance, and $\mathbb{I}_{H(\mathcal{P})}$ is the indicator function that is 1 if $\mathbf{z} \in H(\mathcal{P})$, and 0 otherwise.

4 Architectures

This work focuses on continuous states and discrete actions, such that $\mathcal{S} \subset \mathbb{R}^{d_s}, \quad \mathcal{A} = \{1, \ldots, A\}$. Generalization to small discrete state spaces is straightforward with one-hot encoding, and generalization to continuous actions can be performed analogously to scalar PPO and A2C variants through parameterized action distributions. We consider three architectures for the actor-critic network, presented in Figure 2.

When both actor and critic are learned, an often-used practice is to share weights between them. A popular approach is to share the *trunk* $\mathbf{f}_\zeta : \mathcal{S} \to \mathbb{R}^F$ of a neural network that takes the state s in and produces intermediate *features* $\mathbf{f}_\zeta(s)$. The trunk is parameterized by $\zeta \in \mathbb{R}^{D_b}$. We hope the features are general enough to convey information about both the optimal action and the value of the current state. In this case, we can use a *shared trunk* architecture with two separate linear layers to extract both the action distribution and the predicted value:

$$\pi_\theta(\cdot \mid s, \boldsymbol{\alpha}) = \text{softmax}(\mathbf{W}_a \mathbf{f}_\zeta(s, \boldsymbol{\alpha}) + \mathbf{b}_a), \tag{8}$$

$$\widetilde{\mathbf{V}}_\psi(s, \boldsymbol{\alpha}) = \mathbf{W}_c \mathbf{f}_\zeta(s, \boldsymbol{\alpha}) + \mathbf{b}_c. \tag{9}$$

The learnable parameters of the actor are therefore $\theta = (\zeta, \mathbf{W}_a, \mathbf{b}_a)$. For the critic, the parameters are $\psi = (\zeta, \mathbf{W}_c, \mathbf{b}_c)$. We refer to $(\mathbf{W}_a, \mathbf{b}_a)$ as the *actor head* and to $(\mathbf{W}_c, \mathbf{b}_c)$ as the *critic head*. An

alternative that we consider is that the critic and the actor do not share the trunk's parameters, even though its architecture is the same for both. We will refer to this as a *non-shared trunk* architecture.

When the policy is conditioned on states $s \in \mathcal{S}$ and relative weights $\boldsymbol{\alpha} \in \Delta_K$, it is a priori unclear what would be the best architecture that combines these inputs in $\mathbf{f}_\zeta$. Deep learning architectures provide a useful inductive bias to guide learning and cover the Pareto front efficiently. This can be especially helpful when the rewards interfere with each other, for example, when they compete, i.e., maximizing one reward minimizes the other[1]. As a result, the signs of each objective gradient maximizing each reward may differ, resulting in canceling each other in the shared trunk's parameters. This can disrupt the learning dynamics and representations. An effective way to address this in the continual multi-task learning has been to modify the architectures (see Rusu et al. [2016].) Here, we consider three architectures to understand their impact on learning policies for DMORL: a *multi-body network*, a *merge network*, and a *hypernetwork*.

Multi-body network. This architecture follows the intuition that the policies using different weights are, in some sense, interpolated. However, our preliminary experiments have shown that if we use multiple heads for actors corresponding to different objectives, the interpolation between logits or logprobs is not expressive enough. Instead, we need to interpolate in the higher-dimensional feature space. We thus settled on the formulation

$$\mathbf{f}_\zeta(s, \mathbf{w}) = \mathrm{MLP}\left(\sum_{i=1}^{K} \alpha_i \varphi\left(\mathbf{W}_i s + \mathbf{b}_i\right)\right). \tag{10}$$

Here, φ is the ReLU activation [Nair and Hinton, 2010], $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)^\intercal$, and MLP is a multi-layer perceptron. This architecture has a "body" $(\mathbf{W}_i, \mathbf{b}_i)$ for each objective, which independently processes the input state, and their outputs after activation are linearly interpolated by the weights $\boldsymbol{\alpha}$ assigned for each objective.

Merge network. Here, similarly to Reymond et al. [2022], we pass $\boldsymbol{\alpha}$ through a separate embedding before merging it with the state:

$$\mathbf{f}_\zeta(s, \mathbf{w}) = \mathrm{MLP}\left(\tilde{\varphi}(\mathbf{W}_1 s + \mathbf{b}_1) \odot \tilde{\varphi}(\mathbf{W}_1' \boldsymbol{\alpha} + \mathbf{b}_1')\right), \tag{11}$$

where $\tilde{\varphi}$ is the sigmoid activation, and $\odot$ stands for element-wise multiplication.

Hypernetwork. Finally, we consider a slightly more involved architecture similar to the one described in Navon et al. [2020]. Below, we detail the shared trunk case. The body of the policy $\mathbf{f}_\zeta(s)$ here only takes the state as input, i.e. $\mathbf{f}_\zeta(s) = \mathrm{MLP}(s)$. The last layer parameters $\mathbf{W}_a, \mathbf{b}_a, \mathbf{W}_c, \mathbf{b}_c$ are now produced by a hypernetwork with the structure

$$\mathbf{h}(\boldsymbol{\alpha}) = \varphi(\mathbf{W}_h \boldsymbol{\alpha} + \mathbf{b}_h), \quad \mathbf{W}_a, \mathbf{b}_a = \mathbf{W}_2 \mathbf{h} + \mathbf{b}_2, \quad \mathbf{W}_c, \mathbf{b}_c = \mathbf{W}_2' \mathbf{h} + \mathbf{b}_2'. \tag{12}$$

Above, we slightly abuse the notation in the sense that $\mathbf{W}_2 \mathbf{h} + \mathbf{b}_2$ is a vector of dimension $(F+1)|\mathcal{A}|$, which gets split and reshaped into the corresponding layer parameters $\mathbf{W}_a$ and $\mathbf{b}_a$. Note how the actor and the critic share the network's trunk and the first layer of the hypernetwork. With a non-shared trunk, the critic gets its copy of all parameter groups instead. We also experimented with a version of the shared trunk architecture where the observation gets fed into the hypernetwork along with the weights, i.e., $\mathbf{h}(\boldsymbol{\alpha}, s) = \varphi(\mathbf{W}_h[\boldsymbol{\alpha}; s] + \mathbf{b}_h)$, abbreviated as "Hypernetwork w/obs" in the paper.

5 Algorithms

We chose to focus on two instantiations of our framework for DMORL. However, our actor-critic architectures are more general and could be applied to other policy iteration or policy gradient methods.

5.1 Actor and critic losses

Multi-objective policy gradient Scalar A2C relies on the Policy Gradient theorem to update the actor. This theorem can be generalized to the vector-valued reward case:

$$\nabla_\theta(\boldsymbol{\alpha}^\top \mathbf{J}(\theta, \boldsymbol{\alpha})) = \mathbb{E}_{\tau \sim p^\pi(\tau|\boldsymbol{\alpha})}\left[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{\alpha}^\top \mathbf{A}^\theta(s_t, a_t, \boldsymbol{\alpha}) \nabla_\theta \log \pi_\theta(a_t \mid s_t, \boldsymbol{\alpha})\right]. \tag{13}$$

[1] In Minecart environment, rewards for fuel and ores can have competing dynamics.

To maximize the expected utility, we approximate the above expression by first sampling random relative weight vectors $\boldsymbol{\alpha} \sim \mathcal{D}(\Delta_K)$, and then performing a rollout of the current policy π^{θ} using $\boldsymbol{\alpha}$. In practice, we maintain B trajectories and corresponding $\boldsymbol{\alpha}$-s simultaneously, but in this section we set $B = 1$ for clarity. Let the discounted reward-to-go for the trajectory at timestep t be $\widehat{\mathbf{Q}}_t$. If the trajectory was truncated before the terminal state, reward-to-go is bootstrapped. The estimator of the gradient of expected utility that we use in multi-objective A2C is given by

$$\nabla_\theta J(\theta) \approx \widehat{\nabla}_\theta J(\theta) = \sum_{t=0}^{T} \gamma^t \boldsymbol{\alpha}^\intercal (\widehat{\mathbf{Q}}_t - \widetilde{\mathbf{V}}_\psi(s_t)) \nabla_\theta \log \pi_\theta(a_t \mid s_t, \boldsymbol{\alpha}). \tag{14}$$

Critic loss To optimize the critic over a minibatch of $\{(s_k, \widehat{\mathbf{Q}}_k)\}_k$, we use the least-squares loss:

$$l_c(\psi) = \sum_k \left\| \widehat{\mathbf{Q}}_k - \widetilde{\mathbf{V}}_\psi(s_k, \boldsymbol{\alpha}) \right\|^2. \tag{15}$$

The algorithm dynamically adjusts the weight β_c of the critic loss. Depending on whether the critic shares parameters with the actor, it makes sense to use different strategies for setting β_c. For the generality of our methods, we set β_c dynamically to ensure an approximately constant ratio of norms of the actor and critic gradients. This technique was not as important in our environments, so we defer a discussion to Appendix A.2.

Require: Multi-objective MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathbf{r}, P, \mu, \gamma)$
Require: Actor π_θ, critic $\widetilde{\mathbf{V}}_\psi$ with parameters $\nu = \theta \cup \psi$
while *not terminated* **do**
  Sample reward weights $\boldsymbol{\alpha} \sim U(\Delta_K)$
  Sample a truncated trajectory $\{(s_t, a_t, \mathbf{r}_t)\}_{t=0}^{T}$ from P and $\pi_\theta(\cdot \mid \cdot, \boldsymbol{\alpha})$
  for $e \leftarrow 1$ **to** E **do** `// iterate over epochs`
    Update PopArt $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ and the critic head
    Get the reward-to-go $\widehat{\mathbf{Q}}_t$ from bootstrapped TD(1)
    Compute vector- and scalarized advantage:
      $\hat{\mathbf{A}}_t \leftarrow \widehat{\mathbf{Q}}_t - \widetilde{\mathbf{V}}_\psi(s_t, \boldsymbol{\alpha}),\ \hat{A}_t \leftarrow \boldsymbol{\alpha}^\intercal(\hat{\mathbf{A}}_t / \boldsymbol{\sigma})$
    for $b \leftarrow 1$ **to** B **do**
      Sample a minibatch $\{(s_k, a_k, \widehat{\mathbf{Q}}_k, \hat{A}_k)\}_k$
      Update $\mathbf{g}$ and λ using (19) for entropy control
      Compute $l_a(\psi)$ and its gradient $\nabla_\theta l_a$ from (17)
      Compute $l_c(\psi)$ and its gradient $\nabla_\psi l_c$ from (15)
      Update β_c so that $\|\mathbf{g} + \nabla_\theta l_a\| \approx C\beta_c \|\nabla_\psi l_c\|$
      Feed $\mathbf{g} + \nabla_\theta l_a + \beta_c \nabla_\psi l_c$ into Adam, update ν
    end
  end
end

Algorithm 1: A schematic description of multi-objective PPO for an actor and a critic with a shared trunk. Our PopArt update scheme follows Hessel et al. [2019]. Actor-to-critic gradient ratio C is one of the hyperparameters we tune.

PopArt is an approach to learning value functions across different orders of magnitude, especially in situations where the value scales are not known in advance or change depending on the performance of the policy. Hessel et al. [2019] successfully adapted it to the multi-task RL setup, similar to MORL. In MORL, the differences in scale between rewards also pose a problem. If we have to design trade-offs between objectives of varying magnitude, then the optimization target (6) will unfairly favor the objectives of larger scales. Multi-task PopArt maintains an approximate mean μ_i and variance σ_i^2 of the target values for each task. We chose to use $\boldsymbol{\alpha}$ combine the normalized advantages. This means that for the gradient estimate (14) instead of $\boldsymbol{\alpha}^\intercal(\widehat{\mathbf{Q}}_{t,j} - \widetilde{\mathbf{V}}_\psi(s_{t,j}))$ we use a scalarization of the form

$$\boldsymbol{\alpha}^\intercal \left((\widehat{\mathbf{Q}}_{t,j} - \widetilde{\mathbf{V}}_\psi(s_{t,j})) / \boldsymbol{\sigma} \right), \tag{16}$$

where $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_K)$ and $/$ denotes component-wise division.

Multi-objective PPO The original PPO is formulated by Schulman et al. [2017] using a *surrogate objective* for policy iteration. This objective is optimized for multiple epochs over the same sampled trajectories to achieve higher sample efficiency. The actor loss for MOPPO differs from the standard PPO formulation only by using the scalarized advantage $\hat{A}$, defined in Algorithm 1. Given a minibatch of $\{(s_k, a_k, \hat{A}_k)\}_k$, we employ the following loss of the actor:

$$l_a(\theta) = \sum_k \min \left(r_k(\theta) \hat{A}_k, \text{clip}\left(r_k(\theta), 1-\varepsilon, 1+\varepsilon\right) \hat{A}_k \right), \quad r_k(\theta) = \frac{\pi_\theta(a_k | s_k, \boldsymbol{\alpha})}{\pi_{ref}(a_k | s_k, \boldsymbol{\alpha})}, \tag{17}$$

where π_{ref} is the policy directly after sampling the trajectory.

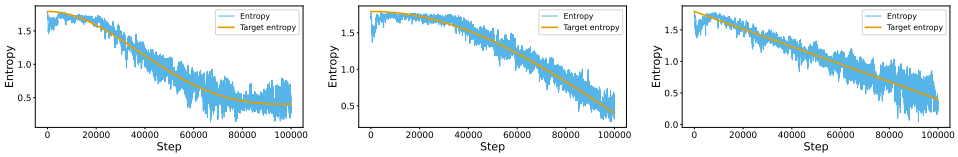


Figure 3: **Entropy control schedules:** Example entropy behaviors on Minecart when using the entropy control method described in Section 5.2. From left to right: custom schedule, cosine schedule and linear schedule of entropy. The custom schedule is designed to have a flat start for exploration and an extended flat end for fine-tuning the behavior. The schedules are discussed in detail in Appendix A.4.

5.2 Entropy control during training

Our experiments show that entropy regularization is necessary to avoid collapse in more challenging environments. The entropy of the current policy embodies the exploration-exploitation trade-off: if the entropy is low, the agent fails to explore, but if it is too high, the agent cannot operate efficiently. Intuitively, a good training run keeps entropy high initially to explore sufficiently and then "anneals" the entropy to lower values as training progresses. This behavior is hard to achieve in practice since training is sensitive to the entropy regularization coefficient. At this point, we can employ a key insight that we do not need to *maximize* the entropy. Rather, we would like to keep it at a pre-defined level. Hence, the framework of constrained optimization is applicable.

We use an algorithm by Platt and Barr [1987] called the *Modified Differential Method of Multipliers (MDMM)*, which allows us to maximize a function subject to approximate equality constraints. Let the expected entropy of the policy and its empirical estimate be

$$H(\theta) = \mathbb{E}_{\boldsymbol{\alpha}\sim\mathcal{D}(\Delta_K), s\sim p^{\pi_\theta}(s|\boldsymbol{\alpha})}\left[H(\pi_\theta(\cdot \mid s, \boldsymbol{\alpha}))\right], \quad \widehat{H}(\theta) = \frac{1}{T+1}\sum_{t=0}^{T} H(\pi_\theta(\cdot \mid s_t, \boldsymbol{\alpha})), \tag{18}$$

where p^{π_θ} is the state visitation distribution. Assume that we would like to satisfy the constraint $H(\theta) \approx H_{target}$, where H_{target} is a desired entropy value that can also depend on our progress in training. To this end, MDMM introduces the Lagrange multiplier $\lambda_i \in \mathbb{R}$, which is dynamically updated at each step i. Vanilla MDMM dictates that we use an update of the form:

$$\mathbf{g} = (\lambda_i + c(H_{target} - \widehat{H}))\nabla_\theta \widehat{H}, \quad \theta_{i+1} = \theta_i + \eta(\mathbf{g} + \nabla_\theta l_a), \quad \lambda_{i+1} = \lambda_i + \tilde{\eta}(H_{target} - \widehat{H}). \tag{19}$$

Here, η is the learning rate, and $\tilde{\eta}, c$ are new hyperparameters. In practice, we don't directly use the update vector $\mathbf{g} + \nabla_\theta l_a$ and instead provide it to Adam. Note that the conventional method of entropy regularization keeps λ constant and positive, while here it can also become negative, thus forcing the entropy to decrease to reach H_{target}. Figure 3 demonstrates how entropy oscillates around the target value during training in the Minecart environment. We discuss the entropy schedules we used in more detail in Appendix A.4. In our experiments, entropy control did not perform well with A2C, so we only enabled it for PPO, which provides benefits as our ablation study demonstrates.

6 Experiments

We implement all architectures described above using the `TorchRL` library [Bou et al., 2023]. As a source of MOMDP environments, we use `MO-Gymnasium` [Alegre et al., 2022], the standard testbed in MORL. We run an ablation study to compare various architectures and algorithm details against each other and perform a comparison against two baselines, Pareto Conditioned Networks [Reymond et al., 2022] and Envelope Q-learning [Yang et al., 2019]. We chose these two approaches to MORL because, to the best of our knowledge, these are the most recent model-free MORL methods that condition a single policy (or value function) to generate the entire Pareto front and that have a public implementation supporting the case of continuous observations and discrete actions. We use the implementation provided in MORL-baselines [Felten et al., 2023] for both methods.

6.1 Preliminary results on deep-sea-treasure

All our methods can solve the standard grid-world environment called "Deep Sea Treasure". In this environment, the agent controls a submarine on a 2D grid. The two objectives are the cost of fuel (-1

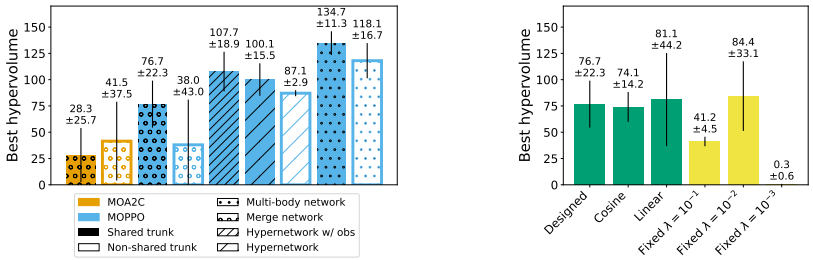


Figure 4: **Architecture ablations on Minecart:** *Left:* comparison of the algorithms and policy architectures on Minecart. *Right:* comparison of entropy control from Section 5.2 using three entropy schedules described in Appendix A.4 with standard entropy regularization using a fixed weight λ.

for every step) and the reward of a treasure that the agent can discover in pre-defined locations. When the agent is willing to spend more fuel, it can find a bigger reward. Precise control over the placement of treasures and the rewards from each one allows us to shape the 2D PF. We use the version of Deep Sea Treasure with a convex PF. We discovered that the results in this simple environment are not sensitive to hyperparameter tuning. All runs used $\approx 10^5$ environment steps and converged to Pareto fronts presented in Figure 1.

6.2 Ablation studies on Minecart

"Minecart" is a harder environment than deep-sea-treasure, also included in MO-Gym. This environment has an agent that operates a cart in a 2D continuous space. There are three rewards: two for bringing different types of ores and one for the consumed fuel. Depending on the agent's location, the probabilities for getting one or the other change, thus leading to a nontrivial multi-objective problem. This environment also has a deterministic version, where each mining action near a mine is guaranteed to produce a fixed amount of ore. We use the stochastic version for ablations on different architectural choices discussed here. For each architecture, we run a hyperparameter search detailed in Appendix B.1. Then, we run training for each method with five seeds to estimate the average and standard deviation. The resulting hypervolume for selecting methods is presented in Figure 4. We want to focus the attention of the reader on multiple conclusions that we can draw if we focus on parts of it:

- PPO significantly outperforms A2C, likely because of its sample-efficient data reuse across multiple epochs. This effect mirrors the corresponding knowledge from the scalar RL community.
- Non-shared trunk architectures consistently perform slightly worse than their shared counterparts.
- Architecture choice seems to influence the performance. Multi-body networks perform the best, followed by hypernetworks and merge networks. This ordering also depends on the environment since it comes out differently for another environment in the following section.

Figure 4 also shows that training is sensitive to the entropy control coefficient λ when using standard entropy regularization. All 12 considered hyperparameter configurations lead to collapse if we fix $\lambda = 10^{-3}$, while $\lambda = 10^{-2}$ seems to lead to optimal performance among methods with constant λ. On the other hand, all entropy control schedules lead to essentially the same performance. We additionally note here that we did not have to perform tuning of the entropy control learning rate $\tilde{\eta}$ for any of our experiments, we just fixed $\tilde{\eta} = \eta/10$ from the start. We also provide the results in expected utility for all of the methods from Figure 4 in Table 3 in the Appendix C.1 and demonstrate all hyperparameter grids in Figures 5 and 6 there.

6.3 Comparison with the baseline methods

We use three environments for comparison: two versions of Minecart (deterministic and non-deterministic) and MO-reacher. The non-deterministic version of Minecart checks that our methods can handle stochasticity, but it cannot be used to compare with PCN, which requires deterministic transitions. MO-reacher is a MuJoCo-based [Todorov et al., 2012] environment where an agent

Table 1: A comprehensive comparison of a selection of our methods to baselines. We do not evaluate PCN on the nondeterministic minecart because the authors of the method claim that its handling of the replay buffer would lead to incorrect behaviors in stochastic environments. The best in the column is marked in bold, and so are all methods whose intervals intersect the interval of the best.

Method			Minecart		Minecart det.		Reacher	
Arch	Shared	Entropy	HV	EU	HV	EU	HV ($/10^7$)	EU
Multi-body	No	Custom	$\mathbf{115.8 \pm 15.5}$	0.19 ± 0.06	$\mathbf{119.6 \pm 17.0}$	0.21 ± 0.05	$\mathbf{3.21 \pm 0.64}$	$\mathbf{21.06 \pm 3.23}$
Multi-body	Yes	Custom	$\mathbf{132.7 \pm 11.3}$	0.27 ± 0.02	$\mathbf{136.2 \pm 10.1}$	0.27 ± 0.02	2.64 ± 0.87	$\mathbf{19.09 \pm 3.48}$
Multi-body	Yes	Linear	$\mathbf{118.5 \pm 24.7}$	0.15 ± 0.09	$\mathbf{134.6 \pm 8.8}$	0.27 ± 0.02	2.53 ± 0.72	$\mathbf{18.26 \pm 3.64}$
Hypernet	Yes	Custom	97.9 ± 16.5	0.18 ± 0.03	93.8 ± 6.0	0.19 ± 0.04	2.45 ± 0.91	$\mathbf{17.34 \pm 4.99}$
Merge net	Yes	Custom	84.2 ± 14.1	0.03 ± 0.05	50.4 ± 42.1	-0.22 ± 0.26	$\mathbf{3.33 \pm 0.17}$	$\mathbf{21.73 \pm 0.72}$
PCN			—	—	106.7 ± 10.0	$\mathbf{0.30 \pm 0.06}$	1.91 ± 0.23	15.41 ± 1.72
Envelope			104.7 ± 9.5	$\mathbf{0.34 \pm 0.03}$	77.4 ± 4.7	$\mathbf{0.38 \pm 0.07}$	2.73 ± 0.60	$\mathbf{19.78 \pm 2.65}$

controls a two-jointed robot arm. There are four rewards, each corresponding to the l_2-distance of the tip of the arm from one of four targets on the 2D-space. The action space consists of nine actions corresponding to one of three torques (positive, negative, and zero) in each joint.

We tune the hyperparameters of each method on each environment separately; the details are presented in Appendix B. Results from all grids for this experiment are also presented in Figures 7-10 in the Appendix. The best hyperparameters are used to run the method again with 5 different seeds. We found that in some cases, envelope Q-learning showed good results early but collapsed later in training, likely due to catastrophic forgetting. Because of this, we demonstrate the results corresponding to the best hypervolume for envelope Q-learning. Our methods or PCN did not suffer from this issue, so we reported the metrics after the training was finished. We present the evaluation results in Table 1. Overall, our methods can outperform the baselines in terms of hypervolume. This is partly because we use non-deterministic policies for evaluation, while PCN and Envelope Q-learning rely on deterministic ones. Learning non-deterministic policies allows us to cover the Pareto front more densely. As one can see, on Minecart, whether our methods outperform Envelope Q-learning is ambiguous. Mirroring the situation in the scalar RL literature, however, our methods perform better on the robotic control task. Note that the authors of PCN also provide hypervolume measurements on deterministic Minecart in [Reymond et al., 2022, Table 1], but they do not mention the reference point $\mathbf{p}_0$ they use, nor the discount factor γ. Based on the true Pareto front that they provide and their measurements, they most likely use $\gamma = 1$, while we use $\gamma = 0.99$. Therefore, the results from their paper are not directly comparable with the ones provided in our Table 1.

7 Conclusion

We proposed several actor-critic architectures and two algorithms for dynamic multi-objective reinforcement learning. All of our approaches can solve a simple MORL environment and provide a continuous parametrization of the space of policies covering the Pareto front. We then performed an extensive comparison study on two more complicated environments, demonstrating the usefulness of the proposed improvements over the naive implementation. We showed that our implementation of multi-objective PPO can outperform the baselines on a robotic control task and perform competitively when delayed rewards and/or stochasticity are present in the environment.

Our contributions are empirical in nature. We compare architectures and technical details of the implementation, but we do not perform a theoretical analysis of our algorithms. Such analysis could also be beneficial for the community: for example, under which conditions does policy gradient or policy iteration converge to a policy that approximately covers the Pareto front, and what would the notion of "misspecification" be? Another important limitation of this work is that we only consider linear scalarizations of the vector return $\mathbf{J}(\pi, \boldsymbol{\alpha})$. This does not allow us to recover concave regions on the Pareto front. It should also be possible to extend our methods to nonlinear utilities, so that they optimize the expected scalarized return $\mathbb{E}\left[u\left(\boldsymbol{\alpha}, \sum_t \gamma^t \mathbf{r}_t\right)\right]$ for a utility u with a free parameter $\boldsymbol{\alpha}$. Finally, scalar PPO works especially well in continuous control environments, and a good extension of our work would be to see whether our methods preserve this strength in the multi-objective setup.

References

A. Abdolmaleki, S. Huang, L. Hasenclever, M. Neunert, F. Song, M. Zambelli, M. Martins, N. Heess, R. Hadsell, and M. Riedmiller. A distributional view on multi-objective policy optimization. In *International conference on machine learning*, pages 11–22. PMLR, 2020.

A. Abels, D. Roijers, T. Lenaerts, A. Nowé, and D. Steckelmacher. Dynamic weights in multi-objective deep reinforcement learning. In *International conference on machine learning*, pages 11–20. PMLR, 2019.

L. N. Alegre, F. Felten, E.-G. Talbi, G. Danoy, A. Nowé, A. L. C. Bazzan, and B. C. da Silva. MO-Gym: A library of multi-objective reinforcement learning environments. In *Proceedings of the 34th Benelux Conference on Artificial Intelligence BNAIC/Benelearn 2022*, 2022.

L. N. Alegre, A. L. Bazzan, D. M. Roijers, A. Nowé, and B. C. da Silva. Sample-efficient multi-objective learning via generalized policy improvement prioritization. *arXiv preprint arXiv:2301.07784*, 2023.

A. Bou, M. Bettini, S. Dittert, V. Kumar, S. Sodhani, X. Yang, G. De Fabritiis, and V. Moens. Torchrl: A data-driven decision-making library for pytorch. *arXiv preprint arXiv:2306.00577*, 2023.

J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

F. Felten, L. N. Alegre, A. Nowé, A. L. C. Bazzan, E. G. Talbi, G. Danoy, and B. C. d. Silva. A toolkit for reliable benchmarking and research in multi-objective reinforcement learning. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023)*, 2023.

Z. Fu, X. Cheng, and D. Pathak. Deep whole-body control: learning a unified policy for manipulation and locomotion. In *Conference on Robot Learning*, pages 138–149. PMLR, 2023.

M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. Van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3796–3803, 2019.

W. Hung, B.-K. Huang, P.-C. Hsieh, and X. Liu. Q-pensieve: Boosting sample efficiency of multi-objective rl through memory sharing of q-snapshots. *arXiv preprint arXiv:2212.03117*, 2022.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

A. Kumar, X. B. Peng, and S. Levine. Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*, 2019.

C. L. Lan, S. Tu, A. Oberman, R. Agarwal, and M. G. Bellemare. On the generalization of representations in reinforcement learning. *arXiv preprint arXiv:2203.00543*, 2022.

S. Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

S. Liu and L. N. Vicente. Accuracy and fairness trade-offs in machine learning: A stochastic multi-objective approach. *Computational Management Science*, 19(3):513–537, 2022.

I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

H. Lu, D. Herman, and Y. Yu. Multi-objective reinforcement learning: Convexity, stationarity and pareto optimality. In *The Eleventh International Conference on Learning Representations*, 2022.

C. Lyle, M. Rowland, and W. Dabney. Understanding and preventing capacity loss in reinforcement learning. *arXiv preprint arXiv:2204.09560*, 2022.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

A. Navon, A. Shamsian, G. Chechik, and E. Fetaya. Learning the pareto front with hypernetworks. *arXiv preprint arXiv:2010.04104*, 2020.

L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

J. Platt and A. Barr. Constrained differential optimization. In *Neural Information Processing Systems*, 1987.

M. Reymond, E. Bargiacchi, and A. Nowé. Pareto conditioned networks. *arXiv preprint arXiv:2204.05036*, 2022.

M. Reymond, C. F. Hayes, D. Steckelmacher, D. M. Roijers, and A. Nowé. Actor-critic multi-objective reinforcement learning for non-linear utility functions. *Autonomous Agents and Multi-Agent Systems*, 37(2):23, 2023.

D. M. Roijers, D. Steckelmacher, and A. Nowé. Multi-objective reinforcement learning for the expected utility of the return. In *Proceedings of the Adaptive and Learning Agents workshop at FAIM*, volume 2018, 2018.

A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

E. Todorov, T. Erez, and Y. 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. doi: 10.1109/IROS.2012.6386109.

P. Vamplew, J. Yearwood, R. Dazeley, and A. Berry. On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts. In *AI 2008: Advances in Artificial Intelligence: 21st Australasian Joint Conference on Artificial Intelligence Auckland, New Zealand, December 1-5, 2008. Proceedings 21*, pages 372–378. Springer, 2008.

P. Vamplew, C. Foale, R. Dazeley, and A. Bignold. Potential-based multiobjective reinforcement learning approaches to low-impact agents for ai safety. *Engineering Applications of Artificial Intelligence*, 100:104186, 2021.

H. Van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

J. Xu, Y. Tian, P. Ma, D. Rus, S. Sueda, and W. Matusik. Prediction-guided multi-objective reinforcement learning for continuous robot control. In *International conference on machine learning*, pages 10607–10616. PMLR, 2020.

R. Yang, X. Sun, and K. Narasimhan. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. *Advances in neural information processing systems*, 32, 2019.

L. M. Zintgraf, T. V. Kanters, D. M. Roijers, F. Oliehoek, and P. Beau. Quality assessment of morl algorithms: A utility-based approach. In *Benelearn 2015: proceedings of the 24th annual machine learning conference of Belgium and the Netherlands*, 2015.

Require: Multi-objective MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathbf{r}, P, \mu, \gamma)$
Require: Actor π_θ, critic $\widetilde{\mathbf{V}}_\psi$ with disjoint sets of parameters
while *not terminated* **do**
  Sample reward weights $\boldsymbol{\alpha} \sim U(\Delta_K)$
  Sample a truncated trajectory $\{(s_t, a_t, \mathbf{r}_t)\}_{t=0}^{T}$
    from P and $\pi_\theta(\cdot \mid \cdot, \boldsymbol{\alpha})$
  for $e \leftarrow 1$ **to** E **do** `// iterate over epochs`
    Update PopArt $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ and the critic head
    Get the reward-to-go $\widehat{\mathbf{Q}}_t$ from bootstrapped $\mathrm{TD}(1)$
    Compute vector- and scalarized advantage:
      $\hat{\mathbf{A}}_t \leftarrow \widehat{\mathbf{Q}}_t - \widetilde{\mathbf{V}}_\psi(s_t, \boldsymbol{\alpha}),\ \hat{A}_t \leftarrow \boldsymbol{\alpha}^\intercal(\hat{\mathbf{A}}_t / \boldsymbol{\sigma})$
    for $i \leftarrow 1$ **to** F **do** `// We run more updates for the critic`
      for $b \leftarrow 1$ **to** B **do**
        Sample a minibatch $\{(s_k, a_k, \widehat{\mathbf{Q}}_k)\}_k$
        Compute $l_c(\psi)$ and its gradient $\nabla_\psi l_c$ from (15)
        Feed $\nabla_\psi l_c$ into critic's Adam, update ψ
      end
    end
    for $b \leftarrow 1$ **to** B **do**
      Sample a minibatch $\{(s_k, a_k, \hat{A}_k)\}_k$
      Update $\mathbf{g}$ and λ using (19) for entropy control
      Compute $l_a(\psi)$ and its gradient $\nabla_\theta l_a$ from (17)
      Feed $\mathbf{g} + \nabla_\theta l_a$ into actor's Adam, update θ
    end
  end
end

Algorithm 2: A schematic description of multi-objective PPO for an actor and a critic with separate trunks. In this case, the actor-to-critic ratio C that we tune stands for the ratio η_a/η_c, where η_a and η_c are the learning rates of the actor and the critic optimizers, respectively. Since we tend to set $C > 1$, we compensate here by running F times more updates on the critic. In all our experiments we set $F = 2$.

A Implementation details

A.1 Other variants of our learning algorithms

Here we describe our versions of non-shared trunk PPO (Algorithm 2) and of A2C (shared trunk version in Algorithm 3, and non-shared — in Algorithm 4). We note that non-shared versions can perform multiple updates on the critic per single actor update, but this would be harder to justify conceptually for a shared trunk architecture.

Since MOA2C diverges when used together with our entropy regularization scheme, for it we have to rely on the standard entropy regularizer loss. Given a trajectory $\{(s_t, a_t, \mathbf{r}_t)\}_t$ sampled with reward weights $\boldsymbol{\alpha}$, it is given by

$$\mathbf{g}_{a2c} = \frac{1}{T+1} \sum_{t=0}^{T} \nabla_\theta H(\pi_\theta(\cdot \mid s_t, \boldsymbol{\alpha}). \tag{20}$$

A.2 Gradients for shared and non-shared trunk architectures

Suppose the actor and critic do not share parameters. In that case, updating them in PPO or A2C is straightforward: the gradient of the value loss is separated from the gradient of the policy with entropy regularization. We can even perform multiple "inner" optimization steps on the critic. Since the critic is aiming at a moving target (the value function depends on the policy, which keeps changing), there should be a clear sweet spot between overfitting the critic to the current policy and making it unable to catch up. Although more extensive experiments would be helpful, for now, we just set the number

Require: Multi-objective MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathbf{r}, P, \mu, \gamma)$
Require: Actor π_θ, critic $\widetilde{\mathbf{V}}_\psi$ with disjoint sets of parameters
while *not terminated* **do**
  Sample reward weights $\boldsymbol{\alpha} \sim U(\Delta_K)$
  Sample a truncated trajectory $\{(s_t, a_t, \mathbf{r}_t)\}_{t=0}^{T}$
    from P and $\pi_\theta(\cdot \mid \cdot, \boldsymbol{\alpha})$
  for $e \leftarrow 1$ **to** E **do** `// iterate over epochs`
    Update PopArt $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ and the critic head
    Get the reward-to-go $\widehat{\mathbf{Q}}_t$ from bootstrapped $\mathrm{TD}(1)$
    Compute vector- and scalarized advantage:
      $\hat{\mathbf{A}}_t \leftarrow \widehat{\mathbf{Q}}_t - \widetilde{\mathbf{V}}_\psi(s_t, \boldsymbol{\alpha}),\ \hat{A}_t \leftarrow \boldsymbol{\alpha}^\intercal(\hat{\mathbf{A}}_t / \boldsymbol{\sigma})$
    Compute $l_c(\psi)$ and its gradient $\nabla_\psi l_c$ from (15) on the entire trajectory
    Compute the entropy regularizing gradient $\mathbf{g}_{a2c}$ from (20)
    Compute the policy gradient estimator $\widehat{\nabla}_\theta J(\theta)$ from (14)
    Update β_c so that $\|\mathbf{g}_{a2c} + \widehat{\nabla}_\theta J(\theta)\| \approx C\beta_c \|\nabla_\psi l_c\|$
    Feed $\mathbf{g}_{a2c} + \widehat{\nabla}_\theta J(\theta) + \beta_c \nabla_\psi l_c$ into Adam, update ν
  end
end

Algorithm 3: A schematic description of multi-objective A2C for an actor and a critic with shared trunks. We discovered that entropy control leads to divergence of MOA2C on our environments, so we disabled it in our experiments.

of inner updates to 2 (relative to one update of the actor) and only studied the relative learning rate we can give to the critic for optimal performance.

When the architectures of the actor and critic are joined, training becomes trickier. Now, multiple updates from the critic do not make much sense because this would unpredictably influence the actor. We want all updates of the actor parameters to be at least partially influenced by the relevant learning signal. Otherwise, we are in danger of a "catastrophic misstep.". Let now $\mathbf{g}_a$ be the actor gradient from the policy loss and the entropy regularization. Let $\mathbf{g}_c = \nabla_\psi l(\psi)$ be the gradient from the critic. Our intuition is that we need to make the norms of these gradients approximately proportional, i.e., $\|\mathbf{g}_c\| \approx C\|\mathbf{g}_a\|$, where $C > 0$ is a hyperparameter. The most straightforward way to ensure this is to set $\beta_c = C\|\mathbf{g}_c\|/\|\mathbf{g}_a\|$. Setting β_c to this value directly at each iteration would lead to a rather unstable learning rate for the critic, so instead, we compute a running average using a recurrent formula

$$\beta_c = \frac{\delta C \|\mathbf{g}_c\|}{\|\mathbf{g}_a\|} + (1 - \delta)\beta_c, \tag{21}$$

where δ is another hyperparameter regulating the smoothness of the critic's learning rate, which we set to 0.001 in all experiments. The hope is that the learning procedure is not as sensitive to δ, while C is the analogue of the relative learning rate that we could compute in the separated actor and critic case.

We use Adam [Kingma and Ba, 2014] for optimization after computing the above gradients. If the critic is separated, we naturally have a separate instance of Adam optimizer. Otherwise, the gradient estimates are fed into it after the adaptive relative weight update (21).

A.3 Step discarding heuristics

During the preliminary experiments, we identified multiple scenarios that occur rather rarely but lead to the collapse of the learning process. Unstable training on toy environments was especially observed with the hypernetwork architecture. In part, it can be remedied by clipping the gradient norms (which we do) or even just by setting a lower learning rate, but this comes at the cost of lower sample efficiency. The collapse of learning that we observed mostly happened rapidly and led to trivial policies and zero entropy. To prevent the missteps that lead to collapse, we employ the following heuristics:

Require: Multi-objective MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathbf{r}, P, \mu, \gamma)$
Require: Actor π_θ, critic $\widetilde{\mathbf{V}}_\psi$ with parameters $\nu = \theta \cup \psi$
Initialize ν
while *not terminated* **do**
  Sample reward weights $\boldsymbol{\alpha} \sim U(\Delta_K)$
  Sample a truncated trajectory $\{(s_t, a_t, \mathbf{r}_t)\}_{t=0}^{T}$
    from P and $\pi_\theta(\cdot \mid \cdot, \boldsymbol{\alpha})$
  for $e \leftarrow 1$ **to** E **do** `// iterate over epochs`
    Update PopArt $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ and the critic head
    Get the reward-to-go $\widehat{\mathbf{Q}}_t$ from bootstrapped $\mathrm{TD}(1)$
    Compute vector- and scalarized advantage:
      $\hat{\mathbf{A}}_t \leftarrow \widehat{\mathbf{Q}}_t - \widetilde{\mathbf{V}}_\psi(s_t, \boldsymbol{\alpha}),\ \hat{A}_t \leftarrow \boldsymbol{\alpha}^\intercal(\hat{\mathbf{A}}_t / \boldsymbol{\sigma})$
    for $i \leftarrow 1$ **to** F **do**
      Compute $l_c(\psi)$ and its gradient $\nabla_\psi l_c$ from (15) on the entire trajectory
      Feed $\nabla_\psi l_c$ into critic's Adam, update ψ
    end
    Compute the entropy regularizing gradient $\mathbf{g}_{a2c}$ from (20)
    Compute the policy gradient estimator $\widehat{\nabla}_\theta J(\theta)$ from (14)
    Feed $\mathbf{g}_{a2c} + \widehat{\nabla}_\theta J(\theta)$ into actor's Adam, update θ
  end
end

Algorithm 4: A schematic description of multi-objective A2C for an actor and a critic with non-shared trunks. We discovered that entropy control leads to divergence of MOA2C on our environments, so we disabled it in our experiments. As in Algorithm 2, the actor-to-critic ratio C that we tune stands for the ratio η_a/η_c, where η_a and η_c are the learning rates of the actor and the critic optimizers, respectively.

1. If the entropy dropped significantly (the change in entropy is negative, and its absolute value is three standard deviations above the average absolute changes in entropy) *and* the average reward did not increase from the previous step, we discard the step.
2. If the entropy or actor gradient has been nearly zero for the past 200 steps, reset learning from the last checkpoint where nonzero statistics were still observed.

To prevent discarding too many steps, we set an upper limit of 5% of the last 100 iterations that can be discarded.

A.4 Entropy control

The method described in Section 5.2 to control entropy during training allows us to shape the entropy according to any schedule. An investigation into principled ways of selecting such schedules would be quite interesting, but it is out of the scope of this paper. We have selected three schedules based on our intuition about exploration-exploitation trade-offs. All of these schedules start from the maximal possible entropy $H_{target}(0) = H_{max} = \log|\mathcal{A}|$ and progress to $H_{target}(1) = H_{min}$. For Minecart and resource gathering experiments, we set $H_{min} = 0.4$; for deep sea treasure, we set it to 0.1. The schedules are defined as $H_{target}(u)$, where $u \in [0, 1]$ denotes the proportion of environment interactions that were used so far from the total budget allocated for training.

The first schedule is a simple linear function going from the maximal entropy $H_{max} = \log|\mathcal{A}|$ to a desired minimal value H_{min}:

$$H_{target,lin}(u) = H_{max} - (H_{max} - H_{min})u. \tag{22}$$

The second schedule is normalized cosine, used to provide extra high-entropy exploration time in the beginning:

$$H_{target,cos}(u) = (H_{max} - H_{min})\cos(\pi u/2) + H_{min}. \tag{23}$$

Finally, we also designed a "custom" schedule with a flat start to provide exploration time and, unlike the previous schedule, has an extended flat end around the lower entropy to provide time for

exploitation. We used the expression

$$H_{target,custom}(u) = (H_{max} - H_{min})(0.5 - \cos\left(\pi(1-u)^{1.3}\right)/2 + H_{min}. \tag{24}$$

The target entropy curves resulting from all of these methods are shown in Figure 3 in the main paper. We also briefly experimented with "resetting" entropy to a high value a few times during training, similar to cosine learning rate annealing [Loshchilov and Hutter, 2016], but did not get promising results.

B Hyperparameter tuning

B.1 Our methods

Here we describe the details of the hyperparameter search that we performed for each architecture when comparing them to produce Figure 4 in Section 6.2. We run a hyperparameter search for all shared trunk architectures and pick the best final hypervolume. We search over the learning rates $\eta \in$\{3e-4, 1e-3, 3e-3\} and over the ratio $C \in \{1, 3, 10, 30\}$ as defined in Algorithm 1. For architectures with non-shared trunks, we instead search over the learning rate of the critic among $\eta_c \in \{\eta, \eta/3, \eta/10, \eta/30\}$. All hyperparameter grids for this experiment are shown in Figures 5 and 6. Final results on more architectures with extra metrics for the Pareto front are shown in Table 3.

For experiments in Section 6.3, we also run a hyperparameter grid search for each pair of methods and environments. For the reacher environment, we search over $\eta \in$\{3e-4, 1e-3, 3e-3\} and $C \in \{0.33, 1, 3\}$. We picked smaller values of C for this environment because the trajectories sampled from it are always infinite; hence, bootstrapping and the critic are more relevant to the algorithm's performance.

B.2 PCN and Envelope Q-learning

We aim to compare our methods fairly to PCN by performing additional hyperparameter sweeps for PCN. We sweep through the same learning rates $\eta \in \{0.0001, 0.0003, 0.001, 0.003\}$ and perform an architecture search. The PCN network consists of embedding layers for the target rewards and for the state, followed by an MLP that combines these into the final action. We experiment with three sizes of the hidden layer of the MLP: $d \in \{64, 128, 256\}$. In addition, we try to add an extra hidden layer. This makes the total number of checked configurations $4 \times 3 \times 2 = 24$.

The default implementation of Envelope Q-learning already relies on a rather deep MLP (4 hidden layers with width 256 each). We expected less sensitivity to the architecture of the MLP than to ways in which one provides the state and $\boldsymbol{\alpha}$ to it (our reference implmentation simply concatenates them), so we instead focused on the ε for exploration and the learning rate. The reference implementation linearly anneals the exploration ε from 1 to some value ε_{min} in the first half of the training, and then keeps it constant. For Envelope Q-learning, we sweeped through the learning rates $\eta \in \{0.0003, 0.001, 0.003\}$ and through $\varepsilon_{min} \in \{0.01, 0.05, 0.1\}$. Other parameters of the baselines were kept at the default values from MORL-Baselines.

C Additional Results

In Table 2, we show the hyperparameters of our algorithms that were not tuned via a grid search.

C.1 Ablations on normalization heuristics

Table 3, apart from the methods discussed in Section 6.2, shows three ablations on parts of our approach. The results include:

- The dynamic gradient reweighting approach for shared trunk architectures described in Appendix A.2 does not provide benefits compared to using a constant relative weight. Although we believe this approach might stabilize learning dynamics in some other environments, we chose not to describe it in the main paper because of this result.

Table 2: Hyperparameter choices for our experiments. Notice that we did not tune any hyperparameters other than the number of samples for individual environments, apart from the learning rate η and the learning rate ratio C as discussed in the main paper.

	Deep-sea-treasure	Minecart	Reacher
Hidden dim. for merge networks	256		
Hidden dim. for multi-body networks	256		
Hidden dim. for hypernetworks	64		
γ	0.99		
Initial λ for dynamic entropy	0.01		
H_{min} for entropy control	0.4		
Dampening c for entropy control	0.01		
Total number of environment interactions	1e5	4e6	1e6
Number of trajectories B in a batch	8		
Number of PPO epochs	4		
Number of PPO minibatches	8		
PPO clip ε	0.2		
PopArt learning rate	0.001		
Critic weight decay	0.01		
Max gradient norm for the actor	0.5		
Max gradient norm for the critic	0.5		
Hypervolume reference	—	(0,0,-200)	(-50,-50,-50,-50)

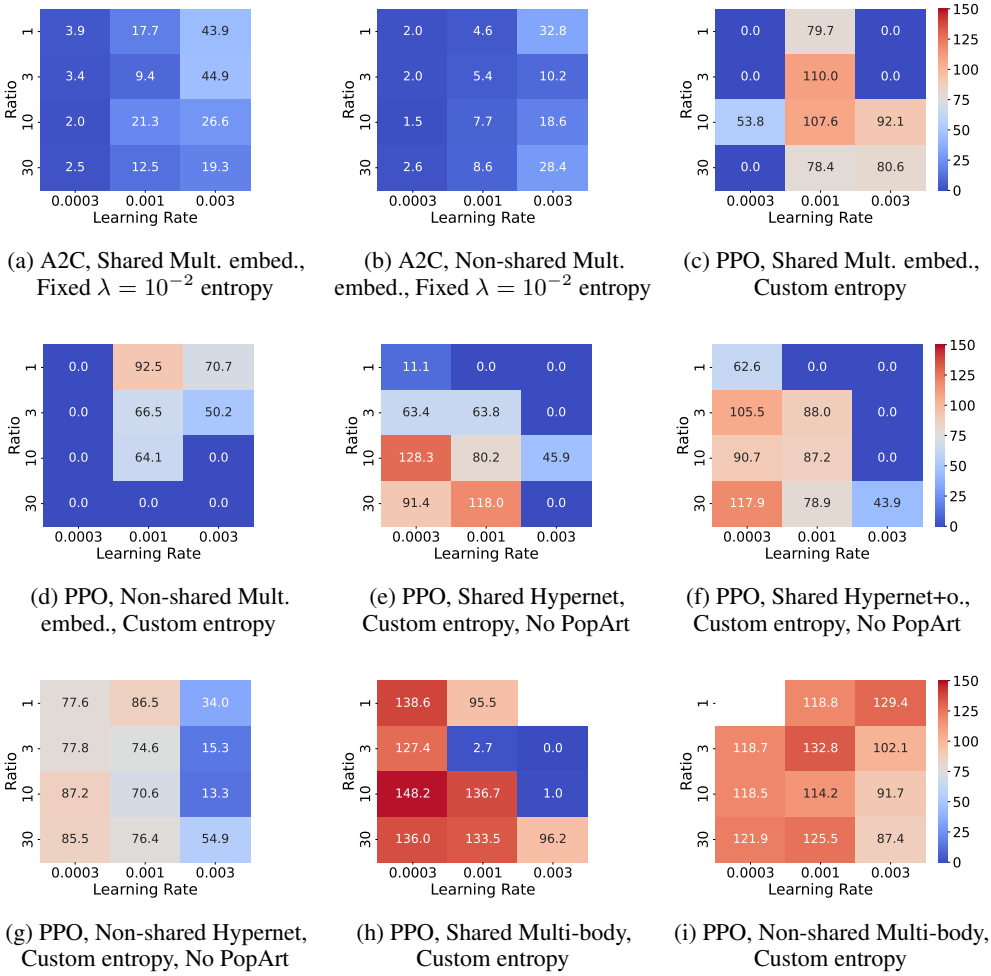
Table 3: Detailed results of ablation studies on a stochastic Minecart and resource gathering. "No renorm." means that PopArt was used for value learning, but normalized scalarization (16) was not used when computing the policy loss. "No dyn. beta" means that the relative gradient weighting scheme from Appendix A.2. HV is hypervolume (larger is better), EU is expected utility (larger is better), and MUL is maximum utility loss (smaller is better). Negative MUL can occur due to a stochastic environment, which allows the sampled reward to be higher than the expected optimal reward.

Method					Metrics on Minecart		
Algo	Arch	Shared trunk	Entropy	Notes	HV	EU	MUL
A2C	Merge net.	Yes	Fixed $\lambda = 10^{-2}$	—	28.3 ± 25.7	-0.34 ± 0.21	0.99 ± 0.21
A2C	Merge net.	No	Fixed $\lambda = 10^{-2}$	—	41.5 ± 37.5	-0.24 ± 0.25	0.81 ± 0.25
PPO	Merge net.	Yes	Custom	—	76.7 ± 22.3	-0.03 ± 0.18	0.59 ± 0.18
PPO	Merge net.	No	Custom	—	38.0 ± 43.0	-0.11 ± 0.10	0.82 ± 0.10
PPO	Hypernet w/ obs.	Yes	Custom	No PopArt	107.7 ± 18.9	0.22 ± 0.05	0.25 ± 0.05
PPO	Hypernet	Yes	Custom	No PopArt	100.1 ± 15.5	0.19 ± 0.03	0.31 ± 0.03
PPO	Hypernet	No	Custom	No PopArt	87.1 ± 2.9	0.14 ± 0.01	0.33 ± 0.01
PPO	Multi-body	Yes	Custom	—	134.7 ± 11.3	0.27 ± 0.01	0.16 ± 0.01
PPO	Multi-body	No	Custom	—	118.1 ± 16.7	0.18 ± 0.05	0.34 ± 0.05
PPO	Merge net.	Yes	Cos	—	74.1 ± 14.2	-0.03 ± 0.07	0.56 ± 0.07
PPO	Merge net.	Yes	Linear	—	81.1 ± 44.2	-0.06 ± 0.31	0.64 ± 0.31
PPO	Merge net.	Yes	Fixed $\lambda = 10^{-1}$	—	41.2 ± 4.5	-0.42 ± 0.08	1.05 ± 0.08
PPO	Merge net.	Yes	Fixed $\lambda = 10^{-2}$	—	84.4 ± 33.1	-0.00 ± 0.10	0.61 ± 0.10
PPO	Merge net.	Yes	Fixed $\lambda = 10^{-3}$	—	0.3 ± 0.6	-0.65 ± 0.09	1.30 ± 0.09
PPO	Merge net.	Yes	Custom	No dyn. beta	82.2 ± 20.9	0.04 ± 0.05	0.56 ± 0.05
PPO	Merge net.	Yes	Custom	No renorm.	92.4 ± 21.4	0.16 ± 0.05	0.37 ± 0.05
PPO	Merge net.	Yes	Custom	No PopArt	59.5 ± 33.9	0.08 ± 0.09	0.50 ± 0.09

- If we do not use the normailzed scalarization (16), the performance slightly improves. Note, however, that this scalarization makes the approach scale-invariant, so the minor performance loss is probably justified.

D Compute requirements

We used a cluster with Nvidia V100 GPU nodes. An average run on MO-reacher took around 3 hours, and we ran 3 of them in parallel. Hence, reproducing the results on MO-reacher would take around 75 GPU hours. A run on minecart took around 5 hours, and we also ran 3 in parallel. In total this



(a) A2C, Shared Mult. embed., Fixed $\lambda = 10^{-2}$ entropy

(b) A2C, Non-shared Mult. embed., Fixed $\lambda = 10^{-2}$ entropy

(c) PPO, Shared Mult. embed., Custom entropy

(d) PPO, Non-shared Mult. embed., Custom entropy

(e) PPO, Shared Hypernet, Custom entropy, No PopArt

(f) PPO, Shared Hypernet+o., Custom entropy, No PopArt

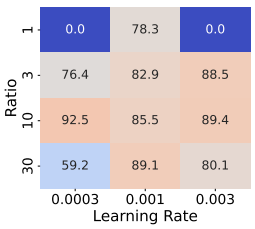
(g) PPO, Non-shared Hypernet, Custom entropy, No PopArt

(h) PPO, Shared Multi-body, Custom entropy

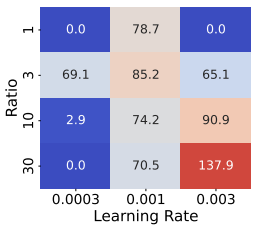
(i) PPO, Non-shared Multi-body, Custom entropy

Figure 5: Hypervolume of all hyperparameter configurations for the ablations demonstrated in Figure 4 and Table 3 run on the non-deterministic Minecart environment. Part 1. "Hypernet+o." is the architecture where the observation is provided to the hypernetwork along with $\boldsymbol{\alpha}$.

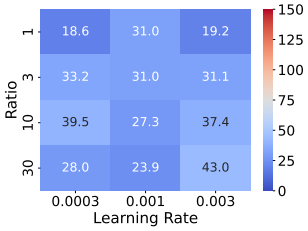
amounds to 490 GPU hours for minecart. Hence, to reproduce the results from the main paper, it takes roughly 565 hours on a V100 GPU.



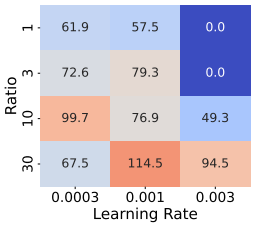
(a) PPO, Shared Mult. embed., Cosine entropy



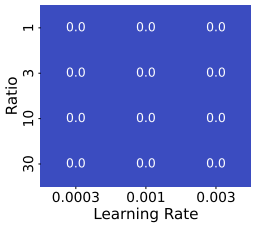
(b) PPO, Shared Mult. embed., Linear entropy



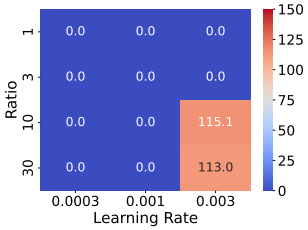
(c) PPO, Shared Mult. embed., Fixed $\lambda = 10^{-1}$ entropy



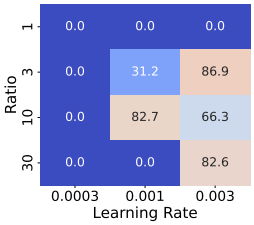
(d) PPO, Shared Mult. embed., Fixed $\lambda = 10^{-2}$ entropy



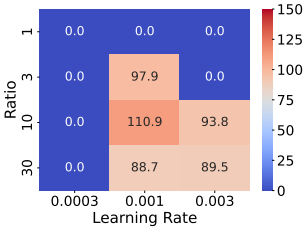
(e) PPO, Shared Mult. embed., Fixed $\lambda = 10^{-3}$ entropy



(f) PPO, Shared Mult. embed., Custom entropy, No renorm.



(g) PPO, Shared Mult. embed., Custom entropy, No PopArt



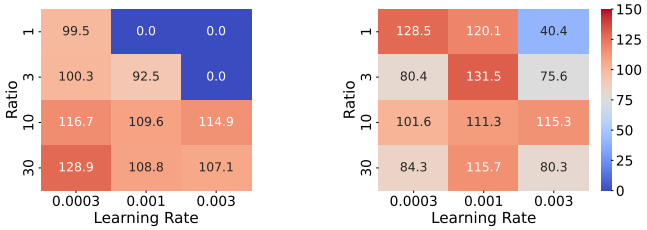
(h) PPO, Shared Mult. embed., Custom entropy, No dyn. beta

Figure 6: Hypervolume of all hyperparameter configurations for the ablations demonstrated in Figure 4 and Table 3 run on the non-deterministic Minecart environment. Part 2.

(a) PPO, Shared Mult. embed., Custom entropy

(b) PPO, Shared Hypernet, Custom entropy, No PopArt

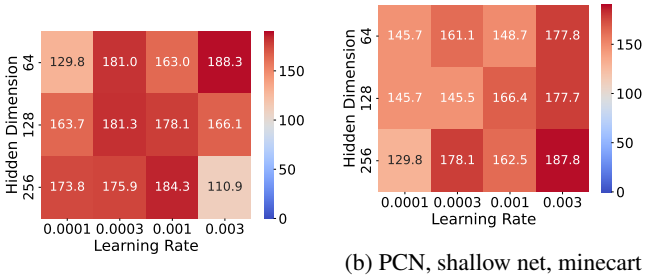
(c) PPO, Shared Multi-body, Custom entropy



(d) PPO, Shared Multi-body, Linear entropy

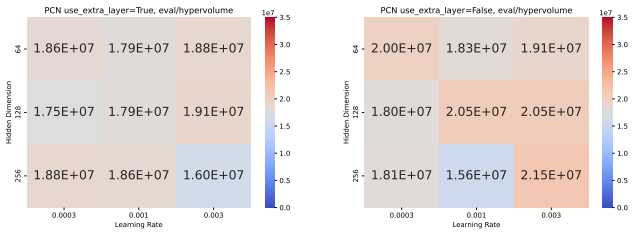
(e) PPO, Non-shared Multi-body, Custom entropy

Figure 7: Hypervolume of our hyperparameter configurations for the ablations demonstrated in Table 1 run on the deterministic Minecart environment.



(a) PCN, deep net, minecart det

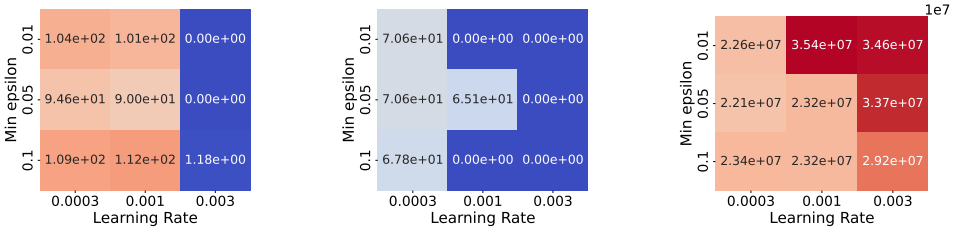
(b) PCN, shallow net, minecart det



(c) PCN, deep net, Reacher

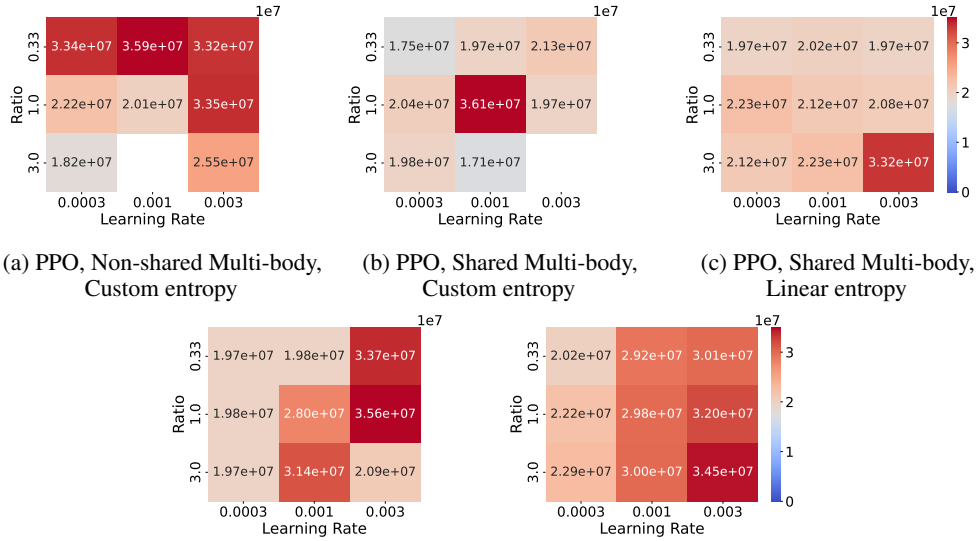
(d) PCN, shallow net, Reacher

Figure 8: Hypervolume of all hyperparameter configurations for the PCN networks on minecart-deterministic and on Reacher.



(a) Minecart (b) Minecart det. (c) Reacher

Figure 9: Hypervolume of all hyperparameter configurations for Envelope Q-learning evaluation.



(a) PPO, Non-shared Multi-body, Custom entropy

(b) PPO, Shared Multi-body, Custom entropy

(c) PPO, Shared Multi-body, Linear entropy

(d) PPO, Shared Hypernet, Custom entropy, No PopArt

(e) PPO, Shared Mult. embed., Custom entropy

Figure 10: Hypervolume of all hyperparameter configurations for our methods on Reacher.