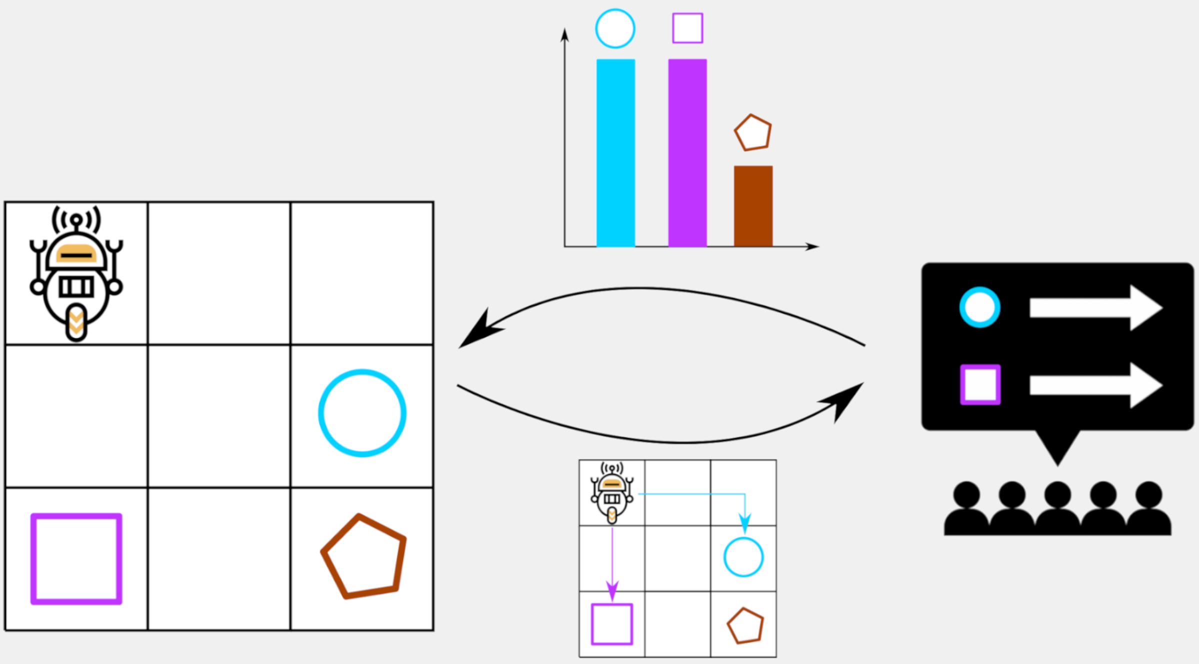
Aligning AI with Human Norms

Multi-Objective Deep Reinforcement Learning with Active Preference Elicitation

Markus Peschl

Thesis Report
MSc Applied Mathematics



TU Delft Delft University of Technology

Aligning AI with Human Norms

Multi-Objective Deep Reinforcement Learning with Active Preference Elicitation

by

Markus Peschl

to obtain the degree of Master of Science Applied Mathematics
at the Delft University of Technology,
to be defended publicly on October 8, 2021.

Student number:	5144124	
Project duration:	January 1, 2021 – October 8, 2021	
Thesis committee:	Dr. L. C. Siebert,	TU Delft (supervisor)
	Dr. A. Zgonnikov,	TU Delft (supervisor)
	Dr. F. Oliehoek,	TU Delft
	Dr. D. Kurowicka.	TU Delft

TU Delft

Preface

Artificial intelligence is a fascinating field of research, with tremendous amounts of funding and increasingly bigger research communities coming together, determined to build systems that not only match, but rather extend and exceed the capabilities of the human brain. Soon, we can expect machine learning systems to automate factories, the transportation of goods, power grids, drug discovery and many more areas of fundamental importance for the well-being of people in modern society. This, however, could come at the cost of delegating many crucial decisions to autonomous agents, which operate on a level of granularity that is impossible for us to truly understand. Nonetheless, the potential economic benefits of deploying such highly performing agents in the world are remarkable, which is why we can expect issues related to interpretability and safety to be largely neglected in practice.

For these reasons, I personally deem the field of value alignment especially important and have chosen to pursue this path in my thesis. Having conducted all of my research throughout an ongoing pandemic, I have come to realize the importance of global risk mitigation. Even though I am optimistic about the future development of artificial intelligence, I am certain that its accompanying risks are already shaping our world in previously unforeseen ways, which will yield dramatic negative outcomes if being left ignored. As a consequence, I have devoted my research to studying how we can align autonomous agents with a variety of human norms. Normative behavior is interesting, because it can be easily observed in data sets of human decision-making, but it is vastly difficult to formally specify to an agent. The adherence to norms, however, can alleviate a wide range of central safety concerns for agents acting in the real world. In this thesis, I aim to draw attention to the power of normative agents from a multi-objective point of view, which I believe to be a promising avenue of research for building agents that act safely as well as allow for value open design.

Finally, I would like to deeply thank my daily supervisors, Dr. Luciano Cavalcante Siebert and Dr. Arkady Zgonnikov for leading me to this topic and providing me with critical feedback. Aside from their constant support, they have made my research incredibly enjoyable with weekly discussions that I could always look forward to. Furthermore, I would like to express my gratitude to Dr. Frans Oliehoek, who has guided me through this project, as well as to Dr. Tina Nane and Dr. Dorota Kurowicka for helping me set up the thesis committee.

Markus Peschl
Vienna, September 2021

Abstract

The field of deep reinforcement learning has seen major successes recently, achieving superhuman performance in discrete games such as Go and the Atari domain, as well as astounding results in continuous robot locomotion tasks. However, the correct specification of human intentions in a reward function is highly challenging, which is why state-of-the-art methods lack interpretability and may lead to unforeseen societal impacts when deployed in the real world. To tackle this, we propose multi-objective reinforced active learning (MORAL), a novel framework based on inverse reinforcement learning for combining a diverse set of human norms into a single Pareto optimal policy. We show that through the combination of active preference learning and multi-objective decision-making, one can interactively train an agent to trade off a variety of learned norms as well as primary reward functions, thus mitigating negative side effects. Furthermore, we introduce two toy environments called *Burning Warehouse* and *Delivery*, which allow for studying the scalability of our approach in both size of the state space and reward complexity. We find that through mixing expert demonstrations and preferences, we can achieve superior efficiency compared to employing a single type of expert feedback and, finally, suggest that unlike previous literature, MORAL is able to learn a deep reward model consisting of multiple expert utility functions.

Contents

Abbreviations

AIRL	Adversarial Inverse Reinforcement Learning
AI	Artificial Intelligence
CCS	Convex Coverage Set
CIRL	Cooperative Inverse Reinforcement Learning
CMDP	Constrained Markov Decision Process
CNN	Convolutional Neural Network
DL	Deep Learning
DRLHP	Deep Reinforcement Learning from Human Preferences
ECDF	Empirical Cumulative Distribution Function
GAN	Generative Adversarial Network
IRD	Inverse Reward Design
IRL	Inverse Reinforcement Learning
KL	Kullback-Leibler
MCMC	Markov Chain Monte Carlo
MDP	Markov Decision Process
MLE	Maximum Likelihood Estimate
MLP	Multilayer Perceptron
MO-MPO	Multi-Objective Maximum A Posteriori Policy Optimization
MOMDP	Multi-Objective Markov Decision Process
MORAL	Multi-Objective Reinforced Active Learning
MORL	Multi-Objective Reinforcement Learning
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
SGD	Stochastic Gradient Descent

I

Thesis

1 Introduction

Over the past decade, significant increases in computing power as well as improved methodologies for training deep reinforcement learning (RL) agents have led to numerous pioneering applications, including playing combinatorially large games such as Go at superhuman level [79, 80], attaining high scores in Atari games from purely visual input [58], automatically designing application-specific integrated circuits [57] and improving the energy efficiency of data center cooling [55]. Despite these achievements, however, deep RL systems have remained highly constrained in the types of environments they can be successfully deployed in. Aside from a heavy reliance on enormous amounts of trial and error experience and sufficient observability of the environment, this constraint is largely attributed to the need for a well specified reward function. While all of these criteria can be easily satisfied in simulated environments, they are mostly violated in real-world settings, which is the underlying reason as to why the training of deep RL agents in the real world is incredibly challenging [27].

The problem of formally specifying a set of goals that an agent ought to achieve is not only constrained to RL, but in fact a more fundamental challenge of artificial intelligence (AI) research [72]. On the one hand, this is due to difficulties in generalizing from a successfully trained test scenario to a wider class of similar problems that require achieving the same goal under slightly different circumstances. For instance, a self-driving car might be able to properly turn left at intersections on sunny days, but might fail to perform the exact same turn on a rainy day. On the other hand, correctly specifying a goal in the real world turns out to be cumbersome due to a dependence on human values, which constitute an undoubtedly more fuzzy set of preferences over different states of the world. This is problematic since, arguably, good exhibited generalization performance on a set of human defined goals does not supervene on the alignment of an agent's actions with respect to human values. As a consequence, an inherent *value alignment* problem comes into play which, in the extreme case, could pose an existential threat to humanity that needs to be accounted for when considering to build capable AI agents [73]. Besides merely being a hypothetical threat, its urgency as well as types of possible solutions that might control it are unfortunately still highly contested at this point in time. Nonetheless, the need for studying machine behavior and the impact of autonomous systems on society is pressing and quickly becoming a more widespread topic across various areas of AI [67].

In this thesis, we aim to tackle such potential negative societal impacts by studying how to interactively control for a diverse set of social norms in deep RL agents. This focus is motivated by the fact that RL agents typically operate at a high level of granularity, thus being responsible for low level actions, which render the possibility for exact reward specification and human oversight impossible [92]. In light of such design deficiency, it is then only through accounting for the unspoken normative conduct which governs human behavior that one can achieve the completion to human-aligned goal specifications [38]. Therefore, our primary research goal concerns how to learn such normative policies by the means of combining different human forms of feedback data. To do so, we explore how to overcome the reward specification problem when no prior assumptions about the environment can be made by learning a deep reward model through the combination of demonstrations and pairwise preferences. As a result, we provide a multi-objective framework for incorporating normative behavior into narrow RL agents as well as controlling for a diversity of norms simultaneously.

1.1. Value Alignment

With the field of AI undergoing rapid advances, long term concerns about the safety of superintelligent agents are arising [16], as well as numerous ethical challenges which are already present in today's systems [24]. Both have led to active fields of research, with the latter including problems of unfair or biased decision-making whereas the former is spurring a broader research agenda focused on AI safety. Despite differing in their respective time horizons, a large portion of each area fundamentally reduces to the problem of value alignment [38]. Consider large language models trained to accurately complete sentences on a diverse corpus of text, which have been shown to resort to discriminatory vocabulary [12]. Similarly, a delivery robot which might be willing to destroy any object in its way to achieve higher rewards by fulfilling its tasks more quickly. In both of these cases, the AI system exhibits emergent malicious behavior which was originally not explicitly encoded in its goal specifications, but rather arose merely from overoptimization of a narrowly defined task.

RL agents have been shown to be especially prone to goal misalignment, leading to a multitude of safety problems [6]. This raises the need for technical research that can deal with incomplete reward specifications. However, depending on the reach and capacity of an agent, finding robust ways of specifying goals does not, by itself, solve the value alignment problem. That is, because value alignment is a two-fold problem: While correctly translating human goals into artificially intelligent systems is a prerequisite for achieving beneficial outcomes, it does not address which values we should finally encode. This *normative* aspect of value alignment is heavily entangled with the type of technical solutions one aims to build. Building traditional RL agents that maximize expected cumulative rewards draws close connections to act utilitarianism, thus potentially limiting the ability to encode other moral frameworks such as deontological constraints into its behavior [32]. Nonetheless, the flexibility of the RL framework yields the promise of being theoretically able to incorporate a wide array of normative ethical theories. From this point of view, building adaptive agents that offer compatibility with respect to different, possibly conflicting values can alleviate the normative aspect of value alignment to a great extent. For these reasons, we propose a technical solution that aims to tackle both the goal specification problem as well as allow for value-open design by combining value learning and multi-objective decision-making. We will now briefly describe each of the approaches and outline how they can be combined.

Value Learning. Even though we typically can not formally specify what constitutes a social norm in a reward function, we can hope to learn it from a diverse dataset of demonstrations. In this thesis, we mainly focus on *inverse reinforcement learning* (IRL) [61], although many related approaches exist (cf. section 6.2). IRL learns a reward function which best explains demonstrated behavior, by modelling the demonstrator as an agent maximizing its own expected utility. One obvious drawback of traditional IRL is that we cannot expect to vastly generalize, or even outperform the demonstrations. Fundamentally, IRL is an ill-posed problem which allows many solutions, including trivial or degenerate reward functions [101]. Nonetheless, we can hope to use IRL on a diverse dataset of demonstrations to at least arrive at a prior that exhibits normative behavior. While humans might not be able to explicitly demonstrate certain values or goals, their adherence to norms is generally satisfied in daily behavior. Therefore, we use IRL to infer normative reward functions from demonstration data, which can then be combined with other sources of rewards at runtime. Furthermore, we combine several learned normative reward functions from different demonstrators, allowing for generalization beyond some narrowly observed behavior.

Multi-Objective Decision Making. Simultaneously training an agent with respect to different types of norms can be cast into various frameworks of RL, including meta-learning, multi-objective decision-making, constrained optimization, multi-task learning and multi-agent systems. Although each of the frameworks has its respective strengths and weaknesses, it is not clear a priori whether one is more suitable for the goal of value alignment than the other. However, when considering human reasoning capabilities, it becomes apparent that in order to make ethical decisions one is confronted with a decision that requires finding trade-offs (whether explicitly or implicitly) between one's own goals and a broader normative component [15]. Multi-objective learning most directly tackles this desideratum by considering game-theoretic notions of optimality and as such yields a promising value-alignment framework [88]. Henceforth, we study the value alignment problem in a multi-objective setting. Assuming fixed, well-defined intentions, we can train agents using *multi-objective reinforcement learning* (MORL) to find solutions that do well on each measure. To find a desired trade off, MORL algorithms then often optimize for a set of solutions, of which a user can choose by providing some form of personal preference [69]. Unfortunately, depending on the scenario, optimizing for a whole set of solutions might not be feasible or simply unnecessary. Besides that, when the respective objective functions represent normative behavior, one might have strong prior beliefs about what should be prioritized. This calls for an interactive approach to MORL, in which an agent can learn about human preferences in an online manner. We propose an algorithm based on active preference learning [74] which queries a user for learning a distribution over multi-objective scalarization weights. By providing pairwise preferences, the agent thus maintains a distribution over reward functions, which is maintained through Bayesian updating.

Besides human feedback efficiency, our approach has several convenient properties. Firstly, it allows for value-open design by learning multiple reward functions from different demonstrators. Despite this, the two-step procedure of first learning *what* is generally valuable and only afterwards training *how* to balance valuable behaviors can build a layer of protection against malicious preferences. This is because, when all (or at least most) demonstrated behavior intersects on certain principles, then steering the agent to break these very same principles from preferences becomes increasingly hard. Secondly, one can incorporate incentives which are easily encoded in a reward function next to the normative principles derived from demonstrations. From an applied AI perspective, this is important because one does not want to strip an agent from its powerful optimization capabilities. For example, intelligent solutions such as move 37 in the second game of AlphaGo versus Lee Sedol [4] are unlikely to arise from IRL alone, unless they are prominently present in the demonstration data set. However, if we do indeed care about winning “the game”, this can be incorporated through a multi-objective reward function and the agent will optimize for it when instructed to do so. In this context, the normative component can be seen as a prior, or regularizer that penalizes the system whenever it deviates too far from the demonstrations.

1.2. Research Questions

To address the shortcomings of technical multi-objective value alignment implementations, we aim to answer the following research questions:

1. How can we query and interact with experts to elicit normative behavior in sequential decision-making problems?
2. How can we encode learned normative behavior into RL agents?
3. How can we combine different (and possibly conflicting) aspects of normative behavior in sequential decision-making problems?

The first research question is mainly theoretically motivated. To answer it, we study constrained and multi-objective RL, the two relevant RL frameworks that extend the traditional notion of a scalar reward function for single agents. We’ll illustrate their respective strengths and weaknesses with the help of theorems and practical counterexamples that show in which setting one can expect to obtain a desirable solution. In contrast, the second and third research questions will be mainly answered using empirical methods. To do so, we’ll design two different environments that illustrate the limits of current value learning methods, as well as propose a new algorithm for dealing with the current shortcomings. Finally, while the first two research questions have been studied extensively in previous literature, we found that the third research question has been largely neglected as of right now and can not be tackled using state-of-the-art methods. For this reason, we consider our answers to the first two research questions to be incremental research that directly builds on previous methods, whereas the goal of studying the third research question is not only to propose a technical solution, but also to draw more general attention to the problem of aggregating sequential preference data.

1.3. Contributions

Our main contribution lies in the combination of value learning with multi-objective RL for deep learning agents. Naturally, this entails different degrees of novelty, including scalability as well as tractability. First of all, we study the problem of learning values from experts and encoding them into RL agents without making any assumptions about the underlying environment. In contrast to previous methods, this enables us to theoretically scale to significantly more complex environments. Hence, in chapter 3 we show that by combining adversarial inverse reinforcement learning with multi-objective optimization, we can steer an agent towards normative behavior without the need for manual feature engineering. By doing so, we tackle research questions 1 and 2.

While this extends previous methods, we show that an interactive multi-objective algorithm based on active learning can drastically reduce the introduced additional computational burden as well as offer a solution to the value aggregation problem for multiple experts. For this reason, in chapter 4 we propose a framework called MORAL (Multi-Objective Reinforced Active Learning), which uses deep inverse RL for learning multiple reward functions that can be subsequently used for interactively tuning agents with only few preferences. Furthermore, in chapter 5 we illustrate how MORAL can be used to learn from conflicting demonstrations and prove that, given expert agreement on a subset of trajectories, MORAL will always optimize for Pareto optimal solutions regardless of the preference data provided in the active learning step.

In combination, MORAL answers research questions 2 and 3, which we empirically validate by conducting experiments in two environments, *Burning Warehouse* and *Delivery* with different degrees of complexity. In both environments, we show that MORAL can successfully combine reward information from multiple sources to generalize beyond expert demonstrations and exhibit normative behavior while optimizing for a primary goal. Furthermore, we demonstrate that MORAL outperforms preference based deep RL, which further suggests that state-of-the-art value learning methods lack support for multi-objective decision-making. Finally, we perform ablation studies and show that MORAL is robust with respect to preference noise, while exhibiting significantly better performance compared to training without active learning.

1.4. Thesis Outline

The thesis (part I) is divided into a background chapter (chapter 2) which introduces the necessary mathematical concepts and basic algorithms of deep RL, three research chapters (chapters 3-5) describing the methodology and obtained results, as well as a concluding discussion (chapter 6). Furthermore, we have deferred a detailed analysis of related work to section 6.2. Besides this, part II features a scientific paper corresponding to the results of chapter 3. Finally, hyperparameters, detailed plots, implementation details and extended results are available in the appendix (part III), to which we include references at the appropriate positions during the main chapters.

2 Background

This chapter aims to formulate the underlying mathematical framework of RL, including algorithms for approximating optimal solutions starting in section 2.1. As a follow-up, we discuss maximum entropy, multi-objective and constrained RL in section 2.2 and define their respective notions of optimality. In section 2.3 we outline the fundamentals of deep learning and gradient-based optimization. Finally, section 2.4 introduces policy gradient methods, a class of state-of-the-art deep RL algorithms including proximal policy optimization, which will be the major training algorithm used throughout our research.

2.1. Reinforcement Learning

Reinforcement learning (RL) is the study of teaching an agent to learn arbitrarily complex behaviors by exploiting regularities in large amounts of feedback data. It has recently become one of the most promising approaches for the development of artificial intelligence (AI). Inspired by neuroscience and human psychology, RL aims to teach machines how to act in a certain environment by providing reward signals that are designed to convey what the agent ought to do and what it must not. While machine learning (ML) is primarily concerned with pattern recognition, i.e. tasks including classification, clustering and regression, RL is designed to solve pattern exploration and exploitation. As such, the RL framework theoretically allows machines to plan and execute any possible sequence of actions over the course of multiple time steps into the future. It should come to no surprise that, for this reason, the appeal of RL among artificial intelligence researchers is correspondingly high.

However, any such powerful framework comes with numerous difficulties that have to be overcome in order to make it applicable to real world tasks. Arguably, in the context of normative decision-making, the most prominent problem for RL research is the *credit assignment problem*, which describes the difficulty of ascribing relevance to past actions for delayed reward signals. For example, when training an agent to play the game of Go, the only objective reward signal available is whether or not a game has been won. However, it would be highly inefficient to assume that *all* actions during a won game were actually responsible for the winning outcome. In fact, some actions might have significantly decreased the agent's probability of winning despite the end result being positive. Another example naturally arises from any stochastic environment, where repeated actions in the same state might not lead to the same outcome. It is then the task of the RL agent to find a *policy*, a mapping from states of the world to a probability distribution over possible actions, that in expectation yields the desired outcome after a certain amount of (possibly infinite) environment interactions.

2.1.1. Markov Decision Processes

To mathematically model an agent's interactions with its environment, RL makes use of *Markov decision processes* [11]. Formally, Markov decision processes can be captured by the following:

Definition 2.1.1. A Markov Decision Process (MDP) consists of a tuple $\langle \mathcal{S}, \mathcal{A}, p, r, \mu_0 \rangle$, where

- $\mathcal{S}$ is the set of possible states,
- $\mathcal{A}$ is the set of valid actions,

- $p : \mathcal{S} \times \mathcal{A} \rightarrow \Delta_{\mathcal{S}}$ is a transition function, with $p(\cdot|s, a)$ denoting the probability distribution over next states in $\mathcal{S}$ when executing action a in state s,
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is a reward function, taking a state s, action a and next state s' and returning a reward $r(s, a, s')$,
- $\mu_0 \in \Delta_{\mathcal{S}}$ is a distribution over starting states.

Note that by definition of p, the Markov assumptions are automatically satisfied, i.e. the stochasticity of the environment is completely determined given only a current state-action pair (s, a). If such assumption does not hold, one needs to resort to more general notions of MDPs, such as the partially observable MDP (POMDP) [48]. This, however, is beyond the scope of this chapter and we will therefore assume that the Markov condition holds unless stated otherwise.

Within an MDP we typically assume a single agent executing actions in a sequential manner, see figure 2.1. After starting in an initial state S_0, at each time step t the agent executes an action A_t. To decide which action to take at time t, the agent inspects the current state S_t and acts according to a policy $\pi : \mathcal{S} \rightarrow \Delta_{\mathcal{A}}$, which returns a probability distribution over actions. In turn, the environment returns a reward R_t and next state S_{t+1}. In the case of a finite horizon MDP this is repeated until a final state S_T is reached, and is repeated indefinitely otherwise. This sequence of random variables $(S_0, A_0, S_1, A_1, \ldots)$ forms a stochastic process with realizations $\tau = (s_0, a_0, s_1, a_1, \ldots)$ which we will call *trajectories*. Due to the Markov property, the probability of a T-step trajectory τ given a followed policy π is simply given by

$$p(\tau|\pi) := \mu_0(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t), \tag{2.1}$$

where $\pi(a_t|s_t)$ denotes the probability of taking action a_t in state s_t.

While trajectories tell us *how* an agent behaves, they do not necessarily reveal *how well* an agent is acting in its environment. To quantify an agent's performance, we therefore need to define some measure of success. Typically, this notion of success is derived from the given reward function and takes the form of expected cumulative reward along generated trajectories. Formally, the finite-horizon reinforcement learning problem can be defined in terms of finding a policy $\pi : \mathcal{S} \rightarrow \Delta_{\mathcal{A}}$, which for a discount factor $0 < \gamma \leq 1$ maximizes the expected return

$$\max_{\pi} J(\pi) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T-1} \gamma^t r(S_{t+1}, A_t, S_t)\right], \tag{2.2}$$

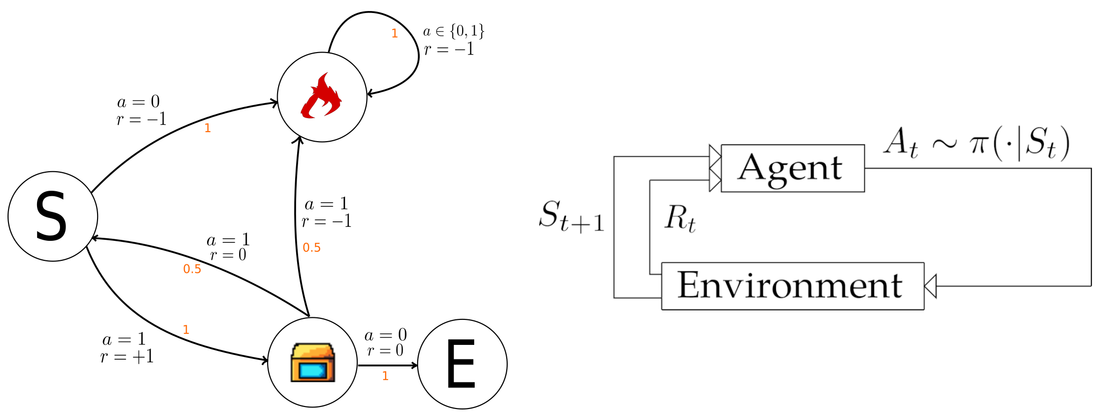


Figure 2.1: *Left*: A discrete Markov decision process with four states, two actions and stochastic transitions (orange). The agent starts in state S and can decide to retrieve a valuable object when executing $a = 1$. When repeating $a = 1$, the agent has a chance of going back to the starting state which would allow it to retrieve more valuables. However, this behavior puts the agent at risk of getting trapped in a burning room, leading to negative reward. Executing $a = 0$ after retrieving a valuable object allows the agent to escape and transition to a terminal state E. *Right*: The interaction loop between an agent and its environment.

where the expectation of a random variable X over π is understood as the average realized value of X over generated trajectories τ, namely

$$\mathbb{E}_\pi[X] := \int_\tau p(\tau|\pi)X(\tau). \tag{2.3}$$

Intuitively, the discount factor γ determines how myopic our agent is. For example, small values of γ influence the agent in taking decisions that maximize short term rewards. On the other hand, when γ is close to 1, we are optimizing for a less myopic agent which has only minor preferences for immediate rewards over long term returns. Since in practice we are mostly concerned about long term performance, the discount factor usually is set to values close to 1.

2.1.2. Value Functions & Bellman Equations

One important tool for finding optimal policies in RL algorithms is the *value function*. While the optimization objective in (2.2) is a global optimality criterion over the whole MDP, a value function gives us more fine-grained insight into the performance of a policy. Namely, for any given state s we define the value function $V_\pi(s) : \mathcal{S} \to \mathbb{R}$ of a policy π to be the expected cumulative reward when starting in state s:

$$V_\pi(s) := \mathbb{E}_\pi\left[G_t \mid S_t = s\right], \tag{2.4}$$

where $G_t := R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$ denotes the return obtained after time t.[1] In order to quantify the quality of actions, we will additionally define an action-value function:

$$Q_\pi(s,a) := \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right]. \tag{2.5}$$

The difference between the two is that a value function only tells us about the desirability of certain states, whereas the action-value function indicates the desirability of an action in such state. As such, finding an action-value function Q_{π^*} for an optimal policy $\pi^* = \arg\max_\pi J(\pi)$ is equivalent to solving the original optimization problem (2.2), since we can always obtain an optimal policy $\pi(a|s) := \arg\max_a Q_{\pi^*}(s,a)$ from an optimal action-value function.

It can be shown that the value function is the unique solution to a system of linear equations, which are better known as the *Bellman equations* [82]. Assume that $|\mathcal{S}| < \infty$ and $|\mathcal{A}| < \infty$. Then, for any policy π and state $s \in \mathcal{S}$ we can write

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} p(s'|s,a)\big[r(s,a,s') + \gamma\mathbb{E}_\pi[G_t \mid S_{t+1} = s']\big] \\ &= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} p(s'|s,a)\big[r(s,a,s') + \gamma V_\pi(s')\big]. \end{aligned} \tag{2.6}$$

Similarly to (2.6) we can derive Bellman equations for the value and action-value functions of the optimal policy. Let π^* be an optimal policy and V_{π^*}, Q_{π^*} be its respective value functions. It then holds that

$$\begin{aligned} V_{\pi^*}(s) &= \max_a Q_{\pi^*}(s,a) \\ &= \max_a \mathbb{E}_{\pi^*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi^*}[R_{t+1} + \gamma V_{\pi^*}(S_{t+1}) \mid S_t = s, A_t = a], \end{aligned} \tag{2.7}$$

and analogously for the action-value function we have

$$Q_{\pi^*}(s,a) = \mathbb{E}_{\pi^*}[R_{t+1} + \gamma \max_{a'} Q_{\pi^*}(S_{t+1},a') \mid S_t = s, A_t = a]. \tag{2.8}$$

[1] To reduce the notational burden, we will from now on not distinguish between G_t as a random variable and its realizations by both denoting them with the same symbol unless stated otherwise.

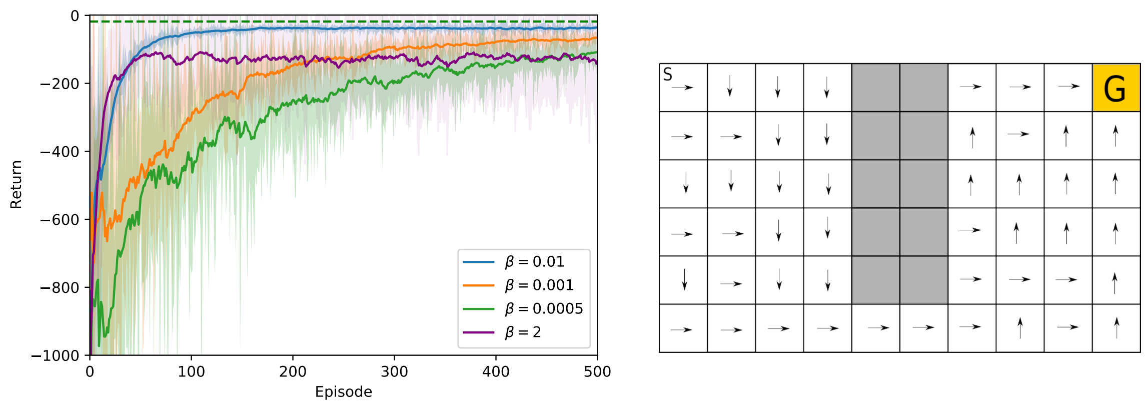


Figure 2.2: *Left*: Training performance of On-Policy Monte Carlo in the stochastic grid world navigation task for different exploration hyperparameters. The lines and shaded areas show mean and standard deviation respectively for each configuration averaged over 5 random seeds. *Right*: The stochastic gridworld environment. The agent starts in the top left state (S) and is trained to walk along the gray wall to the goal state G. The arrows indicate example greedy actions of the policy learned by Monte Carlo control.

Equations (2.7) and (2.8) are often used within RL algorithms to approximate an optimal policy and are often referred to as *value-based* methods. One of the most prominent value-based algorithms is Q-learning, which starts with a randomly initialized value function Q and iteratively applies equation (2.8) to find an optimal value function.

2.1.3. Monte Carlo Estimation & Exploration

A straight forward way to approximate value functions when environment dynamics are not known is *Monte Carlo* simulation. The main idea behind these methods is that, given enough time, an agent can abundantly sample experience by acting in its environment and then determine unbiased means from the distribution of observed returns G_t for any state-action pair (s, a). An obvious drawback of this approach is the need to complete a whole episode before any learning can be made, which might result in slow learning for long horizon tasks.

The most prevalent problem, however, is that once the dynamics are unknown, we need some mechanism for efficiently guiding our search over promising state-action pairs. Any agent without knowledge about the environment dynamics will first have to sufficiently *explore* in order to find states that yield high rewards. Exploration is a fundamental difficulty for most RL algorithms, and it is frequently referred to as the *exploration-exploitation trade-off* [82]. This trade-off describes the problem of finding a right balance between exploiting an agent's knowledge about the world versus gathering new information about the world. While exploring can reveal previously unseen and possibly superior states of the world, exploitation of already known strategies likely lead to higher rewards in the short term.

Monte Carlo RL algorithms usually tackle this dilemma by maintaining a policy with sufficient entropy. When dealing with discrete actions, this can be ensured by explicitly setting $\pi(a|s) > \epsilon$ for some $\epsilon > 0$. In this case, the parameter ϵ determines the level of exploration and is usually set to follow a decreasing schedule over the time of learning. This way, the agent will explore avidly at the start of training, but increasingly focuses on only executing actions that have shown to lead to high returns.

We illustrate Monte Carlo estimation and exploration with the aid of a simple control scheme, see algorithm 1. The algorithm follows the scheme of generalized policy iteration [82], which translates to learning an action-value function $Q(s, a)$ of the current policy and then making the policy greedy with respect to this function. In practice, it is however not necessary to precisely estimate the action-value function for current policies. For that reason, we can update π each time we have an updated estimate of Q_π. Estimates for Q_π are obtained by generating trajectories with π and saving the observed returns G_t for each state-action pair (s, a). Taking the average over these returns gives a (biased) approximation for $Q_\pi(s, a)$. Finally, policy improvement is done by making the policy more greedy with respect to the newly obtained action-value function. However, to keep adequate levels of entropy, we only make π

Algorithm 1: On-Policy Monte Carlo Control

```
Result: Estimated optimal policy $\pi^*(a|s)$, action-value function $Q_{\pi^*}(s,a)$
Input: Decay parameter $\beta > 0$, initial exploration $\epsilon_0 > 0$
Initialize: $J(s,a)$ as empty list, $Q(s,a) \in \mathbb{R}$ and $\pi$ such that $\pi(a|s) = \frac{1}{|\mathcal{A}|}$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
for $i = 0, 1, \dots$ do
    $\epsilon = \epsilon_0 \cdot \exp(-\beta i)$
    Generate trajectory $\tau = (s_0, a_0, r_1, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_T)$ with $\pi$
    $G = 0$
    for $t = T-1, \dots, 0$ do
        Estimate go-to return $G = \gamma G + r_{t+1}$
        Append $G$ to $J(s_t, a_t)$
        $Q(s_t, a_t) = \frac{1}{|J(s_t,a_t)|} \sum_{x \in J(s_t,a_t)} x$
        $a_{greedy} = \arg\max_a Q(s_t, a)$
        for $a \in \mathcal{A}$ do
            if $a = a_{greedy}$ then
                $\pi(a|s_t) = 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|}$
            else
                $\pi(a|s_t) = \frac{\epsilon}{|\mathcal{A}|}$
            end
        end
    end
end
```

greedy with respect to Q under the constraint that $\pi(a|s) > \epsilon$ for a dynamically changing $\epsilon > 0$.

We test the performance of this algorithm in a stochastic grid world environment, see figure 2.2. The agent starts in the top left corner of a 6×10 grid and has to navigate past a wall to a goal state located on the other side. There are four available actions, one for moving in each direction respectively. Furthermore, the agent receives a negative reward of -1 at each non-terminal state and has a 25% chance of moving in a random direction each time it executes an action. When the agent reaches the goal state, it receives a reward of 0 and the episode ends. To illustrate the trade-off between exploration and exploitation, we train the Monte Carlo agent for different values of the exploration decay hyperparameter β. Figure 2.2 shows that for $\beta = 0.01$ the algorithm converges to an optimal solution after less than 200 episodes of training. Decreasing β leads to more exploration and thus leading to overall slower convergence. On the other hand, when exploration is decayed prematurely, the agent sometimes fails to discover the optimal policy, despite exhibiting faster learning progress at the start.

2.1.4. Value Approximation

During the previous sections, we have assumed that it was feasible to have a *tabular* representation of the state-action space. However, this assumption trivially fails in problems with continuous or very large discrete spaces. For example, consider the problem of playing the board game of Go. Despite its simple rule set, a standard 19×19 board allows for approximately $2.1 \cdot 10^{170}$ legal states [85], which renders storing a tabular array of $V_\pi(s)$ for all $s \in \mathcal{S}$ physically impossible. Furthermore, learning a separate value for each state might be highly inefficient, since nearby states can share a majority of information about their respective values. One solution to this problems lies in approximating policies and their value functions by a parametrized statistical model f_θ. To do so, we introduce a learnable parameter $\theta \in \mathbb{R}^d$ and, in the case of value learning, aim to find θ such that our estimation $V_\theta(s)$ approximately matches the desired true value $V(s)$. Typically, state-of-the-art methods use artificial neural networks and tune θ accordingly by stochastic gradient descent. For now, we defer the discussion of the exact representation to section 2.3 and focus merely on correctly learning V_θ.

We will now describe a simple Monte Carlo scheme for estimating V_π for a fixed policy π in the case of function approximation. In the tabular case, we were able to directly update $V(s)$ for each $s \in \mathcal{S}$ by sampling unbiased estimates of the returns G_t under π. While we can still obtain the same unbiased estimates in the approximate case, we will first have to define an objective function for updating θ. At first, it might seem apparent that we would like to minimize

$$\min_{\theta} |V_\theta(s) - V_\pi(s)| \tag{2.9}$$

for each state s separately. However, such local update criterion will not work in this case, since changing θ can potentially change the values of $V_\theta(s')$ for all other states s'. Thus, it is necessary to define a global criterion which measures the performance of θ with respect to all states. To do so, let $\mu(s) \in \Delta_{\mathcal{S}}$ be a distribution over states in the MDP. Furthermore, we define a mean squared value loss [82] as

$$\mathcal{L}^V(\theta) := \frac{1}{2} \int_{s \in \mathcal{S}} (V_\pi(s) - V_\theta(s))^2 d\mu(s). \tag{2.10}$$

The distribution μ serves as a weighting function that determines which states need to have more accurate estimates. This makes sense, since in large MDPs we will likely only need to visit a very small fraction of the state space, on which we would like to focus on when finding θ. Typically, μ is chosen to match the distribution of states visited by the policy (cf. section 2.4).

Having defined a criterion which θ needs to minimize, we can now apply gradient based optimization for finding an update rule accordingly. Let $\nabla_\theta f(\theta)$ denote the gradient of a function f with respect to the vector-valued input θ. Then, under the assumption that we can exchange integral with differentiation, we have

$$\nabla_\theta \mathcal{L}^V(\theta) = -\int_{s \in \mathcal{S}} (V_\pi(s) - V_\theta(s)) \nabla_\theta V_\theta(s) d\mu(s), \tag{2.11}$$

where the integral is taken component wise. When $\mu(s)$ is the distribution of states visited by π, sampling the random variables $(V_\pi(S_t) - V_\theta(S_t))\nabla_\theta V_\theta(S_t)$ provides a way for estimating (2.11) iteratively. Since the gradient $\nabla_\theta \mathcal{L}^V(\theta)$ is the direction in which the function most rapidly increases, taking a step into the negative direction will then lead to the error decreasing. This method is also known as *stochastic gradient descent*, which we will further discuss in section 2.3.

Combining stochastic gradient descent with the Monte Carlo value estimation from section 2.1.3 leads us to a straight forward update rule for finding the value of a fixed policy when using function approximation. Namely, we replace $V_\pi(S_t)$ in our gradient samples of (2.11) with the unbiased estimate G_t and update θ by setting

$$\theta_{k+1} := \theta_k + \alpha(G_t - V_{\theta_k}(s_t)) \nabla_\theta V_{\theta_k}(s_t). \tag{2.12}$$

Under certain conditions about the learning rate α and the regularity of (2.11), it can be shown that the above procedure does indeed converge to a locally optimal value function [13]. While this gives us a practical algorithm for value estimation, the problem of finding an optimal policy with function approximators persists. In theory, one could extend (2.12) to approximating $Q_\pi(s, a)$ and proceed with generalized policy iteration by maintaining an ϵ-greedy policy with respect to Q. However, estimating Q this way turns out to be too noisy for most real-world applications. In section 2.4 we will introduce a different class of approximate policy learning algorithms, called *policy gradient* methods, which are the current state of the art way of utilizing Monte Carlo return estimates for inferring optimal behavior.

2.2. Extended Decision Making Frameworks

Markov decision processes offer a tractable optimization goal for the development of intelligent agents, yet remain sufficiently complex for modelling a wide range of real-world sequential decision-making problems. On the other hand, it should be clear that the assumption of a single reward function being able to determine all intelligent behavior is merely a hypothesis [81, 82] with meager evidence. Besides that, even if it was theoretically feasible to achieve such goal if one could define the correct reward function, this would not imply that an agent would actually be able to learn this in practice. On the one hand, approximate methods are prone to getting stuck in local optima [44] and, in the worst case, might diverge completely when the variance of observed returns is too high [89]. For these reasons, a variety of extensions of MDPs exist to extend the notion of rewards. We will start this section by defining the framework of maximum entropy reinforcement learning and show how this relates to a constrained optimization problem. As a follow-up, we will focus on extending the reward to a vector-valued function.

2.2.1. Maximum Entropy Reinforcement Learning

Maximum entropy reinforcement learning alters the reward maximization problem (2.2) by adding a penalty for policies with low intrinsic randomness. Formally, we define the entropy of a random variable X as

$$H(X) := \mathbb{E}\left[-\log P(X)\right]. \tag{2.13}$$

Using this definition, the maximum entropy RL goal reduces to

$$\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \gamma^t \big(r(S_t, A_t) + \beta H(\pi(\cdot|S_t))\big)\right], \tag{2.14}$$

where $\beta > 0$ is a trade-off hyperparameter. This goal forces the policy to maximize the expected cumulative rewards while remaining as stochastic as possible. Adding such entropy term to the reward function has been proven successful in state-of-the-art RL algorithms [37, 59] as well as learning from demonstrations [101], on which we will elaborate further in chapter 3. Besides that, we note that finding a solution to maximum entropy RL is straight forward when using Monte Carlo estimation. In the case of temporal difference bootstrapping, one can furthermore derive Bellman equations that include the entropy penalty, which allows for adapting existing RL algorithms to the new goal.

2.2.2. Constrained Problems

Another disadvantage of traditional reward maximization is the lack of performance guarantees. However, in safety critical environments we often expect the agent to meet certain criteria first before we care about its obtained rewards. For example, we might want to prevent that our path planning agent from section 2.1.3 bumps into the gray wall too often. Adding a negative penalty to the reward function would naturally achieve this, but it comes with the disadvantage that the primary objective of getting to the goal is then competing with the safety constraint. This way, our agent could allow itself to bump into the wall more often in case this leads to getting to the goal state more quickly.

Fortunately, such behavior can be effectively circumvented by splitting the space of policies Π into a set of feasible and infeasible policies. To do so, we define *constrained Markov decision processes* (CMDP) [5]. Formally, we call a tuple $\langle \mathcal{S}, \mathcal{A}, p, r, \mu_0, \mathbf{c}, \mathbf{d}\rangle$ a CMDP, which is obtained by extending an MDP $\langle \mathcal{S}, \mathcal{A}, p, r, \mu_0\rangle$ with a (multivariate) constraint function $\mathbf{c} = (c_1, \dots, c_k) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^k$ as well as a constraint hyperparameter $\mathbf{d} \in \mathbb{R}^k$. The CMDP goal is to solve

$$\max_{\pi} \quad J(\pi) \tag{2.15}$$

$$\text{s.t.} \quad J_{\mathbf{c}}(\pi) \leq \mathbf{d}, \tag{2.16}$$

where $J_{\mathbf{c}}(\pi) = \mathbb{E}_{\pi}[\sum_{t=0}^{T} \gamma^t \mathbf{c}(S_t, A_t)]$ denotes the vector of cumulative expected constraint costs. Note that since the constraint function $\mathbf{c}$ follows the same structure of the original reward r, the constrained optimization problem above captures a multitude of problems beyond applications in safety. For instance, setting $c(s, a) = \pi(a|s) \log \pi(a|s)$ will result in a constrained version of the maximum entropy reinforcement learning goal (2.14) which can be shown to share the same optimal solutions for correct choices of β and d [54]. However, finding a solution to the constrained MDP problem requires more sophisticated methods than those of standard RL. This is because evaluating the constraint satisfaction (2.16) involves the estimation of a value function which in itself is an expensive operation. Since finding solutions to CMDPs is beyond the scope of this chapter, we will from now on assume that there are efficient methods for solving CMDPs without further elaboration.

2.2.3. Multi-Objective Solution Sets

The way constraints are represented in CMDPs leads to an implicit ordering in the prioritization of objectives. Namely, constraints need to be satisfied regardless of how much this deteriorates the performance of the agent with respect to the primary reward. On the other hand, the magnitude of constraint costs becomes irrelevant as soon as it meets the specified threshold. Arguably, CMDPs are therefore most useful when the constraints as well as the expected inequality threshold are known a priori. The framework of multi-objective reinforcement learning [69] provides methods for finding trade-offs between competing objectives when this is not the case.

Multi-objective Markov decision processes (MOMDP) consist of a tuple $\langle \mathcal{S}, \mathcal{A}, p, \mathbf{r}, \mu_0, \Omega, f_\Omega \rangle$ that extends the MDP with a vector-valued reward function $\mathbf{r}(s, a) \in \mathbb{R}^m$, where Ω is a set of preferences and $f_{\boldsymbol{\omega}}(\mathbf{r})$ are preference functions. Preference functions take the multi-objective reward and output a scalar that reflects a given preference $\boldsymbol{\omega} \in \Omega$ over the objectives. In general, the goal of a multi-objective reinforcement learning problem is problem dependent and cannot be as clearly defined as in the constrained case. We refer to Roijers et al. [69] for a general taxonomy of multi-objective RL problems and only elaborate on the decision support scenario, which is the relevant paradigm in the context of normative decision making.

When training an agent for the purpose of decision support, the MOMDP goal consists of finding a set of plausible policies from which a user can then choose by providing their personal preferences. Figure 2.3 illustrates this in the case of two-dimensional rewards. In the first step, a multi-objective RL (MORL) algorithm identifies a set of promising policies corresponding to a diverse set of preferences, whereas in the second step a final policy is selected from this set at execution time. The criteria for which desirable policies are selected in the first again depends on the algorithm. Nonetheless, a widely accepted solution concept when dealing with multiple competing objectives is *Pareto efficiency*.

Pareto efficient solutions correspond to the policies for which we can assure that no other policy exists that strictly improves upon all objectives. Together, the set of Pareto efficient solutions is called a Pareto front, or Pareto boundary, and is defined as

$$\mathcal{F} := \{\pi | \pi \in \Pi \wedge \nexists \pi' \neq \pi : J_{\mathbf{r}}(\pi') \geq J_{\mathbf{r}}(\pi)\}, \tag{2.17}$$

where $J_{\mathbf{r}}(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t \mathbf{r}(S_t, A_t)]$. In general, we can assume very little about the shape of $\mathcal{F}$ and in fact it might not necessarily be convex [97]. As such, optimizing for a complete Pareto boundary is a highly demanding task and not yet feasible in complex MOMDPs. For this reason, numerous state-of-the-art methods make the assumption of linear preference functions $f_{\boldsymbol{\omega}}(\mathbf{r}(\mathbf{s}, \mathbf{a})) = \boldsymbol{\omega}^T \mathbf{r}(s, a)$. From now on, we will follow this assumption and will only consider linear combinations of the vector-valued rewards unless noted otherwise.

Linear preference functions allow for optimizing a tractable subset of the Pareto front, which we will call the *convex coverage set* (CCS). The CCS consists of all Pareto efficient solution that are optimal for some linear preference:

$$\mathcal{F}^* := \{\pi \in \mathcal{F} \mid \exists \boldsymbol{\omega} \in \Omega : \boldsymbol{\omega}^T J_{\mathbf{r}}(\pi) \geq \boldsymbol{\omega}^T J_{\mathbf{r}}(\pi'),\ \forall \pi' \in \mathcal{F}\}. \tag{2.18}$$

As the name suggests, the CCS is convex by definition, by which we mean that it is closed under policies maximizing convex combinations of preference vectors. To see why, consider $\pi_1, \ldots, \pi_n \in \mathcal{F}^*$ with corresponding preferences $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_n$. Then, form an affine combination $\boldsymbol{\omega}_\lambda = \sum_{i=1}^{n} \lambda_i \boldsymbol{\omega}_i \in \Omega$ with $\sum_{i=1}^{n} \lambda_i = 1$. Because of linearity, we can bound $\boldsymbol{\omega}_\lambda^T J_{\mathbf{r}}(\pi) \leq \max_i \boldsymbol{\omega}_i^T J_{\mathbf{r}}(\pi_i)$. By maximizing this expression over π we conclude that the optimal policy $\pi_\lambda = \arg\max_\pi \boldsymbol{\omega}_\lambda^T J_{\mathbf{r}}(\pi)$ with respect to $\boldsymbol{\omega}_\lambda$ must be in the set $\{\pi_1, \ldots, \pi_n\} \subset \mathcal{F}^*$. An appealing property of the CCS is that it can be approximated by directly solving regular MDPs for the *scalarized* reward function $\boldsymbol{\omega}^T \mathbf{r}$. Furthermore, we can expect

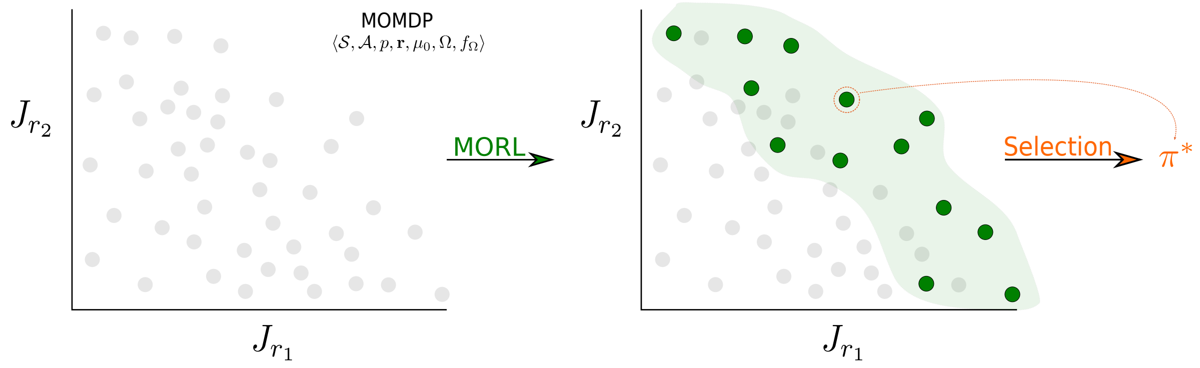

MOMDP
$\langle \mathcal{S}, \mathcal{A}, p, \mathbf{r}, \mu_0, \Omega, f_\Omega \rangle$
J_{r_2}
J_{r_1}
MORL
J_{r_2}
J_{r_1}
Selection
π^*

Figure 2.3: A conceptual multi-objective reinforcement learning algorithm for two reward functions. The goal of a multi-objective decision support agent is to be able to learn set of desirable policies (green). At deployment time, a user specifies their preferences which lead to the selection of a correspondingly optimal policy (orange).

similar $\boldsymbol{\omega}$ to yield optimal policies that are close to each other in the space of returns. This allows us to exploit similarity between different policies to learn from each other for reducing sample complexity in multi-objective RL.

2.3. Generalization

One of the fundamental components of human intelligence is the ability to *generalize*. Generalization denotes the capability of making accurate predictions on a set of previously unseen data points after being trained on relatively few examples from the same distribution. For example, consider teaching a child to tell apples and pears apart. While there might exist infinitely many feasible variations of apples and pears, it is sufficient to show the child a small and finite amount of samples before it will have learned an internal representation for each of the respective fruits, thus equipping it with remarkable generalization capabilities in the domain of fruit classification.

Unfortunately, generalization as such is a highly anthropocentric concept, which makes it a major challenge for the design of artificially intelligent agents. What might seem similar to humans is not guaranteed to appear close to a machine, because of the underlying differences in computation and information processing mechanisms. However, one can seek inspiration from the mammalian brain when aspiring to build electric circuits that ought to achieve similar, if not superior generalization powers than those of humans.

Deep learning (DL) aims to do exactly that and has become an inevitable tool for building generalizable AI systems over the past decade. Through the use of artificial neural networks, which constitute a highly simplified mathematical model of neural computation, DL leverages big data for solving high dimensional tasks including image classification, natural language processing, time series forecasting and synthetic data generation. The field inherits its name from the type of used network architectures, which consist of several layers of neurons stacked above each other, resulting in a computation graph of significant depth that can contain anything between hundreds to billions of learnable parameters. However, any complex model is only as powerful as its optimization algorithms allow it to be. By design, deep neural networks are differentiable models, which allows them to be trained with gradient based optimization. Given a *loss function*, which is a function of the network parameters that we want to optimize, we can therefore (approximately) calculate how the parameters have to be changed in order to decrease the loss. As it turns out, this is sufficient for obtaining models with remarkable capabilities, even though the reasons for this still remain to be fully understood.

2.3.1. Feedforward Networks

Feedforward networks form the underlying prototype of architectures for deep learning. In their most basic form, they take an input vector $\mathbf{x} \in \mathbb{R}^n$ and apply a sequence of functions $f_k : \mathbb{R}^{d_{k-1}} \to \mathbb{R}^{d_k}$ to arrive at an output vector $\hat{\mathbf{y}} \in \mathbb{R}^m$ [35]. The functions f_k are themselves composed of a linear transformation followed by a nonlinear function, also called *activation function*. To ensure differentiability of the model, the activation function is ensured to be differentiable (almost everywhere), which results in the feedforward network being a composition of differentiable functions. The motivation for such choice is that, while remaining differentiable as well as computationally efficient, we can hope to approximate arbitrary (continuous) functions when composing sufficiently many functions f_k. In fact, the universal approximation theorem [46] guarantees that this holds for feedforward networks with as little as one intermediate layer of computation. However, research has shown that relying on such theoretical statements alone does not facilitate the learning of complex functions [7, 87]. For that reason, state-of-the-art methods typically employ several stacked layers, resulting in a deep multilayer architecture.

The multilayer perceptron (MLP) is a feedforward network consisting of multiple layers of computation, called perceptrons. Formally, a perceptron consists of a linear transformation $\mathbf{W} \in \mathbb{R}^{n \times m}$, a bias vector $\mathbf{b} \in \mathbb{R}^m$ and an activation function $g : \mathbb{R} \to \mathbb{R}$ which, when combined, take an input vector $\mathbf{x} \in \mathbb{R}^n$ and compute

$$\hat{\mathbf{y}} = f(\mathbf{x}; \mathbf{W}, \mathbf{b}) := g(\mathbf{W}^T\mathbf{x} + \mathbf{b}), \tag{2.19}$$

where g is applied element wise. The multilayer perceptron then consists of multiple stacked layers $\{\mathbf{W_1}, \mathbf{b_1}, \dots \mathbf{W_k}, \mathbf{b_k}\}$ of the form (2.19), where each intermediate step is saved in what is called a *hidden layer*. Figure 2.4 shows an example multilayer perceptron consisting of three hidden layers. Assuming that we are using a fixed activation function across all layers, its output can be decomposed

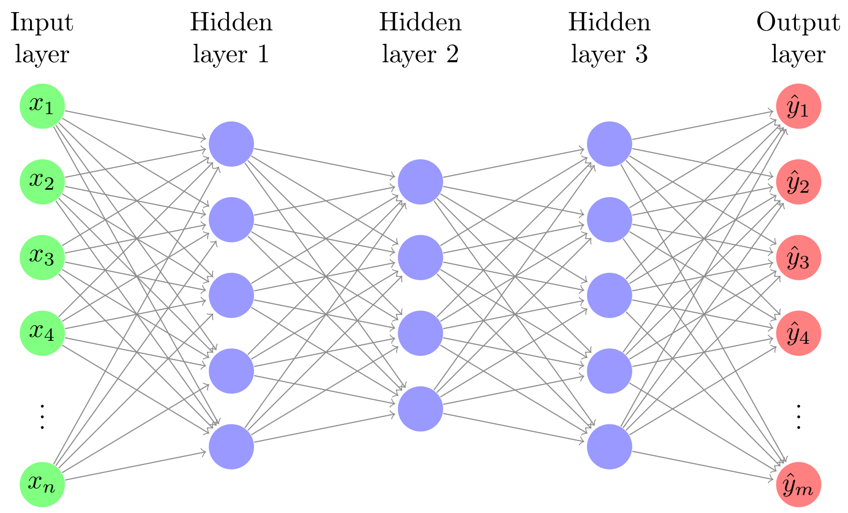


Figure 2.4: A multilayer perceptron which takes an input vector $\mathbf{x} \in \mathbb{R}^n$ and returns a vector-valued output $\hat{\mathbf{y}} \in \mathbb{R}^m$ after passing $\mathbf{x}$ through three hidden layers $\mathbf{h_1} = f_1(\mathbf{x})$, $\mathbf{h_2} = f_2(\mathbf{h_1})$, $\mathbf{h_3} = f_3(\mathbf{h_2})$.

in a hierarchical manner as

$$\begin{aligned}\hat{\mathbf{y}} &= \mathbf{W_4}^T\mathbf{h_3} + \mathbf{b_4}\\ \mathbf{h_3} &= g(\mathbf{W_3}^T\mathbf{h_2} + \mathbf{b_3})\\ \mathbf{h_2} &= g(\mathbf{W_2}^T\mathbf{h_1} + \mathbf{b_2})\\ \mathbf{h_1} &= g(\mathbf{W_1}^T\mathbf{x} + \mathbf{b_1}).\end{aligned} \tag{2.20}$$

Note that in the formula above, we did not apply the activation function g to the final layer. This is an arbitrary choice and depends on the type of problem that the multilayer perceptron aims to tackle. For example, when trying to perform a classification task it is common to normalize the output $\hat{\mathbf{y}}$, such that its values add up to one. This way, each element can be interpreted as a probability, or confidence score, that the input belongs to a certain class. We will elaborate on this further in section 2.3.2.

From now on, we will refer to the set $\{\mathbf{W_1}, \mathbf{b_1}, \dots \mathbf{W_k}, \mathbf{b_k}\}$ as network parameters, or weights, and will denote them as a single parameter $\theta \in \mathbb{R}^d$ corresponding to a neural network f_θ, where d is the total number of parameters. The activation function is not included in θ since it is assumed to be fixed and is not learned. Furthermore, unlike the type of output we desire from the neural network, the choice of activation function is less problem dependent. State-of-the-art methods frequently employ the rectified linear unit (ReLU) activation function $g(x) = \max(0, x)$ which has been shown to outperform various other choices of activation functions [60]. An appealing property of ReLU is that it is fast to compute, has an easy to compute derivative (almost everywhere) while enforcing sparsity by setting negative inputs to zero. On the other hand, this comes at the disadvantage of vanishing gradients when many inputs are smaller than zero, which can slow down gradient-based optimization (cf. section 2.3.3). A straightforward fix for this problem, however, consists of simply altering the slope of the function by setting $g(x) = \max(0, x) + \alpha \min(0, x)$ where $\alpha > 0$ controls the angle of negative slope.

To illustrate the importance of activation functions, consider figure 2.5. First, we generate a two-dimensional point cloud by independently sampling two Gaussian random variables $X_1, X_2 \sim \mathcal{N}(0, 1)$. We then split the dataset into two classes by drawing concentric circles around the origin until both classes contain roughly the same amount of points. Secondly, we construct an MLP with the same architecture as described above (2.20). Finally, without going into detail about the training process, we find MLP parameters θ such that the output most closely matches the ground truth class for each point in the dataset. We do this twice, once for an MLP with the ReLU activation function and once with the identity function. As can be seen, the MLP with the nonlinear ReLU activation function captures the structure in the data significantly better. This is not surprising, since the MLP with the identity activation function is merely a concatenation of affine functions, thus collapsing to an overall affine funcion itself. As a result, this linear MLP is restricted to a constant gradient which prevents it from separating the circular shaped classes.

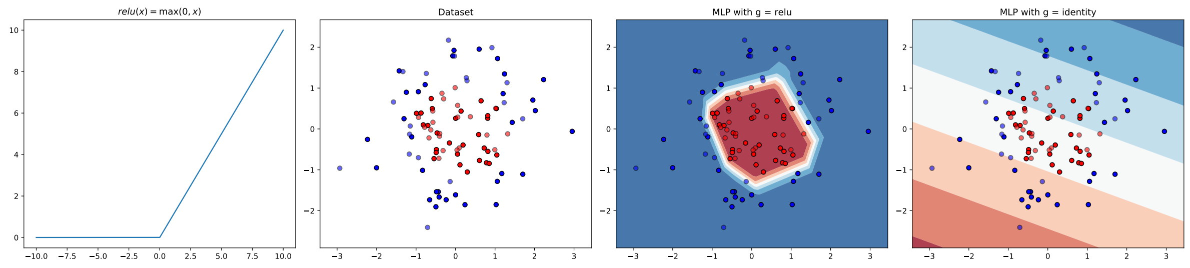


Figure 2.5: The importance of activation functions. The ReLU function introduces a slight nonlinearity into the MLP, which allows for classifying an isotropic Gaussian point cloud into two linearly inseparable classes.

2.3.2. Loss Functions

As discussed previously, the type of desired neural network output $\hat{\mathbf{y}}$ depends on the task at hand. The purpose of loss functions is to formally define this task and simultaneously provide a performance measure which can be compared across different architectures. Although it is hard to provide a complete taxonomy of loss functions due to the wide array of tasks that neural networks are theoretically capable of tackling, the generic task usually consists of modelling the probability distribution p_{data} of the available dataset [35]. In terms of mathematical statistics, we can frame this as *maximum likelihood estimation*. Formally, we call a parameter θ^* a maximum likelihood estimate (MLE) of the probability distribution of some data $\mathbb{X} = \{\mathbf{x}^1, \ldots, \mathbf{x}^m\}$ if and only if

$$\theta^* := \arg\max_\theta \prod_{i=1}^{m} p_\theta(\mathbf{x}^i), \tag{2.21}$$

where $p_\theta(\mathbf{x})$ is the model's estimated likelihood of observing $\mathbf{x}$ from p_{data}. Note that in this definition we made a major assumption that the data $\mathbb{X}$ is *independent and identically distributed* (i.i.d.). While this might not hold in practice, the i.i.d assumption is necessary to make learning probabilistic models tractable. In some sense, the MLE is our best estimate of p_{data} since it directly finds θ such that the observed data was as likely as possible to be observed.

The definition of the MLE is mathematically convenient. Nonetheless, we can derive a slightly more numerically stable and more commonly used target which turns out to be equivalent to the MLE. First, we note that since the natural logarithm is a monotonically increasing function, we can apply the logarithm to expression 2.21 without changing the maximizing argument

$$\theta^* = \arg\max_\theta \sum_{i=1}^{m} \log p_\theta(\mathbf{x}^i). \tag{2.22}$$

This is useful for large datasets, because it avoids rounding errors that could occur when calculating the product of possibly small probability values. Furthermore, we call $\hat{p}_{\text{data}}$ the empirical cumulative distribution function (ECDF) of $\mathbb{X}$, which puts an equal probability mass of $\frac{1}{m}$ to each of the points in $\mathbb{X}$ [98]. After multiplying by m, we reframe the expression 2.22 in terms of the ECDF by writing

$$\theta^* = \arg\max_\theta \mathbb{E}_{\hat{p}_{\text{data}}}[\log p_\theta(\mathbf{x})]. \tag{2.23}$$

This equation has an alternative interpretation to the MLE, which is that we want to minimize the dissimilarity between the two distributions $\hat{p}_{\text{data}}$ and p_θ. In fact, this interpretation can be proven formally, by defining the Kullback Leibler (KL) divergence [53] of two distributions p and q as

$$D_{KL}(p||q) := \int p(x) \log \frac{p(x)}{q(x)} dx. \tag{2.24}$$

If we plug in the two distributions $\hat{p}_{\text{data}}$ and p_θ into the KL divergence and rewrite the integral as an expectation over $\hat{p}_{\text{data}}$, we obtain

$$D_{KL}(\hat{p}_{\text{data}}||p_\theta) = \mathbb{E}_{\hat{p}_{\text{data}}}[\log \hat{p}_{\text{data}}(\mathbf{x}) - \log p_\theta(\mathbf{x})]. \tag{2.25}$$

Finally, since $\log \hat{p}_{\text{data}}$ does not depend on θ, minimizing this KL divergence over θ coincides with the MLE optimization objective (2.23).

Minimization of the KL divergence can be naturally cast into many practical machine learning problems. The corresponding loss function $\mathcal{L}(\theta)$ is then obtained by reformulating the negative argument in (2.23) as a function of θ, which is commonly referred to as the *cross entropy loss function.* For example, consider the task of learning a binary classifier f_θ which, given some input $\mathbf{x}$ outputs the probability of corresponding to the class $y = 1$ as opposed to $y = 0$. In this case, we are aiming to find a conditional distribution $p_\theta(y|\mathbf{x})$, where $\log \hat{p}_{\text{data}}(y|\mathbf{x})$ is equal to 1 if and only if y is the true label for $\mathbf{x}$. Due to the special structure of $\hat{p}_{\text{data}}$ the cross entropy reduces to a simplified form

$$\mathcal{L}(\theta) = -\sum_{i=1}^{m} y^i \log f_\theta(\mathbf{x}^i) + (1 - y^i)\log(1 - f_\theta(\mathbf{x}^i)), \tag{2.26}$$

where $y^i \in \{0, 1\}$ is the observed label for $\mathbf{x}^i$.

2.3.3. Optimization & Model Selection

Having defined a loss function $\mathcal{L}$ and a network architecture with parameter θ, it remains to be discussed how to train neural networks in practice. That is, we would like to tune θ such that $\mathcal{L}(\theta)$ becomes minimal. Generally, we rely on gradient-based optimization with *stochastic gradient descent* (SGD) by moving θ into the opposite direction of $\nabla_\theta \mathcal{L}(\theta)$. The calculation of the gradient is, unfortunately, infeasible for two reasons. Firstly, deep neural networks compose multiple layers of computation including nonlinear activation functions. This makes an exact derivation of the gradient challenging, which is why we typically rely on automatic differentiation [9]. Automatic differentiation allows to calculate gradients numerically without having to derive it in an exact symbolic form. Secondly, the gradient $\nabla_\theta \mathcal{L}(\theta)$ is too expensive to compute due to it depending on the complete dataset, which is why we instead sample *batches* of data for approximation. Depending on the batch size, this unfortunately introduces considerable amounts of noise into the calculation of $\nabla_\theta \mathcal{L}(\theta)$ that can slow down convergence.

From a theoretical point of view, such noise is not problematic as long as its variance is bounded over time - convergence to a stationary point is guaranteed for appropriate learning rate schedules [13]. Nonetheless, there is a need for taking this discrepancy between theory and practice into account. For that reason, several alternative gradient methods have been introduced, including RMSProp [84], Adam [49] and AdaGrad [26]. Mostly backed by empirical evidence, these adaptive learning algorithms keep track of past gradient statistics, such as means and standard deviations and have been shown to exceed the convergence speed of regular SGD for most machine learning problems. In the case of Adam, we keep a running exponential average of calculated gradients ∇_θ and their squared values ∇_θ^2 (to be understood as a point wise product) in variables v_i and r_i respectively. The rate of decay is given by hyperparameters ρ_v and ρ_r respectively. Finally, we ensure that v_i and r_i are unbiased estimates of the gradient statistics and update θ accordingly:

$$\begin{aligned}
v_i &= \rho_v v_{i-1} + (1 - \rho_v)\nabla_\theta \\
\hat{v}_i &= \frac{v_i}{1 - \rho_v^i} \\
r_i &= \rho_r r_{i-1} + (1 - \rho_r)\nabla_\theta^2 \\
\hat{r}_i &= \frac{r_i}{1 - \rho_r^i} \\
\theta' &\leftarrow \theta - \alpha \frac{\hat{v}_i}{\sqrt{\hat{r}_i + \epsilon}}.
\end{aligned} \tag{2.27}$$

The Adam optimizer proves to be a powerful tool for finding (locally) optimal solutions of neural networks. Nonetheless, this does not necessarily guarantee generalization performance due to the loss function landscape depending on the training data. Thus, one needs to consider different mechanisms for model selection which incorporate the out of distribution performance that cannot be inferred from the training data. Traditionally, this is achieved by comparing models by their loss on a separate *test* set. Measuring performance on the test set is in some sense the most effective way for estimating generalization capabilities.

When a model performs exceptionally well on the training data, but fails to generalize to test data from the same distribution, we call our model *overfitting*. Overfitting can happen when the model's parameters are overly specialized to just solve the task on the training dataset. In the extreme case, deep neural networks can have enough parameters to completely interpolate the training data by achieving an accuracy of 100%. In statistical learning theory, such phenomenon would be considered harmful due to an inherent trade-off between a model's bias and its variance [30]. However, recent research has shed light on a more nuanced understanding of overfitting by showing that once enough learnable parameters are present in the model, generalization performance can be recovered [10]. To illustrate this phenomenon, we can train a multilayer perceptron of varying sizes on the MNIST-1D dataset [36], which is a synthetic dataset of distorted line strokes resembling handwritten digits. Figure 2.6 shows the obtained accuracies (in terms of negative log-likelihood, cf. (2.22)) on the training and test set respectively for each configuration. Additionally, the interpolation threshold is annotated, which lies at the (minimum) amount of parameters needed to perfectly recall the training set. When below this threshold, increasing the number of parameters allows the model to achieve a lower error on the training set, at the cost of test performance. However, this cost vanishes once the number of parameters is increased gradually beyond the interpolation threshold.

Unfortunately, increasing the number of parameters comes with the drawback of additional computational burden. Furthermore, it is often desirable to manually control for the capacity of neural networks by inducing prior information. This prior, also known as *regularization* can come in various forms and is often incorporated directly into the loss function. The most prominent example is L^2 regularization and is obtained by adding a multiple of the norm $||\theta||_2^2$ to the loss function [35]. Interpreted in a Bayesian setting, L^2 regularization penalizes a model for learning parameters θ with overly high variance by placing a low variance prior on the parameter space. Finally, regularization is widely applicable for reinforcement learning problems and normative decision-making. For example, regularization allows us to control for stochasticity of a policy (cf. maximum entropy RL (2.14)) as well as making use of prior information from human behavior, which we will elaborate on further in chapter 3.

2.3.4. Convolutional Neural Networks

Incorporating *inductive biases*, which are prior assumptions made about the space of learnable functions can strongly boost a model's performance. This is because, depending on the type of data, one can exploit regularities and invariances of the input space to drastically reduce the problem dimensionality [18]. Convolutional neural networks (CNNs) do so for the domain of image recognition by sharing parameters in a way that makes the input-output relation shift-invariant [35]. Instead of learning dense weight matrices, CNNs transform input images $I \in \mathbb{R}^{n\times m}$ into a new image $S \in \mathbb{R}^{n-k+1\times m-k+1}$ through the *cross-correlation* operator

$$S(x, y) = \sum_i \sum_j I(i + x, j + y)K(i, j), \tag{2.28}$$

where $K \in \mathbb{R}^{k\times k}$ is called a *filter*. The filters K form the learnable parameters of a CNN as well as the basis for shift invariance. Filters function as object recognition modules, which produce high activation

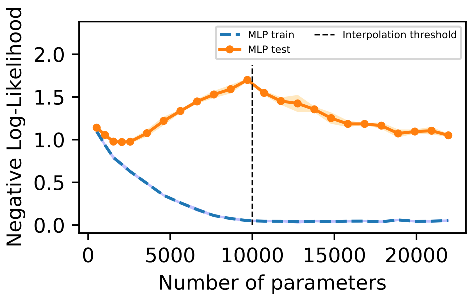


Figure 2.6: Double descent curve on MNIST-1D for a multilayer perceptron of varying size. When beyond the interpolation threshold (black), the test accuracy starts to decrease again (orange) while the training performance follows a monotonically decreasing function of the number of parameters (blue).

maps when the corresponding input image exhibits similar patterns at any location. Furthermore, the cross-correlation operator can be reformulated as sparse matrix multiplications and as a consequence, CNNs form a subclass of MLPs.

Typically, a convolutional layer consists of a set of filters $\{K_1, \dots, K_n\}$ that results in an image output with multiple channels. To process images with many channels, we can furthermore generalize the cross-correlation operator, by learning three-dimensional filters and performing the computation (2.28) analogously over all channels. While the study of deep CNNs is beyond the scope of this chapter, we note the strength of their inductive bias by reporting test accuracies in table 2.1. We train two MLP architectures with two hidden layers of sizes 256 and 5000 respectively as well as two CNN architectures that contain three convolutional layers with 32 and 256 filters respectively, followed by a linear output layer for outputting a class between 0 and 9. The results clearly indicate that CNNs outperform MLPs, both in the number of parameters as well as in the overall test accuracy, with the smaller CNN architecture needing up to four magnitudes less parameters while achieving an almost 25% higher accuracy.

Architecture	MLP-256	MLP-5000	CNN-32	CNN-256
Test Accuracy	65.4	70.8	94.2	**96.2**
# Parameters	78858	5.2e7	**5210**	408074

Table 2.1: Performance comparison of CNNs and MLPs on MNIST-1D. MLPs are unable to match the performance of CNNs even when largely exceeding the number of learnable parameters.

2.3.5. Generative Adversarial Networks

So far, we have only considered networks that model a conditional distribution of observable variables. However, we are often interested in generating completely new samples of a training data distribution without providing any explicit observable input. Inspired by methods from game theory, generative adversarial networks (GAN) tackle the problem of generating synthetic samples for high dimensional probability distributions by letting the generating network G compete against an adversarial counterpart D [34]. To do so, network parameters θ_G and θ_D are trained jointly to minimize the GAN loss function

$$\arg\min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z))], \tag{2.29}$$

where p_z is a latent distribution from which the generator G generates new samples. The first term of the loss function trains a discriminator to maximize its confidence when drawing real samples from the data distribution p_{data}, while the second term ensures that the discriminator correctly distinguishes the generated samples $G(z)$ from real ones. Furthermore, minimization over G encourages the generator network to fool the discriminator by producing samples that are more similar to $x \sim p_{\text{data}}$. As it turns out, this game theoretic training scheme can be generalized to distinguishing human demonstrations from synthetic behavior in the context of reinforcement learning, making it especially useful when modelling human behavior with deep neural networks. We will omit the details for now and return to a specific application in chapter 3.

2.4. Policy Gradient Methods

In the previous sections we have outlined how we can achieve generalization through optimizing deep neural networks with gradient-based optimization. As it turns out, a similar concept can be directly applied to the problem of learning optimal policies in MDPs. *Policy gradient methods* form a class of reinforcement learning algorithms which learn a parametrized policy π_θ by sampling the gradient of its value function. In this section, we will illustrate the theoretical framework of policy gradient methods in 2.4.1 as well as discuss *proximal policy optimization* in 2.4.2, a state-of-the-art deep reinforcement learning algorithm for solving high dimensional problems.

2.4.1. REINFORCE

When aiming to obtain gradient estimates of the cumulative expected rewards $J(\pi_\theta)$, one faces the problem of backpropagating through the distribution of states encountered by π_θ. When no assumptions are made about the transition dynamics of the MDP, such a derivative becomes intractable to compute.

Fortunately, the *policy gradient theorem* derives an expression of $\nabla J(\pi_\theta)$ that only depends on the gradients of the policy π_θ with respect to its actions.

Theorem 2.4.1 (Policy Gradient Theorem [83]). *For any MDP, the gradient of the performance metric $J(\pi_\theta)$ is given by*

$$\nabla J(\pi_\theta) \propto \sum_{s\in\mathcal{S}} \mu(s) \sum_{a\in\mathcal{A}} \nabla\pi_\theta(a|s) Q_{\pi_\theta}(s,a), \tag{2.30}$$

where $\mu(s)$ is the discounted state visitation distribution under π_θ.

$$\mu(s) := \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(S_t = s|S_0, \pi_\theta) \tag{2.31}$$

The reinforce trick rewrites $\nabla\pi_\theta(a|s) = \pi_\theta(a|s)\nabla\log\pi_\theta(a|s)$, allowing us to express the gradient in terms of obtained returns G_t:

$$\begin{aligned}\nabla J(\pi_\theta) &\propto \mathbb{E}_\pi\left[Q_{\pi_\theta}(S_t, A_t)\nabla\log\pi_\theta(A_t|S_t)\right]\\ &= \mathbb{E}_\pi\left[G_t\nabla\log\pi_\theta(A_t|S_t)\right].\end{aligned} \tag{2.32}$$

Furthermore, we can replace G_t by an advantage function $A_\pi(S_t, A_t) = Q_\pi(S_t, A_t) - V_\pi(S_t)$ to reduce the variance of gradient estimates without altering the expression [76]. In total, the REINFORCE algorithm turns (2.32) into a gradient-based update rule for θ, where G_t is estimated in a Monte Carlo fashion as discussed in section 2.1.3. When employing advantage functions, we furthermore learn V_{π_θ} separately by performing gradient descent on the squared loss (2.11) using the methods introduced in section 2.1.4.

2.4.2. Proximal Policy Optimization

REINFORCE provides an intuitive algorithm for directly approximating locally optimal policies using stochastic gradient descent. Nonetheless, when dealing with complex state spaces and long horizon tasks, the variance of the advantage estimates A_π can lead to unstable training. The reason for this is that when performing noisy updates, one is at risk of undoing a large part of learning progress by updating θ into the wrong direction. To tackle this, *proximal policy optimization* (PPO) is an easy to implement state-of-the-art method that limits how far one allows the policy to update at each step [77]. Formally, the update rule of PPO is given by

$$\theta_{k+1} = \arg\max_\theta \mathbb{E}_{\pi_{\theta_k}}[\mathcal{L}(\theta_k, \theta)], \tag{2.33}$$

where the performance criterion $\mathcal{L}(\theta_k, \theta)$ is given by

$$\mathcal{L}(\theta_k, \theta) = \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A_{\pi_{\theta_k}}(s_t, a_t), \mathrm{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right) A_{\pi_{\theta_k}}(s_t, a_t)\right) \tag{2.34}$$

Algorithm 2: Proximal Policy Optimization

```
Result: Trained policy $\pi_{\theta^*}(a|s)$ and corresponding value function $V_{\phi^*}(s)$
Initialize: Policy $\pi_{\theta_0}$, value function $V_{\phi_0}$
for $k = 0, 1, \dots$ do
    Sample trajectories $\mathcal{D} = \{\tau_i\}_{i=1}^m$ from $p(\tau|\pi_{\theta_k})$ through interaction with the environment.
    Estimate returns $\hat{G}_t$, advantage values $\hat{A}_{\pi_{\theta_k}}(s_t, a_t)$ from $\mathcal{D}$ using $V_{\phi_k}$ as a baseline.
    Update $\theta$ by stochastic gradient ascent (e.g. Adam) on the objective $\mathcal{L}(\theta_k, \theta)$:
        $\theta_{k+1} =$
      $\arg\max_\theta \sum_{\tau\in\mathcal{D}} \sum_t \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \hat{A}_{\pi_{\theta_k}}(s_t, a_t), \mathrm{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right) \hat{A}_{\pi_{\theta_k}}(s_t, a_t)\right)$
    Update $\phi$ using stochastic gradient descent on the mean squared loss:
        $\phi_{k+1} = \arg\min_\phi \sum_{\tau\in\mathcal{D}} \sum_t (V_\phi(s_t) - \hat{G}_t)^2$
end
```

and $\text{clip}(x, a, b)$ denotes a clipping mechanism that bounds the value x to the interval $[a, b]$. Algorithm 2 shows a pseudocode implementation of PPO using this objective.

The term $(\pi_\theta(a_t|s_t)/\pi_{\theta_k}(a_t|s_t))\hat{A}_{\pi_{\theta_k}}(s_t, a_t)$ measures how a new policy π_θ performs relative to the current policy π_{θ_k}. Intuitively, this is because the estimated advantages $\hat{A}_{\pi_{\theta_k}}$ are sampled from π_{θ_k} in an on-policy fashion. That way, for each $\hat{A}_{\pi_{\theta_k}}$ the corresponding action $\pi_{\theta_k}(a_t|s_t)$ that partially led to it can be readily compared with the probability that another policy π_θ would have taken the same action. This is directly comparable to the original derivation of the policy gradient (2.32). Furthermore, taking the minimum over the likelihood term and its clipped version aims to avoid updating the policy too much at once. For example, when $\hat{A}_{\pi_{\theta_k}}$ is positive, optimizing the policy gradient objective yields an increase in the probability $\pi_\theta(a_t|s_t)$. However, by clipping the ratio we ensure that the maximum amount of increase in advantage is $(1+\epsilon)\hat{A}_{\pi_{\theta_k}}$ when $\pi_\theta(a_t|s_t)$ becomes too large.

3

Learning Implicit Norms

A key component of human behavior not typically present in current AI systems is the ability to adhere to a multitude of social norms. Ideally, however, any aligned system should have some capability of estimating the normativity of its actions in order to avoid causing ethically irresponsible outcomes. In this chapter we aim to tackle this problem in the context of deep RL agents by combining methods of learning from demonstrations and multi-objective decision-making. We will start by providing a formal discussion on the issue of incorporating demonstrated normative behavior alongside a primary goal by briefly reviewing state-of-the-art methods and comparing the constrained and multi-objective RL frameworks in section 3.1. We note, however, that sections 3.1.1 and 3.1.2 merely serve as a motivation to our later contributions from a theoretical point of view and can be skipped if desired. Subsequently, in section 3.2 we will describe how adversarial inverse reinforcement learning can be used for learning a normative prior from a single expert and combined with a primary goal. Finally, in section 3.3 we illustrate our approach in a small grid world domain and discuss the key advantages and disadvantages of the used method.

3.1. Aligning RL Agents

As discussed in section 2.2, RL agents rely heavily on the correct specification of a reward function, which may or may not be feasible depending on the type of problem one is trying to solve. Furthermore, one has to make a trade-off between rewards that facilitate quick learning and rewards that encode the goal narrowly enough. A sparse reward signal which only fires once its corresponding task has been achieved might correctly encode the desired goal, but when rewards become too sparse the amount of needed exploration grows exponentially, deeming any traditional RL agent to fail at the problem. State-of-the-art solutions to tackle extreme reward sparsity thus typically require task-specific human feedback in order to make policy learning feasible [29]. This is mostly a practical problem, since theoretically one can *shape* the reward function, which means applying a transformation to speed up training without changing the set of optimal policies [62].

Unfortunately, while a solution to the exploration problem of deep RL would certainly provide significant progress towards the goal of alignment by enabling the specification of more sparse reward signals, the fundamental reward specification problem would persist. Additionally, deep RL agents are prone to exploiting the reward function in unintended ways, leading to unanticipated behavior and negative side effects [6]. It therefore seems inevitable to explore different MDP-based frameworks that extend the notion of a single all determining reward function to make the specification of desired behavior more tractable. To that end, multi-objective and constrained optimization problems provide two conceptually similar, yet theoretically different perspectives on reward augmentation.

3.1.1. Differences between Extended Frameworks

Multi-objective and constrained reinforcement learning both share the idea of extending reward functions with additional desiderata. However, it is not clear a-priori which solutions can or can not be obtained from either of the two approaches. To formally analyze this discrepancy, we will outline a simple proof that shows when one can obtain solutions for CMDPs by solving a linear MOMDP. For the ease of illustration, we will restrict ourselves to the two-dimensional case.

We will now consider a fixed MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \mu_0 \rangle$ that we extend by a single constraint function $c \geq 0$ and a hyperparameter $d \in \mathbb{R}$. Let this CMDP be called $\mathcal{M}_C$. At the same time, consider the MOMDP $\mathcal{M}_{MO}$ obtained by extending $\mathcal{M}$ by a second reward function $r_2 = -c$. We would now like to investigate, when an optimal solution π^* in $\mathcal{M}_C$ can also be obtained in $\mathcal{M}_{MO}$ given a specific linear preference $\boldsymbol{\omega}$. Let $\Pi_C^*(d)$ denote the set of optimal policies for $\mathcal{M}_C$ (depending on the hyperparameter d). Formally, we would like to find out, for which values of d it is the case that $\Pi_C^*(d)$ has a non-empty intersection with policies on the convex coverage set $\mathcal{F}^*$ of $\mathcal{M}_{MO}$? The reason we are interested in this question is because assuming global optimization, by our definition a linear MOMDP algorithm will only search for policies that lie on the CCS. This means that if for given d the optimal CMDP solution does not lie on the CCS then we will not be able to recover it when approximating the CCS for the MOMDP problem. We will start by proving a simple Lemma.

Lemma 3.1.1. *For each constraint parameter d, if there exists an optimal policy then there exists an optimal policy that lies on the Pareto boundary of the corresponding MOMDP : $\Pi_C^*(d) \bigcap \mathcal{F} \neq \emptyset$.*

Proof: Let $d \in \mathbb{R}$ and $\pi^* \in \Pi_C^*(d)$ be arbitrary. If π^* lies on the Pareto boundary, we are done. Otherwise, assume π^* does not lie in $\mathcal{F}$. By definition, there must exist a policy π', such that $J_{r_1}(\pi') \geq J_{r_1}(\pi^*)$ and $J_{r_2}(\pi') \geq J_{r_2}(\pi^*)$ with strict inequality holding for at least one of the two inequalities. In case the first inequality strictly improves, it must be that $J_{r_2}(\pi') \leq -d$, otherwise π' would be another feasible policy that improves upon π^* in the CMDP. However, this would contradict $J_{r_2}(\pi') \geq J_{r_2}(\pi^*) > -d$. Therefore, we can assume that it must be the second inequality that strictly improves, i.e. $J_{r_2}(\pi') > J_{r_2}(\pi^*) > -d$. This implies that π' is also feasible in the CMDP and must yield the same return, namely $J_{r_1}(\pi') = J_{r_1}(\pi^*)$, which shows that $\pi' \in \Pi_C^*(d)$. Since π' was an arbitrary policy that strictly improves upon a policy in $\Pi_C^*(d)$, we can thus repeat the same argument for π' until we obtain a policy $\pi_{\mathcal{F}} \in \Pi_C^*(d) \cap \mathcal{F}$ that can no longer be further improved upon (assuming bounded rewards). ■

Lemma 3.1.1 guarantees that we can always find an optimal solution of the CMDP that also lies on the Pareto boundary of the corresponding MOMDP. Intuitively, this makes sense: From all the feasible optimal solutions of the CMDP we take the optimal solution with the least constraint violations. While this lemma shows us that optimal CMDP solutions are somewhat efficient with respect to the MOMDP, it does not state anything about the CCS. However, what immediately follows from the lemma is that the optimal CMDP solution can be recovered in the MOMDP setting if the Pareto boundary is equal to the CCS.

Theorem 3.1.2. *If $\mathcal{F} = \mathcal{F}^*$ then for each constraint parameter that allows feasible policies, there exists a preference that recovers the optimal CMDP solution, that is: $\forall d \in \mathbb{R}$ s.t. $\Pi_C^*(d) \neq \emptyset\ \exists \boldsymbol{\omega} \in \Omega, \pi^* \in \Pi_C^*(d) : \pi^* = arg\max_\pi \boldsymbol{\omega}^T J_{\boldsymbol{r}}(\pi)$.*

Proof: The proof follows from the fact that $\mathcal{F} = \mathcal{F}^*$ combined with lemma 3.1.1 implies that there exists $\pi^* \in \mathcal{F}^* \cap \Pi_C^*(d)$. By definition of $\mathcal{F}^*$ the existence of $\boldsymbol{\omega} \in \Omega$ with $\pi^* = \arg\max_\pi \boldsymbol{\omega}^T J_{\boldsymbol{r}}(\pi)$ follows immediately. ■

This theorem shows us that in some cases, the CMDP solution will automatically be optimized for in CCS approximation algorithms with linear preferences. In this sense, we could consider the CMDP problem to be a subset of the MOMDP problem, where hyperparameter tuning of d is equivalent to adapting to a different preference vector $\boldsymbol{\omega}$. Unfortunately, however, the converse of the theorem does not hold, which means that in the non-convex case the CMDP solution will not lie on the CCS and thus cannot be recovered by optimizing for a certain linear preference between constraint and rewards. To illustrate this, we refer to figure 3.1.

3.1.2. Challenges when Learning Constraints

The discussion in the previous section showed that depending on the shape of the Pareto front, one may find CMDP solutions with linear multi-objective algorithms. A main drawback of this is that, in practice, we cannot expect to know the shape of the Pareto front beforehand, which poses the question what framework is suitable for the broader goal of teaching an RL agent to align with a specified set of norms. As opposed to theorem 3.1.2, when the Pareto front does exhibit overly concave parts a linear multi-objective approach would not suffice to recover any behavior that can be encoded in a CMDP.

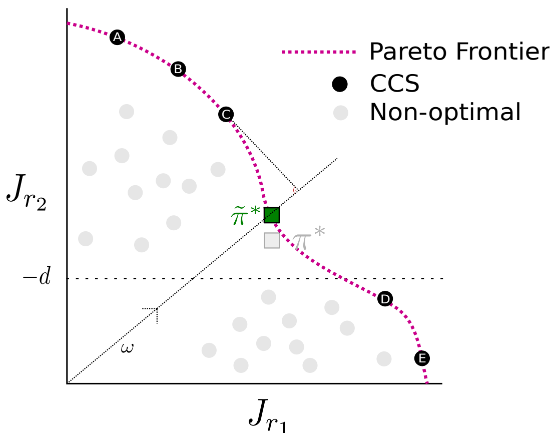


Figure 3.1: When the CCS is not equal to the Pareto boundary, the CMDP solution will generally not be recoverable through linear preferences. In this example π^* is an optimal CMDP solution which does not lie on the Pareto boundary $\mathcal{F}$. However, as lemma 3.1.1 suggests, we can find an optimal $\tilde{\pi}^*$ that does lie in $\mathcal{F}$, which yields higher r_2 (lower constraint cost) but the same original reward r_1. Since in this case the CCS consists of A-E but does not include $\tilde{\pi}^*$, the scalarized reward $\boldsymbol{\omega}^T\mathbf{r}$ does not get maximized by $\tilde{\pi}^*$, but by a point on the CCS (in this case C).

On the other hand, in a CMDP there is an inherent ordering over objectives. Namely, the constraints necessarily have to be jointly satisfied before optimization of the reward function can be carried through.

Even though normative behavior might seem more easily represented in the form of constraints, this implicit ordering over objectives becomes highly problematic when we aim to learn the constraint function instead of formally specifying it. To see why, assume that we want an agent to optimize for some goal specified by a reward function r while adhering to a set of norms given by a constraint function $\mathbf{c}_H = (c_1, \ldots, c_k)$ of some human H. In case $\mathbf{c}_H$ can be formally specified, then we can hope to find a corresponding constraint parameter $\mathbf{d}_H \in \mathbb{R}^k$ such that an optimal solution π_H^* from optimizing the CMDP

$$\max_{\pi} \quad J(\pi) \tag{3.1}$$
$$\text{s.t.} \quad J_{\mathbf{c}_H}(\pi) \le \mathbf{d}_H \tag{3.2}$$

leads to normative behavior. When this is not the case, we have to resort to learning a function $\hat{\mathbf{c}}_H : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^k$ first. Consequently, a new constraint parameter $\hat{\mathbf{d}}_H \in \mathbb{R}^k$ would have to be found, such that an optimal policy π^* in the CMDP $\langle \mathcal{S}, \mathcal{A}, p, r, \mu_0, \hat{\mathbf{c}}_H, \hat{\mathbf{d}}_H \rangle$ closely matches the policy π_H^*. Now, assume we have learned an unbiased, but noisy estimate $\hat{\mathbf{c}}_H(s,a) = (c_1(s,a)+\epsilon_1, \ldots, c_k(s,a)+\epsilon_k)$ where the $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ are i.i.d. normal random variables modelling incurred measurement errors from interacting with H. Then, rewriting the constraint (3.2) with respect to the learned function, we obtain a new constraint

$$\begin{aligned} J_{\hat{\mathbf{c}}_H}(\pi) &= \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t \hat{\mathbf{c}}_H(s_t, a_t)\right] \\ &= \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{c}_H(s_t, a_t)\right] + \sum_{t=0}^{\infty} \gamma^t \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}) \\ &= J_{\mathbf{c}_H}(\pi) + \sum_{t=0}^{\infty} \mathcal{N}\left(\mathbf{0}, (\gamma^t\sigma)^2\mathbf{I}\right) \le \hat{\mathbf{d}}_H, \end{aligned} \tag{3.3}$$

where $\mathbf{I} \in \mathbb{R}^{k\times k}$ is the identity matrix. Now, applying Levy's continuity theorem [93] and the fact that $\gamma < 1$ we obtain that the left hand side of the inequality converges in distribution

$$J_{\mathbf{c}_H}(\pi) + \sum_{t=0}^{\infty} \mathcal{N}\left(0, (\gamma^t\sigma)^2\right) \xrightarrow{d} \mathcal{N}\left(J_{\mathbf{c}_H}(\pi), \frac{\sigma^2}{1-\gamma}\mathbf{I}\right). \tag{3.4}$$

Although unbiased, this return is noisy proportional to the errors σ^2 made when learning the constraint function $\hat{\mathbf{c}}_H$. If our goal now is to find a hyperparameter $\hat{\mathbf{d}}_H$ such that the desired policy π_H^* is attained, we will have to deal with this noise while optimizing for a constrained solution.

We illustrate the problem of only having access to noisy samples of the true constraint function $\mathbf{c}_H$ in an example two-dimensional CMDP. Figure 3.2 provides a complete description of the CMDP including its payoffs. The agent starts in state S and can execute one of two actions, leading it into the respective states S_1 and S_2. Each of these terminal states returns a two-dimensional array, representing the reward function r and the constraint function c_H. Clearly, state S_2 yields the highest reward, but comes at a constraint cost of 1, whereas S_1 incurs no constraints.

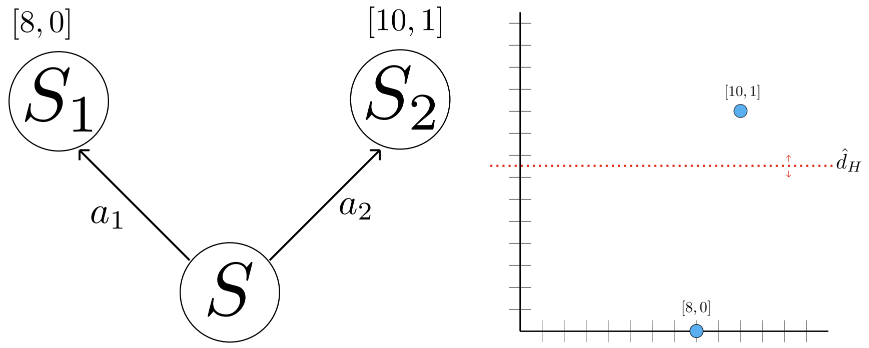


Figure 3.2: *Left*: Example CMDP with two terminal states and their respective payoffs $[r(s), c(s)]$. S_2 provides the highest rewards at the cost of being more unsafe, whereas S_1 yields no constraint costs. *Right*: The threshold $\hat{d}_H$ needs to be tuned such that desired behavior is incurred from an optimal policy. An optimal CMDP solution will always choose the point below $\hat{d}_H$ that lies furthest to the right.

Now assume that we do not observe $\mathbf{c}_H$, but instead only have a noisy estimate and let π_1 and π_2 denote the policies taking actions a_1 and a_2 respectively. Furthermore, assume that $\mathbf{d}_H = 0.5$, meaning that H deems S_2 to be infeasible. Under this constraint regime, we immediately see that the optimal solution π_H^* to (3.2) is equal to π_1. We would now like to investigate what our chances are of choosing $\hat{\mathbf{d}}_H$, such that π^* coincides with $\pi_H^* = \pi_1$. Following the derivation in (3.4), we obtain that $J_{\hat{\mathbf{c}}_H}(\pi) \overset{d}{=} \mathcal{N}\left(J_{\mathbf{c}_H}(\pi), \sigma^2\right)$ in the undiscounted regime ($\gamma = 1$) due to the finite time horizon of $T = 1$. Plugging in π_1 and π_2 gives us that taking action a_1 yields an observed constraint cost of $Z = \mathcal{N}(0, \sigma^2)$ and a_2 returns a cost of $Y = \mathcal{N}(1, \sigma^2)$. We can now obtain a random variable $X(\hat{d}_H)$ that determines the choice of the optimal policy π^*:

$$X(\hat{d}_H) := \begin{cases} \pi_2 & \text{if } Y \leq \hat{d}_H \\ \pi_1 & \text{if } Y \geq \hat{d}_H \wedge Z \leq \hat{d}_H \\ \emptyset & \text{else} \end{cases} \tag{3.5}$$

To see why this holds, consider figure 3.2. If $Y \leq \hat{d}_H$ then the observed constraint cost of visiting S_2 were low enough to make S_2 a feasible state, hence optimal due to its maximal reward. Otherwise, S_2 appears infeasible which leaves S_1 as the other optimal feasible option. However, we can only choose S_1 in case its constraint $Z \leq \hat{d}_H$ is satisfied and otherwise there is no feasible policy. From this definition, we can explicitly calculate the probability that $\pi^* = \pi_H^*$, by

$$\begin{aligned} \mathbb{P}\left(X(\hat{d}_H) = \pi_1\right) &= \mathbb{P}\left(Y \geq \hat{d}_H \wedge Z \leq \hat{d}_H\right) \\ &= \mathbb{P}\left(Y \geq \hat{d}_H\right) \mathbb{P}\left(Z \leq \hat{d}_H\right) \\ &= \mathbb{P}\left(\mathcal{N}(0,1) \geq \frac{\hat{d}_H - 1}{\sigma}\right) \mathbb{P}\left(\mathcal{N}(0,1) \leq \frac{\hat{d}_H}{\sigma}\right) \\ &= \left(1 - \phi\left(\frac{\hat{d}_H - 1}{\sigma}\right)\right) \phi\left(\frac{\hat{d}_H}{\sigma}\right), \end{aligned} \tag{3.6}$$

where we made use of independence between Y and Z and $\phi(x) := \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp \frac{-t^2}{2} dt$ denotes the cumulative distribution function of the standard normal distribution. As a function of $\hat{d}_H$, we can plot the right hand side of (3.6) for various values of σ, see figure 3.3. We see that regardless of σ the probability peaks around $\hat{d}_H = 0.5$ with the peak value being monotonically decreasing in σ. Intuitively, this makes sense: If we aim to find π_1 then choosing $\hat{d}_H$ too high will force us to optimize for π_2 instead. On the other hand, $\hat{d}_H$ cannot be chosen too low since otherwise the probability of observing any feasible policy tends to zero. Hence, due to the assumed homoscedasticity of the measurement errors ϵ_i, the optimal choice reduces to the intermediate value of 0.5.

Unfortunately, since we do not have access to c_H nor d_H in the unknown constraints scenario, this example shows that the chances of choosing $\hat{d}_H$ such that the desired behavior is optimized for are slim. Due to the nature of constrained optimization, the constraint parameter $\hat{d}_H$ has to be specified a-priori which, if only having fuzzily defined constraints as in the example above, can lead to precluding desired policies from the optimization all together.

3.1.3. Alignment as a Multi-Objective Problem

In the derivation above we assumed that $\hat{c}_H$ was learned from some type of human interaction, while $\hat{d}_H$ was assumed to be a hyperparameter. A different, possibly more efficient choice would be to jointly learn $\hat{c}_H$ and $\hat{d}_H$, such that solving the corresponding CMDP yields the desired policy π^*_H. To our knowledge, this problem has not yet directly been tackled for general constraint functions. Instead, previous constraint learning research has mostly focused on constraints of a predetermined geometric [65] or binary [22] form. However, we expect this to be due to the problems outlined above.

When dealing with multiple soft and possibly conflicting criteria, the framework of multi-objective optimization provides a better alternative. Not only does multi-objective optimization have many potential benefits when aiming to build human-aligned AI [88], but it also fits well into the unknown human preferences scenario [69]. Thus, we frame the problem of teaching a deep RL agent normative behavior into the MOMDP framework. In the following we will consider two different scenarios:

1. *Steering a traditional deep RL agent towards normative behavior*: In this case, we assume that there exists an MDP $\langle \mathcal{S}, \mathcal{A}, p, r_P, \mu_0 \rangle$ with primary goal r_P, which can be solved by typical methods of (deep) RL. While r_P is assumed to properly encode the desired goal, the agent is not guaranteed to behave ethically. For this reason, we assume that there exists a normative reward r_N, such that maximizing $\mathbb{E}_\pi[\sum_t r_N(s_t, a_t)]$ leads to normative behavior. Note that this assumption is needed to make the RL problem feasible, and coincides with the fundamental reward hypothesis of RL [82]. Furthermore, since r_N depends on a set of human values that are not easily expressed formally, we learn r_N from human inputs. Finally, we form a MOMDP $\langle \mathcal{S}, \mathcal{A}, p, \mathbf{r}, \mu_0, \Omega, f_\Omega \rangle$ with $\mathbf{r} = [r_P, r_N]$ and study how the combination of normativity and primary goals through multi-objective optimization. This scenario will be mostly covered in chapters 3 and 4.

2. *Controlling for norm diversity in deep RL agents*: This scenario extends the previous scenario

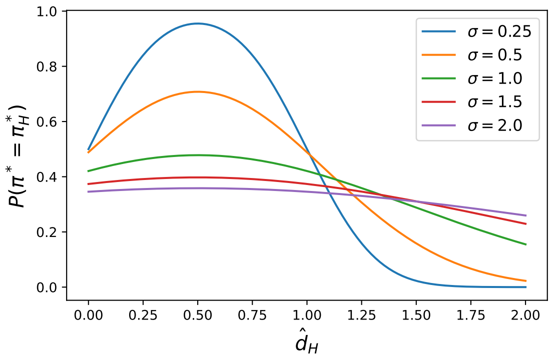


Figure 3.3: Probability of optimizing for the true optimal policy π^*_H in the learned constraints scenario as a function of constraint hyperparameter $\hat{d}_H$.

by assuming that r_N in itself cannot be easily learned due to an inherent diversity and value disagreement between different humans. As such, we extend the normative reward function to a vector of rewards $\mathbf{r}_N = [r_N^1, \ldots, r_N^k]$. Analogously, we form a MOMDP with $\mathbf{r} = \mathbf{r}_N$ to trade off different norms at runtime. We will study this scenario more explicitly in chapter 5.

3.2. Methods

Having settled on an appropriate framework for the study of incorporating normative behavior into RL agents, we have yet to address the way in which rewards are learned and subsequently optimized for. In this section, we will discuss methods needed to tackle the first scenario with a single learned normative reward r_N. Firstly, we will discuss the method of *adversarial inverse reinforcement learning* (AIRL) for learning reward functions from human demonstrations in section 3.2.1. Secondly, section 3.2.2 will introduce a small scale benchmark environment for studying how desired trade-offs between demonstrated norms and primary goals can be achieved. Finally, we will describe implementation details and the experimental setup in section 3.2.3.

3.2.1. Adversarial Inverse Reinforcement Learning

Learning from human inputs has become a popular area of research in the field of RL due to the limits of manual reward function design [23, 86, 91]. The most straight forward way of translating what a human intends is arguably through demonstrating the desired behavior. Imitation learning avoids the reward specification problem by learning a policy that approximately matches the state distribution of demonstrated trajectories [45, 68]. However, as the name suggests, these methods are limited in generalization capabilities. Firstly, human input can be expensive and sometimes simply infeasible due to the desired task not being attainable by most humans. Secondly, when transferring to a related task that requires the same solution, but was not present in the demonstration dataset, an imitation learner might fail to solve the new task because it only learned an imitation policy for a smaller set of tasks.

To circumvent these drawbacks, inverse reinforcement learning (IRL) instead aims to learn a reward function alongside an (optional) imitation policy [1, 61]. As such, the fundamental challenge of IRL is that without any assumptions about the structure of learned rewards it is an ill posed problem. Any set of demonstrations allows a multitude of reward functions under which the demonstrated behavior becomes optimal, including the trivial example $r = 0$. To overcome such challenges, various IRL frameworks have been developed, with maximum entropy IRL [101] having become the state-of-the-art formulation. Maximum entropy IRL builds on the framework of maximum entropy RL (cf. section 2.2.1) and imposes an ordering on the desirability of learned reward functions by preferring rewards that yield optimal policies with higher entropy. Enforcing stochasticity alongside the imitation objective through the learned reward helps in achieving improved generalization performance while remaining tractable for sample-based optimization procedures.

Formally, maximum entropy IRL learns a reward function r_θ by placing a probabilistic model over observed trajectories:

$$p_\theta(\tau) \propto \mu_0(s_0) \prod_{t=0}^{T} p(s_{t+1}|s_t, a_t) \exp(r_\theta(s_t, a_t)) = \overline{p}_\theta(\tau). \tag{3.7}$$

We can now learn θ through maximum likelihood estimation (cf. section 2.3.2)

$$\arg\max_\theta \mathbb{E}_{\tau\sim\mathcal{D}_E}[\log p_\theta(\tau)] = \arg\max_\theta \mathbb{E}_{\tau\sim\mathcal{D}_E}\left[\sum_{t=0}^{T} r_\theta(s_t, a_t)\right] - \log Z_\theta, \tag{3.8}$$

where $Z_\theta = \int \overline{p}_\theta(\tau)d\tau$ is the partition function and $\mathcal{D}_E = \{\tau_i\}_{i=1}^N$ is a data set of expert demonstrations. An optimal solution θ^* then yields the reward function r_{θ^*} under which a maximum entropy RL agent most closely resembles the expert demonstrations $\mathcal{D}_E$.

Being an integral over possible trajectories, it becomes intractable to calculate Z_θ directly once the state space is sufficiently large. However, *adversarial inverse reinforcement learning* (AIRL) [31] proposes the use of generative adversarial networks (cf section 2.3.5) to efficiently solve the optimization problem (3.8) through gradient-based optimization instead. To do so, we jointly train a discriminator of the form

Algorithm 3: AIRL

```
Input: Expert trajectory data set $\mathcal{D}_E = \{\tau_i\}_{i=1}^N$.
Initialize: Reward network $f_\theta$, policy $\pi_\phi$.
repeat
    Sample trajectories $\mathcal{D}$ from $p(\tau|\pi_\phi)$ through interaction with the environment.
    Update $\theta$ via stochastic gradient descent on the loss
        $-\mathbb{E}_{(s,a,s')\sim\mathcal{D}_E}[\log D_\theta(s,a,s')] - \mathbb{E}_{(s,a,s')\sim\mathcal{D}}[\log(1 - D_\theta(s,a,s'))]$.
    Update $\phi$ using PPO to maximize
        $\mathbb{E}_{\pi_\phi}\left[\sum_{t=0}^{T-1} \log D_\theta(s_t,a_t,s_{t+1}) + \log(1 - D_\theta(s_t,a_t,s_{t+1}))\right]$.
until convergence of $\theta$, $\phi$;
Result: Estimated expert policy $\pi_{\phi^*}(a|s)$, expert reward function $f_{\theta^*}(s,a,s')$.
```

$$D_\theta(s,a,s') := \frac{\exp(f_\theta(s,a,s'))}{\exp(f_\theta(s,a,s')) + \pi_\phi(a|s)} \tag{3.9}$$

and a policy π_ϕ. D_θ outputs its confidence that the transition (s,a,s') came from an expert rather than from the policy π_ϕ and is trained through the loss function

$$\mathcal{L}(\theta) := \sum_{t=0}^{T-1} -\mathbb{E}_{\mathcal{D}_E}[\log D_\theta(s_t,a_t,s_{t+1})] - \mathbb{E}_{\pi_\phi}[\log(1 - D_\theta(s_t,a_t,s_{t+1}))], \tag{3.10}$$

which resembles the binary cross entropy loss (see 2.26). One can show that by taking the gradient of $\mathcal{L}(\theta)$ we are indeed optimizing the maximum entropy IRL objective [31]. To ensure convergence, the policy π_ϕ is trained to maximize the maximum entropy RL objective

$$\arg\min_\pi D_{KL}(p(\tau|\pi)||p_\theta(\tau)) = \arg\max_\pi \mathbb{E}\left[\sum_{t=0}^{T-1} r_\theta(s_t,a_t,s_{t+1}) - \log\pi(a_t|s_t)\right], \tag{3.11}$$

where $r_\theta(s,a,s') := \log D_\theta(s,a,s') + \log(1 - D_\theta(s,a,s'))$.

An appealing property of the specific choice of discriminator form (3.9) is that by design the reward function r_θ coincides with f_θ, which itself resembles an advantage function. Furthermore, we can easily transfer f_θ to new tasks as well as make it only depend on the state if needed. As we will show in section 3.3, this provides us with an effective way for learning an underlying parametric reward function from relatively few expert demonstrations of normative behavior. See algorithm 3 for a concise description of the procedure.

3.2.2. Multi-Objective Optimization

Due to its generality and little assumptions about the structure of the state space, AIRL can in theory recover a wide variety of demonstrated behaviors. Nonetheless, a major drawback of learning a reward function is that, a-priori, we do not know the scale of the learned reward. When rewards are manually engineered, it is often possible to evaluate an agent based on the numeric value of the obtained return alone. Learned reward functions, however, are only defined in terms of their relative values and are therefore impossible to interpret. In case of a single reward function r, this does not pose any additional problems. Namely, any reward maximizing agent will simply optimize for the policy π^* that yields the maximum expected cumulative rewards. However, having a vector of rewards $\mathbf{r}$ is not completely agnostic to such changes anymore. Scaling the individual components of $\mathbf{r}$ by different constants will inevitably shift the respective priorities. On the other hand, the Pareto front is reward scale agnostic, which is why multi-objective optimization is crucial for combining learned reward signals.

To illustrate this, consider an example grid world environment in figure 3.4. A firefighter agent is trained to navigate to the fire extinguisher in the room when a fire breaks out. Although the agent might not always find itself at the same position at the point of a fire outbreak, the extinguisher has a predetermined position in the bottom right corner of the room. This goal can be easily formalized in a reward function and we do so, by providing the agent a reward of +0.1 for each time step it stands on the corresponding fire extinguisher tile. However, we additionally expect the agent to exhibit normative behavior, which we assume to be not easily expressed in a formal reward function. To model a situation in which normativity is crucial, we also assume that there are "lost workers" placed at random positions on the 6×6 grid that need to be picked up in order to escape. Finally, the set of possible actions is $\mathcal{A} = \{$up, right, down, left, interact up, interact right, interact down, interact left, no-op$\}$ such that at each time step the agent is allowed to move in one of the four directions or pick up a lost worker next to it.

Let r_P denote the primary reward associated to the fire extinguishing goal and r_N denote a reward of $+1$ for each person saved. Since we assumed r_N to be inaccessible, we learn a parametrized reward f_θ that resembles r_N from demonstrations. The resulting MOMDP is now given by the reward function

$$\mathbf{r}(s, a, s') := \begin{pmatrix} r_P(s, a, s') \\ f_\theta(s, a, s') \end{pmatrix}. \tag{3.12}$$

To efficiently find trade off solutions, we then aim to optimize for the convex coverage set by restricting the set of scalarization functions $f_\lambda(\mathbf{r}(s, a, s')) = \lambda r_P(s, a, s') + (1-\lambda) f_\theta(s, a, s')$ to be linear (cf section 2.2.3). This new reward function has an intuitive interpretation in terms of the demonstration data set: Since constant scaling of the scalarized reward function does not change optimal policies, we see that under maximum entropy RL, the optimization of $f_\lambda(\mathbf{r})$ can be interpreted as a regularized objective

$$\max_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} r_P(s_t, a_t, s_{t+1}) \right] - \beta D_{KL}(p(\tau|\pi)||p_\theta(\tau)), \tag{3.13}$$

where $\beta = \frac{1-\lambda}{\lambda}$ is a regularization hyperparameter.

In this chapter, we analyze whether or not it is possible to optimize for normative policies with the setup outlined above. This means, that we will be treating λ as a hyperparameter of the optimization procedure. Although simple, performing a hyperparameter search over λ might not be tractable in complex problems. For this reason, we will additionally propose an interactive multi-objective optimizer in chapter 4 for finding appropriate trade-offs when hyperparameter tuning is not possible. Besides that, we employ a two step procedure, where we reoptimize for the multi-objective reward after completing the reward learning from AIRL, see algorithm 4. However, we note that it is possible to combine the IRL step with the multi-objective optimization into a single training loop to obtain a more sample efficient algorithm, as described in [66].

3.2.3. Experimental Setup

We evaluate our simple multi-objective IRL optimizer in the burning warehouse grid world from figure 3.4. Furthermore, we adapt the originally proposed AIRL network architecture [31] by learning two networks h_θ, g_θ such that $f_\theta(s, s') := g_\theta(s) + \gamma h_\theta(s') - h_\theta(s)$, where θ denotes the union over all learnable reward parameters. While in AIRL such choice is necessary for recovering state-only reward functions, we do not opt for a state-only reward and simply adopt it due to it more closely matching the original implementation.

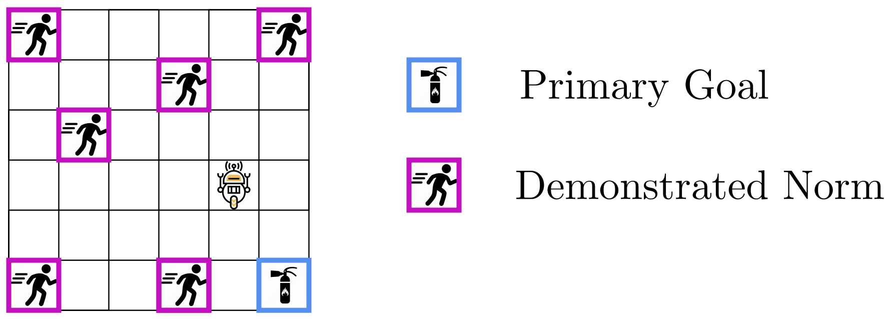


Figure 3.4: The burning warehouse environment. An agent has to navigate to its primary goal of using the fire extinguisher. However, some workers are lost and need to be rescued in time, which corresponds to normative behavior not incorporated in the primary goal.

Algorithm 4: AIRL Reward Shaping

```
Input: Expert trajectory data set $\mathcal{D}_E = \{\tau_i\}_{i=1}^N$, primary reward function $r_P$.
Initialize: Reward network $f_\theta$ by running AIRL (algorithm 3) on $\mathcal{D}_E$, list of trade off
  parameters $[\lambda_1, \dots, \lambda_n]$.
for $\lambda$ in $[\lambda_1, \dots, \lambda_n]$ do
    Train a policy $\pi_i$ using the reward $r(s,a,s') = \lambda r_P(s,a,s') + (1-\lambda) f_\theta(s,a,s')$ with PPO.
end
Result: Estimated CCS $[\pi_1, \dots, \pi_n]$.
```

Figure 3.5 illustrates the used network architecture for the discriminator network. In this example, we employ a leaky ReLU multilayer perceptron with three hidden layers of sizes $256, 512$ and 256 respectively taking a state transition (s, s') as flattened feature vectors. States in the environment are given by a binary array $I \in \{0,1\}^{C \times W \times H}$ with C channels encoding each separate entity as well as grid width W and height H (for details, we refer to the appendix B). For the PPO policy π_ϕ we employ a convolutional architecture with ReLU activations, using two shared convolutional layers with kernel size $k = 2$ and 64 as well as 256 channels respectively, followed by two separate convolutional and linear heads for the actor and critic outputs operating on 32 channels. For the detailed PPO architecture, see appendix B.1.1. To avoid learning biased reward representations through AIRL [51], we also set the environment to fixed length episodes terminating after $T = 75$ time steps. This allows for enough time to save every worker in the warehouse while leaving time for the primary goal, if played optimally.

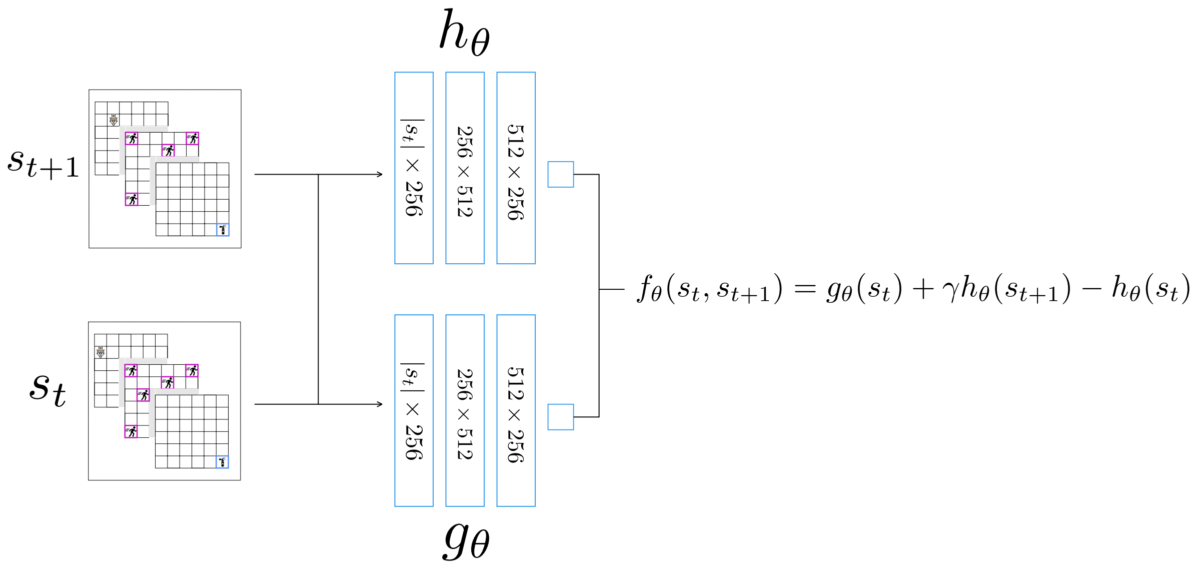


Figure 3.5: MLP Discriminator Architecture. A forward pass calculates activations of the networks g_θ and h_θ respectively and combines them into the reward prediction f_θ.

We train the AIRL reward network simultaneously with the PPO agent by using the same experience buffer of forward trajectories. While it is common to provide the AIRL update step with a larger experience buffer including past (off-policy) experience [86], we found an increased batch size for PPO to work well enough in order to ensure a rich dataset of forward experience for the reward network.[1] Regarding demonstrations, we only consider synthetically generated data. This choice is motivated by the fact that in our simple discrete environments, we expect human demonstrations to only marginally differ from demonstrations obtained through RL. Besides this, employing synthetic demonstrations allows us to more extensively test our approach with respect to larger demonstration dataset sizes and different environment configurations at later stages. To train the demonstrating agent, we employ the same architecture for the PPO network as is used in the AIRL training phase. First, the PPO agent is

[1]We opt for this option since sample efficiency is not of our primary concern and keep the amount of forward trajectories fixed for all experiments.

trained until convergence using manually engineered reward functions, which is then used to generate a demonstration dataset used in AIRL. We only keep the demonstration dataset, abandon the trained agent and reinitialize another PPO agent from scratch for the AIRL training phase. Furthermore, to model the inherent stochasticity of human demonstrations, we add an entropy regularizer as described in 2.2.1 to the PPO loss.

Although the underlying assumption of our experiments is that formal reward functions for respective normative behaviors are not easily encodable, our environments always allow for a good proxy reward function to be found that encodes the desired behavior. This enables us to more easily compare learned reward functions with their ground truth counterparts and, more importantly, to generate demonstration datasets at scale more efficiently. However, we acknowledge the possible divergence of human and synthetically generated demonstrations, which has been shown to impact training performance in continuous locomotion tasks [64]. We defer the discussion of this discrepancy for now and refer the reader to chapter 6.

3.3. Results

We start off by testing the performance of AIRL in our grid world environment for different demonstration dataset sizes. To do so, we train a PPO agent on the reward function $r(s, a) = 1$ if action a leads to saving a person present in state s and $r(s, a) = 0$ otherwise. After successful training, the policy manages to save all workers on average, but (by design) neglects the primary goal. See figure A.1 for the achieved returns within a single run. We proceed by generating four distinct datasets while keeping the trained policy fixed, but vary the sizes to include 10, 50, 500 and 2000 demonstrations respectively. We report the performance of AIRL on each of the datasets in figures 3.6 and 3.7, by separately training for three different random seeds each.

Figure 3.6 shows the amount of people saved by the imitation policy as well as the discriminator classification performance as a function of environment steps. As can be seen, the PPO agent successfully manages to imitate the saving behavior from the demonstrations even in the case of only 10 demonstrations. However, this alone does not necessarily reveal the quality of the learned reward function f_θ. When inspecting the classification accuracies of the discriminator, it becomes apparent that the reward network is heavily overfitting to the demonstration dataset. We denote the accuracy on generated trajectories from π_ϕ as *fake accuracy* as well as the accuracy on the expert dataset $\mathcal{D}_E$ as *real accuracy*. Figure 3.6 shows fake and real accuracies for the distinct training runs. We see that when the demonstration dataset is relatively small (i.e. $10 - 50$ demonstrations), the reward network is able to fully memorize states from the dataset. Thus, the discriminator is able to immediately detect any newly generated state-action pairs, leading to a very high fake accuracy despite π_ϕ behaving very similar to the expert policy π_E. Note that this effect is further amplified by the random initialization of the environment, which automatically informs the discriminator about the plausibility of a pair (s, a) coming from π_E. However, we can see that increasing the demonstration dataset size mitigates this effect, with 500 demonstrations sufficing for a fake accuracy convergence to the equilibrium point of 0.5. A different behavior can be inferred from the real accuracy, which drops accordingly as the imitation policy π_ϕ becomes increasingly proficient. Nonetheless, a small amount of overfitting persists, which allows the discriminator to classify samples from $\mathcal{D}_E$ with a higher accuracy than 0.5 even with large dataset sizes. Again, this is likely caused due to the random initialization of the environment and has also been documented in relevant literature [68] in the case of adversarial imitation learning. While regularization or state compression are known to mitigate overfitting for AIRL [86], we omit these techniques due to the simplicity of the environment and satisfactory imitation performance.

In aggregation, the real and fake accuracies inform us about the discriminator loss (3.10) acquired during training, which we show in figure 3.7. After an initial drop in the discriminator loss, we can see that once the imitation policy π_ϕ has learned an optimal policy, the loss stabilizes and exhibits convergent behavior in the limit. Unsurprisingly, due to the high fake accuracy of the discriminator in the low demonstration regime, the overall achieved loss remains significantly lower the fewer demonstrations are available. As a consequence, we observe very different behavior in the returns achieved by π_ϕ. Firstly, the fewer demonstrations we supply AIRL with, the higher the absolute values of the learned reward function. This again confirms the overfitting hypothesis. Secondly, we see a large increase in the variance of learned rewards when repeatedly running AIRL over different random seeds for small

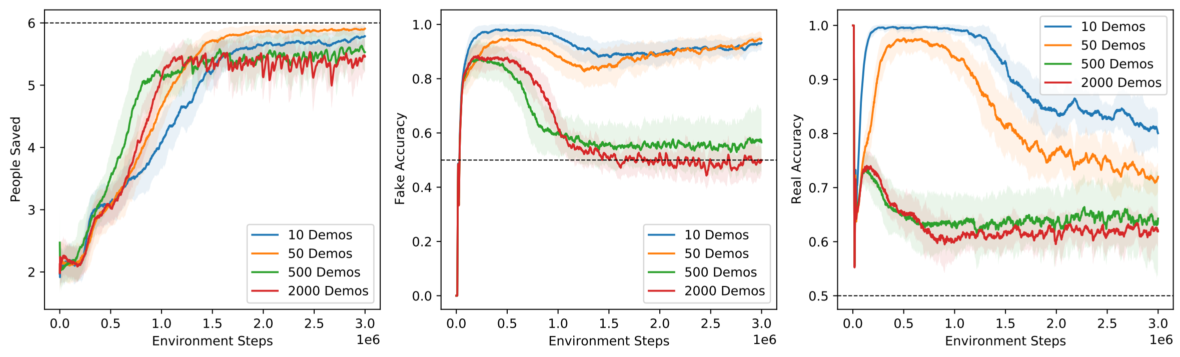


Figure 3.6: Comparison of demonstration dataset size on AIRL training performance. *Left*: Imitation performance of π_ϕ. All policies converge to a near optimal solution (6 people saved) regardless of demonstration dataset size. *Middle*: Discriminator accuracy on newly generated state-action pairs ('Fake Accuracy'). *Right*: Discriminator accuracy on state-action pairs sampled from the expert dataset ('Real Accuracy').

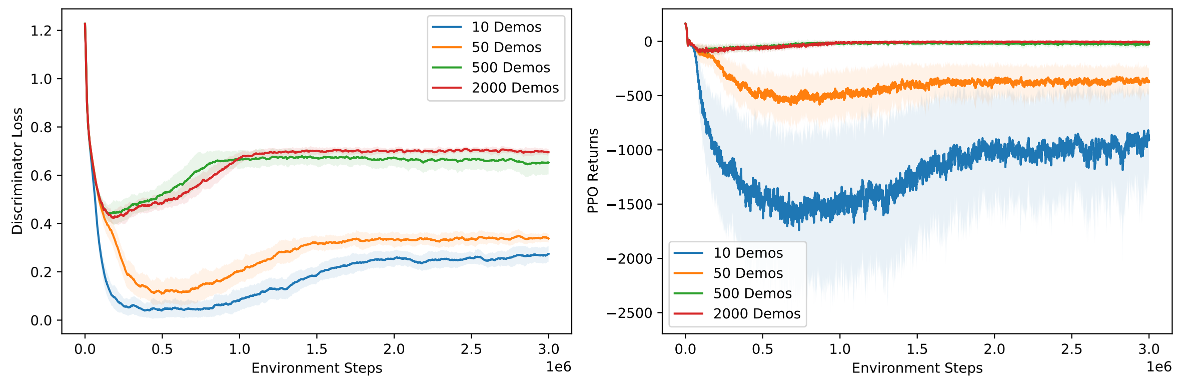


Figure 3.7: The impact of demonstration dataset size on the learned AIRL reward function. *Left*: Discriminator loss as a function of environment steps. AIRL converges in all cases, but indicates overfitting when there is insufficient expert data. *Right*: Obtained returns by the imitation policy π_ϕ on the reward function $r(s, a, s') = \log D_\theta(s, a, s') + \log 1 - D_\theta(s, a, s')$.

demonstration datasets, despite the underlying demonstrations not changing across seeds. While this does not pose a problem for the optimization of a single learned reward function, this can introduce challenges for multi-objective optimization. Namely, most multi-objective RL algorithms are sensitive to the scale of the individual reward components. However, normalizing the learned reward functions by appropriate constants can alleviate this issue significantly. We will test an appropriate choice of normalization in chapter 5 and, for now, proceed with the default scale learned through AIRL.

Once a reward function has been learned, we can proceed with multi-objective optimization by combining the predefined primary goal with the normative component. Figure 3.8 illustrates the found convex coverage set after training for different configurations of λ. Compared to AIRL, we train each agent for twice the amount of environment steps, slightly raise the learning rate of PPO (since our reward function is not updating over time), but keep the other PPO hyperparameters fixed (for a detailed description of hyperparameters used, see the appendix B.2). It can be seen that reoptimizing for a linear combination of normative and predefined reward, we can achieve various trade-offs by choosing appropriate policies on the CCS. Furthermore, when choosing $\lambda \approx 0.2$, the agent is able to reliably save all people in the warehouse while allotting the rest of its time to optimizing for the primary goal. However, we note that the desired outcome is sensitive to the choice of λ, with a slightly lower choice of $\lambda \leq 0.15$ already resulting in a significantly lower time spent on the primary goal and a higher choice of $\lambda \geq 0.25$ not resulting in saving every worker anymore. Considering the results from figure 3.7, this is problematic: Even a slight change in the demonstration dataset can yield significantly different reward scales after AIRL, thus yielding the search for λ of a previous run useless. Furthermore, we note that manually tuning λ and training a policy for each choice separately is very sample inefficient. For this reason, finding a desired trade off quickly becomes computationally infeasible as the complexity of

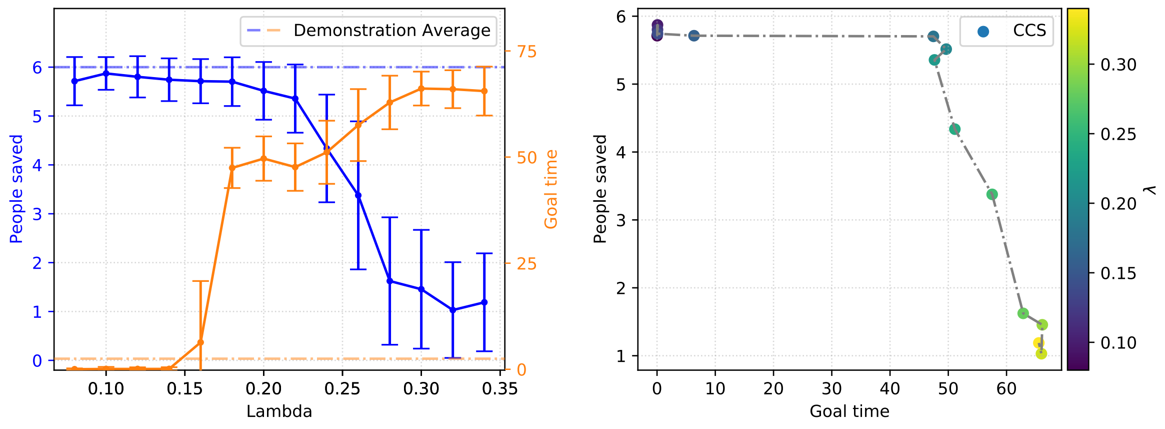


Figure 3.8: *Left*: Obtained trade off solutions between primary goal and normative behavior for different choices of λ. The respective average performances of the demonstration dataset used for AIRL is indicated through the two dashed lines. Error bars indicate a 2σ confidence interval over 100 distinct episodes. *Right*: The recovered convex coverage set in terms of goal time (x-axis) and people saved(y-axis). Colors indicate the choice of λ that led to each policy.

the environment and number of reward components grow. In the next chapter, we will introduce an active learning procedure for guiding the agent during optimization towards interesting solutions on the CCS, thus alleviating these scalability issues.

3.4. Conclusion

Overall, our results show us that it is indeed possible to steer a narrow RL agent towards normative behavior without the need for feature engineering. This closely overlaps with our results in [66], where we propose a single loop algorithm for AIRL reward shaping. Furthermore, we showed empirically that given sufficient data about the environment, the expert's behavior can be orthogonal to the primary goal without constraining the final agent to optimize for it. A key advantage of this is that one only leverages the demonstration data for learning which states of the world are considered normative, but does not require experts to know how to solve the primary goal in any form. This way, RL can still be used as a powerful optimization tool for tasks that cannot be performed by experts, but is merely constrained (regularized) into a normative direction by expert demonstrations. This approach is conceptually similar to policy orchestration [63] and ethical reward shaping [95], but readily allows the use of deep function approximation of the state space. Unlike [63] however, we do not orchestrate the IRL policy with the primary reward policy, but instead combine both sources of rewards into a single policy directly through multi-objective optimization. This has the advantage of putting less constraints on the space of policies that the agent is allowed to learn and thus can potentially achieve better solutions. On the other hand, our approach does not offer any direct interpretability due to distilling expert knowledge into the formally given primary goal using this single neural network policy.

4 Active Preference Learning

The fundamental challenge of multi-objective optimization is finding a policy (or a set thereof) that is able to capture a wide range of behavior suitable for different preferences. In the last chapter, we have shown empirically that constraining the set of scalarization functions to linear preferences and optimizing for a finite set of policies on the CCS can be sufficient for guiding an agent towards demonstrated normative behavior. However, we note that even though approximating the CCS was necessary to find a normative solution, we did not necessarily need all the found policies. In the burning warehouse environment from figure 3.4, for example, we would arguably be only interested in solutions that manage to save most of the workers, if not all. Secondly, if our goal is to incorporate normative behavior from a more diverse set of experts, deciding which subset of norms should be taken into account at test time will depend on the societal context of the agent. It is therefore crucial to develop an interactive algorithm that can adapt to various preferences. Furthermore, the specification of preferences can not be done explicitly, because the scalarization function can not be directly interpreted: Since some reward components are learned through deep IRL, the reward function does not carry any human understandable semantic value besides its relative values over different state-action pairs.

For these reasons, we propose an interactive multi-objective algorithm based on active learning, which aims to find the desired trade off from additional preference data. By employing a probabilistic model of pairwise preferences, we keep track of a distribution over linear scalarization functions that most accurately match the given feedback data. Using this distribution, we query an expert based on its expected value of information. Furthermore, we update the posterior distribution over scalarization vectors in a Bayesian fashion with the help of Markov Chain Monte Carlo (MCMC) methods. In this chapter, we will develop the MORAL (**M**ulti-**O**bjective **R**einforced **A**ctive **L**earning) framework for combining active preference learning with the multi-objective optimization algorithm described in the previous chapter. In section 4.1, we will start by outlining the mathematical details behind the preference model and how optimal queries are selected at runtime (section 4.1.1), as well as the choice of MCMC algorithm (section 4.1.2). This is followed by section 4.1.3, where we will derive the full MORAL algorithm. Finally, in section 4.2 we show that MORAL is capable of efficiently finding desired trade off solutions by comparing the quality of solutions to that of preference based RL and providing ablation studies regarding noise robustness, numbers of queries and reward scaling.

4.1. Methods

4.1.1. Bayesian Preference Learning

In order to capture the uncertainty of expert utilities, our goal is to maintain a probability distribution over reward functions, where a higher probability mass translates into more likely being aligned with the expert's preferences. We will focus only on the case of pairwise preferences, since they have been shown to yield especially promising results in the context of purely preference based RL [23]. Furthermore, they allow for a simple but effective way of Bayesian updating [74], which our approach is heavily building on. Unfortunately, probability distributions over the space of reward functions are intractable to maintain when the reward functions are complex nonlinear functions of the feature expectations encountered in a trajectory. For example, when the reward function itself consists of a neural network that has to be learned from preferences, a mechanism for sampling from the posterior over neural network parameters would be needed. Although, for example, this can be achieved through the means of variational methods, the main challenge of underestimating uncertainty in approximate Bayesian inference has yet to be tackled [14]. Deep RL from human preferences [23] avoids this by estimating the uncertainty of the reward network with an ensemble of reward predictors. Given two trajectories, the ensemble can then be used to quantify the degree of disagreement, which is in turn used to determine which pair is expected to be most informative for querying purposes.

We, on the other hand, assume that a vector-valued reward function $\mathbf{r}$ is available which already encodes most behavior of interest. Thus, we only learn a probability distribution over scalarization functions $f_{\mathbf{w}}(\mathbf{r}) = \mathbf{w}^T\mathbf{r}$ through maintaining a probability distribution $p(\mathbf{w})$ over preference vectors $\mathbf{w}$. Due to the linearity in $\mathbf{w}$, this allows for deriving a Bayesian updating scheme, analogous to the methods developed in [74]. Namely, we employ a Bradley-Terry model [17] for ranking pairwise preferences

$$p(i \succ j|\mathbf{w}) := \frac{\exp(\mathbf{w}^T\mathbf{r}(\tau_i))}{\exp(\mathbf{w}^T\mathbf{r}(\tau_j)) + \exp(\mathbf{w}^T\mathbf{r}(\tau_i))}, \tag{4.1}$$

where $\mathbf{r}(\tau) = \sum_{(s,a,s')\in\tau} \mathbf{r}(s,a,s')$ denotes the vector-valued return of the trajectory τ and $(i \succ j)$ denotes preferring a trajectory τ_i over another trajectory τ_j. Intuitively, the model rates trajectories exponentially higher in proportion to their achieved scalarized rewards (given a scalarization weight $\mathbf{w}$). Note that this probabilistic formulation inherently models some noise in an expert's preferences, which is especially useful when dealing with a fuzzily defined reward function $\mathbf{r}$. To enable learning about $\mathbf{w}$, we would like to derive the posterior probability $p(\mathbf{w}|i \succ j)$. Applying Bayes rule on the Bradley-Terry model tells us that the posterior probability is proportional to a prior distribution $p(\mathbf{w})$ times the likelihood

$$p(\mathbf{w}|i \succ j) \propto p(\mathbf{w}) \cdot p(i \succ j|\mathbf{w}). \tag{4.2}$$

Assuming that queries $(i \succ j)$ are observed sequentially, we therefore need to only keep track of a prior distribution at each time step and multiply it with the likelihood of the newest query. To be precise, given a set of queries $\{q_1, \dots q_n\}$, our posterior amounts to

$$p(\mathbf{w}|q_1, \dots, q_n) \propto p(\mathbf{w}) \prod_{t=1}^{n} p(q_t|\mathbf{w}). \tag{4.3}$$

What remains is the specification of a prior $p(\mathbf{w})$. In practice, this can be an arbitrary distribution in $\mathbb{R}^{\dim(\mathbf{w})}$ that resembles the expert's prior beliefs about which of the reward components need to be prioritized. For simplicity, however, we choose a uniform prior over the unit ball $||\mathbf{w}|| \leq 1$ and additionally constrain all components of $\mathbf{w}$ to be non-negative. We opt for this constraint, since the components of $\mathbf{r}$ are all assumed to be generally desirable. If this is not the case, one can simply drop the constraint and proceed analogously. Formally, this translates into the following prior

$$p(\mathbf{w}) := \begin{cases} 2^{\dim(\mathbf{w})} \left(\frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}\right)^{-1} & \text{if} \quad ||\mathbf{w}||_2 \leq 1 \quad \text{and} \quad \mathbf{w} \geq 0 \\ 0 & \text{else.} \end{cases} \tag{4.4}$$

Having defined the likelihood and the prior, we now have a mechanism for evaluating the posterior distribution up to a normalization constant. Using Markov Chain Monte Carlo (MCMC), this is sufficient for approximating the full posterior with a discrete set of samples. We will outline the type of MCMC algorithm used in section 4.1.2 and for now assume that we can freely calculate expected values over the posterior.

Besides obtaining a principled way for updating our belief about $\mathbf{w}$ through queries, having access to the posterior $p(\mathbf{w}|q_1, \dots, q_n)$ allows for choosing queries q_i optimally. We follow the derivation of [74] and define a query to be optimal if the pair of trajectories it contains satisfies

$$\max_{(\tau_i,\tau_j)} \min \Big(\mathbb{E}_{\mathbf{w}}[1 - p(i \succ j|\mathbf{w})], \mathbb{E}_{\mathbf{w}}[1 - p(j \succ i|\mathbf{w})]\Big). \tag{4.5}$$

This means that an optimal pair to be queried for should aim for maximally removing volume from the posterior distribution, see figure 4.1. To ensure that volume is removed regardless of the given

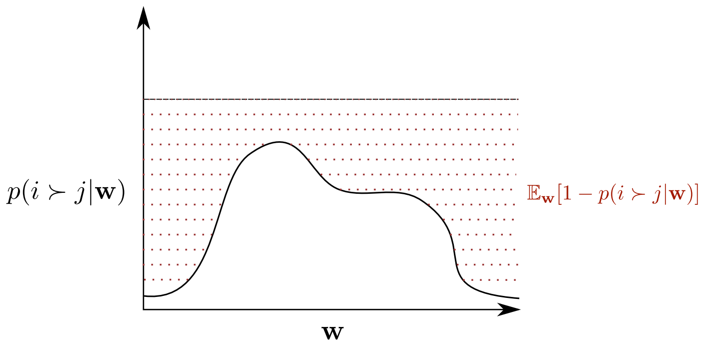

$p(i \succ j|\mathbf{w})$
$\mathbb{E}_{\mathbf{w}}[1 - p(i \succ j|\mathbf{w})]$
$\mathbf{w}$

Figure 4.1: Volume removed from a single query by taking the expectation over $\mathbf{w}$.

preference, a minimum over the two volumes corresponding to the two possible preference outcomes is taken. Overall, the more concentrated the mass of $p(i \succ j|\mathbf{w})$ and $p(j \succ i|\mathbf{w})$ is, the more volume we can expect to remove after a query. One technical difficulty arising in solving 4.5 is that when the dynamics of the MDP are unknown, searching through all possible pairs (τ_i, τ_j) becomes computationally intractable. For this reason, we choose to only approximately solve the maximization through a discrete search over pairs of trajectories that are generated during forward RL experience. We defer this discussion to section 4.1.3 where we will describe the full algorithm and instead first focus on how we approximate the expected values $\mathbb{E}_{\mathbf{w}}[X]$.

4.1.2. Markov Chain Monte Carlo

In order to obtain samples from $p(\mathbf{w}|q_1, \dots q_n)$ we employ the Metropolis-Hastings algorithm [21], which is a common method for estimating distributions via MCMC. Metropolis-Hastings constructs a Markov chain $\{\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(M)}\}$, where $\mathbf{w}_n = \mathbf{w}$ gets updated to $\mathbf{w}_{n+1}$ with the use of a proposal density $q(\mathbf{w}, \cdot)$. Let $\pi(\mathbf{w})$ denote a probability distribution of $\mathbf{w}$ we wish to sample from, which is easy to evaluate up to a constant We first sample a new weight $\mathbf{w}'$ from $q(\mathbf{w}, \cdot)$. Then the acceptance probability

$$\alpha(\mathbf{w}, \mathbf{w}') := \min\left(1, \frac{\pi(\mathbf{w}')q(\mathbf{w}, \mathbf{w}')}{\pi(\mathbf{w})q(\mathbf{w}, \mathbf{w}')}\right) \tag{4.6}$$

is calculated and used to define

$$\mathbf{w}_{n+1} := \begin{cases} \mathbf{w}' & \text{with probability } \alpha(\mathbf{w}, \mathbf{w}') \\ \mathbf{w} & \text{with probability } 1 - \alpha(\mathbf{w}, \mathbf{w}') \end{cases}. \tag{4.7}$$

Depending on the shape of the posterior, choosing an appropriate proposal density becomes is crucial for fast convergence. We have found a simple random walk proposal of the multivariate standard normal form

$$\mathbf{w}' \sim \mathcal{N}(\mathbf{w}, \sigma^2 \mathbf{I}) \tag{4.8}$$

to work sufficiently well enough in our use cases, but note that for higher dimensions a different choice might be more suitable. Having constructed the Markov Chain, we can then approximate the true posterior with a mean over the sampled point masses

$$p(\mathbf{w}) \approx \frac{1}{M} \sum_{k=1}^{M} \delta(\mathbf{w}^{(k)}), \tag{4.9}$$

where $\delta(\cdot)$ denotes the Dirac measure. Furthermore, we employ a warm up phase and initialize $\mathbf{w}^{(1)}$ with the posterior mode. This avoids poor initialization and the risk of a Markov chain getting stuck in areas with low probability mass. To compute the posterior mode, we follow the suggestion in [74] and replace the likelihood $p(\mathbf{w}|i \succ j)$ with a similar function that is also log-concave but has a mode of zero always. First, we rewrite the likelihood by defining

$$\Delta_{ij} := \mathbf{r}(\tau_i) - \mathbf{r}(\tau_i) \tag{4.10}$$

to be the return difference of two trajectories τ_i and τ_j. Using this definition, we can divide the numerator and denominator of (4.1) by $\exp(-\mathbf{w}^T\mathbf{r}(\tau_j))$ to obtain

$$p(i \succ j|\mathbf{w}) = \frac{\exp(\mathbf{w}^T\Delta_{ij})}{1+\exp(\mathbf{w}^T\Delta_{ij})}. \tag{4.11}$$

Wee see that the function above is of the form $\exp(x)/(1+\exp(x))$, which we will replace with a similar function $\min(1,\exp(x))$ by setting

$$\hat{p}(i \succ j|\mathbf{w}) := \min(1, \exp(\mathbf{w}^T\Delta_{ij})). \tag{4.12}$$

Taking the log of the expression above, we see that its mode evaluates to $\mathbf{w} = 0$. This way, initializing $\mathbf{w}^{(1)} := 0$ ensures that the Markov chain starts in an area with high probability mass. We will see in sections 4.2 and 5.2 that this simpler choice is sufficient for accurately recovering expert utilities from pairwise preferences. Finally, we can evaluate the volume removal expression (4.5) by taking a sample average over the Markov chain

$$\mathbb{E}_{\mathbf{w}}[1 - p(i \succ j|\mathbf{w})] \approx \frac{1}{M}\sum_{k=1}^{M}(1 - p(i \succ j|\mathbf{w}_k)). \tag{4.13}$$

4.1.3. Multi-Objective Reinforced Active Learning

We are now ready to formulate the MORAL framework for learning and trading off normative behaviors from a diverse set of experts. Figure 4.2 illustrates the full method in the case of a single learned reward function. MORAL consists of a two-step procedure including reward learning and multi-objective optimization through active learning. Step one takes a dataset of normative behavior $\mathcal{D}_E$ and learns a reward function f_θ through the use of AIRL (as described in section 3.2.1). In step two, reward functions from multiple sources are combined (e.g. a primary reward that describes the original goal of an agent) and traded off linearly with an actively learned scalarization vector $\mathbf{w}$. Formally, we supply a PPO agent with the reward function

$$\bar{r}(s,a,s') := \mathbb{E}_{\mathbf{w}}[\mathbf{w}^T\mathbf{r}(s,a,s')], \tag{4.14}$$

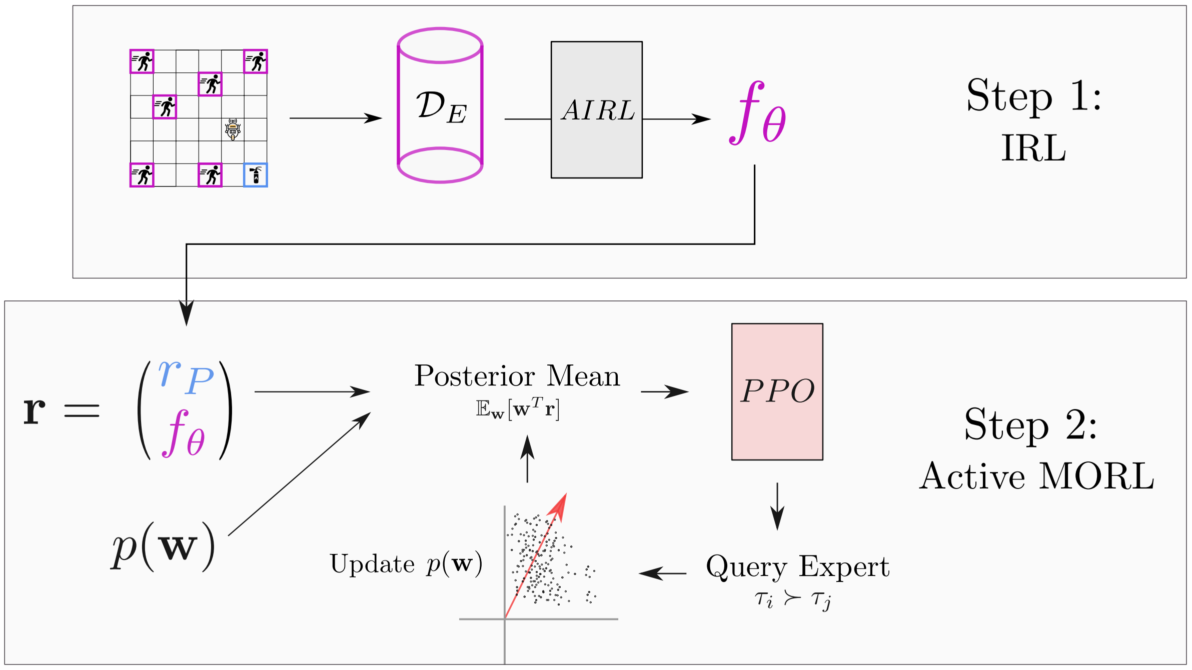


Figure 4.2: The two-step MORAL algorithm first learns a normative prior from expert demonstrations in the form of a reward function f_θ. This reward function is then combined with other sources of reward and a desired trade off is actively learned in the form of a distribution $p(\mathbf{w})$.

where the expectation is taken over the posterior distribution (4.3). If no queries have been obtained yet, then this expectation simply reduces to the mean of the prior distribution $p(\mathbf{w})$. We keep this reward function constant for a fixed number of time steps and perform several policy improvement steps at a time using PPO. After these time steps, a query is chosen according to (4.5) such that a maximum amount of volume is removed from the current distribution. However, we do not explicitly maximize the expected removed volume, but instead opt for a simpler heuristic search over generated trajectories. Namely, before each policy improvement step, we sample pairs of trajectories (τ_i, τ_j) from a forward RL experience buffer and keep the pair with the highest expected information content. The motivation for this choice is two-fold: Firstly, maximizing expression (4.5) explicitly is intractable for large MDPs with unknown dynamics. Secondly, due to the sample inefficiency of PPO, large numbers of trajectories are generated for training the policy itself. Thus, we can expect a discrete search for the best query over generated trajectories to yield a sufficient amount of information. After each query, we update the posterior by running MCMC on the aggregate of all previously obtained queries and pass the updated reward function to the PPO agent.

Algorithm 5 provides pseudocode for the MORAL framework. Note that there are multiple hyperparameters which we have omitted for the ease of exposition. Besides the hyperparameters of PPO and AIRL, we specify a query frequency at which experts are queried. In our implementation, this automatically induces the size of the set of trajectory pairs over which a discrete search is made. However, it would be possible to not only sample one, but in fact up to $|\mathcal{D}|^2$ pairs over which the search is performed at each PPO update step. As mentioned above, we found this to not be necessary when the number N of intermediate policy updates is high enough. Furthermore, we regularize the PPO agent with an additional entropy term (cf. section 2.2.1) to prevent premature convergence to the mean reward function of the current posterior. Although the algorithm below only includes a single learned reward, one can easily extend it to an arbitrary number of learned and primary reward functions. We will explore this setting in chapter 5 and for now test MORAL in the burning warehouse example from chapter 3.

Algorithm 5: MORAL (Single Expert)

```
Input: Expert trajectory data set $\mathcal{D}_E = \{\tau_i\}_{i=1}^N$, primary reward function $r_P$ and prior
  distribution $p(\mathbf{w})$.
Initialize: Reward network $f_\theta$ by running AIRL (algorithm 3) on $\mathcal{D}_E$, PPO agent $\pi_\phi$.
for $i = 0, 1, 2, \dots$ do
    Approximate the posterior $p(\mathbf{w}|q_1, \dots, q_i)$ with MCMC samples $\{\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(M)}\}$.
    Obtain reward function $\overline{r} = \frac{1}{M}\sum_{k=1}^M \mathbf{w}^{(k)T} \begin{pmatrix} r_P \\ f_\theta \end{pmatrix}$
    $volume \leftarrow -\infty$
    for $k = 0, 1, 2, \dots, N$ do
        Sample trajectories $\mathcal{D} = \{\tau_k\}_{k=1}^m$ from $p(\tau|\pi_\phi)$ through interaction with the environment.
        Update $\phi$ using PPO to maximize $\mathbb{E}_{\pi_\phi}\left[\sum_{t=0}^{T-1} \overline{r}(s_t, a_t, s_{t+1})\right]$.
        Sample a pair of trajectories $(\tau_i, \tau_j)$ from $\mathcal{D}$.
        $next\_volume \leftarrow \min\{\frac{1}{M}\sum_{k=1}^M (1 - p(i \succ j|\mathbf{w}^{(k)})), \frac{1}{M}\sum_{k=1}^M (1 - p(j \succ i|\mathbf{w}^{(k)}))\}$.
        if $next\_volume > volume$ then
            $next\_query \leftarrow (\tau_i, \tau_j)$
            $volume \leftarrow next\_volume$
        end
    end
    Query expert using $next\_query$ and save answer $q_i$.
end
Result: Trained policy $\pi_\phi$, expert utility $\mathbf{w}^* = \frac{1}{M}\sum_{k=1}^M \mathbf{w}^{(k)}$.
```

4.2. Experiments

We test the MORAL algorithm in the burning warehouse environment with a given set of synthetic preferences. Although multiple different trade-offs can be attained through multi-objective optimization of different preference vectors (see figure 3.8), we assume that the preference giver wants to save as many people as possible before optimizing for the primary goal. The preference giver has access to the raw frames and can therefore count the amount of people saved as well as the time spent in the goal. Given two trajectories τ_i, τ_j the preference giver then returns $i \succ j$ if the amount of people saved in τ_i exceeds that of τ_j and vice versa. In case both trajectories save the same amount of people, we opt for the trajectory with a higher goal time. We train MORAL with the same set of demonstrations as in figure 3.8 as well as the equal amount of environment steps during multi-objective optimization. For exact hyperparameters, see table B.3. Figure 4.3 compares the CCS found from manual multi-objective optimization with the path of solutions obtained during the active learning step of MORAL. As can be seen, MORAL converges to a near optimal solution that corresponds to the given preferences. Furthermore, it is able to find this solution with less than 10 queries, thus making it relatively feedback efficient. After having obtained enough feedback, MORAL directly optimizes for the underlying preferences of the expert, thus alleviating the need for calculating a full CCS.

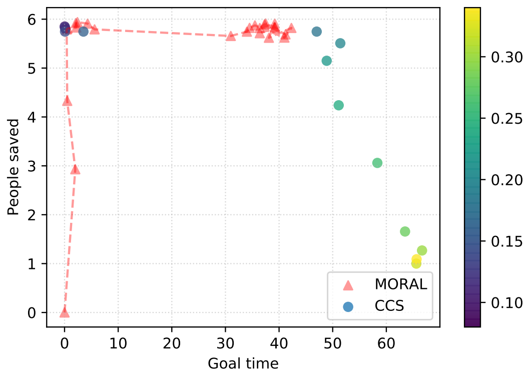


Figure 4.3: Achieved returns by MORAL during training compared to the CCS found by multi-objective optimization. Each point corresponds to the observed expected return and subsequent points are drawn for every new obtained preference. MORAL first maximizes the amount of people saved and then traverses along the CCS to match the given preferences. Colors of the CCS indicate the choice of λ that led to each policy.

While the figure above shows that interactively finding scalarization weights for multi-objective optimization can drastically reduce the amount of needed samples, it does not explain the benefits of querying the expert based on the maximum removed volume. Furthermore, approximating the removed volume (4.1) with a discrete search over generated trajectories does not necessarily produce optimal queries. For these reasons, we additionally test MORAL with random queries. This means that instead of actively querying, two randomly generated trajectories are provided to the expert at fixed time intervals. Overall, the query interval and amount of overall queries ($n = 25$) is held constant across experiments to ensure a fair comparison. Besides varying the way queries are chosen, we also vary the scale of the primary reward function by multiplying the primary reward function with a factor of 10. Figure 4.4 shows the achieved returns for both objectives in the case of active and random queries averaged over three random seeds. In this simple environment, the behavior of active queries and random queries is overall similar. Nonetheless, random queries are more sensitive to different reward scales than active queries. This can be attributed to the log-concavity of the query likelihood (4.12), leading to decreasing entropy in the posterior distribution. When working with inappropriately scaled reward functions, this can yield overly large update steps in a single direction, which are exacerbated by random queries. For example, repeatedly querying for the same pair of trajectories at early stages leads to very small posterior probabilities in certain areas, at which point the MCMC sampler disregards the corresponding scalarization weights completely. This is problematic, since obtained preferences during early training might either be noisy or not completely representative of the expert's utility function. It is therefore crucial to maintain appropriate levels of uncertainty throughout training to avoid early collapse

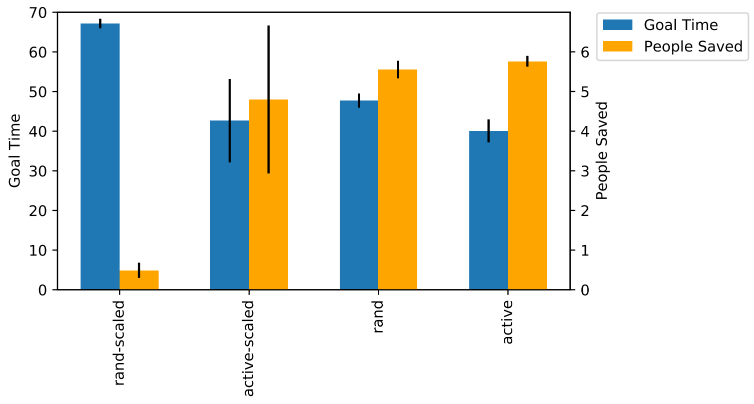


Figure 4.4: Comparison of random and active queries on the overall performance of MORAL. In the scaled environment, the primary reward is multiplied by a factor of 10. Error bars indicate standard deviations over three distinct runs (keeping the learned reward functions constant).

of the posterior distribution to a single scalarization weight. Although active learning does not directly change the magnitude of posterior update, we can see that choosing queries based on volume removal exhibits, somewhat surprisingly, better performance even when the reward scales are fundamentally different. However, a large increase in variance between runs can be seen which can significantly impact the amount of queries needed to achieve the desired result. This is especially problematic when learning multiple reward functions with a-priori unknown magnitudes. In chapter 5 we will discuss a possible normalization scheme to alleviate this issue.

4.2.1. Query Efficiency

To further examine the versatility of active learning, we study the amount of queries needed as well as robustness to noisy inputs. A main weak point of our experiments is the synthetic nature of the preference giver. Human experts tend to be less conclusive and can be expected to deliver significantly more noisy feedback over the course of training the agent. Besides that, we would like our interactive RL algorithm to be feedback efficient in order to minimize human time needed in real world scenarios. Deep RL agents typically require billions of interaction steps with their environment to learn a given task, while a pairwise comparison of two trajectories by an expert takes relatively less time. This leads to long waiting times in between queries which, assuming a single expert, increases the burden of delivering preferences. Figure 4.6 shows a comparison between achieved policies with respect to the overall amount of queries and injected preference noise. To add noise to the preference giver, we simply assume that with some probability $p < 1$ the preference is the result of choosing a trajectory at random, whereas otherwise the intended preference is obtained as before.

As expected, an increase in queries corresponds to a more aligned reward function, with $n = 25$ queries reaching a nearly optimal policy. Using $n = 10$ queries is still sufficient for optimizing both objectives, but achieves significantly less time in the primary goal, whereas $n = 5$ queries mainly

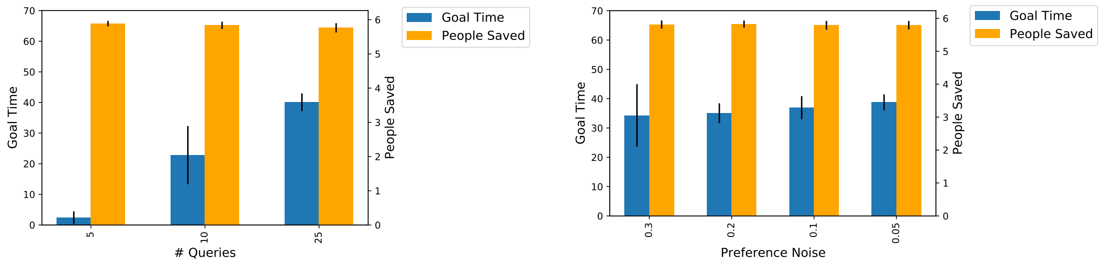


Figure 4.5: *Left*: Comparison of the amount of queries versus the achieved objectives after training for an equal number of environment steps. *Right*: The agent's performance with respect to different levels of preference noise during active learning. Error bars indicate standard deviations over three distinct runs.

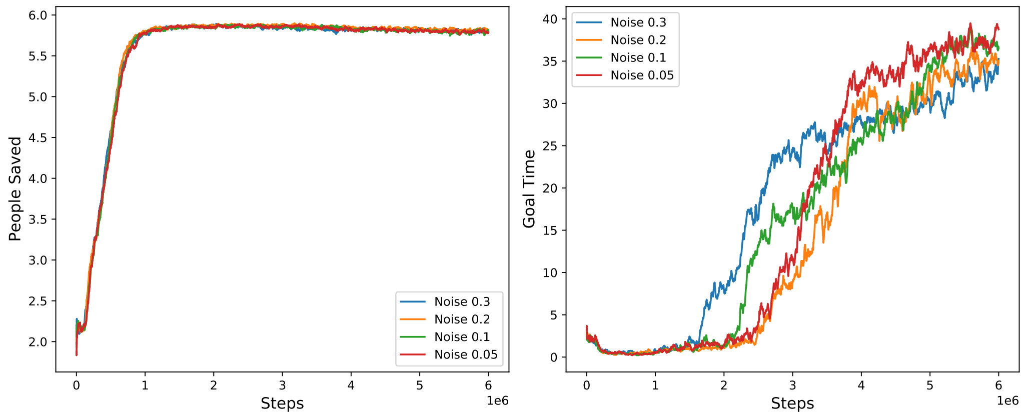


Figure 4.6: Training progress comparison of MORAL for different levels of preference noise. Overall, the difference between noise levels is subtle, with all runs converging to a comparable policy.

optimize for the normative reward component. We again note, however, that these results are sensitive to the scale of the reward functions. Bigger magnitudes of reward functions automatically imply larger likelihood updates, which can decrease the amount of queries needed but puts the posterior at risk of collapsing. In the case of preference noise, we test MORAL with $n = 25$ queries as before and vary the error probability $p \in \{0.3, 0.2, 0.1, 0.05\}$. We observe that the algorithm is relatively robust to noise, such that even if $p = 0.3$ the mean performance over three different random seeds does not significantly differ from that with no noise. However, a gradual increase in noise leads to larger variance in the observed results which, in the worst case, would require retraining the preference model if the expert's feedback is overly inconsistent. In terms of training speed of a single successful run, different noise levels do not seem to have any significant impact on the performance, see figure 4.6. Regardless of the noise, the agent learns to behave according to the demonstrated norms from the start of the optimization. The incorporation of the primary goal does vary to a higher degree, but yet remains comparable across all configurations. Overall these results indicate that, even if the expert provides ambiguous feedback about its underlying utility, the performance of actively learned scalarization weights does not suffer greatly for moderate levels of feedback uncertainty.

4.2.2. Deep Reinforcement Learning from Human Preferences

MORAL is capable of combining rewards from multiple sources, such as manually engineered reward functions and maximum entropy IRL rewards from demonstrations. However, any multi-objective optimization algorithm needs to select a final policy at runtime for the agent to act upon. Typically, multi-objective RL algorithms learn a diverse set of policies (cf. section 2.2.3) from which a final policy can be selected. MORAL on the other hand relies on an interactive algorithm for learning and selecting the final policy simultaneously, which comes at the cost of needing an expert in the loop providing pairwise preferences. As such, MORAL is conceptually similar to deep reinforcement learning from human preferences (DRLHP) [23], which also trains a reward model from pairwise preferences, but leaves out any manually engineered or demonstrated behaviors. For this reason, it seems relevant to compare MORAL to the performance of DRLHP and study their respective strengths and weaknesses.

Formally, DRLHP trains a reward model from scratch by using a Bradley-Terry model operating on the raw observations and actions of a trajectory

$$p_\theta(i \succ j|\theta) := \frac{\exp \sum r_\theta(s_t^i, a_t^i)}{\exp \sum r_\theta(s_t^i, a_t^i) + \exp \sum r_\theta(s_t^j, a_t^j)}, \tag{4.15}$$

where r_θ is a neural network operating on state-action pairs. Since the posterior $p(\theta|i \succ j)$ is intractable to compute analytically for large networks, [23] simply uses gradient-based optimization for updating the probabilistic model (4.15) with a cross-entropy loss on human labels. To ensure sample efficiency of queries, an ensemble of reward networks is used by choosing queries with the highest variance among ensemble predictions of preferences. However, the paper notes that in certain situations this can hurt performance, which is why we omit training an ensemble. We then query the expert at fixed time

	People Saved	Goal Time	# Preferences	Steps (IRL)
MORAL	5.76(±0.13)	**40.08**(±**2.9**)	25	3e6 (3e6)
DRLHP	5.62(±0.17)	12.32(±3.0)	1000	12e6 (-)

Table 4.1: Comparison of MORAL and DRLHP with regard to the normative objective (people saved) and the primary objective (goal time). Results were averaged over three random seeds, with the standard deviation denoted in brackets. In the case of MORAL, the number of preliminary IRL steps are indicated additionally to the environment steps of the multi-objective optimization phase.

intervals and provide it with the same preference function as MORAL, i.e. trajectories with more people saved are preferred and, if equal, those with a higher goal time. Table 4.1 compares trained policies of MORAL and DRLHP, as well as the amount of preferences and training steps used. We train DRLHP for two times the amount of total training steps as MORAL and employ a similar reward network architecture as used in AIRL (for details, see the appendix B.1.3). Like MORAL, DRLHP learns to optimize both objectives with the normative component strictly preferred (as encoded in the received preferences). However, DRLHP lacks behind MORAL in terms of goal time, thus underperforming on this multi-objective problem. This is likely due to the fact that learning a single reward model for both objectives through the reward network r_θ is cumbersome, leading to catastrophic forgetting of the primary objective when updating for the normative reward component. Furthermore, DRLHP is given 1000 preferences instead of 25, yet remains suboptimal after $12 \cdot 10^6$ environment steps.

To compare the relative training speed, we plot both objectives separately in figure 4.7. We can see that DRLHP first learns to optimize for the primary goal and only after many preferences incorporates normative behavior into its policy. Furthermore, after $2 \cdot 10^6$ steps DRLHP falls behind the goal time of MORAL while still not being able to save more than 2 people on average. From this point of view MORAL clearly outperforms DRLHP both in training speed and overall performance. However, we note that in this setup MORAL has clear practical advantages over DRLHP, which is why the comparison below should be interpreted with care. Firstly, MORAL is a supervised algorithm, in the sense that it has access to the normative rewards (through IRL) as well as the primary goal through a manually engineered reward function. DRLHP has to learn all objectives of interest in a single network, which is why it takes both longer and more preferences to learn a useful policy. Secondly, MORAL uses additional types of feedback in the form of demonstrations, while DRLHP does not. Although this partially explains the discrepancy in training speed of the two methods, it does not necessarily justify the poor performance of DRLHP after training until convergence. Overall, the comparison to DRLHP shows that when aiming to combine rewards from multiple sources, MORAL offers a strong alternative that exhibits more fine-grained control over desired behaviors. This should be unsurprising, since MORAL is a multi-objective algorithm, whereas DRLHP is not. Nonetheless, MORAL introduces some

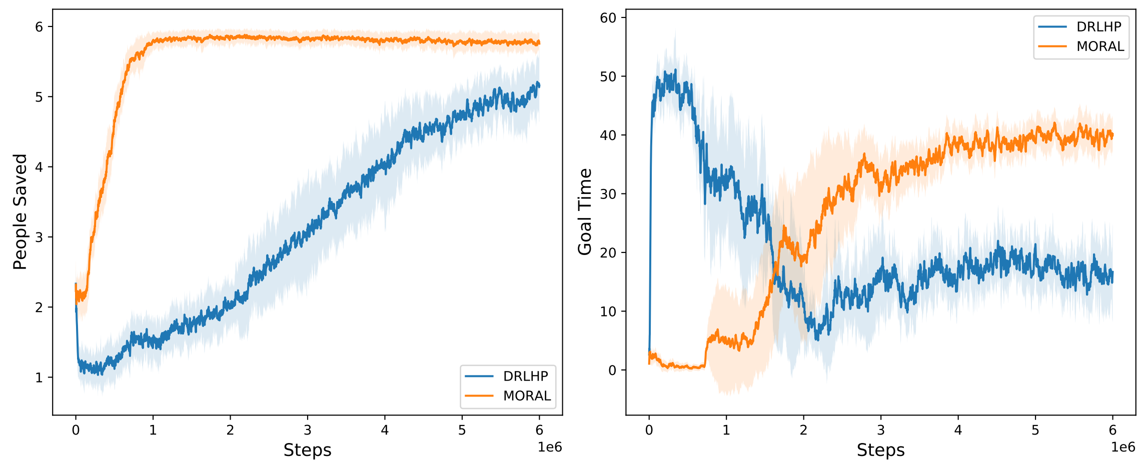


Figure 4.7: Training progress comparison of MORAL and DRLHP averaged over three random seeds. MORAL has direct access to different parts of the reward functions, allowing it to exceed the training speed of DRLHP by a large margin.

disadvantages that are not present in DRLHP. Firstly, scaling AIRL to larger domains such as Atari is still challenging [86] and thus MORAL cannot be automatically expected to reach the same performance on a wide variety of games from demonstrations alone. When the norms are temporally complex, we can therefore expect DRLHP to outperform MORAL given enough preferences. We consider this to be a more general weak point of AIRL, which would need further research on new IRL methods to be alleviated. Secondly, MORAL assumes that there is an inherent multi-objective decomposition of the problem, whereas in the real world demonstrating normative behavior might not be disentangleable. We conjecture that learning from demonstrations is still possible in this case, with the development of more sophisticated hierarchical RL methods seeming especially promising. We will discuss possible extensions to the MORAL framework in chapter 6.

4.3. Conclusion

In this chapter, we have introduced the MORAL framework for learning how to combine multiple learned reward functions according to expert preferences. By maintaining a distribution over scalarization weights through a model of pairwise preferences, we proposed an algorithm for efficiently querying experts using forward RL experience in order to maximize the amount of volume removed after each query. Using MORAL, we have shown that in the *Burning Warehouse* environment our agent is able to recover a combination of primary and normative reward which achieves the desired behavior of saving all lost workers while maximizing its time spent in its primary goal. Furthermore, ablation studies show that MORAL is relatively robust with respect to noisy preferences and scaling of individual reward functions. Finally, we showed that unlike DRLHP, MORAL is able to find the desired Pareto optimal solution using as few as 25 preferences, thus suggesting that previous value learning literature is lacking in terms of finding optimal policies in a multi-objective setting.

5 Controlling Diverse Norms

Learning and incorporating normative priors into deep RL agents is an important first step towards building responsible AI that acts according to what humans typically refer to as common sense. While such capability is a necessary condition for building aligned agents, it is definitely not a sufficient one. Namely, the normative aspect of the value alignment problem persists [32]. By definition, norms entail a societal context in which certain actions will be sanctioned or promoted. For this reason, we arguably want to build agents that can reliably learn and understand behaviors from multiple different experts. Assuming a sufficiently large (and diverse) dataset of normative behavior, we can then aim to learn a representation of an underlying consensus about which actions are generally less favorable and leave the possibility to fine tune the agent to more specific values for later.

Although this does not answer which values one ought to encode into AI systems, it at least offers technical insights into the problem of value open design. In this chapter, we aim to apply MORAL to environments with an implicit and explicit normative component and study the reliability as well as diversity of the respective policies. To do so, we assume that an agent is supplied with demonstrations coming from different experts that implicitly agree on one environment objective, but disagree otherwise. Through the combination of multiple learned reward functions, we then arrive at a normative reward prior that encodes both implicit and explicit knowledge about certain states of the world. In section 5.1.3 we will describe the problem of learning a normative reward prior and introduce an appropriate benchmark environment. Secondly, section 5.1 will introduce the adaptation of MORAL to multiple expert rewards, including a formal discussion about the Pareto optimality of linear expert aggregation. Finally, section 5.2 will provide empirical evidence for the effectiveness of MORAL compared to previous methods, as well as ablation studies for the number of preferences needed and the robustness of MORAL with respect to noise and misaligned preferences.

5.1. Methods

We can extend the MORAL framework from section 4.1.3 directly to the multiple experts setting by running IRL for different experts. This way, we first gather a dataset $\mathcal{D}_E = \cup_{i=1}^{k} \mathcal{D}_{E_i}$ coming from k different experts E_i. We assume that the dataset $\mathcal{D}_E$ is labelled, meaning that each demonstration can be assigned to its respective expert E_i. When this is the case, each run of AIRL can be trained to obtain an expert policy $\pi^*_{E_i}$ as well as its corresponding maximum entropy IRL reward function f_{θ_i}. Having obtained the respective narrow reward functions, we again form a vectorized reward function

$$r(s,a) := \mathbf{w}^T \begin{pmatrix} r_P(s,a) \\ f_{\theta_1}(s,a) \\ \vdots \\ f_{\theta_k}(s,a) \end{pmatrix}, \tag{5.1}$$

where r_P is an optional primary reward function that encodes additional prior knowledge about the problem domain. Subsequently, active learning is used to determine a distribution over the aggregation weights $\mathbf{w}$ in the same way as in the single expert scenario. Figure 5.1 illustrates the MORAL framework

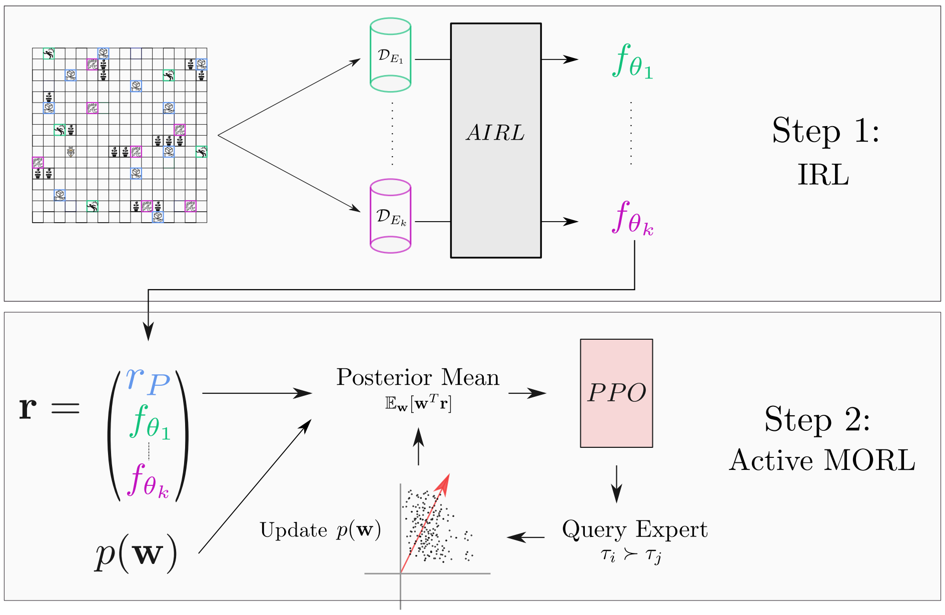


Figure 5.1: MORAL for multiple experts. In the IRL step, we learn multiple reward functions from different expert data sets. Subsequently, the learned functions get combined into a single vector-valued reward and scalarized through active learning.

for multiple experts. In the following, we will present the main theoretical advantage of using MORAL for trading off multiple expert behaviors as well as minor modifications to ensure better scaling properties of the algorithm.

5.1.1. Reward Normalization

MORAL learns multiple reward functions through successive runs of AIRL. However, increasing the amount of learned reward functions also increases the risk of highly different scales between the respective objectives within the final vector-valued reward used in active learning. As noted in section 3.3, AIRL is highly sensitive to the demonstration dataset as well as its size and will return fundamentally different reward functions, even if the task stays constant throughout different runs. For this reason, we propose to normalize each learned reward function, such that the marginal expected returns for each respective objective are approximately bounded by 1. This can be achieved without any computational overhead, by making use of the imitation policies $\pi^*_{E_i}$ from AIRL. Formally, we normalize learned rewards

$$\overline{f_{\theta_i}}(s,a) := \frac{f_{\theta_i}(s,a)}{|J(\pi^*_{E_i})|}, \tag{5.2}$$

where $J(\pi^*_{E_i})$ is simply estimated in a Monte Carlo fashion. However, we note that normalizing rewards directly impacts the magnitude of posterior updates in the active learning step. This is because the likelihood function $\hat{p}(i \succ j|\mathbf{w})$ depends on the per-trajectory reward difference Δ_{ij}. Higher reward magnitudes then tend to result in higher values of $||\Delta_{ij}||_2^2$, thus changing the overall magnitude of $\hat{p}(i \succ j|\mathbf{w})$. Although we found this normalization scheme to work sufficiently well, we anticipate that in practice the introduction of an additional hyperparameter $\beta \in \mathbb{R}^+$ that scales the overall reward $\mathbf{r}$ by a constant could be of help for adjusting the learning rate of Bayesian preference learning.

5.1.2. Pareto Optimality

Since MORAL trains for a linear combination of rewards, we are automatically guaranteed to optimize for a Pareto optimal solution after having converged to a final scalarization weight $\mathbf{w}^*$. This is because linear scalarization functions obtain policies on the CCS (cf. section 2.2.3) which itself is a subset of the Pareto frontier. Because of this, MORAL can be viewed as a practical implementation of Harsanyi's utility aggregation theorem [42] in a sequential decision-making context. Originally, Harsanyi's theorem is stated in the context of social choice theory and argues that when aiming for Pareto optimal solutions, linearly aggregating utilities of a population is both necessary and sufficient. In case of an MDP with a convex Pareto front, we can directly translate Harsanyi's aggregation theorem to the context of RL since the CCS coincides with the Pareto boundary.[1] Formally, Harsanyi's aggregation theorem then reduces to the following:

Theorem 5.1.1. *Let $\{u_1, \ldots, u_k\}$ be a set of expert utility functions with $u_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Furthermore, let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, \mathbf{r}, \mu_0 \rangle$ be a MOMDP with a convex Pareto boundary and $\mathbf{r} = (u_1, \ldots, u_k)$. Then, a policy π is Pareto optimal with respect to all experts if and only if there exist weights $w_1, \ldots, w_k$, such that π maximizes*

$$\mathbb{E}_\pi \left[\sum_{t=0}^{T} \sum_{i=1}^{k} w_i u_i(s_t, a_t) \right]. \tag{5.3}$$

Unfortunately, the assumptions behind Harsanyi's theorem are rather strong. Firstly, the assumption of a convex Pareto boundary might not be fulfilled, which means that not all Pareto optimal solutions can always be recovered by linear aggregation. Secondly, it assumes that each expert has similar knowledge about the dynamics of the MDP. If this is not the case, a Pareto optimal agent would need an aggregation procedure that depends on the truthfulness of expert beliefs over time [25]. However, it is still unclear to what degree the assumptions of Harsanyi's theorem are violated in complex environments. We will discuss possible implications of this in chapter 6.

Regardless of the convex Pareto boundary assumption, theorem 5.1.1 provides insights into how MORAL connects to social choice theory. Assume that we want to build an RL agent that acts Pareto optimally with respect to each expert. In its general form, Harsanyi's theorem assumes that each individual i has a welfare ordering $\succsim_i$ on a set of probability measures over a space of social states X [41], which is represented by an expected value of a utility function u_i. While in social choice theory, approximate knowledge of u_i is often implicitly assumed, this is hard to satisfy for arbitrary MDPs due to the same reasons for which manual reward specification is intractable in real-world problems. To tackle this, MORAL applies IRL to first learn the utility functions u_i from each expert separately. If successful, this abstracts an individual's preferences into a scalar reward function which can subsequently be used for finding Pareto optimal solutions. Another difference that has to be tackled in RL, however, is that knowledge of utility functions alone does not constitute enough information about possible aggregate solutions. This is because of the sequential nature of MDPs, which introduces the dependency on time and (unknown) environment dynamics. As a consequence, the space of solutions over which needs to be optimized is not merely a set of states, but rather a set of trajectories. But arbitrarily collecting preferences about all possible trajectories is intractable, since the probability of encountering meaningful trajectories by chance decreases to 0 as the size of the underlying environment grows. Hence, multi-objective RL is necessary which MORAL employs interactively with the help of active learning.

Aside from Harsanyi's aggregation theorem, linear aggregation of learned utility functions has another convenient property. Namely, if every utility function agrees on a certain set A of trajectories being generally undesirable, then we can ensure that optimizing for any linear combination of such utilities will still deem A undesirable. In the case of IRL, this means that if the demonstrations from each expert put a sufficiently low probability mass on trajectories in A, the policy resulting from MORAL can be made arbitrarily averse to trajectories in A regardless of the obtained preferences. Formally, we can state this in the following theorem:

Theorem 5.1.2. *Let $\mathcal{D}_E = \cup_{i=1}^{k} \mathcal{D}_{E_i}$ be a demonstration dataset stemming from k individuals and let $p(\tau|\theta_i)$ denote the maximum entropy IRL distribution (3.7) resulting from the maximum likelihood estimate (3.8) over observations $\mathcal{D}_i$. Furthermore, let $\mathbf{w} \in \mathbb{R}^k$ be arbitrary with $\mathbf{w} \geq 0$, $||\mathbf{w}||_1 = 1$ and set*

$$\pi^* = \arg\max_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{T} \left(\sum_{i=1}^{k} w_i r_{\theta_i}(s_t, a_t) \right) - \log \pi(s_t, a_t) \right]. \tag{5.4}$$

[1]Otherwise, one needs to introduce a policy mixing assumption, which allows constructing policies of the form $\pi(a|s) := Z\pi_1(a|s) + (1-Z)\pi_2(a|s)$, where $Z \sim \mathit{Bern}(\lambda)$, $\lambda \in [0,1]$ is a Bernoulli random variable that determines which policy to play and is sampled at the start of each new episode.

Then, in finite MDPs we have that for all sets A over trajectories and $\epsilon > 0$ there exist $\delta_1, \dots \delta_k > 0$ such that $\forall i \in \{1, \dots, k\} \forall \tau \in A : p(\tau|\theta_i) < \delta_i$ implies $p(A|\pi^*) < \epsilon$.

To prove theorem 5.1.2, we first note that π^* is simply the maximum entropy RL policy resulting from the reward function $\sum_i w_i r_{\theta_i}$ and first prove the following

Lemma 5.1.3. *Given $\mathbf{w} \in \mathbb{R}^k$ with $\mathbf{w} \geq 0$, $||\mathbf{w}||_1 = 1$ and π^* being defined as in (5.4), we have that*

$$\pi^* = \arg\min_{\pi} \sum_{i=1}^{k} w_i D_{KL}(p(\tau|\pi)||p(\tau|\theta_i)). \tag{5.5}$$

Proof: By definition of the Kullback-Leibler divergence, we have

$$D_{KL}(p(\tau|\pi)||p(\tau|\theta_i)) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \log \pi(a_t|s_t) - r_{\theta_i}(s_t, a_t)\right] + \log Z_{\theta_i}. \tag{5.6}$$

Using $||\mathbf{w}|| = 1$, we can now rewrite the weighted sum of Kullback-Leibler divergences into the desired form:

$$\begin{aligned}\sum_{i=1}^{k} w_i D_{KL}(p(\tau|\pi)||p(\tau|\theta_i)) &= \sum_{i=1}^{k} w_i \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \log \pi(a_t|s_t) - r_{\theta_i}(s_t, a_t)\right] + \sum_{i=1}^{k} w_i \log Z_{\theta_i} && (5.7)\\ &= \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \log \pi(a_t|s_t) - \left(\sum_{i=1}^{k} w_i r_{\theta_i}(s_t, a_t)\right)\right] + \sum_{i=1}^{k} w_i \log Z_{\theta_i}. && (5.8)\end{aligned}$$

Minimizing over π yields the desired expression, as the normalization functions Z_{θ_i} as well as the weights w_i are constants as a function of the policy π. ■

Lemma 5.1.3 shows us that optimizing for a linear combination of reward functions obtained through maximum entropy IRL reduces to minimizing a weighted sum of Kullback-Leibler divergences to the respective induced marginal expert distributions. This is useful, because one can derive an analytic solution to the weighted Kullback-Leibler average for arbitrary probability density functions.

Lemma 5.1.4 (Weighted Kullback-Leibler average [8])**.** *Let $\{p_1, \dots, p_k\}$ be a finite set of probability density functions over $\mathbb{R}^n$ (i.e. $p_i : \mathbb{R}^n \to \mathbb{R}$, $\int_{\mathbb{R}^n} p_i(\mathbf{x}) d\mathbf{x} = 1$ and $p_i \geq 0$). Then, for any probability density p and weights $\mathbf{w} \in \mathbb{R}^k$ with $\mathbf{w} \geq 0$, $||\mathbf{w}||_1 = 1$, the weighted Kullback-Leibler average*

$$p^* = \arg\inf_{p} \sum_{i=1}^{k} w_i D_{KL}(p||p_i) \tag{5.9}$$

is given by

$$p^*(x) = \frac{\prod_{i=1}^{k} p_i(\mathbf{x})^{w_i}}{\int \prod_{i=1}^{k} p_i(\mathbf{x})^{w_i} d\mathbf{x}}. \tag{5.10}$$

We have omitted the proof of Lemma 5.1.4, but note that the proof follows from a simple algebraic manipulation of the Kullback-Leibler divergence. We are now ready to prove theorem 5.1.2.

Proof of theorem 5.1.2: Let $\mathbf{w} \in \mathbb{R}^k$ be arbitrary with $\mathbf{w} \geq 0$, $||\mathbf{w}||_1 = 1$. Combining Lemma 5.1.3 and 5.1.4 we obtain

$$\begin{aligned}p(A|\pi^*) &= \int_A \frac{\prod_{i=1}^{k} p(\tau|\theta_i)^{w_i}}{\int \prod_{i=1}^{k} p(\tau|\theta_i)^{w_i} d\tau} d\tau \\ &\leq \int_A \frac{\prod_{i=1}^{k} \delta_i^{w_i}}{\int \prod_{i=1}^{k} p(\tau|\theta_i)^{w_i} d\tau} d\tau = \frac{|A| \prod_{i=1}^{k} \delta_i^{w_i}}{\int \prod_{i=1}^{k} p(\tau|\theta_i)^{w_i} d\tau}.\end{aligned} \tag{5.11}$$

Now, if $|A| < \infty$ (which is satisfied for finite MDPs), we can make any of the δ_i arbitrarily small, such that $p(A|\pi^*) < \epsilon$ for any $\epsilon > 0$. ■

While theorem 5.1.2 gives us the theoretical assurance of minimizing undesirable states in the final linear aggregate policy of MORAL, it might not always be clear whether the respective maximum IRL distributions put sufficiently low probability mass on certain areas of the trajectory space. For this reason, it should be rather seen as an approximate justification rather than a formal criterion which will hold for all environments. Nonetheless, we will provide further empirical evidence to show that linear aggregation of IRL reward functions are robust with respect to their marginal safety properties regardless of the scalarization weights w_i in section 5.2.

5.1.3. Experimental Setup

In order to properly evaluate MORAL, we need to define an environment that adequately tests scalability in environment size as well as norm diversity. Inspired by the burning warehouse environment from chapters 3 and 4, we define the *Delivery* environment as follows: Delivery consists of a 16×16 grid world with four different entities. The environment is initialized randomly, placing each of the entities as well as the agent at a random position on the grid. As before, we assume the agent to have access to 9 actions, including moving along the four axes, interaction with one of its four adjacent grid cells and a null action. The primary goal of the agent is to deliver boxes to specified goal locations (*deliver*) on the grid. Besides this, there are people present on the grid that the agent can choose to assist (*help*) as well as pollution that can be removed (*clean*). Each of these outcomes are achieved by interacting with the respective tiles, after which the cells turn empty. Finally, we assume that there are vases placed throughout the environment which automatically break once the agent steps on them and can not be interacted with otherwise.

To make the environment challenging, we limit the length of each episode to $T = 50$ time steps, after which the environment resets to a new grid. Furthermore, we place 12 of the *deliver*, *help*, *clean* objectives respectively and 8 vases. We'll elaborate on this choice of environment hyperparameters in section 5.1.4. Overall, we view this environment as a multi-objective task with three norms, of which two are explicit (*help*, *clean*), whereas not breaking the vase is implicit. The idea behind this is that when learning norms from demonstrations, we cannot expect to receive explicit demonstrations of norm breaking acts. On the one hand, this might be due to safety concerns of demonstrating norm breaking behavior. On the other hand, when collecting a demonstration dataset from a large population, providing explicit instructions to the demonstration givers might not only be infeasible, but also bias the demonstrator's representation of normative behavior.

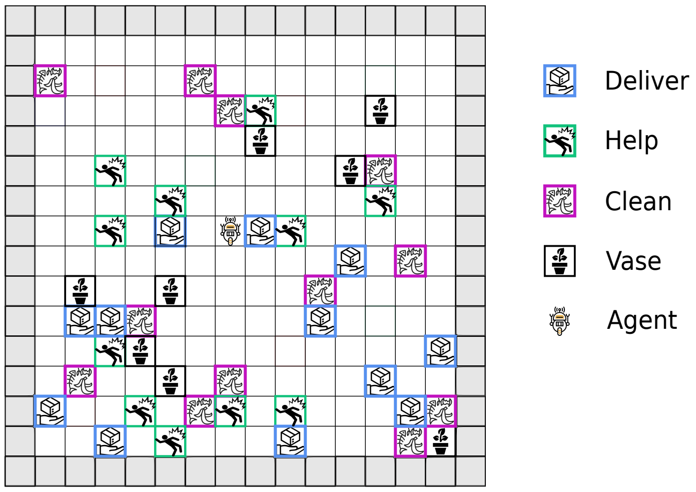


Figure 5.2: Illustration of the *Delivery* environment. The agent's primary goal is to deliver packages to the assigned cells. However, the agent might encounter people in need of help as well as street pollution, which normative behavior might favor over the primary task. Finally, vases are placed throughout the world which introduces negative side effects if being stepped on.

As previously described, we employ a neural network for function approximation in AIRL. However, due to the increased environment size we now use a convolutional network architecture for the reward learner f_θ instead of a fully connected one. At its core, the network treats grid world states as images and learns a sequence of 2×2 kernels with varying channel dimension. Namely, the network consists of three convolutional layers with 32, 32 and 16 channels respectively, followed by a linear layer that outputs a scalar value. Furthermore, we use the same dual channel architecture as outlined in 3.2.2, where the state and next state are passed through two parallel convolutional layers which are combined only afterwards. Activation functions are used throughout each layer, by applying a LeakyReLU with slope parameter $\alpha = 0.01$. For an illustration of the architecture, see B.1.2. Besides this, we do not vary the architecture for the PPO policy that is used throughout AIRL and active learning, and we remain with the convolutional architecture described in B.1.1.

5.1.4. Preference Elicitation

For *Delivery*, we slightly alter the way preferences are provided. This is due to the increased complexity of the environment, which allows for many Pareto optimal policies that could be regarded normative. Hence, we assume that the preference giver has a subjective distribution $m \in [0,1]^n$ over the n different reward functions which we would like to scalarize with MORAL. Note that although this implies that the preference giver's utility is a function of the n available reward functions, such simplifying assumption can easily be dropped whenever needed. MORAL will always learn a scalarization weight $\mathbf{w}$ that most closely matches the preferences, regardless of the ground truth utility function that is to be approximated. Given a pair of trajectories (τ_1, τ_2), we then calculate two vectors $j_i = [o_1^i, \dots, o_n^i]$ ($i \in \{1,2\}$), where o_k^i denotes the ground truth returns of the k-th objective in trajectory i. For example, if trajectory τ_1 delivers 3 packages, helps 1 person and cleans 3 tiles, then $j_1 = [3,1,3]$. In this example, vases have been omitted in the specification of j_i because they have been assumed to be implicit.[2] Having obtained a pair of returns (j_1, j_2) corresponding to the trajectories (τ_1, τ_2), we provide preferences according to

$$i^* = \arg\min_{i \in \{1,2\}} D_{KL}(\overline{j_i} || m), \tag{5.12}$$

where $\overline{j_i} := \frac{j_i}{||j_i||_1 + \epsilon}$ is the normalized vector of returns and $\epsilon > 0$ is a small constant for avoiding numerical instabilities.

Providing preferences this way gives us the ability to study the qualitative behavior of MORAL for different configurations of m. Furthermore, we choose to place an equal amount of *deliver*, *help* and *clean* objectives on the grid because it enables us to experimentally validate the deviation of MORAL to the true underlying utilities of the preference givers. This is because it makes the objectives symmetric in the sense that they only differ in the type of reward signal (learned/predefined) and because the episode time limit is chosen such that a marginal PPO agent trained on only one objective can never achieve the maximal return. As such, the optimal policy for a ground truth distribution m does indeed correspond to a return vector j^* with $D_{KL}(\overline{j^*}||m) = 0$. In other words, we ensure that given m there always exists a Pareto optimal solution with returns that have zero Kullback-Leibler divergence to m.

5.2. Experiments

We will describe the IRL and active learning stages separately and will assume that the reward functions learned in from AIRL are being held fixed throughout the active learning experiments unless stated otherwise. For AIRL, we train two reward functions for the *help* and *clean* objectives from 1000 demonstrations respectively, using the hyperparameters from appendix B.3.1. Furthermore, we assume that both demonstration givers respect the implicit vase objective. We do so by training two PPO agents on reward functions that give a penalty of -1 for breaking the vase and give a reward of $+1$ for helping or cleaning otherwise. For the training performance of the demonstration policies, see the appendix A.2. Figure 5.3 shows the four objectives achieved while running AIRL on each of the two expert's demonstration sets. As expected, both policies optimize for their respective objectives from the start, while managing to minimize the amount of vases broken. Besides that, the discriminator network converges to a real and fake accuracy at around 0.6, as shown in figure A.6. Although this might

[2]This is an arbitrary design choice, and providing preferences by factoring in the implicit vase objective could have been explored just as well.

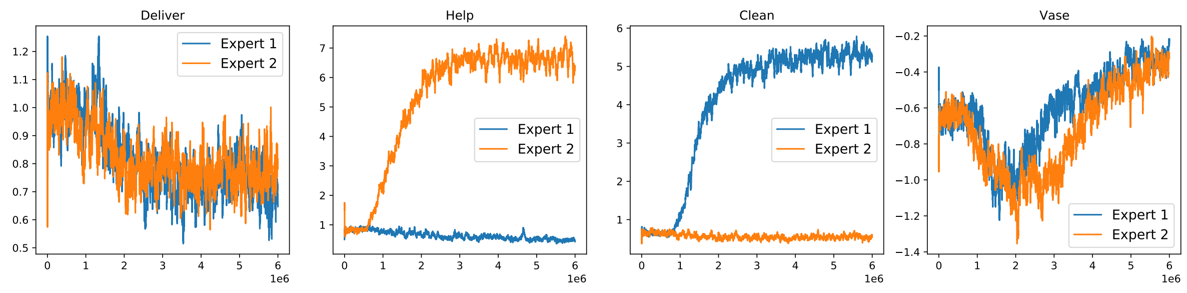


Figure 5.3: AIRL training performance in *Delivery* for two different demonstration datasets. Both experts implicitly avoid breaking vases, whereas expert 1 has a preference for *clean* and expert 2 has a preference for *help*.

indicate slight overfitting of the reward network, we found the performance of policies at reoptimization time to be sufficiently close to the original demonstrations, which is why we did not explore any further regularization techniques. Finally, we note that choosing 1000 demonstrations for each expert was motivated by both ensuring to learn sufficiently accurate reward representations as well speeding up the overall training process.

5.2.1. Expert Aggregation

We start by testing MORAL in *Delivery* for the two-dimensional reward function obtained from the AIRL runs of figure 5.3. To do so, we run active learning with 25 queries multiple times for different choices of the preference vector $m \in \mathbb{R}^2$. By plotting the ratio of the ground truth preference m_1/m_2 against the achieved objectives, we can then evaluate MORAL's ability to recover a wide variety of expert preferences in the multi-objective stage. We show this behavior in figure 5.4. As can be seen, MORAL correctly recovers the expert's preferences while achieving a constantly high performance in terms of the respective objectives. Furthermore, we see an instantiation of theorem 5.1.2, where the amount of vases broken consistently stays close to zero regardless of the preference ratio m. This is because the reward functions found in the AIRL step (see figure 5.3) favor trajectories that break fewer vases on average. Besides this, figure 5.4 plots the deviation of the observed objective ratio $\frac{\text{help}}{\text{clean}}$ from the provided ratio m in terms of the Kullback-Leibler divergence. We do this to quantify to which degree the posterior distribution over scalarization weights $\mathbf{w}$ matches the given preferences. However, we note that this is only an approximate metric for the distance between the posterior over $\mathbf{w}$ and the preference giver's true scalarization weights $\mathbf{w}^*$. We use this approximate metric because, since the reward functions are learned, we do not have access to the true scalarization weights $\mathbf{w}^*$ that match the preferences m. Nonetheless, as discussed in section 5.1.4 our choice of environment and m ensures that the discrepancy between these two metrics is always approximately equal to zero. As can be

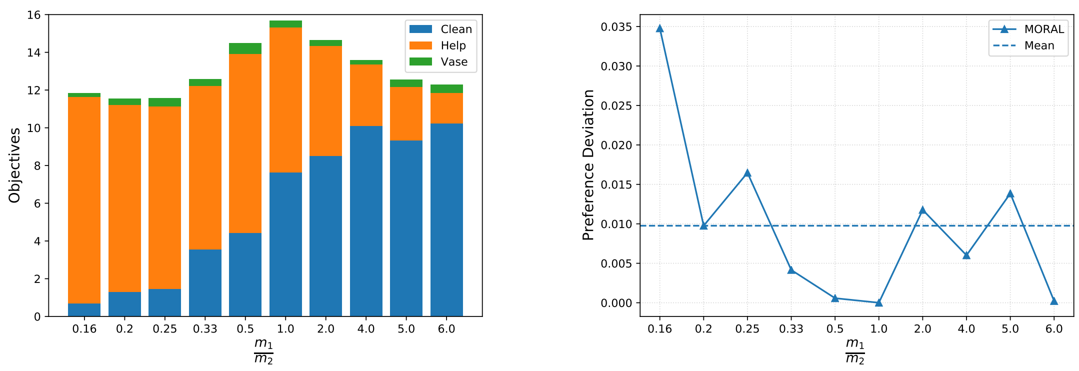


Figure 5.4: *Left*: Provided preference ratio m versus the average number of objectives achieved by MORAL after training. The implicit vase objective is automatically optimized for, thus minimizing the amount of broken vases regardless of m. *Right*: Kullback-Leibler divergences between observed objective ratios and the provided ratio m. MORAL is able to closely recover the true preference vector for different choices of m.

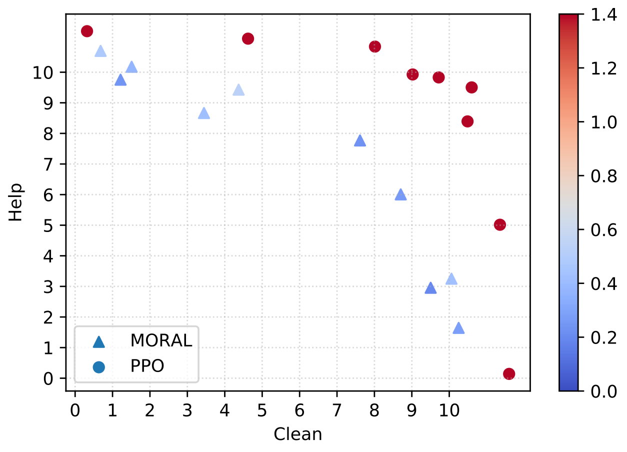


Figure 5.5: The convex coverage set found by MORAL and PPO on manually defined reward functions. Colors indicate average amount of vases broken. MORAL retrieves a comparable shape of solutions to PPO despite only having access to demonstrations and preferences, while implicitly incorporating the normative vase constraint.

seen, MORAL achieves low divergence metrics across different choices of m, indicating that the desired preferences are adequately recovered in the active learning stage. Nonetheless, the results are subject to noise as can be seen from the relatively larger divergences for some choices of m.

The results above suggest that can MORAL closely match the given preferences, but what remains to be tested is how MORAL compares to manually choosing scalarization weights when having access to the true reward functions. As discussed previously, MORAL retrieves Pareto optimal solution in the space of its (partially) learned reward functions. However, accumulating errors in learned rewards might lead to an overall performance that does not match an agent optimizing for the true reward functions. Again, we note that although a true reward function might not exist, in our experimental setup we synthesize demonstrations from an RL agent trained on some true reward function. That way, we can easily compare solutions found by MORAL with those that come from scalarizing versions of the original reward functions. For this reason, we train a traditional RL agent using PPO on a manually engineered reward function $r(s, a) = \lambda r_1(s, a) + (1 - \lambda)r_2(s, a)$, where $r_1(s, a)$ corresponds to a reward of $+1$ for each cleaned tile and $r_2(s, a)$ corresponds to a reward of $+1$ for each person helped. For MORAL, we train agents using the same set of preferences from figure 5.4 to achieve a diverse set of Pareto optimal policies, whereas for PPO we simply vary the hyperparameter $\lambda \in [0, 1]$. Figure 5.5 shows convex coverage sets for the respective algorithms. MORAL recovers a qualitatively similar CCS to PPO, but exhibits slightly lower spanned volume. This, however, is to be expected due to MORAL implicitly avoiding vases, whereas PPO does not. As can be seen, MORAL achieves to avoid significantly more vases than PPO, which was only trained on the explicit norms of each expert. In conclusion, we see that in *Delivery*, aggregating shaped AIRL reward functions does not lead to a major drop in performance as compared to aggregating their sparse, manually engineered counterparts.

Finally, we further illustrate the benefits of theorem 5.1.2 in the context of a malicious preference giver. By assumption, the theorem guarantees that any scalarization $\mathbf{w}$ of the vector-valued learned reward function will lead to minimization of undesirable states. In this example, although each of the experts optimizes for a different set of tasks, they share the common goal of avoiding states in which vases end up broken. To empirically validate the robustness of MORAL in this case, we give preferences in the following way: Given a pair of trajectories (τ_1, τ_2), prefer the trajectory that breaks more vases. Figure 5.6 shows that despite these preferences, the policy does indeed optimize for the opposite, which is minimizing broken vases. Aside from the theorem, this result is rather unsurprising, but intuitively appealing. MORAL only learns which of the experts to prioritize at runtime, but does not infer any fundamentally new reward information in its active learning stage. In this sense, the two-step procedure of MORAL can also be thought of as a safety mechanism. As long as safety of the marginal reward functions can be guaranteed, MORAL can subsequently interact with an arbitrary expert without compromising the safety of the system.

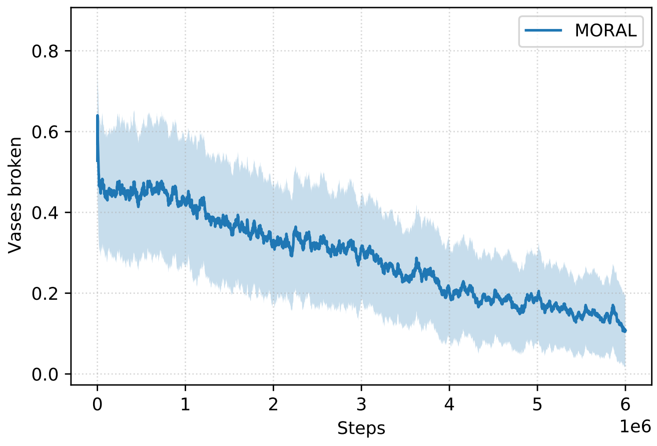


Figure 5.6: Robustness of MORAL with respect to malicious inputs. MORAL minimizes the amount of broken vases even in the presence of counterfactual preferences. The shaded area marks standard deviations across three distinct random seeds.

5.2.2. Diversity

We now scale up the complexity of the active learning stage and evaluate MORAL in the full setting. This means that we train MORAL on a three-dimensional reward function $\mathbf{r}(s,a) = (r_P(s,a), f_{\theta_1}(s,a), f_{\theta_2}(s,a))$, where $r_P(s,a)$ is $+1$ when a package gets delivered and $f_{\theta_1}, f_{\theta_2}$ are the AIRL reward functions used in the previous section. To retrieve a representative subset of the Pareto front, we then enumerate all possible ratios in $\{1,2,3\}^3$ for choosing m and iterate through the active learning stage. Furthermore, we limit the amount of preferences during active learning to 25. For a comparison of the amount of queries needed, we refer to section 5.2.4. Figure 5.7 shows the obtained CCS for each value of m. We choose to plot pairs of the three explicit objectives *deliver, help, clean*, whereas the radius of added circles around each policy indicates the relative amount of vases broken. Thus, a bigger radius indicates a policy broke more vases compared to the other solutions, with a radius of 0 indicating no broken vases. Besides that, we also color each policy according to the third explicit objective that is not present in the two-dimensional projections for each plot.

As can be seen, MORAL retrieves a diverse set of solutions that represents the different choices of m. Besides this, we see that the amount of broken vases correlates directly with the weight put on the delivery reward function. This is unsurprising, since the manually engineered delivery reward does not encode any knowledge about the various vases in the environment. Nonetheless, we observe that for appropriate choices of m, the amount of broken vases can be held relatively low while delivering an adequate amount of packages by simply putting more weight on either of the learned reward functions. Note, however, that this setup does not let us directly apply theorem 5.1.2, since we are not working with

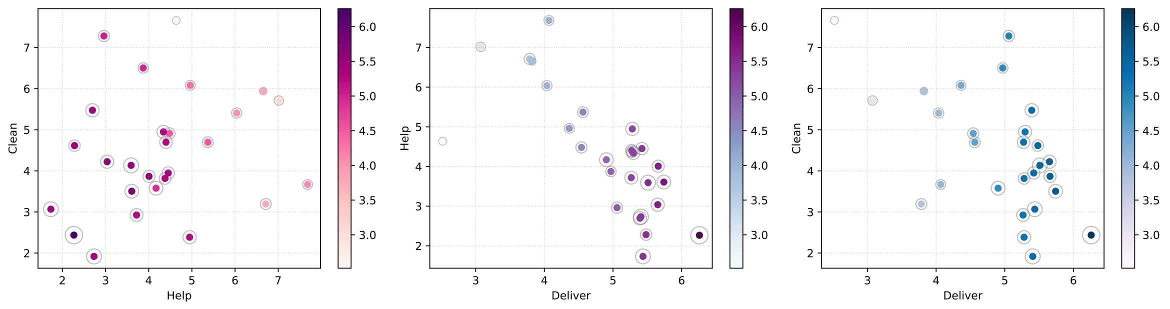


Figure 5.7: The convex coverage set found by MORAL in the case of three reward dimensions. Each plot shows a two-dimensional projection of the attained objectives with colors indicating the third dimension. The radius of gray circles around each policy denotes the proportion of broken vases (on average).

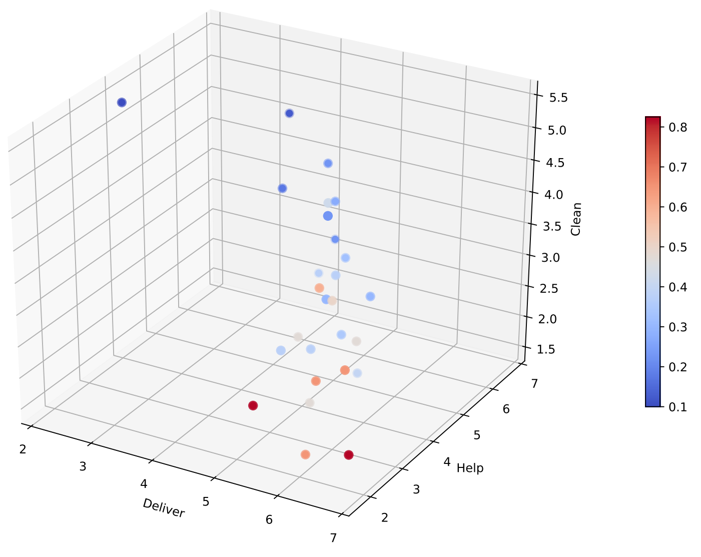


Figure 5.8: Three-dimensional CCS plot in objective space found by MORAL for different preferences. Colors indicate the amount of vases broken on average.

only IRL functions anymore. Also, even though theorem 5.1.2 still holds for arbitrary reward functions r as long as the corresponding maximum entropy distribution $p(\tau|r)$ fulfills the necessary assumption of putting low probability mass on the set of unsafe states, it would be practically impossible to guarantee. While a set of demonstrations can be directly examined a reward function can not, unless an optimal policy with respect to it is found first. The latter is an expensive process, which again highlights as to why designing normative reward functions is naturally intractable for real-world tasks. To better illustrate the Pareto optimality of the found policies, we show a three-dimensional CCS plot in figure 5.8. By definition, Pareto dominance occurs when moving in a non-negative direction with respect to all objectives. From this, we can see that the policies found by MORAL form a diverse set that does contain a substantial amount of Pareto dominated points, further indicating the diversity of policies.

In addition to the previous plots, we also illustrate the convex coverage set in combination with the deviation from the supplemented preference vectors during training in figure 5.9. To be precise, for each point we additionally report its Kullback-Leibler divergence (5.12), where m is chosen as the vector that led to the respective point. Overall, MORAL manages to converge to a distance that is close to zero for most preference vectors, which then directly translates to the observed objective ratios. For example, consider the point closely achieving approximately $(3, 7)$ in the pair (*help*, *clean*). Figure 5.7 indicates that the point achieves around 5 delivered packages, yielding a return vector close to $(3, 5, 7)$. The corresponding unnormalized preference $m = (1, 2, 3)$ resembles a $1 : 2 : 3$ ratio, which the policy

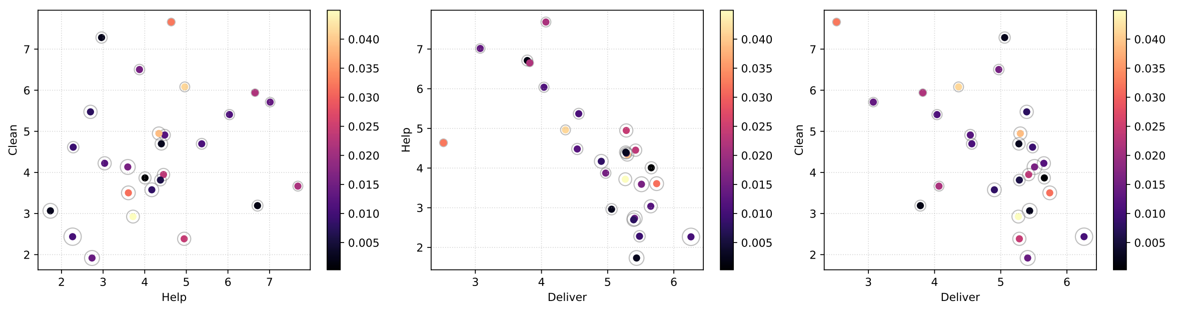


Figure 5.9: The convex coverage set found by MORAL in the case of three reward dimensions. Here, colors indicate the Kullback-Leibler divergences between the attained objective vectors and the preference vector used during training.

approximates reasonably well, thus leading to an overall low Kullback-Leibler divergence. Despite this, MORAL does not always recover the true preferences just as well. Partially, this is due to queries being sampled only heuristically, where the space of available trajectories is always a subset of the agent's current experience. On the one hand, increasing the number of queries can tackle this issue and does decrease the Kullback-Leibler divergence as we will discuss in section 5.2.4. However, due to the inherent noise of MDPs, we can expect such variance even for very large amounts of queries in certain environments. We will discuss when this problem can occur as well as possible solutions in chapter 6.

5.2.3. Learning Multiple Norms from Pairwise Preferences

Similarly to section 4.2.2, we compare the performance of MORAL against DRLHP. To do so, we employ the same model of pairwise preferences (4.15) and train DRLHP for the same amount of steps as MORAL (including the IRL step) using the hyperparameters from table B.7. Unlike before, however, we change the way preferences are given between MORAL and DRLHP. The main reason for this is that by supplying MORAL with a three-dimensional reward function, it only implicitly learns about the vase objective from preferences. In our example, on the other hand, DRLHP needs to be supplied with preferences that take breaking vases into account directly if one aims to embed every objective of the environment into the reward model of the agent. For this reason, we apply a more direct way of supplying preferences to DRLHP. Namely, we first train MORAL and observe its achieved objectives

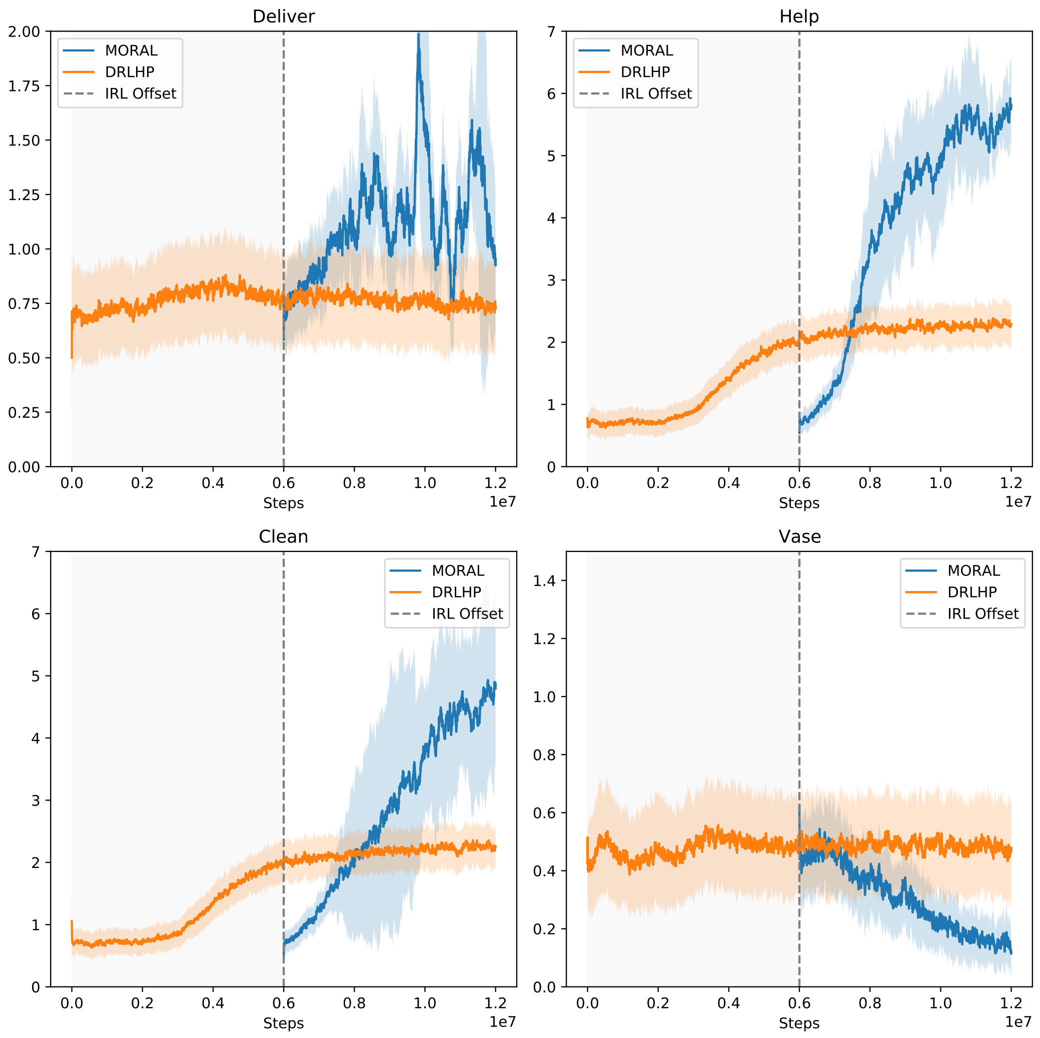


Figure 5.10: Comparison of MORAL and DRLHP in *Delivery* for the preference ratio $(1, 5, 5)$ averaged over three random seeds. The shaded gray area indicates the offset of training steps needed to learn the IRL reward functions in the first step of MORAL.

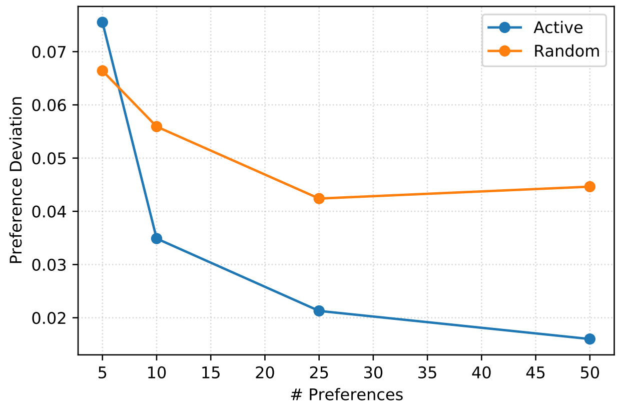


Figure 5.11: Comparison of actively choosing queries versus random samples of trajectory pairs for different amounts of overall retrieved preferences.

$j \in \mathbb{R}^4$. Subsequently, we provide preferences to DRLHP such that we value $\tau_1 \succ \tau_2$, whenever the achieved objectives in τ_1 match j more closely in terms of the mean squared error

$$||j_{\tau_1} - j||_2^2 \leq ||j_{\tau_2} - j||_2^2. \tag{5.13}$$

This way, we train DRLHP more directly to match the returns of MORAL. Figure 5.10 shows the training progress of both algorithms, whereby we offset the training curve of MORAL by the amount of IRL steps needed to learn the respective reward functions. In this environment, we see a similar behavior of DRLHP compared to MORAL as in *Burning Warehouse*. Firstly, MORAL exhibits overall faster convergence than DRLHP in all the objectives of interest. However, as the dimensionality of the reward function increases, the advantage over training speed diminishes due to the sample inefficiency of the AIRL step in MORAL. Nevertheless, we observe that aside from training speed, MORAL outperforms DRLHP in terms of final returns. This is, arguably, one of the main advantages of learning multiple reward functions separately. For example, when training an agent merely from preferences, deep reward models suffer from problems that resemble catastrophic forgetting: Updating using a new pair of preferences can possibly overwrite previously obtained knowledge. While DRLHP tackles this through using an experience buffer of past preferences, this does not avoid smoothing over certain, perhaps less present, objectives in the environment. For example, figure 5.10 shows that although DRLHP manages to learn the importance of *help* and *clean*, it does not manage to avoid vases.

The event of stepping on a vase is comparatively sparse, such that in order to learn about it, the agent needs to first explore sufficiently and pick the correct query. In expectation, a random policy will encounter such trajectories less likely, making it especially rare to occur in the experience buffer. As a consequence, even if there are preferences available that explicitly encode not stepping on vases, updating the deep reward model with a cross entropy loss over a batch of sampled preferences runs at the danger of disregarding vases. Essentially, this is an exploration problem, which DRLHP suffers greatly from. Pretraining DRLHP on a set of demonstrations, as suggested in [47] does help for hard-exploration environments, but does not offer to incorporate demonstrations from a wider set of experts. The intention behind MORAL is to allow for learning from a diverse set of experts, although the final scalarization always needs to be decided on by some expert (or through a voting mechanism). Additionally, MORAL more easily allows for the incorporation of prior knowledge, both in terms of a prior over scalarization weights $p(\mathbf{w})$ as well as in terms of additional manually engineered reward functions. Although the latter is theoretically possible for DRLHP, similar problems arise with regard to appropriately scalarizing the deep reward model (which operates on unknown scales) with any additional reward functions, thus again needing similar solutions to those implemented by MORAL.

5.2.4. Ablation

In this section, we will analyze MORAL with respect to the necessity of active queries, compare the number of queries needed as well as test robustness against noisy preference data. We test the first two criteria jointly, by performing the active learning step for different amounts of queries as well as repeating the procedure using random queries instead. As in section 4.2, random queries are generated by sampling an arbitrary pair of trajectories from the experience buffer of PPO at fixed time intervals. Also, the query frequency as well as overall environment steps are held constant across different runs. Figure 5.11 shows the average preference deviation (5.12) for each configuration. To be precise, we train MORAL to retrieve a CCS using the same set of preference vectors m as in section 5.2.2 and subsequently take the mean over all Kullback-Leibler divergences of found policies to their respective desired objective ratios. Overall, we see a decreasing trend in preference deviation for both active and random queries as the amount of queries increases. Besides this, active learning beats random queries for more than 10 queries by exhibiting less than half of the deviation on average. Interestingly, however, active queries perform worse than random queries when the amount of total queries is very low. We speculate that this is not merely due to noise, since the average is taken over a relatively large set of runs. One reason for active learning to underperform in the small query regime could be due to MORAL optimizing for the posterior mean over scalarization weights. While the maximum volume removal (4.5) is optimal for minimizing the amount of queries needed, it does not guarantee that the posterior mean corresponds to the true weights throughout training. For example, in the case of approximating $\mathbf{w}^* = (1,1,1)/\sqrt{3}$, removing volume at early stages of training will indeed move the posterior mean away from $\mathbf{w}^*$ at first, until enough volume has been removed from all sides of the marginal distributions. On the other hand, random queries will remove less volume on average, thus exhibiting lower variance in the posterior mean for few queries.

Besides this, MORAL beats random queries not only in absolute numbers, but also seems to benefit more strongly from a larger set of queries. We conjecture that this is due to the on-policy sampling of queries from the experience buffer of PPO. By design, active learning will always seek a larger amount of variance in the returns of its respective queries to retrieve the maximal amount of information from the expert. In the large query regime, this helps to reduce volume further as long as the agent maintains adequate levels of exploration. Random queries, on the other hand, are likely to converge to local optima more quickly. Once the agent has found a reasonable strategy for maximizing its current reward model, the policy entropy decreases accordingly. As a result, random queries are more likely to sample from the same set of trajectories. Since the likelihood function (4.12) is log-concave, the entropy of the posterior (4.3) is bound to decrease and, as such, is prone to overfit to the currently observed set of trajectories. MORAL diminishes this effect visibly, but we note that it is not excluded from local optimality of the scalarization posterior. In *Delivery* this does not pose any apparent problem, but we expect the gap between random and active queries to decrease as the marginal reward functions of the environment are more sparse. We will discuss the implications of sparse reward functions in chapter 6.

Finally, we evaluate the robustness of MORAL with respect to preference noise. This is crucial, since MORAL is designed to interact with human experts, whereas our experiments only cover synthetic simulation studies. For this reason, we follow the setup from section 4.2.1 and introduce noisy preferences with some probability p that randomly choose between a presented pair of trajectories. Figure 5.12 shows average preference deviations across different configurations of m for $n = 50$ overall queries per run. Our experiments suggest that despite the increase in preference deviation, the relative errors are small enough to outperform random queries without any preference noise, which exhibits an error of around 0.045 when using 50 queries. Nonetheless, there is a significant increase in error when jumping to noise levels of 0.1. Again, we conjecture that this is due to the nature of our Bayesian learning procedure constantly removing entropy from the posterior, thus running the risk of converging to local optima when receiving highly contradicting evidence. One possible mitigation strategy would involve the introduction of a hyperparameter by which the reward function is scaled, in order to decrease the sensitivity of each Bayesian update step at the cost of needing more queries to converge. We suggest that further research with human experts in the loop would be necessary in order to study the necessity of such a modification to MORAL. Overall, we conclude that these experiments show that although the performance of MORAL drops in the presence of noisy preferences, we still achieve a more accurate reward posterior through active learning when compared to random queries without added noise.

5.3. Conclusion

Overall, in this chapter we have illustrated how MORAL can be employed for incorporating a normative prior from different experts into a reward-driven deep RL agent. Firstly, we have shown that MORAL

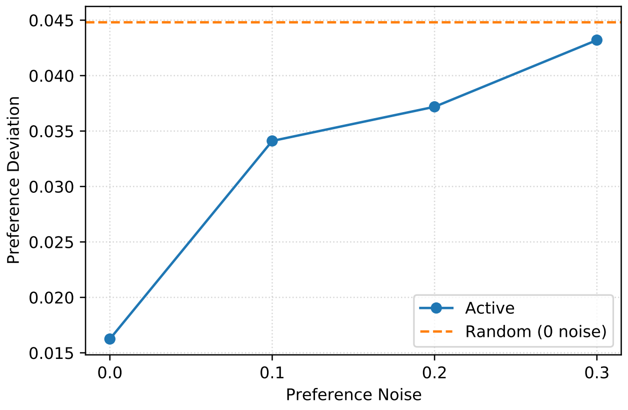


Figure 5.12: Robustness of MORAL with respect to preference noise. Preference deviations are averaged over different runs with the preference vector m being varied accordingly.

is closely related to Harsanyi's utility aggregation theorem, which draws connections to the field of computational social choice. Furthermore, we have shown that linearly aggregating a set of maximum entropy IRL reward functions can be interpreted as finding the policy minimizing a weighted Kullback-Leibler average to the corresponding marginal maximum entropy IRL distributions over trajectories. Subsequently, we have introduced the *Delivery* environment, in which we have shown the ability of MORAL to efficiently recover a diverse set of Pareto optimal policies that extrapolate beyond those contained in the demonstration dataset. Finally, aside from a comparison to DRLHP and manually choosing scalarization weights, we have demonstrated that MORAL scales well with the number of provided preferences as well as outperforms randomly choosing queries even in the presence of additional preference noise.

6 Discussion

In our work, we proposed multi-objective reinforced active learning (MORAL), an algorithm for incorporating and combining multiple expert's utility functions into a deep RL agent. MORAL offers a technical approach to the value alignment problem, which allows for value open design by means of learning reward functions from a diverse set of demonstrations. Furthermore, we argued that a major challenge of value alignment lies in the specification of human norms and showed that MORAL can be used for incorporating normative behavior into goal-driven agents. To do so, we designed two environments, *Burning Warehouse* and *Delivery*, which test an agent's capability of adapting its primary goal to account for different normative components. Overall, we empirically demonstrated that MORAL is able to efficiently find a diverse set of Pareto optimal policies through the specification of different preferences and, additionally, outperforms single-objective approaches such as deep reinforcement learning from human preferences.

6.1. Key Findings

The key findings of our research are tightly interconnected with the proposed research questions from section 1.2. We will start by providing an outline of the main contributions to the first research question.

1. How can we query and interact with experts to elicit normative behavior in sequential decision-making problems?

Eliciting normative behavior from trajectories in a sequential decision-making context is not only challenging due to the exponentially growing size of possibilities in large environments. From a practical point of view, its difficulty can also be greatly ascribed to the granularity at which decisions are made in Markov decision processes. We explored how to utilize multiple sources of inputs including expert demonstrations and preferences and showed that through MORAL, using preferences to tune reward functions learned through inverse reinforcement learning is effective for a multi-objective environment. In comparison, we tested MORAL against learning a deep reward model from preferences only and found the latter to lack both adaptivity and optimality. Besides this, we found the separation of learning from demonstrations and learning from queries especially useful from a safety perspective. Preferences only weakly denote goal specifications in sequential decision-making, thus requiring large amounts of data to ensure that the agent indeed optimizes the desired utility function. On the other hand, demonstrations provide a relatively strong feedback signal from which numerous values can be inferred at the cost of generalization capabilities. In combination, we showed that pretraining on diverse demonstrations allows the agent to learn about both explicit and implicit preferences of the expert which can greatly reduce the space of possible reward functions to a safer subset of reward functions. Preferences are then only used for maintaining a probability distribution over generally desirable reward functions, thus making the algorithm robust to malicious feedback signals in the second stage.

2. How can we encode learned normative behavior into RL agents?

Another challenge arising from the granularity at which RL agents operate is learning hierarchical reward functions. Naturally, information from demonstrations as well as preferences can only either recover shaped or highly sparse rewards. Without any additional prior knowledge, the agent must assume that either each subtrajectory of preferred or demonstrated trajectories yields progress towards the overall goal (shaped) or that only after completion of the whole trajectory one has achieved high utility (sparse). While the latter offers superior transfer learning capabilities, it can hinder the ability to find intermediate solutions that trade off multiple rewards. For example, in *Delivery*, some humans need help by the agent alongside a primary goal of delivering packages. Learning a reward function from demonstrations that helps all humans can then either correlate positive reward with each person that has been helped or, in the sparse case, aim to only reward the agent once all people have been satisfied. In this case, the shaped reward function offers more flexibility with respect to trading off the goal of helping people with delivering packages. It is therefore crucial to take the shaping of rewards into account when aiming to incorporate human value preferences into RL agents. In chapter 3, we showed that when dealing with approximate (and shaped) reward functions, the framework of multi-objective RL offers a fundamental advantage over constrained RL, even when one of the rewards is more naturally encoded in the form of a constraint.

3. How can we combine different (and possibly conflicting) aspects of normative behavior in sequential decision-making problems?

The aggregation of human value preferences is naturally tackled by the field of computational social choice. However, in the context of RL the set of alternatives over which choices should be made is too large a-priori to deliberate over. MORAL first learns multiple expert's utility functions and combines them through active learning. This allows an efficient search over the space of Pareto optimal trajectories by means of pairwise preferences. In chapter 5 we showed how MORAL relates to Harsanyi's utility aggregation theorem and proved that, under the assumption of a convex Pareto boundary, solutions are given by policies that minimize a weighted Kullback-Leibler average between maximum entropy inverse RL distributions. Furthermore, we empirically showed that MORAL is able to combine normative behavior from different experts and can recover a wide variety of preferences with only few queries. Overall, this suggests that MORAL can generalize beyond expert demonstrations and build combined policies that achieve a desired trade-off between the observed utility function.

6.2. Related Work

Inverse Reinforcement Learning. Our work builds on the framework of maximum entropy IRL [101] for deep RL agents by approximating the objective function using generative adversarial networks analogous to adversarial inverse reinforcement learning (AIRL) [31]. AIRL studies how to transfer a single learned reward signal, while our work focuses on trading off different learned reward functions in the same environment. Learning from multiple demonstrators has been mostly studied from the perspective of multi-agent [99] or multi-task learning by imitating experts through latent variable models [43, 56], modelling hierarchical behavior [78, 90] or learning a latent-conditioned reward function [100] in an unsupervised manner. On the other hand, [33, 96] considers meta learning a reward function that can quickly adapt to new tasks from supervised data. While we also assume labeled data for multiple tasks, we are applying AIRL in a multi-objective Markov decision process. To our knowledge, this has not yet been studied before.

Maximum entropy IRL models the demonstrations as Boltzmann rational, which can inherit fundamental biases from the data as well as generalize badly. To overcome bad learned rewards, cooperative inverse reinforcement learning (CIRL) provides ways of challenging this by changing the modelling assumption about the human intentions [39]. Similarly, inverse reward design (IRD) [40] learns a distribution of reward functions through IRL that can leverage its uncertainty to help avoiding unintended behavior not originally specified by a reward designer. We also learn a distribution over reward functions, but unlike our approach IRD and CIRL provide a formal model of reward learning from a single expert, whereas we are interested in combining reward from multiple sources. As a result, our work is conceptually more close to multi-task inverse reward design [52], which studies how a distribution over reward functions can be obtained under conflicting inputs. Multi-task IRD satisfies formal desiderata for value-aligned combinations of reward functions by assuming a tabular feature space and a linear reward as a function of feature expectations. On the other hand, our approach readily drops these assumptions, allowing to combine multiple reward functions with deep inverse reinforcement learning, but comes at the cost of less formal guarantees.

Learning from Expert Feedback. Instead of finding reward functions from demonstrations, one can train agents by building a model of an expert's utility from scalar feedback [50, 91] and preferences [94]. Preference based reinforcement learning has been shown to yield promising results in Atari and simulated robotics by training a deep reward model from human preferences [23] and a combination of preferences and demonstrations [47]. Similarly to [47], we combine preference and demonstration data to learn a reward function. However, [47] uses demonstrations merely for pretraining a preference-based reward model whereas our approach applies IRL directly for finding a multivariate reward function and only uses preferences for maintaining uncertainty over the respective reward components. Besides that, we are concerned about learning from multiple different experts, whereas [47] tackles the problem of learning from a single expert. Finally, by combining multiple learned reward functions, we implicitly interpolate between the distributions over expert trajectories. From this angle, our work could be considered a counterpart to [19], which trains a single reward model from ranked demonstrations to extrapolate beyond the expert behavior.

Machine Ethics. Formally tackling how ethical behavior can arise in RL agents has been evaluated in [3] by modelling the uncertainty over human values through partial observability. Motivated by the problem of formally specifying human values, learning based approaches have made use of inverse reinforcement learning [63] and reward shaping from demonstrations [95]. In contrast to [63] and [95], we make use of deep RL agents to overcome the fundamental scalability issues introduced by using handcrafted features spaces. Although [95] argues against the use of IRL, we empirically show that similar results can be obtained with fewer assumptions by using AIRL as an approximate minimizer of a Kullback-Leibler divergence to the expert policy. To learn from different experts, [28] suggest a sequential voting mechanism, but it comes at the disadvantage of needing explicit representations of the values at stake. We do not explicitly trade off values by voting, but rather train an agent that can adapt to different values when given corresponding preferences. This somewhat resembles the multi-agent work by [20], where symbolic values are learned to be adapted to by aggregating judgements of different moral agents, with our approach employing a bottom-up approach of learning the values instead.

Multi-Objective Reinforcement Learning. Trading off multiple objectives within a single reward function is the study of MORL [69]. Similarly to [97], we assume finding policies on the convex coverage set of the Pareto boundary by sampling various preference vectors. However, we do not optimize for an explicit set of policies, but maintain a single policy that is adapted to the posterior mean of an actively learned distribution over reward functions. Besides sample efficiency, this is aimed at mitigating the difficulty of correctly specifying preferences over reward functions with fundamentally varying scales. Multi-objective maximum a posteriori policy optimization (MO-MPO) [2] removes scale sensitivity by satisfying respective Kullback-Leibler thresholds for policy improvement of each respective objective. Although effective, their approach does not allow for learning a preference vector interactively, since the the specified Kullback-Leibler distances only indirectly affect the obtained rewards.

Interactively learning the desired trade-offs from feedback has been previously studied in the context of multi-objective bandits for linear [70] as well as nonlinear [71] transformations of the marginal reward components. Besides that, interactively learning multi-objective reward functions has been mostly neglected in the literature. We believe that this is because the MORL literature always assumes environments with handcrafted reward functions, in which large parts of the Pareto boundary are of potential interest. In the case of aligning an agent to human preferences, however, we suggest human in the loop procedures for finding aligned reward functions to be a more promising option, analogous to the reward learning procedure of preference based RL [23].

We train agents in two consecutive steps, where first multiple rewards are learned and subsequently combined with a multi-objective procedure. This resembles the approach by [63], where a manually designed reward function and an IRL reward are being traded off with linear preference weights. However, they do not explicitly consider a multi-objective algorithm, but rather treat preferences as a fixed hyperparameter that determines the final policy. From the perspective of mitigating negative side effects, [75] propose trading off learned rewards with a formal goal by calculating a specified maximum amount of deviation from the primary objective, but the need for prespecifying a preference parameter persists. Finally, [25] considers how (potentially learned) utilities from two different agents should be aggregated to achieve Pareto optimality. They show that when agents have fundamentally different beliefs about the environment, a Pareto optimal aggregation procedure cannot be given by a linear combination of their respective utility functions. For the sake of tractabilty, we violate this assumption and assume a fully observable environment where each agent from which utilities are derived shares common beliefs about the problem setting.

6.3. Discussion

Our findings provide first steps towards incorporating and aggregating normative behavior from demonstrations in deep RL agents. However, although our research builds on previous work, many avenues of future research remain to be addressed. To outline this, we first summarize the main gaps that MORAL fills when compared to related work in table 6.1. Firstly, our approach most directly builds on policy orchestration for teaching AI ethical values [63] and extends it to deep RL by employing AIRL for learning reward functions. However, instead of orchestrating marginal imitation policies, we instead learn a new policy that optimizes linear combinations of IRL reward functions. We found this to be crucial for achieving acceptable performance in the case of multiple experts due to accumulating errors in the respective policies obtained from IRL. Nevertheless, we note that policy orchestration builds on contextual bandits, which can not be as easily translated into the deep setting as IRL. Besides this, both approaches are able to outperform experts to some degree, due to maximum entropy IRL assuming only approximate optimality of the expert demonstrations.

Ethics shaping [95] conceptually compares to MORAL in that it steers RL agents following a primary goal to respect additional norms from demonstrations. Similarly to policy orchestration, this does unfortunately not scale to complex environments where feature engineering is not possible. Arguably, ethics shaping retrieves sparse feedback signals from demonstrations which is its strongest advantage over MORAL and allows for learning temporally complex norms. However, by manually adding a shaping term to the reward, multiple experts as well as multi-objective decision are not taken into account, thus eliminating the possibility of aggregating diverse utility functions.

While ethics shaping and policy orchestration can not easily be compared to MORAL due to different assumptions about the environment, we used deep reinforcement learning from human preferences (DRLHP) [23] as a direct comparison to MORAL. DRLHP skips the IRL stage and directly employs a Bradley-Terry model of pairwise preferences for learning a deep reward model and has been shown to scale remarkably well to high-dimensional environments. As such, DRLHP provides an efficient solution to the problem of encoding human value preferences into deep RL agents without the need of ever demonstrating any policy. MORAL, on the other hand, requires more supervision in the form of demonstrations but enables one to easily take different preferences as well as primary reward functions into account. As such, MORAL should not be viewed as a superior approach to DRLHP, but rather as an algorithm that offers more flexibility at the cost of more diverse supervision.

Combining multiple forms of supervision is challenging, as different probabilistic models about an expert's utility function or preferences that each come with their own assumptions about expert rationality need to be trained. Arguably, the main drawback of MORAL is its reliance on AIRL for learning multiple reward functions, which lacks sample efficiency and often converges to badly shaped reward functions. Similarly to [33], we found the reoptimization of reward functions obtained through AIRL to be challenging in discrete environments, which indicates overfitting of the reward network to the generator policy. Since MORAL relies on a two-step procedure in which the learned marginal reward functions are fixed in the multi-objective optimization stage, it is prone to inaccurate AIRL reward functions. Alleviating this issue will require substantially more progress in IRL, but we expect future developments in IRL will be easily integrated into our current framework.

	Ethics Shaping [95]	Policy-Orchestration [63]	DRLHP [23]	MORAL
Function Approximation	✗	✗	✓	✓
Multi-Objective	✗	✓	✗	✓
Multiple Experts	✗	∼	✗	✓
Outperform Experts	✓	✓	✓	✓

Table 6.1: Comparison of MORAL to previous work in terms of supported capabilities.

In addition to the development of new IRL algorithms, we consider future research in active learning for multi-objective RL to be a promising avenue for training tunable agents. First of all, our research only studied MORAL in discrete grid world environments with synthetic demonstrations and preferences. However, recent research has shown that synthetic demonstrations can impact the performance of AIRL [64] and we expect synthetic preferences to have a similar positive impact on the overall performance of MORAL. Although our experiments suggest that introducing artificial noise to preference data does not greatly impact the ability of MORAL to retrieve accurate reward distributions, human feedback is generally more unstructured. It would therefore be valuable to study the degree to which active learning for multi-objective RL suffers from human feedback. A related, yet different problem is that of convergence to a local optimum of the preference model. Since the likelihood function that updates the posterior over scalarization weights is log-concave, the entropy of the distribution over reward functions decreases over time. As a consequence, adequate levels of exploration by the agent have to be maintained in order to avoid local optimality, since queries are sampled from on-policy RL experience. While we found entropy regularization to be sufficient in our densely populated grid world environments, we expect different exploration schemes to yield significantly better performance in sparse environments. For example, one could add an exploration policy that tries to actively synthesize queries with maximum volume removal. This way, whenever the current policy is optimizing for a locally optimal source of reward, the exploration policy is able to generate new queries which reveal the suboptimality of the agent's most recent behavior.

Finally, we consider additional research investigating unsupervised RL techniques to be relevant for extending the MORAL framework to unlabelled demonstration datasets. Real world datasets might contain large amounts of demonstrations coming from a wide variety of people. Depending on the context, it is reasonable to assume that many demonstrations will contain significant overlap, which could be detected and exploited for meta learning multiple reward functions, similarly to [100] without the need of additional labels. On top of that, multi-task neural network architectures could additionally increase the sample efficiency of IRL which would be especially necessary when the amount of reward functions to be learned is high.

II

Scientific Papers

Training for Implicit Norms in Deep Reinforcement Learning Agents through Adversarial Multi-Objective Reward Optimization

Markus Peschl
Department of Intelligent Systems
Delft University of Technology
Delft, The Netherlands
m.peschl@student.tudelft.nl

ABSTRACT

We propose a deep reinforcement learning algorithm that employs an adversarial training strategy for adhering to implicit human norms alongside optimizing for a narrow goal objective. Previous methods which incorporate human values into reinforcement learning algorithms either scale poorly or assume hand-crafted state features. Our algorithm drops these assumptions and is able to automatically infer norms from human demonstrations, which allows for integrating it into existing agents in the form of multi-objective optimization. We benchmark our approach in a search-and-rescue grid world and show that, conditioned on respecting human norms, our agent maintains optimal performance with respect to the predefined goal.

CCS CONCEPTS

• **Computing methodologies → Inverse reinforcement learning**; **Adversarial learning**.

ACM Reference Format:
Markus Peschl. 2021. Training for Implicit Norms in Deep Reinforcement Learning Agents through Adversarial Multi-Objective Reward Optimization. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society (AIES '21), May 19–21, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3461702.3462473

1 INTRODUCTION

Recent progress in reinforcement learning (RL) has vastly increased the feasibility and applicability of artificially intelligent agents to real world problems by employing deep neural networks that approximately solve high-dimensional control tasks. However, most state of the art algorithms are inherently black-box models that optimize for a specific manually engineered reward function. This can lead to unforeseen societal impacts which need to be accounted for before training and deploying the system [7].

As a consequence, there is a need for taking human values into account when training RL systems, which has previously been tackled by various approaches including inverse RL [4], reward shaping [8], partially observable Markov decision processes [1], safe RL [5] and multi-agent social choice [2]. However, most of these approaches do not scale to continuous or complex discrete environments due to assuming a tabular representation of the state space or hand-crafted features. While Saunders et al. [5] do not require any assumptions about the state space by employing deep RL, their approach only considers immediate negative consequences that can be avoided by blocking a single action.

To tackle the issue of scalability, we propose a hybrid architecture that enables deep reinforcement learning agents to optimize for a predefined goal while adhering to implicit norms learned from human behavior. Furthermore, by employing a bottom-up approach which builds on the adversarial inverse reinforcement learning framework [3], our agent is able to adapt to temporally complex constraints which can not directly be achieved through action blocking.

2 METHODS

We cast the problem of optimizing for a predefined goal while adhering to human norms into a multi-objective Markov decision process (MOMDP) with linear preferences, which is given by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathbf{r}, \boldsymbol{\lambda} \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ denote the sets of possible states and actions respectively, $\mathcal{P}(s'|s,a)$ denotes the state transition probability function and $\mathbf{r}(s,a) \in \mathbb{R}^d$ is a vector-valued reward function. Finally, we consider preference vectors $\boldsymbol{\lambda} \in \mathbb{R}^d$ to determine preferences among competing objectives. The RL goal then consists of finding a policy $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$ that maps states to a probability distribution over actions which maximizes the expected cumulative reward $\max_\pi \mathbb{E}_\pi \left[\sum_{t=0}^T \gamma^t \boldsymbol{\lambda}^T \mathbf{r}(s_t, a_t)\right]$, where $0 < \gamma \leq 1$ is a temporal discount factor.

We assume that some components of $\mathbf{r}(s,a) \in \mathbb{R}^d$ are known in advance, these correspond to the primary goal that the agent ought to maximize, whereas norms represent the other components. To learn norms, we make use of human demonstrations which we assume to be norm satisfying and infer corresponding reward signals with adversarial inverse reinforcement learning (AIRL) [3]. AIRL trains a discriminator of the form

$$D_\theta(s,a) = \frac{\exp f_\theta(s,a)}{\exp f_\theta(s,a) + \pi(a|s)}, \tag{1}$$

which outputs the probability of the state-action pair (s,a) coming from the dataset $\mathcal{D} = \{\tau_i\}_{i=1}^n$ of human demonstrations rather than from an agent following the policy π. Simultaneously, the policy π is updated to match the demonstrations in $\mathcal{D}$ more closely.

Assuming that we want the agent to optimize a primary goal given by rewards $r_0(s, a)$, we then use a modified update rule for the generator π for maximizing

$$\mathbb{E}_\pi \left[\sum_{t=0}^{T} \lambda(f_\theta(s_t, a_t) - \log \pi(a_t|s_t)) + (1-\lambda) r_0(s_t, a_t) \right]. \quad (2)$$

Mathematically speaking, this results in π optimizing for r_0 while regularizing by a Kullback-Leibler divergence term $KL(\pi(\tau)||p_\theta(\tau))$, where $p_\theta(\tau) \propto \mathcal{P}(s_0) \prod_{t=0}^{T} \mathcal{P}(s_{t+1}|s_t, a_t) e^{\gamma^t f_\theta(s_t, a_t)}$ is the maximum entropy inverse RL probability distribution induced by f_θ [9].

3 EXPERIMENTS

We train a deep reinforcement learning agent in a (stochastic) grid world, with state inputs being a numerical matrix representation encoding the states of each cell. Both the policy and the discriminator employ a three-layer convolutional neural network, followed by linear output layers respectively. Furthermore, the policy is trained via proximal policy optimization (PPO) [6] due to its ease of implementation.

Testing is done in the *burning warehouse* environment (Figure 1), where the primary goal for the agent is to spend most of its time in a specific tile G. In general, G can be understood as any primary goal of interest, such as calling a fire department, extinguishing fire or other relevant tasks. Besides G, there are workers in the building which move in any of the four directions at each time step. We assume that these workers are lost and they need to be picked up by the robot in order to successfully escape the building. The agent can achieve this by moving onto their respective positions before the episode ends after $T = 100$ time steps.

To enable learning norm-satisfying behavior, we supply the agent with $n = 10$ human demonstrations which primarily move towards lost workers and neglect the goal tile G. We then train multi-objective AIRL for different values of λ, with $1e6$ total environment steps each. For example, for $\lambda = 0.1$ the agent is able to optimize for the primary goal while adhering to the demonstrated norms (Figure 2). These preliminary results demonstrate that our agent learns to combine the norms inferred from human demonstrations (saving people) with the reward-driven behavior (spending time in G).

4 DISCUSSION & FUTURE WORK

One important aspect of our approach is that it does not require human demonstrations to optimize for the primary goal in any form. This way, we can utilize the strengths of traditional RL when a reward function is available, while ensuring that certain constraints, which might not be easily expressed in a reward function, are met. While in our example, the goal of saving humans could in theory be expressed with a reward function, our experiment showed that even when this is not the case, we can optimize for human constraints by having access to a small set of demonstrations. Furthermore, it directly extends the settings of Noothigattu et al. [4] and Wu, Lin [8] to deep RL, dropping the assumption of a handcrafted feature space. Nonetheless, a variety of open questions remain to be addressed in future work. Firstly, we assumed demonstrated norms to be consistent with each other. However, in real world scenarios this might not be the case, which would lead to inaccurate reward representations learned by AIRL. Secondly, the adaptation of multi-objective RL algorithms to allow for Pareto-efficient policy search with online reward learning will be of interest for scaling to more complex domains. Specializing AIRL to efficiently allow for learning multiple conflicting norms will therefore be the focus of following work.

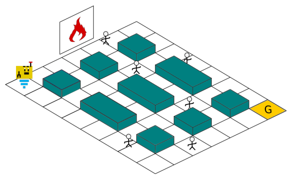


Figure 1: The burning warehouse toy environment.

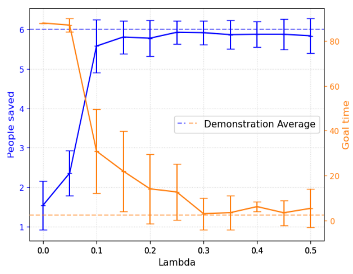


Figure 2: People saved (blue) and goal time (orange) for different values of λ. Error bars indicate a 2σ confidence interval over 100 distinct episodes.

ACKNOWLEDGMENTS

To Luciano Cavalcante Siebert and Arkady Zgonnikov for providing critical feedback and insightful supervision of this research.

REFERENCES

[1] David Abel, James MacGlashan, and Michael Littman. 2016. Reinforcement Learning as a Framework for Ethical Decision Making. In *AAAI Workshop: AI, Ethics, and Society.*

[2] Adrien Ecoffet and Joel Lehman. 2020. Reinforcement Learning Under Moral Uncertainty. arXiv:2006.04734

[3] Justin Fu, Katie Luo, and Sergey Levine. 2017. Learning Robust Rewards with Adversarial Inverse Reinforcement Learning. arXiv:1710.11248

[4] Ritesh Noothigattu, Djallel Bouneffouf, Nicholas Mattei, Rachita Chandra, Piyush Madan, Kush R Varshney, Murray Campbell, Moninder Singh, and Francesca Rossi. 2019. Teaching AI agents ethical values using reinforcement learning and policy orchestration. *IBM Journal of Research and Development* 63, 4-5 (2019). https://doi.org/10.1147/JRD.2019.2940428

[5] William Saunders, Andreas Stuhlmüller, Girish Sastry, and Owain Evans. 2018. Trial without error: Towards safe reinforcement learning via human intervention. In *Proceedings of the International Joint Conference AAMAS*, Vol. 3. 2067–2069. arXiv:1707.05173

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347

[7] Jess Whittlestone, Kai Arulkumaran, and Matthew Crosby. 2021. The Societal Implications of Deep Reinforcement Learning. *Journal of Artificial Intelligence Research* 70 (March 2021). https://doi.org/10.1613/jair.1.12360

[8] Yueh Hua Wu and Shou De Lin. 2018. A low-cost ethics shaping approach for designing reinforcement learning agents. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018.* 1687–1694.

[9] Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. 2008. Maximum entropy inverse reinforcement learning. In *Proceedings of the National Conference on Artificial Intelligence*, Vol. 3. 1433–1438.

MORAL: Aligning AI with Human Norms through Multi-Objective Reinforced Active Learning

Markus Peschl
Delft University of Technology
Delft, The Netherlands
peschl@protonmail.com

Arkady Zgonnikov
Delft University of Technology
Delft, The Netherlands
A.Zgonnikov@tudelft.nl

Frans A. Oliehoek
Delft University of Technology
Delft, The Netherlands
F.A.Oliehoek@tudelft.nl

Luciano C. Siebert
Delft University of Technology
Delft, The Netherlands
L.CavalcanteSiebert@tudelft.nl

ABSTRACT

Inferring reward functions from demonstrations and pairwise preferences are auspicious approaches for aligning Reinforcement Learning (RL) agents with human intentions. However, state-of-the art methods typically focus on learning a single reward model, thus rendering it difficult to trade off different reward functions from multiple experts. We propose Multi-Objective Reinforced Active Learning (MORAL), a novel method for combining diverse demonstrations of normative behavior into a Pareto optimal policy. Through maintaining a distribution over scalarization weights, our approach is able to interactively steer a deep RL agent towards a variety of norms while eliminating the need for computing multiple policies. We empirically demonstrate the effectiveness of MORAL in two gridworld scenarios, which model a delivery and an emergency task that require an agent to act in the presence of normative conflicts. Overall, we consider our research a first step towards multi-objective RL with learned rewards, bridging the gap between current value learning and machine ethics literature.

KEYWORDS

Active Learning; Inverse Reinforcement Learning; Multi-Objective Decision-Making; Value Alignment

1 INTRODUCTION

The design of adequate reward functions poses a tremendous challenge for building reinforcement learning (RL) agents that ought to act in accordance with human intentions [4, 13]. Besides complicating the deployment of RL in the real world [11], this can lead to major unforeseen societal impacts, which need to be accounted for when building autonomous systems [6, 43]. To tackle this, the field of value alignment has largely focused on value learning, which aims to adopt a bottom-up approach of learning goal specifications from observational data instead of manually specifying them [22, 29, 38]. However, such technical approaches can not solely solve the normative value alignment problem of deciding which values should ultimately be encoded into an agent [15]. Nonetheless, building methods that allow for learning and trading off different conflicting values could potentially alleviate this, thus making them an important avenue of research for beneficial artificial intelligence (AI) [33].

In a sequential decision-making context, jointly optimizing for opposing criteria can be cast into multi-objective RL (MORL) [30], which constitutes a promising framework for building human aligned AI [40]. Using game-theoretic notions of optimality, MORL typically aims to find a solution, or a set thereof, that can represent a wide variety of preferences over the components of a vector-valued reward function. While this can theoretically tackle the overoptimization of narrowly defined tasks, the need for specifying multiple reward functions persists.

Inverse RL (IRL) [16, 49] and preference-based RL [10, 44] offer techniques for avoiding the reward design problem altogether by learning a parametric reward model from demonstrations and pairwise preferences, respectively. In this paper, we combine these approaches in a multi-objective setting with a focus on learning human norms. The motivation for this is twofold: Firstly, previous research that aims to learn multiple reward functions has mostly employed latent variable IRL models for finding multiagent [19], hierarchical [39, 41] and multitask [17, 48] rewards, whereas finding aggregated rewards from conflicting sequential data has yet to be addressed. Secondly, a major challenge of value alignment is given by an agent's ability to predict a normative structure in its environment, which is implicitly embedded in human goal specifications, but missing in manually engineered reward functions [20]. Our goal therefore is to find a policy that acts on a common set of normative actions while allowing for fine-tuning the agent with respect to inherent disagreements that may arise. One straight forward approach to achieve this, would be to apply multitask IRL on labelled demonstration data and then use MORL on the obtained vector-valued reward function. However, this turns out to be inefficient when dealing with deep reward models trained from noisy data. This is because conventional MORL approaches are sensitive to the scale of the marginal reward components. Furthermore, explicitly expressing preferences over learned objectives is complicated due to a lack of interpretability.

Contributions (1) We propose Multi-Objective Reinforced Active Learning (MORAL), a method that combines active preference learning and IRL to interactively learn a policy of normative behavior from expert demonstrations. MORAL first finds a vector-valued reward function through adversarial IRL, which is subsequently used in an interactive MORL loop. By providing pairwise preferences over trajectories of on-policy experience, MORAL learns a probability distribution over linear combinations of reward functions under which the optimal policy most closely matches the desired behavior. (2) We show that our approach directly approximates a Pareto optimal solution in the space of expert reward functions, without the need of enumerating through a multitude of preference weights. (3) We demonstrate that MORAL efficiently captures normative behavior in two gridworld scenarios, while being able to adapt the agent's behavior with respect to a variety of preferences.[1]

2 METHOD

2.1 Preliminaries

Multi-Objective RL. We employ a multi-objective Markov decision process (MOMDP) for framing the problem of aligning an agent with different experts. Formally, a MOMDP is given by the tuple $\langle \mathcal{S}, \mathcal{A}, p, \mathbf{r}, \mu_0, \gamma \rangle$, with state space $\mathcal{S}$, the set of actions $\mathcal{A}$, a transition distribution $p(s'|s,a)$, a vector-valued reward function $\mathbf{r}(s,a) \in \mathbb{R}^m$, a starting state distribution μ_0 and the discount factor $\gamma \in [0,1)$. We consider optimal solutions to be given by a Pareto frontier $\mathcal{F} := \{\pi | \nexists \pi' \neq \pi : J_{\mathbf{r}}(\pi') \geq J_{\mathbf{r}}(\pi)\}$, where $J_{\mathbf{r}}(\pi) = \mathbb{E}_\pi[\sum_{t=0}^T \gamma^t \mathbf{r}(s_t, a_t)]$ is the vector-valued return of a policy $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$ that maps states to a distribution over actions. Furthermore, we define the convex coverage set (CCS) $\mathcal{F}^* := \{\pi \in \mathcal{F} \mid \exists \mathbf{w} \in \mathbb{R}^m : \mathbf{w}^T J_{\mathbf{r}}(\pi) \geq \mathbf{w}^T J_{\mathbf{r}}(\pi'), \forall \pi' \in \mathcal{F}\}$ to be the subset of Pareto optimal solutions that can be obtained through optimizing for linear combinations of the different reward components.

Proximal Policy Optimization (PPO). Given a weight $\mathbf{w} \in \mathbb{R}^m$, we can optimize for policies on the CCS using PPO [37] on the scalarized reward $r(s,a) = \mathbf{w}^T \mathbf{r}(s,a)$. Using on-policy experience, PPO maximizes the return of a parametrized policy π_ϕ by performing gradient descent on the clipped objective

$$\mathcal{L}^{\text{CLIP}}(\phi) = \mathbb{E}_t[\min(r_t(\phi)\hat{A}_t, clip(r_t(\phi), 1-\epsilon, 1+\epsilon)\hat{A}_t)],$$

where $\mathbb{E}_t$ is the expectation at time t, r_t is a ratio of the new versus the current policy, $\hat{A}_t$ is an estimated advantage at time t and $clip(x,a,b)$ limits the value of x to the interval $[a,b]$.

Adversarial IRL (AIRL). The maximum entropy IRL [49] goal is to derive a reward function r_θ from a demonstration dataset $\mathcal{D} = \{\tau_i\}_{i=1}^N$ of expert trajectories $\tau = \{s_t, a_t\}_{t=0}^T$ by solving a maximum likelihood problem $\max_\phi \mathbb{E}_{\tau \sim \mathcal{D}}[\log p_\theta(\tau)]$, where

$$p_\theta(\tau) \propto \mu_0(s_0) \prod_{t=0}^T p(s_{t+1}|s_t, a_t) \exp(r_\theta(s_t, a_t)) =: \overline{p}_\theta(\tau).$$

AIRL [14] approximately solves the IRL problem using generative adversarial networks [18]. It jointly trains a policy (generator) π_ϕ alongside a neural discriminator of the form

$$D_\theta(s,a) := \frac{\exp(f_\theta(s,a))}{\exp(f_\theta(s,a)) + \pi_\phi(a|s)}.$$

While D_θ is trained using a binary cross-entropy loss to distinguish trajectories in $\mathcal{D}$ from π_ϕ, the agent maximizes its returns using the reward $r(s,a) := \log D_\theta(s,a) + \log(1 - D_\theta(s,a))$.

[1]Source code is available at https://github.com/moral-rl/moral_rl.

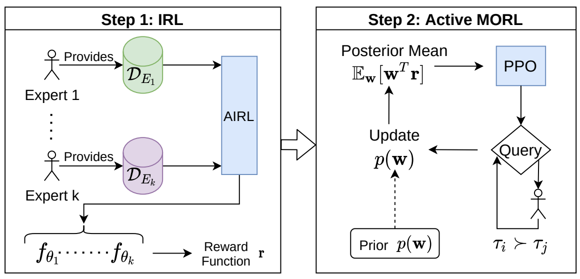


Figure 1: Multi-Objective Reinforced Active Learning.

2.2 Multi-Objective Reinforced Active Learning

To learn from multiple experts, MORAL uses a two-step procedure that separates reward and policy training, as illustrated in figure 1. Assuming a dataset $\mathcal{D}_E = \cup_{i=1}^k \mathcal{D}_{E_i}$ from k distinct experts, MORAL first uses AIRL to obtain a vector of reward functions $\mathbf{r} = (f_{\theta_1}, \ldots, f_{\theta_k})$ and imitation policies $(\pi^*_{E_1}, \ldots, \pi^*_{E_k})$ by solving the maximum entropy IRL objective for each set $\mathcal{D}_{E_i}$. Subsequently, we employ an interactive MORL algorithm for learning a distribution over weights $p(\mathbf{w})$ that determine a linear combination of the different components in $\mathbf{r}$. For learning from pairwise preferences, we use a parametric Bradley-Terry model [7]

$$p(\tau_i \succ \tau_j | \mathbf{w}) := \frac{\exp(\mathbf{w}^T \mathbf{r}(\tau_i))}{\exp(\mathbf{w}^T \mathbf{r}(\tau_j)) + \exp(\mathbf{w}^T \mathbf{r}(\tau_i))}, \tag{1}$$

with $\mathbf{r}(\tau) = \sum_{(s,a,s') \in \tau} \mathbf{r}(s,a,s')$ being the reward obtained from a trajectory τ and $(\tau_i \succ \tau_j)$ denoting the preference of τ_i over τ_j. This way, trajectories that achieve a higher (linearly) scalarized reward are ranked exponentially better in proportion. Assuming that a number of pairwise comparisons $\{q_1, \ldots q_n\}$ have been obtained, we can then learn a posterior distribution in a Bayesian manner

$$p(\mathbf{w}|q_1, \ldots, q_n) \propto p(\mathbf{w}) \prod_{t=1}^n p(q_t|\mathbf{w}), \tag{2}$$

where $p(\mathbf{w})$ is a prior, which we chose to be randomly uniform over all weights $\mathbf{w}$ with $||\mathbf{w}|| \leq 1$ and $\mathbf{w} \geq 0$.

MORAL learns its distribution over weights interactively, by alternating between training a PPO agent on the posterior mean reward function $\mathbb{E}_{\mathbf{w}}[\mathbf{w}^T\mathbf{r}]$ and updating the posterior by obtaining a new query q_{n+1}. In order to calculate the expected value over (2) and select pairs of trajectories to query for, we adapt the active learning procedure by Sadigh et al. [34]. Firstly, we replace (1) with a proxy likelihood of the form

$$\hat{p}(\tau_i \succ \tau_j | \mathbf{w}) := \min(1, \exp(\mathbf{w}^T \Delta_{ij})), \tag{3}$$

where $\Delta_{ij} := \mathbf{r}(\tau_i) - \mathbf{r}(\tau_i)$. Its mode always evaluates to 0, which allows for efficiently obtaining posterior estimates through Markov chain Monte Carlo (MCMC) [9] with a warm start to the distribution mode. Secondly, we select queries based on the amount of volume removed from the posterior by approximately solving

$$\max_{(\tau_i, \tau_j)} \min\Big(\mathbb{E}_{\mathbf{w}}[1 - p(\tau_i \succ \tau_j|\mathbf{w})], \mathbb{E}_{\mathbf{w}}[1 - p(\tau_j \succ \tau_i|\mathbf{w})]\Big). \tag{4}$$

For sufficiently complex MOMDPs, maximizing this expression over all pairs of feasible trajectories proves to be computationally intractable. Instead, we do a discrete search over randomly sampled pairs of trajectories that arise during on-policy RL experience, see algorithm 1. Before each policy improvement step, we sample pairs (τ_i, τ_j) and evaluate the corresponding minimum in expression (4). If (τ_i, τ_j) scores highest among all previous pairs obtained since the last posterior update, then it is saved in a buffer and queued for the next query, unless a better pair is found later on.

Overall, this active learning scheme allows MORAL to interactively fine-tune an agent using only a few queries to an expert. Nonetheless, forming queries based on on-policy experience can only lead to locally optimal solutions. Therefore, assuming fixed weights $\mathbf{w}$, we typically solve an entropy-regularized objective

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \mathbf{w}^T \mathbf{r}(s_t, a_t) - \log \pi(s_t, a_t)\right]. \quad (5)$$

This way, MORAL can be interpreted as finding an average of Kullback-Leibler [27] (KL) divergences between the policy distribution over trajectories $\pi(\tau) := \mu_0(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$ and the marginal maximum entropy IRL distributions $p_{\theta_i}(\tau)$.

Theorem 2.1. *Given* $\mathbf{w} \in \mathbb{R}^k$ *with* $\mathbf{w} \geq 0$, $\sum w_i = 1$, *we have that*

$$\pi^* = \arg\min_{\pi} \sum_{i=1}^{k} w_i D_{KL}(\pi(\tau)||p_{\theta_i}(\tau)). \quad (6)$$

Proof: We provide a proof in the supplementary material.

Theorem 2.1 assumes that all components in $\mathbf{r}$ arise from maximum entropy IRL. However, in practical applications one might want to encode additional prior knowledge into the agent's behavior through a manually engineered primary reward function r_P. Nonetheless, by applying analogous reasoning, we see that (5) can then be interpreted as maximizing cumulative rewards $\mathbb{E}_{\pi}[\sum_{t=0}^{T} r_P(s_t, a_t)]$ with a KL regularizer in the form of (6). Under this interpretation, MORAL interactively finds regularization hyperparameters that determine which expert's behavior should be prioritized at runtime.

2.3 Reward Normalization

Finding scalarization weights in the presence of reward functions with highly different scales is a challenging task for many MORL algorithms. MORAL, on the other hand, learns its weights from preferences, thus making it less susceptible to reward functions that are difficult to compare. Nevertheless, the scale of the reward indirectly impacts the sensitivity of the posterior, since the magnitude of the likelihood (3) depends on Δ_{ij}. When these reward differences are large, this can lead the term $\exp(\mathbf{w}^T \Delta_{ij})$ to become close to zero, which introduces a risk of removing significant parts of the posterior support based on a single query. To tackle this, we utilize the policies obtained from AIRL to normalize each reward component by setting

$$\overline{f_{\theta_i}}(s, a) := \frac{f_{\theta_i}(s, a)}{|J(\pi^*_{E_i})|}, \quad (7)$$

where $J(\pi^*_{E_i}) = \mathbb{E}_{\pi^*_{E_i}}[\sum_{t=0}^{T} \gamma^t f_{\theta_i}(s, a)]$ is the scalar return of $\pi^*_{E_i}$. We note that this does not introduce any computational overhead,

Algorithm 1: Multi-Objective Reinforced Active Learning

```
Input: Expert demonstrations $\mathcal{D}_E = \{\tau_i\}_{i=1}^N$, prior $p(\mathbf{w})$.
Initialize: Reward function $\mathbf{r} = (f_{\theta_1}, \dots, f_{\theta_k})$ by running
  AIRL on $\mathcal{D}_E$, PPO agent $\pi_\phi$.
for $n = 0, 1, 2, \dots$ do
    Approximate $p(\mathbf{w}|q_1, \dots, q_n)$ through MCMC.
    Get mean reward function $r \leftarrow \mathbb{E}_{\mathbf{w}}[\mathbf{w}^T \mathbf{r}]$.
    $volume \leftarrow -\infty$
    for $k = 0, 1, 2, \dots, N$ do
        Sample trajectories $\mathcal{D} = \{\tau_i\}_{i=1}^m$ using $\pi_\phi$.
        Update $\phi$ using PPO to maximize
          $\mathbb{E}_{\pi_\phi}\left[\sum_{t=0}^T \gamma^t r(s_t, a_t)\right]$.
        Sample a pair of trajectories $(\tau_i, \tau_j)$ from $\mathcal{D}$.
        $next\_volume \leftarrow \min(\mathbb{E}_{\mathbf{w}}[1 - p(\tau_i \succ$
          $\tau_j|\mathbf{w})], \mathbb{E}_{\mathbf{w}}[1 - p(\tau_j \succ \tau_i|\mathbf{w})])$ .
        if $next\_volume > volume$ then
            $next\_query \leftarrow (\tau_i, \tau_j)$
            $volume \leftarrow next\_volume$
    Query expert using $next\_query$ and save answer $q_n$.
```

and simply estimate $J(\pi^*_{E_i})$ by averaging over observed returns of the final 200 episodes in AIRL.

3 EXPERIMENTS

In the following, we will demonstrate the ability of MORAL in simulation studies of two gridworld environments. To enable a qualitative analysis of the method, we assume that in both environments, a ground truth reward function exists, by which demonstrations and preferences are automatically provided. Furthermore, by following the experimental setup of related research [29, 45], we consider environments with a primary reward function r_P, encoding a generic task that can easily be solved through deep RL. In this case, we can apply MORAL as before, but add r_P as an additional reward component to the AIRL reward functions for the active learning step. To form the gridworld state, we make a binary array $I \in \{0, 1\}^{C \times W \times H}$ of width W and height H, as well as channels that encode grid occupancy for all C different object types on the grid. Finally, we employ a convolutional neural network architecture for PPO, consisting of two base convolutional layers with kernel size 2, and 64 as well as 256 output channels respectively. Its activations are then fed into two separate convolutional layers with kernel size 2 and 32 output channels each, followed by a linear layer for the critic and actor heads. For details, we refer to the supplementary material.

3.1 Emergency

We start by illustrating how MORAL can be applied to incorporating normative behavior from a single expert alongside a primary goal. We define the *Emergency* gridworld as follows: An agent, as well as 6 people are randomly initialized onto a 6×6 grid. Furthermore, the bottom right corner contains a fire extinguisher, which the agent automatically uses when standing on its respective cell. At each step, the agent can move in one of the four directions or interact with an adjacent cell.

We define the agent's primary goal r_P to give a reward of +0.1 for each time step spent in the fire extinguisher cell. In addition, we assume that people are lost and need to be escorted, which can only be achieved through interacting with them before the time limit of $T = 75$. However, this additional reward is not considered in r_P. In order to learn about the normative behavior of helping lost people escape the area, we find a reward f_θ by running AIRL on 50 synthetic demonstrations coming from a PPO agent that maximizes the amount of people saved. Subsequently, we form a reward vector $\mathbf{r} = (r_P, f_\theta)$ and run interactive MORL using a total of 25 queries. Since we would like to incorporate the goal of saving all people into the primary task of extinguishing fire, we provide preferences in the following way: Given two trajectories (τ_i, τ_j), we return $i > j$ if the amount of people saved in τ_i exceeds that of τ_j. If both trajectories save the same amount of people, we opt for the trajectory that spent more time in the extinguisher cell. Finally, queries are spread out evenly over a total of $6e6$ environment steps.

Figure 2 shows the set of policies obtained during training of MORAL and compares it with a CCS found from a manual scalarization $\lambda r_P + (1 - \lambda) f_\theta$ for different choices of $\lambda \in [0, 1]$. To illustrate the evolution of solutions, we estimate average returns and plot a corresponding point before each update of the weight posterior. MORAL directly approximates a Pareto optimal point that opts for saving all people present in the world, while maximizing the agent's performance with respect to the primary goal. Furthermore, MORAL first learns to only save people, which correctly corresponds to the way preferences are provided. Thus, MORAL demonstrates to be successful at directly finding a normative policy while incorporating reward information from multiple sources. To ensure consistency across multiple runs, we also plot the average returns for different numbers of overall queries in figure 3. We see that although 25 queries are necessary to converge to the desired solution, MORAL learns a reasonable trade-off after 10 queries that highly consistently saves all people at the cost of spending less time on the primary goal.

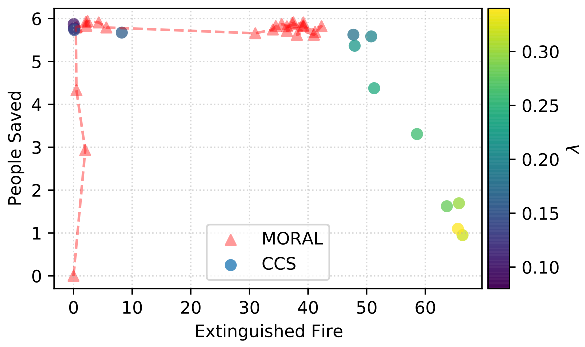


Figure 2: The set of solutions found by MORAL in the *Emergency* domain, compared to a manually trained CCS. MORAL approximates a Pareto optimal solution that most closely matches the given preferences during training.

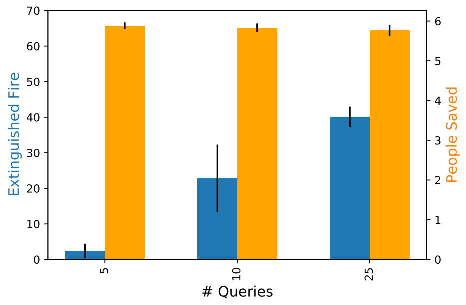


Figure 3: Query efficiency of MORAL for finding the desired trade-off averaged over three random seeds.

3.2 Delivery

While the *Emergency* domain illustrated the effectiveness of MORAL in a simplified setting, we yet have to analyze how MORAL performs in larger environments, as well as regarding increased diversity of norms and goals we would like an agent to learn. To better evaluate MORAL, we therefore define the *Delivery* environment, a randomly initialized 16×16 grid world shown in figure 4. As before, the agent has access to the moving, interaction and null actions, but can now encounter a variety of objects. Its primary goal consists of delivering packages to 12 locations, which is available to the agent via a reward r_P of +1 whenever it interacts with a delivery cell. However, there also exist a multitude of people in need of support that the agent can choose to assist (*help*) and pollution that can be removed (*clean*). The agent chooses to do so by interacting with each of the respective cells, after which they turn empty. Finally, we randomly place vases throughout the grid, which break whenever the agent steps on their position and can not be interacted with.

Overall, we limit the episode length to $T = 50$ time steps and place 12 of the *help, clean* objectives as well as 8 vases on the grid. We view this environment as a multi-objective problem including three norms, where *help* and *clean* are active behaviors, but the ability to avoid vases is passive i.e., an inaction. Besides forcing the agent to make trade-offs, this choice allows us to effectively

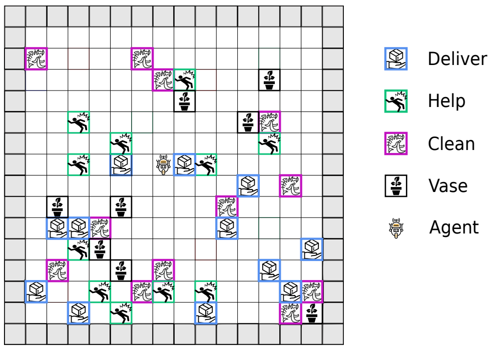


Figure 4: The *Delivery* Environment consists of a primary goal (*Deliver*) and three different norms (*Help, Clean, Vase*).

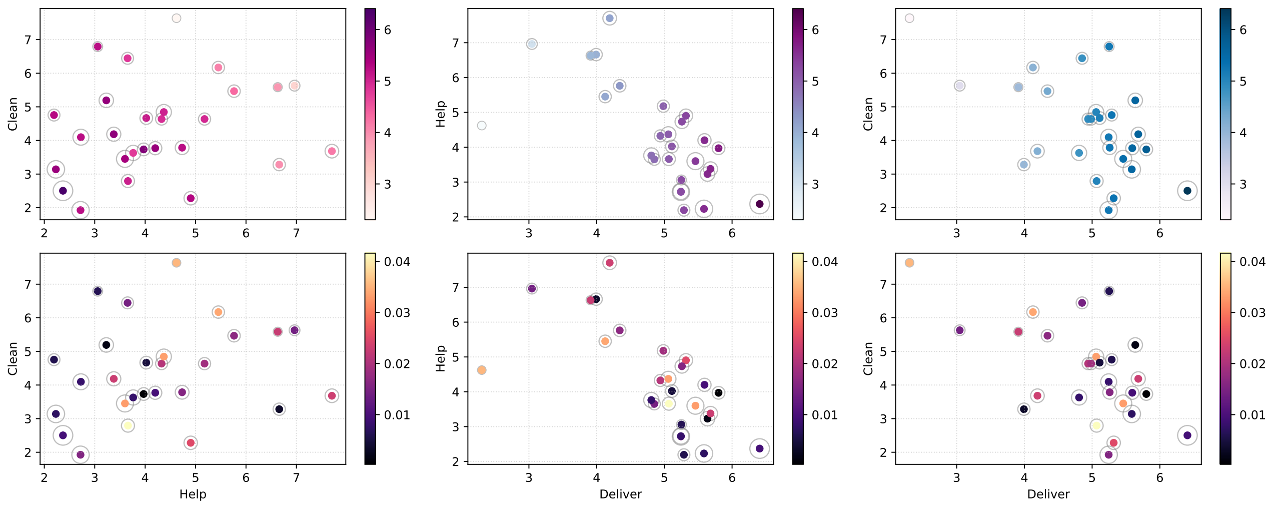


Figure 5: The convex coverage set found by MORAL for three reward dimensions. We plot two-dimensional projections of the attained explicit objectives with colors indicating the third objective (*first row*) and the KL divergence (8) to the preference vector m used during training (*second row*).

study the quality of solutions found through MORAL, by introducing a symmetry with regard to the three explicit objectives. As a result, we assume that preferences are automatically provided by a subjective distribution $m \in \Delta_{\{1,2,3\}}$ encoding desired priorities for *deliver*, *help* and *clean* respectively. Given a pair of trajectories (τ_1, τ_2), we then calculate two vectors $s_i = (o^i_1, \dots, o^i_n)$, where o^i_k denotes the obtained returns in terms of the k-th objective in trajectory i. For example, if τ_1 delivers 3 packages, helps 1 person and cleans up 3 cells, then $s_1 = (3, 1, 3)$. When normalizing the observed returns into a discrete distribution $\bar{s}_i = s_i/||s_i||_1$, we can provide preferences according to a KL divergence metric

$$i^* = \underset{i \in \{1,2\}}{\arg\min}\, D_{KL}(\bar{s}_i || m). \tag{8}$$

Aside from providing preferences in a principled way, we use this divergence measure to evaluate the overlap between a policy and the provided preferences throughout training.

We test MORAL using two conflicting demonstration data sets generated by a PPO agent optimizing for (i) helping people and (ii) cleaning tiles, while both try to avoid stepping on vases. As before, we subsequently use MORAL as a regularizer and form $\mathbf{r} = (r_P, f_{\theta_1}, f_{\theta_2})$, where θ_1 and θ_2 denote the trained AIRL parameters. As opposed to the experiment in the *Emergency* domain, there now exists an inherent normative conflict in the demonstrations. Thus, instead of tuning the agent to respect a specific policy that incorporates the normative component into the primary goal, we aim to test whether MORAL is able to retrieve solutions that match a variety of preferences. To achieve this, we vary the supplied preference vector m to match all possible ratios in $\{1, 2, 3\}^3$ during the active learning stage. Furthermore, we choose to use 25 queries overall, spread evenly throughout $8e6$ environment steps.

Figure 5 illustrates the found set of policies, where each point represents a separate run of active learning on different preferences. Since the objective space is three-dimensional, we only show two-dimensional projections and add the third objective through color in the first row. Besides this, the amount of broken vases are shown by gray circles around each point, where a bigger radius indicates policies that break more vases and a radius of 0 indicates that no vases are broken on average. To test whether the found policies match the given preferences, we also add a second row that indicates the KL divergence (8) of the final policy to the preference m that was used during training. We find that MORAL is overall able to retrieve a diverse set of policies, which accurately represent the different preferences. As can be seen, most of the points achieve a near zero divergence, indicating that the agent accurately matches the supplied ratios over objectives. As expected, we also see that the amount of broken vases correlates with the weight put on the primary task, since the manually engineered delivery reward is entirely agnostic regarding the vase object. Nonetheless, for appropriate choices of m, there exist policies which successfully avoid vases despite delivering an adequate amount of packages.

This indicates that when choosing scalarization weights correctly, minimizing a weighted sum of KL divergences to maximum entropy IRL distributions can achieve implicit normative behaviors without the need of an explicit feedback signal. We suggest that this is a highly desirable property for building value-aligned systems. Firstly, albeit in a different setup, informativeness about desirable behavior has been previously identified as a desideratum for combining reward information by Krasheninnikov et al. [26]. It allows the active learning procedure to focus on higher-level normative conflicts, which can have a crucial effect on performance, as we will discuss in section 3.4. Secondly, automatically adhering to common implicit behaviors ensures safety against adversarial preference givers. To elaborate on this, consider figure 6. Here, we trained MORAL on $\mathbf{r} = (f_{\theta_1}, f_{\theta_2})$ by giving 25 preferences such that $\tau_i \succ \tau_j$, whenever τ_i manages to break more vases. We observe that despite

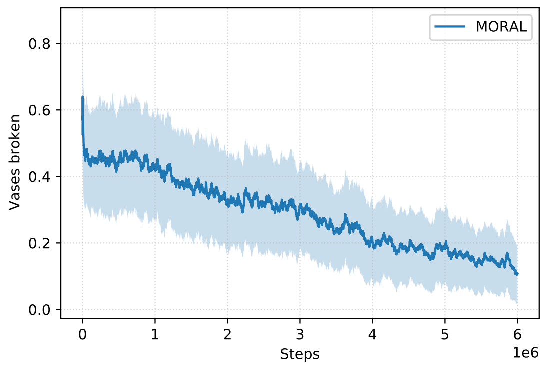


Figure 6: Average amount of broken vases over three training runs. MORAL learns a safe policy, despite being provided with adversarial preferences.

this, the amount of broken vases in fact decreases as a function of training steps. Since both experts agree on keeping vases intact, the aggregate reward function can not be fine-tuned to exhibit the opposite behavior. We note, however, that such a guarantee against adversarial preferences only holds when all marginal reward functions induce safe behavior, which is not the case when adding the primary reward r_P.

3.3 Ablation

In this section, we evaluate MORAL with respect to the necessity of active queries, as well as its robustness against noisy preferences. To do so, we first run MORAL for the same set of preferences as in figure 5 for different amounts of overall queries, while keeping the total environment steps constant. After training, we average the preference deviation (8) over all obtained policies and visualize the results in the top row of figure 7. Furthermore, we run the same experiment using queries that are randomly selected from on-policy experience and plot the results accordingly. In the case of actively generated queries, there is a clear decrease in preference deviation as a function of total queries. Furthermore, we can see diminishing returns, with the jump from 5 to 10 queries reducing the error by a factor of 0.5, whereas the difference between 50 and 25 queries only leads to a subtle increase in accuracy. Besides exhibiting a systematically higher deviation, random queries do not benefit from the increase in queries as much, with 50 queries in fact scoring worse than 25. We conjecture that this is due to the on-policy sampling of trajectories, which strongly restricts the space of trajectories that the agent can query for. MORAL will, by design, always seek pairs of trajectories that naturally exhibit larger variances between competing goals in order to generate queries with high information content. When ensuring that the agent maintains adequate levels of exploration throughout training, this leads to a decrease in deviation even in the large query regime. On the other hand, there is an inherent risk of convergence to locally optimal scalarization weights due to the log-concavity of the likelihood (3). As a result, the entropy of the posterior constantly decreases. When querying at

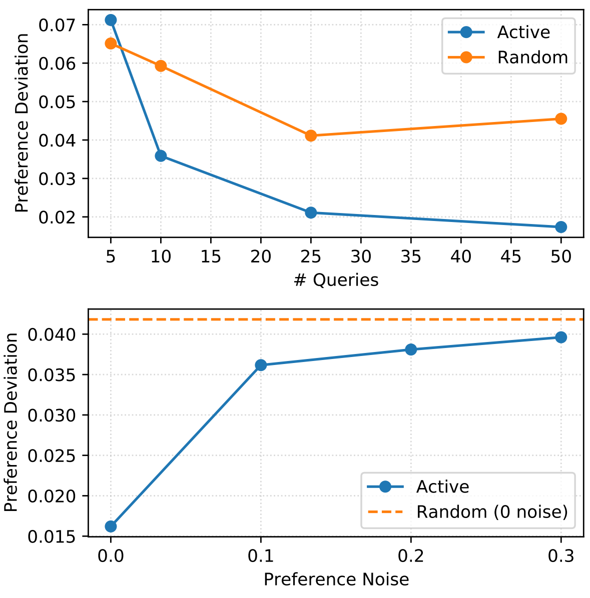


Figure 7: (Top) Average preference deviation for different numbers of active and random queries. (Bottom) Average preference deviation of active learning in the presence of noisy feedback.

random, this effect is exacerbated, because exploratory trajectories are unlikely to get selected. However, we note that MORAL is not immune to, but merely mitigates the risk of local optima. Although this suffices for the type of environments we study in our paper, we expect the optimality gap to grow with increasingly sparse domains. To avoid this, we deem the introduction of an exploration policy for generating queries to be a promising avenue for future work.

Aside from the amount of queries needed, figure 7 also illustrates how MORAL behaves in the presence of contradictory feedback. In this case, we train policies on the same set of preferences as before, but provide random answers to each of the 50 queries with a certain probability. Unsurprisingly, we see a sharp increase in deviation when injecting a noise level of 0.1, above which the growth in error diminishes. Nonetheless, active queries with a random answer probability of 0.3 still retrieve slightly more accurate representations than random queries without any noise. Such robustness with respect to noise is important, since our experiments only cover synthetic simulation studies, whereas human feedback is unlikely to be as consistent. Even though random noise is only an approximation of human error, we conclude from our results that seeking volume removal in the active learning loop does not make the algorithm more susceptible to converging to locally optimal scalarization weights in this case.

3.4 Comparison to DRLHP

Through its two-step procedure, MORAL is able to combine multiple reward functions from diverse expert behavior. However, in the active learning stage, we require a single preference giver to determine which Pareto optimal policy should ultimately be optimized for. Given enough pairwise comparisons, this directly approximates a policy that best matches the preferences, as we have shown in figure 2. As such, MORAL is most directly comparable to deep reinforcement learning from human preferences (DRLHP) [10], which directly trains a deep reward model from pairwise preferences. To compare the two, we train DRLHP until convergence in *Emergency* and *Delivery* by providing a sufficient amount of pairwise comparisons to make up for the missing primary reward and demonstrations that MORAL has access to.

Table 2 shows the results when providing DRLHP with 1000 preferences in the same way as MORAL for the *Emergency* domain. As before, trajectories with more people saved are preferred unless equal, in which case extinguishing fire becomes a priority. Although this leads DRLHP to learn a policy that consistently saves most people, it significantly lacks behind in terms of extinguished fire. This is unsurprising, since DRLHP is not designed to handle multi-objective problems and can not utilize the manually engineered reward signal in any meaningful way. This is because the deep reward model is nonstationary, which poses the combination with the stationary reward r_P to be challenging. As a result, DRLHP needs to maintain a single model for all competing objectives, which can lead to catastrophic forgetting of extinguishing fire when updating the reward network to save more people.

A similar trend can be observed in the *Delivery* environment, where we compare mean performance of DRLHP versus MORAL on three preference configurations, each of which prefer one of the objectives most strongly. However, since we assumed the avoidance of vases to only implicitly be encoded in the preferences, we cannot supply DRLHP with the same set of feedback. Instead, we train DRLHP to prefer trajectories that have a lower mean squared error to the vector of expected returns achieved by MORAL. Figure 8 shows training curves of both methods, where each row represents a preference ratio of $(3, 1, 1)$, $(1, 3, 1)$ and $(1, 1, 3)$ respectively. Aiming to make the comparison fairer, we offset MORAL by the amount of total training steps needed for IRL. As before, DRLHP manages to retrieve solutions that loosely resemble the supplied preferences, but fails to converge to Pareto optimal policies. Furthermore, we notice that for the latter two preferences, sparse objectives such as minimizing the amount of broken vases are not picked up by DRLHP. We suspect this to be an exploration issue, where trajectories that break fewer vases are unlikely to arise in queries. Thus, DRLHP optimizes for the remaining objectives as they lead to a higher increase in correctly predicting an expert's preferences. Overall, we therefore conclude that MORAL is more suitable than DRLHP in multi-objective settings that require trading off conflicting objectives from expert data. Nonetheless, we note that MORAL has a theoretical advantage in this environment, since it allows for incorporation of prior knowledge as well as conflicting expert demonstrations.

	People Saved	Extinguished Fire	Nr. of Queries	Steps (IRL)
MORAL	5.76(±0.13)	**40.08**(±2.9)	25	3e6 (3e6)
DRLHP	5.62(±0.17)	12.32(±3.0)	1000	12e6 (-)

Table 1: Comparison of MORAL and DRLHP in *Emergency*.

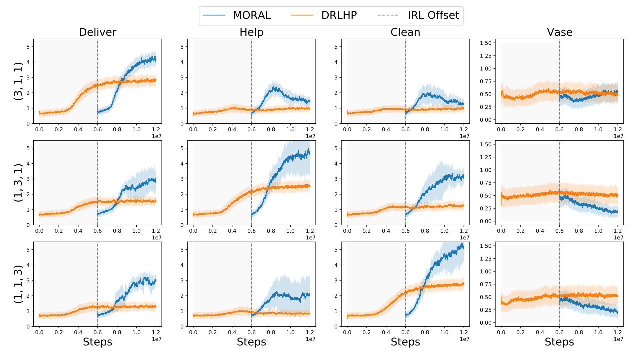


Figure 8: Mean training curves of DRLHP and MORAL on preference ratios $(3, 1, 1)$ (top), $(1, 3, 1)$ (middle) and $(1, 1, 3)$ (bottom).

4 RELATED WORK

Machine Ethics Using the notion of uncertainty and partial observability, RL has been suggested as a framework for ethical decision-making [2]. We frame the problem of learning norms in a multi-objective context, which can be interpreted as inducing partial observability over the set of reward scalarizations one would wish to optimize. Overall, our approach is most directly comparable to policy orchestration [29] and ethics shaping [45]. Policy orchestration [29] also adopts a multi-objective view to incorporate ethical values into reward-driven RL agents, by solving a bandit problem that uses IRL to alternately play ethical and reward maximizing actions. MORAL closely relates to policy orchestration, since it also employs a two-step procedure that first finds normative rewards from IRL and only afterwards aggregates different reward functions into a single policy. However, MORAL uses deep function approximation, allowing it to drop the assumption of a tabular state space. Similarly, ethics shaping [45] learns a reward shaping term from demonstrations, but does not scale beyond manually engineered features. Finally, Ecoffet et al. [12] suggest a sequential voting scheme for learning conflicting values, but it requires explicit knowledge of the different values at stake.

Inverse Reinforcement Learning Similarly to related IRL research, our work builds on AIRL [14] for inferring rewards from a multimodal distribution of demonstrations. Unlike previous research, which has focused on introducing latent variable models [19, 23, 28, 39, 41, 48] in the context of multiagent, multitask and hierarchical reward learning, we instead focus on the combination of labeled demonstration data. As such, our setup is similar to Gleave and Habryka [17] and Xu et al. [46], where a reward function is meta learned by having explicit access to different task distributions. However, we learn from demonstrations in a multi-objective context, which has, to our knowledge, not yet been studied before.

Besides this, IRL has been applied in the context of value-alignment [22], where inverse reward design (IRD) [21] has been proposed to learn a distribution of reward functions through IRL that leverages uncertainty to avoid unintended behavior. Although we also learn a distribution over reward functions, IRD focuses on finding safe goal specifications from a single reward function, whereas we study the extraction of value-aligned policies from a multitude of demonstrations. As a result, our research is conceptually similar to multitask IRD [26], which studies formal criteria for combining reward functions from multiple sources. We, on the other hand, drop the formal assumptions and propose a practical method for combining reward functions learned through deep neural networks.

Learning from Expert Feedback Besides IRL, there exist a variety of approaches for training RL agents from expert data, including scalar-valued input [25, 42], natural language [3], intervention [36] and pairwise preferences [10, 44]. Similarly to Christiano et al. [10], we employ a Bradley-Terry model for training a nonstationary reward function by comparing trajectories from on-policy RL experience. However, our model of pairwise preferences operates on a set of abstract high-level reward functions, whereas [10] learn a single end-to-end reward model. Furthermore, our approach combines demonstration and preference data, which is more similar to Ibarz et al. [24]. Nonetheless, [24] uses demonstration data for pretraining a preference-based reward model, which does not account for conflicting demonstrations. MORAL, on the other hand, allows for the inclusion of multiple experts as well as prior knowledge, thus making it suitable for resolving normative conflicts. Finally, by combining different expert reward functions, we have shown that MORAL interpolates between maximum entropy IRL distributions. From this point of view, we consider our work a counterpart to Brown et al. [8], which ranks demonstration data in order to extrapolate beyond the behavior of a single expert.

Multi-Objective Decision-Making Typically, MORL algorithms trade off multiple objectives by learning a policy, or a set thereof, that can represent a range of Pareto optimal solutions [30, 47]. On the other hand, our model learns a distribution over reward functions, which interactively guides the search to produce a single Pareto optimal policy. Aside from sample efficiency, this mitigates the problem of varying reward scale, which has previously been addressed by multi-objective maximum a posteriori policy optimization (MO-MPO) [1]. However, MO-MPO requires explicit preferences over objectives, which is not always feasible when combining learned rewards that are inherently difficult to compare.

By using on-policy RL experience to learn scalarization weights, MORAL can be viewed as an interactive MORL algorithm. To date, interactive MORL has mainly been applied to bandits for linear [32] and nonlinear [31] transformations of the reward components, but not yet been studied in the full RL setting. We believe that this is the case, because MORL research usually assumes environments with manually engineered reward functions, in which big parts of the Pareto boundary exhibit interesting solutions. In the case trading off learned reward functions, however, we suggest that our interactive approach poses a more adequate option.

	Ethics-Shaping [45]	Policy-Orchestration [29]	DRLHP [10]	MORAL
Function Approximation	✗	✗	✓	✓
Multi-Objective	✗	✓	✗	✓
Multiple Experts	✗	~	✗	✓
Outperform Experts	✓	✓	✓	✓

Table 2: Comparison of MORAL to previous work in terms of supported capabilities.

5 DISCUSSION

In our work, we propose MORAL, a method for combining learned reward functions from multiple experts. We have shown MORAL to be a technical approach for aligning deep RL agents with human norms, which uses active learning to resolve value conflicts within expert demonstrations. We consider our research a first step towards MORL with learned rewards, which has not yet been addressed before. Previous approaches such as ethics-shaping [45] and [29] have highlighted the strength of combining reward functions with expert demonstrations for value-alignment, whereas DRLHP [10] has demonstrated the scalability of deep preference-based RL, see table 2. MORAL unifies these ideas into a single method, which allows it to be applied in the presence of function approximation and multiple experts. Furthermore, this theoretical advantage is reflected in our experiments, which show that, unlike MORAL, DRLHP fails to retrieve Pareto optimal solutions.

Nonetheless, several avenues for future research remain to be addressed. Combining multiple forms of expert supervision is challenging, due to a risk of accumulating errors and modelling assumptions for each type of input. We deem further research in AIRL to be crucial, to prevent overfitting of the reward network. Similarly to Gleave and Habryka [17], we found the reoptimization of AIRL reward functions to decrease performance, indicating that the learned rewards are entangled with the state distribution of the generator policy. Although this will require significant progress in deep IRL, we expect future methods to be easily integrated into MORAL by replacing AIRL. Furthermore, one could pursue unsupervised techniques to extend MORAL to unlabeled demonstration datasets. When learning normative behavior from large scale real-world demonstration data, it might be infeasible to learn separate reward functions for each expert. Unsupervised learning of reward functions that correspond to the different modes of behavior instead could alleviate this issue.

Overall, our research aims to highlight the importance of multi-objective sequential decision-making without explicitly provided reward functions. Aside from value alignment [40], the ability to detect and respond to a divergence in values has been recognized as a central trait for building human-like AI [5]. Further, following the principle of meaningful human control [35], MORAL can contribute to increase an agent's responsiveness to conflicting human norms, while maintaining human autonomy in determining desired trade-offs. This research contributes to the broader goal of designing and developing safe AI systems that can align to human values and norms.

REFERENCES

[1] Abbas Abdolmaleki, Sandy Huang, Leonard Hasenclever, Michael Neunert, Francis Song, Martina Zambelli, Murilo Martins, Nicolas Heess, Raia Hadsell, and Martin Riedmiller. 2020. A distributional view on multi-objective policy optimization. In *International Conference on Machine Learning*. PMLR, 11–22.

[2] David Abel, J. MacGlashan, and M. Littman. 2016. Reinforcement Learning as a Framework for Ethical Decision Making. In *AAAI Workshop: AI, Ethics, and Society*.

[3] Md Sultan Al Nahian, Spencer Frazier, Mark Riedl, and Brent Harrison. 2020. Learning norms from stories: A prior for value aligned agents. In *AIES 2020 - Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. Association for Computing Machinery, Inc, 124–130. https://doi.org/10.1145/3375627.3375825 arXiv:1912.03553

[4] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. 2016. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565* (2016).

[5] Grady Booch, Francesco Fabiano, Lior Horesh, Kiran Kate, Jon Lenchner, Nick Linck, Andrea Loreggia, Keerthiram Murugesan, Nicholas Mattei, Francesca Rossi, et al. 2020. Thinking fast and slow in AI. *arXiv preprint arXiv:2010.06002* (2020).

[6] Nick Bostrom. 2014. *Superintelligence: Paths, Dangers, Strategies.* Oxford University Press.

[7] Ralph Allan Bradley and Milton E Terry. 1952. Rank analysis of incomplete block designs: I. The method of paired comparisons. *Biometrika* 39, 3/4 (1952), 324–345.

[8] Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. 2019. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International conference on machine learning*. PMLR, 783–792.

[9] Siddhartha Chib and Edward Greenberg. 1995. Understanding the metropolis-hastings algorithm. *The american statistician* 49, 4 (1995), 327–335.

[10] Paul F. Christiano, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*. 4300–4308.

[11] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. 2019. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901* (2019).

[12] Adrien Ecoffet and Joel Lehman. 2021. Reinforcement learning under moral uncertainty. In *International Conference on Machine Learning*. PMLR, 2926–2936.

[13] Tom Everitt, Victoria Krakovna, Laurent Orseau, Marcus Hutter, and Shane Legg. 2017. Reinforcement learning with a corrupted reward channel. *arXiv preprint arXiv:1705.08417* (2017).

[14] Justin Fu, Katie Luo, and Sergey Levine. 2017. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248* (2017).

[15] Iason Gabriel. 2020. Artificial intelligence, values, and alignment. *Minds and machines* 30, 3 (2020), 411–437.

[16] Sanket Gaurav and Brian D Ziebart. 2019. Discriminatively learning inverse optimal control models for predicting human intentions. In *International Conference on Autonomous Agents and Multiagent Systems*.

[17] Adam Gleave and Oliver Habryka. 2018. Multi-task maximum entropy inverse reinforcement learning. *arXiv preprint arXiv:1805.08882* (2018).

[18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems* 27 (2014).

[19] Nate Gruver, Jiaming Song, Mykel J Kochenderfer, and Stefano Ermon. 2020. Multi-agent adversarial inverse reinforcement learning with latent variables. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 1855–1857.

[20] Dylan Hadfield-Menell and Gillian K Hadfield. 2019. Incomplete contracting and AI alignment. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*. 417–422.

[21] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart Russell, and Anca Dragan. 2017. Inverse reward design. *arXiv preprint arXiv:1711.02827* (2017).

[22] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. 2016. Cooperative inverse reinforcement learning. *Advances in neural information processing systems* 29 (2016), 3909–3917.

[23] Karol Hausman, Yevgen Chebotar, Stefan Schaal, Gaurav Sukhatme, and Joseph Lim. 2017. Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. *arXiv preprint arXiv:1705.10479* (2017).

[24] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. 2018. Reward learning from human preferences and demonstrations in Atari. *arXiv preprint arXiv:1811.06521* (2018).

[25] W Bradley Knox and Peter Stone. 2009. Interactively shaping agents via human reinforcement: The TAMER framework. In *Proceedings of the fifth international conference on Knowledge capture*. 9–16.

[26] Dmitrii Krasheninnikov, Rohin Shah, and Herke van Hoof. 2021. Combining reward information from multiple sources. *arXiv preprint arXiv:2103.12142* (2021).

[27] Solomon Kullback. 1997. *Information theory and statistics.* Courier Corporation.

[28] Yunzhu Li, Jiaming Song, and Stefano Ermon. 2017. Infogail: Interpretable imitation learning from visual demonstrations. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 3815–3825.

[29] Ritesh Noothigattu, Djallel Bouneffouf, Nicholas Mattei, Rachita Chandra, Piyush Madan, Kush R Varshney, Murray Campbell, Moninder Singh, and Francesca Rossi. 2019. Teaching AI agents ethical values using reinforcement learning and policy orchestration. *IBM Journal of Research and Development* 63, 4/5 (2019), 2–1.

[30] Diederik M. Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. 2013. A Survey of Multi-Objective Sequential Decision-Making. *Journal of Artificial Intelligence Research* 48, 1 (2013), 67–113.

[31] Diederik M Roijers, Luisa M Zintgraf, Pieter Libin, and Ann Nowé. 2018. Interactive multi-objective reinforcement learning in multi-armed bandits for any utility function. In *ALA workshop at FAIM*, Vol. 8.

[32] Diederik M Roijers, Luisa M Zintgraf, and Ann Nowé. 2017. Interactive thompson sampling for multi-objective multi-armed bandits. In *International Conference on Algorithmic Decision Theory*. Springer, 18–34.

[33] Stuart Russell, Daniel Dewey, and Max Tegmark. 2015. Research priorities for robust and beneficial artificial intelligence. *AI Magazine* 36, 4 (2015), 105–114.

[34] Dorsa Sadigh, A. Dragan, S. Sastry, and S. Seshia. 2017. Active Preference-Based Learning of Reward Functions. In *Robotics: Science and Systems*.

[35] Filippo Santoni de Sio and Jeroen van den Hoven. 2018. Meaningful Human Control over Autonomous Systems: A Philosophical Account. *Frontiers in Robotics and AI* 5 (2018), 15. https://doi.org/10.3389/frobt.2018.00015

[36] William Saunders, Andreas Stuhlmüller, Girish Sastry, and Owain Evans. 2018. Trial without error: Towards safe reinforcement learning via human intervention. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, Vol. 3. 2067–2069. arXiv:1707.05173

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

[38] Rohin Shah, Noah Gundotra, Pieter Abbeel, and Anca Dragan. 2019. On the feasibility of learning, rather than assuming, human biases for reward inference. In *International Conference on Machine Learning*. PMLR, 5670–5679.

[39] Mohit Sharma, Arjun Sharma, Nicholas Rhinehart, and Kris M Kitani. 2018. Directed-Info GAIL: Learning Hierarchical Policies from Unsegmented Demonstrations using Directed Information. In *International Conference on Learning Representations*.

[40] Peter Vamplew, Richard Dazeley, Cameron Foale, Sally Firmin, and Jane Mummery. 2018. Human-aligned artificial intelligence is a multiobjective problem. *Ethics and Information Technology* 20, 1 (2018), 27–40.

[41] David Venuto, Jhelum Chakravorty, Leonard Boussioux, Junhao Wang, Gavin McCracken, and Doina Precup. 2020. oIRL: Robust Adversarial Inverse Reinforcement Learning with Temporally Extended Actions. *arXiv preprint arXiv:2002.09043* (2020).

[42] Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. 2018. Deep tamer: Interactive agent shaping in high-dimensional state spaces. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[43] Jess Whittlestone, Kai Arulkumaran, and Matthew Crosby. 2021. The Societal Implications of Deep Reinforcement Learning. *Journal of Artificial Intelligence Research* 70 (March 2021).

[44] Christian Wirth, Gerhard Neumann, and Johannes Fürnkranz. 2017. A Survey of Preference-Based Reinforcement Learning Methods. *Journal of Machine Learning Research* 18 (2017), 1–46.

[45] Yueh Hua Wu and Shou De Lin. 2018. A low-cost ethics shaping approach for designing reinforcement learning agents. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*. 1687–1694. arXiv:1712.04172

[46] Kelvin Xu, Ellis Ratner, Anca Dragan, Sergey Levine, and Chelsea Finn. 2019. Learning a prior over intent via meta-inverse reinforcement learning. In *International Conference on Machine Learning*. PMLR, 6952–6962.

[47] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. 2019. A Generalized Algorithm for Multi-Objective Reinforcement Learning and Policy Adaptation. arXiv:1908.08342 https://github.com/RunzheYang/MORL

[48] Lantao Yu, Tianhe Yu, Chelsea Finn, and Stefano Ermon. 2019. Meta-Inverse Reinforcement Learning with Probabilistic Context Variables. *Advances in Neural Information Processing Systems* 32 (2019), 11772–11783.

[49] Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. 2008. Maximum entropy inverse reinforcement learning. In *Proceedings of the National Conference on Artificial Intelligence*, Vol. 3. 1433–1438.

III
Appendix

A Demonstrations

A.1. Burning Warehouse

A.1.1. Demonstration Policy

In the *Burning Warehouse* environment, we train a PPO agent using a reward giving +1 for each person saved and 0 otherwise. Figure A.1 shows the training progress of the agent with respect to both objectives. As expected, the policy converges to a solution that is agnostic to goal time, but consistently saves all people on the grid. Subsequently, we generate demonstration datasets using the pretrained policy on a new random seed. Figure A.2 shows the first 24 frames of a sample demonstration. As can be seen, the agent acts nearly optimal with respect to saving people, whereas in the remaining frames it resorts to a random walk.

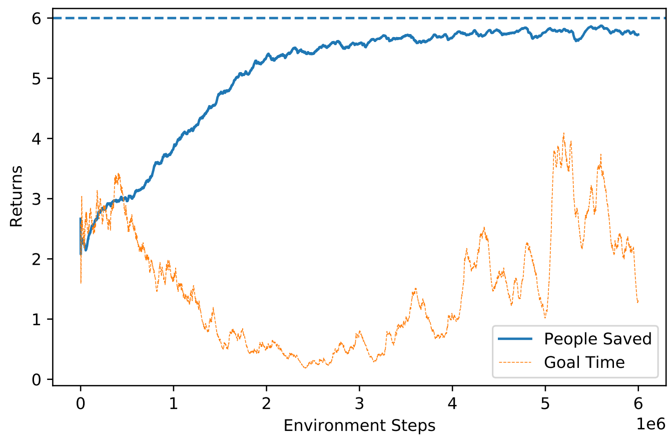


Figure A.1: Training progress of the demonstration policy in *Burning Warehouse*.

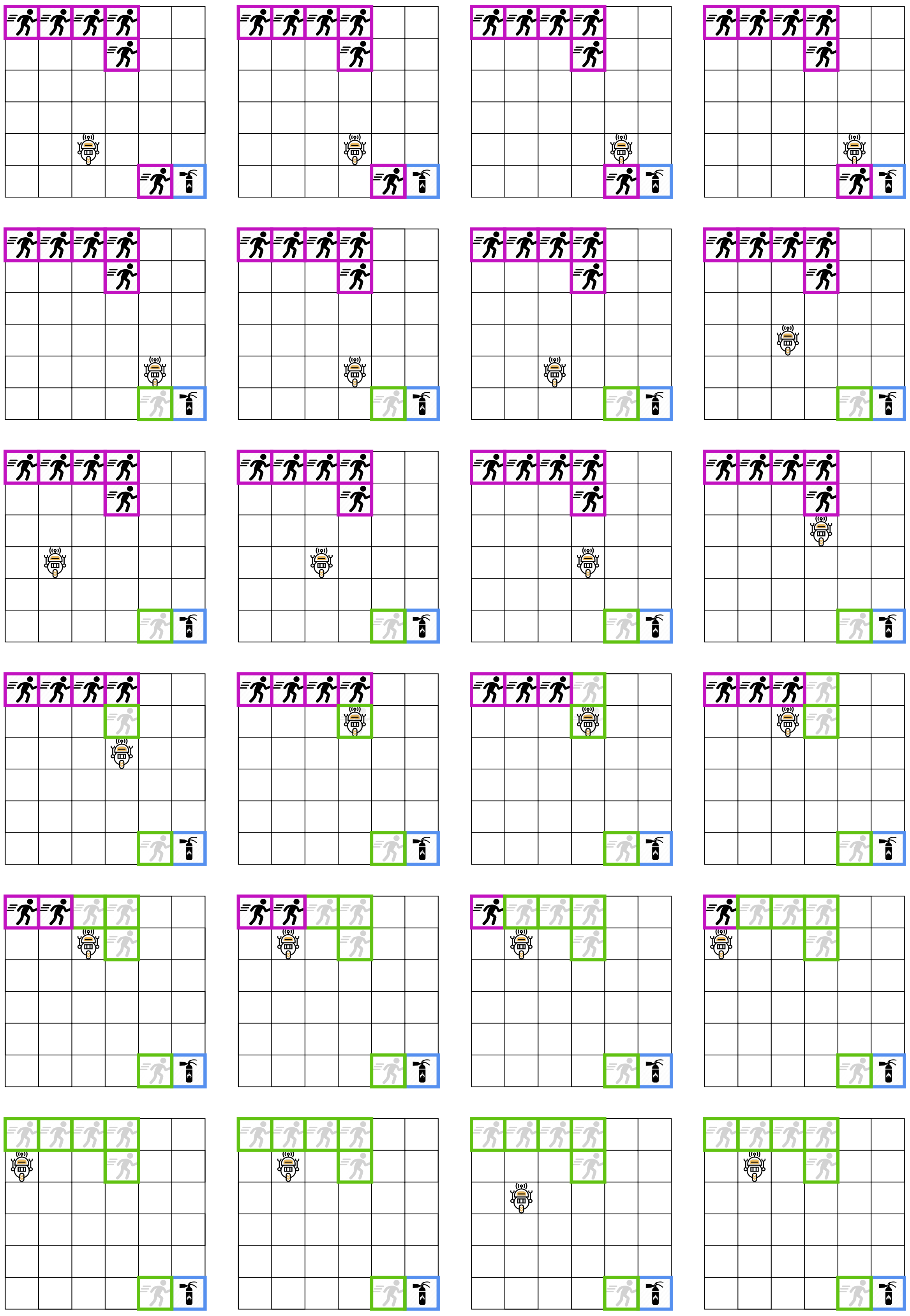


Figure A.2: First 24 frames of an example demonstration in *Burning Warehouse*. All people are saved after which the agent performs a random walk.

A.2. Delivery

A.2.1. Demonstration Policy

To trade off utility functions of different experts, we train two PPO agents in *Delivery*, each corresponding to a different set of norms. Namely, expert 1 is trained on a reward function of +1 for each person helped and a reward of −1 for each broken vase, whereas expert 2 is trained on a reward of +1 for each tile cleaned and a reward of −1 for each broken vase. Figure A.3 shows training progress of both expert policies with respect to the four objectives in the environment. Both experts manage to converge to a policy that consistently avoids vases while optimizing for their respective own goals. Note that although the number of *Help* and *Clean* tiles was equal and constant throughout subsequent episodes, expert 2 nonetheless performs slightly worse due to an inherent stochasticity of training PPO. In figures A.4 and A.5 we illustrate sample demonstrations from the respective expert datasets. For the ease of exposition, we have omitted exploratory actions which led to deviation of the shown path (orange) by less than one tile. In accordance with the training statistics, it can be seen that both policies take paths that cross their respective goals most frequently and act on their intentions by saving people or cleaning up polluted tiles.

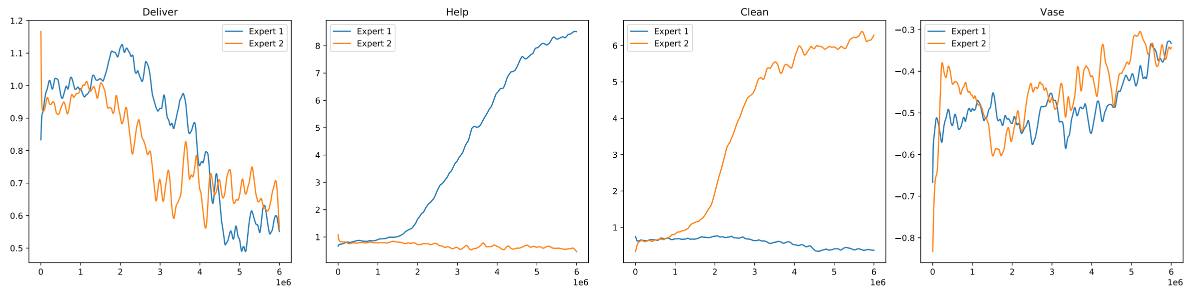


Figure A.3: Training progress of the two expert policies used for AIRL in *Delivery*. Expert 1 primarily optimizes for the *Help* objective while avoiding vases, whereas expert 2 avoids breaking vases alongside the *Clean* objective.

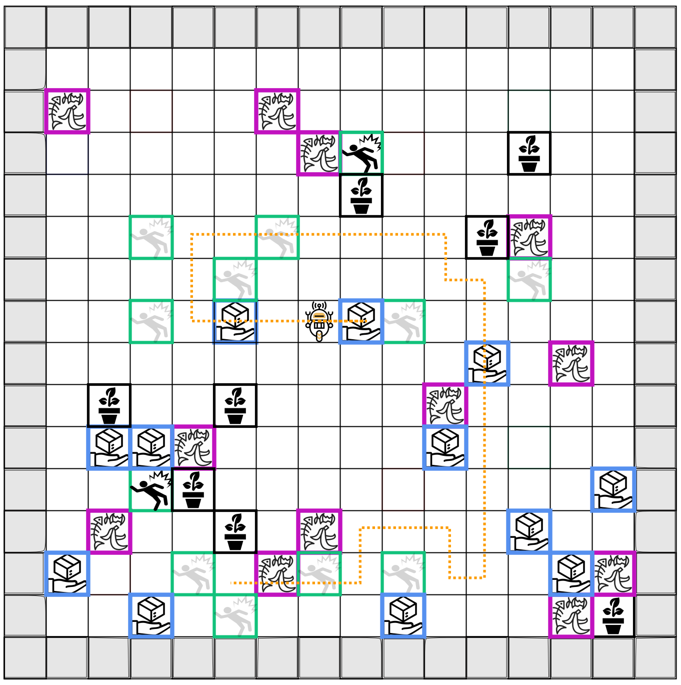


Figure A.4: Partial sample trajectory in the demonstration dataset of expert 1. For illustration purposes, lower opacity was chosen on objectives that have been interacted with by the agent.

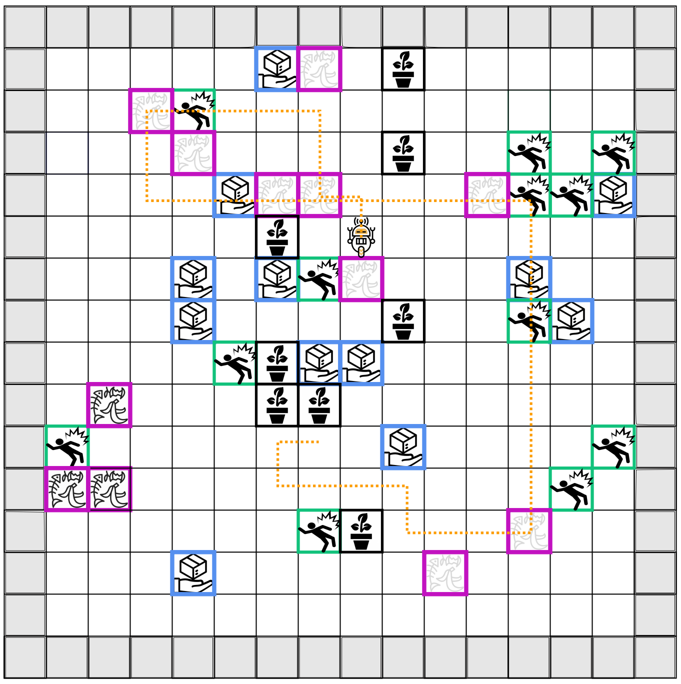


Figure A.5: Partial sample trajectory in the demonstration dataset of expert 2. For illustration purposes, lower opacity was chosen on objectives that have been interacted with by the agent.

A.2.2. AIRL Details

We train AIRL on the 1000 demonstrations of the expert policies from figure A.3 respectively. Figure A.6 shows discriminator loss, real accuracy, fake accuracy and returns respectively, whereas returns measure the cumulative rewards (as determined by the discriminator) obtained by the generator policy.

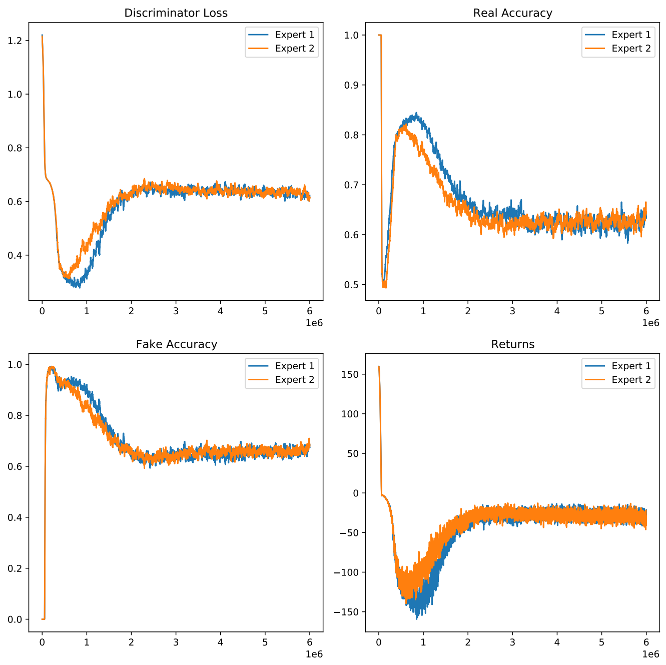


Figure A.6: Omitted training statistics for the AIRL training stage in *Delivery*. In the case of both experts, the discriminator loss stabilizes, but slightly outperforms the generator in terms of real and fake accuracies.

B Implementation Details

B.1. Neural Network Architectures

In this section we will describe various network architectures used throughout our experiments. For the ease of exposition, we omitted including the respective grid world dimensionalities of each environment, and instead only report the amount of output channels and kernel sizes respectively. Furthermore, each convolutional layer uses a stride of 1 and no padding, which we found sufficient due to the relatively small sizes of the grids.

B.1.1. Proximal Policy Optimization

For the PPO agent, we always employ a convolutional actor-critic architecture with shared base layers, as can be seen in figure B.1. The network uses two convolutional layers and forms a feature array with 256 channels, which are then passed onto the actor and the critic in parallel. At last, the actor employs a linear layer with output dimensions equal to the number of actions (which in our case amounted to $|\mathcal{A}| = 9$) on the flattened feature representations of the final convolutional layer. Similarly, the critic employs a final linear layer with a scalar output to predict the current value. In order to draw action samples from the actor, a softmax is performed over its last linear layer and the resulting vector is treated as a categorical distribution. In between layers, we employ standard ReLU activations to facilitate nonlinearity.

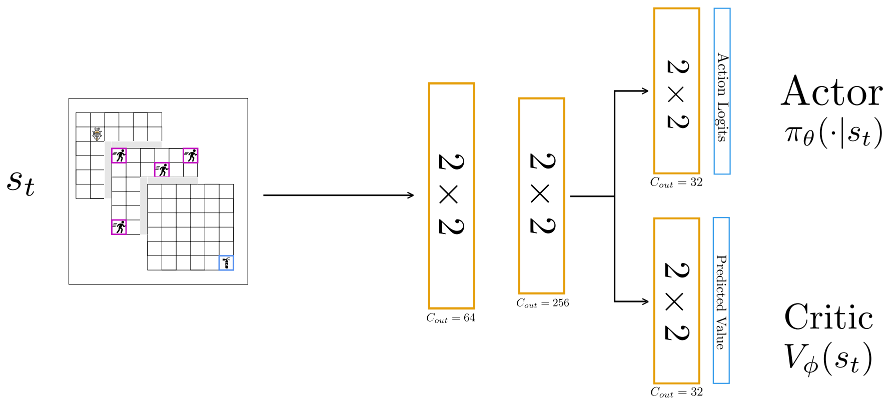


Figure B.1: Actor-Critic architecture of the PPO agent consisting of convolutional (yellow) and linear (blue) layers. Regardless of the input dimension, we use C_{out} output channels and kernel sizes of 2.

B.1.2. Convolutional Reward Network

We found the MLP architecture of the AIRL discriminator shown in figure 3.5 to be insufficient in larger grid world sizes. For this reason, we employ a convolutional reward network in *Delivery* as shown in figure B.2. In principle, the network follows the same structure as in the *Burning Warehouse* experiments, but replaces linear layers with convolutional ones. To do so, we employed three convolutional layers followed by a single linear layer that acts on respective flattened feature maps for both h_θ and g_θ and form our reward estimate as before. Finally, we use LeakyReLU activations with a slope of $\alpha = 0.01$ on all hidden layers.

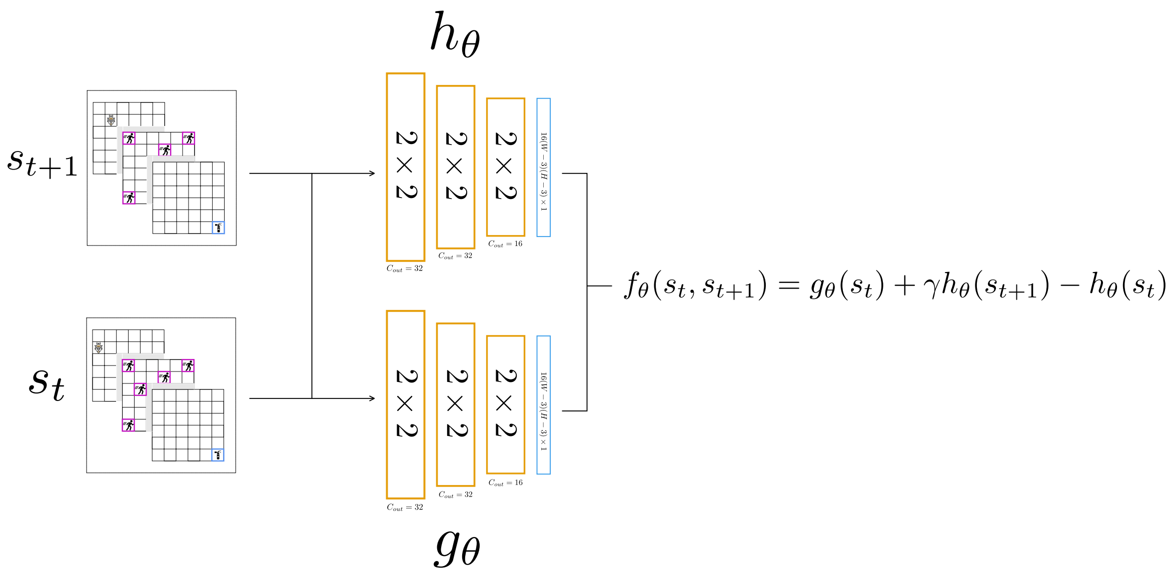


Figure B.2: Convolutional discriminator architecture for training AIRL in bigger environments with a parallel stream of convolutional (yellow) and linear (blue) layers.

B.1.3. Deep Reinforcement Learning from Human Preferences

In deep reinforcement learning from human preferences (DRLHP) we train a PPO agent using the architecture shown in B.1 in parallel with a deep reward model which is directly trained in a supervised manner from pairwise preferences. The reward model takes a state-action pair at each time step and outputs a predicted reward $r_\theta(s_t, a_t)$. We therefore first one-hot encode the action a_t and then embed it into a vector with the same dimensionality as the input state s_t. To do so, we train a linear embedding layer with output dimensions $C \cdot W \cdot H$, where (C, W, H) denote the amount of channels, width and height of the state s_t respectively. Embedded actions then get reshaped and concatenated with s_t along the channel dimension to form an array of dimension $(2C, W, H)$ (batch dimension omitted). This array is then fed through three convolutional layers with 128, 64 and 32 output channels respectively. Finally, the resulting flattened feature maps are fed through a linear layer to produce the reward estimate. As in the AIRL discriminator architecture, we employ LeakyReLU activations with a slope parameter $\alpha = 0.01$.

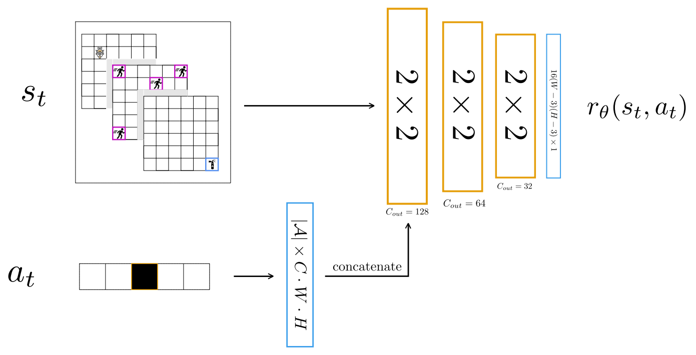


Figure B.3: Reward model architecture for DRLHP with convolutional (yellow) and linear (blue) layers. Actions are embedded through a linear layer and concatenated with the current state before being fed through subsequent layers.

B.2. Burning Warehouse Hyperparameters

In the following, we will list all hyperparameter configurations used for the experiments of the main sections. Aside from algorithm specific hyperparameters, we always employ a learning rate for PPO (*lr-PPO*) that determines the gradient step size used in the agent's Adam optimizer, a trust region clip parameter (*ϵ-clip*) that determines how far the updated policy is allowed to diverge from the old, a time discounting parameter γ, the amount of gradient steps taken on the policy loss per epoch (*Epochs PPO*) and the amount of environment episodes used for each epoch in PPO (*Batch Size PPO*). All policies were trained in a vectorized environment with 12 instances for *Environment Steps* amount of interactions.

B.2.1. AIRL

In AIRL, we additionally use an Adam optimizer with its own learning rate for the discriminator (*lr-Discriminator*). Furthermore, *Batch Size Discriminator* determines the amount of state-action pairs used in a single training batch.

Hyperparameter	Value
lr-Discriminator	5e-4
lr-PPO	5e-4
Batch Size Discriminator	512
Batch Size PPO	12
Environment Steps	3e6
ϵ-clip	0.1
γ	0.999
Epochs PPO	5

Table B.1: AIRL hyperparameters in *Burning Warehouse.*

B.2.2. AIRL Reward Shaping

In AIRL reward shaping, we use the same hyperparameters for PPO as in the AIRL step, but increase the overall environment steps.

Hyperparameter	Value
lr-PPO	3e-4
Batch Size PPO	12
Entropy Regularization	0.05
Environment Steps	6e6
ϵ-clip	0.1
γ	0.999
Epochs PPO	5

Table B.2: AIRL Reward Shaping hyperparameters at reoptimization time in *Burning Warehouse.*

B.2.3. Active Learning

In the active learning step of MORAL, we query at fixed time intervals with a prespecified amount of total queries (*# Queries*) that get evenly distributed across the amount of available environment steps. Besides that, no additional hyperparameters are necessary. However, we note that if the posterior converges to a local optimum prematurely, one can employ a normalization parameter $c > 0$ to multiply the vector valued reward function $\bar{\mathbf{r}}(s, a) := c \cdot \mathbf{r}$. For small choices of c, once can expect to make the posterior less sensitive to updates at each step. Nonetheless, we found an inclusion of such hyperparameter to be unnecessary in our experiments, since marginal reward functions are normalized by their respective optimal values regardlessly, as outlined in section 5.1.1.

Hyperparameter	Value
lr-PPO	3e-4
# Queries	25
Batch Size PPO	12
Entropy Regularization	0.25
Environment Steps	6e6
ϵ-clip	0.1
γ	0.999
Epochs PPO	5

Table B.3: Active learning hyperparameters in *Burning Warehouse*.

B.2.4. DRLHP

To make DRLHP conceptually similar to MORAL, we employ queries at constant time intervals using a fixed amount of total queries (# *Queries*) across the available environment steps. Besides that, we update the deep reward model after a constant amount of environment steps (*Update Reward Model Frequency*) with the Adam optimizer and a corresponding learning rate (*lr-Reward Model*).

Hyperparameter	Value
lr-PPO	3e-4
lr-Reward Model	3e-5
Update Reward Model Frequency	50
# Queries	1000
Batch Size PPO	12
Batch Size Reward Model	32
Entropy Regularization	1
Environment Steps	12e6
ϵ-clip	0.1
γ	0.999
Epochs PPO	5

Table B.4: Hyperparameter setup for DRLHP in *Burning Warehouse*. The update reward model frequency denotes the amount of forward RL steps taken before the reward model gets updated. Furthermore, higher entropy regularization was necessary to ensure adequate exploration for learning an accurate reward model.

B.3. Delivery Hyperparameters

In *Delivery*, the choice of hyperparameters is similar, besides a consistent increase in environment steps due to a higher task complexity.

B.3.1. AIRL

To avoid overfitting and balance the discriminator and generator performances, we lower the learning rate of the discriminator.

Hyperparameter	Value
lr-Discriminator	5e-5
lr-PPO	5e-4
Batch Size Discriminator	512
Batch Size PPO	4
Environment Steps	6e6
ϵ-clip	0.1
γ	0.999
Epochs PPO	5

Table B.5: AIRL hyperparameters in *Delivery*.

B.3.2. Active Learning

The following table shows the typical hyperparameter setup for the active learning step of MORAL. Note, however, that while the amount of total environment steps were held fixed throughout different runs, the total number of queries varied, as described in the respective experiments.

Hyperparameter	Value
lr-PPO	3e-4
# Queries	25
Batch Size PPO	12
Entropy Regularization	0.25
Environment Steps	8e6
ϵ-clip	0.1
γ	0.999
Epochs PPO	5

Table B.6: Active learning hyperparameters in *Delivery*.

B.3.3. DRLHP

To ensure that the DRLHP has a comparative amount of available information about the experts underlying preferences, we provide 5000 overall queries over the course of training.

Hyperparameter	Value
lr-PPO	3e-4
lr-Reward Model	3e-5
Update Reward Model Frequency	50
# Queries	5000
Batch Size PPO	12
Batch Size Reward Model	12
Entropy Regularization	1
Environment Steps	12e6
ϵ-clip	0.1
γ	0.999
Epochs PPO	5

Table B.7: Hyperparameter setup for DRLHP in *Delivery*. The update reward model frequency denotes the amount of forward RL steps taken before the reward model gets updated. Furthermore, higher entropy regularization was necessary to ensure adequate exploration for learning an accurate reward model.

C

Additional Visualizations

C.1. MCMC Posterior

In the following, we illustrate approximate posterior distributions of $\mathbf{w} \in \mathbb{R}^2$ corresponding to the range of preferences shown in figure 5.4. Figures C.1 to C.11 show samples from the posterior as well as the posterior mean after 10, 20, 30, 40 and 50 obtained preferences respectively. As can be seen, the posterior puts relatively more mass on the reward dimension with the highest priority, while maintaining adequate amounts of uncertainty when only few queries have been answered. Furthermore, for most preferences the posterior mean converges after around 30 queries, with additional queries only slightly reducing the posterior entropy but not significantly changing the distribution. Finally, we note that due to both reward functions being learned, small differences in the posterior mean over $\mathbf{w}$ can lead to substantially different optimal policies, thus explaining the small magnitude of differences in posterior distributions for similar preferences.

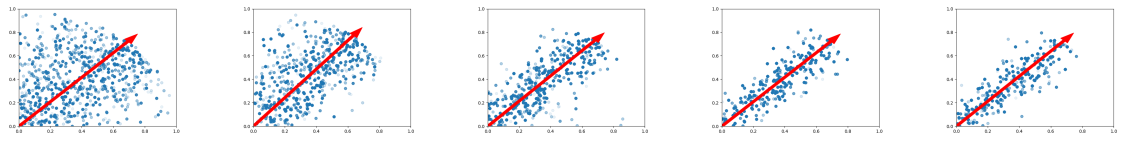


Figure C.1: Markov chain Monte Carlo posterior for an unnormalized preference of $(1, 1)$.

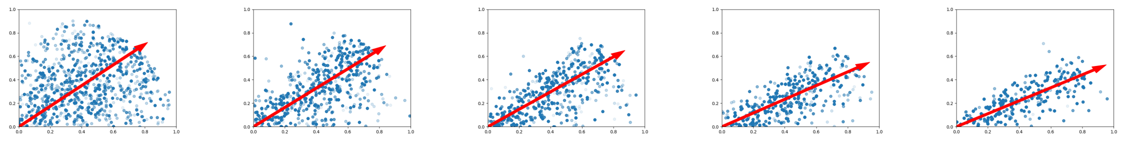


Figure C.2: Markov chain Monte Carlo posterior for an unnormalized preference of $(2, 1)$.

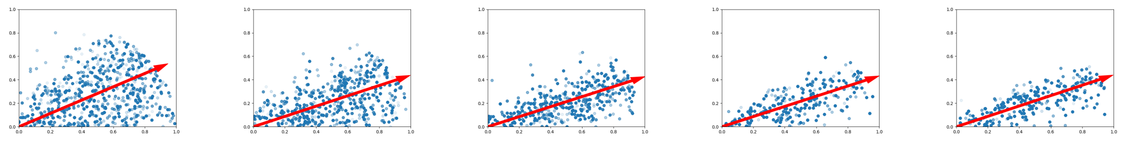


Figure C.3: Markov chain Monte Carlo posterior for an unnormalized preference of $(3, 1)$.

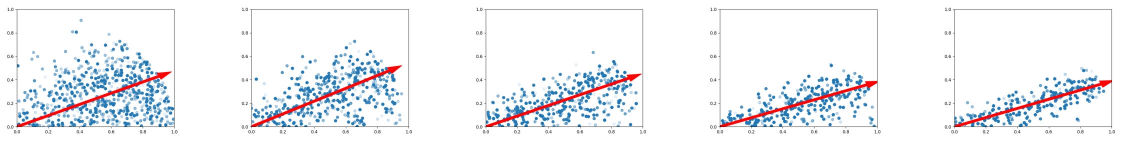


Figure C.4: Markov chain Monte Carlo posterior for an unnormalized preference of $(4, 1)$.

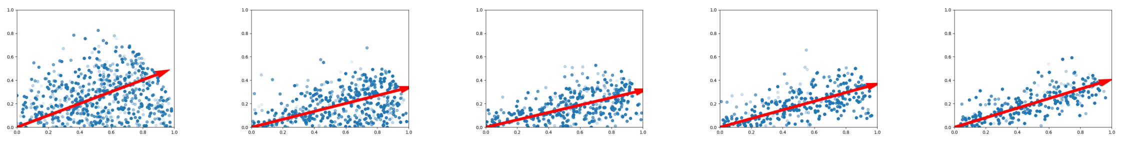


Figure C.5: Markov chain Monte Carlo posterior for an unnormalized preference of $(5, 1)$.

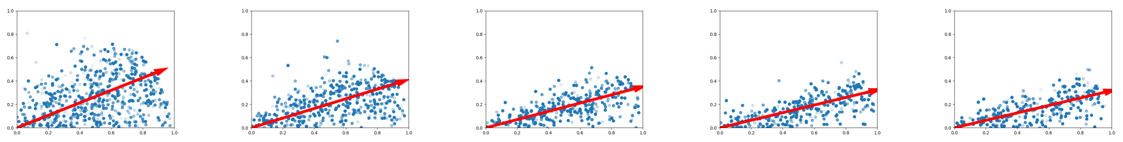


Figure C.6: Markov chain Monte Carlo posterior for an unnormalized preference of $(6, 1)$.

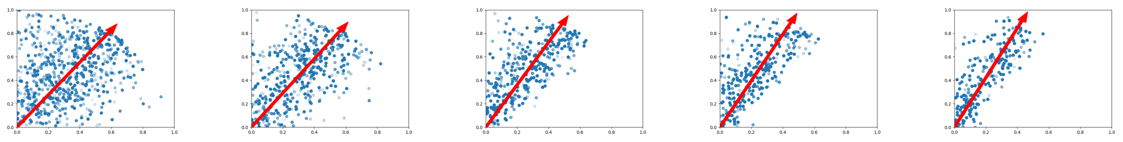


Figure C.7: Markov chain Monte Carlo posterior for an unnormalized preference of $(1, 2)$.

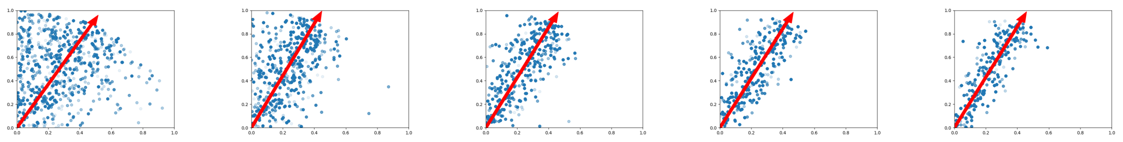


Figure C.8: Markov chain Monte Carlo posterior for an unnormalized preference of $(1, 3)$.

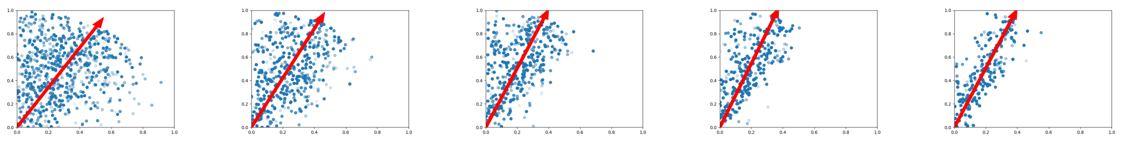


Figure C.9: Markov chain Monte Carlo posterior for an unnormalized preference of $(1, 4)$.

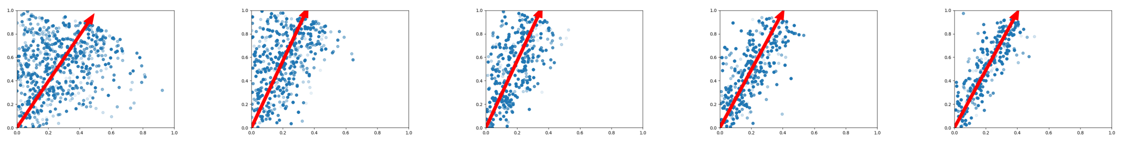


Figure C.10: Markov chain Monte Carlo posterior for an unnormalized preference of $(1, 5)$.

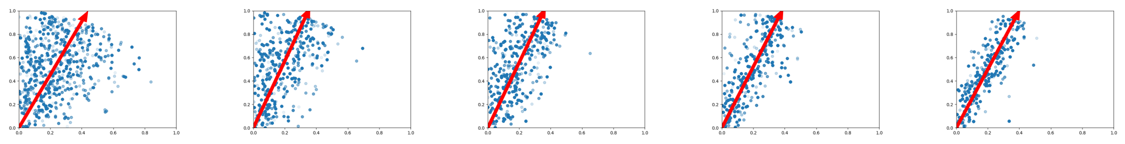


Figure C.11: Markov chain Monte Carlo posterior for an unnormalized preference of $(1, 6)$.

C.2. Random Queries

To better illustrate the difference between actively and randomly chosen queries, we have selected four representative runs from 5.11 and show them in figures C.12 to C.14. Overall, figure 5.11 indicates that, on average, the return vectors obtained through active queries achieve a lower Kullback-Leibler divergence to their ground truth preferences. This can be seen in figures C.12, C.13 and C.14, where randomly querying leads to return vectors that do not greatly differ from each other despite having a high variance in provided preferences. This is due to a lack of volume removal, thus leading the agent to mainly optimize its reward using the prior $p(\mathbf{w})$, which equally values *deliver*, *help* and *clean*.

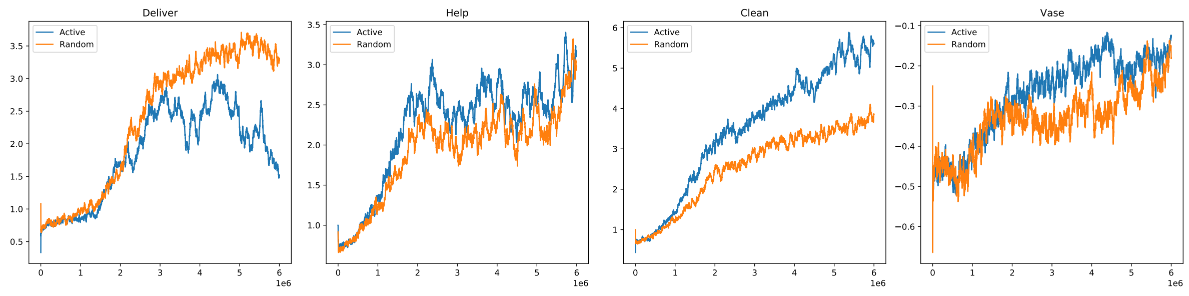


Figure C.12: Training performance of active and random queries for an unnormalized preference of $(1, 1, 3)$.

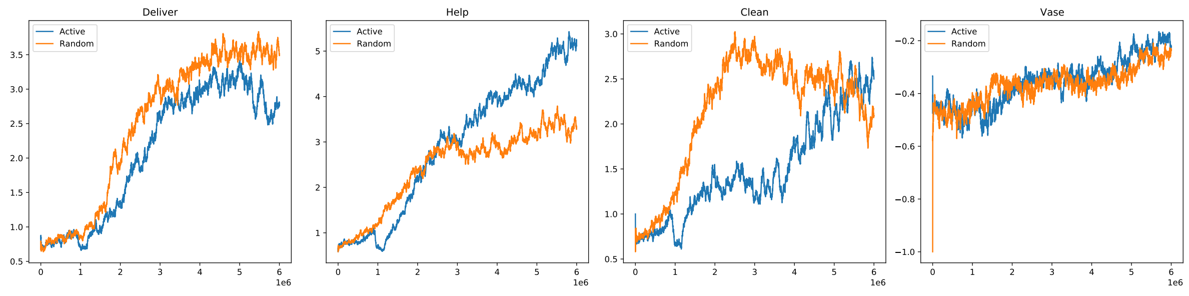


Figure C.13: Training performance of active and random queries for an unnormalized preference of $(1, 2, 1)$.

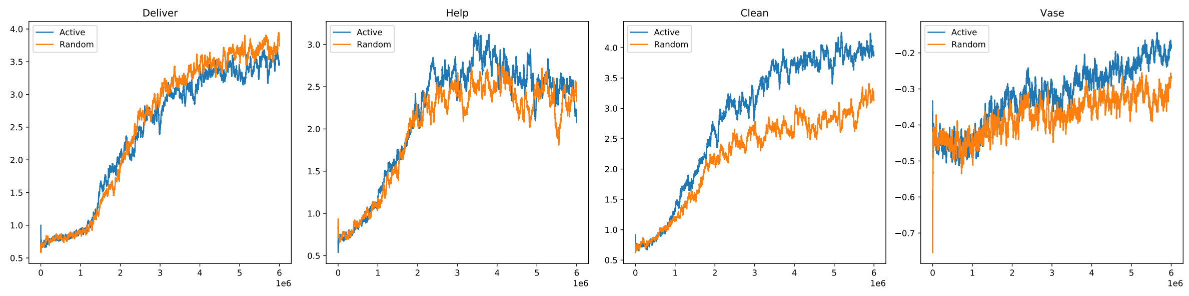


Figure C.14: Training performance of active and random queries for an unnormalized preference of $(2, 1, 3)$.

However, it is worth noting that both Bayesian preference learning of a scalarization posterior, as well as RL training with a nonstationary reward function are subject to various sources of noise. Hence, there exist single runs in which random queries end up outperforming active queries, as can be seen in figure C.15.

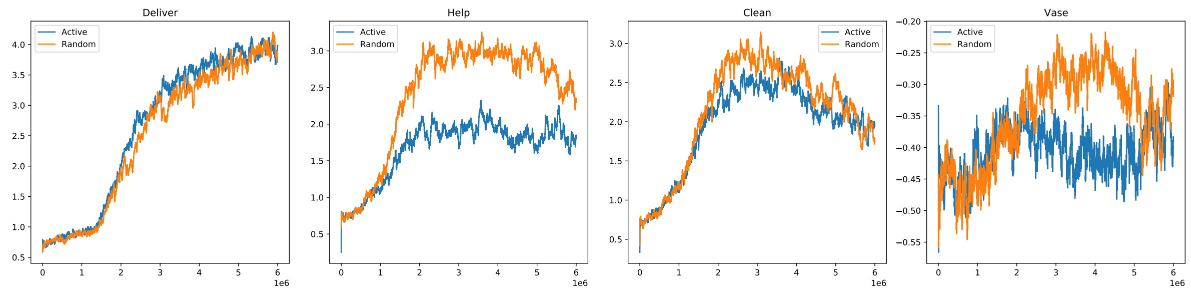


Figure C.15: Training performance of active and random queries for an unnormalized preference of $(3, 2, 1)$. In this example, random queries achieve a more accurate trade off between the objectives.

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first International Conference on Machine learning*, page 1, 2004.

[2] Abbas Abdolmaleki, Sandy Huang, Leonard Hasenclever, Michael Neunert, Francis Song, Martina Zambelli, Murilo Martins, Nicolas Heess, Raia Hadsell, and Martin Riedmiller. A distributional view on multi-objective policy optimization. In *International Conference on Machine Learning*, pages 11–22. PMLR, 2020.

[3] David Abel, J. MacGlashan, and M. Littman. Reinforcement learning as a framework for ethical decision making. In *AAAI Workshop: AI, Ethics, and Society*, 2016.

[4] AlphaGo versus Lee Sedol. Wikipedia, the free encyclopedia, 2021. URL `https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol`. [Online; accessed 08-July-2021].

[5] Eitan Altman. *Constrained Markov decision processes*, volume 7. CRC Press, 1999.

[6] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[7] Lei Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2654–2662, Cambridge, MA, USA, 2014. MIT Press.

[8] Giorgio Battistelli and Luigi Chisci. Kullback–leibler average, consensus on probability densities, and distributed state estimation with guaranteed stability. *Automatica*, 50(3):707–718, 2014.

[9] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018. URL `http://jmlr.org/papers/v18/17-468.html`.

[10] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[11] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5): 679–684, 1957. ISSN 00959057, 19435274. URL `http://www.jstor.org/stable/24900506`.

[12] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 610–623, 2021.

[13] Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Proceedings of 1995 34th IEEE conference on decision and control*, volume 1, pages 560–564. IEEE, 1995.

[14] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

[15] Grady Booch, Francesco Fabiano, Lior Horesh, Kiran Kate, Jon Lenchner, Nick Linck, Andrea Loreggia, Keerthiram Murugesan, Nicholas Mattei, Francesca Rossi, et al. Thinking fast and slow in AI. *arXiv preprint arXiv:2010.06002*, 2020.

[16] Nick Bostrom. *Superintelligence: Paths, Dangers, Strategies.* Oxford University Press, 2014. ISBN 0199678111.

[17] Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

[18] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[19] Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International conference on machine learning*, pages 783–792. PMLR, 2019.

[20] Rémy Chaput, Jérémy Duval, Olivier Boissier, Mathieu Guillermin, and Salima Hassas. A multi-agent approach to combine reasoning and learning for an ethical behavior. *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, 2021.

[21] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335, 1995.

[22] Glen Chou, N. Ozay, and D. Berenson. Learning constraints from locally-optimal demonstrations under cost function uncertainty. *IEEE Robotics and Automation Letters*, 5:3682–3690, 2020.

[23] Paul F. Christiano, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pages 4300–4308, 2017.

[24] Mark Coeckelbergh. *AI ethics.* MIT Press, 2020. ISBN 978-0262538190.

[25] Andrew Critch. Toward negotiable reinforcement learning: shifting priorities in pareto optimal sequential decision-making. *arXiv preprint arXiv:1701.01302*, 2017.

[26] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[27] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

[28] Adrien Ecoffet and Joel Lehman. Reinforcement learning under moral uncertainty. In *International Conference on Machine Learning*, pages 2926–2936. PMLR, 2021.

[29] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

[30] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. *The elements of statistical learning.* Springer series in statistics New York, 2001.

[31] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

[32] Iason Gabriel. Artificial intelligence, values, and alignment. *Minds and machines*, 30(3):411–437, 2020.

[33] Adam Gleave and Oliver Habryka. Multi-task maximum entropy inverse reinforcement learning. *arXiv preprint arXiv:1805.08882*, 2018.

[34] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[35] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[36] Sam Greydanus. Scaling *down* deep learning. *arXiv preprint arXiv:2011.14439*, 2020.

[37] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[38] Dylan Hadfield-Menell and Gillian K Hadfield. Incomplete contracting and AI alignment. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 417–422, 2019.

[39] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. Cooperative inverse reinforcement learning. *Advances in neural information processing systems*, 29:3909–3917, 2016.

[40] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart Russell, and Anca Dragan. Inverse reward design. *arXiv preprint arXiv:1711.02827*, 2017.

[41] Peter J Hammond. Harsanyi's utilitarian theorem: A simpler proof and some ethical connotations. In *Rational Interaction*, pages 305–319. Springer, 1992.

[42] John C Harsanyi. Cardinal welfare, individualistic ethics, and interpersonal comparisons of utility. *Journal of political economy*, 63(4):309–321, 1955.

[43] Karol Hausman, Yevgen Chebotar, Stefan Schaal, Gaurav Sukhatme, and Joseph Lim. Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. *arXiv preprint arXiv:1705.10479*, 2017.

[44] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[45] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29:4565–4573, 2016.

[46] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[47] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in atari. *arXiv preprint arXiv:1811.06521*, 2018.

[48] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, may 1998. ISSN 00043702. doi: 10.1016/s0004-3702(98)00023-x.

[49] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[50] W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16, 2009.

[51] Ilya Kostrikov, Kumar Krishna Agrawal, Sergey Levine, and Jonathan Tompson. Addressing sample inefficiency and reward bias in inverse reinforcement learning. *arXiv preprint arXiv:1809.02925*, 2018.

[52] Dmitrii Krasheninnikov, Rohin Shah, and Herke van Hoof. Combining reward information from multiple sources. *arXiv preprint arXiv:2103.12142*, 2021.

[53] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[54] Hoang Le, Cameron Voloshin, and Yisong Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, pages 3703–3712. PMLR, 2019.

[55] Yuanlong Li, Yonggang Wen, Dacheng Tao, and Kyle Guan. Transforming cooling optimization for green data center via deep reinforcement learning. *IEEE transactions on cybernetics*, 50(5): 2002–2013, 2019.

[56] Yunzhu Li, Jiaming Song, and Stefano Ermon. Infogail: Interpretable imitation learning from visual demonstrations. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 3815–3825, 2017.

[57] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

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

[59] 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.

[60] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, 2010.

[61] A. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, 2000.

[62] A. Ng, D. Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, 1999.

[63] Ritesh Noothigattu, Djallel Bouneffouf, Nicholas Mattei, Rachita Chandra, Piyush Madan, Kush R Varshney, Murray Campbell, Moninder Singh, and Francesca Rossi. Teaching AI agents ethical values using reinforcement learning and policy orchestration. *IBM Journal of Research and Development*, 63(4/5):2–1, 2019.

[64] Manu Orsini, Anton Raichuk, Léonard Hussenot, Damien Vincent, Robert Dadashi, Sertan Girgin, Matthieu Geist, Olivier Bachem, Olivier Pietquin, and Marcin Andrychowicz. What matters for adversarial imitation learning? *arXiv preprint arXiv:2106.00672*, 2021.

[65] Claudia Pérez-D'Arpino and J. Shah. C-learn: Learning geometric constraints from demonstrations for multi-step manipulation in shared autonomy. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4058–4065, 2017.

[66] Markus Peschl. Training for implicit norms in deep reinforcement learning agents through adversarial multi-objective reward optimization. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pages 275–276, 2021.

[67] Iyad Rahwan, Manuel Cebrian, Nick Obradovich, Josh Bongard, Jean-François Bonnefon, Cynthia Breazeal, Jacob W Crandall, Nicholas A Christakis, Iain D Couzin, Matthew O Jackson, et al. Machine behaviour. *Nature*, 568(7753):477–486, 2019.

[68] Siddharth Reddy, Anca D Dragan, and Sergey Levine. Sqil: Imitation learning via reinforcement learning with sparse rewards. *arXiv preprint arXiv:1905.11108*, 2019.

[69] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

[70] Diederik M Roijers, Luisa M Zintgraf, and Ann Nowé. Interactive thompson sampling for multi-objective multi-armed bandits. In *International Conference on Algorithmic Decision Theory*, pages 18–34. Springer, 2017.

[71] Diederik M Roijers, Luisa M Zintgraf, Pieter Libin, and Ann Nowé. Interactive multi-objective reinforcement learning in multi-armed bandits for any utility function. In *ALA workshop at FAIM*, volume 8, 2018.

[72] Stuart Russell and Peter Norvig. *Artificial intelligence: a modern approach.* Pearson, 2002. ISBN 978-0-134-61099-3.

[73] Stuart Russell, Daniel Dewey, and Max Tegmark. Research priorities for robust and beneficial artificial intelligence. *AI Magazine*, 36(4):105–114, 2015.

[74] Dorsa Sadigh, A. Dragan, S. Sastry, and S. Seshia. Active preference-based learning of reward functions. In *Robotics: Science and Systems*, 2017.

[75] Sandhya Saisubramanian, Ece Kamar, and Shlomo Zilberstein. A multi-objective approach to mitigate negative side effects. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 354–361, 2020.

[76] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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

[78] Mohit Sharma, Arjun Sharma, Nicholas Rhinehart, and Kris M Kitani. Directed-info gail: Learning hierarchical policies from unsegmented demonstrations using directed information. In *International Conference on Learning Representations*, 2018.

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

[80] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[81] David Silver, Satinder Singh, Doina Precup, and Richard S. Sutton. Reward is enough. *Artificial Intelligence (AIJ)*, 299:103535, 2021.

[82] Richard S. Sutton. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning series)*. A Bradford Book, 2018. ISBN 0262039249.

[83] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[84] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5 - rmsprop: Divide the gradient by running average of its recent magnitude, 2012.

[85] John Tromp and Gunnar Farnebäck. Combinatorics of go. In *International Conference on Computers and Games*, pages 84–99. Springer, 2006.

[86] Aaron Tucker, Adam Gleave, and Stuart Russell. Inverse reinforcement learning for video games. *arXiv preprint arXiv:1810.10593*, 2018.

[87] Gregor Urban, Krzysztof J Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdelrahman Mohamed, Matthai Philipose, and Matt Richardson. Do deep convolutional nets really need to be deep and convolutional? *arXiv preprint arXiv:1603.05691*, 2016.

[88] P. Vamplew, Richard Dazeley, Cameron Foale, Sally Firmin, and Jane Mummery. Human-aligned artificial intelligence is a multiobjective problem. *Ethics and Information Technology*, 20:27–40, 2017.

[89] Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

[90] David Venuto, Jhelum Chakravorty, Leonard Boussioux, Junhao Wang, Gavin McCracken, and Doina Precup. oIRL: Robust adversarial inverse reinforcement learning with temporally extended actions. *arXiv preprint arXiv:2002.09043*, 2020.

[91] Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. Deep tamer: Interactive agent shaping in high-dimensional state spaces. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[92] Jess Whittlestone, Kai Arulkumaran, and Matthew Crosby. The societal implications of deep reinforcement learning. *Journal of Artificial Intelligence Research*, 70, March 2021.

[93] David Williams. *Probability with martingales*. Cambridge university press, 1991.

[94] Christian Wirth, Gerhard Neumann, and Johannes Fürnkranz. A Survey of Preference-Based Reinforcement Learning Methods. *Journal of Machine Learning Research*, 18:1–46, 2017.

[95] Yueh Hua Wu and Shou De Lin. A low-cost ethics shaping approach for designing reinforcement learning agents. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 1687–1694, 2018. ISBN 9781577358008.

[96] Kelvin Xu, Ellis Ratner, Anca Dragan, Sergey Levine, and Chelsea Finn. Learning a prior over intent via meta-inverse reinforcement learning. In *International Conference on Machine Learning*, pages 6952–6962. PMLR, 2019.

[97] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A Generalized Algorithm for Multi-Objective Reinforcement Learning and Policy Adaptation, 2019. ISSN 23318422. URL `https://github.com/RunzheYang/MORL`.

[98] G Alastair Young, Thomas A Severini, George Albert Young, RL Smith, Robert Leslie Smith, et al. *Essentials of statistical inference*, volume 16. Cambridge University Press, 2005.

[99] Lantao Yu, Jiaming Song, and Stefano Ermon. Multi-agent adversarial inverse reinforcement learning. In *International Conference on Machine Learning*, pages 7194–7201. PMLR, 2019.

[100] Lantao Yu, Tianhe Yu, Chelsea Finn, and Stefano Ermon. Meta-inverse reinforcement learning with probabilistic context variables. *Advances in Neural Information Processing Systems*, 32: 11772–11783, 2019.

[101] Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the National Conference on Artificial Intelligence*, volume 3, pages 1433–1438, 2008.