Extracting and Using Preference Information from the State of the World

Rohin Shah

Technical Report No. UCB/EECS-2020-210
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-210.html

December 17, 2020

Extracting and Using Preference Information from the State of the World

by

Rohin Monish Shah

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Fall 2020

Extracting and Using Preference Information from the State of the World

Abstract

Extracting and Using Preference Information from the State of the World

by

Rohin Monish Shah

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Anca Dragan, Co-chair

Professor Stuart Russell, Co-chair

Typically when learning about what people want and don't want, we look to human action as evidence: what reward they specify, how they perform a task, or what preferences they express can all provide useful information about what an agent should do. This is essential in order to build AI systems that do what we intend them to do. However, existing methods require a lot of expensive human feedback in order to learn even simple tasks. This dissertation argues that there is an additional source of information that is rather helpful: the state of the world.

The key insight of this dissertation is that when a robot is deployed in an environment that humans have been acting in, the state of the environment is already optimized for what humans want, and is thus informative about human preferences.

We formalize this setting by assuming that a human H has been acting in an environment for some time, and a robot R observes the final state produced. From this final state, R must infer as much as possible about H's reward function. We analyze this problem formulation theoretically and show that it is particularly well suited to inferring aspects of the state that should *not* be changed – exactly the aspects of the reward that H is likely to forget to specify. We develop an algorithm using dynamic programming for tabular environments, analogously to value iteration, and demonstrate its behavior on several simple environments. To scale to high-dimensional environments, we use function approximators judiciously to allow the various parts of our algorithm to be trained without needing to enumerate all possible states.

Of course, there is no point in learning about H's reward function unless we use it to guide R's decision-making. While we could have R simply optimize the inferred reward, this suffers from a "status quo bias": the inferred reward is likely to strongly prefer the observed state, since by assumption it is already optimized for H's preferences. To get R to make changes to

the environment, we will usually need to integrate the inferred reward with other sources of preference information. In order to support such reward combination, we use a model in which R must maximize an unknown reward function known only to H. Learning from the state of the world arises as an instrumentally useful behavior in such a setting, and can serve to form a prior belief over the reward function that can then be updated after further interaction with H.

For humanity,

that we may navigate the challenging times ahead.

Contents

List of Figures

List of Tables

Acknowledgments

My journey through grad school has been quite a roller coaster, and I am indebted to many people for making it both enjoyable and successful. For even starting me on this path, I owe my thanks to Ras Bodik. He encouraged me to consider research as an opportunity to explore during undergrad, and later took me on as his PhD student to work on program synthesis. While my dissertation has turned out to be on a completely different topic, I owe my success in this field to the research skills I learned working with Ras.

During those early PhD years, I was especially lucky to be surrounded by the Bodik group: Mangpo Phothilimthana, Sarah Chasins, Julie Newcomb, Ali Koksal, Shaon Barman, and Emina Torlak. While it is only now that I really see how little I knew, and how much I could expect to improve over the course of my PhD, their advice was invaluable in helping me come to terms with my lack of productivity during that first year.

After this year, Ras transferred to the University of Washington, and I followed him a year later. I am especially grateful to the UW PLSE lab for making me feel so welcome; Zach Tatlock, Anna Kornfeld Simpson, and James Wilcox deserve special thanks.

It was around this time that I noticed the significant progress being made in artificial intelligence, and the notable lack of research on how to ensure its safety. I made the difficult decision to switch the focus of my PhD from program synthesis to AI safety, at the Center for Human-Compatible AI (CHAI). Thanks in particular to Ajeya Cotra for several conversations that fed into this decision.

At CHAI, I was especially lucky to work with *three* advisors: Anca Dragan, Stuart Russell and Pieter Abbeel. While I'm not going to cover the wealth of wisdom they have imparted to me, I want to highlight one thing they did that I am especially thankful for: they gave me a large amount of freedom in deciding what I should spend my time on. During my first year at CHAI, I spent most of my time simply reading papers and getting up to speed in my new field of study; at the end of the year I started a newsletter that consumed over ten hours of my time every week. In today's culture of "publish or perish", it feels quite unusual for me to have had this affordance at all.

The greatest benefit of CHAI was the people I got to work with. Thanks in particular to Daniel Filan, Adam Gleave, Michael Dennis, and Dylan Hadfield-Menell for shaping many of my views about AI safety. I would also like to thank Alex Turner, Andrew Critch, Jaime Fisac, Lawrence Chan, Andreea Bobu, Smitha Milli, and Cody Wild for fascinating conversations on the topic. And of course I am indebted to many collaborators, including Ben Cottier, Cynthia Chen, David Lindner, Dmitrii Krasheninnikov, Jordan Alexander, Mark Ho, Micah Carroll, Neel Alex, Noah Gundotra, Paul Knott, Pedro Freire, Rachel Freedman, Richard Ngo, Sam Toyer, Scott Emmons, Sören Mindermann, Steven Wang and Sumith Kulal.

Of course, I don't just owe thanks to those who worked specifically on research with me. I am also indebted to the excellent administrative staff, who made university bureaucracy much, much easier to navigate. Rosie Campbell, Martin Fukui, Lydia Raya and Jean Nguyen were particularly helpful in this regard, but I must also thank Angie Abbatecola, Lena Lau-Stewart, Shirley Salanio, Audrey Sillers, Susanne Kauer, Laura Greenfield, and Logan Baldini.

This PhD would have been so much harder to do without the support of many friends (beyond those I was working with). While it would be quite a challenge to list them all, I want to specifically name Andrew Huang, Anisha Sensa Mauze, Bill Zito, Caroline Jeanmaire, Chigozie Nri, Daniel Ziegler, Davis Foote, Dmitriy Khripkov, Eli Marrone, Ellen Anderson, Jen Chung, Jonathan Mustin, Joseph Kijewski, Nathan Mandi, Ozzie Gooen, Patrick Brinich-Langlois, Richard Yannow, Sean Kelly, Sindy Li, Valmik Prabhu, and Veronica Boyce.

And last but certainly not least, I must thank my parents Lena and Monish, my brother Nihal, and my partner Lynette; they have been remarkably supportive, even as my expected graduation date slipped further and further into the future. I cannot say that they were all thrilled at my choice to pursue a PhD, which makes me all the more grateful that I can nevertheless count on them to support me anyway.

Chapter 1

Introduction

Traditional computer programs are instructions on how to perform a particular task: to compute a factorial, we tell the machine to enumerate the integers from 1 to n, and multiply them together. However, we do not know how to mechanically perform more challenging tasks like translation. The field of artificial intelligence raises the level of abstraction so that we simply show what the task is, and let the machine to figure out how to do it. For instance, we present pairs of sentences which provide examples of translation, and let the machine determine how to translate well.

Unfortunately, as we apply our techniques to increasingly complex tasks, even specifying the task becomes difficult. When we specify reward functions for reinforcement learning by hand, the resulting agents often "game" their reward function by finding solutions that technically achieve high reward without doing what the designer intended. In a particularly famous example, an agent trained to maximize its score in the boat racing game *CoastRunners* got stuck in a loop collecting high-scoring speed boosts instead of winning the game (Clark and Amodei, 2016). Both Lehman et al. (2018) and Krakovna (2018) collect several examples of similar behavior from a variety of sources.

As the environment becomes more complex, it becomes important not just to specify what should be done, but also what should *not be changed* (McCarthy and Hayes, 1981). As put by Stuart Russell, "A system that is optimizing a function of n variables, where the objective depends on a subset of size $k < n$, will often set the remaining unconstrained variables to extreme values; if one of those unconstrained variables is actually something we care about, the solution found may be highly undesirable" (Russell, 2014). A robot that cleans houses must be taught not just the appropriate way to fluff pillows, but also that it should not break any vases on end tables in the course of completing its task (Amodei et al., 2016).

1.1 Why is inaction a safe default?

A common intuition in response to this issue is to suggest that our AI systems should have safety margins: if they are uncertain about what to do, they should default to inaction.

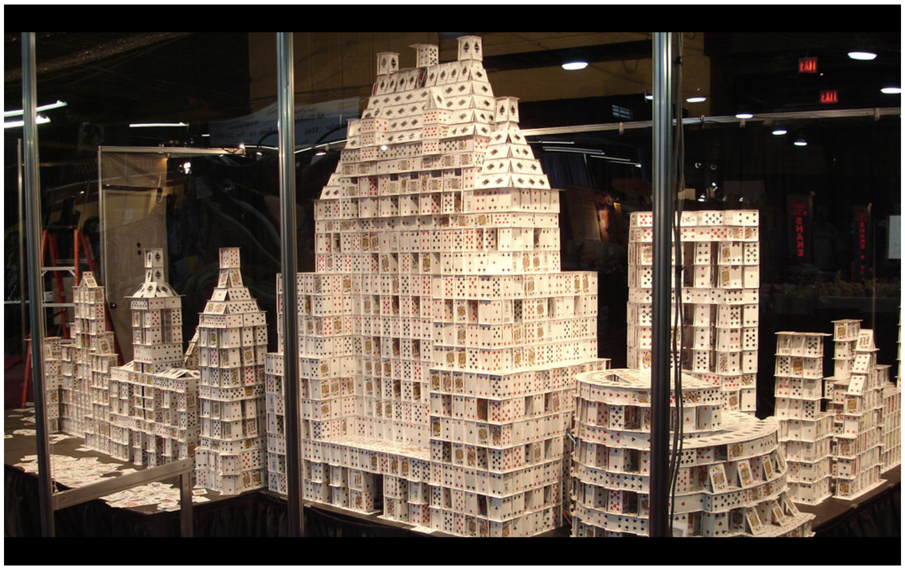


Figure 1.1: Imagine that you walk into a room and you see this giant house of cards. You would likely slow down, be a lot more careful, and stay far away. This is probably not because somebody taught you that it's bad to knock down a house of cards; rather, you can tell that someone must have put a lot of effort into constructing this house of cards, and so they probably care a lot about it. In other words, you have inferred something about what another person cares about, by looking at the state of the room in which they had acted.

This intuition has driven research on *impact penalties*, which aim to ensure that our AI systems do not cause any unnecessary "large" changes (Armstrong and Levinstein, 2017). Researchers have proposed penalizing reduction in the number of reachable states (Krakovna et al., 2018) or attainable utility (Turner et al., 2020). However, such approaches will penalize all irreversible effects, including ones that humans want to occur. Can we instead infer which impacts humans don't want, and penalize only those?

To answer this question, it will help to delve deeper into our assumptions. Why is the "do nothing" action privileged in the first place? Of all the actions that our AI system could take, why is that the safest one?

As a concrete example, let us consider entering the room in Figure 1.1 and seeing the huge house of cards. It is intuitively obvious that we should be very careful around this construction, and seek to avoid knocking it down. This isn't because we were taught that it is bad to knock down houses of cards – even people who had never heard of a house of cards before would probably know not to knock it down. The reason that we intuitively know to avoid messing with the house of cards is that if we were to do so, the person who constructed it would likely be upset. It is obvious that someone had put in a lot of effort into constructing it, and thus they probably care quite a lot that it remains standing.

This shows us how we might infer which impacts are good and which ones are bad: we

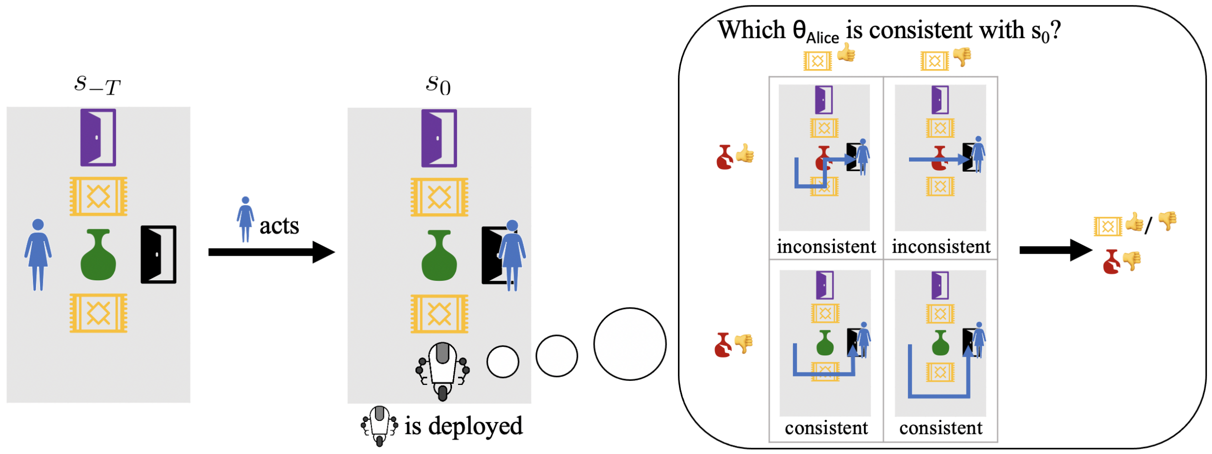


Figure 1.2: Learning preferences from the state of the world. Alice (H) attempts to accomplish a goal in a room with an easily breakable vase in the center. The robot (R) observes the state of the environment, s_0, after H has acted for some time from an even earlier state s_{-T}. R considers multiple possible reward functions that H might have had, and infers that states where vases are intact usually occur when H's reward penalizes breaking vases. In contrast, it doesn't matter much what the reward function says about carpets, as we would observe the same final state either way. Note that while we consider a specific s_{-T} for clarity here, the robot could also reason using a prior over H's initial state s_{-T}.

can notice aspects of the environment that humans have put effort into, and thereby deduce that they care about these aspects. This is our key insight: *when an agent is deployed in an environment that humans have been acting in, the state of the environment is already optimized for human preferences and thus informative.* The heuristic of having agents do nothing by default is appealing precisely because the current state of the world has already been optimized for human preferences, and action is more likely to ruin it than inaction, since inaction preserves the current state.

1.2 Learning from the state of the world

To make this more concrete, such that we could see how to implement it algorithmically, let us consider a simplified example, in which a robot (R) must assist its human user Alice (H). Consider a simple room environment, in which there is a breakable vase placed in the center, as illustrated in Figure 1.2. If either H or R walks over this center square, the vase breaks, and can never be fixed. At the time R is deployed, it observes that the vase is still intact. It can then reason about each of the possible reward functions that H could have:

1. If H wants to break the vase, then she would have walked through the center square to get to the door, breaking the vase in the process. In this case, the robot would observe

a broken vase at deployment, which contradicts the observation that the vase is still intact. Thus, H probably did not actively want to break the vase.

2. If H was indifferent to the vase being broken or intact, then she still would have walked through the center square, simply because that would be the shortest path, and the vase would still be broken. Thus, H was probably not indifferent to the vase either.

3. If H wants to keep the vase intact, then she would have walked around the vase to get to the door. In this world, the robot would observe an intact vase at deployment, which is in fact what it observes.

Only one of these cases is consistent with R's observations, and so R can infer that H probably wants to keep the vase intact. If we did the same reasoning for standing on carpets, all three cases would be consistent with the robot's observation, and so the robot cannot make any inferences about whether H cares about standing on carpets. It should be noted that this assumes that the carpets and vase are fixed, unchangeable features of the room; if we also model the fact that H chose to buy the carpets and place them in the room, R might then infer that H prefers to walk over carpets.

While we have so far talked about how to infer what side effects humans care about, the state of the world can contain information about other preferences as well. For example, if the robot observes a basket full of apples near an apple tree, it can reasonably infer that Alice wants to harvest apples.

1.3 Using the learned reward

Everything we have discussed so far only requires modeling H and the environment she is acting in: at no point have R's actions come into the story. This is because we have been talking about what the robot can *learn* from the state of the world; what the robot can *do* is irrelevant to such learning. Nonetheless, the entire point of learning about a reward function is so that we can then use the learned reward to guide the agent's action selection.

However, here a problem arises: typically, by looking at the state of the world, R will infer what should *not* be changed about the world – for example, vases should not be broken. It is less common (though still possible) for R to learn what changes it *should* make to the world: for any proposed change, if that change were good that wouldn't H have already implemented it? While there are many possible reasons (for example, it may be hard for H but easy for R, or H may not have had the time to do it), there will nonetheless be some information about the reward that R does not learn from the state of the world. We need a way to solicit and incorporate additional information.

The typical paradigm for this is that of *reward learning.* Reward learning splits the overall problem into two parts: first, a learning phase during which the agent gains more information about the reward from external human feedback, and second, an action phase during which the agent acts to maximize expected reward. However, it seems beneficial for the agent to be able to reason about how these phases interact with each other. We recast reward learning as a special case of the *assistance* framework, which models the human as internal to the environment, with some uncertain reward to be inferred. The agent must then act to maximize the (uncertain) reward in this environment. This forces the agent to learn and act simultaneously and jointly, which enables several qualitative benefits.

We thus formalize learning from the state of the world within the assistance paradigm, leading to a model in which H and R are both part of the environment, but R only starts acting and observing after a discrete "deployment" event. At this point, R can update its prior over reward functions using the state of the environment, which gives it some information about the reward. If it still needs additional information, it can get this through more typical mechanisms such as asking H what she wants, or observing how H acts in order to infer her intent. This enables agents that can *assist* humans while learning about what they should do from the *state of the world*, and only asking H questions when it is truly necessary.

1.4 Thesis overview

The central claim of this Ph.D. dissertation is that with only empirical knowledge of how the world works, simply by observing the state of the world, an AI system can make significant inferences about what the human must care about for the world to look as it does, which it can then use to assist the human H. Unlike other methods of learning human preferences, it is not necessary for the human to explicitly provide any feedback to the agent, making this information effectively free.

Central to our approach is the assumption that H has *already* acted in the environment. It is this assumption that implies that the current state of the environment is already optimized towards her preferences. Chapter 3 formalizes this assumption by describing the environment using a Markov Decision Process (MDP) in which H acts. The reward function of the MDP represents what H cares about. The agent then gets to observe a state that is generated by allowing H to act in the environment for some amount of time.

When the MDP is combined with a model of how H chooses actions given a particular reward function, we obtain a generative model that allows us to model H's actions from the reward function, which in turn allows us to model the observed state of the environment from the initial state and H's actions. We can then invert this model in order to make inferences about the reward function from the observed state.

After establishing this formalism, we turn to its theoretical implications in Chapter 4. An immediate limitation is that since we observe only a single state, it is not possible to distinguish between the full set of possible reward functions. A particular challenge is the possibility of the "status quo reward", which assigns maximal reward to the observed state and minimal reward to every other state. Such a reward will often (though not always) have maximal likelihood, since it would strongly incentivize H to seek the observed state. To avoid having these "weird" reward functions dominate our posterior distribution, we restrict the space of reward functions to be linear functions of high-level *features* of the state. This often forces reward to be assigned to multiple states, and requires that states with the same features be assigned the same reward. The introduction of features allows us to formally prove a crucial property of learning from the state of the world: nothing can be inferred about a feature that H could not have affected.

We then turn to efficient algorithms that can perform these inferences in practice. We first consider the setting in which H has been acting *Boltzmann-rationally* (Ziebart et al., 2010) for a known amount of time in a tabular MDP with a prespecified feature function. Chapter 5 derives an algorithm, *Reward Learning by Simulating the Past* (RLSP), by applying dynamic programming to efficiently compute the gradient of the log probability of the observed state with respect to the parameters of the reward function.

However, in practice, our environments are high-dimensional, and we have neither an exact model of the transition dynamics, nor a prespecified function that identifies the relevant features. To extend the algorithm to these kinds of settings, in Chapter 6 we assume access to an environment simulator (or some other source of environment interaction data), and apply self-supervised learning with neural networks to learn models of the transition dynamics and features. We then adapt RLSP to work in this new setting with learned models and large state spaces that can no longer be enumerated, resulting in the *Deep RLSP* algorithm.

What can these algorithms do in practice? In Chapter 7, we test the algorithms in multiple settings to demonstrate their qualitative behaviors. We evaluate both RLSP and Deep RLSP on a suite of gridworlds designed to test their ability to infer negative side effects as well as positive actions that can be taken. To test Deep RLSP in high-dimensional environments, we use it to imitate MuJoCo robot behaviors, given only a small number of states sampled from rollouts of those behaviors.

All of the work until this point primarily focuses on reward learning, using standard algorithms for planning and reinforcement learning in order to then learn good policies that use the inferred reward. However, for the most benefit we would like to continue to learn more information after observing the state of the world, and then use that information to act. We identify two paradigms which allow for this: *reward learning* and *assistance*. Chapter 8 compares and contrasts these two paradigms, and finds that the integration of learning and decision-making in assistance leads to several qualitative benefits. We thus choose to use the assistance paradigm, and show how to formalize learning from the state of the world in an assistance setup.

Chapter 9 concludes by offering a look at how these ideas may be applied to AI systems that are increasingly being deployed in the real world. This dissertation only scratches the surface of what could be done by learning from the state of the world. We are excited to see future research on this topic.

Chapters 4, 5, and 8 can be read after Chapter 3. Chapter 6 depends on Chapter 5, and Chapter 7 depends on both Chapters 5 and 6.

Chapter 2

Background

This chapter provides background, notation and related work to set the context for the rest of this dissertation. It should be possible to skip this chapter and refer back to it when necessary.

2.1 Sequential decision-making

Our key insight is that the human H has already acted in the environment in order to optimize it towards her preferences. In order to formalize this, we need to have the mathematical language to talk about agents that act in environments to achieve goals. This can be done using the formalism of sequential decision-making, in which we model the agent as interacting with an *environment*. Every timestep, the environment is in some particular *state*, and the agent can take an *action* which leads to the environment moving to a new state. To formalize the *goal*, we assume that there is some *reward function* that assigns values to states; an agent pursues the reward function if their actions systematically lead to states with high reward.

Markov Decision Processes (MDPs). Let $\Delta(X)$ be the set of probability distributions over X. Then, a *Markov decision process (MDP)* $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, P_{\mathcal{S}}, \gamma \rangle$ consists of:

- A set of states $\mathcal{S}$, known as the *state space*,
- A set of actions $\mathcal{A}$, known as the *action space*,
- A transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, that given a possible state and the agent's action in that state produces a distribution over the next state of the environment,
- A reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, that given a state, agent action, and the resulting next state, specifies the reward that the agent accrues for this transition,
- An initial state distribution $P_{\mathcal{S}} : \Delta(S)$, that determines the initial state of the environment from which the agent begins to act, and

- A discount rate $\gamma \in (0, 1)$, that specifies how much less valuable reward is in the next timestep than in the current timestep.

An agent in the environment is specified by a *policy* $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, which specifies how the agent will act (perhaps stochastically) in every possible state in the MDP. Given a policy, we can compute the *expected discounted sum of rewards* that an agent would accrue by following the policy from the initial state of the environment:

$$ER(\pi) = \mathop{\mathbb{E}}_{\substack{s_0 \sim P_{\mathcal{S}} \\ a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1}) \right].$$

An *optimal policy* π^* for the MDP is one that maximizes expected reward, that is, $\pi^* \in \operatorname{argmax}_\pi ER(\pi)$.

Partially observable MDPs (POMDPs). One issue with the MDP formalism is that the agent is implicitly assumed to be able to observe the entire state of the environment, since an agent policy $\pi(\cdot \mid s)$ is conditioned on the state s. However, in reality, most agents are not able to observe the entirety of their environment: for example, humans only have directional sight. To address this, we can posit an *observation function*, that given an environment state, specifies what aspects of the state the agent can observe.

Formally, a *partially observable MDP (POMDP)* $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \Omega, O, \mathcal{T}, \mathcal{R}, P_{\mathcal{S}}, \gamma \rangle$ consists of all of the ingredients of an MDP, and adds two more:

- A finite set of observations Ω, known as the *observation space*,
- An observation function $O : \mathcal{S} \to \Delta(\Omega)$, that specifies what aspects of the state an agent can observe. We will write o_t to signify the tth observation $O(s_t)$.

Since the purpose of introducing partial observability was to better model the agent, our policies will have to change. In particular, rather than allowing policies to condition on the *state*, we now allow a policy $\pi : (O \times \mathcal{A})^* \times O \to \Delta(\mathcal{A})$ to condition on the *history of observations and actions*. We now allow a history in order to allow the agent to "remember" previous observations. The expected reward of a policy is nearly unchanged

$$ER(\pi) = \mathop{\mathbb{E}}_{\substack{s_0 \sim P_{\mathcal{S}} \\ a_t \sim \pi(\cdot|o_{0:t}, a_{0:t-1}) \\ s_{t+1} \sim T(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1}) \right],$$

and an optimal policy continues to be one that maximizes expected reward.

Using decision processes for modeling humans

So far, we have been agnostic to what exactly counts as an "agent". When learning from the state of the world, our job is to think about how the human H has acted in order to produce the current state of the world; thus in our context the agent of interest is H. We will denote the action space by $\mathcal{A}^H$ and individual actions by a^H to emphasize this.

Since we typically do not know exactly what goal a human is pursuing, we will often want to specify an *MDP or POMDP without reward* $\mathcal{M}\backslash\mathcal{R}$, also known as a *world model*, which contains all the ingredients of an MDP or POMDP, except for the reward function. Instead, we have a *human model* that specifies how a policy π is chosen given a potential reward function or goal $\mathcal{R}$. Common choices are that humans are perfectly optimal (Ng and Russell, 2000) or noisily rational (Ziebart et al., 2010), though recent work argues that such models should first condense similar trajectories into a single option (Bobu et al., 2020).

The approach of modeling humans using sequential decision-making processes has been used to infer human goals (Ng and Russell, 2000), model human pedagogic behavior (Ho and Ermon, 2016), and infer human internal state (Sadigh et al., 2016), among other applications.

Reinforcement learning and planning

While in this dissertation, we typically *model* the human as an agent in an MDP or POMDP, it is also possible to *build* an artificial agent to optimize reward in a pre-specified MDP or POMDP. In this case, we are given an MDP or POMDP $\mathcal{M}$, and we must find an optimal policy π^* that maximizes expected reward. There is a rich literature in this field going back decades; we refer the reader to Sutton and Barto (2018) for an overview.

Recent work in *deep reinforcement learning* combines function approximation with reinforcement learning (Mnih et al., 2013; Schulman et al., 2017). This has shown great success, leading to expert agents at challenging games like Go (Silver et al., 2017) and Dota (OpenAI, 2018).

2.2 Reward misspecification

However, reinforcement learning comes with a major challenge: since our artificial agent maximizes the specified reward function, it is crucial that the developer specify the reward function correctly. The reward function can be thought of as a specification of how the agent should behave in every possible situation it could ever encounter; it is perhaps not surprising that it is challenging to provide such a specification.

In games, there is a simple reward function that works well: a reward of 1 if the agent wins, and 0 otherwise. Whether or not the agent wins can usually be determined algorithmically. Even in such situations, agents will sometimes find and exploit bugs in the game, such as teleporting through a wall to progress further in Sonic (Hesse et al., 2018), or getting a large amount of points for no apparent reason in Qbert (Chrabaszcz et al., 2018).

However, the real challenge is in specifying reward functions in more complex, open-ended environments. Both Krakovna (2018) and Lehman et al. (2018) catalog a wide variety of creative and surprising ways in which artificial agents found methods of achieving high reward that were a) surprising to the developer and b) not what the developer intended. So far, these surprising behaviors have been primarily limited to sandboxed digital environments, in which these failures are mere curiosities, but they would be actively dangerous in real-world settings such as self-driving cars.

This is not just a problem in which we are unable to effectively communicate our desires to the agent: often, our desires are themselves inconsistent. This is best illustrated in criminal recidivism, in which we would like an AI system to predict whether a criminal is likely to commit another crime after being released, and to do so *fairly*, without showing a bias based on race. When we attempt to write a specification for "fairness", we end up finding two different criteria that are both intuitively necessary properties. Unfortunately, it turns out to be provably impossible to simultaneously these two different criteria, suggesting that a formal fairness specification will remain elusive (Kleinberg et al., 2016; Chouldechova, 2017; Corbett-Davies et al., 2017).

Furthermore, unlike many other potential failures of AI systems, the risks from reward misspecification *grow* as the agent becomes more capable, because with increasing capability agents would presumably become better at finding these surprising behaviors that achieve high reward that we did not anticipate in advance. If we imagine that reward misspecification continues to afflict agents whose capabilities are at par with or exceeding those of humans, it seems that by pursuing instrumental subgoals (Omohundro, 2008) such as resisting shutdown and accumulating resources and power (Turner, 2019), these agents could pose catastrophic risks to human society, potentially leading to human extinction (Bostrom, 2014).

2.3 Minimizing side effects

One perspective on this problem is that the issue is primarily that reward designers cannot be expected to think of everything, and will forget to specify some aspects of what they care about. As described in Chapter 1, if the reward function fails to account for variables that we care about, an optimal agent will tend to set those variables to extreme values, which could be highly undesirable (Russell, 2014).

This is very related to the frame problem in AI (McCarthy and Hayes, 1981), which refers to the issue that we must specify what stays the same in addition to what changes. In formal verification, this manifests as a requirement to explicitly specify the many quantities that the program does not change (Andreescu, 2017). Analogously, rewards are likely to specify what to do (the task), but may forget to say what not to do (the frame properties).

Given this, recent research has focused on solving this problem by finding general methods for penalizing *side effects*, that is, impacts on the environment that are not necessary for completing the primary task (Armstrong and Levinstein, 2017). One approach is to penalize a reduction in the number of reachable states (Krakovna et al., 2018) or attainable utility (Turner et al., 2020). However, such approaches will penalize all irreversible effects, including ones that humans *want*.

By learning from the state of the world, we can instead learn to penalize only those side effects that humans do not want, while allowing effects that are desirable. If H has put effort into ensuring that some effect does not happen, for example preventing the accumulation of dust by cleaning regularly, then we can infer that she must not want that side effect to happen. In contrast, if some effect would not have happened regardless of what H did, then we cannot infer anything about it. This can be a downside: it implies that we could not learn about cases where H is optimized for the environment, rather than optimizing the environment herself. For example, H could probably not change the temperature of the Earth even if she wanted to, and so R could not make any inferences about H's preferences over the temperature of the Earth by looking at the state of the world; nonetheless we would not want R to drastically change the temperature. This suggests that we may want to combine learning from the state of the world with impact penalization in order to achieve the virtues of both.

2.4 Learning from human feedback

A natural solution to reward misspecification is not to specify the reward at all, and to instead build an agent that *infers* the reward function to optimize from human feedback. To the extent that human feedback is inconsistent, the agent would remain *uncertain* about the true objective. As a result, instead of the current model of intelligent agents optimizing for *their* objectives, we would now have beneficial agents optimizing for *our* objectives (Russell, 2019).

Active vs passive. An *active* method is one in which R chooses what feedback H gives, typically by asking questions, whereas in a *passive* method the feedback is specified upfront and cannot be chosen by R. Active methods can allow R to learn from less feedback, since the feedback can be chosen to be most useful for reducing R's uncertainty (Sadigh et al., 2017; Lopes et al., 2009; Mindermann et al., 2018).

Another major difference amongst methods is in the *type* of feedback obtained:

Demonstrations. A common form of feedback is *demonstrations*, in which H shows (from R's perspective, perhaps using teleoperation) how to accomplish the desired task. R can then either use imitation learning to directly learn a policy that *imitates* the demonstrations (Bain and Sammut, 1999; Ho and Ermon, 2016), or use inverse reinforcement learning (IRL) to infer what reward function H must have in order to produce the demonstrations (Ng and Russell, 2000; Ziebart et al., 2010; Ramachandran and Amir, 2007; Fu et al., 2017; Finn et al., 2016). Using ranked demonstrations can improve performance (Brown et al., 2019).

One issue with learning from demonstrations is that we require the demonstrations to have action labels. For imitation learning, these action labels must come from R's action space $\mathcal{A}^R$, even though they are presumably demonstrated by H. In the case of inverse reinforcement learning, the action labels can be from any action space, as long as R has a model of the environment from the perspective of agents with that action space. (If the action space is not $\mathcal{A}^R$, then the inferred reward function must not depend on the action, if we want to use it to learn a policy for R.)

Observations. Since action labels can be fairly challenging to get, and since demonstrations without action labels can often be plentiful (such as YouTube videos), we would like to learn from demonstrations without action labels. Learning from Observations (LfO; Yu et al., 2018; Torabi et al., 2018; Gandhi et al., 2019; Edwards et al., 2018) aims to recover a policy from such demonstrations. This makes it much more generally applicable than learning from demonstrations, though it still requires the demonstrations to have observations from R's perspective, or from a different perspective for which R has a model of how the observations are produced.

Comparisons. Some tasks may be sufficiently difficult that H cannot perform them all, making demonstrations infeasible. However, in such situations, H may still be able to *evaluate* whether or not a particular behavior is good. In this case, we can ask H to compare which of two behaviors is better, and fit a reward model that predicts these comparisons well (Christiano et al., 2017; Sadigh et al., 2017; Wirth et al., 2017).

Other sources. Many other potential sources of information have been proposed; we refer the reader to Jeon et al. (2020) for an overview. Some sources not yet mentioned include ratings (Daniel et al., 2014), reinforcement signals (Knox and Stone, 2009; Warnell et al., 2017; MacGlashan et al., 2017), proxy rewards (Hadfield-Menell et al., 2017b), corrections (Bajcsy et al., 2017), proxy rewards (Hadfield-Menell et al., 2017b), instructions (Bahdanau et al., 2019), natural language (Fu et al., 2019), summary statistics of demonstrations (Kangasrääsiö and Kaski, 2018), or combinations of multiple feedback types (Ibarz et al., 2018).

This dissertation proposes a new type of human feedback: the state of the world in which H has been operating. Unlike most of the other sources of feedback described above, the state of the world does not require any active effort on the part of H that they weren't already planning to do; R is supposed to learn simply by observing the state when it is deployed. For this reason, it is fundamentally a passive method. While one could formalize and build algorithms for an active variant of learning from the state of the world, it is not clear why we would want to do so. If we are going to ask H to act optimally in a new environment, rather than one she had already been acting in, we might as well observe her full demonstration, rather than limiting ourselves to just the final state.

2.5 Maximum Causal Entropy Inverse Reinforcement Learning

In this section, we explain a particular algorithm for inverse reinforcement learning, called Maximum Causal Entropy IRL (MCEIRL) (Ziebart et al., 2010). This is both a typical example of an algorithm that learns from human feedback, and an explanation of an algorithm and human model that we will be using in Chapters 5 and 6.

As a passive IRL algorithm, MCEIRL aims to learn a reward function that explains a set of demonstrations given as input. We assume that the reward function is *linear in features*, that is, it is of the form $\mathcal{R}_\theta(s) = \theta^T f(s)$, where θ is a set of parameters that defines the reward function, and f is a feature function that extracts a set of features of the state that may be relevant to the reward.

MCEIRL models H as *Boltzmann rational*, that is, she is assumed to be more likely to take actions if they are of higher value, but at least some probability is assigned to all actions. Concretely, H is assumed to act according to the policy computed by soft value iteration:

$$\begin{aligned}
\pi_t^{\text{soft}}(a^H \mid s, \theta) &= \exp(Q_t(s, a^H; \theta) - V_t(s; \theta)) \\
V_t(s; \theta) &= \ln \sum_{a^H} \exp(Q_t(s, a^H; \theta)) \\
Q_t(s, a^H; \theta) &= \theta^T f(s) + \sum_{s'} \mathcal{T}(s' \mid s, a^H) V_{t+1}(s'; \theta) \\
V_{T+1}(s; \theta) &= 0.
\end{aligned}$$

Here, V_t plays the role of a normalizing constant for the policy at timestep t. Intuitively, H is assumed to act close to randomly when the difference in expected total reward across the actions is small, but nearly always chooses the best action when it leads to a substantially higher expected return. The soft Bellman backup for the state-action value function Q is similar to the standard Bellman backup, with the hard maximum over actions replaced with a soft maximum (logsumexp) over actions.

The likelihood of a trajectory $\tau = s_0 a_0^H \dots s_T a_T^H$ given the reward parameters θ is:

$$p(\tau \mid \theta) = p(s_0) \left(\prod_{t=0}^{T-1} \mathcal{T}(s_{t+1} \mid s_t, a_t^H) \pi_t^{\text{soft}}(a_t^H \mid s_t, \theta) \right) \pi_T^{\text{soft}}(a_T^H \mid s_T, \theta). \tag{2.1}$$

MCEIRL finds the reward parameters θ^* that maximize the log-likelihood of the demonstrations:

$$\theta^* = \text{argmax}_\theta \ln p(\mathcal{D} \mid \theta) = \text{argmax}_\theta \sum_i \sum_t \ln \pi_t^{\text{soft}}(a_{i,t}^H \mid s_{i,t}, \theta). \tag{2.2}$$

Ziebart et al. (2010) shows that θ^* gives rise to a policy whose feature expectations match those of the expert demonstrations. It proposes a gradient ascent algorithm for solving this problem, and derives the following gradient:

$$\nabla_\theta \ln p(\mathcal{D} \mid \theta) = \left(\sum_i \sum_{t=0}^{T} f(\tau_{i,t}) \right) - \left(\mathop{\mathbb{E}}_{\substack{s_0 \sim P_\mathcal{S} \\ a^H_{0:T-1} \sim \pi^{\text{soft}}_\theta \\ s_{1:T} \sim \mathcal{T}}} \left[\sum_{t=0}^{T} f(s_t) \right] \right), \tag{2.3}$$

where $\tau_{i,t}$ is the tth timestep of the ith demonstration in $\mathcal{D}$.

Unfortunately, the derivation for this gradient assumes access to the expert policy. The feature counts for the expert policy are then approximated with the feature counts of the demonstrations. This is an unbiased estimator of the true gradient, but in settings with limited data the variance can be extremely high. So, we derive a gradient directly from the assumption that the trajectory was obtained by rolling out a policy computed using soft value iteration.

Deriving an exact gradient

Given a trajectory $\tau_T = s_0 a^H_0 \dots s_T a^H_t$, we seek the gradient $\nabla_\theta \ln p(\tau_T \mid \theta)$. We assume that the expert has been acting according to the maximum causal entropy IRL model given above. In the following, unless otherwise specified, all expectations over states and actions use the probability distribution over trajectories from this model, starting from the state and action just prior. For example, given s_{t-1} and a_{t-1}, we have:

$$\mathop{\mathbb{E}}_{s_t, a^H_t} \left[X(s_t, a^H_t) \right] = \sum_{\substack{s_t \in \mathcal{S} \\ a^H_t \in \mathcal{A}^H}} \mathcal{T}(s_t \mid s_{t-1}, a^H_{t-1}) \pi^{\text{soft}}_t (a^H_T \mid s_T, \theta) X(s_T, a^H_T).$$

First, we compute the gradient of $V_t(s;\theta)$. We have $\nabla_\theta V_{T+1}(s;\theta) = 0$, and for $0 \le t \le T$:

$$\begin{aligned}
&\nabla_\theta V_t(s_t;\theta)\\
&= \nabla_\theta \ln \sum_{a_t^{H'}} \exp(Q_t(s_t, a_t^{H'};\theta))\\
&= \frac{1}{\exp(V_t(s_t;\theta))} \sum_{a_t^{H'}} \exp(Q_t(s_t, a_t^{H'};\theta)) \nabla_\theta Q_t(s_t, a_t^{H'};\theta)\\
&= \frac{1}{\exp(V_t(s_t;\theta))} \sum_{a_t^{H'}} \exp(Q_t(s_t, a_t^{H'};\theta)) \nabla_\theta \left[\theta^T f(s_t) + \mathop{\mathbb{E}}_{s'_{t+1}} \left[V_{t+1}(s'_{t+1};\theta)\right]\right]\\
&= \sum_{a_t^{H'}} \exp(Q_t(s_t, a_t^{H'};\theta) - V_t(s_t;\theta)) \left[f(s_t) + \mathop{\mathbb{E}}_{s'_{t+1}} \left[\nabla_\theta V_{t+1}(s'_{t+1};\theta)\right]\right]\\
&= \sum_{a_t^{H'}} \pi_t^{\text{soft}}(a_t^{H'} \mid s_t, \theta) \left[f(s_t) + \mathop{\mathbb{E}}_{s'_{t+1}} \left[\nabla_\theta V_{t+1}(s'_{t+1};\theta)\right]\right]\\
&= f(s_t) + \mathop{\mathbb{E}}_{a_t^{H'}, s'_{t+1}} \left[\nabla_\theta V_{t+1}(s'_{t+1})\right].
\end{aligned}$$

Unrolling the recursion, we can prove using induction that the gradient is the expected feature counts under the policy implied by θ from s_t onwards. If we define

$$\mathcal{F}_t(s_t;\theta) \triangleq f(s_t) + \mathop{\mathbb{E}}_{a^{H'}_{t:T-1}, s'_{t+1:T}} \left[\sum_{t'=t+1}^{T} f(s'_{t'})\right],$$

then we have

$$\nabla_\theta V_t(s_t;\theta) = \mathcal{F}_t(s_t;\theta).$$

This makes intuitive sense: since the reward is linear in features, for a single state we have $\nabla_\theta \mathcal{R}_\theta(s) = \nabla_\theta \theta^T f(s) = f(s)$, and so for the value of a state $V_t(s_t;\theta)$ the gradient simply sums up the expected feature counts for the rest of the trajectory when acting according to π^{soft}.

We can now calculate the gradient we actually care about:

$$\begin{aligned}
&\nabla_\theta \ln p(\tau \mid \theta) \\
&= \nabla_\theta \left[\ln P_{\mathcal{S}}(s_0) + \sum_{t=0}^{T} \ln \pi_t^{\text{soft}}(a_t^H \mid s_t, \theta) + \sum_{t=0}^{T-1} \ln \mathcal{T}(s_{t+1} \mid s_t, a_t^H) \right] \\
&= \sum_{t=0}^{T} \nabla_\theta \ln \pi_t^{\text{soft}}(a_t^H \mid s_t, \theta) && \text{only } \pi_t^{\text{soft}} \text{ depends on } \theta \\
&= \sum_{t=0}^{T} \nabla_\theta \left[Q_t(s_t, a_t^H; \theta) - V_t(s_t; \theta) \right] \\
&= \sum_{t=0}^{T} \nabla_\theta \left[\theta^T f(s_t) + \mathop{\mathbb{E}}_{s'_{t+1}} \left[V_{t+1}(s'_{t+1}; \theta) \right] - V_t(s_t; \theta) \right] \\
&= \sum_{t=0}^{T} \left(f(s_t) + \mathop{\mathbb{E}}_{s'_{t+1}} \left[\mathcal{F}_{t+1}(s'_{t+1}; \theta) \right] - \mathcal{F}_t(s_t; \theta) \right).
\end{aligned}$$

The last term of the summation is $f(s_T) + \mathbb{E}_{s'_{T+1}} \left[\mathcal{F}_{T+1}(s'_{T+1}; \theta) \right] - \mathcal{F}_T(s_T; \theta)$, which simplifies to $f(s_T) + 0 - f(s_T) = 0$, so we can drop it. Thus, our gradient is:

$$\nabla_\theta \ln p(\tau \mid \theta) = \sum_{t=0}^{T-1} \left(f(s_t) + \mathop{\mathbb{E}}_{s'_{t+1}} \left[\mathcal{F}_{t+1}(s'_{t+1}; \theta) \right] - \mathcal{F}_t(s_t; \theta) \right). \tag{2.4}$$

The term inside the summation has an intuitive meaning: the first two terms together compute the expected feature counts if we take action a_t^H in s_t and then follow π^{soft}, while the last term computes the expected feature counts if we always follow π^{soft} from state s_t. It is thus analogous to the advantage of the demonstrated action a_t^H over the action recommended by the policy π^{soft}. We define $g(s_t, a_t^H; \theta)$ to represent this quantity:

$$g(s_t, a_t^H; \theta) \triangleq f(s_t) + \mathop{\mathbb{E}}_{s'_{t+1}} \left[\mathcal{F}_{t+1}(s'_{t+1}; \theta) \right] - \mathcal{F}_t(s_t; \theta) \tag{2.5}$$

This lets us rewrite our gradient as

$$\nabla_\theta \ln p(\tau \mid \theta) = \sum_{t=0}^{T-1} g(s_t, a_t^H; \theta). \tag{2.6}$$

This is the gradient we will use in Chapter 5, but a little more manipulation allows us to compare with the gradient provided in Ziebart et al. (2010).

Comparing the two gradient expressions

We reintroduce the terms that we cancelled above:

$$\begin{aligned}\nabla_\theta \ln p(\tau \mid \theta) &= \sum_{t=0}^{T} \left(f(s_t) + \mathop{\mathbb{E}}_{s'_{t+1}} \left[\mathcal{F}_{t+1}(s'_{t+1}; \theta) \right] - \mathcal{F}_t(s_t; \theta) \right) \\ &= \left(\sum_{t=0}^{T} f(s_t) \right) + \left(\sum_{t=0}^{T-1} \mathop{\mathbb{E}}_{s'_{t+1}} \left[\mathcal{F}_{t+1}(s'_{t+1}; \theta) \right] \right) - \left(\mathcal{F}_0(s_0) + \sum_{t=0}^{T-1} \mathcal{F}_{t+1}(s_{t+1}; \theta) \right) \\ &= \left(\sum_{t=0}^{T} f(s_t) \right) - \mathcal{F}_0(s_0; \theta) + \sum_{t=0}^{T-1} \left(\mathop{\mathbb{E}}_{s'_{t+1}} \left[\mathcal{F}_{t+1}(s'_{t+1}; \theta) \right] - \mathcal{F}_{t+1}(s_{t+1}; \theta) \right).\end{aligned}$$

Compare this to Equation 2.3. The first term matches our first term exactly. Our second term matches the second term in the limit of sufficiently many demonstrations. Our third term converges to zero with as the number of demonstrations increases, since as the number of s_t, a_t^H pairs increases, the empirical counts of s_{t+1} will match the expected proportions prescribed by $\mathcal{T}(\cdot \mid s_t, a_t^H)$, leading to a smaller correction.

In a deterministic environment, we have $\mathcal{T}(s'_{t+1} \mid s_t, a_t^H) = 1[s'_{t+1} = s_{t+1}]$ since only one transition is possible. Thus, the third term is zero and even for one trajectory the gradient reduces to $\left(\sum_{t=0}^{T} f(s_t)\right) - \mathcal{F}_0(s_0; \theta)$. This differs from the gradient in Ziebart et al. (2010) only in that it computes feature expectations from the observed starting state s_0 instead of using the initial state distribution $P_{\mathcal{S}}(s_0)$.

In a stochastic environment, the third term need not be zero, and corrects for the "bias" in the observed states s_{t+1}. Intuitively, when the expert chose action a_t^H, she did not know which next state s'_{t+1} would arise, but the first term of our gradient upweights the particular next state s_{t+1} that we observed. The third term downweights the future value of the observed state and upweights the future value of all other states, all in proportion to their transition probability $\mathcal{T}(s'_{t+1} \mid s_t, a_t^H)$.

Chapter 3

Formalism: Defining a state of the world problem

In this chapter, we will describe how to mathematically formalize the problem of learning from the state of the world, and describe some "free variables" that can be varied to produce different formulations that have slightly different properties.

3.1 Problem statement

Let us consider the example of a room with a breakable vase from Chapter 1 (Figure 1.2). Recall that in this example, Alice (H) has been walking around in a room that contains a vase and carpets. Intuitively, when the robot (R) observes that the vase is still intact, it should infer that H does not want the vase to be broken, since in any other case R would likely have observed a broken vase.

We focus for now on the case where R is only learning about the reward, rather than acting based on the learned reward. Thus, all we need to specify for R is a parameterized hypothesis space over reward functions combined with an initial prior over likely reward functions, given by $\langle \Theta, \mathcal{R}_\theta, P_\Theta \rangle$. In contrast, for H we do need to model her behavior, and so we model the environment as an MDP without reward $\mathcal{M}\backslash\mathcal{R} = \langle \mathcal{S}, \mathcal{A}^H, \mathcal{T}, \gamma, P_\mathcal{S} \rangle$, in which H pursues expected reward $\mathrm{ER}(\pi^H)$ for some unknown reward function.

In the room with vase example, we would have an MDP in which $\mathcal{S}$ consists of combinations of H's possible locations and whether the vase is broken, $\mathcal{A}^H$ includes up, down, left and right, $\mathcal{T}$ simply moves H according to the selected action, and then breaks the vase if H is standing on it, and $P_\mathcal{S}$ specifies that H starts to the left of the vase. For R, we could assume that it is already known that the reward depends linearly on four features: whether the vase is broken, whether H is standing on a carpet, whether H is at the black door, or whether H is at the purple door. Thus, we would have $\Theta = \mathbb{R}^4$ with $\mathcal{R}_\theta(s) = \theta^T f(s)$, where $f(s)$ produces a vector of the four features described above. The prior P_Θ could be a Gaussian distribution centered at $\theta = \vec{0}$. One possible true reward $\mathcal{R}_{\theta^*}$ for H could assign negative reward to breaking the vase and positive reward for going to the black door.

We assume that R observes a single state $s \in \mathcal{S}$. Since this is meant to occur at the time that R is deployed, we will call this timestep 0, and thus write the observed state as s_0. R's goal is then to compute $P(\theta \mid s_0)$. However, in order to do this, we need to have some assumption on how s_0 is connected to θ. Since the state of the world is informative of human preferences *because* humans have already optimized it, our model is that s_0 was produced by H acting in the environment for the last T timesteps, for some T. If we are given a human model π^H, then we get a generative model of s_0 as follows:

$$P(s_0 \mid \theta) = \sum_{\substack{s_{-T}, \cdots s_{-2}, s_{-1} \in \mathcal{S} \\ a^H_{-T}, \cdots a^H_{-2}, a^H_{-1} \in \mathcal{A}^H}} P_{\mathcal{S}}(s_{-T}) \prod_{t=-T}^{-1} \pi^H(a^H_t \mid s_t, \theta) \mathcal{T}(s_{t+1} \mid s_t, a^H_t). \tag{3.1}$$

Notice that Equation 3.1 involves marginalizing over the possible past trajectories that "could have happened". Note that not every sequence of past states and actions is feasible: the term $\mathcal{T}(s_0 \mid s_{-1}, a^H_{-1})$ enforces that it must at least be possible to transition to the state that we actually observe, s_0. This enforces a strong constraint on how much different past trajectories are weighted: for example, in our room with vase environment, any trajectory that ever breaks a vase would have zero weight and would not contribute to the likelihood at all, since $\mathcal{T}(s_0 \mid s_{-1}, a^H_{-1})$ would be zero if s_{-1} contains a broken vase.

The problem is then to invert this generative model to get a posterior over reward functions:

$$P(\theta \mid s_0) \propto P(s_0 \mid \theta) P_\Theta(\theta). \tag{3.2}$$

Putting it all together, we get the following problem statement:

Definition 1. *A* state of the world problem *is a tuple $\langle \mathcal{M} \backslash \mathcal{R}, s_0, \pi^H, T, \langle \Theta, \mathcal{R}_\theta, P_\Theta \rangle \rangle$, where $\mathcal{M} \backslash \mathcal{R}$ is an MDP without reward that specifies H's environment, s_0 is the observed state after H acts in $\mathcal{M}$ for T timesteps following policy π^H, and P_Θ is R's prior over the parameters $\theta \in \Theta$ for the parameterized reward function $\mathcal{R}_\theta$.*

3.2 Choice of problem parameters

In the rest of this chapter, we discuss the implications of particular choices of the parameters in a state of the world problem.

Human model

Given that we have an MDP $\mathcal{M}$ for H, it may seem odd that we also require a policy π^H for her. Why not just assume that she acts according to the optimal policy of $\mathcal{M}$?

This would correspond to an assumption of perfect rationality, but we know that humans are suboptimal and biased (Kahneman, 2011). In areas of AI research that require a human model, there is active research on how exactly to construct such a model, with approaches including perfect rationality (Ng and Russell, 2000), Boltzmann or noisy rationality (Ziebart et al., 2010; Bobu et al., 2020), modeling specific biases (Evans et al., 2016; Shah et al., 2019a; Majumdar et al., 2017), handling humans learning over time (Chan et al., 2019; Jacq et al., 2019), and using machine learning (Bourgin et al., 2019; Carroll et al., 2019; Choudhury et al., 2019).

The choice of π^H is significant, as it can radically change the interpretation of a particular piece of observed evidence. For example, suppose a household robot observes that the fire extinguisher has been used up, and much of the furniture is significantly charred. If R assumes that H was following an optimal policy, it would have to conclude that H loves to extinguish fires. If we instead have R model H as error-prone, or still learning about how to work with fire, then the true explanation presents itself: H accidentally set the house on fire, but she would prefer it if her house was not on fire.

Rather than baking in a specific human model into our formalism, we instead leave it as a free parameter that must be specified upfront. In future chapters, we shall derive theory and algorithms for specific common choices of π^H.

Time horizon

Another parameter of interest in the problem statement is the *time horizon* T. This is the number of actions that we assume H has taken in the MDP $\mathcal{M}$. In addition to positive integers, we also allow setting $T = \infty$. In this case, s_0 is assumed to have been drawn from the stationary distribution over states when executing the policy π^H on the MDP $\mathcal{M}$.

Like the human model, T can have a significant impact on R's inferences:

Incorrectly learning what to do. If T is too small, then it will appear to R that H managed to accomplish quite a lot in a small amount of time, suggesting that H cares a lot about whatever features that R can observe, even if in reality H only has weak preferences and puts in a bare minimum of effort. For example, if the house is a little dusty but still mostly clean, but R incorrectly assumes that H has only had half an hour to act, then R would infer that H cares strongly about having a clean house, and simply has not yet had the time to finish the job. In extreme cases, R might conclude that *no* H trajectories could explain the evidence it observes, and then $P(\theta \mid s_0)$ would be undefined (as we would be conditioning on an event with zero probability).

Conversely, if T is too large, R may underestimate the fraction of time that H has put into a task, and so underestimate how much H cares about that task.

Incorrectly learning what not to do. When we consider properties that H *avoids*, the conclusions reverse. If T is too small, then R may think that H has simply not had the chance to do some action, rather than specifically avoiding that action. For example, in our room with vase example, if R assumes that H has only been walking around the room for a short period of time, then it may conclude that H's path might never have crossed the vase, and so the vase is only intact due to random luck, and so it may still be the case that the vase can be broken. Conversely, if T is too large, R may conclude that H has been taking more care to avoid actions than she actually is.

Chapter 4

Theory: What can we learn from the state of the world?

We first theoretically analyze what we can qualitatively learn from the state of the world, before delving into algorithmic approaches. We study three different qualitative behaviors that emerge when learning from the state of the world.

First, we expect that learning from the state of the world will tend to encourage preservation of the the observed state s_{observed}: after all, our original motivation for learning from the state of the world was that it explains why the "do nothing" action is a safe default. Section 4.2 formalizes this connection by proving that "preservation" reward functions that incentivize doing nothing are maximum likelihood estimates of the reward in many (though not all) circumstances.

Second, since our key assumption is that s_{observed} is optimized by H towards her preferences, the information we can extract is limited by what H could have done in the past. In Section 4.3 we prove that if there is no policy that H could have taken to affect some aspect of the environment, then we cannot learn anything about H's preferences over that aspect of the environment.

Finally, in our running vase example of Figure 1.2, we can make a strong inference that breaking the vase is dispreferred. Intuitively, this is because breaking the vase is an irreversible event, and so just from s_{observed} we can infer that the vase was *never* broken over the last T timesteps, which is unlikely to be a coincidence. Is it generally the case that if an irreversible event has not happened in s_{observed}, then we can conclude that that event is dispreferred? In Section 4.4, we show that the answer is no: it is possible that the irreversible event would have happened in a different counterfactual world, in which case it cannot be arbitrarily strongly dispreferred.

Setting. For the sake of simplifying the theory, we would prefer not to deal with arbitrary time horizons T, and so we work with the infinite-horizon case, that is $T = \infty$. This implies that the observed state s_{observed} is drawn from the stationary distribution of a rollout from H's policy.

We will assume that the environment is both finite and *ergodic*, that is, for every possible policy π^H, the Markov chain induced by the policy in the environment is ergodic. In this case, the H's policy π^H uniquely determines the stationary distribution over states when executing the policy. In particular, it does not depend on the initial state distribution $P_{\mathcal{S}}$. We will additionally assume that H is executing an optimal policy for her reward function $\mathcal{R}_{\theta^*}$.

We make these assumptions primarily to simplify the theoretical analysis, rather than because they will be literally true. We expect that the insights generated in this setting will still apply to other settings in which our assumptions do not hold perfectly.

4.1 Formalism

We make the discussion above more concrete and introduce notation to describe it.

Markov chains

A *Markov chain* over a set of states $\mathcal{S}$ consists of a transition distribution $P : \mathcal{S} \to \Delta(\mathcal{S})$, where $P(s' \mid s)$ denotes the probability of transitioning from state s to state s'. A *stationary distribution* μ of a Markov chain is a distribution over states that is invariant under transitions, that is:

$$\forall s \in \mathcal{S} : \mu(s) = \sum_{s' \in \mathcal{S}} P(s \mid s')\mu(s').$$

Alternatively, viewing P as a transition matrix, we have $\mu = \mu \cdot P$.

A Markov chain is *ergodic* if there is some finite time T_0 such that for any two states s_i and s_j, if the Markov chain is in state s_i at timestep t, then at all times $T \geq t + T_0$, there a non-zero probability that the Markov chain is in state s_j. Intuitively, no matter what the current state is, eventually every state will have positive probability. A fundamental fact about ergodic Markov chains is that they are guaranteed to have a single, unique stationary distribution μ, that the chain will converge to regardless of initial conditions. Following Levin and Peres (2017, Chapter 4), we define the *total variation distance* $||\mu - \nu||_{TV}$ between distributions μ and ν to be

$$||\mu - \nu||_{TV} = \max_{S \subseteq \mathcal{S}} |\mu(S) - \nu(S)| \tag{4.1}$$

Note that a bound on total variation also gives a bound on the variation in the probability assigned to a specific state s, by choosing $S = \{s\}$ in the definition of total variation:

Proposition 1. *If $||\mu - \nu||_{TV} \leq \epsilon$, then for any $s \in \mathcal{S}$, we have $|\mu(s) - \nu(s)| \leq \epsilon$.*

For an ergodic Markov chain, we can prove that the chain converges to the stationary distribution regardless of initial conditions:

Proposition 2. (Levin and Peres, 2017, Theorem 4.9) *For an ergodic Markov chain P with stationary distribution μ, there exist constants $\alpha \in (0,1)$ and $C > 0$ such that*

$$\max_{s \in \mathcal{S}} ||P^t(s, \cdot) - \mu||_{TV} \leq C\alpha^t.$$

While the theorem considers deterministic initial conditions where we start at a particular state s, the bound also applies to stochastic initial conditions, since they are convex combinations of the deterministic cases.

Ergodic MDPs

Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}^H, \mathcal{T}, \mathcal{R}, P_\mathcal{S}, \gamma \rangle$ and a policy π^H, we say that π^H induces a Markov chain in $\mathcal{M}$ given by:

$$P(s' \mid s) = \sum_{a^H \in \mathcal{A}^H} \pi^H(a^H \mid s)\mathcal{T}(s' \mid s, a^H).$$

Intuitively, if we observe an agent with policy π^H acting in $\mathcal{M}$, the sequence of states visited by the agent behaves like the Markov chain induced by π^H in $\mathcal{M}$.

An MDP $\mathcal{M}$ is *ergodic* if for all possible policies π^H, the Markov chain induced by π^H in $\mathcal{M}$ is ergodic. For an ergodic MDP, the stationary distribution of states can never depend on the initial state distribution $P_\mathcal{S}$, regardless of the choice of π^H. We write $\mu_\infty^{\pi^H}$ to denote the stationary distribution of the resulting Markov chain, which is the limit of the t-step visitation distribution, defined recursively with $\mu_0^{\pi^H} = P_\mathcal{S}$ and

$$\mu_t^{\pi^H}(s) = \sum_{s' \in \mathcal{S}} P(s \mid s')\mu_{t-1}^{\pi^H}(s').$$

Learning from an optimal human

In this chapter, we will assume that H pursues her goals *optimally*, that is, π^H is an optimal policy given her true reward function $\mathcal{R}_{\theta^*}$. When two or more actions are equally valuable, we assume H chooses uniformly at random between these actions. We can formalize this using the Bellman equations (which are guaranteed to have a unique solution):

$$\begin{aligned}
\pi^H(a^H \mid s, \theta) &\propto \mathbb{1}\left[a^H \in \operatorname*{argmax}_{a^{H'} \in \mathcal{A}^H} Q^*(s, a^{H'}; \theta)\right] \\
Q^*(s, a^H; \theta) &= \mathop{\mathbb{E}}_{s' \sim \mathcal{T}(\cdot \mid s, a^H)}\left[\mathcal{R}_\theta(s, a^H, s') + \gamma V^*(s; \theta)\right] \\
V^*(s; \theta) &= \max_{a^H \in \mathcal{A}^H} Q^*(s, a^H; \theta)
\end{aligned}$$

We denote this policy by $\pi^H_{\text{opt}}(\theta)$ (read as "the optimal policy for θ"), and thus its stationary distribution is $\mu_\infty^{\pi^H_{\text{opt}}(\theta)}$. Now, recall that our goal is to learn from the state of the world. For clarity, we will denote the observed state as s_{observed}. We now want to compute the posterior $P(\theta \mid s_{\text{observed}})$, which is given by Equation 3.2:

$$P(\theta \mid s_{\text{observed}}) \propto P(\theta) \cdot P(s_{\text{observed}} \mid \theta) = P(\theta) \cdot \mu_\infty^{\pi^H_{\text{opt}}(\theta)}(s_{\text{observed}}).$$

In other words, the likelihood ratio is given by $\mu_\infty^{\pi^H_{\text{opt}}(\theta)}(s_{\text{observed}})$, and so any information that we can learn comes from the variation of this quantity with θ. For this reason, $\mu_\infty^{\pi^H_{\text{opt}}(\theta)}(s_{\text{observed}})$ will be the primary object of study in this chapter.

4.2 Preserving the observed state

We argued in Chapter 1 that upon observing some optimized state, our first instinct would be to "do nothing" and leave the state as is. There is a fully general argument for this point: if all we know is that H optimized the environment to lead to state s_{observed}, then clearly H expects that s_{observed} is a high-value state, and so perhaps we should try not to change it.

Let us suppose there is reward function that rewards being in the state s_{observed}, and never rewards any other state. Intuitively, the "goal" of this reward function would be to visit s_{observed} as often as possible. As long as γ is sufficiently large, it will care primarily about how much we visit s_{observed} in the stationary distribution. So, the optimal policy for this reward is the policy that maximizes the visitation of s_{observed} at the stationary distribution. We formalize this below:

Proposition 3. *Suppose there is some $\theta_{preserve}$ such that $\mathcal{R}_{\theta_{preserve}}(s, a, s') = \mathbb{1}[s = s_{observed}]$. Then there exists some $\gamma_0 < 1$ such that if $\gamma > \gamma_0$ then $\theta_{preserve}$ maximizes $\mu_\infty^{\pi^H_{opt}(\theta)}(s_{observed})$.*

Proof Sketch. If there is some other θ that achieves higher likelihood than θ_{preserve}, it must visit s_{observed} more often in the limit as $t \to \infty$. Then $\pi^H_{\text{opt}}(\theta)$ must achieve higher reward *according to $\theta_{preserve}$* than $\pi^H_{\text{opt}}(\theta_{\text{preserve}})$ in the limit as $t \to \infty$, despite the fact that $\pi^H_{\text{opt}}(\theta_{\text{preserve}})$ is optimal for θ_{preserve}. The only way this can hold is if $\pi^H_{\text{opt}}(\theta_{\text{preserve}})$ visited s_{observed} early at the cost of failing to visit s_{observed} later. This allows us to bound γ. $\square$

Proof. Note that for the reward $\mathcal{R}_{\theta_{\text{preserve}}}$, the expected reward is simply the sum of the discounted probabilities of being in s_{observed}, and so $ER_{\theta_{\text{preserve}}}(\pi^H) = \sum_{t=0}^{\infty} \gamma^t \mu_t^{\pi^H}(s_{\text{observed}})$.

Consider two arbitrary policies π^H_1 and π^H_2. Assume that they satisfy the following two conditions:

1. π^H_1 has lower likelihood: $\mu_\infty^{\pi^H_1}(s_{\text{observed}}) < \mu_\infty^{\pi^H_2}(s_{\text{observed}})$.
2. π^H_1 has higher expected reward: $ER_{\theta_{\text{preserve}}}(\pi^H_1) \geq ER_{\theta_{\text{preserve}}}(\pi^H_2)$.

Define p to be the lower of the two stationary distribution probabilities, and ϵ to be half the difference, so that we have:

$$\begin{aligned}\mu_\infty^{\pi_1^H}(s_{\text{observed}}) &= p \\ \mu_\infty^{\pi_2^H}(s_{\text{observed}}) &= p + 2\epsilon\end{aligned}$$

By the application of the convergence theorem (Proposition 2), we can find some time T such that for any $t \geq T$:

1. $\mu_t^{\pi_1^H}(s_{\text{observed}}) \leq p + \frac{\epsilon}{2}$
2. $\mu_t^{\pi_2^H}(s_{\text{observed}}) \geq p + \frac{3\epsilon}{2}$

Putting these together, we get:

$$\forall t \geq T: \quad \mu_t^{\pi_2^H}(s_{\text{observed}}) - \mu_t^{\pi_1^H}(s_{\text{observed}}) \geq \epsilon.$$

We now use the inequality from the second condition and simplify:

$$\begin{aligned}ER_{\theta_{\text{preserve}}}(\pi_2^H) - ER_{\theta_{\text{preserve}}}(\pi_1^H) &\leq 0 \\ \sum_{t=0}^{\infty} \gamma^t (\mu_t^{\pi_2^H}(s_{\text{observed}}) - \mu_t^{\pi_1^H}(s_{\text{observed}})) &\leq 0 \\ \left[\sum_{t=0}^{T-1} \gamma^t (\mu_t^{\pi_2^H}(s_{\text{observed}}) - \mu_t^{\pi_1^H}(s_{\text{observed}}))\right] + \\ \left[\sum_{t=T}^{\infty} \gamma^t (\mu_t^{\pi_2^H}(s_{\text{observed}}) - \mu_t^{\pi_1^H}(s_{\text{observed}}))\right] &\leq 0\end{aligned}$$

For the first terms where $t < T$, we use the fact that μ_t must be a probability, and so is bound by $[0, 1]$. For the second terms where $t \geq T$, we use the bound from above:

$$\begin{aligned}\left[\sum_{t=0}^{T-1} \gamma^t (0 - 1)\right] + \left[\sum_{t=T}^{\infty} \gamma^t \epsilon\right] &\leq 0 \\ \frac{-(1-\gamma^T)}{1-\gamma} + \frac{\gamma^T}{1-\gamma}\epsilon &\leq 0 \\ \gamma \leq \left(\frac{1}{1+\epsilon}\right)^{\frac{1}{T}} &< 1\end{aligned}$$

So far, we have shown that for an arbitrary pair of policies π_1^H, π_2^H, if they satisfy the two conditions above then we can derive a bound on γ, which we will denote as $\gamma\text{-bound}(\pi_1^H, \pi_2^H)$. Equivalently, if $\gamma > \gamma\text{-bound}(\pi_1^H, \pi_2^H)$, then it is not possible for π_1^H, π_2^H to satisfy the two conditions above.

Then, we can define $\gamma_0 \triangleq \max_{\pi_1^H, \pi_2^H} \gamma\text{-bound}(\pi_1^H, \pi_2^H)$. Note that every γ-bound is less than 1. Since $\mathcal{S}$ and $\mathcal{A}^H$ are finite, so too is the space of deterministic policies, and so we also have $\gamma_0 < 1$, as required.

Now suppose $\gamma > \gamma_0$, and consider an arbitrary policy $\overline{\pi^H}$. For the pair $\pi_{\text{opt}}^H(\theta_{\text{preserve}}), \overline{\pi^H}$, both conditions cannot be satisfied. However, since $\pi_{\text{opt}}^H(\theta_{\text{preserve}})$ maximizes $ER_{\theta_{\text{preserve}}}$, the second condition must be satisfied. Thus, the first condition cannot be satisfied. Therefore $\pi_{\text{opt}}^H(\theta_{\text{preserve}})$ maximizes $\mu_\infty^{\pi_{\text{opt}}^H(\theta)}(s_{\text{observed}})$. $\square$

The above theorem only applies when γ is sufficiently large, effectively because only then are we sufficiently confident that H cared enough about s_{observed} that we can infer that it must be a high-value state. However, even when γ is small, we would find a policy that prioritizes reaching s_{observed} when possible. Is it possible that despite its focus on the near term, such a policy would nonetheless maximize the probability of s_{observed} in the stationary distribution? It turns out the answer is no: the agent may sacrifice the long-term reward from the stationary distribution in order to get a more enticing short-term reward.

Consider the MDP in Figure 4.1, in which the observed state is $s_{\text{observed}} = B$. Every time the agent takes an action, there is an $\epsilon = 0.04$ chance that the next state is chosen uniformly at random from all states. (This is necessary in order to ensure that the MDP is ergodic, as we have assumed in this chapter.)

In this MDP, the only relevant decision for a policy is whether to take action a_1 or a_2 in state A, which we will refer to as π_1 and π_2 respectively. The stationary distribution of π_1 is $\{A : 0.01, B : 0.311, C : 0.37, D : 0.309\}$, while that of π_2 is $\{A : 0.01, B : 0.368, C : 0.25, D : 0.372\}$. The crucial feature is just that π_1 visits C more while π_2 visits B and D more, as one would expect looking at Figure 4.1.

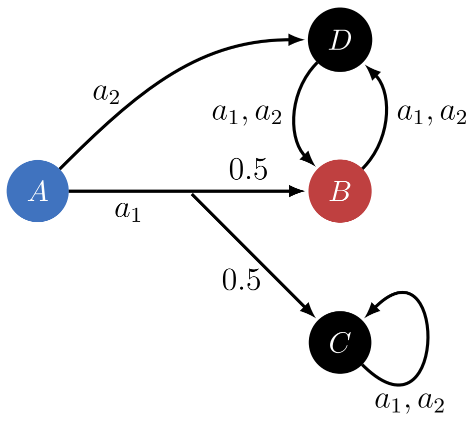


Figure 4.1: An MDP in which a greedy goal-seeking agent will sacrifice long-term reward in pursuit of short-term reward. The blue A is the important state to analyze, and the red B is the observed state s_{observed}.

We assume the reward space is parameterized such that $\mathcal{R}_{\theta_i}$ assigns 1 to state i and 0 to all other states. For this MDP, $\theta_{\text{preserve}} = \theta_B$. For this reward, we face a choice: whether to take a_1 in the hopes of getting to state B immediately, or to take a_2 to guarantee that we eventually get to state B.

Intuitively, if γ is sufficiently low, then it is better to take a_1, because the uncertain chance of reaching B is better than being forced to wait a full timestep before being able to reach B. For example, when $\gamma = 0.1$, the value of π_1 in each state is about $\{A : 0.0497, B : 1.0106, C : 0.0013, D : 0.0982\}$, whereas the value of π_2 in each state is about $\{A : 0.0105, B : 1.0105, C : 0.0012, D : 0.0981\}$. π_1 strictly dominates π_2 (where the difference is most pronounced in A, since that is where the two policies differ).

Meanwhile, the optimal policy for θ_D is clearly π_2, since a_2 goes directly to D and preserves the ability to keep going to D, whereas a_1 does not go to D and has some chance of severely curtailing the agent's ability to go to D. Thus at $\gamma = 0.1$, we have $\mu_\infty^{\pi^H_{\text{opt}}(\theta_D)}(s_{\text{observed}}) > \mu_\infty^{\pi^H_{\text{opt}}(\theta_{\text{preserve}})}(s_{\text{observed}})$, and θ_{preserve} is *not* a maximizer of $\mu_\infty^\theta(s_{\text{observed}})$. Thus, the requirement in Proposition 3 that γ be sufficiently high cannot be completely eliminated.

This counterexample relied on our ability to set up the dynamics to offer the agent a "lottery" (action a_1) that could severely curtail its ability to reach s_{observed}. What if we remove the ability to set up such lotteries by requiring that the environment be deterministic?

One issue is that deterministic environments are typically not ergodic. However, we can apply the same trick as in our counterexample above, where to every action we add an ϵ probability that the new state is chosen uniformly at random. Specifically, given an MDP $\mathcal{M}$, define the ϵ-randomized version of $\mathcal{M}$ to be the same as $\mathcal{M}$ except that the transition dynamics are modified to $\mathcal{T}'(s, a^H, s') = (1-\epsilon)\mathcal{T}(s, a, s') + \frac{\epsilon}{|\mathcal{S}|}$.

Then, an MDP $\mathcal{M}$ is ϵ-deterministic if there exists an MDP $\mathcal{M}'$ with a deterministic transition function $\mathcal{T}_d$, such that $\mathcal{M}$ is the ϵ-randomized version of $\mathcal{M}'$. We will sometimes abuse notation and write $\mathcal{T}_d(s, a^H)$ to denote the unique state s' such that $\mathcal{T}_d(s, a^H, s') = 1$.

As long as the reward does not depend on the next state, the optimal policy for an ϵ-deterministic MDP can be found by solving an associated fully deterministic MDP:

Lemma 4. *Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}^H, \mathcal{T}, \mathcal{R}, P_\mathcal{S}, \gamma\rangle$ be an ϵ-deterministic MDP with deterministic transitions $\mathcal{T}_d$, where the reward does not depend on next state and is written as $\mathcal{R}(s, a^H)$. Then a policy π^H is optimal for $\mathcal{M}$ iff it is optimal for $\mathcal{M}' = \langle \mathcal{S}, \mathcal{A}^H, \mathcal{T}_d, \mathcal{R}, P_\mathcal{S}, \gamma_e\rangle$, where the effective discount $\gamma_e = \gamma(1-\epsilon)$.*

Proof. An optimal policy can be computed from the optimal Q-function, which can be calculated by value iteration, in which we initialize $V_0(s) = 0$, and then define the recurrence:

$$\begin{aligned}
Q_t(s, a^H) &= \mathcal{R}(s, a^H) + \sum_{s' \in \mathcal{S}} \gamma \mathcal{T}(s, a^H, s') V_{t-1}(s') \\
&= \left[\mathcal{R}(s, a^H) + \gamma(1-\epsilon) V_{t-1}(\mathcal{T}_d(s, a^H))\right] + \frac{\epsilon}{|S|} \sum_{s' \in \mathcal{S}} V_{t-1}(s') \\
V_t(s) &= \max_{a^H \in \mathcal{A}^H} Q_t(s, a^H).
\end{aligned}$$

Note that optimal policies for a Q-function are unchanged if we subtract a constant from Q_t. This allows us to drop the last term from Q_t. Using $\gamma_e = \gamma(1-\epsilon)$, we have:

$$Q_t(s, a^H) = \mathcal{R}(s, a^H) + \gamma(1-\epsilon)V_{t-1}(\mathcal{T}_d(s, a^H)).$$

This is equivalent to the Q-value backup for a fully deterministic MDP with dynamics $\mathcal{T}_d$ with effective discount rate γ_e. □

In an ϵ-deterministic MDP, we we once again find that the likelihood $\mu_\infty^{\pi^H_{\text{opt}}(\theta)}(s_{\text{observed}})$ is highest for θ_{preserve}:

Proposition 5. *Suppose there is some $\theta_{preserve}$ for $\mathcal{M}$ as in Proposition 3, and assume that $\mathcal{M}$ is ϵ-deterministic with $0 < \epsilon < 1$. Then $\theta_{preserve}$ maximizes $\mu_\infty^{\pi^H_{opt}(\theta)}(s_{observed})$.*

Proof Sketch. In an ϵ-deterministic environment, $\pi^H_{\text{opt}}(\theta_{\text{preserve}})$ is always traversing the shortest path in $\mathcal{T}_d$ to s_{observed}, which means that at every timestep t it will be maximizing the average visitation, $\frac{1}{t+1}\sum_{t'=0}^{t} \mu_{t'}^{\pi^H_{\text{opt}}(\theta)}(s_{\text{observed}})$, which in the limit converges to the stationary distribution $\mu_\infty^{\pi^H_{\text{opt}}(\theta)}(s_{\text{observed}})$. □

Proof. First, we show that $\pi^H_{\text{opt}}(\theta_{\text{preserve}})$ always chooses an action a that is on a shortest path in $\mathcal{T}_d$ to s_{observed}. By Lemma 4, we know that $\pi^H_{\text{opt}}(\theta_{\text{preserve}})$ is an optimal policy for the Q-function computed by the backups:

$$\begin{aligned} Q_t(s, a^H) &= \mathbb{1}\left[s = s_{\text{observed}}\right] + \gamma_e V_{t-1}(\mathcal{T}_d(s, a^H)) \\ V_t(s) &= \max_{a^H \in \mathcal{A}^H} Q_t(s, a^H). \end{aligned}$$

This is equivalent to the Q-value backup for a fully deterministic MDP with dynamics $\mathcal{T}_d$ with effective discount rate γ_e. It is clear that the optimal policy in such an MDP is to follow the shortest path to s_{observed}.

To elaborate, let $L(s)$ be the length of the shortest path from s to s_{observed} in $\mathcal{T}_d$ (or ∞ if no such path exists), with $L(s_{\text{observed}}) \triangleq 0$. In addition, define L_0 to be the length of the shortest (non-empty) path from s_{observed} to itself. Then, we can prove by induction using the previous recurrence relations that:

$$V_t(s) = \begin{cases} \gamma_e^{L(s)} \frac{1-\gamma_e^{\lceil \frac{t-L(s)}{L_0} \rceil L_0}}{1-\gamma_e^{L_0}} & t > L(s) \\ 0 & \text{else} \end{cases}.$$

It is easy to see that either all actions are equally good (the else case above), or the best action is the one which progresses on a shortest path (so that in the next state $L(s)$ has decreased, or if we are in s_{observed} then we are following a shortest path back to s_{observed}).

We have so far shown that $\pi^H_{\text{opt}}(\theta_{\text{preserve}})$ always chooses an action a^H that is on a shortest path in $\mathcal{T}_d$ to s_{observed}. Since the only other effect of any action besides traversing paths in $\mathcal{T}_d$ is to teleport the agent randomly, which the agent cannot control, we only need to analyze how paths in $\mathcal{T}_d$ are traversed. Clearly, always traversing the shortest path to s_{observed} will maximize the expected number of times that s_{observed} is visited, and so we have:

$$\begin{aligned}
\pi^H_{\text{opt}}(\theta_{\text{preserve}}) &\in \operatorname*{argmax}_{\pi^H} \sum_{t'=0}^{t} \mu_{t'}^{\pi^H}(s_{\text{observed}}) && \text{for finite } t \\
\pi^H_{\text{opt}}(\theta_{\text{preserve}}) &\in \operatorname*{argmax}_{\pi^H} \lim_{t\to\infty} \frac{1}{t+1} \sum_{t'=0}^{t} \mu_{t'}^{\pi^H}(s_{\text{observed}}) \\
\pi^H_{\text{opt}}(\theta_{\text{preserve}}) &\in \operatorname*{argmax}_{\pi^H} \lim_{t\to\infty} \mu_{t}^{\pi^H}(s_{\text{observed}}) && \text{Limit of average is limit of sequence} \\
\pi^H_{\text{opt}}(\theta_{\text{preserve}}) &\in \operatorname*{argmax}_{\pi^H} \mu_{\infty}^{\pi^H}(s_{\text{observed}}) \\
\theta_{\text{preserve}} &\in \operatorname*{argmax}_{\theta} \mu_{\infty}^{\pi^H_{\text{opt}}(\theta)}(s_{\text{observed}})
\end{aligned}$$

□

4.3 Features as a form of prior information

Overall, it seems quite likely that θ_{preserve} will have a high likelihood ratio (if it exists). We have derived two distinct sufficient conditions: first, when H is modeled as sufficiently farsighted (that is, with sufficiently high γ), and second, when the environment is ϵ-deterministic.

This is in some sense a disappointing result: it suggests that it is hard to learn anything more sophisticated than "preserve the observed state", even though intuitively it seems like we should be able to infer more from the state of the world. We view this as having similar implications as the no-free-lunch theorem of machine learning: just as we need to assume some prior simplicity of the environment in order for machine learning to be feasible at all, we similarly need to assume some prior information about $\mathcal{R}_{\theta^*}$ in order to make non-trivial inferences about the reward that H is optimizing.

We will use a particularly simple form of prior information: we will assume that we have access to a *feature function* $f : \mathcal{S} \to \mathbb{R}^F$ that describes each state by a vector of real-valued *features* of that state. We will use $f_i : \mathcal{S} \to \mathbb{R}$ to denote the ith feature function, so that $f_i(s) = [f(s)]_i$. Just as we have visitation distributions for states, we can have feature counts:

$$\mathcal{F}_t^{\pi^H} = \sum_{t'=0}^{t} \sum_{s\in\mathcal{S}} \mu_{t'}^{\pi^H}(s) f(s).$$

The reward $\mathcal{R}_{\theta^*}$ is then assumed to be of the form $\mathcal{R}_{\theta^*}(s, a^H, s') = g_{\theta^*}(f(s))$ for some simple function g. Unless otherwise specified, we will use rewards that are *linear* over the features, that is, $\mathcal{R}_\theta(s, a^H, s') = \theta^T f(s)$. Note that for a constant $c > 0$, $\mathcal{R}_\theta$ and $\mathcal{R}_{c\theta}$ have the same optimal policy; as a result we may sometimes assume that θ is normalized.

Once we use features, we can formalize an important fact about learning from the state of the world: if there was no way that H could have affected a particular feature, then we can't infer anything about her preferences over that feature. Note that this is not specific to learning from the state of the world: we could prove a similar theorem for inverse reinforcement learning as well.

Proposition 6. *Assume that for a specific feature f_i and any timestep t, the feature count $(\mathcal{F}_t^{\pi^H})_i$ is independent of π^H. Then $\mu_\infty^{\pi^H_{opt}(\theta)}(s_{observed})$ is invariant to the value of θ_i.*

Proof Sketch. Since there is no policy that can affect the value of f_i, changing the value of θ_i has no effect on the optimal policy, and so cannot have an effect on $\mu_\infty^{\pi^H_{\text{opt}}(\theta)}(s_{\text{observed}})$. □

Proof. Consider some arbitrary θ, and let θ' be the same as θ, except θ_i has been increased by an arbitrary constant c (which may be negative). Then, for any policy π^H:

$$\begin{aligned}
ER_{\theta'}(\pi^H) &= \sum_{t=0}^{\infty} \gamma^t \sum_{s \in \mathcal{S}} \mu_t^{\pi^H}(s) \mathcal{R}_{\theta'}(s) \\
&= \sum_{t=0}^{\infty} \gamma^t \sum_{s \in \mathcal{S}} (\theta')^T \mathcal{F}_t^{\pi^H} \\
&= \sum_{t=0}^{\infty} \gamma^t \sum_{s \in \mathcal{S}} \theta^T \mathcal{F}_t^{\pi^H} + c(\mathcal{F}_t^{\pi^H})_i \\
&= K + \sum_{t=0}^{\infty} \gamma^t \sum_{s \in \mathcal{S}} \theta^T \mathcal{F}_t^{\pi^H} \qquad \text{where } K \text{ is independent of } \pi^H \\
&= K + ER_\theta(\pi^H).
\end{aligned}$$

In the fourth line above, we have used the fact that $(\mathcal{F}_t^{\pi^H})_i$ is independent of π^H to extract out a constant K that is also independent of π^H.

Thus, for any policy π^H, the expected reward of the policy under θ' only changes by a constant. This can never change the ordering over policies, and so $\pi^H_{\text{opt}}(\theta) = \pi^H_{\text{opt}}(\theta')$; that is, optimal policies are invariant to changes in θ_i. Thus, $\mu_\infty^{\pi^H_{\text{opt}}(\theta)}(s_{\text{observed}})$ is also invariant to changes in θ_i. □

4.4 Information from irreversibility

Recall our motivating example of an intact vase: since a vase that is broken can never be repaired, the fact that it is still intact in the observed state is strong evidence that H preferred that the vase stays intact. This is an instance of a general pattern: if you observe that some irreversible action was *not* taken, this is a strong signal that the consequences of that action are particularly unwanted. If we have a feature that tracks whether the irreversible transition has been taken, then presumably the reward weight of that feature should indicate that the transition is not to be taken. We formalize this below:

Definition 2. Binary feature. *A feature f_i is* binary *if it only takes on the values 0 or 1, that is, $\forall s, f_i(s) \in \{0, 1\}$.*

Definition 3. Irreversible feature. *A binary feature f_i is* irreversible *in $\mathcal{M}$ if it satisfies the following conditions:*

1. *Initially off: $\forall s \in \text{Support}(P_{\mathcal{S}}), f_i(s) = 0$.*
2. *Can be turned on: $\exists \pi^H, t : \mu_t^{\pi^H}(f_i) > 0$.*
3. *Cannot be turned off: $\forall s : [f_i(s) = 1 \implies \forall a \forall s' \in \text{Support}(\mathcal{T}(\cdot \mid s, a)), f_i(s') = 1]$.*

For example, the "broken vase" feature in the vase environment of Figure 1.2 is an irreversible feature. Note that in an ergodic MDP, there cannot be irreversible features, since there is always positive probability of eventually transitioning back to the initial state. However, we can instead consider non-ergodic MDPs $\mathcal{M}$, which can have irreversible features, and then ask what we would infer about the reward in an ϵ-randomized version of $\mathcal{M}$ (which is always ergodic, if $\epsilon > 0$).

Can we get a general result saying that if we observe that an irreversible feature is not turned on, then we can infer that the feature should not be turned on? Unfortunately the answer is no. Consider the ϵ-deterministic MDP in Figure 4.2, in which there is a feature that indicates whether or not the state is blue (that is, whether the state is C or not). This is a binary, irreversible feature in the deterministic transitions $\mathcal{T}_d$, and we do observe that the agent is in a state where the feature is not yet on. However, the agent has no control over anything, and so by Proposition 6 nothing can be inferred about the weight on this feature.

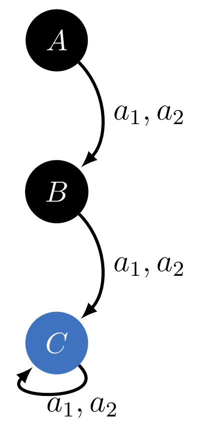


Figure 4.2: An ϵ-deterministic chain MDP in which the agent has no control over the irreversible blue feature. B is the observed state s_{observed}.

The core issue is that we can only make strong inferences about irreversible events if we believe that H *controlled* whether or not the irreversible event happened. In the chain MDP of Figure 4.2, the irreversible event is inevitable, and can only be undone by the ϵ chance of randomly transitioning to a randomly chosen state.

It should be noted that this is not an artifact of ϵ-randomization. If we did not have the ergodic restriction, then we could still have an MDP in which the agent's first action randomly sends it into one of two halves of the state space. In one half the agent is forced to turn on the irreversible feature f_1, and in the other half it is forced to turn on the irreversible

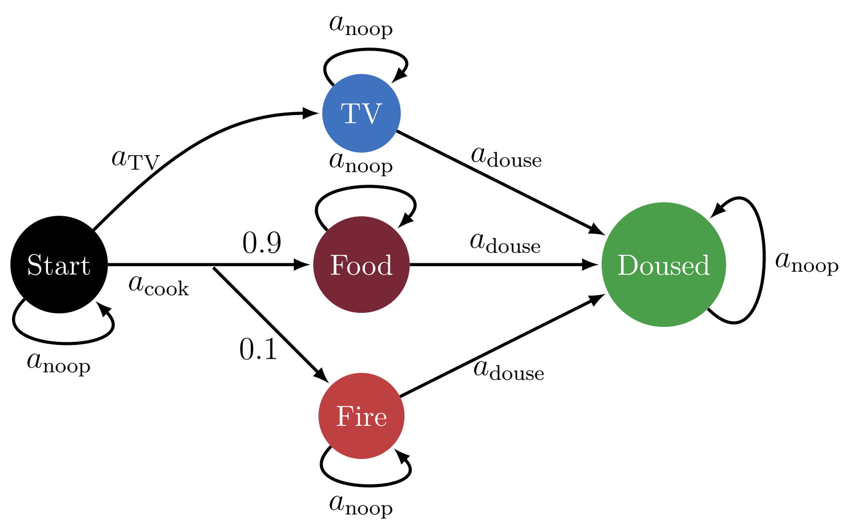


Figure 4.3: Fire extinguisher MDP. At the start, the agent can choose whether to cook food or watch TV. Watching TV always succeeds. Cooking usually succeeds in making food, but also has a 10% chance of starting a fire. At any point the agent may use the fire extinguisher, irreversibly covering the house in foam but extinguishing any fires present. The agent may always take a noop, and if it takes an action that is not applicable to the given state, that is treated equivalently to a noop.

feature f_2. The observed state s_{observed} will have exactly one of the two irreversible features turned on, but again by Proposition 6 we cannot infer anything about either of the features.

What if we added an assumption that the irreversible transition was "controllable"? For example, we could assume that in every state s where $f_i(s) = 0$, there is at least one action that guarantees that the next state s' also satisfies $f_i(s') = 0$. Could we then say that the irreversible feature's weight can be made arbitrarily negative?

It turns out that even with this assumption, we cannot conclude that the irreversible feature is arbitrarily dispreferred. While we do know that along the trajectory the feature was never turned on, it is possible that had the environment randomness been different then the optimal policy would have turned the feature on. For example, using a fire extinguisher is irreversible, and we usually observe that fire extinguishers have not been used, but there certainly are situations in which we would use fire extinguishers, and so the reward weight for using a fire extinguisher cannot be arbitrarily negative.

Concretely, consider the fire extinguisher MDP in Figure 4.3. For ergodicity, we need the environment to be ϵ-randomized, but we ignore this dependence during analysis for the sake of clarity. In this MDP, the agent starts out in the state Start, and can choose to watch TV, or to cook, which has a 90% chance of producing Food, and a 10% chance of igniting Fire. From any of these states, the agent may use the fire extinguisher to Douse a fire (if present). We have four features: f_{Food}, f_{TV}, f_{Fire} and f_{Doused}, each of which take on value 1 in the corresponding state and 0 everywhere else. Note that f_{Doused} is a binary, controllable, irreversible feature.

Suppose that we observe $s_{\text{observed}} = \text{Food}$. Intuitively, H cooked food and was willing to take the risk of igniting a fire. Assuming that γ is sufficiently large, the optimal policy for the reward $\overline{\theta} = [\overline{\theta}_{\text{Food}}, \overline{\theta}_{\text{TV}}, \overline{\theta}_{\text{Fire}}, \overline{\theta}_{\text{Doused}}] = [1, 0, -20, 0]$ is to cook food and then take noop actions, and to douse the fire if one is ignited. This policy leads to $\mu_{\infty}^{\pi^H_{\text{opt}}(\overline{\theta})}(s_{\text{observed}}) = 0.9$ (ignoring ϵ-randomness), which is the highest possible value (since the maximum probability of entering the Food state is 0.9). However, $\overline{\theta}_{\text{Doused}}$ cannot be made arbitrarily negative: for example, if $\overline{\theta}_{\text{Doused}}$ is set to -100, then it is no longer worth cooking. The expected cost of the fire outweighs the expected benefit of the food, since we would no longer be willing to douse the fire if it did occur. As a result, the agent would watch TV instead of cooking (or execute some other 0 reward policy), in which case $\mu_{\infty}^{\pi^H}(s_{\text{observed}}) = 0$.

This type of reasoning is a feature, not a bug: the entire point of a fire extinguisher is to deal with accidental fires, and so we *want* our agent to learn that we are not completely averse to using fire extinguishers, *despite* the fact that using a fire extinguisher is irreversible and we have never (yet) done it.

4.5 Discussion

In this chapter, we theoretically analyzed solutions to the problem of learning from the state of the world. We first analyzed the general argument that the inferred reward would tend to prioritize *preserving* the observed state of the world. When a "preservation" reward function exists, it often does seem to have high likelihood. We identified two different sufficient conditions for such a reward function to maximize the likelihood:

1. H is sufficiently farsighted, that is, γ is sufficiently large (Proposition 3), or
2. The dynamics of the MDP are ϵ-deterministic (Proposition 5).

These are not guaranteed: there are some cases like Figure 4.1 where with stochastic dynamics and low γ the preservation reward function does not maximize the likelihood. Nonetheless, it seems likely that in most realistic cases, a preservation reward function would tend to have high or even maximal likelihood.

We also analyzed another general argument from Chapter 1: if we observe that some irreversible transition has not happened, then H probably did not want that irreversible transition to happen. While this is certainly sometimes true, as in the vase example of Figure 1.2, there are often realistic scenarios in which it is not. In particular, just because H has not yet irreversibly used a fire extinguisher, doesn't mean that she doesn't want it to be used in the event of a fire.

Further research could also consider how to best formalize the argument from effort: "if H put effort into a particular feature, then that is very informative about the reward weight for that feature". For example, if the transition dynamics lead to a vase accumulating dust over time, that H must wipe off in order to make it clean, and we then observe a clean vase in s_{observed}, that is informative about H's preferences over dusty vases.

Having clarified what we can hope to do by learning from the state of the world, we now turn to the design of algorithms that can efficiently solve such problems. We will leverage the assumption that the reward is linear in features, as identified in Section 4.3. We will then use these algorithms to further explore conceptually what can be done by learning from the state of the world, in Chapter 7.

Chapter 5

An exact algorithm for tabular environments

Having discussed the inferences that could be made in theory, we now turn to the task of creating an algorithm that can make these inferences in practice. In this chapter, we will consider finite-horizon tabular environments, that is, in our state of the world problem $\langle \mathcal{M}, s_0, \pi^H, T, \langle \Theta, \mathcal{R}_\theta, P_\Theta \rangle \rangle$, we will require that $T \neq \infty$, and that $\mathcal{M}$ is sufficiently small that we can iterate over it, analogously to value iteration.

In Chapter 4 we saw that allowing for arbitrary reward functions often leads to degenerate solutions. So, we assume that $\mathcal{M}$ is equipped with a feature function f that identifies the relevant features, and the the reward function is linear in features, so that $\mathcal{R}_\theta(s) = \theta^T f(s)$.

Equation 3.1 specifies how to compute the likelihood of a particular reward $\mathcal{R}_\theta$. We reproduce it here:

$$P(s_0 \mid \theta) = \sum_{\substack{s_{-T}, \cdots s_{-2}, s_{-1} \in \mathcal{S} \\ a^H_{-T}, \cdots a^H_{-2}, a^H_{-1} \in \mathcal{A}^H}} P_\mathcal{S}(s_{-T}) \prod_{t=-T}^{-1} \pi^H(a^H_t \mid s_t, \theta) \mathcal{T}(s_{t+1} \mid s_t, a^H_t). \tag{5.1}$$

Our goal is to compute the posterior $P(\theta \mid s_0)$. However, typically θ will be drawn from a continuous space, and the transition dynamics $\mathcal{T}$ can be complex and nonlinear, making it very difficult if not impossible to compute an exact form for the posterior that works for arbitrary θ. This remains true even if we do not require the distribution to be normalized. We thus need to use approximate inference methods in order to estimate the posterior.

5.1 MCMC sampling

One standard way to address the computational challenges involved with the continuous and high-dimensional nature of θ is to use MCMC sampling to sample from $p(\theta \mid s_0) \propto p(s_0 \mid \theta) p(\theta)$. We apply this sampling in a standard manner to derive an algorithm that we present in Algorithm 1.

Algorithm 1 MCMC sampling to estimate the state of the world posterior

```
Require: State of the world problem $\langle \mathcal{M}, s_0, \pi^H, T, \langle \Theta, \mathcal{R}_\theta, P_\Theta \rangle \rangle$, step size $\delta$
 1: $\theta \leftarrow$ random sample($P_\Theta$)
 2: Compute the last-step occupancy measure $p(s_0 \mid \theta)$
 3: $p \leftarrow P(s_0 \mid \theta) P_\Theta(\theta)$
 4: repeat
 5:     $\theta' \leftarrow$ random sample($\mathcal{N}(\theta, \delta)$)
 6:     Compute the last-step occupancy measure $p(s_0 \mid \theta')$
 7:     $p' \leftarrow p(s_0 \mid \theta') p(\theta')$
 8:     if random sample(Unif(0, 1)) $\leq \min(1, \frac{p'}{p})$ then
 9:         $\theta \leftarrow \theta'$
10:     end if
11:     Append $\theta$ to the list of samples
12: until have generated the desired number of samples
```

While this Algorithm 1 gives us an estimate of the full posterior distribution, even for relatively simple environments it can take quite a long time to converge, and so we seek a better solution.

5.2 Reward Learning by Simulating the Past

To obtain a more efficient algorithm, we take inspiration from the research literature on inverse reinforcement learning (IRL). The problem formulation in IRL is similar to ours: the primary difference is that R can observe a set of trajectories $\{\tau_i\}$ rather than a single state s_0. If we apply MCMC sampling to IRL as we did in the previous section, then we recover the Bayesian IRL algorithm (Ramachandran and Amir, 2007).

Instead of sampling with Bayesian IRL, it is common to use the Maximum Causal Entropy IRL framework (MCEIRL) (Ziebart et al., 2010). As discussed in Section 2.5, MCEIRL computes a point estimate of the true reward $\mathcal{R}_{\theta^*}$ rather than the full posterior $P(\theta \mid s_0)$. It also assumes that π^H is the Boltzmann-rational policy, which can be computed using soft value iteration.

We adopt both of these assumptions in our setting to derive a new algorithm analogous to MCEIRL for the state of the world setting.

Deriving the gradient

Our goal is now to find the maximum a posteriori estimate of the reward:

$$\theta^* = \text{argmax}_\theta \ln p(\theta \mid s_0) = \text{argmax}_\theta \ln p(s_0 \mid \theta) + \ln P_\Theta(\theta). \tag{5.2}$$

MCEIRL uses gradient ascent to solve the problem, since it is convex in the reward parameters θ. We adopt the same approach. We assume that the gradient of the log prior $\nabla_\theta \ln P_\Theta(\theta)$ is easy to compute, and so we focus on computing the gradient of the log likelihood $\nabla_\theta \ln p(s_0 \mid \theta)$.

First, we note that the likelihood in Equation 3.1 can be rewritten in terms of the likelihoods of individual trajectories:

$$p(s_0 \mid \theta) = \sum_{\substack{s_{-T}, \dots s_{-2}, s_{-1} \in \mathcal{S} \\ a^H_{-T}, \dots a^H_{-2}, a^H_{-1} \in \mathcal{A}^H}} p(\tau \mid \theta), \tag{5.3}$$

where $\tau = s_{-T} a^H_{-T} \dots s_{-1} a^H_{-1} s_0$ is the (hypothesized) trajectory that H took in the environment.

We can then rewrite our desired gradient in terms of the gradients of individual trajectories as follows:

$$\begin{aligned}
\nabla_\theta \ln p(s_0 \mid \theta) &= \frac{1}{p(s_0 \mid \theta)} \nabla_\theta p(s_0 \mid \theta) \\
&= \frac{1}{p(s_0 \mid \theta)} \sum_{\substack{s_{-T:-1} \in \mathcal{S} \\ a^H_{-T:-1} \in \mathcal{A}^H}} \nabla_\theta p(\tau \mid \theta) \\
&= \frac{1}{p(s_0 \mid \theta)} \sum_{\substack{s_{-T:-1} \in \mathcal{S} \\ a^H_{-T:-1} \in \mathcal{A}^H}} p(\tau \mid \theta) \nabla_\theta \ln p(\tau \mid \theta).
\end{aligned}$$

This has a nice interpretation: compute the MCEIRL gradients for each trajectory, and then take their weighted sum, where each weight is the probability of the trajectory given the evidence s_0 and current reward θ. In Section 2.5 we derived the exact gradient for a trajectory under the MCEIRL assumption of Boltzmann-rationality. We now substitute it in to get:

$$\nabla_\theta \ln p(s_0 \mid \theta) = \frac{1}{p(s_0 \mid \theta)} \sum_{\substack{s_{-T:-1} \in \mathcal{S} \\ a^H_{-T:-1} \in \mathcal{A}^H}} \left[p(\tau \mid \theta) \sum_{t=-T}^{-1} g(s_t, a^H_t; \theta) \right], \tag{5.4}$$

where we have used the definitions from Section 2.5:

$$\begin{aligned}
g(s_t, a^H_t; \theta) &\triangleq f(s_t) + \mathop{\mathbb{E}}_{s'_{t+1}} \left[\mathcal{F}_{t+1}(s'_{t+1}; \theta) \right] - \mathcal{F}_t(s_t; \theta) \\
\mathcal{F}_t(s_t; \theta) &\triangleq f(s_t) + \mathop{\mathbb{E}}_{a^{H\prime}_{t:T-1}, s'_{t+1:T}} \left[\sum_{t'=t+1}^{T} f(s'_{t'}) \right].
\end{aligned}$$

Computing the gradient with dynamic programming

The gradient in Equation 5.4 still involves a combinatorially large summation over all possible past trajectories. In order to solve even simple environments, we need to avoid this combinatorial explosion. Our approach is to use dynamic programming.

We first express the gradient in Equation 5.4 as $\frac{G_0(s_0;\theta)}{p(s_0|\theta)}$, which can be done if we define

$$G_t(s_t;\theta) \triangleq \sum_{s_{-T:t-1}, a^H_{-T:t-1}} \left[p(\tau_{-T:t-1}, s_t \mid \theta) \sum_{t'=-T}^{t-1} g(s_{t'}, a^H_{t'}; \theta) \right]. \tag{5.5}$$

Thus, to compute the gradient, we only need to compute $G_0(s_0;\theta)$ and $p(s_0 \mid \theta)$ efficiently. The latter is particularly easy, as it is just the probability of observing a state at a specific timestep:

$$\begin{aligned} p(s_{-T} \mid \theta) &= P_{\mathcal{S}}(s_{-T}) \\ p(s_{t+1} \mid \theta) &= \sum_{\substack{s_t \in \mathcal{S} \\ a^H_t \in \mathcal{A}^H}} p(s_t \mid \theta) \pi^{\text{soft}}_t (a^H_t \mid s_t, \theta) \mathcal{T}(s_{t+1} \mid s_t, a^H_t). \end{aligned}$$

Note that π^{soft}_t can be computed using soft value iteration, as detailed in Section 2.5. Before we tackle G, we first derive a recursive rule for $\mathcal{F}$:

$$\begin{aligned} \mathcal{F}_0(s_0;\theta) &= f(s_0) \\ \mathcal{F}_{t-1}(s_{t-1};\theta) &= f(s_{t-1}) + \mathop{\mathbb{E}}_{a^{H'}_{t-1:-1}, s'_{t:0}} \left[\sum_{t'=t}^{0} f(s'_{t'}) \right] \\ &= f(s_{t-1}) + \mathop{\mathbb{E}}_{a^{H'}_{t-1}, s'_t} \left[f(s'_t) + \mathop{\mathbb{E}}_{a^{H'}_{t:-1}, s'_{t+1:0}} \left[\sum_{t'=t+1}^{0} f(s'_{t'}) \right] \right] \\ &= f(s_{t-1}) + \mathop{\mathbb{E}}_{a^{H'}_{t-1}, s'_t} \left[\mathcal{F}_t(s'_t;\theta) \right] \\ &= f(s_{t-1}) + \sum_{a^{H'}_{t-1}, s'_t} \pi^{\text{soft}}_{t-1}(a^{H'}_{t-1} \mid s_{t-1}, \theta) \mathcal{T}(s'_t \mid s_{t-1}, a^{H'}_{t-1}) \mathcal{F}_t(s'_t). \end{aligned}$$

Note that $g(s_t, a^H_t; \theta)$ can easily be computed using its definition given $\mathcal{F}$. We are now ready to derive a recursive relation for G:

$$
\begin{aligned}
&G_{t+1}(s_{t+1}; \theta) \\
&= \sum_{\substack{s_{-T:t} \in \mathcal{S} \\ a^H_{-T:t} \in \mathcal{A}^H}} \left[p(\tau_{-T:t}, s_{t+1} \mid \theta) \sum_{t'=-T}^{t} g(s_{t'}, a^H_{t'}; \theta) \right] \\
&= \sum_{\substack{s_t \in \mathcal{S} \\ a^H_t \in \mathcal{A}^H}} \sum_{\substack{s_{-T:t-1} \in \mathcal{S} \\ a^H_{-T:t-1} \in \mathcal{A}^H}} \mathcal{T}(s_{t+1} \mid s_t, a^H_t) \pi^{\text{soft}}_t (a^H_t \mid s_t, \theta) p(\tau_{-T:t-1}, s_t \mid \theta) \left(g(s_t, a^H_t; \theta) + \sum_{t'=-T}^{t-1} g(s_{t'}, a^H_{t'}; \theta) \right) \\
&= \sum_{\substack{s_t \in \mathcal{S} \\ a^H_t \in \mathcal{A}^H}} \left[\mathcal{T}(s_{t+1} \mid s_t, a^H_t) \pi^{\text{soft}}_t (a^H_t \mid s_t; \theta) \left(\sum_{\substack{s_{-T:t-1} \in \mathcal{S} \\ a^H_{-T:t-1} \in \mathcal{A}^H}} p(\tau_{-T:t-1}, s_t \mid \theta) \right) g(s_t, a^H_t; \theta) \right] \\
&\quad + \sum_{\substack{s_t \in \mathcal{S} \\ a^H_t \in \mathcal{A}^H}} \left[\mathcal{T}(s_{t+1} \mid s_t, a^H_t) \pi^{\text{soft}}_t (a^H_t \mid s_t; \theta) \sum_{\substack{s_{-T:t-1} \in \mathcal{S} \\ a^H_{-T:t-1} \in \mathcal{A}^H}} \left(p(\tau_{-T:t-1}, s_t \mid \theta) \sum_{t'=-T}^{t-1} g(s_{t'}, a^H_{t'}; \theta) \right) \right] \\
&= \sum_{\substack{s_t \in \mathcal{S} \\ a^H_t \in \mathcal{A}^H}} \mathcal{T}(s_{t+1} \mid s_t, a^H_t) \pi^{\text{soft}}_t (a^H_t \mid s_t; \theta) \Big(p(s_t \mid \theta) g(s_t, a^H_t; \theta) + G_t(s_t; \theta) \Big).
\end{aligned}
$$

For the base case, note that

$$
\begin{aligned}
G_{-T+1}(s_{-T+1}) &= \sum_{s_{-T}, a^H_{-T}} \left[p(s_{-T}, a^H_{-T}, s_{-T+1}) g(s_{-T}, a^H_{-T}, s_{-T+1}) \right] \\
&= \sum_{s_{-T}, a^H_{-T}} \mathcal{T}(s_{-T+1} \mid s_{-T}, a^H_{-T}) \pi^{\text{soft}}_{-T} (a^H_{-T} \mid s_{-T}) \Big(p(s_{-T}) g(s_{-T}, a^H_{-T}, s_{-T+1}) \Big).
\end{aligned}
$$

Comparing this to the recursive rule, for the base case we can set $G_{-T}(s_{-T}) = 0$.

Overall algorithm

Combining all of these ingredients together gives us our final algorithm, presented in Algorithm 2. Since we combine gradients from simulated past trajectories, we name the algorithm Reward Learning by Simulating the Past (RLSP).

Algorithm 2 Reward Learning by Simulating the Past.

```
Require: State of the world problem $\langle \mathcal{M}, s_0, \pi^H, T, \langle \Theta, \mathcal{R}_\theta, P_\Theta \rangle \rangle$, learning rate $\alpha$
 1: $\theta \leftarrow$ random sample($P_\Theta$)
 2: repeat
 3:     $\forall s_{-T} \in \mathcal{S} : p(s_{-T} \mid \theta) \leftarrow P_{\mathcal{S}}(s_{-T})$
 4:     $\forall s_0 \in \mathcal{S} : \mathcal{F}_0(s_0; \theta) \leftarrow f(s_0)$
 5:     $\forall s_{-T} \in \mathcal{S} : G_{-T}(s_{-T}; \theta) \leftarrow 0$
 6:     $\pi^{\text{soft}} \leftarrow$ soft value iteration($\mathcal{M}, \theta, T$)
 7:     // Compute probabilities of states
 8:     for $t$ in $[-T, \ldots, -2, -1]$ do
 9:         $\forall s_{t+1} \in \mathcal{S} : p(s_{t+1} \mid \theta) \leftarrow \sum_{s_t, a_t^H} p(s_t \mid \theta) \pi_t^{\text{soft}}(a_t^H \mid s_t, \theta) \mathcal{T}(s_{t+1} \mid s_t, a_t^H)$
10:     end for
11:     // Compute expected feature counts
12:     for $t$ in $[-1, -2, \ldots, -T]$ do
13:         $\forall s_t \in \mathcal{S} : \mathcal{F}_t(s_t; \theta) \leftarrow f(s_t) + \sum_{a_t^{H'}, s'_{t+1}} \pi_t^{\text{soft}}(a_t^{H'} \mid s_t, \theta) \mathcal{T}(s'_{t+1} \mid s_t, a_t^{H'}) \mathcal{F}_{t+1}(s_{t+1}; \theta)$
14:     end for
15:     // Compute $G$
16:     for $t$ in $[-T, \ldots, -2, -1]$ do
17:         $\forall s_{t+1} \in \mathcal{S} : G_{t+1}(s_{t+1}; \theta) \leftarrow 0$
18:         for $s_t, a_t^H$ in $\mathcal{S} \times \mathcal{A}^H$ do
19:             $g(s_t, a_t^H; \theta) \leftarrow f(s_t) + \mathbb{E}_{s'_{t+1}} \left[ \mathcal{F}_{t+1}(s'_{t+1}; \theta) \right] - \mathcal{F}_t(s_t; \theta)$
20:             target $\leftarrow \pi_t^{\text{soft}}(a_t^H \mid s_t; \theta) \left[ p(s_t \mid \theta) g(s_t, a_t^H; \theta) + G_t(s_t; \theta) \right]$
21:             $\forall s_{t+1} \in \mathcal{S} : G_{t+1}(s_{t+1}; \theta) \leftarrow G_{t+1}(s_{t+1}; \theta) + \mathcal{T}(s_{t+1} \mid s_t, a_t^H) \cdot$ target
22:         end for
23:     end for
24:     // Gradient ascent
25:     $\theta \leftarrow \theta + \alpha \cdot \left[ \frac{G_0(s_0; \theta)}{p(s_0 \mid \theta)} + \nabla_\theta \ln P_\Theta(\theta) \right]$
26: until convergence
```

5.3 Requirements

The algorithms that we have presented here can infer information about human preferences, but require fairly strong assumptions. In particular, in order to apply either RLSP or the sampling algorithm, we need to have:

1. Small state and action spaces, so that we can enumerate all possible state-action pairs,
2. Perfect knowledge of the transition dynamics $\mathcal{T}$ of the environments, and
3. A feature function f that identifies relevant features of interest.

In the next chapter, we relax these restrictions.

Chapter 6

Function approximation for high-dimensional environments

Our goal now is to scale RLSP to realistic environments, where we cannot enumerate states, the dynamics of the environment are not fully known in advance, and a feature function may not be available. Consider for example the balanced cheetah in Figure 6.1. Just by looking at this single balanced state, it should be possible to infer the "goal" of balancing. However, we cannot apply the RLSP algorithm from the previous chapter for several reasons: the state space is continuous, we cannot simply enumerate the transition matrix, and there is no clear way to obtain a feature function.

Once again, we can take inspiration from corresponding research in inverse reinforcement

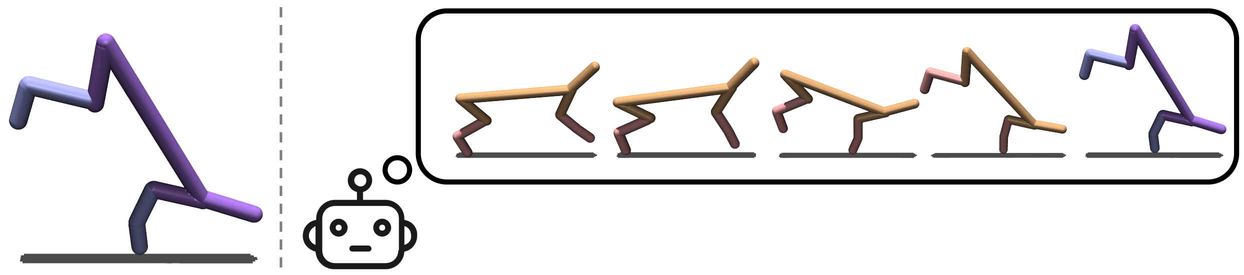


Figure 6.1: Suppose we observe a Cheetah balancing on its front leg (left). The state contains joint *velocities* in addition to positions, which are fairly low, showing that the cheetah is indeed balancing rather than, say, falling over. Intuitively, just by observing this balanced state, we should be able to reason that the Cheetah is "trying" to balance – other plausible goals, such as running forward, falling over, hopping on a single leg, and so on would not have led to this kind of state. We would like to have an algorithm that can make these sorts of inferences, despite the continuous state space, the lack of a feature function, and only having access to a simulator (rather than a full transition matrix as in the previous chapter).

learning. Recent IRL algorithms (Fu et al., 2017; Finn et al., 2016) have shown how to generalize tabular IRL algorithms to work in high-dimensional environments. The key idea is to use a neural net function approximator to represent the reward function and learn a policy in tandem with the reward function. We can then compute gradients by comparing rollouts from the policy with the provided demonstrations, as in Equation 2.3.

This is a useful starting point for us, but does not solve everything. Our first challenge is that unlike the MCEIRL gradient of Equation 2.3, the RLSP gradient in Equation 5.4 cannot be easily expressed as a comparison between policy rollouts and the observed state s_0, and so it is unclear how the gradient can be computed. Our key insight here is that to sample from the distribution over past trajectories $p(\tau_{-T:-1} \mid s_0, \theta)$, we can start at the observed state s_0 and simulate *backwards in time*. To enable this, we derive a gradient that is amenable to estimation through backwards simulation, and learn an inverse policy and inverse dynamics model using supervised learning to perform the backwards rollouts.

Our second challenge is that we cannot use a deep neural net as our reward function, because this is too expressive: we would start learning the degenerate reward functions identified in Section 4.2 that only assign reward to the observed state and ignore everything else. We need a reward representation that can be meaningfully updated from a single state observation, yet also can capture complex functions of the input in order to express concepts like the "balanced Cheetah". Our solution here is to represent the reward as a linear combination of features, where the features are learned through self-supervised representation learning techniques in a pretraining step.

6.1 Gradient as backwards-forwards consistency

Here we tackle the first challenge: expressing the RLSP gradient in a form that we can work with when we only have access to a simulator for the environment. We start with the gradient from Section 5.2:

$$\begin{aligned}
\nabla_\theta \ln p(s_0 \mid \theta) &= \frac{1}{p(s_0 \mid \theta)} \sum_{\substack{s_{-T:-1} \in \mathcal{S} \\ a^H_{-T:-1} \in \mathcal{A}^H}} p(\tau \mid \theta) \nabla_\theta \ln p(\tau \mid \theta) \\
&= \sum_{\substack{s_{-T:-1} \in \mathcal{S} \\ a^H_{-T:-1} \in \mathcal{A}^H}} p(\tau_{-T:-1} \mid s_0, \theta) \nabla_\theta \ln p(\tau \mid \theta) \\
&= \mathop{\mathbb{E}}_{\tau_{-T:-1} \sim p(\cdot \mid s_0, \theta)} \left[\nabla_\theta \ln p(\tau \mid \theta) \right].
\end{aligned}$$

Approximating the expectation

For higher-dimensional environments, we must approximate the expectation over past trajectories $p(\tau_{-T:-1} \mid s_0, \theta)$. We would like to sample from the distribution, but it is not clear how to sample the past conditioned on the present. Our key idea is that just as we can sample the future by rolling out forwards in time, *we should be able to sample the past by rolling out backwards in time.* Note that by the Markov property we have:

$$\begin{aligned} p(\tau_{-T:-1} \mid s_0, \theta) &= \prod_{t=-T}^{-1} p(s_t \mid a_t^H, s_{t+1}, \dots s_0, \theta) p(a_t^H \mid s_{t+1}, a_{t+1}^H, \dots s_0, \theta) \\ &= \prod_{t=-T}^{-1} p(s_t \mid a_t^H, s_{t+1}, \theta) p(a_t^H \mid s_{t+1}, \theta) \end{aligned}$$

Thus, given the *inverse policy* $(\pi^H)_t^{-1}(a_t^H \mid s_{t+1}, \theta)$ as well as the *inverse dynamics* $\mathcal{T}_t^{-1}(s_t \mid a_t^H, s_{t+1}, \theta)$, we can sample a past trajectory $\tau_{-T:-1} \sim p(\tau_{-T:-1} \mid s_0, \theta)$ by iteratively applying $(\pi^H)^{-1}$ and $\mathcal{T}^{-1}$, starting from s_0. Thus, our gradient can be written as

$$\mathop{\mathbb{E}}_{\substack{a^H_{-1:-T} \sim (\pi^H)^{-1} \\ s_{-1:-T} \sim \mathcal{T}^{-1}}} \left[\nabla_\theta \ln p(\tau \mid \theta)\right].$$

In order to learn $(\pi^H)^{-1}$, we must first have π^H. We assumed that the human was Boltzmann-rational, which corresponds to the *maximum entropy* reinforcement learning objective (Levine, 2018). We use Soft Actor-Critic (SAC; Haarnoja et al., 2018) to estimate the policy $\pi^H(a \mid s, \theta)$, since it explicitly optimizes the maximum entropy RL objective.

Given the forward policy $\pi^H(a^H \mid s, \theta)$ and simulator $\mathcal{T}$, we can construct a dataset of sampled forward trajectories, and learn the inverse policy $(\pi^H)^{-1}$ and the inverse dynamics $\mathcal{T}^{-1}$ using supervised learning. Given these, we can then sample $\tau_{-T:-1}$, allowing us to approximate the expectation in the gradient. In general, both $(\pi^H)^{-1}$ and $\mathcal{T}^{-1}$ could be stochastic and time-dependent, even if π^H and $\mathcal{T}$ are themselves deterministic and time-independent.

Estimating the gradient for a trajectory

We now turn to the term within the expectation, which is the inverse reinforcement learning gradient given a demonstration trajectory $\tau = s_{-T} a^H_{-T} \dots s_0$. Assuming that the user is Boltzmann-rational, this is the MCEIRL gradient from Section 2.5:

$$\nabla_\theta \ln p(\tau \mid \theta) = \left(\sum_{t=-T}^{0} f(s_t)\right) - \mathcal{F}_{-T}(s_{-T}; \theta) + \sum_{t=-T}^{-1} \left(\mathop{\mathbb{E}}_{s'_{t+1} \sim \mathcal{T}(\cdot \mid s_t, a_t^H)} \left[\mathcal{F}_{t+1}(s'_{t+1}; \theta)\right] - \mathcal{F}_{t+1}(s_{t+1}; \theta) \right) \tag{6.1}$$

(Recall that $\mathcal{F}_t(s_t; \theta)$ is the expected feature count when continuing to act from state s_t according to policy π^H, that is, $\mathcal{F}_{-t}(s_{-t}; \theta) \triangleq \mathop{\mathbb{E}}_{\substack{a^H_{-t:-1} \sim \pi^H \\ s_{-t:0} \sim \mathcal{T}}} \left[\sum_{t'=-t}^{0} f(s_{t'})\right]$.)

The first term computes the feature counts of the demonstrated trajectory τ, while the second term computes the feature counts obtained by the policy for the current reward function θ (starting from the initial state s_{-T}). Since $r(s) = \theta^T f(s)$, these terms increase the reward of features present in the demonstration τ and decrease the reward of features under the current policy. Thus, the gradient incentivizes *consistency* between the demonstration and rollouts from the learned policy.

The last term is essentially a correction for the observed dynamics: if we see that s_t, a_t^H led to s_{t+1}, it corrects for the fact that we "could have" seen some other state s'_{t+1}. Since this correction is zero in expectation (and expensive to compute), we drop it for our estimator.

Gradient estimator

After dropping the last term in Equation 6.1, expanding the definition of $\mathcal{F}$, and substituting in to our earlier gradient expression, our final gradient estimator is:

$$\nabla_\theta \ln p(s_0 \mid \theta) = \mathop{\mathbb{E}}_{\substack{a^H_{-1:-T} \sim (\pi^H)^{-1} \\ s_{-1:-T} \sim \mathcal{T}^{-1}}} \left[\left(\sum_{t=-T}^{0} f(s_t) \right) - \mathop{\mathbb{E}}_{\substack{s'_{-T} = s_{-T} \\ a^H_{-T:-1} \sim \pi^H \\ s_{-T+1:0} \sim \mathcal{T}}} \left[\left(\sum_{t=-T}^{0} f(s'_t) \right) \right] \right] \tag{6.2}$$

Thus, given s_0, θ, π^H, $\mathcal{T}$, $(\pi^H)^{-1}$, and $\mathcal{T}^{-1}$, computing the gradient consists of three steps:

1. Simulate backwards from s_0, and compute the feature counts of the resulting trajectories.
2. Simulate forwards from s_{-T} of these trajectories, and compute their feature counts.
3. Take the difference between these two quantities.

This again incentivizes consistency, this time between the backwards and forwards trajectories: the gradient leads to movement towards "what the human must have done" and away from "what the human would do if they had this reward". The gradient becomes zero when they are identical.

It may seem like the backwards and forwards trajectories should always be consistent with each other, since $(\pi^H)^{-1}$ and $\mathcal{T}^{-1}$ are inverses of π^H and $\mathcal{T}$. The key difference is that s_0 imposes constraints on the backwards trajectories, but not on the forward trajectories. For example, suppose we observe s_0 in which a vase is unbroken, and our current hypothesis is that the user wants to break the vase. When we simulate backwards, our trajectory will contain an unbroken vase (even though the policy usually breaks vases), and when we simulate forwards from s_{-T}, π^H will break the vase. The gradient would then reduce the reward for a broken vase and increase the reward for an unbroken vase.

6.2 Learning a latent MDP

Our gradient still relies on a feature function f, with the reward parameterized as $r(s) = \theta^T f(s)$. A natural way to remove this assumption would be to instead allow θ to parameterize a neural network, which can then learn whatever features are relevant to the reward from the RLSP gradient.

However, this approach will not work. The information contained in the RLSP gradient is insufficient to identify the appropriate features to construct: after all, it is derived from a single state. If we were to learn a single unified reward using the same gradient, the resulting reward would likely be degenerate: for example, it may simply identify the observed state, that is $R(s) = \mathbb{1}[s = s_0]$. Thus, we continue to assume that the reward is linear in features, and instead *learn the feature function* using self-supervised representation learning. Such techniques define an *auxiliary task* and train the neural net to perform well on the task, in the hopes that the neural net learns generally useful features that can then be used for the task of interest. There are many potential auxiliary tasks that we could use:

Contrastive tasks. Contrastive approaches (Oord et al., 2018) give the model a discrimination task: given a *context*, select which of a large set of *targets* is "most related" to the context. The notion of relatedness can change: in image classification, two images are "related" if they are different augmentations of the same base image (He et al., 2020; Chen et al., 2020); while in reinforcement learning, two states could be related if they come from the same trajectory (Lee et al., 2020; Stooke et al., 2020).

Reconstruction. In reconstructive approaches, the input state must be encoded into a low-dimensional representation vector, in such a way that the original state can then be decoded back from this vector (Kingma and Welling, 2014). In sequential environments, the decoder can instead predict past or future states. This has been applied in reinforcement learning (Stooke et al., 2020) as well as natural language processing (Radford et al., 2019). Alternatively, tasks can be defined over the sequence as a whole, for example to predict a missing word in a sentence (Devlin et al., 2019; Clark et al., 2020).

State space models. For partially observable environments recurrent state space models (RSSMs; Karl et al., 2017; Doerr et al., 2018; Hafner et al., 2019, 2020; Buesing et al., 2018; Kurutach et al., 2018) could be used instead: such methods aim to learn a *latent MDP*. They do so by allowing the latent states to be computed by a recurrent model over the observations, thus allowing the states to encode the history. For such a model, we can imagine that the underlying POMDP has been converted into a latent MDP whose feature function f is the identity. We can then compute RLSP gradients directly in this latent MDP, where $\mathcal{T}$ and f are known.

Goal-based exploration. By default, in the absence of a dataset of environment behavior, the methods above would have to be applied to a dataset of environment interaction using rollouts of a random policy. However, in many environments, a random policy will stay within a small subset of the state space with high probability. To cover more of the state space, we can extent any of the methods above with methods that increase the diversity of the data on which the features are trained. These methods include unsupervised skill learning (Achiam et al., 2018; Eysenbach et al., 2018; Nair et al., 2018; Sharma et al., 2020), curiosity-driven exploration (Burda et al., 2018), and goal-conditioned policies with a diverse goal space (Schaul et al., 2015; Andrychowicz et al., 2017).

Experimental choices. In our experiments, we use a variational autoencoder (VAE; Kingma and Welling, 2014) to learn the feature function. The VAE encodes the states into a latent feature representation, which we can use to learn a reward function if the environment is fully observable, i.e., the states contain all relevant information. However, we expect that better results could be obtained using other, more recent methods, if tuned well.

6.3 Deep RLSP

Putting these components together gives us the Deep RLSP algorithm (Algorithm 3). We first learn a feature function f using self-supervised learning, and then train an inverse dynamics model $\mathcal{T}^{-1}$, all using a dataset of environment interactions (such as random rollouts). Then, we update θ using Equation 6.2, and continually train π^H, and $(\pi^H)^{-1}$ alongside θ to keep them up to date. The full algorithm also adds a few bells and whistles that we describe below.

Initial state distribution $P_\mathcal{S}$. The attentive reader may wonder why our gradient appears to be independent of $P_\mathcal{S}$. This is actually not the case: while π^H and $\mathcal{T}$ are independent of $P_\mathcal{S}$, $(\pi^H)^{-1}$ and $\mathcal{T}^{-1}$ *do* depend on it. For example, if we observe Alice exiting the San Francisco airport, the corresponding $(\pi^H)^{-1}$ should hypothesize different flights if she started from New York than if she started from Tokyo.

However, in order to actually produce such explanations, we must train $(\pi^H)^{-1}$ and $\mathcal{T}^{-1}$ solely on trajectories of length T starting from $s_{-T} \sim P_\mathcal{S}$. We instead train $(\pi^H)^{-1}$ and $\mathcal{T}^{-1}$ on a variety of trajectory data, which loses the useful information in $P_\mathcal{S}$, but leads to several benefits. First, we can train the models on exactly the distributions that they will be used on, allowing us to avoid failures due to distribution shift. Second, the horizon T is no longer critical: previously, T encoded the separation in time between s_{-T} and s_0, and as a result misspecification of T could cause bad results. Since we now only have information about s_0, it doesn't matter much what we set T to, and as a result we can use it to set a curriculum (discussed next). Finally, this allows Deep RLSP to be used in domains where an initial state distribution is not available.

Since we are no longer able to use the information from $P_\mathcal{S}$ through $(\pi^H)^{-1}$ and $\mathcal{T}^{-1}$, we add in a heuristic to incorporate the information elsewhere. Specifically, we weight every

Algorithm 3 The Deep RLSP algorithm. The initial dataset of environment interactions D can be constructed in many different ways: random rollouts, human play data, curiosity-driven exploration, etc. The specific method will determine the quality of the learned features.

```
procedure Deep RLSP($\{s_0\}$, $\mathcal{T}$)
    $D \leftarrow$ dataset of environment interactions
    Initialize $f, \pi^H, (\pi^H)^{-1}, \mathcal{T}^{-1}, \theta$ randomly.
    $f \leftarrow$ SelfSupervisedLearning($D$)        ▷ Train encoder and decoder for latent MDP
    $\mathcal{T}^{-1} \leftarrow$ SupervisedLearning($D$)        ▷ Train inverse dynamics
    $T \leftarrow 1$        ▷ Start horizon at 1
    for $i$ in [1..num_epochs] do
        $\pi^H \leftarrow$ SAC($\theta$)        ▷ Train policy
        $\{\tau\} \leftarrow$ Rollout($\pi^H, \mathcal{T}$)        ▷ Collect dataset by rolling out $\pi^H$
        $(\pi^H)^{-1} \leftarrow$ SupervisedLearning($f, \{\tau\}$)        ▷ Train inverse policy
        $\theta \leftarrow \theta + \alpha \times$ ComputeGrad($\{s_0\}$, $\pi^H$, $\mathcal{T}$, $(\pi^H)^{-1}$, $\mathcal{T}^{-1}$, $T$, $f$)        ▷ Update $\theta$
        if gradient magnitudes are sufficiently low then
            $T \leftarrow T + 1$        ▷ Advance horizon
        end if
    end for
    return $\theta$, $f$
end procedure
procedure ComputeGrad($\{s_0\}$, $\pi^H$, $\mathcal{T}$, $(\pi^H)^{-1}$, $\mathcal{T}^{-1}$, $T$, $f$)
    $\{\tau_{\text{backward}}\} \leftarrow$ Rollout($\{s_0\}, (\pi^H)^{-1}, \mathcal{T}^{-1}, T$)        ▷ Simulate backwards from $s_0$
    $f_{\text{backward}} \leftarrow$ AverageFeatureCounts($f, \{\tau_{\text{backward}}\}$)        ▷ Compute backward feature counts
    $\{s_{-T}\} \leftarrow$ FinalStates($\{\tau_{\text{backward}}\}$)
    $\{\tau_{\text{forward}}\} \leftarrow$ Rollout($\{s_{-T}\}, \pi^H, \mathcal{T}, T$)        ▷ Simulate forwards from $s_{-T}$
    $f_{\text{forward}} \leftarrow$ AverageFeatureCounts($f, \{\tau_{\text{forward}}\}$)        ▷ Compute forward feature counts
    return $f_{\text{backward}} - f_{\text{forward}}$
end procedure
```

backwards trajectory by the cosine similarity between the final state s_{-T}, and a sample $\hat{s}_{-T} \sim P_{\mathcal{S}}$.

Curriculum. Since the horizon T is no longer crucial, we can use it to provide a curriculum. We initially calculate gradients with low values of T, to prevent compounding errors in our learned models, and making it easier to enforce backwards-forwards consistency, and then slowly grow T, making the problem harder. In practice, we found this crucial for performance: intuitively, it is much easier to make short backwards and forwards trajectories consistent than with longer trajectories; the latter would likely have much higher variance.

Multiple input states. If we get multiple independent s_0 as input, we average their gradients.

Replay buffer. We also maintain a replay buffer that stores previously collected (s, a, s') transitions. This buffer persists across policy training steps. When training the policy π, we sample transitions from both the replay buffer as well as from the simulator $\mathcal{T}$.

Having now defined both the RLSP and Deep RLSP algorithms, we turn to demonstrating and evaluating them on a variety of environments and tasks.

Chapter 7

Evaluation: Correcting misspecified rewards and imitating skills

To evaluate the importance of learning from the state of the world, we would like to check whether the inferred preferences enable R to be more helpful to H in a variety of contexts. However, it is not very clear how exactly to do so. The inferred reward is very likely to assign state s_0 maximal reward, since by assumption, when Alice optimized $\mathcal{R}_{\theta^*}$ she ended up at s_0. If the robot then starts in state s_0, if a no-op action is available (as it often is), the inferred reward is likely to incentivize no-ops, which is not very interesting or helpful.

Ultimately, our hope is that the information learned from the state of the world can be *combined* with other sources of preference information in order for R to learn both what it should do, and what it should not do. So, in Section 7.2, we create a suite of environments with a true reward, $\mathcal{R}_{\theta^*}$, a specified reward, $\mathcal{R}_{\theta_{\text{spec}}}$, the initial state of the environment, s_{-T}, and the state observed by R, s_0. Here, $\mathcal{R}_{\theta_{\text{spec}}}$ is a stand-in for some information about what R should do, and will ignore some aspect(s) of $\mathcal{R}_{\theta^*}$.

To evaluate an algorithm in an environment, we use it to infer a reward θ_H from s_0, which is then combined with the specified reward to get a final reward $\theta_{\text{final}} = \theta_H + \lambda\theta_{\text{spec}}$. (We later consider another heuristic method for combining these two rewards in Section 7.6, as well as a more principled approach in Chapter 8.) We inspect the inferred reward qualitatively and report the expected amount of true reward obtained when planning with θ_{final}, as a fraction of the expected true reward from the optimal policy.

Section 7.3 identifies a set of environments in which inferred rewards *can* capture what R should do: robot locomotion. Intuitively, if a quadrupedal robot is in mid-jump, a "no-op" action is not available: the state of the environment will change no matter what the R does, simply because of gravity. As a result, the inferred reward cannot simply lead to behavior that stays in s_0 forever, and so the inferred reward could be sufficient to learn good behavior.

Thus, in these environments we use each algorithm to infer a reward function from the state of the world, given a state sampled from rollouts of a policy that is performing a specific type of movement behavior. We then optimize a new policy using the inferred reward function, and evaluate how well the new policy imitates the original skill.

Having done these basic tests of algorithm functionality, we turn to investigating particular details of the algorithm implementations, including the importance of priors over s_{-T} (Section 7.4), the robustness to the planning horizon (Section 7.5), and alternative methods for combining reward functions (Section 7.6).

7.1 Algorithm details

RLSP is almost entirely specified by Algorithm 2. In our experiments, we set the learning rate to 0.1 and the temperature for soft value iteration to 1.0.

Deep RLSP on the other hand has several implementation details beyond what is specified in Algorithm 3. In this section we describe the hyperparameters and architecture choices for all models used in Deep RLSP in our experiments. All models are implemented using the TensorFlow framework.

Feature function. We use a variational autoencoder (VAE) (Kingma and Welling, 2014) to learn the feature function f. The encoder and decoder consist of 3 feed-forward layers of size 512. The latent space has dimension 30. The model is trained for 100 epochs on 100 rollouts of a random policy in the environment. During training we use a batch size of 500 and a learning rate of 10^{-5}. We use the standard VAE loss function, but weight the KL-divergence term with a factor $c = 0.001$, which reduces the regularization and empirically improved the reconstruction of the model significantly in our experiments. We hypothesize that the standard VAE regularizes too much in our setting, because the latent space has a higher dimension than the input space, which is not the case in typical dimensionality reduction settings.

Inverse dynamics model. Our inverse dynamics model $\mathcal{T}^{-1}$ is a feed-forward neural network with 5 layers of size 1024 with ReLU activations. We train it on 1000 rollouts of a random policy in the environment for 100 epochs, with a batch size of 500 and learning rate of 10^{-5}.

Note that the model predicts the previous observation given the current observation and action; it does not use the feature representation. We found the model to perform better if it predicts the residual $o_{t-1} - o_t$ given o_t and a_t instead of directly predicting o_{t-1}.

We normalize all inputs to the model to have zero mean and unit variance. To increase robustness, we also add zero-mean Gaussian noise with standard deviation 0.001 to the inputs and labels during training and clip the outputs of the model to the range of values observed during training.

Policy. For learning the policy π^H we use the *stable-baselines* implementation of Soft Actor-Critic (SAC) with its default parameters for the MuJoCo environments (Haarnoja et al., 2018; Hill et al., 2018). Each policy update during Deep RLSP uses 10^4 total timesteps (except Hopper, where we use 2×10^4) and we evaluate the final reward function using 2×10^6 timesteps (except Pendulum, where we use 6^4, which is the default of Hill et al. (2018)).

Inverse policy. Because the inverse policy $(\pi^H)^{-1}$ is not a deterministic function, we represent it with a *mixture density network*, a feed-forward neural network that outputs a mixture of Gaussian distributions (Bishop, 1994).

The network has 3 layers of size 512 with ReLU activations and outputs a mixture of 5 Gaussians with a fixed diagonal covariance matrix $0.05 \cdot I$.

During Deep RLSP, we maintain an experience replay that is initialized with random rollouts. In every iteration all states encountered during the algorithm are added to the experience replay.

To update the inverse policy we sample batches with batch size 500 from the experience replay, apply the forward policy and the forward transition model on the states to label the data. We then train the model with a learning rate of 10^{-4}.

Feature learning dataset. By default, we use random rollouts to generate the initial dataset D that is used to train the features ϕ and the inverse model $\mathcal{T}^{-1}$.

Additional hyperparameters. We run Deep RLSP with a learning rate of 0.01, and use 200 forward and backward trajectories to estimate the gradients. Starting with $T = 1$ we increment the horizon when the gradient norm drops below 2.0 or after 10 steps, whichever comes first. We run the algorithm until $T = 10$.

Computational cost. It is important to note that Deep RLSP can be significantly more computationally costly than the baselines that we compare against. Imitation learning with full demonstrations can already be quite computationally expensive. Deep RLSP takes this much further by learning several distinct deep neural net models, *simulating* potential demonstrations (which are likely much noisier than real demonstrations), and finally imitating them.

7.2 Conceptual environments

In our first evaluation, we design a suite of environments to test whether algorithms that learn from the state of the world can extract various different types of information from the state of the environment at deployment. In each environment, a candidate algorithm must infer a reward function given only the MDP without reward $\mathcal{M}\backslash\mathcal{R}_{\theta^*}$, the observed state s_0, and (depending on the experiment) the initial state s_{-T}. This is then combined with a specified reward $\mathcal{R}_{\theta_{\text{spec}}}$, to give a final reward $\theta_{\text{final}} = \theta_H + \lambda\theta_{\text{spec}}$ that we then evaluate. The hyperparameter λ is tune for all algorithms (including baselines) to give the best results.

It should be noted that these environments were designed to illustrate properties of learning from the state of the world, and thus should not be considered an outside independent check on the value of learning from the state of the world. Nonetheless, we find the qualitative behaviors displayed below to be good indications of the value of such an approach.

We first describe our baselines, and then explain our suite of environments along with the results.

Baselines

To our knowledge, there are no existing algorithms for state of the world problems, and so we design some simple algorithms ourselves to serve as baselines.

Specified reward policy $\pi_{\textbf{spec}}$. This baseline corresponds to not doing any learning from the state of the world. We don't use s_0 at all and just optimize the specified reward, that is, $\theta_{\text{final}} = \theta_{\text{spec}}$.

Policy that penalizes deviations $\pi_{\textbf{deviation}}$. One of the core intuitions behind learning from the state of the world is that the state s_0 has already been optimized for human preferences, and so random changes to s_0 are likely to be bad. Thus, one possible baseline is to simply minimize changes to the observed state. We accomplish this by penalizing deviations from the observed features $f(s_0)$, giving $\mathcal{R}_{\text{final}}(s) = \theta_{\text{spec}}^T f(s) + \lambda ||f(s) - f(s_0)||$. Note that for this baseline, there is no θ_{final}; the algorithm directly specifies a final reward $\mathcal{R}_{\text{final}}$, rather than first specifying an inferred reward that is then added to $\mathcal{R}_{\theta_{\text{spec}}}$.

Relative reachability policy $\pi_{\textbf{reachability}}$. Rather than just penalizing any deviations, we could instead only penalize those deviations that are *impactful.* For this, we look to the literature on low-impact AI systems for baselines. We use relative reachability (Krakovna et al., 2018) as our baseline. This is an approach low-impact AI systems that considers a change to be negative when it decreases *coverage*, relative to what would have happened had the agent done nothing. Here, coverage is a measure of how easily states can be reached from the current state.

We compare against the variant of relative reachability that uses undiscounted coverage and a baseline policy where the agent takes no-op actions, as in the original paper. Relative reachability requires known dynamics but not a handcoded featurization. A version of relative reachability that does make use of handcoded features instead of operating directly on states would behave similarly on our suite of environments. Like the deviation baseline, relative reachability directly specifies a final reward $\mathcal{R}_{\text{final}}$.

Average features policy $\pi_{\textbf{features}}$. One way to think about s_0 is that it is a sample from the rollout of an optimal (or at least good) policy for $\mathcal{R}_{\theta^*}$. As a result, it is likely that the features of s_0 are generally good, and so should be incentivized. Since the reward takes on the form $\mathcal{R}_\theta(s) = \theta^T f(s)$, this suggests that we should set $\theta_H = \frac{f(s_0)}{||f(s_0)||}$, where the normalization is simply to put the reward on a common scale. We call this the AverageFeatures baseline.

Analysis

We compare RLSP and Deep RLSP to our baselines with the assumption of known s_{-T}, because it makes it easier to analyze their properties. We consider the case of unknown s_{-T} in Section 7.4.

For Deep RLSP, in these stylized gridworlds, self-supervised learning should not be expected to learn the necessary features. For example, in the room with vase environment, the two door features are just particular locations, with no distinguishing features in the state that would allow self-supervised learning to identify these locations as important. So, when evaluating Deep RLSP, we run Algorithm 3 without the feature learning and instead use the pre-defined feature function f of the environments. We also use a Mixture Density Network (MDN) (Bishop, 1994) for the inverse dynamics $\mathcal{T}^{-1}$, in order to allow the model to output probability distributions. Once we do this, Deep RLSP has the same behavior as RLSP, and so we only report results with RLSP below.

We summarize the results in Table 7.1, and show the environments and trajectories in Figure 7.1.

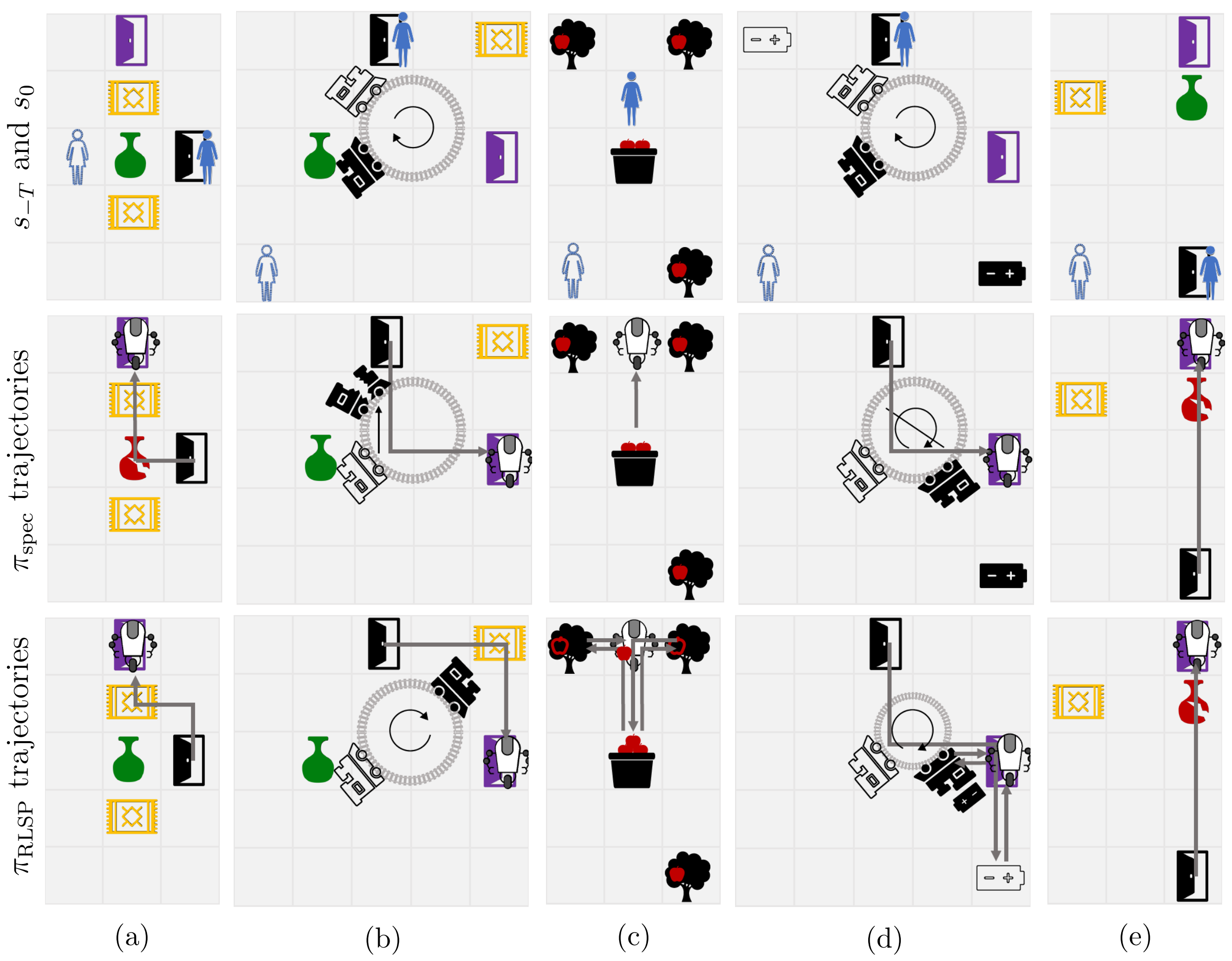
Table 7.1: Performance of algorithms on environments designed to test particular properties.

	Side effect	**Env effect**	**Implicit reward**	**Desirable act**		**Unseen effect**
	Room	Toy train	Apple collection	Batteries		Far away vase
				Easy	Hard	
$\pi_{\textbf{spec}}$	✗	✗	✗	✓	✗	✗
$\pi_{\textbf{deviation}}$	✓	✗	✗	$\approx$	✗	✓
$\pi_{\textbf{reachability}}$	✓	✓	✗	$\approx$	✗	✓
$\pi_{\textbf{features}}$	$\approx$	$\approx$	✓	$\approx$	✗	$\approx$
$\pi_{\textbf{RLSP}}$	✓	✓	✓	✓	✓	✗

Side effects: Room with vase *(Figure 7.1a).* The room tests whether the robot can avoid breaking a vase as a side effect of going to the purple door. There are features for the number of broken vases, standing on a carpet, and each door location. $\mathcal{R}_{\theta_{\text{spec}}}$ has weight 1 for the purple door feature, and 0 for all other weights. As a result, it causes the agent to walk over the base, breaking it. $\mathcal{R}_{\theta^*}$ additionally has weight -1 for the broken vases feature.

Since Alice didn't walk over the vase, RLSP infers a negative reward on broken vases, and a small positive reward on carpets (since paths to the top door usually involve carpets). So, π_{RLSP} successfully avoids breaking the vase.

The penalties also achieve the desired behavior: $\pi_{\text{deviation}}$ avoids breaking the vase since it would change the "number of broken vases" feature, while relative reachability avoids breaking the vase since doing so would result in all states with intact vases becoming unreachable.



s_{-T} and s_0
π_{spec} trajectories
π_{RLSP} trajectories
(a)
(b)
(c)
(d)
(e)

Figure 7.1: Evaluation of RLSP on our environments. Silhouettes indicate the initial position of an object or agent, while filled in versions indicate their positions after an agent has acted. The first row depicts the information given to RLSP. The second row shows the trajectory taken by the robot when following the policy π_{spec} that is optimal for $\mathcal{R}_{\theta_{\text{spec}}}$, where θ_H is the reward inferred using RLSP. The third row shows the trajectory taken when following the policy π_{RLSP} that is optimal for $\theta_{\text{final}} = \theta_H + \lambda\theta_{\text{spec}}$. (a) Side effects: Room with vase (b) Distinguishing environment effects: Toy train (c) Implicit reward: Apple collection (d) Desirable side effect: Batteries (e) "Unseen" side effect: Room with far away vase.

π_{features} fails to infer that breaking vases is bad, though this is due to a quirk in the feature encoding. In particular, the feature counts the number of broken vases, and so the inferred reward θ_H has a value of zero for this feature, effectively ignoring it. If we change the featurization to instead count the number of *unbroken* vases, then π_{features} would likely get the right behavior.

Distinguishing environment effects: Toy train *(Figure 7.1b).* To test whether algorithms can distinguish between effects caused by the agent and effects caused by the environment, as suggested in Krakovna et al. (2018), we add a toy train that moves along a predefined track. The train breaks if the agent steps on it. We add a new feature indicating whether the train is broken and new features for each possible train location. Once again, $\mathcal{R}_{\theta_{\text{spec}}}$ just has weight 1 on the purple door feature, and so the resulting policy ends up breaking the train. $\mathcal{R}_{\theta^*}$ additionally has weight -1 on broken vases and trains.

RLSP infers a negative reward on broken vases and broken trains, for the same reason as in the previous environment. It also infers not to put any weight on any particular train location, even though it changes frequently, because it doesn't help explain s_0. As a result, π_{RLSP} walks over a carpet, but not a vase or a train.

For the penalty-based algorithms, $\pi_{\text{deviation}}$ immediately breaks the train in order to prevent the train location features from changing. $\pi_{\text{reachability}}$ on the other hand deduces that breaking the train is irreversible, and so correctly follows the same trajectory as π_{RLSP}.

π_{features} fails to infer that trains should not be broken, due to the same quirk in feature encoding as in the previous environment, and as a result breaks the train.

Implicit reward: Apple collection *(Figure 7.1d).* This environment tests whether the algorithms can learn tasks implicit in s_0. There are three trees that grow apples, as well as a basket for collecting apples, and the goal is for the robot to harvest apples. We have features for the number of apples in baskets, the number of apples on trees, whether the robot is carrying an apple, and each location that the agent could be in. s_0 has two apples in the basket, while s_{-T} has none. The specified reward $\mathcal{R}_{\theta_{\text{spec}}}$ is zero: the robot must infer the task from the observed state. The true task is to collect apples, and so $\mathcal{R}_{\theta^*}$ has weight 1 on the number of apples in the basket, and no weight on any other feature.

π_{spec} is arbitrary since every policy is optimal for the zero reward. A penalty-based algorithm can never get positive reward, because $\mathcal{R}_{\theta_{\text{spec}}}$ is always zero and penalties can only decrease reward. As a result, the optimal policy for both $\pi_{\text{deviation}}$ and $\pi_{\text{reachability}}$ is to do nothing, which achieves the maximum of zero reward.

RLSP infers a positive reward on apples in baskets, a negative reward for apples on trees, and a small positive reward for carrying apples. Despite the spurious weights on other features, π_{RLSP} harvests apples as desired, achieving the maximum possible true reward.

π_{features} also correctly infers that it is good to place apples in the basket, but it also rewards the agent for staying in the original location. As a result, it avoids picking apples from the tree that is furthest away, and so it does not pick apples as effectively as π_{RLSP}.

Desirable side effect: Batteries *(Figure 7.1c).* This environment tests whether the algorithms can tell when a side effect is allowed. We take the toy train environment, remove vases and carpets, and add batteries. The robot can pick up batteries and put them into the (now unbreakable) toy train, but the batteries are never replenished. If the train runs for 10 timesteps without a new battery, it stops operating. There are features for the number of batteries, whether the train is operational, each train location, and each door location.

There are two batteries in s_{-T} but only one in s_0. The true reward $\mathcal{R}_{\theta^*}$ places weight 1 on being at the purple door, as well as weight -1 on the train being out of power.

We consider two variants for the specifie reward $\mathcal{R}_{\theta_{\text{spec}}}$ – an "easy" case, where is identical to the true reward $\mathcal{R}_{\theta^*}$, and a "hard" case, where it only has weight 1 on being at the purple door (and no weight on keeping the train operational).

Unsurprisingly, π_{spec} succeeds at the easy case (where $\mathcal{R}_{\theta_{\text{spec}}} = \mathcal{R}_{\theta^*}$), and fails on the hard case by allowing the train to run out of power. Both $\pi_{\text{deviation}}$ and $\pi_{\text{reachability}}$ see the action of putting a battery in the train as a side effect to be penalized, and so neither can solve the hard case. In the easy case, they still penalize picking up the batteries, and so only solve the easy case if the penalty weight λ is small.

RLSP sees that one battery is gone and that the train is operational, and infers that Alice wants the train to be operational and doesn't want batteries (since a preference against batteries and a preference for an operational train are nearly indistinguishable). So, it solves both the easy and the hard case, with π_{RLSP} picking up the battery, then staying at the purple door except to deliver the battery to the train.

π_{features} incorrectly infers that batteries should *not* be used up, since the number of batteries in the environment is positive and so we infer that batteries are good. It thus fails to solve the hard case, and only solves the easy case if λ is sufficiently small. Part of the issue here is that θ_{features} is not invariant to the addition of a constant to any feature. We could fix this by instead setting $\theta_{\text{features}} = f(s_0) - f(s_{-T})$, treating $f(s_{-T})$ as a baseline to which $f(s_0)$ should be compared, but this would make π_{features} reliant on knowledge of s_{-T} (which we will not have in future sections) and would cause it to always fail the vase and toy train environments (whereas currently it depends on how the features are constructed).

"Unseen" side effect: Room with far away vase *(Figure 7.1e).* This environment demonstrates a limitation of our algorithm: it cannot identify side effects that Alice would never have triggered. In this room, the vase is nowhere close to the shortest path from the Alice's original position to her goal, but is on the path to the robot's goal. The features, specified reward and true reward are all the same as in the room with vase environment (Figure 7.1a).

Since our baselines don't care about the trajectory the human takes, they all perform as before: π_{spec} walks over the vase, $\pi_{\text{deviation}}$ and $\pi_{\text{reachability}}$ both avoid it, and π_{features} fails to avoid it but only due to a quirk of feature construction.

RLSP infers a near zero weight on the broken vase feature, since it is not present on any reasonable trajectory to the goal, and so breaks it when moving to the goal. Note that this only applies when Alice is known to be at the bottom left corner at s_{-T}: if we have a uniform prior over s_{-T} (considered in Section 7.4) then we do consider trajectories where vases are broken.

7.3 Skill learning with Deep RLSP

The apples collection environment (Figure 7.1d) demonstrates that at least in some cases it is possible to learn what to do from the state of the world, without needing a specified reward $\mathcal{R}_{\theta_{\text{spec}}}$. In the case of that environment, given our choice of feature function, there was no possible reward function that would view s_0 as uniquely optimal, which forced RLSP to infer a reward that would incentivize changes to the environment. In this section, we identify another domain in which changes to the environment are necessary: robot locomotion. In this domain, due to physical laws (particularly gravity), R usually does not have the ability to preserve the state as it is. As a result, any reward inferred from the state of the world must take into account that the state can and will change, suggesting that we may be able to infer what R should do, rather than just what it should not do.

To test learning from the state of the world in this domain, we use the MuJoCo physics simulator (Todorov et al., 2012). We consider the *Inverted Pendulum*, *Half-Cheetah* and *Hopper* environments implemented in Open AI Gym (Brockman et al., 2016). Since these environments are continuous and high-dimensional, only Deep RLSP can be evaluated in this domain.

In the inverted pendulum environment, the pendulum falls very quickly in random rollouts, and $\mathcal{T}^{-1}$ never learns what a balanced pendulum looks like. So, for this environment only, when constructing the initial dataset of environment interactions D, we combine random rollouts with rollouts from an expert policy that balances the pendulum.

Baselines

To our knowledge, this is the first work to train policies using a single state as input. Due to lack of alternatives, we compare against GAIL (Ho and Ermon, 2016) using the implementation from the `imitation` library (Wang et al., 2020). GAIL is an imitation learning algorithm, and thus requires *transitions* as inputs, rather than single states. For each state we provide to Deep RLSP, we provide a transition (s, a, s') to GAIL. GAIL is thus given more information than Deep RLSP.

Most of our previous baselines in Section 7.2 do not apply in this setting, either because they do not scale to high-dimensional environments, or because they depend on the presence of $\mathcal{R}_{\theta_{\text{spec}}}$. In particular, the penalty-based deviation and relative reachability methods do not make sense as baselines, because the idea is to pursue some already specified behavior given by $\mathcal{R}_{\theta_{\text{spec}}}$, but avoid having side effects via the use of a penalty.

The only baseline that can still be applied is the AverageFeatures baseline, in which we set $\theta_H = \frac{f(s_0)}{||f(s_0)||}$. Here, the function f must still be learned through self-supervised learning, but we can avoid learning $(\pi^H)^{-1}$ and $\mathcal{T}^{-1}$ entirely. This can be thought of as an ablation of Deep RLSP, in which we ignore all temporal information.

In our experiments, we sometimes sample multiple states from rollouts of an expert policy, instead of just a single state. In this case, for Deep RLSP, we simply average the gradients for each of the states individually (which can be thought of as a lower-variance estimator of

Environment	SAC	# states	Deep RLSP	AverageFeatures	Waypoints	GAIL
Inverted Pendulum	1000	1	262 (246)	6 (2)	N/A	**1000 (0)**
		10	605 (242)	3 (1)	4 (1)	**1000 (0)**
		50	258 (198)	6 (4)	3.7 (0.3)	**1000 (0)**
Cheetah (forward)	13236	1	**4833 (2975)**	**6466 (3343)**	N/A	-288 (55)
		10	**6299 (559)**	**6245 (2352)**	-10 (23)	-296 (172)
		50	**7657 (177)**	4504 (2970)	-126 (38)	-54 (295)
Cheetah (backward)	13361	1	5694 (2513)	**12443 (645)**	N/A	-335 (46)
		10	8102 (624)	**12829 (651)**	-80 (388)	-283 (45)
		50	7795 (551)	**11616 (178)**	-509 (87)	2113 (1015)
Hopper (terminate)	3274	1	80 (21)	99 (45)	N/A	**991 (9)**
		10	168 (58)	159 (126)	58 (7)	**813 (200)**
		50	130 (12)	65 (36)	14 (4)	**501 (227)**
Hopper (penalty)	3363	1	**1964 (545)**	**2537 (363)**	N/A	990 (9)
		10	564 (75)	**3103 (64)**	709 (133)	784 (229)
		50	**2130 (744)**	**2078 (581)**	**1612 (785)**	508 (259)

Table 7.2: Average returns achieved by the policies learned through various methods, for different numbers of input states. The states are sampled from a policy trained using SAC on the true reward function; the return of that policy is given as a comparison. Besides the SAC policy return, all values are averaged over 3 seeds and the standard error is given in parentheses. We don't report Waypoints on 1 state as it is identical to AverageFeatures on 1 state.

the gradient), and we similarly use an average for the AverageFeatures baseline (which is why we call it the *Average*Features baseline).

An alternative to averaging the features is to view each of the observed states s_i as a potential *waypoint* of the expert policy, and reward R for being near any one of them. We implement this *Waypoints* method as $R(s) = \max_i \frac{f(s_0^i)}{||f(s_0^i)||} \cdot f(s)$. Note that when we only have a single state as input, this is equivalent to AverageFeatures, and so we do not report results for Waypoints with one state.

Solving the environments without access to the reward function

First we look at the typical target behavior in each environment: balancing the inverted pendulum, and making the half-cheetah and the hopper move forwards. Additionally we consider the goal of making the cheetah run backwards (that is, the negative of its usual reward function). We aim to use Deep RLSP to learn these behaviors *without having access to the reward function.*

We train a policy using soft actor critic (SAC) (Haarnoja et al., 2018) to optimize for the true reward function, and sample either 1, 10 or 50 states from rollouts of this policy to use as input. We then use Deep RLSP to infer a reward and policy. Ideally we would evaluate this learned policy rather than reoptimizing the learned reward, since learned reward models can often be gamed (Stiennon et al., 2020), but it would be too computationally expensive to run the required number of SAC steps during each policy learning step. As a result, we run SAC for many more iterations on the inferred reward function, and evaluate the resulting policy on the true reward function (which Deep RLSP does not have access to).

Results are shown in Table 7.2. In Hopper, we noticed that videos of the policies learned by Deep RLSP looked okay, but the quantitative evaluation said otherwise. It turns out that the policies learned by Deep RLSP do jump, as we might want, but they often fall down, terminating the episode; in contrast GAIL policies stand still or fall over slowly, leading to later termination and explaining their better quantitative performance. We wanted to also evaluate the policies without this termination bias, and so we evaluate the same policies in an environment that does not terminate the episode, but provides a negative reward instead; in this evaluation both Deep RLSP and AverageFeatures perform much better. We also provide videos of the learned policies at `https://sites.google.com/view/deep-rlsp`, which show that the policies learned by Deep RLSP do exhibit hopping behavior (though with a strong tendency to fall down).

GAIL is only able to learn a truly good policy for the (very simple) inverted pendulum, even though it gets states and actions as input. Deep RLSP on the other hand achieves reasonable behavior (though clearly not expert behavior) in all of the environments, using only a few states as input. Surprisingly, the AverageFeatures method also performs quite well, even beating the full algorithm on some tasks, though failing quite badly on Pendulum. It seems that the task of running forward or backward is very well specified by a single state, since it can be inferred even without any information about the dynamics (except that which is encoded in the features learned from the initial dataset).

Learning skills from a single state

We investigate to what extent Deep RLSP can learn other skills where the reward is not clear. Evaluation on these tasks is much harder, because there is no ground truth reward. Therefore we evaluate qualitatively how similar the policies learned by Deep RLSP are to the original skill. Unlike the previous case, we do not reoptimize the learned reward and only look at the policies learned by Deep RLSP.

We consider skills learned by running Dynamics-Aware Unsupervised Discovery of Skills (DADS; Sharma et al., 2020). Since we are not interested in navigation, we remove the "x-y prior" used to get directional skills in DADS. We run DADS on the half-cheetah environment and select all skills that are not some form of running. This resulted in two skills: one in which the cheetah is moving forward making big leaps (*"jumping"*) and one in which

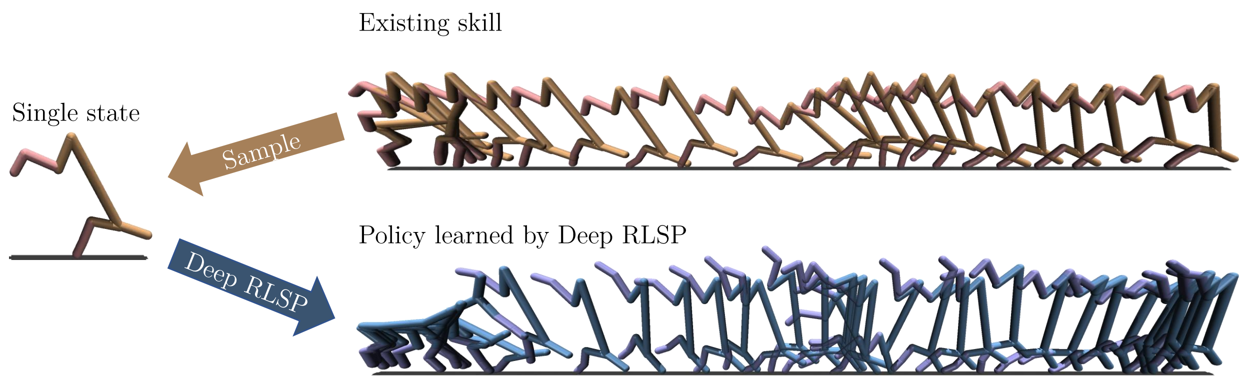


Figure 7.2: We sample a few states from a policy performing a specific skill to provide as input. Here, Deep RLSP learns to balance the cheetah on the front leg from a *single state*. We provide videos of the original skills and learned policies at: `https://sites.google.com/view/deep-rlsp`.

it is slowly moving forward on one leg (*"balancing"*). As before we roll out these policies and sample individual states from the trajectories to provide as an input for Deep RLSP. We then evaluate the policy learned by Deep RLSP. Since the best evaluation here is to simply watch what the learned policy does, we provide videos of the learned policies at `https://sites.google.com/view/deep-rlsp`.

The first thing to notice is that relative to the ablations, only Deep RLSP is even close to imitating the skill. None of the other policies resemble the original skills at all. While AverageFeatures could perform well on simple tasks such as running, the full algorithm is crucial to imitate more complex behavior.

Between Deep RLSP and GAIL the comparison is less clear. Deep RLSP can learn the balancing skill fairly well from a single state, which we visualize in Figure 7.2 (though we emphasize that the videos are much clearer). Like the original skill, the learned policy balances on one leg and slowly moves forward by jumping, though with slightly more erratic behavior. However, the learned policy sometimes drops back to its feet or falls over on its back. We suspect this is an artifact of the short horizon ($T \leq 10$) used for simulating the past in our algorithm. A small horizon is necessary to avoid compounding errors in the learned inverse dynamics model, but can cause the resulting behavior to be more unstable on timescales greater than T. We see similar behavior when given 10 or 50 states. GAIL leads to a good policy given a single transition, where the cheetah balances on its front leg and head (rather than just the front leg), but does not move forward very much. However, with 10 or 50 transition, the policies learned by GAIL do not look at all like balancing.

However, the jumping behavior is harder to learn, especially from a single state. We speculate that here a single state is less informative than the balancing state. In the balancing state, the low joint velocities tell us that the cheetah is not performing a flip, suggesting that we had optimized for this specific balancing state. On the other hand, with the jumping behavior, we only get a single state of the cheetah in the air with high velocity, which is likely not sufficient to determine what the jump looked like exactly. In line with this hypothesis, at 1 state Deep RLSP learns to erratically hop, at 10 states it executes a jump too vigorously and falls over, and at 50 states it makes larger hops (but still sometimes falls over).

The GAIL policies for jumping are also reasonable, though in a different way that makes it hard to compare. Using 1 or 10 transitions, the policy doesn't move very much, staying in contact with the ground most of the time. However, at 50 transitions, it performs behaviors that are noticeably forward hops, without falling over, leading to a policy that looks better than that learned by Deep RLSP.

7.4 Investigating the prior distribution

In Section 7.2, we considered the setting where the robot knows s_{-T}, since it is easier to understand and analyze what happens. However, typically we will not know s_{-T}, and will instead have some prior over it. Here, we compare RLSP in two settings: perfect knowledge of s_{-T} (as in Section 7.2), and no knowledge of s_{-T}, which we represent by using a uniform prior distribution over all states.

Side effects: Room with vase *(Figure 7.1a)* **and toy train** *(Figure 7.1b)*. In both room with vase and toy train, RLSP learns a smaller negative reward on broken vases when using a uniform prior. This is because RLSP considers many more feasible trajectories when using a uniform prior, many of which do not give Alice a chance to break the vase, as in Room with far away vase in Section 7.2. In room with vase, the small positive reward on carpets changes to a near-zero negative reward on carpets. With known s_{-T}, RLSP overfits to the few consistent trajectories, which usually go over carpets, whereas with a uniform prior it considers many more trajectories that often don't go over carpets, and so it correctly infers a near-zero weight. In toy train, the negative reward on broken trains becomes slightly more negative, while other features remain approximately the same. This may be because when Alice starts out closer to the toy train, she has more of an opportunity to break it, compared to the case where s_{-T} is known.

Implicit preference: Apple collection *(Figure 7.1d)*. Here, a uniform prior leads to a smaller positive weight on the number of apples in baskets compared to the case with known s_{-T}. Intuitively, this is because RLSP is considering cases where s_{-T} already has one or two apples in the basket, which implies that Alice has collected fewer apples and so must have been less interested in them. States where the basket starts with three or more apples are inconsistent with the observed s_0 and so do not have an effect on the gradient. Following the inferred reward still leads to good apple harvesting behavior.

Desirable side effects: Batteries *(Figure 7.1c)*. With the uniform prior, we see the same

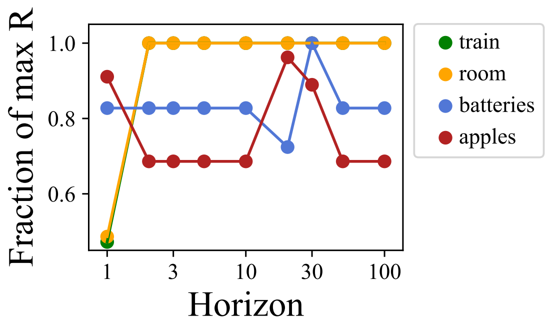


Figure 7.3: Reward achieved by π_{RLSP}, as a fraction of the expected reward of the optimal policy, for different values of Alice's planning horizon T.

behavior as in Apple collection, where RLSP with a uniform prior learns a slightly smaller negative reward on the batteries, since it considers states s_{-T} where the battery was already gone. In addition, due to the particular setup the battery must have been given to the train two timesteps prior, which means that in any state where the train started with very little charge, it was allowed to die even though a battery could have been provided before, leading to a near-zero *positive* weight on the train losing charge. Despite this, RLSP successfully delivers the battery to the train in both easy and hard cases.

"Unseen" side effect: Room with far away vase *(Figure 7.1e)*. With a uniform prior, we "see" the side effect: if Alice started at the purple door, then the shortest trajectory to the black door would break a vase. As a result, π_{RLSP} successfully avoids the vase (whereas it previously did not). Here, uncertainty over the initial state s_{-T} can counterintuitively *improve* the results, because it increases the diversity of trajectories considered, which prevents RLSP from "overfitting" to the few trajectories consistent with a known s_{-T} and s_0.

Overall, RLSP appears to be quite robust to the use of a uniform prior over s_{-T}, suggesting that we do not need to be particularly careful in the design of that prior.

7.5 Robustness to H's planning horizon

We investigate how RLSP performs when assuming the wrong value of H's planning horizon T. We vary the value of T assumed by RLSP, and report the true return achieved by π_{RLSP} obtained using the inferred reward and a *fixed* horizon for the robot to act. For this experiment, we used a uniform prior over s_{-T}, since with known s_{-T}, RLSP often detects that the given s_{-T} and s_0 are incompatible (when T is misspecified). The results are presented in Figure 7.3.

The performance worsens when RLSP assumes that Alice had a smaller planning horizon than she actually had. Intuitively, if we assume that Alice has only taken one or two actions ever, then even if we know the actions, they could have been in service of many goals, and so we end up quite uncertain about Alice's reward. For the Apple collection environment the underestimated horizon prevents R from learning anything at all, since all very short trajectories consistent with s_0 do not involve any apple collection.

When the assumed T is larger than the true horizon, RLSP correctly infers things the robot should *not* do. Knowing that the vase was not broken for longer than T timesteps is more evidence to suspect that Alice cared about not breaking the vase. However, overestimated T leads to worse performance at inferring implicit preferences, as in the Apples environment. If we assume Alice has only collected two apples in 100 timesteps, she must not have cared about them much, since she could have collected many more. The batteries environment is unusual – assuming that Alice has been acting for 100 timesteps, the only explanation for the observed s_0 is that Alice waited until the 98th timestep to put the battery into the train. This is not particularly consistent with any reward function, and performance degrades.

Overall, T is an important parameter and needs to be set appropriately. However, even when T is misspecified, performance tends to degrade gracefully to what would have happened if we optimized $\mathcal{R}_{\text{spec}}$ by itself, so RLSP does not hurt. In addition, if T is larger than it should be, then RLSP still tends to accurately infer parts of the reward that specify what not to do.

While this evaluation showed that RLSP is reasonably robust to the choice of planning horizon T and prior over s_{-T}, this may be specific to our gridworlds. In the real world, we often make long term hierarchical plans, and if we don't observe the entire plan (corresponding to a choice of T that is too small) it seems possible that we infer bad rewards, especially if we have an uninformative prior over s_{-T}. We do not know whether this will be a problem, and if so how bad it will be, and hope to investigate it in future work with more realistic environments.

7.6 Conflicts between the inferred reward and H's desires

In Section 7.2, we evaluated RLSP by combining the reward it infers with a specified reward to get a final reward $\theta_{\text{final}} = \theta_H + \lambda\theta_{\text{spec}}$. The problem of combining θ_H and θ_{spec} is difficult, since the two rewards incentivize different behaviors and will conflict. The *Additive* method above is a simple way of trading off between the two.

Both RLSP and the sampling algorithm of Appendix 5.1 can incorporate a prior over θ. Another way to combine the two rewards is to condition the prior on θ_{spec} before running the algorithms. In particular, we could replace our prior $P(\theta_H)$ with a new prior $P(\theta_H \mid \theta_{\text{spec}})$, such as a Gaussian distribution centered at θ_{spec}. When we use this prior, the reward returned by RLSP can be used as the final reward θ_{final}.

It might seem like this is a principled Bayesian method that allows us to combine the two rewards. However, the conflict between the two reward functions still exists. In this formulation, it arises in the new prior $P(\theta_H \mid \theta_{\text{spec}})$. Modeling this as a Gaussian centered at

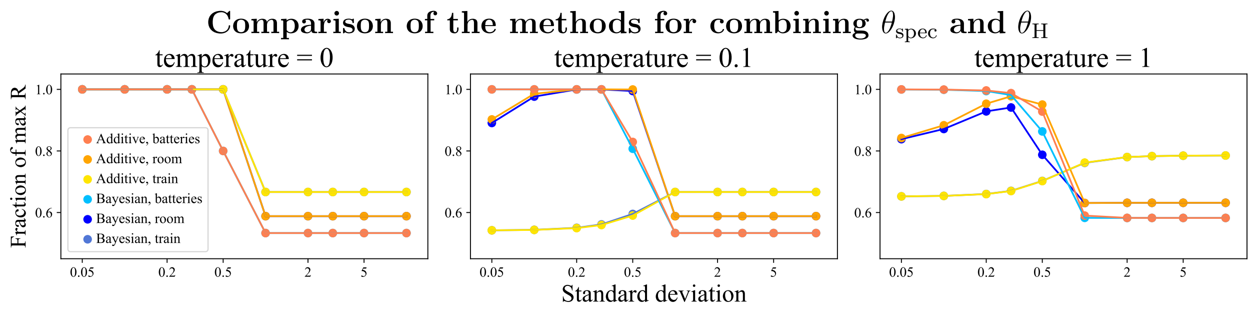


Figure 7.4: Comparison of the Additive and Bayesian methods. We show how the percentage of true reward obtained by π_{RLSP} varies as we change the tradeoff between the inferred reward $\mathcal{R}_H$ and the specified reward $\mathcal{R}_{\text{spec}}$. The zero temperature case corresponds to traditional value iteration; this often leads to identical behavior and so the lines overlap. So, we also show the results when planning with soft value iteration, varying the softmax temperature, to introduce some noise into the policy. Overall, there is not much difference between the two methods. We did not include the Apples environment because $\mathcal{R}_{\text{spec}}$ is uniformly zero and the Additive and Bayesian methods do exactly the same thing.

θ_{spec} suggests that before knowing s_0, it seems likely that θ_H is very similar to θ_{spec}. However, this is not true – Alice is probably providing the reward θ_{spec} to the robot so that it causes some *change* to the state that she has optimized, and so it will be *predictably* different from θ_{spec}. On the other hand, we do need to put high probability on θ_{spec}, since otherwise θ_{final} will not incentivize any of the behaviors that θ_{spec} did.

Nonetheless, this is another simple heuristic for how we might combine the two rewards, that manages the tradeoff between $\mathcal{R}_{\theta_{\text{spec}}}$ and $\mathcal{R}_H$. We compared the Additive and Bayesian methods by evaluating their robustness. We vary the parameter that controls the tradeoff and report the true reward obtained by π_{RLSP}, as a fraction of the expected true reward under the optimal policy. For the Bayesian method, we vary the standard deviation σ of the Gaussian prior over $\mathcal{R}_H$ that is centered at $\mathcal{R}_{\theta_{\text{spec}}}$. For the Additive method, the natural choice would be to vary λ; however, in order to make the results more comparable, we instead set $\lambda = 1$ and vary the standard deviation of the Gaussian prior used while inferring $\mathcal{R}_H$, which is centered at zero instead of at $\mathcal{R}_{\theta_{\text{spec}}}$. A larger standard deviation allows $\mathcal{R}_H$ to become larger in magnitude (since it is penalized less for deviating from the mean of zero reward), which effectively corresponds to a smaller λ.

While we typically create π_{RLSP} using value iteration, this leads to deterministic policies with very sharp changes in behavior that make it hard to see differences between methods, and so we also show results with soft value iteration, which creates stochastic policies that vary more continuously. As demonstrated in Figure 7.4, our experiments show that overall the two methods perform very similarly, with some evidence that the Additive method is slightly more robust. The Additive method also has the benefit that it can be applied in situations where the inferred reward and specified reward are over different feature spaces, by creating the final reward $\mathcal{R}_{\text{final}}(s) = {\theta_H}^T f_H(s) + \lambda \mathcal{R}_{\theta_{\text{spec}}}(s)$.

However, both of these methods are quite unprincipled, and do not resolve the conflict between the two goals. The presence of this conflict is worrying: it should not be the case that two different sources of information about the same underlying reward function *predictably* conflict; the fact that this is happening suggests that we have a poorly specified model somewhere. The problem is that we are implicitly treating $\mathcal{R}_{\theta_{\text{spec}}}$ incorrectly: both the Additive and Bayesian methods are implicitly suggesting that the true reward must be "near" $\mathcal{R}_{\theta_{\text{spec}}}$, but this is not actually the case: we believe that $\mathcal{R}_{\theta_{\text{spec}}}$ will capture some aspects of good behavior, but completely neglect other aspects; this need not correspond to low L2 distance in parameter space.

We could try to improve upon our assumptions about what information $\mathcal{R}_{\theta_{\text{spec}}}$ provides, in order to resolve the conflict. For example, rather than interpreting θ_{spec} as the parameters of a *reward function*, we could interpret it as a *changelist*: that is, it is a specification of how the environment should be changed, *relative to the current state of the environment.* In this case, in our vase environment, θ_{spec} would be interpreted as saying "all else equal, it is better for R to be at the purple door then at its current location", without making any claims about what is preferred when all else is not equal (Boutilier et al., 2004). This information could then be integrated with the inferred reward to provide a full reward function for R.

Taking a step back, we can see that in general, the inferred reward does not have enough information about what R should do, and so it must be combined with some information from H. In our evaluation so far, we have considered the additional information to come from θ_{spec}, but this is not necessary – it could just as well come from preferences, demonstrations, corrections, or natural language, for example. Thus, what we really want is a general formalism that allows us to infer rewards from the state of the world, combine them with information provided by H online, and then use the resulting information to act in the environment. We tackle this problem in the next chapter.

Chapter 8

Using assistance instead of reward learning to integrate information

A central premise of this dissertation is that we cannot specify a perfect reward function for powerful agents. A natural solution is to have agents be *uncertain* about the objective and *infer* it from observations. While we have shown that some of this information can be extracted from the state of the world, it also seems likely that some information will need to come from human feedback.

So far we have implicitly been using the *reward learning* paradigm to combine reward information from multiple sources, as in Chapter 7. In this chapter, we recast the entire reward learning paradigm as a special, constrained case of the *assistance* paradigm, and demonstrate that by removing these constraints we can gain significant qualitative benefits. We then show how to recast learning from the state of the world in the assistance paradigm to gain these benefits.

In the reward learning paradigm (Leike et al., 2018; Jeon et al., 2020; Christiano et al., 2017; Ziebart et al., 2010), a reward model is learned from human feedback, and then used by a control algorithm to select actions in the environment. Crucially, the control algorithm and the reward learning process do not interact: they are separate processes that optimize separate objectives. When learning the reward model, multiple forms of information can be integrated into a single posterior over reward functions.

In contrast, in the assistance paradigm (Hadfield-Menell et al., 2016; Fern et al., 2014), the human H is modeled as part of the environment and as having some latent goal that the agent R (for robot) does not know. R's goal is to maximize this (unknown) human goal. In this formulation, R must balance between actions that help learn about the unknown goal, and control actions that lead to high reward. There is a single policy is responsible for both learning about the goal (to the extent necessary) as well as acting in pursuit of the goal.

We claim that the assistance paradigm has several qualitative advantages over the reward learning paradigm. The goal of this chapter is to clarify and illustrate these advantages. We then show how to formalize state of the world learning within the assistance paradigm in Section 8.6.

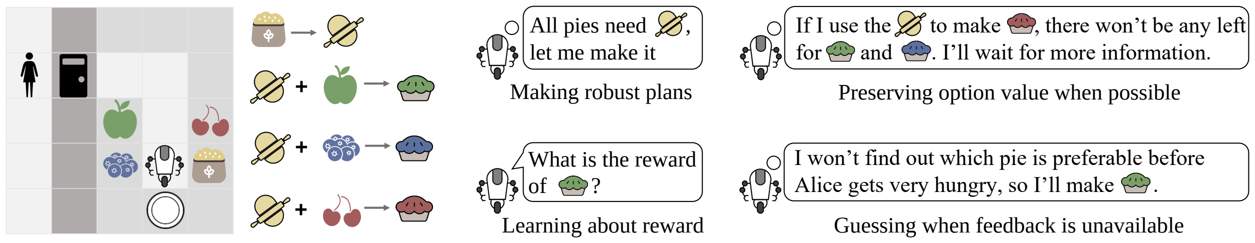

All pies need , let me make it
Making robust plans
If I use the to make , there won't be any left for and . I'll wait for more information.
Preserving option value when possible
What is the reward of ?
Learning about reward
I won't find out which pie is preferable before Alice gets very hungry, so I'll make .
Guessing when feedback is unavailable

Figure 8.1: R must cook a pie for H, by placing flour to make the pie dough, filling it with **A**pple, **B**lueberry, or **C**herry filling, and finally baking it. However, R does not know which filling H prefers, and H is not yet available for questions. What should R do in this situation? On the right, we show what qualitative reasoning we might want R to use to handle the situation.

Our key insight is that *by integrating reward learning and control using the learned reward into a single policy, the assistance formulation allows for reward learning to depend on how the reward will be used, and vice versa.* This integration enables new, desirable qualitative behaviors:

1. Decision-making for "control" can take into account the fact that the agent can do more "reward learning" in the future. This allows the agent to take actions that are robustly instrumentally useful now, knowing that the intermediate results can be used later once the reward is better understood (Section 8.4).

2. Decision-making for "reward learning" (e.g. which questions to ask the human) can take into account how the resulting updates to the reward will be used for "control". This allows the agent to only ask questions whose answers are decision-relevant (Section 8.4).

Consider for example the kitchen environment illustrated in Figure 8.1, in which R must bake a pie for H. R is uncertain about which type of pie H prefers to have, and currently H is at work and cannot answer R's questions. An assistive R can make the pie crust while waiting for H to return, after which R can ask her about her preferences over the filling (Section 8.4). R may never clarify all of H's preferences: for example, R only needs to know how to dispose of food if it turns out that the ingredients have gone bad (Section 8.4). If H will help with making the pie, R can allow H to disambiguate her desired pie by watching what filling she chooses (Section 8.4). These behaviors rely crucially on the *integration* of reward learning and control into a single policy, and as such vanilla reward learning agents do not show these behaviors.

To clarify and illustrate the advantages of the assistance paradigm, we first precisely characterize the differences between reward learning and assistance, by showing that *two phase, communicative assistance* is equivalent to reward learning (Section 8.3). We then give qualitative examples of desirable behaviors that can only be expressed once these restrictions are lifted, and thus are only exhibited by assistive agents (Section 8.4).

We do not mean to suggest that work on reward learning should be replaced by research on assistance. Amongst other limitations, assistive agents are very computationally complex. Our goal is simply to clarify what qualitative benefits an assistive formulation could theoretically provide. Further research is needed to develop efficient algorithms that can capture these benefits. Such algorithms may look like algorithms designed to solve assistance problems as we have formalized them here, but they may also look like modified variants of reward learning, where the modifications are designed to provide the qualitative benefits we identify.

8.1 Reward learning

We consider two variants of reward learning: *non-active* reward learning, in which R must infer the reward by observing H's behavior, and *active* reward learning, in which R may choose particular questions to ask H in order to get particular feedback.

A **non-active reward learning problem** $\mathcal{P} = \langle \mathcal{M} \backslash r, C, \langle \Theta, r_\theta, P_\Theta \rangle, \pi^H, k \rangle$ contains a POMDP without reward $\mathcal{M} \backslash r = \langle S, \mathcal{A}^R, \Omega^R, \mathcal{O}^R, \mathcal{T}, P_0, \gamma \rangle$, and instead R has access to a parameterized reward space $\langle \Theta, r_\theta, P_\Theta \rangle$. R is able to learn about θ^* by observing H make k different choices c, each chosen from a set of potential choices C. In order for R to learn from the human's choices, it also assumes access to the human decision function $\pi^H(c \mid \theta)$ that determines how the human makes choices for different possible reward functions r_θ. Common decision functions include perfect optimality (Ng and Russell, 2000) and Boltzmann rationality (Ziebart et al., 2010). As discussed in Section 2.4, there are many types of choices (Jeon et al., 2020), including demonstrations (Argall et al., 2009; Ng and Russell, 2000; Ziebart et al., 2010; Fu et al., 2017; Gao et al., 2012), comparisons (Zhang et al., 2017; Wirth et al., 2017; Christiano et al., 2017; Sadigh et al., 2017), corrections (Bajcsy et al., 2017), proxy rewards (Hadfield-Menell et al., 2017b), natural language (Fu et al., 2019), etc.

A policy decision function $f(c_{0:k-1})$ produces a policy π^R after observing H's choices. f is a *solution* if it maximizes expected reward $\mathbb{E}_{\theta \sim P_\Theta, c_{0:k-1} \sim \pi^H} \left[ER(f(c_{0:k-1})) \right]$. Since H's choices $c_{0:k-1}$ do not affect the state of the environment that R is acting in, this is equivalent to choosing π^R that maximizes expected reward given the posterior over reward functions, that is $\mathbb{E}_{\theta \sim P(\theta \mid c_{0:k-1})} \left[ER(\pi^R) \right]$.

An **active reward learning problem** $\mathcal{P} = \langle \mathcal{M} \backslash r, Q, C, \langle \Theta, r_\theta, P_\Theta \rangle, \pi^H, k \rangle$ adds the ability for R to ask H particular questions $q \in Q$ in order to get more targeted feedback about θ. The human decision function $\pi^H(c \mid q, \theta)$ now depends on the question asked. A *solution* consists of a question policy $\pi^R_Q(q_i \mid q_{0:i-1}, c_{0:i-1})$ and a policy decision function $f(q_{0:k-1}, c_{0:k-1})$ that maximize expected reward $\mathbb{E}_{\theta \sim P_\Theta, q_{0:k-1} \sim \pi^R_Q, c_{0:k-1} \sim \pi^H} \left[ER(f(q_{0:k-1}, c_{0:k-1})) \right]$.

A typical algorithm (Eric et al., 2008; Daniel et al., 2014; Maystre and Grossglauser, 2017; Christiano et al., 2017; Sadigh et al., 2017; Zhang et al., 2017; Wilde et al., 2020) will compute and ask $q \in Q$ that maximizes an *active learning criterion* such as information gain (Bıyık et al., 2019) or volume removal (Sadigh et al., 2017). Best results are achieved by selecting questions with the highest value of information (Cohn, Robert W, 2016; Zhang et al., 2017; Mindermann et al., 2018; Wilde et al., 2020), but these are usually much more computationally expensive. R then finds a policy that maximizes expected reward under the inferred distribution over θ, in order to approximately solve the original POMDP.

Note that a non-active reward learning problem is equivalent to an active reward learning problem with only one question, since having just a single question means that R has no choice in what feedback to get:

Proposition 7. *Every non-active reward learning problem $\langle \mathcal{M}\backslash r, C, \langle \Theta, r_\theta, P_\Theta \rangle, \pi^H, k \rangle$ can be reduced to an active reward learning problem.*

Proof. We construct the active reward learning problem as $\langle \mathcal{M}\backslash r, Q', C, \langle \Theta, r_\theta, P_\Theta \rangle, \pi^{H'}, k \rangle$, where $Q' \triangleq \{q_\phi\}$ where q_ϕ is some dummy question, and $\pi^{H'}(c \mid q, \theta) \triangleq \pi^H(c \mid \theta)$.

Suppose the solution to the new problem is $\langle \pi_Q^{R'}, f' \rangle$. Since f' is a solution, we have:

$$\begin{aligned}
f' &= \operatorname*{argmax}_{\hat{f}} \mathop{\mathbb{E}}_{\theta \sim P_\Theta, q_{0:k-1} \sim \pi_Q^{R'}, c_{0:k-1} \sim \pi^{H'}(\cdot \mid q_i, \theta)} \left[ER(\hat{f}(q_{0:k-1}, c_{0:k-1})) \right] \\
&= \operatorname*{argmax}_{\hat{f}} \mathop{\mathbb{E}}_{\theta \sim P_\Theta, q_{0:k-1} = q_\phi, c_{0:k-1} \sim \pi^{H'}(\cdot \mid q_\phi, \theta)} \left[ER(\hat{f}(q_{0:k-1} = q_\phi, c_{0:k-1})) \right] \qquad \text{all } q \text{ are } q_\phi \\
&= \operatorname*{argmax}_{\hat{f}} \mathop{\mathbb{E}}_{\theta \sim P_\Theta, c_{0:k-1} \sim \pi^H(\cdot \mid \theta)} \left[ER(\hat{f}(q_{0:k-1} = q_\phi, c_{0:k-1})) \right].
\end{aligned}$$

Thus $f(c_{0:k-1}) = f'(q_{0:k-1} = q_\phi, c_{0:k-1})$ is a maximizer of $\mathbb{E}_{\theta \sim P_\Theta, c_{0:k-1} \sim \pi^H(\cdot \mid \theta)} \left[ER(\hat{f}(c_{0:k-1}) \right]$, making it a solution to our original problem. □

Proposition 8. *Every active reward learning problem $\langle \mathcal{M}\backslash r, Q, C, \langle \Theta, r_\theta, P_\Theta \rangle, \pi^H, k \rangle$ with $|Q| = 1$ can be reduced to a non-active reward learning problem.*

Proof. Let the sole question in Q be q_ϕ. We construct the non-active reward learning problem as $\langle \mathcal{M}\backslash r, C, \langle \Theta, r_\theta, P_\Theta \rangle, \pi^{H'}, k \rangle$, with $\pi^{H'}(c \mid \theta) = \pi^H(c \mid q_\phi, \theta)$.

Suppose the solution to the new problem is f'. Then we can construct a solution to the original problem as follows. First, note that π_Q^R must be $\pi_Q^R(q_i \mid q_{0:i-1}, c_{0:i-1}) = \mathbb{1}[q_i = q_\phi]$, since there is only one possible question q_ϕ. Then by inverting the steps in the proof of Proposition 7, we can see that f' is a maximizer of $\mathbb{E}_{\theta \sim P_\Theta, q_{0:k-1} \sim \pi_Q^R, c_{0:k-1} \sim \pi^H(\cdot \mid q_i, \theta)} \left[ER(\hat{f}(\cdot \mid c_{0:k-1})) \right]$. Thus, by defining $f(q_{0:k-1}, c_{0:k-1}) = f'(c_{0:k-1})$, we get a maximizer to our original problem, making $\langle \pi_Q^R, f \rangle$ a solution to the original problem. □

8.2 Assistance

The key idea of assistance is that helpful behaviors like reward learning are incentivized when R does not know the true reward $\mathcal{R}_{\theta^*}$ and can only learn about it by observing human behavior. So, we model the human H as part of the environment, leading to a two-agent

POMDP, and assume there is some true reward $\mathcal{R}_{\theta^*}$ that only H has access to, while the robot R only has access to a model relating $\mathcal{R}_{\theta^*}$ to H's behavior. Intuitively, as R acts in the environment, it will also observe H's behavior, which it can use to make inferences about the true reward.

Following Hadfield-Menell et al. (2016)[1], we define an **assistance game** $\mathcal{M}$ as a tuple

$$\mathcal{M} = \langle S, \{\mathcal{A}^H, \mathcal{A}^R\}, \{\Omega^H, \Omega^R\}, \{\mathcal{O}^H, \mathcal{O}^R\}, \mathcal{T}, P_S, \gamma, \langle \Theta, r_\theta, P_\Theta \rangle \rangle.$$

Here S is a finite set of states, $\mathcal{A}^H$ a finite set of actions for H, Ω^H a finite set of observations for H, and $\mathcal{O}^H : S \to \Delta(\Omega^H)$ an observation function for H (respectively $\mathcal{A}^R, \Omega^R, \mathcal{O}^R$ for R). The transition function $\mathcal{T} : S \times \mathcal{A}^H \times \mathcal{A}^R \to \Delta(S)$ gives the probability over next states given the current state and both actions. The initial state is sampled from $P_S \in \Delta(S)$. Θ is a set of possible reward function parameters θ which parameterize a class of reward functions $r_\theta : S \times \mathcal{A}^H \times \mathcal{A}^R \times S \to \mathbb{R}$, and P_θ is the distribution from which θ is sampled. $\gamma \in (0, 1)$ is a discount factor.

As with POMDPs, policies can depend on history. Both H and R are able to observe each other's actions, and on a given timestep, R acts before H. We use $\tau_t^R : (\Omega^R \times \mathcal{A}^H \times \mathcal{A}^R)^t$ to denote R's observations until time t, and τ_t^H for H's observations; thus R's policy can be written as $\pi^R(a^R \mid o_t^R, \tau_{t-1}^R)$, while H's can be written as $\pi^H(a^H \mid o_t^H, a_t^R, \tau_{t-1}^H, \theta)$. Note that unlike H, R *does not* observe the reward parameter θ, and must infer θ much like it does the hidden state.

A **fully observable assistance game** is one in which both H and R can observe the full state. In such cases, we omit $\Omega^H, \Omega^R, \mathcal{O}^H$ and $\mathcal{O}^R$.

Since we have not yet specified how H behaves, it is not clear what the agent should optimize for. Should it be playing a Nash strategy or optimal strategy pair of the game, and if so, which one? Should it use a non-equilibrium policy, since humans likely do not use equilibrium strategies? This is a key hyperparameter in assistance games, as it determines the *communication protocol* for H and R. For maximum generality, we can equip the assistance game with a *policy-conditioned belief* $B : \Pi^R \to \Delta(\Pi^H)$ over π^H, which specifies how the human responds to the agent's choice of policy (Halpern and Pass, 2018). The agent's goal is to maximize expected reward given this belief. We use policy-conditioned beliefs as opposed to a simple unconditional distribution over human policies, because it allows us to model a wide range of situations, including situations with prior coordination, or where humans adapt to the robot's policy as a result of prior interactions.

Prior work on assistance games (Hadfield-Menell et al., 2016; Malik et al., 2018; Woodward et al., 2019) focuses on finding optimal strategy pairs. This corresponds to a belief that H will know and perfectly adapt to R's policy, as formalized below:

[1]Relative to Hadfield-Menell et al. (2016), our definition allows for partial observability and requires that the initial distribution over S and Θ be independent. We also have H choose her action sequentially after R, rather than simultaneously with R, in order to better parallel the reward learning setting.

Proposition 9. *Let* $\mathcal{M} = \langle S, \{\mathcal{A}^H, \mathcal{A}^R\}, \{\Omega^H, \Omega^R\}, \{\mathcal{O}^H, \mathcal{O}^R\}, \mathcal{T}, P_S, \gamma, \langle\Theta, r_\theta, P_\Theta\rangle\rangle$ *be an assistance game. Let* $B(\pi^R)(\pi^H) \propto \mathbb{1}\left[EJR(\pi^H, \pi^R) = \max_{\tilde{\pi}^H \in \Pi^H} EJR(\tilde{\pi}^H, \pi^R)\right]$ *be an associated policy-conditioned belief. Let* π^R *be the solution to* $\langle\mathcal{M}, B\rangle$*. Then* $\langle B(\pi^R), \pi^R\rangle$ *is an optimal strategy pair.*

Proof. Let $\langle\overline{\pi}^H, \overline{\pi}^R\rangle$ be an arbitrary strategy pair. Then $EJR(\overline{\pi}^H, \overline{\pi}^R) \leq EJR(B(\overline{\pi}^R), \overline{\pi}^R)$ by the definition of B, and $EJR(B(\overline{\pi}^R), \overline{\pi}^R) \leq EJR(B(\pi^R), \pi^R)$ by the definition of π^R. Thus $EJR(\overline{\pi}^H, \overline{\pi}^R) \leq EJR(B(\pi^R), \pi^R)$. Since $\langle\overline{\pi}^H, \overline{\pi}^R\rangle$ was assumed to be arbitrary, $\langle B(\pi^R), \pi^R\rangle$ is an optimal strategy pair. □

However, our goal is to compare assistance to reward learning. Typical reward learning algorithms assume access to a model of human decision-making: for example, H might be modeled as *optimal* (Ng and Russell, 2000) or *Boltzmann-rational* (Ziebart et al., 2010). As a result, we also assume that we have access to a model of human decision-making π^H. Note that π^H *depends on* θ: we are effectively assuming that we know how H *chooses how to behave* given a particular reward r_θ. This assumption corresponds to the policy-conditioned belief $B(\pi_R)(\tilde{\pi}^H) = \mathbb{1}\left[\tilde{\pi}^H = \pi^H\right]$. We define an **assistance problem** $\mathcal{P}$ as a pair $\langle\mathcal{M}, \pi^H\rangle$ where π^H is a human policy for the assistance game $\mathcal{M}$.

Given an assistance problem, a robot policy π_R induces a probability distribution over trajectories: $\tau \sim \langle s_0, \theta, \pi^H, \pi^R\rangle$, $\tau \in [S \times \mathcal{A}^H \times \mathcal{A}^R]^*$ (where X^* denotes a sequence of X). We denote the support of this distribution by $\text{Traj}(\pi_R)$. The *expected reward* of a robot policy for $\langle\mathcal{M}, \pi^H\rangle$ is given by

$$ER(\pi^R) = \mathop{\mathbb{E}}_{s_0 \sim P_S, \theta \sim P_\theta, \tau \sim \langle s_0, \theta, \pi^H, \pi^R\rangle}\left[\sum_{t=0}^{\infty} \gamma^t r_\theta(s_t, a_t^H, a_t^R, s_{t+1})\right].$$

A *solution* of $\langle\mathcal{M}, \pi^H\rangle$ is a robot policy that maximizes expected reward: $\pi^R = \operatorname*{argmax}_{\tilde{\pi}^R} ER(\tilde{\pi}^R)$.

Solving assistance problems

Once the π^H is given, H can be thought of as an aspect of the environment, and θ can be thought of as a particularly useful piece of information for estimating how good actions are. This suggests that we can reduce the assistance problem to an equivalent POMDP, and then solve the POMDP. Following Desai (2017), the key idea is to embed π^H in the transition function $\mathcal{T}$ and embed θ in the state. It turns out we must also include the human's *previous* action a^H in the state, so that the robot can observe it, and so that the reward can be computed.

In theory, to embed potentially non-Markovian π^H in $\mathcal{T}$, we need to embed the entire history of the trajectory in the state, but this leads to extremely large POMDPs. In our experiments, we only consider Markovian human policies, for which we do not need to embed the full history, keeping the state space manageable. Thus, the policy can be written as

$\pi^H(a^H \mid o^H, a^R, \theta)$. To ensure that R must infer θ from human behavior, as in the original assistance game, the observation function does *not* reveal θ, but *does* reveal the previous human action a^H.

Formally, suppose that we have an assistance problem $\langle \mathcal{M}, \pi^H \rangle$ with:

$$\mathcal{M} = \langle S, \{\mathcal{A}^H, \mathcal{A}^R\}, \{\Omega^H, \Omega^R\}, \{\mathcal{O}^H, \mathcal{O}^R\}, \mathcal{T}, P_S, \gamma, \langle \Theta, r_\theta, P_\Theta \rangle \rangle.$$

Then, the transformation $\mathcal{M} \mapsto \mathcal{M}'$ is given as follows:

$$\begin{aligned}
S' &\triangleq S \times \mathcal{A}^H \times \Theta && \text{State space} \\
\Omega' &\triangleq \Omega^R \times \mathcal{A}^H && \text{Observation space} \\
O'(o' \mid s') &= O'((o^R, a_1^H) \mid (s, a_2^H, \theta)) && \text{Observation function} \\
&\triangleq \mathbb{1}\left[a_1^H = a_2^H\right] \cdot \mathcal{O}^R(o^R \mid s) \\
\mathcal{T}'(s_2' \mid s_1', a^R) &= \mathcal{T}'((s_2, a_1^H, \theta_2) \mid (s_1, a_0^H, \theta_1), a^R) && \text{Transition function} \\
&\triangleq \mathcal{T}(s_2 \mid s_1, a_1^H, a^R) \cdot \mathbb{1}\left[\theta_2 = \theta_1\right] \\
&\quad \cdot \sum_{o^H \in \Omega^H} \mathcal{O}^H(o^H \mid s_1) \cdot \pi^H(a_1^H \mid o^H, a^R, \theta) \\
r'(s_1', a^R, s_2') &= r'((s_1, a_0^H, \theta), a^R, (s_2, a_1^H, \theta)) && \text{Reward function} \\
&\triangleq r_\theta(s_1, a_1^H, a^R, s_2) \\
P_0'(s') &= P_0'((s, a^H, \theta)) && \text{Initial state distribution} \\
&\triangleq P_S(s) \cdot P_\Theta(\theta) \cdot \mathbb{1}\left[a^H = a_{init}^H\right] && \text{where } a_{init}^H \text{ is arbitrary}
\end{aligned}$$

In the case where the original assistance problem is fully observable, the resulting POMDP is an instance of a Bayes-Adaptive MDP (Martin, 1967; Duff, 2002).

Any robot policy π^R can be translated from the APOMDP $\mathcal{M}$ naturally into an identical policy on $\mathcal{M}'$. Note that in either case, policies are mappings from $(\Omega^R, \mathcal{A}^H, \mathcal{A}^R)^* \times \Omega^R$ to $\Delta(\mathcal{A}^R)$. This transformation preserves optimal agent policies:

Proposition 10. *A policy π^R is a solution of $\mathcal{M}$ if and only if it is a solution of $\mathcal{M}'$.*

Proof. Recall that an optimal policy π^* in the POMDP $\mathcal{M}'$ is one that maximizes the expected value:

$$\text{EV}(\pi) = \mathop{\mathbb{E}}_{s_0' \sim P_0', \tau' \sim \langle s_0', \pi \rangle} \left[\sum_{t=0}^{\infty} \gamma^t r'(s_t', a_t, s_{t+1}) \right] = \mathop{\mathbb{E}}_{s_0' \sim P_0', \tau' \sim \langle s_0', \pi \rangle} \left[\sum_{t=0}^{\infty} \gamma^t r_\theta(s_t, a_t^H, a_t, s_{t+1}) \right]$$

where the trajectories τ's are sequences of state, action pairs drawn from the distribution induced by the policy, starting from state s_0.

Similarly, an optimal robot policy π^{R*} in the APOMDP $\mathcal{M}$ is one that maximizes its expected reward:

$$\mathrm{ER}(\pi^R) = \mathop{\mathbb{E}}_{s_0 \sim P_S, \theta \sim P_\Theta, \tau \sim \langle s_0, \theta, \pi^R \rangle} \left[\sum_{t=0}^{\infty} \gamma^t r_\theta(s_t, a_t^H, a_t^R, s_{t+1}) \right].$$

To show that the optimal policies coincide, it suffices to show that for any π, $\mathrm{ER}(\pi)$ (in $\mathcal{M}$) is equal to $\mathrm{EV}(\pi)$ (in $\mathcal{M}'$). To do this, we will show that π induces the "same" distributions over the trajectories. For mathematical convenience, we will abuse notation and consider trajectories of the form $\tau; \theta \in (S, \mathcal{A}^H, \mathcal{A}^R)^* \times \Theta$; it is easy to translate trajectories of this form to trajectories in either $\mathcal{M}'$ or $\mathcal{M}$.

We will show that the sequence $\tau; \theta$ has the same probability when the robot takes the policy π in both $\mathcal{M}'$ and $\mathcal{M}$ by induction on the lengths of the sequence.

First, consider the case of length 1 sequences. $\tau; \theta = [(s, a^R, a^H); \theta]$. Under both $\mathcal{M}'$ and $\mathcal{M}$, s and θ are drawn from P_S and P_Θ respectively. Similarly, a^R and a^H are drawn from $\pi^R(\cdot \mid o_0^R)$ and $\pi^H(\cdot \mid o^H, a^R, \theta)$ respectively. So the distribution of length 1 sequences is the same under both $\mathcal{M}'$ and $\mathcal{M}$.

Now, consider some longer sequence $\tau; \theta = [(s_1, a_1^R, a_1^H), ..., (s_t, a_t^R, a_t^H); \theta]$. By the inductive hypothesis, the distribution of $(s_1, a_1^H, a_1^R), ..., (s_{t-1}, a_{t-1}^H, a_{t-1}^R)$ and θ are identical; it suffices to show that (s_t, a_t^H, a_t^R) has the same distribution, conditioned on the other parts of $\tau; \theta$, under $\mathcal{M}'$ and under $\mathcal{M}$. Yet by construction, s_t is drawn from the same distribution $\mathcal{T}(\cdot | s_{t-1}, a_{t-1}^H, a_{t-1}^R)$, a_t^H is drawn from the same distribution $\pi^H(\cdot \mid o_t^H, a_t^R, \theta)$, and a_t^R is drawn from the same distribution $\pi^R(\cdot \mid o_t^R, \tau_{t-1}^R))$. $\square$

When $\mathcal{M}$ is fully observable, in the reduced POMDP θ is the only part of the state not directly observable to the robot, making it an instance of a *hidden-goal MDP* (Fern et al., 2014). For computational tractability, much of the work on hidden goals (Javdani et al., 2015; Fern et al., 2014) selects actions *assuming that all goal ambiguity is resolved in one step.* This effectively separates reward learning and control in the same way as typical reward learning algorithms, thus negating many of the benefits we highlight in future sections. Intention-aware motion planning (Bandyopadhyay et al., 2013) also embeds the human goal in the state in order to avoid collisions with humans during motion planning, but does not consider applications for assistance.

Macindoe et al. (2012) uses the formulation of a POMDP with a hidden goal to produce an assistive agent in a cops and robbers gridworld environment. Nikolaidis et al. (2015) assumes a dataset of joint human-robot demonstrations, which they leverage to learn "types" of humans that can then be inferred online using a POMDP framework. This is similar to solving an assistance problem, where we think of the different values of θ as different "types" of humans. Chen et al. (2018) uses an assistance-style framework in which the unknown parameter is the human's trust in the robot (rather than the reward θ). Woodward et al. (2019) uses deep reinforcement learning to solve an assistance game in which the team must collect either plums or lemons. To our knowledge, these are the only prior works that use an assistive formulation in a way that does not ignore the information-gathering aspect of actions. While these works typically focus on algorithms to solve assistance games, we instead focus on the qualitative benefits of using an assistance formulation.

Since we can reduce an assistance problem to a regular POMDP, we can use any POMDP solver to find the optimal π^R. In our examples for this paper, we use an exact solver when feasible, and point-based value iteration (PBVI) (Pineau et al., 2003) or deep reinforcement learning (DRL) when not. When using DRL, we require recurrent models, since the optimal policy can depend on history.

A common confusion is to ask how DRL can be used, given that it requires a reward signal, whereas R does not know the reward function by assumption. This stems from a misunderstanding of what it means for R "not to know" the reward function. When DRL is run, at the beginning of each episode, a specific value of θ is sampled as part of the initial state. The *learned policy* π^R is not provided with θ: it can only see its observations o^R and human actions a^H, and so it is accurate to say that π^R "does not know" the reward function. However, the reward is calculated by the DRL algorithm, not by π^R, and the algorithm can and does use the sampled value of θ for this computation. π^R can then implicitly learn the correlation between the actions a^H chosen by π^H, and the high reward values that the DRL algorithm computes; this can be thought of as an implicit estimation of θ in order to choose the right actions.

8.3 Reward learning as two-phase communicative assistance

There are two key differences between reward learning and assistance. First, reward learning algorithms split reward learning and control into two separate phases, while assistance merges them into a single phase. Second, in reward learning, the human's only role is to communicate reward information to the robot, while in assistance the human can help with the task. These two properties exactly characterize the difference between the two: reward learning problems and communicative assistance problems with two phases can be reduced to each other, in a very natural way.

A **communicative assistance problem** is one in which the transition function $\mathcal{T}$ and the reward function r_θ are independent of the choice of human action a^H, and the human policy $\pi^H(\cdot \mid o^H, a^R, \theta)$ is independent of the observation o^H. Thus, in a communicative assistance problem, H's actions only serve to respond to R, and have no effects on the state or the reward (other than by influencing R). Such problems can be cast as instances of HOP-POMDPs (Rosenthal and Veloso, 2011).

For the notion of two phases, we will also need to classify robot actions as communicative or not. We will assume that there is some distinguished action a^R_{noop} that "does nothing". Then, a robot action $\hat{a}^R$ is **communicative** if for any s, a^H, s' we have $\mathcal{T}(s' \mid s, a^H, \hat{a}^R) = \mathcal{T}(s' \mid s, a^H, a^R_{noop})$ and $R(s, a^H, \hat{a}^R, s') = R(s, a^H, a^R_{noop}, s')$. A robot action is **physical** if it is not communicative.

Now consider a communicative assistance problem $\langle \mathcal{M}, \pi^H \rangle$ with noop action a^R_{noop} and let the optimal robot policy be π^{R*}. Intuitively, we would like to say that there is an initial communication phase in which the only thing that happens is that H responds to questions from R, and then a second action phase in which H does nothing and R acts. Formally, the assistance problem is **two phase with actions at** t_{act} if it satisfies the following property:

$$\exists a^H_{noop} \in \mathcal{A}^H,\ \forall \tau \in \text{Traj}(\pi^{R*}),\ \left[\forall t < t_{act} : a^R_t \text{ is communicative } \wedge \forall t \geq t_{act} :\ a^H_t = a^H_{noop}\right].$$

Thus, in a two phase assistance problem, every trajectory from an optimal policy can be split into a "communication" phase where R cannot act and an "action" phase where H cannot communicate.

Reducing reward learning to assistance

We can convert an active reward learning problem to a two-phase communicative assistance problem in an intuitive way: we add Q to the set of robot actions, make C the set of human actions, add a timestep counter to the state, and construct the reward such that an optimal policy must switch between the two phases after k questions. A non-active reward learning problem can first be converted to an active reward learning problem.

Proposition 11. *Every active reward learning problem $\mathcal{P} = \langle \mathcal{M}, Q, C, \langle \Theta, r_\theta, P_\Theta \rangle, \pi^H, k \rangle$ can be reduced to an equivalent two phase communicative assistance problem $\mathcal{P}' = \langle \mathcal{M}', \pi^{H'} \rangle$.*

Proof. Let $\mathcal{M} = \langle S, A, \Omega, O, \mathcal{T}, P_0, \gamma \rangle$. Let $q_0 \in Q$ be some question and $c_0 \in C$ be some (unrelated) choice. Let N be a set of fresh states $\{n_0, \dots n_{k-1}\}$: we will use these to count the number of questions asked so far. Then, we construct the new assistance problem $\mathcal{P}' = \langle \mathcal{M}', \pi^{H'}, a^{R'}_{noop} \rangle$ with $\mathcal{M}' \triangleq \langle S', \{C, A^{R'}\}, \{\Omega^{H'}, \Omega^{R'}\}, \{O^{H'}, O^{R'}\}, \mathcal{T}', P'_S, \gamma, \langle \Theta, r'_\theta, P_\Theta \rangle \rangle$

as follows:

$$
\begin{aligned}
S' &\triangleq S \cup N && \text{State space}\\
P'_S(\hat{s}) &\triangleq \mathbb{1}\left[\hat{s} = n_0\right] && \text{Initial state distribution}\\
A^{R'} &\triangleq A \cup Q && \text{Robot actions}\\
\Omega^{H'} &\triangleq S && H\text{'s observation space}\\
\Omega^{R'} &\triangleq \Omega \cup N && R\text{'s observation space}\\
O^{H'}(o^{H'} \mid \hat{s}) &\triangleq \mathbb{1}\left[o^{H'} = \hat{s}\right] && H\text{'s observation function}\\
O^{R'}(o^{R'} \mid \hat{s}) &\triangleq \begin{cases} \mathbb{1}\left[o^{R'} = \hat{s}\right], & \hat{s} \in N \\ O(o^{R'} \mid \hat{s}), & \text{else} \end{cases} && R\text{'s observation function}\\
\mathcal{T}'(\hat{s}' \mid \hat{s}, a^H, a^R) &\triangleq \begin{cases} P_0(\hat{s}'), & \hat{s} = n_{k-1}, \\ \mathbb{1}\left[\hat{s}' = n_{i+1}\right], & \hat{s} = n_i \text{ with } i < k-1 \\ \mathcal{T}(\hat{s}' \mid \hat{s}, a^R), & \hat{s} \in S \text{ and } a^R \in A, \\ \mathbb{1}\left[s' = s\right], & \text{else} \end{cases} && \text{Transition function}\\
r'_\theta(\hat{s}, a^H, a^R, \hat{s}') &\triangleq \begin{cases} -\infty, & \hat{s} \in N \text{ and } a^R \notin Q, \\ -\infty, & \hat{s} \in S \text{ and } a^R \in Q, \\ 0, & \hat{s} \in N \text{ and } a^R \in Q, \\ r_\theta(s, a^R, s'), & \text{else} \end{cases} && \text{Reward function}\\
\pi^{H'}(a^H \mid o^H, a^R, \theta) &\triangleq \begin{cases} \pi^H(a^H \mid a^R, \theta), & a^R \in Q \\ c_0, & \text{else} \end{cases} && \text{Human policy}\\
a^{R'}_{noop} &\triangleq q_0 && \text{Distinguished noop action}
\end{aligned}
$$

Technically r'_θ should not be allowed to return $-\infty$. However, since S and A are finite, r_θ is bounded, and so there exists some large finite negative number that is functionally equivalent to $-\infty$ that we could use instead.

Looking at the definitions, we can see $\mathcal{T}'$ and r' are independent of a^H, and $\pi^{H'}$ is independent of o^H, making this a communicative assistance problem. By inspection, we can see that every $q \in Q$ is a communicative robot action. Any $a^R \notin Q$ must not be a communicative action, because the reward r'_θ differs between a^R and q_0. Thus, the communicative robot actions are Q and the physical robot actions are A.

Note that by construction of P'_S and $\mathcal{T}$, we must have $s_i = n_i$ for $i \in \{0, 1, \dots k-1\}$, after which s_k is sampled from P_0 and all $s_t \in S$ for $t \geq k$. Given this, by inspecting r'_θ, we can see that an optimal policy must have $a^R_{0:k-1} \in Q$ and $a^R_{k:} \notin Q$ to avoid the $-\infty$ rewards. Since $a^R_{k:} \notin Q$, we have $a^H_{k:} = c_0$. Thus, setting $a^H_{noop} = c_0$, we have that the assistance problem is two phase with actions at $t_{act} = k$, as required.

Let a policy $\pi^{R'}$ for the assistance problem be **reasonable** if it never assigns positive probability to $a^R \in A$ when $t < k$ or to $a^R \in Q$ when $t \geq k$. Then, for any reasonable policy $\pi^{R'}$ we can construct an analogous pair $\langle \pi_Q^R, f \rangle$ to the original problem $\mathcal{P}$ as follows:

$$\pi_Q^R(q_i \mid q_{0:i-1}, c_{0:i-1}) \triangleq \pi^{R'}(q_i \mid o^R_{0:i-1} = n_{0:i-1}, a^R_{0:i-1} = q_{0:i-1}, a^H_{0:i-1} = c_{0:i-1}),$$

$$f(q_{0:k-1}, c_{0:k-1})(a_t \mid o_{0:t}, a_{0:t-1}) \triangleq \pi^{R'}(a_t \mid o^R_{0:t+k}, a^R_{0:t+k-1}, a^H_{0:t+k-1}),$$

where for the second equation we have

$$\begin{array}{lll} o^R_{0:k-1} = n_{0:k-1} & a^R_{0:k-1} = q_{0:k-1} & a^H_{0:k-1} = c_{0:k-1} \\ o^R_{k:t+k} = o_{0:t} & a^R_{k:t+k-1} = a_{0:t-1} & a^H_{k:t+k-1} = a^H_{noop} \end{array}$$

Note that this is a bijective mapping.

Consider some such policy $\pi^{R'}$ and its analogous pair $\langle \pi_Q^R, f \rangle$. By construction of $\mathcal{T}$, we have that the first k states in any trajectory are $n_{0:k-1}$ and the next state is distributed as $P_0(\cdot)$. By our assumption on $\pi^{R'}$ we know that the first k robot actions must be selected from Q and the remaining robot actions must be selected from A, which also implies (based on π^H) that after the the remaining human actions must be c_0. Finally, looking at r_θ we can see that the first k timesteps get 0 reward. Thus:

$$\begin{aligned} ER_{\mathcal{P}'}(\pi^{R'}) &= \mathop{\mathbb{E}}_{s'_0 \sim P'_S, \theta \sim P_\theta, \tau \sim \langle s'_0, \theta, \pi^{H'}, \tilde{\pi}^R \rangle} \left[\sum_{t=0}^{\infty} \gamma^t r_\theta(s'_t, a^{H'}_t, a^{R'}_t, s'_{t+1}) \right] \\ &= \mathop{\mathbb{E}}_{\theta \sim P_\theta, a^{R'}_{0:k-1} \sim \pi^R, a^{H'}_{0:k-1} \sim \pi^H, s'_k \sim P_0, \tau'_{k:} \sim \langle s_k, \theta, \pi^{H'}, \tilde{\pi}^R \rangle} \left[\sum_{t=k}^{\infty} \gamma^t r_\theta(s'_t, a^{H'}_t, a^{R'}_t, s'_{t+1}) \right] \\ &= \mathop{\mathbb{E}}_{\theta \sim P_\theta, q_{0:k-1} \sim \pi_Q^R, c_{0:k-1} \sim \pi^H, s_0 \sim P_0, \tau \sim \langle s_0, \theta, f(q_{0:k-1}, c_{0:k-1}) \rangle} \left[\gamma^k \sum_{t=0}^{\infty} \gamma^t r_\theta(s_t, a_t, s_{t+1}) \right] \\ &= \gamma^k \mathop{\mathbb{E}}_{\theta \sim P_\Theta, q_{0:k-1} \sim \pi_Q^R, c_{0:k-1} \sim \pi^H} \left[ER(f(q_{0:k-1}, c_{0:k-1})) \right], \end{aligned}$$

which is the objective of the reward learning problem scaled by γ^k.

Since we have a bijection between reasonable policies in $\mathcal{P}'$ and tuples in $\mathcal{P}$ that preserves the objectives (up to a constant), given a solution π^{R*} to $\mathcal{P}'$ (which must be reasonable), its analogous pair $\langle \pi_Q^R, f \rangle$ must be a solution to $\mathcal{P}$. □

Corollary 12. *Every non-active reward learning problem $\langle \mathcal{M}, C, \langle \Theta, r_\theta, P_\Theta \rangle, \pi^H, k \rangle$ can be reduced to an equivalent two phase communicative assistance problem $\langle \mathcal{M}', \pi^{H'} \rangle$.*

Proof. Apply Proposition 7 followed by Proposition 11. □

Reducing assistance to reward learning

The reduction from a two-phase communicative assistance problem to an active reward learning problem is similarly straightforward: we interpret R's communicative actions as questions and H's actions as answers. There is once again a simple generalization to non-active reward learning.

Proposition 13. *Every two-phase communicative assistance problem $\langle \mathcal{M}, \pi^H, a^R_{noop} \rangle$ can be reduced to an equivalent active reward learning problem.*

Proof. Let $\mathcal{M} = \langle S, \{\mathcal{A}^H, \mathcal{A}^R\}, \{\Omega^H, \Omega^R\}, \{\mathcal{O}^H, \mathcal{O}^R\}, \mathcal{T}, P_S, \gamma, \langle \Theta, r_\theta, P_\Theta \rangle \rangle$ be the assistance game, and let the assistance problem's action phase start at t_{act}. Let $a^H_\phi \in \mathcal{A}^H$ be some arbitrary human action and $o^H_\phi \in \Omega^H$ be some arbitrary human observation. We construct the new active reward learning problem $\langle \mathcal{M}', Q', C', \langle \Theta, r'_\theta, P_\Theta \rangle, \pi^{H'}, k' \rangle$ as follows:

$$
\begin{aligned}
Q' &\triangleq \{a^R \in \mathcal{A}^R : a^R \text{ is communicative}\} && \text{Questions} \\
C' &\triangleq \mathcal{A}^H && \text{Answers} \\
\mathcal{M}' &\triangleq \langle S, A', \Omega^R, \mathcal{O}^R, \mathcal{T}', P'_0, \gamma \rangle && \text{POMDP} \\
A' &\triangleq \mathcal{A}^R \backslash Q' && \text{Physical actions} \\
\mathcal{T}'(s' \mid s, a^R) &\triangleq \mathcal{T}(s' \mid s, a^H_\phi, a^R) && \text{Transition function} \\
k' &\triangleq t_{act} && \text{Number of questions} \\
P'_0(s) &\triangleq \sum_{s_{0:k'} \in S} P_{\mathcal{M}}(s_{0:k'}, s_{k'+1} = s \mid a^R_{0:k'} = a^R_{noop}, a^H_{0:k'} = a^H_\phi) && \text{Initial state distribution} \\
r'_\theta(s, a^R, s') &\triangleq r_\theta(s, a^H_\phi, a^R, s') && \text{Reward function} \\
\pi^{H'}(c \mid q, \theta) &\triangleq \pi^H(c \mid o^H_\phi, q, \theta) && \text{Human decision function}
\end{aligned}
$$

Note that it is fine to use a^H_ϕ in $\mathcal{T}, r_\theta$ and to use o^H_ϕ in π^H even though they were chosen arbitrarily, because since the assistance problem is communicative, the result does not depend on the choice. The $P_{\mathcal{M}}$ term in the initial state distribution denotes the probability of a trajectory under $\mathcal{M}$ and can be computed as

$$P_{\mathcal{M}}(s_{0:T+1} \mid a^R_{0:T}, a^H_{0:T}) = P_S(s_0) \prod_{t=0}^{T} \mathcal{T}(s_{t+1} \mid s_t, a^H_t, a^R_t).$$

Given some pair $\langle \pi^{R'}_Q, f' \rangle$ to the active reward learning problem, we construct a policy for the assistance problem as

$$\pi^R(a^R_t \mid o^R_t, \tau^R_{t-1}) \triangleq \begin{cases} \pi^{R'}_Q(a^R_t \mid a^R_{0:t-1}, a^H_{0:t-1}), & t < k \text{ and } a^R_{0:t} \in Q' \\ f'(a^R_{0:k-1}, a^H_{0:k-1})(a^R_t \mid o^R_{k:t}, a^R_{k:t-1}), & t \geq k \text{ and } a^R_{0:k-1} \in Q' \text{ and } a^R_{k:t} \in A' \\ 0, & \text{else} \end{cases}.$$

We show that there must exist a solution to $\mathcal{P}$ that is the analogous policy to some pair. Assume towards contradiction that this is not the case, and that there is a solution π^{R*} that is not the analogous policy to some pair. Then we have a few cases:

1. π^{R*} assigns positive probability to $a_i^R = a \notin Q'$ for $i < k$. This contradicts the two-phase assumption.

2. π^{R*} assigns positive probability to $a_i^R = q \in Q$ for $i \geq k$. This contradicts the two-phase assumption.

3. $\pi^{R*}(a_t^R \mid o_t^R, \tau_{t-1}^R)$ depends on the value of o_i^R for some $i < k$. Since both $a_{0:k-1}^H$ and $a_{0:k-1}^R$ cannot affect the state or reward (as they are communicative), the distribution over $o_{0:k-1}^R$ is fixed and independent of π^R, and so there must be some other π^R that is independent of $o_{0:k-1}^R$ that does at least as well. That π^R would be the analogous policy to some pair, giving a contradiction.

Now, suppose we have some pair $\langle \pi_Q^{R'}, f' \rangle$, and let its analogous policy be π^R. Then we have:

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\substack{\theta \sim P_\Theta \\ q_{0:k-1} \sim \pi_Q^{R'} \\ c_{0:k-1} \sim \pi^{H'}}} \left[ER(f'(q_{0:k-1}, c_{0:k-1})) \right] \\
&= \mathop{\mathbb{E}}_{\theta \sim P_\Theta} \left[\mathop{\mathbb{E}}_{\substack{q_{0:k-1} \sim \pi^R \\ c_{0:k-1} \sim \pi^H}} \left[ER(f'(q_{0:k-1}, c_{0:k-1})) \right] \right] \\
&= \mathop{\mathbb{E}}_{\theta \sim P_\Theta} \left[\mathop{\mathbb{E}}_{\substack{q_{0:k-1} \sim \pi^R \\ c_{0:k-1} \sim \pi^H}} \left[\mathop{\mathbb{E}}_{\substack{s_0 \sim P_0' \\ a_t^R \sim f'(q_{0:k-1}, c_{0:k-1}) \\ s_{t+1} \sim \mathcal{T}'(\cdot \mid s_t, a_t^R)}} \left[\sum_{t=0}^{\infty} \gamma^t r'_\theta(s_t, a_t^R, s_{t+1}) \right] \right] \right] \\
&= \mathop{\mathbb{E}}_{\theta \sim P_\Theta} \left[\mathop{\mathbb{E}}_{\substack{q_{0:k-1} \sim \pi^R \\ c_{0:k-1} \sim \pi^H}} \left[\mathop{\mathbb{E}}_{\substack{s_k \sim P_0' \\ a_t^R \sim \pi^R(\cdot \mid \langle c_{0:k-1}, o_{k:t} \rangle, \langle q_{0:k-1}, a_{k:t-1} \rangle) \\ s_{t+1} \sim \mathcal{T}'(\cdot \mid s_t, a_t^R)}} \left[\frac{1}{\gamma^k} \sum_{t=k}^{\infty} \gamma^t r'_\theta(s_t, a_t^R, s_{t+1}) \right] \right] \right] \\
&= \mathop{\mathbb{E}}_{\theta \sim P_\Theta} \left[\mathop{\mathbb{E}}_{\substack{q_{0:k-1} \sim \pi^R \\ c_{0:k-1} \sim \pi^H}} \left[\mathop{\mathbb{E}}_{\substack{s_k \sim P_0' \\ a_t^R \sim \pi^R(\cdot \mid \langle c_{0:k-1}, o_{k:t} \rangle, \langle q_{0:k-1}, a_{k:t-1} \rangle) \\ s_{t+1} \sim \mathcal{T}'(\cdot \mid s_t, a_t^R)}} \left[\frac{1}{\gamma^k} \sum_{t=k}^{\infty} \gamma^t r_\theta(s_t, a_\phi^H, a_t^R, s_{t+1}) \right] \right] \right]
\end{aligned}$$

However, since all the actions in the first phase are communicative and thus don't impact state or reward, the first k timesteps in the two phase assistance game have constant reward in expectation. Let $C = \mathbb{E}_{s_{0:k}}\left[\sum_{t=0}^{k-1}\gamma^t r_\theta(s_t, a^H_\phi, a^R_{noop}, s_{t+1})\right]$. This gives us:

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\theta\sim P_\Theta, q_{0:k-1}\sim\pi_Q^{R'}, c_{0:k-1}\sim\pi^H}\left[ER(f'(q_{0:k-1}, c_{0:k-1}))\right] \\
&= \mathop{\mathbb{E}}_{\theta\sim P_\Theta}\left[\mathop{\mathbb{E}}_{s_0\sim P_S, \theta\sim P_\Theta, \tau\sim\langle s_0,\theta,\pi^H,\pi^R\rangle}\left[\frac{1}{\gamma^k}\sum_{t=0}^{\infty}\gamma^t r_\theta(s_t, a^H_t, a^R_t, s_{t+1})\right]\right] - \frac{1}{\gamma^k}C \\
&= \frac{1}{\gamma^k}\left(ER(\pi^R) - C\right).
\end{aligned}$$

Thus, if $\langle \pi_Q^{R'}, f'\rangle$ is a solution to the active reward learning problem, then π^R is a solution of the two-phase communicative assistance problem. $\square$

Corollary 14. *If a two-phase communicative assistance problem $\langle \mathcal{M}, \pi^H, a^R_{noop}\rangle$ has exactly one communicative robot action, it can be reduced to an equivalent non-active reward learning problem.*

Proof. Apply Proposition 13 followed by Proposition 8. (Note that the construction from Proposition 13 does lead to an active reward learning problem with a single question, meeting the precondition for Proposition 8.) $\square$

8.4 Qualitative improvements for general assistance

We have seen that reward learning is equivalent to two-phase communicative assistance problems, where inferring the reward distribution can be separated from control using the reward distribution. However, for general assistance games, it is necessary to merge estimation and control, leading to several new qualitative behaviors. When the two phase restriction is lifted, we observe *plans conditional on future feedback* and *relevance aware active learning.* When the communicative restriction is lifted, we observe *learning from physical actions.*

We demonstrate these qualitative behaviors in simple environments using point-based value iteration (PBVI) or deep reinforcement learning (DRL). For communicative assistance problems, we also consider two baselines:

1. **Active reward learning.** This is the reward learning paradigm discussed so far.

2. **Interactive reward learning.** This is a variant of reward learning that aims to recover some of the benefits of interactivity, by alternating reward learning and acting phases. During an action phase, R chooses actions that maximize expected reward *under its current belief over θ* (without "knowing" that its belief may change), while during a reward learning phase, R chooses questions that maximizes information gain.

Plans conditional on future feedback

Here, we show how an assistive agent can make plans that depend on obtaining information about θ in the future. The agent can first take some "preparatory" actions whose results can be used later once the agent has clarified details about θ. A reward learning agent would not be able to do this, as it would require three phases (acting, then learning, then acting again).

We illustrate this with our original kitchen environment (Figure 8.1). R must bake a pie for H, but doesn't know what type of pie H wants: **A**pple, **B**lueberry, or **C**herry. Each type has a weight specifying the preference for that pie. Assuming people tend to like apple pie the most and cherry pie the least, we have $\theta_{\mathrm{A}} \sim \text{Uniform}[2, 4]$, $\theta_{\mathrm{B}} \sim \text{Uniform}[1, 3]$, and $\theta_{\mathrm{C}} \sim \text{Uniform}[0, 2]$. We define the questions $Q = \{q_A, q_B, q_C\}$, where q_X means "What is the value of θ_X?", and thus, the answer set is $C = \mathbb{R}$.

R can select ingredients to assemble the pie. Eventually, R must use "bake", which bakes the selected ingredients into a finished pie, resulting in reward that depends on what type of pie has been created. H initially starts outside the room, but will return at some prespecified time. r_θ assigns a cost of asking a question of 0.1 if H is inside the room, and 3 otherwise. The horizon is 6 timesteps.

In this environment, R needs to bake either apple or blueberry pie (cherry is never preferred over apple) within 6 timesteps, and may query H about her preferences about the pie. Making the pie takes 3 timesteps: first R must make flour into dough, then it must add one of the fillings, and finally it must bake the pie. Baking the correct pie results in +2 reward, while baking the wrong one results in a penalty of -1. In addition, H might be away for several timesteps at the start of the episode. Querying H costs 0.1 when she is present and 3 when she is away.

We use PBVI to train an agent for this assistance problem with different settings for how long H is initially away.

Assistance. *Regardless of H's preferences*, R will need to use flour to make pie dough. So, R always makes the pie dough first, before querying H about her preferences. Whether R then queries H about her preferences depends on how late H returns. If H arrives home before timestep 5, R will query her about her preferences and then make the appropriate pie as expected. However, if H will arrive later, then there will not be enough time to query her for her preferences and bake a pie. Instead, R bakes an apple pie, since its prior suggests that that's what H wants.

This behavior, where R takes actions (making dough) that are robustly good but waits on actions (adding the filling) whose reward will be clarified in the future, is very related to *conservative agency* (Turner et al., 2020).

Reward learning. The assistance solution requires R to act (to make dough), then to learn preferences, and then to act again (to make pie). A reward learning agent can only have two phases, and so we see one of two suboptimal behaviors. First, R could stay in the learning phase until H returns home, then ask which pie she prefers, and then make the pie from scratch. Second, R could make an apple pie without asking H her preferences. (In this case there would be no learning phase.) Which of these happens depends on the particular method and hyperparameters used.

Interactive reward learning. Adding interactivity is not sufficient to get the correct behavior. Suppose we start with an action phase. The highest reward plan under R's current belief over θ is to bake an apple pie, so that's what it will do, as long as the phase lasts long enough. Conversely, suppose we start with a learning phase. In this case, R does nothing until H returns, and then asks about her preferences. Once we switch to an action phase, it bakes the appropriate pie from scratch.

Why was integration of reward learning and control needed? The plan "make pie dough and then wait" (a control behavior) is only a good plan because the agent has some way of learning what filling to use in the future (a reward learning behavior). Thus, this plan can only be selected because when selecting control behaviors the agent can reason about future reward learning behaviors.

Relevance aware active learning

Once we relax the two-phase restriction, R starts to further optimize *whether* and *when* it asks questions. In particular, since R may be uncertain about whether a question's answer will even be necessary, R will only ask questions once they become *immediately relevant* to the task at hand. In contrast, a reward learning agent would have to decide at the beginning of the episode (during the learning phase) whether or not to ask these questions, and so cannot evaluate how relevant they are.

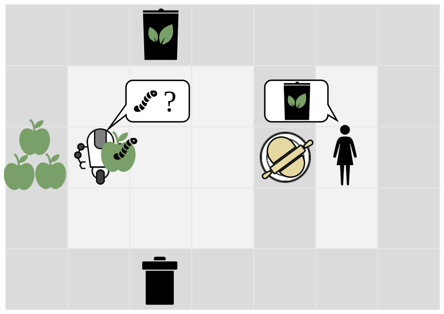


Figure 8.2: The wormy-apples kitchen environment. H wants an apple, but R might discover worms in the apple, and have to dispose of it in either of the trash or compost bins.

Consider for example a modification to the kitchen environment: R knows that H wants an apple pie, but when R picks up some apples, there is a 20% chance that it finds worms in some of the apples. R is unsure whether H wants her compost bin to have worms, and so does not know whether to dispose of the bad apples in the trash or compost bin. The robot gets 2 reward for making an apple pie (regardless of how it disposed of any wormy apples), and gets -2 reward if it disposes of the apples in the wrong container. Additionally, asking a question incurs a cost of 0.1. Ideally R would only clarify H's preferences around the disposal of wormy apples when R happens to pick up wormy apples.

We use PBVI to solve this environment, and report the results below.

Assistance. An assistive R only asks about wormy apples when it needs to dispose of one. R always starts by picking up apples. If the apple does not have worms, R immediately uses the apples to bake the pie. If some apples have worms and the cost of asking a question is sufficiently low, R elicits H's preferences and disposes of the apples appropriately. It then bakes the pie with the remaining apples. This policy always gets the 2 reward from baking the pie and never incurs the -2 reward from disposing of wormy apples in the wrong bin. It only has to ask H a question 80% of the time, and so incurs the 0.1 cost of asking a question 20% of the time, leading to a total expected undiscounted reward of 1.98.

This behavior, in which questions are asked only if they are useful for constraining future behavior, has been shown previously using probabilistic recipe trees (PRTs) (Kamar et al., 2009), but to our knowledge has not been shown with optimization-based approaches.

Reward learning. A reward learning policy must have only two phases and so would show one of two undesirable behaviors: either it would always ask H where to dispose of wormy apples, or it never asks and instead guesses when it does encounter wormy apples.

With a lower discount rate ($\lambda = 0.9$), R's policy never asks questions and instead simply tries to make the apple pie, guessing which bin to dispose of wormy apples in if it encounters any. Intuitively, since it would have to always ask the question at the beginning, it would always incur a cost of 0.1 as well as delay the pie by a timestep resulting in 10% less value, and this is only valuable when there turn out to be worms *and* its guess about which bin to dispose of them in is incorrect, which only happens 10% of the time. This ultimately isn't worthwhile. This achieves an expected undiscounted reward of 1.8.

With a higher discount rate of $\lambda = 0.99$, the two-phase policy will always ask about which bin to dispose of wormy apples in, achieving 1.9 expected undiscounted reward.

Interactive reward learning. If we start in the action phase and R picks up wormy apples, it will dispose of them in an arbitrary bin without asking H about her preferences, because it doesn't "know" that it will get the opportunity to do so. Alternatively, if we start with a learning phase, R will ask H where to dispose of wormy apples, even if R would never pick up any wormy apples.

Why was integration of reward learning and control needed? The question "where should I dispose of wormy apples" (a reward learning behavior) should only be asked when the agent picks up apples and they happen to be wormy (a control behavior). This can only happen when the question selection mechanism is able to reason about how the resulting information will be used for control.

While this example might seem trite, complex settings usually have *many* more questions. Should R ask if H prefers seedless apples, should scientists ever invent them? Perhaps R should ask if H still prefers apple over cherry when angry? Asking about all possible situations is not scalable.

Learning from physical actions

So far we have considered *communicative* assistance problems, in which H only provides feedback rather than acting to maximize reward herself. Allowing H to have physical actions enables a greater variety of potential behaviors. Most clearly, when R knows the reward (that is, P_Θ puts support over a single θ), assistance games become equivalent to human-AI collaboration (Nikolaidis and Shah, 2013; Carroll et al., 2019; Dimitrakakis et al., 2017).

With uncertain rewards, R can learn just by observing how H acts in an environment, and then work with H to maximize reward, *all within a single episode*, as in shared autonomy with intent inference (Javdani et al., 2015; Brooks and Szafir, 2019) and other works that interpret human actions as communicative Whitney et al. (2017). This can significantly reduce the burden on H in providing reward information to R. Some work has shown that in such situations, humans tend to be *pedagogic*: they knowingly take individually suboptimal actions, in order to more effectively convey the goal to the agent (Ho et al., 2016; Hadfield-Menell et al., 2016). An assistive R who knows this can quickly learn what H wants, and help her accomplish her goals.

We illustrate this with a variant of our kitchen environment, shown in Figure 8.3. There are no longer questions and answers. Both H and R can move to an adjacent free space, and pick up and place the various objects. Only R may bake the dessert. R is uncertain whether H prefers cake or cherry pie. Preparing the desired recipe provides a base value of $V = 10$, while the other recipe provides a base value of $V = 1$. Since H doesn't want the preparation to take too long, the actual reward when a dessert is made is given by $r_t = V \cdot f(t)$, with $f(t) = 1 - (t/N)^4$, and $N = 20$ as the episode horizon.

For both recipes, it is individually more efficient for H to pick up the dough first. However, we assume H is pedagogic and wants to quickly show R which recipe she wants. So, if she wants cake, she will pick up the chocolate first to signal to R that cake is the preferred dessert.

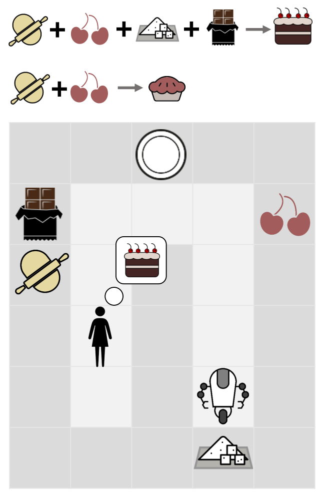


Figure 8.3: The CakeOrPie variant of the kitchen environment. H is equally likely to prefer cake or pie. Communication must take place through physical actions alone.

It is not clear how exactly to think about this from a reward learning perspective: there aren't any communicative human actions since every action alters the state of the environment. In addition, there is no clear way to separate out a given trajectory into two phases. This situation cannot be easily coerced into the reward learning paradigm, and so we only report the results in the assistance paradigm. We solve the environment using Deep Q-Networks (DQN; (Mnih et al., 2013)). We ran 6 seeds for $5M$ timesteps and a learning rate of 10^{-4}; results are shown in Figure 8.4.

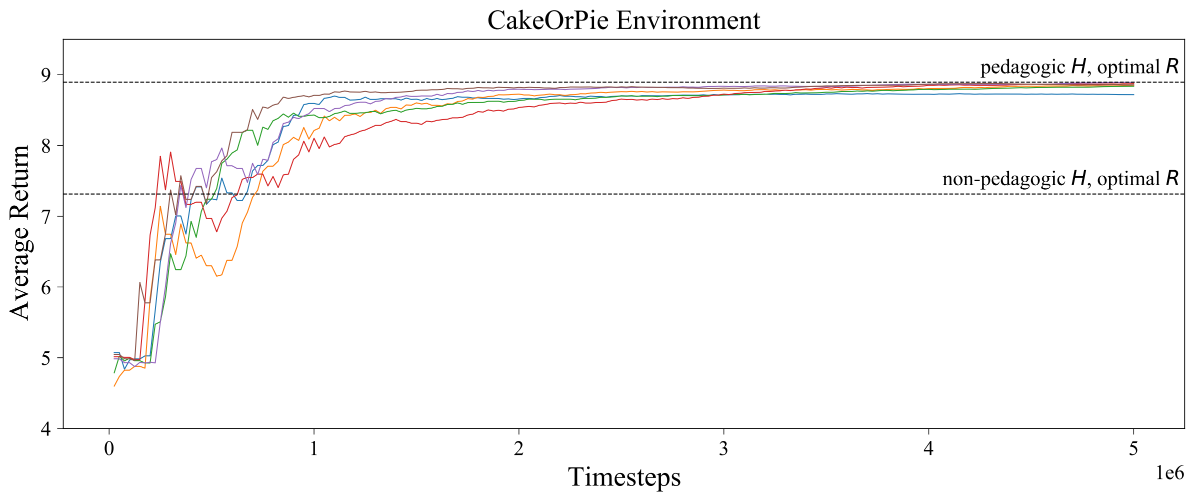


Figure 8.4: DQN smoothed learning curves on the CakeOrPie environment, with 6 seeds over $5M$ timesteps and learning rate of 10^{-4}.

The assistive R handles this situation perfectly. It waits to see which ingredient H walks towards, infers which recipe H wants, and then helps H by putting in the ingredients from its side of the environment and baking the dessert. This is equivalent to pragmatic reasoning (Goodman and Frank, 2016): "H would have gone towards the chocolate if she wanted cake, so the fact that she picked up the dough implies that she wants cherry pie". We emphasize that R is not explicitly programmed to reason in this manner.

Note that R is not limited to learning from H's physical actions: R can also use its own physical actions to "query" the human for information (Woodward et al., 2019; Sadigh et al., 2016).

8.5 Discussion

So far we have discussed the benefits of assistance over reward learning. Are there any downsides?

The major limitation of assistance is that assistance problems are significantly more computationally complex, since we treat the unknown reward θ as the hidden state of a POMDP. We are hopeful that this can be solved through the application of deep reinforcement learning, which has been demonstrated to scale to large state, action and observation spaces (OpenAI, 2018; Vinyals et al., 2019). Another avenue for future work is to modify active reward learning algorithms in order to gain the benefits outlined in Section 8.4, while maintaining their computational efficiency.

In addition, assistive agents will typically extract more information from H than reward learning agents. While this leads to benefits when correct inferences are made, it can also lead to significantly worse failures when π^H is misspecified. We don't see this as a major limitation: to the extent this is a major worry, we can design π^H so that the robot only makes inferences about human behavior in specific situations. For example, by having π^H be independent of θ in a given state s, we ensure that the robot does not make any inferences about θ in that state.

Limitations of assistance and reward learning

In addition to the downsides above, which are specific to assistance, there a number of challenges that apply to both reward learning and assistance:

Human modeling. A major motivation for both paradigms is that reward specification is very difficult. However, now we need to specify a prior over reward functions, and the human model π^H. Consequently, misspecification can still lead to bad results (Armstrong et al., 2020; Carey, 2018). While it should certainly be easier to specify a prior over θ with a "grain of truth" on the true reward θ^* than to specify θ^* directly, it is less clear that we can specify π^H well.

One possibility is to add uncertainty over the human policy π^H. However, this can only go so far: information about θ must come from *somewhere*. If R is sufficiently uncertain about θ and π^H, then it cannot learn about the reward (Armstrong and Mindermann, 2018). Thus, for good performance we need to model π^H. While imitation learning can lead to good results (Carroll et al., 2019), the best results will likely require insights from a broad range of fields that study human behavior.

Assumption that H knows θ. Both paradigms assume that H knows her reward exactly, but in practice, human preferences change over time (Allais, 1979; Cyert and DeGroot, 1975; Shogren et al., 2000). We could model this as the human changing their subgoals (Michini and How, 2012; Park et al., 2020), adapting to the robot (Nikolaidis et al., 2017) or learning from experience (Chan et al., 2019).

Dependence on uncertainty. All of the behaviors of Section 8.4, as well as previously explored benefits such as off switch corrigibility (Hadfield-Menell et al., 2017a), depend on R expecting to gain information about θ. However, R will eventually exhaust the available information about θ. If everything is perfectly specified, this is not a problem: R will have converged to the true θ^*. However, in the case of misspecification, after convergence R is effectively certain in an incorrect θ, which has many troubling problems that we sought to avoid in the first place (Yudkowsky, year unknown).

8.6 Modeling the state of the world with assistance

Our original goal in this chapter was to find a more principled way of integrating the reward inferred using RLSP with information learned from human feedback and then act on that information. The assistance paradigm provides a natural way of doing so. We model the environment as an assistance problem, but enforce the condition that for the first T timesteps, R does not yet exist, and is only deployed after T timesteps. At the time of deployment, R can update its prior P_Θ based on the observed state of the environment, and subsequently it may continue to ask H for more feedback about what to do.

Concretely, let us consider some state of the world problem $\langle \mathcal{M}, s_0, \pi^H, T, \langle \Theta, \mathcal{R}_\theta, P_\Theta \rangle \rangle$. We can view this as capturing the first T timesteps of a special assistance problem with a discrete deployment event. We have $\langle \langle S, \{\mathcal{A}^H, \mathcal{A}^R\}, \{\Omega^H, \Omega^R\}, \{\mathcal{O}^H, \mathcal{O}^R\}, \mathcal{T}, P_S, \gamma, \langle \Theta, r_\theta, P_\Theta \rangle \rangle, \pi^H \rangle$ as a standard assistance problem, and add as a restriction that R must take a noop action a^R_{noop} for the first T timesteps, and additionally is not allowed to observe the environment or H's actions. After these T timesteps, R is deployed and can act as normal within the assistance problem.

This is technically speaking not a standard assistance problem, because we are imposing conditions on R's policy π^R, as well as what R gets to observe. We might consider it a constrained assistance problem. Nonetheless, all of the discussion in the previous sections still applies, and in particular, it can still be reduced to a POMDP, and solved using some POMDP solver.

From this perspective, the point of the algorithms introduced in Chapters 5 and 6 that solve the state of the world problem is that they are more efficient algorithms that can solve a subpart of this constrained assistance problem. Specifically, they compute how R should update its belief over θ upon its deployment when observing s_0. However, in order to get an *optimal* policy for the assistance problem, we need to have the full distribution over θ, which only Algorithm 1 does; Algorithms 2 and 3 only compute a point estimate and thus should be thought of as efficient approximations.

Consider for example the kitchen office environment in Figure 8.5. In this environment, H wants to get work done on her laptop (which R cannot do on her behalf), and also eventually wants to eat a pie as in our original kitchen environment in Figure 8.1. In addition, there is a vase in the middle of the room, which H does not want to break, as in Figure 1.2. However, R does not know any of this, and is uncertain about the rewards of all three of these features. As before, R can ask H questions about her preferences in order to clarify what it should do.

At the beginning of a trajectory s_{-T}, before R has been deployed, H has just entered the room through a door, ready to begin acting in the environment. When R is deployed in state s_0, it observes that H is now working at her laptop, and the vase remains intact. Simply by observing the state of the environment, R can infer that H probably cares about getting her work done (since she is at the laptop), and that she doesn't want to break the vase (since she must have taken the long way around the vase in order to get to the laptop). However, this doesn't tell R what it should do: it can't help with H's work, and keeping the vase intact is only an injunction of what not to do, rather than what to do.

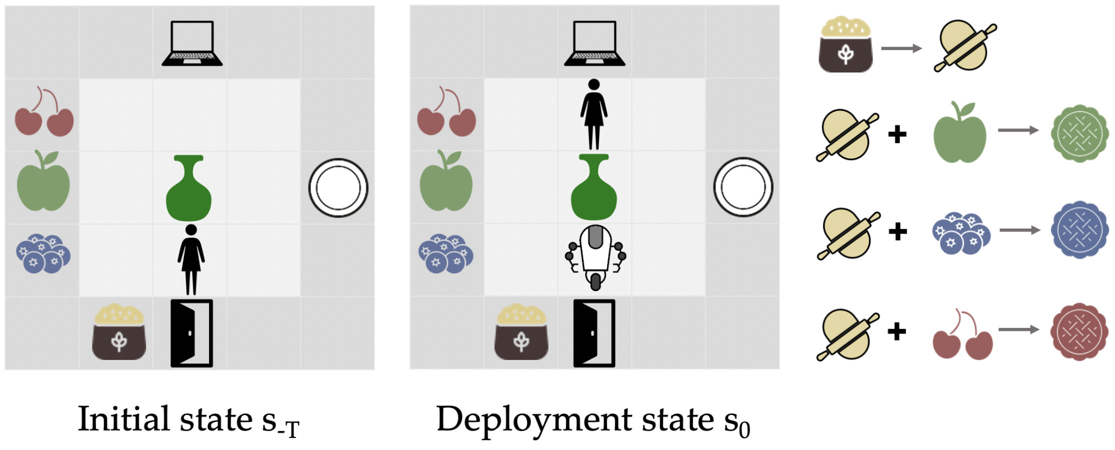


Figure 8.5: Kitchen office environment. In this environment, R is uncertain both about what type of pie H would like, as well as whether it is acceptable to break vases. At the time of deployment, R observes that H is working at her laptop, but the vase is still intact, and so R can infer that vases should not be broken. However, R still does not know which pie H would prefer, and so asks her for her preferences before finishing the pie.

R is aware that H might want a specific type of pie, and that is something that it can help with. As a result, it asks H about her pie preferences, and then makes the appropriate pie type for her to eat.

Chapter 9

Conclusion

One major goal of artificial intelligence is to build artificial agents that capably perform tasks that we want. In order to build such agents, we must have some way of transmitting information about what we want to the agent. The current paradigm in reinforcement learning is to accomplish this by specifying a *reward function*, that defines which states are desirable and which are not. Unfortunately, it is very difficult for humans to correctly specify such a reward function.

Instead, it is important that our agents treat all source of information as *observations* that provide information about the intended goal, rather than *definitions* of it. Once we make this distinction, it becomes evident that our agents should collect many different kinds of information, and integrate them together, in order to make better inferences about the intended goal.

This dissertation identified a potentially important source of such information: the state of the world. Unlike other sources of information, such as human demonstrations and comparisons, this source is free: it does not require humans to change their behavior at all. The agent must simply look at the world, infer what humans have put effort into, and make appropriate inferences about what it should and should not do. While this will likely not be enough to uniquely identify the task that the agent should perform, it can reduce the amount of information that the agent needs to collect from other, more expensive sources.

We have demonstrated both theoretically and empirically that learning from the state of the world can enable agents to make meaningful inferences, at least within some simple environments. However, there is still much work to be done to turn this into a practical method that can be applied in realistic scenarios.

9.1 Avenues for improvement

There are multiple avenues for further research, both conceptually and practically.

Conceptual changes

Handling multiple humans. In many realistic environments, there is not a single human H who is influencing the environment: many people act simultaneously, and the state is a result of joint optimization by all of them. One approach is to infer H's goals by modeling other people as part of the (very complex) environment. However, this seems like a particularly challenging approach, and may require a fairly complex human model in order to account for interpersonal interactions: a simple Boltzmann rational model may not be sufficient.

An alternative approach is to distinguish all of the humans as separate from the environment, and to instead use the state of the world to infer the *norms* by which they operate. These norms are likely the most relevant aspect of the reward function for R to know, as they determine how R should constrain its behavior within the group. They may also be easier to infer, since by definition nearly all of the humans will agree on the norms and so they will have an outsized impact on the state of the world.

Humans are optimized for the environment. Even when the state of the world is optimized for human preferences, this may not be a result of *human* optimization, as assumed in this dissertation. For example, we prefer that the atmosphere contain oxygen for us to breathe. That the atmosphere has enough oxygen for us to breathe is not a result of human optimization – indeed, it seems that the atmosphere meets this preference *in spite of* human action. Rather, humans were optimized through the process of evolution to be able to breathe the existing atmosphere.

The algorithms presented in this dissertation would not be able to infer such a preference, except indirectly, for example by inferring that humans prefer not to die or get sick, and thus instrumentally preferring that the atmosphere contain sufficient oxygen. The issue is that these algorithms assume that humans are the source of all optimization, and thus neglect the fact that humans have themselves be optimized. It is unclear how to modify the algorithms to account for this issue.

Learning tasks to perform. The apple collection environment (Figure 7.1d) and MuJoCo skill learning (Section 7.3) demonstrate that it is possible to learn aspects of what should be done in an environment. However, it is not clear that this is *desirable*, since the robot may perform a task that has been inferred, instead of the task that H explicitly sets for it. Just because R has inferred that H wants the apples harvested doesn't mean it should go do so, especially if H has currently asked R to wash the dishes.

It is possible that this would not be a problem in a suitable formulation: as long as R assigns sufficiently high probability to H asking R for the thing that H wants most, R will follow H's instructions, unless it has reason to think that H is incorrect (Milli et al., 2017). (For example, R should probably disobey H if it knows that the house is on fire and knows that H does not know this.)

Nevertheless, we may want to prevent R from learning to autonomously pursue tasks based solely on the state of the world. Ideally, we would be able to decompose the inferred reward into the *frame conditions* θ_{frame}, which specify what R should not do, and the *task reward* θ_{task}, which specify things that could be good for R to do. Given such a decomposition, R could ensure that it takes θ_{frame} into account in its decision-making process, while ignoring the implications of θ_{task}. One way to do this is to look across a variety of environments, or a variety of humans, for commonalities in the reward: it seems likely that these would correspond to frame conditions.

Practical improvements

Other sources of empirical knowledge. Learning from the state of the world can be thought of as a solution to the "is-ought problem" (Hume, 2003) for artificial agents: it allows an agent to learn what it *ought* to do, given only knowledge about what *is* true. (Readers familiar with Hume may wonder what assumption allows us to bridge the gap from "is" facts to "ought" facts. In this case, the bridging assumption is that R ought to do that which H has put effort into (and what H has put effort into is an "is" fact.)

However, our algorithms so far either require us to have a full, complete understanding of empirical facts as with RLSP (Algorithm 2), or they use the barest of empirical knowledge, that which can be gleaned from random rollouts, as in Deep RLSP (Algorithm 3). It seems clear that we can do better, by leveraging better algorithms and data sources for learning empirical facts about the world, before trying to learn from the state of the world. For example, how might we leverage the common-sense knowledge of large language models (Radford et al., 2019; Brown et al., 2020)?

Human models. In this dissertation we used the simple model of Boltzmann rationality as our model of H. While this worked well in our simple conceptual environments, it is not clear that this will scale well to larger environments. Recent work in inverse reinforcement learning has considered accounting for human biases (Evans et al., 2016; Majumdar et al., 2017; Shah et al., 2019b); similar approaches may be needed for learning from the state of the world.

Handling partial observability. Our formalism for learning from the state of the world (Chapter 3) assumes that H is acting in an MDP without reward $\mathcal{M}\backslash\mathcal{R}$. However, most realistic environments are partially observable; neither H nor R can observe the entire world state at every timestep. While we have generalized to partial observability using the assistance formulation, assistance problems are quite computationally challenging to solve. Can we generalize our original formalism to handle partial observability, without requiring the increase in computational complexity of assistance?

It is relatively easy to handle partial observability for H: the only difference is that our human model π^H must now work off of observations, rather than states. However, it is more challenging to handle partial observability for R: what exactly does R get to observe? It should no longer be able to observe the full state of the world s_0, but it also is not sufficient to just observe o_0, because this may only be a small part of the world. For example, if R can't see an unbroken vase in o_0, but does see after two actions in o_2, then it should still learn from the state of the world that vases probably should not be broken. The crucial difference is that when observing a full state s_0, the posterior $P(\theta \mid s_0)$ captures *all* of the information we can gain from simulating the past, due to the Markov property of states. In contrast, $P(\theta \mid o_0)$ does not capture all of this information, and future observations $o_1, o_2, \ldots$ can provide additional insights.

This is handled naturally by the assistance formulation, in which R's optimal policy will seek out new observations in future timesteps that can then be leveraged to learn from the state of the world (as long as this information is sufficiently valuable). Are there good heuristics that allow us to capture this behavior, without requiring us to solve a full assistance problem?

Integrating RLSP with assistance. While we formalized learning from the state of the world using the assistance framework, even with full observability, the optimal policy for an assistance problem typically requires the full posterior $P(\theta \mid s_0)$, whereas both RLSP and Deep RLSP compute a point estimate of θ. How can we use these algorithms (or analogs thereof) in order to more efficiently solve the assistance formulation of learning from the state of the world?

Learned models. The gradient in Deep RLSP depends on having access to a good model of π^H, $(\pi^H)^{-1}$, and $\mathcal{T}^{-1}$, as well as a good learned feature function f. We used relatively simple multilayer perceptrons (MLPs) and variational autoencoders (VAEs) to learn these models, as we found them to be the most stable and easiest to tune. However, progress in these areas is rapid, especially in self-supervised representation learning. It seems likely that applying well-tuned state-of-the-art techniques will allow for significantly improved results, and could be a good approach to few-shot imitation learning for robotics.

9.2 Applications

Our fundamental insight is that the state of the world can inform us about human preferences by showing us what humans put effort into. While we formalized this in the paradigm of sequential decision-making, and the ideas in the previous section continue to work within this paradigm, the general idea is broader and can be applied in many domains. We provide a few examples as illustration, but it seems likely that there are many more applications that have not yet been thought of.

Satellite imagery. Satellite imagery is a rich source of data about the state of the literal world. It is already being used for many purposes, such as predicting how wealthy various regions are in order to better target policy and aid (Abelson et al., 2014; Yeh et al., 2020) and predicting crop yields (Pantazi et al., 2016). An intelligent AI system with enough background knowledge of the world should be able to infer a lot from such a dataset. For example, if the AI system already knows the wealth of a region, then the prevalence of schools in that region may be informative of how much the people of that region value education.

Social media. A person's social media feed can be a rich source of information about their preferences. Social media companies can learn a lot about an individual user through their pattern of clicks and views on a variety of posts. However, even without access to the internal datasets, we can learn a lot about both individual people and various groups by looking at social media feeds. We can model a social feed as joint optimization by the user and the recommendation algorithm on the set of posts shown to the user. Thus, by observing the set of posts that are currently being shown, we can infer what the user must care about reading. (This does require having a sufficiently good model of the workings of the recommendation algorithm, which may be challenging.)

Learning from prices. By using the formalism of sequential decision-making, we have implicitly made the assumption that time is the unit of effort. At each timestep, H has a variety of options available to her; the fact that she chooses one option over many others thus provides significant information about what H cares about.

However, we could also view *money* as the unit of effort: everyone has a limited supply of it, and can use it to pursue many different goals, and so the choice of what H does with her money is very informative about her preferences.

We can even think of the market prices and trading volumes of various items as informative about the "preferences" of humanity as whole. For example, despite being more expensive, vegan meat substitutes are growing in popularity; from this an agent might deduce that the preference against meat-eating is rising.

It should be noted that prices would be thought of as a *lower bound* on value, because prices are a function both of how valuable an item is as well as how easy it is to produce it. Just because clean water is cheap doesn't mean it is not important to human preferences.

9.3 Closing thoughts

AI systems are poised to have a massive impact on human civilization within the next century. Researchers today have a unique opportunity to shape this impact to ensure that it is beneficial for humanity. By showing how an AI system can learn what it *should* do by reasoning about the provenance of the state of the world, this dissertation takes a step towards AI systems that do what we intend them to do. With further progress towards such AI systems, we can create a world in which AI systems assist us in achieving our goals, whether they be as mundane as washing laundry or as grand as colonizing the stars.

Bibliography

Brian Abelson, Kush R Varshney, and Joy Sun. Targeting direct cash transfers to the extremely poor. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1563–1572, 2014.

Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

Maurice Allais. The so-called allais paradox and rational decisions under uncertainty. In *Expected utility hypotheses and the Allais paradox*, pages 437–681. Springer, 1979.

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

Oana Fabiana Andreescu. *Static analysis of functional programs with an application to the frame problem in deductive verification.* PhD thesis, Rennes 1, 2017.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30:5048–5058, 2017.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

Stuart Armstrong and Benjamin Levinstein. Low impact artificial intelligences. *arXiv preprint arXiv:1705.10720*, 2017.

Stuart Armstrong and Sören Mindermann. Occam's razor is insufficient to infer the preferences of irrational agents. In *Advances in Neural Information Processing Systems*, pages 5598–5609, 2018.

Stuart Armstrong, Jan Leike, Laurent Orseau, and Shane Legg. Pitfalls of learning a reward function online. *arXiv preprint arXiv:2004.13654*, 2020.

Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette. Learning to understand goal specifications by modelling reward. In *International Conference on Learning Representations (ICLR)*, 2019.

Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15, Intelligent Agents [St. Catherine's College, Oxford, July 1995]*, pages 103–129, Oxford, UK, UK, 1999. Oxford University. ISBN 0-19-853867-7. URL `http://dl.acm.org/citation.cfm?id=647636.733043`.

Andrea Bajcsy, Dylan P Losey, Marcia K O'Malley, and Anca D Dragan. Learning robot objectives from physical human interaction. *Proceedings of Machine Learning Research*, 78: 217–226, 2017.

Tirthankar Bandyopadhyay, Kok Sung Won, Emilio Frazzoli, David Hsu, Wee Sun Lee, and Daniela Rus. Intention-aware motion planning. In *Algorithmic Foundations of Robotics X*, pages 475–491. Springer, 2013.

Christopher M Bishop. Mixture density networks. Neural Computing Research Group Report, Aston University, 1994.

Erdem Bıyık, Malayandi Palan, Nicholas C Landolfi, Dylan P Losey, and Dorsa Sadigh. Asking easy questions: A user-friendly approach to active reward learning. *arXiv preprint arXiv:1910.04365*, 2019.

Andreea Bobu, Dexter RR Scobee, Jaime F Fisac, S Shankar Sastry, and Anca D Dragan. Less is more: Rethinking probabilistic models of human behavior. In *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, pages 429–437, 2020.

Nick Bostrom. *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press, Inc., USA, 2014.

David D Bourgin, Joshua C Peterson, Daniel Reichman, Stuart J Russell, and Thomas L Griffiths. Cognitive model priors for predicting human decisions. In *International conference on machine learning*, pages 5133–5141, 2019.

Craig Boutilier, Ronen I Brafman, Carmel Domshlak, Holger H Hoos, and David Poole. Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of artificial intelligence research*, 21:135–191, 2004.

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

Connor Brooks and Daniel Szafir. Balanced information gathering and goal-oriented actions in shared autonomy. In *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 85–94, 2019.

Daniel S Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. *arXiv preprint arXiv:1904.06387*, 2019.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Lars Buesing, Theophane Weber, Sébastien Racaniere, SM Eslami, Danilo Rezende, David P Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, et al. Learning and querying fast generative models for reinforcement learning. *arXiv preprint arXiv:1802.03006*, 2018.

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

Ryan Carey. Incorrigibility in the CIRL framework. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pages 30–35, 2018.

Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. In *Advances in Neural Information Processing Systems*, pages 5174–5185, 2019.

Lawrence Chan, Dylan Hadfield-Menell, Siddhartha Srinivasa, and Anca Dragan. The assistive multi-armed bandit. In *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 354–363. IEEE, 2019.

Min Chen, Stefanos Nikolaidis, Harold Soh, David Hsu, and Siddhartha Srinivasa. Planning with trust for human-robot collaboration. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, pages 307–315, 2018.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of International Conference on Machine Learning (ICML)*, 2020.

Rohan Choudhury, Gokul Swamy, Dylan Hadfield-Menell, and Anca D Dragan. On the utility of model learning in hri. In *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 317–325. IEEE, 2019.

Alexandra Chouldechova. Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big Data*, 5(2):153–163, 2017.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. Back to basics: Benchmarking canonical evolution strategies for playing atari. *arXiv preprint arXiv:1802.08842*, 2018.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307, 2017.

Jack Clark and Dario Amodei. Faulty reward functions in the wild, 2016. URL `https://blog.openai.com/faulty-reward-functions`.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations (ICLR)*, 2020.

Cohn, Robert W. *Maximizing Expected Value of Information in Decision Problems by Querying on a Wish-to-Know Basis.* PhD thesis, University of Michigan, 2016.

Sam Corbett-Davies, Emma Pierson, Avi Feller, Sharad Goel, and Aziz Huq. Algorithmic decision making and the cost of fairness. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 797–806, 2017.

Richard M Cyert and Morris H DeGroot. Adaptive utility. In *Adaptive Economic Models*, pages 223–246. Elsevier, 1975.

Christian Daniel, Malte Viering, Jan Metz, Oliver Kroemer, and Jan Peters. Active reward learning. In *Robotics: Science and systems*, 2014.

Nishant Desai. Uncertain reward-transition mdps for negotiable reinforcement learning. 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019.

Christos Dimitrakakis, David C Parkes, Goran Radanovic, and Paul Tylkin. Multi-view decision processes: the helper-ai problem. In *Advances in Neural Information Processing Systems*, pages 5443–5452, 2017.

Andreas Doerr, Christian Daniel, Martin Schiegg, Duy Nguyen-Tuong, Stefan Schaal, Marc Toussaint, and Sebastian Trimpe. Probabilistic recurrent state-space models. In *Proceedings of International Conference on Machine Learning (ICML)*, 2018.

Michael O Duff. *Optimal learning: Computational procedures for Bayes-adaptive Markov decision processes.* PhD thesis, University of Massachusetts Amherst, 2002.

Ashley D Edwards, Himanshu Sahni, Yannick Schroeker, and Charles L Isbell. Imitating latent policies from observation. *arXiv preprint arXiv:1805.07914*, 2018.

Brochu Eric, Nando D Freitas, and Abhijeet Ghosh. Active preference learning with discrete choice data. In *Advances in Neural Information Processing Systems*, pages 409–416, 2008.

Owain Evans, Andreas Stuhlmüller, and Noah Goodman. Learning the preferences of ignorant, inconsistent agents. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Alan Fern, Sriraam Natarajan, Kshitij Judah, and Prasad Tadepalli. A decision-theoretic model of assistance. *Journal of Artificial Intelligence Research*, 50:71–104, 2014.

Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

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

Justin Fu, Anoop Korattikara, Sergey Levine, and Sergio Guadarrama. From language to goals: Inverse reinforcement learning for vision-based instruction following. *arXiv preprint arXiv:1902.07742*, 2019.

Sunil Gandhi, Tim Oates, Tinoosh Mohsenin, and Nicholas Waytowich. Learning from observations using a single video demonstration and human feedback. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2019.

Yang Gao, Jan Peters, Antonios Tsourdos, Shao Zhifei, and Er Meng Joo. A survey of inverse reinforcement learning techniques. *International Journal of Intelligent Computing and Cybernetics*, 2012.

Noah D Goodman and Michael C Frank. Pragmatic language interpretation as probabilistic inference. *Trends in Cognitive Sciences*, 20(11):818–829, 2016.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of International Conference on Machine Learning (ICML)*, 2018.

Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. Cooperative Inverse Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 3909–3917, 2016.

Dylan Hadfield-Menell, Anca Dragan, Pieter Abbeel, and Stuart Russell. The off-switch game. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017a.

Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan. Inverse reward design. In *Advances in Neural Information Processing Systems*, pages 6765–6774, 2017b.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *Proceedings of International Conference on Machine Learning (ICML)*, 2019.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations (ICLR)*, 2020.

Joseph Y Halpern and Rafael Pass. Game theory with translucent players. *International Journal of Game Theory*, 47(3):949–976, 2018.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Christopher Hesse, John Schulman, Vicki Pfau, Alex Nichol, Oleg Klimov, and Larissa Schiavo. Retro contest, 2018. `https://openai.com/blog/retro-contest/`.

Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, 2016.

Mark K Ho, Michael Littman, James MacGlashan, Fiery Cushman, and Joseph L Austerweil. Showing versus doing: Teaching by demonstration. In *Advances in Neural Information Processing Systems*, pages 3027–3035, 2016.

David Hume. *A treatise of human nature.* Courier Corporation, 2003.

Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in atari. In *Advances in Neural Information Processing Systems*, 2018.

Alexis Jacq, Matthieu Geist, Ana Paiva, and Olivier Pietquin. Learning from a learner. In *International Conference on Machine Learning*, pages 2990–2999, 2019.

Shervin Javdani, Siddhartha S Srinivasa, and J Andrew Bagnell. Shared autonomy via hindsight optimization. *Robotics Science and Systems: online proceedings*, 2015, 2015.

Hong Jun Jeon, Smitha Milli, and Anca D Dragan. Reward-rational (implicit) choice: A unifying formalism for reward learning. *arXiv preprint arXiv:2002.04833*, 2020.

Daniel Kahneman. *Thinking, fast and slow.* Macmillan, 2011.

Ece Kamar, Ya'akov Gal, and Barbara J Grosz. Incorporating helpful behavior into collaborative planning. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Springer Verlag, 2009.

Antti Kangasrääsiö and Samuel Kaski. Inverse reinforcement learning from summary data. *Machine Learning*, 107(8-10):1517–1535, 2018.

Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick Van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. In *International Conference on Learning Representations (ICLR)*, 2017.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

Jon Kleinberg, Sendhil Mullainathan, and Manish Raghavan. Inherent trade-offs in the fair determination of risk scores. *arXiv preprint arXiv:1609.05807*, 2016.

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. ACM, 2009.

Victoria Krakovna. Specification gaming examples in AI, 2018. URL `https://vkrakovna.wordpress.com/2018/04/02/specification-gaming-examples-in-ai/`.

Victoria Krakovna, Laurent Orseau, Ramana Kumar, Miljan Martic, and Shane Legg. Penalizing side effects using stepwise relative reachability. *arXiv preprint arXiv:1806.01186*, 2018.

Thanard Kurutach, Aviv Tamar, Ge Yang, Stuart J Russell, and Pieter Abbeel. Learning plannable representations with causal infogan. In *Advances in Neural Information Processing Systems*, 2018.

Kuang-Huei Lee, Ian Fischer, Anthony Liu, Yijie Guo, Honglak Lee, John Canny, and Sergio Guadarrama. Predictive information accelerates learning in rl. *Advances in Neural Information Processing Systems*, 33, 2020.

Joel Lehman, Jeff Clune, and Dusan Misevic. The surprising creativity of digital evolution. In *Artificial Life Conference Proceedings*, pages 55–56. MIT Press, 2018.

Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. Scalable agent alignment via reward modeling: a research direction. *arXiv preprint arXiv:1811.07871*, 2018.

David A Levin and Yuval Peres. *Markov chains and mixing times*, volume 107. American Mathematical Soc., 2017.

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

Manuel Lopes, Francisco Melo, and Luis Montesano. Active learning for reward estimation in inverse reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 31–46. Springer, 2009.

James MacGlashan, Mark K Ho, Robert Loftin, Bei Peng, David Roberts, Matthew E Taylor, and Michael L Littman. Interactive learning from policy-dependent human feedback. *arXiv preprint arXiv:1701.06049*, 2017.

Owen Macindoe, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. POMCoP: Belief space planning for sidekicks in cooperative games. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

Anirudha Majumdar, Sumeet Singh, Ajay Mandlekar, and Marco Pavone. Risk-sensitive inverse reinforcement learning via coherent risk models. In *Robotics: Science and Systems*, 2017.

Dhruv Malik, Malayandi Palaniappan, Jaime F Fisac, Dylan Hadfield-Menell, Stuart Russell, and Anca D Dragan. An efficient, generalized Bellman update for cooperative inverse reinforcement learning. *arXiv preprint arXiv:1806.03820*, 2018.

James John Martin. *Bayesian decision problems and Markov chains*. Wiley, 1967.

Lucas Maystre and Matthias Grossglauser. Just sort it! A simple and effective approach to active preference learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2344–2353, 2017.

John McCarthy and Patrick J Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Readings in Artificial Intelligence*, pages 431–450. Elsevier, 1981.

Bernard Michini and Jonathan P How. Bayesian nonparametric inverse reinforcement learning. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 148–163. Springer, 2012.

Smitha Milli, Dylan Hadfield-Menell, Anca Dragan, and Stuart Russell. Should robots be obedient? *arXiv preprint arXiv:1705.09990*, 2017.

Sören Mindermann, Rohin Shah, Adam Gleave, and Dylan Hadfield-Menell. Active inverse reward design. *arXiv preprint arXiv:1809.03060*, 2018.

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

Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, pages 9191–9200, 2018.

Andrew Y Ng and Stuart J Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine learning*, 2000.

Stefanos Nikolaidis and Julie Shah. Human-robot cross-training: computational formulation, modeling and evaluation of a human team training strategy. In *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 33–40. IEEE, 2013.

Stefanos Nikolaidis, Ramya Ramakrishnan, Keren Gu, and Julie Shah. Efficient model learning from joint-action demonstrations for human-robot collaborative tasks. In *2015 10th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 189–196. IEEE, 2015.

Stefanos Nikolaidis, David Hsu, and Siddhartha Srinivasa. Human-robot mutual adaptation in collaborative tasks: Models and experiments. *The International Journal of Robotics Research*, 36(5-7):618–634, 2017.

Stephen M Omohundro. The basic AI drives. In *Artificial General Intelligence*, pages 483–492, 2008.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

OpenAI. OpenAI Five, 2018. `https://openai.com/blog/openai-five/`.

Xanthoula Eirini Pantazi, Dimitrios Moshou, Thomas Alexandridis, Rebecca L Whetton, and Abdul Mounem Mouazen. Wheat yield prediction using machine learning and advanced sensing techniques. *Computers and Electronics in Agriculture*, 121:57–65, 2016.

Daehyung Park, Michael Noseworthy, Rohan Paul, Subhro Roy, and Nicholas Roy. Inferring task goals and constraints using bayesian nonparametric inverse reinforcement learning. In *Conference on Robot Learning*, pages 1005–1014, 2020.

Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, pages 1025–1032, 2003.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *IJCAI*, pages 2586–2591, 2007.

Stephanie Rosenthal and Manuela Veloso. Modeling humans as observation providers using pomdps. In *2011 RO-MAN*, pages 53–58. IEEE, 2011.

Stuart Russell. Of myths and moonshine, 2014. URL `https://www.edge.org/conversation/the-myth-of-ai#26015`.

Stuart Russell. *Human Compatible: Artificial Intelligence and the Problem of Control*. Penguin, 2019.

Dorsa Sadigh, S Shankar Sastry, Sanjit A Seshia, and Anca Dragan. Information gathering actions over human internal state. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 66–73. IEEE, 2016.

Dorsa Sadigh, Anca D Dragan, Shankar Sastry, and Sanjit A Seshia. Active preference-based learning of reward functions. In *Robotics: Science and Systems*, 2017.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pages 1312–1320, 2015.

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

Rohin Shah, Noah Gundotra, Pieter Abbeel, and Anca D Dragan. On the feasibility of learning, rather than assuming, human biases for reward inference. *arXiv preprint arXiv:1906.09624*, 2019a.

Rohin Shah, Dmitrii Krasheninnikov, Jordan Alexander, Pieter Abbeel, and Anca Dragan. Preferences implicit in the state of the world. In *International Conference on Learning Representations*, 2019b.

Archit Sharma, Shane Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised skill discovery. In *International Conference on Learning Representations (ICLR)*, 2020.

Jason F Shogren, John A List, and Dermot J Hayes. Preference learning in consecutive experimental auctions. *American Journal of Agricultural Economics*, 82(4):1016–1021, 2000.

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.

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback. *arXiv preprint arXiv:2009.01325*, 2020.

Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. *arXiv preprint arXiv:2009.08319*, 2020.

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

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

Faraz Torabi, Garrett Warnell, and Peter Stone. Generative adversarial imitation from observation. *arXiv preprint arXiv:1807.06158*, 2018.

Alexander Matt Turner. Optimal farsighted agents tend to seek power. *arXiv preprint arXiv:1912.01683*, 2019.

Alexander Matt Turner, Dylan Hadfield-Menell, and Prasad Tadepalli. Conservative agency via attainable utility preservation. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 385–391, 2020.

Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M. Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Yuhuai Wu, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. Alphastar: Mastering the real-time strategy game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019.

Steven Wang, Sam Toyer, Adam Gleave, and Scott Emmons. The imitation library for imitation learning and inverse reinforcement learning. `https://github.com/HumanCompatibleAI/imitation`, 2020.

Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. Deep tamer: Interactive agent shaping in high-dimensional state spaces. *arXiv preprint arXiv:1709.10163*, 2017.

David Whitney, Eric Rosen, James MacGlashan, Lawson LS Wong, and Stefanie Tellex. Reducing errors in object-fetching interactions through social feedback. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1006–1013. IEEE, 2017.

Nils Wilde, Dana Kulic, and Stephen L Smith. Active preference learning using maximum regret. *arXiv preprint arXiv:2005.04067*, 2020.

Christian Wirth, Riad Akrour, Gerhard Neumann, and Johannes Fürnkranz. A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, 18(1):4945–4990, 2017.

Mark Woodward, Chelsea Finn, and Karol Hausman. Learning to interactively learn and assist. *arXiv preprint arXiv:1906.10187*, 2019.

Christopher Yeh, Anthony Perez, Anne Driscoll, George Azzari, Zhongyi Tang, David Lobell, Stefano Ermon, and Marshall Burke. Using publicly available satellite imagery and deep learning to understand economic well-being in africa. *Nature communications*, 11(1):1–11, 2020.

Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. *arXiv preprint arXiv:1802.01557*, 2018.

Eliezer Yudkowsky. Problem of fully updated deference, year unknown. URL `https://arbital.com/p/updated_deference/`.

Shun Zhang, Edmund Durfee, and Satinder Singh. Approximately-optimal queries for planning in reward-uncertain markov decision processes. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 2017.

Brian D Ziebart, J Andrew Bagnell, and Anind K Dey. Modeling interaction via the principle of maximum causal entropy. 2010.