MAXIMUM LIKELIHOOD INVERSE REINFORCEMENT LEARNING

by

MONICA C. VROMAN

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Michael L. Littman

And approved by

New Brunswick, New Jersey

OCTOBER, 2014

ABSTRACT OF THE DISSERTATION

MAXIMUM LIKELIHOOD INVERSE REINFORCEMENT LEARNING

by MONICA C. VROMAN

Dissertation Director:

Michael L. Littman

Learning desirable behavior from a limited number of demonstrations, also known as inverse reinforcement learning, is a challenging task in machine learning. I apply maximum likelihood estimation to the problem of inverse reinforcement learning, and show that it quickly and successfully identifies the unknown reward function from traces of optimal or near-optimal behavior, under the assumption that the reward function is a linear function of a known set of features. I extend this approach to cover reward functions that are a generalized function of the features, and show that the generalized inverse reinforcement learning approach is a competitive alternative to existing approaches covering the same class of functions, while at the same time, being able to learn the right rewards in cases that have not been covered before.

I then apply these tools to the problem of learning from (unlabeled) demonstration trajectories of behavior generated by varying “intentions” or objectives. I derive an EM approach that clusters observed trajectories by inferring the objectives for each cluster using any of several possible IRL methods, and then uses the constructed clusters to quickly identify the intent of a trajectory.

I present an application of maximum likelihood inverse reinforcement learning to the problem of training an artificial agent to follow verbal instructions representing high-level tasks using a set of instructions paired with demonstration traces of appropriate behavior.

Preface

Portions of this dissertation are based on work previously published by the author in ”Apprenticeship Learning about Multiple Intentions” by Monica Babeş-Vroman, Vukosi Marivate, Kaushik Subramanian and Michael Littman (ICML 2011) and ”Learning to Interpret Natural Language Instructions” by Monica Babeş-Vroman, James MacGlashan, Ruoyuan Gao, Kevin Winner, Richard Adjogah, Marie DesJardins, Michael Littman and Smaranda Muresan (NACL 2012).

Acknowledgements

There are many people who were instrumental in the completion of the research that went into this thesis.

My husband, Dave, provided the support and companionship I needed through the sometimes lonely and difficult times in graduate school. He listened when I needed to process things and provided some ideas and feedback. He proofread parts of my papers and helped me phrase things better. Most of all, he was loving and patient with me through the ups and downs of my research process. I couldn't have asked for a better companion and best friend.

My son, Jay, forced me to take valuable breaks, not always appreciated, but certainly needed. He made the last months of work on my thesis more fun and enjoyable. He taught me to be patient and laid back and that he is not a deterministic machine (like the computers I work with) who always has the same behavior (or output) when I do the same thing (or provide the same input). He is unpredictable and lovely.

My mom, Ina, traveled from Romania to spend time with us and give me more time for research by taking care of my son. My mother-in-law, Diana, also spent many days taking care of him so I would have more time to work. They are some of the most generous people I have ever met and I hope I can learn from the way they selflessly and joyfully helped me. Their time and support were extremely valuable in the process of finishing my research.

My sister Laura and my sister-in-law Janna are my best friends. It warms my heart every time I talk to them, which is not often, but it is always a delight. They are sweet and honest, and their genuine love and care means the world to me.

My friends encouraged and supported me, and even provided feed-back when we designed experiments with human subjects. It was very entertaining to read the transcripts of spouses giving each other instructions on how to play Sokoban over instant messenger. Their thoughts and prayers were instrumental in getting my work done. Annette gave me a ride to and from campus every time I needed it, Linh bought groceries for me and kept my son company so I could take a break, Francis and Rebecca took me out in the evenings and provided great company and conversations, Liz and I went through graduate school together and encouraged each other to keep going, Natalie came over to keep me company and cheer me up, Carroll made me laugh, Ruth took me out for coffee every month and challenged and encouraged me to see things outside of the box. I am blessed with great friends.

My collaborators, Kaushik Subramanian, Vukosi Marivate, James MacGlashan, Julie Gao, Kevin Winner, Richard Adjogah, Marie DesJardins, Samaranda Muresan, Mike Wunder, and Enrique Munoz de Cote taught me so much about team work and inspired me to be passionate about my research.

My (former) labmates took the time to attend my practice talks and provided valuable feed-back in terms of my research and the final presentation slides, especially Tom Walsh who helped me gain some intuition about Gaussian processes and helped me interpret some of my results. I also appreciated the feed-back from John Asmuth, Sergiu Goschin, Vukosi Marivate, Ali Nouri and Ari Weinstein.

My committee members, Tina Eliassi-Rad, Alex Borgida, and Brian Ziebart provided valuable feed-back, leading to important changes and additions to this document. My thesis is more solid as a result of their comments and challenges.

My adviser, Michael, was my mentor and friend. He knew exactly how to encourage and motivate me. His passion for research and teaching were contagious. He taught me how important it is to care, not just about my work, but also about the people I work with, and to make a real difference for the better in my workplace. I am privileged to have had such a great role model.

Lastly, none of this work could have happened without the strength and grace that God provided me with. The verse ”I can do all things through Him who strengthens me.” (Philippians 4:13) became more real to me over the last year of my research. I owe Him everything I have, including the intellect I needed to complete this research, the discipline and patience to do it, and the wonderful people He surrounded me with (mentioned above) who supported and encouraged me. ”From Him and through Him and to Him are all things. To Him be the glory forever. Amen.” (Romans 11:36).

Dedication

To Dave, my favorite person in the whole world.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

When an automatic door opens to let us in, the device implicitly assumes we want to go into the store, based on our proximity to the door. It is reasonable to assume that when we are close to a door, our intention is to use it. When we're driving, we signal our intention to make a turn, to help the other drivers safely navigate around us. Some websites monitor our activity to decide which ads to display in order to maximize our exposure to products and services we are likely to buy.

Say we need to declutter and clean our garage, and we would like to program a robot to learn how to do the job. One solution is to design an agent motivated by a reinforcement signal. Each time this agent takes an action leading to progress in accomplishing the job, for example, throwing away trash, we give it a positive reinforcement, encouraging it to take the same or a similar action in the future. However, in many situations, it is difficult and expensive to specify the reward manually. (Imagine specifying the reward amount in every possible situation!)

As an alternative, we could program the agent to learn from demonstrations. Returning to the task of cleaning, we trade off different factors, like trying to minimize the amount of time and effort spent on the task, or minimizing the amount of dust we swallow, maximizing the surface we clean in the amount of time we have, or trying to clean each spot as thoroughly as we can. To describe a reward function in the cleaning task, we would have to decide exactly what factors matter in getting the job done, and describe exactly how each of the factors contribute to the reward. We refer to these factors as *environment state features* or just *features*.

A more efficient solution could be to observe the behavior of an expert in the cleaning task, and infer their reward function from demonstrations. One of the central assumptions in this inverse reinforcement learning problem is that the expert's behavior expresses what it means to perform a task well.

Inverse reinforcement learning [Russell, 1998], or IRL, tackles the problem of learning the motivation of an agent given examples of its behavior in a known environment. Motivations are conceptualized as reward functions in the Markov decision process formalism, which maps states (or states and actions) to scalar values whose expected discounted sum the agent seeks to maximize. Because of the assumption that the agent is successfully maximizing reward, we refer to it as the *expert*.

Apprenticeship learning [Abbeel and Ng, 2004], or AL, addresses the task of learning a policy from expert demonstrations, but the expert's reward function is unknown to the apprentice. From the demonstrations, the apprentice strives to derive a policy that performs well with respect to the expert's private reward function.

A basic assumption in IRL and AL is that the expert's intent can be expressed as a reward function that is a combination of a known set of features.

The statement of this thesis is the following:

> By casting inverse reinforcement learning as a maximum likelihood problem, we can provide a unified approach to linear inverse reinforcement learning, non-linear inverse reinforcement learning, and multiple intentions inverse reinforcement learning leading to effective and natural solutions.

I start by providing the background and defining some of the terms I will use in the dissertation in Chapter 2. In Chapter 3, I present a survey of existing Inverse Reinforcement Learning approaches. The maximum likelihood linear inverse reinforcement learning algorithm is described in Chapter 4. I relax the assumption of a linear reward function and present a modular inverse reinforcement learning algorithm that adds a supervised learning component to the inverse reinforcement learning setup in Chapter 5. Learning from multiple experts with varying intentions is explored in Chapter 6. I show an application of the maximum likelihood IRL algorithm in Chapter 7 and conclude in Chapter 8. A bibliography containing all cited references is included at the end of the dissertation.

Chapter 2
Background and Definitions

In this chapter, I define inverse reinforcement learning (IRL) and the closely related problem of apprenticeship learning (AL). I start by describing the reinforcement-learning setting (Section 2.1), with terminology that I use again when describing AL and IRL. The AL and IRL settings are described in Section 2.2, and maximum likelihood estimation is defined in Section 2.3.

2.1 The Reinforcement-Learning Setting

I start by describing the reinforcement-learning setting, and use as an example the task of automatically driving a car. In reinforcement learning, there is an *agent* (which is also called the *learner* or *apprentice*), for example the driver, acting in a known *environment*, for example driving a car on a road. Using the Markov decision process formalism [Puterman, 1994], the environment is characterized by a set of variables (S, A, r, T, γ): *states* S, *actions* A, *rewards* r, *transition probabilities* T, and *discount factor* γ.

A *state* is a sufficient statistic for transitions and rewards. In the driving task, it can include information like be the speed of the car, the amount of gas left in the tank, the lane where the car is driving, and the speeds and locations of other cars on the same road that are up to a certain distance away from the car that the agent is driving.

A is the set of actions the agent can perform in the environment. The set of actions for the agent driver could be: changing the speed of the car, using the blinker, going left, going right, etc. Each time the agent takes an action, the environment provides a new state, describing how the environment has changed as a result of the agent's action, and a reinforcement signal, also called *reward* (see Figure 2.1). Rewards can be either positive or negative, though a negative reward is sometimes referred to as a *cost*. For example, the driver might pay a cost for inefficient driving (high gas mileage), but obtain a high reward when arriving to the destination within a certain time.

The *transitions* (or *transition dynamics*) describe how the environment's states change as the agent is taking actions. In some environments, when an agent is taking the same action in a certain state, it always leads to the same new state. In this case, the transitions are called *deterministic*. For example, if the road is empty, the agent is driving in nice weather with no wind, and the car is working properly, if the driver steers left to change the lane, the car will end up in the next lane to the left. However, in certain environments, the driving conditions are such that taking the same action can lead to multiple outcomes, each outcome with a different probability. For example, if the road is slippery or there is a very strong wind, the agent taking the *left* action might end up in the lane it is trying to reach with high probability, but there may also be a small chance of not being able to steer and to end up going straight. Such transitions are called *stochastic*.

The *transition probability matrix* T encodes the environment's transition dynamics. Each matrix entry $T(s, a)$ is a probability distribution over next states s' when action a is taken in state s. We can also write $T : S \times A \times S \rightarrow [0, 1]$.

The discount factor γ weights the outcome of future actions versus present

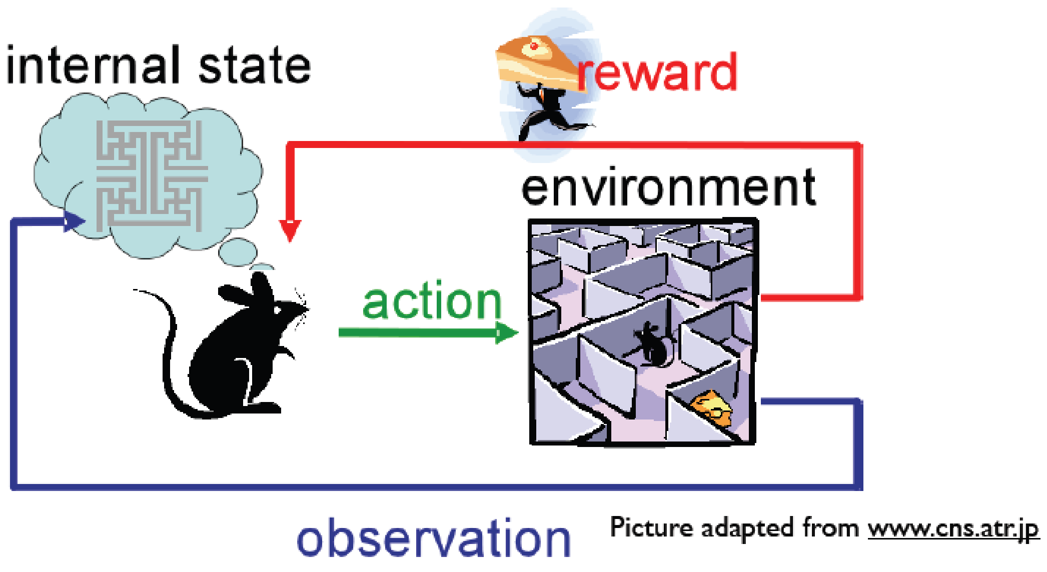

Figure 2.1: The Reinforcement-Learning Setting. The mouse (**agent**) is moving around (**actions**) looking for food (the **reward**) in a maze (the **environment**).

actions. Receiving \$100 next month may not be worth to the agent as much as receiving \$100 tomorrow. In our settings, the discount factor is $\gamma \in [0, 1)$, with lower numbers decreasing the value of future rewards more. If future actions are worth as much as present actions, then $\gamma = 1$.

The RL problem is illustrated in Figure 2.1. The RL agent is drawn as a mouse, acting in an environment shown as a maze and seeking to maximize its reward, here shown as food. This picture is meant to illustrate the exchange of information between the agent and the environment: the information that the agent sends to the environment is its action, and the information that the environment sends in response is the next state, and the reward or cost is the immediate value for taking that action.

When an MDP (S, A, r, T, γ) is given, one can use the value-iteration algorithm [Puterman, 1994] to compute the corresponding action-value function $Q : S \times A \to \mathbb{R}$ and the optimal policy. The action-value function for a state

s and an action a, $Q(s,a)$, provides an estimate of the total discounted reward when taking action a in state s, and following the optimal policy defined by reward function r thereafter. Using the optimal policy and an initial state distribution $p_s : S \rightarrow [0,1]$ (with $\Sigma_s p_s = 1$), it is straightforward to generate trajectories starting in states drawn according to p_s with transitions selected according to the policy. For example, after the starting state is drawn from p, we could choose the greedy action at each step in state s (that is, the action a' that maximizes $Q(s,a)$), or we could choose actions according to a Boltzmann distribution for action selection:

$$\pi(s,a) = e^{\beta Q(s,a)} / \sum_{a'} e^{\beta Q(s,a')}. \tag{2.1}$$

The advantage of using a Boltzmann distribution for action selection is that it is commonly used as a way of inducing variability in the behavior that is tied to the values of the actions themselves. It is also used as a model for human decision making [Luce, 1959].

In our IRL and AL setups, I will assume the availability of a set of trajectories coming from expert agents taking actions in the MDP in the form $D = \{\xi_1, ..., \xi_N\}$. A trajectory consists of a sequence of state-action pairs $\xi_i = \{(s_1, a_1), \ldots\}$.

2.2 IRL and AL

Algorithms for apprenticeship learning and inverse reinforcement learning take as input a model of the environment and the observed behavior of another agent in the form of a set of *demonstrations*, or *trajectories*. More formally, the model of the environment is a Markov decision process (MDP) without the reward function (MDP\r: (S, A, T, γ)), and the demonstrations are a set

$D = \{\xi_1, \dots \xi_N\}$. Each demonstration ξ_i is a sequence of state–action pairs $\xi_i = \{(s,a)_j^i\}$, where $s \in S$, and $a \in A$. Sometimes, the MDP$\backslash r$ is called a *Markov decision process*, and the MDP with the reward function is sometimes called a *Markov decision problem*. Because of the assumption that the demonstrations come from an agent that is successfully maximizing reward, this agent is referred to as the *expert*. The learner does not have access to the expert's reward function. The state space is sometimes compactly represented by a state-feature function $\Phi = \{\phi_i : S \to \mathbb{R}\}$. Each feature assigns a value to every state. In the car driving domain, the state features could be (for each state) the speed of the car, the amount of gas left in the tank, the lane where the car is driving, and the speeds and locations of other cars on the same road that are up to a certain distance away from the car that the agent is driving.

The goal in AL is to find a policy that performs well with respect to the expert's reward function. The goal in IRL is to find a proxy for the expert's reward function. As is common in earlier work, I focus on IRL as a means to solving AL problems. IRL is finding application in a broad range of problems from inferring people's moral values [Moore et al., 2009] to interpreting verbal instructions [Branavan et al., 2009].

IRL can be seen as a reward-estimation problem. The reward-estimation problem is the following: Given an MDP$\backslash r$ and a set of expert demonstrations $D = \{\xi_1, \dots \xi_N\}$, find a reward function r that makes behavior D optimal in the MDP. This problem has an infinite set of solutions, for example, a reward function $r(s) = c$, $\forall s \in S$ and $c \in \mathbb{R}$, therefore, the reward-estimation problem is ill-posed. A well-posed problem is a problem that admits a solution, and its solution is unique and stable [Hadamard, 1902]. The challenge of IRL algorithms is to find the optimal solution in this infinite set. IRL algorithms differ in the optimization criterion they use, but they all seek a solution that is unique and optimal with respect to their criterion.

In general, reward functions are parameterized by a vector of reward weights θ applied to a feature vector for each state-action pair $\phi(s, a)$. Thus, a reward function is written $r_\theta(s, a) = f_\theta(\phi(s, a))$. If the expert's reward function is given by θ_E, the apprentice's objective is to behave in a way that maximizes the discounted sum of expected future rewards with respect to r_{θ_E}. However, the apprentice does not know θ_E and must use information from the observed trajectories to decide how to behave. It can, for example, hypothesize its own reward weights θ_A and behave accordingly.

2.3 Maximum Likelihood Estimation

As stated in Chapter 1, I cast inverse reinforcement learning as a maximum likelihood estimation problem.

The goal of maximum likelihood estimation (or MLE) is to find a set of parameters $\hat{\theta} \in \Theta$ given a mapping $f : \Theta \to \mathbb{R}$, the observed data D, and the process $M(f)$ that generated the data. Here, θ stand for the true parameters that generated the data. MLE finds parameters

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, P(D|M, f, \theta). \tag{2.2}$$

In some cases, it is easy to compute θ analytically. For example, if f and M are known linear mappings and assuming the data is large enough, finding θ involves solving a system of linear equations. In settings like IRL, $f(\theta)$ is the expert's reward function, and M involves finding the optimal policy in the given MDP$\backslash r$, which is typically done by dynamic programming (for example, using the value-iteration algorithm [Puterman, 1994]). The optimal policy is then used to generate demonstrations in the MDP (see Section 2.1). In this case, it is intractable to compute θ analytically, and we need to estimate $\hat{\theta}$ using Equation 2.2.

In Chapter 4, the expert's reward is assumed to be a linear combination of a known set of state features, therefore f is a linear mapping. I use θ_E to stand for the feature parameters in the expert's reward function. The data is a set of demonstrations
$D = \{(s_1, a_1)^1, \ldots (s_{j_1}, a_{j_1})^1, (s_1, a_1)^2, \ldots (s_{j_2}, a_{j_2})^2, \ldots (s_1, a_1)^N, \ldots (s_{j_N}, a_{j_N})^N)\}$, where N is the number of expert demonstrations, and j_i is the length of the ith demonstration. The expert is assumed to be acting optimally with respect to its reward function, but to allow for slight mistakes or noise in the observation of the expert's behavior, a Boltzmann distribution for action selection is used. The data-generation process M is therefore the following: Given a current estimate $\hat{\theta}$ of θ_E, compute an estimate of the expert's reward, $f(\hat{\theta})$, then compute an estimate of the optimal policy according to $f(\hat{\theta})$ by running value iteration for a fixed number of steps. This calculation gives us an estimate of the Q-function, which can be used to compute an estimate of the expert's policy, $\pi(s, a) = e^{\beta Q(s,a)} / \sum_{a'} e^{\beta Q(s,a')}$. Using Equation 2.2, $\hat{\theta}$ can be computed by using gradient ascent to maximize the probability of the data given the parameters.

The same process is used in Chapter 5, except using a mapping f that is a non-linear function of the state features. In Chapter 6, MLE is applied to the problem of inferring multiple intentions from a set of unlabeled demonstrations.

2.4 Conclusions

In this chapter, I have described the settings and defined some of the terms needed to support my thesis statement. I have described the reinforcement-learning, inverse reinforcement learning and apprenticeship-learning settings and defined maximum likelihood estimation.

In the following chapters, I show that maximum likelihood is a justified optimization criterion for IRL in the single expert linear setting (Chapter 4), in the single expert non-linear setting (Chapter 5), and in the multiple expert setting (Chapter 6), thereby providing a unified approach to IRL problems.

Chapter 3
Survey of Inverse Reinforcement Learning Algorithms

This chapter presents a survey of existing inverse reinforcement learning (IRL) or inverse optimal control (IOC) and apprenticeship learning (AL) algorithms. IRL and AL are examples of imitation learning [Argall et al., 2009], which is the problem of inferring a policy from expert demonstrations. The expert is generally assumed to be acting optimally according to their intention or reward function, although many IRL and AL algorithms include a model of the expert that allows for slight deviations from optimal behavior, or for noise in the observations of the expert's behavior.

As discussed in Chapter 2, the input to an IRL or AL algorithm is a Markov decision process without the reward function, MDP$\backslash R$, a state-feature function $\Phi = \{\phi_i : S \to \mathbb{R}\}$, and a set of expert demonstrations $D = \{\xi_1 \ldots \xi_N\}$, each demonstration represented as a sequence of state-action pairs, $\xi_i = \{(s, a)_j^i\}$. The output is the computed estimate of the expert's reward function R_E for an IRL algorithm, or the computed estimate of the expert's policy π_E for an AL algorithm.

Models for human goal inference and action understanding from sequences of behavior have also been studied [Baker et al., 2007, 2009]. When understanding the actions of another rational agent, humans assume that the agent behaves as optimally as possible to accomplish its goal. The authors use a computational model of inverse probabilistic planning, the process of inferring an agent's "intention, given observations of an agent behavior, by inverting a model of the agent's planing process." The authors use three models of people's prior knowledge about goals, and show that a model assuming that goals can change over time correlates best with human judgments.

3.1 Imitation Learning. Direct methods.

Early approaches to imitation learning used demonstrations from the expert as input to a supervised learning algorithm, and learned the policy as a direct mapping from states to actions. One paper [Sammut et al., 1992] presents an application of inductive learning from human demonstrations in the task of flying an airplane–a task that involves complex motor skills. The actions of the human using the flight simulator are used as input to an induction program. Since each person has a different flying style, the autopilot is trained from each of the human subjects who demonstrate the task. Then, a decision tree is learned that maps between the states of the simulator and the actions that need to be taken in those states. The trees are pruned to obtain the smallest trees that allow the plane to fly correctly. The authors show the learned rules for taking off, attaining flight level, turning right and taking a sharp left turn. The performance is measured by comparing the flight profiles for the human pilot versus the autopilot. The goal is to mimic the trajectory of the human expert.

In another paper [Amit and Matari, 2002], the learner is a robot expected to learn to imitate the movements of the demonstrator. It uses a hierarchical reward structure, where higher levels use the information from the lower levels for policy inference. The lowest level encodes movement primitives, like moving a certain joint, the next level learns particular motions that are frequently demonstrated, and the last layer is able to represent complex movements. During the demonstration, the lowest layer adjusts its parameters to imitate the demonstrated movements as closely as possible. The second layer learns specific movements, composed of primitive joint movements encoded in the first layer. The highest level is implemented using hidden Markov models, where each motion from the lower level is treated as a symbol in a set of symbols for the HMM. Performance is measured by the system's accuracy in recognizing demonstrated motions.

The disadvantage of these methods is that they learn a policy that essentially copies the behavior of the demonstrator, and does not provide generalization in regions of the state space that have not been demonstrated. This means that for guaranteeing optimal performance, the learner has to be given an exhaustive set of demonstrations, and is unable to generalize to situations it has not seen.

Learning the reward function provides more opportunity for this kind of generalization. Maximum likelihood IRL methods (Chapter 4, Chapter 5, and Chapter 6) find a reward function as a combination of state features. Even if the demonstrations do not cover the state space, the algorithms are able to learn which features are important to the demonstrated task, and extend this knowledge to unseen states.

3.2 Atypical Setups

Several IRL and AL approaches do not follow the typical IRL setup. In one approach [Ratliff et al., 2006], instead of using one MDP and a set of multiple demonstrations, the training data consists of multiple training instances. Each training instance is a tuple containing an MDP$\backslash r$, a state-feature function, a demonstrated trajectory, policy or vector of state-action counts, and a loss vector, giving the learner a measure of closeness between policies or behaviors. The goal is to find the parameters of a linear mapping between features and rewards such that for each training instance, the resulting reward is "close" to the reward that generated the demonstrated behavior. To compute the reward parameters, the authors define a set of constraints using the structured large margin criterion [Taskar et al., 2005], which allows only for weight vectors for which the example policies have a higher expected reward than any other policy by a margin that scales with the loss. The parameters can be found by solving a quadratic program, but to make the learning more efficient, the authors use the subgradient method, a generalization of gradient descent. The algorithm iteratively computes the optimal policy and state-action visitation frequencies for each input map using the current estimate of the reward function weights, then computes the subgradient of the cost function, and updates the weights according to the subgradient, with an optional step of projecting the weights according to any additional constraints.

In another IRL framework, instead of having access to demonstrations of optimal behavior, the agent receives as feed-back the result of the comparison between two possible behaviors, in the form of feature expectations [da Silva et al., 2006]. The reward function is assumed to be a linear combination of a known set of features, and the goal is to find the expert's reward function, or, equivalently, the feature parameters of the expert's reward function. As the agent is acting in the environment and learning an estimate of the expert's reward function, it is able to query an evaluation module, giving it two different feature expectation vectors and receiving back the result of their comparison. The reward parameters are computed by solving a linear program, and gradually building a set of linear constraints until a given optimal policy is contained in the convex set with boundaries defined by the constraints.

In a hierarchical AL approach [Kolter et al., 2007], the reward function is decomposed into two levels of control, with the low level reward described as a linear combination of a known set of features, and the high level reward as the average of the low level rewards. At the high level, the expert demonstrates full policies, and at the low level, the expert provides the greedy action in certain states. At the low level, the reward is computed by setting the constraint that the reward for the expert's demonstrated action be much higher than the reward of any other action. The authors describe an optimization problem that provides a solution incorporating the rewards of both the high and low levels in the hierarchy.

The problem of inverse optimal control in linearly solvable MDPs (LMDPs) has also been explored [Dvijotham and Todorov, 2010]. An LMDP is defined by a state space X, a state cost $q(x)$, and passive dynamics $x' \sim p(\cdot|x)$ characterizing the transitions in the absence of control. The controller can impose dynamics $x' \sim \pi(\cdot|x)$ by paying a cost which is the KL divergence between π and p. In LMDPs, there is a direct dependence between the value function $v(x)$, the cost function $q(x)$ and the control law $\pi^*(x)$, so only determining one quantity is necessary to compute all of them. The authors note that v is the most efficient to compute, and they use maximum likelihood to estimate it by computing the log likelihood of v. The authors also show a method for computing the cost function directly, and how a traditional IRL problem can be embedded into an LMDP in order to be solved more efficiently. In the case of continuous state spaces, the IRL method uses time discretization which reduces it to a discrete LMDP. The authors note that algorithms in the LMDP framework are more efficient because they recover the value function, and "recovering values is easier than recovering costs because solving the forward problem is avoided". The disadvantage of this approach over approaches that compute an estimate of the expert's reward function is that the value function transfers poorly to new MDPs with similar feature sets or to variants of the same MDP.

Inverse reinforcement learning has also been studied in partially observable MDPs (POMDPs) [Choi and Kim, 2009]. The authors show how previous IRL algorithms [Ng and Russell, 2000; Abbeel and Ng, 2004], can be adapted for POMDPs. The representation of the policy is a finite state controller (FSC), which is a directed graph with one node per action, and one edge per observation. Given an FSC policy, the value function V^{π} is the expected discounted return for executing π, and is defined over the joint space of FSC nodes and POMDP states. Following prior work [Ng and Russell, 2000], the authors describe three IRL algorithms in the case where the expert's policy is known. First, a Q-function based approach, where the reward is found by setting constraints on the value function that make the value of the expert's policy higher than the value of any other policy. Since there are infinitely many policies to choose from, the authors choose to use the subset of constraints that compare the expert's policy only with policies that deviate by one step from the expert's policy, and only the finite sampled beliefs reachable by the expert's policy. This process leads to a finite set of constraints that can be solved to find the estimate of the expert's reward function. Second, a dynamic programming update based approach, where the policies that are to be compared with the expert's policy are chosen among the policies that arise during the dynamic programming update of the expert's policy. The authors show that the expert's policy is optimal if its value is not improved for any belief by the dynamic programming update. Third, using the witness theorem, which provides a more computationally efficient way to find the reward function estimate, a set of nodes is found that define a feasible region for the true reward function. To put these results together, the authors present a modification of a previous IRL algorithm [Ng and Russell, 2000], where an estimate of the reward function is computed by maximizing the sum of margins between the expert's policy and all other policies, with a penalty term that encourages sparse reward functions.

In the case of IRL from sample trajectories, the authors show how three existing approaches can be extended for POMDPs: a method that uses the maximum margin between values, following the IRL algorithm from sample trajectories from Ng and Russell [2000], a maximum margin method between feature expectations following the maximum margin IRL algorithm from Abbeel and Ng [2004], and a projection method adapted from the IRL method with the same name by Abbeel and Ng [2004].

Imitation learning in multiagent settings have also been addressed [Waugh et al., 2011].

Some of the approaches presented in this section have setups that, although different, are comparable to the setup of our maximum likelihood IRL algorithms. They address the case where the reward is a linear combination of a known set of features [Ratliff et al., 2006; da Silva et al., 2006; Kolter et al., 2007], but do not provide an overarching set of tools that can be applied in the non-linear or the multiple intentions case, like the unified approach presented in this thesis.

Most of these approaches also have the disadvantage of involving complex algorithms and being computationally expensive, involving solving quadratic programs [Ratliff et al., 2006], repeatedly querying a module that compares the performance of pairs of behaviors [da Silva et al., 2006], and requiring demonstrations at different levels of complexity in terms of the dependence of the reward to features [Kolter et al., 2007]. In contrast, MLIRL (Chapter 4) is an intuitive gradient-based algorithm, with simple computations at each step.

In the rest of the chapter, I present IRL and AL algorithms that, more or less, use the typical setup described in Chapter 2.

3.3 Indirect methods. Early Approaches

The motivation for learning the reward function rather than the policy comes from the assumption that reward functions are a more succinct, robust and transferable ways to represent tasks [Abbeel and Ng, 2004]. Inverse reinforcement learning was first defined taking as inputs measurements of an agent's behavior over time, measurements of the agent's sensory inputs, and a model of the environment, and finding an estimate of the reward function that the agent is trying to optimize. In the paper, Russell [1998] enumerates a few open research problems, and sketches an algorithm for IRL. The proposed algorithm uses maximum likelihood estimation for the parameters of the reward function, maximizing the likelihood of the observed behavior. To do so, it expresses the likelihood of the behavior as a function of the reward parameters, then computes the gradient of the likelihood with respect to the parameters to improve the parameter estimates. Conceptually, this algorithm is very close to Maximum Likelihood IRL (Chapter 4).

Interestingly, the author does not implement the maximum likelihood IRL algorithm in his next IRL paper [Ng and Russell, 2000], but instead uses a linear programming formalism to compute the reward function parameters. The IRL problem is explored in three different settings. In the first setting, one with a finite state space, known model of the environment and known expert policy, the authors show the equation that needs to be solved to obtain the reward function. The equation admits an infinite number of solutions, including degenerate ones assigning the same reward to each state. Therefore, the IRL problem is ill-posed. To choose a meaningful reward, the authors require a solution that makes a single step deviation from the optimal policy as costly as possible, and favor solutions for which rewards are small numbers. These constraints are framed in a linear programming formulation, with a unique solution that assigns non-zero rewards to only a small number of states. In settings with very large state spaces, the authors use function approximation to represent the reward function as a linear combination of a known set of state features, and the problem becomes that of estimating the feature parameters. Since the reward is a linear combination of features, a linear programming approach can again be applied to find a reward function such that the value of the expert's policy is greater than the value of any other policy. The third setting is one where the agent does not have access to the expert's policy, but rather to a set of demonstrations from the expert. The goal is to find the reward function such that the value of the unknown expert policy is maximal. Under the assumption that the reward is a linear combination of a known set of features, the algorithm starts with a random policy, then iteratively improves the reward parameters by solving a linear program, this time using estimates of the expert's policy computed from the trajectories. The performance of the three algorithms is measured in three experimental settings, a discrete 5×5 grid world with stochastic transitions and one goal state, the mountain car domain, and a continuous version of the grid world.

The assumption that the expert's reward function is a linear combination of a known set of features is very common in the IRL literature [Abbeel and Ng, 2004; Ratliff et al., 2006; da Silva et al., 2006; Syed and Schapire, 2007; Kolter et al., 2007; Syed et al., 2008; Ziebart et al., 2008; Babeş-Vroman et al., 2011; Klein et al., 2012; Almingol et al., 2013]. The problem becomes that of estimating the feature parameters in the linear dependence.

In some settings, the reward's dependence on features is non-linear, and the linearity assumption can lead to solutions that result in poor performance. A few IRL approaches address non-linear rewards. Neu and Szepesvári [2007] provide a method for differentiable classes of reward functions, but only test it in a linear setting. Other approaches using non-linear mappings of features for the reward are described in specific settings, for example regression trees [Levine et al., 2010], Gaussian processes [Levine et al., 2011], and Bayesian frameworks [Ramachandran and Amir, 2007; Lopes et al., 2009]. In contrast to these algorithms, our modular IRL approach (Chapter 5) allows any linear on non-linear regression algorithm to be used in the IRL context.

3.4 IRL and AL Approaches. Different Objectives

IRL and AL algorithms differ not only in their algorithmic approach, but also in the objective function they seek to optimize [Neu and Szepesvári, 2009]. In early IRL work [Ng and Russell, 2000] the expert's policy is given, and the reward is chosen to make any one-step deviations from the expert's policy as costly as possible. If the expert's policy is unknown, other objectives need to be considered. In projection [Abbeel and Ng, 2004], the objective is to make the features encountered by the apprentice's policy match those of the expert. Policy matching [Neu and Szepesvári, 2007] tries to make the actions taken by the learned policy as close as possible to those of the expert. LPAL and MWAL [Syed et al., 2008] behave in such a way that they outperform the expert according to the computed reward parameters. Maximum entropy [Ziebart et al., 2008] defines a probability distribution over behaviors as a function of the reward parameters, and finds the parameters that maximize the likelihood of the expert's demonstrations. I discuss these approaches in detail below.

A unified IRL framework for linear IRL algorithms has been developed [Neu and Szepesvári, 2009] to describe early work. In this framework, each algorithm takes as input an $MDP \setminus r$, that is, an MDP without the reward function, and a set of trajectories $D = \{\xi_1, \ldots \xi_N\}$, and outputs a reward function. The algorithms incrementally use an update rule with step size α_k at iteration k, update function g, also called the link-function (for example, the exponential function for multiplicative updates, and the identity function for additive updates), and parameter update Δ_k at iteration k. The authors claim that the update of any IRL algoritm can be written as $\theta_{k+1} = g(g^{-1}(\theta_k) + \alpha_k \Delta_k)$. The last algorithm parameter is the dissimilarity function $J = J(r, D)$, measuring the similarity between the optimal behavior corresponding to r and the demonstrations D. The authors show how each of these parameters is instantiated for projection [Abbeel and Ng, 2004], MWAL [Syed et al., 2008], Max-margin planning [Ratliff et al., 2006], policy matching [Neu and Szepesvári, 2007], and maximum entropy IRL [Ziebart et al., 2008].

The expressions for some of the quantities mentioned above in the case of these algorithms are complicated, and they may not seem straightforward at first glance. In contrast, these functions can be easily computed for MLIRL (Chapter 4). MLIRL uses a constant step size of α at each iteration, and an additive update rule, $g(x) = x$. The parameter update is $\Delta_k = \log(P(D|\theta_k))$, where θ_k is the current estimate of the feature parameters and the dissimilarity function is $J(\theta, D) = -\frac{1}{N}\Sigma_{i=1}^{N} \log P(\xi_i|\theta)$, with lower values when the optimal behavior with respect to the reward function given by θ is more similar to the dataset D.

So far, I have given a general overview of linear IRL algorithms, and the methods they use in computing an estimate of the expert's reward function. In the following, these approaches are presented in detail.

One of the first applications of IRL is in solving apprenticeship learning (AL), the problem of learning an estimate of the expert's policy from demonstrations [Abbeel and Ng, 2004]. As mentioned before, the motivation for learning the expert's underlying reward function instead of just a mapping from states to actions is that reward functions provide a better generalization of the task, especially in areas of the state space where the expert has not demonstrated it. A key quantity defined in this approach are the feature expectations of a policy, more specifically, the vector of discounted accumulated feature expectations when following the policy. The authors show that if two policies have close feature expectations, then their values are also close, therefore computing a policy with an optimal value can be solved by finding a policy that matches the feature expectations according to the expert's policy. The paper describes two algorithms, *maximum margin* and *projection*, taking as input the expert's feature expectations, a model of the environment, and a feature mapping. *Maximum margin* iteratively builds a set of policies until a policy is found for which the distance between its value and the estimate of the expert's policy (using the expert's feature expectations) is smaller than a given threshold. If such a policy is not found, a new policy is generated as the optimal policy corresponding to a reward function for which the expert does better by a margin than any of the policies previously found by the algorithm. This step involves solving a quadratic program. To avoid using a quadratic programming solver, the authors propose the *projection* algorithm, which generates a new feature expectation vector by computing the orthogonal projection of the expert's feature expectations onto the line through the last two previously generated feature expectation vectors. The authors show that the algorithms always converge given enough data, and they provide a sample complexity bound.

There are a number of applications of projection and maximum margin. One paper [Abbeel et al., 2008] applies maximum margin to the problem of navigating though a parking lot. Another paper [Chandramohan et al., 2011] applies projection to dialogue systems. Another one applies projection to learn behavior styles from playing a video game with various characters [Lee and Popović, 2010]. These algorithms have also been extensively applied to the problem of teaching an autonomous helicopter to fly [Abbeel et al., 2010].

Matching the expert's feature expectation is the goal of another IRL approach [Ziebart et al., 2008]. Since this computation can result in multiple behaviors, the principle of maximum entropy is used to select a distribution over behaviors that matches the expert's feature expectations. This distribution is softmax or Boltzmann, assigning the same probability to plans yielding the same reward, and higher probabilities to choices with higher rewards. The parameters of the reward function are chosen such that the likelihood of the observed behavior is maximized and computed using gradient descent.

Another approach [Ziebart et al., 2010] uses the idea of matching feature expectations as part of the optimization for maximum causal entropy, an extension of maximum entropy in settings where side-information is revealed over time. Side-information is a set of variables that are not predicted, but are related to the predicted variables (for example, the environment's transition dynamics which the agent discovers as it interacts with the environment). In the context of an agent interacting with a stochastic environment, the objective function is the causal entropy based on the probability of the actions causally conditioned on the side information available at each step. The optimization can be solved recursively to obtain the distribution $P(a_t|s_t)$, and the parameters of the reward function are found using a gradient method, similar to the one used in previous work [Ziebart et al., 2008].

The same principle of maximum entropy is used in a continuous inverse optimal control algorithm [Levine and Koltun, 2012], which I describe later in Section 3.7.

In a game-theoretic approach to AL [Syed and Schapire, 2007], the authors note that AL algorithms from previous work [Abbeel and Ng, 2004] seek to mimic the expert's behavior, therefore their performance is both upper and lower bounded by the performance of the expert. To address the situation where the expert's demonstrations are not optimal, the AL problem is cast as a zero-sum game, in which the environment adversarially chooses a reward function to minimize the performance of the learner, and the learner chooses a mixed policy with the goal of maximizing its performance relative to the reward function chosen by the environment. The learner is guaranteed to do no worse than the expert according to the unknown reward function. The authors use the multiplicative weights algorithm (MWAL) to find an approximately optimal strategy, which provides the learner with a policy that can outperform the expert under certain conditions.

A related approach frames the AL problem as linear programming (LPAL) [Syed et al., 2008], using the Bellman flow constraints to compute a stationary policy with the same value as the mixed policy computed by MWAL. The authors also describe MWAL-Dual, a modification of MWAL that uses the dual of the linear programming formulation for solving MDPs as an intermediate step. The LPAL algorithm computes an estimate of the expert's value function from demonstrations, and uses a linear program to compute the occupancy measures of a policy that outperforms the expert, using the same maximization step as the one for MWAL.

Learning from suboptimal demonstrations has also been addressed in the context of teaching an autonomous helicopter to fly [Coates et al., 2008, 2009]. The assumption made in these papers is that the optimal behavior is encoded in the joint set of suboptimal demonstrations. The paper presents an algorithm for extracting optimal trajectories from many suboptimal demonstrations, and building a model of the system's dynamics in the vicinity of the optimal trajectory, allowing the learner to perform better than the expert. The paper proposes a generative model for the expert's demonstrations as noisy observations of the unobserved, intended trajectories, then an EM algorithm to infer both the unobserved ideal trajectories, and a time alignment of the demonstrations, used to learn a local model in the vicinity of the demonstration. The generative model uses a normal distribution over initial states, and an approximate model of the dynamics, including a term for noise, which is normally distributed with mean 0. Each demonstration is represented as a set of independent observations of the hidden ideal trajectory z. The authors extend the generative model to account for other sources of error like unintentional position drifting and including prior knowledge. With this extended generative model, the algorithm successfully finds the most likely hidden trajectory.

A gradient method for AL has been developed [Neu and Szepesvári, 2007], combining direct imitation learning methods which learn a mapping from states to actions (policy) by supervised learning with indirect imitation learning methods learning a policy by assuming the expert is acting optimally in the environment by using an IRL algorithm. The authors use a loss function that penalizes deviations from the expert's policy like in supervised learning, but the policy is computed by tuning a reward function instead of finding the parameters of the policy, such that it can generalize to parts of the state space that are not visited by the expert during the demonstrations. The mapping between the Q-function and the policy is the Boltzmann action-selection policy, chosen because it is smooth and differentiable. To find the reward parameters, the authors use gradient descent to minimize the distance between the estimate of the expert's policy and the apprentice's policy.

The IRL algorithms described in this section assume the expert's reward is a linear combination of a known set of features [Abbeel and Ng, 2004; Ziebart et al., 2008, 2010; Syed and Schapire, 2007; Syed et al., 2008; Neu and Szepesvári, 2007]. These algorithm only address learning a linear function from demonstrations coming from a single expert, and do not provide a unified framework for learning non-linear or multiple rewards. In contrast, I show how maximum likelihood can be applied not only to the linear case (Chapter 4), but also to the non-linear (Chapter 5) and multiple intentions case (Chapter 6).

3.5 Bayesian Approaches

Bayesian IRL [Ramachandran and Amir, 2007] casts IRL as a Bayesian inference problem, where prior knowledge about the expert's reward function can be included in the inference. The posterior is updated using the expert's demonstrations as evidence. The authors point out that the normalizing factor is hard to compute, and therefore, the posterior is estimated by using a sampling technique. IRL becomes a reward-estimation problem, where the loss function can be computed as the norm of the distance between the estimated and actual rewards. The estimated reward that minimizes this loss is the mean of the posterior. If the problem is policy estimation, the computed policy is the one that minimizes the expected loss over the posterior reward distribution, where the loss is defined as a function of a reward and a policy, and is computed as the norm of the distance between the value of the optimal policy and the value of the policy for the current reward function estimate. The authors claim that the policy minimizing this norm is the optimal policy for the MDP with reward function the expected value of the posterior reward distribution. Instead of computing the posterior distribution on reward functions, the authors use a computationally efficient method to derive the mean of the posterior by MCMC sampling and return the sample mean as its estimate, while at the same time keeping track of the optimal policy as the reward changes along the Markov chain. While moving along the Markov chain, the optimal policy for the current reward R is either known, or, when a new reward vector in the chain is encountered, the new optimal policy is only slightly different than the old one and its update can be computed by using a few steps of policy iteration, thus avoiding solving the MDP at each step.

A number of other IRL algorithms build on the ideas from Bayesian IRL [Ramachandran and Amir, 2007]. Active learning reward estimation algorithms rely on this algorithm [Lopes et al., 2009; Cohn et al., 2011] (see Section 3.6) and it is also used in non-parametric approaches for learning about multiple intentions also rely on it [Choi and Kim, 2012] (see Section 3.8).

An application of inverse reinforcement learning to preference elicitation generalizes the Bayesian IRL framework [Rothkopf and Dimitrakakis, 2011] to allow for obtaining the agent's preferences and policy as well as their reward function. The paper proposes a Bayesian formulation for inverse reinforcement learning as preference elicitation, with a structured prior on the expert's utilities and policies, then derives two different Markov chain procedures for preference elicitation. Two generative models are proposed, one for just the data, and another one for both the data and the reward function. The observables are the prior on reward functions ξ, the prior on policies given a reward function ψ, and the data D, which is a set of demonstrations from the expert, (and, in the reward-augmented case, the rewards r^T). The latent variables are the actual reward function ρ (drawn from the prior) and corresponding policy π used to generate the data.

This setup is extended to the problem of multitask IRL [Dimitrakakis and Rothkopf, 2012] using a hierarchical population model.

None of these papers provide a unified approach to all the subproblems addressed in this thesis—linear IRL, non-linear IRL and multiple intentions IRL, but one such approach could be built by using the algorithm described by Ramachandran and Amir [2007] for single intention IRL and the non-parametric method by Choi and Kim [2012] for multiple intentions IRL.

3.6 Active Learning

Inverse reinforcement learning has also been studied in the context of active learning. The first method introduced the active sampling algorithm [Lopes et al., 2009]. In this approach, the agent receives a small number of demonstrations from the expert to begin with, then repeatedly chooses a state where its uncertainty about the optimal action is maximized, and queries the expert about the optimal action in that state. Two Bayesian IRL algorithms are used for learning, a gradient approach which computes the posterior probability of the reward function given the data, $Pr(r|D)$ by maximizing the likelihood of the data under a uniform prior over reward functions and an MCMC sampling approach that computes the estimate of the mean of the posterior over the reward functions by sampling from the posterior and then averaging over the samples [Ramachandran and Amir, 2007]. To compute the uncertainty about the expert's policy, the authors define a distribution over possible values for a policy in each state, $\mu_{sa}(p)$, the probability that the value of the policy for state action pair (s, a) is p. They discretize this distribution by splitting the interval $[0, 1]$ into K sub-intervals I_k, with $k = 1 \ldots N$, and changing $\mu_{sa}(k)$ to be the probability that $\pi(s, a)$ belongs to interval I_k. The mean-entropy corresponding to the discretized distribution quantifies the uncertainty that the agent has about the action it should take in that state, therefore the state that is chosen for query is the one that maximizes this mean-entropy over all the states. The distribution is estimated by generating N samples using Monte-Carlo sampling, then averaging over the samples to obtain the probability of each interval, $\mu_{sa}(k)$, according to a previously proved theorem [Ramachandran and Amir, 2007]. Two active learning IRL algorithms are presented, the first one computes an estimate of the posterior $Pr(r|D)$ from the initial (assumed small) data set, then uses this posterior to compute the mean-entropy for each state, and queries the expert for the action to be taken in that state. The new sample is used to re-estimate $Pr(r|D)$, and so on. The second algorithm considers the case of large dimensional state spaces, where estimating the posterior using MC sampling is computationally too expensive. The sampling step is replaced by estimating the expert's reward function using the gradient ascent method mentioned above, then estimating the posterior in a neighborhood of the expert's reward.

A a similar setup is used in a related approach to action-queries [Cohn et al., 2011], this time the focus being on the uncertainty over reward functions. When choosing which state to query, given the current probability distribution over reward functions, the agent can compute the mean entropy for each state, and query the state with the maximum mean-entropy. The expected myopic gain quantifies the uncertainty about states [Cohn et al., 2010] and assesses the goodness of a query in terms of its long term expected gain. The gain for knowing that the answer to query q is o given the current state and the current distribution over reward functions is the difference between the expected value at the current state of the policy calculated according to the new information and the policy calculated beforehand. Since the answer to the query q is not known ahead of time, the agent computes an expectation of the gain for

q over all the possible answers, also called the expected myopic gain (EMG) of query q, and chooses the query with the maximum expected myopic gain. To compute the EMG, the authors use the Bayesian inverse reinforcement learning algorithm [Ramachandran and Amir, 2007], which provides a way to update the probability distribution over reward functions given demonstrations of action choices from the expert. They also use a previous result [Ramachandran and Amir, 2007] to replace the expected value of a policy over a reward distribution with its value for the mean of the distribution to simplify the formula for the gain corresponding to a query q.

The two algorithms presented in this section do not limit the reward function estimate to linear functions, but instead see the reward as an arbitrary mapping between states and values. Lopes et al. [2009] use a gradient approach similar to the MLIRL algorithm (Chapter 4), but the gradient is taken with respect to the reward of each state. Therefore, if a region of the state space hasn't been visited by the expert in the demonstrations, the estimate of the reward in that region will be inaccurate. Ramachandran and Amir [2007] and Cohn et al. [2010] have the same issue. In contrast, by expressing the reward as a function of features, MLIRL (Chapter 4) and Modular IRL (Chapter 5) recover the expert's reward from fewer demonstrations, that do not necessarily need to cover the entire state space. No active learning IRL algorithm for multiple intentions has been published to my knowledge, although the application of an active learning IRL algorithm to a multi-class classification has been discussed by Melo and Lopes [2013].

3.7 Learning Non-linear Reward Functions

A few notable IRL algorithms do not make the assumption that the reward is a linear combination of a known set of features.

I start with two approaches to IRL that build a feature set to best describe the expert's reward function from a larger set of basis features or feature components. The computed reward function becomes a non-linear combination of the feature components. One approach [Ratliff et al., 2007] iteratively builds a model of the reward or cost function using the current feature set, initially the set of feature components, then runs a planner to obtain the path with the lowest loss. If the features are not expressive enough to represent the task, the generated path will differ significantly from the expert demonstrated path. If new features are needed, a training set is built by gathering feature vectors encountered along the generated path as positive examples, and feature vectors encountered along the expert demonstrated path as negative examples. This training set is used to build a classifier which can be queried in every cell of all the example maps to build new features. The new feature is the one that best raises the cost of the current erroneous path, and at the same time lowers the cost of the example path, therefore contributing to better explaining the decisions made in the paths demonstrated by the expert.

Another approach [Levine et al., 2010] uses a similar iterative structure but the optimization is different. The algorithm starts with an empty feature set, and iteratively runs two steps: the optimization step builds a reward function given the current feature set, and the fitting step generates a new feature hypothesis that better captures the variations in the reward function. In the optimization step, a reward function R is computed for which the optimal policy is consistent with the expert demonstrations, while, at the same time, minimizing the sum of squared errors between R and its projection unto the linear basis of the current feature set. This calculation is accomplished using a quadratic programming solver. The fitting step chooses to generate new features or merge old ones, such that the resulting feature set gives greater resolution to regions of the state space where the old features are too coarse, and lower resolution to regions where the old features are too fine. The new feature set is obtained by building a regression tree for the current reward function estimate, with the feature components acting as tests at the nodes. Each leaf represents a subset of the state space ϕ_l, and the new features are the set of indicator functions for membership in ϕ_l. The smallest tree is chosen that produces a sufficiently rich feature set to represent the current reward function.

Another non-linear approach casts IRL as a Gaussian process regression problem [Levine et al., 2011]. The observations of the expert's demonstrations are assumed to be noisy, and the model learns the true rewards and the parameters of the model by maximizing their probability under the expert demonstrations. The kernel function used is the automatic relevance detection kernel (a variant of the RBF kernel), with hyper-parameters β, the overall variance, and Λ, the feature weights. For large state spaces, a subset of the features is chosen to contain the feature values of all states visited during the demonstrations. Other kernels can be used to encode prior knowledge of the reward function.

In a continuous inverse optimal control approach [Levine and Koltun, 2012], a probabilistic algorithm for continuous domains is developed, using an approximation of the reward function that allows for learning from examples that are globally optimal, but only requires locally optimal examples. The algorithm only considers the shape of the reward function in the neighborhood of the expert demonstrations. The goal of the algorithm is to find a reward function under which the optimal actions match the expert's demonstrations, and since the demonstrated trajectories are assumed to be suboptimal, the maximum entropy model is used for the expert's behavior, to account for "noise" in the demonstrations. In this model, the probability of the expert's actions is proportional to the exponential of the rewards encountered along the trajectory, and the log likelihood of this probability is maximized to find an estimate of the expert's reward function. Computing this quantity requires access to the full policy under an estimated reward function, which becomes intractable in high-dimensional state spaces. Instead of computing the log likelihood of the action probability exactly, this quantity is approximated by using a second order Taylor expansion of the reward function around the demonstrated actions. Under this approximation, reward functions with small gradients are more likely.

While these approaches are suited for learning linear and non-linear reward functions, they do not address the problem of learning about multiple intentions. In the next section, I describe a few IRL algorithms that infer the rewards corresponding to demonstrations coming from multiple experts with varying intentions.

3.8 Learning about Multiple Intentions

Our work on learning from demonstrations generated by multiple intentions or reward functions (Chapter 6) has been extended to address the requirement that the number of reward functions needs to be known in advance by using non-parametric methods [Choi and Kim, 2012; Almingol et al., 2013]. The first approach extends Bayesian IRL [Ramachandran and Amir, 2007] with the

Dirichlet process model, and clusters the trajectories coming from the same reward function, while at the same time computing the reward function corresponding to each cluster. The number of clusters is flexible and does not need to be specified in advance, and the framework allows for inferring the reward function for a trajectory even when its reward function is novel. The inference is done using MCMC sampling for the latent variables. The algorithm has two steps, first the cluster assignment is updated, then the reward functions for each cluster are updated in light of the new cluster assignments. If a new cluster is drawn, then its reward function is a new reward function drawn from the prior.

The second approach [Almingol et al., 2013] focuses on motion planning, a setting with very large action and parameter spaces. The function to be estimated is the potential function V dictating the motion of a particle, and it is assumed to be a linear combination of a known set of features. Under the assumption that V is smooth, the continuous dynamical system describing the motion can be discretized with respect to time, and the equations corresponding to a whole trajectory can be represented as a system of linear equations, $Y = X\beta$. Each potential generating the input trajectories has its own set of parameters β_k. To compute these parameters, the model assumes that the trajectory set is generated by a mixture model of K linear dynamical systems, that is potentials V_k, with parameters β_k, and a Dirichlet process is used to estimate the mixture model over the parameters of each controller with a Laplacian prior over the parameters.

These approaches do not provide a unified framework for linear, non-linear, and multiple intentions IRL. As mentioned before, to build such a framework, one could use Bayesian IRL [Ramachandran and Amir, 2007] for single intention IRL, and Choi and Kim [2012] for multiple intentions IRL.

3.9 Conclusions

My thesis statement is:

> By casting inverse reinforcement learning as a maximum likelihood problem, we can provide a unified approach to linear inverse reinforcement learning, non-linear inverse reinforcement learning, and multiple intentions inverse reinforcement learning leading to effective and natural solutions.

In this chapter, I have surveyed some of the existing IRL and AL algorithms, and shown that no such unified approach to IRL currently exists. While some algorithms cover two of these subproblems, the linear and non-linear single intentions cases, [Ramachandran and Amir, 2007; Neu and Szepesvári, 2007; Ratliff et al., 2007; Lopes et al., 2009; Levine et al., 2010, 2011], they do not address learning from demonstrations coming from multiple expert with varying intentions.

One could argue that the non-parametric Bayesian inverse reinforcement learning approach could be applied to all three settings: the linear case, the non-linear case, and the multiple intentions case. If all the demonstrations came from one expert, the non-parametric approach for multiple reward functions may figure out the commonality of all the trajectories in the dataset. However, the underlying IRL algorithm for trajectories coming from a single expert is Bayesian IRL [Ramachandran and Amir, 2007], which uses MCMC to estimate the posterior in each iteration and is, therefore computationally expensive. If it is known whether the demonstrations come from one or more expert, and the shape of the reward function, it is more efficient to use specialized IRL algorithms for each case.

Another advantage of the unified IRL framework is that the algorithms are very simple. MLIRL is simply a gradient ascent algorithm that iteratively changes the feature parameters to maximize the probability of the expert demonstrations (Chapter 4). Modular IRL adds to each iteration a simple and intuitive step of fitting the rewards to the hypothesis class that the reward function belongs to (Chapter 5). Multiple intentions IRL is an EM-based clustering method, and MLIRL naturally fits as the M-step (Chapter 6). Maximum likelihood algorithms are simple and intuitive, yet we have not seen them described anywhere else.

Chapter 4
Maximum Likelihood Inverse Reinforcement Learning

In this chapter I describe maximum likelihood inverse reinforcement learning (or MLIRL) and show that it is a justified approach for linear, single expert IRL. I start by revisiting several algorithms for IRL and AL and briefly describing a new linear programming based algorithm and will compare this algorithm and MLIRL with with existing work in Section 4.1. MLIRL is described in Section 4.2 with implementation details in Section 4.3. In Section 4.4 I describe an experiment showing that MLIRL obtains optimal performance in an envionment where other IRL and AL algorithms have been previously tested and conclude in Section 4.5.

4.1 Related Work

As discussed previously in Chapter 2, IRL algorithms receive as input a model of the environment in the form of an MDP$\backslash r$:(S, A, T, γ), and demonstrations of the task to be learned $D = \{\xi_1, \ldots \xi_N\}$. They then compute an estimate of the reward function used to generate the demonstrations, under the assumption that the demonstrator is a reinforcement learning agent acting optimally or nearly optimally with respect to their reward function.

Reward functions are parametrized by a vector of reward weights θ applied to a feature vector for each state-action pair $\phi(s, a)$. Thus, in the linear case, a reward function is written $r_\theta(s, a) = \theta^T \phi(s, a)$. If the expert's reward function is given by θ_E, the apprentice's objective is to behave in a way that maximizes the discounted sum of expected future rewards with respect to r_{θ_E}. However, the apprentice does not know θ_E and must use information from the observed trajectories to decide how to behave. It can, for example, hypothesize its own reward weights θ_A and behave accordingly.

As shown in Chapter Chapter 3, IRL algorithms differ not just in their algorithmic approach but also in the objective function they seek to optimize [Neu and Szepesvári, 2009]. I briefly revisit IRL and AL algorithms I will use in our comparison in Section 4.4. In Projection [Abbeel and Ng, 2004], the objective is to make the features encountered by the apprentice's policy match those of the expert. LPAL and MWAL [Syed et al., 2008] behave in such a way that they outperform the expert according to θ_A. Policy matching [Neu and Szepesvári, 2007] tries to make the actions taken by its policy as close as possible to those observed from the expert. Maximum Entropy IRL [Ziebart et al., 2008, 2010] defines a probability distribution over complete trajectories as a function of θ_A and produces the θ_A that maximizes the likelihood of the observed trajectories.

I devised two new IRL algorithms for our comparisons. The linear program that constitutes the optimization core of LPAL (Linear Programming Apprenticeship Learning) is a modified version of the standard LP dual for solving MDPs [Puterman, 1994]. It has as its variables the "policy flow" and a minimum per-feature reward component. Taking the dual of this LP results in a modified version of the standard LP primal for solving MDPs. It has as its variables the value function and θ_A. Because it produces explicit reward weights instead of just behavior, this algorithm is called Linear Programming Inverse

Reinforcement Learning (or LPIRL). Because its behavior is defined indirectly by θ_A, it can produce slightly different answers from LPAL. The second algorithm seeks to maximize the likelihood of the observed trajectories, as described in the next section.

4.2 Maximum Likelihood Inverse Reinforcement Learning (MLIRL)

In this section I present a simple IRL algorithm called Maximum Likelihood Inverse Reinforcement Learning (MLIRL). Like Bayesian IRL, it adopts a probability model that uses θ_A to create a value function and then assumes the expert randomizes at the level of individual action choices. Like Maximum Entropy IRL, it seeks a maximum likelihood model. Like Policy matching, it uses a gradient method to find optimal behavior.

To define the algorithm more formally, I start by detailing the process by which a hypothesized θ_A induces a probability distribution over action choices and thereby assigns a likelihood to the trajectories in D. First, θ_A provides the rewards from which discounted expected values are derived:

$$Q_{\theta_A}(s,a) = \theta_A^T \phi(s,a) + \gamma \sum_{s'} T(s,a,s') \bigotimes_{a'} Q_{\theta_A}(s',a').$$

Here, the "max" in the standard Bellman equation is replaced with an operator that blends values via Boltzmann exploration [John, 1994]: $\bigotimes_a Q(s,a) = \sum_a Q(s,a) e^{\beta Q(s,a)} / \sum_{a'} e^{\beta Q(s,a')}$. This approach makes the likelihood (infinitely) differentiable, although, in practice, other mappings could be used. The Botzmann exploration is commonly used as a way of inducing variability in behavior that is tied to the values of the actions themselves. It is also used as a model for human decision making [Luce, 1959].

Algorithm 1 Maximum Likelihood IRL

```
Input: MDP$\backslash r$, features $\phi$, trajectories $\{\xi_1, \ldots, \xi_N\}$, number of iterations $M$,
step size for each iteration ($t$) $\alpha_t$, $1 \leq t < M$.
Initialize: Choose random set of reward weights $\theta_1$.
for $t = 1$ to $M$ do
    Compute $Q_{\theta_t}$, $\pi_{\theta_t}$.
    $L = \sum_i \sum_{(s,a)\in\xi} \log(\pi_{\theta_t}(s,a))$.
    $\theta_{t+1} \leftarrow \theta_t + \alpha_t \nabla L$.
end for
Output: Return $\theta_A = \theta_M$.
```

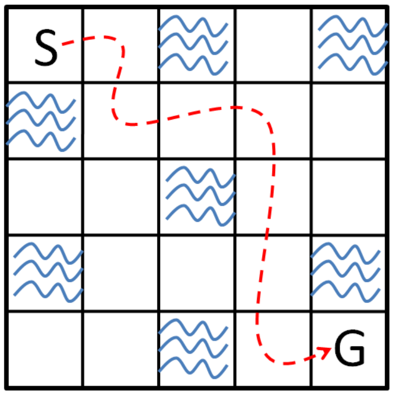

Figure 4.1: A single trajectory from start to goal.

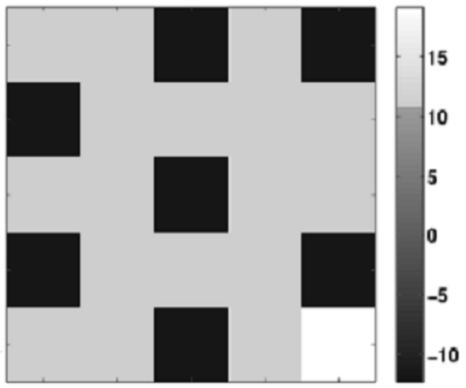

Figure 4.2: Reward function computed using MLIRL.

I calculate these values via value iteration and use $\beta = 0.75$.

The Boltzmann exploration policy is $\pi_{\theta_A}(s,a) = e^{\beta Q_{\theta_A}(s,a)} / \sum_{a'} e^{\beta Q_{\theta_A}(s,a')}$. Under this policy, the log likelihood of the trajectories in D is $L(D|\theta) =$

$$\log \prod_{i=1}^{N} \prod_{(s,a)\in\xi_i} \pi_\theta(s,a) = \sum_{i=1}^{N} \sum_{(s,a)\in\xi_i} \log \pi_\theta(s,a). \tag{4.1}$$

MLIRL seeks $\theta_A = \text{argmax}_\theta\, L(D|\theta)$—the maximum likelihood solution. Here, this function is optimized via gradient ascent (although I experimented with several other optimization approaches). These pieces come together in Algorithm 1. I provide more detail in Section 4.3.

It is open whether infinite-horizon discounted value iteration with the Boltzmann operator will converge. In our finite-horizon setting, it is well-behaved and produces a well-defined answer, as illustrated later in this section and in our experiments (Section 4.4).

To illustrate the functioning of the MLIRL algorithm, I use the example shown in Figure 4.1. It depicts a 5×5 grid with puddles (indicated by wavy lines), a start state (S) and an absorbing goal state (G). The dashed line shows the path taken by an expert from S to G. The algorithm is now faced with the task of inferring the parameters of the expert's reward function θ_E using this trajectory. It appears that the expert is trying to reach the goal by taking the shortest path and at the same time avoid any intermediate puddles. The assignment of reward weights to the three features—ground, puddle, and goal—that makes this trajectory maximally likely is one that assigns the highest reward to the goal. (Otherwise, the expert would have preferred to travel somewhere else in the grid.) The probability of the observed path is further enhanced by assigning lower reward weights to puddles than to ground. Thus, although one explanation for the path is that it is one of a large number of possible shortest paths to the goal, the trajectory's probability is maximized by assuming the expert intentionally missed the puddles. The MLIRL-computed reward function is shown in Figure 4.2, which assigns high likelihood (0.1662) to the single demonstration trajectory.

One of the challenges of IRL is that, given an expert policy, there are an infinite number of reward functions for which that policy is optimal in the given MDP. Like several other IRL approaches, MLIRL addresses this issue by searching for a solution that not only explains why the observed behavior is optimal, but also by explaining why the other possible behaviors are suboptimal. The advantage of probabilistic IRL methods, like MLIRL, over non-probabilistic ones is that they can account for stochasticity in the data. They are, therefore, able to learn optimal behavior from noisy data and suboptimal demonstrations.

4.3 Implementation Details

In this section I describe the MLIRL algorithm in detail.

Given:

- MDP (States S, Actions A, Transitions T, discount factor γ)
- Trajectories, $D = \{\tau_k\}$
- Features $\Phi = \{\phi_1, \ldots, \phi_d\}$
- Number of iterations N
- Step size for iteration t, α_t
- Boltzmann temperature β

Initialize: Choose an arbitrary set of weights, $w^0 = \{w_1^0, w_2^0, \ldots\}$, $r_0 = \sum_j \phi_j w_j^0$.

```
**For each iteration** $t = 0 \ldots n$ :
    $Q_0(s,a) = r_t(s) = \sum_j \phi_j w_j^t$.
    $V_0(s) = r_t(s) = \sum_j \phi_j w_j^t$ .
    $\frac{d}{dw_j} V_0(s) = \phi_j(s)$
    **For each iteration** $i = 1 \ldots K$ :
        $\forall s \in S, a \in A, Q_i(s,a) = r_t(s) + \gamma \sum_{s'} T(s,a,s') V_{i-1}(s')$.
        $\frac{dQ_i(s,a)}{dw_j} = \phi_j(s) + \gamma \sum_{s'} T(s,a,s') \frac{dV_{i-1}(s')}{dw_j}$.
        $Z_i(s) = \sum_a e^{\beta Q_i(s,a)}$.
        $\frac{d}{dw_j} Z_i(s) = \beta \sum_a e^{\beta Q_i(s,a)} \frac{d}{dw_j} Q_i(s,a) = \beta \sum_a e^{\beta Q_i(s,a)} (\phi_j(s)$
    $+ \gamma \sum_{s'} T(s,a,s') \frac{dV_{i-1}(s')}{dw_j})$.
        $\pi_i(s,a) = \frac{e^{\beta Q_i(s,a)}}{Z_i(s)}$.
        $\frac{d}{dw_j} \pi_i(s,a) = \frac{\beta Z_i(s) e^{\beta Q_i(s,a)} \frac{d}{dw_j} Q_i(s,a) - e^{\beta Q_i(s,a)} \frac{d}{dw_j} Z_i(s)}{Z_i^2(s)}$
        $V_i(s) = \sum_a \pi_i(s,a) Q_i(s,a)$.
        $\frac{d}{dw_j} V_i(s) = \sum_a (Q_i(s,a) \frac{d}{dw_j} \pi_i(s,a) + \pi_i(s,a) \frac{d}{dw_j} Q_i(s,a))$.
        $L_i = \log Pr(D) = \sum_{\tau \in D} Pr(\tau) \sum_{(s,a) \in \tau} \log(\pi_i(s,a))$.
        $\frac{dL_i}{dw_j} = \sum_{\tau \in D} Pr(\tau) \sum_{(s,a) \in \tau} \frac{1}{\pi_i(s,a)} \frac{d\pi_i(s,a)}{dw_j}$.
        $\frac{dL}{dw_j} = \frac{dL_i}{dw_j}$.
      **End For each iteration** $i$.

      $\forall j, w_j^{t+1} = w_j^t + \alpha_t \frac{dL}{dw_j}$ .

    $r_{t+1}(s) = \sum_j \phi_j(s) w_j^{t+1}$.
    $w_j = w_j^{t+1}, \forall j$.
    **End For each iteration** $t$.

    **Output:** $w = \{w_1, w_2, \ldots\}$.
```

The MLIRL algorithm behaved well in the experimental settings I tested it in (See Section 4.4 and Section 6.2). These settings are suitable for applying the gradient, because the likelihood of the demonstrations as a function of the parameters is differentiable.

4.4 Experiment

This experiment was designed to compare the performance of the MLIRL and LPIRL algorithms with five existing IRL/AL approaches summarized in the beginning of this chapter, and described in more detail in Chapter 3. I compare these seven approaches to assess how well they perform apprenticeship learning in a grid world with a single expert (single intention). In Chapter 6, I show that MLIRL is a natural fit in a clustering approach computing the reward functions used to generate trajectories coming from multiple experts, and illustrate its performance in a highway car domain, where a few existing approaches have already been tested [Abbeel and Ng, 2004; Syed et al., 2008].

I used implementations of the MLIRL, LPIRL, Maximum Causal Entropy IRL, LPAL, MWAL, Projection, and Policy Matching algorithms, and obtained implementations from the original authors wherever possible.

The grid world environment used in this experiment is similar to one used by Abbeel and Ng (2004) and Syed et al. (2008), with a grid of size of 16×16. Movement of the agent is possible in the four compass directions with each action having a 30% chance of causing a random transition. The grid is further subdivided into non-overlapping square regions, each of size 4×4. Using the same terminology as Abbeel and Ng (2004), I refer to the square regions as "macrocells". The partitioning of the grid results in a total of 16 macrocells. Every cell in the gridworld is characterized by a 16-dimensional feature vector ϕ indicating, using a 0 or 1, which macrocell it belongs to. A

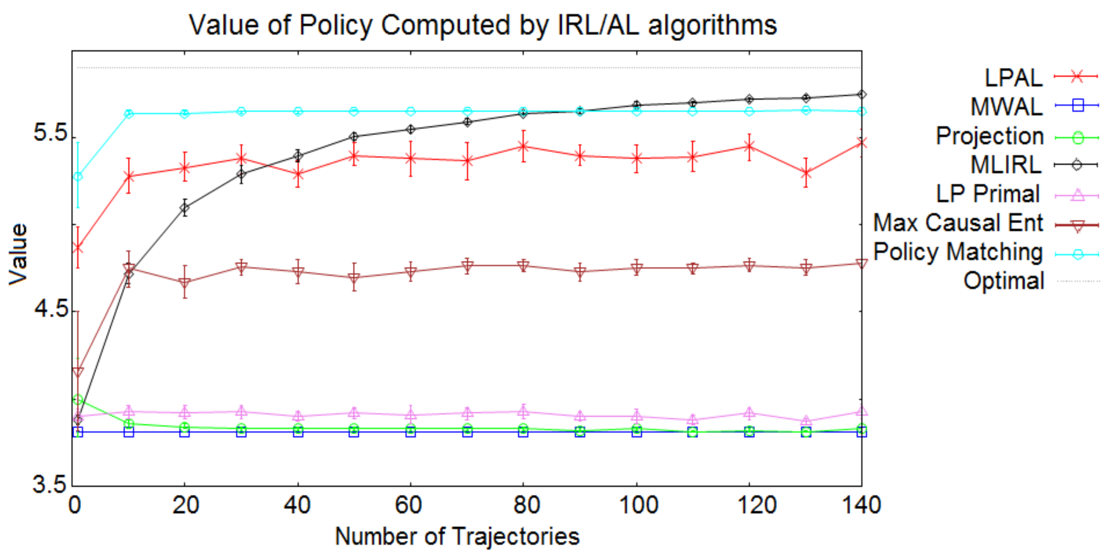

Figure 4.3: A plot of the average reward computed with increasing number of sample trajectories.

random weight vector is chosen such that the true reward function just encodes that some macrocells are more desirable than others. The optimal policy π^* is computed for the true reward function and the single expert trajectories are acquired by sampling π^*. To maintain consistency across the algorithms, the start state is drawn from a fixed distribution and the lengths of the trajectories are truncated to 60 steps. Each algorithm is run for 1000 iterations, except for Max Causal Entropy, which was run for 100 iterations.

Of particular interest is the ability of the seven IRL/AL algorithms to learn from a small amount of data. Thus, I illustrate the performance of the algorithms by varying the number of sample trajectories available for learning. Results are averaged over 5 repetitions and standard error bars are given, corresponding to a 95% confidence interval for the mean. In this and the following experiments, Boltzmann exploration polices (with temperature parameter

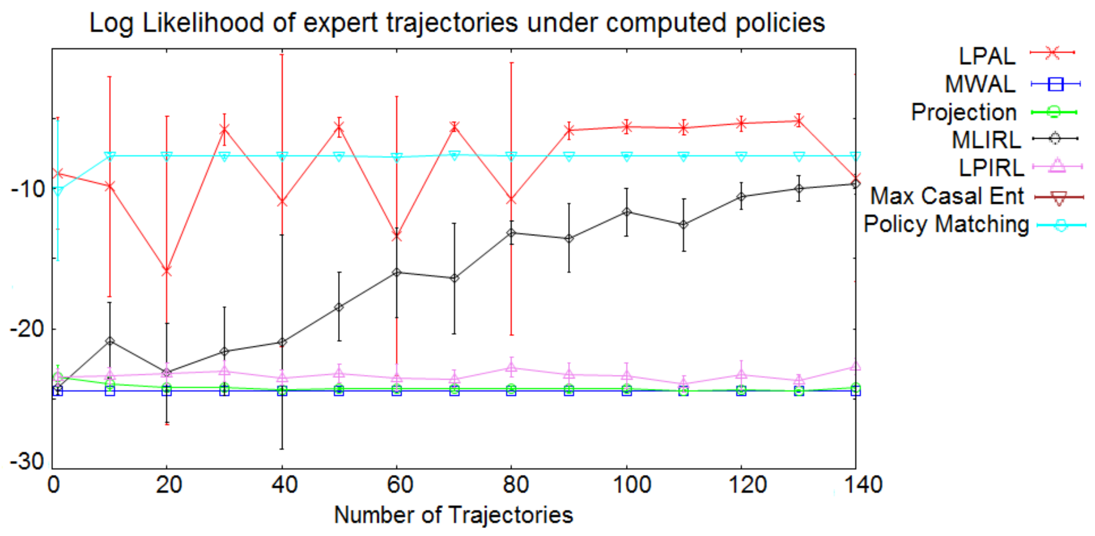

Figure 4.4: A plot of the average trajectory likelihood computed with increasing number of sample trajectories.

$\beta = 0.75$) is used to transform the reward functions computed by the IRL algorithms into policies when required.

Figure 4.3 shows the value of the policy computed by each algorithm as more trajectories are available for training. With enough training data, MLIRL and Policy matching outperform the other six. LPAL also performs well. The reward function computed by Max Causal Entropy assigns the highest weight to a non-goal feature, the one with the second highest weight in the expert's reward function, therefore its policy has a relatively high value.

Figure 4.4 shows that for the most part, in this dataset, the better an algorithm does at assigning high probability to the observed trajectories, the more likely it is to obtain higher rewards. Here, I do not show the logarithm of the probability of the observed trajectories corresponding to Max Causal Entropy, which is around -200. This number is low due to the fact that the feature with the highest weight in the reward computed by Max Causal Entropy is different than the feature with the highest weight in the expert's reward function, and so the actions that Max Causal Entropy wants to take most often are suboptimal.

4.5 Conclusions

In this chapter I have shown that maximum likelihood is a justified approach for inverse reinforcement learning. I have described the MLIRL algorithm, a maximum likelihood approach to IRL in the linear, single expert setting and shown that it is a simple and intuitive gradient method computing an estimate of the expert's reward function that makes the expert demonstrations maximally likely. I used gradient ascent to iteratively change the parameter values in the direction of the gradient of the likelihood of the expert's demonstrations under the current parameter estimate. In the experiment, MLIRL was competitive with 6 existing IRL and AL algorithms in a grid domain.

The fact that MLIRL is a gradient method has the advantage that is it simple and easy to implement. Each iteration is computationally inexpensive, and it does not require a lot of storage. The disadvantages of gradient methods are that they can be very slow , and that the number of iterations necessary might vary with the size of the problem. For the experiment described above, I varied the number of iterations, and chose the number that gave the best results.

Chapter 5
Inverse Reinforcement Learning with Modular Reward Function Learners

In this chapter, I examine a modular approach to inverse reinforcement learning, with a flexible hypothesis class for the reward function. Existing IRL algorithms (see Chapter 3) wed the learning of the mapping from features to reward values (*regression*) to the inference of rewards values from behavior (*intention inference*). By joining these two elements together, the IRL community is effectively disconnected from advances in the broader machine-learning field. Our objective in this chapter is to create an approach to intention inference that can be flexibly combined with arbitrary approaches to regression, building more directly on the existing foundation in the supervised learning community.

A gradient approach is used to hone in on the reward function that maximizes the likelihood of the observed expert behavior with regard to the provided hypothesis class. I compare the performance of the resulting modular IRL algorithm with existing approaches, showing that the approach is a viable alternative to existing IRL methods.

I start by describing the Modular IRL algorithm in Section 5.1, with implementation details in Section 5.2. In Section 5.3 I describe three experiments, showing that the Modular IRL algorithm is able to learn the right reward function where other algorithms fail. I show an example of the algorithm overfitting and underfitting when its hypothesis class has too many or too few parameters in Section 5.4, show a sample complexity analysis in Section 5.5, and conclude in Section 5.6.

5.1 A Modular IRL Algorithm

Our modular IRL framework combines elements of supervised learning with the standard inverse reinforcement learning setup. As in the IRL setup, we are given an MDP$\backslash r$: (S, A, T, γ), a labeling function $L : S \rightarrow X$[1], and information about the expert's optimal policy in the MDP in the form of a set of action selections, $\{s \rightarrow a\}$, also called trajectories or demonstrations. From supervised learning, we add the concept of an input space X, a hypothesis space H containing functions $h : X \rightarrow \mathbb{R}$, and a learning algorithm A that takes a training set of (x, y) pairs, with $x \in X$, and $y \in \mathbb{R}$, and produces a hypothesis $h \in H$ that fits the training data well.

We seek an algorithm that finds the $h \in H$ that assigns the highest probability to the observed transitions. Algorithm 2 outlines the basic steps of our algorithm as it generates a sequence of hypotheses to explain the expert data. Line 2 initializes hypothesis h_0 to be 0 in all states. In the linear IRL setting, this step is equivalent to choosing a set of feature weights that is 0 for each feature. Alternatively, an arbitrary initial h_0 can be chosen, but in that case, the number of iterations used by the modular IRL algorithm may need to be increased, to

[1] The labeling function L is a mapping from states to feature values. $L(s) \in \mathbb{R}^d$, where d is the number of state features. At the same time, for any state $s \in S$, $L(s)$ can be used as input to a classifier or regression module. This notation helps us make the connection with the supervised learning setup.

bring the relative values of the computed rewards close to the values of the expert's rewards (for example, if the expert's reward has a high value in one state and a low value in another, the initial arbitrary values may be reversed, that is a low value for the first state and a high value for the second state, and more iterations will be needed to adjust those values if starting with arbitrary values than if starting with zero values).

During each iteration t, the algorithm uses the current hypothesis h to construct a reward function r_t (Line 4). It then computes an approximation of the corresponding optimal policy $\hat{\pi}_t^*$ by performing a fixed number of steps (K) of value iteration (Line 5). In Line 6, it computes the logarithm of the probability of the expert's demonstrations under π_t^*, then the derivative of the probability (Line 7), and performs one step of gradient ascent (Line 8) to update the reward function r. In Line 9, these rewards are used to label the training examples and update hypothesis h. The Regression module has inputs $\{(x_i, y_i)\}$, where $x_i \in X$, $X = \{x_i = L(s_i) | s_i \in S\}$, and $y_i \in \mathbb{R}$, $y_i = r(s_i)$.

Our method can be seen as a kind of projected gradient algorithm [Rockafellar, 1976], although it uses function approximation instead of projection to map the parameters back into their constrained space.

In practice, the *occupancy measures* of a policy can be used to compute the probability of the expert's demonstrations. The occupancy measures can be computed exactly when the optimal policy is known, or estimated from the expert's demonstrations. I show how these quantities can be computed below.

The occupancy measures of policy π are

$$x^{\pi}_{s',a'} = (p_{s'} + \gamma \Sigma_{s,a} x^{\pi}_{s,a} \cdot T(s,a,s')) \cdot \pi(s',a'), \forall s', a' \tag{5.1}$$

where p_s is the initial state distribution.

More implementation details are given in the following section.

Algorithm 2 Modular IRL.

```
1: **Input:** MDP$\backslash r$ (States $S$, Actions $A$, Transitions $T$, Discount factor $\gamma$), Labeling function $L$, Trajectories $D = \{\tau_k\}$, Number of iterations $N$, Number of forward RL steps $K$, Learning rate $\alpha$, Hypothesis class $H$, Learning algorithm Regression for $H$
2: **Initialize:** Choose $h_0 \in H, h_0 = 0, \forall s \in S$.
3: **for** $t = 1$ **to** $N$ **do**
4:     $r_t(s) \leftarrow h_{t-1}(L(s)), \forall s \in S$.
5:     Use $K$ steps of VI to compute $\hat{\pi}_t^*$ from $r_t$.
6:     Compute $G_t \leftarrow \log(\Pr(D|\hat{\pi}_t^*))$.
7:     Compute $\frac{dG_t}{dr_t(s)}, \forall s \in S$.
8:     $r_t'(s) \leftarrow r_t(s) + \alpha \frac{dG_t}{dr_s}, \forall s \in S$.
9:     $h_t \leftarrow$Regression$(L(s) \rightarrow r_t'(s))$,$\forall s \in S$.
10: **end for**
11: **Output:** $h_N$.
```

5.2 Implementation Details

Given:

- MDP consisting of states S, actions A, transitions T, discount factor γ, initial states distribution p,
- Labeling function $L : S \rightarrow \mathbb{R}^d$, where d is the number of features,
- Trajectories, $D = \{\tau_k\}$, with $|D|$ being the number of trajectories in D,
- Number of iterations N,
- Number of forward RL steps K,
- Learning rate α,
- Hypothesis class H,
- Learning algorithm REGRESSION for H.

Find: Hypothesis $h \in H$ such that $P(D|h)$ is maximized.

Note: Assumption: trajectories are generated by starting with a state according to distribution p, then following π^*, the optimal policy corresponding to the expert's reward r^* (with a Boltzmann distribution for action selection), and continue after each step with probability γ. This choice of termination rule assures that this truncated expected sum of rewards matches the standard exponentially discounted infinite horizon value.

Initialize:

Choose reward function $r_0(s) = 0, \forall s \in S$.

Compute $\hat{x}^{\pi^*}_{s,a} = \frac{(\#(s,a) \in D)}{|D|}$

For each iteration $t = 0 \ldots N$:

$Q_0(s, a) = r_t(s)$.

$V_0(s) = r_t(s)$.

$\frac{d}{dr_j} V_0(s) = 1$, if $j = s$, and 0 otherwise , $\forall j = 1 \ldots |S|$.

For each iteration $i = 1 \ldots K$:

$\forall s \in S, a \in A, Q_i(s, a) = r_t(s) + \gamma \sum_{s'} T(s, a, s') V_{i-1}(s')$.

$$\frac{dQ_i(s,a)}{dr_j} = \{1 \text{ if } s{==}j, 0 \text{ otherwise}\} + \gamma \sum_{s'} T(s,a,s') \frac{dV_{i-1}(s')}{dr_j}.$$

$$Z_i(s) = \sum_a e^{\beta Q_i(s,a)}.$$

$$\frac{d}{dr_j} Z_i(s) = \beta \sum_a e^{\beta Q_i(s,a)} \frac{d}{dr_j} Q_i(s,a)$$

$$\pi_i(s,a) = \frac{e^{\beta Q_i(s,a)}}{Z_i(s)}.$$

$$\frac{d}{dr_j} \pi_i(s,a) = \frac{\beta Z_i(s) e^{\beta Q_i(s,a)} \frac{d}{dr_j} Q_i(s,a) - e^{\beta Q_i(s,a)} \frac{d}{dr_j} Z_i(s)}{Z_i^2(s)}$$

$$V_i(s) = \sum_a \pi_i(s,a) Q_i(s,a).$$

$$\frac{d}{dr_j} V_i(s) = \sum_a (Q_i(s,a) \frac{d}{dr_j} \pi_i(s,a) + \pi_i(s,a) \frac{d}{dr_j} Q_i(s,a)).$$

$$P_i = Pr(D|\pi_i) = \prod_{\tau \in D} Pr(\tau|\pi_i) = \prod_{\tau \in D} p_{s_0}^{\tau} \cdot \prod_{(s,a) \in \tau} \pi_i(s,a) \cdot \prod_{(s,a,s') \in \tau} T(s,a,s').$$

$$\begin{aligned} G_i &= \log P_i = \log(\prod_{\tau \in D} Pr(\tau|\pi_i)) \\ &= \log(\prod_{\tau \in D} (p^{\tau}_{s_0} \prod_{(s,a)\in\tau} \pi_i(s,a) \cdot \prod_{(s,a,s')\in\tau} T(s,a,s'))) \\ &= \sum_{\tau\in D} (\log(p^{\tau}_{s_0}) + \sum_{(s,a)\in\tau} \log(\pi_i(s,a)) + \sum_{(s,a,s')\in\tau} \log(T(s,a,s'))) \\ &= \sum_{\tau\in D} (\log(p^{\tau}_{s_0}) + \sum_{(s,a,s')\in\tau} \log(T(s,a,s'))) + \sum_{\tau\in D} \sum_{(s,a)\in\tau} \log(\pi_i(s,a)) \\ &= \sum_{\tau\in D} (\log(p^{\tau}_{s_0}) + \sum_{(s,a,s')\in\tau} \log(T(s,a,s'))) + \sum_{(s,a)\in S\times A} (\#(s,a)\in D) \cdot \log(\pi_i(s,a)) \\ &= \sum_{\tau\in D} (\log(p^{\tau}_{s_0}) + \sum_{(s,a,s')\in\tau} \log(T(s,a,s'))) + |D| \cdot \sum_{s,a} \hat{x}^{\pi^*}_{s,a} \cdot \log(\pi_i(s,a)) \end{aligned} \tag{5.2}$$

$$\frac{dG_i}{dr_j} = |D| \sum_{s,a} \hat{x}^{\pi^*}_{s,a} \frac{d}{dr_j}(\log(\pi_i(s,a)) = |D| \sum_{s,a} \hat{x}^{\pi^*}_{s,a} \frac{d\pi_i(s,a)}{dr_j} \frac{1}{\pi_i(s,a)}. \tag{5.3}$$

$\frac{d}{dr_j}G = \frac{d}{dr_j}G_i$.

End For each iteration i.

$r'_t(s) = r_{t-1}(s) + \alpha_t \frac{d}{dr_j}G, \forall s \in S, j = 1 \ldots |S|$.

$h =$REGRESSION$(L(s) \to r'_t(s)), \forall s \in S$.

$r_{t+1}(s) = h(f(s)), \forall s \in S.$

End For each iteration t.

Output: h.

To show that the gradient is a reasonable tool in obtaining optimal rewards, I make the following claim:

Theorem 1. *The expected gradient starting from the optimal policy is zero.*

Proof. When the optimal policy is known, one can compute the true occupancy measures $x^*_{s,a}$:

$$x^{\pi^*}_{s',a'} = (p_{s'} + \gamma \sum_{s,a} x^{\pi^*}_{s,a} \cdot T(s,a,s')) \cdot \pi^*(s',a'), \forall s,a \tag{5.4}$$

where $\pi^*(s,a) = \frac{e^{\beta \cdot Q(s,a)}}{\sum_{a'} e^{\beta \cdot Q(s,a')}}$, and Q is the Q-function computed using Value Iteration in the MDP (S, A, T, γ, r^*), where r^* is the expert's reward.

Equation 5.3 becomes:

$$\frac{dG_i}{dr_j} = |D| \sum_{s,a} \frac{x^*_{s,a}}{\pi_i(s,a)} \frac{d\pi_i(s,a)}{dr_j} = |D| \sum_{s} (\sum_{a} \frac{x^*_{s,a}}{\pi_i(s,a)} \frac{d\pi_i(s,a)}{dr_j}). \tag{5.5}$$

When $\pi_i = \pi^*$,

$$\frac{x^*_{s,a}}{\pi_i(s,a)} = \frac{x^*_{s,a}}{\pi^*(s,a)} = \frac{(p_s + \gamma \sum_{s',a'} x^*_{s',a'} T(s',a',s)) \pi^*(s,a)}{\pi^*(s,a)} = x^*_s. \tag{5.6}$$

Therefore,

$$\sum_{a} \frac{x^*_{s,a}}{\pi_i(s,a)} \frac{d\pi_i(s,a)}{dr_j} = \sum_{a} x^*_s \frac{d\pi^*(s,a)}{dr_j} = x^*_s \sum_{a} \frac{d\pi^*(s,a)}{dr_j}, \text{when } \pi = \pi^*.$$

Because for any fixed state s,$\pi(s,a)$ is a probability distribution over $a \in A$, we have that $\sum_a \pi(s,a) = 1$, therefore, $\sum_a \frac{d\pi(s,a)}{dr_j} = 0$.
So,

$$\sum_a \frac{x^*_{s,a}}{\pi^*(s,a)} \frac{d\pi^*(s,a)}{dr_j} = x^*_s \cdot 0 = 0. \tag{5.7}$$

Equation 5.5 becomes:

$$\begin{aligned}
\frac{dG}{dr_j} &= |D| \sum_{s,a} \frac{x^*_{s,a}}{\pi^*(s,a)} \frac{d\pi^*(s,a)}{dr_j} \\
&= |D| \sum_s (\sum_a \frac{x^*_{s,a}}{\pi^*(s,a)} \frac{d\pi^*(s,a)}{dr_j}) && (5.8) \\
&= |D| \sum_s (\sum_a x^*_s \frac{d\pi^*(s,a)}{dr_j}) && (5.9) \\
&= |D| \sum_s x^*_s (\sum_a \frac{d\pi^*(s,a)}{dr_j}) && (5.10) \\
&= |D| \sum_s x^*_s \cdot 0 && (5.11) \\
&= |D| \sum_s 0 = 0. && (5.12)
\end{aligned}$$

The gradient at the optimal policy is 0, and is therefore a fixed point of the update equation. □

5.3 Experiments

I implemented Algorithm 2 in Java, with the Regression module using off-the-shelf regression modules from the Weka Application Programming Interface (API) [Hall et al., 2009]. I performed IRL experiments with linear and polynomial regression, multilayer perceptrons, regression trees, and Gaussian process regression in several illustrative environments.

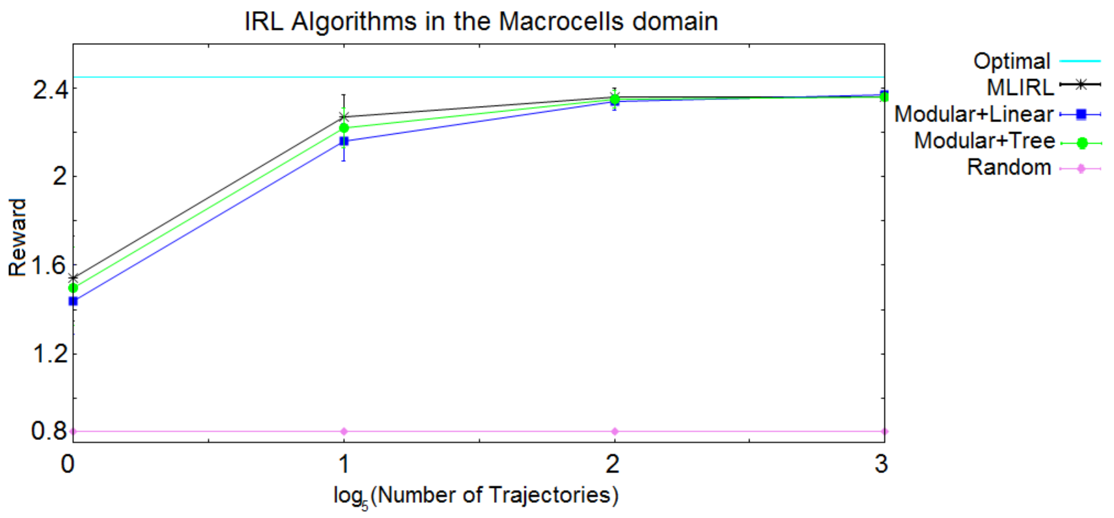

Figure 5.1: The performance of MLIRL, modular+linear and modular+tree in the Grid World with Macrocells.

5.3.1 Modular IRL is Competitive with Linear IRL

Our first experiment evaluates the performance of the modular IRL framework in the setting closest to most existing work—using a linear hypothesis space. I wish to demonstrate that the resulting algorithm is competitive with existing algorithms that use a linear representation. I use the MLIRL algorithm of Chapter 4 as the representative of this class of algorithms, as it performs on par with the best existing algorithms (Section 4.4). I used the original "macrocell" grid-world domain [Abbeel and Ng, 2004] as a testbed, with an 8×8 grid divided into 16 2×2 non-overlapping macrocells. There is one state per grid location, and each state is represented by a set of 16 features, one feature per macrocell. The feature values for a state are 0 if the state's location does not belong to the corresponding macrocell, and 1 if it does. There are 4 available actions to the agent, allowing it to move in the grid north, south, east and west. Each action has a 70% probability of succeeding and a 30% probability of moving the agent randomly according to one of the other actions.

The expert's reward function is assumed to be a linear combination of the features. To measure the performance of the algorithms, an arbitrary expert reward function is built by assigning to each feature a weight that is 0 with probability 90% and a small positive number (in the interval $[0, 1]$) with probability 10%. If there are fewer than two features with non-zero weights, the reward function is discarded, and a new one is generated. After the expert's reward is generated, its optimal policy is computed, and this policy is used to generate the expert's demonstrations or trajectories, which are given as input to the IRL algorithms. Each of the IRL algorithms computes its estimate of the expert's reward function and this estimate is used to compute the optimal Q-function using value iteration. The estimate of the optimal policy is the Boltzmann distribution using the optimal q-function with respect to the estimated reward and a temperature parameter of $\beta = 2$. I generated trajectories according to this distribution, and trajectories are terminated with probability $1 - \gamma$ at each step. The performance of the IRL algorithms is evaluated using the average of discounted reward for this set of trajectories. More specifically, each trajectory receives a score equal to the sum of rewards for the states in the trajectory. The overall performance of the algorithm is the average of the scores of its trajectories.

Each IRL algorithm was run for 1000 iterations with K (the number of forward RL steps, see 2) set to 15. The discount factor was 0.9, and the learning rate was set to 0.1. To evaluate the algorithms, $10,000$ trajectories were used, generated according to the Boltzmann distribution for action selection corresponding to the reward function computed by each algorithm. The generated trajectories were ended at each step with probability $1 - \gamma$, with an average trajectory length of 10. The starting state for each trajectory was chosen uniformly among all the states. For the modular IRL with linear regression learner (modular+linear), I used the Weka `LinearRegression` class. I also ran modular IRL with a decision-tree learner (modular+tree), using the Weka `REPTree` class as the regression module.

The IRL algorithms were trained using increasing numbers of training trajectories (Figure 5.1), and measured their performance over 100 runs. All three algorithms were able to learn the optimal behavior. Consistently with previous results, MLIRL approached optimal performance with increasing data. The modular algorithm was also able to do equally well with either a linear regression algorithm or a decision-tree regression algorithm to learn the reward function.

Note that modular+linear runs about 3 times more slowly than MLIRL; MLIRL takes 40 seconds while modular+linear takes 120 seconds on a Intel Core with a 3.2 Gigahertz processor.

5.3.2 Modular IRL Can Solve Non-linear Problems

The advantage of a modular approach over classic IRL algorithms is that it can be used in more general settings. In many cases, a linear IRL algorithm is adequate even in the face of a non-linear reward function—the best linear fit can produce the right behavior even though the rewards themselves are non-linear. However, I devised a simple environment that forces any linear-based IRL algorithm to produce suboptimal behavior. I find that IRL algorithms with non-linear hypothesis classes are able to learn optimal behavior in this example.

Our testing domain is the small MDP shown in Figure 5.2. This domain has

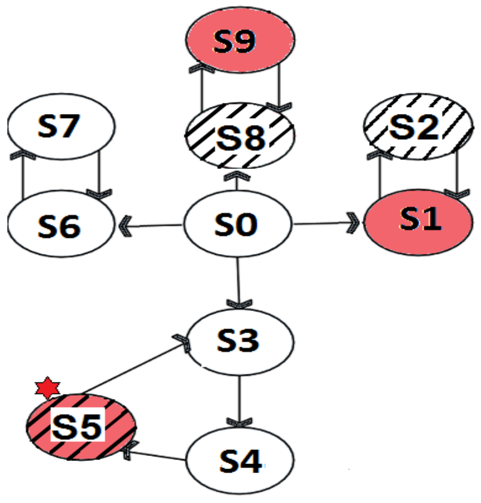

Figure 5.2: The "AND" MDP.

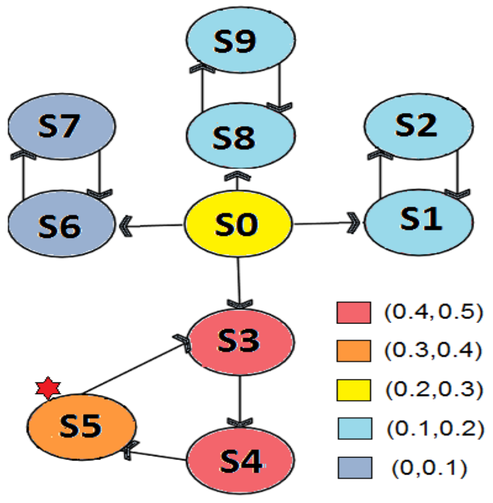

Figure 5.3: The initial updates in reward values suggested by the gradient.

10 states. There are 4 actions available to the agent in State S0, and one action available in every other state. Each action has a 90% probability of succeeding and a 10% probability of taking the agent back to State S0. The states are described by two features: *red* and *striped*, and the expert's reward function is $r(s) = red(s)$ AND $striped(s)$.

This MDP is designed to thwart a linear representation. Any linear representation of reward will prefer the sub-optimal action (going right, left, or up from State S0) to the optimal action (going down from State S0). To see why, let x and y be the weights learned for the two features. In State S0, the algorithm will assign returns $\frac{\gamma x+\gamma^2 y}{1-\gamma^2}$ to the right action, $\frac{\gamma y+\gamma^2 x}{1-\gamma^2}$ to the up action, 0 to the left action, and $\frac{\gamma^3(x+y)}{1-\gamma^3}$ to the down (optimal) action

For the down action to have the highest return, the following inequalities have to be true at the same time for $\gamma = 0.9$: $y < -1.3011x$, $y < 0.09x$, and $y > -x$, which is impossible for $x, y \in \mathbb{R}$. Therefore, any linear algorithm will make sub-optimal choices in this domain.

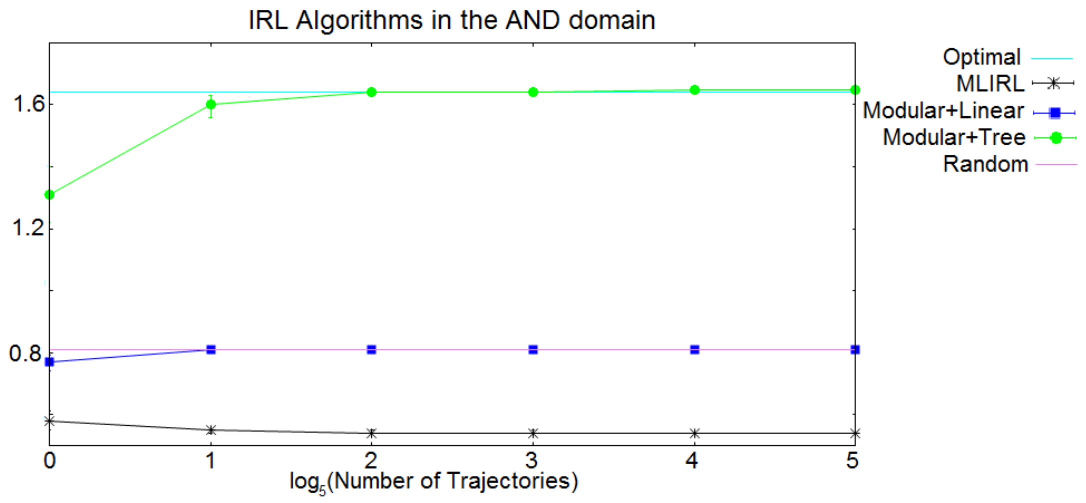

Figure 5.4: The performance of MLIRL, modular+linear and modular+tree in the "AND" MDP. The expert reward function is *red* AND *striped*.

To evaluate performance on this MDP, I again generated training trajectories for MLIRL, modular+linear, and modular+tree. The number of training trajectories was 1, 5, 25, 125, and 625, each trajectory generated using the Boltzmann distribution, and ending with probability $1 - \gamma$ at each step, with starting states chosen uniformly among all the states of the MDP. The algorithms computed their estimates of the expert's reward function, and each of the estimates was used to generate $10,000$ trajectories (again using the Boltzmann distribution), which were scored the same way as mentioned in the previous section. Then, the scores were averaged to compute the reward of each algorithm. I show these results in Figure 5.4.

We see that MLIRL, which uses a linear representation of reward, is clearly suboptimal here. MLIRL receives a small reward due to the fact that the generated trajectories start in arbitrary states including states that lead to the goal, but it always chooses the wrong action in State 0, therefore its performance is worse than that of the random algorithm, since random chooses the right

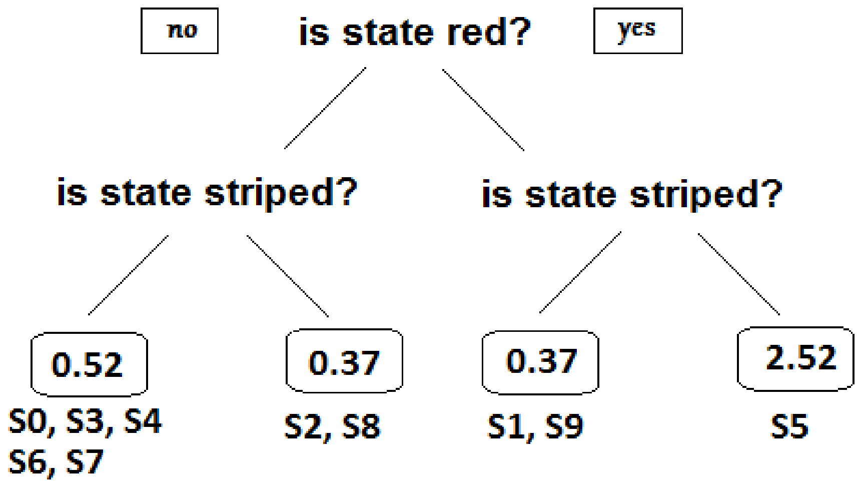

Figure 5.5: The tree computed by modular+tree, estimating the expert's reward function $r(s) = red(s)$ AND $striped(s)$.

action more often than MLIRL. The modular+linear algorithm is also unable to achieve high reward on this problem. The hypothesis it computes assigns a weight of 0 for the two features, therefore it chooses randomly where to go from State 0, and thus obtains the same performance as an algorithm that always chooses its actions randomly.

In contrast, even with a small number of training trajectories, modular+tree is able to learn a reward function that enables it to act optimally. It consistently learns to assign the highest reward to states where both features *red* and *striped* are true, thus drawing the agent to the goal state. I show the learned decision tree in Figure 5.5.

The white states get a relatively high reward due to the fact that States 4 and 5 are chosen often on the way to the goal, therefore the gradient is trying to increase their values (see Figure 5.3).

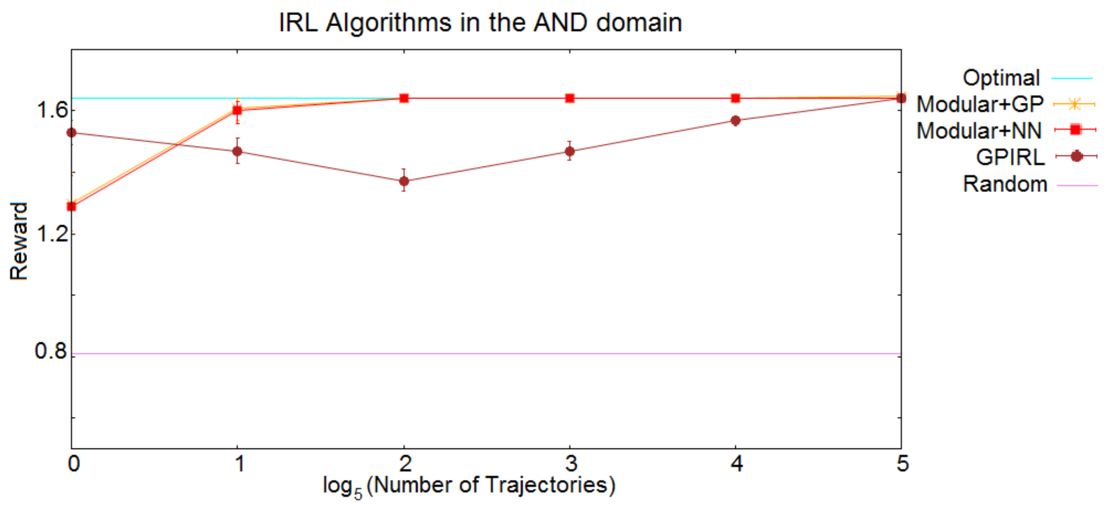

Figure 5.6: The performance of GPIRL, modular+GP and modular+NN in the "AND" MDP. The expert reward function is $r(s) = red(s)$ AND $striped(s)$

.

Although this example shows that the vast majority of existing IRL algorithms—those based on linear representations—can be thwarted, there are existing algorithms that are successful. I evaluated Gaussian Process IRL [Levine et al., 2011] on this same task, comparing its performance to Gaussian Process Regression Modular IRL (modular+GP) and Multilayer Perceptron Modular IRL (modular+NN). I used Weka's `GaussianProcesses` class as the regression module for the modular+GP learner and the `MultilayerPerceptron` class for the modular+NN learner.

Figure 5.6 shows that all three of these algorithms are successful at learning the non-linear reward function for this task.

In Figure 5.3, I show the updates in values that the gradient suggests during the first iteration. Trajectories from an expert trying to reach goal State 5 will choose to go down from State 0. Since the gradient is relative to state rewards, and not features, to increase the likelihood of the demonstrations, one has to increase the value of State 3 the most. Since the gradient increases the value of

State 3 and State 4 by a much larder quantify than the decrease in the value of State 7, the linear algorithm is led to believe that white states are desirable, and therefore, red states should be avoided.

5.3.3 Modular IRL Can Solve Even Harder Problems

In the "AND" MDP from the previous section, GPIRL was able to produce optimal behavior in spite of the non-linear nature of the reward function. In this section, consider a problem that thwarts GPIRL but that the modular approach can still solve given an appropriate hypothesis class.

The experimental setup is a 4×3 grid (Figure 5.7) with an agent and a movable block. There is one state for each combination of agent and block location. The features associated with the states are the agent's location in each room (3 features, one for each of the rooms), and the block's location in each room (another 3 features, one for each of the rooms). For example, the feature *Agent-in-Red-Room* is true for states in which the location of the agent is the red room. The transitions are deterministic, and the dynamics are similar to those used in the game of Sokoban [Junghanns and Schaeffer, 2001]—the agent can push the block if the agent is next to the block and the block is in the direction the agent is moving and there is no obstacle (for example, a wall) next to the block in the direction that the agent is trying to move.

To measure the performance of the IRL algorithms, I first generate trajectories following the optimal policy under the expert's reward. The expert reward is $r(s) =$*Agent-in-Red-Room*(s) XOR *Block-in-Blue-Room*(s) (the state shown in the figure satisfies this goal condition). The performance of each algorithm is

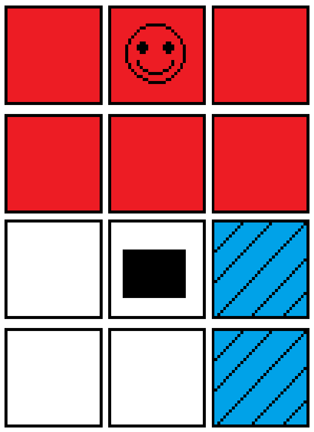

Figure 5.7: The “XOR” MDP (the blue squares are shaded).

measured over 100 runs. For each run, a number of expert trajectories is generated according to the corresponding training size (1,5,5^2,5^3, 5^4, and 5^5 trajectories). These are given as input to modular+GP, modular+NN, and GPIRL. Each algorithm outputs its estimate of the expert reward function. The optimal policy is computed for each of these estimates (using the Boltzmann distribution) and is used to generate $10,000$ trajectories. Each trajectory is evaluated by computing the sum of rewards over the trajectory. For each algorithm, its score is computed by averaging the score for its $10,000$ trajectories. Each trajectory ends with probability $1-\gamma$ at each step.

I show the performance of the three algorithms in Figure 5.8. Both the modular+NN and the modular+GP algorithms are able to learn the true reward function and obtain optimal performance, the GPIRL algorithm is not, and it is unable to lift itself up above random.

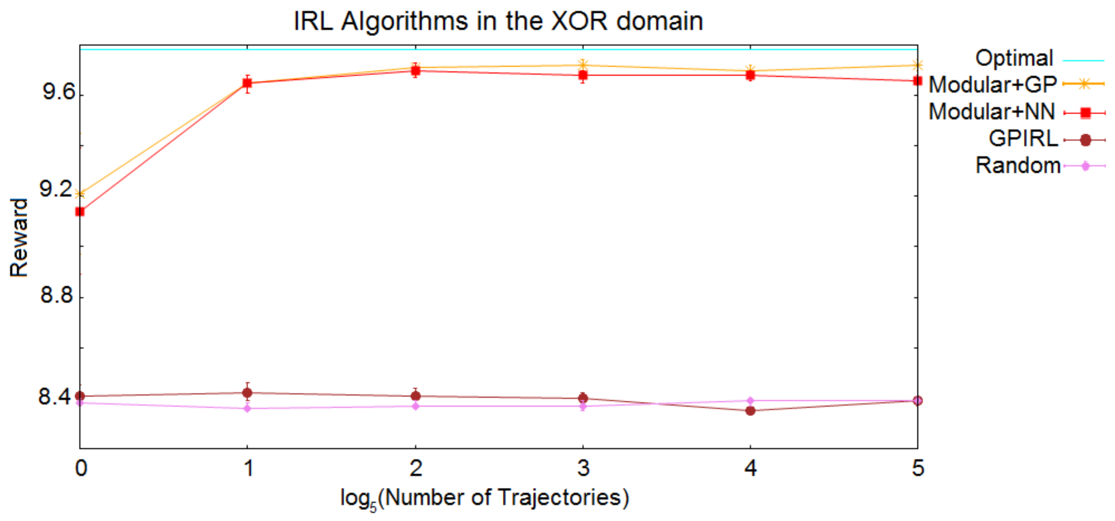

Figure 5.8: The performance of GPIRL, modular+GP, and modular+NN IRL approaches in the "XOR" MDP. The expert reward function is *Agent-in-Room-1* XOR *Block-in-Room-3*.

5.4 Issues of overfitting and underfitting

As mentioned in the beginning of this chapter, the modular IRL algorithm provides a framework that is flexible enough to plug in a variety of supervised learning algorithms, depending on the problem we are trying to solve. The framework leverages new research in regression that other frameworks cannot incorporate. Integrating a supervised learning component raises questions like how does one choose the right hypothesis class (or model) for the function that is being learned? Models with too few parameters will output a function that is too simplistic, and the prediction error on new data will be too large, an issue called *underfitting* [Mitchell, 1997]. Models with too many parameters, will fit the training data well, but their output will be an overcomplicated function that will not generalize well on new data, or *overfit*.

Existing IRL algorithms use only one function class to fit the expert's reward function, and it is hard to tell if that function class overfits or underfits.

Indeed, in most existing experiments, the hypothesis class is chosen to be precisely correct. With the Modular IRL framework, the function class is flexible, so finding the right fit becomes a challenge. Another issue in existing IRL work is that the algorithms are tested in the same MDP used to generate the training data, and the ability of the algorithm to generalize is often overlooked. I noticed this point when testing the modular IRL algorithm with different hypothesis classes in the same MDP where the expert demonstrations were generated, and noticed no overfitting or underfitting. To highlight the importance of this issue, I changed the Modular IRL algorithm used in this setup to allow for training data consisting of pairs of MDPs and trajectories. The testing was done in MDPs that were different than the training MDPs, but all the MDPs shared the same feature set. The setup is described in detail below.

To illustrate overfitting and underfitting, I designed an experiment where the expert's reward function is a linear combination of a given set of features. I used four models of increasing complexity to learn the target function, and tested their performance with increasing training data.

3×3 grids were used with three features: a *goal* feature that is 1 in one of the states chosen arbitrarily and 0 in all the other states, and so-called *distraction* features that are 1 in approximately half of the states (arbitrarily) and 0 in all the other states. Algorithm 3 outlines the experiment.

Our feature set contains the 3 features described above. The learners used polynomial regression with different degree polynomials and therefore different number of parameters (features). I tested the following algorithms: the *constant* learner uses a model with a constant reward in all the states, the *linear* learner finds a linear function of the feature set, the *quadratic* learner also

Algorithm 3 Overfitting/underfitting experiment setup.

```
1: Input: Size of grid $g$, Number of features $nF$, Number of training examples $nT$, Number of testing trajectories $tT$, Boltzmann temperature $\beta$ , Number of iterations $N$, Number of forward RL steps $K$, Learning rate $\alpha$, Learning algorithm Regression for $H$
2: Initialize: Training set $T \leftarrow \emptyset$.
3: for $t = 1$ to $nT$ do
4:     Initialize feature set, $F \leftarrow \emptyset$
5:     choose an arbitrary state as a goal state, and set feature $f_1$ accordingly
6:     choose $nF - 1$ arbitrary binary features, $f_2 \ldots f_{nF}$
7:     $F \leftarrow f_1 \ldots f_{nF}$
8:     Build grid $G_t$ of size $g \times g$ using $F$
9:     Build expert reward $r(s) = f_1(s)$
10:    Compute optimal policy for $r$, and generate 1 expert trajectory $\xi_t$.
11:    Add $(G_t, \xi_t)$ to training set $T$
12: end for
13: Classifier $\leftarrow$ ModularIRL($G$,$T$,$N$,$K$,$\alpha$,$H$)
14: $sum \leftarrow 0$
15: for $gt = 1$ to 10 do
16:    Initialize feature set, $F \leftarrow \emptyset$
17:    choose an arbitrary state as a goal state, and set feature $f_1$ accordingly
18:    choose $nF - 1$ arbitrary binary features, $f_2 \ldots f_{nF}$
19:    $F \leftarrow f_1 \ldots f_{nF}$
20:    Build grid $G_t$ of size $g \times g$ using $F$
21:    $r'(s) \leftarrow$ Classifier($s$), $\forall s$, states of $G_t$.
22:    trajectories$\leftarrow$generate($r'$,$nT$)
23:    $sum \leftarrow sum$ + reward(trajectories)
24: end for
25: Output: Average performance $sum/10$.
```

learns a linear function of the pair-wise products of the feature set (for example, $f_1 \cdot f_1, f_1 \cdot f_2, \ldots$), and the *cubic* learner uses as features the set of all possible 3-term products of the feature set (for example $f_1 \cdot f_1 \cdot f_1, f_1 \cdot f_1 \cdot f_2, \ldots$). In Algorithm 3, these models are called the "learning algorithm", and their hypotheses are used as input to the Modular IRL Algorithm 5.

Algorithm 3 receives as input the size of the grids g, which in our setup is 3, the number of features nF is also 3, the number of training examples, which I increased for each run: 1, 5, 10, 15 and 20, the number of testing trajectories

tT (10, 000) used to compute the performance of the learners, the inverse Boltzmann temperature (3), and a few other parameters used by the modular IRL algorithm: the number of iterations (5, 000), the number of forward RL steps (6), the learning rate for the gradient update (0.1), and the hypothesis class H, which is the feature set used by each learner (for example, the linear feature set for the linear learner, the quadratic feature set for the quadratic learner, etc).

I first build a training set T (lines 3–12) by generating nT arbitrary 3×3 grids (lines 4-8), and an optimal trajectory for each grid (lines 9–10). Each training example is a pair (G, ξ), where G is a grid, and ξ is an optimal trajectory in G. The trajectory is generated (line 10) by computing the optimal policy for the expert's reward, r, then using the corresponding Boltzmann distribution with temperature β as the policy in each state (Equation 2.1). Each grid G is represented as an MDP (S, A, T, γ) as in Section 5.

To make the connection with the previous sections clearer, in the following, I refer to the grid in each training instance as an MDP. I made a slight change to the modular IRL algorithm so it can take as input multiple pairs (MDP, ξ), a Markov decision problem representing a grid G, and an optimal trajectory in the MDP, ξ, all with the same features and actions sets (line 13). I tested the classifier on 10 arbitrary grids (lines 15–24) using the same feature set (lines 16–20). The testing data for the classifier are the states of each testing grid, the classifier providing the reward for each of these states as its prediction (line 21). This reward is used to generate trajectories in the grid, again using the Boltzmann distribution as the estimate of the optimal policy (line 22). The performance of the algorithm in each testing grid is computed as the average reward over these trajectories (line 23). The overall performance of the learner is the average over its performance on all 10 testing grids (line 25).

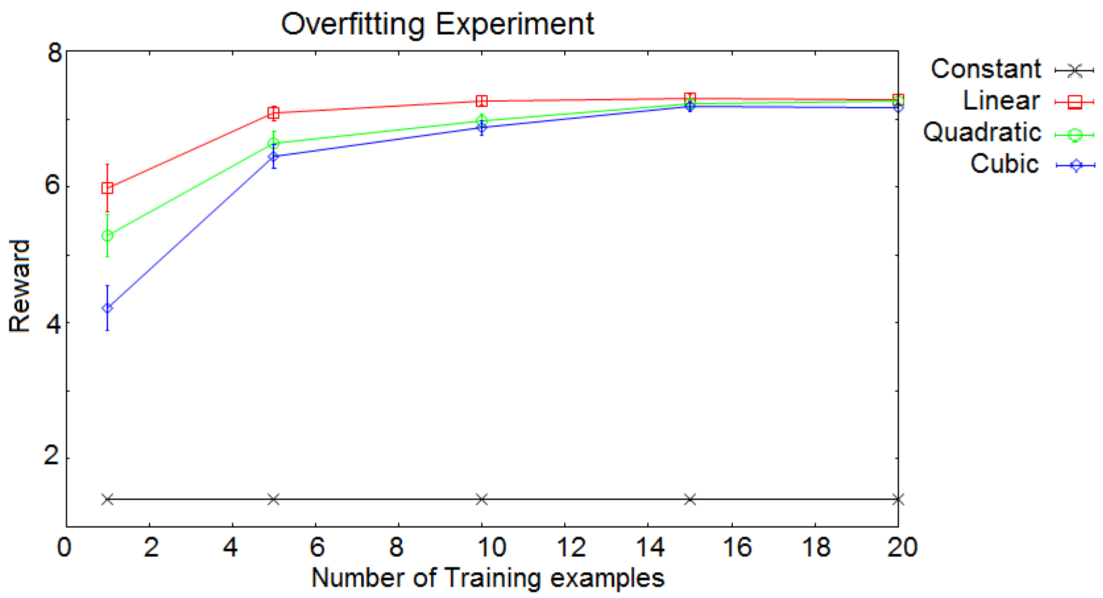

Figure 5.9: The performance of different degree learners in 3×3 grids

In Figure 5.9, I plot the reward obtained by the four learners with increasing amounts of training data. As expected, the *constant* model is not able to learn a usable reward function, because its best explanation for the observed trajectories is that the reward is 0 in all the states; therefore, its trajectories miss the goal state. The *linear*, *quadratic*, and *cubic* learners can all model the linear reward function, and are therefore able to learn the target function with enough data, but the simpler hypotheses learn the function with less data. More specifically, the linear algorithm obtains optimal performance with just 5 training examples, whereas the quadratic and cubic algorithms need at least 25 training examples to attain optimal performance.

5.5 Sample complexity analysis

Our sample complexity analysis closely follows that of Abbeel and Ng [2004]. The following theorem states our sample complexity claim.

Theorem 2. *Let an MDP (S, A, T, γ), a labeling function $L : S \to \mathbb{R}^d$ (where d in the number of state features), and any $\epsilon > 0$, $\delta > 0$ be given. Let r^* be the true expert reward, π_E, the corresponding optimal policy estimate using the Boltzmann distribution for action selection, m, the number of demonstrations generated following π_E. If the MDP and trajectories are given as input to the modular IRL algorithm, let h be the function output by the algorithm and $\tilde{\pi}$ the Boltzmann policy corresponding to reward function $r(s) = h(L(s)), \forall s \in S$. To ensure that for any expert reward $r^*(s)$ we have*

$V^{\tilde{\pi}} \geq V^{\pi_E} - \epsilon,$

it suffices that

$m \geq \frac{9\|S\|\|A\|}{2(1-\gamma)^2 \cdot \epsilon^2} \log \frac{\delta}{2\|S\|\|A\|}.$

Proof. The definition of the occupancy measure for state s and action a is :

$$x^{\pi}_{s,a} = \mathbb{E}[\Sigma^{\infty}_{t=0}\gamma^t \cdot \mathbb{1}_{s_t = s \wedge a_t = a} | \pi]$$

m samples or trajectories are used to estimate x, the vector of all occupancy measures for the expert's policy. We denote this estimate by $\hat{x}$. Our goal is to find the number of samples m for which the estimate $\hat{x}$ is ϵ-close with probability $1 - \delta$ from x. Formally, we need to find m, such that

$$Pr(\|x - \hat{x}\| \leq \epsilon) \geq 1 - \delta$$

.

From this definition, $x^{\pi}_{s,a} \geq 0$; that is, in the worst case, policy π never visits state action pair (s, a), and $x^{\pi}_{s,a} \leq 1 + \gamma + \gamma^2 + \gamma^3 + \ldots = \frac{1}{1-\gamma}$, if at each step policy π visits state action pair (s, a). Therefore, we can write $x^{\pi}_{s,a} \in [0, \frac{1}{1-\gamma}]$, where γ is the discount factor used in the MDP.

Since each $x^{\pi}_{s,a}$ is bounded, $(1-\gamma)x^{\pi}_{s,a}$ is also bounded, and we can apply the Hoeffding's inequality for the m-sample estimate $(1-\gamma)\hat{x}^{\pi}_{s,a}$ of $(1-\gamma)x^{\pi}_{s,a}$:

$$Pr((1-\gamma)|x^{\pi}_{s,a} - \hat{x}^{\pi}_{s,a}| > \tau) \leq 2 \cdot exp(-2\tau^2 m). \tag{5.13}$$

Applying the union bound for the probabilities given by Equation 5.13 for all the components of x, $x^{\pi}_{s,a}$, $\forall (s, a) \in S \times A$, we get

$$Pr(\exists (s,a) \in S \times A.(1-\gamma)|x^{\pi}_{s,a} - \hat{x}^{\pi}_{s,a}| > \tau) \leq 2 \cdot |S||A| \cdot exp(-2\tau^2 m). \tag{5.14}$$

We subtract both sides of Equation 5.14 from 1, and obtain

$$1 - Pr(\exists (s,a) \in S \times A.(1-\gamma)|x^{\pi}_{s,a} - \hat{x}^{\pi}_{s,a}| > \tau) \geq 1 - 2 \cdot |S||A| \cdot exp(-2\tau^2 m). \tag{5.15}$$

$$Pr(\neg\exists (s,a) \in S \times A.(1-\gamma)|x^{\pi}_{s,a} - \hat{x}^{\pi}_{s,a}| > \tau) \geq 1 - 2 \cdot |S||A| \cdot exp(-2\tau^2 m). \tag{5.16}$$

If none of the components of x is greater than $\frac{\tau}{1-\gamma}$, then the component with the maximum value will also be less than $\frac{\tau}{1-\gamma}$, therefore

$$Pr((1-\gamma)\left\|x^{\pi}_{s,a} - \hat{x}^{\pi}_{s,a}\right\|_{\infty} \leq \tau) \geq 1 - 2 \cdot |S||A| \cdot exp(-2\tau^2 m). \tag{5.17}$$

I substitute τ for $\frac{(1-\gamma)\epsilon}{3\sqrt{|S||A|}}$

$$Pr((1-\gamma)\left\|x^{\pi}_{s,a} - \hat{x}^{\pi}_{s,a}\right\|_{\infty} \leq \frac{(1-\gamma)\epsilon}{3\sqrt{|S||A|}}) \geq 1 - 2 \cdot |S||A| \cdot exp(-2(\frac{(1-\gamma)\epsilon}{3\sqrt{|S||A|}})^2 m) \tag{5.18}$$

or

$$Pr(\left\|x^{\pi}_{s,a} - \hat{x}^{\pi}_{s,a}\right\|_{\infty} \leq \frac{\epsilon}{3\sqrt{\|S\|\|A\|}}) \geq 1 - 2 \cdot \|S\|\|A\| \cdot exp(-2(\frac{(1-\gamma)\epsilon}{3\sqrt{\|S\|\|A\|}})^2 m). \tag{5.19}$$

Above, I simplified the quantity under the probability on the left hand side by $(1-\gamma)$.

For $m \geq \frac{9\|S\|\|A\|}{2(1-\gamma)^2 \cdot \epsilon^2} \log \frac{\delta}{2\|S\|\|A\|}$, we get

$$Pr(\left\|x^{\pi}_{s,a} - \hat{x}^{\pi}_{s,a}\right\|_{\infty} \leq \frac{\epsilon}{3\sqrt{\|S\|\|A\|}}) \geq 1 - \delta. \tag{5.20}$$

Since $\|.\|_2 \leq \sqrt{|S||A|}\|.\|_{\infty}$ in $|S||A|$-dimensional spaces,

$$\left\|x^{\pi}_{s,a} - \hat{x}^{\pi}_{s,a}\right\|_2 \leq \frac{\epsilon}{3} \tag{5.21}$$

with probability at least $1 - \delta$.

So far, I have shown that if the number of samples m used to estimate the occupancy measures of the expert's policy is greater or equal to $\frac{9\|S\|\|A\|}{2(1-\gamma)^2 \cdot \epsilon^2} \log \frac{\delta}{2\|S\|\|A\|}$, then the distance between the occupancy measures of the expert's policy and its estimate (using the m samples) is smaller than ϵ with probability at least $1 - \delta$, where ϵ and δ are parameters chosen ahead of time. I still need to show that if our estimate of the occupancy measures for the expert's policy is ϵ-close to the actual occupancy measures, then the values of the policies are also close (this is an analogue of the simulation lemma [Kearns and Singh, 2002])

The definition of the value of a policy is:

$$V^{\pi} = \mathbb{E}[\Sigma_{t=0}^{\infty}\gamma^t R(s_t, a_t)|\pi].$$

The occupancy measure for state, action pair (s, a) under policy π is:

$$x_{s,a}^{\pi} = \mathbb{E}[\Sigma_{t=0}^{\infty}\gamma^t \mathbb{1}_{s_t=s,a_t=a}|\pi].$$

Therefore, we have that $V^{\pi} = \Sigma_{s,a} R(s, a) x_{s,a}^{\pi}$.

The distance between the values of two policies π_E and $\tilde{\pi}$ is:

$$\begin{aligned} \left\| V^{\pi_E} - V^{\tilde{\pi}} \right\| &= \left\| \Sigma_{s,a} R(s,a) x_{s,a}^{\pi_E} - \Sigma_{s,a} R(s,a) x^{\tilde{\pi}_{s,a}} \right\| \\ &\leq \Sigma_{s,a} R(s,a) \left\| x_{s,a}^{\pi_E} - x^{\tilde{\pi}_{s,a}} \right\| . \end{aligned} \tag{5.22}$$

If the occupancy measures for the expert's policy π_E and for policy $\tilde{\pi}$ are ϵ-close, then the distance between the value of the expert's policy π_E and the value of policy $\tilde{\pi}$ will be:

$$\left\| V^{\pi_E} - V^{\tilde{\pi}} \right\| \leq \Sigma_{s,a} R(s,a)\epsilon. \tag{5.23}$$

□

5.6 Conclusions

In this chapter, I have described a modular approach to IRL and showed it working in concert with several off-the-shelf regression algorithms and performing well across several small but challenging problems.

I have shown that maximum likelihood IRL is a justified approach in the non-linear case and provided a framework for learning when the expert's reward is an unknown function from an arbitrary and known hypothesis class.

There are several shortcomings of our approach. First, I found that the modular versions of the linear and Gaussian process IRL algorithms tended to run much more slowly than their integrated counterparts. Of course, it is not surprising that the modular IRL algorithms, with their external calls to Weka, would be slower.

In addition, since they are based on local search, the modular algorithms share shortcomings with other local search algorithms, including the possibility of being trapped in local maxima. I did not see local maxima cropping up in the evaluations I performed, likely due to the small size of our test problems. A problem I did encounter, however, is that regularizing function approximators can cause the overall algorithm to fail to make progress when faced with small data sets (few states). The reason for this behavior is that the regression algorithms require sufficient data to be convinced that the differences are not due to chance and so can end up producing the same reward functions even after the gradient is applied. And, of course, if the hypothesis class is a bad fit for the data, any IRL algorithm will exhibit difficulties.

Chapter 6
Inverse Reinforcement Learning about Multiple Intentions

In this chapter, I present a maximum likelihood IRL approach to the problem of learning about multiple intentions and show that maximum likelihood is a justified approach to multiple intentions IRL.

Most IRL approaches assume that the demonstrations come from a single expert (Chapter 3). In many natural scenarios, however, the learner observes the expert acting with different intents at different times. For example, a driver might be trying to get to the store safely one day or rushing to work for a meeting on another. If trajectories are labeled by the expert to identify their underlying objectives, the problem can be decomposed into a set of separate IRL problems. However, more often than not, the learner is left to infer the expert's intention for each trajectory.

In this chapter, I formalize the problem of inverse reinforcement learning about multiple intentions and adopt a clustering approach in which observed trajectories are grouped so their inferred reward functions are consistent with observed behavior.

I start by describing the multiple intentions setting in Section 6.1, present an EM approach to clustering demonstrations coming from multiple experts and inferring the reward function corresponding to each cluster in Section 6.1.1, and show how the clusters can be used for apprenticeship learning in Section 6.1.2. Two experiments (Section 6.2) show that the clustering algorithm is able to successfully learn the intentions used to generate the demonstrated trajectories. I report results using seven IRL/AL approaches including MLIRL (Chapter 4) and I conclude in Section 6.3.

6.1 Apprenticeship Learning about Multiple Intentions

The motivation for tackling the problem of inferring multiple intentions from a set of unlabeled demonstrations comes from settings like surveillance, in which observed actors are classified as “normal” or “threatening” depending on their behavior. I contend that a parsimonious classifier results by adopting a generative model of behavior—assume actors select actions that reflect their intentions and then categorize them based on their inferred intentions. For example, the behavior of people in a train station might differ according to their individual goals: some have the goal of traveling causing them to buy tickets and then go to their trains, while others may be picking up passengers causing them to wait in a visible area. I adopt the approach of using unsupervised clustering to identify the space of common intentions from a collection of examples, then mapping later examples to this set using Bayes rule.

Similar scenarios include decision making by automatic doors that infer when people intend to go through them, a home climate control system that sets temperature controls appropriately by reasoning about the home owner’s likely destinations when driving. A common theme in these applications is that unlabeled data—observations of experts with varying intentions—are much easier to come by than trajectories labeled with their underlying goal. I define the formal problem accordingly.

In the problem of inverse reinforcement learning about multiple intentions, I assume there exists a finite set of K or fewer intentions each represented by reward weights θ_k. The apprentice is provided with a set of $N > K$ trajectories $D = \{\xi_1, ..., \xi_N\}$. Each intention is represented by at least one element in this set and each trajectory is generated by an expert with one of the intentions. An additional trajectory ξ_E is the test trajectory—the learner's objective is to produce behavior π_A that obtains high reward with respect to θ_E, the reward weights that generated ξ_E. Many possible clustering algorithms could be applied to attack this problem. I show that Expectation-Maximization (EM) is a viable approach.

6.1.1 A Clustering Algorithm for Intentions

EM [Dempster et al., 1977] is a straightforward approach to computing a maximum likelihood model in a probabilistic setting in the face of missing data. The missing data in this case are the cluster labels—the mapping from trajectories to one of the intentions. Next, an EM algorithm is derived.

Define z_{ij} to be the probability that trajectory i belongs in cluster j. Let θ_j be the estimate of the reward weights for cluster j, and ρ_j to be the estimate for the prior probability of cluster j. Following the development in Bilmes (1997), define $\Theta = (\rho_1, \ldots, \rho_K, \theta_1, \ldots, \theta_K)$ as the parameter vector we are searching for and Θ^t as the parameter vector at iteration t. Let $y_i = j$ if trajectory i came from following intention j and $y = (y_1, \ldots, y_N)$. Let $z_{ij}^t = \Pr(\xi_i | \theta_j^t)$, the probability, according to the parameters at iteration t, that trajectory i was generated by intention j.

The E step of EM simply computes

$$z_{ij}^t = \prod_{(s,a)\in\xi_i} \pi_{\theta_j^t}(s,a)\rho_j^t/Z, \tag{6.1}$$

where Z is the normalization factor.

To carry out the M step, define the EM Q function (distinct from the MDP Q function):

$$\begin{aligned}
& Q(\Theta,\Theta^t) \\
= \; & \sum_y L(\Theta|D,y))\Pr(y|D,\Theta^t) \\
= \; & \sum_y \sum_{i=1}^N \log(\rho_{y_i}\Pr(\xi_i|\theta_{y_i}))\prod_{i'=1}^N \Pr(y_{i'}|\xi_{i'},\Theta^t) \\
= \; & \sum_{y_1}\cdots\sum_{y_N}\sum_{i=1}^N\sum_{l=1}^K \delta_{l=y_i}\log(\rho_l\Pr(\xi_i|\theta_l)) \\
& \times \prod_{i'=1}^N \Pr(y_{i'}|\xi_{i'},\Theta^t) \\
= \; & \sum_{l=1}^K\sum_{i=1}^N \log(\rho_l\Pr(\xi_i|\theta_l))\sum_{y_1}\cdots\sum_{y_N}\delta_{l=y_i} \\
& \times \prod_{i'=1}^N \Pr(y_{i'}|\xi_{i'},\Theta^t) \\
= \; & \sum_{l=1}^K\sum_{i=1}^N \log(\rho_l\Pr(\xi_i|\theta_l))z_{il}^t \\
= \; & \sum_{l=1}^K\sum_{i=1}^N \log(\rho_l)z_{il}^t + \sum_{l=1}^K\sum_{i=1}^N \log(\Pr(\xi_i|\theta_l))z_{il}^t.
\end{aligned} \tag{6.2}$$

In the M step, we need to pick Θ (ρ_l and θ_l) to maximize Equation 6.2. Since they are not interdependent, we can optimize them separately. Thus, we can set $\rho_l^{t+1} = \sum_i z_{il}^t/N$ and $\theta_l^{t+1} = \operatorname{argmax}_\theta \sum_{i=1}^N z_{il}^t \log(\Pr(\xi_i|\theta_l))$. The key observation is that this second quantity is precisely the IRL log likelihood, as seen in Equation 4.1. That is, the M step demands that we find reward weights that make the observed data as likely as possible, which is precisely what MLIRL

Algorithm 4 EM Trajectory Clustering

```
**Input:** Trajectories $\{\xi_1, ..., \xi_N\}$ (with varying intentions), number of clusters $K$.
**Initialize:** $\rho_1, \ldots, \rho_K, \theta_1, \ldots, \theta_K$ randomly.
**repeat**
    E Step: Compute $z_{ij} = \prod_{(s,a)\in\xi_i} \pi_{\theta_j}(s,a)\rho_j/Z$, where $Z$ is the normalization factor.
    M step: For all $l$, $\rho_l = \sum_i z_{il}/N$. Compute $\theta_l$ via MLIRL on $D$ with weight $z_{ij}$ on trajectory $\xi_i$.
**until** target number of iterations completed.
```

seeks to do. As a result, EM for learning about multiple intentions alternates between calculating probabilities via the E step (Equation 6.1) and performing IRL on the current clusters. Algorithm 4 pulls these pieces together. This EM approach is a fairly direct interpretation of the defined clustering problem. It differs from much of the published work on learning from multiple experts, however, which starts with the assumption that all the experts have the same intentions (same reward function), but perhaps differ in their reliability (Argall et al. 2009, Richardson and Domingos 2003).

6.1.2 Using Clusters for AL

The input of the EM method of the previous section is a set of trajectories D and a number of clusters K. The output is a set of K clusters. Associated with each cluster i are the reward weights θ_i, which induce a reward function r_{θ_i}, and a cluster prior ρ_i. Next, let's consider how to carry out AL on a new trajectory ξ_E under the assumption that it comes from the same population as the trajectories in D.

By Bayes rule, $\Pr(\theta_i|\xi_E) = \Pr(\xi_E|\theta_i)\Pr(\theta_i)/\Pr(\xi_E)$. Here, $\Pr(\theta_i) = \rho_i$ and $\Pr(\xi_E|\theta_i)$ is easily computable (z in Section 6.1.1). The quantity $\Pr(\xi)$ is a simple normalization factor. Thus, the apprentice can derive a probability distribution over reward functions given a trajectory [Ziebart et al., 2008]. How should it behave? Let $f^{\pi}(s,a)$ be the (weighted) fraction of the time policy π spends taking action a in state s. Then, with respect to reward function r, the value of policy π can be written $\sum_{s,a} f^{\pi}(s,a)r(s,a)$. We should choose the policy with the highest expected reward:

$$\begin{aligned}
&\operatorname*{argmax}_{\pi} \sum_{i} \Pr(\theta_i|\xi_E) \sum_{s,a} f^{\pi}(s,a) r_{\theta_i}(s,a) \\
&= \operatorname*{argmax}_{\pi} \sum_{s,a} f^{\pi}(s,a) \sum_{i} \Pr(\theta_i|\xi_E) r_{\theta_i}(s,a) \\
&= \operatorname*{argmax}_{\pi} \sum_{s,a} f^{\pi}(s,a) r'(s,a),
\end{aligned}$$

where $r'(s,a) = \sum_i \Pr(\theta_i|\xi_E) r_{\theta_i}(s,a)$. That is, the optimal policy for the apprentice is the one that maximizes the sum of the reward functions for the possible intentions, weighted by their likelihoods. This problem can be solved by computing the optimal policy of the MDP with this averaged reward function. Thus, to figure out how to act given an initial trajectory and collection of example trajectories, our approach is to cluster the examples, use Bayes rule to figure out the probability that the current trajectory belongs in each cluster, create a merged reward function by combining the cluster reward functions using the derived probabilities, and finally compute a policy for the merged reward function to decide how to behave.

6.2 Experiments

These experiments are designed to compare the performance of the MLIRL and LPIRL algorithms (Chapter 4) with four existing IRL/AL approaches described in Chapter 3 and summarized in Chapter 4. In this section, I compare these seven approaches in several ways to assess how well they function in the setting of learning about multiple intentions. I first use a grid world example with puddles to demonstrate the performance of the MLIRL algorithm as part of the EM approach (Section 6.1) to cluster trajectories from multiple intentions—each corresponding to a different reward function. In the second experiment, I compare the performance of all the IRL/AL algorithms as part of the EM clustering approach in the simulated Highway Car domain (Abbeel and Ng 2004, Syed et al. 2008), an infinite-horizon domain with stochastic transitions.

6.2.1 Learning about Multiple Intentions—Grid World with Puddles

The first experiment is designed to demonstrate the performance of MLIRL as part of the EM approach (Section 6.1) to cluster trajectories from multiple intentions—each corresponding to a different reward function. It tests the ability of the proposed EM approach, described in Section 6.1, to accurately cluster trajectories associated with multiple intentions.

I make use of a 5×5 discrete grid world shown in Figure 6.1 (Left). The world contains starting states (shown in gray), a goal state, and blue wavy patches in the middle indicating puddles. Furthermore, states are characterized by three features, one for the goal, one for the puddles, and another for the remaining states. For added expressive power, the negations of the features are also included in the set thereby doubling the number of features to six.

Imagine data comes from two experts with different intentions. Expert 1 goes to the goal avoiding the puddles at all times and Expert 2 goes to the goal completely ignoring the puddles. Sample trajectories from these experts are shown in Figure 6.1 (Left). Trajectory T1 was generated by Expert 1, T2 and T3, by Expert 2. This experiment used a total of $N = 12$ sample trajectories of varying lengths, 5 from Expert 1, 7 from Expert 2. The EM algorithm was initiated by setting the value of K, the number of clusters, to 5 to allow some flexibility in clustering. After the clustering was run, I hand-identified the two experts. Figure 6.1 (Right) shows the algorithm's estimates that the three trajectories, T1, T2 and T3, belong to Expert 1. The EM approach was able to successfully cluster all of the 12 trajectories in the manner described above: the unambiguous trajectories were accurately assigned to their clusters and the ambiguous ones were "properly" assigned to multiple clusters. Since the value of K was set to 5, EM produced 5 clusters. On analyzing these clusters, I found that the algorithm produced 2 unique policies along with 3 copies. Thus, EM correctly extracted the preferences of the experts using the input sample trajectories.

The probability values were computed at intermediate steps during the 10 iterations of the EM algorithm. After the 1^{st} iteration, EM estimated that T1 belongs to Expert 1 with high probability and T2 belongs to Expert 1 with very low probability (implying that it therefore belongs to Expert 2). It is interesting to note here that EM estimated that trajectory T3 belongs to Expert 1 with probability 0.3. The uncertainty indicates that T3 could belong to either Expert 1 or Expert 2.

6.2.2 Learning about Multiple Intentions—Highway Car Domain

In this second experiment, I instantiated the EM algorithm in an infinite horizon domain with stochastic transitions, the simulated Highway Car domain (Abbeel and Ng 2004, Syed et al. 2008). This domain consists of a three-lane highway with an extra off-road lane on either side, a car driving at constant speed and a set of oncoming cars. Figure 6.2 shows a snapshot of the simulated highway car domain. The task is for the car to navigate through the busy highway using three actions: left, right and stay. The domain consists of three features: speed, number of collisions, number of off-road visits. The experiment uses these three features along with their negations, making a total of six features. The transition dynamics are stochastic. Four different experts were used for this experiment: **Safe**: Avoids collisions and avoids going off-road. **Student**: Avoids collisions and does not mind going off-road. **Demolition**: Collides with every car and avoids going off-road. **Nasty**: Collides with every car and does not mind going off-road. Sample trajectories of 60 steps were collected from a human subject emulating each of the four experts, with three trajectories per expert, for a total of 12 trajectories. Using these sample trajectories, the EM approach performed clustering ($K = 5$) for 5 iterations. The trajectory used for evaluation ξ_E was generated by **Student**.

Table 6.1: Highway Car Experiment Results

Algorithm	EVD Mean	EVD Standard Deviation
EM+MWAL	1.78	0.01
EM+LPIRL	1.78	0
EM+Policy Gradient	1.71	0.02
EM+Projection	1.50	0.57
Single Expert	1.45	—
EM+LPAL	1.37	0.19
EM+Max Causal Ent	0.44	0.01
EM+MLIRL	0.32	0.05

The actions selected by the approach outlined in the previous section were evaluated according to the expected value difference (or EVD), which is the difference between the value of the **Student** policy and the values of the policies computed by EM with various IRL/AL algorithms. I show the mean and standard deviation of the EVD for each algorithm over 10 runs in Table 6.1. Although the MLIRL algorithm is best suited to carry out the M step in the EM algorithm, any IRL can be used to approximately optimize the likelihood. Indeed, even AL algorithms can be used in the EM framework where a probabilistic policy takes the place of the reward weights as the hidden parameters. Thus, I instantiated each of the 7 AL/IRL approaches within the EM algorithm. I also included the ”single expert” algorithm, an approach that uses the data from all the demonstrations to learn a single reward function. It is interesting to note that maximum likelihood algorithms (MLIRL and MaxEnt) are the most effective for this task. This time, MaxEnt was provided with longer trajectories, leading to an improvement in its performance compared to Section 4.4 and Figure 4.3.

6.3 Conclusions

In this chapter, I have described a maximum likelihood approach to IRL in the multiple intention setting, where demonstrations come from experts with varying reward functions. I have shown that maximum likelihood is a justified approach in the multiple intentions IRL case—the maximum likelihood algorithm was able to successfully cluster demonstrations based on the intention that generated them and compute the reward parameters for each cluster.

The advantages of an EM approach are that it is fast (with a generally low cost per iteration), it is simple and easy implement and understand, requiring small storage space, it is numerically stable, with each iteration guaranteed to increase the likelihood of the data, it can be used to provide estimates for the missing data, and, under certain conditions, it can provide global convergence. The disadvantages are that it can be slow to converge, and in most settings it can find a local maximum instead of the global one.

In the setting I tested it, the multiple intentions maximum likelihood algorithm generally behaved well when the data was representative of each cluster.

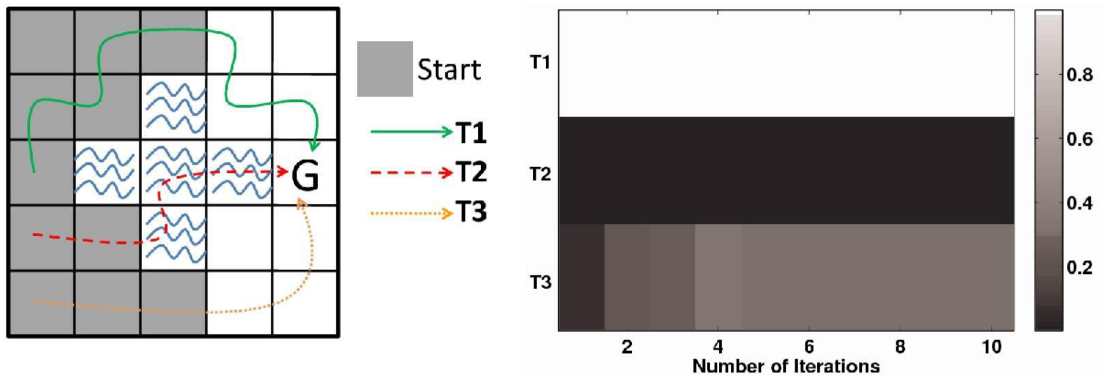

Figure 6.1: **Left:** Grid world showing the start states (grey), goal state (G), puddles and three sample trajectories. **Right:** Posterior probabilities of the three trajectories belonging to Expert 1.

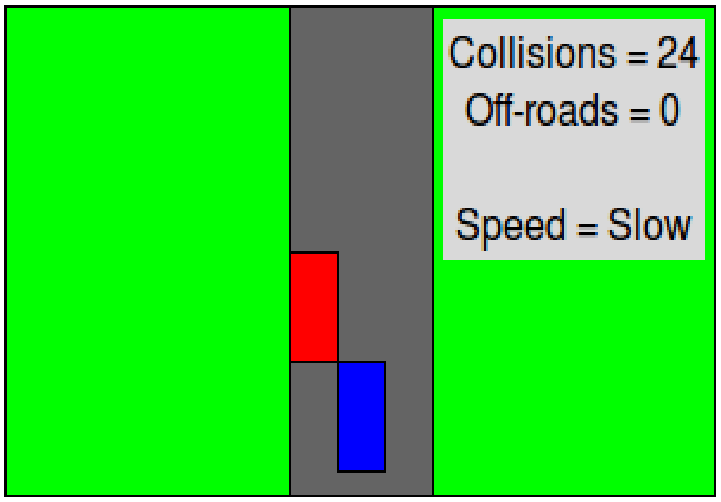

Figure 6.2: Simulated Highway Car.

Chapter 7
Applications for Maximum Likelihood Inverse Reinforcement Learning

In this chapter I describe an application of maximum likelihood inverse reinforcement learning, or MLIRL (Chapter 4) in training an artificial agent to follow verbal commands representing high-level tasks. The agent learns from a set of training data which consists of pairs of a command given in natural language and a trace of behavior corresponding to carrying out the command.

The system has three modules: the semantic parsing module (or SP) processes the command using natural language processing (or NLP) tools, the inverse reinforcement learning module (or IRL) infers the intention behind the command from the behavior, and the task abstraction tool (or TA) combines information from SP and IRL to learn demonstrated tasks. In this chapter, I describe a simplified model of the system with a unigram language model and minimal abstraction.

7.1 Introduction

For robots to be useful to humans, they need to be able to carry out useful tasks. One way this can happen is if the robots are pre-programmed for certain jobs, like vacuuming, or carrying drugs to patients in a hospital. Some artificial agents are able to recognize simple commands from a pre-programmed list, or recognize certain words in a sentence and make informed decisions based on those words. We would like to train an artificial agent to carry out tasks specified in natural language, by demonstrating the task to the agent, and having it learn the meaning of words from the pairings of the command in natural language and the task demonstration. A trained robot would be able to generalize its knowledge to understand and carry out new commands.

These agents will ground the meaning of words from pairings of language and behavior. From each demonstration, the IRL component can infer the intention of the demonstrating agent as a function of a set of state features (Chapter 4). The NLP component processes the verbal command, identifying important words and modifiers. The TA component takes as input the intention computed by IRL and the sentence information from NLP, and learns to map language to abstract tasks. When a new command is given to the agent, it gets mapped to its corresponding intention. Any ”off-the-shelf” planning algorithm can, then, be used to generate the corresponding optimal behavior.

7.2 Related Work

Our work relates to the broad class of methods that aim to learn to interpret language from a situated context Branavan et al. [2009, 2010, 2011]; Clarke et al. [2010]; Chen and Mooney [2011]; Vogel and Jurafsky [2010]; Grubb et al. [2011]; Goldwasser and Roth [2011]; Liang et al. [2011]; Hewlett et al. [2010]; Tellex et al. [2011]; Atrzi and Zettlemoyer [2011]. Instead of using annotated training data consisting of sentences and their corresponding logical forms Kate and Mooney [2006]; Wong and Mooney [2007]; Zettlemoyer and Collins [2005, 2009], most of these approaches leverage non-linguistic information from a situated context as their primary source of supervision. These approaches have been applied to various tasks such as: interpreting verbal commands in the context of navigational instructions Vogel and Jurafsky [2010]; Chen and Mooney [2011]; Grubb et al. [2011], robot manipulation Tellex et al. [2011], puzzle solving and software control Branavan et al. [2009, 2010]; semantic parsing Clarke et al. [2010]; Liang et al. [2011]; Atrzi and Zettlemoyer [2011], learning game strategies from text Branavan et al. [2011], and inducing knowledge about a domain based on text Goldwasser and Roth [2011]. The task closest to ours is interpreting navigation instructions. However, our goal is to move away from low-level instructions that correspond directly to actions in the environment Branavan et al. [2009]; Vogel and Jurafsky [2010] to high-level task descriptions expressed using complex language.

Early work on grounded language learning used the bag-of-words approach to represent the natural language input Branavan et al. [2009, 2010]; Vogel and Jurafsky [2010]. More recent methods have relied on a richer representation of linguistic data, such as syntactic dependency trees Branavan et al. [2011]; Goldwasser and Roth [2011] and semantic templates Grubb et al. [2011]; Tellex et al. [2011] to address the complexity of the natural language input. Our approach uses a flexible framework that allows us to incorporate various degrees of knowledge available at different stages in the learning process (e.g., from dependency relations to a full-fledged semantic model of the domain learned during training).

7.3 Background

The object-oriented Markov decision process formalism is used:
OOMDP$(S, A, T, r, \gamma, C, X, P)$, where S is the set of states, A is the set of actions, T are the transition probabilities, r is the reward function, γ is the discount

factor (see Chapter 2), C is a set of classes, X is a set of attributes describing objects in each class, and P is a set of propositional functions, describing high-level information about the states (the equivalent of the state-feature mapping in the MDP in Chapter 2). Objects in the environment are typed, in that they each belong to a class $c \in C$. For example, in the car driving domain, C could be the set: $\{$car, lane, road$\}$, with different attributes in X. Each car would have a color, make, and highest speed it can reach. Each lane would have a location (right lane, left lane, etc). The attributes of the road could be the number of lanes, how busy it is, and some coefficient encoding the driving conditions on it, for example, how slippery it is. In each state, the propositional functions in P would describe the current speed of the agent's car, how much gas is left in the tank, whether it is within safe driving distance of the other cars, etc.

To illustrate our approach, the Cleanup World domain is used. This domain is a grid world with one agent, a few rooms, each room with a differently colored carpet, and doorways between the rooms. Some rooms contain large toys. The transition dynamics are similar to those in the game of Sokoban: the agent can move a toy by pushing it, unless there is another toy or a wall on the other side of the toy in the direction the agent is trying to move. The Cleanup World domain can be represented as an OO-MDP with four object classes: agent, room, doorway, and toy, and a set of propositional functions that specify whether a toy is a specific shape (such as `isStar(toy)`), the color of a room (such as `isGreen(room)`), whether a toy is in a specific room (`toyIn(toy, room)`), and whether an agent is in a specific room (`agentIn(room)`). These functions belong to *shape*, *color*, *toy position* or *agent position* classes.

The training data for the overall system is a set of pairs of verbal instructions and behavior. For example, one of these pairs could be the instruction

Push the star to the green room with a demonstration of the task being accomplished in a specific environment containing various toys and rooms of different colors. We assume the availability of a set of features for each state represented using the OO-MDP propositional functions described previously. These features play an important role in defining the tasks to be learned. For example, a robot being taught to move furniture around would have information about whether or not it is currently carrying a piece of furniture, what piece of furniture it needs to be moving, which room it is currently in, which room contains each piece of furniture, etc.

The system builds on the MLIRL algorithm (Chapter 4). Given even a small number of trajectories, MLIRL finds a weighting of the state features that (locally) maximizes the probability of these trajectories. Here, these state features consist of one of the sets of propositional functions provided by the TA component. For a given task and a set of sets of state features, MLIRL evaluates the feature sets and returns to the TA component its assessment of the probabilities of the various sets.

7.4 The System Model

In this section, I present a simplified version of our system with a unigram language model, inverse reinforcement learning and minimal abstraction. This version is called Model 0. The input to Model 0 is a set of verbal instructions paired with demonstrations of appropriate behavior. It uses an EM-style algorithm Dempster et al. [1977] to estimate the probability distribution of words conditioned on reward functions (the parameters). With this information, when the system receives a new command, it can behave in a way that maximizes its reward given the posterior probabilities of the possible reward functions given the words.

Algorithm 5 shows our EM-style Model 0. For all possible reward–demonstration pairs, the E-step of EM estimates $z_{ji} = \Pr(R_j|(S_i, T_i))$, the probability that reward function R_j produced sentence-trajectory pair (S_i, T_i), This estimate is given by the equation below:

$$\begin{aligned} z_{ji} = \Pr(R_j|(S_i, T_i)) &= \frac{\Pr(R_j)}{\Pr(S_i, T_i)} \Pr((S_i, T_i)|R_j) \\ &= \frac{\Pr(R_j)}{\Pr(S_i, T_i)} \Pr(T_i|R_j) \prod_{w_k \in S_i} \Pr(w_k|R_j) \end{aligned}$$

where S_i is the i^{th} sentence, T_i is the trajectory demonstrated for verbal command S_i, and w_k is an element in the set of all possible words (vocabulary). If the reward functions R_j are known ahead of time, $\Pr(T_i|R_j)$ can be obtained directly by solving the MDP and estimating the probability of trajectory T_i under a Boltzmann policy with respect to R_j. If the R_js are not known, EM can estimate them by running IRL during the M-step Chapter 6.

The M-step in Algorithm 5 uses the current estimates of z_{ji} to further refine the probabilities $x_{kj} = \Pr(w_k|R_j)$:

$$x_{kj} = \Pr(w_k|R_j) = \frac{1}{X} \frac{\Sigma_{w_k \in S_i} \Pr(R_j|S_i) + \epsilon}{\Sigma_i N(S_i) z_{ji} + \epsilon}$$

where ϵ is a smoothing parameter, X is a normalizing factor and $N(S_i)$ is the number of words in sentence S_i.

While the final system is fully designed, it is still being implemented. As such, we only have results for specific parts of the system. I illustrate a simplified version of our Model 0 in the Cleanup Domain below. To collect a corpus of training data that would be linguistically interesting, we crowdsourced the task of generating instructions for example trajectories using Amazon Mechanical Turk. Example trajectories were presented to users as an animated image

Algorithm 5 EM-style Model 0

```
Input: Demonstrations $\{(S_1, T_1), ..., (S_N, T_N)\}$, number of reward functions $J$, size of vocabulary $K$.
Initialize: $x_{11}, \ldots, x_{JK}$, randomly.
repeat
    E Step: Compute
    $z_{ji} = \frac{\Pr(R_j)}{\Pr(S_i,T_i)} \Pr(T_i|R_j) \prod_{w_k \in S_i} x_{kj}$.
    M step: Compute
    $x_{kj} = \frac{1}{X} \frac{\Sigma_{w_k \in S_i} \Pr(R_j|S_i)+\epsilon}{\Sigma_i N(S_i) z_{ji}+\epsilon}$.
until target number of iterations completed.
```

of the agent interacting in the world, and users were asked to provide a corresponding instruction. This process had predictably mixed results: about 1/3 of the resulting instructions were badly malformed or inappropriate. For the results shown here, we have used "human-inspired" sentences, consisting of a manually constructed subset of sentences we received from our Turk experiment. These sentences were additionally simplified and clarified by retaining only the last verb and by pruning irrelevant portions of the sentence. Instructions are typically of the form, "Move the green star to the red room"; the trajectories in the training data consist of a sequence of states and actions that could be performed by the agent to achieve this goal.

To illustrate our Model 0 performance, I selected as training data six sentences for two tasks (three sentences for each task) from the dataset collected using Amazon Mechanical Turk. The training data is shown in Figure 7.1. The reward function for each task is obtained using MLIRL, computed the $\Pr(T_i|R_j)$, then ran Algorithm 5 and obtained the parameters $\Pr(w_k|R_j)$. After this training process, the agent is presented with a new task. She is given the instruction S_N: *Go to green room.* and a starting state, somewhere in the same grid. Using parameters $\Pr(w_k|R_j)$, the agent can estimate:

$\Pr(S_N|R_1) = \prod_{w_k \in S_N} \Pr(w_k|R_1) = 8.6 \times 10^{-7}, \Pr(S_N|R_2) = \prod_{w_k \in S_N} \Pr(w_k|R_2) = 4.1 \times 10^{-4}$, and choose the optimal policy corresponding to reward R_2, thus successfully carrying out the task. Note that R_1 and R_2 corresponded to the two target tasks, but this mapping was determined by EM. To illustrate the limitation of the unigram model, the trained agent is told to *Go with the star to green*, (this sentence is labeled S'_N). Using the learned parameters, the agent computes the following estimates:
$\Pr(S'_N|R_1) = \prod_{w_k \in S'_N} \Pr(w_k|R_1) = 8.25 \times 10^{-7}, \Pr(S'_N|R_2) = \prod_{w_k \in S'_N} \Pr(w_k|R_2) = 2.10 \times 10^{-5}$. The agent wrongly chooses reward R_2 and goes to the green room instead of taking the star to the green room. The problem with the unigram model in this case is that it gives too much weight to word frequencies (in this case *go*) without taking into account what the words mean or how they are used in the context of the sentence. Using the system described in Section 7.1, we can address these problems and also move towards more complex scenarios.

7.5 Conclusions

I have presented a three-component architecture for interpreting natural language instructions, where the learner has access to natural language input and demonstrations of appropriate behavior. I showed that with a very simple language model and minimal task abstraction, MLIRL can be used to infer the intentions behind demonstrated tasks. With a more complex language model on the SP side, possibly using grammars and semantic parsing, and a task abstraction component that can map the language processed by SP to abstract tasks, I believe the system can be successful in interacting with humans, and carrying out commands on behalf of its users.

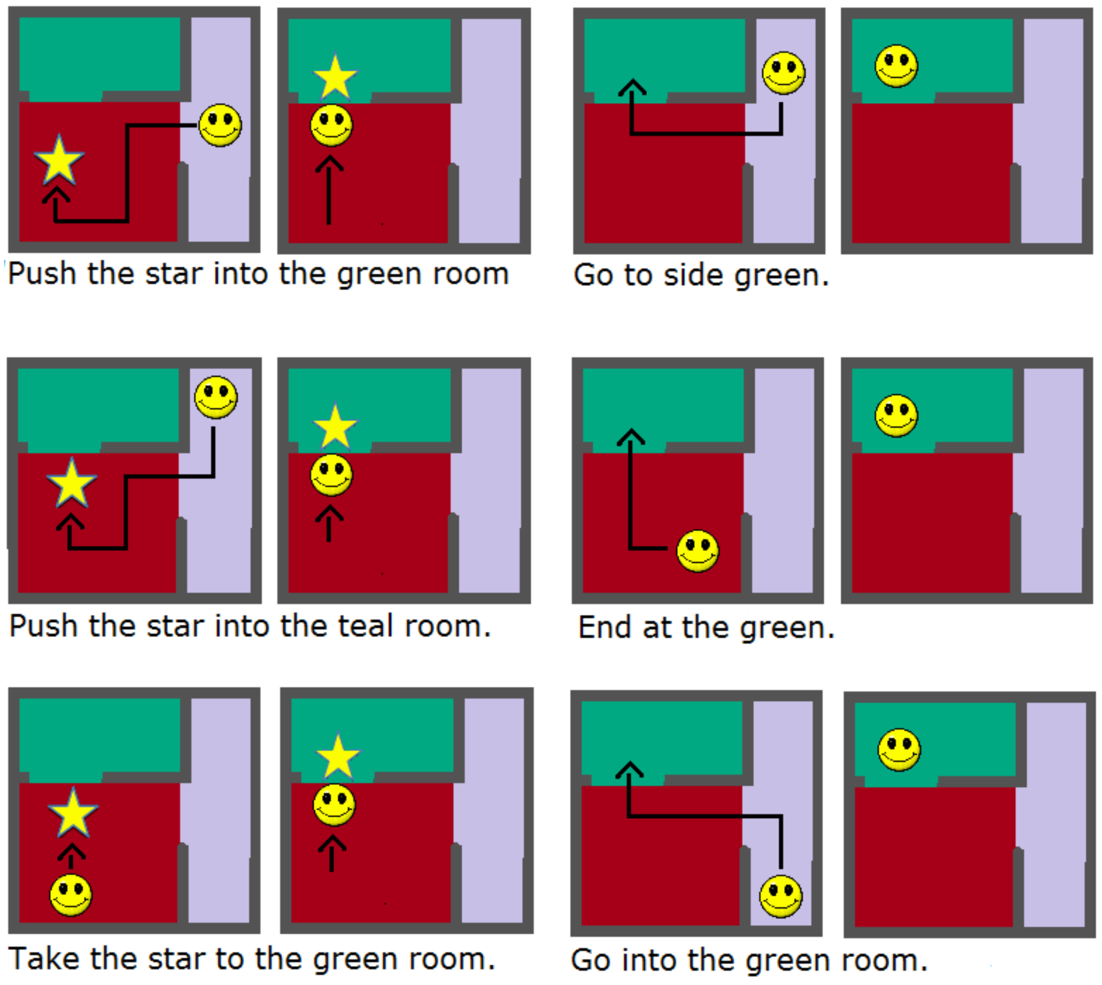
Push the star into the green room
Go to side green.
Push the star into the teal room.
End at the green.
Take the star to the green room.
Go into the green room.

Figure 7.1: Training data for 2 tasks: Taking the star to the green room (left) and Going to the green room (right).

Current work uses a generative model for behavior and tasks, and learns the parameters of the model from the training data using weakly supervised learning. Future work includes fully implementing the system to be able to build abstract tasks from language information and feature relevance, and collecting more data (using Amazon Mechanical Turk) to find out if there are different clusters for users in a cooking task.

Chapter 8
Conclusions and Future Work

My thesis statement is:

> By casting inverse reinforcement learning as a maximum likelihood problem, we can provide a unified and justified approach to linear inverse reinforcement learning, non-linear inverse reinforcement learning, and multiple intentions inverse reinforcement learning leading to effective and natural solutions.

The dissertation started by defining inverse reinforcement learning and the related problem of apprenticeship learning in Chapter 2. I mentioned that the reward-estimation problem (finding a reward that makes a given behavior optimal in a known environment) is ill-posed in that it admits an infinite number of solutions, including degenerate ones. Therefore, every IRL algorithm needs to address how a unique and meaningful reward function is chosen from this infinite set. In this dissertation, rewards were chosen to maximize the likelihood of the data given as a set of traces of optimal behavior.

In Chapter 3, I showed that no existing work has provided a unified approach to linear IRL, non-linear IRL, and multiple intentions IRL. To do so, I surveyed existing algorithms in various settings, describing the approach to solving IRL that each one took. I have shown that, even though approaches that tackle the problem in the non-linear setting could be applied to the linear setting as a special case, and approaches that tackle the problem in the multiple intentions settings could be applied to the single intention setting as a special case, none of them provided specialized algorithms for each setup, under a unified approach, like the one I introduced in this work.

In Chapter 4, I described MLIRL, a maximum likelihood algorithm for the single intention linear case. I showed that it is effective in recovering the expert's reward, even with small amounts of training data, outperforming most existing IRL and AL algorithms in a grid-world environment. MLIRL is an effective gradient ascent algorithm that iteratively tweaks the reward-function parameters to maximize the probability of the expert demonstrations under the current reward-function estimate. To account for noise in the observation or for slight deviations from optimal behavior in the demonstrations, MLIRL uses a Boltzmann distribution for action selection, with a parameter that can be tweaked according to the amount of noise in the observations.

The non-linear case was addressed in Chapter 5. Instead of focusing on computing a reward from a single hypothesis class (for example a decision tree, a Gaussian process or a neural network), the modular IRL algorithm combines a supervised-learning component with a flexible hypothesis class with maximum likelihood IRL. At each gradient ascent step, the rewards are projected through the supervised-learning component so that the learned reward both maximizes the probability of the expert demonstrations and fits in the hypothesis class given as input. I showed that if we start with the optimal policy, the gradient does not change the rewards, presented a sample complexity analysis for the algorithm, and demonstrated how issues of overfitting and underfitting apply to the modular algorithm.

The maximum likelihood algorithm for the multiple intentions case was described in Chapter 6. The expectation maximization algorithm was applied to cluster demonstrations according to their intention, and to infer the reward parameters for each intention. I showed that MLIRL is a natural fit in the M-step for computing the parameters of the reward functions corresponding to each cluster. The algorithm is successful in teasing apart demonstrations corresponding to two experts in a grid-world domain and finding their reward-function parameters.

In Chapter 7, I described an application of MLIRL to teaching an artificial agent to follow verbal commands. I showed how MLIRL can be used in an EM-like setup, together with a simple language model, to infer reward functions corresponding to novel verbal commands. In this setting, the training data consist of pairings of a verbal command and the corresponding optimal behavior, so it is important that MLIRL can learn the reward from only one trajectory.

Maximum likelihood proved to be an effective approach in all three IRL settings it was applied to—the linear case, the non-linear case, and the multiple intentions case. We computed the reward parameters that maximized the likelihood of the data (the expert's demonstrations) via gradient ascent, a simple and step-wise computationally inexpensive algorithm. Gradient ascent is intuitive, and can lead to good results, even when the amount of data is limited, which is what we observed in the settings we have tested. Gradient methods can be very slow, and require a number of iterations that increases with the size of the problem. Like other local search algorithms, they can get stuck in a local maximum, instead of finding the global one.

The Expectation Maximization algorithm is also simple to implement and fast, with a relatively low computational cost per iteration. Its disadvantages are that it can be slow to converge, and can also find a local maximum instead of the global one. In the multiple intentions settings in which it was tested, the multiple intentions maximum likelihood algorithm generally behaved well when the data was representative of each cluster.

8.1 Future Work

Future work can apply the maximum likelihood algorithms to larger datasets and in the context of transfer, which will require making the implementations more efficient and scalable.

In the linear setting, it is open whether infinite-horizon value iteration with the Boltzmann operator will converge. In the finite horizon setting I used, it was well-behaved and produced a meaningful answer, as illustrated by the experiments.

In the non-linear case, it would be useful to understand the conditions under which the composition of a function approximator and a gradient algorithm can be guaranteed to find optimal or even locally optimal solutions.

In the multiple intentions setting, it would be interesting to pursue using the learned intentions to predict the behavior and better interact with other agents in multiagent environments.

Bibliography

Pieter Abbeel, Adam Coates, and Andrew Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *I. J. Robotic Res.*, 29(13):1608–1639, 2010. 24

Pieter Abbeel, Dmitri Dolgov, Andrew Y. Ng, and Sebastian Thrun. Apprenticeship learning for motion planning with application to parking lot navigation. In *IROS*, pages 1083–1090. 2008. 24

Pieter Abbeel and Andrew Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference in Machine Learning (ICML 2004)*. 2004. 17, 18

Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning*. 2004. 40, 46, 47, 59, 73, 84, 85, 2, 19, 21, 22, 23, 25, 27

Javier Almingol, Luis Montesano, and Manuel Lopes. Learning multiple behaviors from unlabeled demonstrations in a latent controller space. In *ICML (3)*, volume 28 of *JMLR Proceedings*, pages 136–144. JMLR.org, 2013. 21, 35, 36

R. Amit and M. Matari. Learning movement sequences from demonstration. In *Development and Learning, 2002. Proceedings. The 2nd International Conference on*. 2002. 13

Brenna Argall, Brett Browning, and Manuela M. Velos. Automatic weight learning for multiple data sources when learning from demonstration. In *Proceedings of the International Conference on Robotics and Automation*, pages 226–231. 2009. 82

Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robot. Auton. Syst.*, 57(5):469–483, May 2009. ISSN 0921-8890. 12

Yoav Atrzi and Luke Zettlemoyer. Bootstrapping semantic parsers for conversations. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. 2011. 91, 92

Monica Babeş-Vroman, Vukosi Marivate, Kaushik Subramanian, and Michael Littman. Apprenticeship learning about multiple intentions. In *Proceedings of the Twenty Eighth International Conference on Machine Learning (ICML 2011)*. 2011. 21

Chris Baker, Joshua Tenenbaum, and Rebecca Saxe. Goal inference as inverse planning. In *Proceedings of the Thenty-Ninth Annual Conference of the Cognitive Sceince Society*, pages 779–784. 2007. 12

Chris L. Baker, Rebecca Saxe, and Joshua B. Tenenbaum. Action understanding as inverse planning. *Cognition*, 113(3):329–349, December 2009. 12

Jeff A. Bilmes. A gentle tutorial of the EM algorithm and its application to parameter estimation for gaussian mixture and hidden Markov models. Technical Report TR-97-021, International Computer Science Institute, 1997. 80

S. R. K. Branavan, Harr Chen, Luke S. Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09. 2009. 91, 92, 8

S. R. K. Branavan, Luke S. Zettlemoyer, and Regina Barzilay. Reading between the lines: Learning to map high-level instructions to commands. In *Association for Computational Linguistics (ACL 2010)*. 2010. 91, 92

S.R.K. Branavan, David Silver, and Regina Barzilay. Learning to win by reading manuals in a monte-carlo framework. In *Association for Computational Linguistics (ACL 2011)*. 2011. 91, 92

Senthilkumar Chandramohan, Matthieu Geist, Fabrice Lefvre, and Olivier Pietquin. User simulation in dialogue systems using inverse reinforcement learning. In *INTERSPEECH*, pages 1025–1028. ISCA, 2011. 24

David L. Chen and Raymond J. Mooney. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-2011).*, pages 859–865. 2011. 91, 92

Jaedeug Choi and Kee-Eung Kim. Inverse reinforcement learning in partially observable environments. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 1028–1033. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009. 17

Jaedeug Choi and Kee-Eung Kim. Nonparametric bayesian inverse reinforcement learning for multiple reward functions. In Peter L. Bartlett, Fernando

C. N. Pereira, Christopher J. C. Burges, Lon Bottou, and Kilian Q. Weinberger, editors, *NIPS*, pages 314–322. 2012. 29, 30, 35, 37

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. Driving semantic parsing from the world's response. In *Proceedings of the Association for Computational Linguistics (ACL 2010)*. 2010. 91, 92

Adam Coates, Pieter Abbeel, and Andrew Y. Ng. Learning for control from multiple demonstrations. 2008. 26

Adam Coates, Pieter Abbeel, and Andrew Y. Ng. Apprenticeship learning for helicopter control. *Commun. ACM*, 52(7):97–105, July 2009. ISSN 0001-0782. doi: 10.1145/1538788.1538812. URL `http://doi.acm.org/10.1145/1538788.1538812`. 26

Robert Cohn, Edmund H. Durfee, and Satinder P. Singh. Comparing action-query strategies in semi-autonomous agents. In Wolfram Burgard and Dan Roth, editors, *AAAI*. AAAI Press, 2011. 29, 31

Robert Cohn, Michael Maxim, Edmund H. Durfee, and Satinder P. Singh. Selecting operator queries using expected myopic gain. In Jimmy Xiangji Huang, Ali A. Ghorbani, Mohand-Said Hacid, and Takahira Yamaguchi, editors, *IAT*, pages 40–47. IEEE Computer Society Press, 2010. ISBN 978-0-7695-4191-4. 31, 32

Valdinei Freire da Silva, Anna Helena Reali Costa, and Pedro U. Lima. Inverse reinforcement learning with evaluation. In *ICRA*, pages 4246–4251. IEEE, 2006. 15, 18, 19, 21

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977. 80, 94

Christos Dimitrakakis and Constantin Rothkopf. Bayesian multitask inverse reinforcement learning. 2012. 29

Krishnamurthy Dvijotham and Emanuel Todorov. Inverse optimal control with linearly-solvable mdps. In Johannes Frnkranz and Thorsten Joachims, editors, *ICML*, pages 335–342. Omnipress, 2010. 16

Dan Goldwasser and Dan Roth. Learning from natural instructions. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*. 2011. 91, 92

Alexander Grubb, Felix Duvallet, Stephanie Tellex, Thomas Kollar, Nicholas Roy, Anthony Stentz, and J. Andrew Bagnel. Imitation learning for natural language direction following. In *Proceedings of the ICML Workshop on New Developments in Imitation Learning*. 2011. 91, 92

Jacques Hadamard. Sur les problèmes aux dérivés partielles et leur signification physique. *Princeton University Bulletin*, 13:49–52, 1902. 8

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009. 58

Derek Hewlett, Thomas J. Walsh, and Paul R. Cohen. Teaching and executing verb phrases. In *Proceedings of the First Joint IEEE International Conference on Development and Learning and on Epigenetic Robotics (ICDL-Epirob-11)*. 2010. 91

George H. John. When the best move isn't optimal: Q-learning with exploration. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, page 1464. Seattle, WA, 1994. 41

Andreas Junghanns and Jonathan Schaeffer. Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence*, 129:2001, 2001. 66

Rohit J. Kate and Raymond J. Mooney. Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44. 2006. 91

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, November 2002. ISSN 0885-6125. 75

Edouard Klein, Matthieu Geist, BILAL PIOT, and Olivier Pietquin. Inverse Reinforcement Learning through Structured Classification. In P Bartlett, F C N Pereira, C J C Burges, L Bottou, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1016–1024. 2012. URL `http://books.nips.cc/papers/files/nips25/NIPS2012_0491.pdf`. 21

J. Zico Kolter, Pieter Abbeel, and Andrew Y. Ng. Hierarchical apprenticeship learning with application to quadruped locomotion. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *NIPS*. Curran Associates, Inc., 2007. 16, 18, 19, 21

Seong Jae Lee and Zoran Popović. Learning behavior styles with inverse reinforcement learning. *ACM Trans. Graph.*, 29(4):122:1–122:7, July 2010. ISSN 0730-0301. doi: 10.1145/1778765.1778859. URL `http://doi.acm.org/10.1145/1778765.1778859`. 24

Sergey Levine and Vladlen Koltun. Continuous inverse optimal control with locally optimal examples. In *ICML '12: Proceedings of the 29th International Conference on Machine Learning*. 2012. 25, 34

Sergey Levine, Zoran Popovi, and Vladlen Koltun. Feature construction for inverse reinforcement learning. In *Advances in Neural Information Processing Systems 23*, pages 1342–1350. 2010. 21, 33, 37

Sergey Levine, Zoran Popovi, and Vladlen Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems 24*, pages 19–27. 2011. 65, 21, 34, 37

Percy Liang, Michael Jordan, and Dan Klein. Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL 2011)*. 2011. 91, 92

Manuel Lopes, Francisco S. Melo, and Luis Montesano. Active learning for reward estimation in inverse reinforcement learning. In *ECML/PKDD*, pages 31–46. 2009. 21, 29, 30, 32, 37

R. Duncan Luce. *Individual Choice Behavior: A theoretical analysis*. Wiley, 1959. URL `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/B0007DELQY`. 41, 7

Francisco S. Melo and Manuel Lopes. Multi-class generalized binary search for active inverse reinforcement learning. *CoRR*, abs/1301.5488, 2013. 32

Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072. 68

Adam B. Moore, Michael T. Todd, and Andrew R. A. Conway. A computational model of moral judgment, 2009. Poster at Psychonomics Society Meeting. 8

Gergely Neu and Csaba Szepesvári. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proceedings of the Conference of Uncertainty in Artificial Intelligence*. 2007. 40, 21, 22, 23, 27, 37

Gergely Neu and Csaba Szepesvári. Training parsers by inverse reinforcement learning. *Machine Learning*, 77(2–3):303–337, 2009. 40, 22

Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *in Proc. 17th International Conf. on Machine Learning*, pages 663–670. Morgan Kaufmann, 2000. 17, 18, 20, 22

Martin L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994. 40, 4, 6, 9

Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *Proceedings of IJCAI*, pages 2586–2591. 2007. 21, 28, 29, 30, 31, 32, 35, 37, 38

Nathan Ratliff, David Bradley, J Andrew (Drew) Bagnell, and Joel Chestnutt. Boosting Structured Prediction for Imitation Learning. In B Schölkopf, J C Platt, and T Hofmann, editors, *Advances in Neural Information Processing Systems 19*. MIT Press, Cambridge, MA, 2007. URL `http://www-clmc.usc.edu/publications/B/bagnell-NIPS2006.pdf`. 33, 37

Nathan D. Ratliff, J. Andrew Bagnell, and Martin A. Zinkevich. Maximum margin planning. In *In Proceedings of the 23rd International Conference on Machine Learning (ICML06*. 2006. 15, 18, 19, 21, 23

Matthew Richardson and Pedro Domingos. Learning with knowledge from multiple experts. In *Proceedings of the International Conference on Machine Learning*, pages 624–631. 2003. 82

R. Tyrrell Rockafellar. Monotone operators and the proximal point algorithm. 14(5):877–898, 1976. 52

Constantin A. Rothkopf and Christos Dimitrakakis. Preference elicitation and inverse reinforcement learning. In *ECML/PKDD (3)*, volume 6913 of *Lecture Notes in Computer Science*, pages 34–48. Springer, 2011. 29

Stuart Russell. Learning agents for uncertain environments (extended abstract). In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, pages 101–103. ACM, New York, NY, USA, 1998. ISBN 1-58113-057-0. doi: 10.1145/279943.279964. URL `http://doi.acm.org/10.1145/279943.279964`. 2, 19

Claude Sammut, Scott Hurst, Dana Kedzier, and Donald Michie. Learning to fly. In *In Proceedings of the Ninth International Conference on Machine Learning*, pages 385–393. Morgan Kaufmann, 1992. 13

Umar Syed, Michael Bowling, and Robert E. Schapire. Apprenticeship learning using linear programming. In *Proceedings of the International Conference on Machine Learning*, pages 1032–1039. 2008. 40, 46, 47, 84, 85, 21, 22, 23, 26, 27

Umar Syed and Robert E. Schapire. A game-theoretic approach to apprenticeship learning. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *NIPS*. 2007. 21, 25, 27

Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. Learning structured prediction models: A large margin approach. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 896–903. ACM, New York, NY, USA, 2005. ISBN 1-59593-180-5. 15

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the Twenty-Fifth AAAI Conference on Articifical Intelligence*. 2011. 91, 92

Adam Vogel and Dan Jurafsky. Learning to follow navigational directions. In *Association for Computational Linguistics (ACL 2010)*. 2010. 91, 92

Kevin Waugh, Brian D. Ziebart, and Drew Bagnell. Computational rationalization: The inverse equilibrium problem. In *ICML*, pages 1169–1176. 2011. 18

Yuk Wah Wong and Raymond Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-2007)*. 2007. 91

Luke Zettlemoyer and Michael Collins. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Association for Computational Linguistics (ACL'09)*. 2009. 91

Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of UAI-05*. 2005. 91

Brian D. Ziebart, J. Andrew Bagnell, and Anind K. Dey. Modeling interaction via the principle of maximum causal entropy. In *Proceedings of the 27th International Conference on Machine Learning*. 2010. 40, 25, 27

Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, pages 1433–1438. 2008. 40, 83, 21, 22, 23, 24, 25, 27