Apprenticeship Learning via Inverse Reinforcement Learning

Pieter Abbeel
Computer Science Department
Stanford University
pabbeel@cs.stanford.edu

Andrew Y. Ng
Computer Science Department
Stanford University
ang@cs.stanford.edu

Abstract

We consider learning in a Markov decision process where we are not explicitly given a reward function, but where instead we can observe an expert demonstrating the task that we want to learn to perform. This setting is useful in applications (such as the task of driving) where it may be difficult to write down an explicit reward function specifying exactly how different desiderata should be traded off. We think of the expert as trying to maximize a reward function that is expressible as a linear combination of known features, and give an algorithm for learning the task demonstrated by the expert. Our algorithm is based on using "inverse reinforcement learning" to try to recover the unknown reward function. We show that our algorithm terminates in a small number of iterations, and that even though we may never recover the expert's reward function, the policy output by the algorithm will attain performance close to that of the expert, where here performance is measured with respect to the expert's *unknown* reward function.

1 Introduction

Given a sequential decision making problem posed in the Markov decision process (MDP) formalism, a number of standard algorithms exist for finding an optimal policy, or an approximation for larger problems. In this setting, we typically assume that a reward function is given. The reward function, together with the state-action-state transition probabilities determine optimal policies and the value function. One of the key benefits of this approach is that for most tasks, it is much easier to specify the reward function than to directly specify the value function (and/or optimal policy).

But, we believe that even the reward function is frequently difficult to specify manually. Consider, for example, the task of highway driving. When driving, we typically trade off many different desiderata, such as maintaining safe following distance, keeping away from the curb, staying far from any pedestrians, maintaining a reasonable speed, perhaps a slight preference for driving in the middle lane, not changing lanes too often, and so on .... To specify a reward function for the driving task, we would have to assign a set of weights stating exactly how we would like to trade off these different factors. Despite being able to drive competently, the authors do not believe they can confidently specify a specific reward function for the task of "driving well."[1]

[1]We note that this is true even though the reward function may often be easy to state in English. For instance, the 'true" reward function that we are trying to maximize when driving is, perhaps, our "personal happiness." The practical problem however is how to model this (i.e., our happiness)

In practice, this means that the reward function is often manually tweaked (cf. reward shaping [8]) until the desired behavior is obtained. From conversations with engineers in industry and our own experience in applying reinforcement learning algorithms to several robots, we believe that, for many problems, the difficulty of manually specifying a reward function represents a significant barrier to the broader applicability of reinforcement learning and optimal control algorithms.

When teaching a young adult to drive, rather than telling them what the reward function is, it is much easier and more natural to demonstrate driving to them, and have them learn from the demonstration. The task of learning from an expert is called *apprenticeship learning* (also learning by watching, imitation learning, or learning from demonstration).

Various researchers have tackled the task of apprenticeship learning for different applications. Most of these approaches try to directly mimic the demonstrator by applying supervised learning algorithm to learn a direct mapping from the states to the actions. (This literature is too wide to survey here, but some examples include [11, 6, 4, 2, 10].) One notable exception is given in [3]. They considered the problem of controlling a robot arm, and used a reward function that quadratically penalizes deviation from the trajectory demonstrated by the expert. Note however, that this method is applicable only to problems where the task is to follow the expert's trajectory. For highway driving, blindly following the expert's trajectory would not work, because the pattern of traffic encountered is different each time.

Given that the entire field of reinforcement learning is founded on the presupposition that the reward function, rather than the policy or the value function, is the most succinct, robust, and transferable definition of the task, it seems natural to consider an approach to apprenticeship learning whereby the reward function is learned. [2]

The problem of deriving a reward function from observed behavior is referred to as inverse reinforcement learning. [9] (For other work in economics and other fields related to inverse reinforcement learning, see the discussion in [9].) In this paper, we assume that the expert is trying to optimize (without necessarily succeeding) an unknown reward function that can be expressed as a linear combination of known "features." Even though we cannot guarantee that our algorithms will correctly recover the expert's true reward function, we show that our algorithm will nonetheless find a policy that performs as well as the expert, where performance is measured with respect to the expert's *unknown* reward function.

2 Preliminaries

A (finite-state) Markov decision process (MDP) is a tuple (S, A, T, γ, D, R), where S is a finite set of states; A is a set of actions; $T = \{P_{sa}\}$ is a set of state transition probabilities (here, P_{sa} is the state transition distribution upon taking action a in state s); $\gamma \in [0, 1)$ is a discount factor; D is the initial-state distribution, from which the start state s_0 is drawn; and $R : S \mapsto A$ is the reward function, which we assume to be bounded in absolute value by 1. We let MDP\R denote a MDP without a reward function, i.e., a tuple of the form (S, A, T, γ, D).

We assume that some feature mapping $\phi : S \to [0, 1]^k$ is given, and that a "true" reward function can be written as $R^*(s) = w^* \cdot \phi(s)$, where $w^* \in \mathbb{R}^k$. [3] In order to ensure that the rewards are bounded by 1, we also assume $\|w^*\|_1 \leq 1$. In the driving domain, ϕ might be

explicitly as a function of the problems' states, so that a reinforcement learning algorithm can be applied.

[2] A related idea is also seen in the biomechanics and cognitive science literatures, in which researchers have pointed out that simple reward functions (usually ones constructed by hand) often suffice to explain complicated behaviour (policies). Some examples include the minimum jerk principle to explain limb movement in primates [5], and the minimum torque-change model to explain trajectories in human multijoint arm movement.[12] Related examples are also found in economics and some other literatures. (See the discussion in [9].)

[3] The case of state-action rewards $R(s, a)$ offers no additional difficulties. Here, we can simply write our feature mapping as $\phi : S \times A \to [0, 1]^k$, and our algorithms apply straightforwardly.

a vector of features indicating the different desiderata in driving that we would like to trade off, such as whether we have just collided with another car, whether we're driving in the middle lane, and so on. The (unknown) vector w^* specifies the relative weighting between these desiderata.

A policy π is a mapping from states to probability distributions over actions. The value of a policy π is given by

$$E_{s_0 \sim D}[V^\pi(s_0)] = E[\sum_{t=0}^\infty \gamma^t R(s_t)|\pi] \tag{1}$$

$$= E[\sum_{t=0}^\infty \gamma^t w \cdot \phi(s_t)|\pi] = w \cdot E[\sum_{t=0}^\infty \gamma^t \phi(s_t)|\pi] \tag{2}$$

Here, the expectation is taken with respect to the random state sequence $s_0, s_1, \ldots$ drawn by starting from a state $s_0 \sim D$, and picking actions according to π. We define the expected discounted accumulated feature value vector $\mu(\pi)$, or more succinctly the **feature distribution**, to be

$$\mu(\pi) = E[\sum_{t=0}^\infty \gamma^t \phi(s^{(t)})|\pi] \in \mathbb{R}^k. \tag{3}$$

Thus, the value of a policy may be written $E_{s_0 \sim D}[V^\pi(s_0)] = w \cdot \mu(\pi)$. Given that the reward R is expressible as a linear combination of the features ϕ, the feature distribution for a given policy π completely determines the expected sum of discounted rewards for acting according to that policy.

Let Π denote the set of stationary policies for an MDP. Given two policies $\pi_1, \pi_2 \in \Pi$, we can construct a new policy π_3 by mixing them together. Specifically, imagine that π_3 operates by flipping a coin with bias λ, and with probability λ picks and always acts according to π_1, and with probability $1-\lambda$ always acts according to π_2. From linearity of expectation, clearly we have that $\mu(\pi_3) = \lambda\mu(\pi_1) + (1-\lambda)\mu(\pi_2)$. Note that the randomization step selecting between π_1 and π_2 occurs only once at the start of a trajectory, and *not* on every step taken in the MDP. More generally, if we have found some set of policies $\pi_1, \ldots, \pi_d$, and want to find a new policy whose feature distribution is a convex combination $\sum_{i=1}^n \lambda_i \mu(\pi_d)$ ($\lambda_i \geq 0, \sum_i \lambda_i = 1$) of these policies', then we can do so by mixing together the policies $\pi_1, \ldots, \pi_d$, where the probability of picking π_i is given by λ_i.

We assume access to demonstrations by some expert π_E. Specifically, we assume the ability to observe trajectories (state sequences) generated by the expert starting from $s_0 \sim D$, and taking actions according to π_E. We can think of the expert as the optimal policy under the reward function $R^* = {w^*}^T \phi$, though we do not require this to hold. For our algorithm, we will require an estimate of the expert's feature distribution $\mu_E = \mu(\pi_E)$. This is easily obtained from Monte Carlo samples/observations of the expert. Specifically, given a set of m trajectories $\{s_0^{(i)}, s_1^{(i)}, \ldots\}_{i=1}^m$ generated by the expert, we let

$$\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^m \sum_{t=0}^\infty \gamma^t \phi(s_i^{(t)}) \tag{4}$$

be the empirical estimate for μ_E.[4]

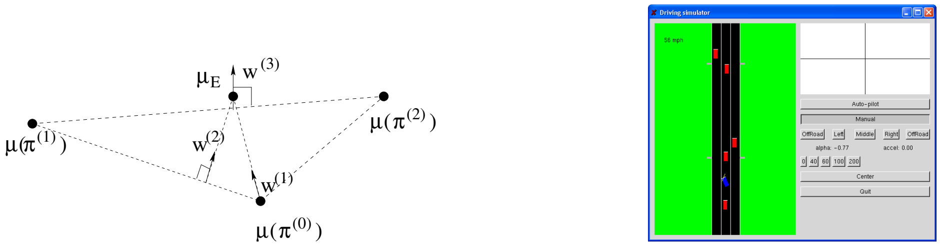
In the sequel, we also assume access to a reinforcement learning (RL) algorithm that can be used to solve an MDP\R augmented with a reward function $R = w^T\phi$. For simplicity of exposition, we will assume that the RL algorithm returns the optimal policy. (The generalization to using approximate RL algorithms offers no special difficulties; see the long version of this paper. [1])

3 Algorithm

The problem is the following: Given a MDP\R, a feature mapping ϕ and the expert's feature distribution μ_E, find a policy whose performance is close to that of the expert's, on the *unknown* reward function $R^* = {w^*}^T\phi$. To accomplish this, we will find a policy $\tilde{\pi}$ such that $\|\mu(\tilde{\pi}) - \mu_E\|_2 \leq \epsilon$. For such a $\tilde{\pi}$, we have that for any $w \in \mathbb{R}^k$ ($\|w\|_1 \leq 1$),

$$\begin{aligned} |E[\sum_{t=0}^\infty \gamma^t R(s^{(t)})|\pi_E] - E[\sum_{t=0}^\infty \gamma^t R(s^{(t)})|\tilde{\pi}]| &= |w^T\mu(\tilde{\pi}) - w^T\mu_E| \\ &\leq \|w\|_2 \|\mu(\tilde{\pi}) - \mu_E\|_2 \leq 1 \cdot \epsilon = \epsilon \end{aligned} \tag{5}$$

[4] In practice we truncate the trajectories after a finite number H of steps. If $H = H_\epsilon = \log_\gamma(\epsilon(1-\gamma))$ is the ϵ-horizon time, then this introduces at most ϵ error into the approximation.



(a) (b)

Figure 1: (a) 3 iterations for max-margin algorithm. (b) Screenshot of driving simulator

The first inequality follows from the fact that $|x^T y| \leq \|x\|_2 \|y\|_2$, and the second from $\|w\|_2 \leq \|w\|_1 \leq 1$. So the problem is reduced to finding a policy $\tilde{\pi}$ that induces a feature distribution $\mu(\tilde{\pi})$ close to μ_E's. Unfortunately, such a policy $\tilde{\pi}$ may be hard to find. Our apprenticeship learning algorithm for doing so is as follows:

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the reinforcement learning algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or approximate via Monte Carlo) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Upon termination by $t^{(n+1)} < \epsilon$, the algorithm returns $\{\pi^{(i)} : i = 0 \ldots n\}$.

Let us examine the algorithm in detail. On iteration i, we have already found some policies $\pi^{(0)}, \ldots, \pi^{(i-1)}$. The optimization in step 2 can be viewed as an inverse reinforcement learning step in which we try to guess the reward function being optimized by the expert. The maximization in that step is equivalently written

$$\text{maximize}_{t,w} \quad t \tag{6}$$
$$\text{s.t.} \quad w^T \mu_E \geq w^T \mu^{(j)} + t, \; j = 0, \ldots, i-1 \tag{7}$$
$$||w||_2 \leq 1 \tag{8}$$

From Equation 7, we see the algorithm is trying to find a reward function $R = w^{(i)} \cdot \phi$ such that $E_{s_0 \sim D}[V^{\pi_E}(s_0)] \geq E_{s_0 \sim D}[V^{\pi^{(i)}}(s_0)] + t$. I.e., a reward on which the expert does better, by a "margin" of t, than any of the i policies we had found previously. This step is similar to one of the algorithms presented in [9], but unlike the algorithms given there, because of the 2-norm constraint on w it cannot be posed as a linear program (LP), but only as quadratic program. The 2-norm constraint is critical for the proofs of the convergence theorems (to follow).[5]

Readers familiar with support vector machines (SVMs) will recognize this optimization as being equivalent to finding the maximum margin hyperplane separating two sets of points. [13] The equivalence is obtained by associating a label 1 with the expert's feature distribution μ_E, and a label -1 with the feature distributions $\{\mu(\pi^{(j)}) : j = 0..(i-1)\}$. The vector $w^{(i)}$ we want is the unit vector orthogonal to the maximum margin separating hyperplane. So an SVM solver can also be used to find $w^{(i)}$. (The SVM problem is a quadratic programming problem (QP), so we can also use any generic QP solver.)

[5]Note that although the theory assumes $\|w^*\|_1 \leq 1$, for the *algorithm* we must use $\|w\|_2 \leq 1$.

In Figure 1a we show an example of what the first 3 iterations of the algorithm could look like geometrically. Shown are several example $\mu(\pi^{(i)})$, and the $w^{(i)}$'s given by different iterations of the algorithm.

Now, suppose the algorithm terminates, with $t^{(n+1)} \leq \epsilon$. (Whether the algorithm terminates is discussed in Section 4.) Then directly from Equations (6-8) we have:

$$\forall w \text{ with } \|w\|_2 \leq 1 \ \exists i \text{ such that } w^T(\mu_E - \mu^{(i)}) \leq \epsilon. \tag{9}$$

This means that for any underlying reward function, among the set of policies returned by the algorithm, there is at least one with performance at least as good as the expert's performance minus ϵ. Thus, at this stage, we can ask the agent designer to manually test/examine the policies found by the algorithm, and pick one with acceptable performance. A slight extension of this method ensures that the agent designer has to examine at most $k+1$, rather than all $n+1$, different policies (see footnote 6).

If we do not wish to ask for human help to select a policy, alternatively we can find the point closest to μ_E in the convex closure of $\mu^{(0)}, \ldots, \mu^{(n)}$ by solving the following QP:

$$\min ||\mu_E - \mu||_2, \text{s.t. } \mu = \textstyle\sum_i \lambda_i \mu^{(i)}, \lambda_i \geq 0, \sum_i \lambda_i = 1.$$

Because μ_E is "separated" from the points $\mu^{(i)}$ by a margin of at most ϵ, we know that for the solution μ we have $||\mu_E - \mu||_2 \leq \epsilon$. Further, by "mixing" together the policies $\pi^{(i)}$ according to the mixture weights λ_i as discussed previously, we obtain a policy whose feature distribution is given by μ. Following our previous discussion, this policy attains performance near that of the expert's on the unknown reward function.[6]

3.1 A simpler algorithm

The algorithm described above requires access to a QP (or SVM) solver. It is also possible to slightly change the algorithm, such that no QP solver is needed. We will call the previous, QP-based, algorithm the **max-margin** method, and the new algorithm the **projection** method. Briefly, the projection method replaces step 2 of the algorithm with the following:

- set $\bar{\mu}^{(i-1)} = \bar{\mu}^{(i-2)} + \frac{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T(\mu_E - \bar{\mu}^{(i-2)})}{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T(\mu^{(i-1)} - \bar{\mu}^{(i-2)})}(\mu^{(i-1)} - \bar{\mu}^{(i-2)})$ (This finds the orthogonal projection of μ_E onto the line through $\bar{\mu}^{(i-2)}$ and $\mu^{(i-1)}$.)
- set $w^{(i)} = \mu_E - \bar{\mu}^{(i-1)}$
- set $t^{(i)} = \|\mu_E - \bar{\mu}^{(i-1)}\|_2$

In the first iteration, we also set $w^{(1)} = \mu_E - \mu^{(0)}$ and $\bar{\mu}^{(0)} = \mu^{(0)}$. The projection method was motivated by a convergence proof for the max-margin version. We will describe convergence proofs for this method also, and compare empirically the max-margin and the projection methods.

4 Theoretical results

Most of the results in the previous section were predicated on the assumption that the algorithm terminates with $t \leq \epsilon$. If the algorithm sometimes does not terminate, or if it sometimes takes a very (perhaps exponentially) large number of iterations to terminate, then it would not be useful. The following show that this is not the case.

Theorem 1. *Let an MDP\R, features $\phi : S \mapsto [0,1]^k$, and any $\epsilon > 0$ be given. Then the apprenticeship learning algorithm (both max-margin and projection versions) will terminate with $t^{(i)} \leq \epsilon$ after at most*

$$n = O\left(\frac{k}{(1-\gamma)^2\epsilon^2} \log \frac{k}{(1-\gamma)\epsilon}\right) \tag{10}$$

iterations.

[6] In k-dimensional space, any point that is a convex combination of a set of x points, with $x > k+1$, can be written as a convex combination of a subset of only $k+1$ points of the original x points. Applying this to $\mu = \arg\min ||\mu_E - \mu||_2$ and $\{\mu(\pi^{(i)})\}_{i=0}^n$ we obtain a set of $k+1$ policies which is equally close to the expert's feature distribution and thus have same performance guarantees.

The previous result (and all of Section 3) had assumed that μ_E was exactly known or calculated. In practice, it has to be estimated from Monte Carlo samples (Equation 4). We can thus ask about the sample complexity of this algorithm; i.e., how many trajectories m we must observe of the expert before we can guarantee we will approach its performance.

Theorem 2. *Let an MDP\R, features $\phi : S \mapsto [0,1]^k$, and any $\epsilon > 0, \delta > 0$ be given. Suppose the apprenticeship learning algorithm (either max-margin or projection version) is run using an estimate $\hat{\mu}_E$ for μ_E obtained by m Monte Carlo samples. In order to ensure that with probability at least $1-\delta$ the algorithm terminates after at most a number of iterations n given by Equation (10), and outputs a policy $\tilde{\pi}$ so that for any true reward $R^*(s) = {w^*}^T\phi(s)$ ($\|w^*\|_1 \leq 1$) we have*

$$E[\textstyle\sum_{t=0}^{\infty}\gamma^t R^*(s_t)|\tilde{\pi}] \geq E[\sum_{t=0}^{\infty}\gamma^t R^*(s_t)|\pi_E] - \epsilon, \tag{11}$$

it suffices that

$$m \geq \tfrac{9k}{2(\epsilon(1-\gamma))^2}\log\tfrac{2k}{\delta}.$$

Theorem 1 is proved by showing that $t^{(i)}$ decreases exponentially quickly as the algorithm is run, so that the algorithm converges with only a small number of iterations. In the setting of Theorem 2, because the estimated $\hat{\mu}_E$ is noisy, it may not be a valid feature distribution for any policy. I.e., there may be no (mixture or otherwise) policy $\tilde{\pi}$ that achieves a feature distribution $\mu(\tilde{\pi}) = \hat{\mu}_E$. We show that the algorithm will however converge exponentially quickly (in feature distribution space) towards the policy $\arg\min_\pi \|\hat{\mu}_E - \mu(\pi)\|_2$. The proof is then completed by arguing that the minimum in the preceding expression is small.

Due to space constraints, the proofs of these two theorems are deferred to the full paper [1] (available online).

In the case where the true reward function R^* does not lie exactly in the span of the basis functions ϕ, the algorithm still enjoys a graceful degradation of performance. Specifically, if $R^*(s) = w^* \cdot \phi(s) + \varepsilon(s)$ for some residual (error) term $\varepsilon(s)$, then our algorithm will have performance that is worse than the expert's by no more than $O(\|\varepsilon\|_\infty)$.

5 Experiments

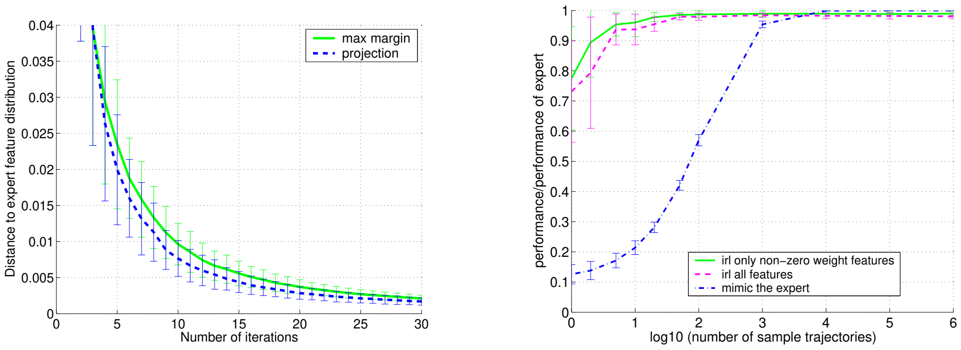
5.1 Gridworld

In our first set of experiments, we used 128 by 128 gridworlds with multiple/sparse rewards. The reward is not known to the algorithm, but we can sample trajectories from an expert's (optimal) policy. The agent has four actions to try to move in each of the four compass directions, but with 30% chance an action fails and results in a random move. The grid is divided into non-overlapping regions of 16 by 16 cells; we call these 16x16 regions "macrocells." A small number of the resulting 64 macrocells have positive rewards. For each value of $i = 1, \ldots, 64$, there is one feature $\phi_i(s)$ indicating whether that state s is in macrocell i. Thus, the rewards may be written $R^* = (w^*)^T\phi$. The weights w^* are generated randomly so as to give sparse rewards, which leads to fairly interesting/rich optimal policies for this problem.[7]

In the basic version, the algorithm is run using the 64-dimensional features ϕ. We also tried a version in which the algorithm knows exactly which macrocells have non-zero rewards (but not their values), so that the dimension of ϕ is reduced to contain only features corresponding to non-zero rewards.

In Figure 2a, we compare the max-margin and projection versions of the algorithm, when μ_E is known exactly. (Results reported are averages over 40 experiments). We plot the margin $t^{(i)}$ (distance to expert's policy) vs. the number of iterations, using all 64 macrocells

[7] Details: We used $\gamma = 0.99$, so the expected horizon is of the order of the gridsize. The true reward function was generated as follows: For each macrocell i ($i = 1, \ldots, 64$), with probability 0.9 the reward there is zero ($w_i^* = 0$), and with probability 0.1 a weight w_i^* is sampled uniformly from [0,1]. Finally, w^* is renormalized so that $\|w^*\|_1 = 1$. Instances with fewer than two non-zero entries in w^* are non-interesting and were discarded.



(a) (b)

Figure 2: (a) A comparison of convergence speed of max-margin and projection version of the algorithm on a 128x128 grid. Euclidean distance to the expert's feature distribution is plotted as a function of the number of iterations. We rescaled the feature distributions by $(1-\gamma)$ such that they are in $[0,1]^k$. 1 s.d. errorbars are also shown. (b) Plot of performance vs. number of sample trajectories of expert. Averages over 10 instances are plotted, errorbars correspond to 1 standard deviation. Note the base-10 logarithm scale on the x-axis.

as features. The expert's policy is defined as the optimal policy with respect to the given MDP. The errorbars overlap significantly, indicating that the two algorithms empirically have fairly similar rates of convergence.

The second set of experiments illustrates the performance of the algorithm as we vary the number m of sampled trajectories used to estimate μ_E. We ran the algorithm once using all 64 features, and once using only the features that truly correspond to non-zero rewards.[8] We also report on the performance of a simple "mimic the expert" algorithm, which picks whatever action the expert had taken if it finds itself in a state in which it had previously observed the expert, and picks an action randomly otherwise. Results (averages of 10 experiments) are shown in Figure 2b. For our algorithm only a few sampled trajectories—far fewer than the simple "mimic the expert" method—are needed to attain performance approaching that of the expert. (Note log scale on x-axis.) Thus, by learning a compact representation of the reward function, our algorithm significantly outperforms the naive "mimic the expert" method. We also observe that when the algorithm is told in advance which features have non-zero weight in the true reward function, it is able to learn using fewer expert trajectories.

5.2 Car driving simulation

For our second experiment, we implemented a car-driving simulation, and applied apprenticeship learning to try to learn different "driving styles." A screenshot of our simulator is shown in Figure 1b. We are driving on a highway at 25m/s (56mph), which is faster than all the other cars. The MDP has five different actions, three of which cause the car to drive smoothly to one of the lanes, and two of which cause us to drive off (but parallel to) the road, on either the left or the right side. Because our speed is fixed, if we want to avoid hitting other cars it is sometimes necessary to drive off the road.

The simulation runs at 10Hz, and in the experiments that follow, the expert's features were

---

[8]Note that, as in the text, our apprenticeship learning algorithm assumes the ability to call a reinforcement learning subroutine (in this case, an exact MDP solver using value iteration). Thus, in these experiments, we are interested mainly in the question of how many times an expert must demonstrate a task before we learn to perform the same task. In particular, we are not relying on the expert's demonstrations to learn the state transition probabilities.

estimated from a single trajectory of 1200 samples (corresponding to 2 minutes of driving time). There were features indicating what lane the car is currently in (including offroad-left and offroad-right, for a total of five features), and the distance of the closest car in the current lane.[9] Note that a distance of 0 from the nearest car implies a collision. When running the apprenticeship learning algorithm, the step in which a reinforcement learning is required was implemented by solving a discretized version of the problem. In all of our experiments, the algorithm was run for 30 iterations, and a policy was selected by inspection (per the discussion in Section 3).

We wanted to demonstrate a variety of different driving styles (some corresponding to highly unsafe driving) to see if the algorithm can mimic the same "style" in every instance. The driving styles that we considered were:

1. Nice: The highest priority is to avoid collisions with other cars; we also prefer the right lane over the middle road lane over the left lane, over driving off-road.
2. Nasty: Try to collide with as many other cars as possible.
3. Right lane nice: Almost always stay in the right lane, but go off-road when necessary to avoid hitting cars in the right lane.
4. Right lane nasty: Generally drive off-road on the right, but get back onto the road to try to hit cars in the right lane.
5. Middle lane: Drive in the middle lane, ignoring all other cars (thus crashing into all other cars in the middle lane).

After each style was demonstrated to the algorithm (by one of the authors driving in the simulator for 2 minutes), apprenticeship learning was used to try to find a policy that mimics demonstrated style. Videos of the demonstrations and of the resulting learned policies are available at

`http://robotics.stanford.edu/~abbeel/irl/`

In every instance, the algorithm was qualitatively able to mimic the demonstrated driving style. Note that since no "true" reward was ever specified or used in the experiments, we cannot report on the results of the algorithm according to R^*. However, Table 1 shows, for each of the five driving styles, the feature distribution of the expert (as estimated from the 2 minute demonstration), and the feature distribution of the learned controller for the more interesting features. Also shown are the weights w used to generate the policy shown. While our theory makes no guarantee about any set of weights w found, we note that the values there generally make intuitive sense. For instance, in the first driving style, we see negative rewards for collisions and for driving offroad, and larger positive rewards for driving in the right lane than for the other lanes.

6 Discussion and Conclusions

We assumed access to demonstrations by an expert that is trying to maximize a reward function expressible as a linear combination of known features, and presented an algorithm for apprenticeship learning. Our method is based on inverse reinforcement learning, terminates in a small number of iterations, and guarantees that the policy found will have performance comparable to or better than that of the expert, on the expert's unknown reward function.

Our algorithm assumed the reward function is expressible as a linear function of known features. If the set of features are sufficiently rich, this assumption is fairly unrestrictive. (In the extreme case where there is a separate feature for each state-action pair, fully general reward functions can be learned.) However, it remains an important problem to develop methods for learning reward functions that may be non-linear functions of the features,

[9] Actually, we picked distance to the single closest car in its current lane, discretized to the nearest car length between -7 to +2, for a total of 10 features.

		Collision	Offroad Left	LeftLane	MiddleLane	RightLane	Offroad Right
1	$\hat{\mu}_E$	0	0	0.1325	0.203333	0.598333	0.0658333
	$\mu(\tilde{\pi})$	5e-05	0.00041	0.09043	0.22869	0.60405	0.07642
	$\tilde{w}$	-0.0767	-0.0439	0.0077	0.0078	0.0318	-0.0035
2	$\hat{\mu}_E$	0.116667	0	0.0633333	0.466667	0.47	0
	$\mu(\tilde{\pi})$	0.13321	0	0.10448	0.31962	0.5759	0
	$\tilde{w}$	0.2340	-0.1098	0.0092	0.0487	0.0576	-0.0056
3	$\hat{\mu}_E$	0	0	0	0.00333333	0.705833	0.290833
	$\mu(\tilde{\pi})$	0	0	0	0	0.74465	0.25535
	$\tilde{w}$	-0.1056	-0.0051	-0.0573	-0.0386	0.0929	0.0081
4	$\hat{\mu}_E$	0.06	0	0	0.00333333	0.290833	0.705833
	$\mu(\tilde{\pi})$	0.05694	0	0	0	0.26657	0.73343
	$\tilde{w}$	0.1079	-0.0001	-0.0487	-0.0666	0.0590	0.0564
5	$\hat{\mu}_E$	0.06	0	0	1	0	0
	$\mu(\tilde{\pi})$	0.05416	0	0	1	0	0
	$\tilde{w}$	0.00936197	-0.0107928	-0.276471	0.812638	-0.509989	-0.0153853

Table 1: Feature distribution of teacher $\hat{\mu}_E$ and of selected/learned policy $\mu(\tilde{\pi})$ (as estimated by Monte Carlo). and weights w corresponding to the reward function that had been used to generate the policy shown. (Note for compactness, only 6 of the more interesting features, out of a total of 15 features, are shown here.)

and to incorporate automatic feature construction and feature selection ideas into our algorithms.

We note also that it might be possible to derive an alternative apprenticeship learning algorithm using the dual to the LP that is used in solving Bellman's equations. [7] Specifically, in this problem the variables are the state/action visitation rates, and it is possible to place constraints on the learned policy's stationary distribution. While there are few algorithms for approximating this dual LP for large MDPs and exact solutions would be feasible only for small MDPs, we consider this an interesting direction for future work.

References

[1] P. Abbeel and A.Y. Ng. Apprenticeship learning via inverse reinforcement learning. (Extended version.) `http://robotics.stanford.edu/~abbeel/irl/`, 2003.

[2] R. Amit and M.J. Mataric. Learning movement sequences from demonstration. In *Proc. ICDL*, 2002.

[3] C. Atkeson and S. Schaal. Robot learning from demonstration. In *Proc. ICML*, 1997.

[4] J. Demiris and G. Hayes. A robot controller using learning by imitation, 1994.

[5] N. Hogan. An organizing principle for a class of voluntary movements. *Journal of Neuroscience*, 4:2745–2754, 1984.

[6] Y. Kuniyoshi, M. Inaba, and H. Inoue. Learning by watching: Extracting reusable task knowledge from visual observation of human performance. *T-RA*, 10:799–822, 1994.

[7] A. Manne. Linear programming and sequential decisions. *Management Science*, 6, 1960.

[8] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proc. 16th ICML*, pages 278–287, 1999.

[9] A.Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *Proc. ICML*, 2000.

[10] Dean Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems 1*. Morgan Kaufmann, 1989.

[11] C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In *Proceedings of the Ninth International Conference on Machine Learning*, Aberdeen, 1992. Morgan Kaufmann.

[12] Y. Uno, M. Kawato, and R. Suzuki. Formation and control of optimal trajectory in human multijoint arm movement. minimum torque-change model. *Biological Cybernetics*, 61:89–101, 1989.

[13] V. N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.