Inverse Reinforcement Learning

Arthur Fischer
Department of Computer Science
Technische Universität Darmstadt
arthurf@hotmail.com

Abstract

Recently researches on imitation learning have shown that Markov Decision Processes (MDPs) are a powerful way to characterize this problem. Inverse reinforcement learning tries to describe observed behavior by ascertaining a reward function (or respectively a cost function) by solving a Markov Decision Problem. This paper shows three different approaches to find an optimal policy which mimics observed behavior. The differences and issues will be pointed out and compared on some applications. The first approach handles different cases in which the policy and states are finite and known, the state size is continuous, and the policy is only known through a finite set of observed trajectories. The second approach LEARCH extends Maximum Margin Planning and is simpler to implement like many other approaches while satisfying constraints on the cost function in a more naturally way. The last approach is based on the principle of maximum entropy and reduces learning to the problem of recovering utility function that closely mimics demonstrated behavior.

1 Introduction

Reinforcement learning

Reinforcement learning (RL) is the learning and improving of ways to achieve a desired goal. For this purpose the reinforcement learner has to pass a number of state-action-pairs and unlike in many other machine learning approaches the learner is not told which action to take in which state. The learner has to find the "optimal" action in any state by himself in order to maximize a numerical reward[4].

The learner needs to sensor his environment to determine the state he is in and how the action he accordingly takes is affecting his environment. For each action he can calculate a reward with which he will be able to compare the possible actions with each other. And here is the first problem. In order to maximize the reward the learner is aiming to exploit the action he has already obtained and assumes to be the highest rewarded one. But, to find the "best" action he also needs to explore other actions considering the risk to get a lower reward. Therefore, a balance between exploitation and exploration is indispensable to achieve the maximum reward[4]. Now assuming a learner has visited all states and actions in a finite system (or model), and is aware of all rewards, another problem occurs. At any state s the learner will now take the highest rewarded action to the next state. Unfortunately this could lead to a state with very low rewarded successor-actions and finally to a non-optimal solution.

For this reason it is necessary to consider the long-term rewards - the value. So the learner doesn't choose due to the reward, but the value. To make a decision what action to choose the learner needs a policy.

Summarized a reinforcement learning system consists of four elements: a policy, a reward function, a value function and a model[4]: A

policy defines the behavior of the learner at a given time within a state of the model and maps the state to an action. While the

reward function maps the immediate reward for taking action a from state s into state s' the

value function calculates the long-term value of the rewards. Or in other words the estimated total reward from an given state to the goal. So the value also considers the rewards from actions which can be taken from state s' to succeeding states and so on until the final goal is achieved. And the

model of the environment is used for planning. The model tries to mimic the environment and predict the future state s' after taking action a from a state s.

Markov decision process (MDP)

One possibility of describing such a reinforcement learner is to use a method (respectively an algorithm) that solves a Markov decision process. A MDP can be defined as a reward function and a model, where the model is given by the states, actions and transition probabilities conditioned on each possible action[5]. Or more formally[1]:

A (finite) MDP is a tuple $(S,A,\{P_{sa}\},\gamma,R)$, where

- S is a finite set of N **states**.
- $A = \{a_1,\ldots,a_k\}$ is a set of k **actions**.
- $P_{sa}(\cdot)$ are the state **transition probabilities** upon taking action a in state s.
- $\gamma \in [0,1)$ is the **discount factor**.
- $R : S \mapsto \mathbb{R}$ is the **reinforcement function** (reward function), bounded in absolute value by R_{max}.

To decide which action to take at any state s a **policy** is defined as any map $\pi : S \mapsto A$. The total reward (value) is calculated by the following **value function**:

$V^{\pi}(s_1) = \mathrm{E}\left[R(s_1) + \gamma R(s_2) + \gamma^2 R(s_3) + \cdots \mid \pi\right]$. Therewith a reinforcement learner can be fully described. The goal of a reinforcement learner, now, is to find a policy π for which V^{π} is being maximized.

Although reinforcement learning methods are largely used to fully observable MDPs where the sensor input is sufficient to identify the states, RL can also be applied on partially observable MDPs (POMDP)[5].

2 Inverse Reinforcement Learning (IRL)

Reinforcement learning is a powerful method for finding the optimal solution for a desired goal. However, in RL the reward function is assumed to be fixed and known but for many problems this assumption is not applicable. The reward function is rather unknown. To solve this problem Russel defined the inverse reinforcement learning (IRL) problem as follows[5]:

Given 1) measurement of agent's behavior over time, in a variety of circumstances, 2) measurements of the sensory inputs to that agent; 3) a model of the physical environment (including the agent's body).

Determine the reward function that the agent is optimizing.

The purpose of IRL is to find a reward function that describes observed behavior.

Approaches for IRL

There are some different approaches to solve an IRL problem. The first one shown here is directly applied on the Markov decision process.

2.1 MDP

This approach makes use of two basic properties of MDPs[1] described by the

Bellman Equations: *Let an MDP $M = (S, A, \{P_a\}, \gamma, R)$ and a policy $\pi : S \mapsto A$ be given. Then, for all $s \in S$,$a \in A$, V^π and Q^πsatisfy*

$$V^\pi(s) = R(s) + \gamma \sum_{s'} P_{s\pi(s)}(s') V^\pi(s')$$

$$Q^\pi(s) = R(s) + \gamma \sum_{s'} P_{sa}(s')\, V^\pi(s')$$

Bellman Optimality: *Let an MDP $M = (S, A, \{P_a\}, \gamma, R)$ and a policy $\pi : S \mapsto A$ be given. Then π is an optimal policy for M if and only if, for all $s \in S$,*

$$\pi(s) \in \arg\max_{a \in A} Q^\pi(s, a)$$

It is formulated by Andrew Ng and Stuart Russel[1] and provides the basis for some other approaches. Three different cases will be shown in this section.

IRL in Finite State Spaces

To begin with a simple case, the state size is assumed to be finite, the model is known, and the policy is fully observed. The agent now needs to find a reward function R for which the given policy π is an optimal policy. To simplify the algorithm the policy $\pi(s) \equiv a_1$can be assumed without loss of generality, considering a renaming of the actions might become necessary. This policy is optimal if and only if the reward $\boldsymbol{R}$ satisfies

$(P_{a_1} - P_a)\left(I - \gamma P_{a_1}\right)^{-1} R \succeq 0$, for all $a = (a_2, \ldots, a_k)$. This equation is necessary and sufficient for the policy to be the unique optimal policy[1].

The result will be a set of reward functions solving the given Problem. Unfortunately this set of reward functions is degenerating due to the fact that if $R = 0$ (or any other constant vector), any given policy would be an optimal policy. Thus, a simple heuristic which solves this degeneracy and gives a solution for the IRL problem needs to be applied.

One way to favor a reward function from the set of solutions is to choose those which are penalizing any single step deviation from π by high costs. This can be done by maximizing the sum of the differences between the quality of the optimal action and the quality of the next-best action[1]:

$$\sum_{s \in S} \left(Q^\pi(s, a_1) - \max_{a \in A \setminus a_1} Q^\pi(s, a) \right)$$

for all reward functions R in the set of solutions. An additional penalty λ can lead to a refined preference on reward functions. As "simpler" rewards are more desirable solutions with mainly small rewards will be preferred, assuming all other things are equal. Therefore, a penalty term $-\lambda \parallel R \parallel_1$can be applied where λ can be used to balance between the two mentioned selection criteria. Further, it can be shown that there is a border λ_0 for which the optimal R is bounded away from 0 for $\lambda < \lambda_0$ and $R = 0$ for $\lambda > \lambda_0$. As a sufficient large λ causes $R = 0$ in most states, a λ_0^- just before λ_0 seems to be an appropriate choice since it gives the "simplest" R such that R is not 0 everywhere. Hence, the full optimization problem is to maximize

$$\sum_{i=1}^{N} \min_{a \in \{a_2, \ldots, a_k\}} \left\{ (P_{a_1}(i) - P_a(i))(I - \gamma P_{a_1})^{-1} R \right\} - \lambda \parallel R \parallel_1$$

such that $(P_{a_1} - P_a)(I - \gamma P_{a_1})^{-1} R \succeq 0,\ \forall a \in A \setminus a_1,\ |R_i| \leq R_{\max},\ i = 1, \ldots, N$ where $P_a(i)$ denotes the ith row of P_a.

Linear Function Approximation in Large State Spaces

In this case the state space is infinite. However, the infinite-state state MDP can be defined just like the finite-state MDP but (under the restriction that $S \in \mathbb{R}^n$) R is a mapping $R : S = \mathbb{R}^n \mapsto \mathbb{R}$. Further, a subroutine for approximating the value V^π of a given policy π is assumed to be available for any particular MDP. Therefore, R can be expressed as a function

$$R(s) = \alpha_1\phi_1(s) + \alpha_2\phi_2(s) + \ldots + \alpha_d\phi_d(s)$$

where $\phi_1, \ldots, \phi_d$ are fixed, known, and bounded basic functions mapping from $S \mapsto \mathbb{R}$. The α_is are unknown parameters which need to be fitted. For a reward function $R = \phi_i$, a given policy π, and the corresponding value V_i^π, the value function of the MDP can be described as

$$V^\pi = \alpha_1 V_1^\pi + \ldots + \alpha_d V_d^\pi.$$

Since an optimal policy $\pi \equiv a_1$ consequently has the highest value, the expected value (calculated at state s with respect to a following state s') of the value function along the optimal policy is higher, than along other policies. So a reward function R has to satisfy

$$\mathrm{E}_{\mathrm{s}'\sim\mathrm{P}_{\mathrm{sa}_1}}\left[V^\pi(s')\right] \geq \mathrm{E}_{\mathrm{s}'\sim\mathrm{P}_{\mathrm{sa}}}\left[V^\pi(s')\right]$$

for all states s and actions $a \in A \setminus a_1$. Unfortunately for an infinite state space the condition above would have been needed to be applied infinite times which makes it impossible to calculate it algorithmically. For this reason only a large but finite subset S_0 is sampled and the constraints above are applied on this subset. Another occurring problem is that reward function is now being expressed by a linear function approximator. Thus, any other reward function than $R = 0$ for which a given policy is optimal can no longer be expressed. To resolve this problem the constraint above is relaxed and expanded to pay a penalty when it is violated. Finally, the resulting optimization problem is to maximize

$$\sum_{s\in S_0} \min_{a\in\{a_2,\ldots,a_k\}} \left\{p(\mathrm{E}_{\mathrm{s}'\sim\mathrm{P}_{\mathrm{sa}_1}}\left[V^\pi(s')\right] - \mathrm{E}_{\mathrm{s}'\sim\mathrm{P}_{\mathrm{sa}}}\left[V^\pi(s')\right]\right\}$$

such that $|\alpha_i| \leq 1$, $i = 1, \ldots, d$ where p is the penalty function given by $p(x) = x$ for $x \geq 0$ and $p(x) = 2x$ otherwise 2 is the penalty weight and has been heuristically chosen[1].

IRL from Sampled Trajectories

The third and more realistic case is where only some sampled policies are available. Under some initial fixed state distribution D the goal is to find a reward function R for an unknown policy π such that π maximizes $\mathrm{E}_{s_0\sim D}[\,V^\pi(s_0)\,]$, where s_0 is a simplified notation but indeed can be applied without loss of generality since $s_0$can be a dummy state with an action independent distribution D. Further, it is assumed to be able to find an optimal policy under any reward. Due to the incomplete set of policies trajectories need to be simulated under any policy π beginning at state s_0. For this reason m Monte Carlo trajectories are executed under π. This is neccessary to estimate $V^\pi(s_0)$ for any setting of the α_is. As well needed is a average over the empirical rewards returned on these trajectories if the reward would have been $R = \phi_i$. Therefore $\hat{V}_i^\pi(s_0)$ defines just this for all $i = 1, ..., d$. In the simple case $m = 1$ $\hat{V}_i^\pi(s_0)$ would (on a sequence of states $(s_0, s_1, s_2, ...)$) look like

$$\hat{V}_i^\pi(s_0) = \phi_i(s_0) + \gamma\phi_i(s_1) + \gamma^2\phi_i(s_2) + \cdots$$

The estimated value function $V^\pi(s_0)$ is defined as

$$\hat{V}^\pi(s_0) = \alpha_1\hat{V}_1^\pi(s_0) + \ldots + \alpha_d\hat{V}_d^\pi(s_0)$$

for any setting of the α_is[1]. This looks very similar to the value function in the previous case, but instead of comparing the expected value along two given policies for every state, in this case the estimated value of two policies from a set of polices $\{\pi_1, ..., \pi_k\}$ is being compared. Thus, the optimization problem is to maximize

$$\sum_{i=1}^{k} p\left(\hat{V}^{\pi^*}(s_0) - \hat{V}^{\pi_i}(s_0)\right)$$

such that $|\alpha_i| \leq 1$, $i = 1, ..., d$, where π^* is the (assumed) optimal policy and π_i is any other policy, and p is defined the same as before. From this optimization problem a new setting of α_is can be gained, and hence a new reward function $R = \alpha_1\phi_1 + ... + \alpha_d\phi_d$. Now a new policy π_{k+1} which maximizes $V^\pi(s_0)$ under R is tried to be found and added to the current set of policies. And with this new policy a new setting of α_is can be found and so on. This procedure is repeated until an "satisfying" reward function R is found.

This approach (using methods directly applied on MDPs) shows that the IRL problem can be solved for discrete and continuous domains. However, this problem is generally ill-posed. For example if the reward function is all 0 any policy is 0 and thereby has the same value. Another problem is that noise in the measurement of the agent's sensors inputs or even in the decision making are not considered. Further, there could be more optimal policies which are never observed. And finally many imitation learning algorithms are nonlinear and need a fundamentally different problem definition[2]. Being aware of this issues LEARCH tries to give an alternative solution for this problem.

2.2 LEARCH

LEArn to seaRCH (LEARCH) is another approach to solve the imitation learning problem, but unlike the previous approach it results in a single, deterministic policy. Although, this sounds like the solution for RL, LEARCH can be applied on the inverse RL problem as well. The idea of LEARCH differs from the previous one and is more common to many other imitation learning solutions, where a system consists of two sub-systems - perception and planning.

Maximum Margin Planning (MMP)

Maximum margin planning is based on a structured prediction problem over a space of policies. Even though it extends an MDP it rather considers a cost function than a reward function. Unlike many other imitation learning approaches, MMP can be used to automate the mapping from the world model features to costs, and therefore is able to create a cost-map itself. For planning algorithms (e.g. in mobile robotics) a cost-map is very important to calculate a minimal risk route, and since Ratliff, Bagnell, and Zinkevich (2006) described MMP as a method for planning, it is for MMP[7]. After generating a cost map, any black-box planning algorithm can be used to calculate a minimal cost path and the resulting total cost. MMP started off as a linear approach with an efficient optimization procedure. It was than generalized to handle nonlinear problems as well and then, for the time being, culminated in LEARCH[2].

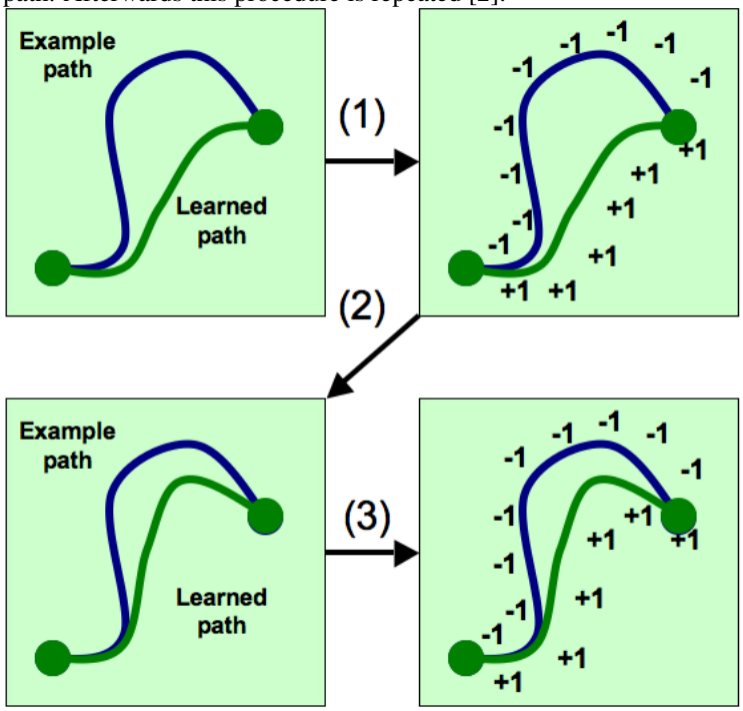
Basic properties

LEARCH is an approach of imitation learning which is applied on a set of training data $\mathcal{D} = \{(\mathcal{M}_i, \mathcal{E}_i)\}_{i=1}^N$, each consisting of an MDP $\mathcal{M}$ (reduced to states, actions, and dynamics to model the environment) and the corresponding expert policy $\mathcal{E}$.

For a given set of training data a cost function evaluates at each feature vector (within one sample) to calculate a cost map c_i^{sa}. Based on this cost map a planner can create a path with minimal costs. Thus, there are two paths (or policies) for every sample, the sample path and the minimum cost path. To minimize the gap between this two, and make the calculated path converge against the sample path, the cost function needs to be modified. More concrete, the cost of the sample path have to be decreased while increasing the costs of the planned path. Hence, the goal is to find a cost function which makes the sample path "cheaper" than any other path. Since the cost of a path is generated by evaluating at the feature vector associated to the state-action-pairs, the path cost can be manipulated by decreasing and increasing the costs at the feature vectors along the path. The easiest way of doing this is to find a regress functions which lowers the cost function in regions close to the sample path and raises the costs in regions apart (e.g. as shown in Figure 1). To generate a more significant cost function an additional loss value can be added to the non-optimal paths. The loss value reduces the costs of bad paths, so that it is harder for the algorithm to keep the sample path the "cheapest".

Knowing this the data set can be expanded. For each state-action-pair there is a fully observed feature vector $f^{sa} \in \mathbb{R}^d$ which describes the distinguish qualities of that pair. These vectors are collected in a feature matrix $F \in \mathbb{R}^{d \times |\mathcal{S}||\mathcal{A}|}$. An expert policy is denoted as $\mu_i \in \mathcal{G}_i$, where $\mathcal{G}_i$

Figure 1: This pictures show the derivation between a sample path and a learned path. The second pictures shows how the costs are reduced along the sample path and increased at the learned path. In the third pictures the new costs are considered an the learned path converges against the sample path. Afterwards this procedure is repeated [2].



is the set of all possible flow vectors. Further, the lost function is given by $\mathcal{L}_i(\mu) = l_i^T \mu$, where $l_i \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$. Hence, the data set is denoted as $\mathcal{D} = \{(\mathcal{M}_i, F_i, \mathcal{G}_i, \mu_i, l_i)\}_{i=1}^N$[2].

MMP objective

Assuming the cost function to be linear, it is denoted as $c(\mu) = w^T F_i \mu$ for a policy $\mu \in \mathcal{G}_i$ in $\mathcal{M}_i$, where $w \in \mathbb{R}^d$ is a real valued weight vector, the cost of the sample path has to be lower than the costs of any other path reduced by a loss $l_i^T \mu$. This constraint can be formulated as

$$w^T F_i \mu_i \leq w^T F_i \mu - l_i^T \mu$$

for each MDP $\mathcal{M}_i$ and each policy $\mu \in \mathcal{G}_i$. This kind of problem is called loss-augmented and the resulting cost map from the vector $w^T F_i \mu - l_i^T \mu$ is called loss-augmented cost map. For this loss-augmented problem the constraints can be transformed in a more compact nonlinear but convex form which makes it possible to efficiently optimize it using the sub-gradient method:

$$\forall i,\ w^T F_i \mu_i \leq \min_{\mu \in \mathcal{G}_i} \{w^T F_i \mu - l_i^T \mu\}.$$

However, this constraint is not sufficient, since a little scaling on w could raise the costs of a sample-near path and generate a massive gap between the sample path costs and the calculated path. Thus, w needs additional constraints on its size. Therefore, w should be as small as possible while still satisfying the given constraints. Since, there might be no vector w satisfying this constraints for every sample an further variable is needed - the slack variable. So any sample is given a slack variable from a set $\{\zeta_i\}_{i=1}^N$ that allows constraint violations for a penalty. The new constraint on the size of w can be described

$$\min_{w \in \mathcal{W}, \zeta_i \in \mathbb{R}_+} \frac{1}{N} \sum_{i=1}^{N} \zeta_i + \frac{\lambda}{2} \quad \| \quad w \|^2$$

$$\forall i,\ w^T F_i \mu_i \leq \min_{\mu \in \mathcal{G}_i} \{w^T F_i \mu - l_i^T \mu\} + \zeta_i$$

where $\lambda \geq 0$ is a constant for the trade off between penalties on constraint violations and the desire for a small weight vector w[2]. The slack variables itself are also supposed to be as small as possible. Hence, the constraint

$$\zeta_i = w^T F_i \mu_i - \min_{\mu \in \mathcal{G}_i} \{w^T F_i \mu - l_i^T \mu\}$$

can be defined. Finally, putting this definition into the constraints of w leads to a regularized risk function - the Maximum Margin Planning Objective:

$$R(w) = \frac{1}{N} \sum_{i=1}^{N} (w^T F_i \mu_i - \min_{\mu \in \mathcal{G}_i} \{w^T F_i \mu - l_i^T \mu\}) + \frac{\lambda}{2} \parallel w \parallel^2 .$$

Now these function needs to be optimized. This can be done using existing sub-gradient methods.

So compared to the first approach where difference of the transition probabilities between the optimal policy and the next-best policy with an additional penalty term was maximized (for finite state and fully observed policies), this approach tries to reduce the cost of the sample path (or policy) so that, the cost of the next-best path, reduced by a loss, is still the "cheapest" path. This solution is less degenerating than the first approach, but also has some key issues. This approach assumes the demonstrated behavior to be optimal (or at least optimal-close). Otherwise it quite frequently fails in finding a reward function which makes a observed policy optimal and significantly better than any other policy. This problem also occurs, when not the whole state space can be observed, and therefore, the observed behavior can't be described by the planning algorithm. And after all, this approach needs a well observed environment which may often not be given.

2.3 Maximum Entropy IRL

In the first two approaches reward values and cost derivations has been used to compare policies. In this approach by Brian D. Ziebart, Andrew Maas, J.Andrew Bagnell, and Anind K. Dey distributions over feature counts are compared. Since this comparator has been used in previous works and is known to be ambiguous, they employed the principle of maximum entropy (MaxEnt) to resolve this ambiguities[6].

Like in the previous approach the reward function is characterized by an reward weight vector applied on feature counts $\mathbf{f}_\zeta = \sum_{s_j \in \zeta} \mathbf{f}_{s_j}$, which sums the features at any state along a path, and can be displayed as

$$\text{reward}(\mathbf{f}_\zeta) = \theta^T \mathbf{f}_\zeta = \sum_{s_i \in \zeta} \theta^T \mathbf{f}_{s_i}.$$

For deterministic MDPs the reward weight vector θ is not calculated upon the value a policy can achieve, but on the distribution over the entire class of possible behaviors [6]. Though, for a sub-optimal demonstrated behavior there can be several distributions over paths matching the feature counts, where some of them might prefer some paths over others, for reasons which are not implied by the feature counts. This is the problem of ambiguity mentioned before and the point where maximum entropy IRL interferes. From all possible distributions MaxEnt chooses the one, which has no other preferences on paths, beyond matching features expectations

$$\sum_{\text{Path } \zeta_i} P(\zeta_i) \mathbf{f}_{\zeta_i} = \tilde{\mathbf{f}}.$$

To satisfy the additional constraint paths with equal reward will have equal probabilities and higher rewards will be exponentially more preferred. Additionally parameterizing the resulting distribution with the reward weight vector θ, the distribution can be described as

$$P(\zeta_i | \theta) = \frac{1}{Z(\theta)} e^{\theta^T \mathbf{f}_{\zeta_i}} = \frac{1}{Z(\theta)} e^{\sum_{s_j \in \zeta_i} \theta^T \mathbf{f}_{s_j}},$$

where $Z(\theta)$ is the partition function and always converges for finite horizon problems, but may fail to converge on infinite horizon problems with zero-reward absorbing states [6].

In the case of non-deterministic MDPs the paths are not only depending on the choice an agent makes at any state, but also on the output of an action which is no longer deterministic. Instead, the output of an action is linked to an transition distribution T. Thus, the agent must consider this

Algorithm 1 Expected Edge Frequency Calculation[6]

Backward Pass

1. Set $Z_{s_i,0}$ =1
2. Recursively compute for N iterations

$Z_{a_{i,j}} = \sum_k P(s_k|s_i, a_{i,j}) e^{\text{reward}(s_i|\theta)} Z_{s_k}$

$Z_{s_i} = \sum_{a_{i,j}} Z_{a_{i,j}}$

Local action probability computation

3. $P(a_{i,j}, |s_i) = \frac{Z_{a_{i,j}}}{Z_{s_i}}$

Forward Pass

4. Set $D_{s_i,t} = P(s_i = s_{\text{initial}})$
5. Recursively compute for $t = 1$ to N

$D_{s_i,t+1} = \sum_{a_{i,j}} \sum_k D_{s_k,t} P(a_{i,j}|s_i) P(s_k|a_{i,j}, s_i)$

Summing frequencies

6. $D_{s_i} = \sum_t D_{s_i,t}$

randomness, and therefore, the reward weight maximizing entropy has to do so as well. Additionally, $\mathcal{T}$ denotes the set of all action outcomes, and o specifies the next state for every action. Thereby, an MDP is deterministic over paths where the action output of the path matches o. This behavior is described by an indicator function $I_{\zeta \in o}$ which is 1 when ζ is compatible with o (i.e. when the MDP is deterministic) and 0 otherwise. Taking all this into account the path distribution can be formulated as

$$P(\zeta|\theta, T) = \sum_{o \in \mathcal{T}} P_T(o) \frac{e^{\theta^T \mathbf{f}_\zeta}}{Z(\theta, o)} I_{\zeta \in o}.$$

But since, computing this distribution is intractable, a more tractable approximation is[6]:

$$P(\zeta|\theta, T) \approx \frac{e^{\theta^T \mathbf{f}_\zeta}}{Z(\theta, T)} \prod_{s_{t+1}, a_t, s_t \in \zeta} P_T(s_{t+1}|a_t, s_t).$$

However, this requires the transition randomness to have a limited effect on the behavior, but indeed this condition is often given (e.g. noise).

Learning from Demonstrated Behavior

Now the goal is to maximize the entropy of the distribution above. More formally

$$\theta^* = \arg\max_\theta L(\theta) = \arg\max_\theta \sum_{\text{examples}} \log P(\zeta|\theta, T).$$

Similar to LEARCH, this is a convex function and can be optimized using gradient-based optimization methods. This leads to

$$\nabla L(\theta) = \tilde{\mathbf{f}} - \sum_\zeta P(\zeta|\theta, T) \mathbf{f}_\zeta = \tilde{\mathbf{f}} - \sum_{s_i} D_{s_i} \mathbf{f}_{s_i},$$

where D_{s_i} denotes the expected state visitation frequencies, i.e. the gradient is the difference between an expected empirical feature counts $\tilde{\mathbf{f}}$ and the agent's expected feature counts $\mathbf{f}_{s_i}$at a state s_i [6]. A way to calculate D_{s_i} is given by algorithm 1. This approach provides a powerful framework for solving the IRL problem, however, there are many distributions for which MaxEnt is not applicable (e.g. distributions with elements of interaction and feedback) [3].

3 Applications

The different approaches have been tested on different problems and under different conditions. Hence, it is difficult to declare an best approach. The first approach has been tested on the mountain

Figure 2: IRL: Mountain car problem and resulting reward function.[5]

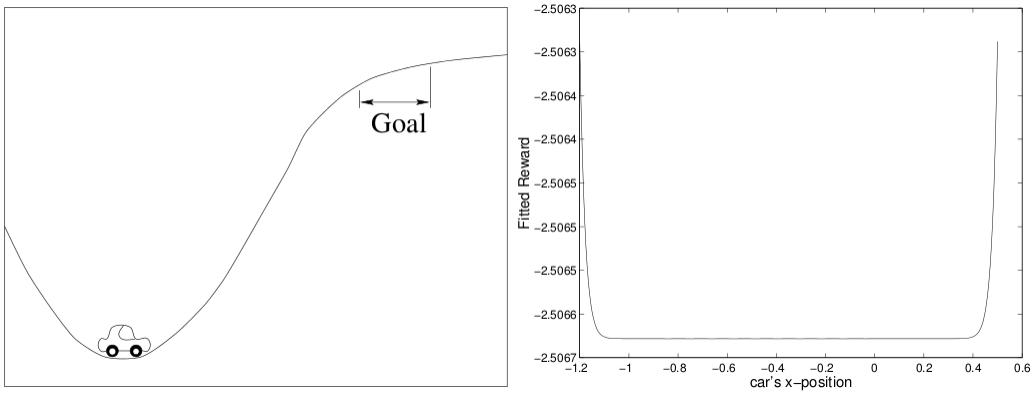
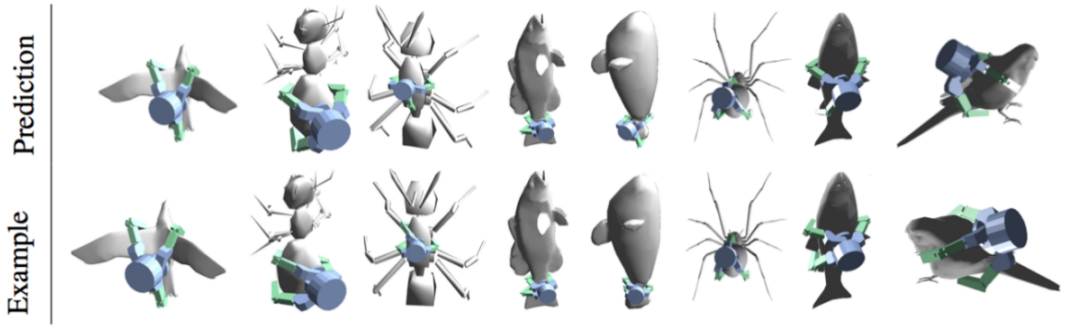


Figure 4: Path prediction of different approaches compared. First column: average percentage of distance matching, second column: percentage of examples where at least 90% of the paths' distances match[6].

	Matching	**90% Match**	**Log Prob**
Time-based	72.38%	43.12%	N/A
Max Margin	75.29%	46.56%	N/A
Action	77.30%	50.37%	-7.91
Action (costs)	77.74%	50.75%	N/A
MaxEnt paths	**78.79%**	**52.98%**	**-6.85**

car problem and provided satisfying data. LEARCH is probably less suitable, since it imitates a supervisor's behavior, and does not optimize a reward function. However, after a view iterations the first IRL approach returns a reward function which is really close to the optimal reward and describes the behavior very well (Figure 2). But on the other side, LEARCH is tested and applied on route and grasp prediction. For this problems LEARCH provides satisfying data, as it can be assumed that the supervisor's behavior is roughly optimal. The results are shown in Figure 3.

Figure 3: LEARCH: Grasp prediction for a training set of 23 examples[2]



Maximum Entropy IRL is specialized on driver prediction, and therefore, provides very good result for this problem. From Figure 4 can be seen that it outruns many other approaches including Maximum Margin Planning. However, it was aware of a training set of 100,000 miles and 3000 hours of driving mapped to 300,000 states and 900,000 actions in an MDP[6].

4 Future Work

Since, no approach is outrunning the others in any given problem, the different approaches are refined separated. The first approach was further developed to consider feature expectations to reduce the degeneracy. However, there were still some issues which MaxEnt is trying to solve. LEARCH is applicable to a lot of problems in robotic and provides fast training and generalization for applicable problems. But for large problem spaces planners necessarily become approximations and one goal is to expand the algorithm to these settings based on recent advances in gradient-based trajectory optimization. Another problem is that most environments are dynamic, and therefore, it must be considered how the environment is structured and perceived over time [2]. Maximum Entropy IRL, is going to be extended in the feature space to consider incorporating contextual factors, like day time and weather, in its specialized driver prediction problem [6].

References

[1] Andrew Y. Ng, Stuart Russell (2000). Algorithms for Inverse Reinforcement Learning. In Proceedings of the 17th International Conference on Machine Learning.

[2] Nathan Ratliff, David Silver, J. Andrew Bagnell (2009). Learning to Search: Functional Gradient Techniques for Imitation Learning. Autonomous Robots, Vol 27, No 1, pp 25-53.

[3] Brian D. Ziebart, J. Andrew Bagnell, and Anind K. Dey (2010). Modeling Interaction via the Principle of Maximum Causal Entropy. In Proceedings of the 27th International Conference on Machine Learning.

[4] Richard S. Sutton, Andrew G. Barto. Reinforcement Learning: An Introduction. MIT Press (Bradford Book), Cambridge, 1998.

[5] Stuart Russel (1998). Learning agents for uncertain environments (extended abstract). Proceedings of the Eleventh Anual Conference on Computational Learning Theory. ACM Press.

[6] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, Anind K. Dey (2008). Maximum Entropy Inverse Reinforcement Learning. Proceedings of the 23rd AAAI Conference on Artificial Intelligence.

[7] Nathan D. Ratliff, J. Andrew Bagnell, Martin A. Zinkevich (2006). Maximum Margin Planning. Proceedings of the 23rd International Conference on Machine Learning.