

Introduction to Reinforcement Learning DA671

Topic : Inverse Reinforcement Learning

Presented By:

Harsh Samoliya - 190104039

Bhavik Chandna - 200102018

Rahul - 190107055

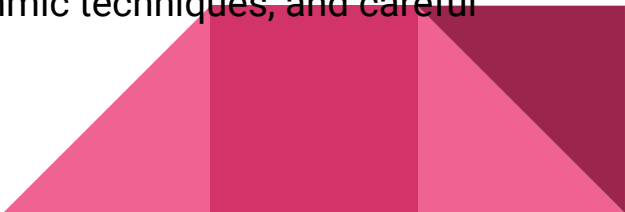
Ketan - 200107037

Chandra Mohan Suman - 190102018

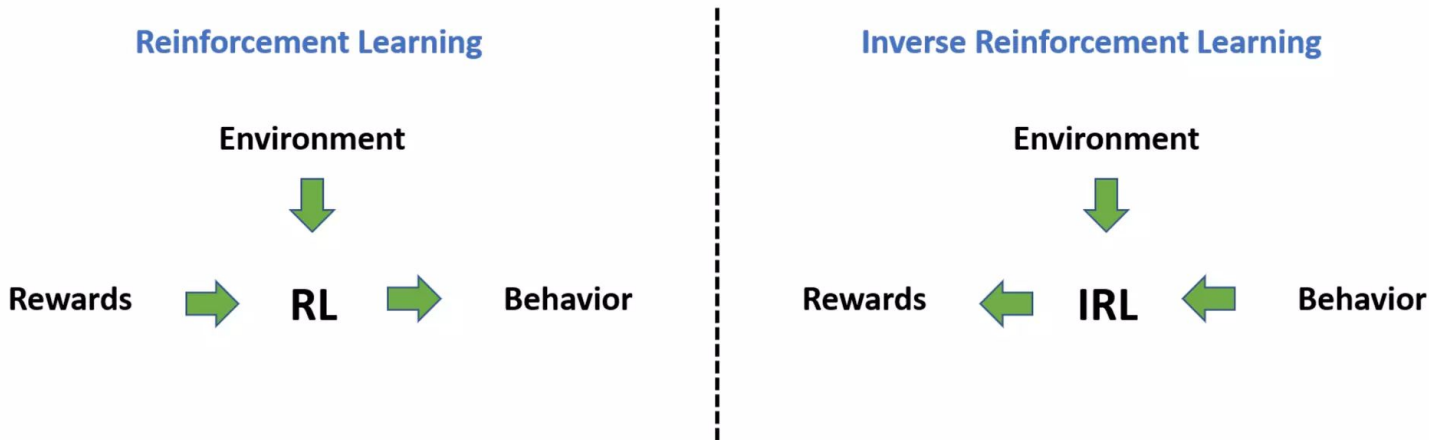
Vanja Vivek Vardhan - 190101097



1: Introduction

- IRL is a machine learning technique used to infer the reward function of an environment from observed behavior or demonstrations of an agent.
 - The goal of IRL is to learn the reward function that motivates an agent's behavior, rather than prescribing it directly.
 - IRL can be useful when the reward function is difficult to specify or when it's not clear what the optimal behavior should be.
 - IRL has applications in robotics, autonomous systems, and game design.
 - IRL can be used to learn the preferences of human operators, infer the goals of other agents, or generate new game levels that challenge players in new ways.
 - The main challenge in IRL is to infer the reward function accurately from the available data.
 - IRL typically requires a combination of domain knowledge, algorithmic techniques, and careful analysis of the data.
- 

The problem of inverse reinforcement learning (IRL) in Markov decision processes is construction of reward function given observed , except behaviour.



RL and IRL

2: The Motivation Behind IRL

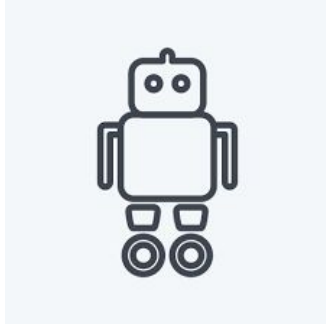
- The main idea behind IRL is that humans are often able to conclude the underlying reward structure of a task simply by observing the behavior of others, even if the task itself is not explicitly defined. By contrast, traditional reinforcement learning assumes that the reward function is known, which can be a limitation in situations where the reward function is difficult to specify or where it may change over time.
- IRL seeks to overcome this limitation by allowing agents to learn the reward function from examples of optimal behavior, such as demonstrations from human experts. This can be particularly useful in situations where the optimal behavior is difficult to define or where it may depend on important factors that are difficult to anticipate.

Lets us take this by an Example



CLASSIC EXAMPLE OF IRL

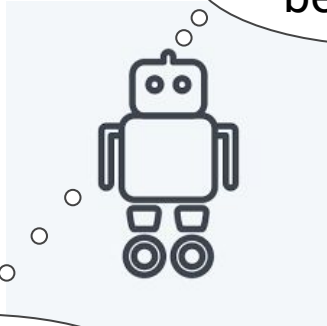
- A classic example of inverse reinforcement learning involves a robot navigating a room to reach a target location. In traditional reinforcement learning, the robot would be given a reward function that defines how much reward it receives for each action it takes. For example, it might receive a reward for moving closer to the target location, and a penalty for colliding with obstacles.



Reward Function: Known

- In Inverse reinforcement learning, the robot does not initially have access to a reward function. Instead, it observes the behavior of a human expert who is able to navigate the room to the target location. By observing the expert's behavior, the robot tries to infer the reward function that the expert is optimizing, and then uses this reward function to guide its own behavior.

Observing
behavior



Ok, now I got
the idea

Once the Robot has accrued the reward function, it can use this function to guide its own behavior in a way that is similar to the expert's behavior. By doing so, the robot can learn to navigate the room more efficiently.



Reward Function: unknown



Another example of inverse reinforcement learning (IRL) is autonomous driving, where a self-driving car must learn to drive safely and efficiently by observing the behavior of human drivers. The goal of the IRL algorithm in this case is to infer the underlying reward function that human drivers are optimizing, based on their observed behavior.

Sim2Real project takes a complex driving simulation inspired by real life — think of it as a basic, and terrible-looking video game — and teaches it to a self-driving car's AI. Voila: Wayne's car now knows how to drive on the real-life streets of Cambridge in the UK — without ever having driven them before. It's like that scene in "The Matrix" where they download kung fu into Neo's brain, except in this case, Neo is an AI.

An Autonomous Car Learned How to Drive IRL Inside a Simulation

That's a lot safer than testing it on real world streets.

[Sci-Fi Visions](#) / [Autonomous Car](#) / [Self Driving](#) / [Simulation](#)



General Notations and Definitions

- A (finite) MDP is a tuple $(S, A, \{P_{sa}\}, \gamma, R)$ where
 - S is a finite set of N states.
 - $A = \{a_1, \dots, a_k\}$ 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.
 - $R: S \rightarrow \mathbb{R}$ is the reinforcement function.
- policy $\pi: S \rightarrow A$ (map)
- Value Function : $V_\pi(s_1) = E[R(s_1) + \gamma R(s_2) + \gamma^2 R(s_3) \dots]$
- Q-Function : $Q_\pi(s, a) = R(s) + \gamma E_{s' \sim P_{sa}(\cdot)}[V_\pi(s')]$

Bellman Equations and Optimality

- We are given an MDP $M = (S, A, \{P_{sa}\}, \gamma, R)$ and policy $\pi : S \rightarrow A$. We can define two theorems :-
- ❖ **Theorem 1 (Bellman Equations)** :- Then for all $s \in S$, V_π and Q_π satisfy
$$V_\pi(s) = R(s) + \gamma \sum_{s'} P_{s\pi}(s') V_\pi(s') \text{ and } Q_\pi(s,a) = R(s) + \gamma \sum_{s'} P_{sa}(s') V_\pi(s')$$
- ❖ **Theorem 2 (Bellman Optimality)** :- Then π is an optimal policy for M iff for all $s \in S$:-

$$\pi(s) \in \operatorname{argmax}_{a \in A} Q_\pi(s,a)$$

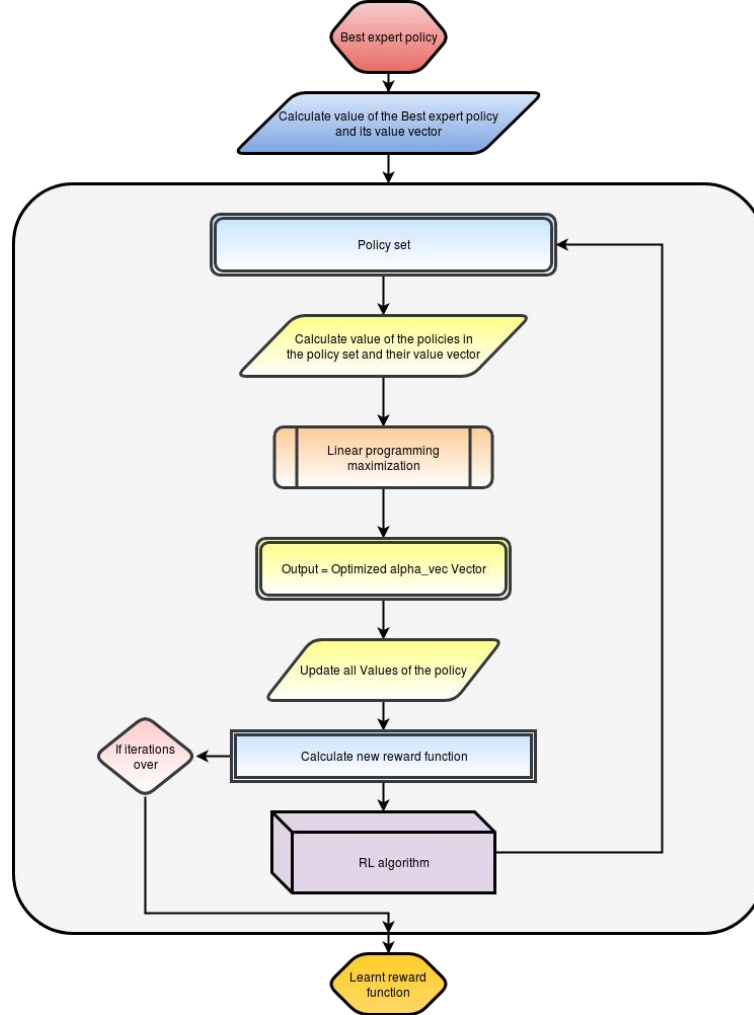


Problem Statement and Steps Taken

- **Given:**
 - Measurements of an agent's behaviour over time, in a variety of circumstances.
 - Measurements of the sensory inputs to that agent.
 - If available, a model of the environment
- **Determine:** The reward function of the observed optimised behaviour.

- Here we will try to convert the problem statement into a problem of linear Programming for all states.
- **Steps involved in solving through IRL**
 - Find the set of all reward functions for which a given policy is optimal.
 - The set contains many degenerate solutions.
 - With a simple heuristic for removing this degeneracy, resulting in a linear programming solution to the IRL problem.





- The inverse reinforcement learning problem is to find a reward function that can explain observed behaviour. We are given:
 - A finite state space S .
 - A set of k actions, $A=\{a_1, \dots, a_k\}$.
 - Transition probabilities $\{P_{sa}\}$.
 - A discount factor γ and a policy π .
- First step is to find the set of possible reward functions R such that π is a optimal policy.



Implementation of IRL in Finite State Space

Find the set of all reward functions for which a given policy is optimal.

Theorem 3 :- Let a finite state S , a set of actions $A=\{a_1, \dots, a_k\}$, transition probability matrices $\{P_a\}$, and a discount factor $\gamma \in (0,1)$ be given. Then the policy π is given by $\pi(s) \equiv a_1$ is optimal if and only if for all $a = \{a_2, a_3, \dots, a_k\}$, the reward R satisfies the below equation

Eq 1

$$(P_{a_1} - P)(I - \gamma P_{a_1})^{-1} R \geq 0$$

Problems with above Theorem

- If you remove equality from theorem, you can proof that it becomes a sufficient condition that $\pi(s) \equiv a_1$ is a unique optimal policy.
- Still some problems with above result.
 - $\mathbf{R}=0$ is always a solution.
 - For most MDPs it also seems likely that there are many choices of \mathbf{R} that meet criteria. (Degeneracy)

How do we decide which one of these many reinforcement functions to choose?



Heuristic Approach to choose suitable R

- Linear programming can be used to find a feasible point of the constraints.
- To remove degeneracy (existence of large set of R), we suggest some natural heuristics that attempt to pick a reward function that maximally differentiates the observed policy from other sub-optimal policies.
- One natural intuitive way to choose R is to first :- (i) make π optimal, (ii) each term bounded by R_{\max} and (iii) favour solution that make any single step deviation from π as costly as possible i.e maximise the below :-

Eq 2

$$\sum_{s \in S} (Q_{\pi}(s, a_1) - \max_{a \in A \setminus a_1} Q_{\pi}(s, a))$$

Penalty Terms

- Small rewards are “simpler” and preferable. Optionally add to the objective function a weight decay like penalty:

$$-\lambda ||\mathbf{R}||_1$$

- Here we are using L1 norm which have a side effect that as λ increases to large values, \mathbf{R} will often be non-zero for some states, which help in removing degeneracy.
- When we go to fine tuning value of λ , we observed that as it increases, there will be a phase transition at some point λ_0 such that optimal \mathbf{R} is bounded away from 0 for $\lambda < \lambda_0$ and $\mathbf{R}=0$ for $\lambda > \lambda_0$. We can choose the λ value by binary searching the value just before λ_0 as it gives the simplest \mathbf{R} such that \mathbf{R} is nonzero everywhere.



Final Algorithm for IRL in Finite State Space

Putting together the theorem 1) and the penalty term that balances small reinforcements and maximising the equation 2) gives the final optimization problem to be solved :-

$$\text{maximise } \sum_{i=1}^N \min_{a \in \{a_2, \dots, a_k\}} \{(\mathbf{P}_{a1}(\mathbf{i}) - \mathbf{P}_a(\mathbf{i}))(\mathbf{I} - \gamma \mathbf{P}_{a1})^{-1} \mathbf{R}\} - \lambda \|\mathbf{R}\|_1$$

Eq 3

$$\begin{aligned} \text{s.t} \quad & (\mathbf{P}_{a1} - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_{a1})^{-1} \mathbf{R} \geq 0 \quad \forall a \in \{a_2, \dots, a_k\} \\ & |\mathbf{R}_i| < \mathbf{R}_{\max} \quad \forall i = 1, \dots, N \end{aligned}$$

Clearly, this may easily be formulated as a linear program and solved efficiently.

Linear Function Approximation in Large State Spaces

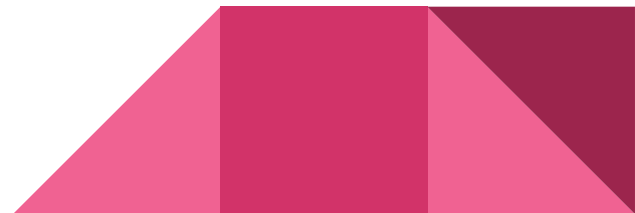
- The reward function is now assumed a map from $\mathbb{R}^n \mapsto \mathbb{R}$ as we have entered the case of infinite MDP which can be represented in form of finite MDP also.
- Here it is difficult to work with calculus of variations algorithmically, so we shall use a linear approximation of Reward function :-

Eq 4

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

where $\phi_1, \phi_2, \dots, \phi_d$ are *fixed, known, bounded basis functions* mapping from $\mathbb{R}^n \mapsto \mathbb{R}$

α_i s are parameters to be fit.



Generalized Version of Theorem 3

- We can use linearity of expectation to represent $V_\pi(s)$ as linear combination of $V_{\pi,d}(s)$.
- Using the Theorem 2 and above linearity we can verify that for \mathbf{R} to make $\pi(s) \equiv a_1$ optimal, the appropriate generalization of Theorem 3 as :-

Eq 5

$$E_{s' \sim P_{sa_1}}[V_\pi(s')] \geq E_{s' \sim P_{sa}}[V_\pi(s')] \quad \forall a \in A \setminus a_1$$

- Two problems with above equation :-
 - Infinite state spaces implies infinite constraints to be computed. (*Problem 1*)
 - Linear function approximator now can't express any \mathbf{R} for which π is optimal. (*Problem 2*)

Final Algorithm for IRL in Large State Spaces


- Problem 1 can be eliminated by sampling a large but finite states S_0 and use constraints only for those states. Problem 2 can be tackled by paying a penalty by relaxing some of the constraints when they are violated.
- Our final Linear Programming formulation is then:

$$\begin{aligned} &\text{maximise} && \sum_{s \in S_0} \min_{a \in A \setminus a_1} \{ p(E_{s' \sim P_{sa_1}}[V_\pi(s')] - E_{s' \sim P_{sa}}[V_\pi(s')]) \} \\ &\text{s.t.} && |\alpha_i| < 1 \quad i=1, 2, \dots, d \end{aligned}$$

Eq 6

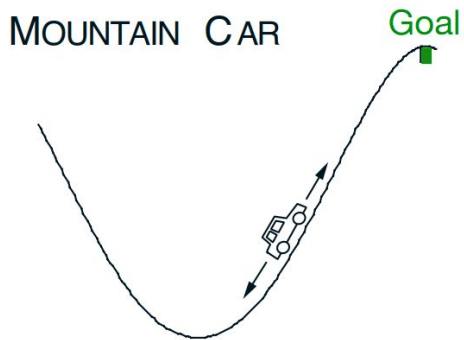
- Here S_0 is subsample of space S . $p(x)$ is given by $=x$ for $x \geq 0$ and $=2x$ for $x < 0$ which penalizes violation of constraints. 2 is penalty weight and can be heuristically chosen.
- In one iteration, by equation 6, we will get new values of α_i s which will be used to construct new reward function to start with.

IRL from Sampled Trajectories

- In realistic case, the policy π is given only through a set of actual trajectories in state space where transition probabilities (P_a) are unknown, instead assume we are able to simulate trajectories.
 - We shall start with any setting of α_i s and generate initial R to simulate Monte Carlo Trajectories under a set basis function.
 - The **algorithm** can be defined in steps :-
 - *For each π_k run the simulation starting s_0*
 - *Calculate $V_{\pi}(s_0)$ for each π_k*
 - *Our objective is to find α_i s from taking average of empirical reward function for m trajectories.*
 - *A new setting α_i s, hence a new Reward function.*
 - *Repeat until large number of iterations or R with which we are satisfied.*
- 

Experiments

- Now we shall try to conduct experiments related to all 3 algorithms and try to replicate results for :-
 - Finite State Space: Gridworld
 - Large State Space: Mountain Car
 - Through Sampled Trajectories: Continuous Gridworld



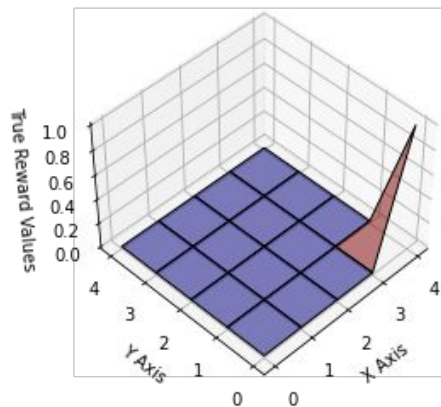
Environment, Agents, Rewards, and etc.

- Discrete Gridworld Problem :
 - 5 x 5 grid world with agent starting from lower left corner.
 - Actions corresponds to Up, Left, Right and Left - noisy, 30% chance of random direction movement.
 - Reward - +1 at top right grid square.
- Mountain Car Problem :
 - Goal is to reach top of hill.
 - Reward per step is -1 until it reaches top.
 - Continuous State space
 - Approximation class - 26 Evenly spaced Gaussian shaped basis function.
- Continuous Gridworld Problem:
 - 5x5 gridworld. But state is $[0,1] \times [0,1]$.
 - Agent moves 0,2 in intended direction with $[-0.1,0.1]$ random noise.
 - True reward is 1 in $[0.8,1] \times [0.8,1]$

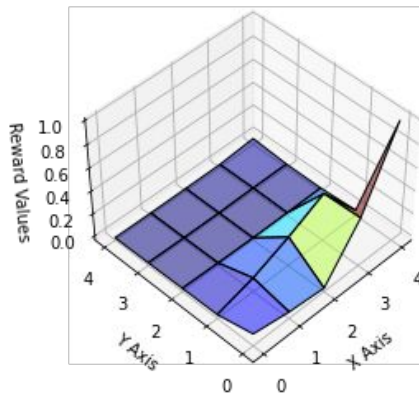


Results: Linear Programming IRL-Discrete Gridworld

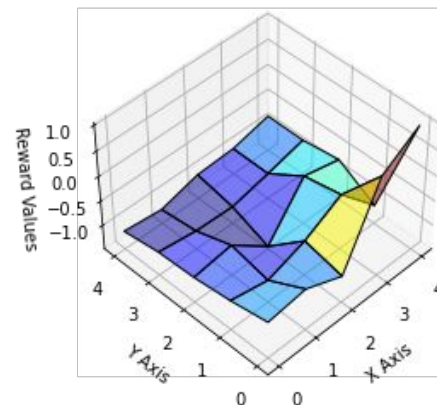
True Reward Function



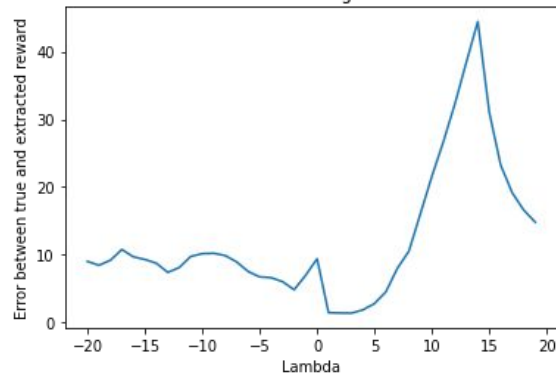
Extracted Reward Function for $\lambda = 2$



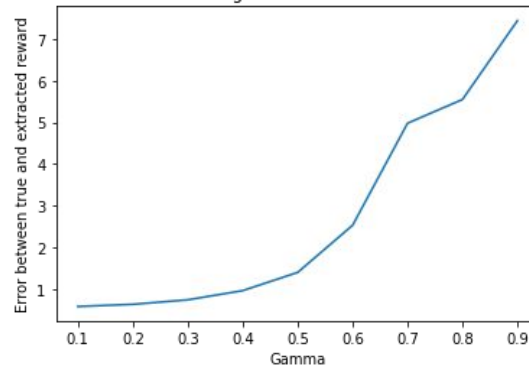
Extracted Reward Function for $\lambda = 10$



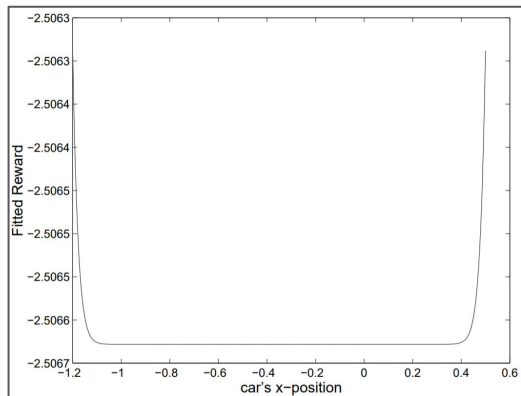
Error vs Lambda for $\gamma = 0.5$



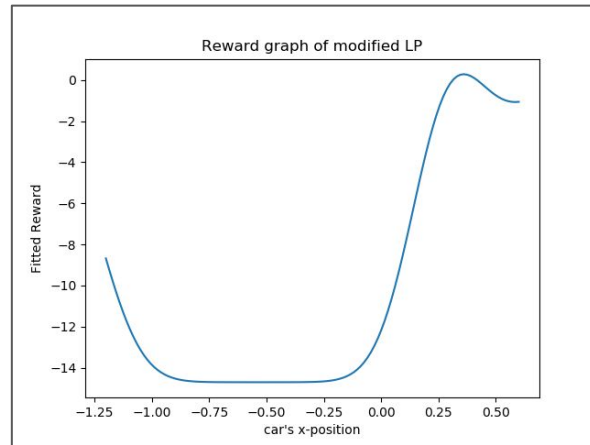
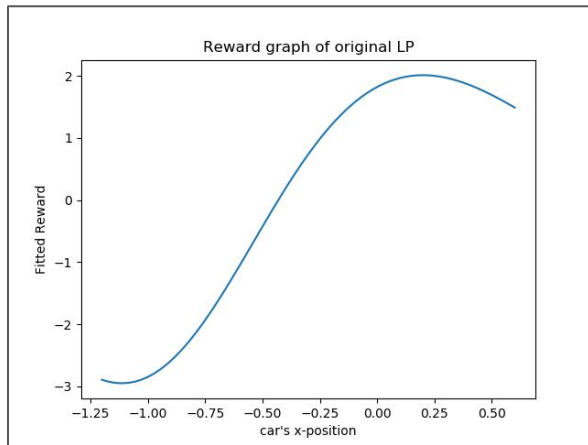
Error vs gamma for $\lambda = 3$



Results: Linear Programming IRL - Mountain Car

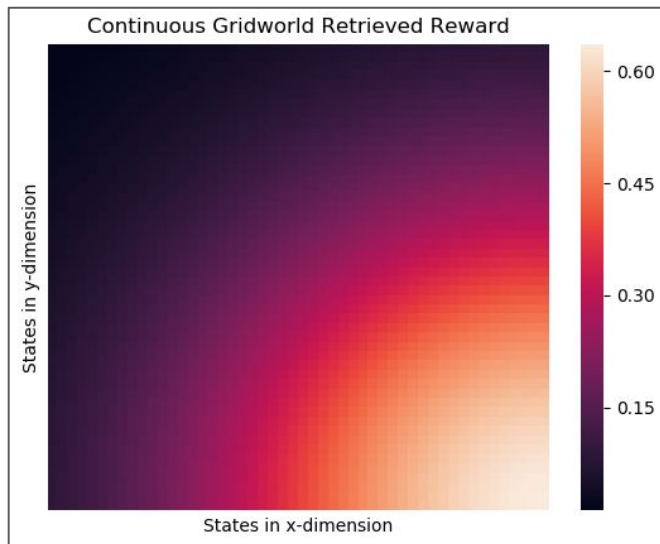
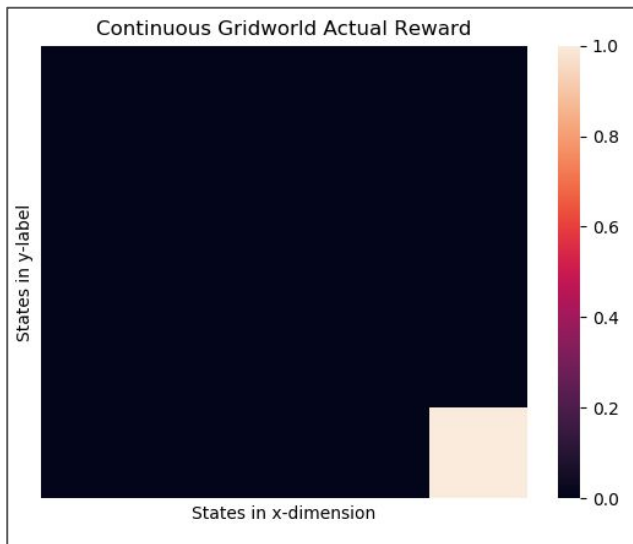


From Paper



Simulations

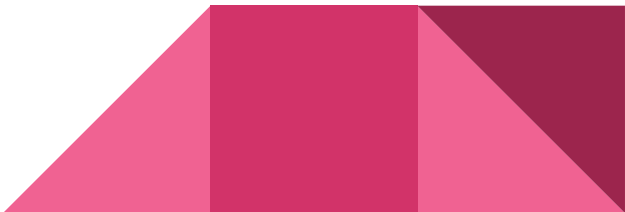
Results: Linear Programming IRL: Cont. Gridworld



The error of original policy vs. optimal policy of retrieved reward was found to be 0.297

FUTURE RESEARCH

Our results show that the inverse reinforcement learning problem is solvable, at least for moderate-sized discrete and continuous domains. A number of open questions remain to be addressed:

- Potential-based shaping rewards (Ng et al., 1999) can produce reward functions that make it dramatically easier to learn a solution to an MDP, without affecting optimality. Can we design IRL algorithms that recover "easy" reward functions?
 - In real-world empirical applications of IRL, there may be substantial noise in the observer's measurements of the agent's sensor inputs and actions; moreover, the agent's own action selection process may be noisy and/or suboptimal. Finally, there may be many optimal policies, of which only a few are observed. What are appropriate metrics for fitting such data?
 - If behavior is strongly inconsistent with optimality, can we identify locally consistent" reward functions for specific regions in state space?
- 

FUTURE RESEARCH

- How can experiments be designed to maximize the identifiability of the reward function?
- How well does our algorithmic approach carry to the case of partially observable environments?

