IRL

Sumit Chaturvedi

Abstract. The Inverse Reinforcement Learning problem tries to uncover the reward mechanism that led to an agent's behaviour. Given a behavior (policy), the paper characterizes the reward functions under which that behavior is optimal. Assuming that agents choose to act according to their best interest, finding such reward functions seems to be the logical thing to do. In this document, I describe my experiments based on this early paper by Andrew Ng and Stuart Russell [NR00].

Contents

1 Comments on Original Paper

1.1 On the Problem

In an MDP, as the paper [NR00] remarks, the most succinct description of a task is the Reward Function. This remark makes the study of Inverse Reinforcement Learning algorithms very interesting.

My fascination stems from the observation that these algorithms, in a sense, help us quantify how we "work". They give us a way to understand why we act in a certain way.

Through my primitive experiments, I learned that even the act of lifting an arm is a tough control problem. I used a policy gradient algorithm on the Acrobot problem from OpenAI gym [Bro+16] and didn't obtain a single reliable agent to solve it. And yet, we are able to master this task (well not exactly this task) with ease. It must be that the rewards that my brain relies on to learn this action are extremely well designed.

Then there are other behaviours, like drug addiction, that are very easy to learn. If we could uncover the reward mechanisms that cause people to stick to drugs, then we would be able to engineer those mechanisms to help the people give them up as well.

Obviously the paper doesn't go far in order to achieve this goal. But it does set the vocabulary to tackle the problem.

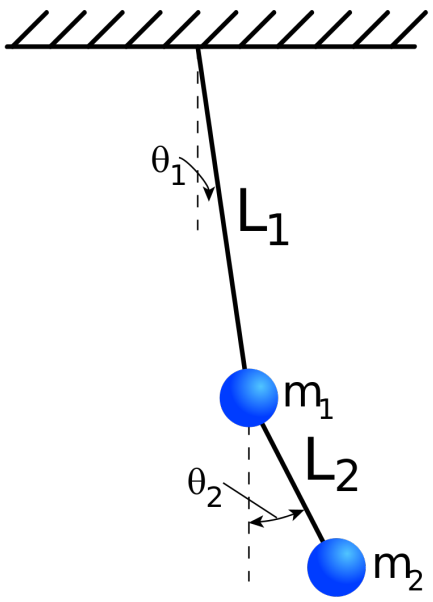

Figure 1: *The Acrobot Problem*: The object is to raise the lower link of the pendulum to a certain height by continuously apply a torque.

1.2 On the LP Formulation

I am interested in section 4 in the paper. It seems to fit real world examples better. The main theorem in the paper is that for a policy π to be optimal under a reward function R, the reward function should, for all states s and actions a, satisfy:

$$\mathbf{E}_{s' \sim T(s,\pi(s),.)}[V^{\pi}(s')] \geq \mathbf{E}_{s' \sim T(s,a,.)}[V^{\pi}(s')] \tag{1}$$

The problem is that this characterization admits some trivial solutions. For example under a constant reward function (i.e. $R(S) = c$), any policy would be optimal. In this case, there is an equality in (1). To avoid this, we'll find reward functions which maximise the sum of the following equation over all s and a.

$$\mathcal{L}(s,a) = \mathbf{E}_{s' \sim T(s,\pi(s),.)}[V^{\pi}(s')] - \mathbf{E}_{s' \sim T(s,a,.)}[V^{\pi}(s')] \tag{2}$$

Ok, I lied a bit over here. Indeed we are going to maximize the above objective, but in a soft sense. We are going to be constrained by how we choose our family of reward functions. It may be possible that no reward function from the family suffices. To make amends, in our search, we'll penalize those reward functions which violate (1).

Now, I'll copy-paste the LP objective presented by the paper and then explain it subsequently:

$$\begin{aligned} \max \sum_{s \in S_0} \min_{a \in A \setminus \{\pi(s)\}} & p(\mathcal{L}(s,a)) \\ \text{s.t. } & |\alpha_i| \leq 1 \text{ for } i = 1 \cdots d \end{aligned} \tag{3}$$

$\mathcal{L}(s,a)$ is given by (2).

1.2.1 Reward function family

The family of reward functions that we consider are linear combinations of d basis functions ϕ_i's.

$$R(s) = \sum_{i=0}^{d} \alpha_i \phi_i(s) \tag{4}$$

1.2.2 Value function are linear over a fixed basis

Due to the choice of our family of reward functions, we get the value function for R is linear over a fixed basis. Following the notation in the paper, let V_i^π denote the value function with ϕ_i as the reward function.

$$V^\pi(s) = \sum_{i=0}^{d} \alpha_i V_i^\pi \tag{5}$$

To see that this is the case, observe that V^π can be written as:

$$V^\pi(s) = \sum_{k=0}^{\infty} \sum_{s' \in S} \gamma^k P(s'|\pi, k) R(s') \tag{6}$$

Where $P(s'|\pi, k)$ denotes the probability that the agent will reach state s' starting from s in k steps. Now, by substituting (4) in (6), we get:

$$\begin{aligned} V^\pi(s) &= \sum_{k=0}^{\infty} \sum_{s' \in S} \gamma^k P(s'|\pi, k) \sum_{i=0}^{d} \alpha_i \phi_i(s') \\ &= \sum_{i=0}^{d} \alpha_i \sum_{k=0}^{\infty} \sum_{s' \in S} \gamma^k P(s'|\pi, k) \phi_i(s') \\ &= \sum_{i=0}^{d} \alpha_i V_i^\pi(s) \end{aligned} \tag{7}$$

In the derivation, first we rearrange the order of the summation. Then we re-apply the definition of value function as present in (6). This makes sure that we can apply LP on V^π. To do so, we pre-compute the basis for V^π using standard value function approximation algorithms present in [SB11].

1.2.3 S_0, a sample of states

Since we are dealing with a large number of states, it is very hard to enforce the constraint in (1) over all the states. Instead, we'll sample a finite set of states and try to enforce the constraint over them.

1.2.4 Penalty function

Since we are using a restricted set of functions, it may be possible that if we require (1) to be applicable at all the sampled states, then the LP problem becomes infeasible. So instead, when the constraint is violated, we'll penalize by doubling the cost of the violation.

$$p(x) = \begin{cases} x & x \geq 0 \\ 2x & x < 0 \end{cases} \tag{8}$$

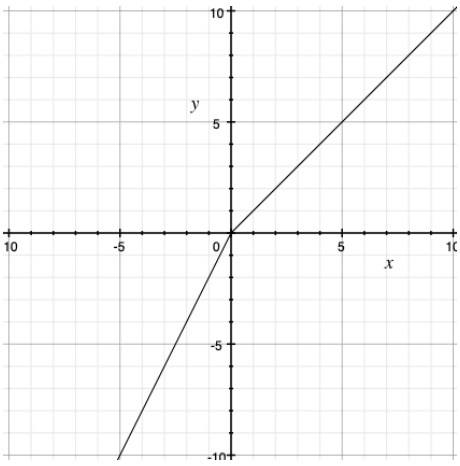

Figure 2: $p(x) = \min(x, 2x)$: Plot of the penalty function. When $\mathcal{L}$ is less than 0, we penalize twice as much.

1.2.5 LP Objective and Constraints

As a detour, let us see how to solve the following problem using LP:

$$\begin{aligned} &\max\{\min(2x+3, 5x-7, -2x+5)\} \\ &\text{s.t.} \quad -10 \leq x \leq 10 \end{aligned} \tag{9}$$

To set this in a form relying only a linear objective and linear constraints, we'll introduce a new variable t and transform the linear program to the follows:

$$\begin{aligned} &\max t \\ \text{s.t.} \quad -10 \leq x \leq 10, \\ t \leq 2x+3, \\ t \leq 5x-7, \\ t \leq -2x+5 \end{aligned} \tag{10}$$

t will be maximized when it reaches one of its upper bounds and will equal exactly $\min(2x+3, 5x-7, -2x+5)$ for that value of x.

We'll use this exact technique. To start of, notice that $p(x) = \min(x, 2x)$. With this, 3 can be written as:

$$\begin{aligned} &\max \sum_{s \in S_0} \min_{a \in A \setminus \{\pi(s)\}} \min(\mathcal{L}(s,a), 2\mathcal{L}(s,a)) \\ &\text{s.t. } |\alpha_i| \leq 1 \text{ for } i = 1 \cdots d \end{aligned} \tag{11}$$

Now we are in good shape because we can apply the same transformation that we applied in 9. For each $s \in S_0$, we introduce

a variable b_s. We then put less-than-equal-to constraints on b_s to make sure that the value it attains satisfies the minimum over all those quantities.

Now let us expand $\mathcal{L}$ so that we can see it's connection with the α_i's.

$$\begin{aligned}
\mathcal{L}(s,a) &= \mathbf{E}_{s' \sim T(s,\pi(s),.)}[V^\pi(s')] - \mathbf{E}_{s' \sim T(s,a,.)}[V^\pi(s')] \\
&= \sum_{i=0}^{d} \alpha_i V_i^\pi(\mathsf{next}(s,\pi(s))) - \sum_{i=0}^{d} \alpha_i V_i^\pi(\mathsf{next}(s,a))) \\
&= \sum_{i=0}^{d} \alpha_i (V_i^\pi(\mathsf{next}(s,\pi(s))) - V_i^\pi(\mathsf{next}(s,a))) \\
&= \sum_{i=0}^{d} \alpha_i \beta_{sa}
\end{aligned} \tag{12}$$

Recall that V_i^π are the basis functions that we have pre-computed (albeit approximately) using the reward bases. Also, since the MDPs I use in my experiment are deterministic, we can simply use the deterministic transition (provided here by the function next). Finally, I have replaced the term $V_i^\pi(\mathsf{next}(s,\pi(s))) - V_i^\pi(\mathsf{next}(s,a))$ by β_{sa} to minimize clutter.

Time to see the LP in all its glory:

$$\begin{aligned}
&\max \sum_{s \in S_0} b_s \\
&\text{s.t. } |\alpha_i| \leq 1 \text{ for } i = 1 \cdots d, \\
&\forall s \in S_0, a \in A \setminus \{\pi(s)\} : b_s \leq \sum_{i=0}^{d} \alpha_i \beta_{sa}, \\
&\forall s \in S_0, a \in A \setminus \{\pi(s)\} : b_s \leq 2 \sum_{i=0}^{d} \alpha_i \beta_{sa}
\end{aligned} \tag{13}$$

In all, there are $2|S_0|(|A|-1) + 2d$ constraints. With this out of the way, lets check out what I did.

2 Experiments

2.1 Getting a Policy

Initially I tried REINFORCE [Wil92], a policy gradient algorithm to learn an optimal policy for Acrobot and the Mountain Car problem.

It wasn't working well and I was growing more and more restless. I thought why not just solve the problem manually and record the trajectory and then learn a model to regress to it.

Using arrow keys, I solved both the problems and saved the trajectory. Next, I trained a neural network to classify the which action to take based on the state using my trajectory as the training data.

Unsurprisingly, this worked well and I was able to get a stationary policy for these problems.

2.2 Approximating Value Functions

I coded up three algorithms.

1. Gradient Monte Carlo
2. Semi-Gradient TD(0)
3. TD(1)

I don't have anything interesting to say about these algorithms. I finally ended up using TD(0) for MountainCar and TD(1) for Acrobot.

2.3 Mountain Car

The episode terminates when the car reaches the terminal state. We can interpret this as the agent getting a 0 reward after reaching the terminal state. This behaviour can be adequately captured by a step function. Let $\mathbf{1}[\alpha, \beta]$ denote such a step function:

$$\mathbf{1}[\alpha, \beta](x) = \begin{cases} 0 & \alpha \leq x \leq \beta \\ -1 & \text{else} \end{cases} \tag{14}$$

I chose the family of reward functions to be a linear combination of such equitably distributed step functions. The width of each step is 0.2 units.

$$R((x, v)) = \sum_{i=0}^{7} \alpha_i \mathbf{1}[-1.2 + 0.2i, -1.0 + 0.2i](x) \tag{15}$$

Next, to sanity check that value function approximation is working correctly, I calculate the values under the trained policy with the original rewards i.e. -1 at each step till termination. A linear model for the value function is used. Refer to Figure 4 for the result.

This plot looks reasonable to me because although a constant per-step reward of -1 is being given, from the positions closer to the goal state, the number of time steps left in the horizon would be fewer and their expected discounted returns would be lower.

Using these value function approximations, I set up the LP objective, exactly as described. The reward landscape that I obtained can be seen in Figure 5.

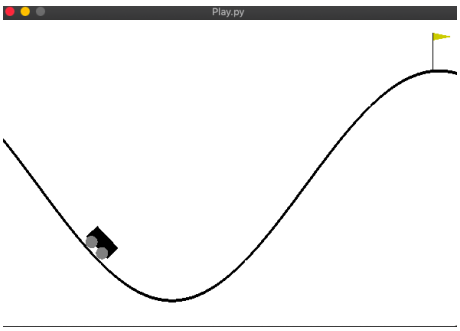

Figure 3: *The Mountain Car Problem*: The objective is to guide the car to the flag by propelling either to the left or the right. A reward of -1 is given at each step till the terminal step at which the episode ends. The state of the agent is (x, v) where x is the position of the car and v is the velocity in the x-direction. x takes values between -1.2 and 0.6, while v lies between -0.07 and 0.07.

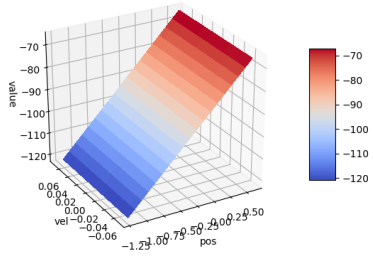

Figure 4: *Value Function Approximation*: Since the episode terminates as the x-position reaches 0.6, it makes sense that positions close to this terminal state will have higher value then those further away. Notice that the approximation is independent of the velocity.

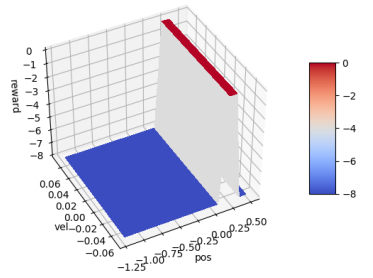

Figure 5: *Reward Landscape*: Quite close to what you would expect! High reward for when the car is close to the goal state and low reward otherwise. There is a dip at the end which seems inexplicable to me. I can live with that.

2.4 Acrobot

As earlier, this is an episodic task. The goal is to make the tip of the lower link reach the grey line. To do this, the agent has the ability to either give clockwise / counter-clockwise torque or do nothing.

The state that is available to the agent is a 6-tuple.

$$(cos(\theta_1), sin(\theta_1), cos(\theta_2), sin(\theta_2), \dot{\theta_1}, \dot{\theta_2}) \tag{16}$$

Where θ_1, θ_2 are as described by 1. To reiterate:

1. $\theta_1 \in [-\pi, \pi]$, is the angle that the top link makes with the vertical.
2. $\theta_2 \in [-\pi, \pi]$, is the angle that the bottom link makes with respect to the top link.

A reward of 0 is given upon reaching the terminal states, else, for each step, the agent incurs a reward of -1. The reward landscape can be seen in Figure 7. Although the state is a 6D vector, the reward only depends on θ_1, θ_2.

Again, the reward function bases were chosen as step functions, equitably distributed in the interval $[-\pi, \pi] \times [-\pi, \pi]$. In Figure 8, we see that the reward computed by the algorithm are quite random and don't match the actual reward landscape at all. I suspect that this is because I used a linear model to approximate the value function. I did this because it was the easiest thing to do and because I'm bound by computational resources.

For now, since I wasn't able to fix this, I'm leaving these experiments without a conclusion.

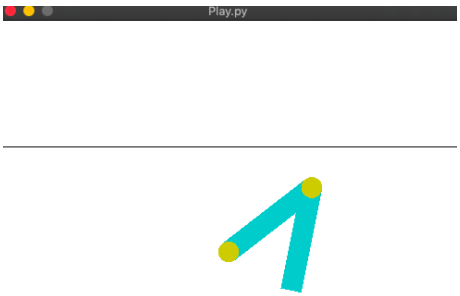

Figure 6: *Acrobot Environment*: The goal is to make the tip of the lower link reach the grey line.

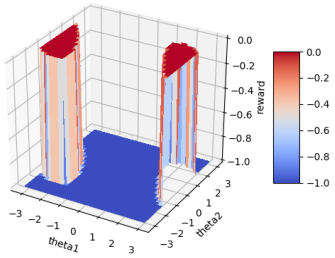

Figure 7: *Reward Landscape*: Reward as function of θ_1 and θ_2.

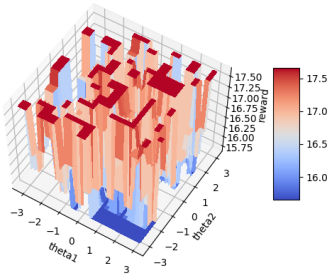

Figure 8: *Estimated Reward*: Not a good result. Doesn't match the expected reward landscape at all.

3 Discussion

Obviously, one crucial area of study is why this algorithm, as I have implemented doesn't work for Acrobot. The code used can be found at https://github.com/Vrroom/IRL. Experimenting with better and more robust value function approximation methods seems to be the next step.

Second thing that I'm interested in is how do we apply this algorithm on real world phenomena. How can we use this algorithm to concretely unravel human behaviour.

One idea that I had in mind was what if we could log all the websites that we visit during a day and then how which websites we visit helps us stay productive.

For example, after a long stretch of work, I may choose to reward myself by a video of cats (cats don't work for me but seem to be the popular choice).

For example, let us say we categorize all websites into two categories.

1. Work Related
2. Entertainment

And at each time step, we have two actions:

1. Stay in this category.
2. Switch to a different category.

Then by tracking our activity and doing Inverse Reinforcement Learning, we'll find out how our brain's productivity depends on how we give ourselves breaks.

References

[Wil92] Ronald J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Mach. Learn.* 8.3–4 (May 1992), pp. 229–256. ISSN: 0885-6125. DOI: 10.1007/BF00992696. URL: https://doi.org/10.1007/BF00992696.

[NR00] Andrew Y. Ng and Stuart J. Russell. "Algorithms for Inverse Reinforcement Learning". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 663–670. ISBN: 1558607072.

[SB11] Richard S Sutton and Andrew G Barto. "Reinforcement learning: An introduction". In: (2011).

[Bro+16] Greg Brockman et al. *OpenAI Gym*. cite arxiv:1606.01540. 2016. URL: http://arxiv.org/abs/1606.01540.