INVERSE REINFORCEMENT LEARNING ON GRIDWORLD

YIFEI LI [LIYIFEI@SEAS], ZIYU GUO [ZIYUGUO@SEAS], ZEYU CHEN [CHENZEY@SEAS]

ABSTRACT. This work compares Reinforcement Learning (RL) and Inverse Reinforcement Learning (IRL) models on different Gridworld environments in order to explore the difference of them and find the limitations and the potential improvements of Inverse Reinforcement Learning algorithm. By formulating it as an optimization problem, we extra a reward function given the trained expert policy to mimic the observed behavior. We shows empirically that the while IRL can generate a similar policy, it's not likely the identical policy and thus is sub-optimal. The more complex the scenario is, the worse the IRL performs. As for the further work, it's necessary to research more how to define the optimal policy and to numericalize and compare the policy difference.

1. PREFACE

We were in the half way of the old project Reinforcement Learning in Computer Systems (To see what we have done, click *the link of GitHub repo*) but unfortunately the API is harder than we thought and it's quite intractable to get a decent loss result despite via different reinforcement learning approaches such as A2C and DDPG. Therefore, we switched to this project.

The overall contributions are listed as below:

- Yifei Li: Code, Experimental Results, Discussion, and Future Work (also, DDPG in old project)
- Ziyu Guo: Theory, Approach, and Discussion (also, A2C in old project)
- Zeyu Chen: Introduction, Background, Related Work, and Future Work (also, DQN/DDQN in old project)

2. INTRODUCTION

In this project, Reinforcement Learning (RL) and Inverse Reinforcement Learning (IRL) are explored and compared. RL is learning from interaction to achieve a goal like learning the optimal policy of an agent in a given environment. And IRL is the process of learning an expert's reward function by observing the optimal policy of the expert.

For the RL part, we generate a Markov Decision Process (MDP) environment we have used in homework3 problem1 and implement the code of the initialization and estimation portions based on value iteration algorithm. For the IRL part, based on the IRL Linear Programming (LP) formulation, we carry out the IRL algorithm to extract reward functions to compute the optimal policy of agent using the value iteration algorithm. The heatmap of the optimal state values and optimal actions displayed as arrows of two reward functions (based on a basic gridworld environment and an obstacle gridworld environment)are displayed.

Finally, we explore the adjustable penalty coefficient in the algorithm to find the optimal combination of the coefficient to maximum accuracy.

3. BACKGROUND

In a traditional RL setting, the goal is to learn a decision process to produce behavior that maximizes some predefined reward function. RL is learning from interaction to achieve a goal like learning the optimal policy of an agent in a given environment.

IRL, as described by Andrew Ng and Stuart Russell in 2000 [2], flips the problem and instead attempts to extract the reward function from the observed behavior of an agent. In the IRL framework, the task is to take a set of human-generated data(expert data) and extract an approximation of that expert's reward function for the task. Of course, this approximation necessarily deals with a simplified model. Still, much of the information necessary for solving a problem is captured within the approximation of the true reward function. As Ng and Russell put it, "the reward function, rather than the policy, is the most succinct, robust, and transferable definition of the task," since it quantifies how good or bad certain actions are. Once we have the right reward function, the problem is reduced to finding the right policy, and can be solved with standard reinforcement learning methods.

The main problem when converting a complex task into a simple reward function is that a given policy may be optimal for many different reward functions. That is, even though we have the actions from an expert, there exist many different reward functions that the expert might be attempting to maximize. Some of these functions are just silly: for example, all policies are optimal for the reward function that is zero everywhere, so this reward function is always a possible solution to the IRL problem. But for our purposes, we want a reward function that captures meaningful information about the task and is able to differentiate clearly between desired and undesired policies.

To solve this, we can formulate inverse reinforcement learning as an optimization problem. We want to choose a reward function for which the given expert policy is optimal. But given this constraint, we also want to choose a reward function that additionally maximizes certain important properties.

4. Related Work

As described by Andrew Ng and Stuart Russell in 2000 [2], First, consider the set of optimal policies for a Markov Decision Process (MDP) with a finite state space S, set of actions A, and transition probability matrices P_α for each action. They proved that a policy π given by $\pi(s) \equiv a_1$ is optimal if and only if for all other actions a , the reward vector (which lists the rewards for each possible state) satisfies the condition

$$(P_{a_1} - P_a)(I - \gamma P_{a_1})^{-1} R \succeq 0$$

It shows that we can efficiently select the "best" reward function for which the policy is optimal, using linear programming algorithms. This lets the authors formulate IRL as a tractable optimization problem, where we're trying to optimize the following heuristics for what makes a reward function "fit" the expert data well:

(1) Maximize the difference between the quality of the optimal action and the quality of the next best action (subject to a bound on the magnitude of the reward, to prevent arbitrarily large differences). The intuition here is that we want a reward function that clearly distinguishes the optimal policy from other possible policies.
(2) Minimize the size of the rewards in the reward function/vector. Roughly speaking, the intuition is that using small rewards encourages the reward function to be simpler, similar to regularization in supervised learning. Ng and Russell choose the L1 norm with an adjustable penalty coefficient, which encourages the reward vector to be non-zero in only a few states.

Other useful heuristics may exist. In particular, later work, such as Maximum Entropy Inverse Reinforcement Learning [1], and Bayesian Inverse Reinforcement Learning [3], are two examples of different approaches that build upon the basic IRL framework.

In this work, we focus on Andrew Ng and Stuart Russell's heuristic and may explore other heuristics in the future.

5. Approach

In RL algorithm, a MDP model contains the state space S, action space A, transfer probability $P_{sa}(s^{'})$ which indicates the probability of reaching state $s^{'}$ from state$s$ after taking action a, the decay factor γ and the reward function R. The goal is to find the optimal policy π. While in IRL, we are given π and search for R. Recall that in RL, V^π and Q^π are defined as

$$V^\pi(s) = R(s) + \gamma \sum_{s'} P_{s\pi(s)}(s') V^\pi(s') \tag{1}$$

$$Q^\pi(s,a) = R(s) + \gamma \sum_{s'} P_{sa}(s') V^\pi(s') \tag{2}$$

Also, from the Bellman Optimality we know

$$\pi(s) \in \arg\max_{a \in A} Q^\pi(s,a) \tag{3}$$

Now we shall find the characteristics of feasible solutions. Let's assume that $\pi(s) = a_1$ and rewrite equation (1) as $\boldsymbol{V}^\pi = \boldsymbol{R} + \gamma \boldsymbol{P}_{a_1} \boldsymbol{V}^\pi$, thus

$$\boldsymbol{V}^\pi = (\boldsymbol{I} - \gamma \boldsymbol{P}_{a_1})^{-1} \boldsymbol{R} \tag{4}$$

Similarly, we can rewrite equation (3) and derive as

$$a_1 \equiv \pi(s) \in \arg\max_{a \in A} \sum_{s'} P_{sa}(s') V^\pi(s') \quad \forall s \in S \tag{5}$$

$$\Leftrightarrow \sum_{s'} P_{sa_1}\left(s'\right) V^{\pi}\left(s'\right) \geq \sum_{s'} P_{sa}\left(s'\right) V^{\pi}\left(s'\right) \quad \forall s \in S, a \in A \tag{6}$$

$$\Leftrightarrow \boldsymbol{P}_{a_1} \boldsymbol{V}^{\pi} \succeq \boldsymbol{P}_a \boldsymbol{V}^{\pi} \quad \forall a \in A \backslash a_1 \tag{7}$$

$$\Leftrightarrow \boldsymbol{P}_{a_1}\left(\boldsymbol{I}-\gamma \boldsymbol{P}_{a_1}\right)^{-1} \boldsymbol{R} \succeq \boldsymbol{P}_a\left(\boldsymbol{I}-\gamma \boldsymbol{P}_{a_1}\right)^{-1} \boldsymbol{R} \quad \forall a \in A \backslash a_1 \tag{8}$$

Which gives the first restriction on solutions. However, there are some solutions that satisfy equation (8) but do not meet our anticipation: for example, $R=0$ would meet criteria (8) anyway. One natural way to narrow our search space is to find R that yields the most distinguishable π. In another word, we want to maximize the sum of the differences between the quality of the optimal action and the quality of the next-best action.

$$\sum_{s \in S}\left(Q^{\pi}\left(s, a_1\right)-\max_{a \in A \backslash a_1} Q^{\pi}(s, a)\right) \tag{9}$$

We can also use equation (2) and (4) to rewrite (9) and get our second restriction

$$\text{maximize} \quad \textstyle\sum_{i=1}^{N} \min_{a \in\left\{a_2, \ldots, a_k\right\}}\left\{\left(\boldsymbol{P}_{a_1}(i)-\boldsymbol{P}_a(i)\right)\left(\boldsymbol{I}-\gamma \boldsymbol{P}_{a_1}\right)^{-1} \boldsymbol{R}\right\} \tag{10}$$

Another idea is that we should keep R small. By intuition, if we scale a reward map by a scalar, the ratio between different states' value should keep the same and thus yields approximately the same optimal policy. To achieve this, we can add a penalty factor $-\lambda\|\boldsymbol{R}\|_1$ on (10). Further more, we can put a scale limit on values of R:

$$\left|\boldsymbol{R}_i\right| \leq R_{\max }, i=1, \ldots, N \tag{11}$$

To make our implementations of restrictions easier, we use t_i to indicate boundaries of $\left(\boldsymbol{P}_{a_1}(i)-\boldsymbol{P}_a(i)\right)\left(\boldsymbol{I}-\gamma \boldsymbol{P}_{a_1}\right)^{-1} \boldsymbol{R}$ and u_i to indicate boundaries of $\|\boldsymbol{R}\|$. Putting all these together, we have a Linear Programming problem:

$$\left\{\begin{array}{c}
t_i+\left(\boldsymbol{P}_{a 1}(i)-\boldsymbol{P}_a(i)\right)\left(I-\gamma \boldsymbol{P}_{a 1}\right)^{-1} \boldsymbol{R} \leq \mathbf{0} \\
\left(\boldsymbol{P}_{a 1}-\boldsymbol{P}_a\right)\left(I-\gamma \boldsymbol{P}_{\boldsymbol{a} 1}\right)^{-1} \boldsymbol{R} \leq 0 \\
-\boldsymbol{u}+\boldsymbol{R} \leq \mathbf{0} \\
-\boldsymbol{u}-\boldsymbol{R} \leq \mathbf{0} \\
\boldsymbol{R} \leq R_{\max } \\
-\boldsymbol{R} \leq R_{\max }
\end{array}\right. \tag{12}$$

Thus we can easily construct matrices and feed them to LP problem solver to find the optimal solution of R function.

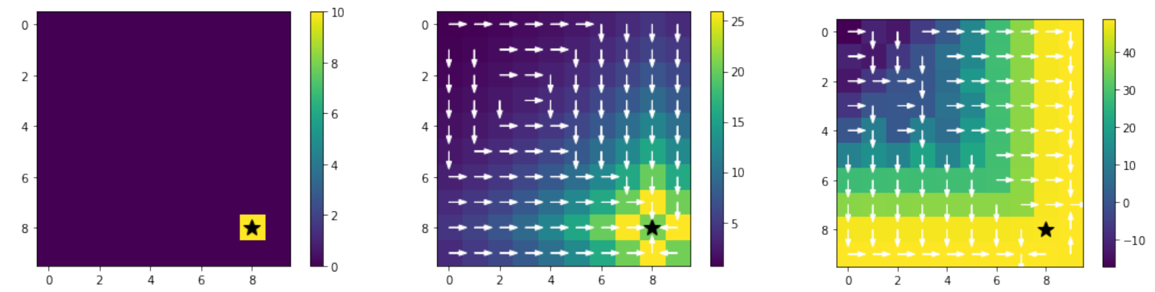
6. Experimental Results

Following the work of problem 1 of homework 3, we try to compare the rendering policy from RL and IRL. Here, we introduce two kinds of 10×10 gridworlds. The first one is called *basic gridworld*, that is, the simplest gridworld without anything but one terminal. The other is called *obstacle gridworld*, which, besides the terminal, contains some obstacles with negative rewards. In both of them, we set the moving reward as 0, terminal value as 10, and obstacle value as -10. The terminal is notated as a black star and the obstacles are marked as red crosses.

Figure 1 shows the initial value heatmap of the basic gridworld and its optimal state values and final policy rendered by RL and IRL respectively. Figure 2 shows the similar results but with respect to the obstacle gridworld. Notes that the heatmap visualizes the value magnitude of each cell by hue and the colorbar ranges are various and thus cannot be compared among different plots directly.

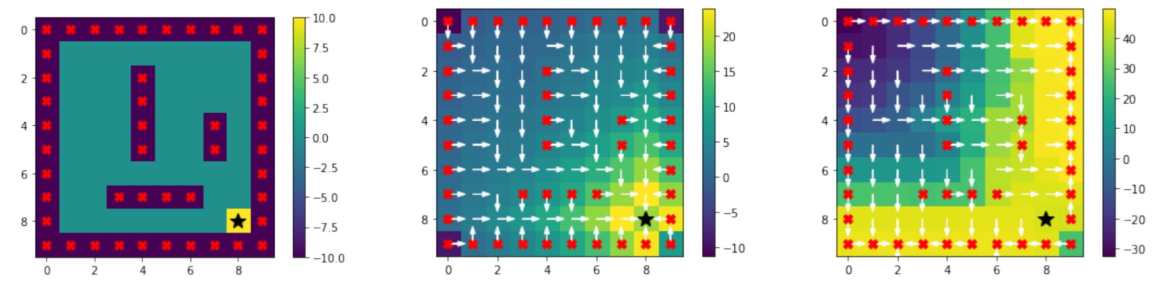
It's obvious that the policy generated iteratively from the RL are optimal in both gridworlds. The values around the terminal and obstacle gradually increase and decrease respectively and the arrow patterns lead to the shortest path with the least cost. Then the IRL algorithm tries to extract the reward functions via the optimal behavior of the expert at each state, here it's from the policy we generated above. In detail, we feed the optimal policy to the algorithm following the aforementioned approach and pick the best parameter λ by comparing the accuracy of different λ candidate as showed in Figure 3. Here, we let λ be 0.59 and 0.15, corresponding to the highest accuracy 76% and 85% respectively in the two gridworlds. The accuracy is computed as the consistency rate between the optimal policy given by RL and the new policy generated by IRL.

By the ultimate IRL formula, we render the best optimal state values demonstrated in heatmap (Part C of Figure 1 and 2). Overall speaking, the hue gradient is largely the same as the optimal ones, indicating that the IRL works to some extent. However, the local policy is not always the best and the increasing complexity (e.g. the number of obstacles here) makes the policy even worse. For instance, in Figure 2.C, the policy on the bottom boundary goes the opposite way and the agents around the in-the-middle obstacles fail to walk far away from them as soon as possible. Also, the agents around the terminal don't walk towards it directly.



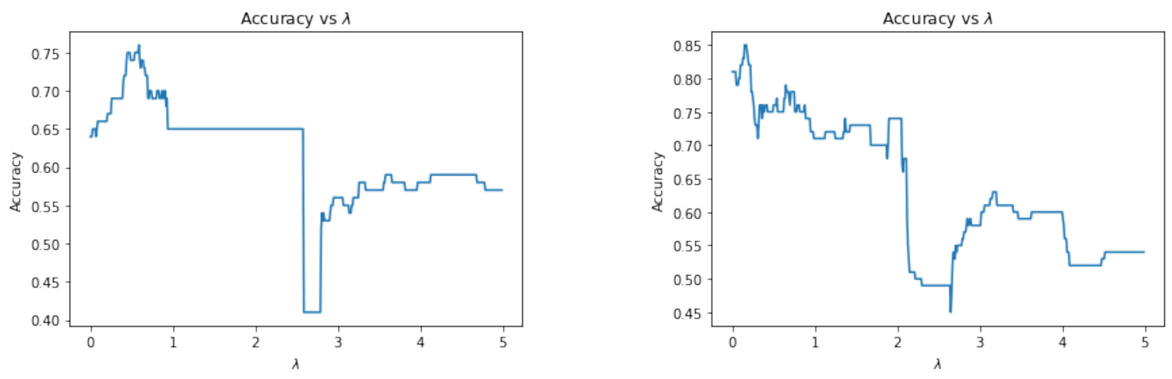
(A) Original Heatmap (B) RL Policy (C) IRL Policy

FIGURE 1. Basic Gridworld Result



(A) Original Heatmap (B) RL Policy (C) IRL Policy

FIGURE 2. Obstacle Gridworld Result



(A) Basic Gridworld (B) Obstacle Gridworld

FIGURE 3. Accuracy vs Lambda

Now we try to use some tricks to improve the IRL policy. Since IRL policy seems to be insensitive to the obstacles, we try to decrease the obstacle value from -10 to -100. But the policy showed in Figure 4.A is even worse then the ones with less-intense obstacles. We also tried the slight obstacle value -1 but get the same plot as Figure 2.C. Also, since we observe that small λ value usually yields good accuracy, then we try to remove the weight decay-like penalty term $-\lambda\|R\|_1$ in the maximization formula (that is, let $\lambda = 0$). The policy is showed in Figure 4.B and not much different than before.

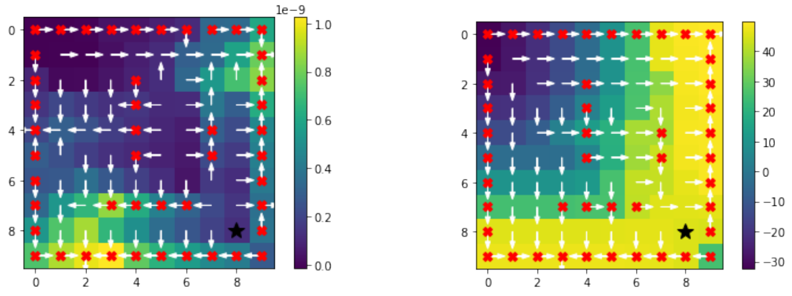

(A) Intense Obstacle Value (B) No Penalty

FIGURE 4. Obstacle Gridworld Variants

7. DISCUSSION

Our experiment shows that while IRL can generate a similar policy based on the extracted reward functions of the optimal policy, it's not likely the identical policy and thus is sub-optimal. The more complex the scenario is (e.g. the increasing number of obstacles), the harder IRL to mimic the optimal policy of RL. The process of reward extraction is quite heuristic and need more analysis and explanation.

A drawback is the way to compute the accuracy of λ candidates, since we directly compare the new policy with the old policy deterministically, we miss the scenario that different policies may be equally optimal. For instance, on the top-left state of the terminal, it doesn't matter the direction is either first-right-then-down or first-down-then-right as long as there's no obstacle. But current algorithm may force to pick one as optimal while penalizing the other option. We may compare the policy effectiveness by counting the cost-to-go in a narrow domain but once the distance is large, it's quite hard to know whether the two next steps are equally efficient or not without enumerating all the path combinations.

8. FUTURE WORK

Since there are other useful heuristics to implement IRL algorithm like Entropy Inverse Reinforcement Learning, and Bayesian Inverse Reinforcement Learning, we may explore different kinds of heuristics to see if IRL can be improved.

Furthermore, the foundational methods of inverse reinforcement learning is able to achieve the results by leveraging information gleaned from a policy executed by a human expert. However, in the long run, the goal is for machine learning systems to learn from a wide range of human data and perform tasks that are beyond the abilities of human experts. In the context of human experts, it's clear that while experts may be able to provide a baseline level of performance that RL algorithms could learn from, it will rarely be the case that the expert policy given is actually the optimal one for the environment. The next problem to solve in IRL would be to extract a reward function without the assumption that the human policy is optimal.

How to compare the current policy and the optimal one is also extremely challenging. Since the optimal policy is not always unique and different step combinations may lead to the same result. In the real world, this problem is more significant and intractable. Like, when feeling hungry, humans can eat apples or oranges. But the trained policy may think only eating the apple is the best. We may abstract them to the higher level like "eating some food" is optimal.

Last but not least, even if we know which policy is worse, how to numericalize the policy difference is another challenging problem. And we may need to choose or fine-tune different scoring schemes in different scenarios.

Many recent papers have aimed to do just this which use fully convolutional neural networks to approximate reward functions or apply neural networks to solve inverse optimal control problems in a robotics context. We may explore these inverse reinforcement learning methods to work with deep learning systems.

References

[1] J.Andrew Bagnell Brian D. Ziebart, Andrew Maas and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, volume 1, page 2, 2008.
[2] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.
[3] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *IJCAI*, volume 07, page 2587, 2008.