Better Gaming: Policy Control with Reward Approximation

Dan Pechi, Jeremy Shih, Rui Sun

1 Project Overview

In this project, we will teach an agent to not only generate the optimal policy of a game, but also learn the rewards itself in the first place.

Almost all kinds of games have a reward system of some sort. Whether the player is a human or an intelligent agent, the actions executed are optimal only when the reward system is taken into considerations. While most game solver projects provide the full gaming environment to the agent, we aim to enable the agent to learn the rewards itself by observing demonstrations by human players. Our preliminary goals are as follows:

- Aim 1: Construct a problem architecture of a GridWorld with puddles. This includes the model of the MDP process and the reward system of the of the GridWorld.
- Aim 2: Construct a inverse reinforcement learning (IRL) agent that approximates rewards through observing demonstrations. This will be done through Maximum Likelihood Inverse Reinforcement Learning
- Aim 3: Construct a reinforcement learning (RL) agent that solves the game with the rewards learned.
- Aim 4: Compare and contrast the ground-truth reward system and the approximated reward system.

Further experimentation include:

- Aim 5: Construct a IRL agent and apply it to Ms. PacMan to approximate a reward function. Then, apply a RL agent to Ms. PacMan using the approximate reward function.
- Aim 6: Construct a RL agent to Ms. PacMan that learns from the ground-truth reward function.
- Aim 7: Compare the scores of using the approximate reward function to the ground-truth reward function.

Thus, we will aim to provide empirical evidence for efficacy of an inverse reinforcement learning agent through a simple GridWorld setting as a benchmark and then a more complex setting of Ms.PacMan.

2 Background and Related Work

Inverse reinforcement learning (IRL) is essentially a reward-estimation problem. Given a Markov Decision Process (MDP) without a reward function and a set of actions, find the reward function that make the set of actions optimal in the given MDP. A relevant topic is Apprenticeship Learning (AL) in which an expert is assumed to be acting to maximize a reward function and have an agent learn a policy based off of the experts actions. If the agent's goal is to learn the reward function, the problem becomes an IRL problem[1].

A reward function is assumed to parameterized by a vector of weights θ, applied to a feature vector for each state-action pair given by the MDP. If each state-action pair is defined as $\phi(s, a)$, then a reward function can be defined as below:

$$R_\theta(s, a) = \theta^T \phi(s, a)$$

Thus, in an AL problem, the expert's reward weights can be defined as θ_E. In AL, the agent, without knowledge of θ_E will try to learn how to behave in a way that maximizes the discounted sum of future rewards from R_{θ_E}. In IRL, the agent will try to approximate R_{θ_E} with its own weights, θ_A.

Several IRL/AL algorithms have been presented and differ not only in their algorithm but also in the objective function they optimize. These include matching the policy of the expert[3], trying to outperform the expert[4] and more. Following, we will define a problem space and will use Maximum Likelihood Inverse Reinforcement Learning (MLIRL) [1] to solve them.

3 Problem Formulation and Technical Approach

In this section, we will briefly explain the mechanism of the Maximum likelihood Inverse Reinforcement Learning (MLIRL), and lay out the settings of the puddle mini-game and the PacMan gaming environments.

MLIRL Agent

MLIRL will be the backbone of the IRL agent. Like Bayesian IRL, it adopts a probability model that uses θ_A to create a value function and then assumes the expert randomizes at the level of individual action choices. It also seeks a maximum likelihood model.

The process by which a hypothesized θ_A induces a probability distribution over action choices and thereby assigns a likelihood to the trajectories in D. First, θ_A provides the rewards from which discounted expected values are derived:

$$Q_{\theta_A}(s, a) = \theta_A^T \phi(s, a) + \gamma \sum_{s'} T(s, a, s') \bigotimes Q_{\theta_A}(s', a')$$

The "max" in the standard Bellman equation is replaced with an operator that blends values via Boltzmann exploration [2]. This approach makes the likelihood (infinitely) differentiable.

One of the challenges of IRL is that, given an expert policy, there are an infinite number of reward functions for which that policy is optimal in the given MDP. Like several other IRL approaches, MLIRL addresses this issue by searching for a solution that not only explains why the observed behavior is optimal, but also by explaining why the other possible behaviors are suboptimal. In particular, by striving to assign high probability to the observed behavior, it implicitly assigns low probability to unobserved behavior.

Algorithm 1 Maximum Likelihood IRL

```
2
1: procedure Choose random set of reward weights $\theta_1$
2:     for $t$ = 1 to $M$ do
3:         Compute $Q_{\theta_t}$, $\pi_{\theta_t}$
4:         $L$ = $\sum_i w_i \sum_{(s,a)\in\xi} log(\pi_{\theta_t}(s,a))$.
5:         $\theta_{t+1} \leftarrow \theta_t + \alpha_t \bigtriangledown L$
6:     Output: Return $\theta_A = \theta_M$
```

Environments

Puddle Mini-game

As a benchmark, besides the start state and an goal state, other states in the puddle mini-game environment would either have a positive reward or a negative reward. The specific underlying settings are as follows:

- State space: A 5x5 gridworld containing puddle and clean grids
- Action space: {left, right, up, down, hold}
- Reward function: {end state:+20, start state:+15, paths:+15, obstacles:-10}

PacMan Game

We would use the Ms.PacMan environment and implementation available from the course. However, because of the complexity of the game, we cannot have a pre-defined reward space for the Ms.PacMan environment. We would leave the "ground-truth" reward function blank and let the IRL agent approximate the best reward function through Apprenticeship learning. The problem settings for the Ms.PacMan are as follows:

- State space: All locations that Ms.PacMan can occupy on the game board
- Action space: {left, right, up, down, hold}
- Reward function: An approximation function that maps each state-action pair in the game to a real number

4 Evaluation and Expected Outcomes

Upon deriving the IRL agent's reward function, this approximate reward function will be compared to that of the ground truth reward function of the RL agent whose actions were observed by the IRL agent. In this sense, the agent will be evaluated on how closely they approximated the reward function. In the case of the GridWorld example, this will take the form of the mean squared error between the GridWorld and the approximations of the two agents to the values assigned to the GridWorld environment. These errors will be tested across multiple experiments.

In the non-tabular case of Ms. PacMan, this is not feasible. Thus, a different metric for the IRL algorithm's performance will be used. The approximate reward function developed by the agent will still be compared to that of the RL agent whose actions were observed by the IRL agent. In this case however, another RL agent will be instantiated that has the approximate reward function of the IRL agent as its reward function. To ensure that this reward function is not updated, any gamma values will be set to 0 so that there is no error correction by the agent. This way the efficacy of the reward function approximated by the IRL algorithm alone can be compared to the reward function of the ground truth RL agent, the gamma value of which will also be set to 0 come testing time. This testing time will be based off of some convergence observed in the reward function approximated by the IRL agent. These two agents, one with the approximate reward function devised by the IRL agent, and the other with the reward function of the agent the IRL agent was observing will be pitted against each other in a Ms. PacMan game. The agents' scores at the end of the game will then be compared across multiple experiments.

In both of these experiments, it is expected that the RL agent will fare better than the IRL agent. Because the IRL agent is basing its decisions off of the behavior of the RL agent, the IRL agent has less information than the RL agent. At best, the IRL agent can replicate the RL agent's reward function, and at worst, it can derive a reward function that is sub-optimal to that derived by its RL counterpart. It is expected that the reward function can be replicated in the easier case of GridWorld, but it will likely be much harder to replicate the reward function in Ms. PacMan. There remains the possibility than in attempting to approximate the RL agent's sub-optimal reward function that the approximation unintentionally proves more robust to the environment of Ms. PacMan, but this is unlikely.

References

[1] Monica Babes-Vroman, Vukosi Marivate, Kaushik Subramanian, and Michael Littman. Apprenticeship learning about multiple intentions. ICML, 2011.

[2] George H. John. When the best move isn't optimal: Q-learning with explo-

ration. Proceedings of the Twelfth National Conference on Artificial Intelligence, 1994.

[3] Gergely Neu and Csaba Szepesvari. Apprenticeship learning using inverse reinforcement learning and gradient methods. Proceedings of the Conference of Uncertainty in Artificial Intelligence, 2007.

[4] Umar Syed, Michael Bowling, and Robert E. Schapire. Apprenticeship learning using linear programming. Proceedings of the Conference of Uncertainty in Artificial Intelligence, 2008.