Better Gaming: Policy Control with Reward Approximation

Dan Pechi, Jeremy Shih, Rui Sun

1 Abstract

Reinforcement learning (RL) algorithms have proved incredibly effective in learning to play video games. However, applying inverse reinforcement learning (IRL) algorithms in these domains has only recently been applied to improve gaming performance. We tested standard IRL algorithms in two gaming environments: a Puddle-Game environment and a Toy-Car environment. In the Puddle-Game, we tested the ability of an IRL algorithm to accurately capture a fixed goal. In Toy-Car, we tested a more complex scenario in which the end goal can be changed by the demonstrator. We found in the Puddle-Game that the reward approximation generated by the IRL agent is proportional to the actual reward function of the Puddle-Game, leading to an optimal policy. The IRL agent is also successful in the more complex obstacle environment, learning to avoid walls entirely.

2 Introduction

Many video games have an underlying reward system that guides a player's policy. Whether the player is a human or an intelligent agent, the actions executed are optimal only when this reward system is taken into consideration. While most experimental setups involve an agent exploring and exploiting the gaming environment to learn this reward system, we aim to limit the agent's reward approximation by only observing expert behavior in the game.

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, IRL algorithms build an approximation of the reward function that makes the set of expert actions optimal in the given MDP. IRL is heavily related to Apprenticeship Learning (AL) in which an expert is assumed to be acting to maximize a reward function. In AL, an agent then approximates a policy based off of the expert's actions. If the agent's goal is to approximate the reward function, the problem becomes an IRL problem [2].

A reward function is assumed to be parameterized by a vector of weights θ, applied to a feature vector for each state-action pair defined by the MDP. If each

state-action pair is defined as $\phi(s, a)$, then a reward function can be defined as follows:

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

Thus, in an AL problem, the expert's reward function 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 and AL algorithms have been presented and differ not only in their underlying mathematical representation, but their objective functions as well. These include matching the policy of the expert [5], trying to outperform the expert [8] and other approaches.

In this report, we conduct two experiments to evaluate the performance of two IRL algorithms on two gaming environments: Maximum Likelihood Inverse Reinforcement Learning (MLIRL) [2] on Puddle-Game and a traditional IRL algorithm proposed by P. Abbeel and A. Ng [1] on Toy-Car.

Puddle Game

The Puddle-Game environment represents a 5 x 5 gridworld with puddle states, a start state, and a goal state. The objective of the agent is to go from the start state to the end state without stepping in the puddle states. The demonstrator must behave accordingly, leading the IRL agent to assign negative rewards to the puddle states and positive rewards to the goal state and other, non-puddle states.

The demonstrator in Puddle-Game can record multiple episodes before the IRL agent is trained. After training with MLIRL, the state is colored according to its estimated reward. We evaluate the results based on the following criteria:

- Reward(puddle states) < 0
- Reward(goal state) $>$ Reward(path states)
- The rewards of puddle states reflect the frequency of the demonstrator's map traversals

Toy-Car Game

In the Toy-Car Game, the demonstrator is given an environment with three types of walls (yellow, red, brown), and one background (black). The objective of the game is dependent on the goal of the demonstrator. The setup of the game is as follows:

- Agent: A green circle equipped with three sensors: left, right, and center.
- States: Each state of the agent has 8 features, consisting of three types.
  - Distance sensor reading (left, middle, right)

  - No. of sensors seeing black/yellow/brown/red color
  - Boolean to indicate a crash/bump into an obstacle

- Rewards: A weighted linear combination of the feature values observed in a frame. The reward r_t in the t^{th} frame is obtained by the weight vector w with the vector of feature values in t^{th} frame, that is

$$r_t = w^T * \phi_t$$

- Actions: Frame progression moves the agent forward a step; the agent can choose to go left, right, or do nothing in which case the agent continues on its current, straight trajectory.

In this setting, we try to teach the agent to aim for five different goals: agents that prefer different wall colors, and agents that prefer crashing or not crashing. As mentioned in the section above, each state of the agent contains 8 features. It would thus be reasonable to assume that agents with different preferences will also have different preferences for these features. Specifically, the color-loving agents should have higher weights on the number of sensors seeing the corresponding colors. The crash-loving and crash-avoidant agents should have the highest and lowest weights for the crash feature, respectively. Our experiment results show the validity of using IRL algorithms for learning a fixed reward function but a less satisfactory performance when the IRL agent needs to adjust its estimation when the reward function is not fixed.

3 Related Work

Inverse reinforcement learning (IRL) was first proposed by Ng et al. [6]. It assumed demonstrations to be optimal, and that the reward function is linear with a set of features. Researchers have expanded on this seminal work with Bayesian IRL and Maximum Entropy IRL. Bayesian methods aim to resolve the ambiguity of the reward function by inferring a posterior distribution over rewards instead of a single function [7]. Maximum Entropy IRL looks for a reward function that matches the expected feature counts, leading to a higher-entropy stochastic policy [14]. More recently, progress in IRL has arisen due to the integration of deep learning with these algorithms. Deep IRL can infer an non-linear reward function, but requires a finite state space with known dynamics[13].

Because of its promise as a means of transfer learning from experts, IRL methods have been actively applied to a wide range of video games. Tastan et al. used IRL to learn policies in first person shooter games [9]. This work tries to distinguish the differences between the models generated by bots and human players. It uses IRL to capture the internal model of expert human players to evaluate the benefits of different actions. Wang et al. uses IRL in playing zero-sum multi-agent games [12]. It considers the situation in which the expert demonstrations are known to be sub-optimal, and proposes an objective

function that directly pits experts against Nash Equilibrium strategies. Tucker et al. also applied IRL to improve video game performance. They proposed a method that integrates convolutional neural networks to an adversarial IRL (AIRL) as its generator and discriminator.

The popular Atari game environment has also been used as a test bed for learning from demonstration. Although they do not use IRL, Hester et al. use deep Q-learning to learn from demonstrations [10]. This method uses demonstrations to bootstrap RL learning against a known reward function, instead of directly learning the rewards from demonstrations. Further progress has been made with active preference learning to Atari games, in which users are asked to select the best of two policies generated from a larger set of policies [11].

4 Technical Approach

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 [4]. 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 explains why the other possible behaviors are sub-optimal. In particular, by assigning high probability to the observed behavior, it implicitly assigns low probability to unobserved behavior.

Projection Method

The Projection Method is a simpler version of the IRL algorithm proposed by P. Abbeel and A. Ng in 2004 ([1]). It is the basis of many popular IRL algorithms more recently proposed. The central component of the algorithm is the integration of **feature expectations**, an expected discounted accumulated feature value vector $\mu(\pi)$:

$$\mu(\pi) = E[\Sigma_{t=0}^{\infty} \gamma^t \phi(s_t|\pi)] \in R^k,$$

Algorithm 1 Maximum Likelihood IRL

procedure CHOOSE RANDOM SET OF REWARD WEIGHTS θ_1
  for t = 1 to M **do**
    Compute Q_{θ_t} , π_{θ_t}
    $L = \sum_i w_i \sum_{(s,a)\in\xi} log(\pi_{\theta_t}(s,a))$.
    $\theta_{t+1} \leftarrow \theta_t + \alpha_t \bigtriangledown L$
  Output: Return $\theta_A = \theta_M$

In which γ is the discount factor and $\phi(s_t)$ denotes the feature vector at state t. Given the demonstration trajectories, the empirical estimate μ_E is defined as

$$\hat{\mu}_E = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{\infty}\gamma^t\phi(s_t^{(i)}).$$

The goal of the algorithm is to find a policy $\tilde{\pi}$ such that $\|\mu(\tilde{\pi}) - \mu_E\|_2 < \epsilon$. The program then is reduced to finding a policy $\tilde{\pi}$ that is as close to μ_E as possible.

Algorithm 2 Projection IRL Algorithm

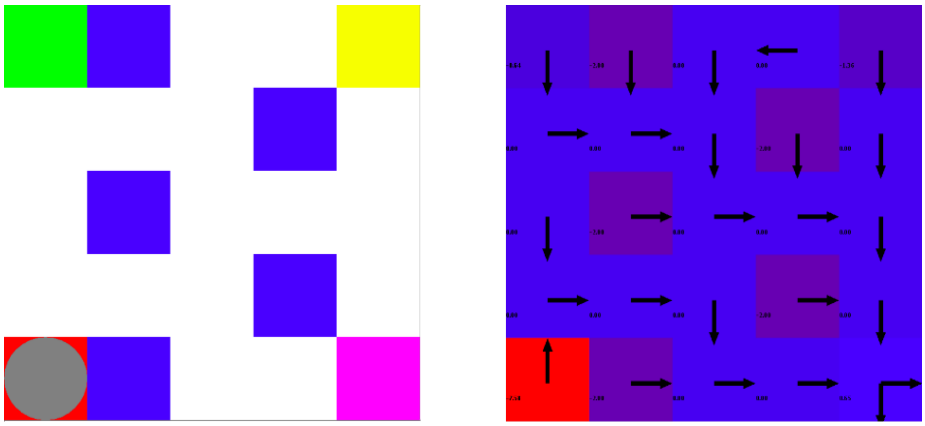
procedure RANDOMLY PICK A POLICY $\pi^{(0)}$
  Compute $\mu^{(0)} = \mu(\pi^{(0)})$ and set i = 1
  while t = 1 to M **do**
    $t^{(i)} \leftarrow \max_{w:\|w\|_2\leq 1} \min_{j\in\{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$
    $w^{(i)} \leftarrow \text{argmax}_{w:\|w\|_2\leq 1} \min_{j\in\{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$
    if $t^{(i)} \leq \epsilon$ **then** Terminate
    Compute $\pi^{(i)}$ using rewards $R = (w^{(i)})^T\phi$
    Compute $\mu^{(i)} = \mu(\pi^{(i)})$
    $i = i + 1$

5 Experiments and Results

Puddle Game

In order to test the ability for an MLIRL agent to infer the optimal reward function, we first let the human player traverse the map and collect 10 trajectories in the gridworld Puddle-Game. After training with 100 training episodes, reward function and the corresponding policy will be shown on the map, as shown in Figure 1.

In figure (b), we set the starting cell to have a negative reward so that the RL agent will not stuck in the starting cell. It is clear that the MLIRL agent assigns negative rewards for all puddle cells and a positive reward for the goal



(a) Puddle-Game map setup

(b) Estimated reward function and policy

Figure 1: Puddle-Game map setup and the estimated rewards and policy inferred by the MLIRL agent. In figure (a), the red cell is the starting point and the pink cell at the bottom-right is the goal cell. Blue cells represent puddle cells. Green and yellow cells represent wall cells.

location. The policy inferred by the RL agent also reflects a clear tendency to go from the bottom-left to the bottom-right, avoiding all puddle and wall cells. It is fair to say that the reward function generated by the MLIRL agent from demonstrations is very close to the true reward function underlying the puddle game.

One thing worth noticing is that in the middle of the training process we found that the MLIRL algorithm is highly dependent on the frequency by which the human player traverses a particular cell. The result in figure (b) was obtained with trajectories of the human player actually going back-and-forth in the path cells to reinforce the positive reward of the path cells and the goal location. If all training trajectories only have the human player going straight from the start to the goal location, the results would be less ideal. Even when the trajectories are all optimal path from start to goal, the MLIRL agent still will not be able to infer a reward function that would generate an optimal policy.

Toy-Car Game

A significant downside to IRL algorithm's is the potential for expensive or time-consuming expert demonstrations. Moreover, these "experts" may not even be readily available in your respective domain. As we saw in our Puddle-Game setup, we had to frame the expert behavior to influence the IRL agent to learn the correct rewards. However, in a more complex environment, this may not be possible and a true expert may be needed. Thus, for the Toy-Car game that utilizes the Projection Method that takes in a feature expectation and finds a policy, we will experiment with editing a preexisting feature expectation

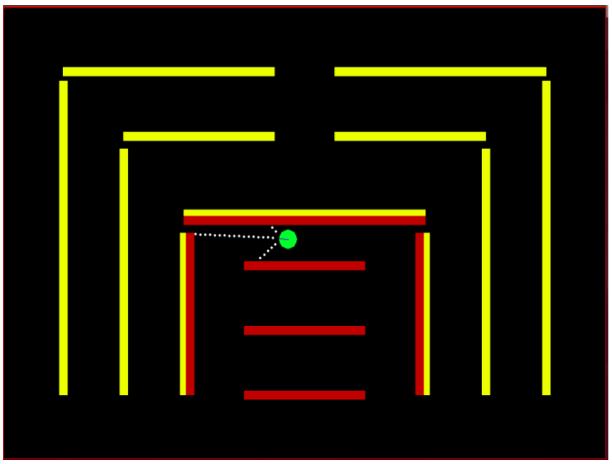


Figure 2: Environment setup of the Toy-Car game.

generated by demonstration to alter the learned policy. These features are defined below:

1. Distance sensor 1 reading
2. Distance sensor 2 reading
3. Distance sensor 3 reading
4. No. of sensors seeing black color
5. No. of sensors seeing yellow color
6. No. of sensors seeing brown color
7. No. of sensors seeing red color
8. Boolean to indicate a crash/bump into an obstacle. (1:crash, 0:alive)

x‘

More specifically, we will take feature expectations of an agent that is supposed to learn to continuously hit the obstacles (defined as the bumping agent) and then invert the sign of the weight on the feature that dictates collision (and define a new agent called nobumping agent). Therefore, our hypothesis is that flipping the sign on the feature that dictates collision should lead to an agent that does not collide with any obstacles. The Toy-Car game is represented in the environment setup of Figure 2 where an agent moves around various obstacles. This is implemented from source code derived from a blog post by Jangir[3].

Below in Table 1 are the results of the feature expectations and learned features for both agents. One thing to take into account is that the algorithm for the nobumping agent has not fully converged. According to Jangir[3], the

algorithm converges within 10-15 iterations. However, the nobumping agent has ran for 18 iterations and still has not converged. Keeping this in mind, we will still analyze the learned results. For a video showing the example of the nobumping agent see **here**.

It is evident that although the agent has learned a negative weight for collision, the agent itself still exhibits colliding behavior. This does not follow our hypothesis and thus we could conclude that changing an individual feature weight does not completely dictate the behavior of the agent. Thus, interactions between the weights of the features may explain more of the behavior than just a single feature and bypassing the expert demonstration effort is not a trivial task.

Table 1: Feature Expectation and Learned Features of Different Behaviors

Behavior	Feature Expectations	Learned Features
Bumping Agent	$\begin{pmatrix} 7.5313 \\ 8.2716 \\ 8.0021 \\ 0.0026 \\ 24.300 \\ 95.962 \\ 15.814 \\ 1553.8 \end{pmatrix}$	$\begin{pmatrix} -0.5892 \\ -0.3672 \\ -0.4660 \\ -0.0299 \\ -0.1528 \\ -0.0368 \\ -0.5239 \\ 0.0256 \end{pmatrix}$
Nobumping Agent	$\begin{pmatrix} 7.5313 \\ 8.2716 \\ 8.0021 \\ 0.0026 \\ 24.300 \\ 95.962 \\ 15.814 \\ -1553.8 \end{pmatrix}$	$\begin{pmatrix} -0.3482 \\ -0.6381 \\ -0.4575 \\ -0.2044 \\ -0.3515 \\ -0.0625 \\ -0.2859 \\ -0.1052 \end{pmatrix}$

6 Conclusion

In the Puddle-Game, the IRL agent's ability to converge to the optimal policy demonstrated the effectiveness of IRL algorithms in simpler environments, however the inability to induce demonstrated behaviors in the case of the Toy-Car environment demonstrates IRL algorithms' shortcomings when environments become more complex. This suggests that IRL agents struggle in their approximation due to this increased complexity.

It should also be noted that to induce an optimal policy in the Puddle-Game, a great deal of exploration had to be demonstrated to the agent that avoided puddles. These trajectories are not part of an optimal trajectory, but

without this instruction, puddles outside of the trajectory would never have been learned. Instituting this exploration in the Toy-Car environment would have proved far more difficult, even as it relates to avoiding walls given the increased complexity of the environment.

Regardless, altering feature expectations proved to be ineffective as a means of bypassing this exploration. The failure in assigning negative and positive weights on non-preferable and preferable features respectively proves overly simplistic. This suggests a more complex interaction of these features in determining optimal behavior.

7 Future Work

The relationship between these aforementioned feature expectations and the policies they induce should be further explored in future work. Furthermore, alternative features may also prove beneficial in improving agent performance. For example, current features in the Toy-Car game are all about the sensor information, further experiments can explore how adding extra environmental information would change the performance of the IRL agent.

The IRL algorithm used in the Toy-Car game is the standard version proposed by Ng et al. in 2000. Future work would involve testing modified versions and other computationally efficient IRL algorithms in the same environment so that the learning advantages of different algorithms can be compared on the same baseline. Aside from the algorithms, more experiments can be carried out in more complex gaming environments, like multi-agent games and games with high-dimensional reward functions.

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. Proceedings of the 21 st International Conference on Machine Learning, 2004.

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

[3] Rishabh Jangir. Apprenticeship learning using inverse reinforcement learning, 2016.

[4] George H. John. When the best move isn't optimal: Q-learning with exploration. Proceedings of the Twelfth National Conference on Artificial Intelligence, 1994.

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

[6] Andrew Y. Ng and Stuart J. Russell. Apprenticeship learning via inverse reinforcement learning. Proceedings of the Seventeenth International Conference on Machine Learning, 2000.

[7] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. Proceedings of the 20th international joint conference on Artificial intelligence, 2007.

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

[9] Bulent Tastan and Gita Sukthankar. Learning policies for first person shooter games using inverse reinforcement learning. Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 2011.

[10] Hester et al. Todd. Deep q-learning from demonstrations. Association for the Advancement of Artificial Intelligence, 2018.

[11] Hester et al. Todd. Deep reinforcement learning from human preferences. Association for the Advancement of Artificial Intelligence, 2018.

[12] Xingyu Wang and Klabjan Diego. Competitive multi-agent inverse reinforcement learning with sub-optimal demonstrations. International Conference on Machine Learning, 2018.

[13] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. arXiv preprint arXiv:1507.04888, 2015.

[14] Brian D. Ziebart, Andrew Maas, Andrew Bagnell, and Dey Anind K. Maximum entropy inverse reinforcement learning. Association for the Advancement of Artificial Intelligence, 2008.