A Proposal For Benchmarking Inverse Reinforcement Learning Agents

CMSC5707 Advanced Topics in Artificial Intelligence

1155160971 - Moin Ahmed Qidwai

Abstract.

Inverse reinforcement learning (IRL) is the task of determining the reward function of an expert agent given it's policy or observations of it's behavior. Inverse reinforcement learning can be both classified as a problem and a set of solutions to that problem. This article first introduces required background information on RL and IRL. It then elaborates on some of the central challenges faced by IRL such as the reward function ambiguity, scalability under large or continuous state sets, incomplete model of the environment or lack of standard benchmarking of solutions. We further propose a benchmarking strategy that can help develop standard testbeds similar to UCI's machine learning repository and OpenAI's Gym library, that are playing significant roles in advancing the progress of supervised and reinforcement learning techniques, respectively. This article concludes with a discussion of potential future extensions to the proposed strategy and some open research questions.

1 Introduction

The most natural form of learning we encounter in the world is likely to be learning by interactions with our environments. When an infant puts something in the mouth that tastes bad, he/she learns through the sensory stimulus to avoid that action in the future. Reinforcement learning is the approach that best encompasses the aforementioned form of learning, in it an agent learns what actions to take in it's environmental context based on reward signals it receives for it's actions. The reward signals that the agent receives must be provided to the system in reinforcement learning, we call the producer of the reward signals the reward function for a environment and set of actions. This reward mechanism is what differentiates reinforcement learning from the other forms of machine learning, specially supervised and unsupervised learning. As such reinforcement learning is said to be the third form of machine learning architecture alongside those two. While reinforcement learning has been quite successful in areas where we know the reward function (for example Chess), it cannot be applied to real world practical

problems where the reward function and/or policy is not well known (for example developing an application to automate someones daily tasks). In such cases, we need to be able to determine the reward function given observations of an expert's behavior. This particular problem is called inverse reinforcement learning (IRL). There are many challenges with regards to IRL, but discussing all of them would be out of the scope of this essay, hence we will restrict our discussion to the issue of the lack of standard benchmarking or testing avenues for IRL models. Quality testbeds allow for quick evaluation of different methodologies along various dimensions leading to isolation of weaknesses, and typically lead to a faster iterative development cycle in the field. For example, UCI's machine learning repository and OpenAI's Gym library are playing significant roles in advancing the progress of supervised and reinforcement learning techniques, respectively [1]. To our knowledge we have not found any other significant proposal for resolving this issue, though it is quite common to use empirical methods of evaluation for IRL solutions as is done by Justin Fu et al [2]. This is partially due to the complexity of creating standardized environments as testbeds. In Chapter 2 we will discuss the theoretical background for RL and IRL, along with concise overview of some of the other challenges faced by IRL. In Chapter 3 we will elaborate on our proposed solution and provide the structure for it while expanding on the potential results. Finally in Chapter 4 we will conclude with outlining the potential benefits of our proposal when compared to empirical methods and outline related future work that can be undertaken to improve upon our approach.

2 Theory And Background

2.1 Reinforcement Learning

2.1.1 Overview

In Reinforcement Learning the learner is not told the "correct" actions, but instead must discover which actions lead to the optimal reward trajectory by trying them. In more complex environments actions may mutate the environment such that the potential subsequent reward changes along with it, in such cases the learner must balance the immediate reward with the delayed reward potential to find the optimal set of actions that lead to the highest overall reward. The trial-and-error search and delayed reward are the two most important distinguishing features of reinforcement learning [3]. A reinforcement learning system has the following core components.

- The Agent: This is the entity that is carrying out the learning task.
- The Environment: This is the context within which the agent is carrying out it's learning task. It can further be divided into the following components.

  - The State Set (S): This is the set of all the possible states that an agent in this environment can be in.

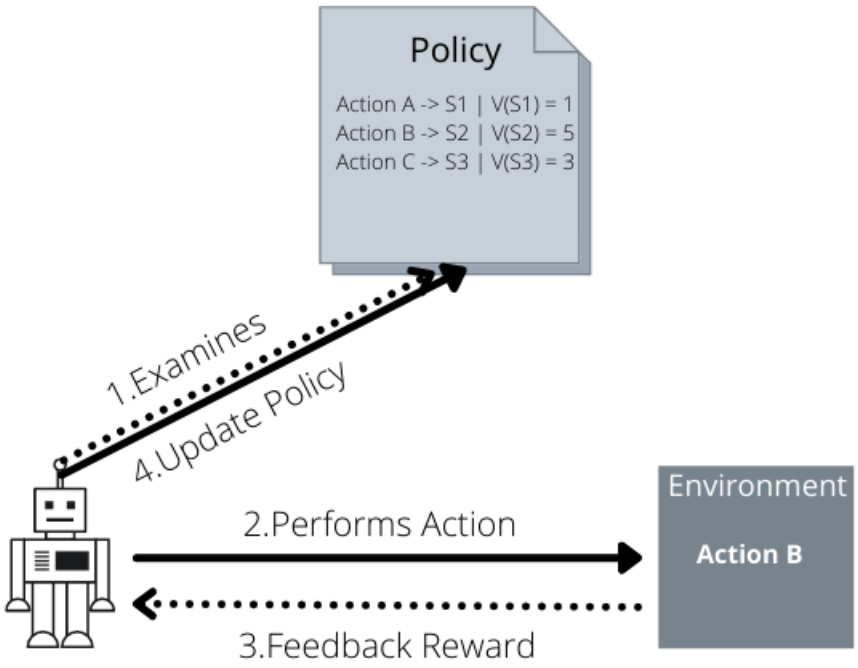
  - The Action Set (A): This is the set of all the possible actions that an agent in this environment can perform.

  - The Model (M): This is an optional concrete implementation of an environment, that allows inferences to be made about how the environment will behave.

- The Policy (π): This is the set of actions that can be taken provided the agent is in a given state, it may also include probabilistic information about the likelihood of an action given a particular state.

- The Reward Function ($R : (S, A) \rightarrow \Re$): This is the actual reward received by the agent for performing an action in a given state.
- The Value Function (V): This is the value function, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. A major challenge in reinforcement learning is effective estimation of this function.

The below flowchart is meant to demonstrate all the above components in a graphical illustration, the number represent the sequence of the steps.



2.1.2 Formal Representation

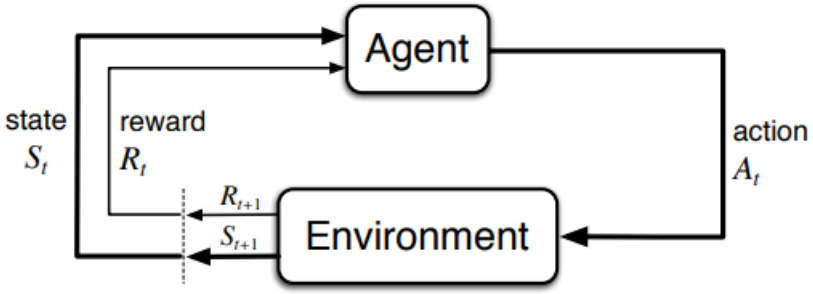
Finite Markov Decision Processes (F-MDP) are the most common mathematical tools used to formally represent reinforcement learning. In MDP the agent estimates the value represented by $V_*(s,a)$ for each action a in each state s, or it estimates the value $V_+(s)$ of each state given optimal action selections. This estimation is carried out by the agent in order to select an action. Formally an MDP can be represented using the below equation.

$$\mathcal{M} = \langle S, A, T, R, \gamma \rangle \tag{1}$$

In the above equation, S is the state set, A is the action set, R is the reward function, T is the transition function such that $T(s_*|s,a) \in [0,1]$ provides the probabilty of transitioning to state s_* from state s via action a. γ is the Discount factor that represents the weight of past rewards accumulated in a trajectory [1]. The expected value of a policy π represented as V^π for starting state s_0 can be found using the below formula also illustrated in [1].

$$V^\pi(s_0) = E_{s,\pi(s)}[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))|s_0] \tag{2}$$

Representing time as a sequence of discrete steps of interaction between the agent and the environment allows us to formulate the problem as follows. At each time step the agent receives the representation of the current state $S_t \in S$ from the environment. After consulting the policy and optionally estimating the Value (V) for each of the potential actions using a model of the environment (M), the agent selects the estimated optimal Action $A_t \in \pi_*(S_t)$ where π_* represents the policy function that returns the set of possible actions given a particular state. Once it has performed the action A_t, the system moves to time step $t+1$ as the state changes to S_{t+1} and the agent receives a numerical reward $R_{t+1} \in R(S_t, A_t) \subset \Re$ from the environment for it's prior action. This whole process is nicely illustrated by (Richard S. Sutton and Andrew G. Barto. [3]) in the following diagram.



The above process leads to the below sequence as outlined in [3].

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ... \tag{3}$$

If we filter out the rewards, while retaining the states and actions from the sequence above we get the below.

$$S_0, A_0, S_1, A_1, S_2, A_2 ... \tag{4}$$

This is the sequence that we are most interested in for purposes of Inverse Reinforcement Learning. We will restrict our discussion of Reinforcement Learning to the above, for further exploration we refer the reader to [3].

2.2 Inverse Reinforcement Learning

2.2.1 Overview

Inverse reinforcement learning (IRL) is the task of determining the reward function of an expert agent given it's policy or observations of it's behavior. As we saw in the previous section the interaction between the agent and the environment provides us with a sequence of states and actions illustrated in equation (4). This sequence is representative of the behavior of our agent. In the case of IRL we call this sequence the demonstrated trajectory and use it to recover the reward function R^* that explain the demonstrated trajectory assuming the expert performed optimally. For an optimal IRL model, the reward function R and R^* would be equivalent given that our agent functioned as per the reward function R.

2.2.2 Formal Representation

As we wish to find the reward function R^* which best explains the expert's behavior while assuming that the expert behaves in a optimal fashion, it follows that the expected value for that reward function under the expert's policy would be higher than the expected value for that reward function under all other policies. Formally we can illustrate this idea using the below equation as mentioned in [4].

$$E_{s,\pi^*(s)}[\sum_{t=0}^{\infty}\gamma^t R^*(s_t,\pi^*(s_t))|s_i] \geq E_{s,\pi(s)}[\sum_{t=0}^{\infty}\gamma^t R^*(s_t,\pi(s_t))|s_i]\ \forall\pi \tag{5}$$

2.2.3 Challenges

- The expert policy π^* is not known to us, hence calculating the left hand side of the above equation may not be possible. In order to resolve this issue we can use a feature based reward function and utilize observed behavior samples along with a neural net to estimate feature expectations. The feature based reward function can be obtained by substituting $R(s,\pi(s)) = \mathcal{W}^T\phi(s)$ in equation (5), where $\mathcal{W} \in \Re^n$ and $\phi : S \to \Re^n$

  $$E_{s,\pi^*(s)}[\sum_{t=0}^{\infty}\gamma^t R^*(s_t,\pi^*(s_t))|s_i] = E_{s,\pi(s)}[\sum_{t=0}^{\infty}\gamma^t \mathcal{W}^T\phi(s_t)|s_i] = \mathcal{W}^T\mu(\pi) \tag{6}$$

  Where $\mu(\pi)$ is equal to $E_{s,\pi(s)}[\sum_{t=0}^{\infty}\gamma^t\phi(s_t)|s_i]$ for $\forall i$. For a further discussion of feature expectations please refer to [4].

- Reward function ambiguity, this challenge refers to the fact that many different reward functions may satisfy the above formulation. As there is only one true reward function we need to estimate the closest approximation to this reward function. This can be achieved using the standard max margin approach first popularized in SVMs. Essentially we increase the "difference" between our pick for the expert policy and the other policies by the maximization of a margin. This can be represented as below

  $$\mathcal{W}^T\mu(\pi^*) \geq \mathcal{W}^T\mu(\pi) + D(\pi^*,\pi)\ \forall\pi \tag{7}$$

  In the above equation $D(\pi^*,\pi)$ represents the margin as the magnitude of difference between π^* and π. This is because margin should be higher for policies that are very different from π^* given that we are searching for π^*.

- High computational complexity. Given that we have to enumerate all policies IRL tends to be computationally very expensive and does not scale well with higher dimensions of features. In order to solve this challenge we can employ Auto-encoders for dimensionality reduction. Discussion of this solution is out of scope of this article, but we refer the reader to Henry Maathuis's Thesis [5] for a deep exploration of utilization of Auto-encoders among other tools for purposes of improving scalability of IRLs.
- Lack of standardized testbeds for Inverse Reinforcement Learning Models. This is quite a practical issue as the lack thereof of such testbeds slows the pace of development of a field. In the next chapter we will propose a solution to this challenge.

3 Proposed Solution

3.1 Overview

In this article we propose a methodology to test inverse reinforcement learning agents using a standardized approach, similar to OpenAI's Gym. While this methodology can be implemented using any popular technology, we will outline the approach using the Godot game engine in this essay. The approach requires the creation of both environments and reinforcement learning agents. It borrows from role playing games specially in regards to goal oriented motivations, by creating an environment with both primary and secondary goals for a reinforcement learning agent we can train the agent on that particular environment with a adjustable setting of rewards for each goal. This can help us create an agent that follows a very particular reward function and we can then repeat this process a few times with different configuration of rewards to obtain a number of agents with different reward functions. Then we can allow users to inject their own inverse reinforcement learning agents in simulations of the environment with respect to each of the different pre-trained expert agents. As we already know the reward function of our trained experts, we simply calculate the difference between the reward function of the pre-trained expert and the estimated reward function of the user's agent.

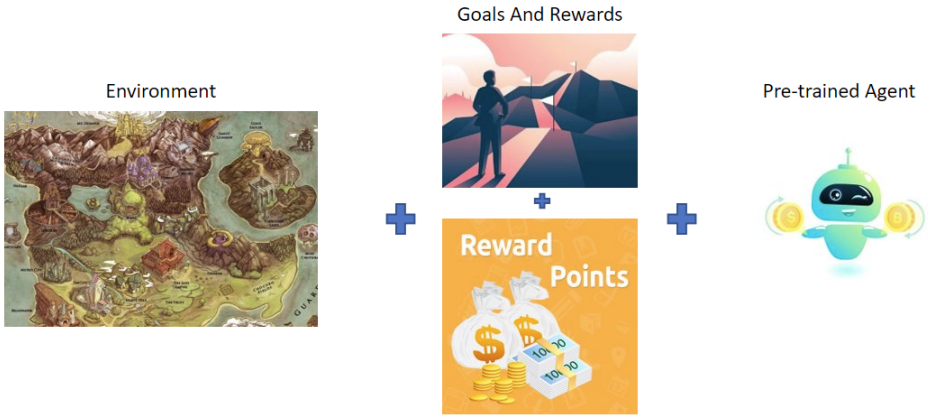
3.2 Software And Hardware

This approach will require a few different resources. As the environments will be created using Godot, we will need to have it installed in order to create the core of the platform. We will also need to create a website where users can submit their inverse reinforcement learning agents. For the creation of the website we can use major frontend frameworks, along with a backend orchestrator for the environment simulators. The users will also need to install Godot and create their inverse reinforcement learning agents using pre-defined scenes from our plugin. In terms of hardware we would need to host the services in cloud in order to be able to scale horizontally, and the users should be able to create their agents using most laptops or home computers.

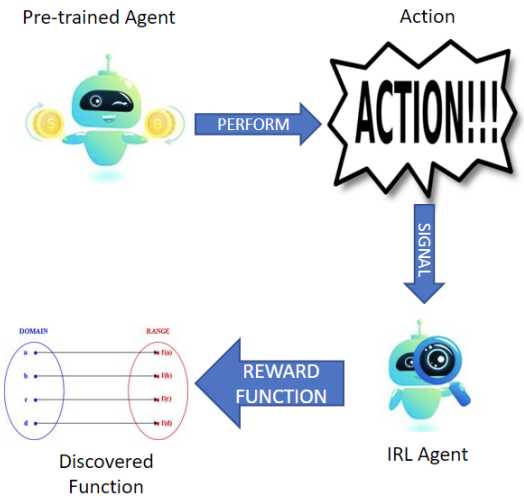
3.3 Structure

The core structure will consist of the environment, the goals/actions with adjustable rewards, and the pre-trained agent all run using the Godot engine in a simulation. The pre-trained agent will emit a signal for every action that it performs. This signal will have details about the action performed, any rewards that are associated to that action and additional contextual details about the environment (the objects acted upon for example). The user's agent process will be able to listen to these signals and will have to build a reward function. In essence these signals

will be the observations for the user's agent. The figure below summarizes the structure for the suggested simulation.



As the pre-trained agent performs actions in the environment it sends signals to the user's agent. This agent will then use the information from these signals to reverse engineer the reward function of the observed agent. The below figure illustrates this part of the simulation.



Each of the state transitions that occur during the simulation are recorded, in other words the states and actions performed. In order to be able to compare and rank the best IRL agents from users we need a methodology to score the resulting reward function provided by the user's agent. If the states for the simulations are represented by the set S and the action by the set A, then given two reward functions R_p (pre-trained agent's reward function) and R_u (user agent's reward function), we can calculate the total error between the two by the below formula.

$$\forall_{s \in S} \forall_{a \in A} \sum_{(s,a)} |R_p(s,a) - R_u(s,a)|^2 \tag{8}$$

We perform the simulations with the same agent a large number of times and calculate the error for each. We find the average of the error rate for the user's agent and that gives us a way to compare multiple user agent's. The IRL agent with the lowest overall error rate for the given simulations will be considered better than the rest. We can also test transfer learning using this approach by comparing the performance of a pre-trained agent and the user's agent on a different environment using the same formula above. As this is a highly repeatable approach we can perform a large number of tests easily to determine the best agents in the public domain, at the same time the generic nature of the environments enable us to test performance over a set of very diverse learning tasks.

4 Conclusion

In this article we have discussed the problem of a lack of benchmarking platforms for inverse reinforcement learning agents. We have described and illustrated the background of both topics of reinforcement learning and inverse reinforcement learning. We have outlined a few challenges facing the IRL community and discussed the reasons for selecting the specific problem for this article.

In the final section we have outlined our proposal for an approach that can effectively solve this problem. We have included information about the technologies that would be required to build this solution and the structure that the proposed solution will have once implemented. While implementing the proposed solution is a large undertaking it provides a direct benchmarking methodology between two different IRL agents, something that in our investigation we found to be lacking at this point in time. We provided the mathematical representation of the comparison to ensure it's well defined for the purposes of understanding the solution for an implementer.

The primary undertaking in the future for this proposal would be to implement the proposed solution and demonstrate comparison results for multiple IRL agent's to prove that the agents with low error rate using this methodology also have better performance for empirical testing methods currently popular in the community.

References

1. A Survey of Inverse Reinforcement Learning: Challenges, Methods and Progress. Saurabh Arora, Prashant Doshi. arXiv:1806.06877v3 [cs.LG]

2. Learning Robust Rewards with Adversarial Inverse Reinforcement Learning. Justin Fu, Katie Luo, Sergey Levine. arXiv:1710.11248 [cs.LG]

3. Reinforcement Learning: An Introduction, Second Edition. Richard S. Sutton, Andrew G. Barto.

4. Inverse Reinforcement Learning. Pieter Abbeel - UC Berkeley EECS - CS287. `http://people.eecs.berkeley.edu/~pabbeel/cs287-fa12/slides/inverseRL.pdf`

5. On the scalability of Deep Inverse Reinforcement Learning In High Dimensional State Spaces. Henry Maathuis