Statistical Techniques in Robotics (16-831, S22) Lecture #22 (Monday, April 11)

Imitation Learning & LP-IRL

Lecturer: Kris Kitani *Scribes: Jonathan Schwartz, Husam Wadi*

1 Review

In the previous lectures, we went over Reinforcement Learning (RL) and had explored several types of RL:

1. Policy-based
2. Hybrid
3. Value-based

We then explored these topics in depth and how each approach has subsets, such as Model-based and Model-free, and within these subsets there are various approaches such as Monte Carlo, Actor-Critic, Policy Gradient, N-step TD learning, and the list goes on. In this lecture, we switch over to Imitation Learning (IL) and explore from a top-down high-level perspective what IL addresses. We will also delve into what Inverse Reinforcement Learning (IRL) is and how it is used in the context of IL. The advantages IL has over RL is that instead of creating and penalizing specific rewards, such as joint toques and motor speeds, which may be difficult to design around, IL instead rewards based on imitating a whole subset of actions. This is advantageous because IL simplifies the setup and training procedure for complex tasks like robot grasping, where it is not clear what specific value should be the reward function. It also makes systems with large state spaces feasible, as with traditional RL it is costly to train specific rewards across massive tensors.

1.1 Reinforcement Learning: Markov Decision Process

RL is an algorithmic approach described as an agent learning (or closely approximating) an optimal policy that maximizes a reward function. RL problems are often modeled as grid based Markov Decision Processes (MDP) that allows us to use powerful equations such as the Bellman Equation: An equation which defines an inductive/recursive relationship between value functions and constraint optimizations. To setup a MDP we can define the problem with (S, A, P, r, H, ρ), where S is equal to the state, A is equal to the action, P is the transition dynamics, $P(s'|s, a)$ is equal to the probability of achieving s' from state s through action a, r is equal to the reward, H is equal to the planning horizon, and ρ is equal to the probability distribution of the initial state S.

$\pi(a|s)$ is the probability that action a takes place at state s. Starting from the initial state, a trajectory routes the path from that state to the planning horizion defined by this relationship: $\tau = \{s_1, a_1, s_2, ..., s_H, a_H, s_{H+1}\}$ given $s_1 \sim \rho$, $a_t \sim \pi(\cdot|s_t)$ and $s_{t+1} \sim \pi(\cdot|s_t, a_t)$. Through this MDP setup, an RL algorithim finds the optimal policy that maximizes the reward: $R(\tau) = \sum_t r(s_t, a_t)$.

1.2 Reinforcement Learning: Value vs Policy Function

To delve into the difference between Value vs Policy Functions, let us review the three main archetypes of RL in detail:

- **Policy-based**: Back deriving reward optimization directly through reward interactions with the environment, without an explicit value function.
- **Hybrid**: A mix of Policy-based and Value-based approaches.
- **Value-based**: Through interactions with the environment, we define a value function that scores the outcome of state s and action a to s' and the reward r that is achieved through this interaction. This value function is used to improve the policy which then increases the reward output.

The issue with these approaches is that we have no prior to use, so we must use stochastic and predictive methods to explore the state space. This may take quite a bit of time, and that is where IL becomes really effective: with an expert to help guide the process, we can quickly imitate the desired outcome and accelerate learning.

1.3 Reinforcement Learning: Model-based vs Model-free

There are also two sub categories of RL algorithms:

- **Model-based**: Having a model of the transition dynamics $P(s'|s, a)$, and the reward function to describe the state $r(s, a, s')$.
- **Model-free**: Estimating the state directly from the reward interactions with the environment.

Compared to IL, there are many parallels between RL and IL in terms of having a model vs doing without one. While having a model is useful and cuts back on the sheer amount of data required to optimize a policy, it usually requires a PhD student several days to even come up with a model for the state. Even worse, there may be times when coming up with a model is nearly impossible, and as such a model-free approach is required. Model-free approaches are much easier to set up, but require a significant premium in quantity and quality of data to produce reasonable results.

IL also has model-based and model-free approaches, which faces similar challenges to what was mentioned above. We will also touch upon IRL where we solve for the reward function which is not exposed to the IL model. For this, we require the transition dynamics of the IL algorithm to apply this IRL approach.

2 Summary

2.1 Imitation Learning: Overview

For context we will show how we have traveled through different machine learning approaches in this class to where we are now with IL in Figure 1:

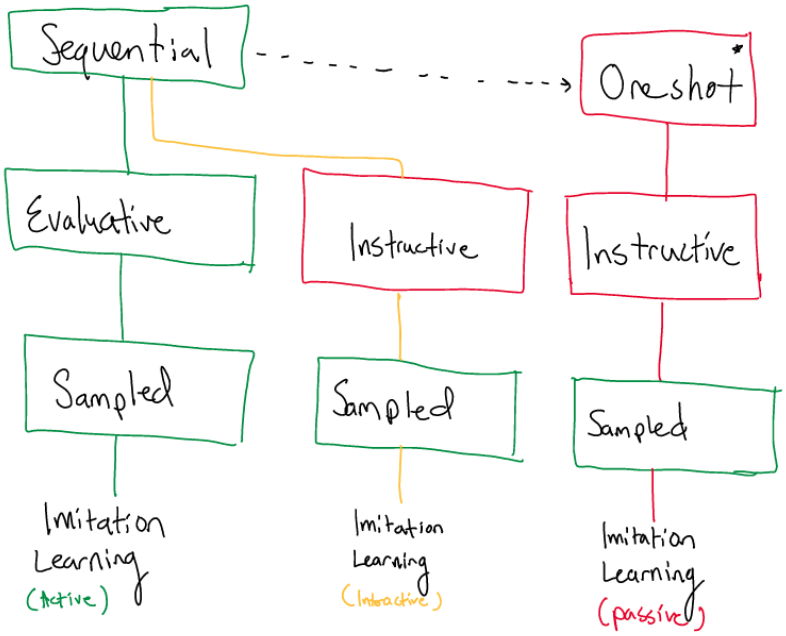

Figure 1: Imitation Learning vs. Learning Trajectory[2]

Imitation learning can be described using the same MDP we used for RL with some minor modifications. Similar to RL: S is equal to the state, A is equal to the action, P is the transition dynamics, $P(s'|s,a)$ is equal to the probability of achieving s' from state s through action a and r is equal to the reward. We will now delve into the details of each part of the IL structure:

- **Expert Demonstrations**

$$D^* = \{\zeta_n\}_{n=1}^{N} \sim \varepsilon|\pi^* \tag{1}$$

Expert demonstrations are the new addition to our previous RL algorithms and are one of the centerpieces of IL. Expert demonstrations are sequences of states and actions that are sampled from the optimal policy. The learner has to find a policy that fits the trajectories sampled at all or most states without over-fitting to a single sample. This is difficult and is why there are many techniques to solve this problem. The trajectories are defined as follows: $\zeta = \{s_0, s_1, ...s_T\}$ where $s_0 \rightarrow s_1$ is a transition between one state to another. $\zeta = \{s_0, a_0, s_1, a_1, ..., a_{T-1}, s_T\}$ is equal to the state-action steps in the sequence and explicitly states the action required to transition from one stat to the next.

Note that the expert demonstrations are often assumed to be optimal, but that is not always

the case in practice. They may just provide a decent starting policy for the IL algorithm to interact with its environment, as was the case in [3].

- **Oracle**

$$\pi^*(a|s) \tag{2}$$

The oracle knows π^*, the optimal policy for the system. It uses this policy to help the IL algorithm improve. The IL algorithm learns from the oracle by interacting with its environment, and getting feedback on if its actions were optimal or not. Through these interactions, the IL algorithm can improve its policy toward the optimal policy.

- **Dynamics**

$$\mathcal{T} = P(s'|s, a) \tag{3}$$

The dynamics function for IL is the same as the one used for RL. It is the probability of getting to state s' from state s through action a. One thing to note in IL is that the dynamics function is not always given. An example of this in passive IL is the behaviour cloning algorithm (also known as supervised learning), which does not assume a knowable dynamics function. We will delve further into this later on in the scribe notes.

- **Reward**

$$\mathcal{R} = R(s, a, s') \tag{4}$$

Reward functions are also defined similarly to how we defined them in RL. Based on state s, doing action a and iterating to the next state s' a reward r is generated. RL and IL both choose actions through policies and the policies can be chosen scholastically. However, as we have mentioned above, IL uses some form of expert/oracle feedback directly or in conjunction with RL to estimate the policy, and in the case of IRL, to also re-estimate the reward function.

2.2 Imitation Learning: Types

In Table 1 below we outline the three major types of IL and how they differ with regards to state parameters and information exposure:

Parameter	**Passive**	**Active**	**Interactive**
Demonstration $\mathcal{D}^*$	yes	yes	optional
Environment ε	no	yes	yes
Oracle π^*	no	no	yes
Dynamics $\mathcal{T}$	no	optional	optional
Reward $\mathcal{R}$	no	optional	optional

Table 1: Imitation Learning Types

We will now delve a bit deeper into those definitions and explore what each type of IL described above entails:

- **Passive** Passive reinforcement learning, also known as supervised learning or behavior cloning, uses only trajectory demonstrations provided by the expert to learn a policy. Policy optimization's sole purpose in these algorithms is to match the behavior of the expert (supervised

data). This can be formulated as:

$$\arg\min_{\theta} \sum_{\tau=\{s_t,a_t\}_{t=1}^{T}} l(\pi_\theta(a|s_t), a_t) \tag{5}$$

One thing to be wary of when using passive reinforcement learning algorithms is **covariate shift**, which is when error accumulates due to several successive states that were not seen in training. Since the agent has not seen these states before, it will likely not behave optimally, resulting in error at each state. In sequential settings like driving, this error can compound across several adjacent states and result in the agent failing.

- **Active** For the active version of IL, we have access to the environment and the expert demonstrations. This means that the learner can train an initial policy using the expert demonstrations, and then use RL techniques to improve the policy by interacting with the environment. This allows the learner to identify differences between the demonstrations it received and the environment it acts in, and update its policy accordingly. It also allows the learner to improve upon its policy if the expert demonstrations were not optimal.

  In active IL, the learner also sometimes has access to the state transition dynamics and reward function of the environment.

- **Interactive** In interactive IL, the learner can access feedback from an oracle that knows π^*, the optimal policy. This allows it to directly improve its policy after each non-optimal action that it takes. With the oracle, initial expert demonstrations are not necessary since oracle feedback can effectively provide those in hindsight. Note that this is not a good approach in environments where failures can be expensive (e.g. driving). The state transition dynamics and reward function are optional for these algorithms, but often are not needed due to the presence of the oracle.

2.3 Inverse Reinforcement Learning: Overview

A form of active IL is "Inverse Reinforcement Learning" (IRL). The idea behind IRL is that the reward function, which is usually elusive, can be learned in reverse through IRL then fed into RL to derive the optimal policy π. IRL learns the optimal reward function from the expert, the learner, and the behaviors they both demonstrate. This type of reward function does well in generalizing the problem to different (but similar) environments and is a very powerful solution.

The outline of the architecture of Inverse Reinforcement Learning is shown in Figure 2:

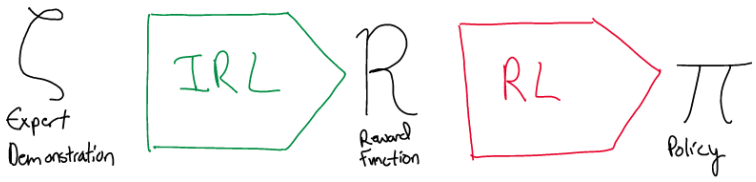

Figure 2: Inverse Reinforcement Learning

2.4 Inverse Reinforcement Learning: Types

- **Finite State Space** In finite state space IRL we assume that the set of all possible states is finite, that the transition model is known, and that the complete policy is knowable for our learner. This is the simplest case and a great base to start with.
- **Large State Space** In large state space IRL we assume that the set of all possible states is very large or infinite, that the transition model is known, and that the complete policy is knowable for our learner. This will be a bit more challenging and will require some derivation to approximate.
- **Sampled Trajectories** This will be covered in the following lecture, and will relax more assumptions that will allow the IRL algorithm to become even more generalizable.

2.5 Inverse Reinforcement Learning: Finite Spaces

In lecture, we discussed the parking lot trajectory planning problem as an example of a finite space IRL problem. In this case, we discretized the parking lot into a grid area, labelled each grid with some features, and then trained a reward function using this information and expert demonstrations. This reward function allowed us to classify the rewards of different parts of a scene, which we could use to plan trajectories in different scenes. Note that these scenes still need to be similar to the original unless we are given more demonstrations related to the new scenes. For example, the parking lot trajectory demonstrations will not train a good reward function for generating trajectories on hikes in wooded areas. This framework is depicted in Figure 3.

When solving finite space inverse reinforcement learning problems, we assume that:

- state space is finite
- transition model is known
- optimal policy is known

These problems can be represented with the following optimization problem:

$$\max_R \sum_s Q^\pi(s, a^*) - \max_{a \neq a^*} Q^\pi(s, a)$$

This means we want to find a Q function that maximizes the improvement of the optimal policy over the second best action at every state. In other words, we are trying to find the reward function that makes the given policy optimal by the largest margin. One thing to be wary of is that, if left unconstrained, this optimization will result in massive positive or negative rewards at different states. To fix this, you can add a penalty term on the sum of the absolute values of the rewards (also known as the L1 norm):

$$\max_R \sum_s Q^\pi(s, a^*) - \max_{a \neq a^*} Q^\pi(s, a) - \lambda ||R||_1$$

This problem can be solved with a linear program. To do so, you first need to use the matrix form of the Q table:

$$Q^{\pi}(s,a) = R(s) + \gamma \sum_{s'} p(s'|s,a)V^{\pi}(s')\forall s, a$$
$$\mathbf{Q}^{\pi}(a) = (\mathbf{R} + \gamma \mathbf{P}_a \mathbf{V}^{\pi})$$

We can do the same thing with the Value Function $V^{\pi}(s)$:

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} p(s'|s, a = \pi(s))V^{\pi}(s')\forall s$$
$$\mathbf{V}^{\pi} = (\mathbf{I} - \gamma \mathbf{P}_a)^{-1}\mathbf{R}$$

This allows us to write the objective function as:

$$\mathbf{Q}^{\pi}(a^*) - \mathbf{Q}^{\pi}(a)$$
$$= (\mathbf{R} + \gamma \mathbf{P}_{a^*}\mathbf{V}^{\pi}) - (\mathbf{R} + \gamma \mathbf{P}_a \mathbf{V}^{\pi})$$
$$= \gamma(\mathbf{P}_{a^*} - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_{a^*})^{-1}\mathbf{R}$$

Before we can write the objective function in linear form, there is one more constraint we need to add. This is that the future payoff for the optimal action is greater than the future payoff for any other action. We can write this in matrix form like so:

$$\mathbf{P}_{a^*}\mathbf{V}_{\pi} \geq \mathbf{P}_a \mathbf{V}_{\pi}$$

Plugging in the expression we derived for $\mathbf{V}_{\pi}(s)$:

$$\mathbf{P}_{a^*}(\mathbf{I} - \gamma \mathbf{P}_{a^*})^{-1}\mathbf{R} \geq \mathbf{P}_a(\mathbf{I} - \gamma \mathbf{P}_{a^*})^{-1}\mathbf{R}$$

Now we can write out the full description of the linear program:

$$\hat{R} = \arg\max_R \Big(\sum_s \min_a \left\{ \gamma(\mathbf{P}_{a^*} - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_{a^*})^{-1}\mathbf{R} \right\} \Big) - \lambda ||R||_1$$
$$\text{s.t.} \quad (\mathbf{P}_{a^*} - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_{a^*})^{-1}\mathbf{R} \geq 0$$
$$|R(s)| \leq R_{max} \forall s \in S$$

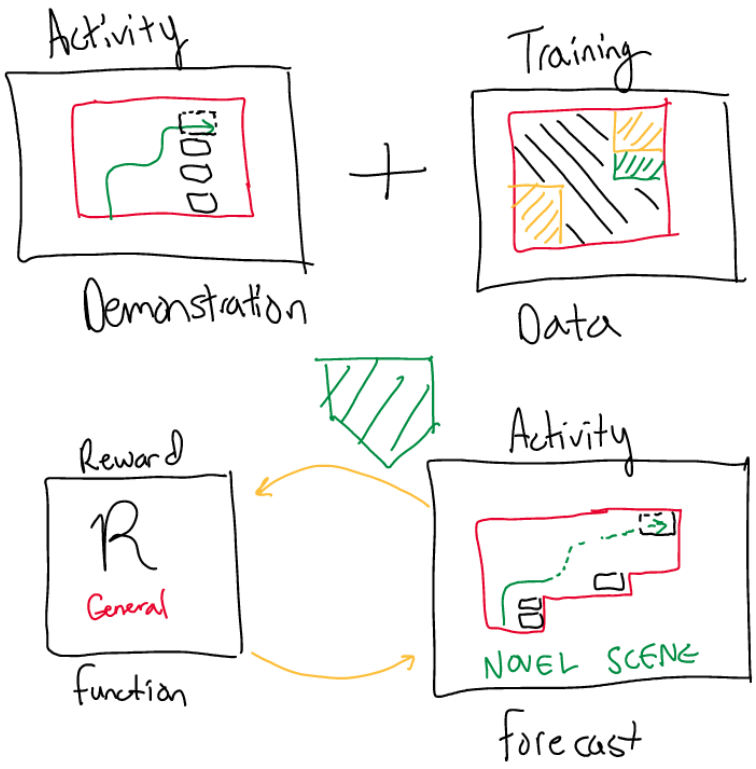

Figure 3: Finite Space IRL Parking Lot Example [1]

References

[1] K. Kitani. Activity forecasting. *Proceedings of the 12th European conference on Computer Vision*, IV, 2012.

[2] M. Littman. Reinforcement learning improves behavior from evaluative feedback. *Nature*, 521:445–451, 2015.

[3] J. Peters. Reinforcement learning of motor skills with policy gradients. *Robotics and Neuroscience*, 21(4):682–697, 2008.