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

Imitation Learning

Lecturer: Kris Kitani *Scribes: Mansi Agarwal, Ashwin Misra*

1 Review

In review, the last few lectures were on Reinforcement Learning. Reinforcement Learning is a learning paradigm where an agent tries to learn an optimal, or nearly-optimal, policy that maximizes short-term and long term rewards. Theoretically, we often assume the RL environment as a Markov Decision Process(MDP).

The three main types of Reinforcement Learning Algorithms, namely-

- Policy-based Methods- In these methods, the stochastic policy is directly learned from the state and actions. The actions are taken by sampling from the policy.We explicitly build a representation of a policy (mapping $\pi : s \rightarrow a$) which maps the state to action, and keep this in cache while the learning is happening.
- Hybrid-based methods- In these methods, the policy proposes a set of possible actions that can be taken for a given state and the feasibility is calculated by the estimated value function, which evaluates the actions taken based on the given policy.
- Value-based methods- These methods don't learn the policy but learn the states or state-action values. The action is chosen by the best reward in that state. Due to this, it is necessary that all the actions are explored for the state.Unlike the policy based methods, the policy is not stored in the cache, only the value function. This value function is used to derive the policy.

Depending on the underlying assumptions of the problem that we have, RL algorithms can also be classified into-

- Model-based RL Algorithms: Such algorithms have full access to the environment including state transition dynamics and reward function. Such algorithms represent the environment as a Markov Decision Process. Value based or policy based iteration algorithms come under model-based algorithms. In policy iteration, the rewards (used to estimate the policy or value functions) are not the result of the interaction with the environment but directly given by the reward function (accessible to the algorithms). Furthermore, since we have access to the dynamics model, the agent need not interact with the environment to retrieve the next state given current state-action pair.
- Model-free RL Algorithms: Such algorithms do not have access to the dynamics model (state transition dynamics) and can only draw samples from the environment. Model-free methods, such as Monte Carlo methods do not use have access to dynamics model, even though the assumption that the environment can be represented as an MDP is implicitly made. There

can be two kinds of model-free algorithms: off-policy vs on-policy. On policy algorithms are those algorithms where the policy needs to be updated after every parameter update and the updated policy needs to interact with the environment for updated sample trajectories., i.e., every time the policy parameters are updates, the agent has to collect more data and cannot use the past experience. On the other hand, off policy algorithms are those where no new data needs to be collected after a policy parameter update. In such algorithms, there are often two policies, one that interacts with the environment to sample trajectories and the other that is optimised using the experience of the other policy. There is an assumption of full coverage between the two policies.

1.1 Last Lecture

In the last lecture, we went through Policy gradient methods- namely Monte Carlo Policy Gradient and REINFORCE. Policy Gradient methods don't require models and learn from interactions. They take the estimated return in the gradient to update the policy step. We see how large rewards result in large gradient variance, and how it can be solved with enforcing causality or a baseline offset. And when the value function estimates are used as the baseline, for policy gradient methods, the methods are called actor-critic methods. These methods use a function approximator return estimate in their policy update step. As discussed before in the hybrid policy based methods, the critic is the evaluation of the actions to take which is done by the value function, and the actor is the policy which gives us possible actions to take from a state.

2 Summary

2.1 Imitation Learning

2.1.1 Why Imitation Learning?

Reinforcement learning is a widely used machine learning paradigm where an agent learns from its own experience. The agent perceives the state, and using a policy it takes the next action. Based on this action, it receives a reward and the environment transitions to the next state. Reinforcement learning algorithms aim at learning the optimal policy which maximises long term as well as short term rewards. Typically, RL methods work well when the state and action space is small and finite. However in most cases, it is tough to deploy RL algorithms. For example, in situations where reward is only obtained after the last step in the horizon (say a game of chess where the reward is 0 or 1 depending on whether the agent lost or won). There could also be scenarios where constructing a reward function is complex and not intuitive. Furthermore, even if we have access to a reward function that returns rewards at every step, RL algorithms can take a long time to converge since they start from trial and error and try to figure out what's best in a given scenario through experience (interacting with the environment).

A feasible solution to all these problems is Imitation Learning (IL). Imitation Learning is also known as Learning from Demonstration, Apprenticeship Learning, Behavior Cloning, and Inverse Reinforcement Learning. In Imitation Learning, the agent learns a policy from demonstrated expert behavior. So, the goal of IL is same of RL to learn an optimal policy but instead of using its own experience, it learns from an expert's demonstrations.

As we'll see later, Imitation Learning employs expert demonstrations and henceforth speeds up the learning process (as compared to RL algorithms when the state and action space is not finite). Also, since IL uses demonstrations, there is no need to construct reward functions which saves a lot of time and complexity.

2.1.2 Problem Formulation

Let us first formalise the Imitation Learning(IL) problem.

$$\mathcal{A}_{\text{IL}} : \langle \mathcal{D}^*, \pi^*, \mathcal{T}, \mathcal{R} \rangle \rightarrow \pi$$

Here, $\mathcal{A}_{\text{IL}}$ denotes the Imitation Learning algorithm. Let us now discuss what each element describes.

1. Expert demonstrations ($\mathcal{D}^*$):

   $$\mathcal{D}^* = \{\boldsymbol{\zeta}_n\}_{n=1}^N \sim \mathcal{E} \mid \pi^*$$

   Expert demonstrations are defined as a sequence of variables such as a sequence of states and actions. These sequences are sampled from the optimal policy π^*. The representation form of demonstrations is flexible though, as it could be represented either as

   $$\text{state sequence} \quad \boldsymbol{\zeta} = \left\{ s^{(0)}, s^{(1)}, \ldots, s^{(T)} \right\}$$

   or

   $$\text{a sequence of state-action pairs} \boldsymbol{\zeta} = \left\{ s^{(0)}, a^{(0)}, s^{(1)}, \ldots, a^{(T-1)}, s^{(T)} \right\}$$

   Note that in off-policy RL algorithms, the policy used to sample trajectories might not be optimal, whereas for IL, the optimal policy is employed. Also, these demonstrations do not include rewards. We assume that the action taken per state optimises some underlying function. In other words, the action is the best action possible for the given state. Another thing to pay attention is that these demonstrations might be from one single episode or N episodes, and could also be shuffled. So, it is not necessary that an action a_1 on state s_1 would lead to state s_2 from the sequence.

2. Oracle (Optimal Policy π^*):

   $$\pi^*(a \mid s)$$

   Optimal policy is the expert policy where the expert demonstrations are sampled from. One thing to question is that if we have access to the optimal policy, why do we need an algorithm to find the policy at all? It should be noted that we do not necessarily have access to the optimal policy, but we can get samples from it during training. In other words, we might not have access to the full parameterization of the optimal policy and thereby can only use it to get access to the best action at a given state. Such type of IL is known as Interactive IL.

3. Dynamics function $\mathcal{T}$ and Reward function $\mathcal{R}$:

$$\mathcal{T}\left(s' \mid s, a\right), R\left(s, a\right)$$

The dynamics model and the reward function together constitute the environment $\mathcal{E}$, which still needs to be a Markov Decision Process (MDP). This means that the states in the environment follow the Markov property, where the next state is only dependent on the current state and action and is independent of the previous state and actions. $\mathcal{T}(s'|s, a)$ denotes the transition model which defines the probability that an action a in the state s leads to state s'. Reward function ($R(s, a)$) returns the reward for the state-action pair.

2.1.3 A Classification on IL Algorithms

We can roughly divide IL algorithms into three categories depending on which elements (discussed above) the agent has access to.

Access to	Passive IL	Active IL	Interactive IL
Demonstrations D^*	yes	yes	optional
Environment ε	no	yes	yes
Oracle π^*	no	no	yes
Dynamics T	no	optional	optional
Reward R	no	optional	optional

Table 1: A Classification on IL Algorithms

1. Passive IL: As we can see from Table 1, Passive IL only has access to demonstrations. Since, we use the expert demonstrations and supervise our model to clone the behaviour of the expert, passive IL is also known as behavior cloning or supervised learning. So, essentially, we have a set of demonstrations (dataset), we use that to train an ML algorithm and learn the policy. This idea has been employed by researchers since 1960s. It was first discussed in the paper ”Pattern Recognising Control Systems” [5]. Another important example of behaviour cloning is ALVINN (Autonomous Land Vehicle In a Neural Network) [3], a vehicle equipped with sensors, which learned to map the sensor inputs into steering angles and drive autonomously. This project was carried out in 1989 by Dean Pomerleau, and it was also the first application of imitation learning in general. More recently, a paper called ”Learning by Cheating” [1] employed the idea of behaviour cloning to improve autonomous driving. The paper proposes an algorithm that entails two steps. In the first step, the method first trains a ”privileged agent” which has access to the full state of the environment including meta-data. Therefore, this step does not need to take care of the perception component of autonomous driving. Once the privileged agent is trained, it is treated as the optimal policy and is then used to sample expert demonstrations. In step 2, the sensorimotor agent is trained using these expert demonstrations via Behaviour Cloning. This method beat the state-of-the-art results.

   The way behavioural cloning works is quite simple. Given the expert's demonstrations, we divide these into state-action pairs and treat these pairs as independent and identically distributed (i.i.d.) examples. Using supervised learning, we learn the optimal policy. Passive IL lies with online linear classification algorithms with sampled one-shot feedback.

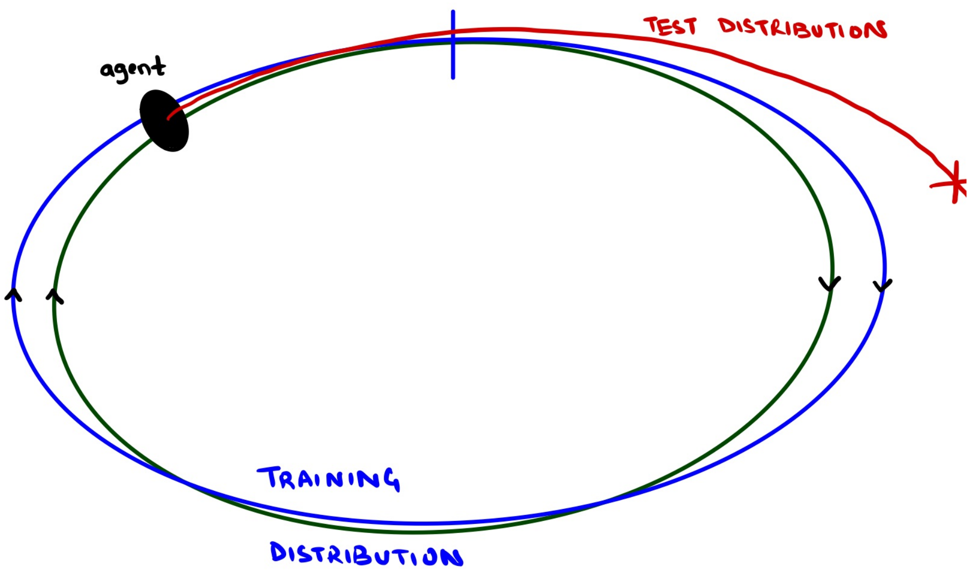

Figure 1: Covariate Shift

Passive IL is simple and intuitive to use but it has a major disadvantage. The learned policy only works well on states it has been trained on. So if one uses a policy trained using Behaviour Cloning for long horizon planning, the method might fail due to this covariate shift. For example, in Fig 1, we see that at every step the policy makes small errors which when accumulated over time result in a shift in test distribution (from the training data distribution). As the errors accumulate, the policy reaches a state it has never seen before. This results in a bogus action which further results in a state the policy has not been trained on thereby, leading to failure.

Hence, passive IL only models short-term (one-shot) behaviour and cannot model long-term behavior well. We could also say that passive IL works best when 1. The problem is simple 2. The state space is finite and demonstrations cover all possible states.

2. Active IL: Active IL has access to expert demonstrations as well as the environment, but does not have access to the optimal policy. In active IL, the method employs the expert demonstrations to initialise its policy parameters and then interacts with the environment (just like RL algorithms) to further finetune the policy. Paper "Reinforcement Learning of motor skills with policy gradients" [2] employs this idea. It first uses behaviour cloning to supervise the agent to learn how to hit a ball using a bat, and then uses RL to improve the agent's performance. This helps in speeding up the process of finding the optimal policy.

3. Interactive IL: Interactive IL has all other access to the environment, making the expert demonstrations optional to the agent as it has access to the oracle policy, serving as an alternative of expert demonstrations. In essence, the agent interacts with the environment and there is an interactive oracle feedback to help improve the policy.

2.2 Inverse Reinforcement Learning

2.2.1 Why Inverse Reinforcement Learning

Inverse reinforcement learning is the process of studying an agent's objectives, values, or rewards with the aid of using insights of its behavior. This algorithm estimates the policy and the reward function by learning from Expert Demonstrations and how they act in the environment. It helps in obtaining a reward function which can be used to generalize to other unknown environments for which reward functions cant be defined easily. This knowledge can be transferred to new but similar environments. For example, a reward function for a self-driving scenario on a busy road cannot be transferred to a robotic manipulation problem.

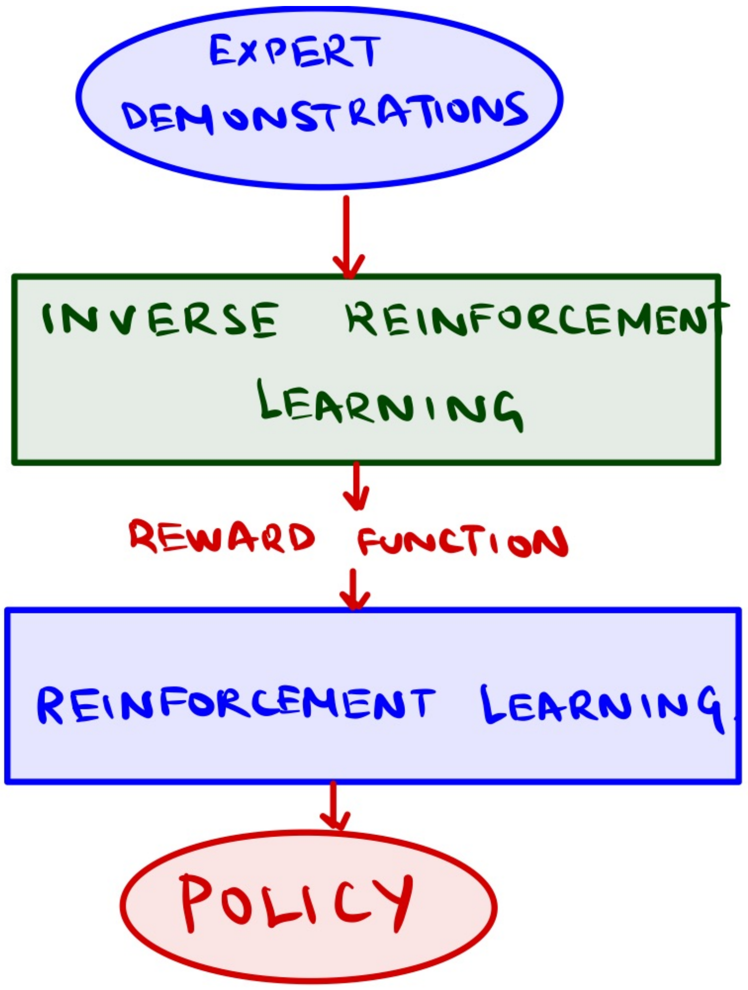

Figure 2: Inverse Reinforcement Learning

2.2.2 Trajectory Forecasting

Let us assume a scene in which a squirrel has to reach a nut in a park. The park has a lot of obstacles in it as shown in the figure. We know the start and the end positions and we need to find a path for the squirrel to reach the destination.

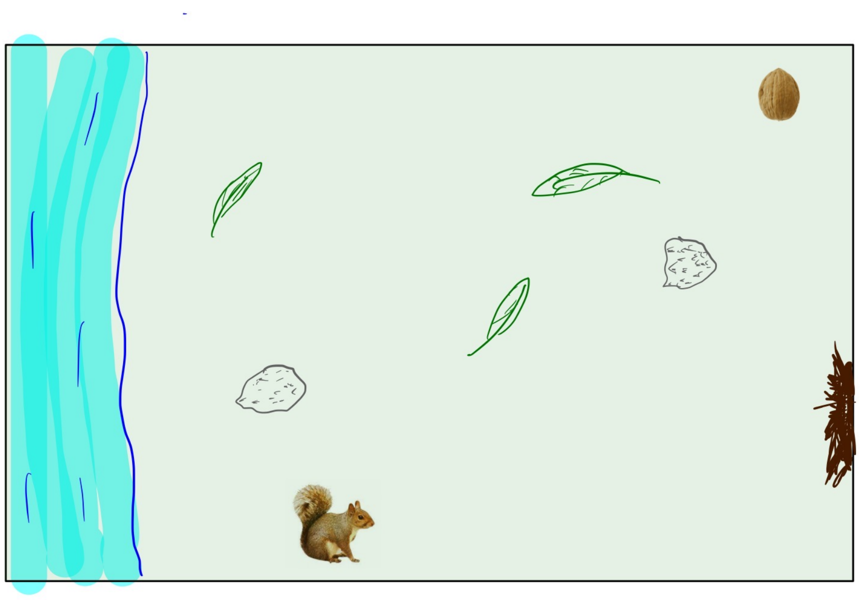

Figure 3: Novel scene

The training data that we have is the expert's demonstration and the physical features of the park such as the grass, the stones and the neighboring trees. This input gives the trajectory as the reward function for the squirrel to reach the nut. This same reward function, which is not scene specific, can be used to forecast activities in novel scenes.

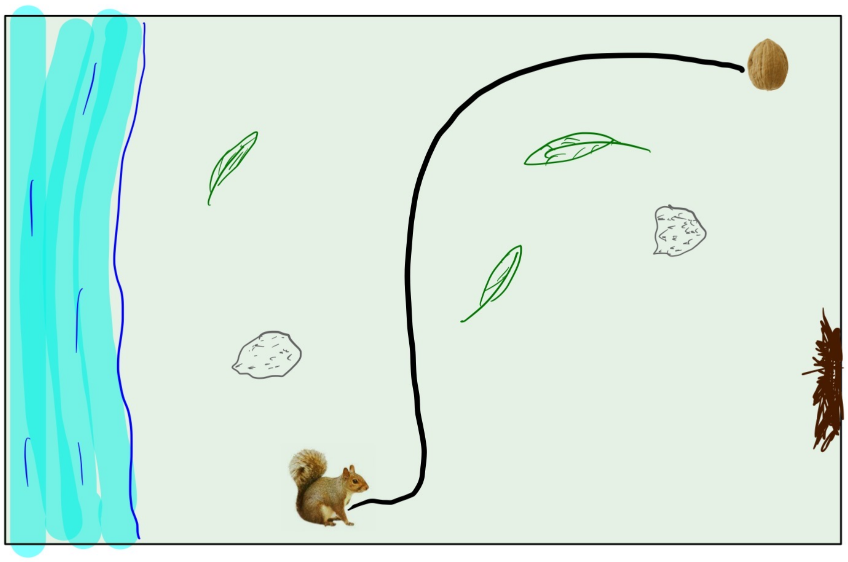

Figure 4: Demonstrated Activity

In this example, This combined with the physical features i.e. the scene segmentation data comprise of the training data.

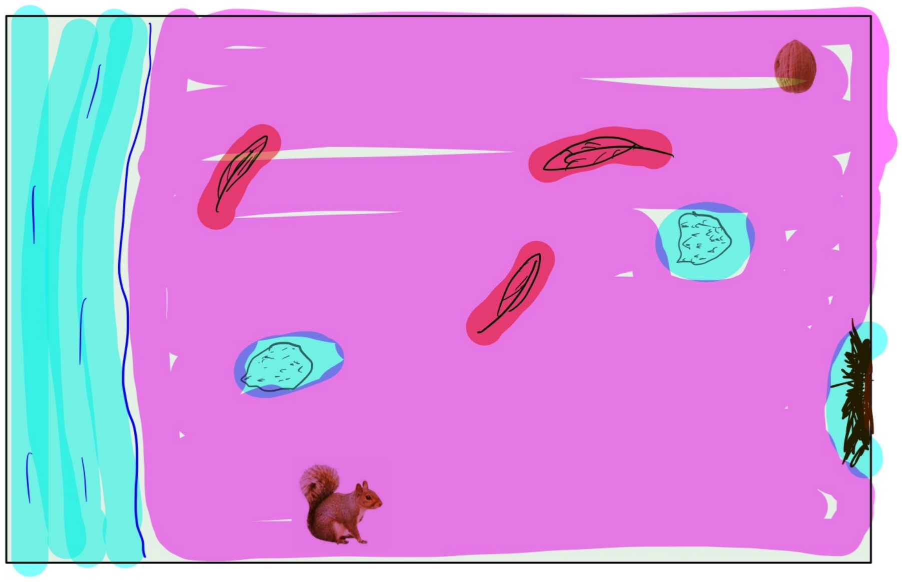

Figure 5: Physical Features

Inverse Reinforcement Learning assumes a Markov Decision Process defined over a grid world, which is fitted across the scene. The gridworld on our running example is shown in the figure-

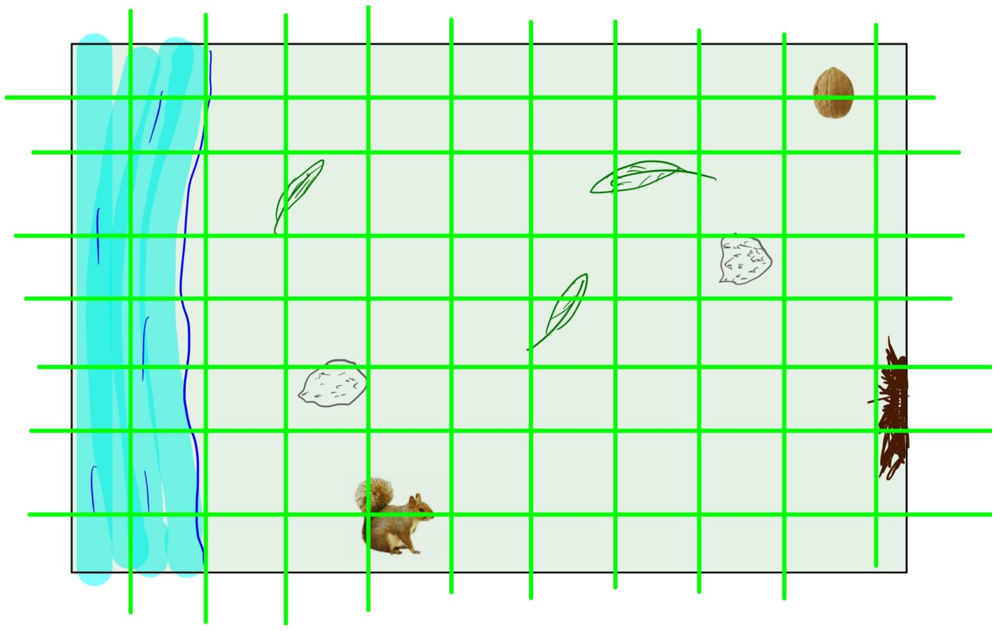

Figure 6: MDP defined over a gridworld

$$\mathcal{S} = \{x_0, a_0, R(x_0), x_1, a_1, R(x_1), ..., x_g, a_g, R(x_g)\}$$

where $x = state$, $a = action$, and $R = reward$. The state is the the co-ordinates on the grid, the actions being the velocity in pre-defined directions. The dynamics are the probability of going to state x' from x while we take the action a $P(x'|x, a)$. To further understand the MDP, the features of a state (f(x)) could be the semantic segments of the scene for each location. The value function of each state (V(x)) could be the expected feature reward over a trajectory originating from that state, also known as cost-to-go.
As shown in the figure, the reward function highlights the segments that are high reward and the segments which are low reward. This is the output of the INverse Reinforcement Learning Algorithm.

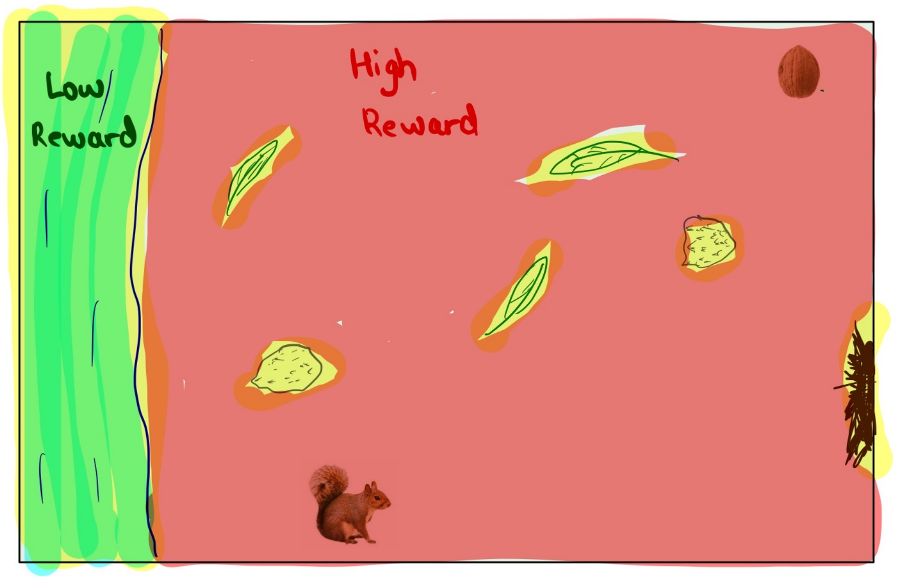

Figure 7: Output of IRL

References

[1] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020.

[2] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[3] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

[4] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on arti-*

ficial intelligence and statistics, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[5] B. Widrow. Pattern-recognizing control systems. *Compurter and Information Sciences*, 1964.

3 Appendix

We briefly discussed the problem of covariate shift in Passive IL. Here, we will discuss ways of curbing this problem.

One way of reducing the covaiate shift is by making the test distribution equivalent to the training distribution. We know that the test distribution comes from the learned policy and the training distribution comes from the expert (optimal policy). Mathematically, we want to make

$$p_{\pi^*}(s_t) = p_{\pi_\theta}(s_t)$$

The main idea is to instead being clever about $p_{\pi_\theta}(s_t)$, we will be clever about $p_{\pi^*}(s_t)$.

DAgger (Dataset Aggregation) [4] employs this idea. The goal is to collect training data (states) from the learned policy rather than the optimal policy.

Algorithm 1 DAgger

```
1: Train $\pi_\theta$ using expert demonstrations ($\mathcal{D} = s_1, a_1, s_2, a_2, ...$).
2: Run $\pi_\theta$ to get dataset ($\mathcal{D}_\pi = s_1, s_2, ...$).
3: Ask the expert to label $\mathcal{D}_\pi$ with actions $a_t$.
4: Aggregate $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$
5: Repeat until $p_{\pi^*}(s_t) = p_{\pi_\theta}(s_t)$
```

This paper shows how the algorithm helps solving the covariate shift. However, there is one big disadvantage. This method involves an expert to annotate new states and this step is expensive, time-consuming and not always practical.

DAgger addresses the problem of distributional ”drift”. What if our model is so good that there is no small errors that get accumulated over time leadig to covariate shift? So, another way of dealing with this issue is to mimic expert behaviour very accurately (but not overfitting).

There are two major reasons why we might fail to mimic the expert.

1. Non-Markovian Behaviour: It could be that the state is fully Markovian, but the expert decision is not fully Markovian and uses past history in its decision making process. We could use an LSTM/RNN architecture to incorporate history and mitigate the non-Markovian property.
2. Multimodal behaviour: It is very unnatural for wxperts (especially humans) to do the same thing twice, if we see the same thing twice, regardless of what happened before. There are various ways of dealing with this issue like employing output mixture of gaussians, latent variable models and autoregressive discretization.