22
Imitation Learning

As discussed in the previous chapter, the goal of reinforcement learning is to determine closed-loop control policies that result in the maximization of an accumulated reward, and RL algorithms are generally classified as either model-based or model-free. In both cases it is generally assumed that the reward function is known, and both typically rely on collecting system data to either update a learned model (model-based), or directly update a learned value function or policy (model-free).

While successful in many settings, these approaches to RL also suffer from several drawbacks. First, determining an appropriate reward function that can accurately represent the true performance objectives can be challenging[1]. Second, rewards may be *sparse*, which makes the learning process expensive in terms of both the required amount of data and in the number of failures that may be experienced when exploring with a suboptimal policy[2]. This chapter introduces the *imitation learning* approach to RL, where a reward function is not assumed to be known *a priori* but rather it is assumed the reward function is described implicitly through expert demonstrations.

[1] RL agents can sometimes learn how to exploit a reward function without actually producing the desired behavior. This is commonly referred to as *reward hacking*. Consider training an RL agent with a reward for each piece of trash collected. Rather than searching the area to find more trash (the desired behavior), the agent may decide to throw the trash back onto the ground and pick it up again!

[2] This issue of sparse rewards is less relevant if data is cheap, for example when training in simulation.

Imitation Learning

The formulation of the imitation learning problem is quite similar to the RL problem formulation from the previous chapter. The main difference is that instead of leveraging an explicit reward function $r_t = R(\boldsymbol{x}_t, \boldsymbol{u}_t)$ it will be assumed that a set of demonstrations from an expert are provided.

22.1 *Problem Formulation*

It will be assumed that the system is a Markov Decision Process (MDP) with a state $\boldsymbol{x}$ and control input $\boldsymbol{u}$, and the set of admissible states and controls are denoted as $\mathcal{X}$ and $\mathcal{U}$. The system dynamics are expressed by the probabilistic transition model:

$$p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}), \tag{22.1}$$

The field of RL often uses $\boldsymbol{s}$ to express the state and $\boldsymbol{a}$ to represent an action, but $\boldsymbol{x}$ and $\boldsymbol{u}$ will be used here for consistency with previous chapters.

which is the conditional probability distribution over $\boldsymbol{x}_t$, given the previous state and control. As in the previous chapter, the goal is to define a *policy* π that defines the closed-loop control law[3]:

[3] This chapter will consider a stationary policy for simplicity.

$$\boldsymbol{u}_t = \pi(\boldsymbol{x}_t). \tag{22.2}$$

The primary difference in formulation from the previous RL problem is that we do not have access to the reward function, and instead we have access to a set of expert demonstrations where each demonstration ξ consists of a sequence of state-control pairs:

$$\xi = \{(\boldsymbol{x}_0, \boldsymbol{u}_0), (\boldsymbol{x}_1, \boldsymbol{u}_1), \dots\}, \tag{22.3}$$

which are drawn from the expert policy π^*. The imitation learning problem is therefore to determine a policy π that imitates the expert policy π^*:

Definition 22.1.1 (Imitation Learning Problem)**.** *For a system with transition model* (22.1) *with states* $\boldsymbol{x} \in \mathcal{X}$ *and controls* $\boldsymbol{u} \in \mathcal{U}$*, the imitation learning problem is to leverage a set of demonstrations* $\Xi = \{\xi_1, \dots, \xi_D\}$ *from an expert policy* π^* *to find a policy* $\hat{\pi}^*$ *that imitates the expert policy.*

There are generally two approaches to imitation learning: the first is to directly learn how to imitate the expert's policy and the second is to indirectly imitate the policy by instead learning the expert's reward function. This chapter will first introduce two classical approaches to imitation learning (*behavior cloning* and the *DAgger* algorithm) that focus on directly imitating the policy. Then a set of approaches for learning the expert's reward function will be discussed, which is commonly referred to as *inverse reinforcement learning*. The chapter will then conclude with a couple of short discussions into related topics on learning from experts (e.g. through comparisons or physical feedback) as well as on interaction-aware control.

22.2 *Behavior Cloning*

Behavior cloning approaches use a set of expert demonstrations $\xi \in \Xi$ to determine a policy π that imitates the expert. This can be accomplished through supervised learning techniques, where the difference between the learned policy and expert demonstrations are minimized with respect to some metric. Concretely, the goal is to solve the optimization problem:

$$\hat{\pi}^* = \arg\min_{\pi} \sum_{\xi \in \Xi} \sum_{\boldsymbol{x} \in \xi} L(\pi(\boldsymbol{x}), \pi^*(\boldsymbol{x})),$$

where L is the cost function[4], $\pi^*(\boldsymbol{x})$ is the expert's action for at the state $\boldsymbol{x}$, and $\hat{\pi}^*$ is the approximated policy.

[4] Different loss functions could include p-norms (e.g. Euclidean norm) or f-divergences (e.g. KL divergence) depending on the form of the policy.

However this approach may not yield very good performance since the learning process is only based on a set of samples provided by the expert. In many cases these expert demonstrations will not be uniformly sampled across the

entire state space and therefore it is likely that the learned policy will perform poorly when not close to states found in ξ. This is particularly true when the expert demonstrations come from a *trajectory* of sequential states and actions, such that the *distribution* of the sampled states $\boldsymbol{x}$ in the dataset is defined by the expert policy. Then, when an estimated policy $\hat{\pi}^*$ is used in practice it produces its own distribution of states that will be visited, which will likely not be the same as in the expert demonstrations! This distributional mismatch leads to compounding errors, which is a major challenge in imitation learning.

22.3 DAgger: Dataset Aggregation

One straightforward idea for addressing the issue of distributional mismatch in states seen under the expert policy and the learned policy is to simply collect new expert data as needed[5]. In other words, when the learned policy $\hat{\pi}^*$ leads to states that aren't in the expert dataset just query the expert for more information! The behavioral cloning algorithm that leverages this idea is known as DAgger[6] (Dataset Aggregation).

[5] Assuming the expert can be queried on demand.

[6] 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 Artificial Intelligence and Statistics*. 2011, pp. 627–635

Algorithm 1: DAgger: Dataset Aggregation

Data: π^*
Result: $\hat{\pi}^*$
$\mathcal{D} \leftarrow 0$
Initialize $\hat{\pi}$
for $i = 1$ **to** N **do**
  $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}$
  Rollout policy π_i to sample trajectory $\tau = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \dots\}$
  Query expert to generate dataset $\mathcal{D}_i = \{(\boldsymbol{x}_0, \pi^*(\boldsymbol{x}_0)), (\boldsymbol{x}_1, \pi^*(\boldsymbol{x}_1)), \dots\}$
  Aggregate datsets, $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
  Retrain policy $\hat{\pi}$ using aggregated dataset $\mathcal{D}$
return $\hat{\pi}$

As can be seen in Algorithm 1, this approach iteratively improves the learned policy by collecting additional data from the expert. This is accomplished by rolling out the current learned policy for some number of time steps and then asking the expert what actions they would have taken at each step along that trajectory. Over time this process drives the learned policy to better approximate the true policy and reduce the incidence of distributional mismatch. One disadvantage to the approach is that at each step the policy needs to be retrained, which may be computationally inefficient.

22.4 Inverse Reinforcement Learning

Approaches that learn policies to imitate expert actions can be limited by several factors:

1. Behavior cloning provides no way to understand the underlying reasons for the expert behavior (no reasoning about outcomes or intentions).
2. The "expert" may actually be suboptimal[7].
3. A policy that is optimal for the expert may not be optimal for the agent if they have different dynamics, morphologies, or capabilities.

[7] Although the discussion of inverse RL in this section will also assume the expert is optimal, there exist approaches to remove this assumption.

An alternative approach to behavioral cloning is to reason about and try to learn a representation of the underlying reward function R that the expert was using to generate its actions. By learning the expert's intent, the agent can potentially outperform the expert or adjust for differences in capabilities[8]. This approach (learning reward functions) is known as *inverse reinforcement learning*.

[8] Learned reward representations can potentially generalize across different robot platforms that tackle similar problems!

Inverse RL approaches assume a specific parameterization of the reward function, and in this section the fundamental concepts will be presented by parameterizing the reward as a linear combination of (nonlinear) features:

$$R(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u}),$$

where $\boldsymbol{w} \in \mathbb{R}^n$ is a weight vector and $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u}) : \mathcal{X} \times \mathcal{U} \to \mathbb{R}^n$ is a feature map. For a given feature map $\boldsymbol{\phi}$, the goal of inverse RL can be simplified to determining the weights $\boldsymbol{w}$. Recall from the previous chapter on RL that the total (discounted) reward under a policy π is defined for a time horizon T as:

$$V_T^\pi(\boldsymbol{x}) = E\big[\sum_{t=0}^{T-1} \gamma^t R(\boldsymbol{x}_t, \pi(\boldsymbol{x}_t)) \mid \boldsymbol{x}_0 = \boldsymbol{x}\big].$$

Using the reward function $R(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u})$ this value function can be expressed as:

$$V_T^\pi(\boldsymbol{x}) = \boldsymbol{w}^\top \mu(\pi, \boldsymbol{x}), \quad \mu(\pi, \boldsymbol{x}) = E_\pi\big[\sum_{t=0}^{T-1} \gamma^t \boldsymbol{\phi}(\boldsymbol{x}_t, \pi(\boldsymbol{x}_t)) \mid \boldsymbol{x}_0 = \boldsymbol{x}\big],$$

where $\mu(\pi, \boldsymbol{x})$ is defined by an expectation over the trajectories of the system under policy π (starting from state $\boldsymbol{x}$) and is referred to as the *feature expectation*[9]. One insight that can now be leveraged is that by definition the optimal expert policy π^* will always produce a greater value function:

[9] Feature expectations are often computed using a Monte Carlo technique (e.g. using the set of demonstrations for the expert policy).

$$V_T^{\pi^*}(\boldsymbol{x}) \geq V_T^\pi(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \mathcal{X}, \quad \forall \pi,$$

which can be expressed in terms of the feature expectation as:

$$\boldsymbol{w}^{*\top} \mu(\pi^*, \boldsymbol{x}) \geq \boldsymbol{w}^{*\top} \mu(\pi, \boldsymbol{x}), \quad \forall \boldsymbol{x} \in \mathcal{X}, \quad \forall \pi. \tag{22.4}$$

Theoretically, identifying the vector $\boldsymbol{w}^*$ associated with the expert policy can be accomplished by finding a vector $\boldsymbol{w}$ that satisfies this condition. However this can potentially lead to ambiguities! For example, the choice $\boldsymbol{w} = 0$ satisfies this condition trivially! In fact, reward ambiguity is one of the main challenges associated with inverse reinforcement learning[10]. The algorithms discussed in the following chapters will propose techniques for alleviating this issue.

[10] A. Ng and S. Russell. "Algorithms for Inverse Reinforcement Learning". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. 2000, pp. 663–670

22.4.1 *Apprenticeship Learning*

The apprenticeship learning[11] algorithm attempts to avoid some of the problems with reward ambiguity by leveraging an additional insight from condition (22.4). Specifically, the insight is that it doesn't matter how well $\boldsymbol{w}^*$ is estimated as long as a policy π can be found that *matches the feature expectations*. Mathematically, this conclusion is derived by noting that:

$$\|\mu(\pi, \boldsymbol{x}) - \mu(\pi^*, \boldsymbol{x})\|_2 \leq \epsilon \implies |\boldsymbol{w}^\top \mu(\pi, \boldsymbol{x}) - \boldsymbol{w}^\top \mu(\pi^*, \boldsymbol{x})| \leq \epsilon$$

for any $\boldsymbol{w}$ as long as $\|\boldsymbol{w}\|_2 \leq 1$. In other words, as long as the feature expectations can be matched then the performance will be as good as the expert *even if the vector $\boldsymbol{w}$ does not match $\boldsymbol{w}^*$*. Another practical aspect to the approach is that it will be assumed that the initial state $\boldsymbol{x}_0$ is drawn from a distribution D such that the value function is also considered in expectation as:

$$E_{\boldsymbol{x}_0 \sim D}\big[V_T^\pi(\boldsymbol{x}_0)\big] = \boldsymbol{w}^\top \mu(\pi), \quad \mu(\pi) = E_\pi\big[\sum_{t=0}^{T-1} \gamma^t \boldsymbol{\phi}(\boldsymbol{x}_t, \pi(\boldsymbol{x}_t))\big].$$

This is useful to avoid having to consider all $\boldsymbol{x} \in \mathcal{X}$ when matching features[12].

To summarize, the goal of the apprenticeship learning approach is to find a policy π that matches the feature expectations with respect to the expert policy (i.e. makes $\mu(\pi)$ as similar as possible to $\mu(\pi^*)$)[13]. This is accomplished through Algorithm 2, which uses an iterative approach to finding better policies.

[11] P. Abbeel and A. Ng. "Apprenticeship Learning via Inverse Reinforcement Learning". In: *Proceedings of the Twenty-First International Conference on Machine Learning*. 2004

[12] Trying to find a policy that matches features for every possible starting state $\boldsymbol{x}$ is likely intractable or even infeasible.

[13] See Example 22.4.1 for an example of why matching features is intuitively useful.

Algorithm 2: Apprenticeship Learning

Data: $\mu(\pi^*)$, ϵ
Result: $\hat{\pi}^*$
Initialize policy π_0
for $i = 1$ **to** $\ldots$ **do**
  Compute $\mu(\pi_{i-1})$ (or approximate via Monte Carlo)
  Solve problem (22.5) with policies $\{\pi_0, \ldots, \pi_{i-1}\}$ to compute $\boldsymbol{w}_i$ and t_i

$$(\boldsymbol{w}_i, t_i) = \arg\max_{\boldsymbol{w}, t} \; t, \quad \text{s.t. } \boldsymbol{w}^\top \mu(\pi^*) \geq \boldsymbol{w}^\top \mu(\pi) + t, \quad \forall \pi \in \{\pi_0, \ldots, \pi_{i-1}\}, \quad \|\boldsymbol{w}\|_2 \leq 1. \tag{22.5}$$

  if $t_i \leq \epsilon$ **then**
    $\hat{\pi}^* \leftarrow$ best feature matching policy from $\{\pi_0, \ldots, \pi_{i-1}\}$
    return $\hat{\pi}^*$
  Use RL to find an optimal policy π_i for reward function defined by $\boldsymbol{w}_i$

To better understand this algorithm it is useful to further examine the optimization problem (22.5)[14]. Suppose that instead of making $\boldsymbol{w}$ a decision variable

[14] This problem can be thought of as an inverse RL problem that is seeking to find the reward function vector $\boldsymbol{w}$ such that the expert *maximally outperforms* the other policies.

it was actually fixed, then the resulting optimization would be:

$$t^*(\boldsymbol{w}) = \max_t \; t,$$
$$\text{s.t. } \boldsymbol{w}^\top \mu(\pi^*) \geq \boldsymbol{w}^\top \mu(\pi) + t, \quad \forall \pi \in \{\pi_0, \pi_1, \dots\},$$

which is essentially computing the smallest performance loss among the candidate policies $\{\pi_0, \pi_1, \dots\}$ with respect to the expert policy, *assuming the reward function weights are* $\boldsymbol{w}$. If $\boldsymbol{w}$ was known, then if $t^*(\boldsymbol{w}) \leq \epsilon$ it would guarantee that one of the candidate policies would effectively perform as well as the expert.

Since $\boldsymbol{w}$ is not known, the actual optimization problem (22.5) maximizes the smallest performance loss across *all vectors* $\boldsymbol{w}$ *with* $\|\boldsymbol{w}\|_2 \leq 1$. Therefore, if $t_i \leq \epsilon$ (i.e. the termination condition in Algorithm 2), then there must be a candidate policy whose performance loss is small *for all possible choices of* $\boldsymbol{w}$! In other words, there is a candidate policy that matches feature expectations well enough that good performance can be guaranteed without assuming the reward function is known, and without attempting to estimate the reward accurately.

Example 22.4.1 (Apprenticeship Learning vs. Behavioral Cloning)**.** Consider a problem where the goal is to drive a car across a city in as short of time as possible. In the imitation learning formulation it is assumed that the reward function is not known, but that there is an expert who shows how to drive across the city (i.e. what routes to take). A behavioral cloning approach would simply try to mimic the actions taken by the expert, such as memorizing that whenever the agent is at a particular intersection it should turn right. Of course this approach is not robust when at intersections that the expert never visited!

The apprenticeship learning approach tries to avoid the inefficiency of behavioral cloning by instead identifying features of the expert's trajectories that are more generalizable, and developing a policy that experiences the same feature expectations as the expert. For example it could be more efficient to notice that the expert takes routes without stop signs, or routes with higher speed limits, and then try to find policies that also seek out those features!

22.4.2 *Maximum Margin Planning*

The maximum margin planning approach[15] uses an optimization-based approach to computing the reward function weights $\boldsymbol{w}$ that is very similar to (22.5) but with some additional flexibility. In its most standard form the MMP optimization is:

[15] N. Ratliff, J. A. Bagnell, and M. Zinkevich. "Maximum Margin Planning". In: *Proceedings of the 23rd International Conference on Machine Learning*. 2006, pp. 729–736

$$\hat{w}^* = \arg\min_{\boldsymbol{w}} \; \|\boldsymbol{w}\|_2^2,$$
$$\text{s.t. } w^\top \mu(\pi^*) \geq w^\top \mu(\pi) + 1, \quad \forall \pi \in \{\pi_0, \pi_1, \dots\}.$$

Again this problem computes the reward function vector $\boldsymbol{w}$ such that the expert policy *maximally outperforms* the policies in the set $\{\pi_0, \pi_1, \dots\}$.

However the formulation is also improved in two ways: it adds a slack term to account for potential expert suboptimality and it adds a similarity function that gives more "margin" to policies that are dissimilar to the expert policy. This new formulation is:

$$\begin{aligned} \hat{w}^* = \arg\min_{\boldsymbol{w},v} \ & \|\boldsymbol{w}\|_2^2 + Cv, \\ \text{s.t. } & w^\top \mu(\pi^*) \geq w^\top \mu(\pi) + m(\pi^*, \pi) - v, \quad \forall \pi \in \{\pi_0, \pi_1, \dots\}, \end{aligned} \tag{22.6}$$

where v is a slack variable that can account for expert suboptimality, $C > 0$ is a hyperparameter that is used to penalize the amount of assumed suboptimality, and $m(\pi^*, \pi)$ is a function that quantifies how dissimilar two policies are.

One example of where this formulation is advantageous over the apprenticeship learning formulation (22.5) is when the expert is suboptimal. In this case it is possible that there is no $\boldsymbol{w}$ that makes the expert policy outperform all other policies, such that the optimization (22.5) returns $\boldsymbol{w}_i = 0$ and $t_i = 0$ (which is obviously not the appropriate solution). Alternatively the slack variables in the MMP formulation allow for a reasonable $\boldsymbol{w}$ to be computed.

22.4.3 *Maximum Entropy Inverse Reinforcement Learning*

While the apprenticeship learning approach shows that matching feature counts is a necessary and sufficient condition to ensure a policy performs as well as an expert, it also has some ambiguity (similar to the reward weight ambiguity problem discussed before). This ambiguity is associated with the fact that there could be different policies that lead to the same feature expectations!

This issue can also be thought of in a slightly more intuitive way in terms of distributions over trajectories. Specifically, a policy π induces a distribution over trajectories[16] $\tau = \{(\boldsymbol{x}_0, \pi(\boldsymbol{x}_0)), (\boldsymbol{x}_1, \pi(\boldsymbol{x}_1)), \dots\}$ that is denoted as $p_\pi(\tau)$. The feature expectations can be rewritten in terms of this distribution as:

$$\mu(\pi) = E_\pi\big[f(\tau)\big] = \int p_\pi(\tau) f(\tau) d\tau,$$

where $f(\tau) = \sum_{t=0}^{T-1} \gamma^t \phi(\boldsymbol{x}_t, \pi(\boldsymbol{x}_t))$. Now suppose a policy π was found that matched feature expectations[17] with an expert policy π^* such that:

$$\int p_\pi(\tau) f(\tau) d\tau = \int p_{\pi^*}(\tau) f(\tau) d\tau.$$

Crucially this condition is not sufficient to guarantee that $p_\pi(\tau) = p_{\pi^*}(\tau)$ (which would be ideal). In fact, the distribution $p_\pi(\tau)$ could also have an *arbitrary preference* for some paths that is *unrelated* to the feature matching objective.

The main idea in the maximum entropy inverse RL approach[18] is to not only match the feature expectations, but also remove ambiguity in the path distribution $p_\pi(\tau)$ by trying to make $p_\pi(\tau)$ *as broadly uncommitted as possible*. In other words, find a policy that matches feature expectations but otherwise has no additional path preferences. This concept is known as the maximum entropy principle[19].

[16] This distribution can be visualized as a set of paths generated by simulating the system many times with policy π (i.e. using a Monte Carlo method).

[17] For example by using apprenticeship learning.

[18] B. D. Ziebart et al. "Maximum Entropy Inverse Reinforcement Learning". In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*. 2008, pp. 1433–1438

[19] A maximum entropy distribution can be thought of as the *least informative distribution* of a class of distribution. This is useful in situations where it is undesirable to encode unintended prior information.

The maximum entropy IRL approach finds a minimally preferential, feature expectation matching distribution by solving the optimization problem:

$$\begin{aligned} p^*(\tau) = \arg\max_p \; & \int -p(\tau)\log p(\tau)d\tau, \\ \text{s.t.} \; & \int p(\tau)f(\tau)d\tau = \int p_{\pi^*}(\tau)f(\tau)d\tau, \\ & \int p(\tau)d\tau = 1, \\ & p(\tau) \geq 0, \quad \forall \tau, \end{aligned} \tag{22.7}$$

where the objective is the mathematical definition of a distribution's entropy, the first constraint requires feature expectation matching, and the remaining constraints ensure that $p(\tau)$ is a valid probability distribution. It turns out that the solution to this problem has the exponential form:

$$p^*(\tau, \boldsymbol{\lambda}) = \frac{1}{Z(\boldsymbol{\lambda})} e^{\boldsymbol{\lambda}^\top f(\tau)}, \quad Z(\boldsymbol{\lambda}) = \int e^{\boldsymbol{\lambda}^\top f(\tau)} d\tau,$$

where $Z(\boldsymbol{\lambda})$ normalizes the distribution, and where $\boldsymbol{\lambda}$ must be chosen such that the feature expectations match:

$$\int p^*(\tau, \boldsymbol{\lambda}) f(\tau) = \int p_{\pi^*}(\tau) f(\tau) d\tau.$$

In other words the maximum entropy IRL approach tries to find a distribution parameterized by $\boldsymbol{\lambda}$ that match features, but also requires that the distribution $p^*(\tau, \boldsymbol{\lambda})$ belong to the exponential family.

To determine the value of $\boldsymbol{\lambda}$ that matches features, it is assumed that the expert also selects trajectories with high reward with exponentially higher probability:

$$p_{\pi^*}(\tau) \propto e^{\boldsymbol{w}^{*\top} f(\tau)},$$

and therefore ideally $\boldsymbol{\lambda} = \boldsymbol{w}^*$. Of course $\boldsymbol{w}^*$ (and more generally $p_{\pi^*}(\tau)$) are not known, and therefore a maximum likelihood estimation approach is used to compute $\boldsymbol{\lambda}$ to best approximate $\boldsymbol{w}^*$ based on the sampled expert demonstrations[20].

In particular, an estimate $\hat{\boldsymbol{w}}^*$ of the reward weights is computed from the expert demonstrations $\Xi = \{\xi_0, \xi_1, \dots\}$ (which each demonstration ξ_i is a trajectory) by solving the maximum likelihood problem:

$$\begin{aligned} \hat{\boldsymbol{w}}^* &= \arg\max_{\boldsymbol{\lambda}} \prod_{\xi_i \in \Xi} p^*(\xi_i, \boldsymbol{\lambda}), \\ &= \arg\max_{\boldsymbol{\lambda}} \sum_{\xi_i \in \Xi} \boldsymbol{\lambda}^\top f(\xi_i) - \log Z(\boldsymbol{\lambda}), \end{aligned}$$

which can be solved using a gradient descent algorithm where the gradient is computed by:

$$\nabla_{\boldsymbol{\lambda}} J(\boldsymbol{\lambda}) = \sum_{\xi_i \in \Xi} f(\xi_i) - E_{\tau \sim p^*(\tau, \boldsymbol{\lambda})}[f(\tau)].$$

[20] By assuming the expert policy is also exponential, the maximum likelihood estimate is theoretically *consistent* (i.e. $\boldsymbol{\lambda} \to \boldsymbol{w}^*$ as the number of demonstrations approaches infinity).

The first term of this gradient is easily computable since the expert demonstrations are known, and the second term can be approximated through Monte Carlo sampling. However, this Monte Carlo sampling estimate is based on sampling trajectories from the distribution $p^*(\tau, \boldsymbol{\lambda})$. This leads to a iterative algorithm:

1. Initialize $\boldsymbol{\lambda}$ and collect the set of expert demonstrations $\Xi = \{\xi_0, \xi_1, \dots\}$.
2. Compute the optimal policy[21] $\pi_{\boldsymbol{\lambda}}$ with respect to the reward function with $\boldsymbol{w} = \boldsymbol{\lambda}$.
3. Using the policy $\pi_{\boldsymbol{\lambda}}$, sample trajectories of the system and compute an approximation of $E_{\tau \sim p^*(\tau, \boldsymbol{\lambda})}[f(\tau)]$.
4. Perform a gradient step on $\boldsymbol{\lambda}$ to improve the maximum likelihood cost.
5. Repeat until convergence.

[21] For example through traditional RL methods.

To summarize, the maximum entropy inverse reinforcement learning approach identifies a distribution over trajectories that matches feature expectations with the expert, but by restricting the distribution to belong to the exponential family ensures that spurious preferences (path preferences not motivated by feature matching) are not introduced. Additionally, this distribution over trajectories is parameterized by a value that is an estimate of the reward function weights.

22.5 Learning From Comparisons and Physical Feedback

Both behavioral cloning and inverse reinforcement learning approaches rely on expert demonstrations of behavior. However in some practical scenarios it may actually be difficult for the expert to provide complete/quality demonstrations. For example it has been shown[22] that when humans are asked to demonstrate good driving behavior in simulation they retroactively think their behavior was too aggressive! As another example, if a robot has a high-dimensional control or state space it could be difficult for the expert to specify the full high-dimensional behavior. Therefore another interesting question in imitation learning is to find a way to learn from alternative data sources besides complete demonstrations.

[22] C. Basu et al. "Do You Want Your Autonomous Car to Drive Like You?" In: *12th ACM/IEEE International Conference on Human-Robot Interaction*. 2017, pp. 417–425

22.5.1 Learning from Comparisons

One alternative approach is to use *pairwise comparisons*[23], where an expert is shown two different behaviors and then asked to rank which behavior is better. Through repeated queries it is possible to converge to an understanding of the underlying reward function. For example, suppose two trajectories τ_A and τ_B are shown to an expert and that trajectory τ_A is preferred. Then assuming that the reward function is:

$$R(\tau) = \boldsymbol{w}^\top f(\tau),$$

[23] D. Sadigh et al. "Active Preference-Based Learning of Reward Functions". In: *Robotics: Science and System*. 2017

where $f(\tau)$ are the collective feature counts (same as in Section 22.4), this comparison can be used to conclude that:

$$\boldsymbol{w}^\top f(\tau_A) > \boldsymbol{w}^\top f(\tau_B).$$

In other words, this comparison has split the space of possible reward weights $\boldsymbol{w}$ in half through the hyperplane:

$$(f(\tau_A) - f(\tau_B))^\top \boldsymbol{w} = 0.$$

By continuously querying the expert with new comparisons[24], the space of possible reward weights $\boldsymbol{w}$ will continue to shrink until a good estimate of $\boldsymbol{w}^*$ can be made. In practice the expert decision may be a little noisy and therefore the hyperplanes don't define hard cutoffs, but rather can be used to "weight" the possible reward vectors $\boldsymbol{w}$.

[24] The types of comparisons shown can be selectively chosen to maximally split the remaining space of potential $\boldsymbol{w}$ in order to minimize the total number of expert queries that are required.

22.5.2 *Learning from Physical Feedback*

Another alternative to learning from complete expert demonstrations is to simply allow the expert to physically interact with the robot to correct for undesirable behavior[25]. In this approach, a physical interaction (i.e. a correction) is assumed to occur when the robot takes actions that result in a lower reward than the expert's action.

[25] A. Bajcsy et al. "Learning Robot Objectives from Physical Human Interaction". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 217–226

For a reward function of the form $R(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{w}^\top \phi(\boldsymbol{x}, \boldsymbol{u})$ the robot maintains an estimate of the reward weights $\hat{\boldsymbol{w}}^*$ and the expert is assumed to have act according to a true set of optimal weights $\boldsymbol{w}^*$. Suppose the robot's policy, which is based on the estimated reward function with weights $\hat{\boldsymbol{w}}^*$, yields a trajectory τ_R. Then, if the expert physically interacts with the robot to make a correction the resulting actual trajectory τ_H is assumed to satisfy:

$$\boldsymbol{w}^{*\top} f(\tau_H) \geq \boldsymbol{w}^{*\top} f(\tau_R),$$

which simply states that the reward of the new trajectory is higher. This insight is then leveraged in a maximum a posteriori approach for updating the estimate $\hat{\boldsymbol{w}}^*$ after each interaction. Specifically, this update takes the form:

$$\hat{\boldsymbol{w}}^* \leftarrow \hat{\boldsymbol{w}}^* + \beta(\boldsymbol{f}(\tau_H) - \boldsymbol{f}(\tau_R)),$$

where $\beta > 0$ is a scalar step size. The robot then uses the new estimate to change its policy, and the process iterates. Note that this idea yields an approach that is similar to the concept of matching feature expectations from inverse reinforcement learning, except that the approach is iterative rather than requiring a batch of complete expert demonstrations.

22.6 *Interaction-aware Control and Intent Inference*

Yet another interesting problem in robot autonomy arises when robots and humans are interacting to accomplish shared or individual goals. Many classical

examples of this problem arise in autonomous driving settings, when human-driven vehicles interact with autonomous vehicles in settings such as highway merging or at intersections. While the imitation learning problems from the previous sections are focused on understanding the expert's behavior for the purpose of *imitating* the behavior, in this setting the human's behavior needs to be understood in order to ensure safe interactions. However there is an additional component to understanding interactions: *the robot's behavior can influence the human's behavior*[26].

[26] It is particularly important in interaction-aware robot control to understand the effects of the robot's actions on the human's behavior. Otherwise the human's could simply be modeled as dynamic obstacles!

22.6.1 Interaction-aware Control with Known Human Model

One common approach is to model the interaction between humans and robots as a dynamical system that has a combined state $\boldsymbol{x}$, where the robot controls are denoted $\boldsymbol{u}_R$ and the human decisions or inputs are denoted as $\boldsymbol{u}_H$. The transition model is therefore defined as:

$$p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{u}_{R,t-1}, \boldsymbol{u}_{H,t-1}).$$

In other words the interaction dynamics evolve according to the actions taken by both the robot and the human. In this interaction the robot's reward function is denoted as $R_R(\boldsymbol{x}, \boldsymbol{u}_R, \boldsymbol{u}_H)$ and the human's reward function is denoted as $R_H(\boldsymbol{x}, \boldsymbol{u}_R, \boldsymbol{u}_H)$, which are both functions of the combined state and both agent's actions[27].

[27] While R_R and R_H do not have to be the same, choosing $R_R = R_H$ may be desirable for the robot to achieve human-like behavior.

Under the assumption that both the robot and the human act optimally[28] with respect to their cost functions:

[28] While not necessarily true, this assumption is important to make the resulting problem formulation tractable to solve in practice.

$$\boldsymbol{u}_R^*(\boldsymbol{x}) = \arg\max_{\boldsymbol{u}_R} \; R_R(\boldsymbol{x}, \boldsymbol{u}_R, \boldsymbol{u}_H^*(\boldsymbol{x})),$$
$$\boldsymbol{u}_H^*(\boldsymbol{x}) = \arg\max_{\boldsymbol{u}_H} \; R_H(\boldsymbol{x}, \boldsymbol{u}_R^*(\boldsymbol{x}), \boldsymbol{u}_H).$$

Additionally, assuming both reward functions R_R and R_H are known[29], computing $\boldsymbol{u}_R^*$ is still extremely challenging due to the two-player game dynamics of the decision making process. However this problem can be made more tractable by modeling it as a *Stackelberg game*, which restricts the two-player game dynamics to a leader-follower structure. Under this assumption it is assumed that the robot is the "leader" and that as the follower the human acts according to:

[29] The reward function R_H could be approximated using inverse reinforcement learning techniques.

$$\boldsymbol{u}_H^*(\boldsymbol{x}, \boldsymbol{u}_R) = \arg\max_{\boldsymbol{u}_H} \; R_H(\boldsymbol{x}, \boldsymbol{u}_R, \boldsymbol{u}_H). \tag{22.8}$$

In other words the human is assumed to see the action taken by the robot *before* deciding on their own action. The robot policy can therefore be computed by solving:

$$\boldsymbol{u}_R^*(\boldsymbol{x}) = \arg\max_{\boldsymbol{u}_R} \; R_R(\boldsymbol{x}, \boldsymbol{u}_R, \boldsymbol{u}_H^*(\boldsymbol{x}, \boldsymbol{u}_R)), \tag{22.9}$$

which can be solved using a gradient descent approach. For the gradient descent approach the gradient of:

$$J(\boldsymbol{x}, \boldsymbol{u}_R) = R_R(\boldsymbol{x}, \boldsymbol{u}_R, \boldsymbol{u}_H^*(\boldsymbol{x}, \boldsymbol{u}_R)),$$

can be computed using the chain rule as:

$$\frac{\partial J}{\partial \boldsymbol{u}_R} = \frac{\partial R_R}{\partial \boldsymbol{u}_R} + \frac{\partial R_R}{\partial \boldsymbol{u}_H^*} \frac{\partial \boldsymbol{u}_H^*}{\partial \boldsymbol{u}_R}.$$

Since the reward function R_R is known the terms $\partial R_R / \partial \boldsymbol{u}_R$ and $\partial R_R / \partial \boldsymbol{u}_H^*$ can be easily determined. In order to compute the term $\partial \boldsymbol{u}_H^* / \partial \boldsymbol{u}_R$, which represents how much the robot's actions impact the human's actions, an additional step is required. First, assuming the human acts optimally according to (22.8) the necessary optimality condition is:

$$g(\boldsymbol{x}, \boldsymbol{u}_R, \boldsymbol{u}_H^*) = 0, \quad g = \frac{\partial R_H}{\partial \boldsymbol{u}_H},$$

which for the fixed values of $\boldsymbol{x}$ and $\boldsymbol{u}_R$ specifies $\boldsymbol{u}_H^*$. Then, by implicitly differentiating this condition with respect to the robot action $\boldsymbol{u}_R$:

$$\frac{\partial g}{\partial \boldsymbol{u}_R} + \frac{\partial g}{\partial \boldsymbol{u}_H^*} \frac{\partial \boldsymbol{u}_H^*}{\partial \boldsymbol{u}_R} = 0,$$

which can be used to solve for:

$$\frac{\partial \boldsymbol{u}_H^*}{\partial \boldsymbol{u}_R}(\boldsymbol{x}, \boldsymbol{u}_R, \boldsymbol{u}_H^*) = -\Big(\frac{\partial g}{\partial \boldsymbol{u}_H^*}\Big)^{-1} \frac{\partial g}{\partial \boldsymbol{u}_R}.$$

Notice that every term in this expression can be computed[30] and therefore it can be substituted into the gradient calculation:

$$\frac{\partial J}{\partial \boldsymbol{u}_R} = \frac{\partial R_R}{\partial \boldsymbol{u}_R} - \frac{\partial R_R}{\partial \boldsymbol{u}_H^*}\Big(\frac{\partial g}{\partial \boldsymbol{u}_H^*}\Big)^{-1} \frac{\partial g}{\partial \boldsymbol{u}_R},$$

which can then be computed as long as it is possible to compute $\boldsymbol{u}_H^*(\boldsymbol{x}, \boldsymbol{u}_R)$.

To summarize, one approach to interaction-aware control is to model the interaction as a Stackelberg game, where it is assumed that both the human and the robot act optimally with respect to some reward functions. This formulation of the problem enables the robot to choose actions based on an implicit understanding of how the human will react.

[30] Assuming the human's reward function is known.

22.6.2 *Intent Inference*

One disadvantage to the approach for interaction-aware control from the previous section is that it assumes the human acts optimally with respect to a *known* reward function. While a reward function could be learned through inverse reinforcement learning, this is not practical for real-world settings where different humans behave differently. Returning to the example of interaction between human drivers and autonomous vehicles, the human could exhibit drastically different behavior depending on whether they have an aggressive or passive driving style. In these settings the problem of *intent inference* focuses on identifying underlying behavioral characteristics that can lead to more accurate behavioral models[31].

[31] This problem can be formulated as a partially observable Markov decision process (POMDP) since the underlying behavioral characteristic is not directly observable, yet influences the system's behavior.

One approach to intent inference[32] is to model the underlying behavioral differences through a set of unknown parameters $\boldsymbol{\theta}$ which need to be inferred by observing the human's behavior. Mathematically this is expressed by defining the human's reward function $R_H(\boldsymbol{x}, \boldsymbol{u}_R, \boldsymbol{u}_H, \boldsymbol{\theta})$ to be a function of $\boldsymbol{\theta}$, and assuming the human chooses actions according to:

$$p(\boldsymbol{u}_H \mid \boldsymbol{x}, \boldsymbol{u}_R, \boldsymbol{\theta}) \propto e^{R_H(\boldsymbol{x}, \boldsymbol{u}_R, \boldsymbol{u}_H, \boldsymbol{\theta})}.$$

In other words this model assumes the human is exponentially more likely to pick optimal actions[33], but that they may pick suboptimal actions as well.

The objective of intent inference is therefore to estimate the parameters $\boldsymbol{\theta}$, which can be accomplished through Bayesian inference methods. In the Bayesian approach a *probability distribution* over parameters $\boldsymbol{\theta}$ is updated based on observations. Specifically the belief distribution is denoted as $b(\boldsymbol{\theta})$, and given an observation of the human's actions $\boldsymbol{u}_H$ the belief distribution is updated as:

$$b_{t+1}(\boldsymbol{\theta}) = \frac{1}{\eta} p(\boldsymbol{u}_{H,t} \mid \boldsymbol{x}_t, \boldsymbol{u}_{R,t}, \boldsymbol{\theta}) b_t(\boldsymbol{\theta}),$$

where η is a normalizing constant. This Bayesian update is simply taking the prior belief over $\boldsymbol{\theta}$ and updating the distribution based on the likelihood of observing human action $\boldsymbol{u}_H$ under that prior. Note that this concept is quite similar to the concepts of inverse reinforcement learning: a set of parameters that describe the human's (experts) behavior are continually updated when new observations of their actions are gathered.

While the robot could sit around and *passively* observe the human act to collect samples for the Bayesian updates, it is often more efficient for the robot to *probe* the human to take interesting actions that are more useful for revealing the intent parameters $\boldsymbol{\theta}$. This can be accomplished by choosing the robot's reward function to be:

$$R_R(\boldsymbol{x}, \boldsymbol{u}_R, \boldsymbol{u}_H, \boldsymbol{\theta}) = I(b(\boldsymbol{\theta}), \boldsymbol{u}_R) + \lambda R_{\text{goal}}(\boldsymbol{x}, \boldsymbol{u}_R, \boldsymbol{u}_H, \boldsymbol{\theta})$$

where $\lambda > 0$ is a tuning parameter and $I(b(\boldsymbol{\theta}), \boldsymbol{u}_R)$ denotes a function that quantifies the amount of information gained with respect to the belief distribution from taking action $\boldsymbol{u}_R$. In other words the robot's reward is a tradeoff between exploiting the current knowledge of $\boldsymbol{\theta}$ to accomplish the objective and taking exploratory actions to improve the intent inference. With this robot reward function the robot's actions are chosen to maximize the expected reward:

$$\boldsymbol{u}_R^*(\boldsymbol{x}) = \arg\max_{\boldsymbol{u}_R} \; E_{\boldsymbol{\theta}}[R_R(\boldsymbol{x}, \boldsymbol{u}_R, \boldsymbol{u}_H, \boldsymbol{\theta})].$$

To summarize, this robot policy will try to simultaneously accomplish the robot's objective and gather more information to improve the inference of the human's intent (modeled through the parameters $\boldsymbol{\theta}$). In a highway lane changing scenario this type of policy might lead the robot to nudge into the other lane to see if the other car will slow down (passive driving behavior) or try to

[32] D. Sadigh et al. "Planning for cars that coordinate with people: leveraging effects on human actions for planning and active information gathering over human internal state". In: *Autonomous Robots* 42.7 (2018), pp. 1405–1426

[33] This assumption was also used in the Maximum Entropy IRL approach.

block the lane change (aggressive driving behavior). Once the robot has a strong enough belief about the human's behavior it may choose to either complete the lane change or slow down to merge behind the human driver.

Bibliography

[1] P. Abbeel and A. Ng. "Apprenticeship Learning via Inverse Reinforcement Learning". In: *Proceedings of the Twenty-First International Conference on Machine Learning*. 2004.

[2] A. Bajcsy et al. "Learning Robot Objectives from Physical Human Interaction". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 217–226.

[3] C. Basu et al. "Do You Want Your Autonomous Car to Drive Like You?" In: *12th ACM/IEEE International Conference on Human-Robot Interaction*. 2017, pp. 417–425.

[4] A. Ng and S. Russell. "Algorithms for Inverse Reinforcement Learning". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. 2000, pp. 663–670.

[5] N. Ratliff, J. A. Bagnell, and M. Zinkevich. "Maximum Margin Planning". In: *Proceedings of the 23rd International Conference on Machine Learning*. 2006, pp. 729–736.

[6] 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 Artificial Intelligence and Statistics*. 2011, pp. 627–635.

[7] D. Sadigh et al. "Active Preference-Based Learning of Reward Functions". In: *Robotics: Science and System*. 2017.

[8] D. Sadigh et al. "Planning for cars that coordinate with people: leveraging effects on human actions for planning and active information gathering over human internal state". In: *Autonomous Robots* 42.7 (2018), pp. 1405–1426.

[9] B. D. Ziebart et al. "Maximum Entropy Inverse Reinforcement Learning". In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*. 2008, pp. 1433–1438.