17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 20: Imitation Learning In Real Life

Lecturer: Sanjiban Choudhury
Scribe: Yang Zhou, Miaosi Dong, Pranjal, Harshita Diddee

20.1 Why Imitation Learning?

There are two main reasons why imitation learning (IL) is used in real-world setups:

20.1.1 The Alignment Problem

It is intractable to enumerate all the rules and rewards necessary to completely and correctly incentivize the right actions to an agent. Hand-designing rewards often relies on implicit "common sense" or shared values that the agent doesn't possess.

- **For example,** When tasked with making a cup of coffee, a reward that awards a cup of coffee may need to be accompanied with an explicit penalty for procuring the coffee through other means than actually making the coffee (Stealing someone else's coffee!).
- **Another example:** Trying to encode driving rules from an official handbook would fail in complex situations. (e.g., in gridlock intersections)

The above example also demonstrates *explicitly* programming reward is tedious, error prone, hackable. Instead, robots should infer rewards from natural human feedback. That's why imitation learning is needed. Finally, a rule-based approach to maximizing rewards is also unlikely to be sustainable in the noisy, real world where humans and other agents may not comply with the rules expected during the agent's policy training.

Summary: Given this brittleness and impracticality of hand engineering rewards, it is much more desirable to design a way of learning from the ideal behavior demonstrated by humans. This learning should help the agent infer the rewards from demonstrations provided by the human and also allow the agent to actively query the human to seek demonstrations in scenarios where it isn't able to infer a reward. Imitation Learning is a step in this direction.

In nutshell, alignment problem can be defined as:

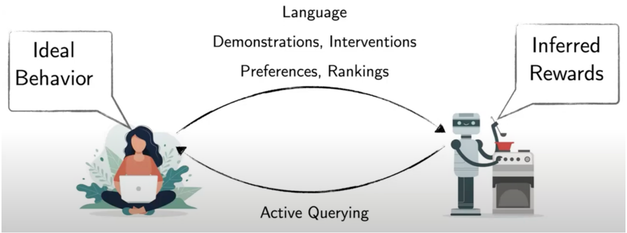


Figure 20.1: Alignment Problem

Definition 1 (Alignment Problem) *We have two entities: humans and robots. Humans have some ideal behavior in mind, which is difficult to express in functional form, but can be conveyed naturally through language, demonstrations, preferences, or rankings. The goal of the Robot is to actively query the human to gather more information to infer the reward.*

(as Figure 20.1 illustrates.)

20.1.2 The Collaboration Problem

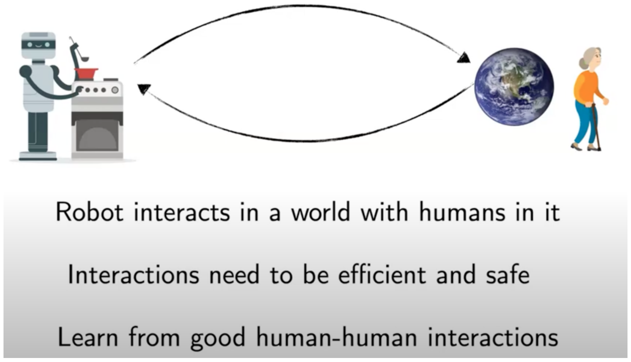
We want to consider the human in the environment as first-class citizens.

- **Example of the MOSAIC system (A Modular System for Assistive and Interactive Cooking)**: A human user interacts with a team of robots assisting them in cooking a recipe. The recipe exists in the user's mind and is not explicitly known to the robots. There is a high degree of personalization—such as different ways of cooking—and the robot's goal is to help the human achieve this personalized outcome. The robots must infer what the human wants to do, and equally what they would prefer not to do because it is too boring or repetitive, allowing the human to focus on tasks that are creative and engaging. The robot observes the human through cameras, predicts their intent, and collaborates accordingly.

Based on this example, the definition of the collaboration problem is as follows:

Definition 2 (Collaboration Problem) *Robots interact in a world with humans in it. Hence, all their interactions must be efficient and safe. This makes it impossible for them to explore with the human in the loop, as is necessitated by traditional reinforcement learning. We must be able to learn from good human-human interaction.*

(as Figure 20.2 illustrates.)



Robot interacts in a world with humans in it

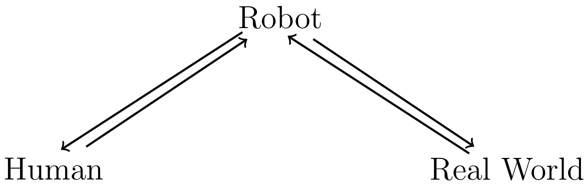
Interactions need to be efficient and safe

Learn from good human-human interactions

Figure 20.2: Alignment Problem

20.1.3 Key Challenge in IL

The key challenge in IL is jointly optimizing the performance and efficiency of both the (a) agent-human interaction and (b) agent-real-world interaction.



20.2 Challenge 1: Agent-Human Interactions

Setup: We are attempting to train a policy for a self-driving robotic race car [1].

- **Data Collection:** Collect human demonstration data $(s_1^*, a_1^*), (s_2^*, a_2^*), ...$
  - **State Space(s):** Occupancy Map (an observation about what parts of its environment are occupied/obstacles, while which are not).
  - **Action Space(a):** Joystick Directions.
- **Train a policy(classifier)** $\pi : \Phi \to \Delta(\mathcal{A})$ (Select one out of a set of possible trajectories that closely match what the human was trying to show)

We review the space of IL approaches for training such a policy.

20.2.1 Behavior Cloning (BC)

The first method is behavior cloning [2] (i.e. directly regressing from states to actions). This is the supervised learning / maximum likelihood estimation approach to IL.

Result: Empirically, BC often achieves a reasonable validation error, but during rollouts, the car crashes quite frequently.

Reason: When the policy makes a small mistake and enters a state not seen in the expert data (e.g., heading towards a wall), it lacks information on how to recover. It often defaults to the most common action seen in demonstrations (e.g., "drive straight"), leading to a crash. This is because the policy is only evaluated on the expert's state distribution, not its own induced state distribution. More formally,

$$\text{Training Loss: } \sum_{t=0}^{T-1} \mathbb{E}_{s_t \sim d_t^{\pi_E}} \left[\ell(s_t, \pi(s_t))\right] \neq \text{Test Loss: } \sum_{t=0}^{T-1} \mathbb{E}_{s_t \sim d_t^{\pi}} \left[\ell(s_t, \pi(s_t))\right] \tag{20.1}$$

20.2.2 Dataset Aggregation (DAgger)

To mitigate this, we naturally gravitate towards a system that increases the coverage of the learned policy to observe how the expert will respond in states that the learned policy is likely to encounter. This involves querying the expert in all the states where the car visits and then aggregating the expert's labels for these instances with the original data.

Training Algorithm: DAgger

1. Roll out the current learner policy to sample robot states.
2. Collect post-hoc expert action labels at ***all robot states***.
3. Aggregate the (learner state, expert action) data with that from previous rounds.
4. Update policy on aggregate data $\mathcal{D}$:

$$\min_{\pi} \mathbb{E}_{s,a \sim \mathcal{D}}[\mathbb{I}(\pi(s) \neq a_E)]. \tag{20.2}$$

By training our policy on the states it actually visits rather than just the ideal ones visited by the expert, we can eliminate compounding errors.

$$\begin{aligned} \text{Naive BC:} \quad & J(\pi) - J(\pi_E) \leq \mathcal{O}(\varepsilon T^2), \\ \text{DAgger:} \quad & J(\pi) - J(\pi_E) \leq \mathcal{O}(\varepsilon T). \end{aligned}$$

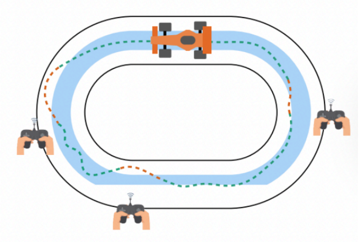


Figure 20.3: HG-DAgger: Interactive Imitation Learning with Human Experts

Practical Concern: While DAgger provides a nice performance bound, querying the expert for every action that the learning policy visits can be expensive, risky (e.g., driving in the real world), and hard to scale (as the environment becomes more complex).

20.2.3 Human-Gated DAgger (HG-DAgger) [3]

Instead of querying the expert *everywhere* the learner goes, we can instead only have the expert intervene when necessary, as in Figure 20.3.

Training Algorithm: HG-DAgger (i.e. *learning form interventions*):

1. Roll out the current learner policy to sample robot states.
2. Collect action labels at states where the expert intervened.
3. Aggregate the (learner state, expert action) data with that from previous rounds.
4. Update policy on aggregate data $\mathcal{D}$:

$$\min_{\pi} \mathbb{E}_{s,a\sim\mathcal{D}}[\mathbb{I}(\pi(s) \neq a_E)]. \tag{20.3}$$

Concern: Though more efficient in terms of querying the expert, the agent would learn to make a mistake and then recover from it, rather than avoiding the mistake in the first place.

20.2.4 Expert Intervention Learning (EIL)

An alterative perspective on expert interventions is that they tell us information about the expert's *latent Q-value function.* Specifically, we can model the expert as intervening at the last moment before even they couldn't fix the agent's behavior. We could then try and infer this Q_E-function from data of when and how they intervene. More formally,

$$\begin{aligned}
\min_{Q_E \in \mathcal{Q}} \quad & \mathbb{E}_{(s,a_E)\sim P_{\text{expert}}}\left[\ell(Q_E(s), a_E)\right] \\
\text{subject to:} \quad & Q_E(s,a) \leq \delta_{\text{good}} \quad \forall (s,a) \in \text{(I)} \text{ (before expert intervenes)} \\
& Q_E(s,a) \geq \delta_{\text{good}} \quad \forall (s,a) \in \text{(II)} \quad \text{(after expert intervenes)} \\
& Q_E(s,a) \leq \min_{a'} Q_E(s,a') \quad \forall (s,a) \in \text{(III)} \text{ (expert intervenes optimally)}
\end{aligned}$$

Note we're using costs rather than rewards above – a low Q_E is a good thing. By inferring such a Q_E, we also know what states we should avoid – those where $\min_a Q_E(s,a)$ is big. We can then augment the HG-DAgger loss (i.e. match the intervention action) with an additional term to avoid the states where interventions happened – see paper for more.

20.3 Scaling to the Real World

When training a real self-driving car, one of the most important differences from our above toy setup is the architecture used, which borrows from developments in language modeling. These days, SDC policies are often transformers [4], as Figure 20.4 illustrates:

1. A Scene Encoder might be pretrained on vast amounts of (suboptimal) driving data. We want to be able to forecast how *all* actors act, not just those who act optimally.

2. A Decoder is fine-tuned to predict future actions on *good* driving data.

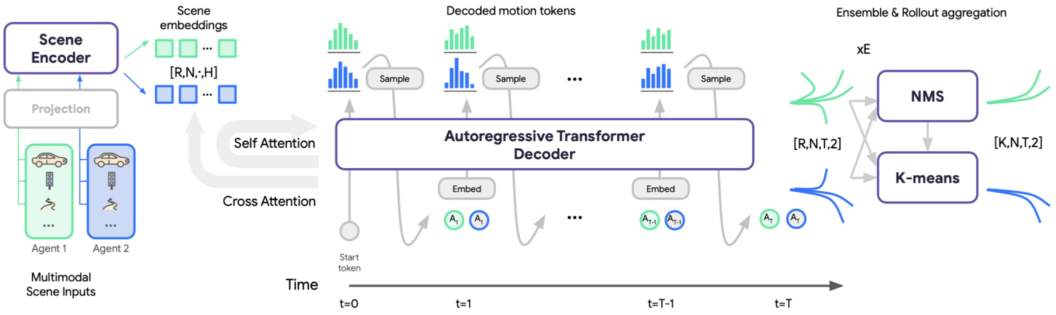


Figure 20.4: MotionLM Architecture

However, demonstrations alone are often missing some information we'd like our SDC to be trained on. First, demonstrations only tell you what to do – they don't tell you what *not* to

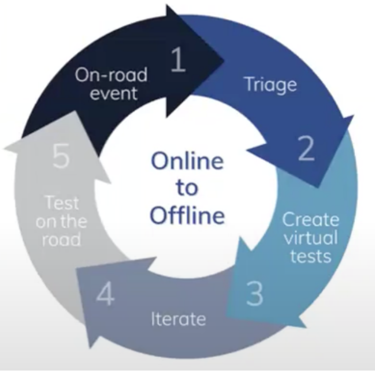


Figure 20.5: The real-world analog of EIL is using interventions to define unit tests.

do. Second, demonstrations don't show recovery behavior from mistakes the expert did not make themselves. We will now discuss how we can use ideas like EIL to address this.

When deploying a self-driving car in the real world, a human operator will sometimes need to intervene and correct the behavior of the system. We can then triage such events and use them to generate tests that specify what right / wrong behavior is in such situations. More generally, we can think of interventions as *automatic preference generation*: we prefer not to enter intervention states and we prefer what the expert did to what the robot was going to do. We can then use this preference data to back out a reward function to apply RL to, similar to the RL form Human Feedback pipeline we will discuss soon.

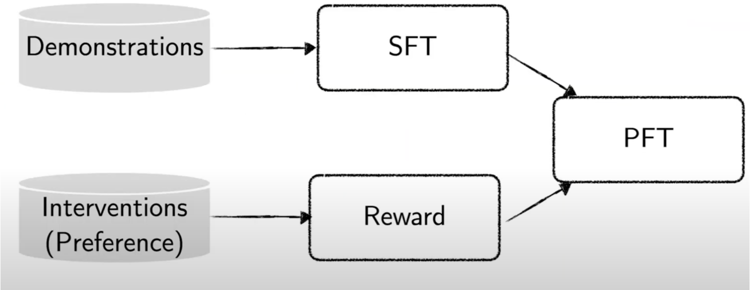


Figure 20.6: An RLHF Pipeline for Self-Driving.

20.4 *

References

[1] Siddhartha S. Srinivasa, Patrick Lancaster, Johan Michalove, Matt Schmittle, Colin Summers, Matthew Rockett, Rosario Scalise, Joshua R. Smith, Sanjiban Choudhury,

Christoforos Mavrogiannis, and Fereshteh Sadeghi. Mushr: A low-cost, open-source robotic racecar for education and research, 2023.

[2] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991.

[3] Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. Hg-dagger: Interactive imitation learning with human experts, 2019.

[4] Ari Seff, Brian Cera, Dian Chen, Mason Ng, Aurick Zhou, Nigamaa Nayakanti, Khaled S Refaat, Rami Al-Rfou, and Benjamin Sapp. Motionlm: Multi-agent motion forecasting as language modeling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8579–8590, 2023.