Statistical Techniques in Robotics (16-831, S21) Lecture #26 (Wednesday, May 5)

DAgger, AggreVaTe, Wrap-Up

Lecturer: Kris Kitani *Scribes: Xiaochen Han, Ingrid Navarro-Anaya*

1 Review

In this section, we briefly survey the topics covered during the previous lecture which include: Maximum Entropy Inverse Reinforcement Learning (MaxEnt IRL) and Generative Adversarial Imitation Learning (GAIL) two types of active Imitation Learning (IL) algorithms.

1.1 MaxEnt IRL

1.1.1 Entropy

We first discussed the notion of entropy, $\mathcal{H}$, which is central to the MaxEnt IRL [5] algorithm. Intuitively, such notion states that the most appropriate distribution to model a set of data is the one with the highest entropy among all those satisfying our prior knowledge.

Formally, the entropy of a discrete distribution is given by:

$$\mathcal{H}(p(X)) = \sum_{x \in sup(p(X))} [-p(x)log(p(x))] \tag{1}$$

Alternatively, entropy can be expressed in a continuous setting:

$$\mathcal{H}(p(X)) = \int_x -p(x)log(p(x))dx \tag{2}$$

1.1.2 MaxEnt IRL

We discussed that an unconstrained IRL problem has many solutions for a given expert policy π_E. To address this, we have learned that previous algorithms (e.g., LP-IRL) introduce a regularizer to their objective. In particular, MaxEnt IRL proposes an entropic regularizer. The general idea of this regularizer follows from the principle of maximum entropy, which states that given a set of distributions that explain a set of data equally well, the best choice is the one that has the highest entropy.

MaxEnt IRL assumes that we are given a set of expert trajectories $\{\zeta_1, \ldots, \zeta_N\} \sim \hat{p}(\zeta)$ and wants to find a distribution $p(\zeta)$ that closely resembles the underlying expert distribution $\hat{p}(\zeta)$. As such, we derived the following objective for MaxEnt IRL:

$$\max_{p(\zeta)}[\mathcal{H}(p(\zeta))] = \min_{p(\zeta)}[-\mathcal{H}(p(\zeta))] \tag{3}$$

$$\max_{p(\zeta)}[\mathcal{H}(p(\zeta))] = \min_{p(\zeta)}[\sum_{\zeta} p(\zeta)log(p(\zeta))] \tag{4}$$

Then, we formulated the MaxEnt IRL as the following constrained optimization problem:

$$\begin{aligned} \min_{p(\zeta)} \quad & \sum_{\zeta} p(\zeta) log(p(\zeta)) \\ \text{s.t.} \quad & \sum_{d} [p(\zeta_d)\mu(\zeta_d)] - \sum_{\zeta} [p(\zeta)\mu(\zeta)] = 0 \\ & \sum_{\zeta} [p(\zeta)] - 1 = 0 \end{aligned} \tag{5}$$

, which can be solved using the Lagrangian constrained optimization:

$$L(p(\zeta), \theta, V) = \sum_{\zeta} [p(\zeta) log(p(\zeta))] + \theta^T (\sum_{d} [p(\zeta_d)\mu(\zeta_d)] - \sum_{\zeta} [p(\zeta)\mu(\zeta)]) + V \sum_{\zeta} [p(\zeta)] - 1 \tag{6}$$

Finally, by solving the Lagrangian above, we derived the algorithm's update by parametrizing $p(\zeta)$ by θ:

$$L(\theta) = \theta^T \sum_{d} [p(\zeta_d)\mu(\zeta_d)] - log(\sum_{\zeta} [exp(\theta^T \mu(\zeta)]) \tag{7}$$

$$= \nabla_\theta L(\theta) = \mu_d - \bar{\mu} \tag{8}$$

1.2 GAIL

1.2.1 Problem Setup

In the previous lecture, we also reviewed Generative Adversarial Imitation Learning (GAIL) [2] framework. This paradigm is a technique used to extract a policy directly from demonstrations but as if it was obtained using RL followed by IRL. The algorithm and exploits generative adversarial training to fit distributions of state-action pairs that define expert behavior. As such, this approach is closely connected to the GAN [1] framework.

Put simply, GAIL defines a policy that tries to generate trajectories similar to those produced by an expert. Further, it defines a scoring function which attempts to detect the generated trajectories.

1.2.2 Formulation

Discriminator Objective. To formulate the discriminator (or cost function), assume we are given a dataset of expert trajectories $\mathcal{D}^* = \{s_1, a_1, \ldots, s_N, a_N\}$, as well as fake trajectories $\mathcal{D}_\theta = \{s_1, a_1, \ldots, s_M, a_M\}$. The goal is to learn a discriminator, $D_\phi(s, a) \in [0, 1]$, that is able to distinguish between a true state-action pair and generated one. This is expressed through the following objective:

$$\max_{\phi} \Big\{ \sum_{s,a \sim \mathcal{D}_\theta} [D_\phi(s, a)] + \sum_{s,a \sim \mathcal{D}^*} [1 - D_\phi(s, a)] \Big\} \tag{9}$$

i.e., we want to maximize the distance function parametrized by ϕ such that the difference between discriminating against a true trajectory and a fake one point is as large as possible.

Generator Objective. GAIL introduces a neural-based policy $G_\theta(s)$ that produces an action given a state and interacts auto-regressively with the environment $\mathcal{E}$ to generate a trajectory. We want to learn a policy that generates trajectories indistinguishable from the expert trajectories. To do so, assume we are given the cost function $D_\phi(s, a)$. The policy objective becomes:

$$\min_\theta \Big\{ \sum_{s,a \sim \mathcal{D}_\theta} [D_\phi(s, a)] \Big\} \tag{10}$$

i.e., we want to minimize the cost function given the parameters θ used to generate the data.

GAIL Objective. Finally, by combining both of the objectives derived above, we get the GAIL objective shown below:

$$\min_\theta \max_\phi \Big\{ \sum_{s,a \sim \mathcal{D}_\theta} [\ln\{D_\phi(s, a)\}] + \sum_{s,a \sim \mathcal{D}^*} [\ln\{1 - D_\phi(s, a)\}] \Big\} \tag{11}$$

2 Summary

In this section, after discussing a lot on Active IL, we are going to learn about the other two types of imitation learning: Passive IL and Interactive IL. Figure 1 compares differences among 3 types of imitation learning.

	Passive IL	**Active IL**	**Interactive IL**
Demonstrations $\mathcal{D}$	yes	yes	optional
Environment $\mathcal{E}$	no	yes	yes
Oracle π	no	no	yes
Dynamics $\mathcal{T}$	no	optional	optional
Reward $\mathcal{R}$	no	optional	optional

Figure 1: 3 Types of Imitation Learning.

2.1 Passive IL

2.1.1 Problem Setup

As we can observe from Figure 1, passive IL sets a scenario where an agent only has access to demonstrations. The agent does not have access to the environment, and as such, it does not get any kind of feedback, nor has has access to the dynamics.

One example of this type of IL, is behavioral cloning (supervised learning) shown in Figure 2. In this paradigm, an agent gets expert demonstrations $\mathcal{D}^* = \{s_n, a_n\}_{n=1}^N$ which are passed through some sort of black-box (e.g. a neural network) from which we want to learn a function $\pi(a|s)$ representing (cloning) expert behavior.

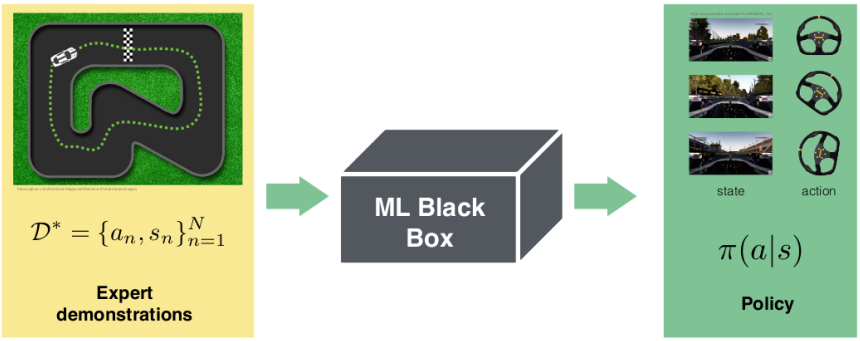

Figure 2: Passive IL.

2.1.2 Discussion

Behavioral cloning suffers from covariate shift issues. At a high-level, covariate shift arises from having different test and training distributions. Now, since the agent (most likely) does not have data from all possible states, it can only model short-term behavior. Thus, we only train agents from current (short-term) inputs often lacking diversity. This may cause the agent to overfit to current inputs and is specially problematic in sequential problems where agents might accumulate small errors that ultimately lead to *catastrophic* scenarios.

For instance, imagine we train an agent to navigate on a road using this paradigm. Imagine the expert demonstrations only include driving "perfectly" straight, and thus, the agent is only shown that type of behavior. The agent in this scenario will learn to *always* navigate straight. However, if at some point the agent starts accumulating errors by steering slightly less straight (or because it has to turn and it has not learned to do so) this could eventually lead the agent to a set of states which it has never seen. Thus, in not knowing how to respond to such unknown set, the agent might end up in a state from which it might not be able to recover, like crashing into a wall.

2.2 Interactive IL

2.2.1 Problem Setup

As mentioned above, one potential problem of Passive IL is that it only models short-term (one-shot) behavior. During test, the agent is likely to get into unseen states with accumulation of small errors. Unless there are data from all states, the problem still remain. Moreover, collecting enough data is challenging because data with negative label (e.g. samples lead to crash) are rare.

To solve the problem, we introduce Interactive IL. Compared with Passive IL, Interactive IL has the ability to interact with the environment, i.e. it executes current policy and queries experts to get feedback. We will introduce two Interactive IL algorithms DAgger and AggreVaTe in the following.

2.2.2 DAgger

DAgger (Data Aggregation) [4] is proposed to solve two main problems of IL: data dependencies (not i.i.d.) and errors accumulation. Figure 3 illustrates the framework of DAgger and Algorithm 1 shows the pseudo code of DAgger. It first mixtures policy between expert policy and hindsight optimal policy, then samples state-action pairs using mix policy from the environment. DAgger aggregates all sampled states with former data and update the hindsight optimal policy via minimizing the zero-one loss function.

Algorithm 1 DAgger

1: **for** $k = 1, \cdots, K$ **do**
2: $\pi_k \triangleq \beta\pi^* + (1-\beta)\hat{\pi}_k$
3: $\{s^{(t)}, a^{(t)}\}_{t=1}^{T} \sim \varepsilon|\pi_k$
4: $\mathcal{D} \leftarrow \mathcal{D} \cup \{s^{(t)}\}_{t=1}^{T}$
5: $\hat{\pi}_{k+1} = \arg\min_{\pi\in\Pi} \sum_{d=1}^{|\mathcal{D}|} f(\pi(s^{(d)}), \pi^*(s^{(d)}))$
6: **end for**
7: **return** $\hat{\pi}$

Recall the general framework for Online Learning where the online learner updates parameters according to the obtained loss from the nature in each iteration, we notice that Interactive IL can be solved with the FTL (Follow-The-Leader) algorithm and DAgger is just FTL algorithm which is no-regret given a convex loss function.

DAgger addresses domain shift problem by sampling trajectories which implicitly accounts for sequential dependence of actions and reduces IL to online learning results in no-regret bounds. However, it solves for each iteration for the optimal policy which is costly and treats all mistakes equally.

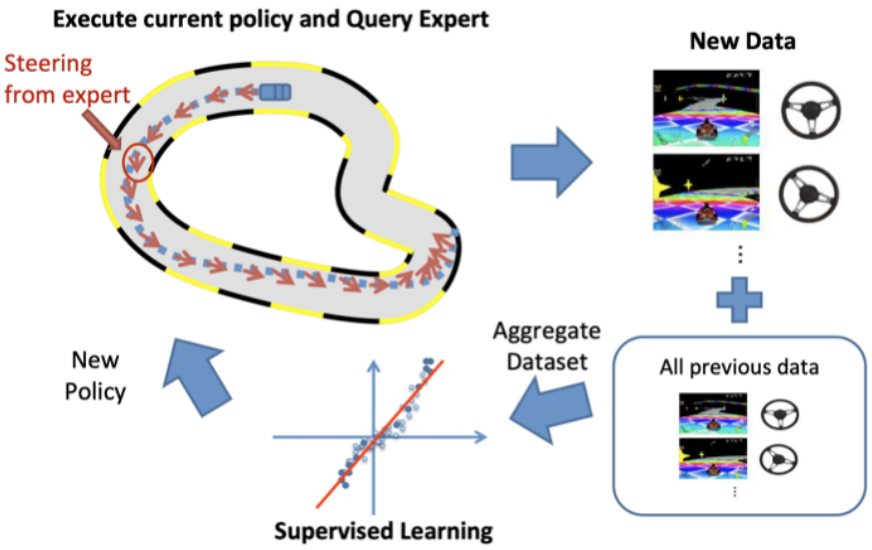

Figure 3: Illustration of DAgger.

2.2.3 AggreVaTe

DAgger minimizes the number of mistakes of the learned policy compared to expert and treats all mistakes equally. In reality, mistakes are different, some mistakes can be recovered while others may be fatal. We would like to improve DAgger to assign different penalties to different mistakes.

AggreVaTe (Aggregate Values to Imitate) [3] is proposed to overcome DAgger's deficiencies. Algorithm 2 shows the pseudo code of AggreVaTe. Firstly, it follows the aggregated policy π_k and stop at a random time step t. Then it takes each exploratory action and follows expert policy π^* to obtain cumulative cost from the expert. After that it aggregates the state and cumulative cost into a cost sensitive dataset, indicating different weights of predictions, implying that some mistakes are more costly. AggreVaTe loops this procedure by resetting the state and recording all the cost Q values. Different from DAgger, AggreVaTe uses Q function as loss function, it updates the optimal policy which minimizes the expert's cost-to-go.

Algorithm 2 AggreVaTe

1: **for** $k = 1, \cdots, K$ **do**
2: $\quad \pi_k \triangleq \beta\pi^* + (1-\beta)\hat{\pi}_k$
3: $\quad t_k \sim [T]$
4: $\quad \{s^{(i)}\}_{i=0}^{t_k} \sim \varepsilon|\pi_k$
5: $\quad \mathcal{D} \leftarrow \mathcal{D} \cup \{s^{(t)}, Q(s^{(t_k)})\}_{t=1}^{T}$
6: $\quad \hat{\pi}_{k+1} = \arg\min_{\pi\in\Pi} \sum_{d=1}^{|\mathcal{D}|} \sum_a \pi(a|s_d) Q^*(s_d, a)$
7: **end for**
8: **return** $\hat{\pi}$

2.2.4 Conclusion

The difference between DAgger and AggreVaTe can be summarized as follows:

- DAgger treats all the mistakes equally while in AggreVaTe some mistakes are worse (fatal) than others (can be recovered).
- AggreVaTe asks expert the consequence of mistakes (i.e. Q function) rather than the correctness of the action.
- AggreVaTe minimizes the cost of mistakes made by the policy instead of the number of mistakes the policy make.

Since the data are sequential (not i.i.d.), traditional statistical learning tools are not ideal for understanding analyzing sequential decisions. To analyze sequential decision making problems, online learning with no-regret analyses would be a proper choice. Imitation and reinforcement learning enable super-expert performance. Imitation learning avoids global exploration, thus it is more sample efficient.

3 Wrap-Up

This section serves as a brief summary of the content discussed in this course.

3.1 Hitchhiker's guide to RoboStats

The goal of *16-831 Statistical Techniques in Robotics* was to provide a theoretical introduction for decision making and statistical learning approaches that are commonly used in robotics. The course is mainly focused in modeling data driven sequential decision processes, and in particular, this course followed (in the order indicated by the arrows) the topics outlined in the Figure 4.

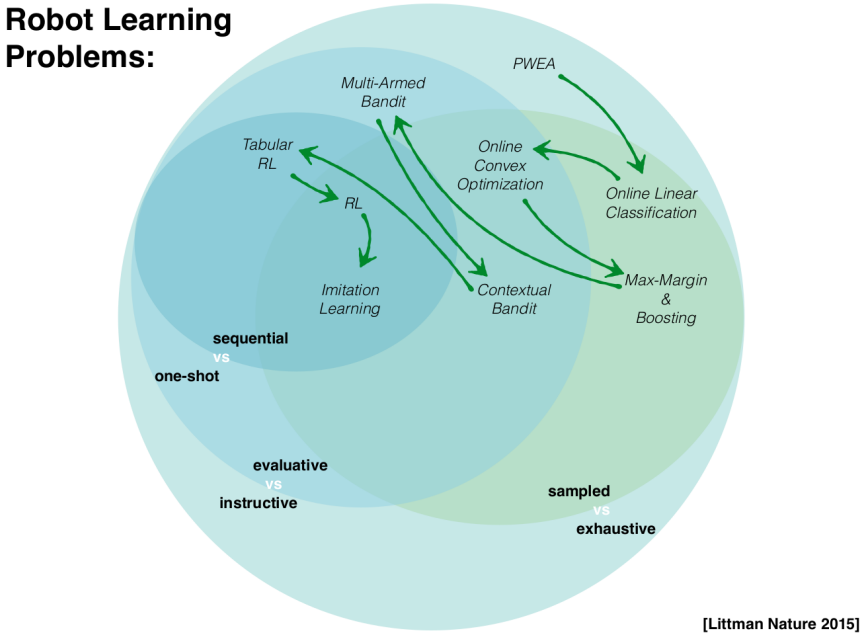

Figure 4: Statistical Techniques in Robotics

We started off the class by setting up the problem that a robot system requires to operate and learn under different conditions and it is up to us, the algorithm designers, to understand such problem settings and its appropriate solutions. To do so, we need to know what type of feedback do we get from the environment(s) our agents interact with. For this reason, we defined three questions to guide ourselves in making a decision as to what type of approach to use in a given problem:

1. Does the **learner** get to see all the data?
   - *Exhaustive feedback.* The learner gets exposed to all possible situations (e.g., tabular reinforcement learning).
   - *Sample feedback.* The learner only gets to access a subset of the situations (e.g, online linear classification).
2. What kind of **feedback** do we get from the **environment**?
   - *Evaluative feedback.* The learner does not have information for all alternative actions (e.g., reinforcement learning, imitation learning algorithms)
   - *Instructive feedback.* The learner receives a score for all actions possible (e.g., prediction with expert advice)

3. Is **feedback** sequence dependent?

    - *Sequential feedback.* The learner's current action affects the future state/action (e.g., reinforcement learning, imitation learning).
    - *One-Shot feedback.* The learner's current action does not affect the next action (e.g., bandits).

For reference, the following list includes all the algorithms covered during class:

1. **Prediction with Expert Advice**
    - Greedy
    - Randomized Greedy
    - Majority
    - Weighted Majority
    - Randomized Weighted Majority
2. **Online Classificaton**
    - Perceptron
    - Winnow
    - SVM
    - AdaBoost
3. **Multi-Armed Bandits**
    - Upper Confidence Bound
    - Thompson Sampling
    - EXP3
    - EXP4
4. **Online Convex Optimization**
    - Follow the Leader
    - Follow the Regularized Leader
    - Online Gradient Descent
    - Online Mirror Descent
5. **Reinforcement Learning**
    - Policy Iteration
    - Value Iteration
    - TD Control
    - Monte-Carlo Control
    - Value Approximation
    - Policy Gradient
    - Actor-Critic
6. **Imitation Learning**
    - Linear Program IRL
    - Game Matrix IRL
    - Max Margin IRL
    - Structured Max Margin IRL
    - Max Entropy IRL
    - GAIL
    - DAgger
    - AggreVate

References

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial networks. *Commun. ACM*, 63(11):139–144, 2020.

[2] J. Ho and S. Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016.

[3] S. Ross and J. A. Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

[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 artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[5] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.