DAgger: Interactive Experts and No-Regret Learning

Sanjiban Choudhury
Let’s travel to the INFINITE data limit!

The Three Regimes of Covariate Shift
Easy

Expert is realizable

\[ \pi^E \in \Pi \]

As \( N \to \infty \), drive down

\[ \epsilon = 0 \quad \text{(or Bayes error)} \]

Nothing special.
Collect lots of data and
do Behavior Cloning
**Easy**

- Expert is realizable
  \[ \pi^E \in \Pi \]

- As \( N \to \infty \), drive down \( \epsilon = 0 \) (or Bayes error)

**Solution**

- Nothing special.
- Collect lots of data and do Behavior Cloning

**Hard**

- Non-realizable expert + limited expert support
Non-realizable expert + limited support?

No label for what to do in this state!
Non-realizable expert + limited support?

Behavior Cloning compounds in error $O(\epsilon T^2)$

[Ross & Bagnell '10]
**Easy**

Expert is realizable

\[ \pi^E \in \Pi \]

As \( N \to \infty \), drive down

\[ \epsilon = 0 \] (or Bayes error)

Nothing special.

Collect lots of data and do Behavior Cloning

---

**Hard**

Non-realizable expert + limited expert support

Even as \( N \to \infty \), behavior cloning \( O(\epsilon T^2) \)
Why can’t we just collect data \((s, a^*)\) on the distribution of states the learner visits?
Introducing an *interactive* expert!
To know the distribution, you need a learner.
To train a learner, you need a distribution.
Activity!
How can we solve the chicken and egg problem, i.e. train the learner on a distribution of states it visits?
An *embarrassingly* simple algorithm: FORWARD

Idea: Train a different learner policy at every timestep by interactively querying expert

Get start state samples $s_0 \sim d^0(.)$

for $t = 0 \ldots T-1$

Query interactive expert to get

$a_t^* = \pi^*(s_t)$

Train a learner policy at time $t$

$\pi^t = \text{Train}(s_t, a_t^*)$

Execute learner policy $\pi^t$ to get

next state samples $s_{t+1} \sim d_{\pi^t}^{t+1}(.)$
But what if we want ONE policy?
DAGGER: A meta-algorithm for imitation learning

A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning

Stéphane Ross  
Robotics Institute  
Carnegie Mellon University  
Pittsburgh, PA 15213, USA  
stephaneross@cmu.edu

Geoffrey J. Gordon  
Machine Learning Department  
Carnegie Mellon University  
Pittsburgh, PA 15213, USA  
ggordon@cs.cmu.edu

J. Andrew Bagnell  
Robotics Institute  
Carnegie Mellon University  
Pittsburgh, PA 15213, USA  
dbagnell@ri.cmu.edu
DAgger: Iteration 0

Data

Human drives

Policy $\pi_0$ [Ross et al'11]
DAgger: Iteration 1

[Ross et al'11]

Robot $\pi_0$ drives

Human corrects!

AGGREGATE DATA

Policy $\pi_1$
DAgger: Iteration 2

Robot $\pi_1$ drives

AGGREGATE DATA
DAgger: Iteration 1

Robot $\pi_N$ drives

After many iterations ....
we are able to drive like a human!
Dagger (Dataset Aggregation)

- Aggregated Training Distribution
- Test Distribution
- Human Distribution
But why does aggregating data work?
The Imitation Game

Learner

Initialize policy

Update policy

Adversary

Chooses loss

\[ \pi_1 \text{ [policy]} \]

\[ \pi_2 \]

\[ l_1(\cdot) \text{ [loss]} \]

\[ l_2(\cdot) \]
Imitation learning is just a game

Be stable

Slowly change predictions
Let’s prove!
How can I customize DAGGER to be more practical?
Q1. The policy iteration at step 1 is crappy and visits irrelevant states. What do I do?

Blend the expert and learner policy \[ \pi_i = (1 - \beta_i)\hat{\pi}_i + \beta_i\pi^* \]

Q2. What if I can’t afford to store all the aggregated data?

Online gradient descent!
Original results from DAGGER!
DAGGER is a foundation

Imitation under uncertainty

SAIL
ExPLORE
STROLL
Counterfactual Teaching

Agnostic
SysID
DaaD

Model learning

DAEQUIL
AGGREGVATE(D)

Imitation learning

DPI
LOLS
NRPI

Reinforcement Learning

EIL
HG-DAGGER
SHIV

Query efficient imitation learning

DAGGER

Imitation under uncertainty

Counterfactual Teaching

Agnostic
SysID
DaaD

Model learning

DAEQUIL
AGGREGVATE(D)

Imitation learning

DPI
LOLS
NRPI

Reinforcement Learning

EIL
HG-DAGGER
SHIV

Query efficient imitation learning

DAGGER
Many cool applications of DAGGER in robotics

Lee et al, Learning quadrupedal locomotion over challenging terrain (2020)

Choudhury et al, Data Driven Planning via Imitation Learning (2018)

Chen et al, Learning by Cheating (2020)

Pan et al, Imitation learning for agile autonomous driving (2019)
DAGGER is not just for imitation learning!

Model-based Reinforcement Learning
DAGGER is not just for imitation learning!

(a) Forward simulation of learned model (gray) introduces error at each prediction step compared to the true time-series (red)

(b) Data provides a demonstration of corrections required to return back to proper prediction
Hidden charges from DAGGER
Hidden charge #1: Not all mistakes are equal

Figure 1: Problem with the imitation learning formulation. The expert policy stays perfectly on the race track. The learner immediately goes off the race track and visits bad states.

Consider the example shown in Fig. 1. The expert stays on the track and never visits bad states $S_{bad}$. The learner, on the other hand, immediately drifts off into $S_{bad}$. Moreover, for all $s \in S_{bad}$, the learner can perfectly imitate the expert. In other words, $(s, \hat{\pi}) = 0$ for these states. In fact, it is likely that for certain policy classes, this is the optimal solution! At the very least, DAGER is susceptible to learn such policies as is shown in the counter example in Laskey et al. [8].

Various alternate formulations for imitation learning

The phenomenon discussed in the previous section leads to a host of interesting questions and problems. We will simply list them here and then tackle them one by one.

4.1 Constrained policy search

It seems we can overcome the 'cheating in bad states' phenomenon by constraining the space of policies to not enter $S_{bad}$ (or have state visitation probabilities to be low under the induced distribution). How do we specify such constraints and how do we solve IL under such constraints?

4.2 Distribution matching

If we dig a little deeper, we realize that instead of minimizing imitation loss as we do in (1), what we really care about is making sure our policy distribution is the same as the experts. This is exactly the premise of GAIL [5].

4.3 Expert in the loop

Say we have an 'expert-in-the-loop' that overrides the learner during roll-out. This would cause a train test distribution mismatch? How should we handle this?

4.4 Actively minimizing calls to the expert

It seems wasteful for the expert to have to label corrective actions that are not relevant to the task, e.g. $s_2 \in S_{bad}$. That being said, it seems wasteful for the expert to have to label states where the policy has

Dagger minimizes 0-1 loss, but what we really want to optimize are advantages! (More next lecture)
Hidden charge #2: Dagger asks the expert for queries everywhere

We’ll see how to learn from limited human feedback (interventions)
Hidden charge #3: Dagger expects at least one policy to be good everywhere

... but can’t cross the bridge! ...

... but can take the long way round.
To know the distribution, you need a learner.
To train a learner, you need a distribution.

---

**The Imitation Game**

**Learner**
- Initialize policy
- Update policy

**Adversary**
- Chooses loss
- Chooses loss

---

**DAgger: Iteration 1**

[Ross et al’11]

Data

Robot $x_0$ drives

Human corrects!

Old Data

AGGREGATE DATA

Policy $\pi_1$