Imitation Learning

A Practical Perspective

Overview

- Understand the <u>concepts</u> behind the 3 main imitation methods:
  - Behavioural Cloning (BC)
  - DAGGER
  - Inverse RL (incl. GAIL)
- Describe each methods strengths and disadvantages:
  - Why use DAGGER over BC?
- Outline practical considerations:
  - Even though using DAGGER is always better, when does using BC/IRL make more sense?

Why Imitation Learning?

- Labels are expensive:
  - A large practical motivation underlying RL's development.
  - RL reduces the required supervision to a <u>scalar, potentially sparse function.</u>
- But sometimes RL isn't enough by itself:
  - "Hard Exploration" environments - Montezuma's Revenge, Starcraft 2, Go, etc.
  - Environments without rewards - (useful) reward functions difficult to define.
  - Interaction is expensive - you can't get 100 million states off a robot in a reasonable time.
  - Random exploration is impossible - random actions on a robot can destroy it.
- Imitation learning enables:
  - Fast initial learning / model iteration, bypassing random exploration phase
  - Can be used to **learn reward functions** - "Inverse RL"
  - Can always finetune imitation policy with RL.

Imitation Learning Pseudocode

```
1. Collect minibatch of states $\{X_1, \ldots, X_N\}$ using policy $\pi_{\mathcal{D}}$.
2. Label each state $X_i$ using $Y_i = \pi_E(X_i)$ with expert policy $\pi_E$.
3. Minimize some loss $L_\theta(\{X_1, \ldots, X_N\}, \{Y_1, \ldots, Y_N\})$
        wrt some model $\pi_\theta$ parameterized by $\theta$.
4. Repeat from 1.
```

What changes between BC/DAGGER?

1. Collect minibatch of states $\{X_1, \ldots, X_N\}$ using policy $\boxed{\pi_{\mathcal{D}}}$.
2. Label each state X_i using $Y_i = \pi_E(X_i)$ with expert policy π_E.
3. Minimize some loss $L_\theta(\{X_1, \ldots, X_N\}, \{Y_1, \ldots, Y_N\})$

   wrt some model π_θ parameterized by θ.
4. Repeat from 1.

Behavioural Cloning: Sample from the Expert

1. Collect minibatch of states $\{X_1, \ldots, X_N\}$ using policy $\boxed{\pi_E}$.
2. Label each state X_i using $Y_i = \pi_E(X_i)$ with expert policy π_E.
3. Minimize some loss $L_\theta(\{X_1, \ldots, X_N\}, \{Y_1, \ldots, Y_N\})$

   wrt some model π_θ parameterized by θ.
4. Repeat from 1.

Train/Test Distribution Mismatch

- We optimize our loss over the distribution of data sampled from π_E.
- At test time, our states are sampled from the policy π_θ.

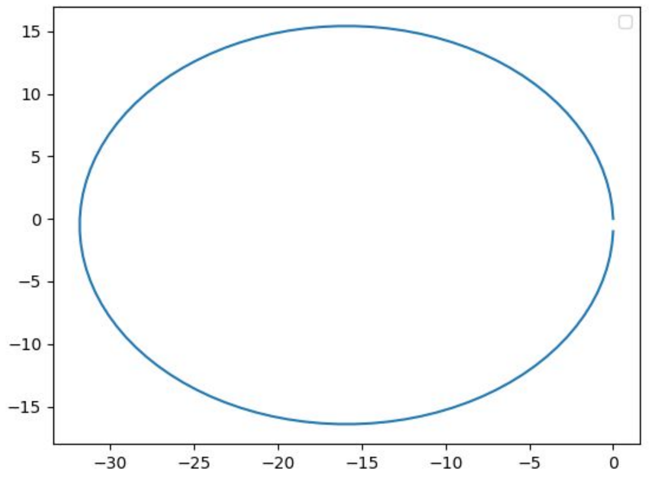
If there's any error in our model: $p_\theta(s) \neq p_E(s)$

There will **always be error** (model error, label noise, observation noise, etc.)

Even if somehow there's no error, there might not be enough data to accurately model π_E, i.e. highly stochastic environment.

What this means practically

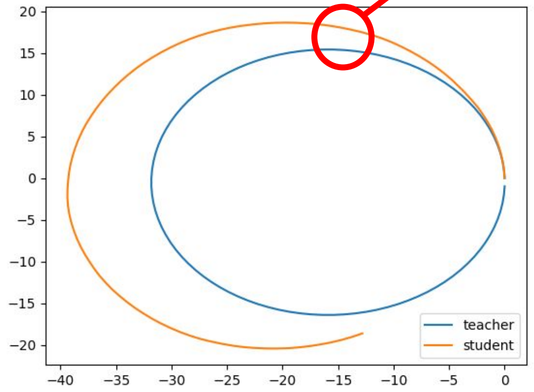
- Suppose we want to train a driving policy to go around in a loop.
- We provide a demonstration of a human driving around the loop once.
- The states are (x,y) and the action is turning angle (with speed fixed).



What this means practically

- By accident, the steering sensor was calibrated incorrectly.
- It introduces systematic label noise: $\hat{y} = y + \epsilon, \epsilon \sim \mathcal{N}(0.01, 0.01)$
  - I.e. noise has non-zero expectation.
- Even though **our model has 0 training error**, because of (even small) label noise **the policy is catastrophic when executed**.

By this point the car has went off-road and crashed.



The Problem with Behavioural Cloning

- BC’s data distribution is too optimistic:
  - It tells the student what to do **when things are going correctly**.
  - The accumulation of model error, label noise, observation noise, etc., means at test time **things will always go wrong when sampling from the student**.
  - Near catastrophe, **BC provides no labeled supervision.**
- DAGGER was designed to fix this:
  - Always **sample states from the student** π_θ.
  - **Query the expert** π_E **to relabel** the student-sampled states.

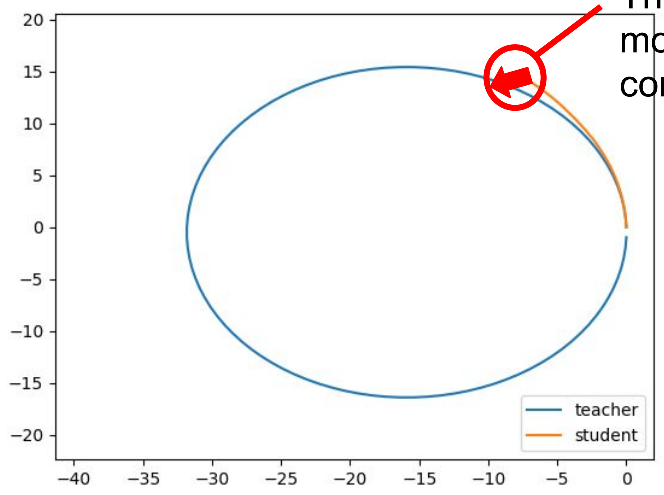
DAGGER: Sample from the Student

1. Collect minibatch of states $\{X_1, \ldots, X_N\}$ using policy $\boxed{\pi_\theta}$.
2. Label each state X_i using $Y_i = \pi_E(X_i)$ with expert policy π_E.
3. Minimize some loss $L_\theta(\{X_1, \ldots, X_N\}, \{Y_1, \ldots, Y_N\})$

   wrt some model π_θ parameterized by θ.
4. Repeat from 1.

What this means practically

- Back to the label noise example, as soon as the student starts veering off the road, the expert will relabel and tell the student to move back onto the road.



The agent will receive instructions from the human to move back onto the road, eventually learning to correct away from its original catastrophic trajectory.

So why use BC at all?

- We have just shown that BC can cause significant problems during test time.
- We have also shown DAGGER fixes these problems - so why use BC?
  - Expert **re-labeling is even more expensive than demonstration** - you need an **expert queryable at any state**. Why not just use the expert at test time if this is feasible?
  - Needs expert to be able to recover from near-catastrophe - this is often the hardest part of control within the environment, so **the expert needs near-perfect competency** (discounts crowdsourcing).
- BC is simpler and just works if you have enough data:
  - Used to train initial version of AlphaGo, superhuman Go player.
  - Used in initial training phase of Alphastar, grandmaster Starcraft II agent.
- You **can always fine-tune the BC-trained policy using RL**.
  - Often called "**bootstrapping**" (not to be confused with TD "bootstrapping")

Inverse Reinforcement Learning

- An alternative to imitation learning:
  - Use demonstrations to **learn a reward function**.
  - **Train a policy** using learnt reward function.
- Least expensive form of supervision - **don’t need full demonstrations**:
  - RL phase can “fill in” missing behavior given partial demonstrations.
- Argued to be a more comprehensive model of expert behavior:
  - **Learning why** the expert did something instead of mapping states to actions.
  - This can potentially **generalize better**.

Generative Adversarial Imitation Learning

- GAIL is a particular form of inverse RL that learns a reward function that tries to **match state distributions between the expert and student**.
- We will avoid going into the theoretical background here and instead focus on the concepts underlying its practical implementation.
  - The paper focuses a lot of text on the theoretical implications of the method.
  - The implementation is actually very straightforward.

Generative Adversarial Networks

- Generative models (model a distribution over inputs, typically images).
- GANs have achieved some of the most impressive results in deep learning.
- Images below are completely synthetic - generated by “BigGAN”.
- You can run this model and sample new images:
    - https://github.com/ajbrock/BigGAN-PyTorch

Generative Adversarial Networks

- Similar in concept to GAIL:
  - A generator G_θ generates images that aim to be indistinguishable from a training dataset.
  - A discriminator $\mathcal{D}_\phi$ discriminates between real images and generator-generated images.

- Generator takes as input a sample from a noise distribution: $z \sim p_N(\cdot)$
  - This is what allows sampled images to be different between each generation.

GAN Pseudocode

Initialize $\mathcal{D}_\phi$,G_θ only once at start of execution.

1. Generate images sampled from G_θ, $\{X_i^\theta\}_{i=1}^N$ and from dataset $\{X_i^E\}_{i=1}^N$.
2. Train a classifier (discriminator) $\mathcal{D}_\phi$ to output 0 on $\{X_i^\theta\}_{i=1}^N$ and 1 on $\{X_i^E\}_{i=1}^N$.
3. Train generator G_θ maximizing $\mathbb{E}_{z\sim p_N(\cdot)}[\log D(G(z))]$
4. Repeat from 1.

Generative Adversarial Imitation Learning

- Given an MDP:
  - A policy π_θ **samples states** that aim to be indistinguishable from expert-sampled states.
  - A discriminator $\mathcal{D}_\phi$ discriminates **between states sampled from student and expert**.

- The generative process is now sampling from an MDP using a policy.
- **No longer differentiable!** ..because of MDP dynamics:
  - Can't use gradients from discriminator.
  - Generator needs to be trained using RL - using log discriminator probability.

GAIL Pseudocode

Initialize $\mathcal{D}_\phi$, π_θ only once at start of execution.

```
1. Generate states sampled from student $\{X_i^\theta\}_{i=1}^N$ and from expert $\{X_i^E\}_{i=1}^N$.
2. Train a classifier (discriminator) $\mathcal{D}_\phi$ to output 0 on $\{X_i^\theta\}_{i=1}^N$ and 1 on $\{X_i^E\}_{i=1}^N$.
3. Train policy $\pi_\theta$ using reward function $r(s) = \log \mathcal{D}_\phi(s)$
4. Repeat from 1.
```

GAIL Pseudocode

Initialize $\mathcal{D}_\phi$, π_θ only once at start of execution.

```
1. Generate states sampled from student $\{X_i^\theta\}_{i=1}^N$ and from expert $\{X_i^E\}_{i=1}^N$.
2. Train a classifier (discriminator) $\mathcal{D}_\phi$ to output 0 on $\{X_i^\theta\}_{i=1}^N$ and 1 on $\{X_i^E\}_{i=1}^N$.
3. Train policy $\pi_\theta$ using reward function $r(s) = \log \mathcal{D}_\phi(s)$
4. Repeat from 1.
```

Implementing this for HW2 might be easier than it seems:

- For 2., use the supervised learning example code you wrote in the "Preliminaries" part of colab.
- For 3., use CMAES - you'll need to pass in the custom reward function.