Imitation Learning (IL)

What:

- Part of Machine Learning, applied in Reinforcement Learning
- Used when manually designing a reward function that satisfies the desired behaviour is extremely complicated.
  (usually complex, graphical / sequential actions)
- Learn the optimal policy directly from expert decisions (which is known as trajectory: $\tau = (s_0, a_0, s_1, a_1, \ldots)$)

Classifications:

- Behavioural Cloning (BC)
- Directly Policy Learning
- Inverse Reinforcement Learning (IRL)

Differences between IL algorithms:

① Learning algorithm

② Loss function

Advants · Drawbacks:

① Behavioural Cloning:

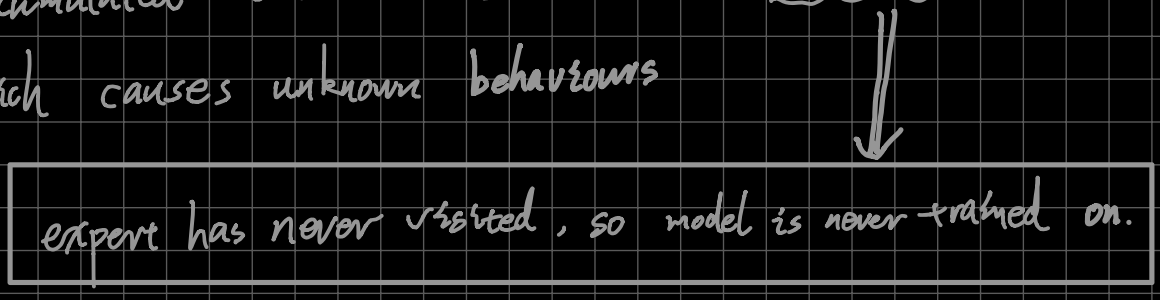
Advantages:

Simple and efficient on short term planning.

Drawbacks:

MDP doesn't satisfy the IID assumption ⇒
(独立同分布)

Accumulated Reward can lead to unknown states,
which causes unknown behaviours



② Directly Policy Learning

Advantages:

Efficient in long term planning / complex planning

Drawbacks:

Needs iterative experts.

③ Inverse Reinforcement Learning (IRL)

Advantages:

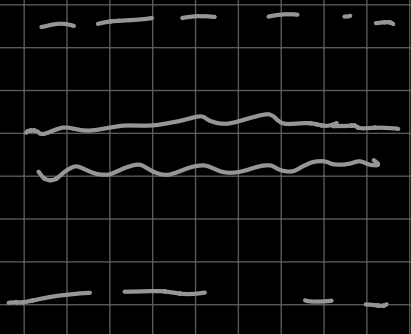
- Efficient in long term planning / complex planning
- No need iterative experts.

Drawbacks:

Difficult to train.

Real World Application by Pieter Abbeel:

- High variability / long tail
- Always changing
- Need to know what don't know

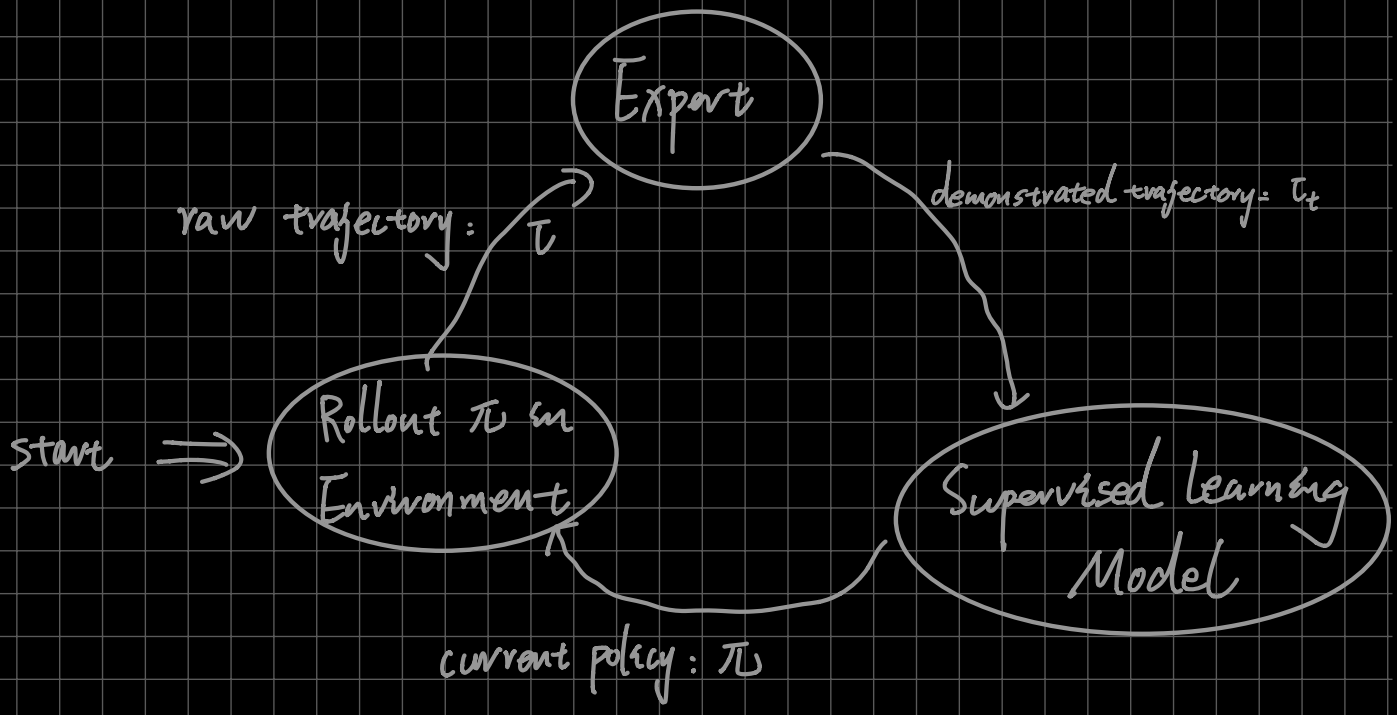


Research : 0 → 1 - 70% of benchmark then move on

Real World Application: 0.99 → 0.999 - 90% of benchmark is not enough.

Direct Policy Learning:

Process:



Algorithm:

Initial predictor: π_0

For $m = 1$:

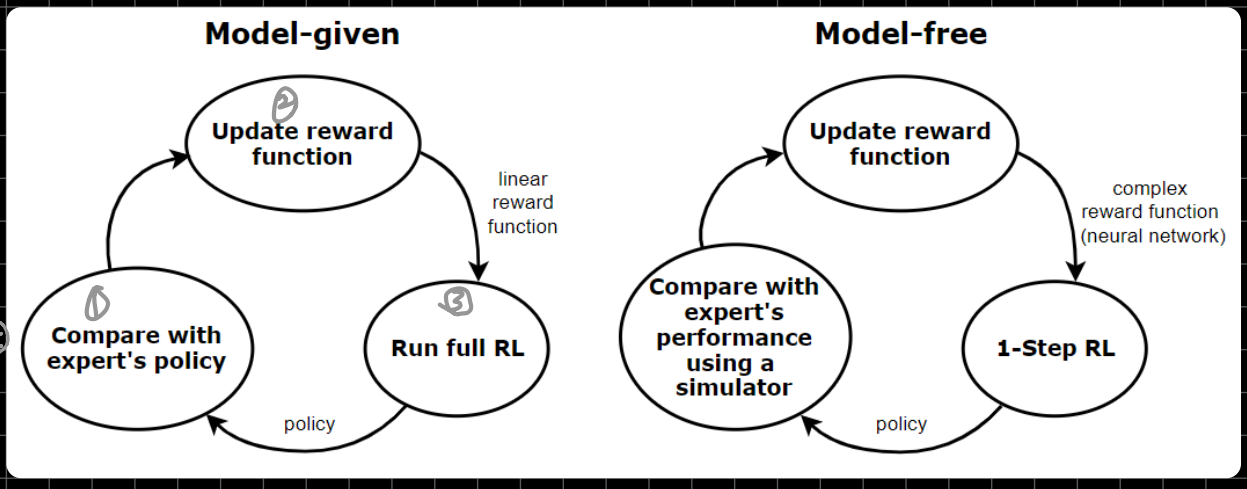
- Collect trajectories τ by rolling out π_{m-1}
- Estimate state distribution P_m using $s \in \tau$
- Collect interactive feedback $\{\pi^*(s) \mid s \in \tau\}$
- Data Aggregation (e.g. Dagger)
  - Train π_m on $P_1 \cup \ldots \cup P_m$
- Policy Aggregation (e.g. SEARN & SMILe)
  - Train π'_m on P_m
  - $\pi_m = \beta\pi'_m + (1-\beta)\pi_{m-1}$

Inversed Reinforcement Learning (IRL)

What:

Estimate the parameterized reward function represented by a neural network from expert's decisions to cause the expert's decisions

Classification:



Algorithm:

(i) Brief Explanation

- Collect expert demonstrations: $D = \{\tau_1, \tau_2, \ldots, \tau_m\}$
- In a loop:
  - Learn reward function: $r_\theta(s_t, a_t)$
  - Given the reward function r_θ, learn π policy using RL
  - Compare π with π^* (expert's policy)
  - STOP if π is satisfactory

(三) Detailed Algorithm:

Input: $\underbrace{E}_{\text{env}}$, $\underbrace{X}_{\text{state space}}$, $\underbrace{A}_{\text{action space}}$, $\underbrace{D = \{\tau_1, \tau_2 \cdots \tau_m\}}_{\text{expert demonstration}}$

Output: $\underbrace{R(x) = w^{*T} x}_{\text{assume linear function, easy to turn to non-linear by adding kernel function.}}$, $\underbrace{\pi}_{\text{given reward function, optimal policy in RL.}}$

Process:

1. $\bar{x}^* = \dfrac{\overbrace{(s_1^1 + s_2^1 \cdots + s_{n_1}^1)}^{\tau_1\text{'s states}} + (s_1^2 + s_2^2 \cdots + s_{n_2}^2) \cdots}{n_1 + n_2 \cdots + n_m}$

Expert Demonstration can be viewed as a sample of optimal policy. Thus, we can use MC to calculate the estimate expected $\bar{x}$. Then for optimal reward function $R(x) = w^* x$, we have: $w^* (\bar{x}^* - \bar{x}^\pi) \geq 0$. Thus, we can update the reward function $R(x)$ by:

($\bar{x}^*$: expected x from expert; $\bar{x}^\pi$: expected x from π)

$$\begin{cases} w^* = \arg\max_{w} \min_{\pi} w^T (\bar{x}^* - \bar{x}^\pi) \\ \text{s.t. } \|w\| \leq 1 \end{cases} \quad ②$$

2. $\pi \leftarrow$ random policy
3. for $t = 1, 2 \cdots T$ do
4. Rollout π to get trajectories: $\tau_1^\pi, \tau_2^\pi \cdots$

5. $\bar{X}_t^{\pi}$ ← ①

6. w^* ← ②

7. Given reward function $R(x) = w^* x$, get the optimal solution π.

8. end for