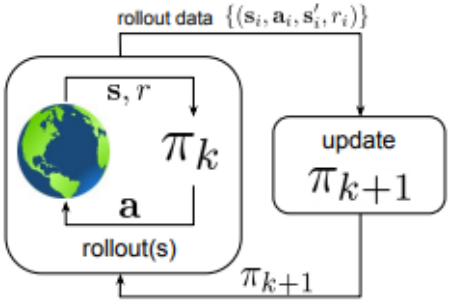
More on Batch / Offline Reinforcement Learning

With thanks to Emma Brunskill, Scott Fujimoto, Pieter Abeell, George Tucker, Sergey Levine, Bilal Piot, Yuxin Chen, Yuejie Chi

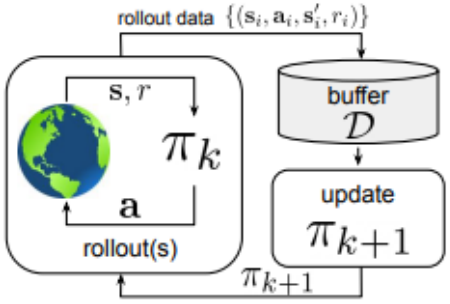
COMP579 Lecture 22, 2026

Recall: On-policy vs off-policy vs offline RL

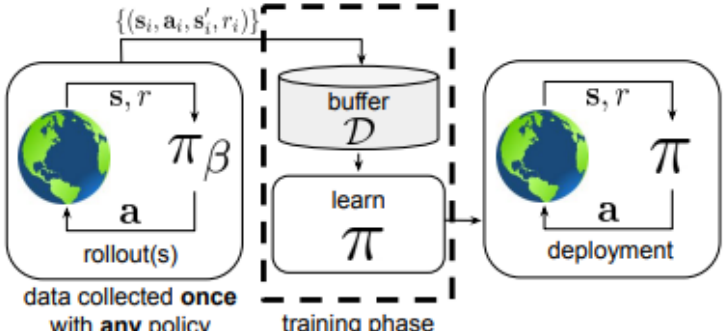
on-policy RL



off-policy RL



offline reinforcement learning



Formally:

$$\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$$

$$\mathbf{s} \sim d^{\pi_\beta}(\mathbf{s})$$
$$\mathbf{a} \sim \pi_\beta(\mathbf{a}|\mathbf{s})$$ ← generally **not** known
$$\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$$
$$r \leftarrow r(\mathbf{s}, \mathbf{a})$$

RL objective: $\max_\pi \sum_{t=0}^{T} E_{\mathbf{s}_t \sim d^\pi(\mathbf{s}), \mathbf{a}_t \sim \pi(\mathbf{a}|\mathbf{s})}[\gamma^t r(\mathbf{s}_t, \mathbf{a}_t)]$

Classes of algorithms

1. Behavior cloning (no rewards required) - just copy the actions recorded (like SFT!)
2. Learn a model, use it for model-based RL (LSTD, LSPI)
3. Pessimistic algorithms (require rewards)
4. Inverse RL (learn reward function from data, use it for RL agent)

Part 2: Model-based RL

- Take the data and fit a model (transition model, or reward model, or both)
- Use your favourite model-based RL algorithm to plan/learn! See past lectures!
- Today - a more well understood case (linear)

Model-based algorithms for least-squares regression

- Given value function approximation $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$
- And *experience* $\mathcal{D}$ consisting of $\langle \textit{state}, \textit{value} \rangle$ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

- Which parameters $\mathbf{w}$ give the *best fitting* value fn $\hat{v}(s, \mathbf{w})$?
- Least squares algorithms find parameter vector $\mathbf{w}$ minimising sum-squared error between $\hat{v}(s_t, \mathbf{w})$ and target values v_t^π,

$$\begin{aligned} LS(\mathbf{w}) &= \sum_{t=1}^{T} (v_t^\pi - \hat{v}(s_t, \mathbf{w}))^2 \\ &= \mathbb{E}_{\mathcal{D}} \left[(v^\pi - \hat{v}(s, \mathbf{w}))^2 \right] \end{aligned}$$

Least-squares regression

- At minimum of $LS(\mathbf{w})$, the expected update must be zero

$$\mathbb{E}_{\mathcal{D}}\left[\Delta \mathbf{w}\right] = 0$$

$$\alpha \sum_{t=1}^{T} \mathbf{x}(s_t)(v_t^{\pi} - \mathbf{x}(s_t)^{\top}\mathbf{w}) = 0$$

$$\sum_{t=1}^{T} \mathbf{x}(s_t) v_t^{\pi} = \sum_{t=1}^{T} \mathbf{x}(s_t)\mathbf{x}(s_t)^{\top}\mathbf{w}$$

$$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}(s_t)\mathbf{x}(s_t)^{\top}\right)^{-1} \sum_{t=1}^{T} \mathbf{x}(s_t) v_t^{\pi}$$

- For N features, direct solution time is $O(N^3)$
- Incremental solution time is $O(N^2)$ using Shermann-Morrison

Least-squares algorithms RL-style

- We do not know true values v_t^π
- In practice, our "training data" must use noisy or biased samples of v_t^π

  LSMC Least Squares Monte-Carlo uses return
  $v_t^\pi \approx G_t$

  LSTD Least Squares Temporal-Difference uses TD target
  $v_t^\pi \approx R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

  LSTD(λ) Least Squares TD(λ) uses λ-return
  $v_t^\pi \approx G_t^\lambda$

- In each case solve directly for fixed point of MC / TD / TD(λ)

LSMC and LSTD

LSMC

$$0 = \sum_{t=1}^{T} \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)$$

$$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}(S_t)\mathbf{x}(S_t)^\top \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t) G_t$$

LSTD

$$0 = \sum_{t=1}^{T} \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)$$

$$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}(S_t)(\mathbf{x}(S_t) - \gamma \mathbf{x}(S_{t+1}))^\top \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t) R_{t+1}$$

LSMC and LSTD

LSMC

$$0 = \sum_{t=1}^{T} \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)$$

$$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}(S_t)\mathbf{x}(S_t)^\top \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t) G_t$$

LSTD

$$0 = \sum_{t=1}^{T} \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)$$

$$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}(S_t)(\mathbf{x}(S_t) - \gamma \mathbf{x}(S_{t+1}))^\top \right)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t) R_{t+1}$$

Theoretical properties: Policy evaluation

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✓	✗
	LSTD	✓	✓	-
Off-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✗	✗
	LSTD	✓	✓	-

Theoretical properties: Control

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
LSPI	✓	(✓)	-

(✓) = chatters around near-optimal value function

Part 3: Pessimism in the face of uncertainty

- *Conservative* approach
- Assume that states or state-action pairs not visited are bad
- Use a penalty to avoid the new policy visiting them

Value iteration with lower confidence bounds

Pessimism in the face of uncertainty: penalize value estimate of those (s, a) pairs that were poorly visited [Jin et al., 2021, Rashidinejad et al., 2021]

Algorithm: value iteration w/ lower confidence bounds

- compute empirical estimate $\widehat{P}$ of P
- initialize $\widehat{Q} = 0$, and repeat

$$\widehat{Q}(s,a) \;\leftarrow\; \max\Big\{r(s,a) + \gamma\big\langle \widehat{P}(\cdot \,|\, s,a), \widehat{V}\big\rangle - \underbrace{b(s,a;\widehat{V})}_{\text{Bernstein-style confidence bound}},\; 0\Big\}$$

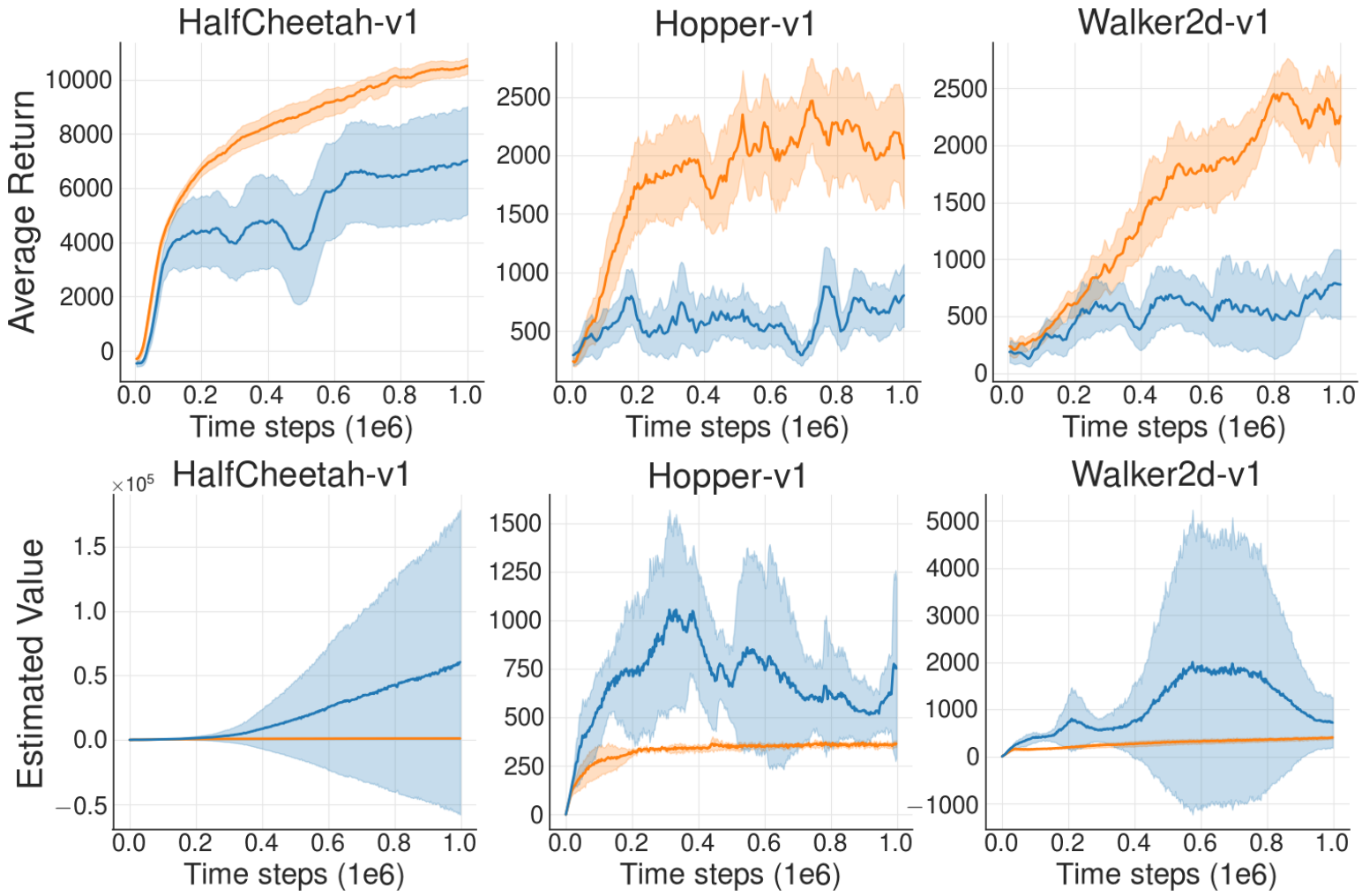
for all (s, a), where $\widehat{V}(s) = \max_a \widehat{Q}(s,a)$

Q-learning version exists as well

Alternative approach: Batch-Constrained RL

- Do NOT try to optimize value function/policy everywhere, if you only have a limited batch
- Instead, only look at policies π such that the batch contains pairs (s, a) that π visits

Motivation: Two versions of DDPG



Motivation: Two versions of DDPG

- Orange agent interacts with the environment in a standard RL loop: collect data, put it in replay buffer, train, repeat
- Blue agent is trained with data collected by the orange agent concurrently
- Even though the data and algorithm are the same, being off-policy throws the learning off!!!

Why? Extrapolation error

$$Q(s,a) \leftarrow r + \gamma Q(s',a')$$

$$Q(s,a) \leftarrow r + \gamma Q(s',a')$$

$$(s',a') \notin Dataset \rightarrow Q(s',a') = \textbf{bad}$$
$$\rightarrow Q(s,a) = \textbf{bad}$$

Batch-constrained RL

1. a$\sim\pi(s)$ such that $(s, a) \in Dataset$.
2. a$\sim\pi(s)$ such that $\left(s', \pi(s')\right) \in Dataset$.
3. a$\sim\pi(s)$ such that $Q(s, a)$ is maxed.

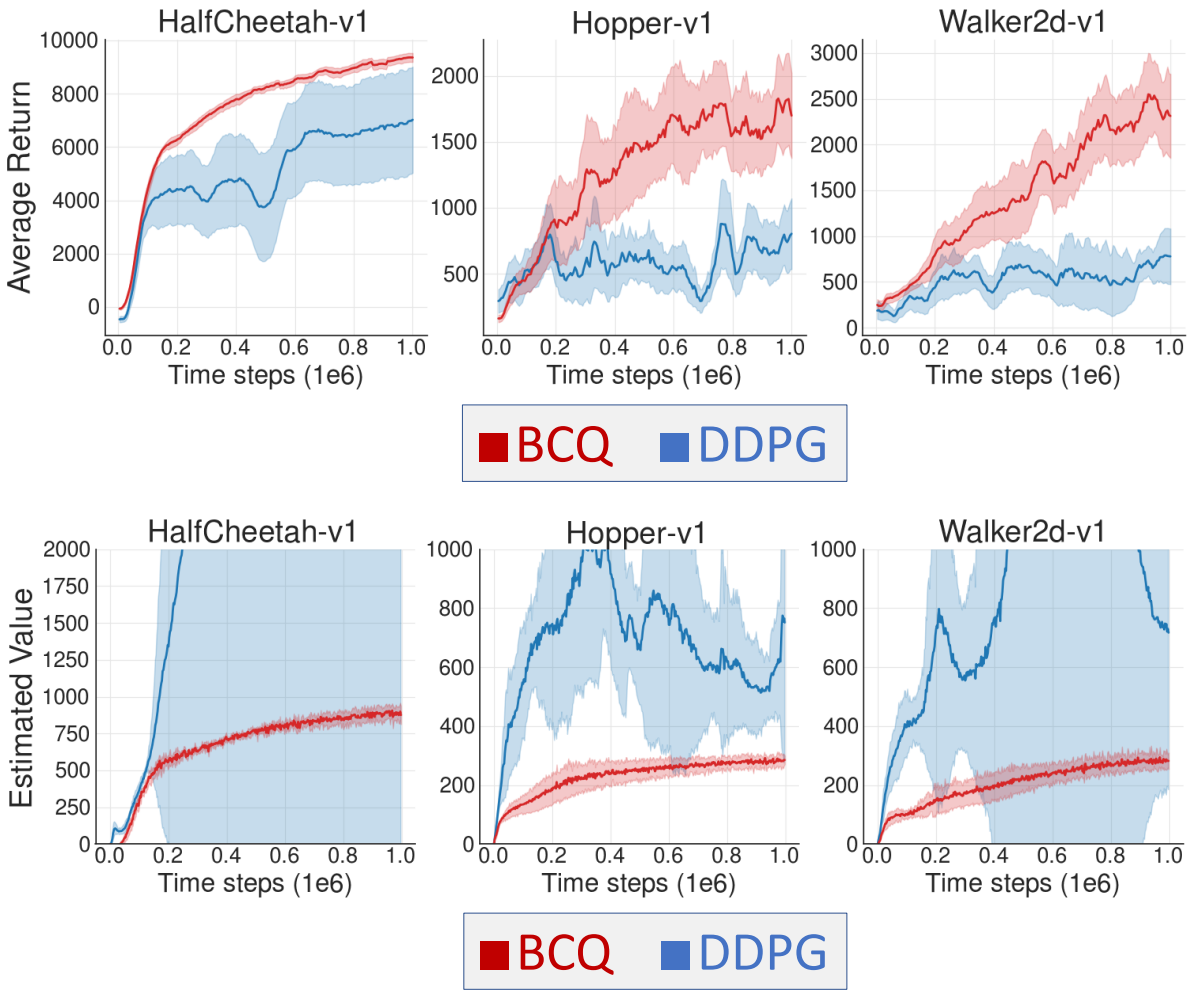
Batch-constrained Q-learning

First imitate dataset via generative model:
$G(a|s) \approx P_{Dataset}(a|s)$.

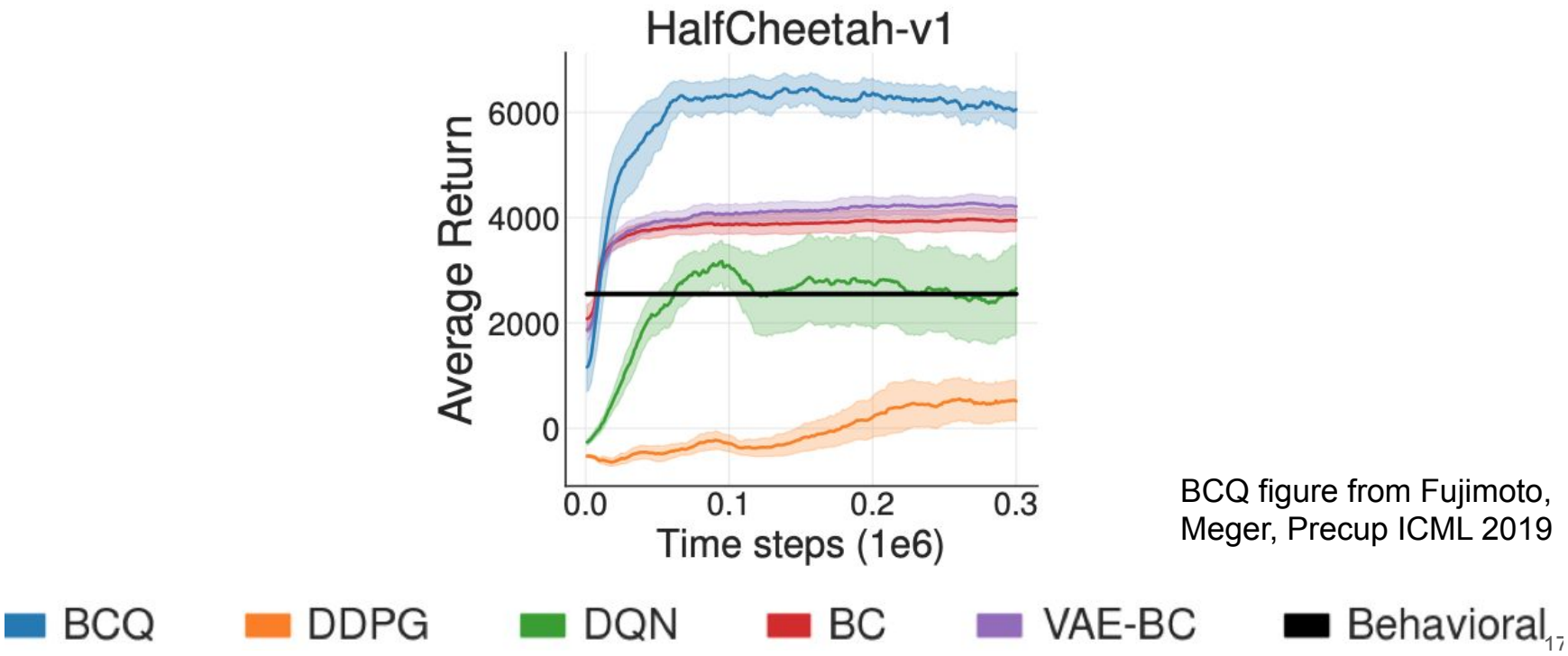
$\pi(s) = \text{argmax}_{a_i} Q\ (s, a_i)$, where $a_i \sim G$
(I.e. select the best action that is likely under the dataset)

(+ some additional deep RL magic)

Revisiting previous example

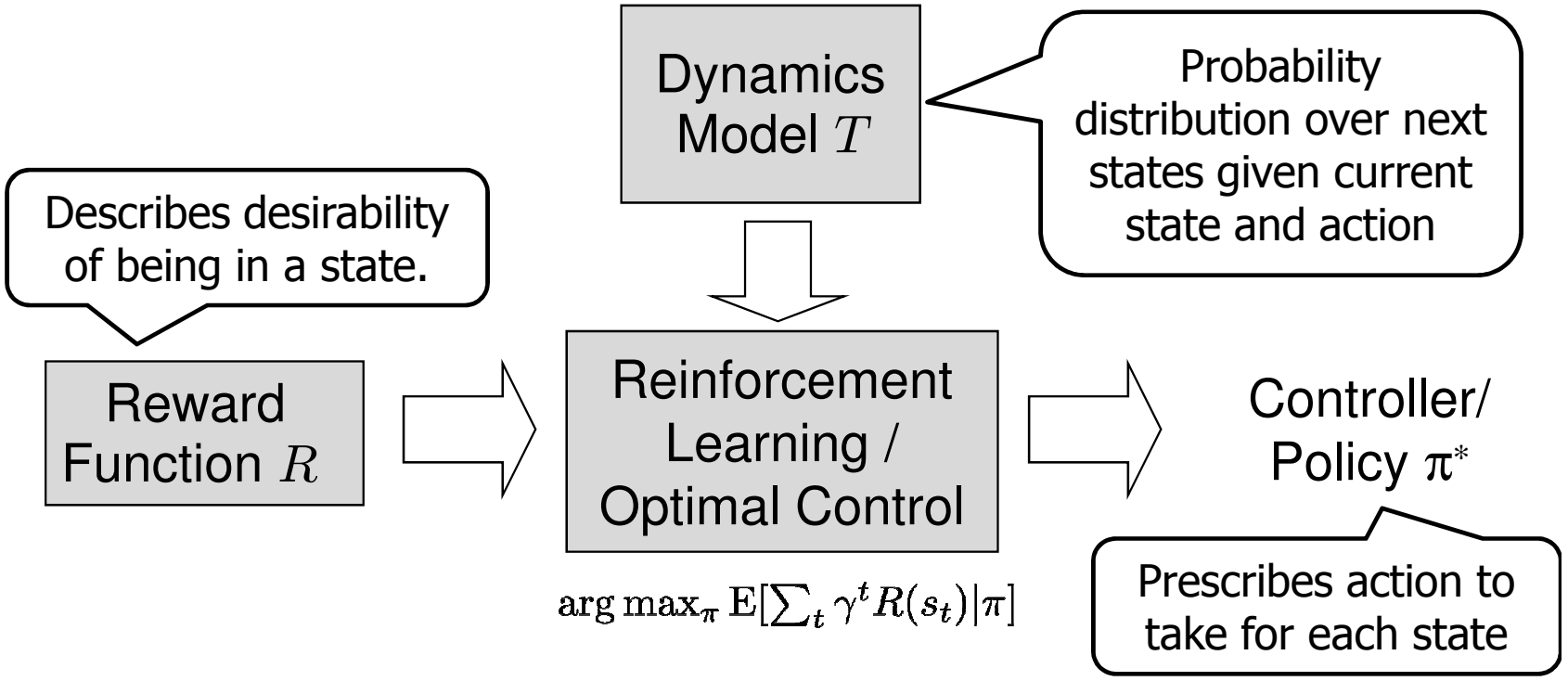


BCQ comparison



BCQ figure from Fujimoto, Meger, Precup ICML 2019

Part 4: Inverse RL



Inverse RL:

Given $\pi^*$and T, can we recover R?

More generally, given execution traces, can we recover R?

IRL problem formulation

- Input:
  - State space, action space
  - Transition model $P_{sa}(s_{t+1} \mid s_t, a_t)$
  - *No* reward function
  - Teacher's demonstration: $s_0, a_0, s_1, a_1, s_2, a_2, \ldots$
    (= trace of the teacher's policy π^*)
- Inverse RL:
  - Can we recover R ?
- Apprenticeship learning via inverse RL
  - Can we then use this R to find a good policy ?
- Behavioral cloning
  - Can we directly learn the teacher's policy using supervised learning?

IRL vs. Imitation learning: which one is better?

- It depends if the policy or the reward is more complicated!
- If the policy is simple learning it is easy supervised learning
- If the reward is simpler, IRL is better
- IRL also allows you to optimize if eg the transition function changes (eg autonomous driving, complex map navigation)

Main idea

- Assume the trajectories we have come from an optimal expert
- Therefore, the expert must have a reward function R^* such that:

$$E\left[\sum_{t=0}^{\infty}\gamma^t R^*(s_t)|\pi^*\right] \geq E\left[\sum_{t=0}^{\infty}\gamma^t R^*(s_t)|\pi\right], \forall\pi$$

  Now solve this type of equation for R^*!
- Problem: reward function ambiguity!
  - Many reward functions satisfy this equation, including eg $R^*(s) = 0 \forall s$
  - We only have some traces, not all of π^*
  - The expert has to be optimal

Feature-based reward functions

- Let $R(s) = w^\top \phi(s)$, where $w \in \Re^n$, and $\phi : S \to \Re^n$.

$$\begin{aligned} \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi] &= \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t w^\top \phi(s_t)|\pi] \\ &= w^\top \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi] \\ &= w^\top \mu(\pi) \end{aligned}$$

Expected cumulative discounted sum of feature values or "feature expectations"

- Subbing into $\mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R^*(s_t)|\pi^*] \geq \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R^*(s_t)|\pi] \quad \forall \pi$

  gives us:

$$\text{Find } w^* \text{ such that } w^{*\top} \mu(\pi^*) \geq w^{*\top} \mu(\pi) \quad \forall \pi$$

Notice that μ can be learned from data (called successor features)

Feature matching

- Abbeel and Ng (2004): for a policy π to perform almost as well as the optimal policy π^*, it suffices that the successor feature expectations match:

$$||\mu(\pi) - \mu(\pi^*)||_1 \leq \epsilon$$

  where $0 \leq \epsilon < 1$
- If so, $\forall w$ with norm less than 1:

$$|w^T \mu(\pi) - w^T \mu(\pi^*)| \leq \epsilon$$

- Optimization problem can be solved in complexity that depends on $1/\epsilon^2$ and on the number of features in the reward function, NOT depending on complexity of π^* or the number of states
- Approximation property even if the true reward cannot be represented through linear combination of features

Apprenticeship learning (Abbeel and Ng, 2004)

- Assume $R_w(s) = w^\top \phi(s)$ for a feature map $\phi : S \rightarrow \Re^n$.
- Initialize: pick some controller π_0.
- Iterate for i = 1, 2, … :
  - **"Guess" the reward function**:

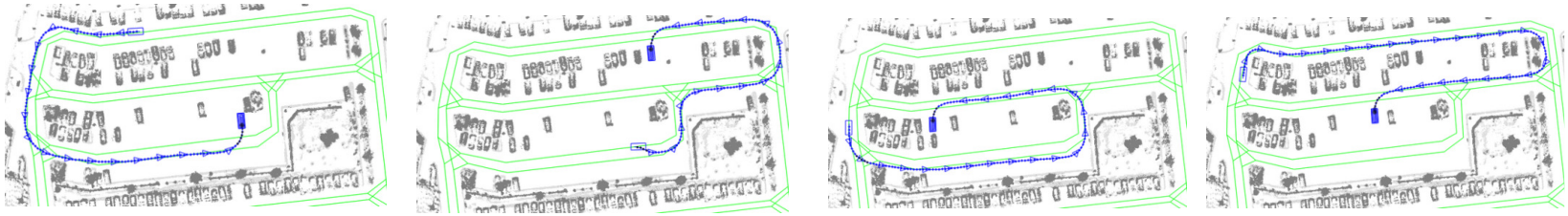
    Find a reward function such that the teacher maximally outperforms all previously found controllers.

    $$\max_{\gamma, w: \|w\|_2 \leq 1} \gamma$$
    $$s.t. \quad w^\top \mu(\pi^*) \geq w^\top \mu(\pi) + \gamma \quad \forall \pi \in \{\pi_0, \pi_1, \ldots, \pi_{i-1}\}$$

  - **Find optimal control policy** π_i for the current guess of the reward function R_w.
  - If $\gamma \leq \varepsilon/2$ exit the algorithm.

Example: Learning to park (Abbeel and Ng, 2004)

- Demonstrate parking lot navigation on "train parking lots."



- Run our apprenticeship learning algorithm to find the reward function.
- Receive "test parking lot" map + starting point and destination.
- Find the trajectory that maximizes the *learned reward function* for navigating the test parking lot.

Learned policies reflect the data!

- Training on nice data (left) vs sloppy data (middle) vs reverse as well (right)

Helping with disambiguation: Maximum-entropy IRL

- IRL is essentially trying to match the feature distribution in expert demonstrations
- But we need to add further constraints to make this well conditioned (previously discussed work uses a margin)
- Principle of maximum entropy: Given prior information about a distribution, the best approximation is the distribution matching the data with the largest uncertainty
- Because this makes the fewest assumptions about the true distribution!
- Another interpretation: we want to avoid overfitting the data

Helping with disambiguation: Maximum-entropy IRL (Ziebart et al, 2008)

- Main idea: match feature distributions but otherwise maximize the entropy of trajectory distributions
- Recall the definition of entropy: $H(P) = -\sum_x P(x) \log P(x)$
- Recall the probability of a trajectory $\tau = s_0 a_0 \ldots s_T$:

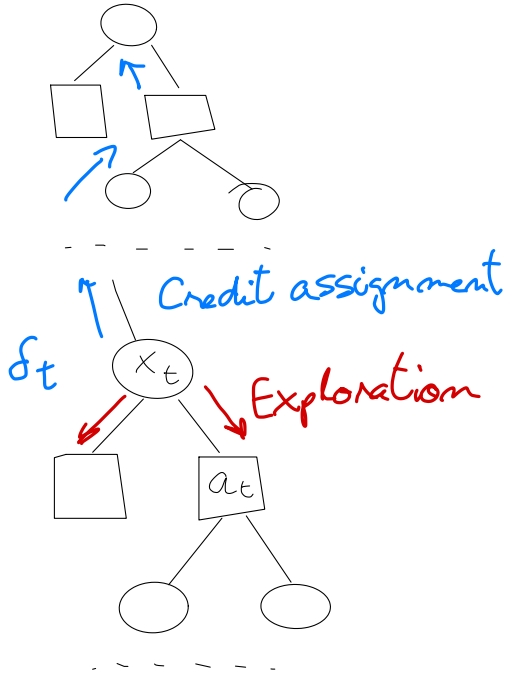
$$P^\pi(\tau) = p_0(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t) P(s_{t+1}|s_t, a_t)$$

- We want to maximize the entropy of P^π while matching feature expectations:

$$\max_\pi H(P^\pi) \text{ s.t. } \mu(\pi) = \mu(\pi^*)$$

- Can be re-written as: $\min_\pi E_{s,a\sim\mu^\pi}[\log(\pi(a|s)]$
- Can be solved using a Lagrangian and gradients (various flavors exist)

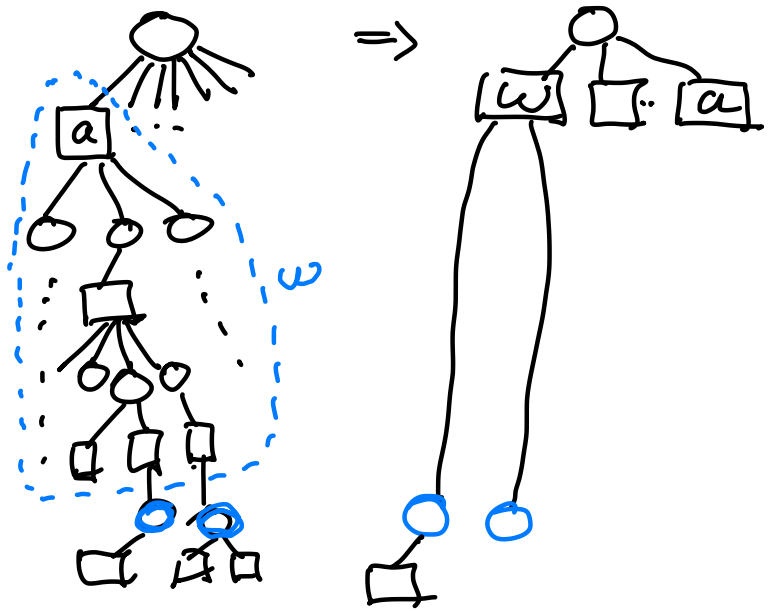
Going back to the big picture



Temporal abstraction:

- Reduce tree width - helps exploration!
- Reduce tree depth - helps make planning/reasoning faster
- Generalize between different branches of the tree - improves learning

Options Intuition



- Package a whole sub-tree as an option ω
- Jumps are to the state *at the end* of the sub-tree
- Primitive actions are a special case (one-step tree)
- Two components: (sub)policy and model

Both abstraction and generalization!