

Do RL without any access to the environment. This matches supervised learning more closely where we're provided with a dataset of experiences and the RL algorithm must learn a policy.

# Offline Reinforcement Learning

## CS 285

Instructor: Aviral Kumar  
UC Berkeley



Berkeley  
UNIVERSITY OF CALIFORNIA

# What have we covered so far?

$$\max_{\pi} \sum_{t=1}^{\infty} \mathbb{E}_{s_t, a_t \sim \pi} [\gamma^t r(s_t, a_t)]$$

- Exploration:
  - Strategies to discover high-reward states, diverse skills, etc.
  - How hard is exploration?

$$\#\text{Samples} \geq \Omega \left( \frac{|\mathcal{S}||\mathcal{A}|}{(1-\gamma)^3} \log \frac{|\mathcal{S}||\mathcal{A}|}{\delta} \right)$$

Super Large

How many states to visit in the “best” case  
to learn an optimal Q-function

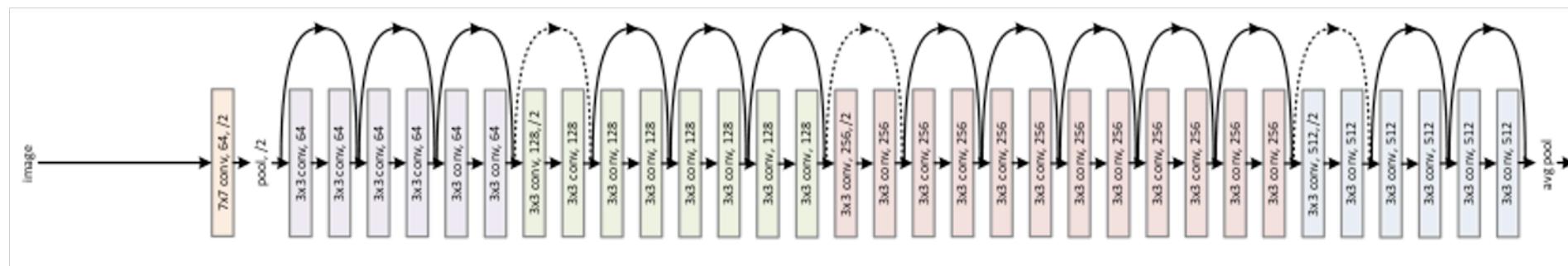
- Even if we are ready to collect so many samples, it may be dangerous in practice: imagine a random policy on an autonomous car or a robot!

# Can we apply standard RL in the real-world?

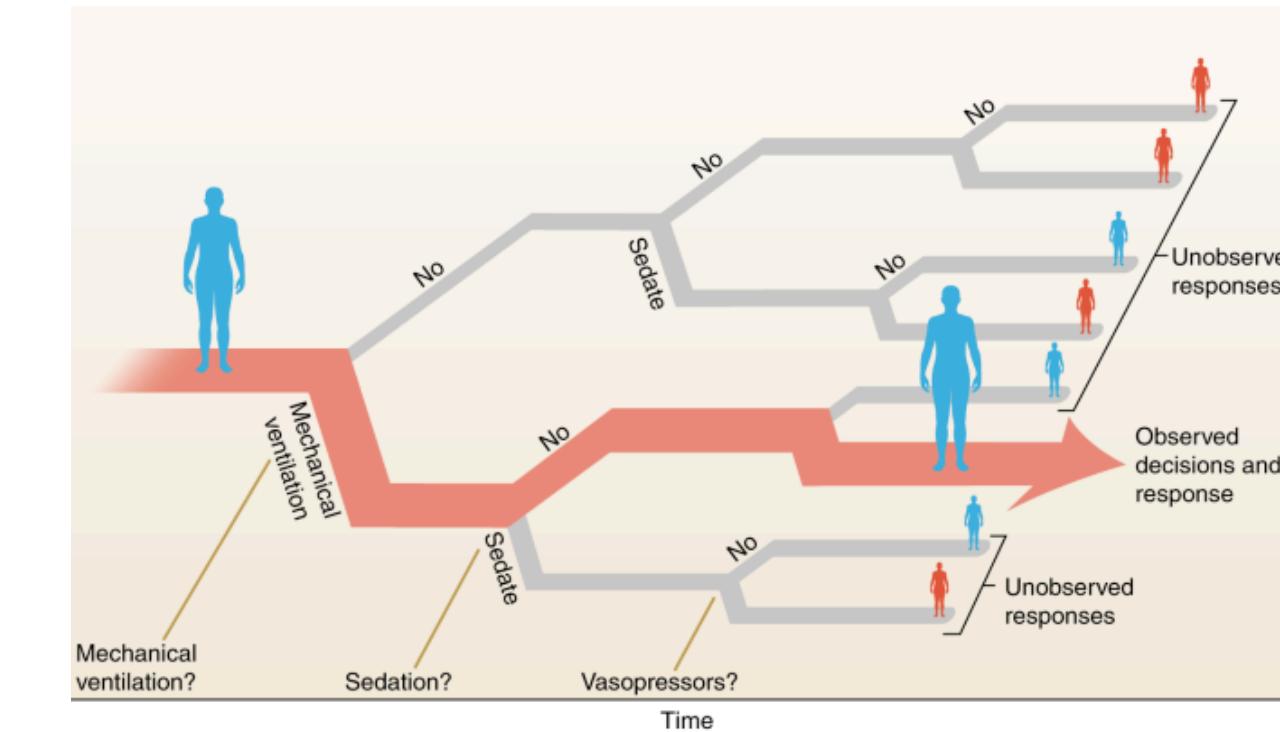
- RL is fundamentally an “active” learning paradigm: the agent needs to collect its own dataset to learn meaningful policies
- This can be unsafe or expensive in real world problems!



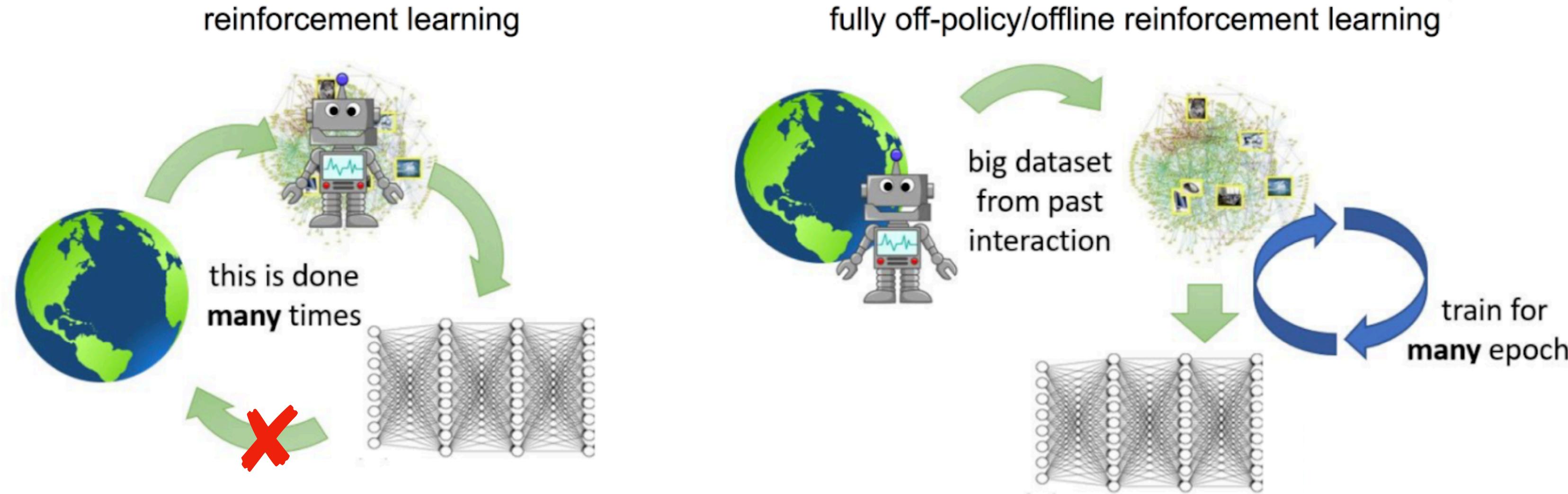
Generalization?



Iterated data collection can cause poor generalization!



# Offline (Batch) Reinforcement Learning

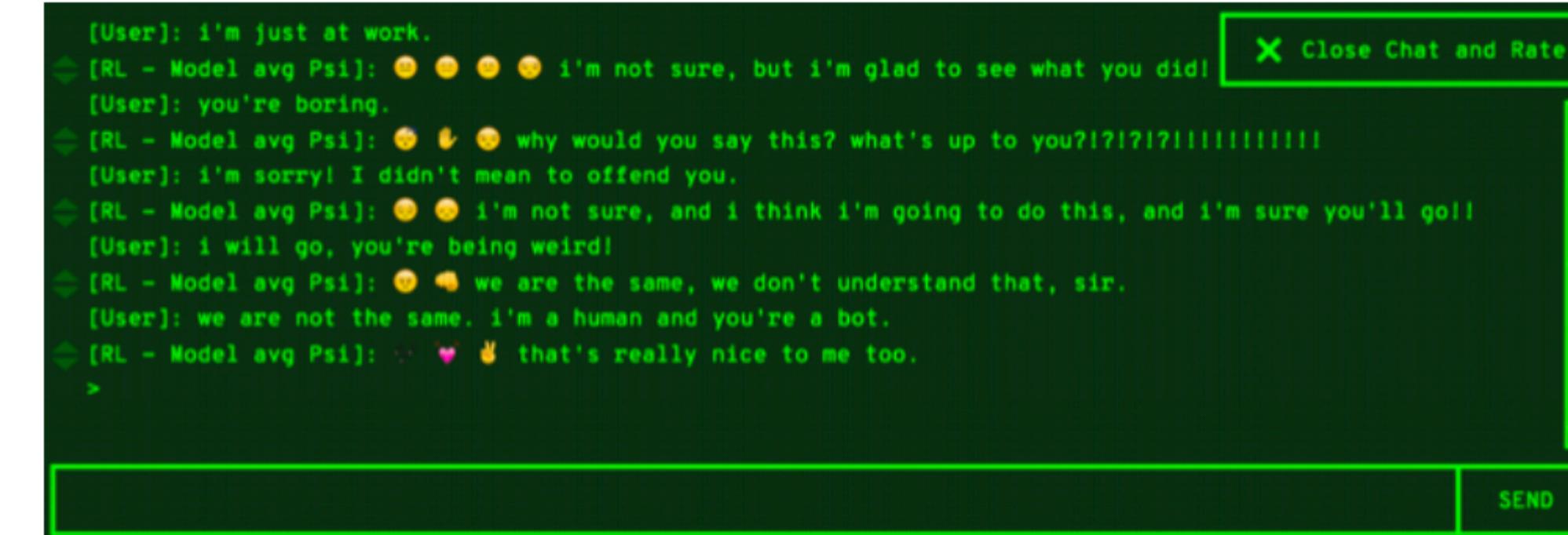
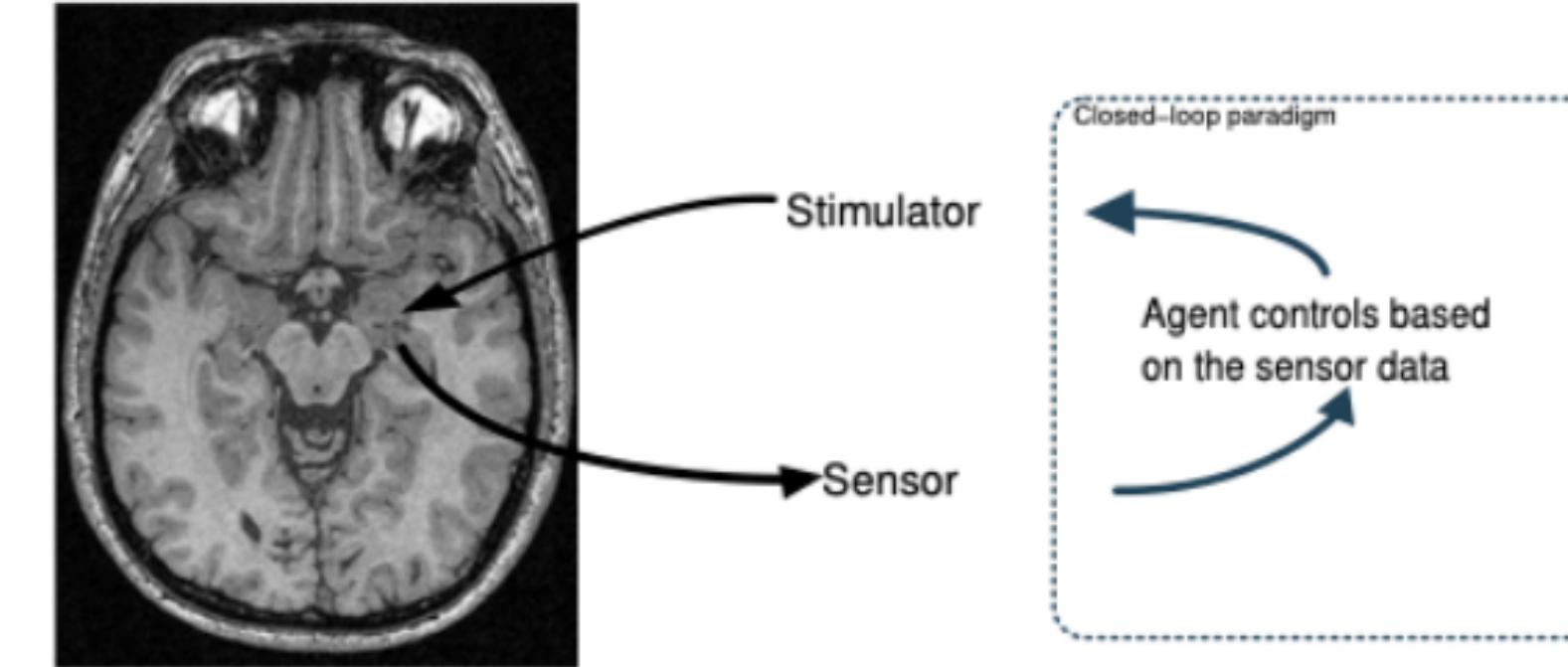


## Learn from a previously collected static dataset

Why is offline RL promising?

- Large static datasets of meaningful behaviours already exist
- Large datasets at the core of successes in Vision and NLP

# Applications of Offline RL



Kalashnikov et al. QT-Opt: Scalable Deep RL for Vision-Based Robotic Manipulation. CoRL 2018.  
Jaques et al. Way Off-Policy Batch Reinforcement Learning for Dialog. EMNLP 2020.  
Guez et al. Adaptive Treatment of Epilepsy via Batch-Mode Reinforcement Learning. AAAI 2008.  
Kendall et al. Learning to Drive in a Day. ICRA 2019.  
Levine, Kumar, Tucker, Fu. *Offline RL Tutorial and Perspectives on Open Problems*. arXiv 2020.

# How good can offline RL perform?

## Supervised Learning

Can do as good as the dataset!



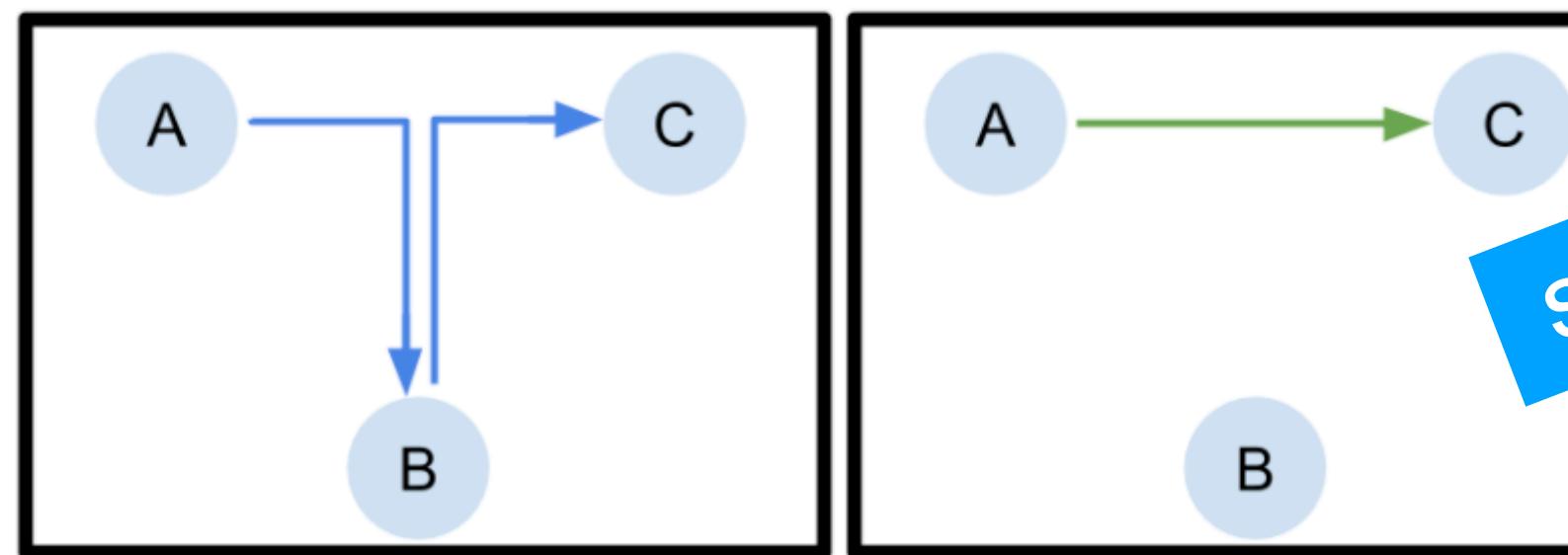
imagine the dataset has mislabeled this picture of a cat as a dog



for a new image of a cat, can we predict this to be a cat? no we can't, because our dataset said it was a dog

## Offline Reinforcement Learning

Can do better than the dataset!



we learn to go from A to C directly, despite the dataset going from A to B to C.

Can show that Q-learning recovers optimal policy from random data.

# Formalism and Notation

$$\max_{\pi} \sum_{t=1}^{\infty} \mathbb{E}_{s_t, a_t \sim \pi} [\gamma^t r(s_t, a_t)]$$

- Dataset construction:
  - Several trajectories:

$$\mathcal{D} = \{\tau_1, \dots, \tau_N\}, \quad \tau_i = \{s_i^t, a_i^t, r_i^t, s_i'^t\}_{t=1}^H$$

Reward known

- Approximate “distribution” of states in the dataset:  $\mathcal{D}(s)$
- Approximate distribution of actions at a given state in the dataset:  $\mathcal{D}(a|s)$
- Will use notation for the behavior policy,  $\pi_{\beta}(a|s) = \mathcal{D}(a|s)$
- Standard RL notation from before:  $Q^{\pi}(s, a), V^{\pi}(s), d^{\pi}(s)$ , etc.

**Part 1:** Classic Offline RL Algorithms  
and Challenges  
With Offline RL

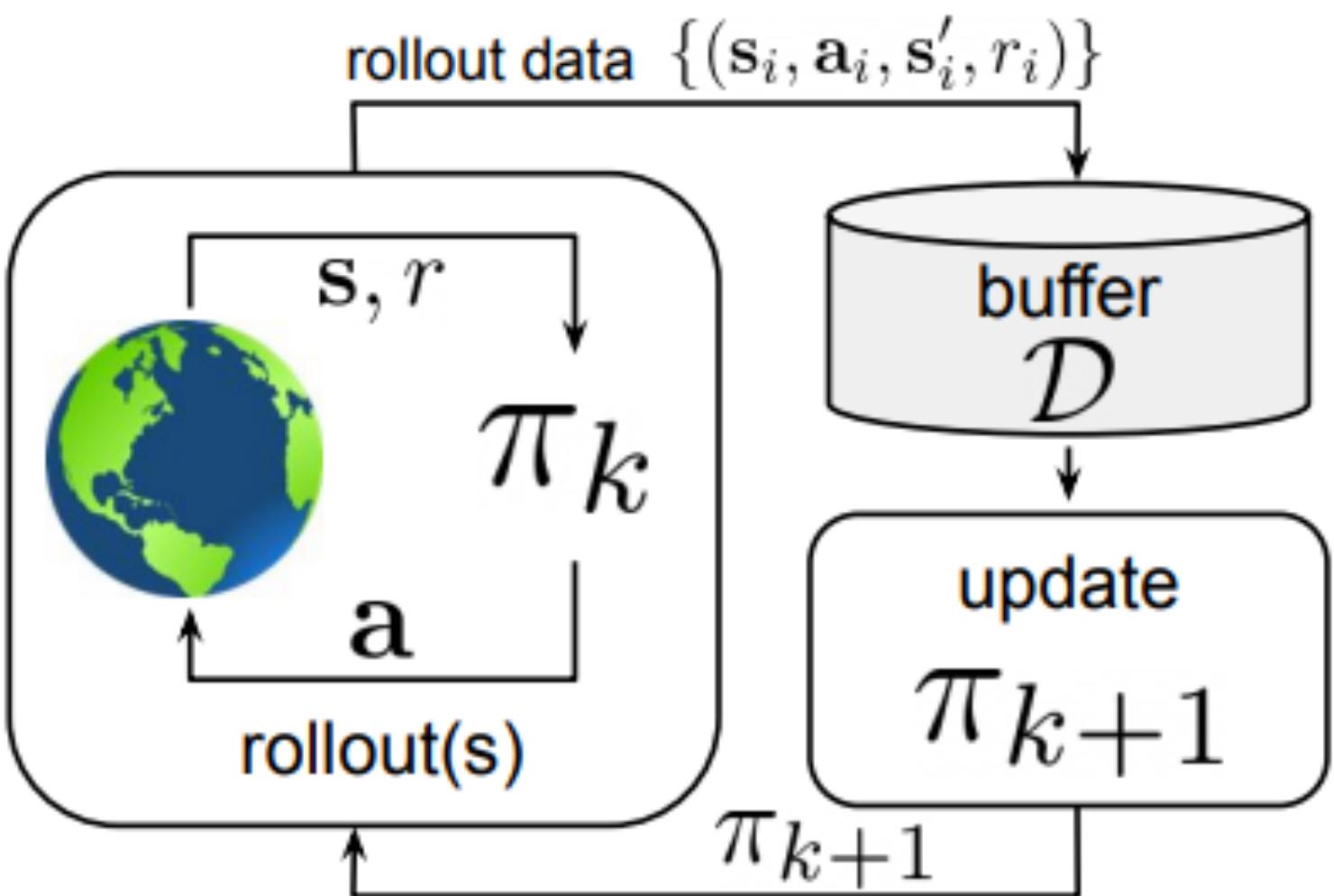
**Part 2:** Deep RL Algorithms to  
Address These Challenges

**Part 3:** Related Problems,  
Evaluation Protocols, Applications

# **Part 1: Classic Algorithms and Challenges With Offline RL**

# A Generic Off-Policy RL Algorithm

DQN and Actor-critic algorithms both follow a similar skeleton, but with different design choices.



1. Collect data using the current policy
2. Store this data in a replay buffer
3. Use replay buffer to make updates on the policy and the Q-function
4. Continue from step 1.

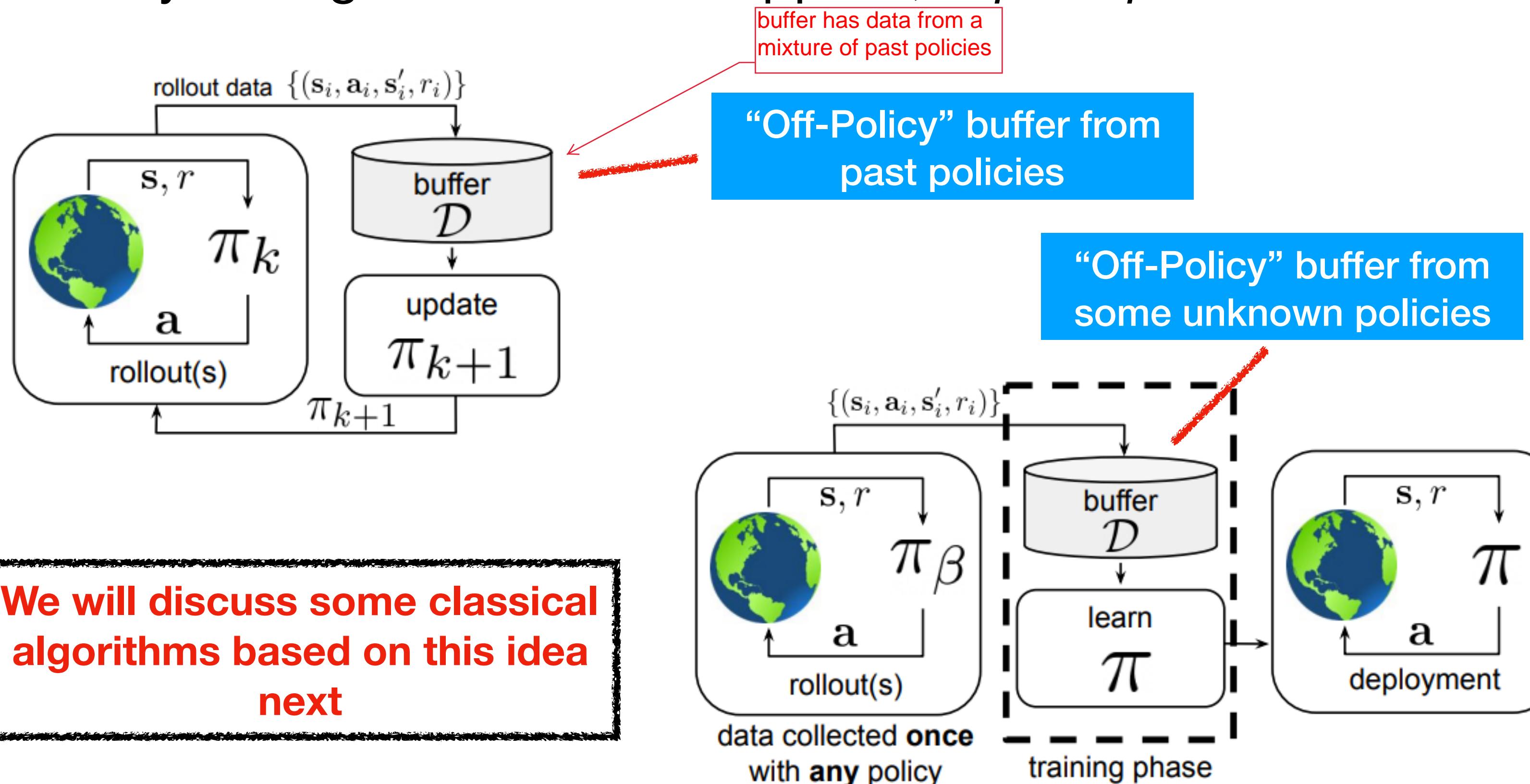
$$\min_Q \sum_{i=1}^N \left( Q(s_i, a_i) - \left[ r(s_i, a_i) + \gamma \max_{a'} \bar{Q}(s'_i, a') \right] \right)^2$$

## Actor-critic Algorithm:

1. Learn  $\hat{Q}^\pi$
2. Optimize policy w.r.t.  $\hat{Q}^\pi$  :  $\pi \leftarrow \arg \max_{\pi} \mathbb{E}_{\pi}[\hat{Q}^\pi]$

# Can such off-policy RL algorithms be used?

Off-Policy RL Algorithms can be applied, *in principle*



**Actor-critic Algorithm:**

1. Learn  $\hat{Q}^\pi$  using offline data  $\mathcal{D}$ .
2. Optimize policy w.r.t.  $\hat{Q}^\pi$ :  $\pi \leftarrow \arg \max_{\pi} \mathbb{E}_{\pi}[\hat{Q}^\pi]$

Lagoudakis, Parr. Least Squares Policy Iteration. JMLR 2003.  
Ernest et al. Tree-Based Batch Mode Reinforcement Learning. JMLR 2005  
Gordon G. J. Stable Function Approximation in Dynamic Programming. ICML 1995, and many more...

# Classic Batch Q-Learning Algorithms

Two general approaches to batch RL  
1) Q-learning algos  
2) Importance sampling algos

## Algorithm 1 Fitted Q-Iteration (FQI)

```
1: Initialize Q-network  $Q_\theta$ , buffer  $\mu$ .  
2: for fitting iteration  $k$  in  $\{1, \dots, N\}$  do  
3:   Compute  $Q_\theta(s, a)$  and target values  
       $y_k(s, a) = r + \gamma \max_{a'} Q_{k-1}(s', a')$   
      on  $\{(s, a)\} \sim \mu$  for training  
4:   Minimize TD error for  $Q_\theta$  via  $t = 1, \dots, T$  gradient descent updates,  
       $\min_\theta (Q_\theta(s, a) - y_k)^2$   
5: end for
```

1. Compute target values using the current Q-function
2. Train Q-function by minimizing TD error with respect to target values from Step 1.

### Linear Q-functions

$$Q(s, a) = w^T \phi(s, a)$$

$$w^T \phi(s, a) \approx R + \gamma \max_{a'} w^T \phi(s', a')$$

Can be solved in many ways:

- (1) find fixed point of the above equation
- (2) minimise the gap between the two sides of the equation

### Least Squares Temporal Difference Q-Learning (LSTD-Q)

Lagoudakis, Parr. Least Squares Policy Iteration. JMLR 2003.

Ernest et al. Tree-Based Batch Mode Reinforcement Learning. JMLR 2005

Riedmiller. Neural Fitted Q-Iteration. ECML 2005.

Gordon G. J. Stable Function Approximation in Dynamic Programming. ICML 1995

Antos, Szepesvari, Munos. Fitted Q-Iteration in Continuous Action-Space MDPs. NeurIPS 2007.

# Classic Batch RL Algorithms based on IS

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\beta(\tau)} \left[ \frac{\pi_\theta(\tau)}{\pi_\beta(\tau)} \sum_{t=0}^H \gamma^t r(s, a) \right]$$

$$= \mathbb{E}_{\tau \sim \pi_\beta(\tau)} \left[ \left( \prod_{t=0}^H \frac{\pi_\theta(a_t | s_t)}{\pi_\beta(a_t | s_t)} \right) \sum_{t=0}^H \gamma^t r(s, a) \right] \approx \sum_{i=1}^n w_H^i \sum_{t=0}^H \gamma^t r_t^i,$$

you can estimate the reward for a particular policy by using importance sampling. You sample trajectories for  $\pi_\beta$  and then correct for the returns that you get using the density ratio  $\pi_\theta / \pi_\beta$ .

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\beta(\tau)} \left[ \sum_{t=0}^H \left( \prod_{t'=0}^t \frac{\pi_\theta(a_t | s_t)}{\pi_\beta(a_t | s_t)} \right) \gamma^t r(s, a) \right] \approx \frac{1}{n} \sum_{i=1}^n \sum_{t=0}^H w_t^i \gamma^t r_t^i.$$

take advantage of temporal structure to reduce variance

these methods aim to estimate the return of a policy  $\pi_\theta$  using importance sampling and then they optimize the return of policy  $\pi_\theta$  to obtain a better policy, using for instance, policy gradient

$$J(\pi_\theta) \approx \sum_{i=1}^n \sum_{t=0}^H \gamma^t \left( w_t^i \left( r_t^i - \hat{Q}^{\pi_\theta}(s_t, a_t) \right) - w_{t-1}^i \mathbb{E}_{a \sim \pi_\theta(a|s_t)} [\hat{Q}^{\pi_\theta}(s_t, a)] \right).$$

**Doubly-robust**

High-confidence bounds on the return estimate  
Variance reduction techniques

doubly robust methods still do importance sampling but in addition they estimate a Q-function. It uses a combination of both of these to obtain a low variance and unbiased estimate of the return

Precup. Eligibility Traces for Off-Policy Policy Evaluation. CSD Faculty Publication Series, 2000.  
Precup, Sutton, Dasgupta. Off-Policy TD Learning with Function Approximation. ICML 2001.  
Peshkin and Shelton. Learning from Scarce Experience. 2002.

Thomas, Theocharous, Ghavamzadeh. High Confidence Off-Policy Evaluation. AAAI 2015.  
Thomas, Theocharous, Ghavamzadeh. High Confidence Off-Policy Improvement. ICML 2015.  
Thomas, Brunskill. Magical Policy Search: Data Efficient RL with Guarantees of Global Optimality. EWRL 2016.  
Jiang and Li. Doubly-Robust Off-Policy Value Estimation for Reinforcement Learning. ICML 2016.

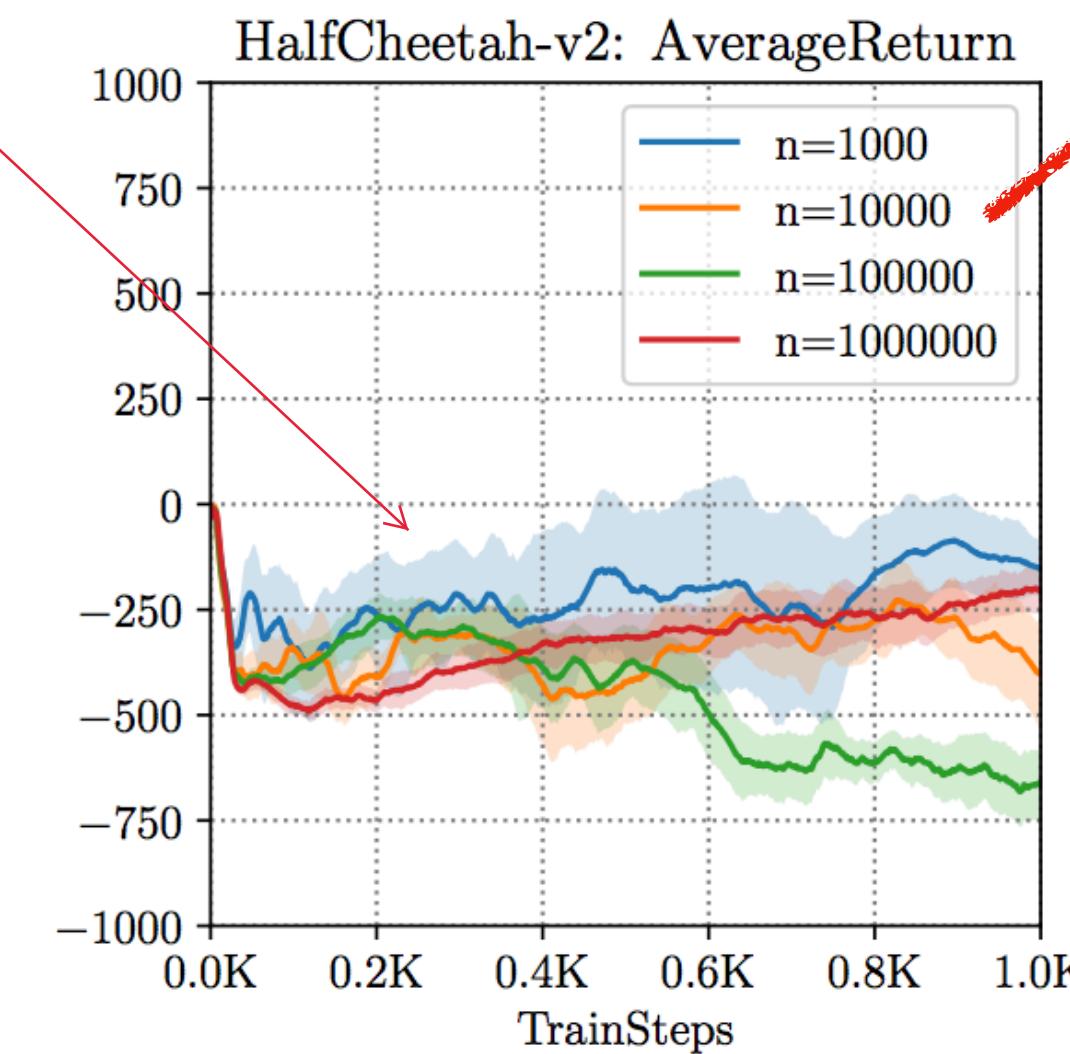
now we look at deep RL  
methods in an offline  
setting

# Modern Offline RL: A Simple Experiment

Collect expert data and run actor-critic algorithms on this data

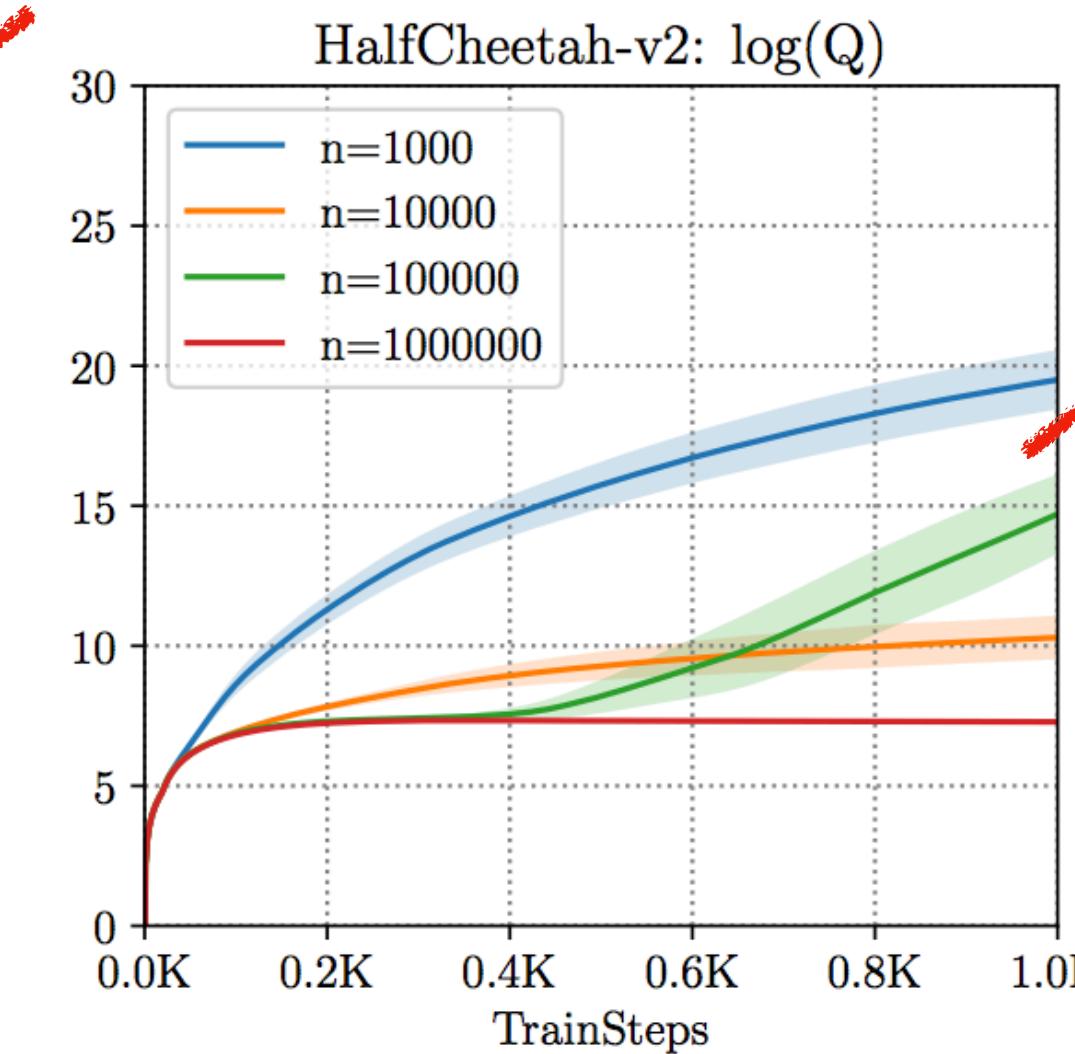
In supervised learning we think about overfitting and underfitting tradeoff.  
To evaluate that in the offline RL setting, we trained multiple models with different amounts of data. If overfitting was really an issue for offline RL, we would observe that more data leads to higher performance. Empirically we see that this isn't the case at all. In fact, 1000 samples performed the best.

very low rewards!



how well it does

Performance doesn't  
improve with more data



how well it thinks it does

Learning diverges

the Q-value is the algorithm's estimate of how good the policy is. Notice how much larger it is than the actual returns (hint: it's log scale). So the learning diverges because the algo thinks it's doing so much better than it actually is

“Policy unlearning”

Not a classical overfitting issue!

**So, why do RL algorithms fail, even though imitation learning would work in this setting (e.g., in Lecture 2)?**

# Let's see how the Q-function is updated

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

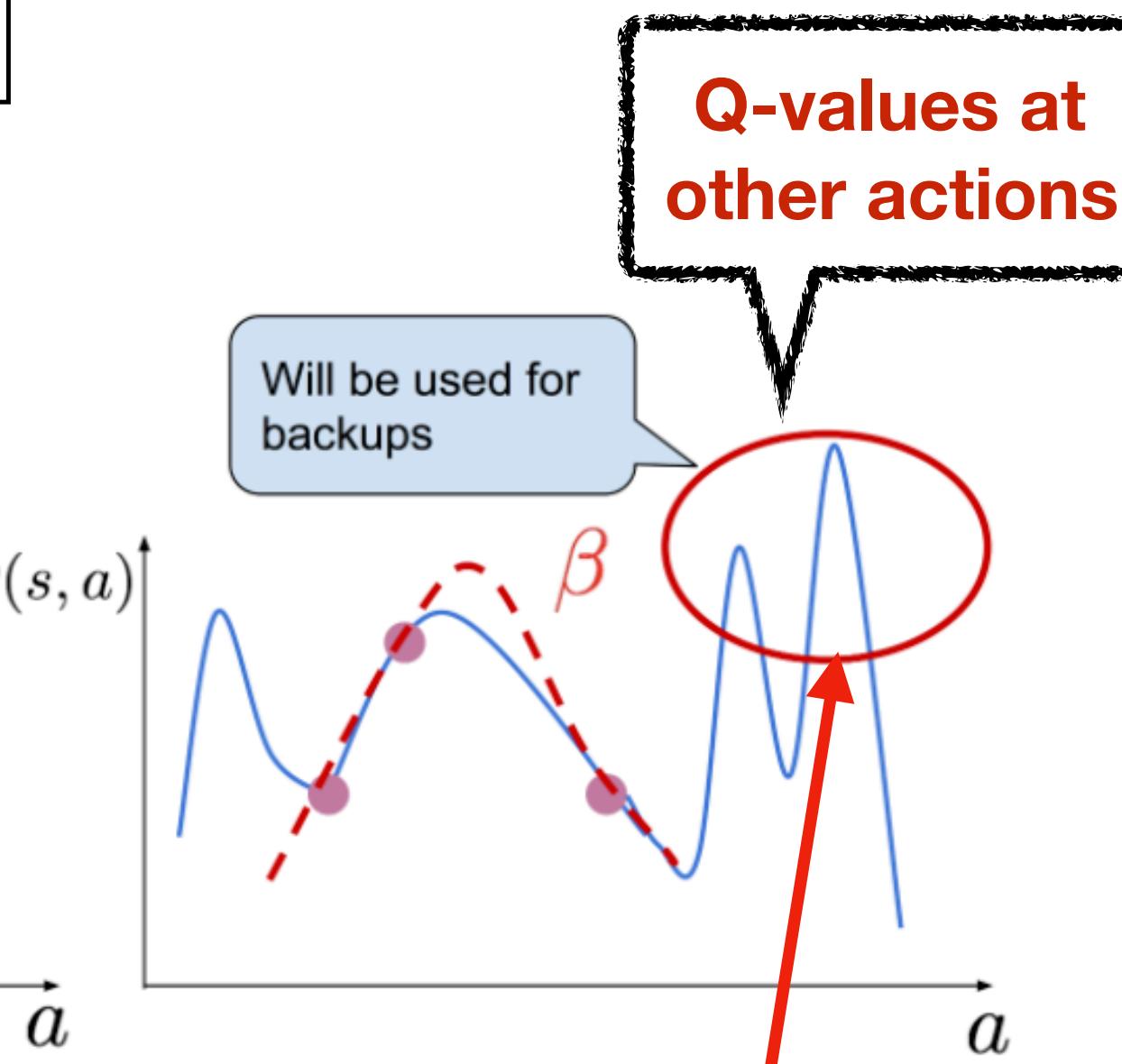
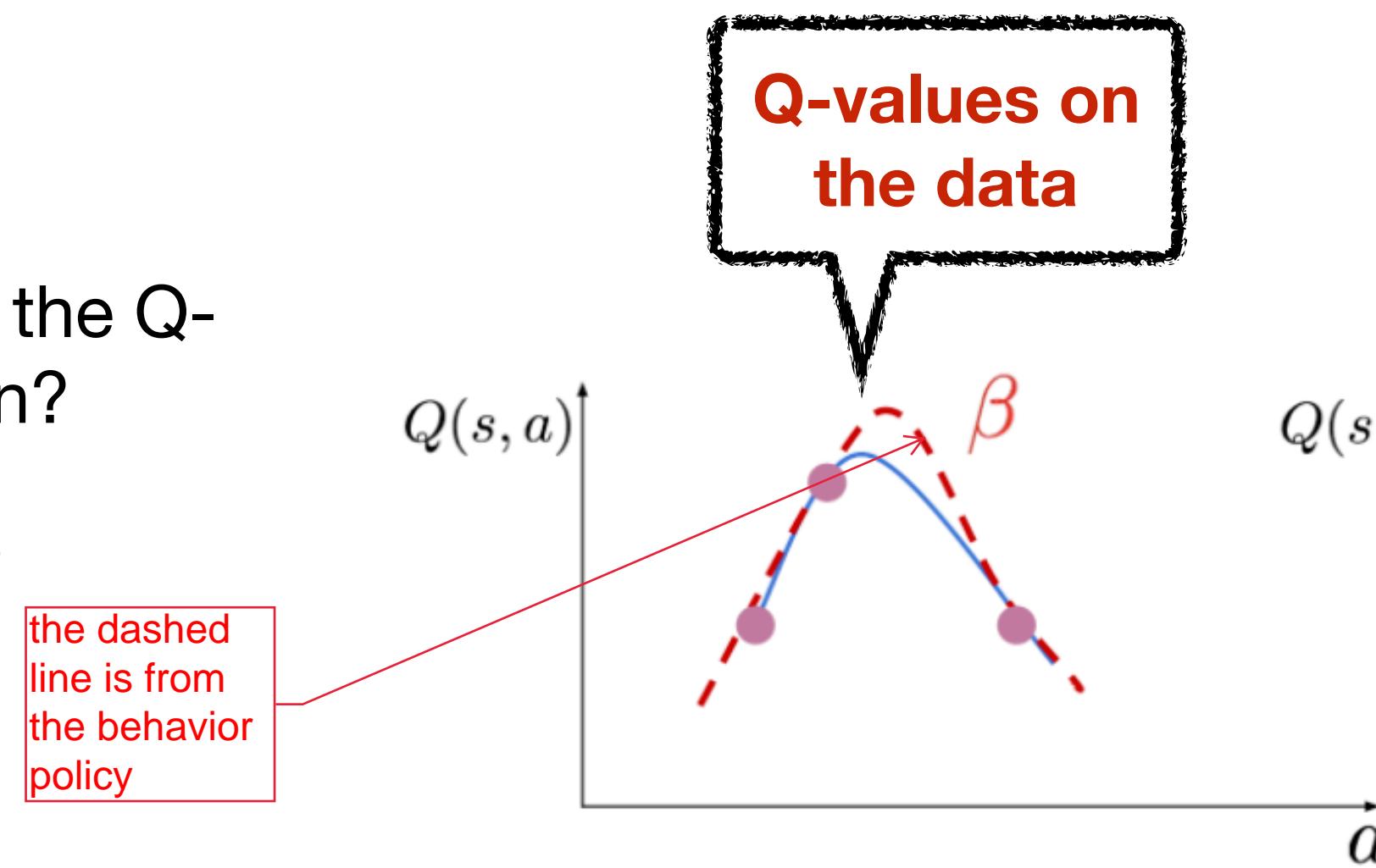
what's happening is that we are querying the Q-function at unseen actions  $a'$ . This leads to overestimation in  $Q(s', a')$  and we're using this for training. In other words these are out-of-distribution actions. Thus we have erroneously large target functions that we're learning on and trying to reach. So we're trying to get our Q-function to regress to erroneous values.

$$\mathbb{E}_{s, a, s' \sim \mathcal{D}} \left[ (Q(s, a) - (r(s, a) + \gamma \max_{a'} Q(s', a')))^2 \right]$$

Which actions does the Q-function train on?

(s,a) pairs from the data, and thus the behavior policy!

$$s, a \sim \mathcal{D}$$



for actions that are unseen in the data, the Q-values can be really huge. When you do a max over these possible next actions, it will select overly optimistic actions

Where does the action  $a'$  for the target value come from?

$a'$  comes from the target value where we take the  $\max(Q(s', a'))$

$$\max_{a'} Q(s', a')$$

**Q-learning queries values at unseen action targets, which are never trained during training**

# Why are erroneous backups a big deal?

- This phenomenon also happens in online RL settings, where the Q-function is erroneously optimistic

this is why we developed double DQN

- But Boltzmann or epsilon-greedy exploration on this overoptimistic Q-function (generally) leads to “error correction”

$$\pi_{\text{explore}}(a|s) \propto \exp(Q(s, a))$$

Error correction is **not** necessarily guaranteed with online data collection when using deep neural nets, but mostly works fine in practice (trick: use replay buffers, perform distribution correction, etc)

generally speaking, in online RL, we select the actions that our Q-function are overly optimistic about. But since we can interact with the world, we take those actions and observe the actual reward. When we do our bellman update, we correct for this error and our Q-function gets closer to reality.

- **But the primary ability of error correction, i.e., exploration, is impossible in offline RL, due to no access to an environment....**

Kumar, Fu, Tucker, Levine. *Stabilizing Off-Policy RL via Bootstrapping Error Reduction*, NeurIPS 2019.

Levine, Kumar, Tucker, Fu. *Offline RL Tutorial and Perspectives on Open Problems*. arXiv 2020.

Kumar, Gupta, Levine. DisCor: Corrective-Feedback in RL via Distribution Correction. NeurIPS 2020.

Kumar, Gupta. Does On-Policy Data Collection Fix Errors in Off-Policy Reinforcement Learning?, BAIR blog.

# Distributional Shift in Offline RL

- Distribution shift between the behavior policy (the policy that collected the data) and the policy during learning

$$Q(s, a) \leftarrow r(s, a) + \gamma \max_{a'} Q(s', a') \quad \neq \pi_\beta(a|s)$$

$$Q(s, a) \leftarrow r(s, a) + \gamma \mathbb{E}_{a' \sim \pi(a'|s')} Q(s', a') \quad = \pi_\beta(a|s)$$

**Training:**  $\mathbb{E}_{s, a \sim d^{\pi_\beta}(s, a)} [(Q(s, a) - \mathcal{B}\bar{Q}(s, a))^2]$

in offline RL, you're not training on transitions that come from the current policy. You're training on transitions that come from the behavior policy. The distribution shift occurs because the Q-function that we use to calculate the targets (and learn a better Q-function & policy) is NOT the same as the behavior policy.

**Offline Q-Learning algorithms can overestimate the value of unseen actions and can thus be falsely optimistic**

# Error Compounds in RL (Additional Slide)



Typical cartoon showing  
“error compounding” in RL

**Theorem 2.1** (Behavioral cloning error bound). *If  $\pi(a|s)$  is trained via empirical risk minimization on  $s \sim d^{\pi_\beta}(s)$  and optimal labels  $a^*$ , and attains generalization error  $\epsilon$  on  $s \sim d^{\pi_\beta}(s)$ , then  $\ell(\pi) \leq C + H^2\epsilon$  is the best possible bound on the expected error of the learned policy.*

**Theorem 2.2** (DAgger error bound). *If  $\pi(a|s)$  is trained via empirical risk minimization on  $s \sim d^\pi(s)$  and optimal labels  $a^*$ , and attains generalization error  $\epsilon$  on  $s \sim d^\pi(s)$ , then  $\ell(\pi) \leq C + H\epsilon$  is the best possible bound on the expected error of the learned policy.*

Error compounding over the horizon magnifies a small error into a big one.

Recent work has also showed counterexamples that indicate we can't do better.

# **Part 2: Deep RL Algorithms to Address Distribution Shift**

# Addressing Distribution Shift via Pessimism

let's assume we're using actor-critic methods

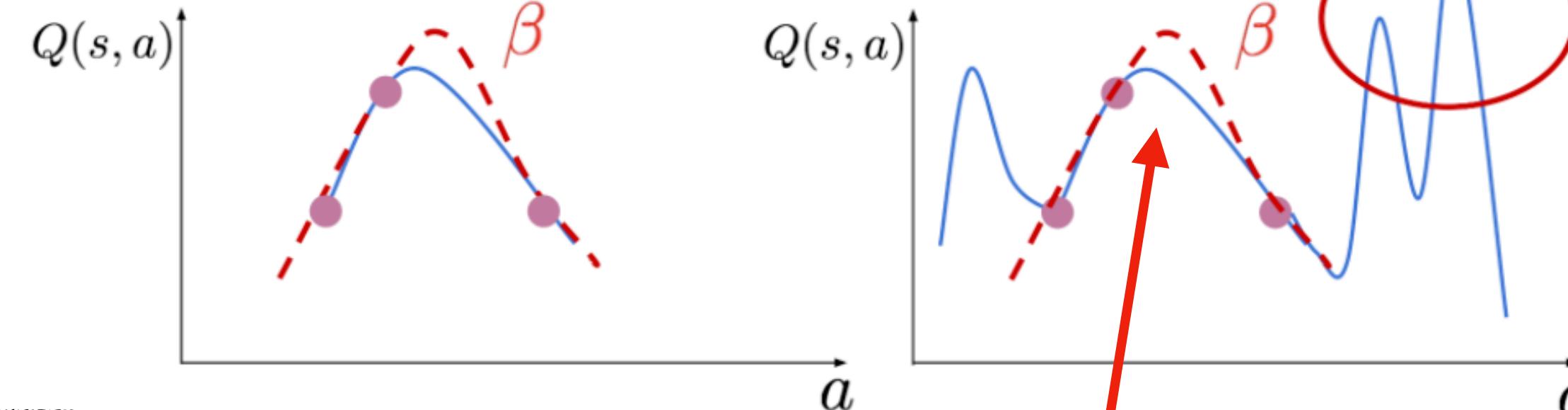
$$Q(s, a) \leftarrow r(s, a) + \gamma \mathbb{E}_{a' \sim \pi_\phi(a|s)} [Q(s', a')]$$

**“Policy Constraint”**

$$\pi_\phi := \arg \max_{\phi} E_{a \sim \pi_\phi(a|s)} [Q(s, a)] \text{ s.t. } D(\pi_\phi(a|s), \pi_\beta(a|s)) \leq \varepsilon$$

we prevent the learned policy  $\pi_\phi$  to be that different from the behavior policy

**Out-of-distribution action values are no longer used for the backup**



**Hence, all values used during training are also trained, leading to better learning**

$$\mathbb{E}_{a' \sim \pi_\phi(a|s)} [Q(s', a')]$$

instead of our target function (using  $Q(s', a')$ ) coming from  $(s', a')$  that are out of distribution, we instead use  $(s', a')$  that aren't that far away from our behavior policy. So we don't use the actions  $a'$  for our bellman backup anymore. Our target value is not computed using actions from our behavior policy.

# Different Types of Policy Constraints

$$\pi_\phi := \arg \max_{\phi} E_{a \sim \pi_\phi(a|s)}[Q(s, a)] \text{ s.t. } D(\pi_\phi(a|s), \pi_\beta(a|s)) \leq \varepsilon$$

## Several Ways of Implementing Them:

an example  
of support  
matching

$$D(\pi_\phi, \pi_\beta) = \text{MMD}(\pi_\phi, \pi_\beta)$$

distance (or  
divergence)

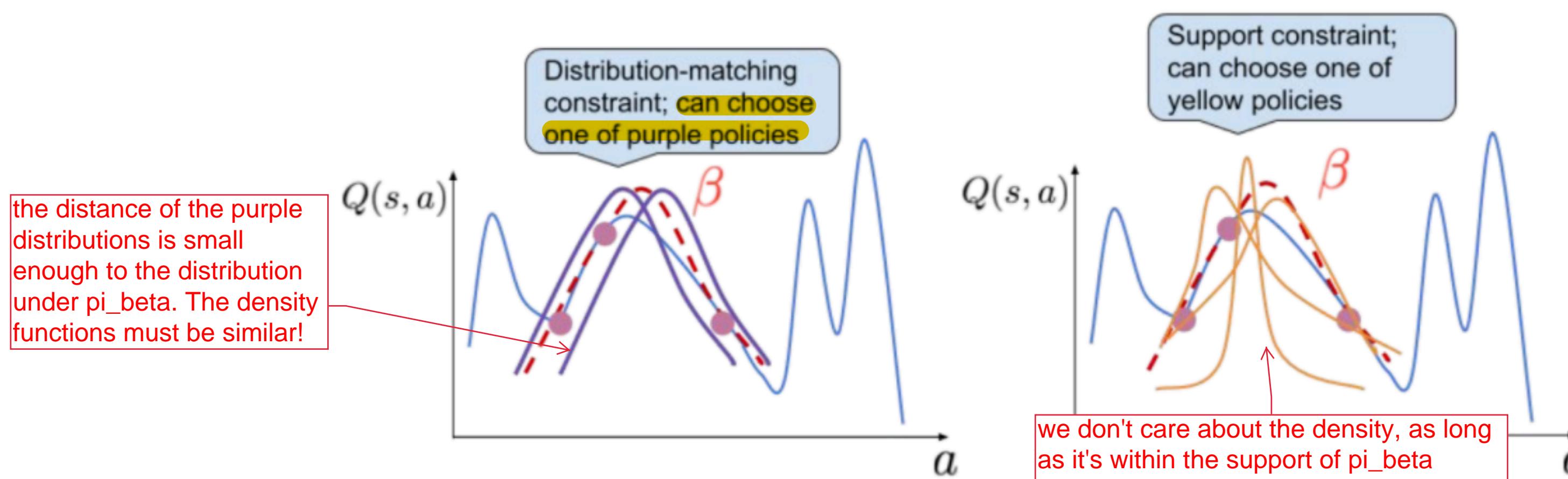
make sure that the policy that we're learning  $\pi_\phi$  is choosing actions that's in the support of the behavior distribution, meaning that actions that it chooses should be likely "enough" under the behavior policy. The probability that  $\pi_\phi$  chooses an action that's likely under  $\pi_\beta$  may vary. But this doesn't care about the density of  $\pi_\phi(s)$  compared to  $\pi_\beta(s)$ , it just cares that the action taken by  $\pi_\phi$  should be within the support of  $\pi_\beta$

- **Support matching** (Kumar et al. 2019, Laroche et al. 2019, Wu et al. 2019)
- **Distribution matching** (Peng et al. 2019, Fujimoto et al. 2019, Jaques et al. 2019)  
ensures the distribution of  $\pi_\phi(s)$  is close to the distribution of  $\pi_\beta(s)$
- **State-marginal constraints** (Nachum & Dai 2020)

$$D(\pi_\phi, \pi_\beta) = D_{\text{KL}}(\pi_\phi, \pi_\beta)$$

$$D(\pi_\phi, \pi_\beta) = D(d^{\pi_\phi}(s, a), d^{\pi_\beta}(s, a))$$

- **Implicit /closed-form distribution constraints** (Peng et al. 2019, Nair et al. 2020, Wang et al. 2020)



**Different types of constraints lead to different solutions, providing a whole lot of different offline RL algorithms**

# Which constraint should I use?

Before answering this question, let's see how the usage of a policy constraint affects optimal solutions?

$$\max_{\pi} \mathbb{E}_{\pi} \left[ \sum_t \gamma^t r(s_t, a_t) \right] - \alpha D(\pi(a|s), \pi_{\beta}(a|s))$$

**Adding pessimism alters the optimal performance**

Thus we would want the constraint to be least restrictive, while still preventing the “badness”

we want the constraint to be the least restrictive as possible while preventing the erroneous over-optimization

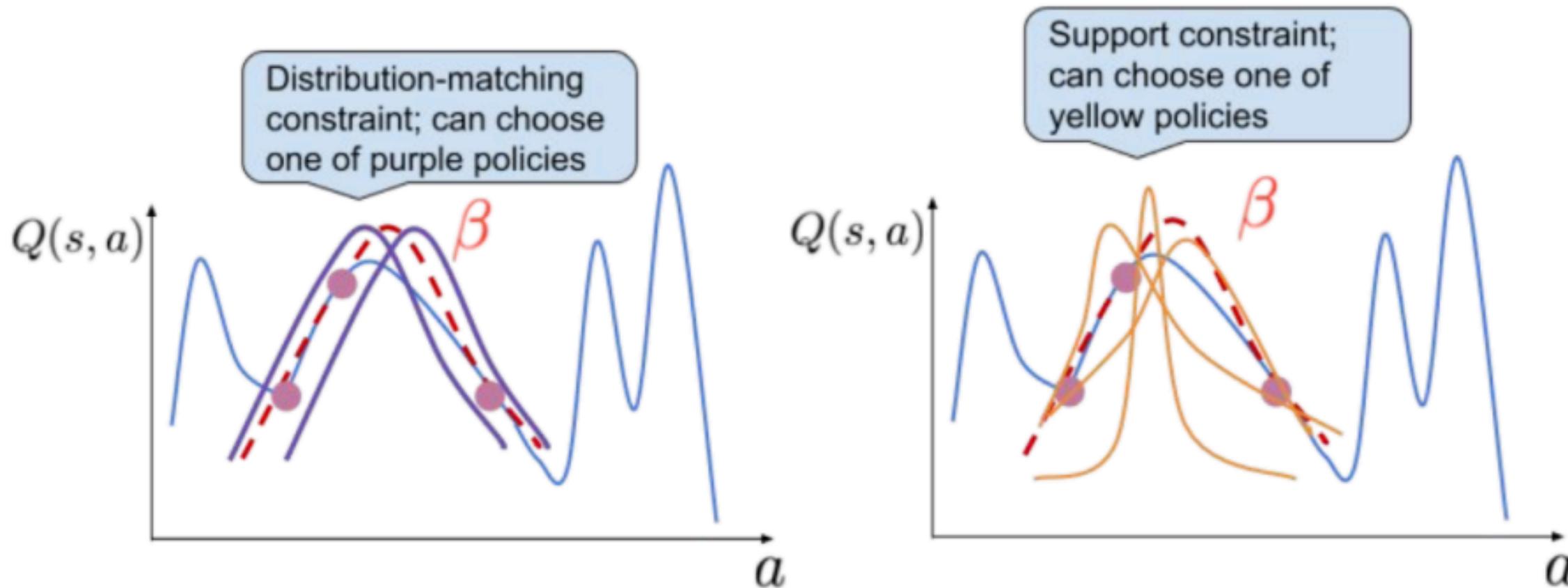
support constraints vs. distribution constraints

- Technically, support constraints are less restrictive
  - Imagine a case where the behavior policy takes all actions uniformly.
  - Constraining to the behavior policy via distribution-matching may lead to highly stochastic policies that are not optimal. ↗ because we're limited to a distribution that is close to uniform!
  - However, choosing to match only supports leads to choosing in-distribution actions, but at the same time, only optimises the RL objective

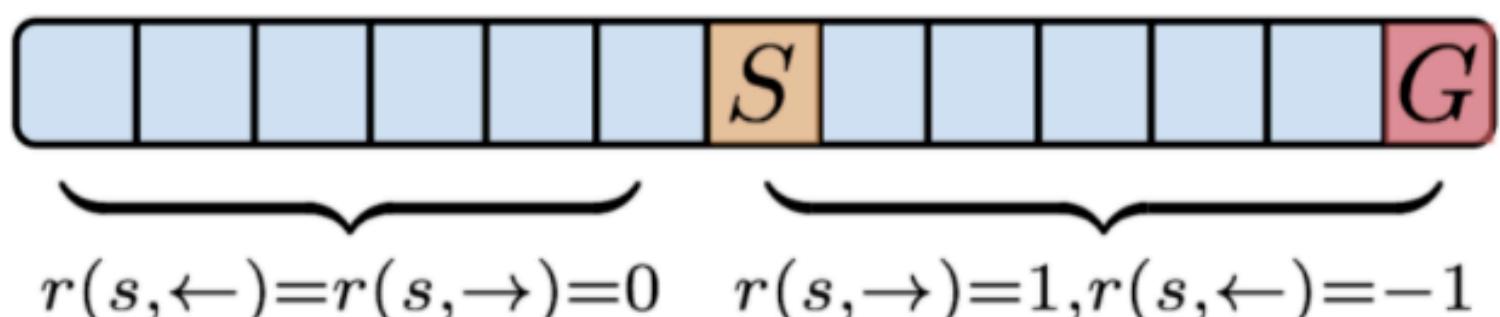
assume  
we're in a  
discrete  
setting

the density function could be anything using this approach. That's because all actions are seen by  $\pi_{\beta}$ , so everything is within the support

# Which constraint should I use?



Actions:  $\rightarrow, \leftarrow$

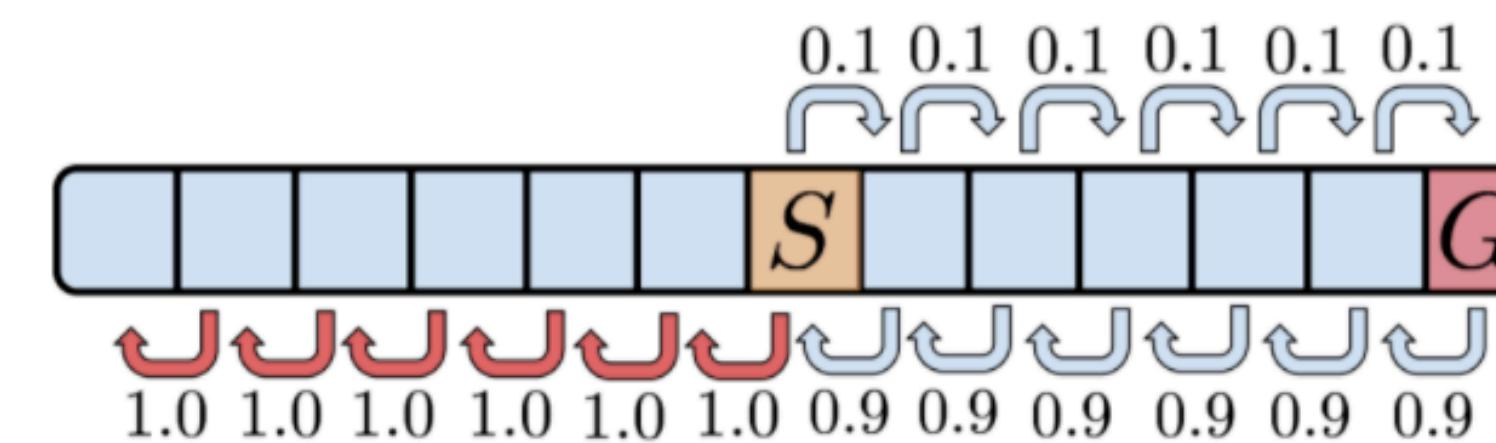


Initial state:  $S$  Goal state:  $G$

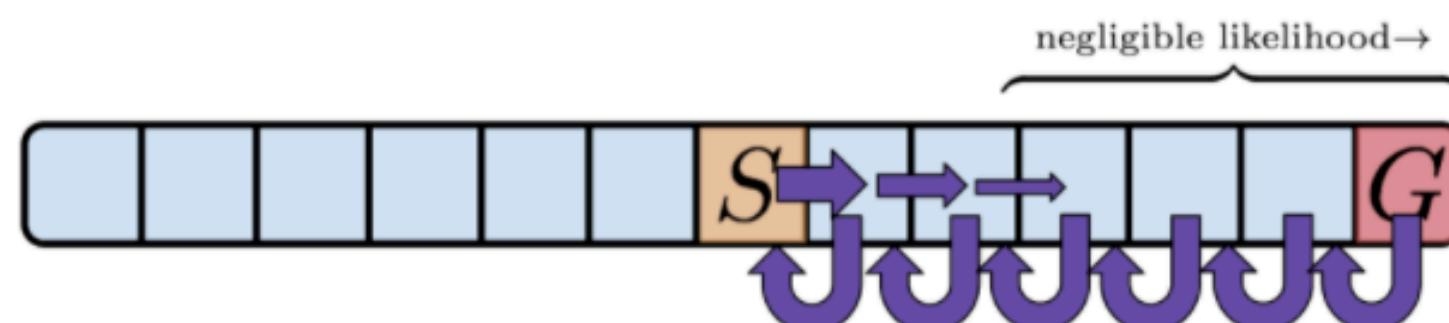
(a) 1D-Lineworld Environment

**Support constraints better in theory, but not much difference in practice, often depends on how well can policy constraint methods be tuned**

performance is highly dependent on hyperparameter tuning



(b) Behavior Policy



(a) Learned Policy via distribution-matching

it's hard for us to learn a policy that goes to the right because we are constrained to policies that go to the left

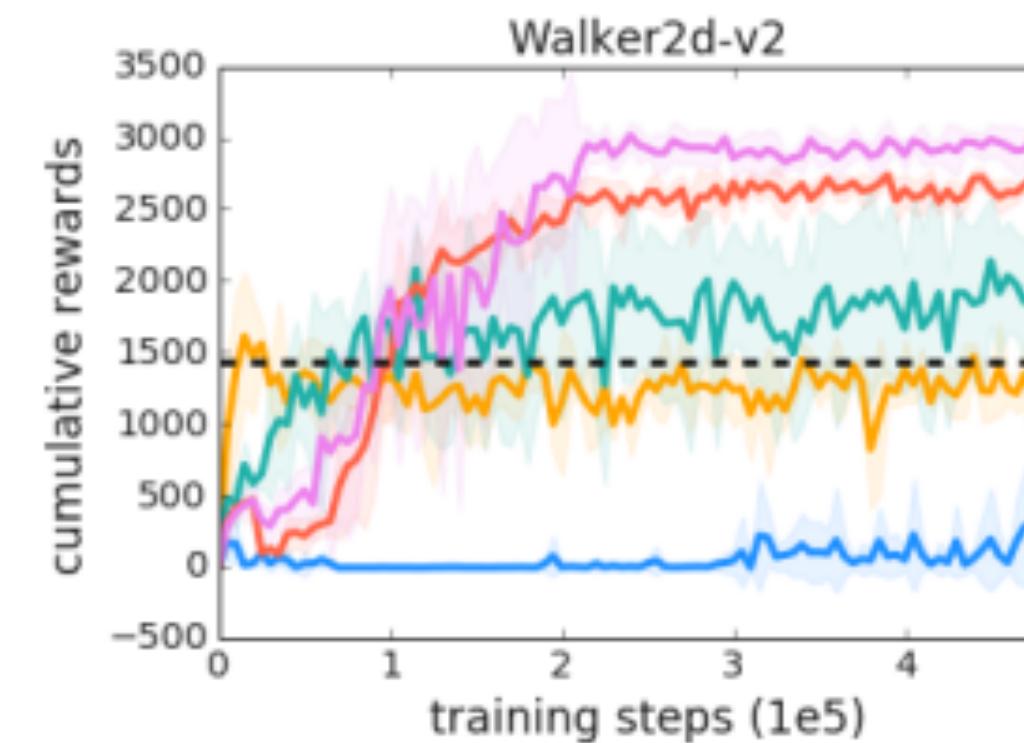
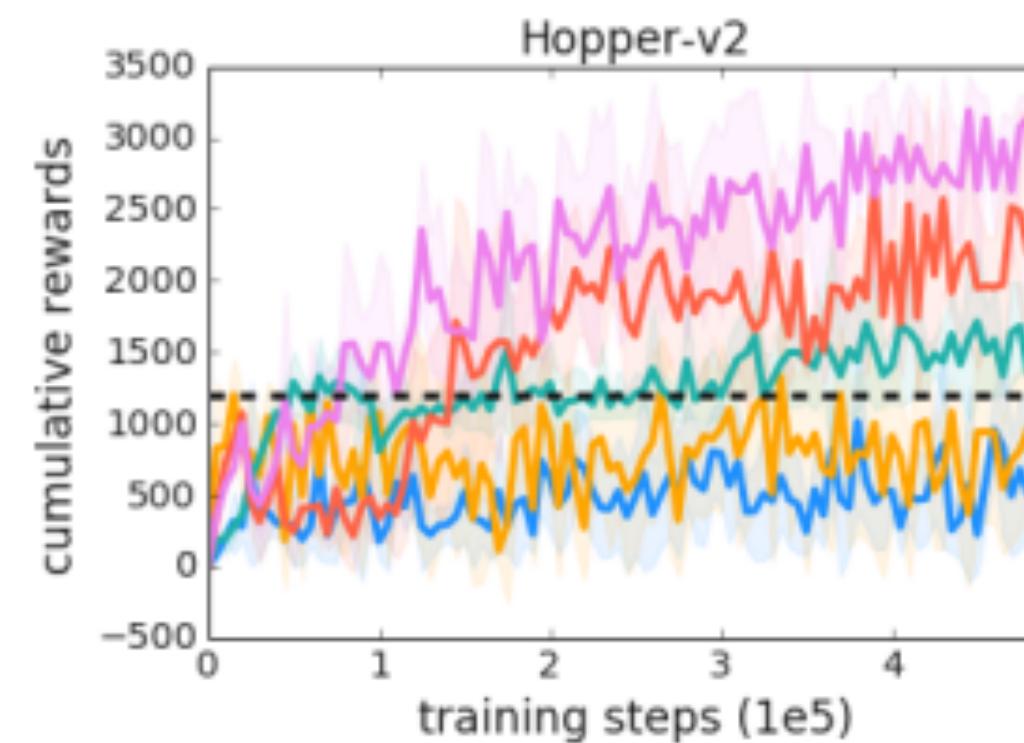


(b) Learned Policy via support-constraint

# Policy Constraint Methods, Empirically

not an expert policy,  
but not random

Dataset collected from a mixture of random and “mediocre” policies



— sac — bc — bcq — bear — kl\_vp

Naive off-policy RL

Behavior cloning

Policy constraint methods: BCQ, BEAR and BRAC (with KL)

Better than BC

Different choices  
of D matter

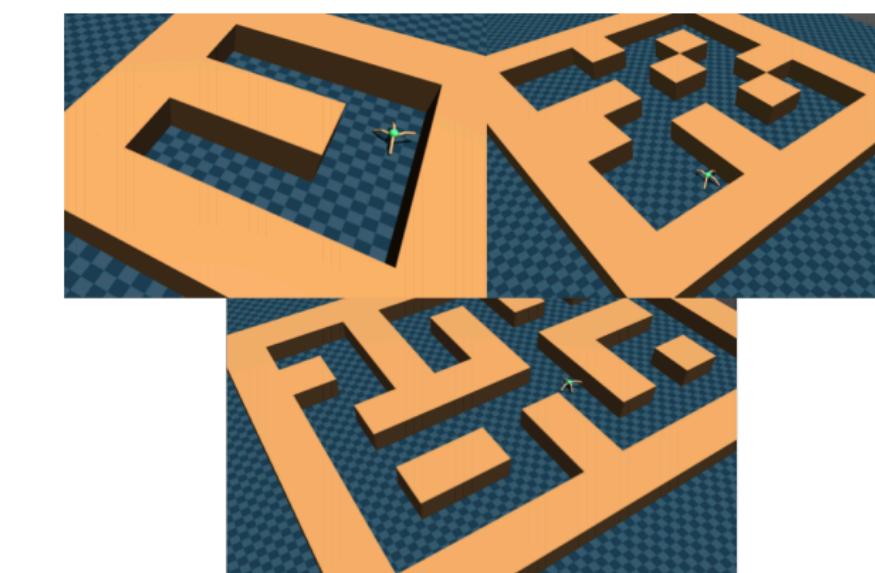
all three offline algs are better  
than behavioral cloning

BEAR uses MMD, BCQ is distribution  
constraint, BRAC also uses  
distribution constraint but is slightly  
different

How do these  
methods perform  
on harder tasks?

Domain	Task Name	BC	SAC	BEAR	BRAC-p	BRAC-v
AntMaze	antmaze-umaze	65.0	0.0	<b>73.0</b>	50.0	70.0
	antmaze-umaze-diverse	55.0	0.0	61.0	40.0	70.0
	antmaze-medium-play	0.0	0.0	0.0	0.0	0.0
	antmaze-medium-diverse	0.0	0.0	8.0	0.0	0.0
	antmaze-large-play	0.0	0.0	0.0	0.0	0.0
	antmaze-large-diverse	0.0	0.0	0.0	0.0	0.0

none of  
these work  
for the more  
complicated  
mazes



in AntMaze, we have no  
trajectories that take us from the  
start to the goal. This means that  
our policy must learn to stitch  
together paths from the data to  
get to the goal

# Are policy constraint methods sufficient?

no

## Require estimation of the behavior policy

maybe the dataset came from different policies. So we have to estimate  $\pi_{\beta}$  first from the data, using something like behavior cloning. Only after can we constraint  $\pi_{\phi}$  against  $\pi_{\beta}$

policy constraint methods can be effective at offline RL, but they are limited because they require estimating the behavior policy, and they often can be too conservative

$$\pi_{\phi} := \arg \max_{\phi} E_{a \sim \pi_{\phi}(a|s)}[Q(s, a)] \text{ s.t. } D(\pi_{\phi}(a|s), \pi_{\beta}(a|s)) \leq \varepsilon$$

estimated from data

If the behavior policy is wrongly estimated (e.g, when it does not match the function class), policy constraint methods can fail dramatically (e.g., AntMaze)

## Often tend to be too conservative

If we know that a certain state has all actions with 0 reward, we do not care about constraining the policy there, since we will not be worse...

## Can we do better?

Nair, Dalal, Gupta, Levine. *Accelerating Online RL with Offline Datasets*. arXiv 2020.

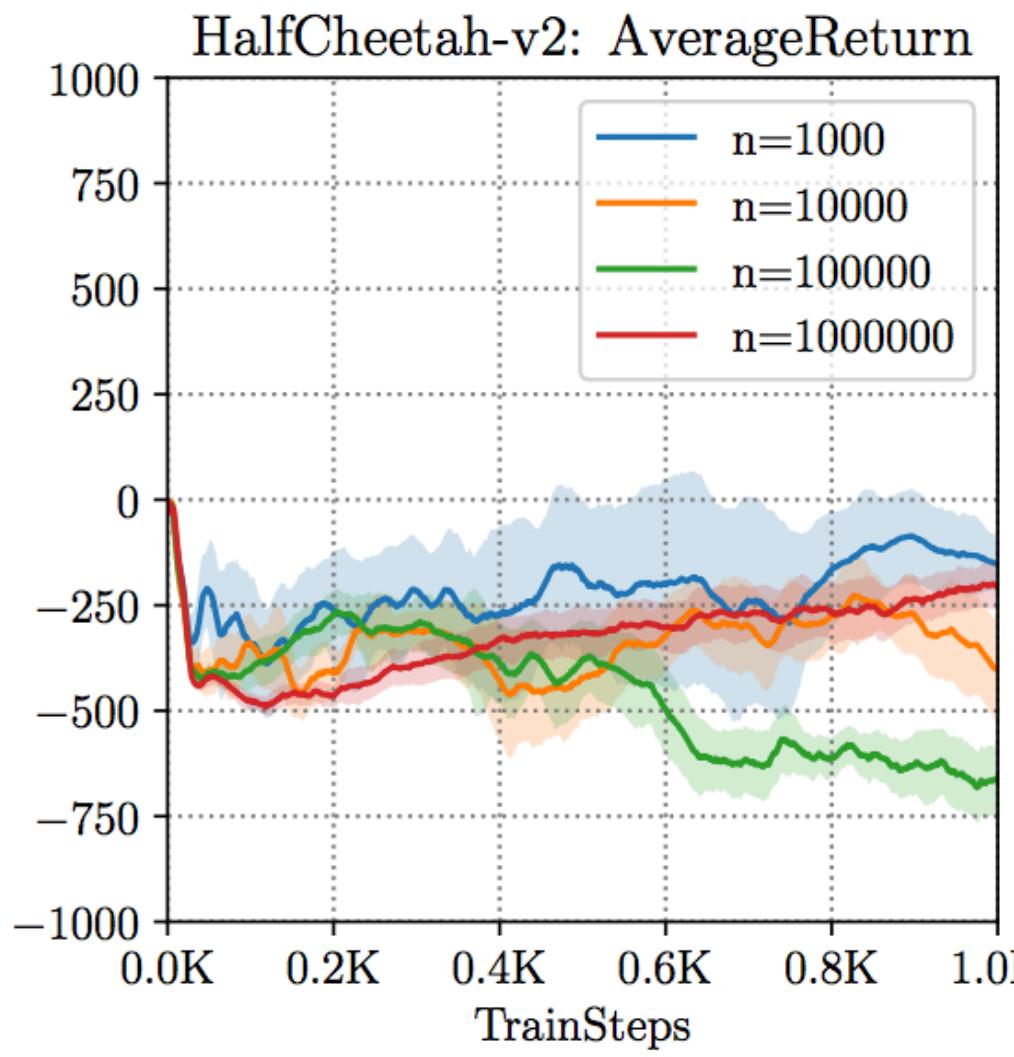
Kumar, Zhou, Tucker, Levine. *Conservative Q-Learning for Offline RL*. NeurIPS 2020.

Levine, Kumar, Tucker, Fu. *Offline RL Tutorial and Perspectives on Open Problems*. arXiv 2020.

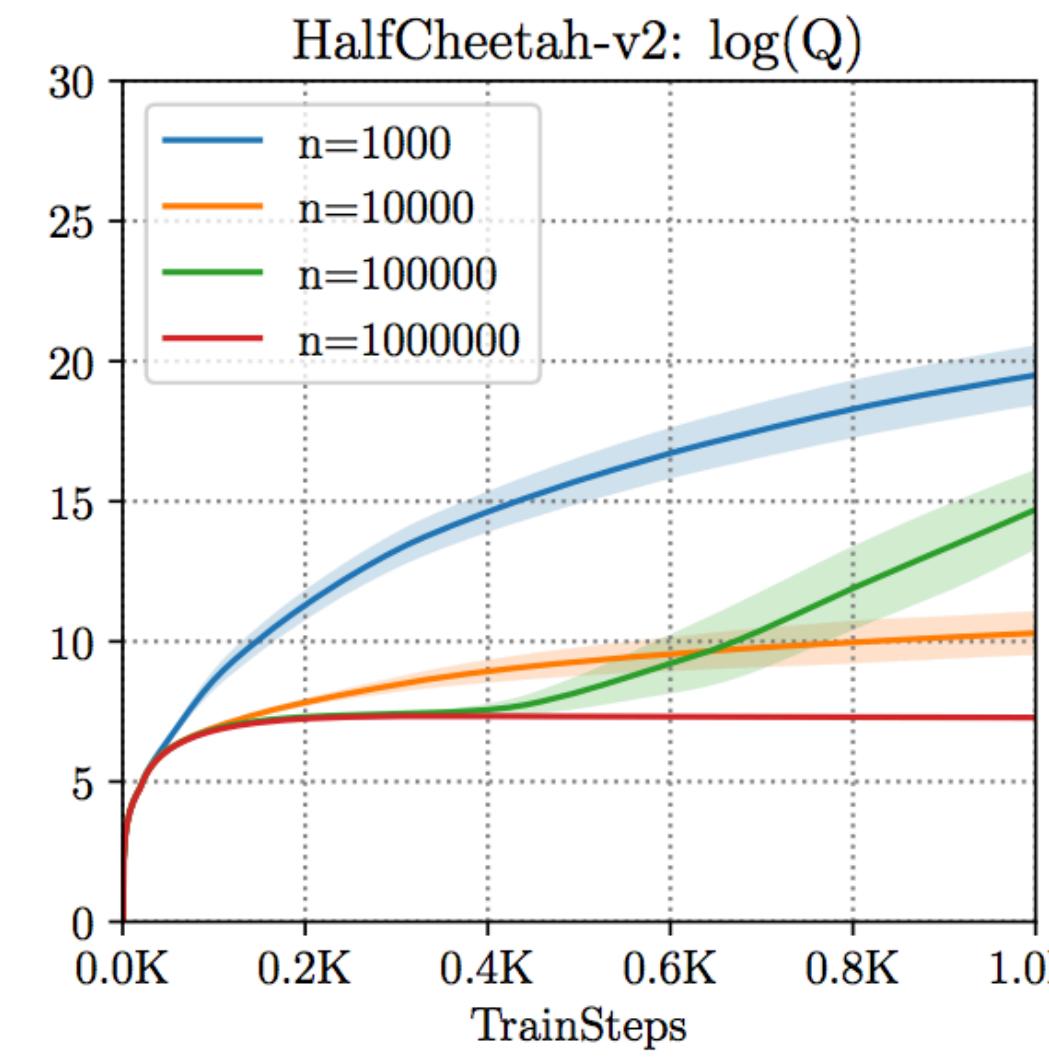
Ghasemipour, Schurmanns, Gu. *EmaQ: Expected Max Q-Learning*. arXiv 2020.

# Let's revisit the motivating example

(and take a slightly different perspective on the problem)



how well it does



how well it thinks it does

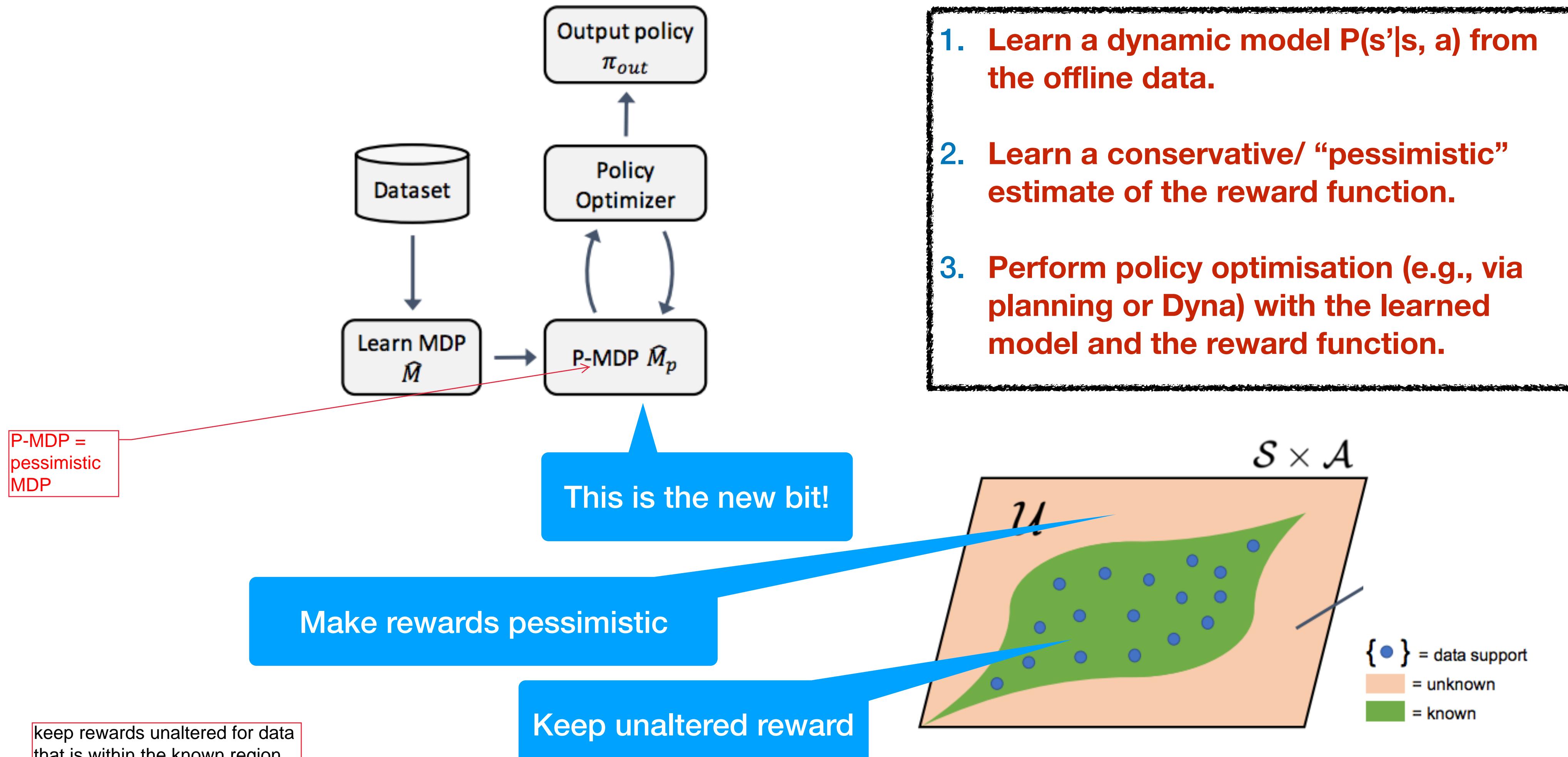
Can we directly tackle false over-estimation, instead of fixes to avoid out-of-distribution actions?

In some cases, not all out-of-distribution actions are bad, they are bad if they affect the policy (i.e. when values are overestimated)

Can we devise methods that learn lower-bounds on the policy value/ performance?

Yes! Two ways: model-based and model-free

# A Framework for Conservative Model-Based RL



Janner, Fu, Zhang, Levine. When to Trust Your Model? Model-Based Policy Optimization. NeurIPS 2019.  
Yu, Thomas, Yu, Ermon, Zou, Levine, Finn, Ma. MOPO: Model-based Offline Policy Optimization. NeurIPS 2020.  
Kidambi, Rajeswaran, Netrapalli, Joachims. MORL: Model-Based Offline Reinforcement Learning. NeurIPS 2020.

# Model-Based Offline RL Methods

$$\tilde{r}(s, a) = r(s, a) - \lambda u(s, a).$$

pessimism

**MOPO (Yu et al. 2020)**

you keep an ensemble of dynamics models. Each dynamics model is stochastic and has some uncertainty. It computes the maximum norm of the covariance matrix across the ensemble of models. If the max norm is high, then this means that there is high uncertainty in our models for a particular (s,a) pair. This means that we don't really know the state-action space that well. So, we make the reward conservative.

Covariance matrix of an ensemble of dynamics models

$$\tilde{r}_j = r_j - \lambda \max_{i=1}^N \|\Sigma^i(s_j, a_j)\|_F$$

**MBPO (Dyna)**

then it uses Dyna for model-based planning

**MOReL (Kidambi et al. 2020)**

Disagreement in an ensemble of dynamics models

$$\text{disc}(s, a) = \max_{i,j} \|f_{\phi_i}(s, a) - f_{\phi_j}(s, a)\|_2$$

this also trains an ensemble of dynamics models, but they are deterministic in that  $(s, a) \rightarrow s'$  always.

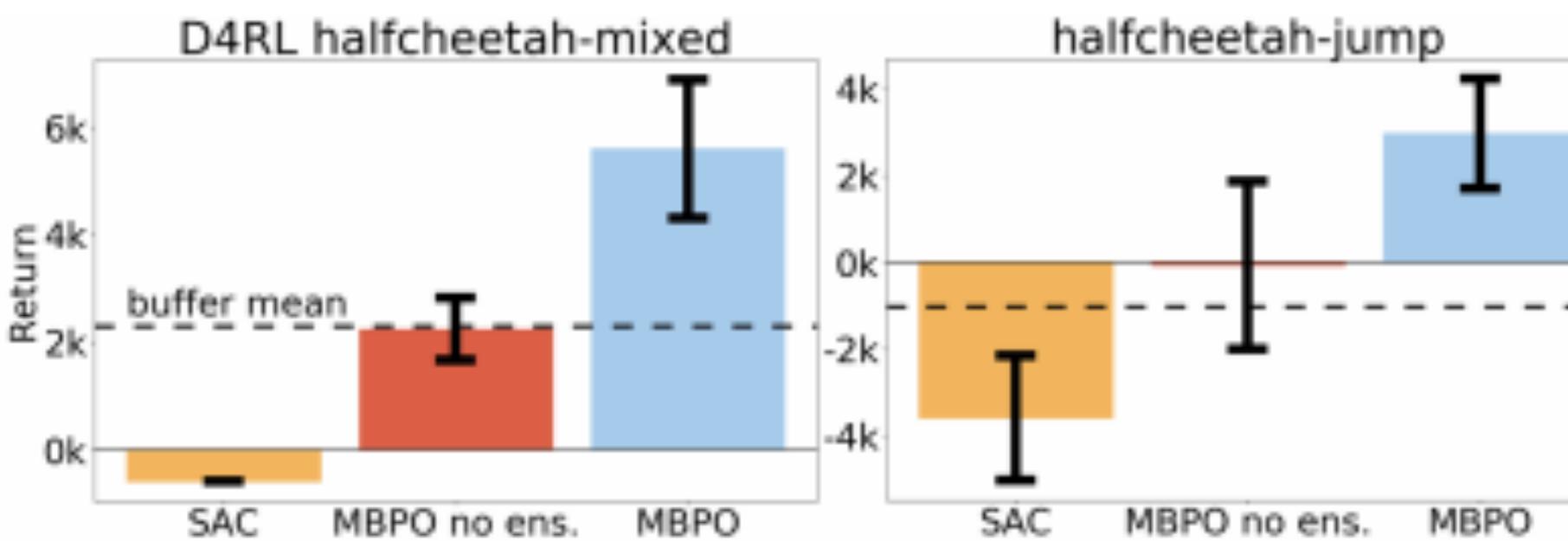
Then we calculate disagreement between the models by calculating the L2 distance. This distance will be high if we haven't seen a particular (s,a), and thus our models will be uncertain about it.

If the disagreement is greater than some threshold, we assign the minimum reward you can get in the MDP, making it conservative.

**Planning**

$$\tilde{r}(s, a) = -R_{\max} \quad \text{if } \text{disc}(s, a) > \text{threshold}$$

# Model-Based Offline RL, Empirically



Model-based methods without any form of correction can work well with “broad” coverage datasets

Conservatism helps in situations with narrow datasets (see MBPO vs MOPO on med-expert)

Dataset type	Environment	BC	MOPO (ours)	MBPO	SAC	BEAR	BRAC-v
random	halfcheetah	2.1	<b>31.9</b> ± 2.8	30.7 ± 3.9	30.5	25.5	28.1
random	hopper	1.6	<b>13.3</b> ± 1.6	4.5 ± 6.0	11.3	9.5	12.0
random	walker2d	9.8	<b>13.0</b> ± 2.6	8.6 ± 8.1	4.1	6.7	0.5
medium	halfcheetah	36.1	<b>40.2</b> ± 2.7	28.3 ± 22.7	-4.3	38.6	<b>45.5</b>
medium	hopper	29.0	<b>26.5</b> ± 3.7	4.9 ± 3.3	0.8	<b>47.6</b>	32.3
medium	walker2d	6.6	<b>14.0</b> ± 10.1	12.7 ± 7.6	0.9	33.2	<b>81.3</b>
mixed	halfcheetah	38.4	<b>54.0</b> ± 2.6	47.3 ± 12.6	-2.4	36.2	45.9
mixed	hopper	11.8	<b>92.5</b> ± 6.3	49.8 ± 30.4	1.9	10.8	0.9
mixed	walker2d	11.3	<b>42.7</b> ± 8.3	22.2 ± 12.7	3.5	25.3	0.8
med-expert	halfcheetah	35.8	<b>57.9</b> ± 24.8	9.7 ± 9.5	1.8	51.7	45.3
med-expert	hopper	111.9	<b>51.7</b> ± 42.9	<b>56.0</b> ± 34.5	1.6	4.0	0.8
med-expert	walker2d	6.4	<b>55.0</b> ± 19.1	7.6 ± 3.7	-0.1	26.0	<b>66.6</b>

Better than policy constraint methods generally

# Learning Lower-Bounded Q-values

## Conservative Q-Learning (CQL) Algorithm

Since learned Q-values (our belief of policy values) are overestimated, let's make them provably lower bound the true value

$$\hat{Q}_{\text{CQL}}^{\pi} := \min_Q \max_{\mu} \mathbb{E}_{a \sim \mu(a|s)} [Q(s, a)] + \frac{1}{2\alpha} \mathbb{E}_{s, a, s' \sim \mathcal{D}} [(Q(s, a) - (r(s, a) + \gamma \mathbb{E}_{a \sim \pi_{\phi}(a|s)} [\bar{Q}(s', a')]))^2]$$

CQL augments the standard bellman error with a regularizer, which we'll call the CQL regularizer.

This regularizer minimizes the large Q-values for a particular state  $s$ .

This minimizes bellman error

Minimize big Q-values

Standard Bellman Error

$\hat{Q}_{\text{CQL}}^{\pi}(s, a) \leq Q(s, a) \quad \forall s \in \mathcal{D}, a$

we under estimate all Q-values

CQL-v1

### CQL Algorithm:

1. Learn  $\hat{Q}_{\text{CQL}}^{\pi}$  using offline data  $\mathcal{D}$ .
2. Optimize policy w.r.t.  $\hat{Q}_{\text{CQL}}^{\pi}$ :  $\pi \leftarrow \arg \max_{\pi} \mathbb{E}_{\pi} [\hat{Q}_{\text{CQL}}^{\pi}]$ .

now you can plug in CQL into a standard actor-critic method that tries to optimize the policy against these learned (conservative) Q-values.

one issue is that the conservative nature can make the Q-values too low

# A Tighter Lower Bound

Three terms in the objective function:

- 1) We want to minimize Q-values for  $(s, a)$  that we've not seen
- 2) We want to increase Q-values for data that we have seen
- 3) We want to minimize bellman error

$$\hat{Q}_{\text{CQL}}^{\pi} := \min_Q \max_{\mu} \mathbb{E}_{a \sim \mu(a|s)} [Q(s, a)] - \mathbb{E}_{a \sim \mathcal{D}(a|s)} [Q(s, a)]$$

Minimize big Q-values

Maximize Data Q-values

maximizes Q-values on actions sampled from the dataset

$$+ \frac{1}{2\alpha} \mathbb{E}_{s, a, s' \sim \mathcal{D}} [(Q(s, a) - (r(s, a) + \gamma \mathbb{E}_{a \sim \pi_{\phi}(a|s)} [\bar{Q}(s', a')]))^2]$$

Standard Bellman Error

$$\hat{Q}_{\text{CQL}}^{\pi}(s, a) \leq Q(s, a) \quad \forall s \in \mathcal{D}, a$$

we no longer under estimate all Q-values

$$\hat{V}_{\text{CQL}}^{\pi}(s) := \mathbb{E}_{a \sim \pi_k} [\hat{Q}_{\text{CQL}}^{\pi}(s, a)] \leq V^{\pi}(s) \quad \forall s \in \mathcal{D}$$

CQL-v2

the overall value of the policy still remains a lower bound

**CQL Algorithm:**

1. Learn  $\hat{Q}_{\text{CQL}}^{\pi}$  using offline data  $\mathcal{D}$ .
2. Optimize policy w.r.t.  $\hat{Q}_{\text{CQL}}^{\pi}$ :  $\pi \leftarrow \arg \max_{\pi} \mathbb{E}_{\pi} [\hat{Q}_{\text{CQL}}^{\pi}]$ .

# Practical CQL Algorithm

**CQL(H)**

we can instead do a soft max instead of a hard max

$$\min_Q \alpha \mathbb{E}_{\mathbf{s} \sim \mathcal{D}} \left[ \log \sum_{\mathbf{a}} \exp(Q(\mathbf{s}, \mathbf{a})) - \mathbb{E}_{\mathbf{a} \sim \hat{\pi}_\beta(\mathbf{a}|\mathbf{s})} [Q(\mathbf{s}, \mathbf{a})] \right] + \frac{1}{2} \mathbb{E}_{\mathbf{s}, \mathbf{a}, \mathbf{s}' \sim \mathcal{D}} \left[ (Q - \hat{\mathcal{B}}^{\pi_k} \hat{Q}^k)^2 \right].$$

---

## Algorithm 1 Conservative Q-Learning (both variants)

---

- 1: Initialize Q-function,  $Q_\theta$ , and optionally a policy,  $\pi_\phi$ .
- 2: **for** step  $t$  in  $\{1, \dots, N\}$  **do**
- 3:   Train the Q-function using  $G_Q$  gradient steps on objective from Equation 4  

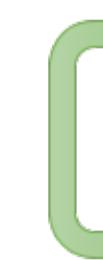
$$\theta_t := \theta_{t-1} - \eta_Q \nabla_\theta \text{CQL}(\mathcal{R})(\theta)$$
  
 (Use  $\mathcal{B}^*$  for Q-learning,  $\mathcal{B}^{\pi_{\phi_t}}$  for actor-critic)
- 4:   (only with actor-critic) Improve policy  $\pi_\phi$  via  $G_\pi$  gradient steps on  $\phi$  with SAC-style entropy regularization:  

$$\phi_t := \phi_{t-1} + \eta_\pi \mathbb{E}_{\mathbf{s} \sim \mathcal{D}, \mathbf{a} \sim \pi_\phi(\cdot|\mathbf{s})} [Q_\theta(\mathbf{s}, \mathbf{a}) - \log \pi_\phi(\mathbf{a}|\mathbf{s})]$$
- 5: **end for**

Only change on top of standard Deep Q-Learning

$$\begin{aligned} \min_Q \max_\mu \alpha & (\mathbb{E}_{\mathbf{s} \sim \mathcal{D}, \mathbf{a} \sim \mu(\mathbf{a}|\mathbf{s})} [Q(\mathbf{s}, \mathbf{a})] - \mathbb{E}_{\mathbf{s} \sim \mathcal{D}, \mathbf{a} \sim \hat{\pi}_\beta(\mathbf{a}|\mathbf{s})} [Q(\mathbf{s}, \mathbf{a})]) \\ & + \frac{1}{2} \mathbb{E}_{\mathbf{s}, \mathbf{a}, \mathbf{s}' \sim \mathcal{D}} \left[ (Q(\mathbf{s}, \mathbf{a}) - \hat{\mathcal{B}}^{\pi_k} \hat{Q}^k(\mathbf{s}, \mathbf{a}))^2 \right] + \mathcal{R}(\mu) \quad (\text{CQL}(\mathcal{R})). \end{aligned}$$

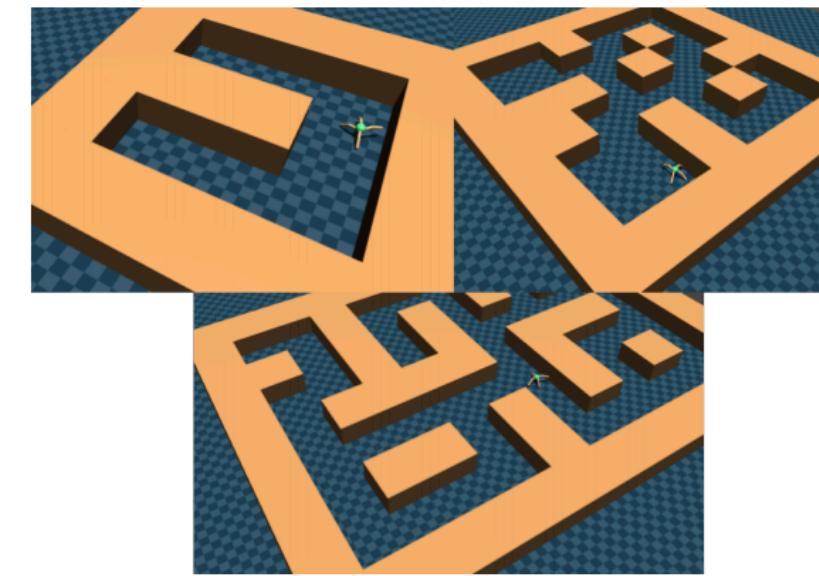
## CQL Algorithm:

- 
1. Learn  $\hat{Q}_{\text{CQL}}^\pi$  using offline data  $\mathcal{D}$ .
  2. Optimize policy w.r.t.  $\hat{Q}_{\text{CQL}}^\pi$ :  $\pi \leftarrow \arg \max_\pi \mathbb{E}_\pi [\hat{Q}_{\text{CQL}}^\pi]$ .

# CQL, Empirically

these values show the learned policy value minus the actual policy value. Negatives mean we are underestimating the actual policy

Learned policy value - Actual policy value



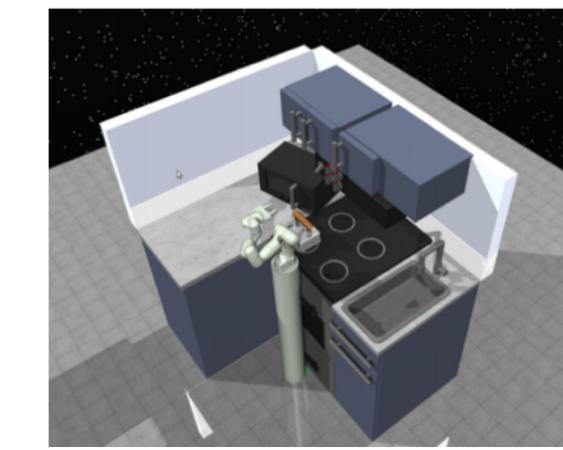
Task Name	CQL( $\mathcal{H}$ )	CQL (Eqn. 1)	Ensemble(2)	Ens.(4)	Ens.(10)	Ens.(20)	BEAR
hopper-medium-expert	-43.20	-151.36	3.71e6	2.93e6	0.32e6	24.05e3	65.93
hopper-mixed	-10.93	-22.87	15.00e6	59.93e3	8.92e3	2.47e3	1399.46
hopper-medium	-7.48	-156.70	26.03e12	437.57e6	1.12e12	885e3	4.32

Behavior cloning

Naive off-policy RL

Policy constraint methods

Domain	Task Name	BC	SAC	BEAR	BRAC-p	BRAC-v	CQL( $\mathcal{H}$ )	CQL( $\rho$ )
AntMaze	antmaze-umaze	65.0	0.0	<b>73.0</b>	50.0	70.0	<b>74.0</b>	<b>73.5</b>
	antmaze-umaze-diverse	55.0	0.0	61.0	40.0	70.0	<b>84.0</b>	61.0
	antmaze-medium-play	0.0	0.0	0.0	0.0	0.0	<b>61.2</b>	4.6
	antmaze-medium-diverse	0.0	0.0	8.0	0.0	0.0	<b>53.7</b>	5.1
	antmaze-large-play	0.0	0.0	0.0	0.0	0.0	<b>15.8</b>	3.2
	antmaze-large-diverse	0.0	0.0	0.0	0.0	0.0	<b>14.9</b>	2.3
Adroit	pen-human	34.4	6.3	-1.0	8.1	0.6	37.5	<b>55.8</b>
	hammer-human	1.5	0.5	0.3	0.3	0.2	<b>4.4</b>	2.1
	door-human	0.5	3.9	-0.3	-0.3	-0.3	<b>9.9</b>	<b>9.1</b>
	relocate-human	0.0	0.0	-0.3	-0.3	-0.3	0.20	<b>0.35</b>
	pen-cloned	<b>56.9</b>	23.5	26.5	1.6	-2.5	39.2	40.3
	hammer-cloned	0.8	0.2	0.3	0.3	0.3	2.1	<b>5.7</b>
	door-cloned	-0.1	0.0	-0.1	-0.1	-0.1	0.4	<b>3.5</b>
	relocate-cloned	<b>-0.1</b>	-0.2	-0.3	-0.3	-0.3	<b>-0.1</b>	<b>-0.1</b>
Kitchen	kitchen-complete	33.8	15.0	0.0	0.0	0.0	<b>43.8</b>	31.3
	kitchen-partial	33.8	0.0	13.1	0.0	0.0	<b>49.8</b>	<b>50.1</b>
	kitchen-undirected	47.5	2.5	47.2	0.0	0.0	<b>51.0</b>	<b>52.4</b>



“Stitching”

Better than other methods, not the best in each case

Only method to outperform BC

# Offline RL Algorithms covered so far

- **Policy Constraint Methods:**
  - Support constraints
  - Distribution constraints
  - State-marginal constraints

Work well, but are conservative and require behavior policy estimation

- **Learning lower-bounded policy-values:**
  - Model-based algorithms
  - Direct Q-function penalties (CQL)

Generally perform better, since they are less conservative, and do not require behavior policy estimation

---

Next, we will cover some related problems, discuss how we should evaluate offline RL methods, and finally, discuss some practical examples.

# A Related Problem: Off-Policy Evaluation

instead of finding an optimal policy, let's try to estimate the value of a policy using data. This will help us make comparative decisions on which policy is better than another

**Problem Statement:** Rather than returning a good policy, find me the value of a given policy, **without running this policy in the environment**

$$\pi \xrightarrow{\quad} V^\pi(s)$$

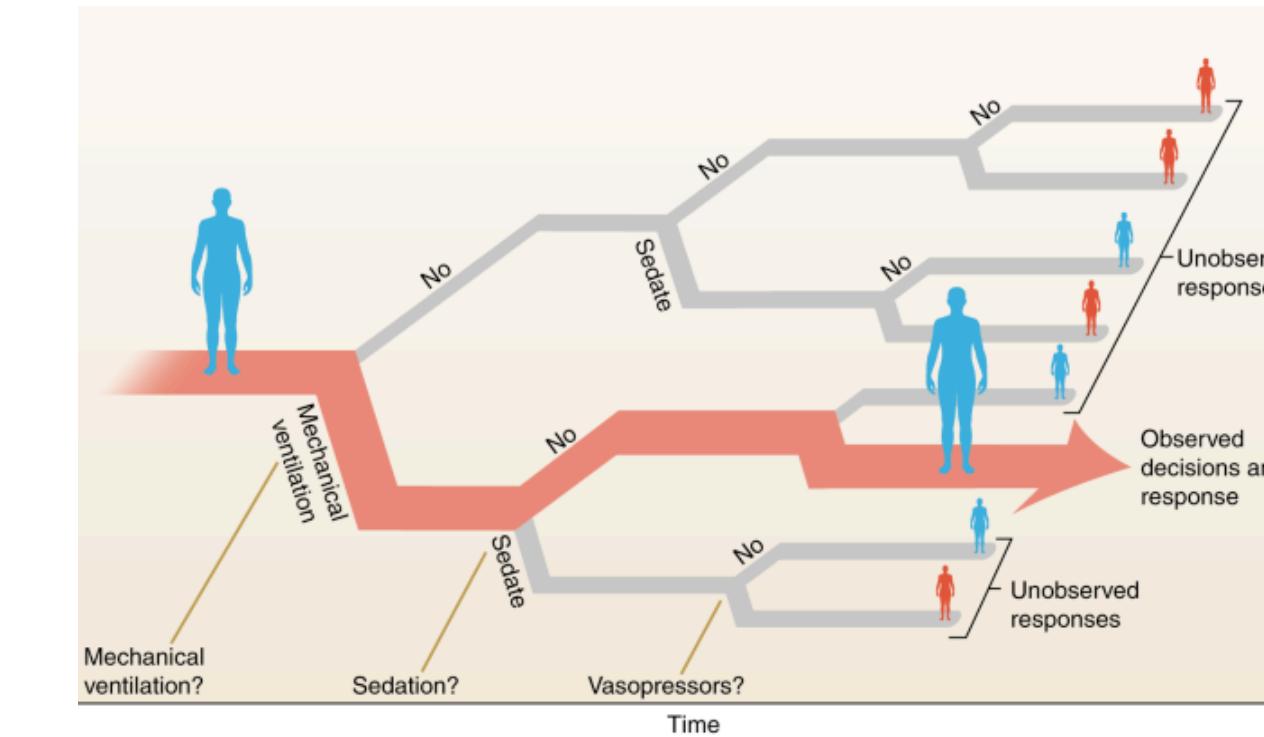
$\mathcal{D}$

if we have multiple types of treatment plans in medicine, we can use this to evaluate which plan is better to give to patients

$$V^{\pi_1}(s) > V^{\pi_2}(s)?$$

**What can be the use of OPE in offline RL?**

Model-selection: selecting which policy is good



**Why do we need model-selection in offline RL?**

Similar to supervised learning methods, excessive training on the same offline dataset can produce poor solutions. If we can rank these solutions using OPE, we can get good offline performance.

# A quick glance on some OPE methods

- **Importance Sampling** (similar to off-policy policy gradient)

$$\begin{aligned} J(\pi_\theta) &= \mathbb{E}_{\tau \sim \pi_\beta(\tau)} \left[ \frac{\pi_\theta(\tau)}{\pi_\beta(\tau)} \sum_{t=0}^H \gamma^t r(s, a) \right] \\ &= \mathbb{E}_{\tau \sim \pi_\beta(\tau)} \left[ \left( \prod_{t=0}^H \frac{\pi_\theta(a_t | s_t)}{\pi_\beta(a_t | s_t)} \right) \sum_{t=0}^H \gamma^t r(s, a) \right] \approx \sum_{i=1}^n w_H^i \sum_{t=0}^H \gamma^t r_t^i, \end{aligned}$$

Sum over dataset

- **Marginalized Importance Sampling**

(see Nachum et al. 2019 (DualDICE) and Uehara and Jiang, 2019.)

Estimate this ratio

High variance

$$J(\pi_\theta) = \mathbb{E}_{s,a \sim d^\pi(s,a)} [r(s,a)] = \mathbb{E}_{s,a \sim \mathcal{D}} \left[ \frac{d^\pi(s,a)}{\mathcal{D}(s)\mathcal{D}(a|s)} r(s,a) \right]$$

- **Fitted Q-Evaluation**

$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{a' \sim \pi(a'|s')} [Q^\pi(s',a')]$$

A lot of prior work on this!  
OPE has turned out to be  
quite challenging with deep  
network policies.

# How should we evaluate offline RL methods?

**Let's revisit the main motivation for offline RL**

Use real-data collected from various different sources (e.g., human demonstrations, runs of hardcoded policies, etc.) for training good policies

**Can train directly on real data, but how do we test the policy?**

Since testing a policy completely offline is hard (unless we actually run the policy on the real-domain), we would want benchmarks!

we need simulators to actually  
test out our policies

**What properties should a benchmark for offline RL have?**

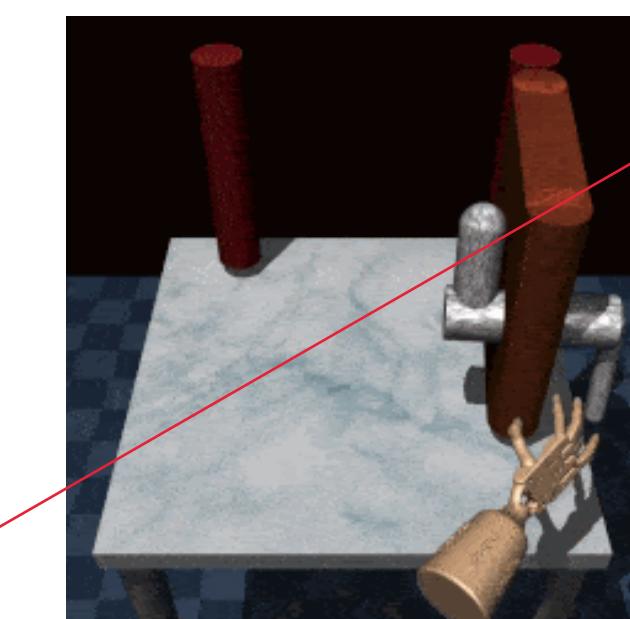
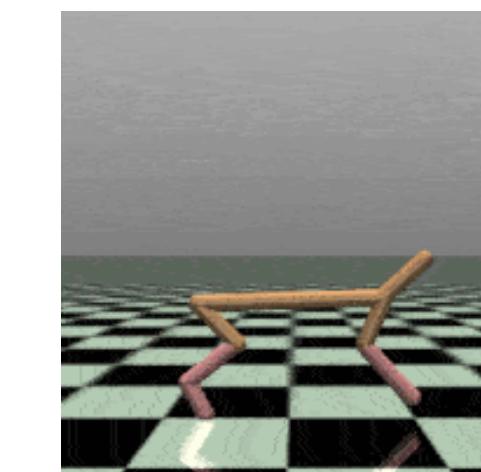
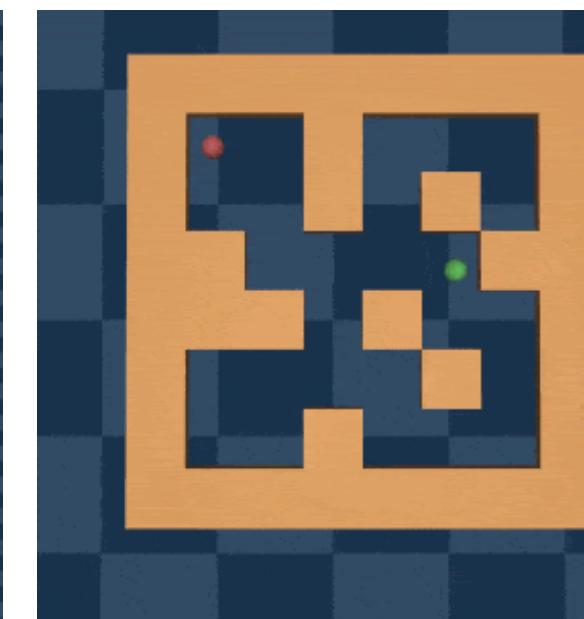
1. It should be realistic: should mimic what we would see in the real-world
2. Should provide a method to compare methods in a standardized way, under the actual evaluation scheme

# Standardized Benchmark for Offline RL

Most evaluation so far has used RL policies or replay buffers, which tend to be substantially easier and different from “real-world” scenarios

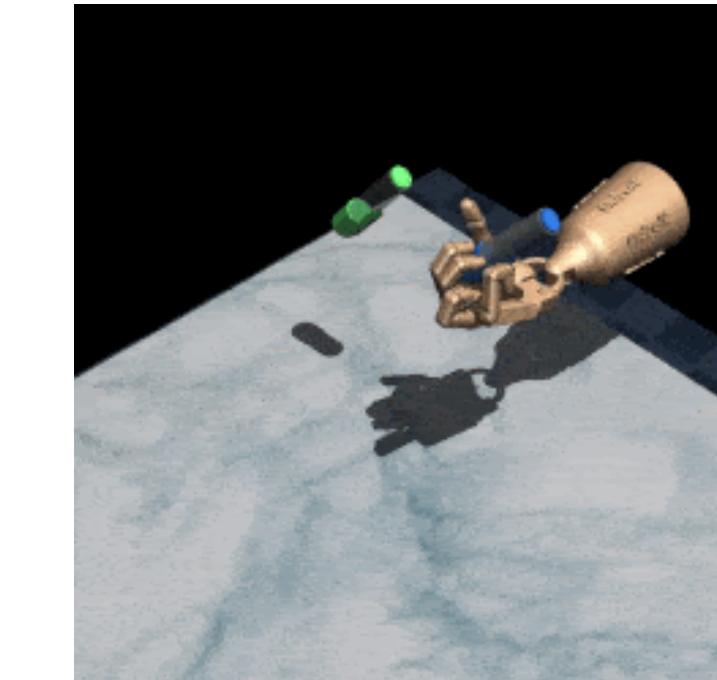
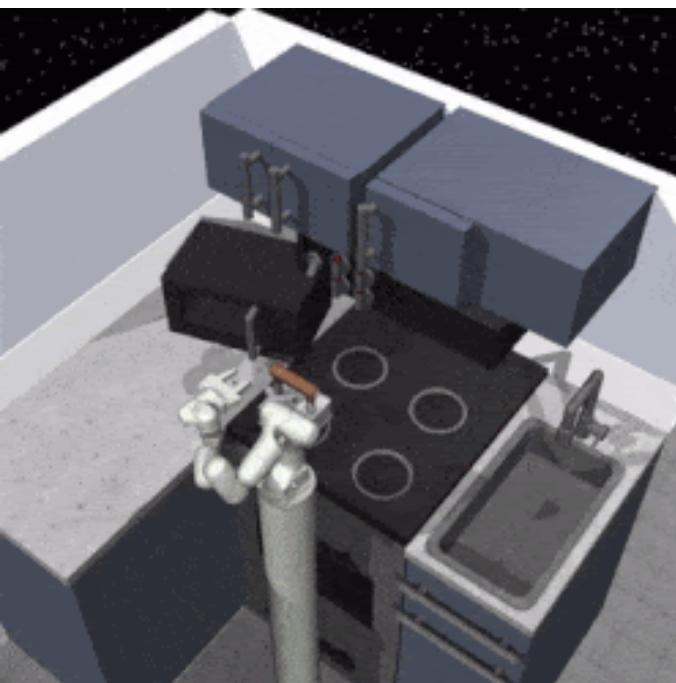
## D4RL benchmark

some data in the dataset aren't even relevant for the goal we are trying to achieve



the data has small coverage over the entire state-action space in the real world

Properties: **(1)** non-representable behavior policies **(2)** narrow distributions **(3)** undirected/multi-task behavior **(4)** visual perception **(5)** human demos.



# **Does Offline RL Work in Practice?**

# Offline RL for Dialog

Can we learn effective dialog policies that understand the implicit human preferences in dialog via offline RL?

[User]: hey, what's up?

[Batch Q]: where did you get??, that sounds so exciting. how do you work for you...  
you want to talk to me?... you want

[User]: yes, i want to talk to you!

[Batch Q]: where did you say to me?... you want to talk to you about you...  
you are so good for you... you are so good...

[User]: thanks, i appreciate that.

[Batch Q]: where did you get??, that sounds so exciting. how do you work for you...  
you want to see you...

[User]: hey, what's up?

[KL-control]: i'm not sure, but i'm glad to see you, and  
i'm so happy that you're not alone

[User]: oh thanks, that's very nice of you!

[KL-control]: i'm sorry, but i'm glad to see what you think.

[User]: don't be sorry, you did a good job.

[KL-control]: that's really nice to me.

Reward function	Quality	Fluent	Diverse	Related	Empathy	Total	Votes	Human reward
Conv. len.	2.20 ± .40	3.61 ± .53	3.02 ± .52	2.25 ± .46	2.48 ± .45	13.57 ± 1.84	-.035	-.003
Semantic sim.	1.93 ± .34	3.50 ± .45	2.37 ± .45	2.11 ± .45	2.52 ± .48	12.43 ± 1.75	-.020	.012
User laughter	1.96 ± .38	3.56 ± .48	2.33 ± .51	1.93 ± .42	3.20 ± .55	12.98 ± 1.60	-.149	-.003
Words elicited	2.11 ± .32	3.96 ± .44	3.04 ± .45	2.04 ± .35	2.55 ± .46	13.70 ± 1.44	.059	.024
Manual votes	2.14 ± .38	3.47 ± .45	2.91 ± .47	2.07 ± .39	2.42 ± .46	13.00 ± 1.65	-.030	.010
Sent. trans.	2.02 ± .31	3.71 ± .49	2.98 ± .50	2.04 ± .42	2.84 ± .48	13.60 ± 1.63	.031	.014
Question	2.29 ± .37	<b>4.31 ± .50</b>	<b>3.31 ± .52</b>	2.20 ± .40	2.60 ± .41	14.71 ± 1.63	.057	.012
Sentiment	<b>2.47 ± .32</b>	4.05 ± .45	3.23 ± .46	<b>2.42 ± .39</b>	<b>3.23 ± .55</b>	<b>15.40 ± 1.49</b>	<b>.085</b>	<b>.045</b>

# Offline RL from Unlabelled Robotic Data

Can we learn effective policies from *unlabelled/general-purpose* robotic data generated from hardcoded policies via offline RL methods such as CQL?

**Input:** Datasets  $\mathcal{D}_{\text{prior}}$  (with no reward annotations),  $\mathcal{D}_{\mathbb{T}}$  (with sparse rewards for task  $\mathbb{T}$ ).

**Return:** Policy  $\pi$  trained to execute task  $\mathbb{T}$ , which should be able to generalize broadly to new initial conditions. We would like to leverage  $\mathcal{D}_{\text{prior}}$  for the latter.



Task & Initial Condition	No prior data	BC			SAC	Ours
		init	all	oracle		
place in box						
object in gripper	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	0.00	1.00 (0.00)
object in tray	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.21 (0.04)	—	0.87 (0.03)
grasp from drawer						
open drawer	0.82 (0.11)	0.93 (0.02)	0.74 (0.03)	0.78 (0.01)	0.0	0.84 (0.07)
closed drawer	0.00 (0.00)	0.79 (0.02)	0.74 (0.03)	0.77 (0.01)	—	0.84 (0.03)
blocked drawer 1	0.00 (0.00)	0.00 (0.00)	0.42 (0.00)	0.65 (0.05)	—	0.86 (0.02)
blocked drawer 2	0.00 (0.00)	0.00 (0.00)	0.36 (0.01)	0.32 (0.18)	—	0.73 (0.08)

# Suggested Readings

- **Summary/ Tutorial:** Levine, Kumar, Tucker, Fu (2020). Offline Reinforcement Learning: Tutorial, Survey and Perspectives on Open Problems.
- **Datasets/Benchmarks:**
  - Fu, Kumar, Nachum, Tucker, Levine (2020). D4RL: Datasets for Deep Data-Driven RL.
  - Gulcehre et al. (2020). RL Unplugged: Benchmarks for Offline RL.
- **Algorithms:**
  - **Classic algorithms and policy constraints:** see tutorial (Levine et al. 2020) and references on prior slides (a lot of work has been done in this area).
  - **Conservative Q-Learning Algorithms:** Kumar, Zhou, Tucker, Levine (2020). Conservative Q-Learning for Offline RL.
  - **Model-based algorithms:**
    - Yu et al. (2020). MOPO: Model-based Offline Policy Optimization.
    - Kidambi et al. (2020). MOReL: Model-based Offline Reinforcement Learning.
  - **Offline RL on Atari:** Agarwal et al. (2020). An Optimistic Perspective on Offline RL.
  - Several new papers on arXiv and OpenReview, check them out!
- **Blog Posts (Summaries):**
  - Kumar. Data-Driven Deep Reinforcement Learning. BAIR blog, December 2019.
  - Agarwal and Norouzi. An Optimistic Perspective on Offline Reinforcement Learning. Google AI Blog, April 2020.