Offline Reinforcement Learning

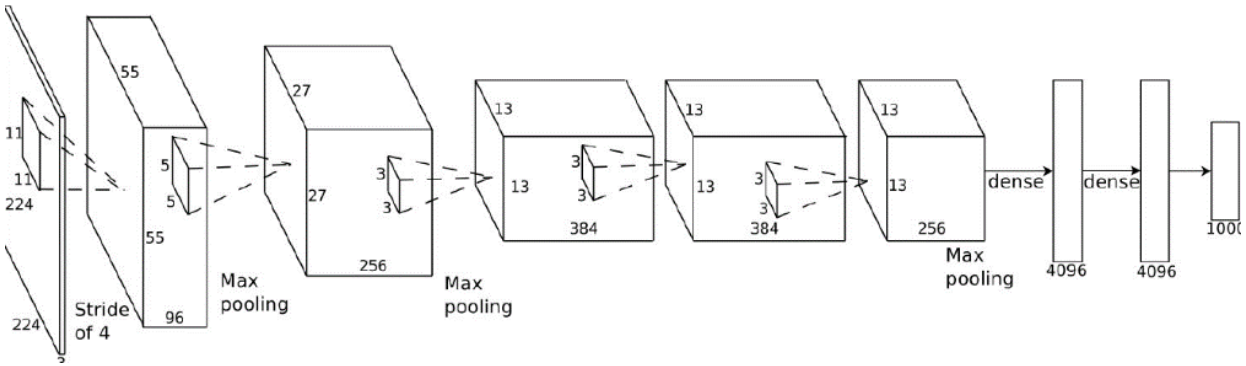
From Algorithms to Practical Challenges

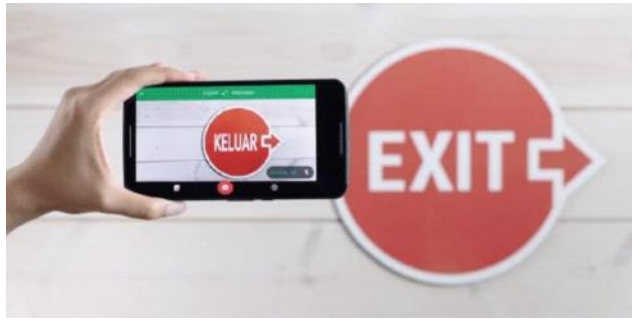
NeurIPS 2020 Tutorial

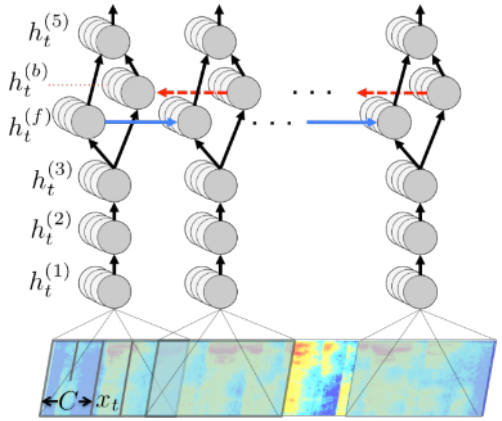
Aviral Kumar **Sergey Levine**

UC Berkeley

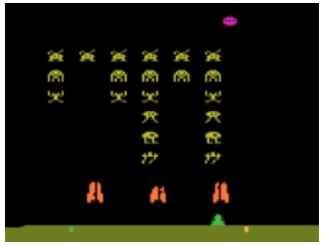
What makes modern machine learning work?



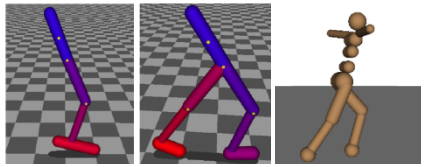




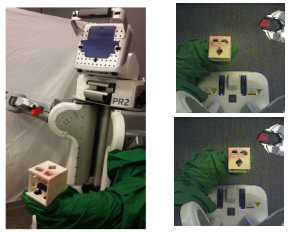
What about reinforcement learning?



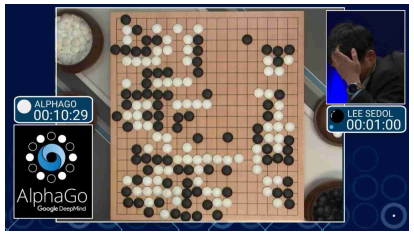
Mnih et al. ‘13

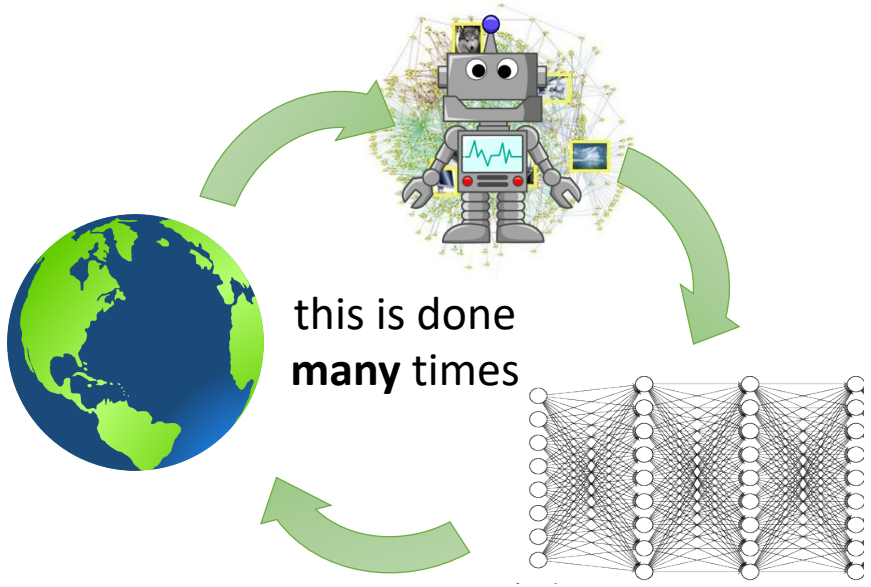


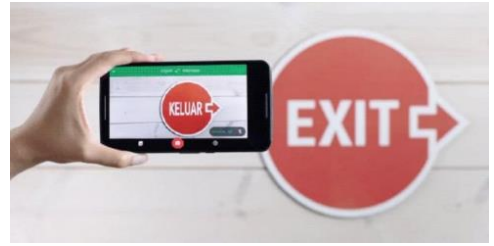
Schulman et al. ’14 & ‘15

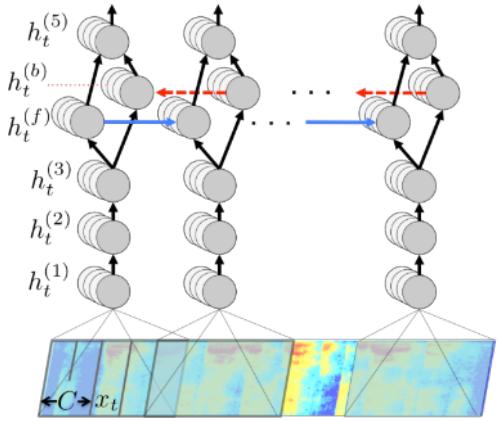


Levine*, Finn*, et al. ‘16

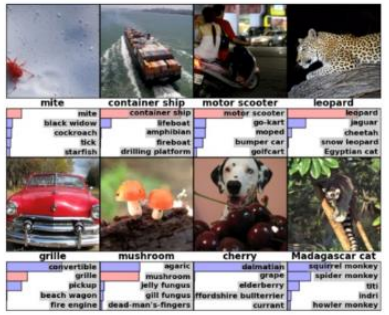








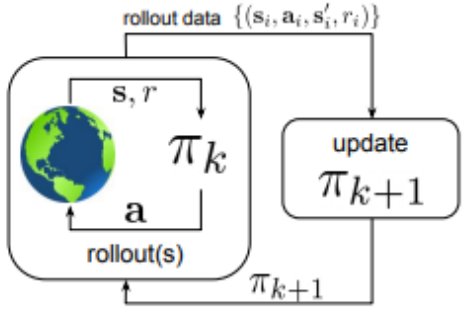
enormous gulf



Can we develop **data-driven** RL methods?

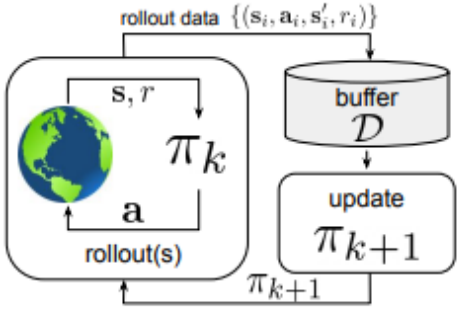
policy interacts with env and updates the policy, then collects new data

on-policy RL

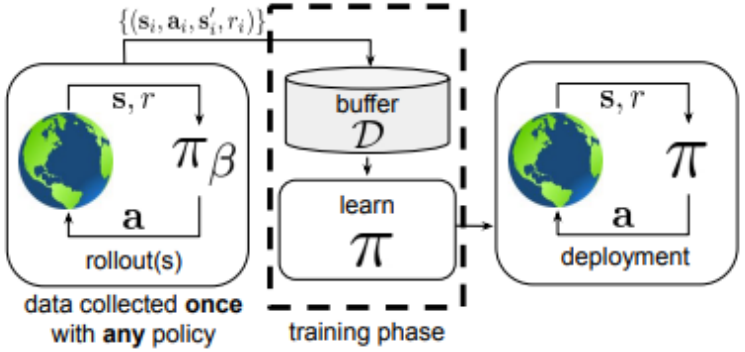


can use data collected by other (usually previous) policies. Still interacts with the world.

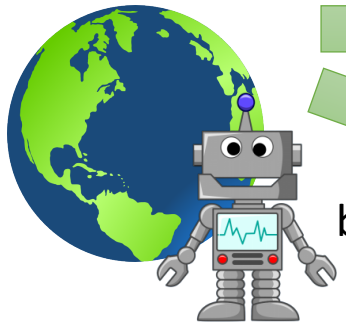
off-policy RL



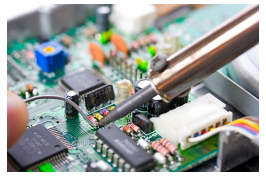
offline reinforcement learning



new data is not added. Behavior policy adds data to the buffer. We don't know what the policy is



big datasets from past interaction



train for **many** epochs

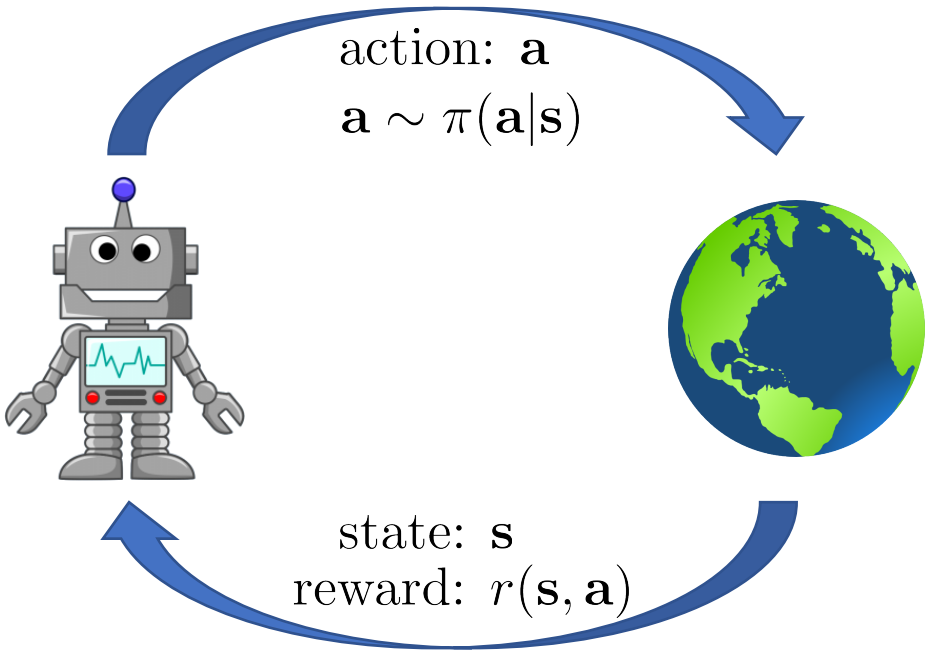
occasionally get more data

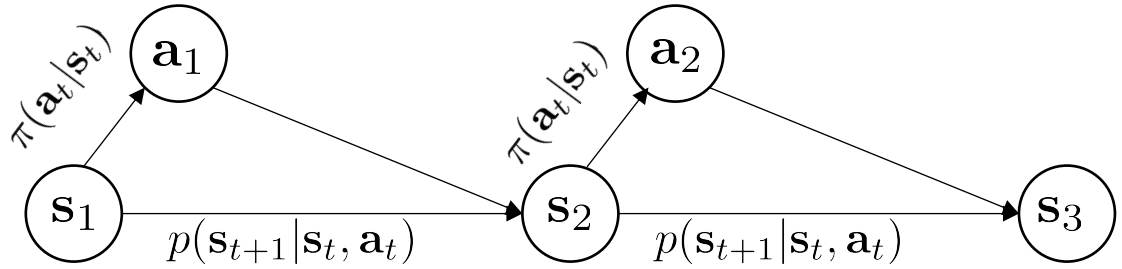
Levine, Kumar, Tucker, Fu. **Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems.** '20

1. Reinforcement learning primer
2. The offline reinforcement learning problem
3. Classic batch reinforcement learning algorithms
    i. Importance-sampled policy gradients
    ii. Least-squares temporal difference (LSTD)
4. Modern offline reinforcement learning algorithms
    i. Policy constraint methods
    ii. Model-based offline RL
    iii. Value function regularization
    iv. Uncertainty-based methods
    v. Comparisons and discussion
5. Off-policy evaluation and model selection
    i. Problem statement
    ii. Direct methods and importance sampling methods
    iii. Doubly robust methods
6. How should we evaluate offline RL methods?
    i. When and why does offline RL work?
    ii. How do we benchmark offline RL methods?
7. Conclusions, discussion, and open problems

1. Reinforcement learning primer
2. The offline reinforcement learning problem
3. Classic batch reinforcement learning algorithms
    i. Importance-sampled policy gradients
    ii. Least-squares temporal difference (LSTD)
4. Modern offline reinforcement learning algorithms
    i. Policy constraint methods
    ii. Model-based offline RL
    iii. Value function regularization
    iv. Uncertainty-based methods
    v. Comparisons and discussion
5. Off-policy evaluation and model selection
    i. Problem statement
    ii. Direct methods and importance sampling methods
    iii. Doubly robust methods
6. How should we evaluate offline RL methods?
    i. When and why does offline RL work?
    ii. How do we benchmark offline RL methods?
7. Conclusions, discussion, and open problems

Policies and objectives





could be ∞ — discount factor

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

state distribution under π

some definitions:

$\mathbf{s} \in \mathcal{S}$ – discrete or continuous state

$\mathbf{a} \in \mathcal{A}$ – discrete or continuous action

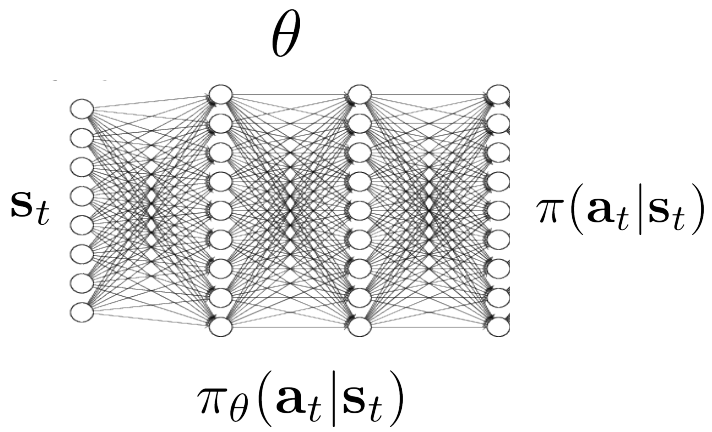
$\tau = \{\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, ..., \mathbf{s}_T, \mathbf{a}_T\}$ – *trajectory*

$$\underbrace{\pi(\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi(\tau)} = p(\mathbf{s}_1) \prod_{t=0}^{T} \pi(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$d_t^{\pi}(\mathbf{s}_t)$ – state marginal of $\pi(\tau)$ at t

$d^{\pi}(\mathbf{s}) = \frac{1}{1-\gamma} \sum_{t=0}^{T} \gamma^t d_t^{\pi}(\mathbf{s}_t)$ – "visitation frequency"

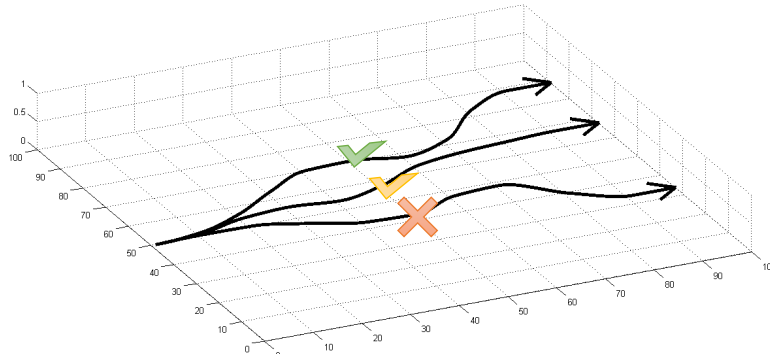
Direct policy gradient



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

↗ exactly the same thing!

$$J(\theta) = E_{\tau \sim \pi_\theta(\tau)}\left[\sum_{t=0}^{T} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T} \gamma^t r(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$$



$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)}\left[\nabla_\theta \log \pi_\theta(\tau) \sum_{t=0}^{T} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right]$$ simple algebraic derivation

$$= E_{\tau \sim \pi_\theta(\tau)}\left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=0}^{T} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$ from definition of τ

$$= E_{\tau \sim \pi_\theta(\tau)}\left[\sum_{t=0}^{T} \nabla_\theta \gamma^t \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

↗

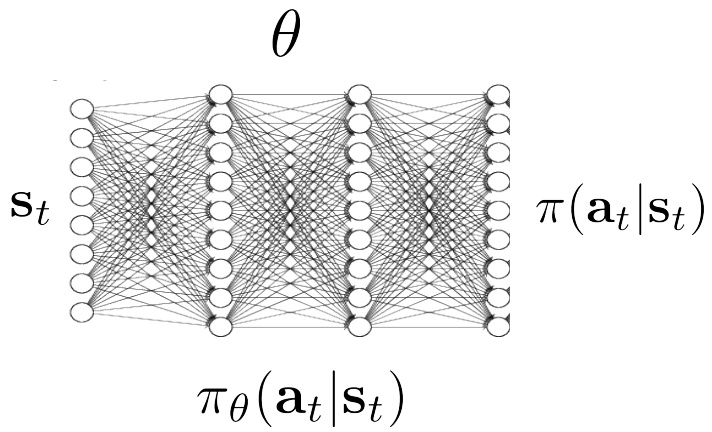
$$= E_{\tau \sim \pi_\theta(\tau)}\left[\sum_{t=0}^{T} \nabla_\theta \gamma^t \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \hat{Q}(\mathbf{s}_t, \mathbf{a}_t)\right]$$

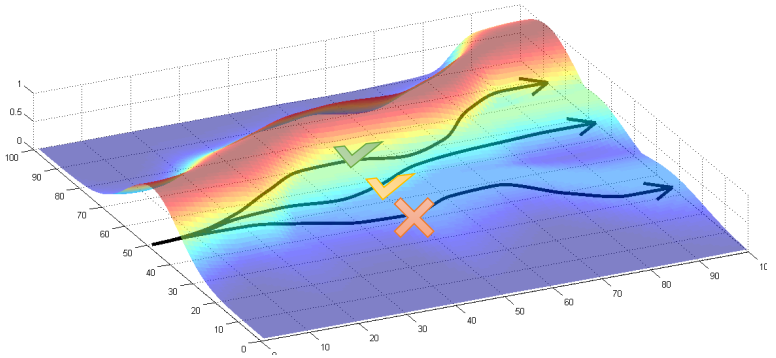
Policy gradient in practice

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

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[\sum_{t=0}^{T} \nabla_\theta \gamma^t \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \hat{Q}(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \nabla_\theta \gamma^t \log \pi_\theta(\mathbf{a}_{t,i}|\mathbf{s}_{t,i}) \hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$$





REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) \right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) \right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Value functions and Q-functions

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}\left[\sum_{t=0}^{T}\nabla_\theta\gamma^t\log\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\hat{Q}(\mathbf{s}_t,\mathbf{a}_t)\right] = \frac{1}{1-\gamma}E_{\mathbf{s}\sim d^{\pi_\theta}(\mathbf{s}),\mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{s})}\left[\nabla_\theta\log\pi_\theta(\mathbf{a}|\mathbf{s})Q^{\pi_\theta}(\mathbf{s},\mathbf{a})\right]$$

constant

???

$$Q^\pi(\mathbf{s}_t,\mathbf{a}_t) = E\left[\sum_{t'=t}^{T}\gamma^{t'-t}r(\mathbf{s}_t,\mathbf{a}_t)\right]$$

$$V^\pi(\mathbf{s}_t) = E\left[\sum_{t'=t}^{T}\gamma^{t'-t}r(\mathbf{s}_t,\mathbf{a}_t)\right]$$

these are really really useful!

Bellman equations:

$$Q^\pi(\mathbf{s}_t,\mathbf{a}_t) = r(\mathbf{s}_t,\mathbf{a}_t) + \gamma E_{\mathbf{s}_{t+1}\sim p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t),\mathbf{a}_{t+1}\sim\pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})}[Q^\pi(\mathbf{s}_{t+1},\mathbf{a}_{t+1})]$$

Basic actor-critic algorithm

1. $Q^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \leftarrow r(\mathbf{s}_t, \mathbf{a}_t) + \gamma E[Q^{\pi_\theta}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})]$
2. $\nabla_\theta J(\theta) = E_{\mathbf{s}\sim d^{\pi_\theta}(\mathbf{s}), \mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{s})}\left[\nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) Q^{\pi_\theta}(\mathbf{s}, \mathbf{a})\right]$
3. $\theta = \theta + \eta \nabla_\theta J(\theta)$

in general:

update Q^{π_θ} to reduce

$E_{(\mathbf{s},\mathbf{a})\sim\pi_\beta(\mathbf{s},\mathbf{a})}\left[\left(Q^{\pi_\theta}(\mathbf{s}, \mathbf{a}) - (r(\mathbf{s}, \mathbf{a}) + \gamma E[Q(\mathbf{s}', \mathbf{a}')])\right)^2\right]$

???

for now: $d^{\pi_\beta}(\mathbf{s}) = d^{\pi_\theta}(\mathbf{s})$

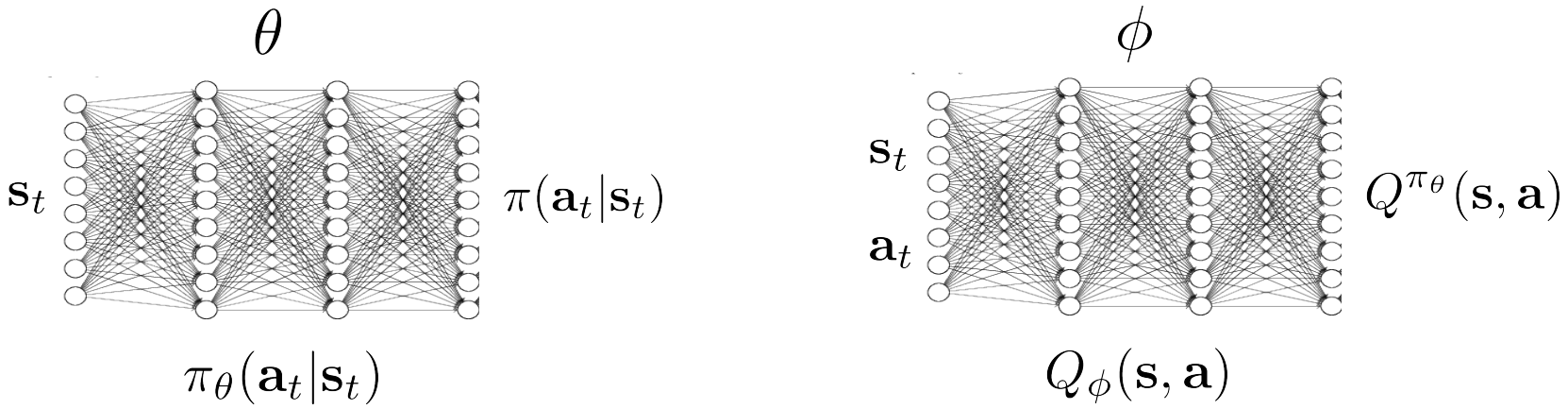
on-policy

in general:

update π_θ to improve

$E_{\mathbf{s}\sim d^{\pi_\theta}(\mathbf{s}), \mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{s})}\left[Q^{\pi_\theta}(\mathbf{s}, \mathbf{a})\right]$

General actor-critic with function approximation



1. update Q_ϕ to decrease $E_{(\mathbf{s},\mathbf{a})\sim\pi_\beta(\mathbf{s},\mathbf{a})}\left[(Q_\phi(\mathbf{s},\mathbf{a}) - (r(\mathbf{s},\mathbf{a}) + \gamma E[Q_\phi(\mathbf{s}',\mathbf{a}')]))^2\right]$
2. update π_θ to increase $E_{\mathbf{s}\sim d^{\pi_\theta}(\mathbf{s}),\mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{s})}\left[Q_\phi(\mathbf{s},\mathbf{a})\right]$

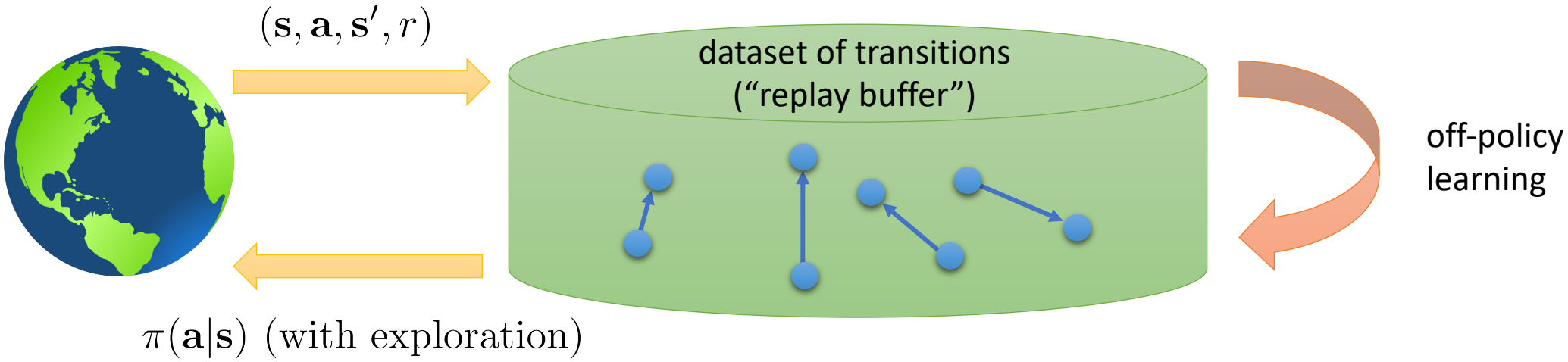
Switching to the *off-policy* setting

1. update Q_ϕ to decrease $E_{(\mathbf{s},\mathbf{a})\sim\pi_\beta(\mathbf{s},\mathbf{a})}\left[(Q_\phi(\mathbf{s},\mathbf{a}) - (r(\mathbf{s},\mathbf{a}) + \gamma E[Q_\phi(\mathbf{s}',\mathbf{a}')]))^2\right]$
2. update π_θ to increase $E_{\mathbf{s}\sim\cancel{d^{\pi_\theta}}(\mathbf{s}),\mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{s})}[Q_\phi(\mathbf{s},\mathbf{a})]$

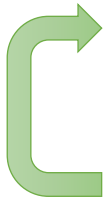
$d^{\pi_\beta}(\mathbf{s})$

is this still correct?

not exactly, but it works well in practice



Removing the actor (Q-learning/fitted Q-iteration)



1. update Q_ϕ to decrease $E_{(\mathbf{s},\mathbf{a})\sim\pi_\beta(\mathbf{s},\mathbf{a})}\left[(Q_\phi(\mathbf{s},\mathbf{a}) - (r(\mathbf{s},\mathbf{a}) + \gamma E[Q_\phi(\mathbf{s}',\mathbf{a}')]))^2\right]$
2. ~~update π_θ to increase $E_{\mathbf{s}\sim d^{\pi_\beta}(\mathbf{s}),\mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{s})}[Q_\phi(\mathbf{s},\mathbf{a})]$~~

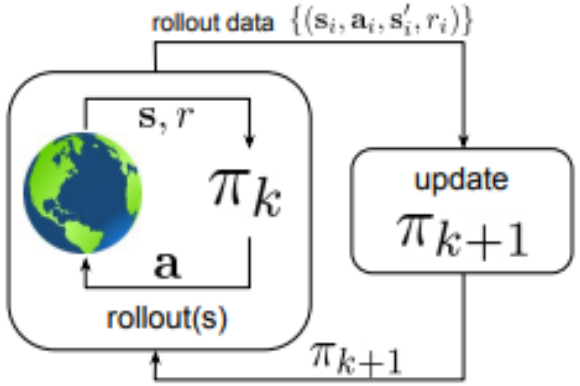
choose π according to: $\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t,\mathbf{a}_t) \\ 0 \text{ otherwise} \end{cases}$

everything we discussed before still applies

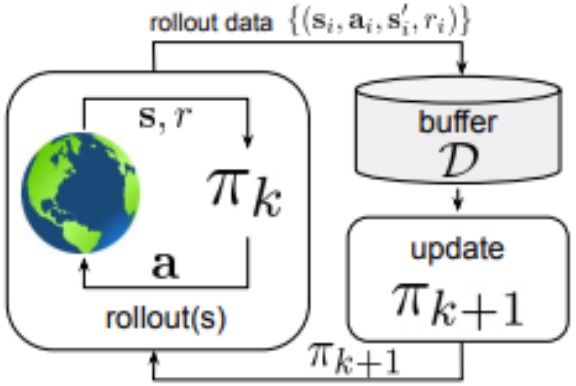
1. Reinforcement learning primer
2. The offline reinforcement learning problem
3. Classic batch reinforcement learning algorithms
    i. Importance-sampled policy gradients
    ii. Least-squares temporal difference (LSTD)
4. Modern offline reinforcement learning algorithms
    i. Policy constraint methods
    ii. Model-based offline RL
    iii. Value function regularization
    iv. Uncertainty-based methods
    v. Comparisons and discussion
5. Off-policy evaluation and model selection
    i. Problem statement
    ii. Direct methods and importance sampling methods
    iii. Doubly robust methods
6. How should we evaluate offline RL methods?
    i. When and why does offline RL work?
    ii. How do we benchmark offline RL methods?
7. Conclusions, discussion, and open problems

What does offline RL mean?

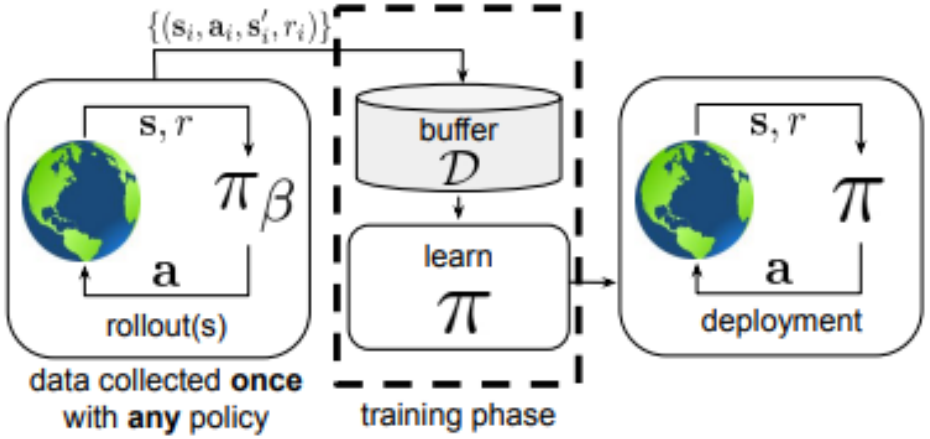
on-policy RL



off-policy RL



offline reinforcement learning



Formally:

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

states are distributed by an unknown distribution d_pi

$$\mathbf{s} \sim d^{\pi_\beta}(\mathbf{s})$$

$$\mathbf{a} \sim \pi_\beta(\mathbf{a}|\mathbf{s})$$

generally **not** known

$$\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$$

$$r \leftarrow r(\mathbf{s}, \mathbf{a})$$

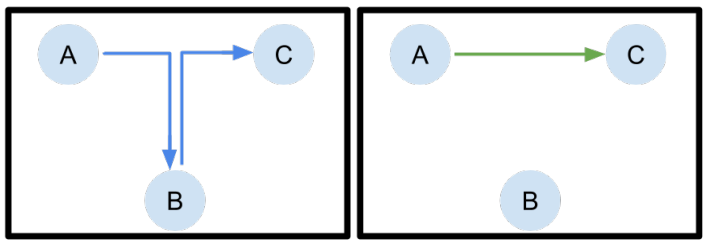
RL objective: $\max_\pi \sum_{t=0}^{T} E_{\mathbf{s}_t \sim d^\pi(\mathbf{s}), \mathbf{a}_t \sim \pi(\mathbf{a}|\mathbf{s})}[\gamma^t r(\mathbf{s}_t, \mathbf{a}_t)]$

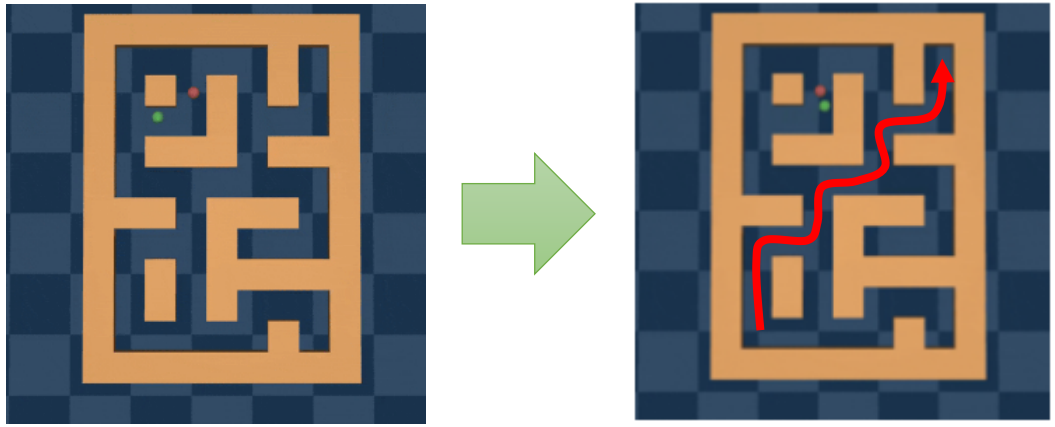
How is this even possible?

1. Find the “good stuff” in a dataset full of good and bad behaviors

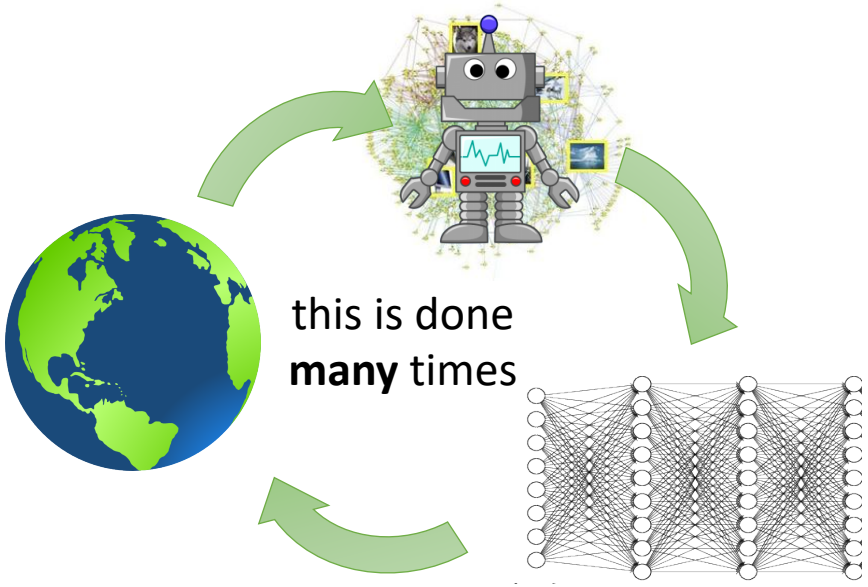
2. Generalization: good behavior in one place may suggest good behavior in another place

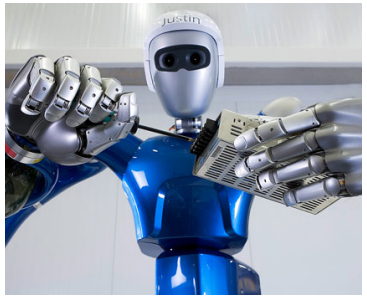
3. “Stitching”: parts of good behaviors can be recombined





Why should we care?





Does it work?

live data collection

stored data from all past experiments

$\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i)\}_i$

training buffers

off-policy $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$

on-policy $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$

labeled $(\mathbf{s}, \mathbf{a}, Q_T(\mathbf{s}, \mathbf{a}))$

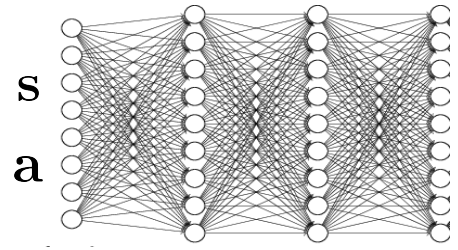
Bellman updaters

compute $Q_T(\mathbf{s}, \mathbf{a}) =$

$r + \max_{\mathbf{a}'} Q_\theta(\mathbf{s}', \mathbf{a}')$

training threads

$\min_\theta ||Q_\theta(\mathbf{s}, \mathbf{a}) - Q_T(\mathbf{s}, \mathbf{a})||^2$



Q_θ

Kalashnikov, Irpan, Pastor, Ibarz, Herzong, Jang, Quillen, Holly, Kalakrishnan, Vanhoucke, Levine. **QT-Opt: Scalable Deep Reinforcement Learning of Vision-Based Robotic Manipulation Skills**

Does it work?

Method	Dataset	Success	Failure
Offline QT-Opt	580k offline	87%	13%
Finetuned QT-Opt	580k offline + 28k online	**96%**	**4%**

Kalashnikov, Irpan, Pastor, Ibarz, Herzong, Jang, Quillen, Holly, Kalakrishnan, Vanhoucke, Levine. **QT-Opt: Scalable Deep Reinforcement Learning of Vision-Based Robotic Manipulation Skills**

Why is offline reinforcement learning hard?

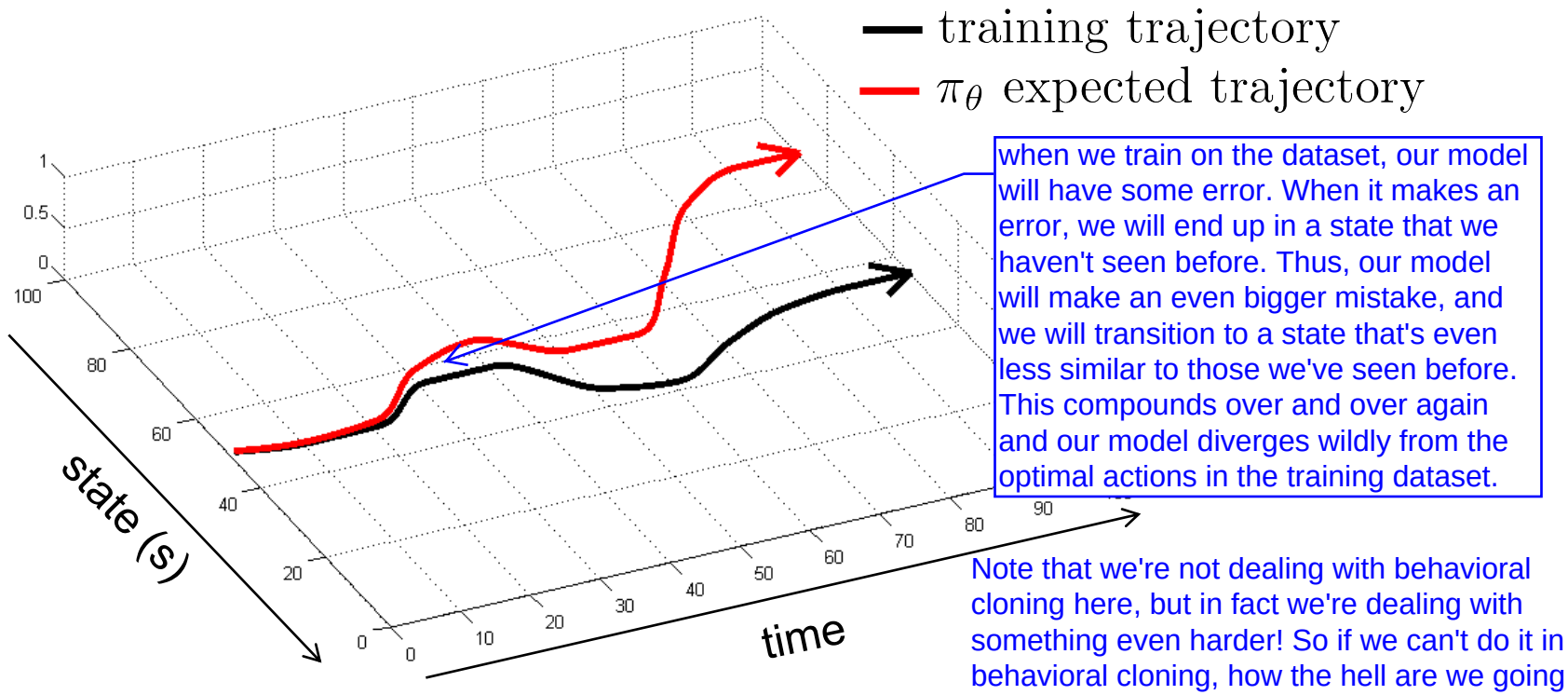
Feedback: when we use a learned model/policy/etc. to act, it changes what we see

Simple example:

behavioral cloning
train π_θ to *copy*
assume data is *optimal*

assume we have a dataset with optimal actions. All we have to do is copy those optimal actions

error scales as $O(T^2)$



when we train on the dataset, our model will have some error. When it makes an error, we will end up in a state that we haven't seen before. Thus, our model will make an even bigger mistake, and we will transition to a state that's even less similar to those we've seen before. This compounds over and over again and our model diverges wildly from the optimal actions in the training dataset.

Note that we're not dealing with behavioral cloning here, but in fact we're dealing with something even harder! So if we can't do it in behavioral cloning, how the hell are we going to do it for offline RL?

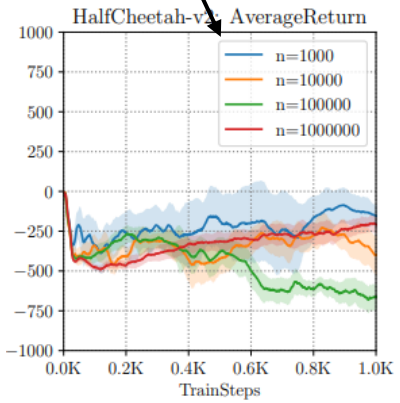
Ross et al. **A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning.** '11

Why is offline reinforcement learning hard?

Hypothesis 1: Overfitting

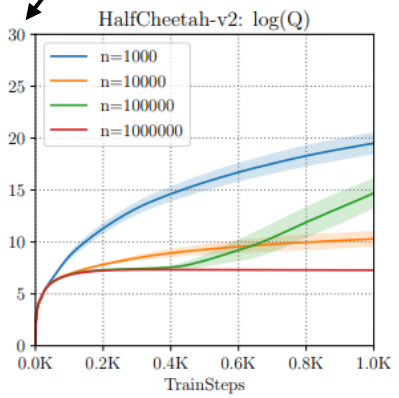
we overfit on the dataset we collect initially and bad stuff happens. They varied the amount of data available to the offline agent, and increasing the amount of data doesn't reduce the problem, so it seems that overfitting is not the issue.

amount of data



how well it does

log scale (massive overestimation)



how well it *thinks* it does (Q-values)

Hypothesis 2: Training data is not good

Usually not the case: behavioral cloning of best data does better!

Kumar, Fu, Tucker, Levine. **Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction.** NeurIPS ‘19

The issue is DISTRIBUTIONAL SHIFT!

Distribution shift in a nutshell

Example empirical risk minimization (ERM) problem:

$$\theta \leftarrow \arg\min_\theta E_{\mathbf{x}\sim p(\mathbf{x}),y\sim p(y|\mathbf{x})}\left[(f_\theta(\mathbf{x})-y)^2\right]$$

given some $\mathbf{x}^\star$, is $f_\theta(\mathbf{x}^\star)$ correct?

$E_{\mathbf{x}\sim p(\mathbf{x}),y\sim p(y|\mathbf{x})}\left[(f_\theta(\mathbf{x})-y)^2\right]$ is low

$E_{\mathbf{x}\sim \bar{p}(\mathbf{x}),y\sim p(y|\mathbf{x})}\left[(f_\theta(\mathbf{x})-y)^2\right]$ is not, for general $\bar{p}(\mathbf{x}) \neq p(\mathbf{x})$

what if $\mathbf{x}^\star \sim p(\mathbf{x})$? not necessarily...

you can't expect the error to be low if x was given by another distribution p_bar(x)

usually we are not worried – neural nets generalize well!

what if we pick $\mathbf{x}^\star \leftarrow \arg\max_{\mathbf{x}} f_\theta(\mathbf{x})$?

Kumar, Fu, Tucker, Levine. **Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction.** NeurIPS '19

Where do we suffer from distribution shift?

q-learning setting

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

$Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + E_{\mathbf{a}' \sim \pi_{\text{new}}}[Q(\mathbf{s}',\mathbf{a}')]$

$y(\mathbf{s},\mathbf{a})$

actor-critic setting

what is the objective?

$\min_Q E_{(\mathbf{s},\mathbf{a}) \sim \pi_\beta(\mathbf{s},\mathbf{a})} \left[(Q(\mathbf{s},\mathbf{a}) - y(\mathbf{s},\mathbf{a}))^2 \right]$

behavior policy

target value

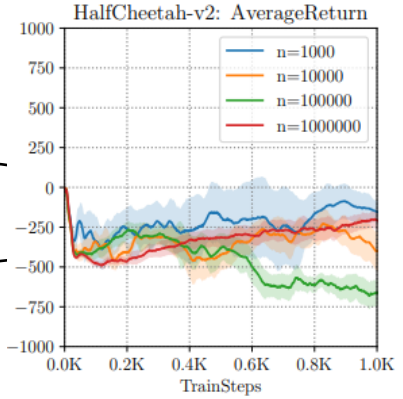
expect good accuracy when $\pi_\beta(\mathbf{a}|\mathbf{s}) = \pi_{\text{new}}(\mathbf{a}|\mathbf{s})$

how often does *that* happen?

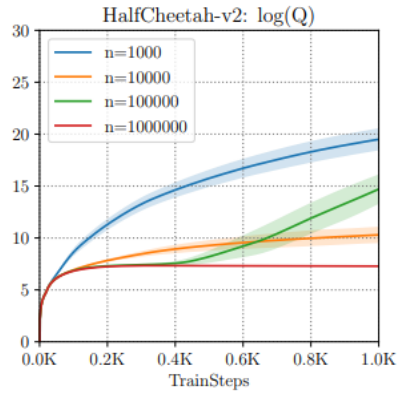
even worse we pick pi_new that maximizes the Q-function

even *worse*: $\pi_{\text{new}} = \arg\max_\pi E_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})]$

(what if we pick $\mathbf{x}^\star \leftarrow \arg\max_{\mathbf{x}} f_\theta(\mathbf{x})$?)



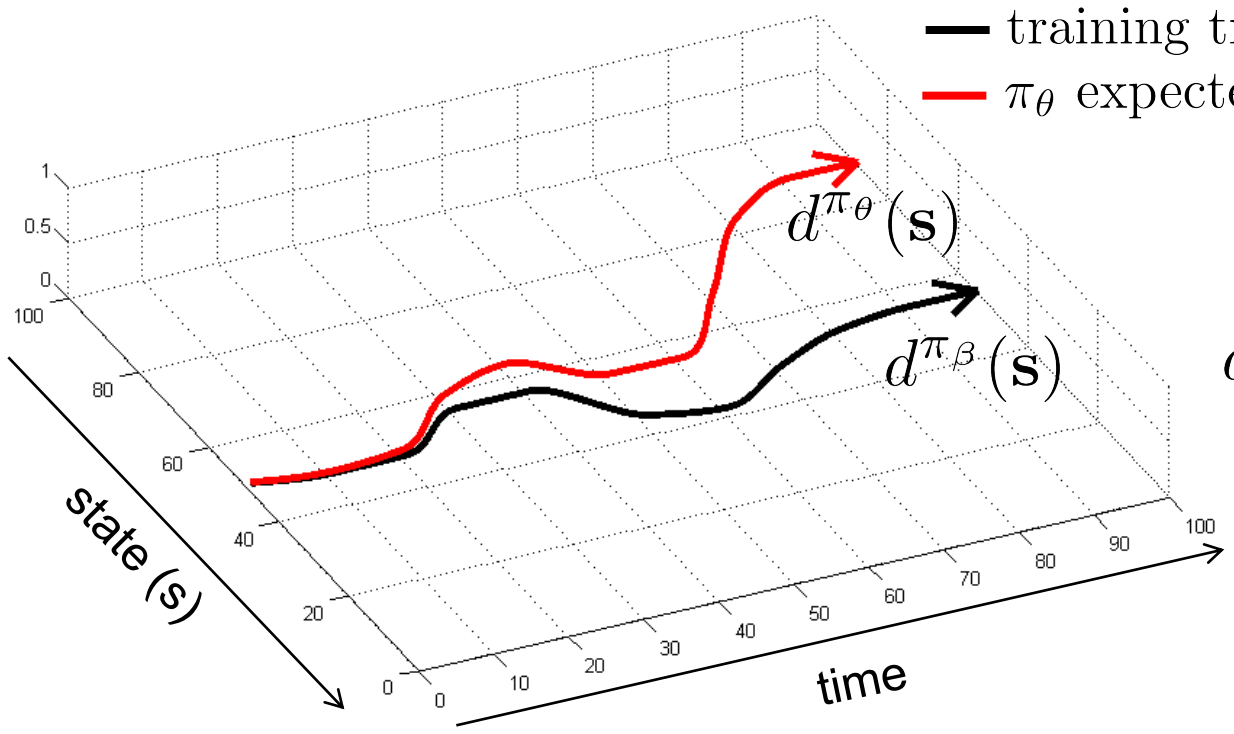
how well it does



how well it *thinks* it does (Q-values)

Kumar, Fu, Tucker, Levine. **Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction.** NeurIPS '19

Where **else** do we suffer from distribution shift?

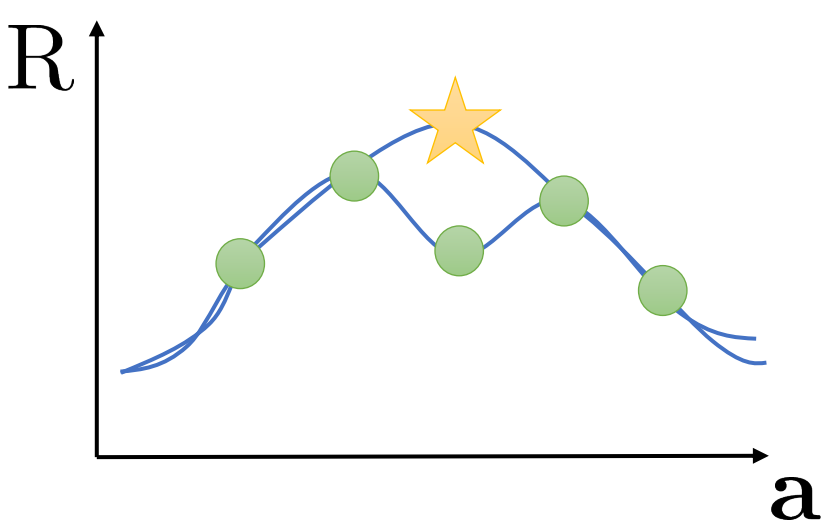


$$d^{\pi_\beta}(\mathbf{s}) \neq d^{\pi_\theta}(\mathbf{s})$$

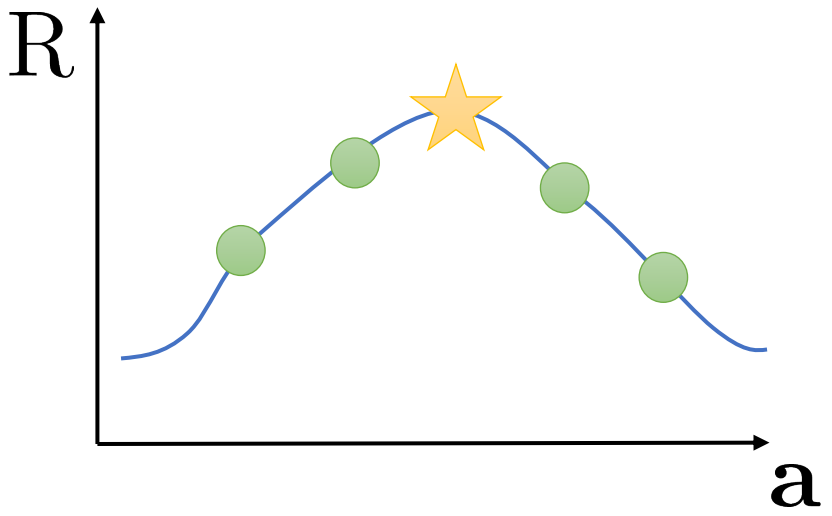
Even if π_θ is *great* on training data, it might be bad when used for control!

Sampling & function approximation error

online RL setting



offline RL setting



Existing challenges with sampling error and function approximation error in standard RL become **much more severe** in offline RL

1. Reinforcement learning primer
2. The offline reinforcement learning problem
3. Classic batch reinforcement learning algorithms
    i. Importance-sampled policy gradients
    ii. Least-squares temporal difference (LSTD)
4. Modern offline reinforcement learning algorithms
    i. Policy constraint methods
    ii. Model-based offline RL
    iii. Value function regularization
    iv. Uncertainty-based methods
    v. Comparisons and discussion
5. Off-policy evaluation and model selection
    i. Problem statement
    ii. Direct methods and importance sampling methods
    iii. Doubly robust methods
6. How should we evaluate offline RL methods?
    i. When and why does offline RL work?
    ii. How do we benchmark offline RL methods?
7. Conclusions, discussion, and open problems

Offline RL with policy gradients

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

Trick when we want the expected value under one distribution but we have samples from another: Importance Sampling

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[\sum_{t=0}^{T} \nabla_\theta \gamma^t \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \hat{Q}(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \nabla_\theta \gamma^t \log \pi_\theta(\mathbf{a}_{t,i}|\mathbf{s}_{t,i}) \hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$$

requires sampling from π_θ!

what if we only have samples from π_β?

importance sampling

importance sampling:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \underbrace{\frac{\pi_\theta(\tau_i)}{\pi_\beta(\tau_i)}}_{\text{importance weight}} \sum_{t=0}^{T} \nabla_\theta \gamma^t \log \pi_\theta(\mathbf{a}_{t,i}|\mathbf{s}_{t,i}) \hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$$

high variance in practice

this is just a sum over future rewards

Offline RL with policy gradients

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\frac{\pi_\theta(\tau_i)}{\pi_\beta(\tau_i)}\sum_{t=0}^{T}\nabla_\theta\gamma^t \log\pi_\theta(\mathbf{a}_{t,i}|\mathbf{s}_{t,i})\hat{Q}(\mathbf{s}_{t,i},\mathbf{a}_{t,i}) \leftarrow E_{\pi_\theta}\left[\sum_{t'=t}^{T}\gamma^{t'-t}r_{t'}\right] \approx \sum_{t'=t}^{T}\gamma^{t'-t}r_{t',i}$$

$$\frac{\pi_\theta(\tau)}{\pi_\beta(\tau)} = \frac{\cancel{p(\mathbf{s}_1)\prod_t p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\cancel{p(\mathbf{s}_1)\prod_t p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}\pi_\beta(\mathbf{a}_t|\mathbf{s}_t)}$$

we need to know pi_beta

this is exponential in T

weights likely to be degenerate as T becomes large

can we fix this?

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}\underbrace{\cancel{\left(\prod_{t'=0}^{t-1}\frac{\pi_\theta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})}{\pi_\beta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})}\right)}}\nabla_\theta\gamma^t \log\pi_\theta(\mathbf{a}_{t,i}|\mathbf{s}_{t,i})\underbrace{\left(\prod_{t'=t}^{T}\frac{\pi_\theta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})}{\pi_\beta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})}\right)}\hat{Q}(\mathbf{s}_{t,i},\mathbf{a}_{t,i})$$

accounts for difference in probability of landing in $\mathbf{s}_{t,i}$

we have $\mathbf{s}_t \sim d^{\pi_\beta}(\mathbf{s}_t)$, but want $\mathbf{s}_t \sim d^{\pi_\theta}(\mathbf{s}_t)$

why is this a reasonable approximation?

if pi_theta lands you in similar states as pi_beta, you can do alright by removing this. In math you can prove this

accounts for having the incorrect $\hat{Q}(\mathbf{s}_{t,i},\mathbf{a}_{t,i})$

corrects for the fact that Q_hat is an estimate of pi_beta not pi_theta

Estimating the returns

recall that we want q_hat to be an estimate of pi_theta, not pi_beta. So we keep the importance sampling weights here.

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}\nabla_\theta \gamma^t \log \pi_\theta(\mathbf{a}_{t,i}|\mathbf{s}_{t,i}) \underbrace{\left(\prod_{t'=t}^{T}\frac{\pi_\theta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})}{\pi_\beta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})}\right)\hat{Q}(\mathbf{s}_{t,i},\mathbf{a}_{t,i})}$$

$$\leftarrow E_{\pi_\theta}\left[\sum_{t'=t}^{T}\gamma^{t'-t}r_{t'}\right] \approx \sum_{t'=t}^{T}\gamma^{t'-t}r_{t',i}$$

$$\sum_{t'=t}^{T}\left(\prod_{t''=t}^{\cancel{T}\,t'}\frac{\pi_\theta(\mathbf{a}_{t'',i}|\mathbf{s}_{t'',i})}{\pi_\beta(\mathbf{a}_{t'',i}|\mathbf{s}_{t'',i})}\right)\gamma^{t'-t}r_{t',i}$$

but this is *still* exponential!

To avoid exponentially exploding importance weights, we must use value function estimation!

imagine we knew $Q^{\pi_\theta}(\mathbf{s},\mathbf{a})$

We'll see how to do this shortly, but first let's conclude our discussion of importance sampling

The doubly robust estimator

$$V^{\pi_\theta}(\mathbf{s}_0) \approx \sum_{t=0}^{T} \left(\prod_{t'=0}^{t'} \frac{\pi_\theta(\mathbf{a}_{t'}, \mathbf{s}_{t'})}{\pi_\beta(\mathbf{a}_{t'}, \mathbf{s}_{t'})} \right) \gamma^t r_t \gamma^{t'-t} r_{t',i}$$

$$= \sum_{t=0}^{T} \left(\prod_{t'=0}^{t} \rho_{t'} \right) \gamma^t r_t$$

$$= \rho_0 r_0 + \rho_0 \gamma \rho_1 r_1 + \rho_0 \gamma \rho_1 \gamma \rho_2 r_2 + \ldots$$

$$= \rho_0 (r_0 + \gamma(\rho_1 (r_1 + \gamma(\rho_2 (r_2 + \gamma \ldots)))))$$

$$= \bar{V}^T \qquad \text{where } \bar{V}^{T+1-t} = \rho_t (r_t + \gamma \bar{V}^{T-t})$$

this is exponential!

doubly robust estimation (bandit case)

$$V_{\mathrm{DR}}(s) = \hat{V}(s) + \rho(s,a)(r_{s,a} - \hat{Q}(s,a))$$

model or function approximator

$$\bar{V}_{\mathrm{DR}}^{T+1-t} = \hat{V}(\mathbf{s}_t) + \rho_t (r_t + \gamma \bar{V}_{\mathrm{DR}}^{T-t} - \hat{Q}(\mathbf{s}_t, \mathbf{a}_t))$$

model or function approximator

Jiang, N. and Li, L. (2015). Doubly robust off-policy value evaluation for reinforcement learning.

Marginalized importance sampling

Main idea: instead of using $\prod_t \frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\beta(\mathbf{a}_t|\mathbf{s}_t)}$, estimate $w(\mathbf{s},\mathbf{a}) = \frac{d^{\pi_\theta}(\mathbf{s},\mathbf{a})}{d^{\pi_\beta}(\mathbf{s},\mathbf{a})}$

if we can do this, we can estimate $J(\theta) \approx \frac{1}{N}\sum_i w(\mathbf{s}_i,\mathbf{a}_i) r_i$

typically this is done for off-policy evaluation, rather than policy learning

how to determine $w(\mathbf{s},\mathbf{a})$? typically solve some kind of consistency condition

example (Zhang et al., GenDICE):

$$d^{\pi_\beta}(\mathbf{s}',\mathbf{a}')w(\mathbf{s}',\mathbf{a}') = (1-\gamma)\underbrace{p_0(\mathbf{s}')\pi_\theta(\mathbf{a}'|\mathbf{s}')}_{\text{probability of starting in } (\mathbf{s}',\mathbf{a}')} + \gamma \sum_{\mathbf{s},\mathbf{a}} \underbrace{\pi_\theta(\mathbf{a}'|\mathbf{s}')p(\mathbf{s}'|\mathbf{s},\mathbf{a})d^{\pi_\beta}(\mathbf{s},\mathbf{a})}_{\text{probability of transitioning into } (\mathbf{s}',\mathbf{a}')} w(\mathbf{s},\mathbf{a})$$

solving for $w(\mathbf{s},\mathbf{a})$ typically involves some fixed point problem

Additional readings: importance sampling

Classic work on importance sampled policy gradients and return estimation:
Precup, D. (2000). Eligibility traces for off-policy policy evaluation.
Peshkin, L. and Shelton, C. R. (2002). Learning from scarce experience.

Doubly robust estimators and other improved importance-sampling estimators:
Jiang, N. and Li, L. (2015). Doubly robust off-policy value evaluation for reinforcement learning.
Thomas, P. and Brunskill, E. (2016). Data-efficient off-policy policy evaluation for reinforcement learning.

Analysis and theory:
Thomas, P. S., Theocharous, G., and Ghavamzadeh, M. (2015). High-confidence off-policy evaluation.

Marginalized importance sampling:
Hallak, A. and Mannor, S. (2017). Consistent on-line off-policy evaluation.
Liu, Y., Swaminathan, A., Agarwal, A., and Brunskill, E. (2019). Off-policy policy gradient with state distribution correction.

Additional readings in our offline RL survey: Section 3.1, 3.2, 3.3, 3.4: https://arxiv.org/abs/2005.01643

Offline value function estimation

How have people thought about it before?

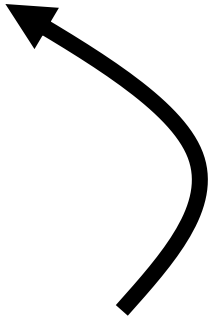
Extend existing ideas for approximate dynamic programming and Q-learning to offline setting

Derive tractable solutions with simple (e.g., linear) function approximators

How are people thinking about it now?

Derive approximate solutions with highly expressive function approximators (e.g., deep nets)

The primary challenge turns out to be **distributional shift**



generally not concerned with distributional shift before

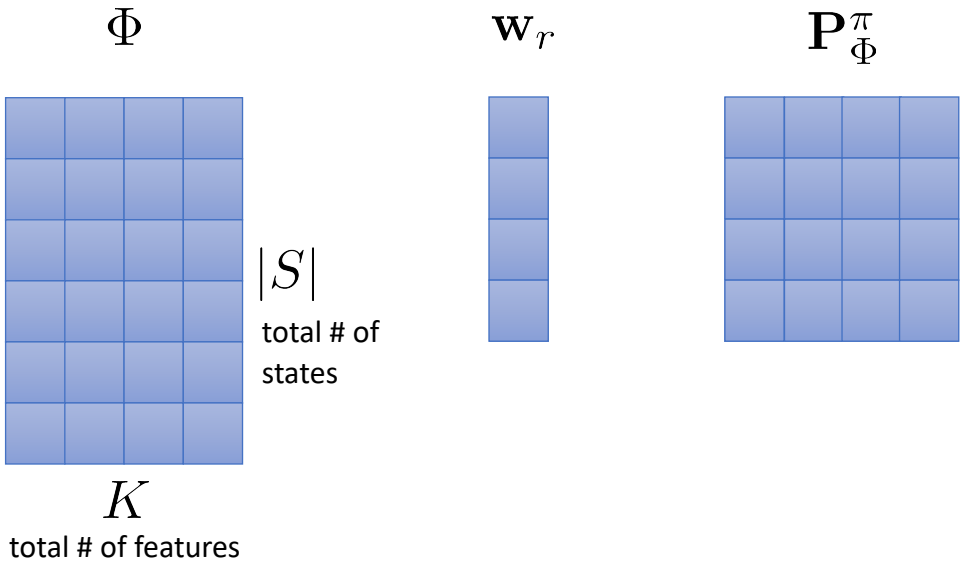
(maybe it was not such a big problem with linear models)

We'll discuss some older offline/batch RL methods next for completeness

Warmup: linear models

Φ – feature matrix, $|S| \times K$

could also think of as a vector-valued function $\Phi(\mathbf{s})$



Can we do (offline) model-based RL in feature space?

1. Estimate the reward
2. Estimate the transitions
3. Recover the value function
4. Improve the policy

vector of rewards for all state-action tuples
but we'll talk about sample-based setting soon!

1. Reward model: $\Phi \mathbf{w}_r \approx r$ least squares: $\mathbf{w}_r = (\Phi^T \Phi)^{-1} \Phi^T \vec{\mathbf{r}}$
2. Transition model: $\Phi \mathbf{P}_\Phi \approx \mathbf{P}^\pi \Phi$ least squares: $\mathbf{P}_\Phi = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{P}^\pi \Phi$

estimated feature-space transition matrix
$K \times K$

real transition matrix (on states)
$|S| \times |S|$

all of this is for a *fixed* policy π

material adapted from Ron Parr

Recovering the value function

1. Reward model: $\Phi \mathbf{w}_r \approx r$

   least squares: $\mathbf{w}_r = (\Phi^T \Phi)^{-1} \Phi \vec{\mathbf{r}}$

2. Transition model: $\Phi \mathbf{P}_\Phi \approx \mathbf{P}^\pi \Phi$

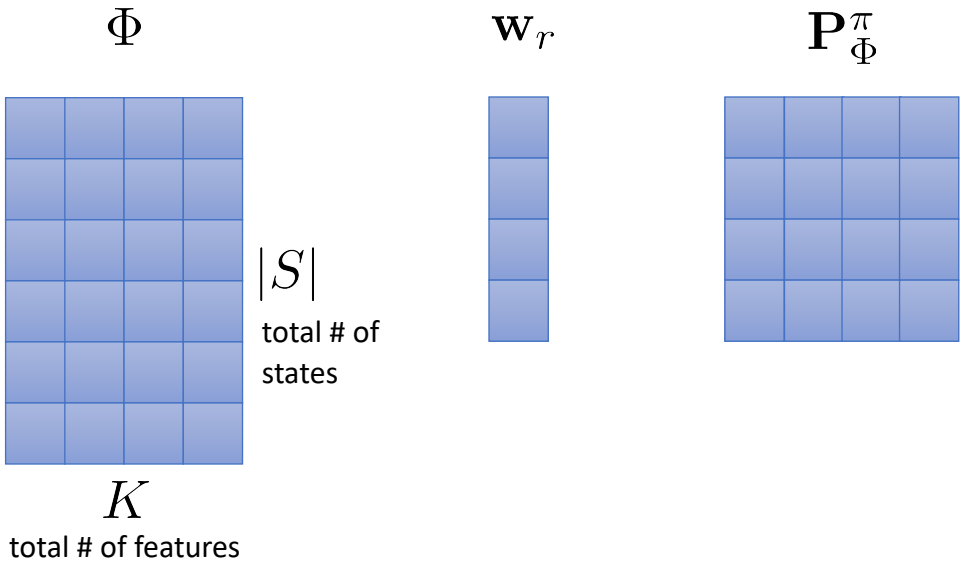
   least squares: $\mathbf{P}_\Phi = (\Phi^T \Phi)^{-1} \Phi \mathbf{P}^\pi \Phi$

3. Estimate $V^\pi \approx V^\pi_\Phi = \Phi \mathbf{w}_V$

   can apply the same equation in feature space:

   $\mathbf{w}_V = (\mathbf{I} - \gamma \mathbf{P}_\Phi)^{-1} \mathbf{w}_r$

substitute



but wait – do we even *need* the model?

$$\mathbf{w}_V = (\mathbf{I} - \gamma(\Phi^T \Phi)^{-1} \Phi^T \mathbf{P}^\pi \Phi)^{-1} (\Phi^T \Phi)^{-1} \Phi^T \vec{\mathbf{r}}$$

after a bit of algebra…

$$\mathbf{w}_V = (\Phi^T \Phi - \gamma \Phi^T \mathbf{P}^\pi \Phi)^{-1} \Phi^T \vec{\mathbf{r}}$$

this is called least-squares temporal difference (LSTD)

Aside: solving for V^π in terms of $\mathbf{P}^\pi$ and $\mathbf{r}$:

$$V^\pi = \mathbf{r} + \gamma \mathbf{P}^\pi V^\pi$$
$$(\mathbf{I} - \gamma \mathbf{P}^\pi) V^\pi = \mathbf{r}$$
$$V^\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1} \mathbf{r}$$

material adapted from Ron Parr

Doing it all with samples

$$\mathbf{w}_V = (\Phi^T\Phi - \gamma\Phi^T\mathbf{P}^\pi\Phi)^{-1}\Phi^T\vec{\mathbf{r}}$$

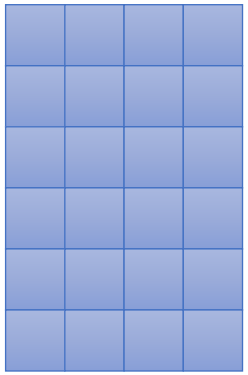
$$\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$$

$$\vec{\mathbf{r}}_i = r(\mathbf{s}_i, \mathbf{a}_i)$$

replace with Φ'

$$\Phi'_i = \phi(\mathbf{s}'_i)$$

Φ $\mathbf{w}_r$ $\mathbf{P}^\pi_\Phi$



$|\mathcal{D}|$

total # of samples

K

total # of features

$$\Phi_i = \phi(\mathbf{s}_i)$$

Everything else works **exactly** the same way, only now we have some sampling error

material adapted from Ron Parr

Improving the policy

1. Estimate the reward
2. Estimate the transitions
3. Recover the value function

} or just do these together with LSTD!

4. Improve the policy

typical policy improvement step:

$\pi'(\mathbf{s}) \leftarrow \text{Greedy}(\Phi \mathbf{w}_V)$

estimate $V^{\pi'}$

That's not going to work for offline RL!

$$\mathbf{w}_V = (\Phi^T \Phi - \gamma \Phi^T \mathbf{P}^\pi \Phi)^{-1} \Phi^T \vec{\mathbf{r}}$$

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

$$\vec{\mathbf{r}}_i = r(\mathbf{s}_i, \mathbf{a}_i)$$

replace with Φ'

$$\Phi'_i = \phi(\mathbf{s}'_i)$$

this requires samples from π!

material adapted from Ron Parr

Least-squares policy iteration (LSPI)

Main idea: replace LSTD with LSTDQ – LSTD but for Q-functions

$$\mathbf{w}_Q = (\Phi^T\Phi - \gamma\Phi^T\Phi')^{-1}\Phi^T\vec{\mathbf{r}}$$

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

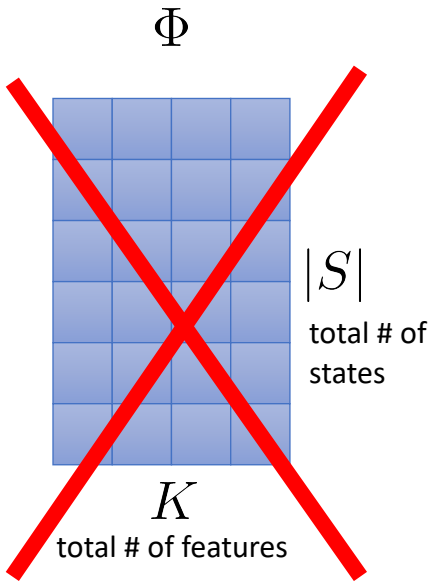
$$\vec{\mathbf{r}}_i = r(\mathbf{s}_i, \mathbf{a}_i)$$

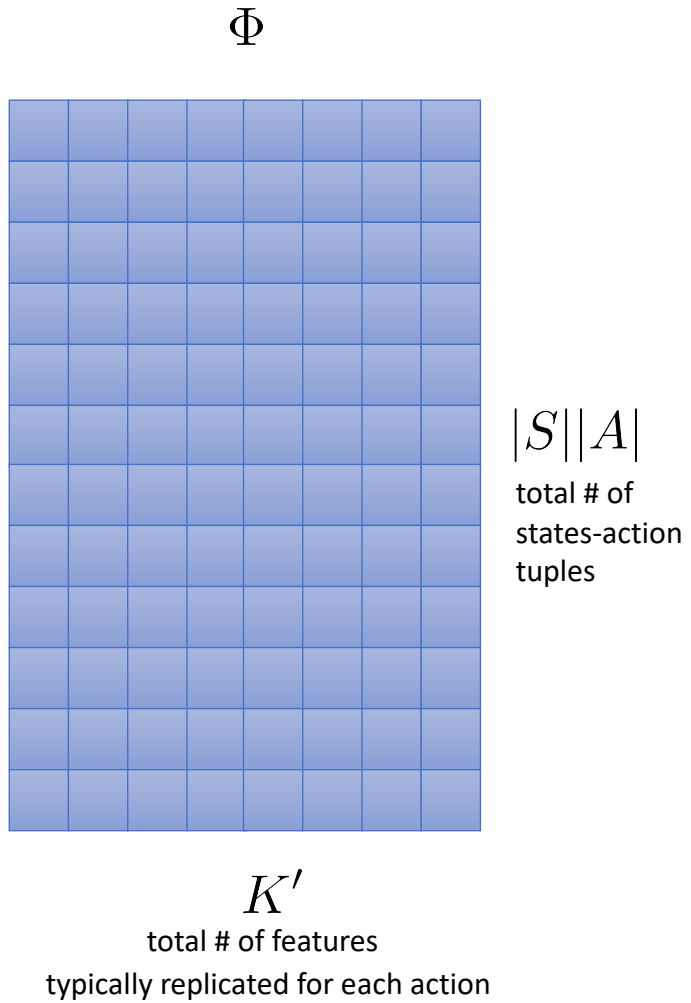
$$\Phi'_i = \phi(\mathbf{s}'_i, \pi(\mathbf{s}'_i))$$

encode the action π would take not the action in the data

LSPI:

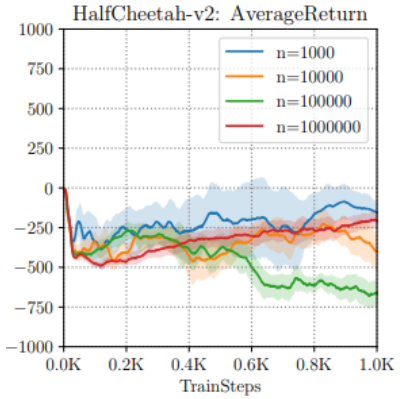
1. compute $\mathbf{w}_Q$ for π_k
2. $\pi_{k+1}(\mathbf{s}) = \arg\max_{\mathbf{a}} \phi(\mathbf{s}, \mathbf{a})\mathbf{w}_Q$
3. Set $\Phi'_i = \phi(\mathbf{s}'_i, \pi_{k+1}(\mathbf{s}'_i))$



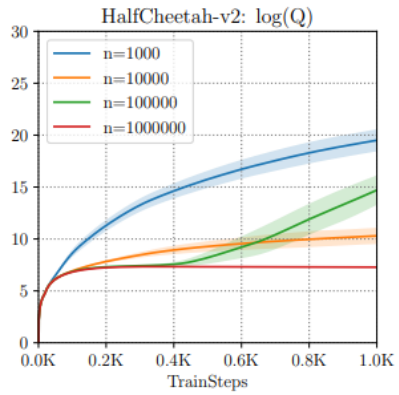


material adapted from Ron Parr

What's the issue?



how well it does



how well it *thinks* it does (Q-values)

In general, all approximate dynamic programming (e.g., fitted value/Q iteration) methods will suffer from action distributional shift, and we **must** fix it!

$$Q(\mathbf{s},\mathbf{a}) \leftarrow \underbrace{r(\mathbf{s},\mathbf{a}) + E_{\mathbf{a}' \sim \pi_{\text{new}}}[Q(\mathbf{s}',\mathbf{a}')]}_{y(\mathbf{s},\mathbf{a})}$$

$$\min_Q E_{(\mathbf{s},\mathbf{a}) \sim \pi_\beta(\mathbf{s},\mathbf{a})} \left[(Q(\mathbf{s},\mathbf{a}) - y(\mathbf{s},\mathbf{a}))^2 \right]$$

behavior policy

target value

expect good accuracy when $\pi_\beta(\mathbf{a}|\mathbf{s}) = \pi_{\text{new}}(\mathbf{a}|\mathbf{s})$

how often does *that* happen?

even *worse*: $\pi_{\text{new}} = \arg\max_\pi E_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})]$

1. Reinforcement learning primer
2. The offline reinforcement learning problem
3. Classic batch reinforcement learning algorithms
    i. Importance-sampled policy gradients
    ii. Least-squares temporal difference (LSTD)
4. Modern offline reinforcement learning algorithms
    i. Policy constraint methods
    ii. Model-based offline RL
    iii. Value function regularization
    iv. Uncertainty-based methods
    v. Comparisons and discussion
5. Off-policy evaluation and model selection
    i. Problem statement
    ii. Direct methods and importance sampling methods
    iii. Doubly robust methods
6. How should we evaluate offline RL methods?
    i. When and why does offline RL work?
    ii. How do we benchmark offline RL methods?
7. Conclusions, discussion, and open problems

Addressing Action Distributional Shift in Offline RL

so we're effectively sampling alphas from the pi_theta distribution

a' can be OOD from pi_beta

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

so when we do the bellman backup, we backup values that are erroneous and not in the dataset, because pi_beta was used to generate the data!

"Policy constraint"

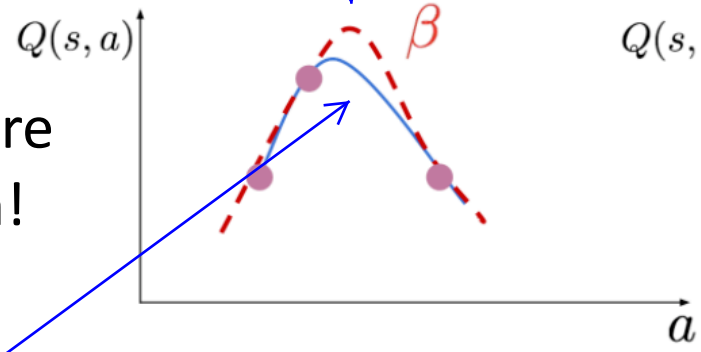
$$\pi_\theta := \arg\max_{\pi_\theta} \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] \quad \text{s.t. } D(\pi_\theta(\mathbf{a}|\mathbf{s}), \pi_\beta(\mathbf{a}|\mathbf{s})) \leq \varepsilon$$

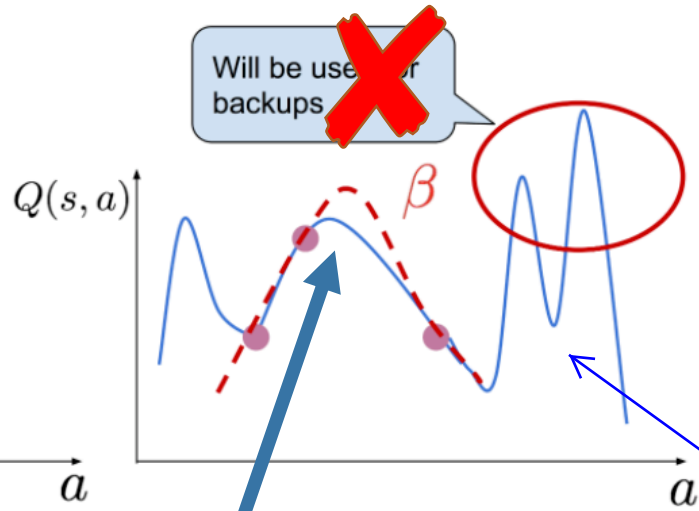
Some notion of divergence, we will discuss some next

Limits OOD actions used for training the Q-function.

Actions used for training are also (almost) trained upon!

red line represents behavior policy





blue represents the Q-function

when we use a policy constraint, we ensure that the chosen action a' is close to the distribution of the behavior policy. This prevents the action distribution shift

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

these actions are OOD for pi_beta. As such, our Q-value is bad at estimating them. Some of our estimates will be erroneously high. But using the bellman backup, we will pick the actions that maximize these!

Kumar, Fu, Tucker, Levine. **Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction.** NeurIPS '19

Different Types of Policy Constraint Methods

$$\pi_\theta := \arg\max_{\pi_\theta} \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] \quad \text{s.t. } D(\pi_\theta(\mathbf{a}|\mathbf{s}),\pi_\beta(\mathbf{a}|\mathbf{s})) \le \varepsilon$$

Distribution constraints

(Fujimoto et al. 2019: action clipping)

(Wu et al. 2019, Jaques et al. 2019)

Several other forms. Different benefits over each other

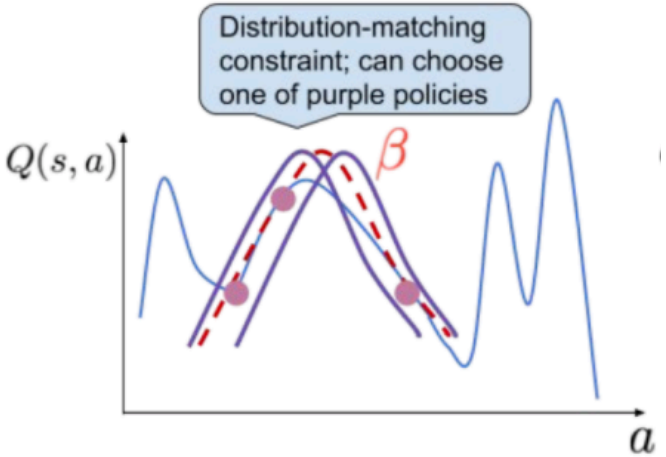
$$D(\pi_\theta,\pi_\beta) = \mathrm{D}_{\mathrm{KL}}(\pi_\theta,\pi_\beta)$$

KL divergence

$$D(\pi_\theta,\pi_\beta) = \mathrm{D}_{\mathrm{f}}(\pi_\theta,\pi_\beta) = \mathbb{E}_{\pi_\beta}\left[f\left(\frac{\pi_\theta}{\pi_\beta}\right)\right]$$

f-divergence

(Wu et al. 2019)



$$\max_{\pi_\theta(\mathbf{a}|\mathbf{s})} \mathbb{E}_{\pi_\theta}[Q(\mathbf{s},\mathbf{a})] - \alpha \mathrm{D}_{\mathrm{KL}}(\pi_\theta(\mathbf{a}|\mathbf{s}),\pi_\beta(\mathbf{a}|\mathbf{s}))$$

(solve in closed form)

$$\pi_\theta(\mathbf{a}|\mathbf{s}) \propto \pi_\beta(\mathbf{a}|\mathbf{s}) \exp(Q(\mathbf{s},\mathbf{a})/\alpha)$$

you can implement this in code

(Peng et al. 2019, Seigel et al. 2019, Wang et al. 2020, Nair et al. 2020)

"Implicit" distributional constraint

Different Types of Policy Constraint Methods

$$\pi_\theta := \arg\max_{\pi_\theta} \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] \quad \text{s.t. } D(\pi_\theta(\mathbf{a}|\mathbf{s}),\pi_\beta(\mathbf{a}|\mathbf{s})) \le \varepsilon$$

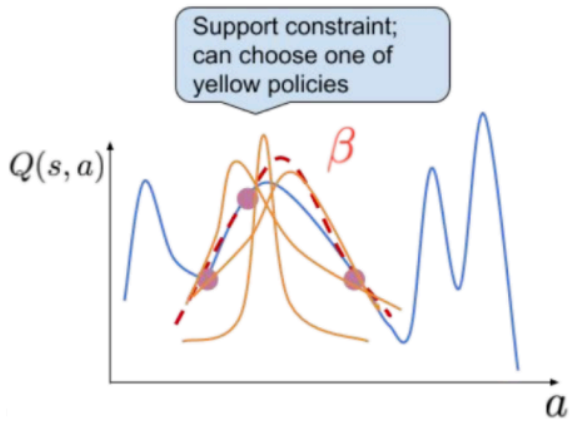
Support constraints

$$D(\pi_\theta,\pi_\beta) = \mathbf{D}(\pi_\theta, \text{supp}(\pi_\beta))$$

constrain the learned policy pi_theta to the support of pi_beta

Can also use other integral probability metrics
(Sriperumbudur et al. 2009)

$$\text{IPM}_\mathcal{F}(\pi_\theta,\pi_\beta) = \max_{f\in\mathcal{F}} \left|\mathbb{E}_{\pi_\theta}[f(\mathbf{s},\mathbf{a})] - \mathbb{E}_{\pi_\beta}[f(\mathbf{s},\mathbf{a})]\right|$$



Discrete Action Spaces

$$D(\pi_\theta,\pi_\beta) = \sum_{\mathbf{a}\notin\mathcal{D}[\mathbf{s}]} \pi_\theta(\mathbf{a}|\mathbf{s})$$

Support at state s

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

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

(Kumar et al. 2019)

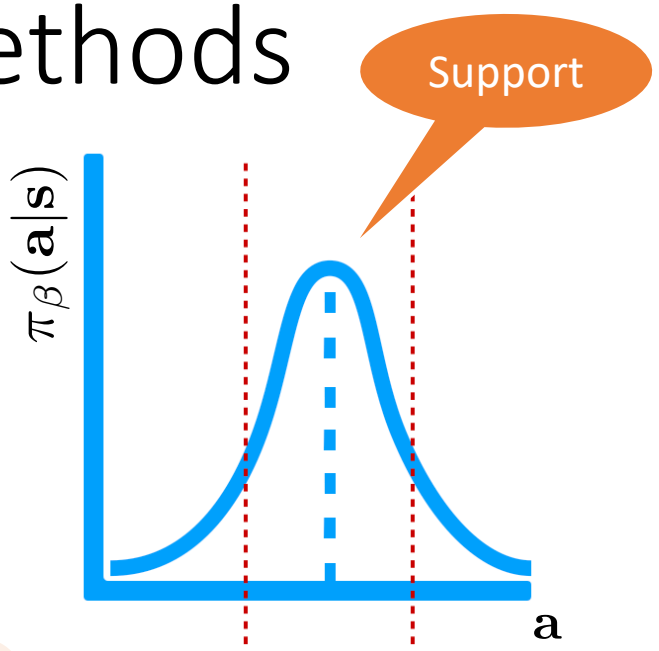
Maximum mean discrepancy

(Gretton et al. 2012)

we can implement this constraint by calculating the MMD

Constrain $\pi_\theta(\mathbf{a}|\mathbf{s}) = \pi_\beta(\mathbf{a}|\mathbf{s})$

on out-of-support actions
(Laroche et al. 2019)



Intuition: compare samples regardless of density

we don't really care about the densities

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

$$\max_{a'\in\mathcal{D}[\mathbf{s}']} Q(\mathbf{s}',\mathbf{a}')$$

the "support" of the behavior policy is the set of all possible actions that are highly likely under the behavior policy

Discrete BCQ (Fujimoto et al. 2019),
EMaQ (Ghasemipour et al. 2020)

restrict the support of the actions that you compute the max over!

Comparison of Policy Constraint Methods

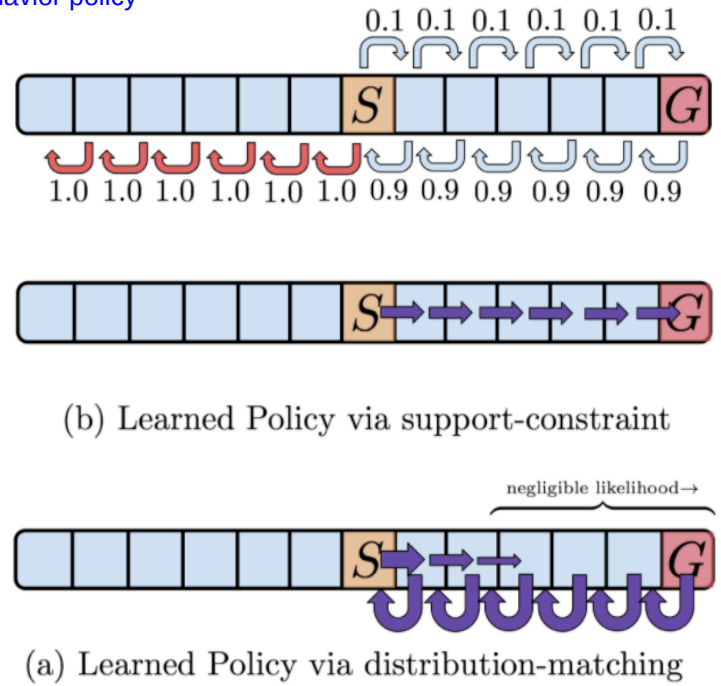
Do different policy constraint methods behave differently?

policy constraint methods alter the optimization function. We're no longer solving the traditional RL objective because we introduce another term penalizing us for deviating too far from the behavior policy

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

Standard RL objective

Conservative behavior



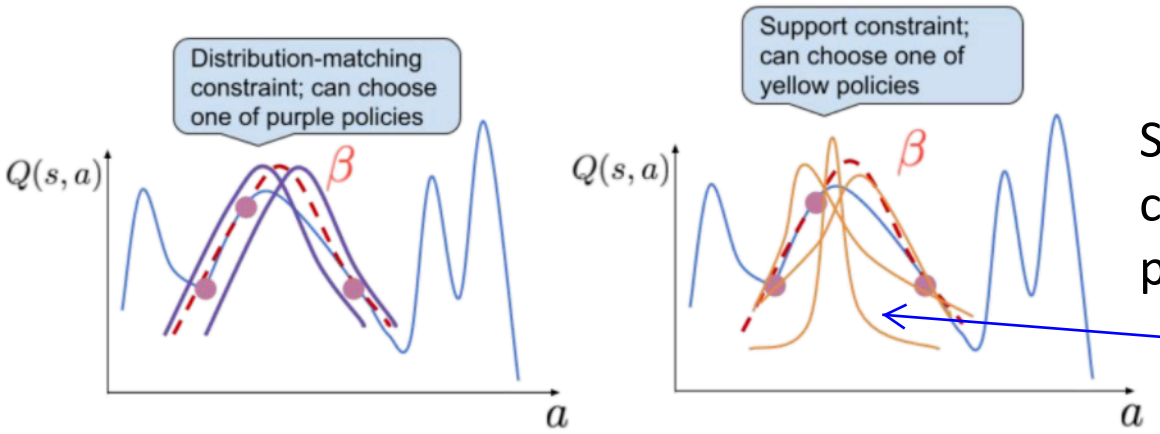
(b) Learned Policy via support-constraint

(a) Learned Policy via distribution-matching

Different constraints alter the behavior of the algorithm differently!

Support vs. distribution constraints

Better in theory, since more flexibility to choose policies



Support constraints still allow convergence to the optimal policy with full coverage

notice that the densities are really different, but they all are within the support of pi_beta

Kumar. **Data-driven Deep Reinforcement Learning.** BAIR Blog, December 2019.

so far we've looked at constraining pi_beta and pi_theta. So we look at pi_theta(a|s) and compare it to pi_beta(a|s)

A different variant of policy constraints

So far, we have discussed constraints on the policy distribution.

Can also define constraints on state-action marginal distributions

$$D(\pi_\theta, \pi_\beta) = \mathrm{D}_{\mathrm{KL}}(d^{\pi}(\mathbf{s}, \mathbf{a}), d^{\pi_\beta}(\mathbf{s}, \mathbf{a}))$$

marginal state-action distribution under pi_beta

Also a "policy constraint" since we technically constrain policies

Primarily explored for off-policy evaluation, but can also be used for offline RL

Lots of work:
"DICE" methods: DualDICE (Nachum et al. 2019), GenDICE (Zhang et al. 2020), AlgaeDICE (Nachum et al. 2019), GradientDICE (Zhang et al. 2020) etc.

Minimax weight learning: Uehera and Jiang, 2019

Since marginals are not computable directly in offline settings

$$\text{estimate } w(\mathbf{s}, \mathbf{a}) = \frac{d^{\pi_\theta}(\mathbf{s},\mathbf{a})}{d^{\pi_\beta}(\mathbf{s},\mathbf{a})}$$

(implicitly use this for RL with a modified objective)

$$d^{\pi_\beta}(\mathbf{s}', \mathbf{a}')w(\mathbf{s}', \mathbf{a}') = (1-\gamma)p_0(\mathbf{s}')\pi_\theta(\mathbf{a}'|\mathbf{s}')+\gamma \sum_{\mathbf{s},\mathbf{a}} \pi_\theta(\mathbf{a}'|\mathbf{s}')p(\mathbf{s}'|\mathbf{s}, \mathbf{a})d^{\pi_\beta}(\mathbf{s}, \mathbf{a})w(\mathbf{s}, \mathbf{a})$$

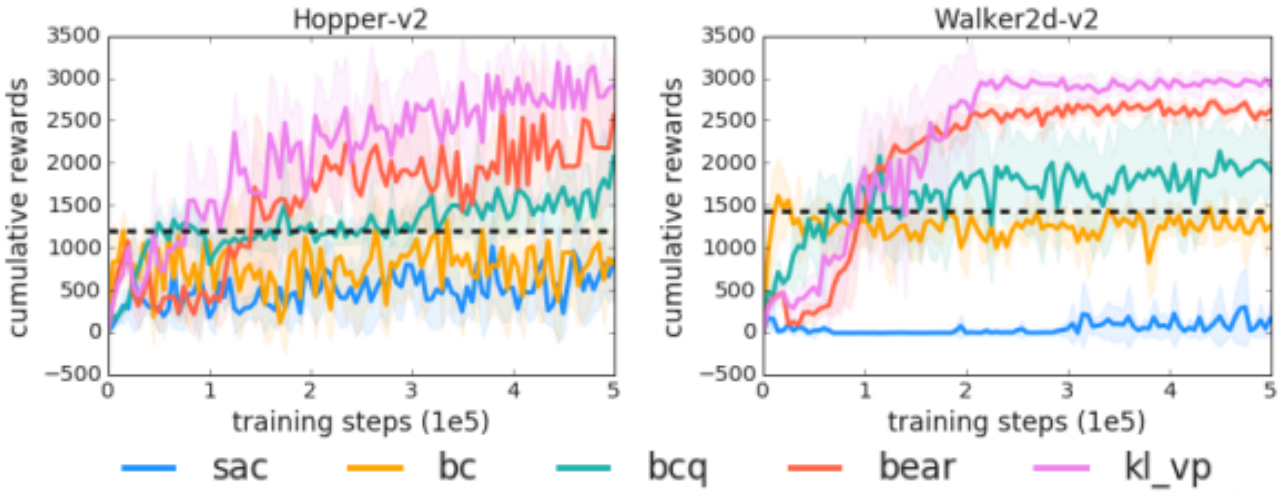
As discussed before…

Nachum and Dai. **Reinforcement Learning via Fenchel-Rockafeller Duality.** '2019

How do policy constraint methods perform?

partially trained RL agent

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



Naïve off-policy RL

Behavioral cloning

Policy constraint methods (BEAR: support, BCQ, BRAC: distribution)

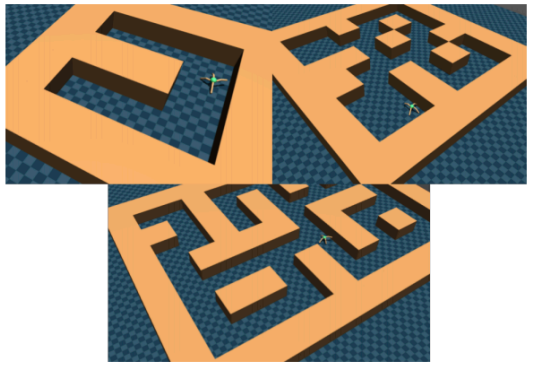
Task Name	BC	SAC	BEAR	BRAC-p
antmaze-umaze	65.0	0.0	**73.0**	50.0
antmaze-umaze-diverse	55.0	0.0	61.0	40.0
antmaze-medium-play	0.0	0.0	0.0	0.0
antmaze-medium-diverse	0.0	0.0	8.0	0.0
antmaze-large-play	0.0	0.0	0.0	0.0
antmaze-large-diverse	0.0	0.0	0.0	0.0

Results from BRAC (Wu et al. 2020) and D4RL (Fu et al. 2020)

Better than BC! (shows offline RL can do better than supervised learning)

Different choices of D matter, though not totally as expected, so other factors at play

But do not solve all tasks, especially harder ones, so work to do!



notice that kl_vp (i.e. BRAC) is a distribution matching algo and performs better than BEAR. Even though in theory, support constraint methods should be better, in practice that's not necessarily the case

Shortcomings of Policy Constraint Methods

Need estimation of the behavior policy

we don't know what this is, we have to estimate it from the dataset

$$\pi_\theta := \arg\max_{\pi_\theta} \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] \quad \text{s.t. } D(\pi_\theta(\mathbf{a}|\mathbf{s}), \pi_\beta(\mathbf{a}|\mathbf{s})) \leq \varepsilon$$

estimated from data

If the behavior policy is wrongly estimated (e.g, when we fit unimodal policy to multimodal data), policy constraint methods can fail dramatically (e.g., AntMaze results from before)

Often tend to be too conservative

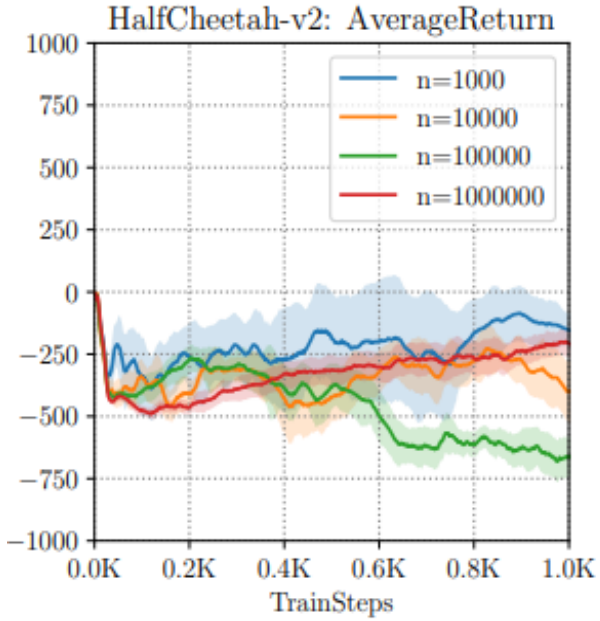
If we know that a certain state has all actions with 0 reward, we do not care about constraining the policy there, since we will not be worse…

Can we do better?

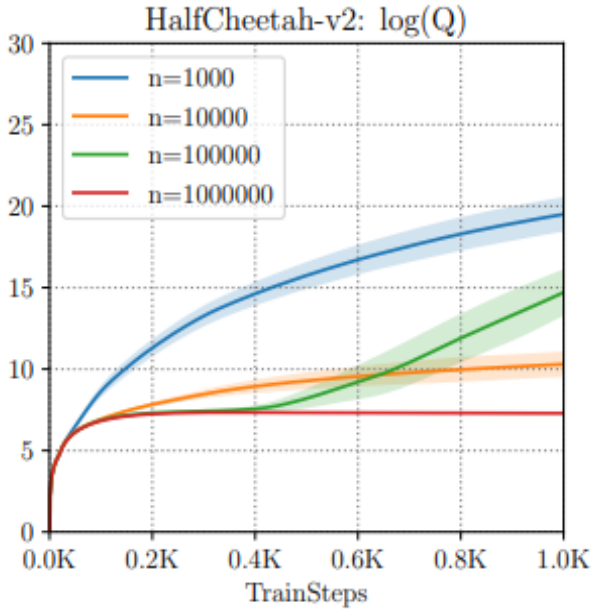
Also see: Nair et al. 2020. **Accelerating Online RL with Offline Datasets.** '2020
and Ghasemipour et al. 2020. **EMaQ: Expected-Max Q-learning Operator for Simple Yet Effective Offline and Online RL.** '2020

Let's revisit the motivating example....

policy constraint methods seek to constrain the actions taken by the policy pi_theta such that they are similar to the actions you'd take using pi_beta. By doing this, they indirectly tackle the overestimation issue.



how well it does



how well it *thinks* it does (Q-values)

The issue is overestimation of the value of unseen, "bad" outcomes, which policy constraint methods aim to solve

But policy constraints "over-do" it: we don't need to constrain policies for some outcomes if the algorithm believes that certain unseen outcomes are bad

Can we directly tackle the overestimation issue?

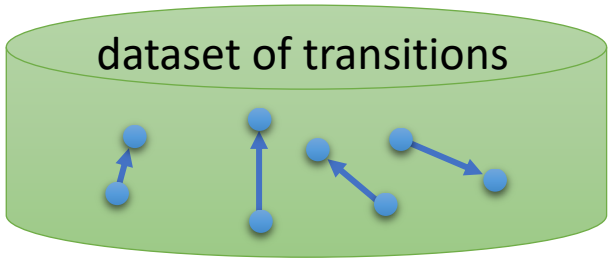
instead of constraining the action distributions, which is not the most optimal way to solve this

"Conservative" Methods

Yes! Model-based and model-free methods

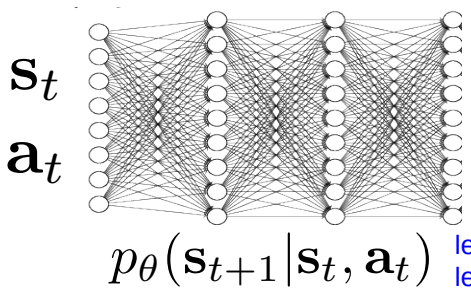
Conservative Model-Based Offline RL

How can we avoid unseen outcomes?



Option 1: Policy constraint (as before) — as we discussed, this tends to be overly conservative

Option 2: Make the learned reward function conservative — essentially, make the reward function take into account OOD or unseen outcomes. Then run typical model-based RL using this

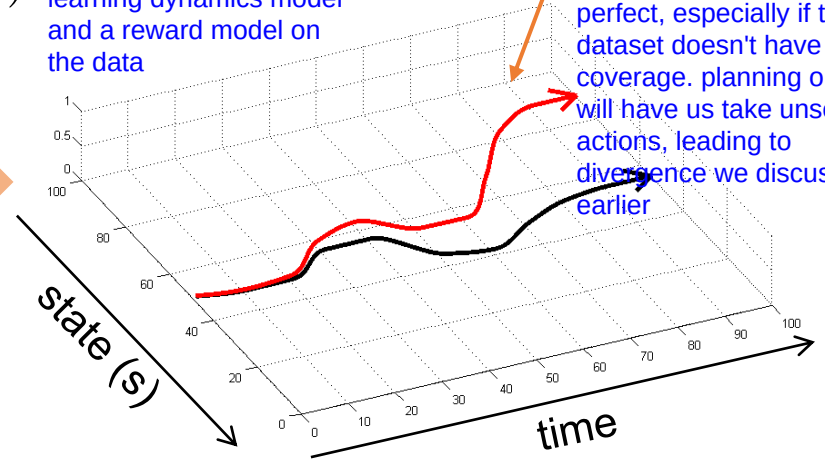


let's train a supervised learning dynamics model and a reward model on the data

Planning/ policy optimization under the model can lead to unseen outcomes

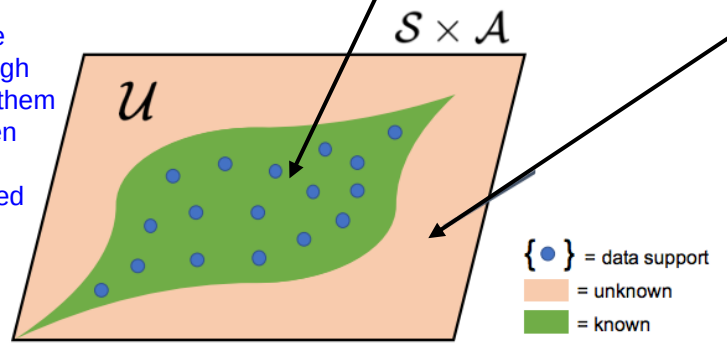
these models won't be perfect, especially if the dataset doesn't have high coverage. planning on them will have us take unseen actions, leading to divergence we discussed earlier

"proxy" world



Keep unaltered reward function

Make reward function conservative



(Figure from Kidambi et al. 2020)

Conservative Reward Functions in Offline RL

How do we obtain conservative reward functions?

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

Some measure of "uncertainty"

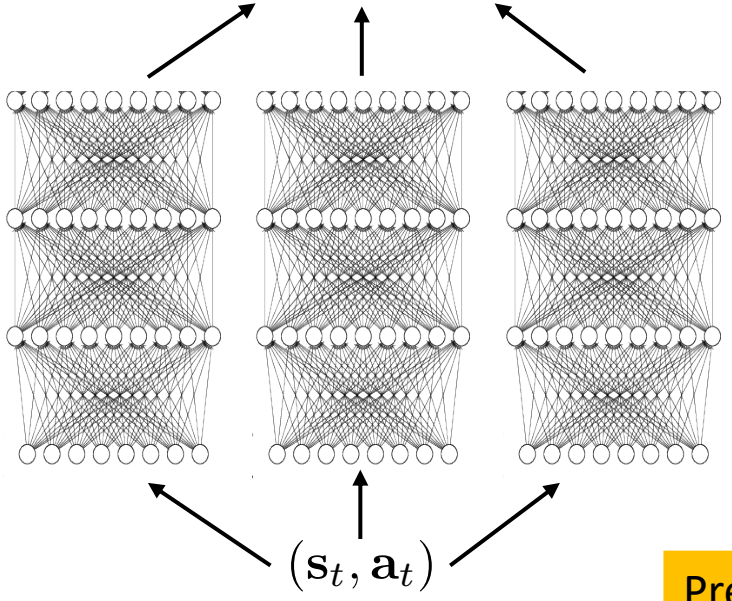
u(s,a) provides some indication of whether (s,a) is seen or unseen/ OOD

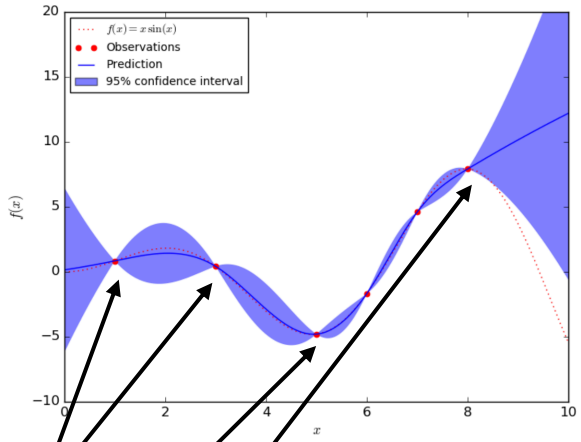
Concrete choices of u(s, a)?

How can we estimate uncertainty in model-based offline RL?

check agreement in the predictions to check if a particular (s,a) pair is in distribution or OOD

Train an ensemble of dynamics models and check agreement in their predictions





Predictions are certain at observed points

Different models will make mistakes differently on **unseen** points

the the variance in predictions on unseen data points will be high

Kidambi et al. 2020, Yu et al. 2020

Model-Based Offline RL Algorithms

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

Option 1: Disagreement

$$\text{dis}_{i,j}(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\mathbf{s}^i_{t+1} \sim p_i(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a_t})} \mathbb{E}_{\mathbf{s}^j_{t+1} \sim p_j(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a_t})} \left[||s^i_{t+1} - s^j_{t+1}|| \right]$$

some L1/L2 distance

if the disagreement is greater than some threshold, subtract the maximum possible reward

$$u(\mathbf{s}, \mathbf{a}) = r_{\max} \quad \text{if} \quad \max_{i,j} \text{dis}_{i,j}(\mathbf{s}, \mathbf{a}) > \text{threshold}$$

MOReL (Kidambi et al. 2020)

Option 2: Prediction uncertainty

$$p_i(\mathbf{s}'|\mathbf{s}, \mathbf{a}) = \mathcal{N}(\mu_i(\mathbf{s}, \mathbf{a}), \Sigma_i(\mathbf{s}, \mathbf{a}))$$

introduce uncertainty in the models by making predictions based on a gaussian distribution. Make u(s,a) that max frobenius norm of the covariance matrix

$$u(\mathbf{s}, \mathbf{a}) = \max_j ||\Sigma_j(\mathbf{s}, \mathbf{a})||_F$$

MOPO (Yu et al. 2020)

We obtained the reward function, how do we solve the RL problem now?

Generate model data, run model-free off-policy RL algorithm (**MBPO, Dyna**)

Run trajectory optimization or plan under the model (**Planning**)

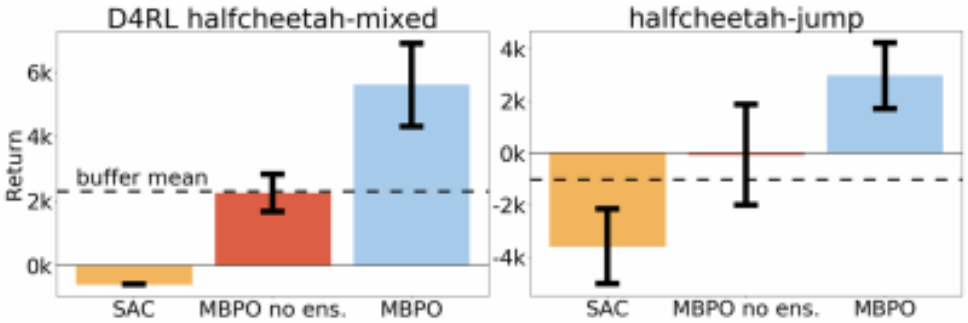
now, after altering the reward function by subtracting u(s,a), we can either use the models in a Dyna-style algo, or we can run a straight planning algo. Both options have been explored in literature and they have trade offs.

Both methods do not alter the dynamics at all….

Should we? Or not?

Kidambi, Rajeshwaran, Netrapalli, Joachims. **Model-Based Offline Reinforcement Learning.** NeurIPS 2020.
Yu, Thomas, Yu, Ermon, Levine, Finn, Ma. **MOPO: Model-Based Offline Policy Optimization.** NeurIPS 2020.

How does Model-Based Offline RL perform?



Naïve model-based methods can help with sampling error challenges if models can be learned accurately

Conservatism **does** help with model-based RL methods on "narrow" datasets

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

Generally **better** than policy constraint methods: less conservative in many cases!

Results from MOPO (Yu et al. 2020)

Value Function Regularization for Offline RL

So far, we have looked at handling overestimation via penalizing the reward function

Can we penalize value functions or Q-functions directly?

KL-control (Kakade '02, Fox et al. '16, Haarnoja et al. '17)

BRAC-v (Wu et al. 2019), KL-control (Jaques et al. 2019)

Way 1: Add penalty based on policy constraint to the value function

$$Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{\mathbf{a}' \sim \pi_\theta(\mathbf{a}'|\mathbf{s}')}[Q(\mathbf{s}',\mathbf{a}')] - \alpha D(\pi_\theta, \pi_\beta)$$

But this still uses a policy constraint…..

Why can this work?

TD error will push values on **seen** state-action pairs up, while we minimize others

Way 2: Change the objective to make this behavior automatically kick in!

Minimize the **"big" Q-values**

Standard TD error objective

$$\min_Q \max_\mu \quad \mathbb{E}_{\mathbf{s}\sim\mathcal{D}} \mathbb{E}_{\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] + \frac{1}{2\alpha} \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}} \left[(Q(\mathbf{s},\mathbf{a}) - y(\mathbf{s},\mathbf{a}))^2 \right]$$

additional regularizer that serves to minimize the "big" Q-values. So it identifies actions with high Q-values, and minimizes and pulls those Q-values down

$$y(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{\mathbf{a}' \sim \pi_\theta(\mathbf{a}'|\mathbf{s}')}[Q(\mathbf{s}',\mathbf{a}')]$$

Target values

essentially, the TD error pushes up Q-values on seen state-action pairs. This happens through the bellman backup. We then pull the Q-values down for unseen actions where the Q-values are high. So you make sure the Q-values on actions we haven't seen are not overestimated.

Conservative Q-Learning

CQL-v1

$$Q_{\mathrm{CQL}}(\mathbf{s},\mathbf{a}) := \arg \min_{Q} \max_{\mu} \; \mathbb{E}_{\mathbf{s}\sim\mathcal{D}}\mathbb{E}_{\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] + \frac{1}{2\alpha}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[(Q(\mathbf{s},\mathbf{a}) - y(\mathbf{s},\mathbf{a}))^2\right]$$

$$\forall \mathbf{s}\in\mathcal{D},\mathbf{a},\; Q_{\mathrm{CQL}}(\mathbf{s},\mathbf{a}) \le Q(\mathbf{s},\mathbf{a})$$

✚ Conservative (so no overestimation)
✖ Can be too conservative!

CQL-v2

Minimize the big Q-values | Maximize the data Q-values | Standard TD error

$$Q_{\mathrm{CQL}}(\mathbf{s},\mathbf{a}) := \arg \min_{Q} \max_{\mu} \; \left(\mathbb{E}_{\mathbf{s}\sim\mathcal{D}}\mathbb{E}_{\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] - \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q(\mathbf{s},\mathbf{a})]\right) + \frac{1}{2\alpha}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[(Q(\mathbf{s},\mathbf{a}) - y(\mathbf{s},\mathbf{a}))^2\right]$$

$$\forall \mathbf{s}\in\mathcal{D},\mathbf{a},\; Q_{\mathrm{CQL}}(\mathbf{s},\mathbf{a}) \le Q(\mathbf{s},\mathbf{a})$$ ✖

$$\forall\, \mathbf{s}\in\mathcal{D},\; V_{\mathrm{CQL}}(\mathbf{s}) \le V(\mathbf{s})$$ ✓

Value function is still a lower-bound, i.e. policy value is lower-bounded,

Much “tighter”

CQL Algorithm:

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

Related ideas: Piot et al. ‘14, Hester et al. ‘17: use ”margin loss” instead of Q-function difference

Value Regularization in Practice

Minimize ”soft”-maximum Q-values and maximize data Q-values

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

1. update Q_ϕ to decrease $\diamond$
2. update π_θ to increase $E_{\mathbf{s}\sim d^{\pi_\theta}(\mathbf{s}),\mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{s})}\left[Q_\phi(\mathbf{s},\mathbf{a})\right]$

Policy constraint penalty vs. regularized objectives

$$Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \gamma\mathbb{E}_{\mathbf{a}'\sim\pi_\theta}[Q(\mathbf{s}',\mathbf{a}')] - \alpha D(\pi_\theta,\pi_\beta)$$

A different penalty, but very similar

With tabular Q-functions

$$Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \gamma\mathbb{E}_{\mathbf{a}'\sim\pi_\theta}[Q(\mathbf{s}',\mathbf{a}')] - \alpha D_{\mathrm{CQL}}(\pi_\theta,\pi_\beta)$$

With Q-function approximators (in practice)

Value Regularization = “Q-function aware” penalty

“projection” operator, a concise notation to represent function approximators

$$Q(\mathbf{s},\mathbf{a}) \leftarrow \prod_\phi \left(r(\mathbf{s},\mathbf{a}) + \gamma\mathbb{E}_{\mathbf{a}'\sim\pi_\theta}[Q(\mathbf{s}',\mathbf{a}')] - \alpha D_{\mathrm{CQL}}(\pi_\theta,\pi_\beta)\right)$$

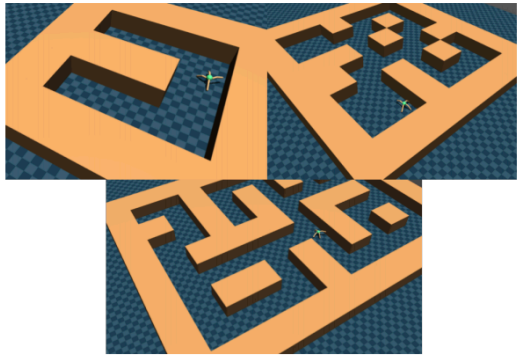
$$Q(\mathbf{s},\mathbf{a}) \leftarrow \prod_\phi \left(r(\mathbf{s},\mathbf{a}) + \gamma\mathbb{E}_{\mathbf{a}'\sim\pi_\theta}[Q(\mathbf{s}',\mathbf{a}')]\right) - \alpha f(\pi_\theta,\pi_\beta,Q_\phi)$$

How does CQL perform?

Learned policy value - Actual policy value

CQL learns conservative Q-values

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



Imitation | Naïve off-policy RL | Policy constraints | "Way 1" | CQL variants

Domain	Task Name	BC	SAC	BEAR	BRAC-p	BRAC-v	CQL($\mathcal{H}$)	CQL(ρ)
AntMaze	antmaze-umaze	65.0	0.0	**73.0**	50.0	70.0	**74.0**	**73.5**
	antmaze-umaze-diverse	55.0	0.0	61.0	40.0	70.0	**84.0**	61.0
	antmaze-medium-play	0.0	0.0	0.0	0.0	0.0	**61.2**	4.6
	antmaze-medium-diverse	0.0	0.0	8.0	0.0	0.0	**53.7**	5.1
	antmaze-large-play	0.0	0.0	0.0	0.0	0.0	**15.8**	3.2
	antmaze-large-diverse	0.0	0.0	0.0	0.0	0.0	**14.9**	2.3

CQL performs better due to stitching!

Uncertainty-Based Offline RL Methods

so far we've seen three approaches: 1) policy constraint methods, 2) Model-based methods, 3) Model-free methods.

All three methods seek to do the same thing: Penalize the value for unseen behaviors as much as possible! So either we prevent the policy from taking unseen actions (i.e. policy constrant), or we ensure the Q-value is small for those unseen actions

But what if our model or Q-function can generalize on unseen (state, action) pairs? In that case, we wouldn't necessarily want to penalize them and be conservative! But figuring out whether or not we can generalize is tricky.

Promise: Can do better by taking into account **when** our Q-function/model can generalize!

two-sides of the spectrum: A) traditional off-policy, B) add conservatism. By understanding how well we can generalize, we can modulate where we are between the two extremes.

Model-based methods

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

We already do it! – ensembles of models, disagreement, etc

model-based methods fall into the uncertainty-based category.

How can we model uncertainty sets over Q-functions?

Model-free methods

Computing an uncertainty function and running explicit RL on it tends to be not be much different than policy constraint methods

A different approach:
directly model Q-function uncertainty

$\mathcal{P}_{\mathcal{D}}(Q^{\pi})$ = uncertainty set of **Q-functions** given the dataset

expected Q-value (standard)

conservative when uncertain

$$\pi_{k+1} \leftarrow \arg\max_{\pi} \mathbb{E}_{\mathbf{s}\sim\mathcal{D}} \underbrace{\left[\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})} \left[\mathbb{E}_{Q^{\pi}_{k+1}\sim\mathcal{P}_{\mathcal{D}}(Q^{\pi})}[Q^{\pi}_{k+1}(\mathbf{s}, \mathbf{a})] - \alpha \mathrm{Unc}(\mathcal{P}_{\mathcal{D}}(Q^{\pi}))\right]\right]}_{\text{conservative estimate}}$$

Uncertainty-Based Offline RL Methods

Bootstrap ensembles of Q-functions

average dirac delta distribution over all possible q-functions

$$\mathcal{P}_{\mathcal{D}}(Q^{\pi}) \approx \frac{1}{N}\sum_{j=1}^{N}\delta[Q^{\pi} = Q_{\phi_j}]$$

N Q-functions $Q_{\phi_1}, Q_{\phi_2}, \cdots, Q_{\phi_N}$

we partition our dataset into independent partitions (i.e. bootstrapping). We make sure that each Q-function has their own, independent target function too.

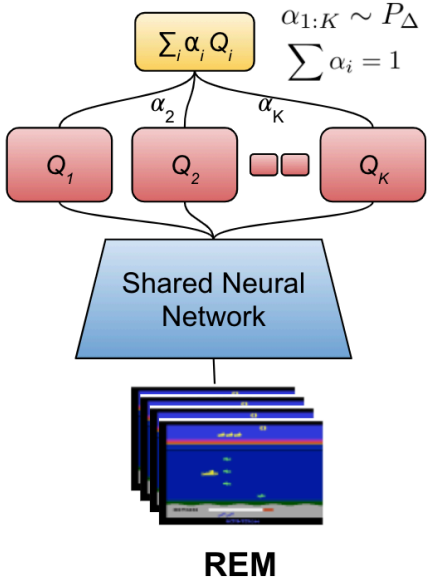
Train each Q-function independently

$$\min_{\phi_i} \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[(Q_{\phi_i}(\mathbf{s},\mathbf{a}) - y_i(\mathbf{s},\mathbf{a}))^2]$$

$$\pi_{k+1} \leftarrow \arg\max_{\pi} \mathbb{E}_{\mathbf{s}\sim\mathcal{D}}\underbrace{\left[\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\mathbb{E}_{Q^{\pi}_{k+1}\sim\mathcal{P}_{\mathcal{D}}(Q^{\pi})}[Q^{\pi}_{k+1}(\mathbf{s},\mathbf{a})] - \alpha\mathrm{Unc}(\mathcal{P}_{\mathcal{D}}(Q^{\pi}))\right]\right]}_{\text{conservative estimate}}$$

so we underestimate uncertainty and has shown not to work well in practice so far

Very little diversity in an ensemble and thus incorrect uncertainty (Fujimoto et al. 2019, Kumar et al. 2019)



REM

Random Ensemble Mixture (REM)

$$Q(\mathbf{s},\mathbf{a}) = \sum_{j=1}^{N}\alpha_j Q_{\phi_j}(\mathbf{s},\mathbf{a})$$

$$\mathrm{Unc}(Q_{\phi_1}, \cdots, Q_{\phi_N}) = 0$$

…but weights randomly chosen!

this approach has been shown to work. they didn't explain this well

Overall, a general solution that works well in each case has been hard to obtain

Works well with high-coverage datasets

Agarwal, Schuurmans, Norouzi. **An Optimistic Perspective on Offline Reinforcement Learning.** ICML 2020.

Summary of Offline RL Methods

- **Policy constraint** often work well when the behavior policy distribution is "easy" to model, but tend to be more conservative
- **Model-based offline RL** often work quite well when the data distribution is relatively high-coverage, and it is easy to learn a model you can approximate the dynamics and reward function with high accuracy
- **Value-regularized offline RL** methods tend to be less conservative and can be augmented with either models or policy constraints, if needed

Method Type	Modifies	Modification Type	Extra requirements
Policy constraints	Learned policy	Probabilistic constraint, support constraint, state-marginal constraints	Behavior policy estimation (except implicit constraints)
Model-based	Reward function	Subtracts an uncertainty function	Defining an uncertainty/ conservatism metric
Value regularization	Q-function	Modified Q-function training objective	Defining a regularizer
Uncertainty-based	Q-function or model	Subtract an uncertainty (one approach)	Uncertainty estimation

Additional readings: Modern Offline RL Methods

- **Summary/Survey:** Levine, Kumar, Tucker, Fu (2020). Offline Reinforcement Learning: Tutorial, Survey and Perspectives on Open Problems. https://arxiv.org/abs/2005.01643

- **Policy Constraint Algorithms**
  - Fujimoto et al. (2019) Off-Policy Reinforcement Learning without Exploration. ICML 2019
  - Laroche et al. (2019) Safe Policy Improvement via Baseline Bootstrapping. ICML 2019.
  - Nadjahi et al. (2019). Safe Policy Improvement with Soft Baseline Bootstrapping. ECML-PKDD 2019.
- - Simao et al. (2019). Safe Policy Improvement with an Estimated Behavior Policy.
  - Kumar et al. (2019) Stabilizing Off-Policy Reinforcement Learning via Bootstrapping Error Reduction. NeurIPS 2019
  - Wu et al. (2019). Behavior Regularized Offline Reinforcement Learning.
  - Jaques et al. (2019). Way Off-Policy Batch Deep Reinforcement Learning of Implicit Human Preferences in Dialog.
  - Peng et al. (2019). Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning.
  - Siegel et al. (2019). Keep Doing What Worked: Behavior Modelling Priors for Offline RL. ICLR 2020.
  - Ghasemipour et al. (2020). EMaQ: Expected-Max Q-Learning Operator for Simple Yet Effective Offline and Online RL.
  - Liu et al. (2020). Provably Good Batch RL without Great Exploration. NeurIPS 2020.
  - Wang et al. (2020). Critic-regularized Regression. NeurIPS 2020.
  - Chen et al. (2020). BAIL: Best-Action Imitation Learning for Batch Deep Reinforcement Learning. NeurIPS 2020.

…and many more

- **Model-Based [MB]/Value Regularization [V]/Uncertainty-aware [U] algorithms**
  - [MB] Yu et al. (2020). MOPO: Model-Based Offline Policy Optimization. NeurIPS 2020.
  - [MB] Kidambi et al. (2020). Model-Based Offline Reinforcement Learning. NeurIPS 2020.
  - [V] Kumar et al. (2020). Conservative Q-Learning for Offline Reinforcement Learning. NeurIPS 2020.
  - [U] Agarwal et al. (2020). An Optimistic Perspective on Offline Reinforcement Learning. ICML 2020.
  - [U] Sonabend-W et al. (2020). Expert-Supervised RL for Offline Policy Learning and Evaluation. NeurIPS 2020.

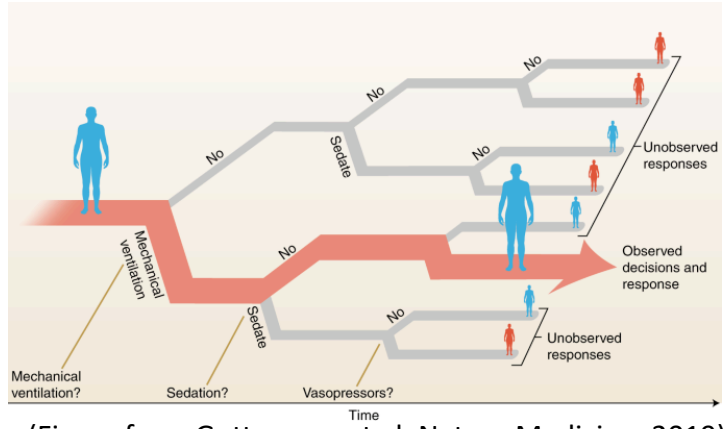
1. Reinforcement learning primer
2. The offline reinforcement learning problem
3. Classic batch reinforcement learning algorithms
    i. Importance-sampled policy gradients
    ii. Least-squares temporal difference (LSTD)
4. Modern offline reinforcement learning algorithms
    i. Policy constraint methods
    ii. Model-based offline RL
    iii. Value function regularization
    iv. Uncertainty-based methods
    v. Comparisons and discussion
5. Off-policy evaluation and model selection
    i. Problem statement
    ii. Direct methods and importance sampling methods
    iii. Doubly robust methods
6. How should we evaluate offline RL methods?
    i. When and why does offline RL work?
    ii. How do we benchmark offline RL methods?
7. Conclusions, discussion, and open problems

Off-Policy Evaluation and Model Selection

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

$$\pi \longrightarrow V^{\pi}(s)$$
$$\mathcal{D} \nearrow$$

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

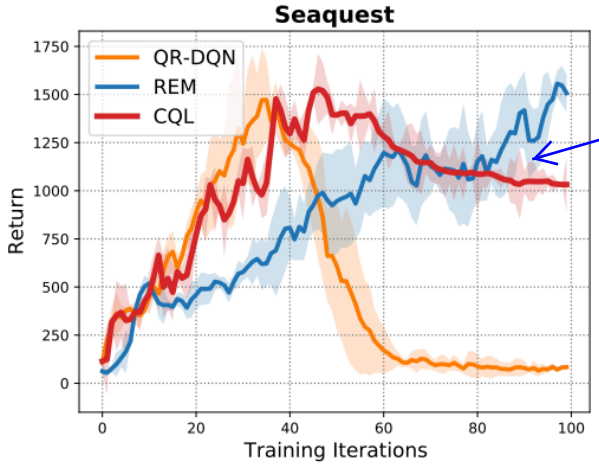


(Figure from Gottesman et al. Nature Medicine, 2019)

What is the use of OPE in offline RL?
Model selection: selecting hyperparameters/policies

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. Moreover, it is unclear how to select hyperparameters.



notice how performance drops after a certain number of training iterations

offline algorithms can be extremely sensitive to hyperparameters, and can be the difference between a method working and not working!

NeurIPS 2020)

Wu, Tucker, Nachum. **Behavior Regul** **t Learning.** '2019

Agarwal, Schuurmans, Norouzi. **An O** **ine Reinforcement Learning.** ICML 2020

Direct Methods for Off-Policy Evaluation

- **Model-based approach:** Fit a transition and reward function model. Compute return of the policy under the model.

$$\hat{p}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \qquad \hat{r}(\mathbf{s}_t, \mathbf{a}_t)$$

policy to evaluate

$$\hat{J}(\pi) = \mathbb{E}_{\mathbf{s}_0, \mathbf{s}_{t+1} \sim \hat{p}, \mathbf{a}_t \sim \pi} \left[\sum_t \gamma^t \hat{r}(\mathbf{s}_t, \mathbf{a}_t) \right]$$

- **Fitted-Q Evaluation (FQE):** Evaluate a policy by training a Q-function using Bellman consistency for the chosen policy

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

$$\hat{Q}_\phi := \arg\min_\phi \; \mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}' \sim \mathcal{D}} \left[(Q_\phi(\mathbf{s}, \mathbf{a}) - (r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{a}' \sim \pi}[Q_\phi(\mathbf{s}', \mathbf{a}')]))^2 \right]$$

Works quite well in practice

$$\hat{J}(\pi) = \mathbb{E}_{\mathbf{s}_0, \mathbf{a}_0 \sim \pi(\mathbf{a}_0|\mathbf{s}_0)} \left[\hat{Q}_\phi(\mathbf{s}_0, \mathbf{a}_0) \right]$$

DM: Jiang, Li. **Doubly robust off-policy value evaluation for reinforcement learning**. ICML 2016 (and many others)
FQE: Paine*, Paduaru* et al. **Hyperparameter Selection for offline reinforcement learning.** ’2020 (and others)

Importance Sampling OPE Methods

- **Per-decision importance sampling/ Inverse propensity scoring (IPS):** Compute importance ratios and re-weight trajectories

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

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

Sum over the dataset

High variance

Markovian structure?

- **Step-wise importance sampling estimator**: Can replace with step-wise weights, rather than trajectory-level weights

$$J(\pi_\theta) \approx \frac{1}{|\mathcal{D}|}\sum_{i=1}^{n}\sum_{t=0}^{H}\gamma^t\underbrace{\left(\prod_{j=0}^{t}\frac{\pi_\theta(\mathbf{a}_j|\mathbf{s}_j)}{\pi_\beta(\mathbf{a}_j|\mathbf{s}_j)}\right)}_{\text{"surrogate" for state frequency}} r_t(\mathbf{s}_t,\mathbf{a}_t)$$

Improves over trajectory level weights

Doubly-Robust Importance Sampling

Let's add a learned Q-function to reduce variance, which still retaining the unbiased nature

Need to evaluate $V(\mathbf{s}) := \mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}[r(\mathbf{s},\mathbf{a})]$

$$\hat{V}(\mathbf{s}) = \mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}[\hat{r}(\mathbf{s},\mathbf{a})]$$

Reward model, separate estimator

Importance sampling

$$\hat{V}_{\text{IS}}(\mathbf{s}) := \frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})}[r(\mathbf{s},\mathbf{a})]$$

Importance weights + a different estimator

$$\hat{V}_{\text{DR}}(\mathbf{s}) = \hat{V}(\mathbf{s}) + \frac{\pi(\mathbf{a}|\mathbf{s})}{\pi_\beta(\mathbf{a}|\mathbf{s})}\left(r(\mathbf{s},\mathbf{a}) - \hat{r}(\mathbf{s},\mathbf{a})\right)$$

Unbiased in expectation

Lower variance

Take step-wise IS estimator

$$V^\pi_{\text{step-IS}}(\mathbf{s}_t) = \frac{\pi(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\beta(\mathbf{a}_t|\mathbf{s}_t)}\left(r(\mathbf{s}_t,\mathbf{a}_t) + \gamma V^\pi_{\text{step-IS}}(\mathbf{s}_{t+1})\right)$$

$$V^\pi_{\text{DR}}(\mathbf{s}_t) = \hat{V}(\mathbf{s}_t) + \frac{\pi(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\beta(\mathbf{a}_t|\mathbf{s}_t)}\left(r(\mathbf{s}_t,\mathbf{a}_t) + \gamma V^\pi_{\text{DR}}(\mathbf{s}_{t+1}) - \hat{Q}(\mathbf{s}_t,\mathbf{a}_t)\right)$$

Jiang, Li. **Doubly robust off-policy value evaluation for reinforcement learning**. ICML 2016
Thomas, Brunskill. **Data-efficient off-policy policy evaluation for reinforcement learning**. ICML 2016

Marginalized Importance Sampling

Importance sampling over a trajectory suffers from high variance. Can we do better via state distributions?

$\textbf{Main idea}$: instead of using $\prod_t \frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\beta(\mathbf{a}_t|\mathbf{s}_t)}$, estimate $w(\mathbf{s},\mathbf{a}) = \frac{d^{\pi_\theta}(\mathbf{s},\mathbf{a})}{d^{\pi_\beta}(\mathbf{s},\mathbf{a})}$

$$J(\pi_\theta) \approx \frac{1}{N}\sum_i w(\mathbf{s}_i,\mathbf{a}_i) r(\mathbf{s}_i,\mathbf{a}_i)$$

DualDICE: importance weights are solutions to:

$$w(\mathbf{s},\mathbf{a}) = \arg\min_{x:\mathcal{S}\times\mathcal{A}\to\mathbb{R}} \frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[x(\mathbf{s},\mathbf{a})^2\right] - \mathbb{E}_{\mathbf{s}\sim d^\pi(\mathbf{s}),\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[x(\mathbf{s},\mathbf{a})\right]$$

$$x(\mathbf{s},\mathbf{a}) = \nu(\mathbf{s},\mathbf{a}) - \underbrace{\mathbb{E}_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a}),\mathbf{a}'\sim\pi(\mathbf{a}'|\mathbf{s}')}[\nu(\mathbf{s}',\mathbf{a}')]}_{(\hat{\mathcal{B}}^\pi\nu)(\mathbf{s},\mathbf{a})}$$

(Change of variables)

$$\min_{\nu:\mathcal{S}\times\mathcal{A}\to\mathbb{R}} \frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[\left(\nu(\mathbf{s},\mathbf{a}) - \tilde{\mathcal{B}}^\pi\nu(\mathbf{s},\mathbf{a})\right)^2\right] - \mathbb{E}_{\mathbf{s}_0\sim d_0(\mathbf{s}_0),\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s}_0)}\left[\nu(\mathbf{s}_0,\mathbf{a})\right]$$

Very much like a Q-function

Nachum, Chow, Dai, Li. **DualDICE: Behavior-Agnostic Estimation of Discounted Stationary Distribution Corrections**. NeurIPS 2019.
Nachum and Dai. **Reinforcement Learning Via Fenchel-Rockafeller Duality.** '2020 (and many others)

Summary of OPE and Model Selection

- **Direct Methods (e.g., FQE)** have been shown to work well, but potentially suffer from distributional shift similar to policy learning methods
- **Marginalized importance sampling (MIS)** methods enjoy good theoretical guarantees, however they tend to be a bit challenging in practice with function approximation (saddle point problems)
- **Importance sampling (IS)** methods suffer from high variance and produce incorrect estimates in many cases, especially when the behavior policy needs to be estimated.

(See Voloshin et al. (2019). Empirical Analysis of Off-Policy Policy Evaluation for Reinforcement Learning)

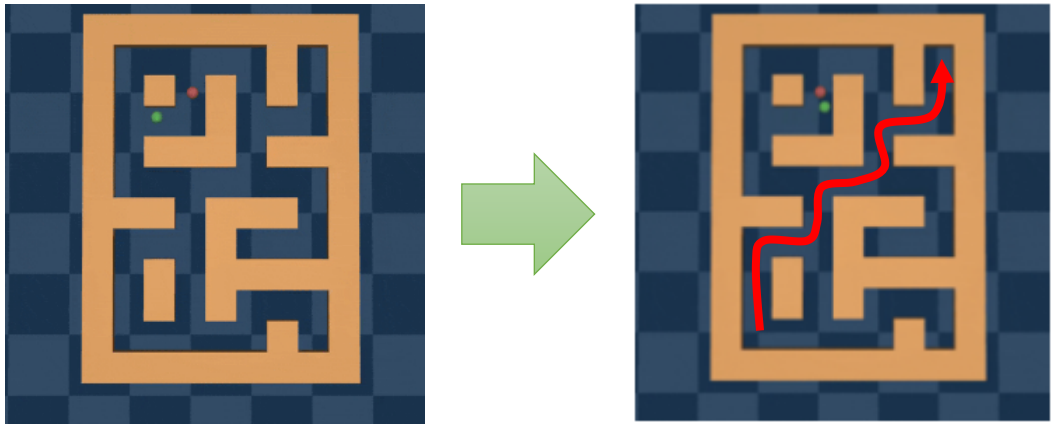
1. Reinforcement learning primer
2. The offline reinforcement learning problem
3. Classic batch reinforcement learning algorithms
    i. Importance-sampled policy gradients
    ii. Least-squares temporal difference (LSTD)
4. Modern offline reinforcement learning algorithms
    i. Policy constraint methods
    ii. Model-based offline RL
    iii. Value function regularization
    iv. Uncertainty-based methods
    v. Comparisons and discussion
5. Off-policy evaluation and model selection
    i. Problem statement
    ii. Direct methods and importance sampling methods
    iii. Doubly robust methods
6. How should we evaluate offline RL methods?
    i. When and why does offline RL work?
    ii. How do we benchmark offline RL methods?
7. Conclusions, discussion, and open problems

When and why does offline RL work?

1. Find the “good stuff” in a dataset full of good and bad behaviors

2. Generalization: good behavior in one place may suggest good behavior in another place

3. “Stitching”: parts of good behaviors can be recombined



What about stochastic settings?

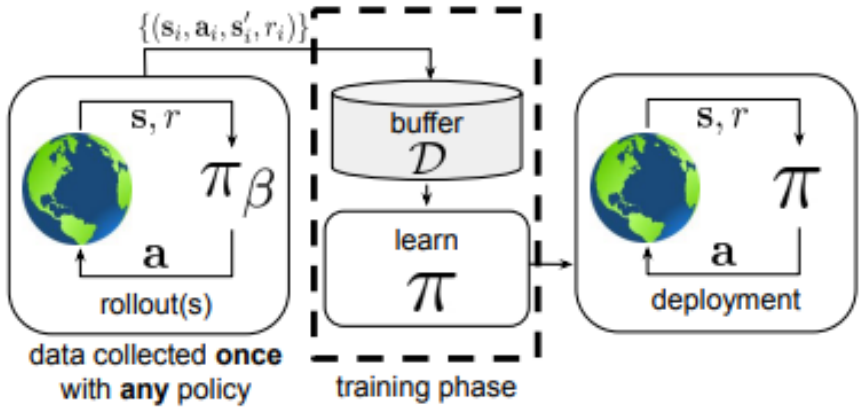
Just taking the best seen action is not always best!

We show that dynamic programming alone can leverage prior datasets

How do we know if it's going to work?

- **Major challenge:** if we can't roll out the policy, how do we know if it will work?
- This is a huge barrier to practical use of offline RL methods
- There is no single great solution, though off-policy evaluation (OPE) methods can help
- Usually this means that when we evaluate offline RL methods, we would still use a simulator/rollouts to see if our method works
- No great answer for hyperparameter tuning just yet, but active area of research (see, e.g., the OPE section before)

How do we evaluate offline RL methods?



maybe just train a reference policy with RL?

typical protocol in prior work:

1. train π_β with *online* RL
2. either collect data throughout training

OR

2. collect data from final policy π_β

this is not good

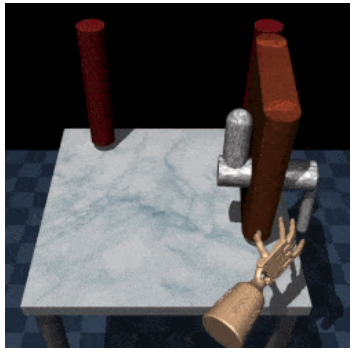
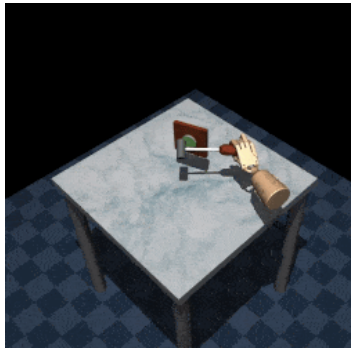
- In the real world, data might come from non-Markovian "policies"
  - Human users
  - Hand-engineered policies
- Must use data that is **representative of real-world settings** and **leaves lots of room for improvement**

  data must be suboptimal, or else the best you can do i behavioral cloning
- Offline RL **must learn policies that are much better than the behavior policy!** otherwise there's no point in doing offline RL

without testing these properties, we **cannot** trust that our algorithms are good!

D4RL: Datasets for Data-Driven Deep RL

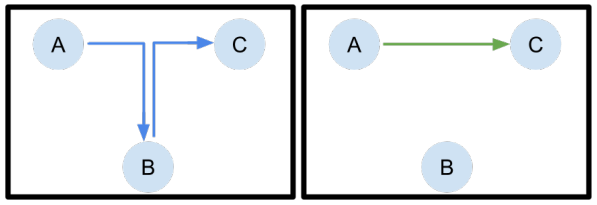
What are some important principles to keep in mind?

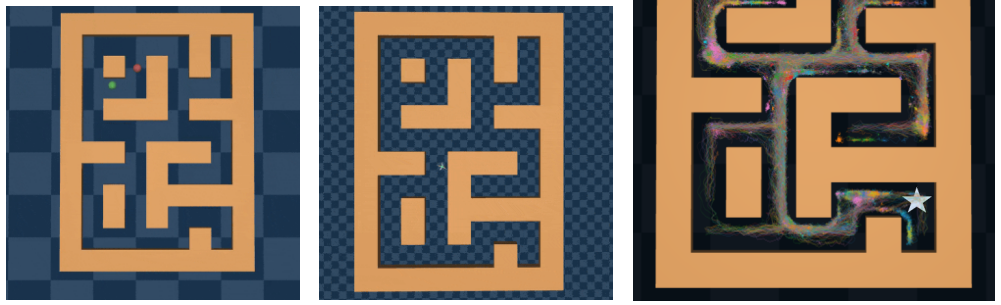



simulation & human data from Rajeswaran et al.

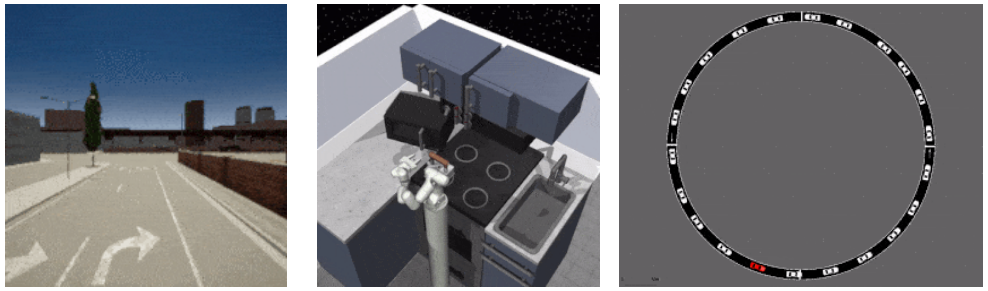
Data from non-RL policies, including data from humans

Stitching: data where dynamic programming can find much better solutions





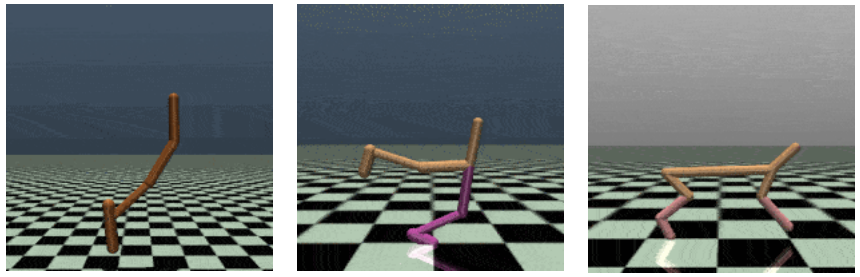
Realistic tasks



Fu, Kumar, Nachum Tucker, Levine. **D4RL: Datasets for Data-Driven Deep Reinforcement Learning.** '20

The D4RL Tasks

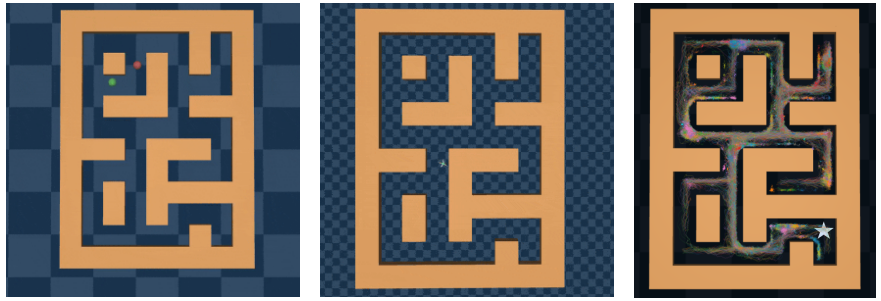
standard MuJoCo Gym-style benchmarks



standardized “expert,” “medium,” and “random” datasets introduced by Kumar et al.

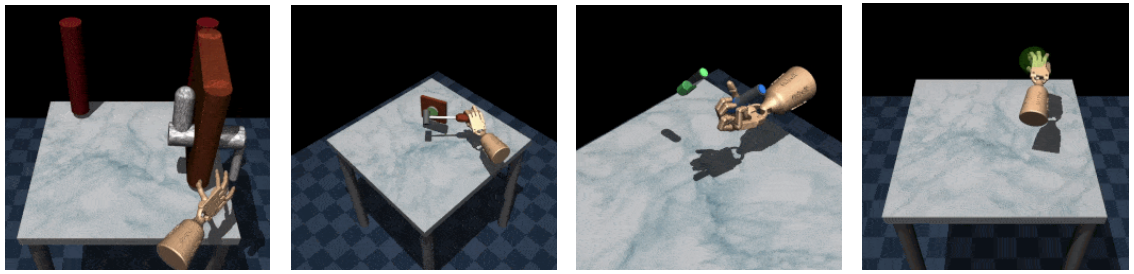
+ additional **mixed** datasets that are more difficult

mazes (2D maze & ant maze)



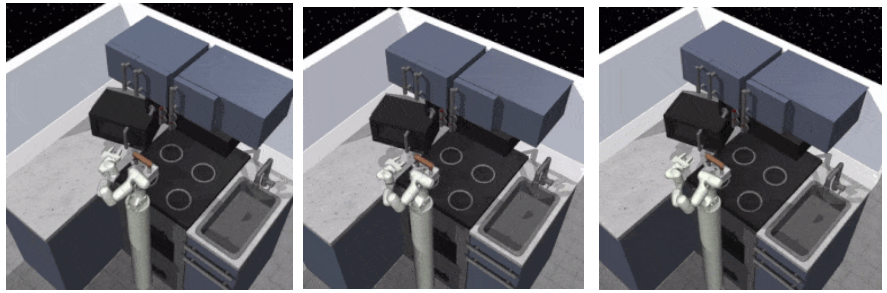
multi-task data, tests “stitching” ability

dexterous manipulation tasks



high dimensionality, human data, looks really awesome

whole-arm manipulation data



multi-task human data, realistic tasks

traffic control (Flow)

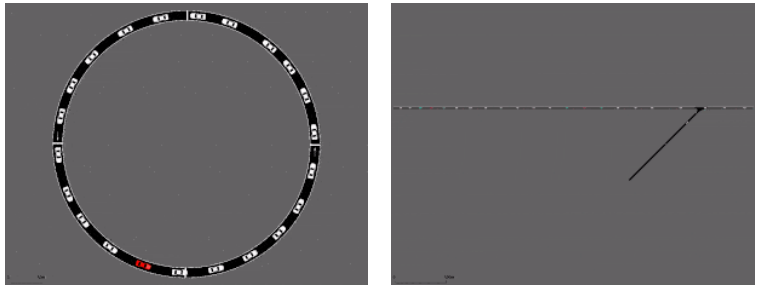
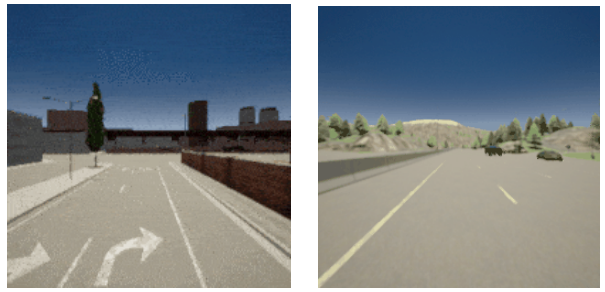


image-based driving (Carla)



Fu, Kumar, Nachum Tucker, Levine. **D4RL: Datasets for Data-Driven Deep Reinforcement Learning.** ‘20

Other properties that are important

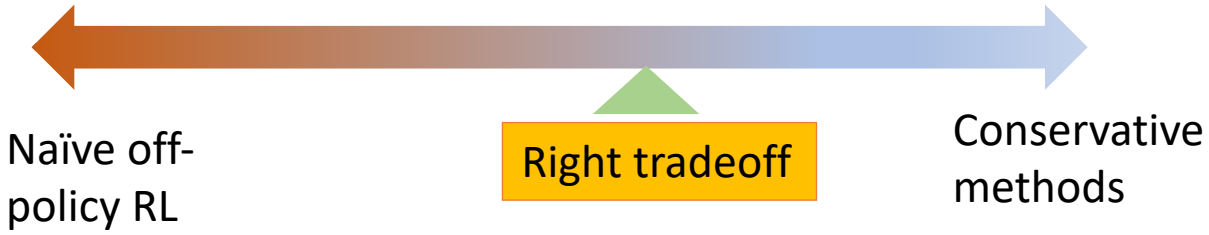
- **Stochasticity:** Common in real-world settings (economics, healthcare, education), often absent in commonly used benchmarks, high potential for impact from offline RL
- **Non-stationarity:** Things change over time, very common feature in the real world
- **"Risky" dataset biases:** If your driving data never shows a car crash, can you understand to avoid car crashes?
  - Some methods can handle this! But important to test
- **Partial observability:** Not discussed in this tutorial, but very important in the real world

1. Reinforcement learning primer
2. The offline reinforcement learning problem
3. Classic batch reinforcement learning algorithms
    i. Importance-sampled policy gradients
    ii. Least-squares temporal difference (LSTD)
4. Modern offline reinforcement learning algorithms
    i. Policy constraint methods
    ii. Model-based offline RL
    iii. Value function regularization
    iv. Uncertainty-based methods
    v. Comparisons and discussion
5. Off-policy evaluation and model selection
    i. Problem statement
    ii. Direct methods and importance sampling methods
    iii. Doubly robust methods
6. How should we evaluate offline RL methods?
    i. When and why does offline RL work?
    ii. How do we benchmark offline RL methods?
7. Conclusions, discussion, and open problems

Open Problems: Over-Conservatism?

- Methods covered so far act over-conservatively: there is no need to for conservatism if our learned models can generalize.

- Uncertainty-aware methods can be used to tradeoff conservatism for generalization



- But we need calibrated uncertainty estimates, calibrated across time!

Ways to measure and optimize uncertainty (consistent with RL)

Can we ever avoid conservatism?

"Expected case" scenarios

Open Problems: Hyperparameter Tuning

- **General Purpose OPE:** Do we need general OPE for hyperparameter tuning in offline RL?
  Can we instead utilize a "validation error" metric during training?
  Can we evaluate a subset of specific policies in OPE?

- **Goal:** Often we care about ranking policies, so do we actually need to train the policy return or value function?

Ranking correlation

Regret @ k

$$\text{Regret @ } 1 = \max_{\pi \in \Pi} J(\pi) - J\left(\arg\max_{\pi \in \Pi} \hat{J}(\pi)\right)$$

Optimal policy in a set of policies

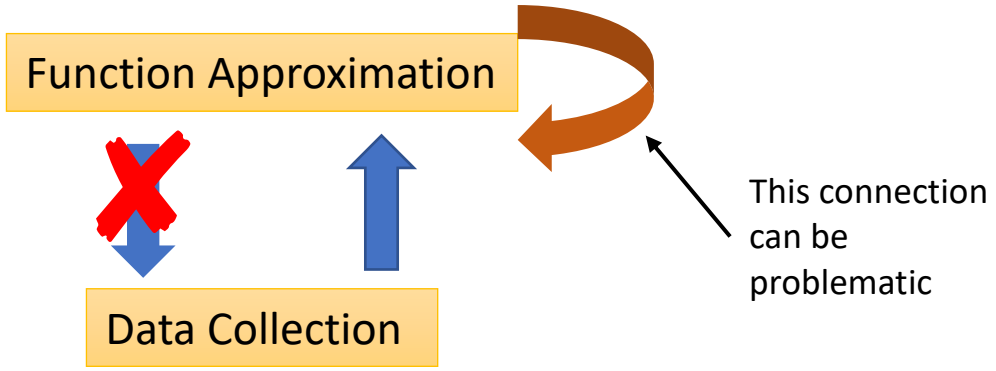
Pick out top k policies according to predicted return

Paine*, Paduaru* et al. **Hyperparameter Selection for offline reinforcement learning.** '2020

Open Problems: Function Approximation

We want to leverage ”good generalization” properties of deep neural networks in offline RL

Function approximation can have drastic impact on the performance of RL in offline settings in particular



Using one to train itself can lead to “bad stuff”

Optimization issues (lack of stability)

i.e. bellman equations are recursive and use previous Q-values to affect future Q-values

“Bad” generalization (lack of robustness)

…and lots of more open questions

Representation learning

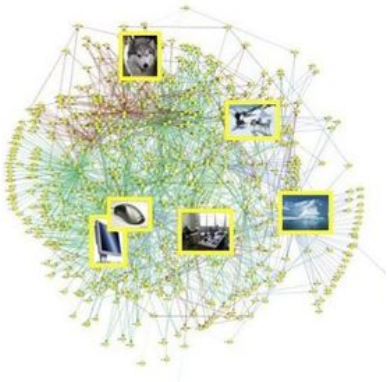
Neural network design

Handling non-i.i.d. data

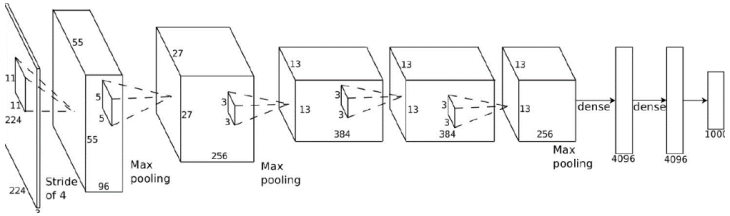
Causal inference

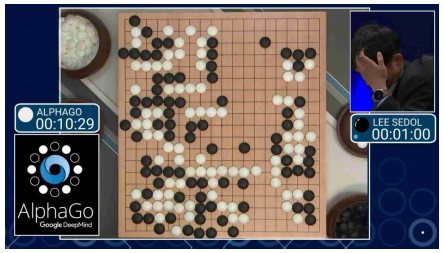
Structures (Hierarchy, binary rewards)

Conclusion



+





enormous gulf

Offline RL methods aim to bridge this gulf by converting datasets to effective decision-making engines

Thank You!

Resources:

- Tutorial website (slides/video/more content): https://sites.google.com/view/offline-rl-tutorial

- Offline RL survey paper: https://arxiv.org/abs/2005.01643

- Hands-on coding assignment at: https://sites.google.com/view/offline-rl-tutorial