IASD M2 at Paris Dauphine

Deep Reinforcement Learning

16: Offline Reinforcement Learning

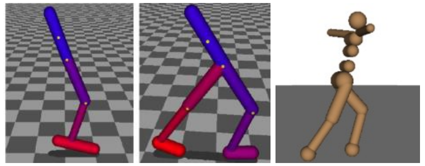
Eric Benhamou David Saltiel

Acknowledgement

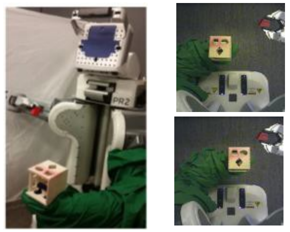
These materials are based on the seminal course of Sergey Levine CS285

The generalization gap

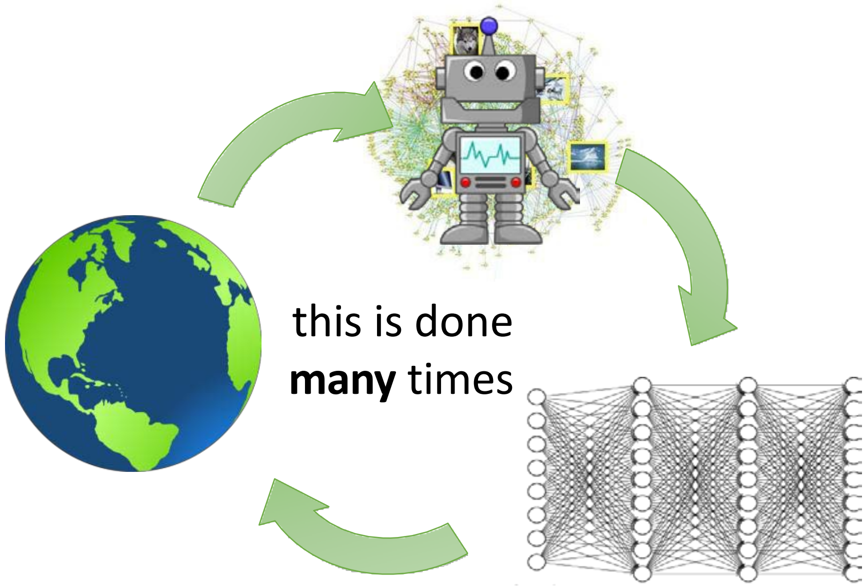
Mnih et al. ‘13



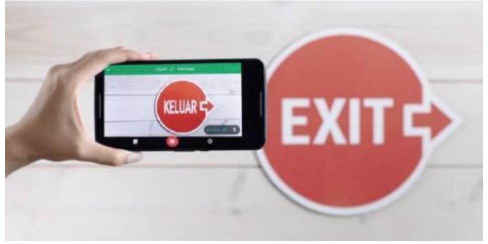
Schulman et al. ’14 & ‘15

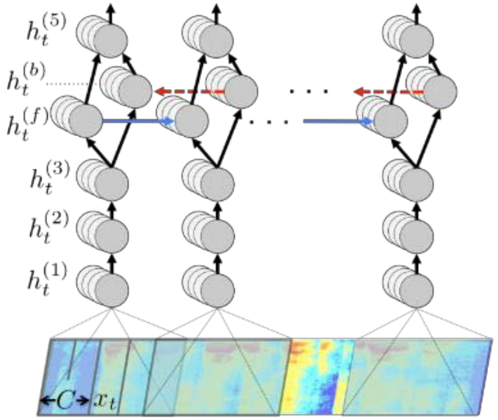


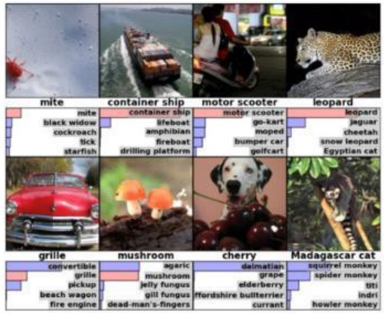
Levine*, Finn*, et al. ‘16



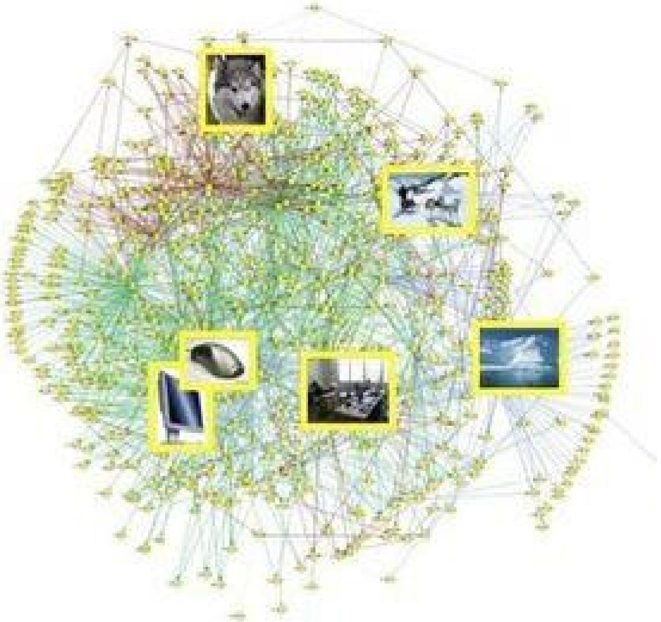
enormous gulf

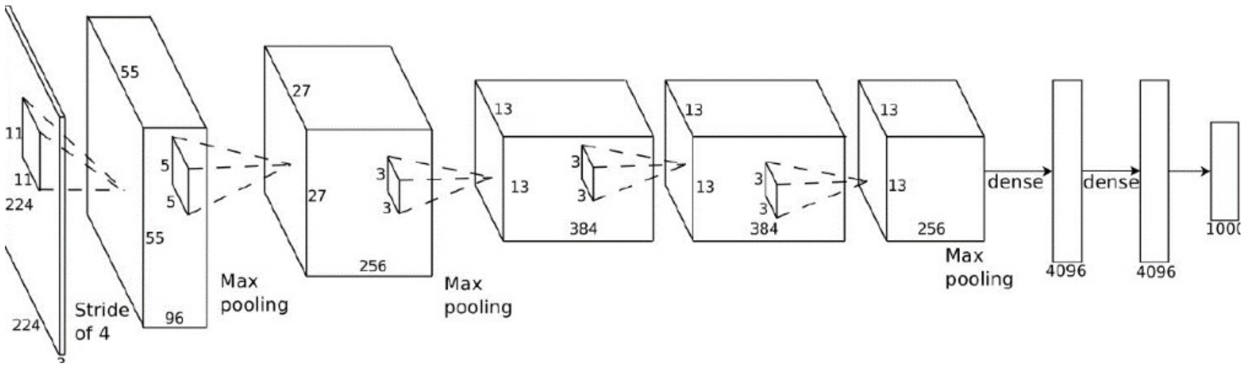


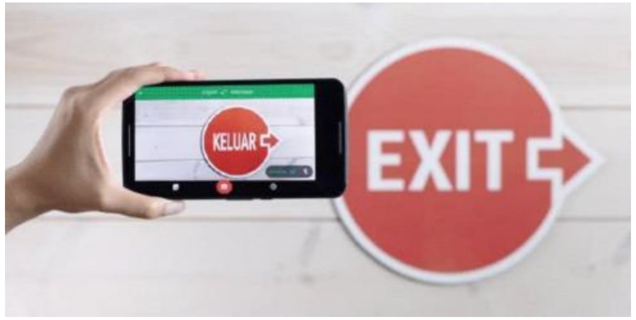


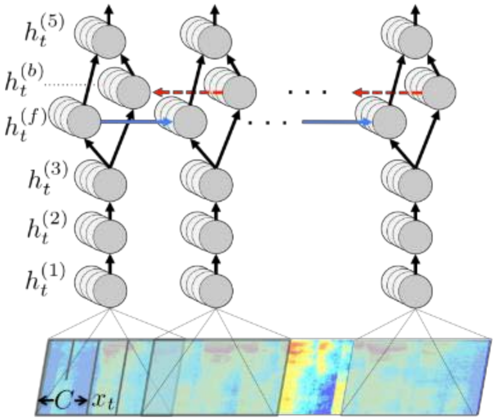


What makes modern machine learning work?



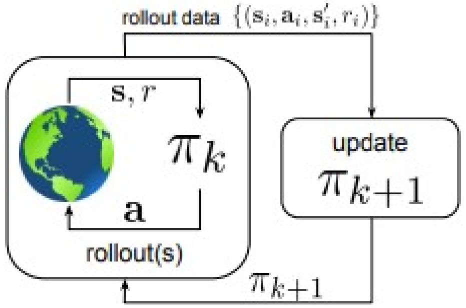




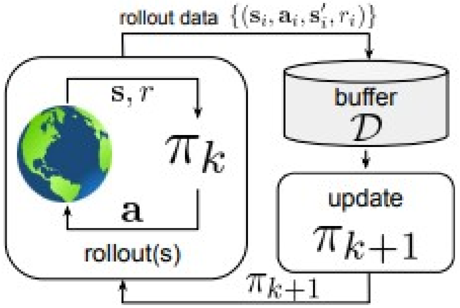


Can we develop data-driven RL methods?

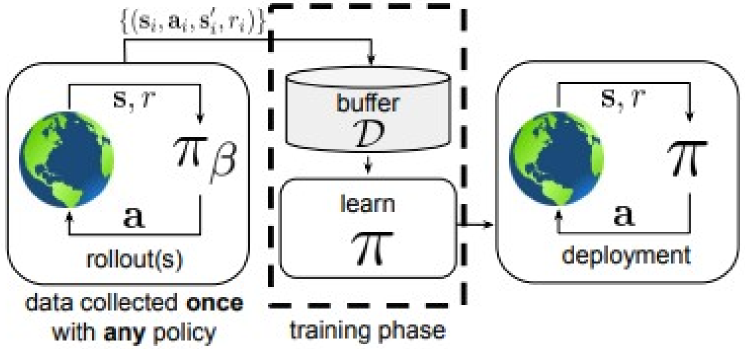
on-policy RL

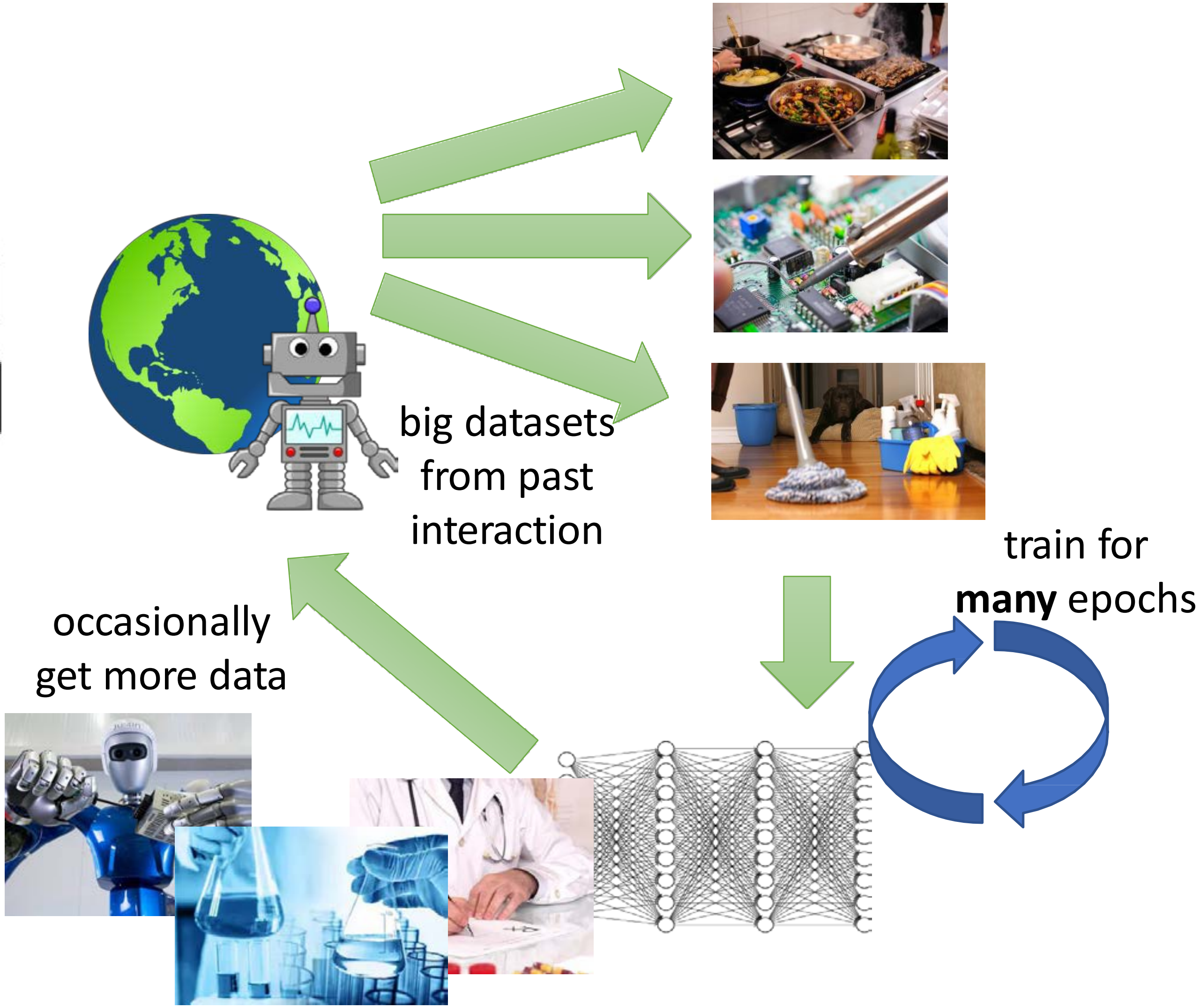


off-policy RL



offline reinforcement learning

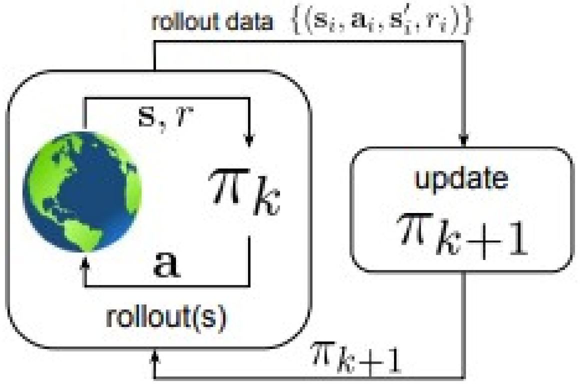




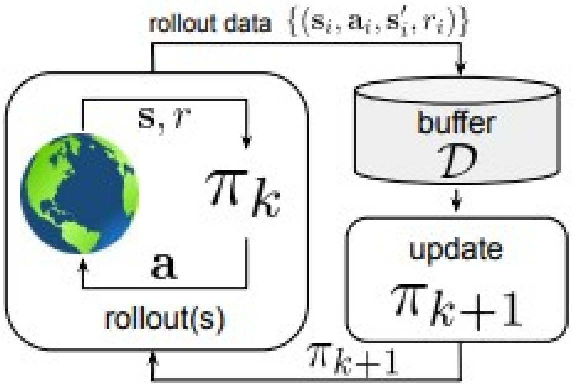
Levine, Kumar, Tucker, Fu. **Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems.** ‘20

What does offline RL mean?

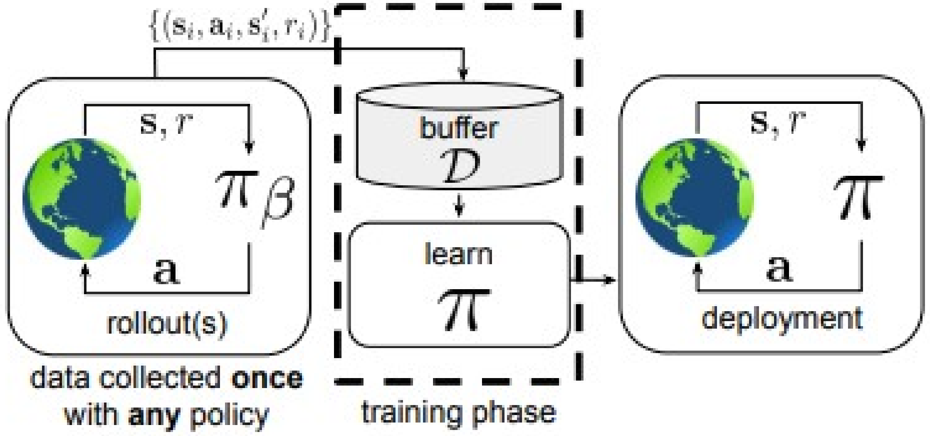
on-policy RL



off-policy RL



offline reinforcement learning



Formally:

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

$$\mathbf{s} \sim d^{\pi_\beta}(\mathbf{s})$$
$$\mathbf{a} \sim \pi_\beta(\mathbf{a}|\mathbf{s})$$
$$\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$$
$$r \leftarrow r(\mathbf{s}, \mathbf{a})$$

generally **not** known

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

Types of offline RL problems

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

$$\mathbf{s} \sim d^{\pi_\beta}(\mathbf{s})$$
$$\mathbf{a} \sim \pi_\beta(\mathbf{a}|\mathbf{s})$$
$$\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$$
$$r \leftarrow r(\mathbf{s}, \mathbf{a})$$

off-policy evaluation (OPE):

given $\mathcal{D}$, estimate $J(\pi) = E_\pi\left[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right]$

offline reinforcement learning: (a.k.a. batch RL, sometimes fully off-policy RL)

given $\mathcal{D}$, learn the best possible policy π_θ

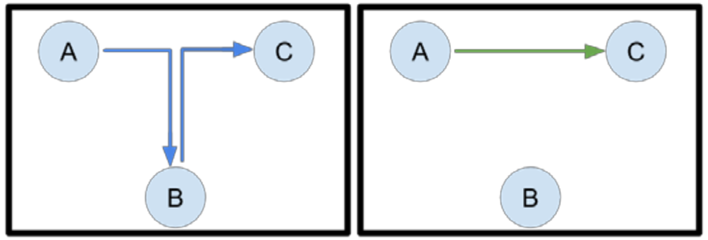
not necessarily obvious what this means

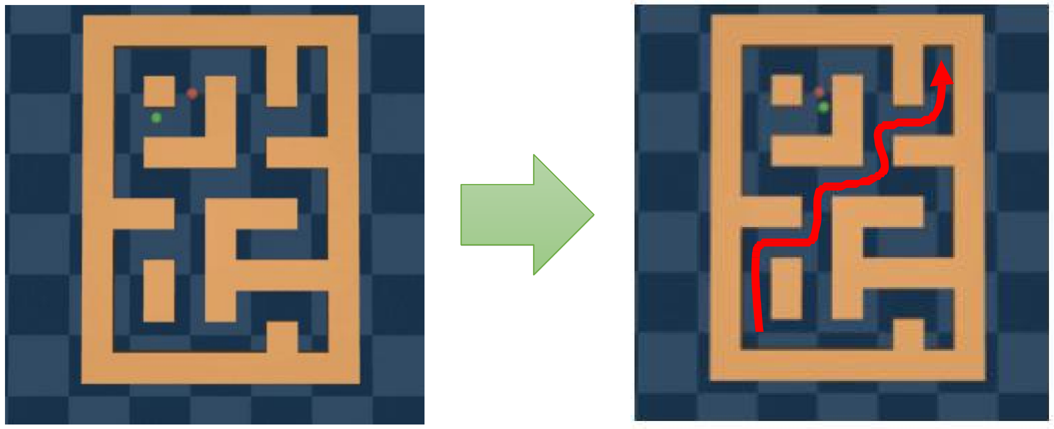
How is this even possible?

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



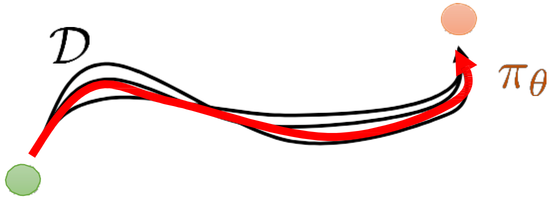


What do we expect offline RL methods to do?

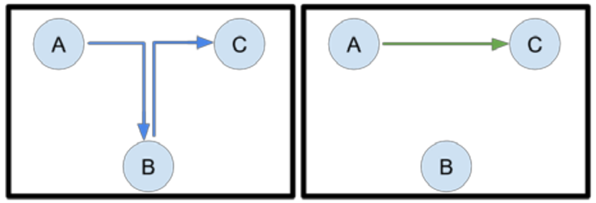
Bad intuition: it's like imitation learning

Though it can be shown to be **provably** better than imitation learning even with optimal data, under some structural assumptions!

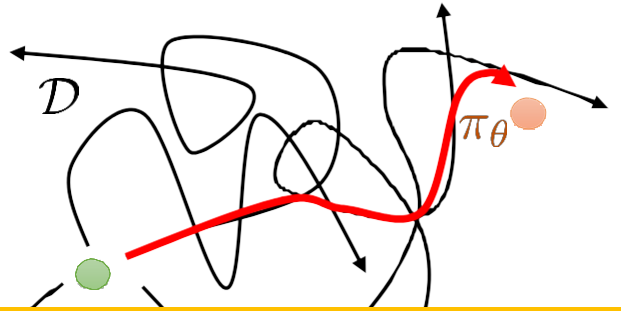
See: Kumar, Hong, Singh, Levine. Should I Run Offline Reinforcement Learning or Behavioral Cloning?



Better intuition: get order from chaos



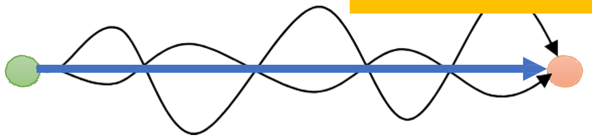
"Macro-scale" stitching



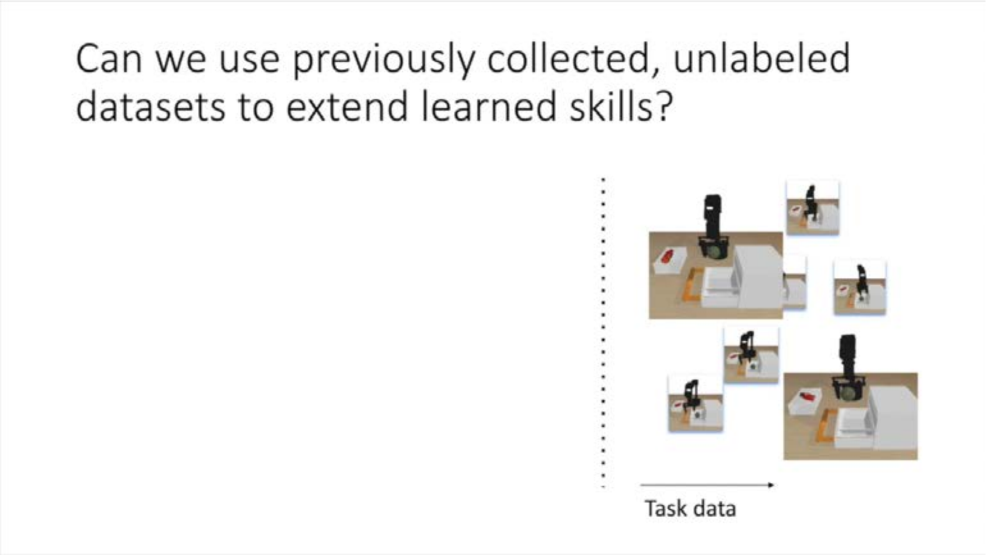
If we have algorithms that properly perform dynamic programming, we can take this idea much further and get near-optimal policies from highly suboptimal data

But this is just the clearest example!

"Micro-scale" stitching:



A vivid example



training task

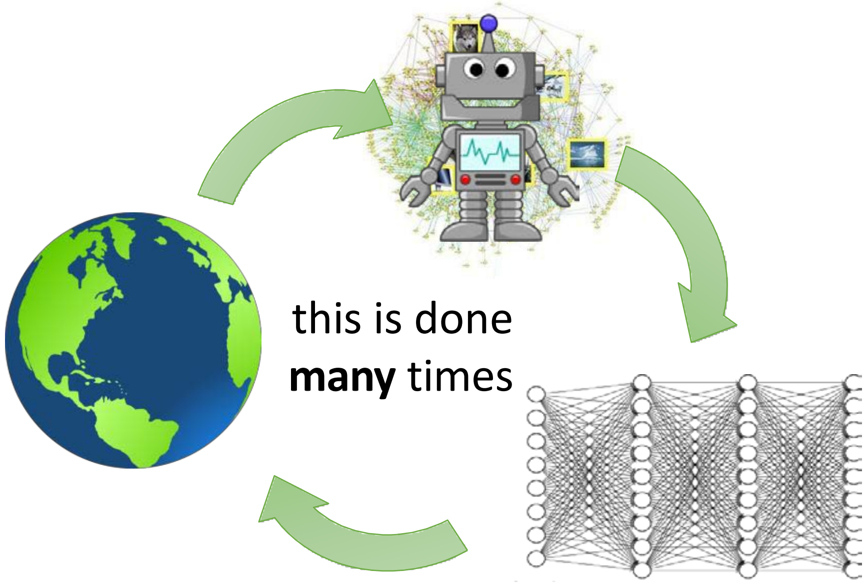
closed drawer

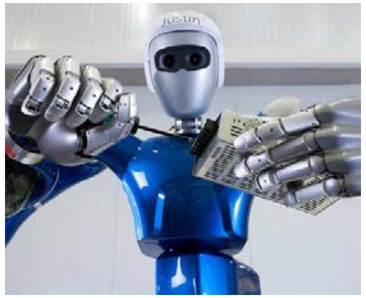
blocked by drawer

blocked by object

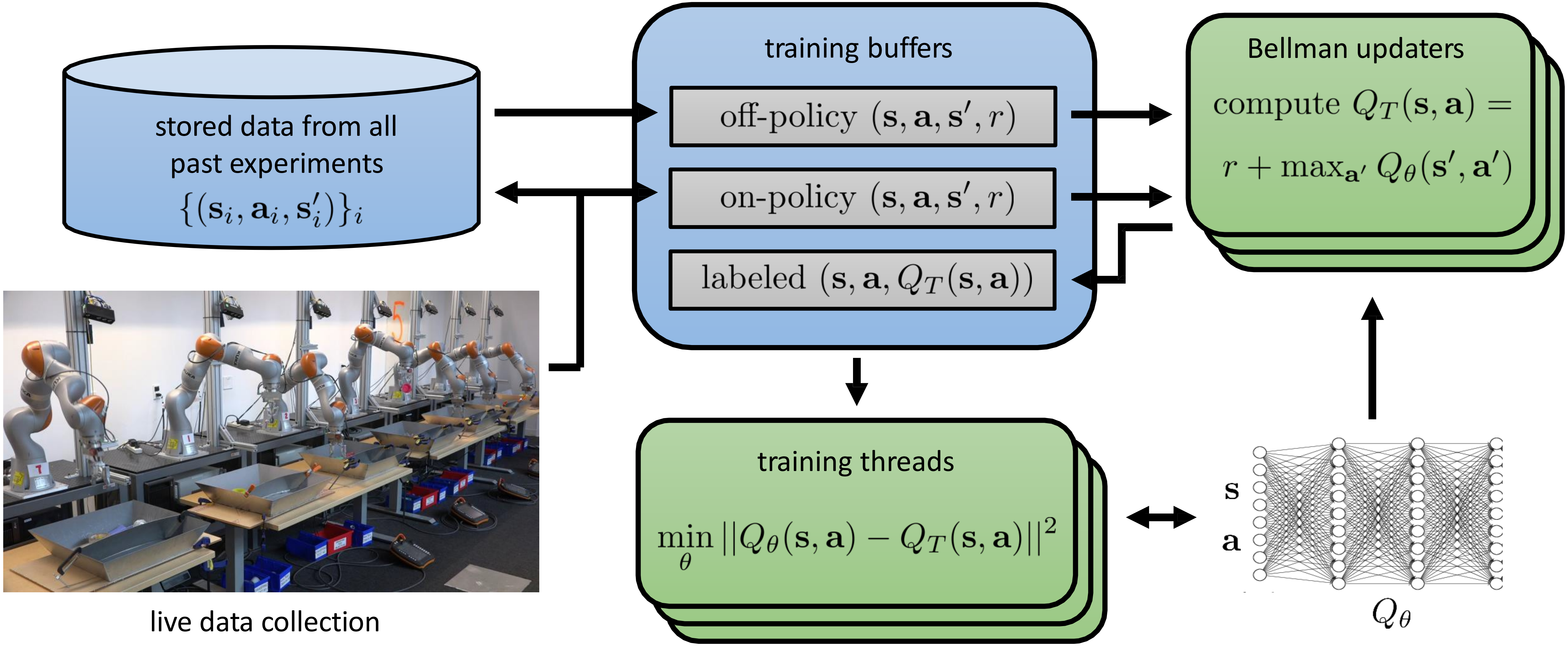
Singh, Yu, Yang, Zhang, Kumar, Levine. **COG: Connecting New Skills to Past Experience with Offline Reinforcement Learning.** '20

Why should we care?





Does it work?



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?

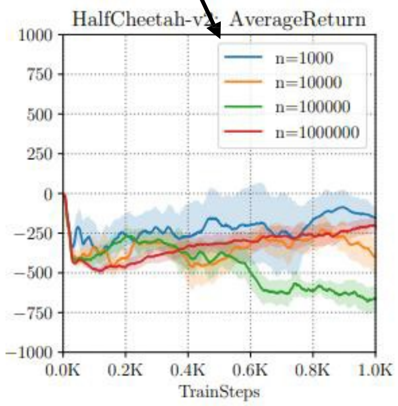
Method	Dataset	Success	Failure
Offline QT-Opt	580k offline	87%	13%
Finetuned QT-Opt	580k offline + 28k online	**96%**	**4%**

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

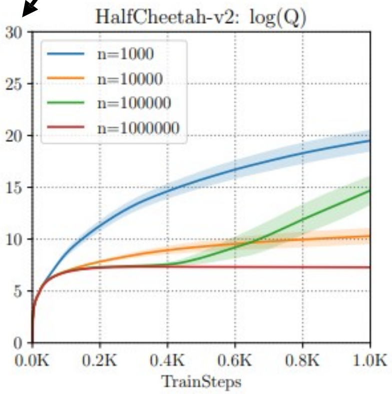
Why is offline RL hard?

amount of data

log scale (massive overestimation)



how well it does



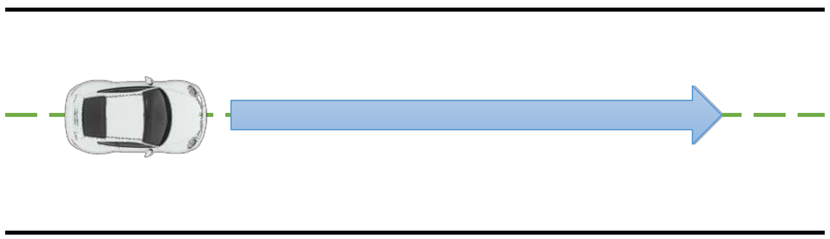
how well it *thinks* it does (Q-values)

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

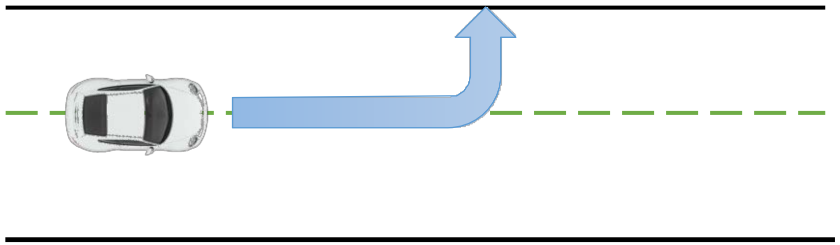
Why is offline RL hard?

Fundamental problem: counterfactual queries

Training data



What the policy wants to do



Is this good? Bad?

How do we know if we didn’t see it in the data?

Online RL algorithms don’t have to handle this, because they can simply **try** this action and see what happens

Offline RL methods must somehow account for these unseen (“out-of-distribution”) actions, ideally in a safe way

…while still making use of generalization to come up with behaviors that are better than the best thing seen in the data!

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

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?

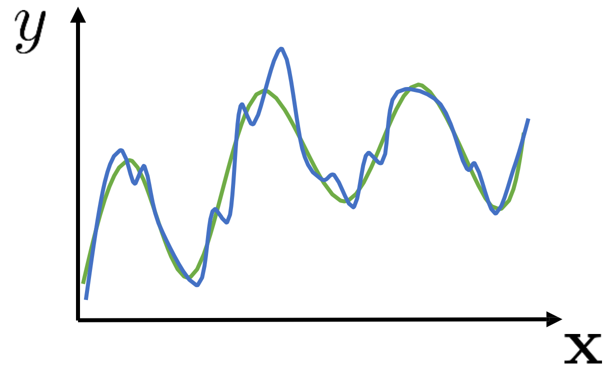
$E_{\mathbf{x}\sim p(\mathbf{x}),y\sim p(y|\mathbf{x})}\left[(f_\theta(\mathbf{x})-y)^2\right]$ is low

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

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

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?

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

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

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

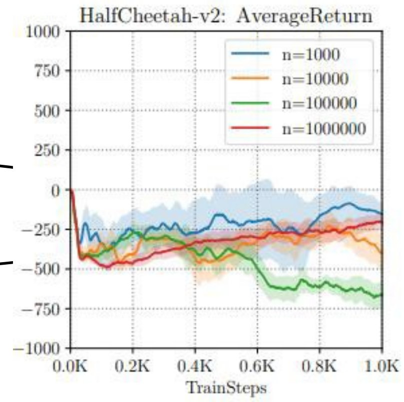
target value

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

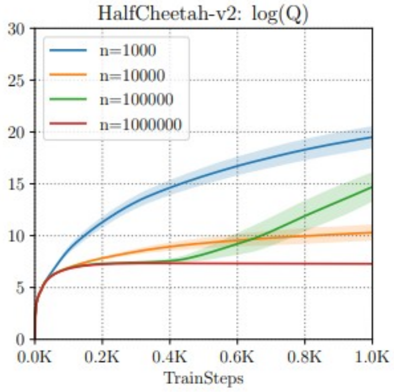
how often does *that* happen?

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

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



how well it does

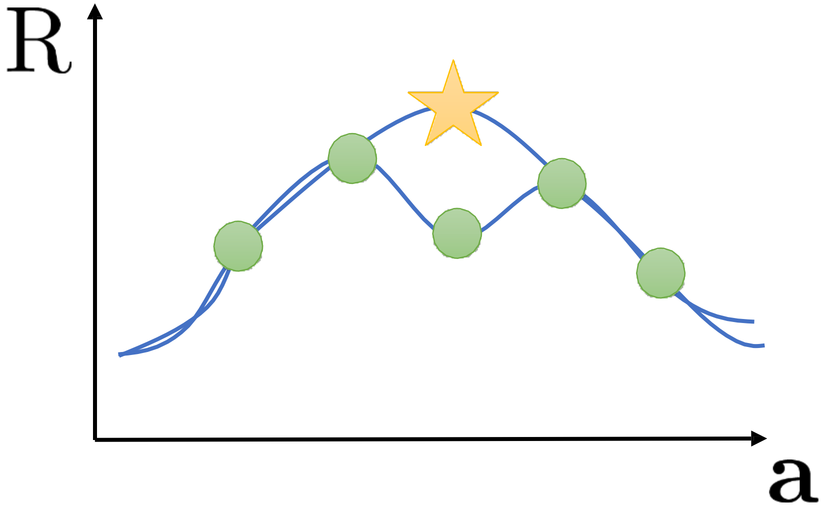


how well it *thinks* it does (Q-values)

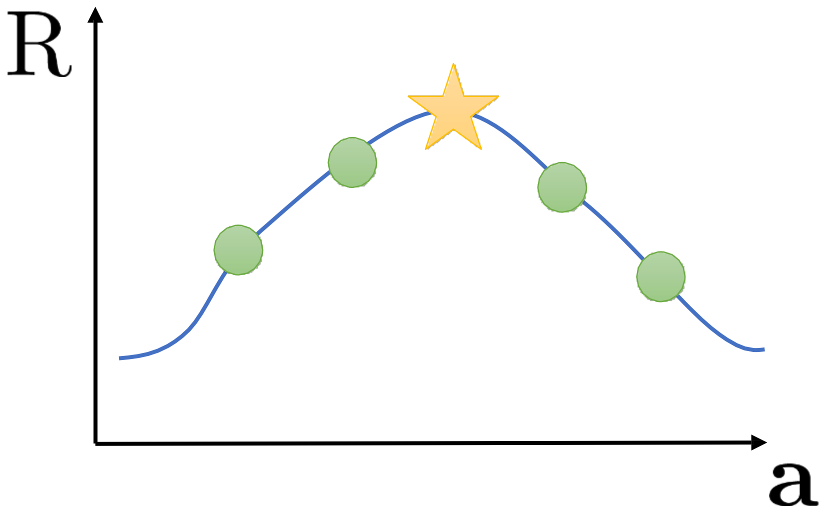
Kumar, Fu, Tucker, Levine. **Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction.** NeurIPS '19

Issues with generalization are not corrected

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

Batch RL via Importance Sampling

Offline RL with policy gradients

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

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

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

requires sampling from π_θ!

what if we only have samples from π_β?

importance sampling:

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

Offline RL with policy gradients

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

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

this is exponential in T

weights likely to be degenerate as T becomes large

can we fix this?

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \underbrace{\cancel{\left(\prod_{t'=0}^{t-1} \frac{\pi_\theta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})}{\pi_\beta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})}\right)}}_{\substack{\text{accounts for difference in probability of landing in } \mathbf{s}_{t,i} \\ \text{we have } \mathbf{s}_t \sim d^{\pi_\beta}(\mathbf{s}_t) \text{, but want } \mathbf{s}_t \sim d^{\pi_\theta}(\mathbf{s}_t)}} \nabla_\theta \gamma^t \log \pi_\theta(\mathbf{a}_{t,i}|\mathbf{s}_{t,i}) \boxed{\underbrace{\left(\prod_{t'=t}^{T} \frac{\pi_\theta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})}{\pi_\beta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})}\right)}_{\text{accounts for having the incorrect } \hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})} \hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})}$$

why is this a reasonable approximation?

Estimating the returns

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

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

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

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

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

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

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

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

this is exponential!

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

model or function approximator

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

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

Batch RL via Linear Fitted Value Functions

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

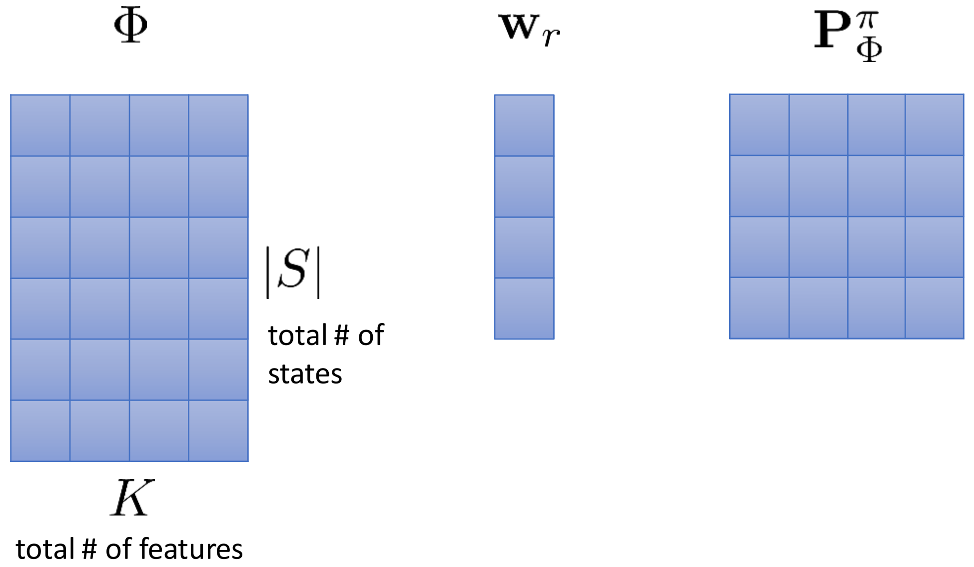
Warmup: linear models

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: $\Phi \mathbf{P}_\Phi \approx \mathbf{P}^\pi \Phi$

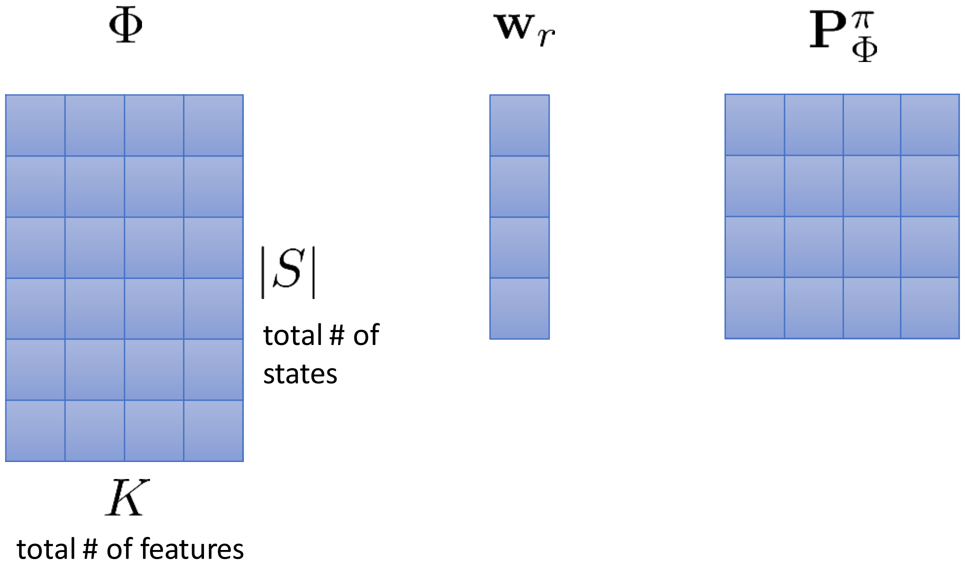
   least squares: $\mathbf{P}_\Phi = (\Phi^T \Phi)^{-1} \Phi \mathbf{P}^\pi \Phi$

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



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

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

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)

material adapted from Ron Parr

Doing it all with samples

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

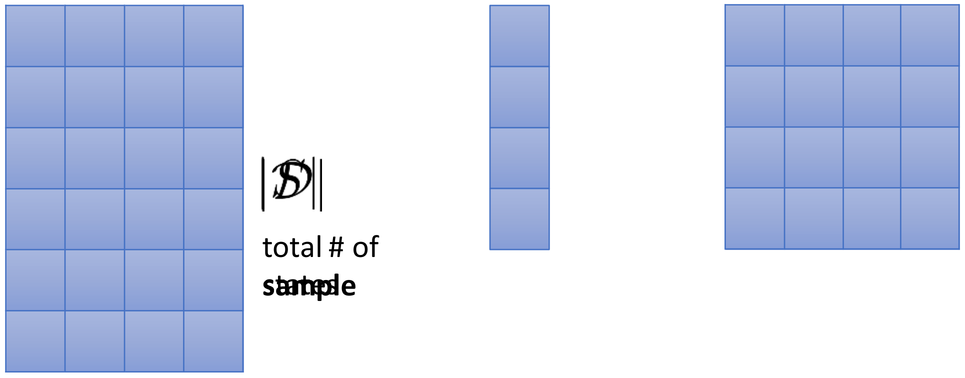
$$\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$$

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

replace with Φ'

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

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



K

total # of features

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

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

Improving the policy

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

or just do these together with LSTD!

~~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!

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

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

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

replace with Φ'

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

this requires samples from π!

Least-squares policy iteration (LSPI)

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

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

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

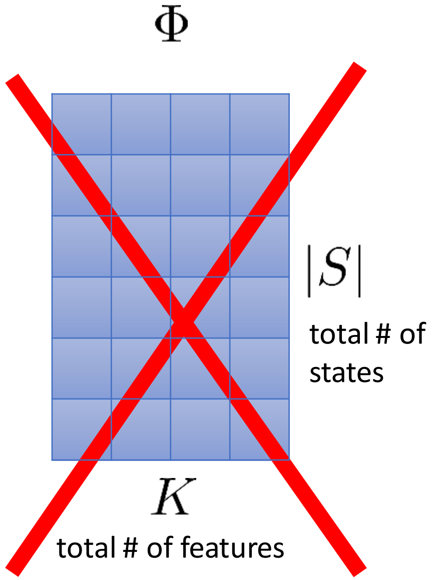
$$\vec{\mathbf{r}}_i = r(\mathbf{s}_i, \mathbf{a}_i)$$

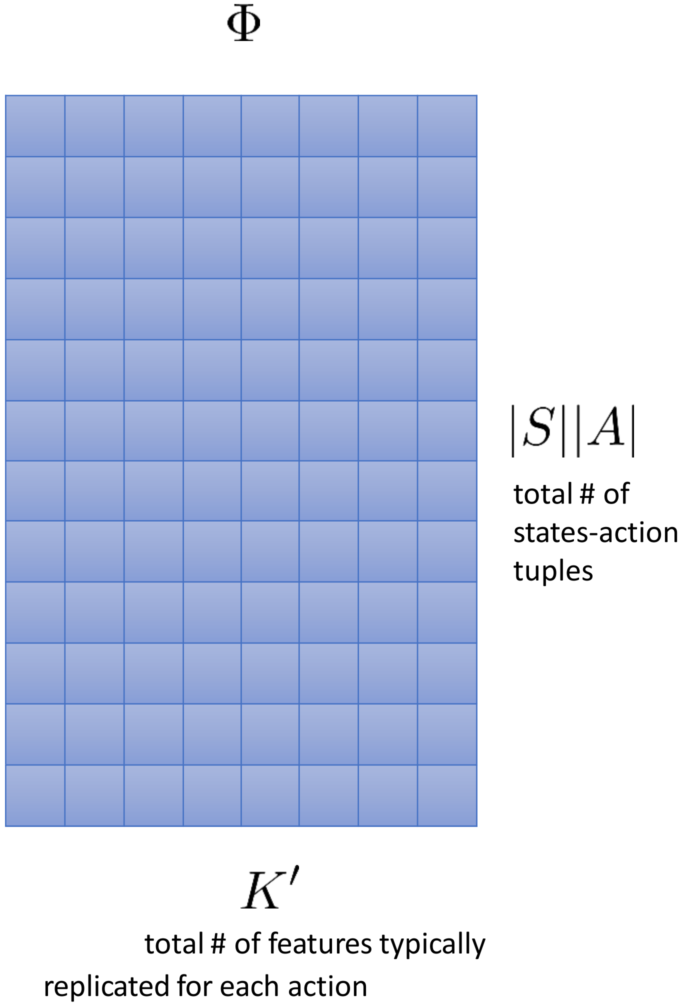
$$\Phi'_i = \phi(\mathbf{s}'_i, \pi(\mathbf{s}'_i))$$

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

LSPI:

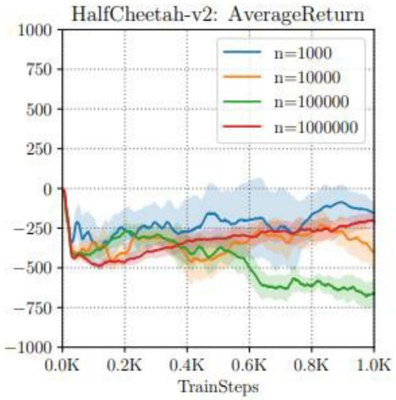
1. compute $\mathbf{w}_Q$ for π_k
2. $\pi_{k+1}(\mathbf{s}) = \arg\max_{\mathbf{a}} \phi(\mathbf{s}, \mathbf{a})\mathbf{w}_Q$
3. Set $\Phi'_i = \phi(\mathbf{s}'_i, \pi_{k+1}(\mathbf{s}'_i))$



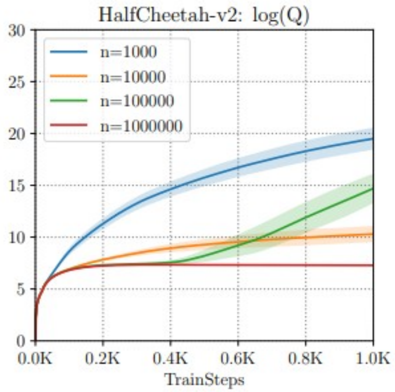


material adapted from Ron Parr

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