Behavior Cloning and Offline Reinforcement Learning

模仿学习与离线强化学习

朱圆恒

国科大 强化学习课程第10讲

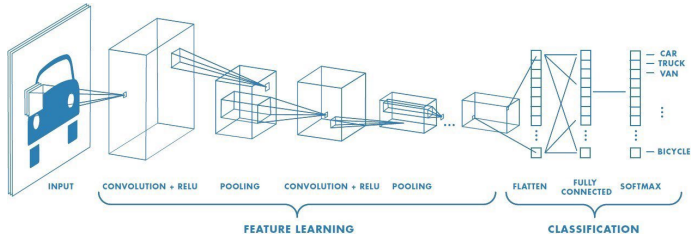
Content

- What is Behavior Cloning
- What is Offline Reinforcement Learning
- Challenges of Offline Reinforcement Learning
- Solution Approaches
  - Policy Constraint
  - Conservative Methods
  - Implicit Policy Constraint
  - Variant Behavior Cloning

Challenge of Reinforcement Learning

Motivation

Why is Deep Learning so successful in real-world applications?



Why is this not the same for Reinforcement Learning?

- Active data collection can be costly, impossible or unethical
  - Medical treatment,
  - Autonomous driving, …
- This hinders the application of active reinforcement learning to real world problems
- Simulators are hard to design and the resulting observations inferior to real world data

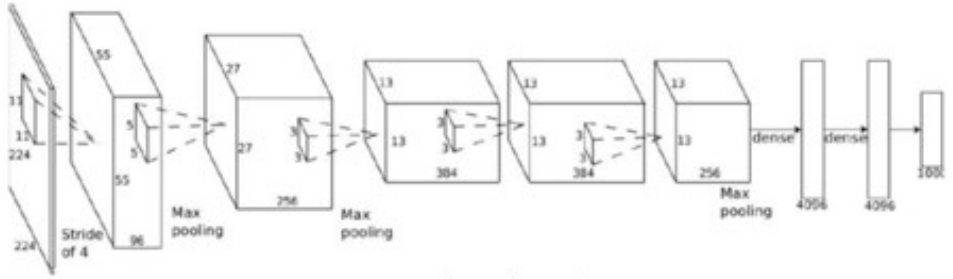
→ Need for data-driven reinforcement learning that uses passively collected real world data

[1] http://vladlen.info/projects/scene-understanding/
[2] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Can we do **supervised learning** in decision-making process?

- Yes! Behavior Cloning / Imitation Learning
- Key idea: Train policy using supervised learning with **collected data**

$\mathbf{o}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

$\mathbf{a}_t$

$\mathbf{o}_t$

$\mathbf{a}_t$

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

Behavior Cloning / Imitation Learning

- Key idea: Train policy using supervised learning

Data: Given trajectories collected by an expert

"*demonstrations*" $\mathcal{D} := \{(s_1, a_1, \ldots, s_T)\}$

Training: Train policy to mimic expert: $\min_\theta -\mathbb{E}_{(s,a)\sim\mathcal{D}}[\log \pi_\theta(a \mid s)]$

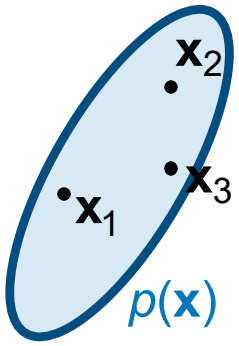
i.e. minimize cross-entropy loss or ℓ_2 loss between predicted & expert actions.

What can go wrong in imitation learning?

What can go wrong in imitation learning?

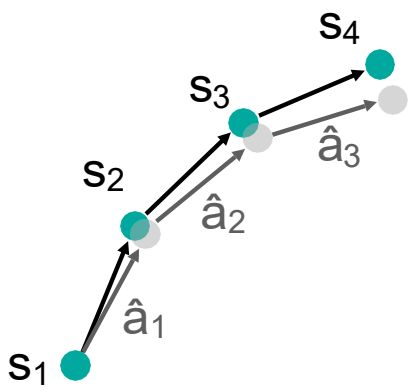
- **1. Compounding errors**

Supervised learning



Inputs independent of predicted labels $\hat{y}$

Supervised learning of behavior



Predicted actions affect next state.

Errors can lead to drift away from the data distribution!

Errors can then compound!

$$p_{expert}(\boldsymbol{s}) \neq p_{\pi}(\boldsymbol{s})$$

states visited by expert

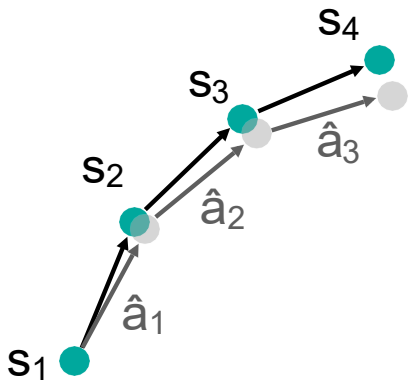
states visited by learned policy

"covariate shift"

What can go wrong in imitation learning?

- **1. Compounding errors**

Supervised learning of behavior



Predicted actions affect next state.

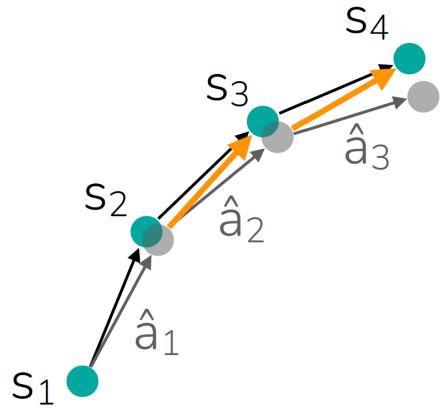
Errors can lead to drift away from the data distribution!

Errors can then compound!

$$p_{expert}(\boldsymbol{s}) \neq p_{\pi}(\boldsymbol{s})$$

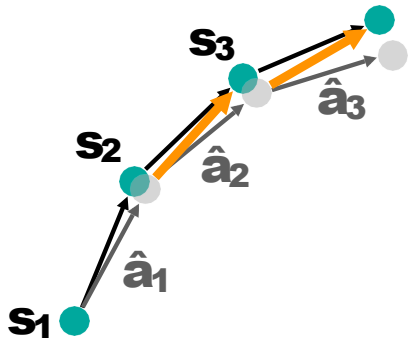
Solutions?

1. Collect A LOT of demo data & hope for the best.
2. Collect **corrective behavior data**



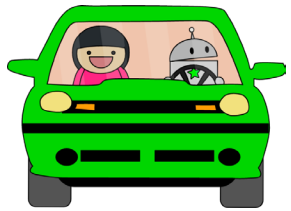
Addressing Compounding Errors with DAgger

- Compounding error caused by $p_{expert}(\boldsymbol{s}) \neq p_{\pi}(\boldsymbol{s})$
- **Solution**: Collect corrective behavior data to make $p_{expert}(\boldsymbol{s})$ close to$p_{\pi}(\boldsymbol{s})$



1. Roll-out learned policy π_θ: $s'_1, \hat{a}_1, \ldots, s'_T$
2. Query expert action at visited states $a^* \sim \pi_{expert}(\cdot|s')$
3. Aggregate corrections with existing data $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s', a^*)\}$
4. Update policy $\min_\theta \mathcal{L}(\pi_\theta, \mathcal{D})$

"dataset aggregation" (DAgger)



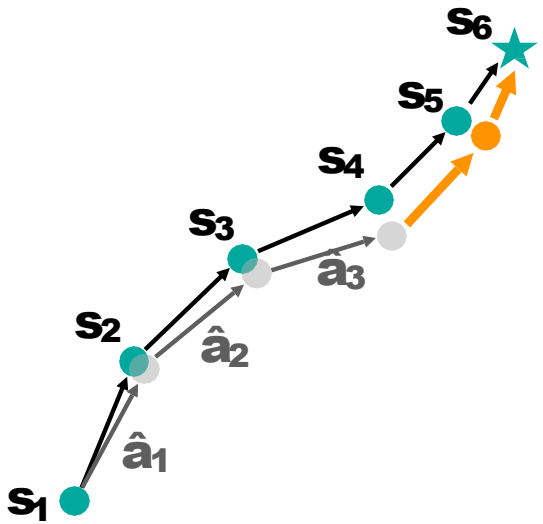
+ data-efficient way to learn from an expert

- can be challenging to query expert when agent has control
  Is there another way to collect corrective data?

A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, AISTATS'11

Addressing Compounding Errors with **DAgger**

- **Alternative**: Collect corrective behavior data while ***taking full control***



1. Start to roll-out learned policy π_θ: $s'_1, \hat{a}_1, \ldots, s'_t$
2. Expert intervenes at time t when policy makes mistake
3. Expert provides (partial) demonstration $s'_t, a^*_t, \ldots, s'_T$
4. Aggregate new demos with existing data $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s'_i, a^*_i)\}; i \geq t$
5. Update policy $\min_\theta \mathcal{L}(\pi_\theta, \mathcal{D})$

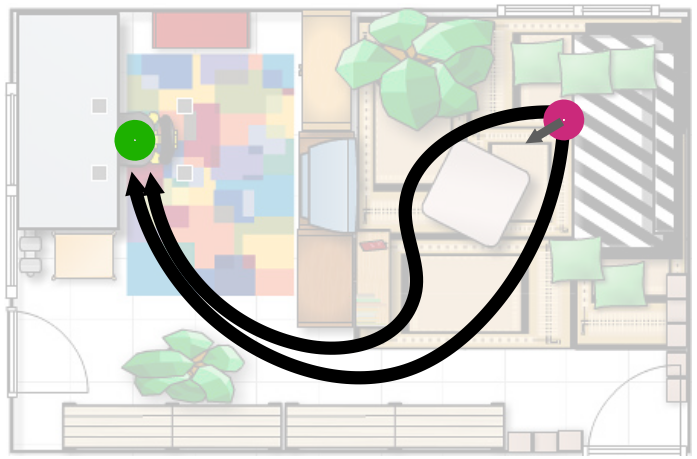
"human gated DAgger"

+ (much) easier interface for providing corrections

- can be hard to catch mistakes quickly in some application domains

What can go wrong in imitation learning?

- **2. Multimodal demonstration data**

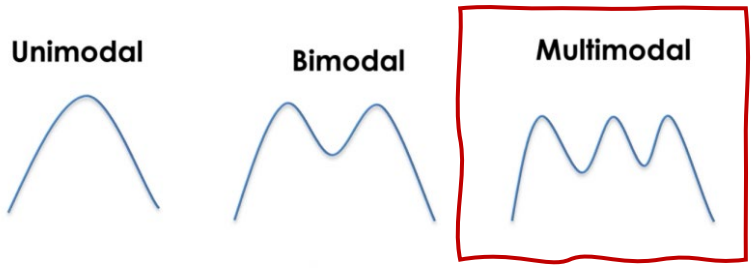


- The data takes two different actions **here**!
- If we use ℓ_2 loss, what action will the agent take?
- When does this happen in practice? All time!
- Esp. when data collected by multiple people.

Solution? Use expressive distribution class to fit p(a | s).

– capture all modes of the data distribution

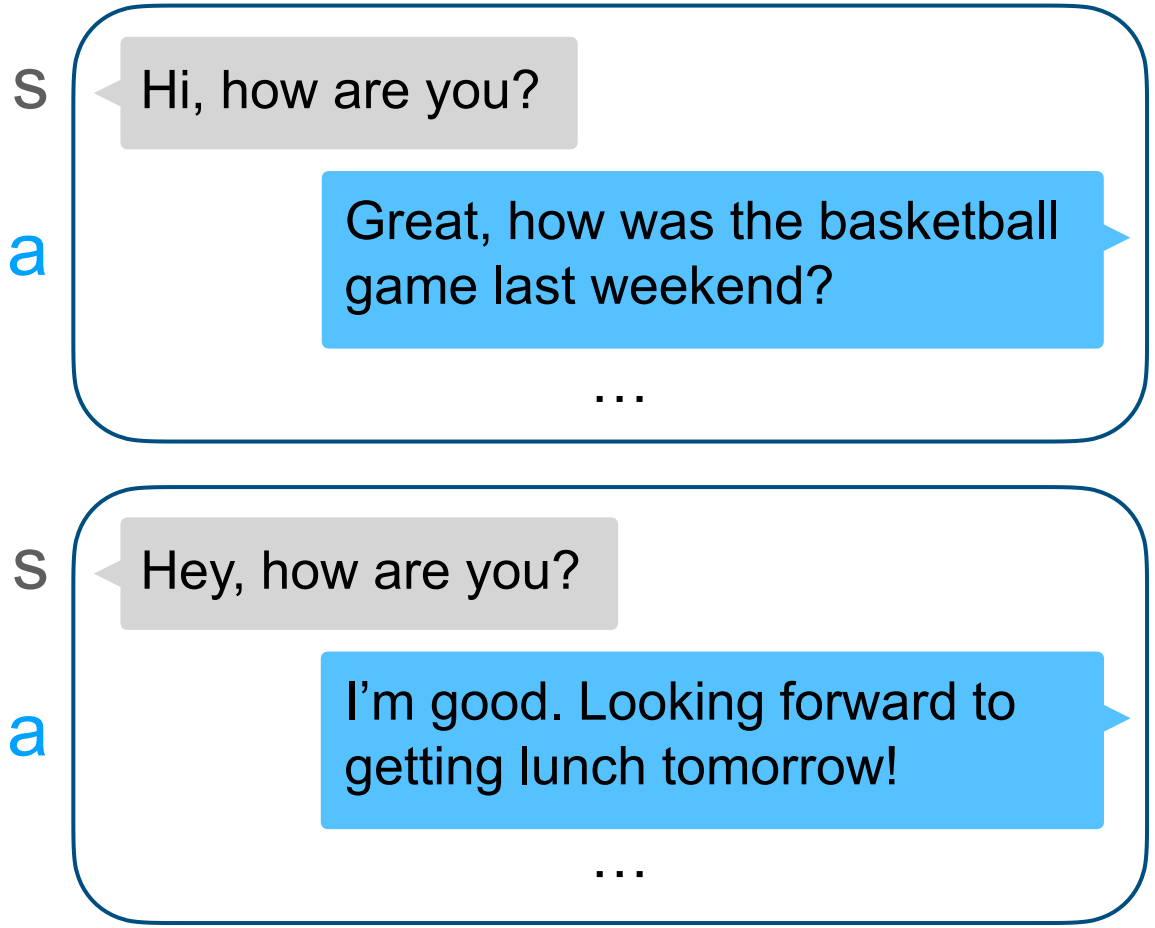
– e.g. Gaussian mixture, Categorical, VAEs, diffusion models



What can go wrong in imitation learning?

- **3. Mismatch in observability between expert & agent**

Example demos scraped from conversations:



Problem: Expert has more information than is observed by the agent.

Impossible to accurately imitate.

Solutions:

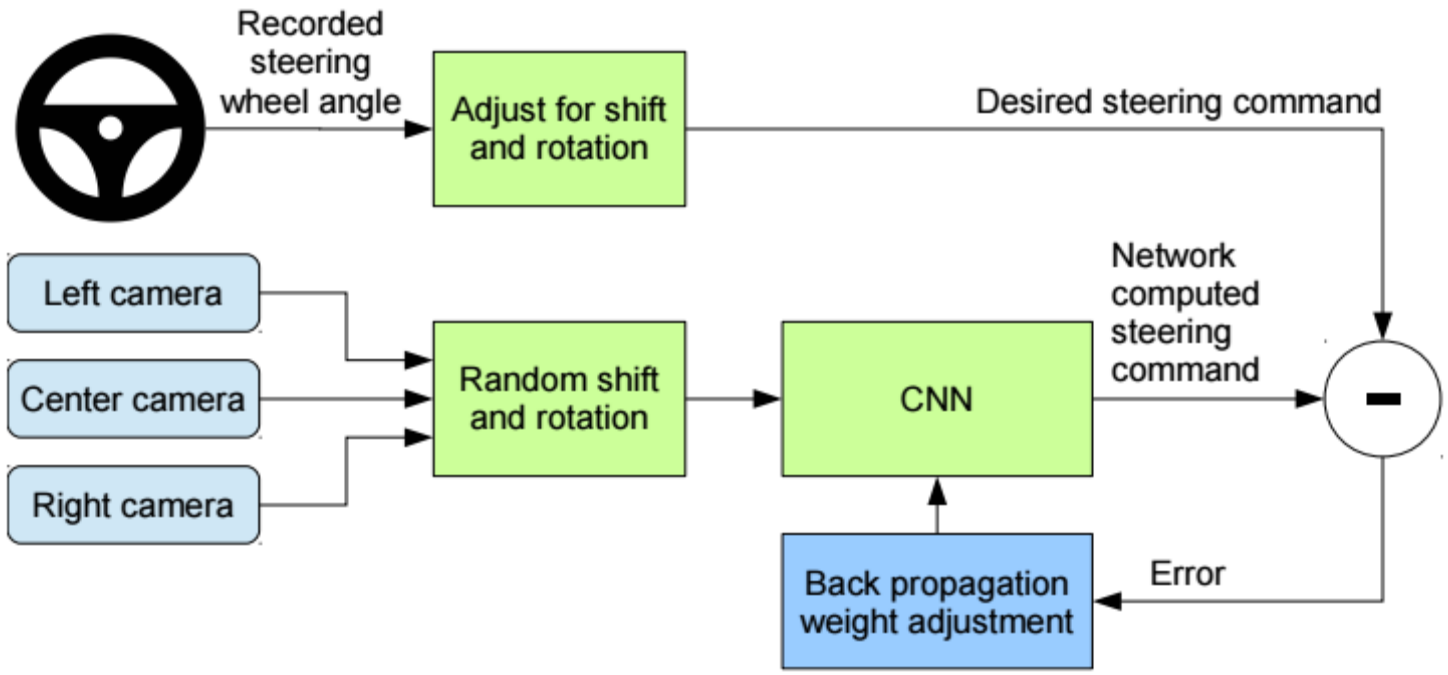
- Give as much contextual information to the agent as possible.
- Collect demos in a way that gives expert same information as agent.

Case Study: Nvidia 2016

- End to End Learning for Self-Driving Cars

Case Study: Nvidia 2016

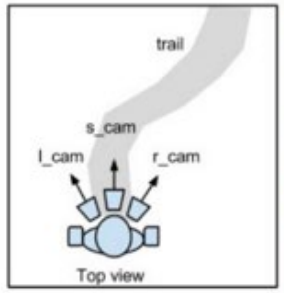
- End to End Learning for Self-Driving Cars
- 72 hours of driving data as training data



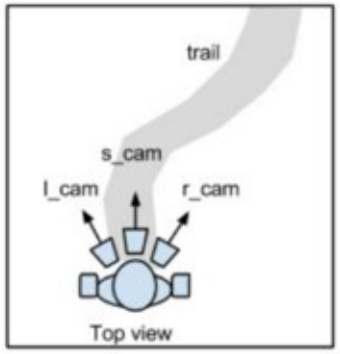
End-to-End Deep Learning for Self-Driving Cars
Mariusz Bojarski, Ben Firner, Beat Flepp, Larry Jackel, Urs Muller, Karol Zieba and Davide Del Testa | August 17, 2016

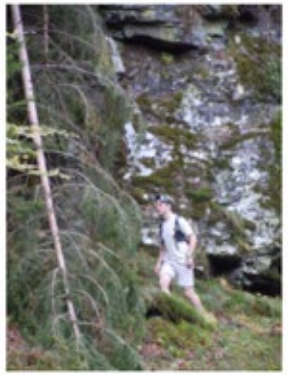
Case Study: Trail following as a classification

- How to train a drone to fly



Human walking data





A. Giusti, et al. A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robot. IEEE trans on Robotics and Automation

Case Study: Trail following as a classification

- How to train a drone to fly

A. Giusti, et al. A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robot. IEEE trans on Robotics and Automation

Is Imitation Learning All You Need?

- A simple & powerful framework for learning behavior!
- But!
  - Collecting expert demonstrations can be difficult or impossible in some scenarios
  - Learned behavior will never be better than expert
  - Does not provide a framework for learning from experience, indirect feedback
  - Can agents learn autonomously, from their own mistakes?

We need a better way to learn from collected data:

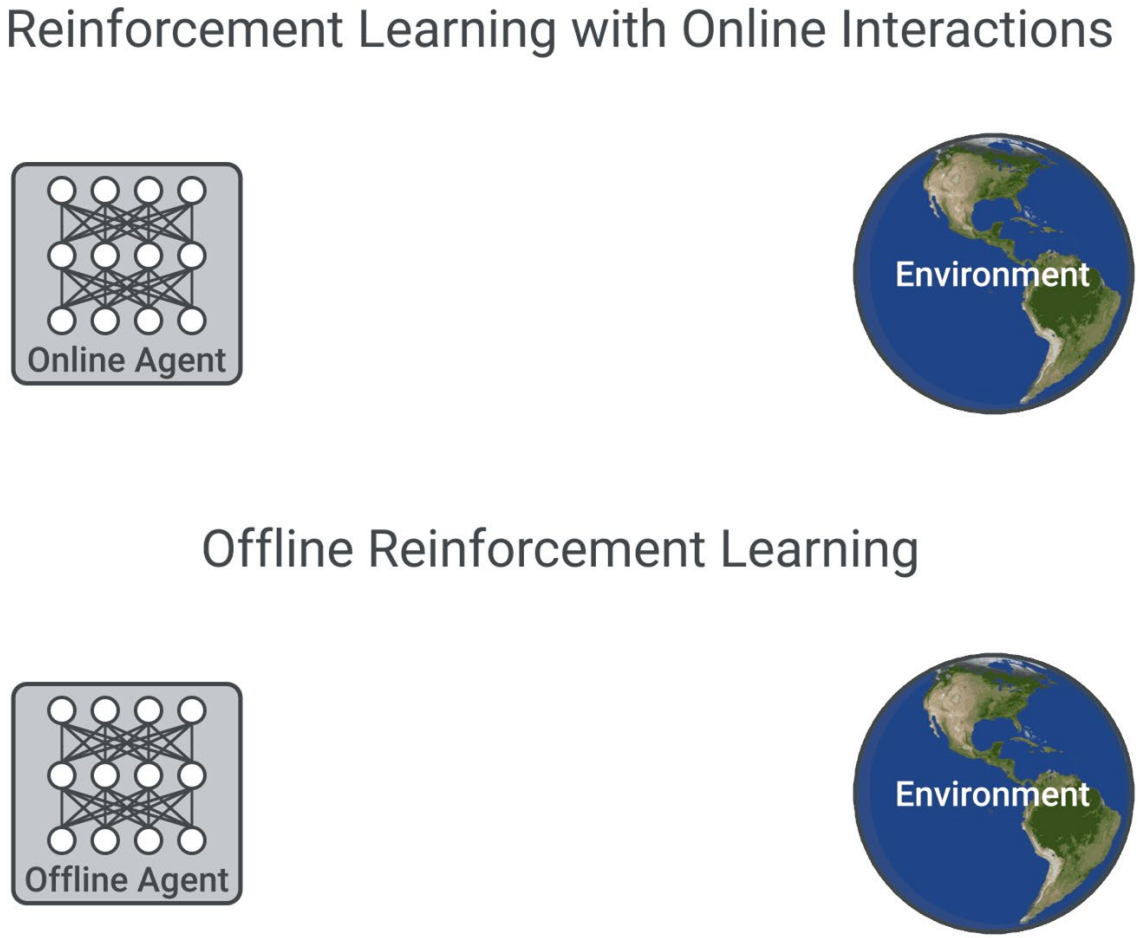
Offline RL!

Offline Reinforcement Learning

Motivation

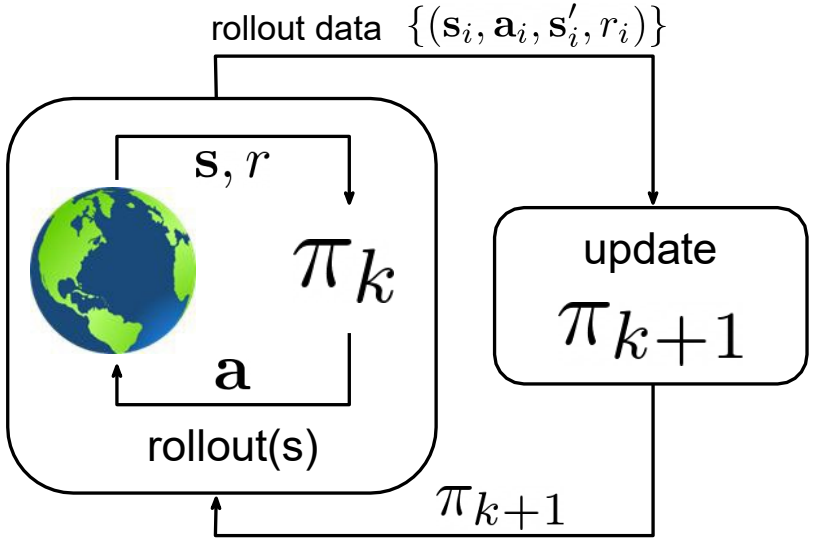
Offline RL can help:

- Pretrain the agents on existing logged data.
- Evaluate RL algorithms on the basis of exploitation alone on common datasets.
- Deliver real-world impact.



Offline Reinforcement Learning

Online (On-Policy) Reinforcement Learning

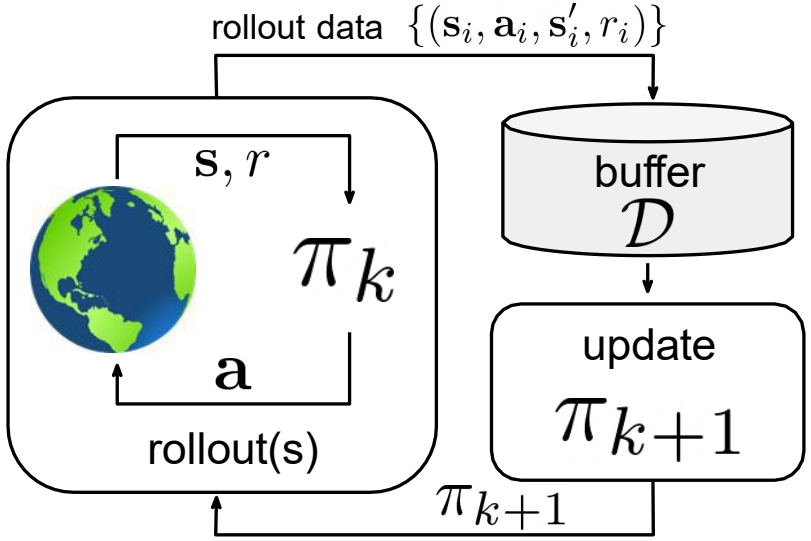


- π_k is updated with streaming data collected by π_k itself
- The transitions (s, a, r, s') are collected using the current policy π_k
- Policy and transitions are perfectly related

Levine et al.: "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems"

Offline Reinforcement Learning

Off-Policy Reinforcement Learning

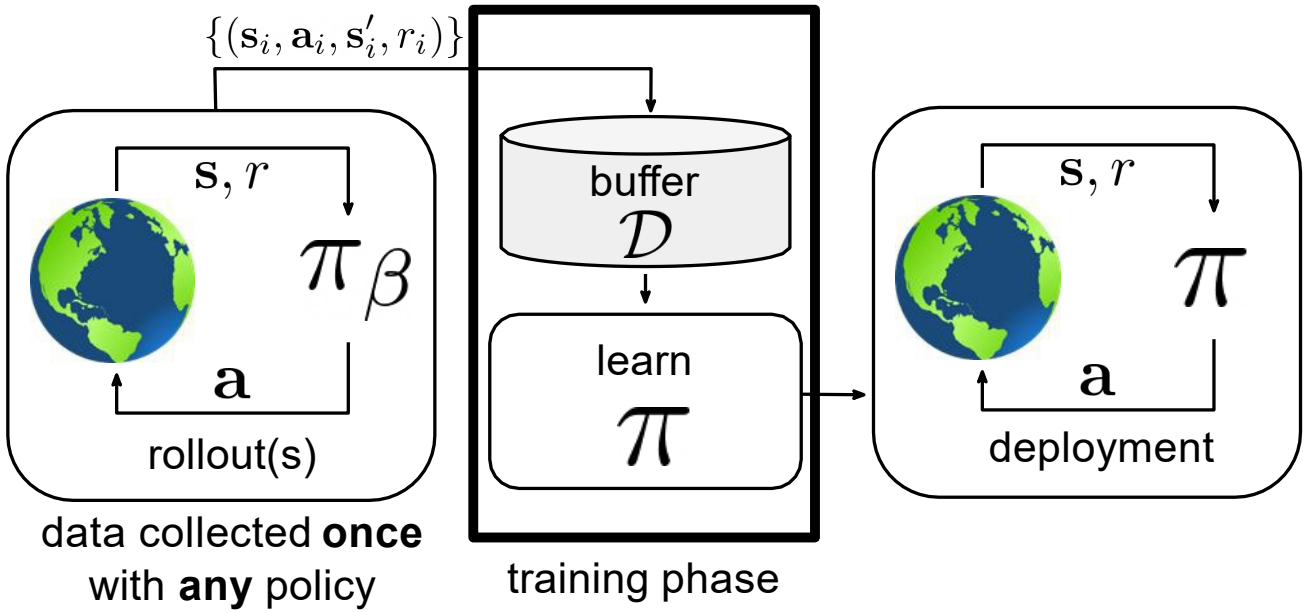


- New transitions (s, a, r, s') are appended to a buffer $\mathcal{D}$
  - $\mathcal{D}$ consist of samples collected under the policies $\pi_0, \pi_1, \ldots, \pi_k$ (with, e.g., an ϵ-greedy sampling)
- All the data from $\mathcal{D}$ is used to train an updated policy π_{k+1} (i.e., transitions are sampled from $\mathcal{D}$)
- Policy and transitions are dependent (strength depends on the size of $\mathcal{D}$ and ϵ)

Levine et al.: "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems"

Offline Reinforcement Learning

Fully Offline Reinforcement Learning



- Offline RL uses a dataset $\mathcal{D}$ collected by some behavior policy π_β
  - π_β is potentially (or often assumed to be) unknown
- $\mathcal{D}$ is **collected once** and **not changed** during training
  - Transitions are sampled from $\mathcal{D}$
  - No interaction with the MDP; Policy is deployed after being fully trained.
- Policy and transitions are independent

Levine et al.: "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems"

Offline Reinforcement Learning

Offline Reinforcement Learning vs. Behavioural Cloning

Behavioural Cloning

- Scenario: Data $\mathcal{D}$ collected from a behavioural agent π_β
- Goal: Learn policyπ_β from $\mathcal{D}$ with supervised learning
  1. Needs expert data to get a good policy
  2. Unexpectedly high error bound due to generalization that usually leads to poor performance in practice

Offline Reinforcement Learning

- Scenario: Data $\mathcal{D}$ collected from a behavioural agent π_β
- Goal: Learn the best possible policy π using only data $\mathcal{D}$

→ Does not necessarily need expert data

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

Proof. The proof follows from Theorem 2.1 from Ross et al. (2011) using the 0-1 loss, and the bound is the best possible bound following the example from Ross and Bagnell (2010). □

Interestingly, if we allow for additional data collection, where we follow the learned policy $\pi(\mathbf{a}|\mathbf{s})$ to gather additional states $\mathbf{s} \sim d^\pi(\mathbf{s})$, and then access optimal action labels for these new *on-policy* states, the best possible bound becomes substantially better:

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

Proof. The proof follows from Theorem 3.2 from Ross et al. (2011). This is the best possible bound, because the probability of a mistake at any time step is at least ϵ, and $\sum_{t=1}^{H} \epsilon = H\epsilon$. □

Levine et al.: Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems.

OfflineRL Dataset

- D4RL: Datasets for deep data-driven reinforcement learning, arXiv, 2020.

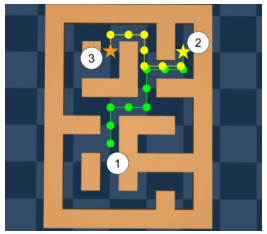
Domain	Task Name	Controller Type	# Samples
Maze2D	maze2d-umaze	Planner	10^6
	maze2d-medium	Planner	$2 * 10^6$
	maze2d-large	Planner	$4 * 10^6$
AntMaze	antmaze-umaze	Planner	10^6
	antmaze-umaze-diverse	Planner	10^6
	antmaze-medium-play	Planner	10^6
	antmaze-medium-diverse	Planner	10^6
	antmaze-large-play	Planner	10^6
	antmaze-large-diverse	Planner	10^6
Gym-MuJoCo	hopper-random	Policy	10^6
	hopper-medium	Policy	10^6
	hopper-medium-replay	Policy	200920
	hopper-medium-expert	Policy	2×10^6
	halfcheetah-random	Policy	10^6
	halfcheetah-medium	Policy	10^6
	halfcheetah-medium-replay	Policy	101000
	halfcheetah-medium-expert	Policy	2×10^6
	walker2d-random	Policy	10^6
	walker2d-medium	Policy	10^6
	walker2d-medium-replay	Policy	100930
	walker2d-medium-expert	Policy	2×10^6

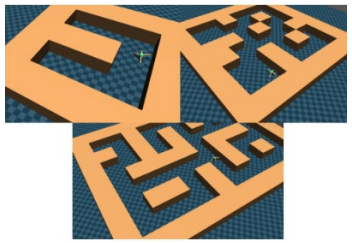
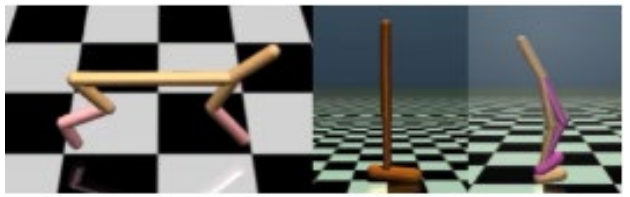
“random” random policy

“medium” partially-trained policy
(SAC+early-stopping the training)

“medium-replay” partially-trained policy
(the policy reaches the “medium” level)

“medium-expert” medium+expert demonstrations



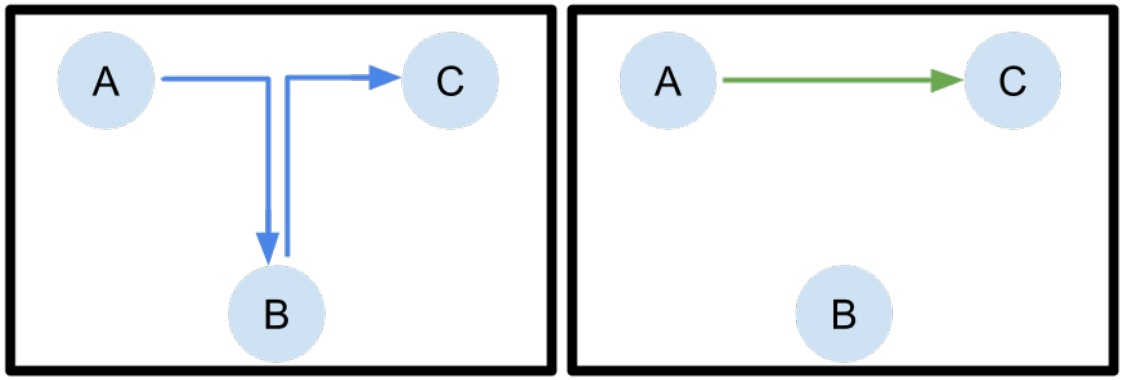



Challenges of Offline Reinforcement Learning

Offline Reinforcement Learning

Why Offline RL Should Work in Principle

- Find the good behaviour in a dataset of good and bad behaviour
- Generalize to similar states and actions
- Stich together good and bad behaviour



https://bair.berkeley.edu/blog/2020/06/25/D4RL/

D4RL:AntMaze

https://sites.google.com/view/d4rl/

Offline Reinforcement Learning

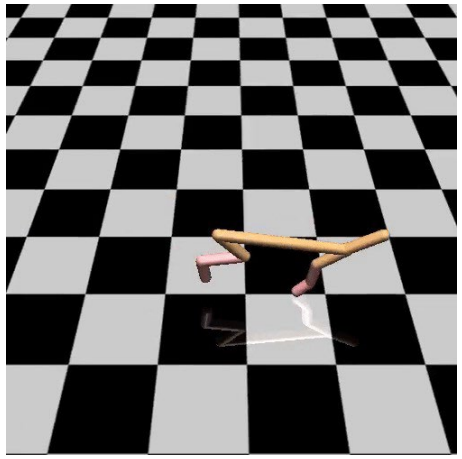
Challenges of Offline Reinforcement Learning

Question

- Can we just use off-policy methods to learn from static datasets?

Experimental Setup

- Train Soft Actor-Critic (SAC) in a fully off-policy setting
- Environment: Mujoco's HalfCheetah-v2
- Use a trained policy to generate *expert demonstrations*
  - Dataset contains successful task completions



https://bair.berkeley.edu/blog/2019/12/05/bear/

Offline Reinforcement Learning

Off-Policy Algorithms for Offline RL?

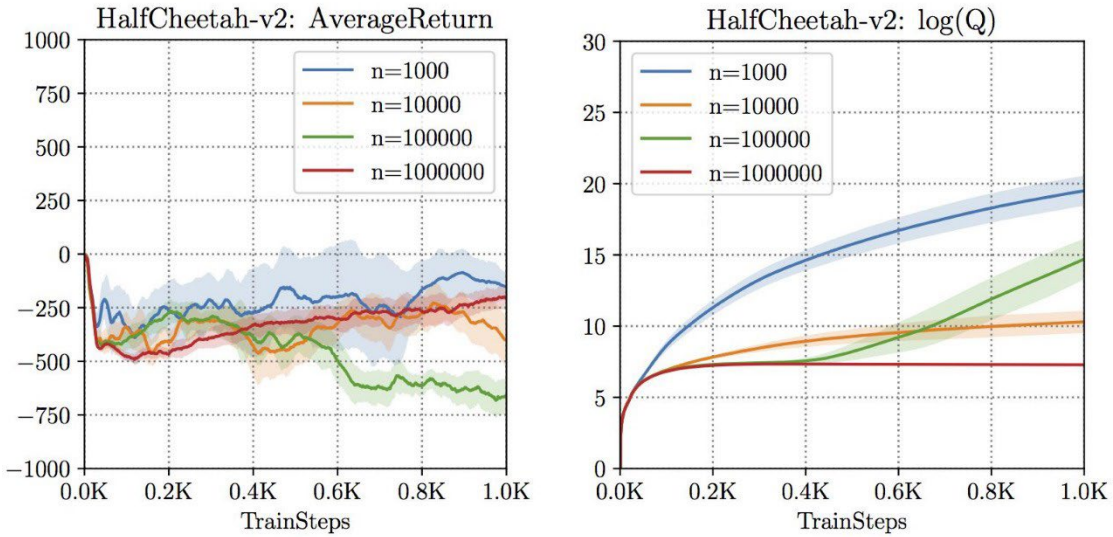
Question

- Can we just use off-policy methods to learn from static datasets? **No!**

Experimental results

- n = size of dataset
- None of the runs succeeds
- Evaluation performance deteriorates with more training (green vs. blue/orange)
  - but cannot be the main reason (see red curve)
- Q-values even diverge for some datasets

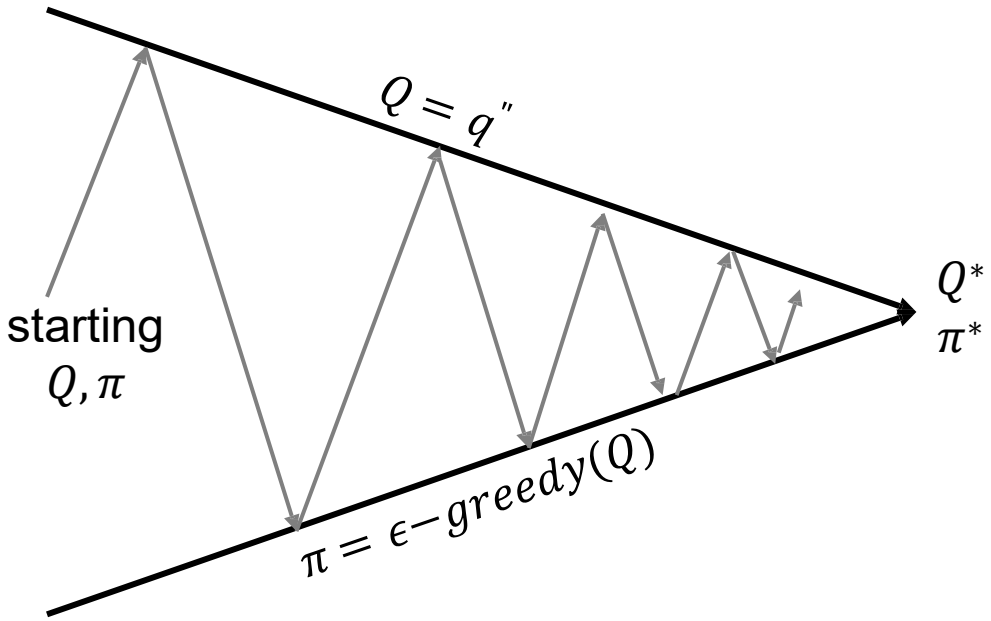
→ **Then why does it fail?**

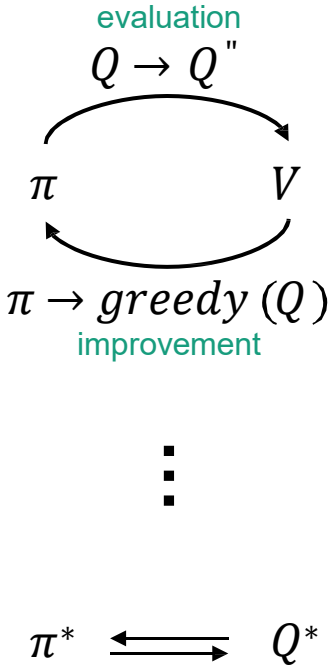


https://bair.berkeley.edu/blog/2019/12/05/bear/

Offline Reinforcement Learning

Recap: Generalized Policy Iteration (GPI)

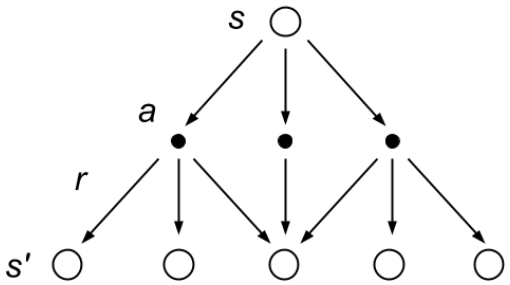




Offline Reinforcement Learning

Recap: Generalized Policy Iteration (GPI)

Policy Evaluation: For a given policy π, find Q



- The true state-action-value function satisfies the Bellman Equation:

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

- Approximate dynamic programming is used to minimize the Mean Squared Bellman Error:

$$\min_\theta \mathbb{E}_{s \sim d_\pi,\, a \sim \pi} \left[\left(Q_\theta(s,a) - T_\pi Q_\theta(s,a) \right)^2 \right]$$

Policy Improvement: For a given value function Q for π, find a better policy π_{new}

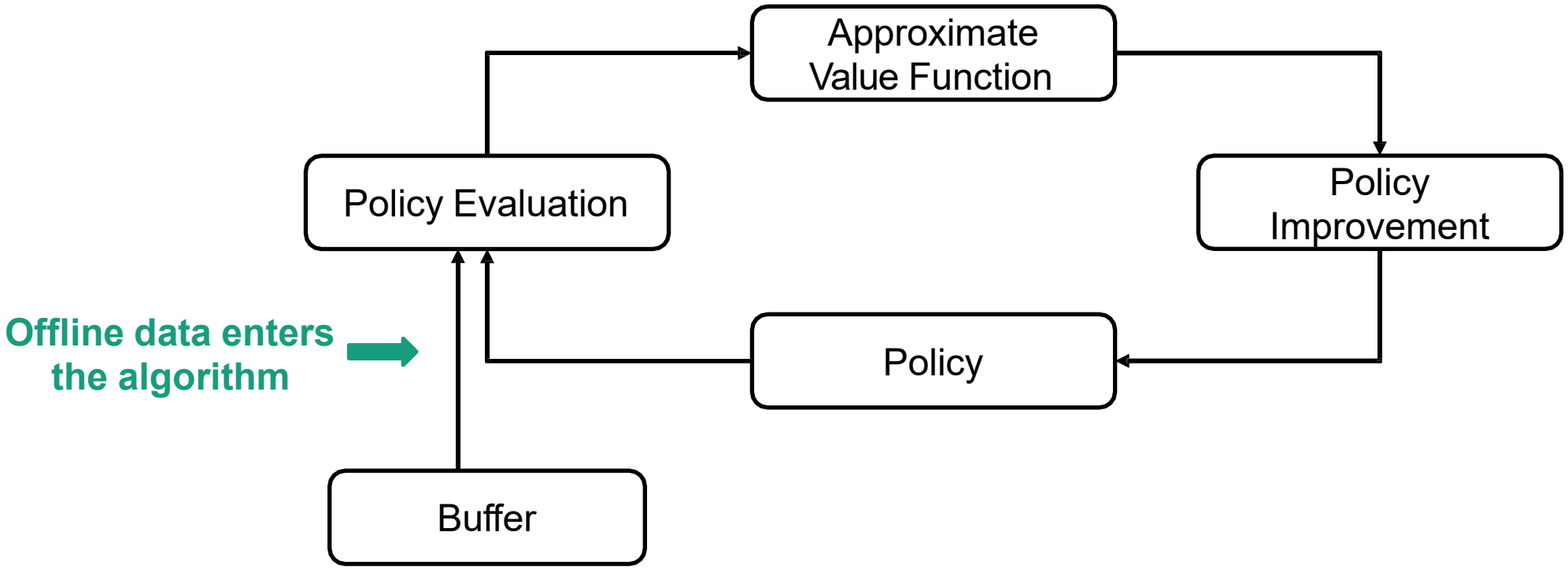
$$\pi_{new}(s) = \arg\max_a Q(s,a)$$

Policy Improvement Theorem:

- Let π and π' be two policies with $\forall\, s \in S$: $Q_\pi(s, \pi'(s)) \geq V_\pi(s)$, then $\forall\, s \in S$: $V_{\pi'}(s) \geq V_\pi(s)$

Offline Reinforcement Learning

GPI in the Offline Setting



Offline Reinforcement Learning

Offline Policy Evaluation and Distribution Shift

- What we want to do:

$$\min_{\theta} \mathbb{E}_{s\sim d_{\pi},\, a\sim\pi}\left[\left(r(s,a)+\gamma\mathbb{E}_{s'\sim p(s'|s,a),\, a'\sim\pi(s')}[Q_{\theta}(s',a')]-Q_{\theta}(s,a)\right)^{2}\right]$$

- What we would be doing naively:

$$\min_{\theta}\sum_{s\in S,a\in A}\left[\sum_{(s,a,r,s')\in\mathcal{D}|s,a}\left(r+\gamma Q_{\theta}(s',\pi(s'))-Q_{\theta}(s,a)\right)\right]^{2}$$

- Which is the empirical approximation for:

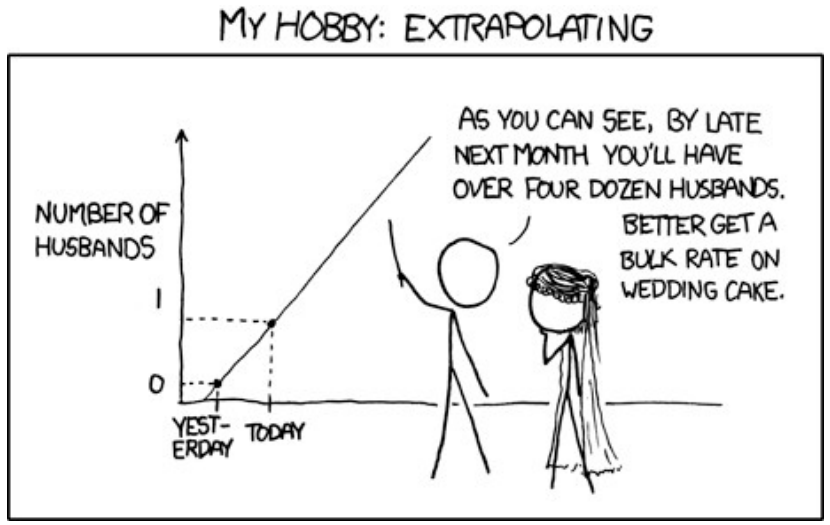
$$\min_{\theta} \mathbb{E}_{s\sim d_{\pi_{\beta}},\, a\sim\pi_{\beta}}\left[\left(r+\gamma\mathbb{E}_{s'\sim p(s'|s,a'),a'\sim\pi(s')}[Q_{\theta}(s',a')]-Q_{\theta}(s,a)\right)^{2}\right]$$

→**State and action distribution shift (the next action can be controlled)**

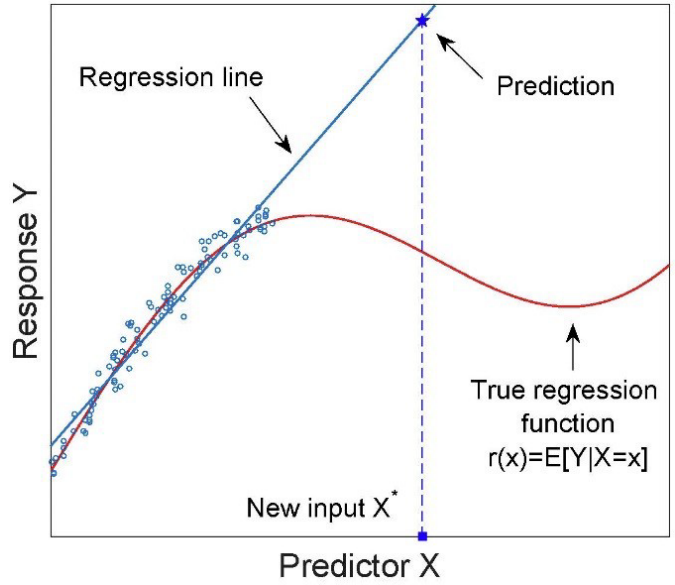
Offline Reinforcement Learning

Offline Policy Evaluation and Distribution Shift

- **The problem:** $P_{test} \neq P_{train}$
- Problematic in many domains:
  - How well does my model extrapolate
  - Adversarial examples
- In RLwe have no ground-truth
  - bootstrapping the target makes the situation even worse



https://xkcd.com/605/



https://stats.stackexchange.com/questions/219579/what-is-wrong-with-extrapolation

Offline Reinforcement Learning

Offline Policy Evaluation and Distribution Shift

What we want to do:

$$\mathbb{E}_{s\sim d_\pi,\, a\sim\pi}\left[\left(r+\gamma\mathbb{E}_{s'\sim p(s'|s,a),\, a'\sim\pi(s')}[Q_\theta(s',a')]-Q_\theta(s,a)\right)^2\right]$$

What we would naively do:

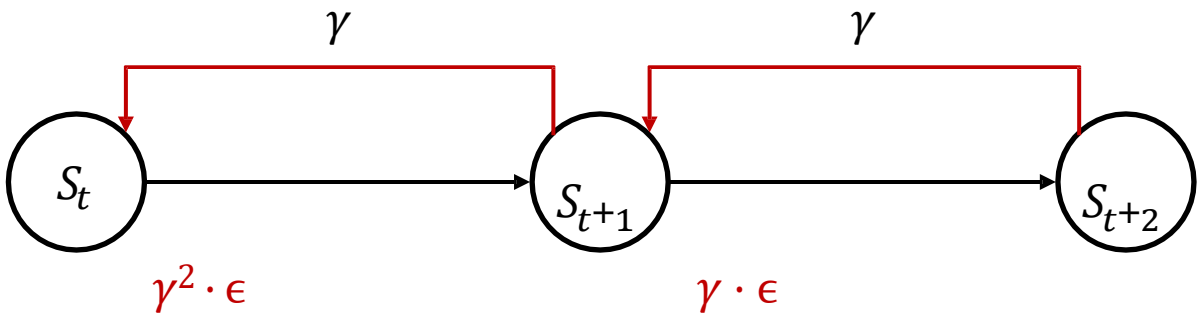
$$\mathbb{E}_{s\sim d_{\pi_\beta},\, a\sim\pi_\beta}\left[\left(r+\gamma\mathbb{E}_{s'\sim p(s'|s,a),\, a'\sim\pi(s')}[Q_\theta(s',a')]-Q_\theta(s,a)\right)^2\right]$$

- **State distribution shift:**
  - Problem arises during **test time**
  - Does not invalidate the learned strategy on the states in $\mathcal{D}$ because unobserved states are never queried during training
- **Action distribution shift:**
  - Already problematic during **training** as inaccurate action values are used as bootstrapped targets
  - Can invalidate the learned strategy even on states in $\mathcal{D}$

Offline Reinforcement Learning

Bootstrapping Out of Distribution Actions

- We can view policy evaluation as a regression problem with a bootstrapped target: $y = r + \gamma Q_\theta(s', \pi(s'))$
- The objective is a (variant of) the mean squared error: $\left(y - Q_\theta(s, a)\right)^2$
- Due to distribution shift the bootstrapped y can be inaccurate as $\pi(s')$ might be an out of distribution action
- Errors are propagated through the state space and can potentially *pollute* everything



$$\epsilon = Q_\theta(s, a) - Q(s, a)$$

Offline Reinforcement Learning

Importance Sampling

- A classical method for off-policy training is **importance sampling**
- Define the **importance ratio**

$$\rho(s,a) = \frac{\pi(a|s) \times d_\pi(s)}{\pi_\beta(a|s) \times d_{\pi_\beta(s)}}$$

- Adjust the policy evaluation objective

$$\sum_{s \in S, a \in A} \rho(s,a) \left[\sum_{(s,a,r,s') \in \mathcal{D}_{|s,a}} r + \gamma Q_\theta(s', \pi(s')) - Q_\theta(s,a) \right]^2$$

- This has the actual loss in expectation

$$\mathbb{E}_{s \sim d_\pi,\, a \sim \pi} \left[\left(r + \gamma \mathbb{E}_{s' \sim p(s'|s,a), a' \sim \pi(s')} [Q_\theta(s', a')] - Q_\theta(s,a) \right)^2 \right]$$

Offline Reinforcement Learning

Importance Sampling

- The adjusted objective becomes

$$\sum_{s\in S, a\in A}\left(\sum_{(s,a,r,s')\in\mathcal{D}_{|s,a}}\frac{\pi(a|s)\times d_{\pi}(s)}{\pi_{\beta}(a|s)\times d_{\pi_{\beta}(s)}}(r+\gamma Q_{\theta}(s',\pi(s'))-Q_{\theta}(s,a))\right)^{2}$$

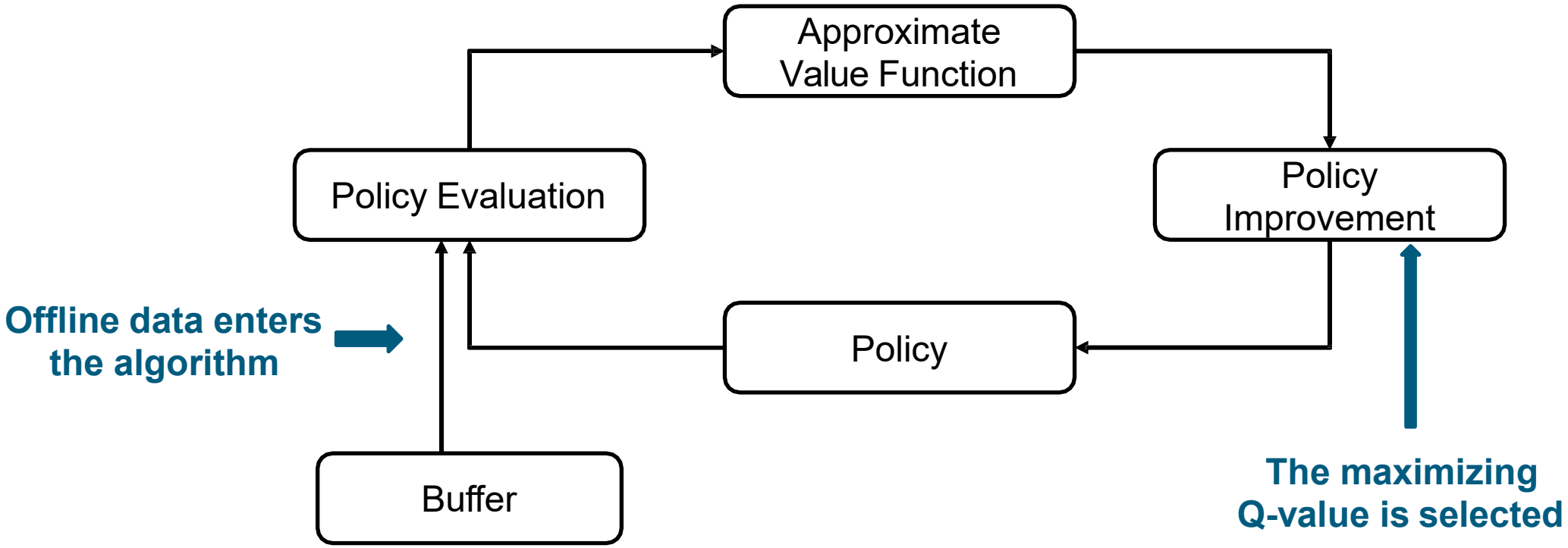
  - $a \in A$ with $\pi_{\beta}(a|s)\approx 0$ is an out of distribution action
  - Consider $\pi_{\beta}(a|s)=0$ and $\pi(a|s)>0$

Shortcomings

- Importance sampling induces high variance if the importance ratios are large
- The importance ratio is large if π and π_{β} differ strongly
- Density estimation in high-dimensional states is notoriously difficult
- There are methods that use importance sampling
  (see e.g. Off-Policy Policy Gradient with State Distribution Correction by Liu et al. (2019))
- In many cases we do not know π_{β}!

Offline Reinforcement Learning

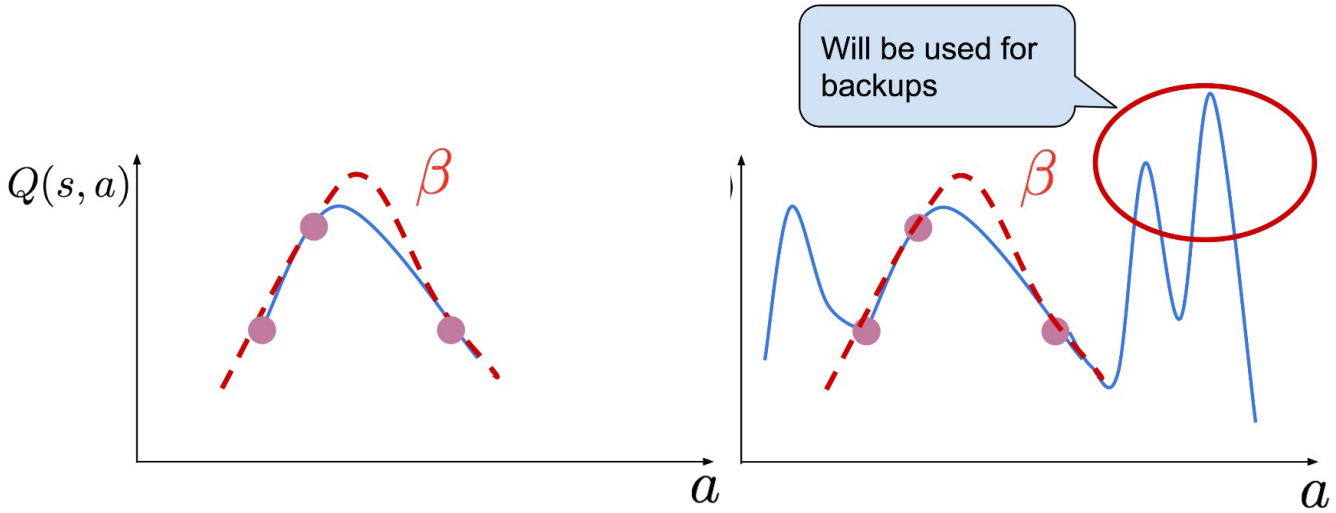
Offline RLand Policy Improvement



Offline Reinforcement Learning

Policy Improvement and Bootstrapping

- We can view policy evaluation as a regression problem with a bootstrapped target: $y = r + \gamma Q_\theta(s', \pi(s'))$
- The objective is a (variant of) the mean squared error: $\left(y - Q_\theta(s, a)\right)^2$
- The **maximizing** action is bootstrapped
  - This potentially draws the policy to out of distribution actions with large positive extrapolation error
  - The importance ratio $\rho(s, a)$ becomes worse



https://bair.berkeley.edu/blog/2019/12/05/bear/

Offline Reinforcement Learning

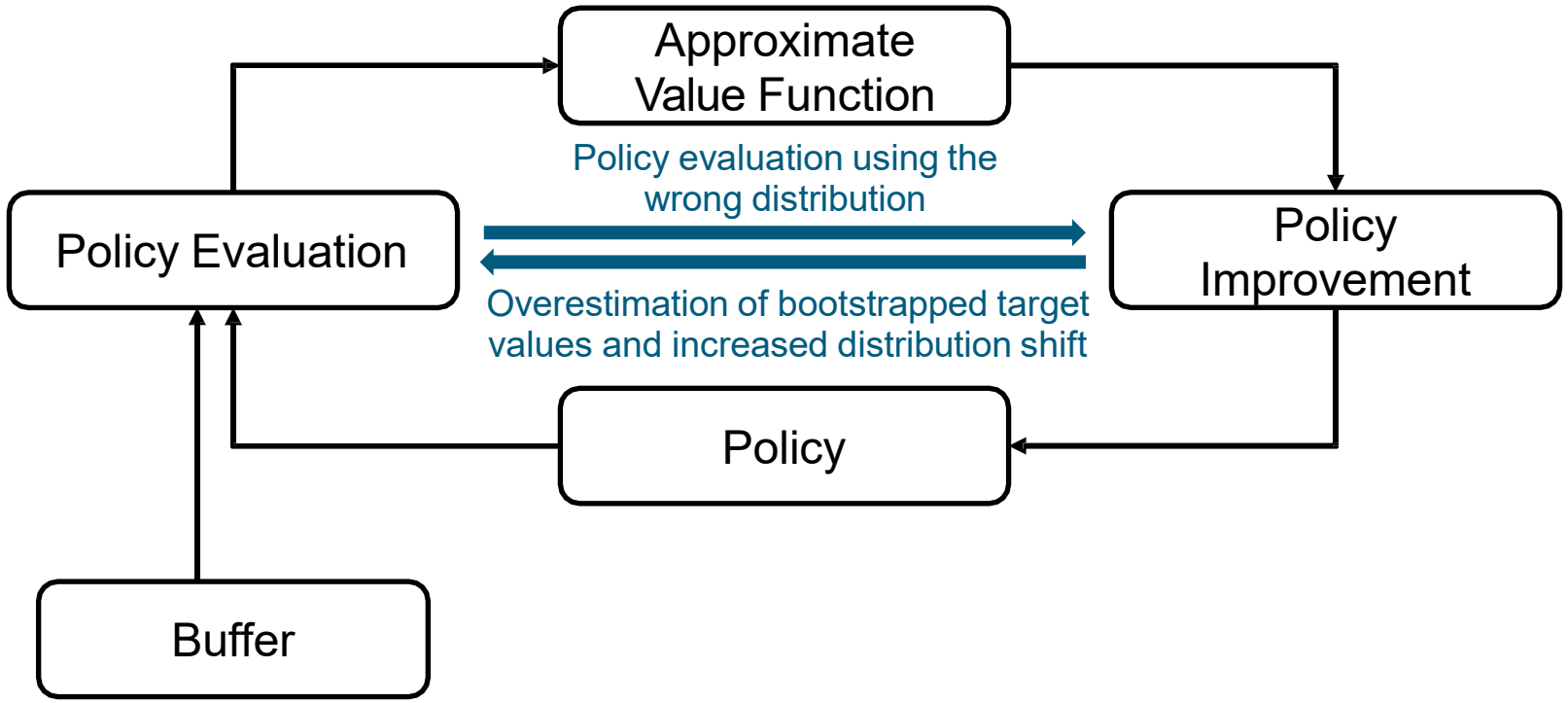
Extrapolation Error in Active Reinforcement Learning

- In active reinforcement learning data is e.g. sampled using ϵ-greedy strategies:
  - Feedback for (potentially overestimated) actions is collected
  - Overestimated Q-values are going to be corrected downwards

But in OfflineRL, No New Corrective Feedback

Offline Reinforcement Learning

GPI in the Offline Setting



Offline Reinforcement Learning

The Dilemma

- We want to improve upon the behaviour policy
  → π should differ from π_β

- We want to be able to evaluate π using data from π_β
  → π should be similar to π_β

→ A tricky trade-off!

Offline Reinforcement Learning

The Deadly Triad

- The main challenges in Offline Reinforcement Learning arise because of the *Deadly Triad:*

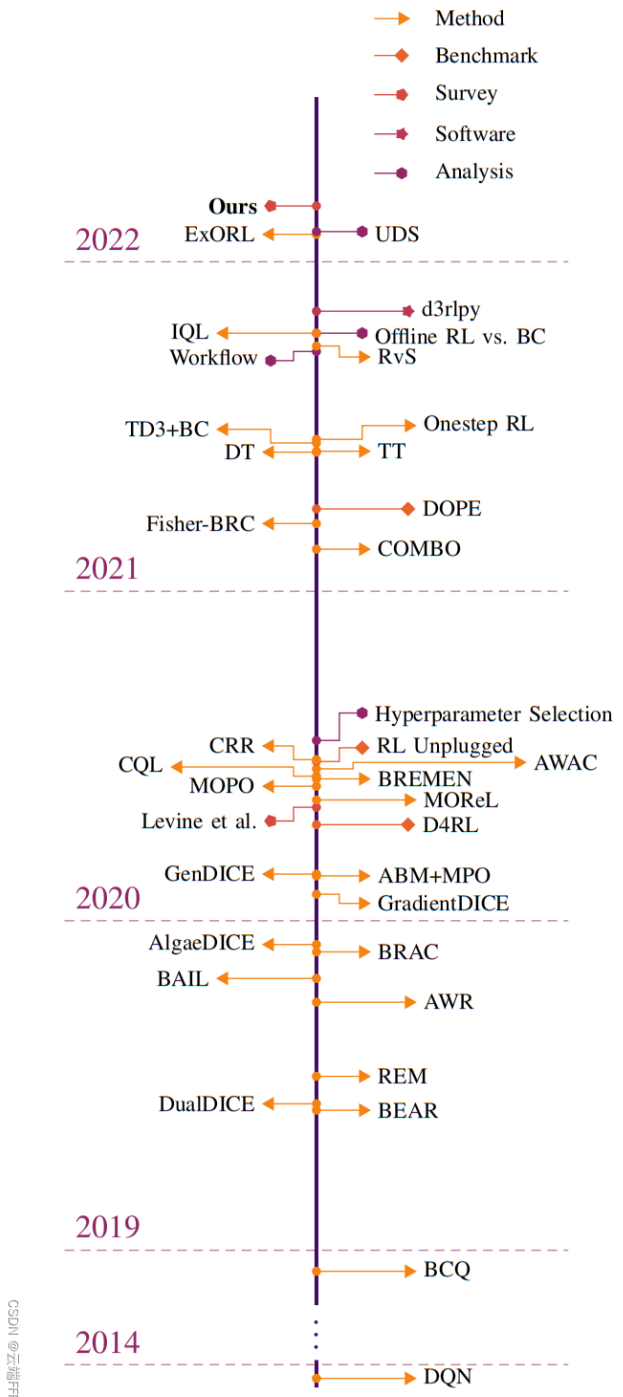
- **Bootstrapping**
  - Needed for efficient learning

- **Function Approximation**
  - Needed for generalization

- **Distribution Shift**
  - A result of off-policy learning

Offline RL development timeline

- A Survey on Offline Reinforcement Learning: Taxonomy, Review, and Open Problems, arXiv.2203.01387



Offline RL solution approaches

- **Policy-constrained methods**
  - bound the distributional shift in the **policy improvement** step
  - by constraining how much the learned policy might differ from the behavior, i.e. $d(\pi, \pi_\beta) \leq \epsilon$
  - What to use as d?
    - Match the distributions: Batch-constrained Q-Learning (BCQ)
    - Match the support: Bootstrapping Error Accumulation Reduction (BEAR)
    - Match the distance: TD3+BC
  - Con: learned policy is similar to behavior policy, limiting performance
- **Conversative methods**
  - Tackle the problem in the **policy evaluation** step
  - by learning a conservative Q-function in areas of high uncertainty, avoids overestimation issues without constraining policy to stay close to behavior distribution
  - Approaches
    - Learn a lower bound for the true Q-Function: Conservative Q-Learning (CQL)
    - Learn a model with a pessimistic reward function: Model-based Offline Policy Optimization (MOPO)
    - Learn a lower bound for the true Q-Function and use additional data from a model: Conservative Offline Model-Based Policy Optimization (COMBO)
  - Con: complex implementation, need to tune extra hyperparameter

Offline RL solution approaches

- **Implicit policy-constraint methods**
  - **implicitly** apply a constraint on the policy to dissuade it from selecting out-of-distribution actions
  - Common approach: Advantage weighted regression
    - only train on collected actions
    - weighted behavioral cloning objective where bad actions are discarded and good ones are used to train the agent.
  - Pro: no explore on unseen actions, avoid extrapolation

- **Behavior Cloning variants**
  - filter out/recognize bad behaviors (low return) in offline trajectories, only imitate good behavior (high return)
  - Approaches
    - Filtered behavior cloning
    - Return-conditioned behavior cloning, e.g. Decision Transformer
  - Pro: simple and effective

Policy-constrained methods

BCQ, BEAR, TD3+BC

Offline Reinforcement Learning

Policy-Constrained Methods

- The problems arise because the maximizing action is selected without uncertainty considerations

$$\pi_{new}(s) = \arg\max_a Q_\theta(s, a)$$

- Define the admissible set of policies $\Pi_\epsilon = \{\pi | d(\pi, \pi_\beta) \leq \epsilon\}$ where d is a distance measure

- Consider a constrained policy improvement step

$$\pi_{new} = \arg\max_{\pi \in \Pi_\epsilon} \mathbb{E}[Q_\theta(s, \pi(s))]$$

Policy-Constrained Offline RL

Batch-Constrained Q-Learning (BCQ)

- **Batch-Constrained Q-Learning**[1] was originally introduced 2019
- General idea:
  - Restrict the action space in order to force the agent towards behaving close to on-policy with respect to the subset of the given data
  - Assumption: if bootstrapped actions are close, then the extrapolation error will be low

- The algorithm will be explained in three steps:
  1. Theoretical BCQ in the tabular case
  2. BCQ with function approximation in the discrete case
  3. BCQ with function approximation in the continuous case

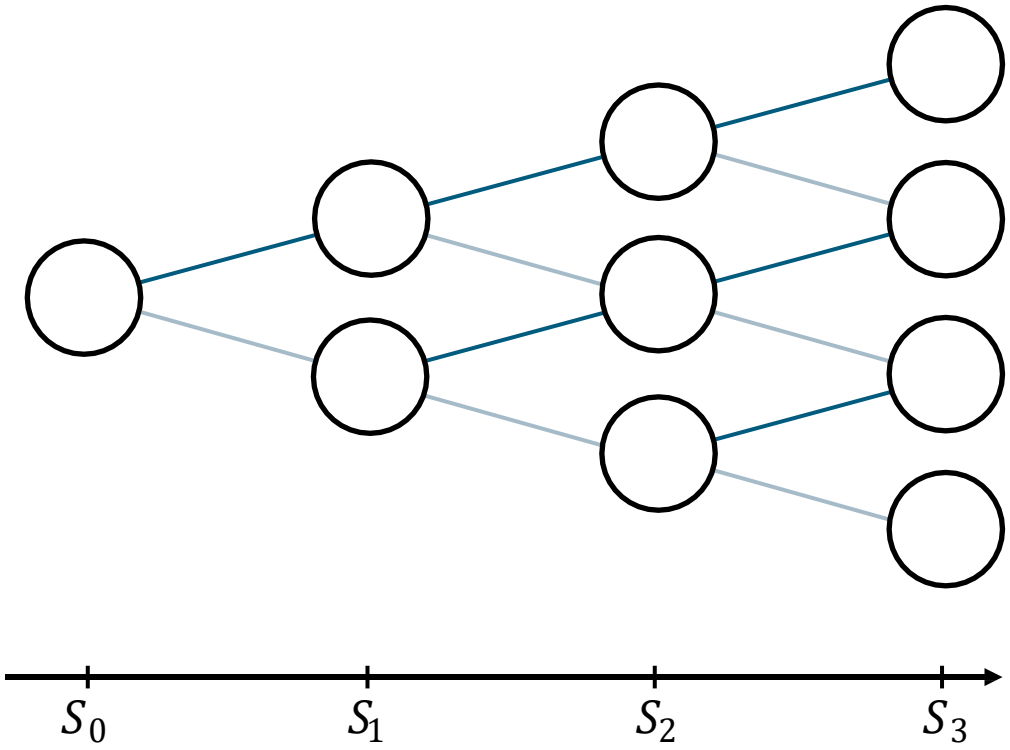
[1] *Fujimoto et a.: Off Policy Deep Reinforcement Learning without Exploration. ICML. 2019.*

Policy-Constrained Offline RL

BCQ – Deterministic Tabular Case

Finite, Deterministic MDP

- Actions: up, down
- Tabular Q-Function
- Rewards are neglected for simplicity



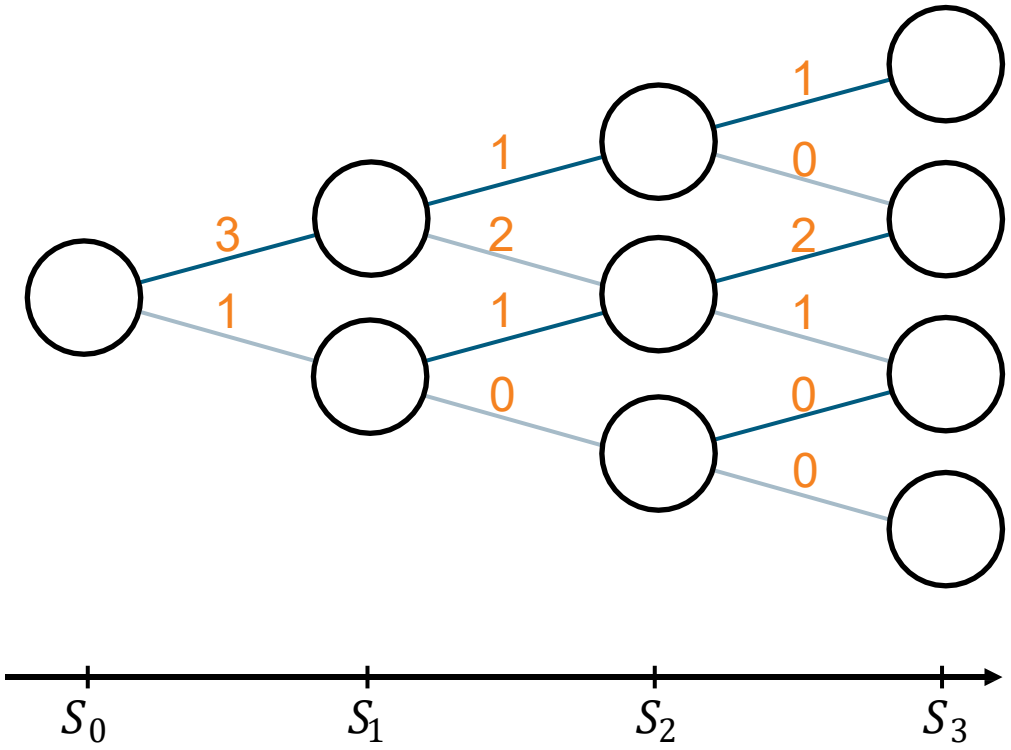
Policy-Constrained Offline RL

BCQ – Deterministic Tabular Case

Finite, Deterministic MDP

- Actions: up, down
- Tabular Q-Function
- Rewards are neglected for simplicity

- Orange numbers are the observed transition counts t_i
- $\mathcal{D}$ consists of the transitions, where $|t_i| > 0$



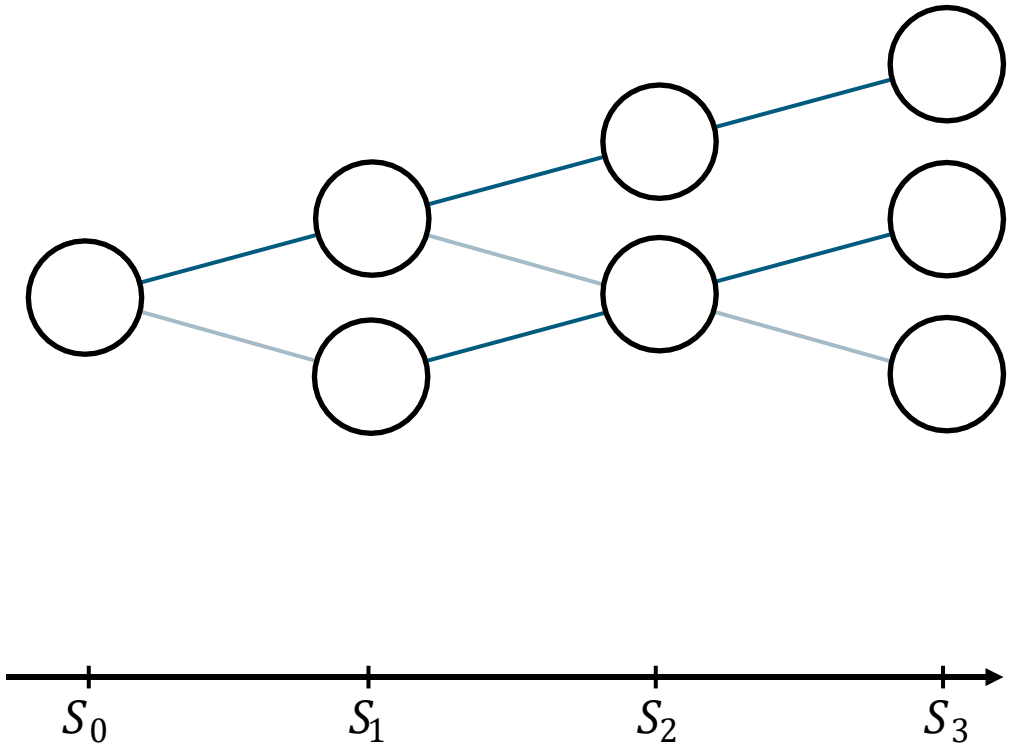
Policy-Constrained Offline RL

BCQ – Deterministic Tabular Case

Finite, Deterministic MDP

- Actions: up, down
- Tabular Q-Function
- Rewards are neglected for simplicity

- Estimate the restricted MDP
- Solve the restricted MDP (e.g., using Q-Learning)
- All policies that stay lie in the restricted MDP can be accurately assessed because of the determinism

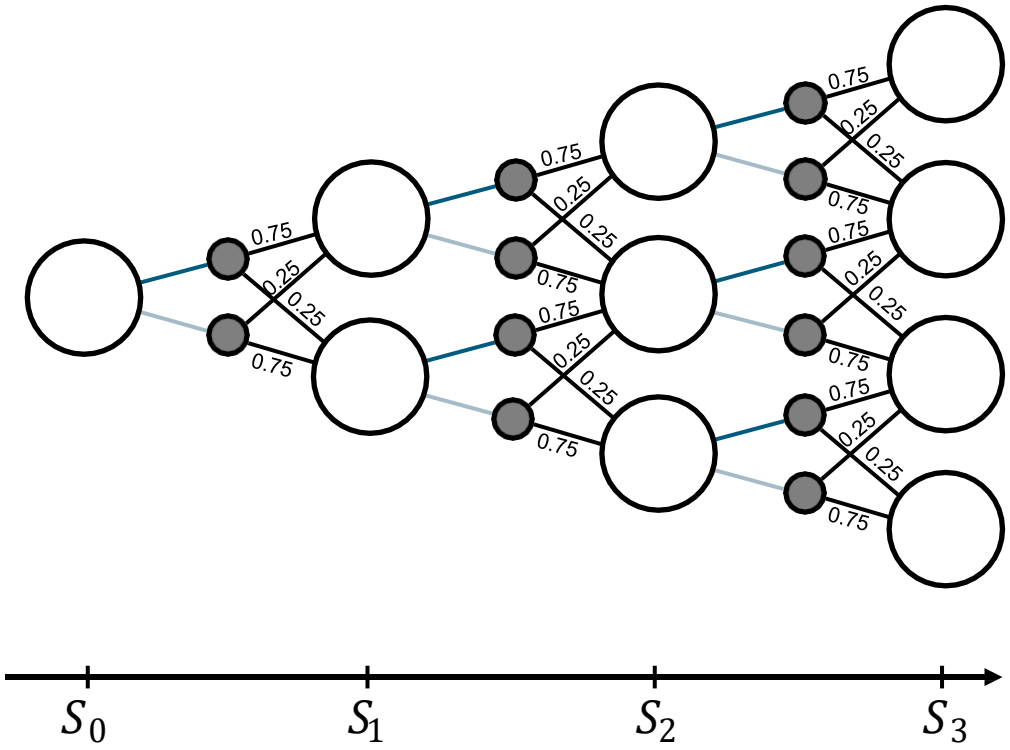


Policy-Constrained Offline RL

BCQ – Stochastic Tabular Case

Finite, Stochastic MDP

- Actions: up, down
- Tabular Q-Function
- Rewards are neglected for simplicity



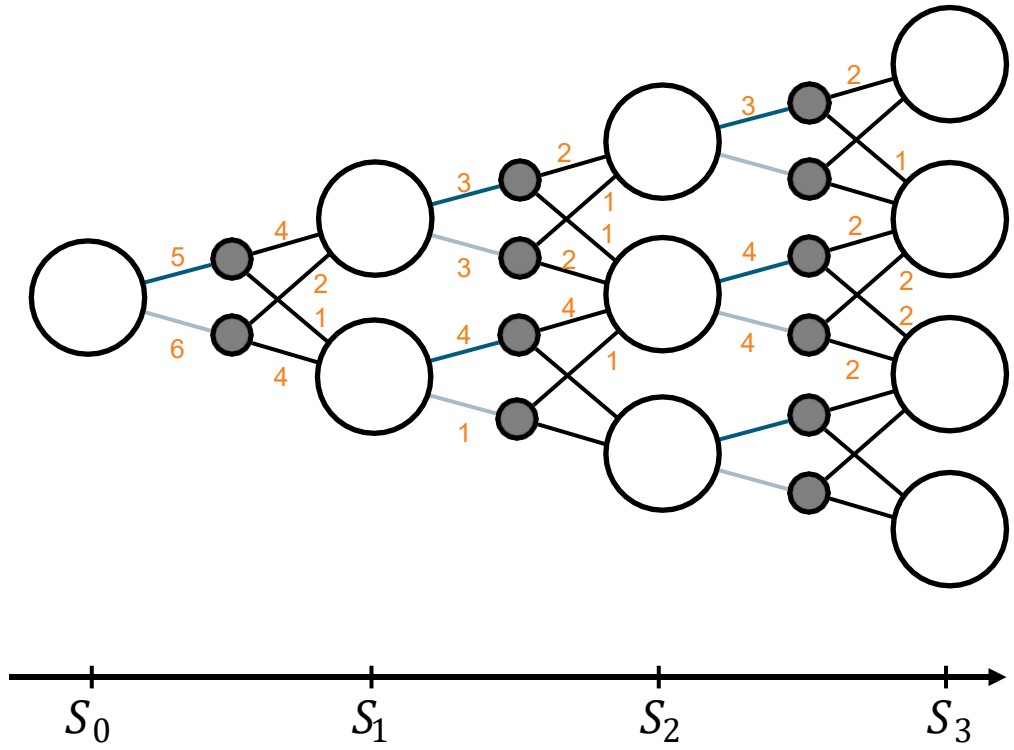
Policy-Constrained Offline RL

BCQ – Stochastic Tabular Case

Finite, Stochastic MDP

- Actions: up, down
- Tabular Q-Function
- Rewards are neglected for simplicity

- Orange numbers are the observed transition counts t_i
- $\mathcal{D}$ consists of the transitions, where $|t_i| > 0$



Policy-Constrained Offline RL

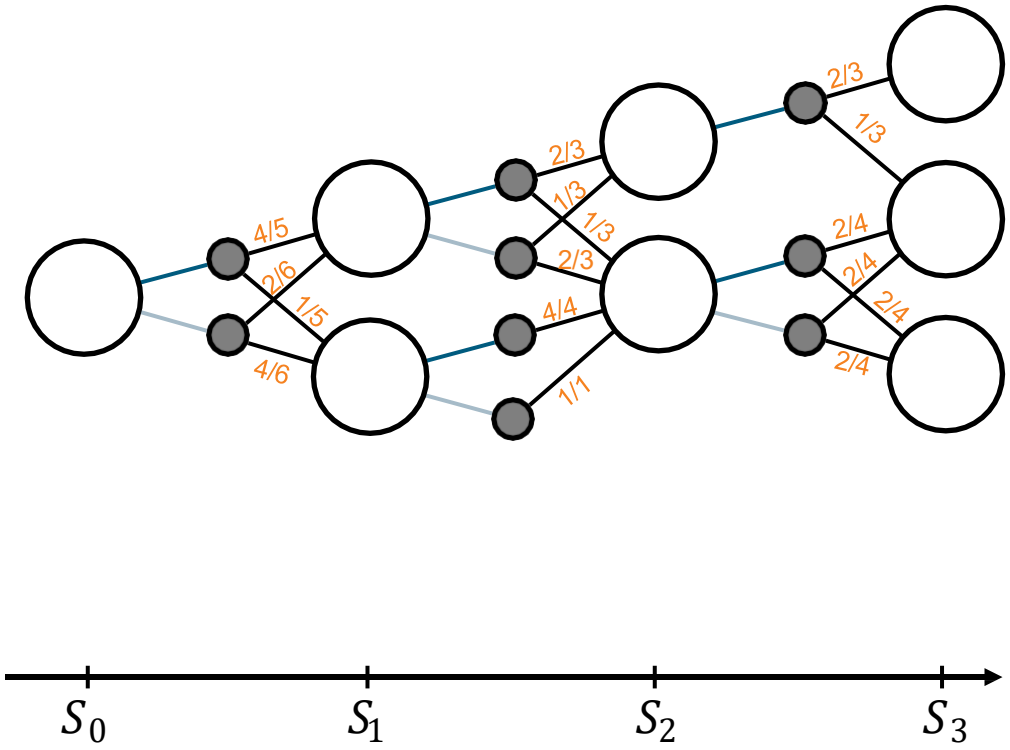
BCQ – Stochastic Tabular Case

Finite, Stochastic MDP

- Actions: up, down
- Tabular Q-Function
- Rewards are neglected for simplicity

- Estimate the restricted MDP
- Solve the estimated MDP (e.g., using Q-Learning)
- Quality of the learned policy depends on the quality of the estimation

- **When deploying the policy, the agent might be swept into the unknown!**



Policy-Constrained Offline RL

BCQ with Function Approximation – Discrete Case

- Q-Learning:

$$\min_{\theta}(r + \gamma \max_{a' \in A(s')} Q_\theta(s', a') - Q_\theta(s, a))^2$$

- Let us define

$$A_\epsilon^{BCQ}(s) = \left\{ a \in A(s): \frac{\hat{\pi}_\beta(a|s)}{\max_a \hat{\pi}_\beta(a|s)} \geq \epsilon \right\},$$

where $\epsilon \in [0,1]$ is the threshold parameter and $\hat{\pi}_\beta$ an estimate for the behaviour policy

- The **constrained target** is $y = r + \gamma \cdot \max_{a' \in A_\epsilon^{BCQ}(s')} Q(s', a')$
  - $\epsilon = 1$ → behavioural cloning
  - $\epsilon = 0$ → Q-Learning

- The learned policy is $\pi(s) = \arg\max_{a \in A_\epsilon^{BCQ}(s)} Q(s, a)$

Policy-Constrained Offline RL

BCQ with Function Approximation – Continuous Case

- **Problem:**
  - Computing the restriction $\frac{\hat{\pi}_\beta(a|s)}{\max_a \hat{\pi}_\beta(a|s)} > \epsilon$ is not so straightforward in the continuous case

- **BCQ addresses this using three counter-measures:**
  - A **Conditional Variational Autoencoder** $\hat{\pi}_\beta(\cdot|s)$ is used to sample actions
  - Additionally, a **perturbation model** improves the sampled actions $a_i \sim \hat{\pi}_\beta(\cdot|s)$ in a neighbourhood
  - **(Soft) Clipped Double Q-Learning**[1] fights overestimation in continuous action spaces

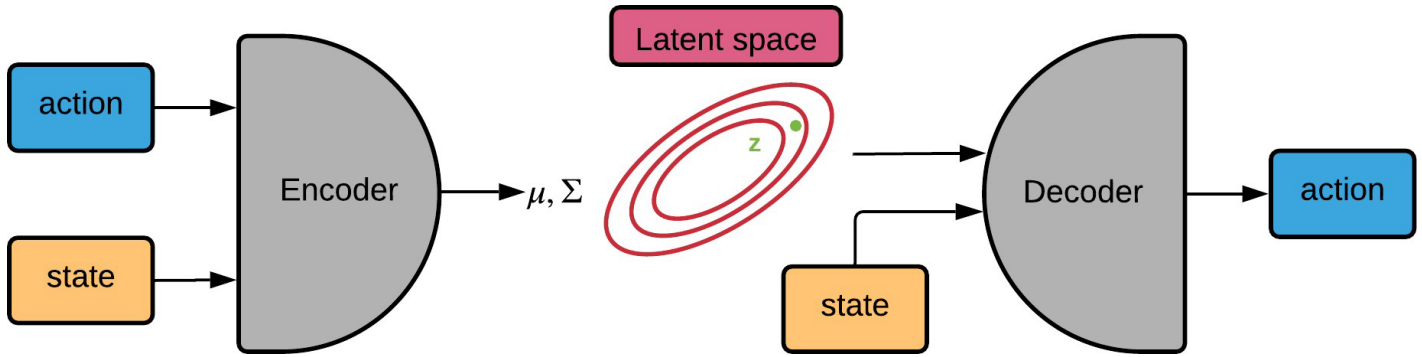
[1] *Fujimoto et al.: Addressing Function Approximation Error in Actor-Critic Methods. ICML. 2018.*

Policy-Constrained Offline RL

BCQ – Sampling from the Behaviour Policy

- **Goal:** Sample from the behavioural policy $\pi_\beta(\cdot\,|s)$
- **Problem:** $\pi_\beta(\cdot\,|s)$ is unknown and difficult to sample from
- **Solution:** **Conditional Variational Autoencoder**

- Generate samples from a trained VAE:
  1. Sample z_i from the latent space (encoder is not used)
  2. Decode $a_i' = dec(z_i, s)$

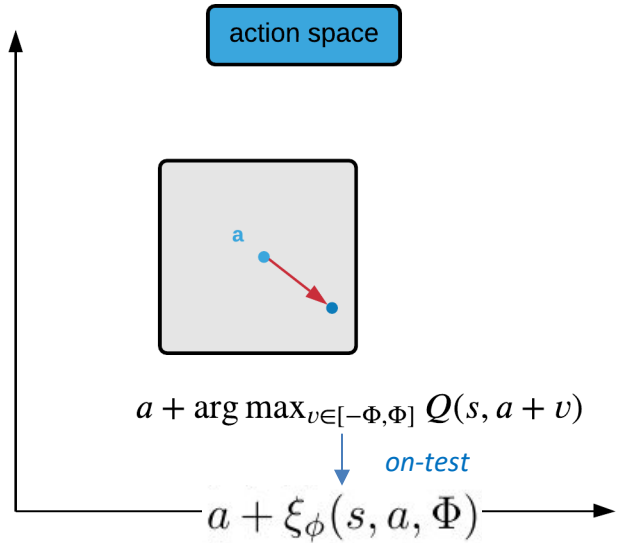


Policy-Constrained Offline RL

BCQ – Perturbation Model

- **Situation:** In state s' we have obtained sampled actions a_i' from $\pi_\beta(\cdot\,|s')$
- **Problem:** Finding a good action requires many samples
- **Solution:**
  - Train an additional perturbation model ξ that improves the Q-value of a_i in a neighbourhood
  - The goal is $\xi_\phi(s, a, \Phi) = \arg\max_{v \in [\Phi, \Phi]} Q(s, a + v)$
    - $\Phi = 0$ → Behavioural cloning
    - $\Phi = \infty$ → Q-Learning
  - It is trained with the DDPG algorithm

$$\phi \leftarrow \text{argmax}_\phi \sum Q_{\theta_1}(s, a + \xi_\phi(s, a, \Phi)), a \sim G_\omega(s)$$



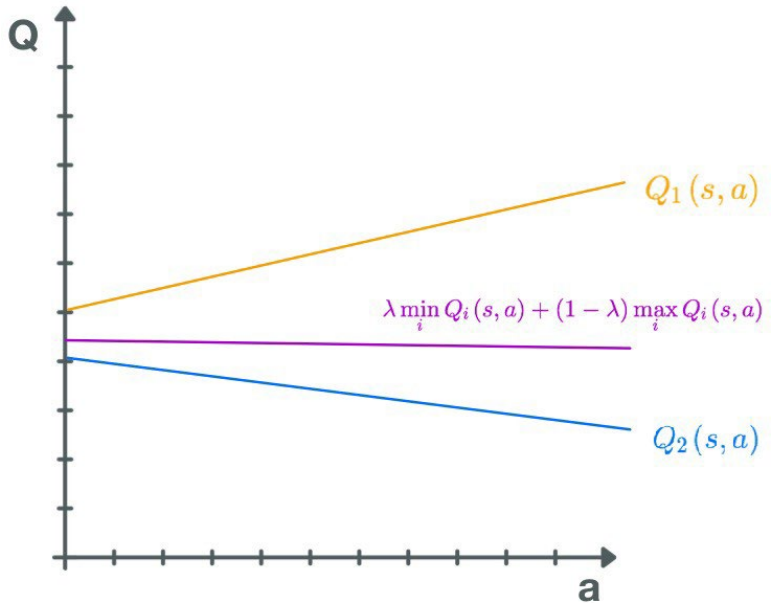
Policy-Constrained Offline RL

BCQ – Soft Clipped Double Q-Learning

- **Situation:** In state s' we have perturbed actions a_i' from $\hat{\pi}_\beta(\cdot\,|s')$ resulting in $a_i^* = a_i' + \xi(s', a_i', \Phi)$
- **Problem:** Vanilla Double Q-Learning can be ineffective when the networks are too similar
  → Overestimation of Q-values
- **Solution:**
  - Use a convex combination of the minimum and the maximum

$$y = \max_{a'=a_1^*,\ldots,a_k^*} \left\{ \lambda \cdot \min_i Q_{Q_{\theta_i'}}(s', a') + (1-\lambda) \cdot \max_i Q_{\theta_i'}(s', a') \right\}$$

  - When $\lambda \in [0,1]$ is large this penalizes **uncertain** regions



Policy-Constrained Offline RL

Continuous BCQ – Algorithm

Algorithm 1 BCQ

Input: Batch $\mathcal{B}$, horizon T, target network update rate τ, mini-batch size N, max perturbation Φ, number of sampled actions n, minimum weighting λ.

Initialize Q-networks $Q_{\theta_1}, Q_{\theta_2}$, perturbation network ξ_ϕ, and VAE $G_\omega = \{E_{\omega_1}, D_{\omega_2}\}$, with random parameters θ_1, θ_2, ϕ, ω, and target networks $Q_{\theta'_1}, Q_{\theta'_2}$, $\xi_{\phi'}$ with $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$.

for $t = 1$ **to** T **do**

Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$

$\mu, \sigma = E_{\omega_1}(s, a), \quad \tilde{a} = D_{\omega_2}(s, z), \quad z \sim \mathcal{N}(\mu, \sigma)$
$\omega \leftarrow \text{argmin}_\omega \sum (a - \tilde{a})^2 + D_{\text{KL}}(\mathcal{N}(\mu, \sigma) || \mathcal{N}(0, 1))$

Sample n actions: $\{a_i \sim G_\omega(s')\}_{i=1}^n$
Perturb each action: $\{a_i = a_i + \xi_\phi(s', a_i, \Phi)\}_{i=1}^n$

Set value target y (Eqn. 13)

$\theta \leftarrow \text{argmin}_\theta \sum (y - Q_\theta(s, a))^2$

$\phi \leftarrow \text{argmax}_\phi \sum Q_{\theta_1}(s, a + \xi_\phi(s, a, \Phi)), a \sim G_\omega(s)$

Update target networks: $\theta'_i \leftarrow \tau\theta + (1 - \tau)\theta'_i$
$\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$

end for

train CVAE

sample from CVAE and perturb

clipped target

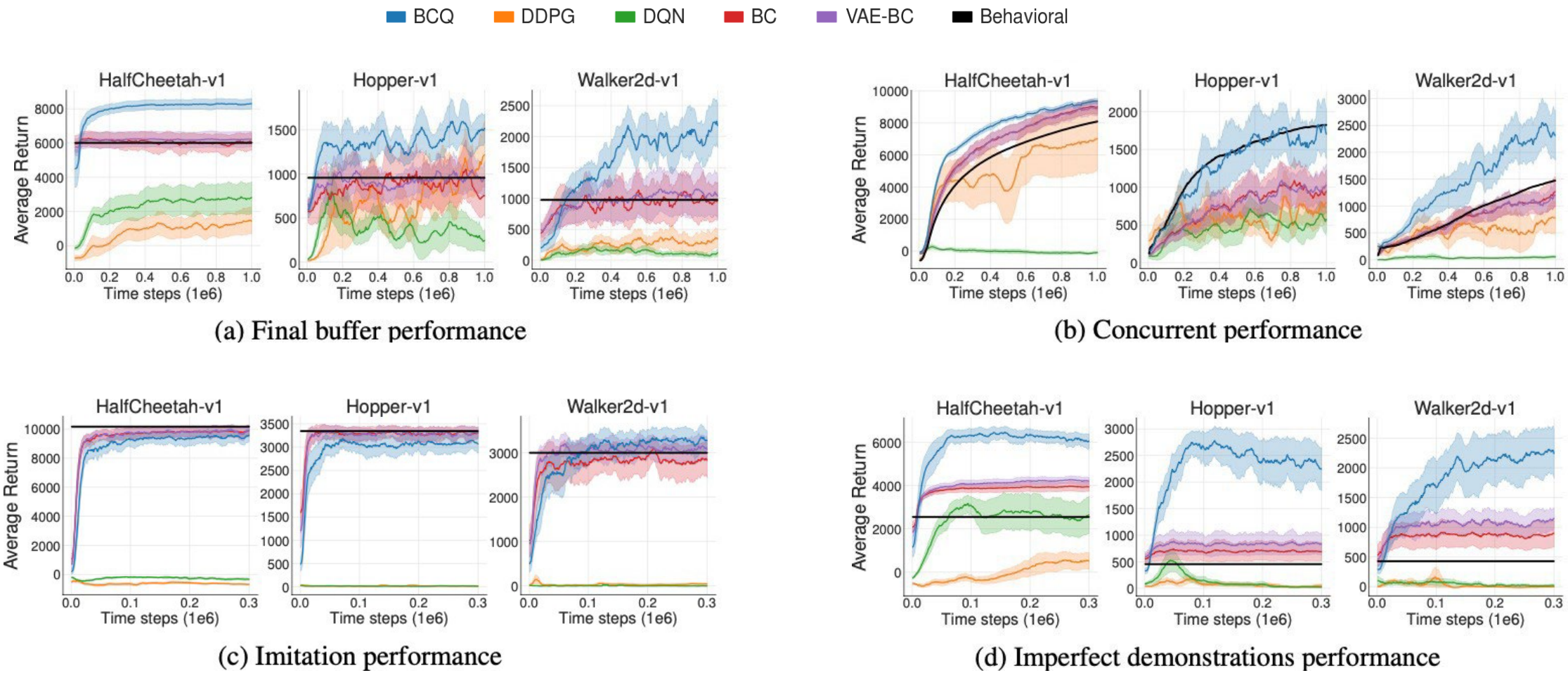
train Q

train perturbation

update targets

Policy-Constrained Offline RL

Continuous BCQ – Results



(a) Final buffer performance

(b) Concurrent performance

(c) Imitation performance

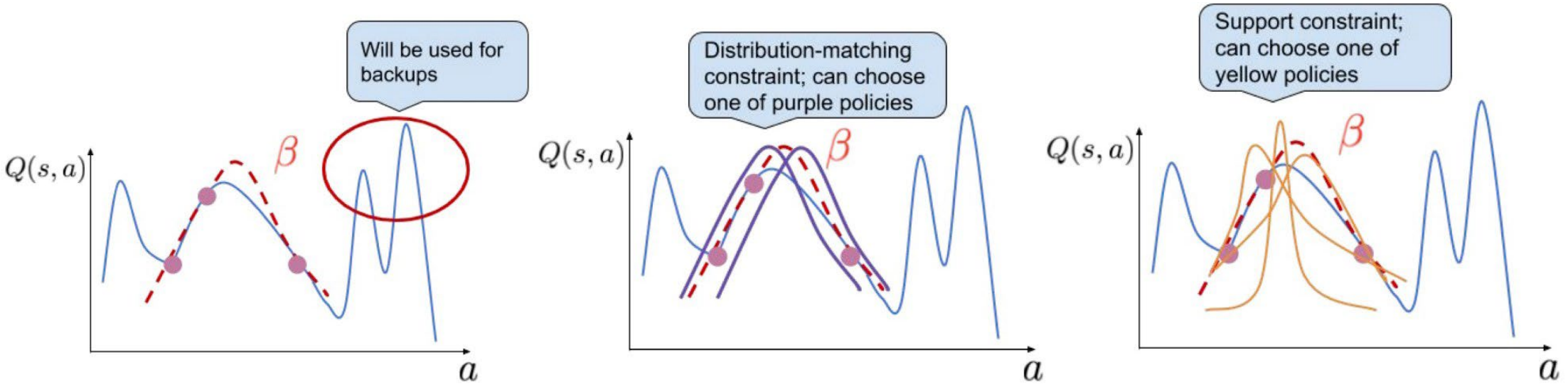
(d) Imperfect demonstrations performance

Policy-Constrained Offline RL

Bootstrapping Error Accumulation Reduction (BEAR)

General motivation of BEAR[1]:

- The constraint in BCQ is overly restrictive:
  If the behaviour policy is uniform, the learned policy must also be close to uniform!
- Only require that π lies in the support of π_β: $\pi(a|s) > 0 \Rightarrow \pi_\beta(a|s) > \epsilon$



https://bair.berkeley.edu/blog/2019/12/05/bear/

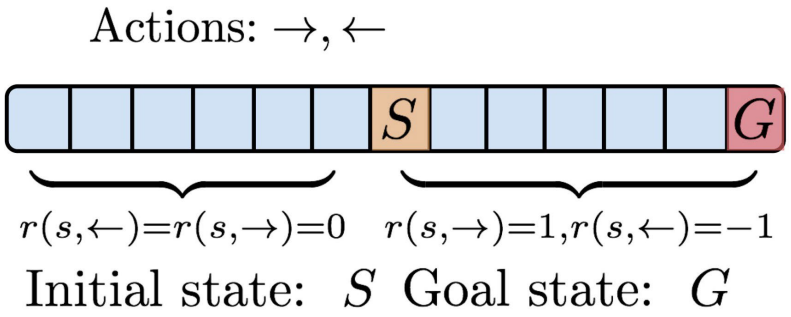
[1] *Kumar et al.: Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. NeurIPS 2019.*

Policy-Constrained Offline RL

Bootstrapping Error Accumulation Reduction (BEAR)

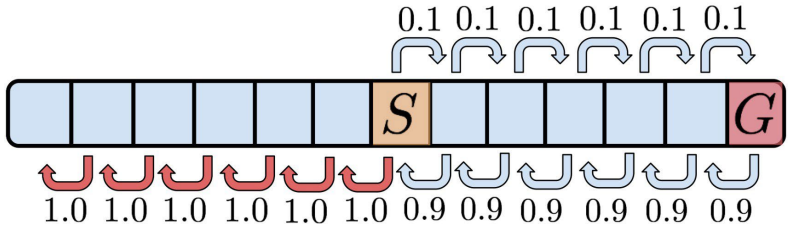
Example:

- 1-dimensional grid-world, two actions: left & right
- The agent starts in **S** and its goal is to reach **G**



1D-Lineworld Environment

- The behavior policy is used to generate our dataset
  - Right side: sub-optimal actions are more likely (90%), *but both actions are in-distribution at all the states*
  - Left side: only the left action is used



Behavior Policy

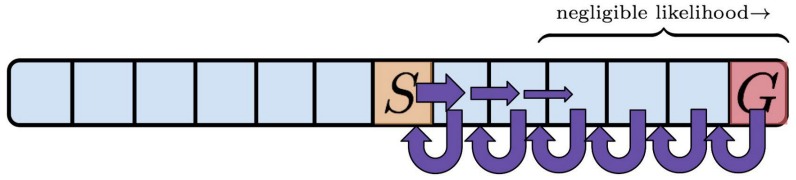
https://bair.berkeley.edu/blog/2019/12/05/bear/

Policy-Constrained Offline RL

Bootstrapping Error Accumulation Reduction (BEAR)

Example:

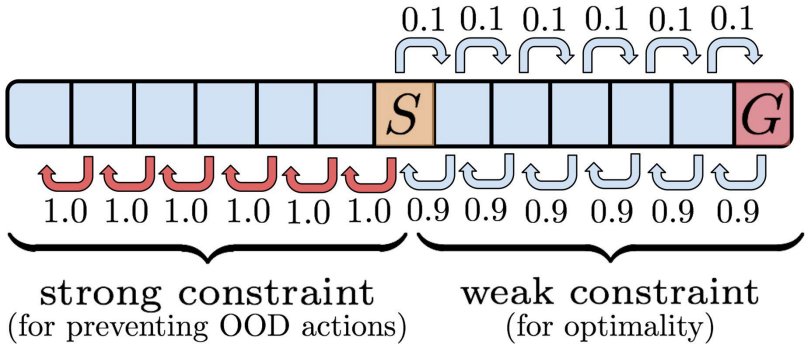
- **Distribution-matching** constraint can be arbitrarily sub-optimal
- Chances of reaching the goal become very small
- With bigger grid-worlds the chances approach 0



Learned Policy via distribution-matching

Why does distribution-matching fail here?

- Assume we use a penalty for distribution-matching
- If penalty tight: agent goes **left** between **S** and **G**
  → suboptimal behavior
- If penalty is generous (in hope for getting a better policy): agent will start to take OOD actions to left of **S**
  → affects Q-value of **S**
  → maybe the policy will even go left when being in **S**!



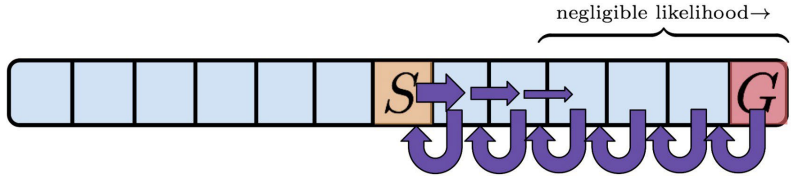
https://bair.berkeley.edu/blog/2019/12/05/bear/

Policy-Constrained Offline RL

Bootstrapping Error Accumulation Reduction (BEAR)

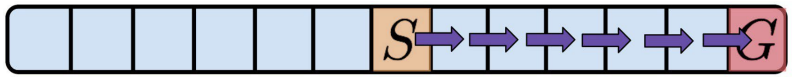
Example:

- **Distribution-matching** constraint can be arbitrarily sub-optimal
- Chances of reaching the goal become very small
- With bigger grid-worlds the chances approach 0



Learned Policy via distribution-matching

- In contrast, a **support-constraint** can recover the optimal policy with probability 1!



Learned Policy via support-constraint

https://bair.berkeley.edu/blog/2019/12/05/bear/

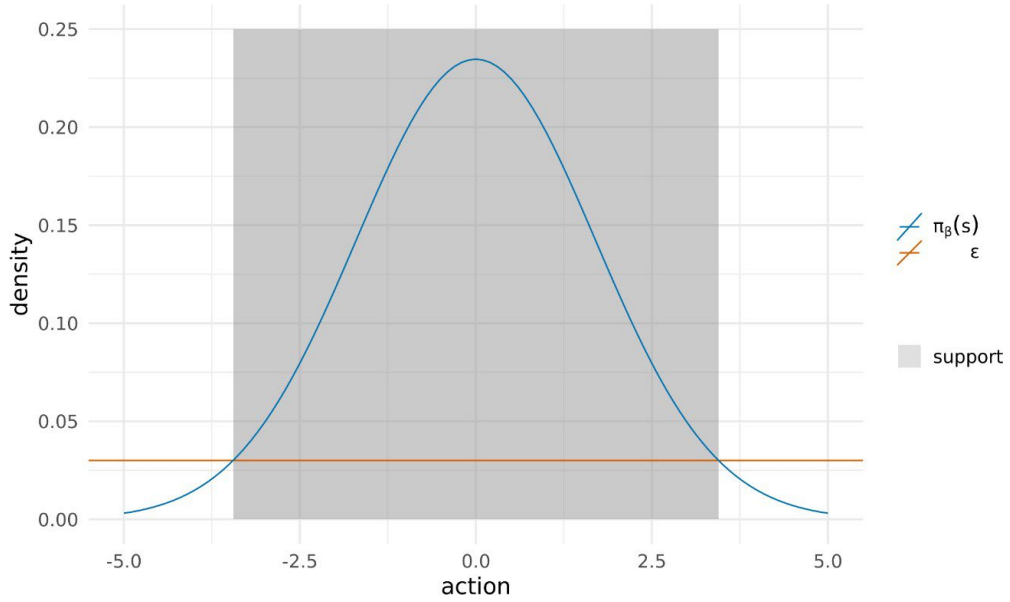
Policy-Constrained Offline RL

Bootstrapping Error Accumulation Reduction (BEAR)

- **Q-Learning**
  - Objective: $\min_{\theta}(r + \gamma \max_{a' \in A(s')} Q_\theta(s', a') - Q_\theta(s, a))^2$
  - Policy: $\pi(s) = \text{argmax}_{a \in A(s)} Q_\theta(s, a)$

Remember: $A_\epsilon^{BCQ}(s) = \left\{ a \in A(s): \frac{\hat{\pi}_\beta(a|s)}{\max_a \hat{\pi}_\beta(a|s)} \geq \epsilon \right\}$,

- **BEAR**
  - Policy constraint: $A_\epsilon^{BEAR}(s) = \{a \in A(s): \hat{\pi}_\beta(a|s) \geq \epsilon\}$
  - Objective: $\left(r + \gamma \cdot \max_{a' \in A_\epsilon^{BEAR}(s')} Q_\theta(s', a') - Q_\theta(s, a)\right)^2$
  - Policy: $\pi(s) = \underset{a \in A_\epsilon^{BEAR}(s)}{\text{argmax}} Q_\theta(s, a)$



Policy-Constrained Offline RL

BEAR – Support Constraint

- **Goal:**
  - get $\max_{a' \in A_\epsilon^{BEAR}(s')} Q(s', a')$ in a continuous action space
- **Approach:**
  - Train an actor π_ϕ that satisfies $\pi_\phi(a|s) > 0 \Rightarrow \pi_\beta(a|s) \geq \epsilon$
  - *Optimization problem:*

$$\max_\phi Q\big(s, \pi_\phi(s)\big) \quad s.t.\ \pi_\phi(a|s) > 0 \Rightarrow \pi_\beta(a|s) > \epsilon$$

→ This requires a distance metric $d(\pi, \pi_\beta)$ that measures the violation of the support constraint

- **Solution in BEAR:**
  - This constraint is implemented via **Maximum Mean Discrepancy** $\mathrm{MMD}(\pi_\phi, \pi_\beta)$
  - This turns the policy improvement step into a *constrained optimization problem:*

$$\max_\phi Q_\theta\left(s, \pi_\phi(s)\right) \quad s.t.\ \mathrm{MMD}^2\left(\pi_\phi, \pi_\beta\right) < C(\epsilon)$$

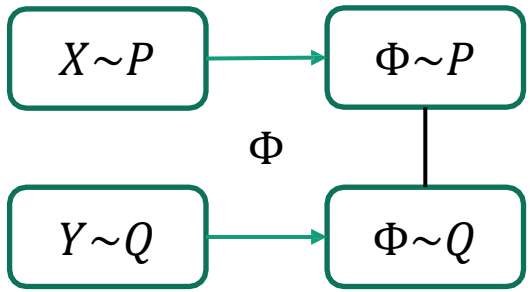
Policy-Constrained Offline RL

Parenthesis: Maximum Mean Discrepancy

only for reference

- **Goal:** Measure the distance between distributions Q and P
  - Let $X \sim P$ and $Y \sim Q$
- **Idea of MMD:**
  1. Map X and Y into a Hilbert space $(H, \langle \cdot \rangle)$ using a feature map Φ
  2. Calculate difference between their expectations in H:

$$||E_{X \sim P}[\Phi(X)] - E_{Y \sim Q}[\Phi(Y)]||$$



$$||E_{X \sim P}[\Phi(X)] - E_{Y \sim Q}[\Phi(Y)]||$$

Policy-Constrained Offline RL

Parenthesis: Maximum Mean Discrepancy

only for reference

- **Goal:** Measure the distance between distributions Q and P
  - Let $X \sim P$ and $Y \sim Q$
- **Kernel trick:** For certain feature maps Φ it holds that $\langle \Phi(x), \Phi(y) \rangle = k(x, y)$
  - Here k is a RBF kernel (e.g. the Gaussian kernel)
  - In these cases it holds that[1]

$$||\mathbb{E}_{X \sim P}[\Phi(X)] - \mathbb{E}_{Y \sim Q}[\Phi(Y)]|| = \mathbb{E}_{X,X' \sim P}[k(X, X')] + \mathbb{E}_{Y,Y' \sim Q}[k(Y, Y')] + 2\mathbb{E}_{X \sim P, Y \sim Q}[k(X, Y)]$$

[1] *Tolstikhin et al.: Minimax Estimation of Maximum Mean Discrepancy with Radial Kernels. NIPS. 2016.*

Policy-Constrained Offline RL

Parenthesis: Maximum Mean Discrepancy

only for reference

- **Goal:** Measure the distance between distributions Q and P
  - Let $X \sim P$ and $Y \sim Q$
  - Estimate $\mathrm{MMD}^2(\pi_\phi, \pi_\beta)$ from samples $a_1^\phi, \ldots, a_n^\phi \sim \pi_\phi(s)$ and $a_1^\beta, \ldots, a_m^\beta \sim \hat{\pi}_\beta(s)$

$$\frac{1}{n^2}\sum_{i,i'} k\left(a_i^\phi, a_{i'}^\phi\right) - \frac{2}{nm}\sum_{i,j} k\left(a_i^\phi, a_j^\beta\right) + \frac{1}{m^2}\sum_{j,j'} k\left(a_j^\beta, a_{j'}^\beta\right)$$

  - For k they use the Gaussian kernel:

```
def gaussian_kernel(x, y, sigma=0.1):
  return exp(-(x - y).pow(2).sum() / (2 * sigma.pow(2)))

def compute_mmd(x, y):
  k_x_x = gaussian_kernel(x, x)
  k_x_y = gaussian_kernel(x, y)
  k_y_y = gaussian_kernel(y, y)
  return sqrt(k_x_x.mean() + k_y_y.mean() - 2*k_x_y.mean())
```

  - Empirically, this sample-estimated distance matches the support in the low-intermediate sample regime[1]

https://bair.berkeley.edu/blog/2019/12/05/bear/

[1] *Kumar et al.: Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. NeurIPS 2019.*

Policy-Constrained Offline RL

Dual Gradient Descent

only for reference

- Objective for the actor:

$$\max_{\phi} Q_\theta\left(s, \phi(s)\right) \qquad s.t.\ MMD^2\left(A^\phi, A^\beta\right) \leq \epsilon$$

  - This is a **constrained optimization** problem
  - Many algorithms exist to solve this, e.g., penalty and barrier methods
  - BEAR uses **Dual Gradient Descent**

Policy-Constrained Offline RL

Dual Gradient Descent in a Nutshell

only for reference

- Optimization problem (for equality constraints):

$$\max_x f(x) \quad s.t.\ C(x) = 0$$

- Lagrangian:

$$L(x, \lambda) = f(x) + \lambda C(x)$$

- The dual gradient descent algorithm solves the **dual problem** instead of the original (primal) problem
  - The dual problem is: $\max_x \min_\lambda L(x,\lambda)$
  - Define $g(\lambda) = L(x^*(\lambda), \lambda)$ where $x^*(\lambda) = \arg\max_x L(x, \lambda)$
  - **Algorithm**
    1. Find $x^* \leftarrow \arg\max_x L(x, \lambda)$
    2. Compute $\frac{dg}{d\lambda} = \frac{dL}{d\lambda}(x^*, \lambda)$
    3. $\lambda \leftarrow \lambda + \beta \frac{dg}{d\lambda}$

see also https://jonathan-hui.medium.com/rl-dual-gradient-descent-fac524c1f049

Policy-Constrained Offline RL

BEAR – Algorithm

Algorithm 1 BEAR Q-Learning (BEAR-QL)

input : Dataset $\mathcal{D}$, target network update rate τ, mini-batch size N, sampled actions for MMD n, minimum λ

Initialize Q-ensemble $\{Q_{\theta_i}\}_{i=1}^{K}$, actor π_ϕ, Lagrange multiplier α, target networks $\{Q_{\theta'_i}\}_{i=1}^{K}$, and a target actor $\pi_{\phi'}$, with $\phi' \leftarrow \phi, \theta'_i \leftarrow \theta_i$

for t in $\{1, \ldots, \mathrm{N}\}$ **do**

  Sample mini-batch of transitions $(s, a, r, s') \sim \mathcal{D}$

  Q-update:

  Sample p action samples, $\{a_i \sim \pi_{\phi'}(\cdot|s')\}_{i=1}^{p}$

  Define $y(s,a) := \max_{a_i}[\lambda \min_{j=1,..,K} Q_{\theta'_j}(s', a_i) + (1-\lambda)\max_{j=1,..,K} Q_{\theta'_j}(s', a_i)]$

  $\forall i, \theta_i \leftarrow \arg\min_{\theta_i}(Q_{\theta_i}(s,a) - (r + \gamma y(s,a)))^2$

  Policy-update:

  Sample actions $\{\hat{a}_i \sim \pi_\phi(\cdot|s)\}_{i=1}^{m}$ and $\{a_j \sim \mathcal{D}(s)\}_{j=1}^{n}$, n preferably an intermediate integer(1-10)

  Update ϕ, α by minimizing Equation 1 by using dual gradient descent with Lagrange multiplier α

  Update Target Networks: $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$; $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$

end for

$$\pi_\phi := \max_{\pi \in \Delta_{|S|}} \mathbb{E}_{s\sim\mathcal{D}} \mathbb{E}_{a\sim\pi(\cdot|s)} \left[\min_{j=1,..,K} \hat{Q}_j(s,a) \right] \quad \text{s.t. } \mathbb{E}_{s\sim\mathcal{D}}[\mathrm{MMD}(\mathcal{D}(s), \pi(\cdot|s))] \leq \varepsilon \qquad (1)$$

Policy-Constrained Offline RL

BEAR – Results

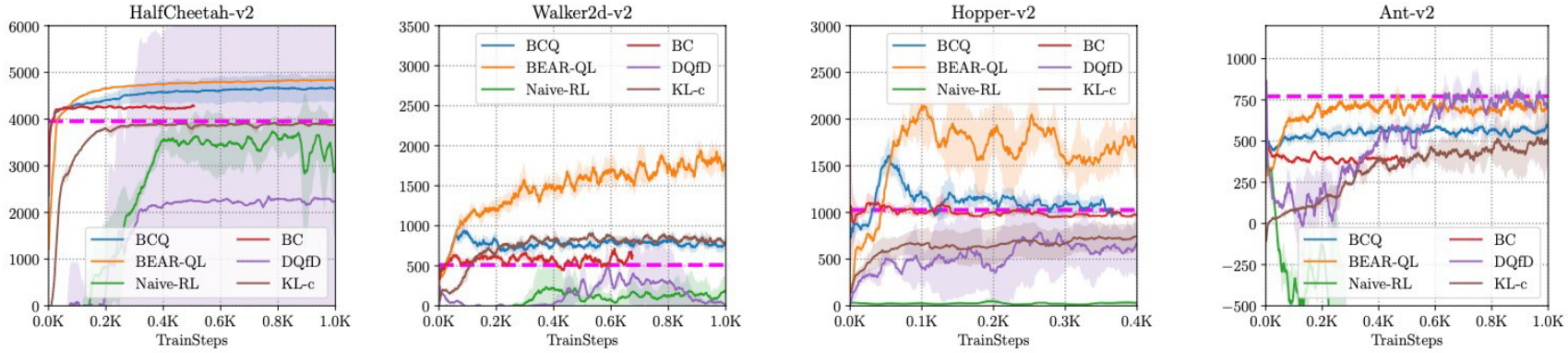


Figure 3: Average performance of BEAR-QL, BCQ, Naïve RL and BC on medium-quality data averaged over 5 seeds. BEAR-QL outperforms both BCQ and Naïve RL. Average return over the training data is indicated by the magenta line. One step on the x-axis corresponds to 1000 gradient steps.

Policy-Constrained Offline RL

BEAR – Results

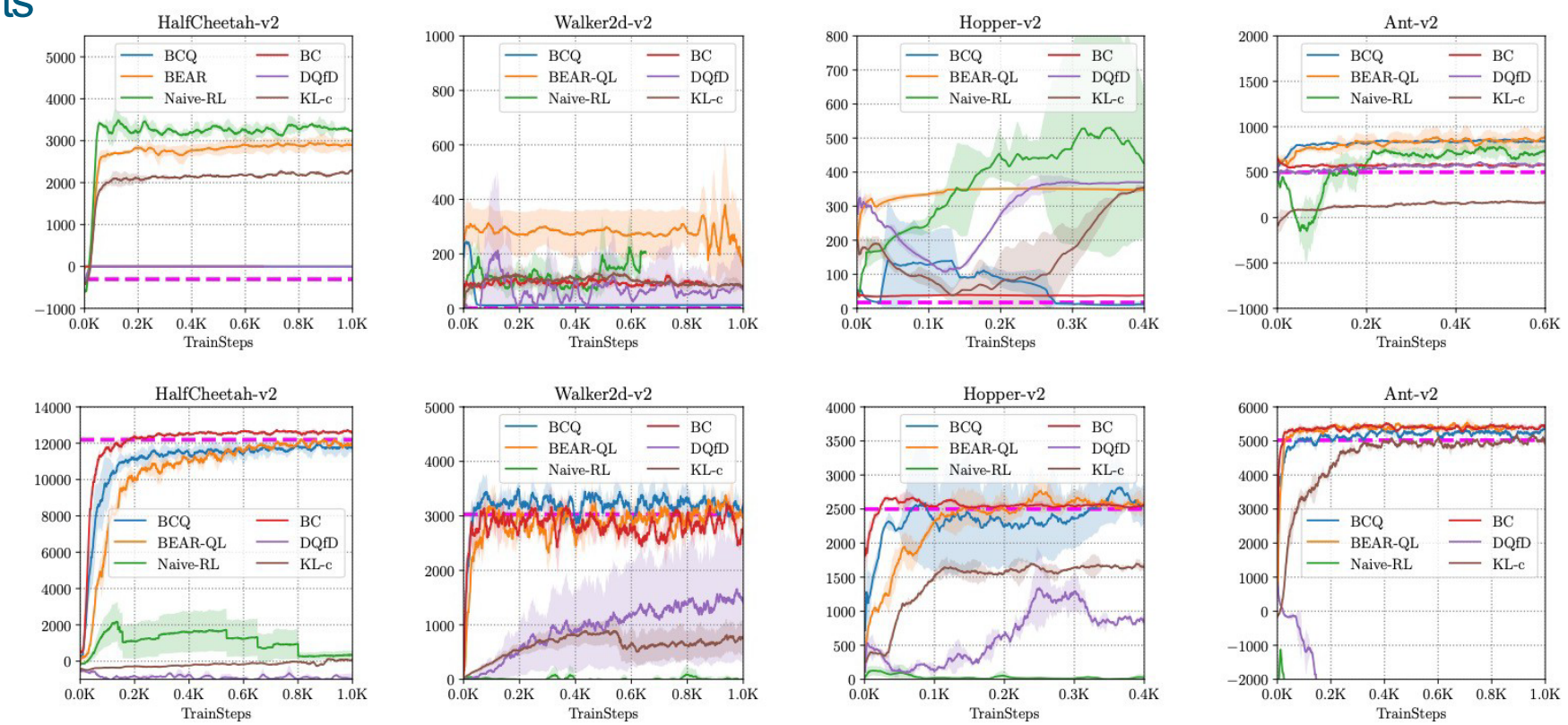


Figure 5: Average performance of BEAR-QL, BCQ, Naïve RL and BC on random data (top row) and optimal data (bottom row) over 5 seeds. BEAR-QL is the only algorithm capable of learning in both scenarios. Naïve RL cannot handle optimal data, since it does not illustrate mistakes, and BCQ favors a behavioral cloning strategy (performs quite close to behaviour cloning in most cases), causing it to fail on random data. Average return over the training dataset is indicated by the dashed magenta line.

A minimalist approach to offlineRL

- Can we make a deep RL algorithm work offline with minimal changes?

A Minimalist Approach to Offline Reinforcement Learning

Scott Fujimoto[1,2] **Shixiang Shane Gu**[2]
[1]Mila, McGill University
[2]Google Research, Brain Team
scott.fujimoto@mail.mcgill.ca

Abstract

Offline reinforcement learning (RL) defines the task of learning from a fixed batch of data. Due to errors in value estimation from out-of-distribution actions, most offline RL algorithms take the approach of constraining or regularizing the policy with the actions contained in the dataset. Built on pre-existing RL algorithms, modifications to make an RL algorithm work offline comes at the cost of additional complexity. Offline RL algorithms introduce new hyperparameters and often leverage secondary components such as generative models, while adjusting the underlying RL algorithm. In this paper we aim to make a deep RL algorithm work while making minimal changes. We find that we can match the performance of state-of-the-art offline RL algorithms by simply adding a behavior cloning term to the policy update of an online RL algorithm and normalizing the data. The resulting algorithm is a simple to implement and tune baseline, while more than halving the overall run time by removing the additional computational overhead of previous methods.

Idea: TD3+BC

- **TD3**: Twin Delayed Deep Deterministic Policy Gradient
  - Reduce overestimation in Q evaluation by

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{i,targ}}(s', a_{TD3}(s'))$$

    - double Q (target-)networks
    - the minimum between Q1 and Q2 as TD target
    - target policy to decouple the greedy a' selection and maximum $Q(s', a')$ estimation
    - add small noise in a' to smoothen target estimation

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s) + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \min_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

Addressing Function Approximation Error in Actor-Critic Methods, ICML'18

Idea: TD3+BC

- **TD3+behavior cloning**
  - Add BC loss in actor update loss

$$\pi = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{(s,a)\sim\mathcal{D}}[Q(s,\pi(s))]. \Rightarrow \pi = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\lambda Q(s,\pi(s)) - (\pi(s)-a)^2\right]$$

Stay close to offline action as policy constraint

  - Other implementation tricks
    - State normalization $s_i = \frac{s_i - \mu_i}{\sigma_i + \epsilon}$
    - Hyper-parameter selection $\lambda = \frac{\alpha}{\frac{1}{N}\sum_{(s_i,a_i)} |Q(s_i,a_i)|}$

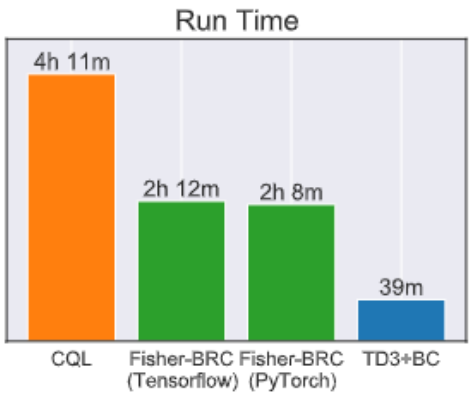
Idea: TD3+BC

- **TD3+behavior cloning**
  - **Simple and effective** (just one line of modification to TD3)

```
# Delayed policy updates
if self.total_it % self.policy_freq == 0:

        # Compute actor loss
        pi = self.actor(state)
        Q = self.critic.Q1(state, pi)
        lmbda = self.alpha/Q.abs().mean().detach()

        actor_loss = -lmbda * Q.mean() + F.mse_loss(pi, action)
```



		BC	BRAC-p	AWAC	CQL	Fisher-BRC	TD3+BC
Random	HalfCheetah	2.0 ±0.1	23.5	2.2	21.7 ±0.9	32.2 ±2.2	10.2 ±1.3
	Hopper	9.5 ±0.1	11.1	9.6	10.7 ±0.1	11.4 ±0.2	11.0 ±0.1
	Walker2d	1.2 ±0.2	0.8	5.1	2.7 ±1.2	0.6 ±0.6	1.4 ±1.6
Medium	HalfCheetah	36.6 ±0.6	44.0	37.4	37.2 ±0.3	41.3 ±0.5	42.8 ±0.3
	Hopper	30.0 ±0.5	31.2	72.0	44.2 ±10.8	99.4 ±0.4	99.5 ±1.0
	Walker2d	11.4 ±6.3	72.7	30.1	57.5 ±8.3	79.5 ±1.0	79.7 ±1.8
Medium Replay	HalfCheetah	34.7 ±1.8	45.6	-	41.9 ±1.1	43.3 ±0.9	43.3 ±0.5
	Hopper	19.7 ±5.9	0.7	-	28.6 ±0.9	35.6 ±2.5	31.4 ±3.0
	Walker2d	8.3 ±1.5	-0.3	-	15.8 ±2.6	42.6 ±7.0	25.2 ±5.1
Medium Expert	HalfCheetah	67.6 ±13.2	43.8	36.8	27.1 ±3.9	96.1 ±9.5	97.9 ±4.4
	Hopper	89.6 ±27.6	1.1	80.9	111.4 ±1.2	90.6 ±43.3	112.2 ±0.2
	Walker2d	12.0 ±5.8	-0.3	42.7	68.1 ±13.1	103.6 ±4.6	101.1 ±9.3
Expert	HalfCheetah	105.2 ±1.7	3.8	78.5	82.4 ±7.4	106.8 ±3.0	105.7 ±1.9
	Hopper	111.5 ±1.3	6.6	85.2	111.2 ±2.1	112.3 ±0.2	112.2 ±0.2
	Walker2d	56.0 ±24.9	-0.2	57.0	103.8 ±7.6	79.9 ±32.4	105.7 ±2.7
	Total	595.3 ±91.5	284.1	-	764.3 ±61.5	974.6 ±108.3	979.3 ±33.4

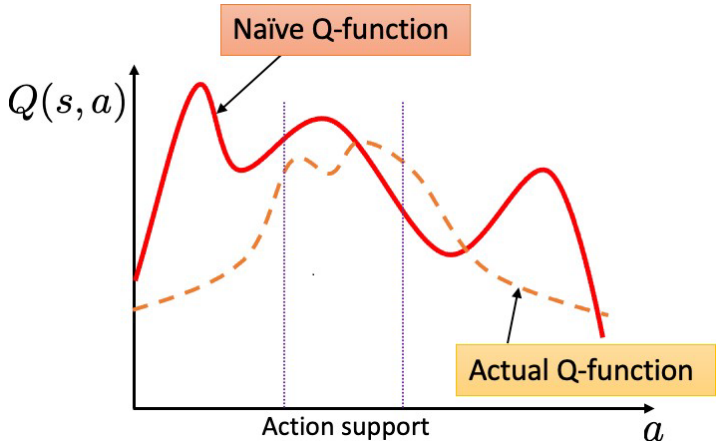
Conservative methods
(Q value regularization)

Conservative Offline RL

BCQ and BEAR – Uncertainty Perspective

- Different areas of the Q-Function are associated with different degrees of (un-)certainty
- BEAR and BCQ address the uncertainty by restricting the policy to certain areas where we think we are certain

How else could we address uncertainty?



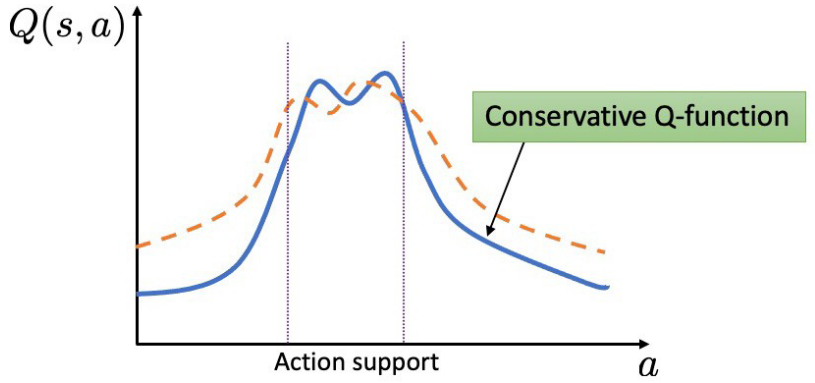
https://bair.berkeley.edu/blog/2020/12/07/offline/

Conservative Offline RL

Conservative Q-Learning

Idea of CQL[1]:

- Instead of constraining the action set for bootstrapping,…
- ...be conservative in the estimation of state-action-values that are not in the dataset!
  - Tackles the problem in the policy evaluation by staying conservative in uncertain areas
  - *Implicitly* keeps the policy away from out of distribution actions



→ ***Learn a value function such that the estimated performance of the policy under this learned value function lower-bounds its true value!***

https://bair.berkeley.edu/blog/2020/12/07/offline/

[1] *Aviral Kumar et al.: Conservative Q-Learning for Offline Reinforcement Learning. NeurIPS. 2020.*

Conservative Offline RL

Conservative Evaluation

- **Goal:** find lower bound to Q_π using data $\mathcal{D}$ collected from π_β
- Let $\hat{T}_\pi$ be the empirical Bellman operator for policy π estimated from $\mathcal{D}$
- **CQL regularization**
  - Minimize $Q(s, a)$ on $s \in \mathcal{D}$, $a \sim \pi(s)$
  - For $a \sim \pi_\beta(s)$ there is no need to be conservative → additionally maximize
- CQL optimization problem for policy evaluation

$$\min_Q \alpha \cdot \left(\mathbb{E}_{s\sim\mathcal{D},a\sim\pi(s)}[Q(s,a)] - \mathbb{E}_{s,a\sim\mathcal{D}}[Q(s,a)]\right) + \frac{1}{2}\mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[\left(Q(s,a) - \hat{T}_\pi Q(s,a)\right)^2\right],$$

where α is a trade-off parameter to control how conservative we are.

Conservative Offline RL

Conservative Control

- **Goal:** Find a good policy π from offline data $\mathcal{D}$ collected from π_β
- Iterate between policy improvement and conservative policy evaluation
- Online Conservative Q-Learning (π is the current policy)

$$\max_{\mu} \min_{Q} \left(\alpha \mathbb{E}_{s \sim \mathcal{D}, a \sim \mu(a|s)}[Q(s,a)] - \alpha \mathbb{E}_{s,a \sim \mathcal{D}}[Q(s,a)] \right) + \frac{1}{2} \mathbb{E}_{s,a,s' \sim \mathcal{D}} \left[\left(Q(s,a) - \hat{T}^{\pi} \hat{Q}^{k}(s,a) \right)^2 \right]$$

- Policy improvement can be done using an actor or standard Q-Learning

Conservative Offline RL

CQL - Performance

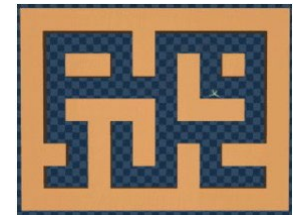
- Ant Maze:
  - Navigate from start to goal
  - Dataset consists of random motions and no single trajectory solves the task
  - The Offline RL algorithm needs to “stich” different sub-trajectories

D4RL Ant Maze	BC	SAC	BEAR	BRAC	AWR	BCQ	AlgaeDICE	CQL
U-Maze	65.0	0.0	73.0	70.0	56.0	**78.9**	0.0	74.0
U-maze + diverse	55.0	0.0	61.0	70.0	70.3	55.0	0.0	**84.0**
Medium + play	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**61.2**
Medium + diverse	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**53.7**
Large + play	0.0	0.0	0.0	0.0	0.0	6.7	0.0	**15.8**
Large + diverse	0.0	0.0	0.0	0.0	0.0	2.2	0.0	**14.9**

U-Maze

Medium

Large Maze



https://bair.berkeley.edu/blog/2020/12/07/offline/

Conservative Offline RL

Conservative Q-Learning

- It is proven that the return-estimate of the learned policy π under Q^{π}_{CQL} is a lower-bound on the actual policy performance:

$$J_{\mathrm{CQL}}(\pi) = \mathbb{E}_{s_0 \sim \mathrm{init}, a_0 \sim \pi}\left[Q^{\pi}_{\mathrm{CQL}}(s_0, a_0)\right] \leq E_{s_0 \sim \mathrm{init}, a_0 \sim \pi}[Q^{\pi}(s_0, a_0)] = J(\pi)$$

- We only need to add a regularizer term (which can be estimated from the dataset) during training
  - No need to estimate the behavior policy (as previous work needs)

Standard actor-critic algorithm

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

CQL algorithm

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

https://bair.berkeley.edu/blog/2020/12/07/offline/

Conservative Offline RL

Model-based Offline Policy Optimization

Model-based Offline Reinforcement Learning

1. Estimate $r(s, a)$ and $p(s'|a, s)$ from $\mathcal{D}$
2. Apply active reinforcement learning to the model

- The transitions (s, a, r, s') can now be collected "online" (established model) → no distribution shift
- Nonetheless if π is very different from π_β our model is queried in unknown regions

→ We still need to address the uncertainty!

Conservative Offline RL

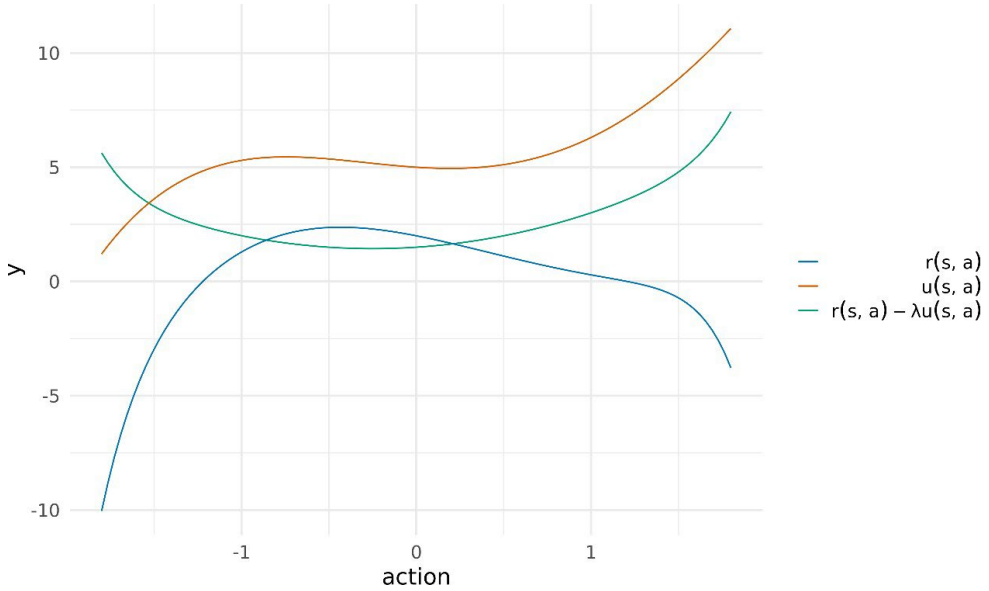
Model-based Offline Policy Optimization

- By Yu, Thomas, Yu, Ermon, Zou, Levine, Finn and Ma, 2020

Algorithm

1. Estimate $\mathrm{r}(s, a)$ and $p(s^{"}|a, s)$ from $\mathcal{D}$
2. Learn uncertainty quantification $u(s, a)$ ← This is the hard part
3. Apply active reinforcement learning to the model using a pessimistic reward function

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



- Practical algorithm uses ensemble of models:

$$\{\widehat{T}^i_{\theta,\phi} = \mathcal{N}(\mu^i_\theta, \Sigma^i_\phi)\}^N_{i=1} \qquad \widehat{T}_{\theta,\phi}(s_{t+1}, r|s_t, a_t) = \mathcal{N}(\mu_\theta(s_t, a_t), \Sigma_\phi(s_t, a_t))$$

- Penalize reward function using heuristic uncertainty estimate:

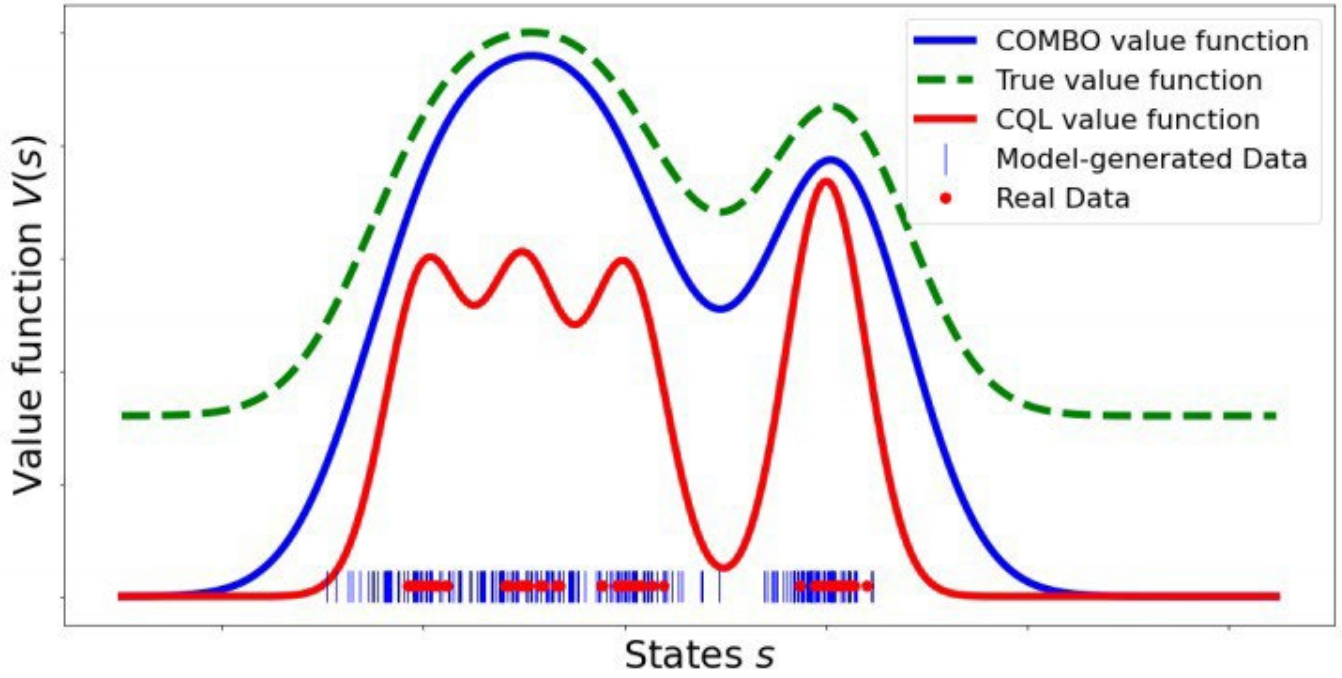
$$u(s, a) = \max^N_{i=1} \|\Sigma^i_\phi(s, a)\|_{\mathrm{F}}$$

the maximum standard deviation of the learned models in the ensemble

Conservative Offline RL

Conservative Offline Model Based Policy Optimization

- by Yu, Kumar, Rafailov, Rajeswaran, Levine and Finn, 2021
- **Problem in MOPO:** Uncertainty quantification $u(s, a)$ is notoriously difficult for neural networks
- COMBO addresses this by combining CQL with rollouts from a model



Conservative Offline RL

Conservative Offline Model Based Policy Optimization

CQL objective

$$\min_Q \left(\alpha \mathbb{E}_{s\sim\mathcal{D},a\sim\pi(s)}[Q(s,a)] - \alpha \mathbb{E}_{s,a\sim\mathcal{D}}[Q(s,a)]\right) + \frac{1}{2}\mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[\left(Q(s,a) - \hat{T}_\pi Q(s,a)\right)^2\right]$$

- Situation: Provided with buffer $\mathcal{D}$ and a model estimated from $\mathcal{D}$
- Let $\hat{d}_\pi$ be the marginal state distribution under π under our model
- Let $d_\pi^f(s,a)$ be the f-interpolation between the offline data and rollouts from the learned model
  - Sample from $\mathcal{D}$ with probability f
  - Sample from $d_\pi^f(s,a)$ with probability $1-f$

COMBO objective

$$\min_Q \left(\alpha \mathbb{E}_{s\sim d_\pi^f,a\sim\pi(s)}[Q(s,a)] - \alpha \mathbb{E}_{s,a\sim\mathcal{D}}[Q(s,a)]\right) + \frac{1}{2}\mathbb{E}_{s,a,s'\sim d_\pi^f}\left[\left(Q(s,a) - \hat{T}_\pi Q(s,a)\right)^2\right]$$

- Note that we only maximize on $s, a\sim\mathcal{D}$ → Higher trust in original data than model rollouts!

Conservative Offline RL

Conservative Offline Model Based Policy Optimization

CQL objective

$$\min_Q \left(\alpha \mathbb{E}_{s\sim\mathcal{D},a\sim\pi(s)}[Q(s,a)] - \alpha \mathbb{E}_{s,a\sim\mathcal{D}}[Q(s,a)]\right) + \frac{1}{2}\mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[\left(Q(s,a) - \hat{T}_\pi Q(s,a)\right)^2\right]$$

COMBO objective

$$\min_Q \left(\alpha \mathbb{E}_{s\sim d_\pi^f,a\sim\pi(s)}[Q(s,a)] - \alpha \mathbb{E}_{s,a\sim\mathcal{D}}[Q(s,a)]\right) + \frac{1}{2}\mathbb{E}_{s,a,s'\sim d_\pi^f}\left[\left(Q(s,a) - \hat{T}_\pi Q(s,a)\right)^2\right]$$

COMBO advantages over CQL

- Uses more data (i.e. from the model)
- The data is correlated with the policy

Conservative Offline RL

COMBO – Algorithm

Algorithm 1 COMBO: Conservative Model Based Offline Policy Optimization

Require: Offline dataset $\mathcal{D}$, rollout distribution $\mu(\cdot|\mathbf{s})$, learned dynamics model $\widehat{T}_\theta$, initialized policy and critic π_ϕ and Q_ψ.

1: Train the probabilistic dynamics model $\widehat{T}_\theta(\mathbf{s}', r|\mathbf{s}, \mathbf{a}) = \mathcal{N}(\mu_\theta(\mathbf{s}, \mathbf{a}), \Sigma_\theta(\mathbf{s}, \mathbf{a}))$ on $\mathcal{D}$.
2: Initialize the replay buffer $\mathcal{D}_{\text{model}} \leftarrow \varnothing$.
3: **for** $i = 1, 2, 3, \cdots$, **do**
4: Perform model rollouts by drawing samples from μ and $\widehat{T}_\theta$ starting from states in $\mathcal{D}$. Add model rollouts to $\mathcal{D}_{\text{model}}$.
5: Conservatively evaluate current policy by repeatedly solving eq. 4 to obtain $\hat{Q}_\psi^{\pi_\phi^i}$ using data sampled from $\mathcal{D} \cup \mathcal{D}_{\text{model}}$.
6: Improve policy under state marginal of d_f by solving eq. 5 to obtain π_ϕ^{i+1}.
7: **end for**

$$\hat{Q}^{k+1} \leftarrow \arg\min_Q \beta \left(\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\rho(\mathbf{s},\mathbf{a})}[Q(\mathbf{s},\mathbf{a})] - \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q(\mathbf{s},\mathbf{a})]\right) + \frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim d_f}\left[\left(Q(\mathbf{s},\mathbf{a}) - \widehat{\mathcal{B}}^\pi \hat{Q}^k(\mathbf{s},\mathbf{a})\right)^2\right]. \tag{4}$$

$$\pi' \leftarrow \arg\max_\pi \mathbb{E}_{\mathbf{s}\sim\rho,\mathbf{a}\sim\pi(\cdot|\mathbf{s})}\left[\hat{Q}^\pi(\mathbf{s},\mathbf{a})\right] \tag{5}$$

Conservative Offline RL

COMBO – Experimental Results

Dataset type	Environment	BC	COMBO (ours)	MOPO	CQL	SAC-off	BEAR	BRAC-p	BRAC-v
random	halfcheetah	2.1	**38.8**	35.4	35.4	30.5	25.1	24.1	31.2
random	hopper	1.6	**17.9**	11.7	10.8	11.3	11.4	11.0	12.2
random	walker2d	9.8	7.0	**13.6**	7.0	4.1	7.3	-0.2	1.9
medium	halfcheetah	36.1	**54.2**	42.3	44.4	-4.3	41.7	43.8	46.3
medium	hopper	29.0	**94.9**	28.0	86.6	0.8	52.1	32.7	31.1
medium	walker2d	6.6	75.5	17.8	74.5	0.9	59.1	77.5	**81.1**
medium-replay	halfcheetah	38.4	**55.1**	53.1	46.2	-2.4	38.6	45.4	47.7
medium-replay	hopper	11.8	**73.1**	67.5	48.6	3.5	33.7	0.6	0.6
medium-replay	walker2d	11.3	**56.0**	39.0	32.6	1.9	19.2	-0.3	0.9
med-expert	halfcheetah	35.8	**90.0**	63.3	62.4	1.8	53.4	44.2	41.9
med-expert	hopper	111.9	**111.1**	23.7	111.0	1.6	96.3	1.9	0.8
med-expert	walker2d	6.4	96.1	44.6	**98.7**	-0.1	40.1	76.9	81.6

Implicit Policy Constraint

Recall: Policy Improvement

- Suppose we already have an evaluation $Q(s,a)$, a new improved policy should be

$$\pi'(s) = \arg\max_{\pi} \sum_{a} \pi(s,a) \cdot Q(s,a)$$

- But in offline RL, we are uncertain of $Q(s,a)$ in OOD of $\mathcal{D}$, instead, we select greedy actions with the support of behavior π_β

$$\pi'(s) = \arg\max_{\pi \in \Pi_\beta} \sum_{a} \pi(s,a) \cdot Q(s,a)$$

  - where e.g. $\Pi_\beta = \{\pi | \forall a, \pi(s,a) > 0 \Rightarrow \frac{\pi_\beta(s,a)}{\max_a \pi_\beta(s,a)} \geq \epsilon\}$ (BCQ) or $\Pi_\beta = \{\pi | \forall a, \pi(s,a) > 0 \Rightarrow \pi_\beta(s,a) \geq \epsilon\}$ (BEAR)

- Q function can be replaced by advantage function for policy improvement

$$Q(s,a) = V(s) + A(s,a)$$

Construct the constrained policy space Π_β is ***computationally cumbersome***, is there a simple way to restrict the policy improvement process? **Yes!**

Advantage-Weighted Objective

- **Advantage-weighted objective approximates KL-constrained objective**

$$\pi'(s) = \arg\max_{\pi} \sum_{a} \pi(s,a) \cdot A(s,a) \qquad \text{s.t. } D_{KL}(\pi || \pi_\beta) \leq \epsilon$$

  - Closed-form solution (Lagrange's method)

$$\pi'(s,a) = \frac{1}{Z(s)} \pi_\beta(s,a) \exp\left(\frac{1}{\epsilon} A(s,a)\right)$$

$Z(s)$ for normalization

- In practice, policy is parameterized and the objective is to find θ that maximized

$$\theta = \arg\max_{\theta} \mathbb{E}_{(s,a)\sim\mathcal{D}}[D_{KL}(\pi'(\cdot|s)||\pi(\cdot|s))]$$

$$= \arg\max_{\theta} \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[\log \pi(a|s) \times \exp\left(\frac{1}{\epsilon} A(s,a)\right)\right]$$

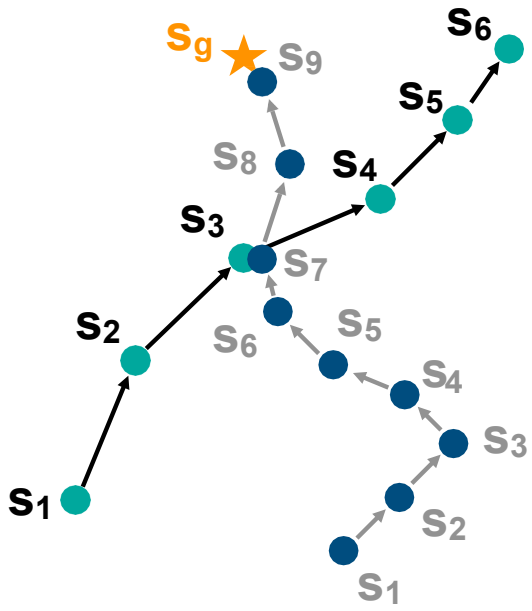
standard imitation learning with weighted advantage

(only learn on seen/known actions)

Relative Entropy Policy Search, AAAI'10

Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning, 2019

Example



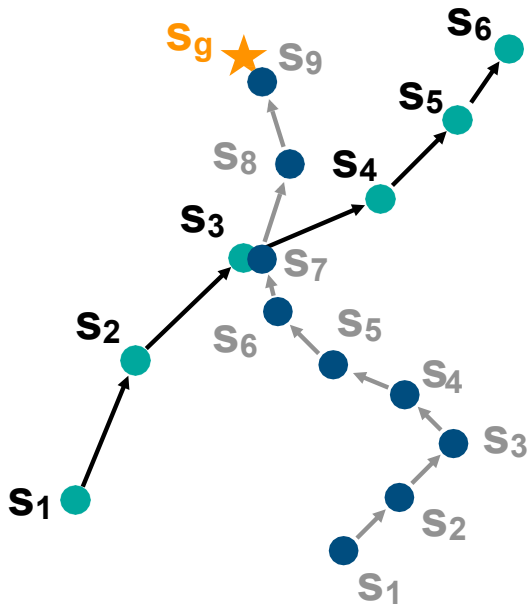
Given two offline trajectories

- s1, s2, s3, …, s6
- s1, s2, s3, …, …, s9(sg)

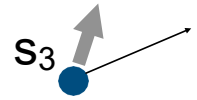
Neither trajectories are perfect, but both includes perfect *segments*

- s1 -> s3 is good behavior
- s7 -> s9 is good behavior

Example



Adv. weights for s_3



Given two offline trajectories

- s1->s2->s3-> …->s6
- s1->s2->s3->…-> …->s9(sg)

Neither trajectories are perfect, but both includes perfect *segments*

- s1 -> s3 is good behavior
- s7 -> s9 is good behavior

Offline RL methods can learn a policy that goes from s_1 to s_9!

$$\theta = \underbrace{\arg\max_\theta \mathbb{E}_{(s,a)\sim\mathcal{D}}}_{\text{standard imitation learning}} \Big[\log \pi(a|s) \times \underbrace{\exp\left(\frac{1}{\epsilon} A(s,a)\right)}_{\text{weighted advantage}} \Big]$$

Advantage-Weighted Regression

$$\theta = \arg\max_{\theta} \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[\log \pi(a|s) \times \exp\left(\frac{1}{\epsilon} A(s,a) \right) \right]$$

- **Another question**: how to estimate the *advantage* function?
  - Approach 1
    - Estimate V^{π_β} with Monte Carlo $\min_{V} \mathbb{E}_{s,a\sim\mathcal{D}}[(G_{s,a} - V(s))^2]$
    - Approximate $A(s,a) = G_{s,a} - V(s)$

G: trajectory return

+ Simple
+ Avoids querying or training on any OOD actions!

- Monte Carlo estimation is noisy
- A^{π_β} assumes weaker policy than A^{π_θ}

Advantage-Weighted Regression

$$\theta = \arg\max_{\theta} \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[\log \pi(a|s) \times \exp\left(\frac{1}{\epsilon} A(s,a) \right) \right]$$

- **Another question**: how to estimate the *advantage* function?
  - Approach 2: Temporal Difference instead of MC

1. Estimate Q^π-function: $\min_Q E_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim D} \left[\left(Q(\mathbf{s},\mathbf{a}) - \left(r + \gamma E_{\mathbf{a}'\sim\pi(\cdot|\mathbf{s})}[Q(\mathbf{s}',\mathbf{a}')] \right) \right)^2 \right]$
2. Estimate advantage as: $\hat{A}^\pi(\mathbf{s},\mathbf{a}) = \hat{Q}^\pi(\mathbf{s},\mathbf{a}) - E_{\bar{\mathbf{a}}\sim\pi(\cdot|\mathbf{s})}[\hat{Q}^\pi(\mathbf{s},\bar{\mathbf{a}})]$
3. Update policy as before: $\hat{\pi} \leftarrow \arg\max_\pi E_{\mathbf{s},\mathbf{a}\sim D} \left[\log \pi(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\alpha} \hat{A}^\pi(\mathbf{s},\mathbf{a}) \right) \right]$

"advantage weighted actor critic"

+ Policy still only trained on actions in data.
+ Temporal difference updates instead of Monte Carlo.

- Possibly querying OOD actions!

Nair, Gupta, Dalal, Levine. AWAC. '20 / Wang et al. Critic Regularized Regression. NeurIPS '20

Can we do better?

- Want to estimate advantages using TD updates, without querying Q on OOD actions

AWAC: Estimate Q-function: $\min_Q E_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim D}\left[\left(Q(\mathbf{s},\mathbf{a}) - \left(r + \gamma E_{\mathbf{a}'\sim\pi(\cdot|\mathbf{s})}[Q(\mathbf{s}',\mathbf{a}')]\right)\right)^2\right]$

$\mathbf{a}' \sim D$

"SARSA algorithm"

SARSA update: $\hat{Q}^{\pi_\beta} \leftarrow \arg\min_Q E_{(\mathbf{s},\mathbf{a},\mathbf{s}',\mathbf{a}')\sim D}\left[\left(Q(\mathbf{s},\mathbf{a}) - \left(r + \gamma Q(\mathbf{s}',\mathbf{a}')\right)\right)^2\right]$

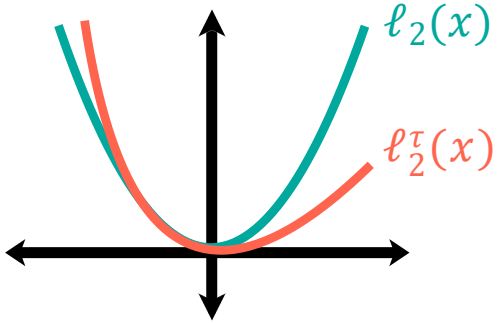
a sample of $V^{\pi_\beta}(\mathbf{s}')$

- Can we estimate Q for a policy that is better than π_β?

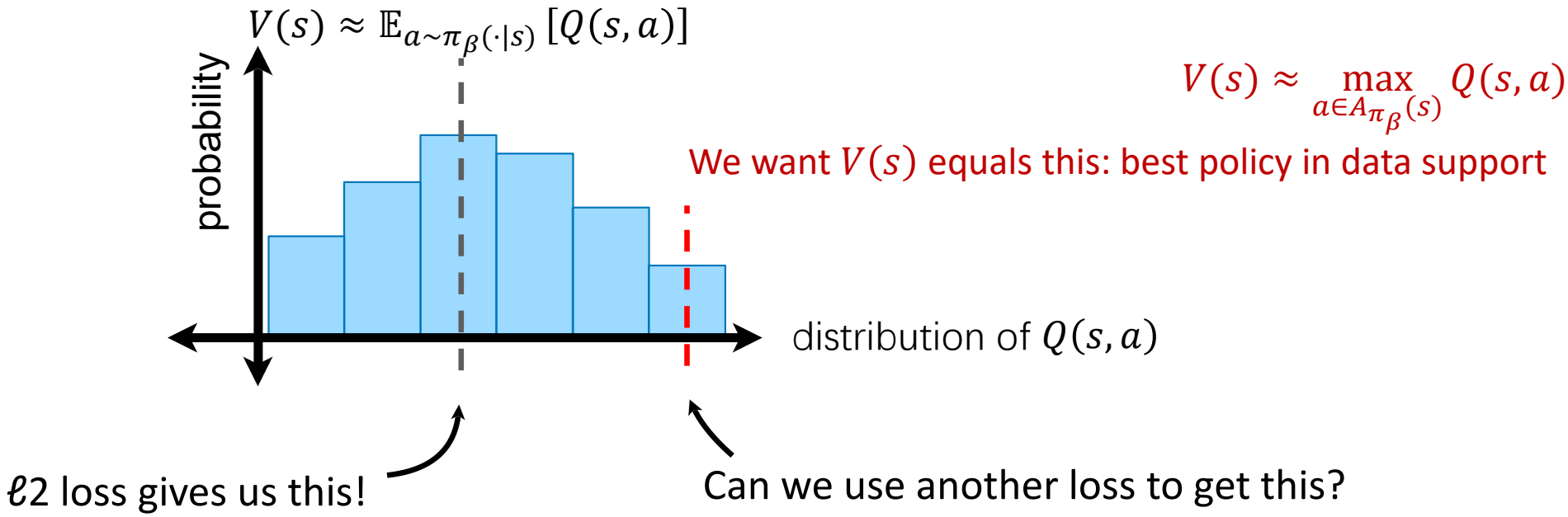
$$Q(s,a) \longleftarrow r + \gamma Q(s',a')$$

$$V^{\pi_\beta}(s') \longleftarrow \min_V \mathbb{E}_{s,a\sim\mathcal{D}}\left[\left(V(\mathbf{s}) - Q(s,a)\right)^2\right]$$

loss ℓ_2



Aside: Expectile regression



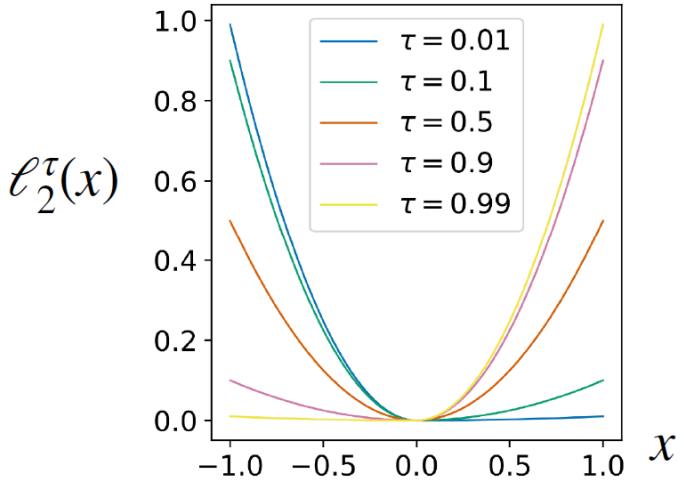
- Idea: Use an asymmetric loss function

Aside: Expectile regression

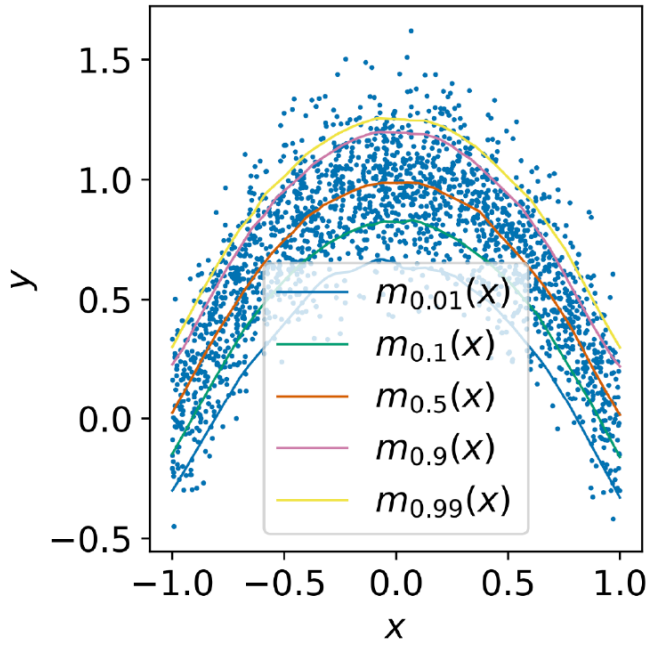
- Instead of getting the mean of a random variable, we can get a higher or lower expectile

Expectile regression loss:

$$\ell_2^\tau(x) = \begin{cases} (1-\tau)x^2 & \text{if } x < 0 \\ \tau x^2 & \text{otherwise} \end{cases}$$



Example with a 2D random variable



Kostrikov, Nair, Levine. Implicit Q-Learning. ICLR ‘22

Can we do better? Implicit Q-Learning (IQL)!

- Want to estimate advantages using TD updates, without querying Q on OOD actions

Full algorithm

Fit V with expectile loss: $\hat{V}(\mathbf{s}) \leftarrow \arg\min_V E_{(\mathbf{s},\mathbf{a})\sim D}\left[\ell_2^\tau\left(V(\mathbf{s}) - \hat{Q}(\mathbf{s},\mathbf{a})\right)\right]$ using small $\tau < 0.5$

Update Q with typical MSE loss: $\hat{Q}(\mathbf{s},\mathbf{a}) \leftarrow \arg\min_Q E_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim D}\left[\left(Q(\mathbf{s},\mathbf{a}) - \left(r + \gamma\hat{V}(\mathbf{s}')\right)\right)^2\right]$

Extract policy with AWR: $\hat{\pi} \leftarrow \arg\max_\pi E_{\mathbf{s},\mathbf{a}\sim D}\left[\log\pi(\mathbf{a}\,|\,\mathbf{s})\exp\left(\frac{1}{\alpha}\left(\hat{Q}(\mathbf{s},\mathbf{a}) - \hat{V}(\mathbf{s})\right)\right)\right]$

+ Never need to query OOD actions!
+ Policy (still) only trained on actions in data.
+ Decoupling actor & critic training —> computationally fast

Kostrikov, Nair, Levine. Implicit Q-Learning. ICLR ‘22

Revisit Behavior Cloning

Recall: Behavior Cloning

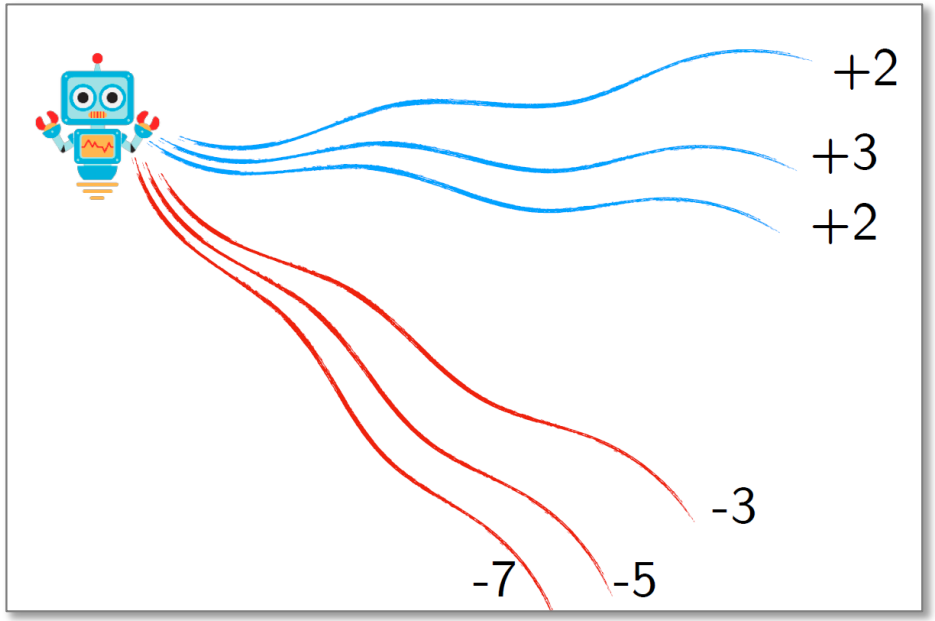
- A simple & powerful framework for learning behavior!
  - Scenario: Data $\mathcal{D}$ collected from a behavioural agent π_β
  - Goal: Learn policyπ_β from $\mathcal{D}$ with supervised learning

- But!
  - Collecting expert demonstrations can be difficult or impossible in some scenarios
  - Learned behavior will never be better than expert
  - Does not provide a framework for learning from experience, indirect feedback

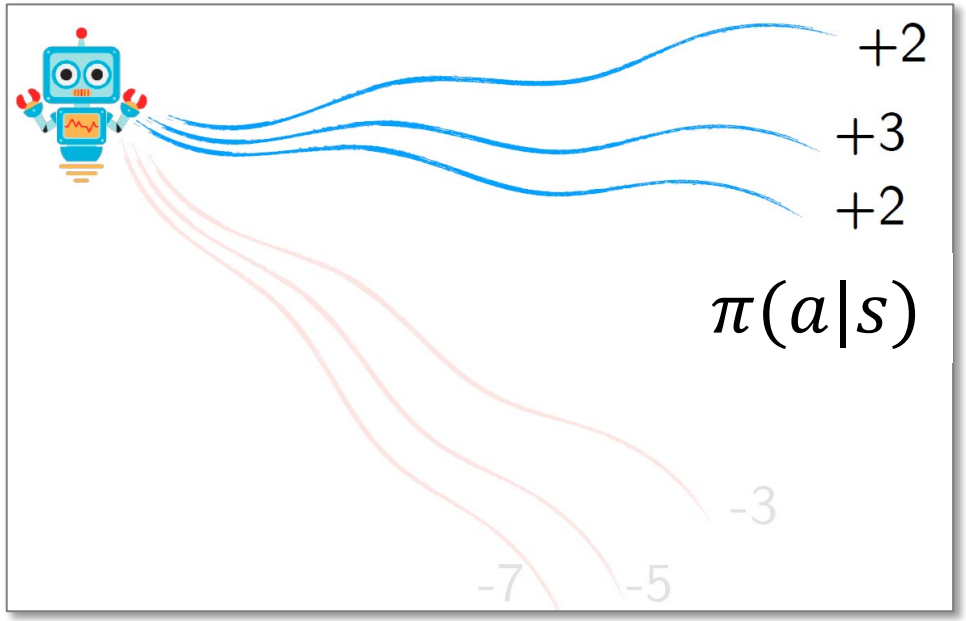
- What about imitate only the good trajectories?

Filtered Behavior Cloning

Supervised Learning
on all rollouts (good/bad)



Supervised Learning
only on the top % rollouts



Filtered Behavior Cloning

- BAIL: Best-Action Imitation Learning

Algorithm 1 Static BAIL

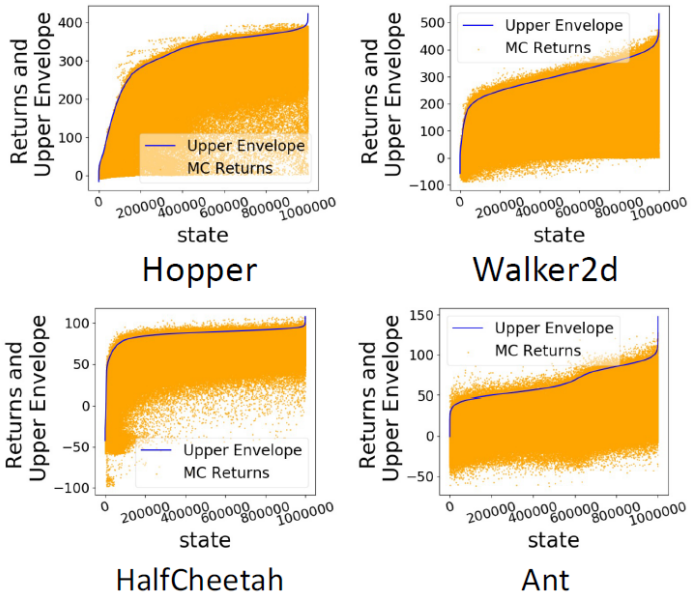
Initialize upper envelope parameters ϕ, ϕ', policy parameters θ. Obtain batch data $\mathcal{B}$. Randomly split data into training set $\mathcal{B}_t$ and validation set $\mathcal{B}_v$ for the upper envelope.
Compute return G_i for each data point i in $\mathcal{B}$.
Obtain upper envelope by minimizing the loss $L^K(\phi)$:
for $j = 1, \ldots, J$ **do**
  Sample a mini-batch B from $\mathcal{B}$.
  Update ϕ using the gradient: $\nabla_\phi \sum_{i \in B} (V_\phi(s_i) - G_i)^2 \{\mathbb{1}_{(V_\phi(s_i) > G_i)} + K \mathbb{1}_{(V_\phi(s_i) < G_i)}\} + \lambda \|\phi\|^2$
  if time to do validation for the upper envelope **then**
    Compute validation loss on B_v
    Update ϕ and ϕ' according to the validation loss
  end if
end for
Select data point i if $G_i > x V_\phi(s_i)$, where x is such that $p\%$ of data in $\mathcal{B}$ are selected. Let $\mathcal{U}$ be the set of selected data points.
for $l = 1, \ldots, L$ **do**
  Sample a mini-batch U of data from $\mathcal{U}$.
  Update θ using the gradient: $\nabla_\theta \sum_{i \in U} (\pi_\theta(s_i) - a_i)^2$
end for

Step 1: learn an *upper envelope* of state by solving a constrained optimization problem:

$$\min_{\phi} \sum_{i=1}^{m} [V_\phi(s_i) - G_i]^2 + \lambda \|w\|^2$$
$$s.t. \qquad V_\phi(s_i) \geq G_i, \qquad i = 1, 2, \ldots, m$$

or its unconstrained penalty-loss version

$$L^K(\phi) = \sum_{i=1}^{m} (V_\phi(s_i) - G_i)^2 \{\mathbb{1}_{(V_\phi(s_i) \geq G_i)} + K \cdot \mathbb{1}_{(V_\phi(s_i) < G_i)}\} + \lambda \|w\|^2$$



X. Chen et al. BAIL: Best-action imitation learning for batch deep reinforcement learning. NeuIPS 2020.

Filtered Behavior Cloning

- BAIL: Best-Action Imitation Learning

Algorithm 1 Static BAIL

Initialize upper envelope parameters ϕ, ϕ', policy parameters θ. Obtain batch data $\mathcal{B}$. Randomly split data into training set $\mathcal{B}_t$ and validation set $\mathcal{B}_v$ for the upper envelope.
Compute return G_i for each data point i in $\mathcal{B}$.
Obtain upper envelope by minimizing the loss $L^K(\phi)$:
for $j = 1, \ldots, J$ **do**
  Sample a mini-batch B from $\mathcal{B}$.
  Update ϕ using the gradient: $\nabla_\phi \sum_{i \in B} (V_\phi(s_i) - G_i)^2 \{\mathbb{1}_{(V_\phi(s_i) > G_i)} + K \mathbb{1}_{(V_\phi(s_i) < G_i)}\} + \lambda \|\phi\|^2$
  if time to do validation for the upper envelope **then**
    Compute validation loss on B_v
    Update ϕ and ϕ' according to the validation loss
  end if
end for
Select data point i if $G_i > xV_\phi(s_i)$, where x is such that $p\%$ of data in $\mathcal{B}$ are selected. Let $\mathcal{U}$ be the set of selected data points.
for $l = 1, \ldots, L$ **do**
  Sample a mini-batch U of data from $\mathcal{U}$.
  Update θ using the gradient: $\nabla_\theta \sum_{i \in U} (\pi_\theta(s_i) - a_i)^2$
end for

Environment	BAIL	BCQ	BEAR	BC	MARWIL
$\sigma = 0.1$ Hopper B1	**2173 ± 291**	1219 ± 114	505 ± 285	626 ± 112	827 ± 220
$\sigma = 0.1$ Hopper B2	**2078 ± 180**	1178 ± 87	985 ± 3	579 ± 141	620 ± 336
$\sigma = 0.1$ Walker B1	**1125 ± 113**	576 ± 309	610 ± 212	514 ± 17	436 ± 24
$\sigma = 0.1$ Walker B2	**3141 ± 300**	2338 ± 388	2707 ± 425	1741 ± 239	1810 ± 200
$\sigma = 0.1$ HC B1	**5746 ± 29**	**5883 ± 43**	0 ± 0	**5546 ± 29**	**5573 ± 35**
$\sigma = 0.1$ HC B2	**7212 ± 43**	**7562 ± 31**	0 ± 0	6765 ± 108	**6828 ± 111**
$\sigma = 0.5$ Hopper B1	**2054 ± 158**	1145 ± 300	203 ± 42	919 ± 52	946 ± 103
$\sigma = 0.5$ Hopper B2	**2623 ± 282**	1823 ± 555	241 ± 239	694 ± 64	818 ± 112
$\sigma = 0.5$ Walker B1	**2522 ± 51**	1552 ± 455	1248 ± 181	2178 ± 178	2111 ± 52
$\sigma = 0.5$ Walker B2	**3115 ± 133**	2785 ± 123	2302 ± 630	2483 ± 94	2364 ± 228
$\sigma = 0.5$ HC B1	1055 ± 9	**1222 ± 38**	924 ± 579	570 ± 35	512 ± 43
$\sigma = 0.5$ HC B2	**7173 ± 120**	5807 ± 249	−114 ± 140	**6545 ± 171**	**6668 ± 93**
SAC Hopper B1	**3296 ± 105**	2681 ± 438	1000 ± 110	2853 ± 318	2897 ± 227
SAC Hopper B2	1831 ± 915	**2134 ± 917**	1139 ± 317	**2240 ± 367**	**2063 ± 168**
SAC Walker B1	**2455 ± 211**	**2408 ± 84**	−3 ± 5	1674 ± 277	1484 ± 140
SAC Walker B2	**4767 ± 130**	3794 ± 398	325 ± 75	2599 ± 145	2651 ± 268
SAC HC B1	**10143 ± 77**	8607 ± 473	7392 ± 257	8874 ± 221	9105 ± 90
SAC HC B2	**10772 ± 59**	**10106 ± 134**	7217 ± 273	9523 ± 164	9488 ± 136
SAC Ant B1	**4284 ± 64**	**4042 ± 113**	3452 ± 128	**3986 ± 112**	**4033 ± 130**
SAC Ant B2	**4946 ± 148**	**4640 ± 76**	3712 ± 236	**4618 ± 111**	**4589 ± 130**
SAC Humanoid B1	**3852 ± 430**	1411 ± 250	0 ± 0	543 ± 378	589 ± 121
SAC Humanoid B2	**3565 ± 153**	1221 ± 207	0 ± 0	1216 ± 826	1033 ± 257

Step 2: Select actions satisfying $G_i > xV(s_i)$ to perform simple imitation learning (BC). x is set such that 25% samples are selected.

X. Chen et al. BAIL: Best-action imitation learning for batch deep reinforcement learning. NeuIPS 2020.

Filtered Behavior Cloning

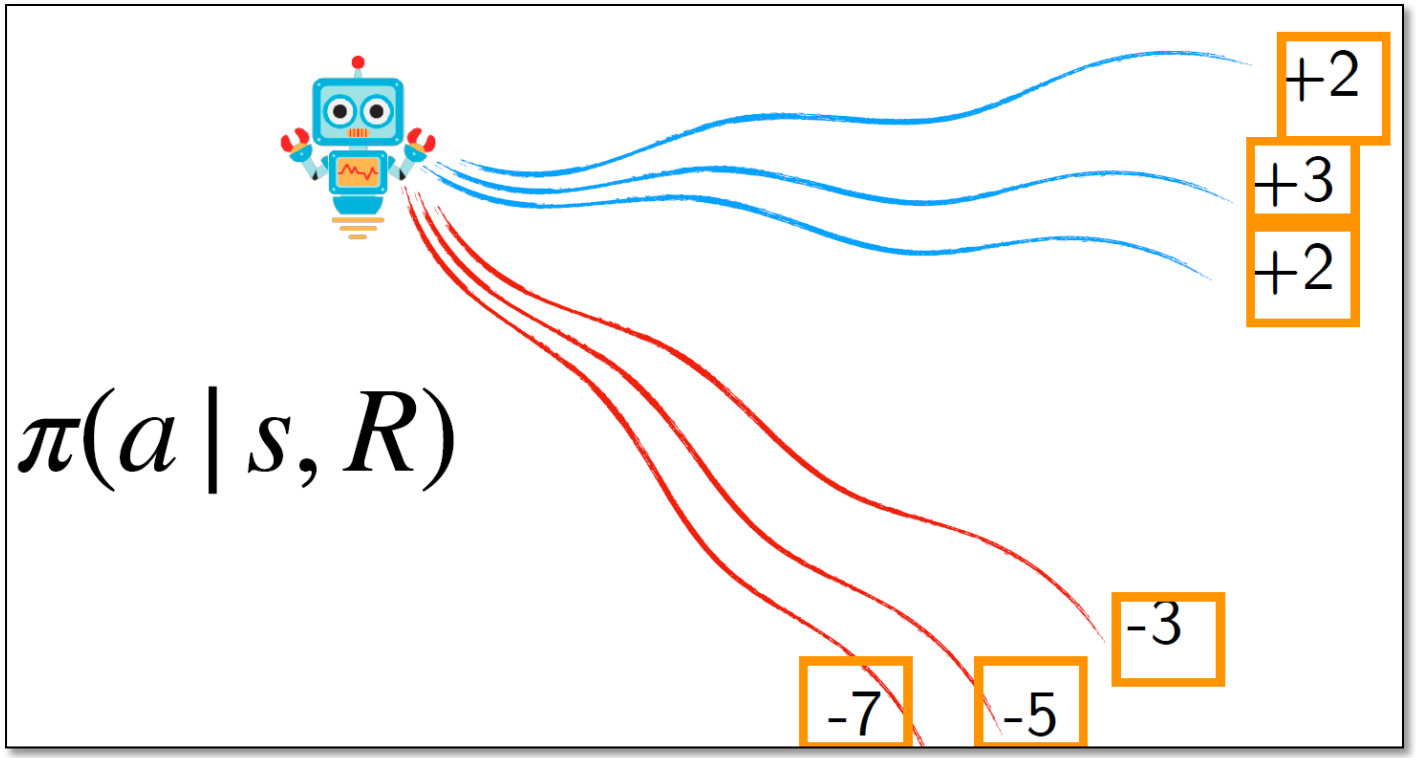
+ A very primitive approach to using reward information.

+ For some datasets, filtered BC can actually work really well!

- For the rest non-top% data, they are just discarded
  (in policy improvement stage, not policy evaluation stage)

- What if we feel bad about discarding data?

Idea: Train a policy conditioned on the returns



Recall that: What can go wrong in imitation learning?

Problem3: Expert has more information than is observed by the agent.

Solution: With return as condition, policy has more information to replicate behavior.

Return-conditioned policies

At train stage:

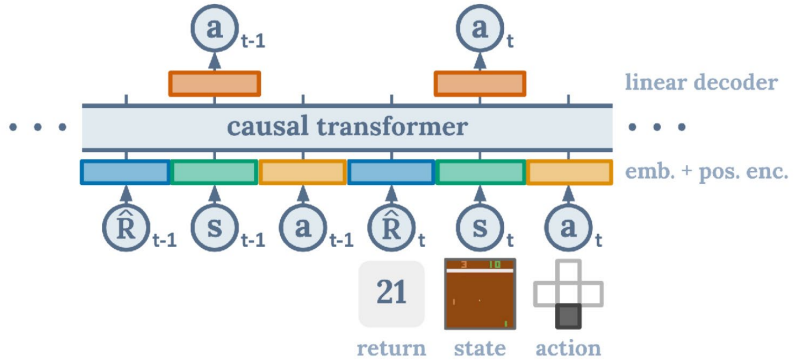
Imitate entire dataset:

$$\max_{\pi} \sum_{s,a \in \mathcal{D}} \log \pi(a|s, R_{s,a})$$

Condition policy on (empirical) return to go

$$\hat{R}_0 = \sum_{t=0}^{T-1} r_t \qquad \hat{R}_1 = \sum_{t=1}^{T-1} r_t$$

- Policy will learn to mimic good and poor behaviors (and everything in between!)
- Can use a sequence model:

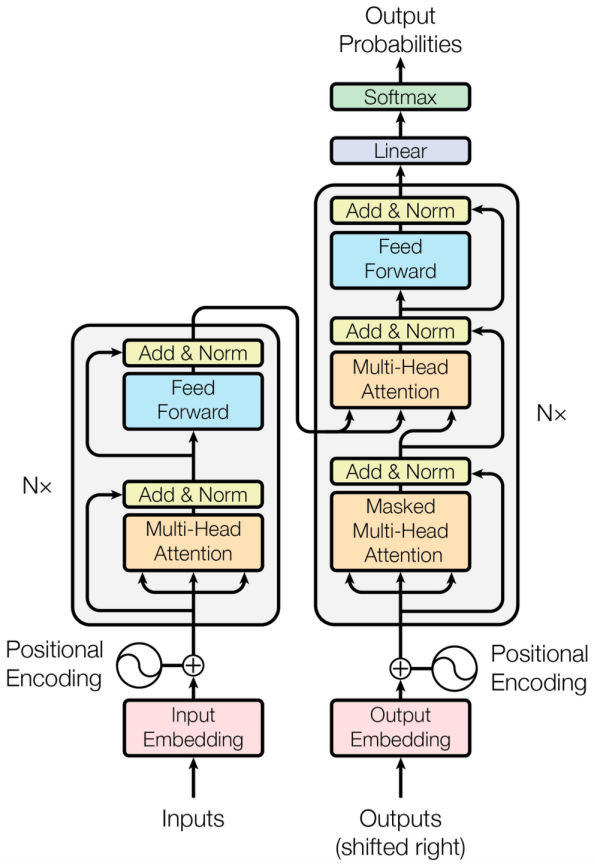


Decision Transformer: Reinforcement Learning via Sequence Modeling

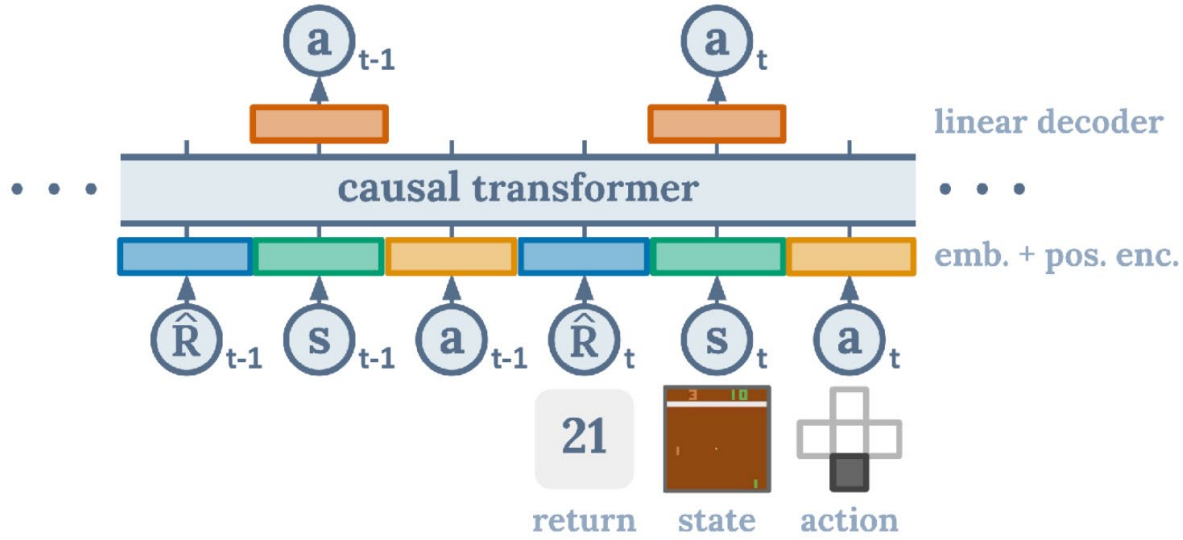
Lili Chen*,1, Kevin Lu*,1, Aravind Rajeswaran2, Kimin Lee1,
Aditya Grover2, Michael Laskin1, Pieter Abbeel1, Aravind Srinivas†,1, Igor Mordatch†,3
*equal contribution †equal advising
1UC Berkeley 2Facebook AI Research 3Google Brain
{lilichen, kzl}@berkeley.edu

- Referred to as: upside-down RL, reward-conditioned policies, decision transformers

Transformer structure



- Transformer is an architecture to efficiently model sequential data.
- It consists of stacked self-attention layers with residual connections.



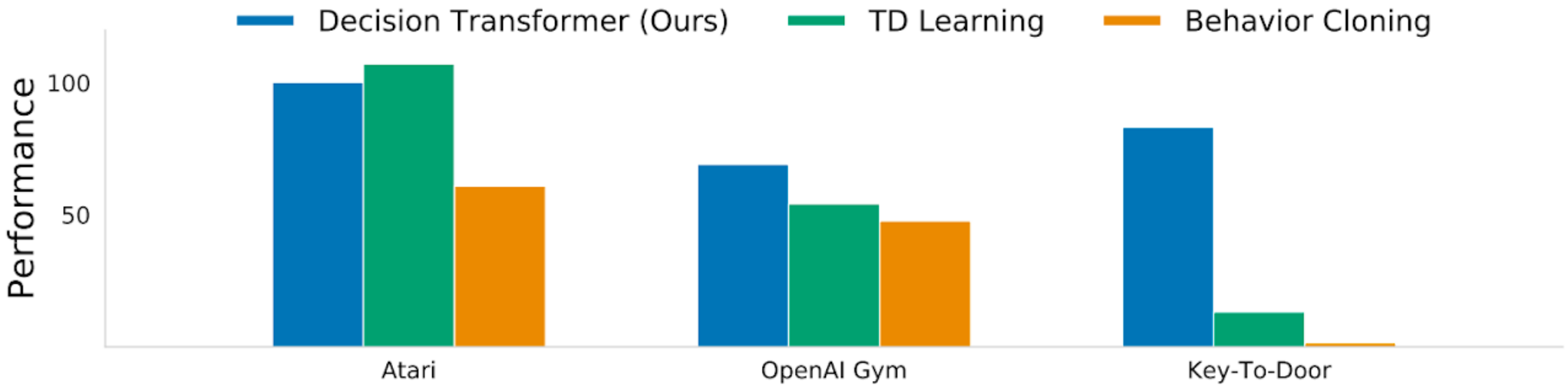
- DT uses the GPT architecture,
- It modifies the transformer architecture with a causal self attention mask to enable autoregressive generation
- replacing the summation/softmax over the n tokens with only the previous tokens in the sequence ($j \in [1, i]$).

Attention is all you need. NeurIPS'17.
Improving language understanding by generative pre-training. 2018.

Return-conditioned policies

At test stage:

1. Start at initial state s_0
2. Specify the desired target return R_0
3. $a_0 = Transformer(R_0, s_0)$
4. Execute action, observe reward and next state (r_0, s_1)
5. Decrement the target return $R_1 = R_0 - r_0$
6. $a_1 = Transformer(R_0, s_0, a_0, R_1, s_1)$
7. (repeat above process)

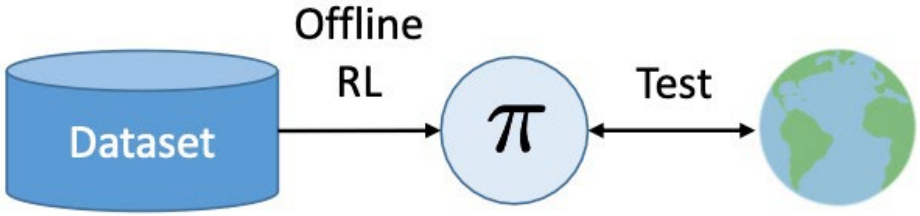


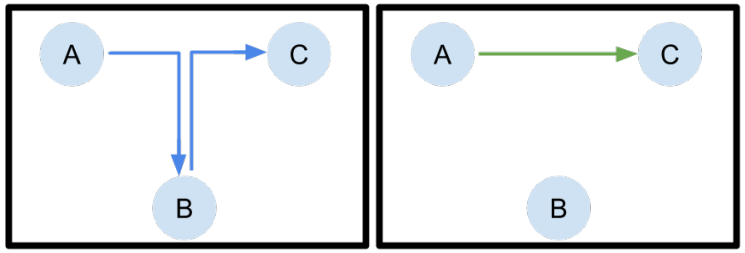
Decision transformer: reinforcement learning via sequence modeling, NIPS'21

Offline RL / Summary

Why Offline RL Should Work in Principle

- Offline RL trains a policy from a batch of pre-collected data, which makes RL closer to real applications
  - Find the good behaviour in a dataset of good and bad behaviour
  - Generalize to similar states and actions
  - Stich together good and bad behaviour

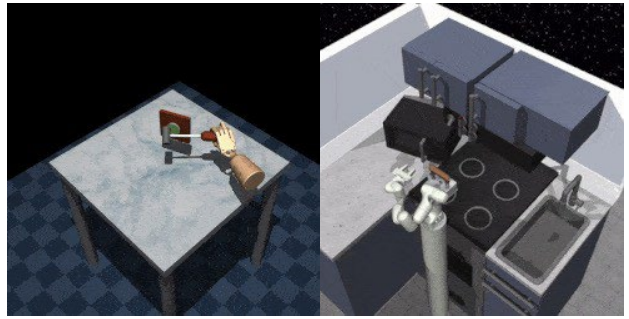




https://bair.berkeley.edu/blog/2020/06/25/D4RL/

AntMaze

Carla



Kitchen

Offline RL / Summary

But it is still challenging

- The most important problem studied in offline RL is about extrapolation (or out-of-distribution) problem
- Model-free offline RL methods build constraints on value function or policies
- Model-based offline RL methods measure the uncertainty of state transition and use it to penalize the out-of-distribution actions
- Offline RL performance is still far from perfect

Reference

- Christopher Mutschler, Offline Reinforcement Learning
- CS 224R, Imitation Learning/Offline Reinforcement Learning: Part 1/Offline Reinforcement Learning: Part 2
- Sanjiban Choudhury, Offline Reinforcement Learning
- Bolei Zhou, Imitation Learning
- Sergey Levine, Offline Deep Reinforcement Learning Algorithms