# Robot Learning

Offline Reinforcement Learning

Prof. Songhwai Oh
ECE, SNU

# Contents



Handling extrapolation error

- Off-Policy Deep Reinforcement Learning without Exploration
- Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction
- Conservative Q-learning for offline reinforcement learning
- Implicit Constraint Approach for Offline Multi-Agent Reinforcement Learning

Data augmentation

- S4RL: Surprisingly Simple Self-Supervision for Offline Reinforcement Learning

Model-based

- MOReL: Model-Based Offline Reinforcement Learning

Supervised learning

- Decision Transformer: Reinforcement Learning via Sequence Modeling
- Trajectory Transformer
- Learning to generalize across long-horizon tasks from human demonstrations

Fujimoto, Scott, David Meger, and Doina Precup. "**Off-policy deep reinforcement learning without exploration**." ICML, 2019.

# OFF-POLICY DEEP REINFORCEMENT LEARNING WITHOUT EXPLORATION

# Main Idea



- Problem
  - In batch reinforcement learning (=offline RL), off-policy algorithms (DQN, DDPG) do not learn well.
  - This is due to an extrapolation error.
  - The cause of this problem is that the distribution of data induced by the policy is different from the distribution of batch data.

- Proposed method
  - Propose batch-constrained deep Q-learning (BCQ).
  - Use a state-conditioned generative model to produce actions similar to those in the batch.
  - Can learn successfully in a setting without interaction.

# Extrapolation Error



- Extrapolation Error
  - Error caused by the mismatch between the dataset and state-action visitations of the current policy
  - The value estimate Q(s, a) is affected by the extrapolation error during the value update

- Cause of extrapolation error
  1. **Absent Data**
     - The estimate of Q(s',a') may be arbitrarily bad because there is not enough data near s', a'.
  2. **Model Bias**
     - Occurs due to the difference between the transitions of the true MDP and the transition distribution within the data.
  3. **Training Mismatch**
     - Occurs when distribution under the current policy and data distribution are different

- In case of on-policy setting, it is possible to reduce error by exploring overestimated part due to extrapolation error, but it is impossible in offline setting.

# Batch-Constrained Deep Q-learning (BCQ)



- To avoid extrapolation error, a policy should induce a similar state-action to the batch (batch-constrained).
- Batch-constrained policies are trained to select actions with respect to three objectives:
  1. Minimize the distance of selected actions to the data in the batch. (most important)
  2. Lead to states where familiar data can be observed.
  3. Maximize the value function.
- BCQ approaches the notion of batch-constrained through a generative model (produce likely actions under the batch).

# BCQ Details



- Generative model for a batch (in this paper, use VAE) : $G_\omega(s)$
- Perturbation model : $\xi_\phi(s, a, \Phi)$
  - Outputs an adjustment to an action a in the range [-Φ, Φ]
- Policy π : $\pi(s) = \underset{a_i+\xi_\phi(s,a_i,\Phi)}{\operatorname{argmax}} Q_\theta(s, a_i + \xi_\phi(s, a_i, \Phi)),$

$$\{a_i \sim G_\omega(s)\}_{i=1}^n.$$

- Perturbation model training: actions are sampled from a generative model.

$$\phi \leftarrow \underset{\phi}{\operatorname{argmax}} \sum_{(s,a)\in\mathcal{B}} Q_\theta(s, a + \xi_\phi(s, a, \Phi))$$

- To penalize high variance in uncertainty region, use clipped double Q-Learning.

$$r+\gamma \max_{a_i} \left[\lambda \min_{j=1,2} Q_{\theta'_j}(s', a_i) + (1-\lambda) \max_{j=1,2} Q_{\theta'_j}(s', a_i)\right]$$

# BCQ Algorithm



**Algorithm 1** BCQ

**Input:** Batch $\mathcal{B}$, horizon $T$, target network update rate $\tau$, mini-batch size $N$, max perturbation $\Phi$, number of sampled actions $n$, minimum weighting $\lambda$.
Initialize Q-networks $Q_{\theta_1}, Q_{\theta_2}$, perturbation network $\xi_\phi$, and VAE $G_\omega = \{E_{\omega_1}, D_{\omega_2}\}$, with random parameters $\theta_1$, $\theta_2$, $\phi$, $\omega$, 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**

Kumar, Aviral, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. "**Stabilizing off-policy Q-learning via bootstrapping error reduction**." NeurIPS, 2019.

# STABILIZING OFF-POLICY Q-LEARNING VIA BOOTSTRAPPING ERROR REDUCTION

# Proposed Method



- The proposed method, Bootstrapping Error Accumulation Reduction (BEAR), restricts the policy to ensure that it outputs actions that lie in the support of the training distribution.

- The existing methods (e.g., BCQ) implicitly constrain the distribution of the learned policy to be close to the behavior policy.

→ It is overly restrictive.

For example, if the behavior policy is close to uniform, the learned policy will behave randomly, even if the data is sufficient.

- BEAR enforces the learned policy $\pi(a|s)$ to have a positive density only where the density of the behavior policy $\beta(a|s)$ is more than a threshold.

(i.e., $\forall a, \beta(a|s) \leq \varepsilon \implies \pi(a|s) = 0$)

- Support (policy) set $\Pi$: the set of restricted policies
- Upper bound on the performance of distribution-constrained Q-iteration:

current Bellman error

$$\lim_{k\to\infty} \mathbb{E}_{\rho_0}[|V^{\pi_k}(s) - V^*(s)|] \leq \frac{\gamma}{(1-\gamma)^2}\left[C(\Pi)\mathbb{E}_{\mu}[\max_{\pi\in\Pi} \mathbb{E}_{\pi}[\delta(s,a)]] + \frac{1-\gamma}{\gamma}\alpha(\Pi)\right]$$

concentrability coefficient
: measures how far $\Pi$ is from the training data distribution

suboptimality constant
: measures how far $\pi^*$ is from $\Pi$

This bound formalizes the tradeoff between keeping policies chosen during backups close to the data and keeping the set $\Pi$ large enough to capture well-performing policies.

→ BEAR uses the following support set: $\Pi_\epsilon = \{\pi \mid \pi(a|s) = 0 \text{ whenever } \beta(a|s) < \epsilon\}$

$\beta$: the behavior policy

→ Therefore, using the support set $\Pi_\epsilon$ gives us a single lever, $\epsilon$, which simultaneously trades off the value of $C(\Pi_\epsilon)$ and $\alpha(\Pi_\epsilon)$.

# BEAR: Practical Implementation



- BEAR uses a set of Q-functions: $\hat{Q}_1, \cdots, \hat{Q}_K$
- Then, the policy is updated to maximize the conservative estimate of Q-values within $\Pi_\epsilon$.

$$\pi_\phi(s) := \max_{\pi \in \Pi_\epsilon} \mathbb{E}_{a \sim \pi(\cdot|s)} \left[ \min_{j=1,..,K} \hat{Q}_j(s,a) \right]$$

- Since the behavior policy $\beta$ is unknown, they use the maximum mean discrepancy (MMD) approximation to measure distance between two distributions using samples.

$$\mathrm{MMD}^2(\{x_1, \cdots, x_n\}, \{y_1, \cdots, y_m\}) = \frac{1}{n^2} \sum_{i,i'} k(x_i, x_{i'}) - \frac{2}{nm} \sum_{i,j} k(x_i, y_j) + \frac{1}{m^2} \sum_{j,j'} k(y_j, y_{j'})$$

k: universal kernel
In BEAR, k is Laplacian or Gaussian Kernel

- Finally, the optimization problem in the policy improvement step becomes

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

→ Equation 1

# BEAR: Algorithm

- Equation 1:

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

**Algorithm 1** BEAR Q-Learning (BEAR-QL)

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

1: Initialize Q-ensemble $\{Q_{\theta_i}\}_{i=1}^K$, actor $\pi_\phi$, Lagrange multiplier $\alpha$, target networks $\{Q_{\theta'_i}\}_{i=1}^K$, and a target actor $\pi_{\phi'}$, with $\phi' \leftarrow \phi, \theta'_i \leftarrow \theta_i$
2: **for** $t$ in {1, . . . , N} **do**
3: Sample mini-batch of transitions $(s, a, r, s') \sim \mathcal{D}$
**Q-update:**
4: Sample $p$ action samples, $\{a_i \sim \pi_{\phi'}(\cdot|s')\}_{i=1}^p$
5: Define $y(s,a) := \max_{a_i}[\lambda \min_{j=1,..,K} Q_{\theta'_j}(s', a_i) + (1-\lambda) \max_{j=1,..,K} Q_{\theta'_j}(s', a_i)]$
6: $\forall i, \theta_i \leftarrow \arg\min_{\theta_i}(Q_{\theta_i}(s,a) - (r + \gamma y(s,a)))^2$
**Policy-update:**
7: 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)
8: Update $\phi$, $\alpha$ by minimizing Equation 1 by using dual gradient descent with Lagrange multiplier $\alpha$
9: **Update Target Networks:** $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i; \phi' \leftarrow \tau\phi + (1-\tau)\phi'$
10: **end for**

Kumar, Aviral, Aurick Zhou, George Tucker, and Sergey Levine. "**Conservative Q-learning for offline reinforcement learning**." NeurIPS 2020.

# CONSERVATIVE Q-LEARNING FOR OFFLINE REINFORCEMENT LEARNING

# Introduction



- Offline RL often fails due to *extrapolation errors*
  - Overestimate Q-value on actions not appearing in dataset

- In order to prevent overestimation, conservative q-learning (CQL) penalizes the value of out-of-distribution state-action pairs.

- By doing so, the agent's policy tends to follow the actions explored by the behavior policy.

# Conservative Off-Policy Evaluation



- Learning the lower-bound of Q-function by minimizing the expected Q-value under a particular distribution,

$$\mu(s,a) = d^{\pi_\beta}(s)\mu(a|s),$$

$\pi_\beta$: behavior policy

$$\hat{Q}^{k+1} \leftarrow \arg\min_Q \alpha\, \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}\left[Q(\mathbf{s},\mathbf{a})\right] + \frac{1}{2}\,\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}\left[\left(Q(\mathbf{s},\mathbf{a}) - \hat{\mathcal{B}}^\pi \hat{Q}^k(\mathbf{s},\mathbf{a})\right)^2\right]$$

Penalizing term

- When $\mu(a|s) = \pi_\beta(a|s)$, the resulting Q-function $\lim_{k\to\infty} \hat{Q}_k(s,a)$ lower-bounds $Q^\pi$ at all $s \in D, a \in A$.

**Theorem 3.1.** *For any $\mu(\mathbf{a}|\mathbf{s})$ with $\operatorname{supp}\mu \subset \operatorname{supp}\hat{\pi}_\beta$, with probability $\geq 1-\delta$, $\hat{Q}^\pi$ (the Q-function obtained by iterating Equation 1) satisifies:*

$$\forall \mathbf{s}\in\mathcal{D},\mathbf{a},\ \ \hat{Q}^\pi(s,a) \leq Q^\pi(\mathbf{s},\mathbf{a}) - \alpha\left[(I-\gamma P^\pi)^{-1}\frac{\mu}{\hat{\pi}_\beta}\right](\mathbf{s},\mathbf{a}) + \left[(I-\gamma P^\pi)^{-1}\frac{C_{r,T,\delta}R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}|}}\right](\mathbf{s},\mathbf{a}).$$

*Thus, if $\alpha$ is sufficiently large, then $\hat{Q}^\pi(\mathbf{s},\mathbf{a}) \leq Q^\pi(\mathbf{s},\mathbf{a}), \forall \mathbf{s}\in\mathcal{D},\mathbf{a}$. When $\hat{\mathcal{B}}^\pi = \mathcal{B}^\pi$, any $\alpha > 0$ guarantees $\hat{Q}^\pi(\mathbf{s},\mathbf{a}) \leq Q^\pi(\mathbf{s},\mathbf{a}), \forall \mathbf{s}\in\mathcal{D},\mathbf{a}\in\mathcal{A}$.*

# Conservative Off-Policy Evaluation

- To make $\hat{Q}^{\pi}$ be a tighter lower-bound, we introduce an additional Q-value maximization term, under the data distribution.

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \; \alpha \cdot \left(\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}\left[Q(\mathbf{s},\mathbf{a})\right] - \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\hat{\pi}_{\beta}(\mathbf{a}|\mathbf{s})}\left[Q(\mathbf{s},\mathbf{a})\right]\right) + \frac{1}{2}\,\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[\left(Q(\mathbf{s},\mathbf{a}) - \hat{\mathcal{B}}^{\pi}\hat{Q}^{k}(\mathbf{s},\mathbf{a})\right)^{2}\right]. \quad (2)$$

- Then, $\lim_{k\to\infty} \hat{Q}_k\,(s,a)$ is not pointwise lower-bound of $Q^{\pi}$, however, $\hat{V}^{\pi}(s)$ becomes a lower-bound of $V^{\pi}(s)$

**Theorem 3.2** (Equation 2 results in a tighter lower bound). *The value of the policy under the Q-function from Equation 2, $\hat{V}^{\pi}(\mathbf{s}) = \mathbb{E}_{\pi(\mathbf{a}|\mathbf{s})}[\hat{Q}^{\pi}(\mathbf{s},\mathbf{a})]$, lower-bounds the true value of the policy obtained via exact policy evaluation, $V^{\pi}(\mathbf{s}) = \mathbb{E}_{\pi(\mathbf{a}|\mathbf{s})}[Q^{\pi}(\mathbf{s},\mathbf{a})]$, when $\mu = \pi$, according to:*

$$\forall \mathbf{s}\in\mathcal{D},\;\; \hat{V}^{\pi}(\mathbf{s}) \leq V^{\pi}(\mathbf{s}) - \alpha\left[(I-\gamma P^{\pi})^{-1}\,\mathbb{E}_{\pi}\left[\frac{\pi}{\hat{\pi}_{\beta}} - 1\right]\right](\mathbf{s}) + \left[(I-\gamma P^{\pi})^{-1}\frac{C_{r,T,\delta}R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}|}}\right](\mathbf{s}).$$

*Thus, if* $\alpha > \frac{C_{r,T}R_{\max}}{1-\gamma}\cdot\max_{\mathbf{s}\in\mathcal{D}}\frac{1}{\sqrt{|\mathcal{D}(\mathbf{s})|}}\cdot\left[\sum_{\mathbf{a}}\pi(\mathbf{a}|\mathbf{s})\left(\frac{\pi(\mathbf{a}|\mathbf{s})}{\hat{\pi}_{\beta}(\mathbf{a}|\mathbf{s})}-1\right)\right]^{-1}$, $\forall \mathbf{s}\in\mathcal{D}$, $\hat{V}^{\pi}(\mathbf{s}) \leq V^{\pi}(\mathbf{s})$, *with probability* $\geq 1-\delta$. *When* $\hat{\mathcal{B}}^{\pi} = \mathcal{B}^{\pi}$, *then any* $\alpha > 0$ *guarantees* $\hat{V}^{\pi}(\mathbf{s}) \leq V^{\pi}(\mathbf{s}), \forall \mathbf{s}\in\mathcal{D}$.

# Conservative Q-Learning for Offline RL

- Now we can define CQL problem as,

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

  where $R(\mu)$ is a policy regularizer.

- For example, for maximum entropy regularization, $R(\mu) = -\mathrm{D}_{KL}\big(\mu|\mathrm{Unif}(a)\big)$, CQL becomes

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

Yang, Yiqin, Xiaoteng Ma, Li Chenghao, Zewu Zheng, Qiyuan Zhang, Gao Huang, Jun Yang, and Qianchuan Zhao. "**Believe what you see: Implicit constraint approach for offline multi-agent reinforcement learning**." NeurIPS 2021.

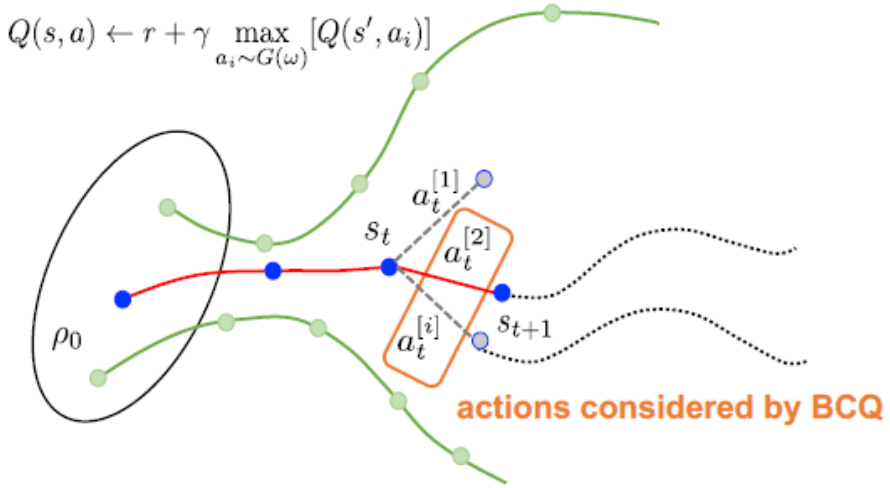
# BELIEVE WHAT YOU SEE: IMPLICIT CONSTRAINT APPROACH FOR OFFLINE MULTI-AGENT REINFORCEMENT LEARNING

# Introduction

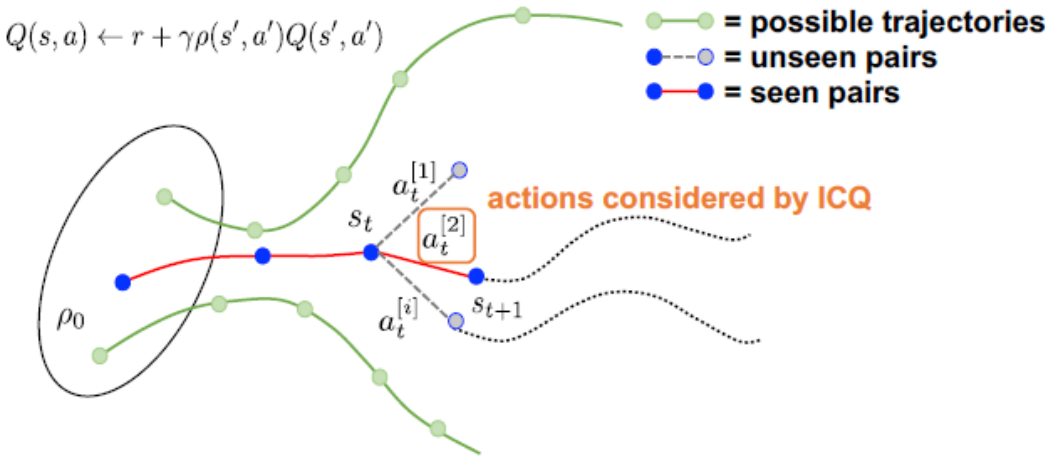
- *Extrapolation error* is the greatest obstacle of offline RL.

$$Q(s,a) \leftarrow r + \gamma \mathrm{E}_{a' \sim \pi}[Q(s',a')]$$

Can be inaccurate for unseen $a'$




(a) BCQ.

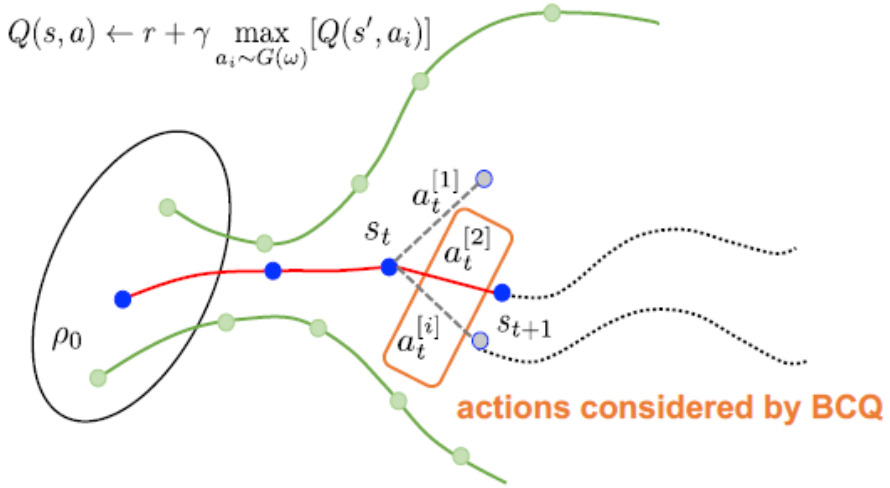



(b) ICQ.

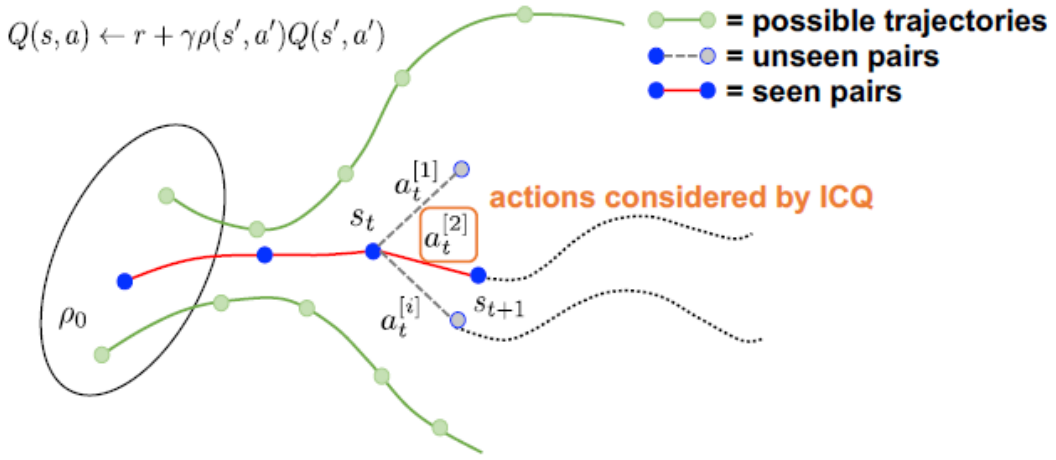
- Implicit constraint Q-learning (ICQ) aims to estimate Q-value using only the *seen* state-action pairs.

# Implicit Constraint Approach for Offline RL






(a) BCQ.




(b) ICQ.

- Implicit constraint Q-learning (ICQ) utilizes only the seen state-action pairs using *importance sampling*.

$$(\mathcal{T}^{\pi} Q)(\tau, a) = Q(\tau, a) + \mathbb{E}_{\tau'}[r + \gamma \mathbb{E}_{a' \sim \mu}[\rho(\tau', a') Q(\tau', a')] - Q(\tau, a)],$$

Where, $\rho(\tau', a') = \frac{\pi(a'|\tau')}{\mu(a'|\tau')}$ ($\mu$: behavior policy used for collecting data)

# Implicit Constraint Q-learning



- How to estimate $\rho(\tau, a)$

- Policy optimization with the behavior regularized constraint.

$$\pi_{k+1} = \arg\max \mathbb{E}_{a\sim\pi(\cdot|\tau)}[Q^{\pi_k}(\tau, a)], \quad \text{s.t.} \quad D_{\mathrm{KL}}(\pi \,\|\, \mu)[\tau] \leq \epsilon.$$

- Then the optimal policy $\pi^*$ becomes,

$$\pi^*_{k+1}(a \mid \tau) = \frac{1}{Z(\tau)}\mu(a \mid \tau)\exp\left(\frac{Q^{\pi_k}(\tau, a)}{\alpha}\right)$$

- And we can estimate $\rho(\tau, a)$ as,

$$\rho(\tau, a) = \frac{\pi^*_{k+1}(a \mid \tau)}{\mu(a \mid \tau)} = \frac{1}{Z(\tau)}\exp\left(\frac{Q^{\pi_k}(\tau, a)}{\alpha}\right)$$

- Implicit Constraint Q-learning operator:

$$\mathcal{T}_{\mathrm{ICQ}}Q(\tau, a) = r + \gamma\mathbb{E}_{a'\sim\mu}\left[\frac{1}{Z(\tau')}\exp\left(\frac{Q(\tau', a')}{\alpha}\right)Q(\tau', a')\right]$$

# Extending ICQ to Multi-Agent Tasks



- CTDE paradigm : Centralized Training and Decentralized Execution

- Value decomposition assumption:
  - Global q-value is expressed as a linear combination of the q-values of each agent.

$$Q^{\pi}(\boldsymbol{\tau}, \boldsymbol{a}) = \sum_i w^i(\boldsymbol{\tau}) Q^i(\tau^i, a^i) + b(\boldsymbol{\tau}),$$

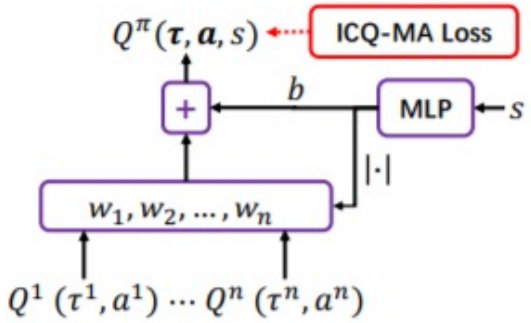



Figure 3: Mixer Network.

- Then, the joint policy can be expressed as a combination of the optimal policies of the all agent (for i=1,2,…,n)

**Theorem 3.** *Assuming the joint action-value function is linearly decomposed, we can decompose the multi-agent joint-policy under implicit constraint as follows*

$$\boldsymbol{\pi} = \arg\max_{\pi^1,\ldots,\pi^n} \sum_i \mathbb{E}_{\tau^i, a^i \sim \mathcal{B}} \left[ \frac{1}{Z^i(\tau^i)} \log(\pi^i(a^i \mid \tau^i)) \exp\left( \frac{w^i(\boldsymbol{\tau}) Q^i(\tau^i, a^i)}{\alpha} \right) \right], \qquad (51)$$

*where* $Z^i(\tau^i) = \sum_{\tilde{a}^i} \mu^i(\tilde{a}^i \mid \tau^i) \exp\left(\frac{1}{\alpha} w^i(\boldsymbol{\tau}) Q^i(\tau^i, \tilde{a}^i)\right)$ *is the normalizing partition function.*

# Extending ICQ to Multi-Agent Tasks



- Q-function objective:

$$\mathcal{J}_Q(\phi,\psi)=\mathbb{E}_{\mathcal{B}}\left[\sum_{t\geq 0}(\gamma\lambda)^t\left(r_t+\gamma\frac{1}{Z(\boldsymbol{\tau}_{t+1})}\exp\left(\frac{Q(\boldsymbol{\tau}_{t+1},\boldsymbol{a}_{t+1})}{\alpha}\right)Q(\boldsymbol{\tau}_{t+1},\boldsymbol{a}_{t+1})-Q(\boldsymbol{\tau}_t,\boldsymbol{a}_t)\right)\right]^2 \tag{19}$$

where $Q(\boldsymbol{\tau}_{t+1},\boldsymbol{a}_{t+1})=\sum_i w^i(\boldsymbol{\tau}_{t+1};\psi')Q^i(\tau^i_{t+1},a^i_{t+1};\phi'_i)-b(\boldsymbol{\tau}_{t+1};\psi')$.

- Policy gradient:

$$\mathcal{J}_{\boldsymbol{\pi}}(\theta)=\sum_i \mathbb{E}_{\tau^i,a^i\sim\mathcal{B}}\left[-\frac{1}{Z^i(\tau^i)}\log(\pi^i(a^i\mid\tau^i;\theta_i))\exp\left(\frac{w^i(\boldsymbol{\tau})Q^i(\tau^i,a^i)}{\alpha}\right)\right]$$

- Multi-agent value estimation with $\lambda$-return

$$(\mathcal{T}^{\lambda}_{\mathrm{ICQ}}Q)(\boldsymbol{\tau},\boldsymbol{a})\triangleq Q(\boldsymbol{\tau},\boldsymbol{a})+\mathbb{E}_{\mu}\left[\sum_{t\geq 0}(\gamma\lambda)^t\left(r_t+\gamma\rho(\boldsymbol{\tau}_{t+1},\boldsymbol{a}_{t+1})Q(\boldsymbol{\tau}_{t+1},\boldsymbol{a}_{t+1})-Q(\boldsymbol{\tau}_t,\boldsymbol{a}_t)\right)\right]$$

Sinha, Samarth, Ajay Mandlekar, and Animesh Garg. "**S4RL: Surprisingly simple self-supervision for offline reinforcement learning in robotics**." Conference on Robot Learning (CoRL), 2021.

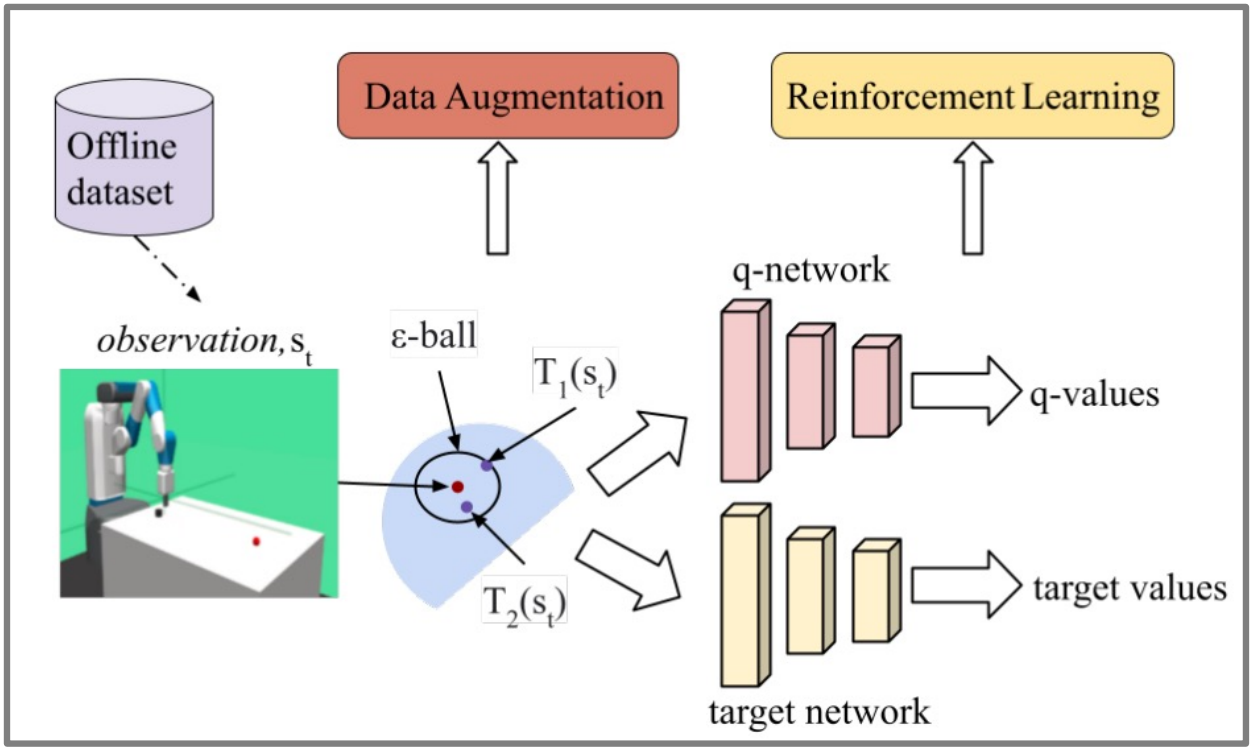
# S4RL: SURPRISINGLY SIMPLE SELF-SUPERVISION FOR OFFLINE REINFORCEMENT LEARNING IN ROBOTICS

# Main Problem



- Existing offline RL methods mainly focus on reducing Q-value estimation errors due to out-of-distribution from offline data.
- However, due to the use of parameterized neural networks, approximation errors are also introduced.
- In this paper, a data augmentation method for offline RL is proposed to reduce the approximation errors
- "Existing offline RL methods + proposed data augmentation" outperforms baselines in the D4RL experiments.

# Proposed Method




**Overview of the proposed method**

- A data augmentation transformation is denoted by: $\mathcal{T}(\tilde{s}_t|s_t)$ where $s_t \sim \mathcal{D}$.
- The authors propose seven different data augmentation transformations and compare the results of each transformation method.

# Proposed Method



- Proposed Q-learning objective:

$$\min_{Q} \mathop{\mathbb{E}}_{s_t, a_t \sim \mathcal{D}} \left[ r_t + \gamma \frac{1}{i} \sum_i Q(\mathcal{T}_i(\tilde{s}_{t+1} | s_{t+1}), a_{t+1}) - \frac{1}{i} \sum_i Q(\mathcal{T}_i(\tilde{s}_t | s_t), a_t) \right]$$

  - Q values and target values are used as the average of each augmented state value.
  - In the experiments, however, use only one augmentation transformation.

- Proposed candidates for augmentation transformation:

1. **zero-mean Gaussian noise** $\tilde{s}_t \leftarrow s_t + \epsilon$, where $\epsilon \in \mathcal{N}(0, \sigma I)$
2. **zero-mean Uniform noise** $\tilde{s}_t \leftarrow s_t + \epsilon$, where $\epsilon \in \mathcal{U}(-\alpha, \alpha)$
3. **random amplitude scaling** $\tilde{s}_t \leftarrow s_t * \epsilon$, where $\epsilon \in \mathcal{U}(\alpha, \beta)$
4. **dimension dropout** $\tilde{s}_t \leftarrow s_t \cdot \mathbf{1}$,
   where $\mathbf{1}$ is a vector of 1s with one 0 randomly sampled from a Bernoulli
5. **state-switch** where we flip the value of 2 randomly selected dimensions
6. **state mix-up** $\tilde{s}_t \leftarrow \lambda s_t + (1 - \lambda) s_{t+1}$, where $\lambda \sim \texttt{Beta}(\alpha, \alpha)$
7. **adversarial state training** $\tilde{s}_t \leftarrow s_t + \epsilon \nabla_{s_t} \mathbb{J}_Q(\pi(s_t))$

# Experiment: baselines

RLLAB
http://rllab.snu.ac.kr

- Existing offline RL methods:
  - Conservative Q-learning (CQL)
  - Behavior regularized actor-critic (BRAC)
- Proposed methods: “[offline RL] + S4RL (transformation method)”

  ( “CQL+S4RL (Adv)”: update CQL with the adversarial state training transformation)
- Baseline augmentation (self-supervision) methods:
  - Contrastive unsupervised reinforcement learning (CURL): use another data augmentation transformation.
  - Variational auto-encoder (VAE): feed the augmented states to the auto-encoder to learn generalizable representations.
- Offline Dataset:
  - Use D4RL dataset.

  (“cheetah-medium”: environment – cheetah, data – collect using a medium-level policy)

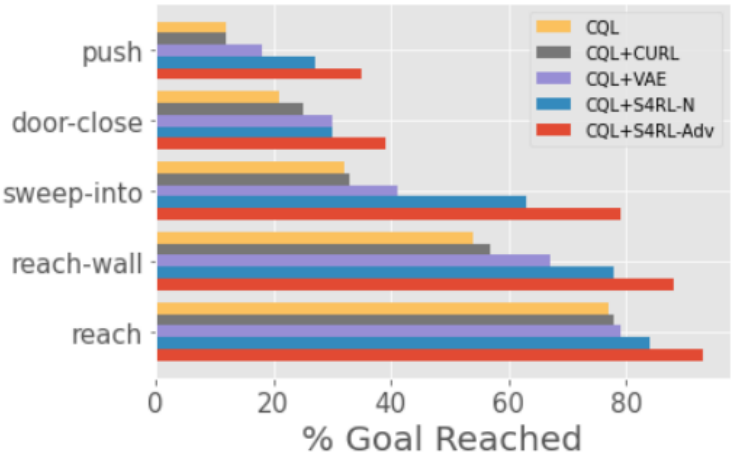
# Experiment: results

RLLAB
http://rllab.snu.ac.kr

| Task Name | Normal | CQL +S4RL ($\mathcal{N}$) | CQL +S4RL ($\mathcal{U}$) | CQL +S4RL (Amp-Scale) | CQL +S4RL (Dim-Drop) | CQL +S4RL (State-Switch) | CQL +S4RL (MixUp) | CQL +S4RL (Adv) |
|---|---|---|---|---|---|---|---|---|
| cheetah-random | 35.4 | **52.3** | **50.3** | 46.4 | 45.5 | 40.3 | 45.2 | **53.9** |
| cheetah-medium | 44.4 | **48.8** | **47.0** | 42.5 | 45.6 | 41.2 | 46.2 | **48.6** |
| cheetah-medium-replay | 42.0 | **51.4** | **50.8** | 43.1 | 49.6 | 41.0 | 46.2 | **51.7** |
| cheetah-medium-expert | 62.4 | **79.0** | **78.5** | 71.2 | 66.1 | 68.3 | 73.1 | **78.1** |
| hopper-random | **10.8** | **10.8** | 10.7 | **10.8** | 9.3 | 9.5 | **11.0** | 10.7 |
| hopper-medium | 58.0 | 78.9 | **80.6** | 60.3 | 45.3 | 54.6 | **79.2** | **81.3** |
| hopper-medium-replay | 29.5 | **35.4** | 35.2 | 28.6 | 20.4 | 20.9 | **35.6** | **36.8** |
| hopper-medium-expert | 111.0 | **115.2** | 112.3 | 102.6 | 98.3 | 108.6 | **113.5** | **117.9** |
| walker-random | 7.0 | **24.9** | **20.5** | 6.9 | 3.2 | 9.9 | 10.2 | **25.1** |
| walker-medium | 79.2 | **93.6** | **93.9** | 81.6 | 65.3 | 65.3 | 88.6 | **93.1** |
| walker-medium-replay | 21.1 | **30.3** | **31.9** | 25.1 | 8.5 | 9.3 | 27.6 | **35.0** |
| walker-medium-expert | 98.7 | **112.2** | 104.1 | 100.2 | 86.8 | 103.8 | **104.3** | **107.1** |
| **average-score** | 49.96 | **60.57** | **59.65** | 51.60 | 45.31 | 47.73 | 56.73 | **61.6** |
| **average-ranking** | 5.83 | **1.92** | **3.00** | 5.00 | 6.42 | 6.25 | 3.17 | **1.83** |

→ show that adversarial state is the best transformation method.

| Domain | Task Name | CQL Normal | CQL +CURL ($\mathcal{N}$) | CQL +VAE ($\mathcal{N}$) | CQL +S4RL ($\mathcal{N}$) | CQL +S4RL (Mix-Up) | CQL +S4RL (Adv) | BRAC-v Normal | BRAC-v +S4RL ($\mathcal{N}$) | BRAC-v +S4RL (Mix-Up) | BRAC-v +S4RL (Adv) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| AntMaze | antmaze-umaze | 74.0 | 70.3 | 86.1 | **91.3** | 86.7 | **94.1** | 70.0 | 76.5 | 66.4 | **80.3** |
| | antmaze-umaze-diverse | 84.0 | 81.4 | 85.1 | **87.8** | **86.9** | **88.0** | 70.0 | **81.3** | 65.4 | **80.9** |
| | antmaze-medium-play | **61.2** | **60.9** | **62.1** | **61.9** | **61.3** | **61.6** | 0.0 | 0.0 | 0.0 | 0.0 |
| | antmaze-medium-diverse | 53.7 | 45.4 | 59.9 | 78.1 | 62.3 | **82.3** | 0.0 | 0.0 | 0.0 | 0.0 |
| | antmaze-large-play | 15.8 | 12.3 | 15.2 | **24.4** | 16.7 | **25.1** | 0.0 | 0.0 | 0.0 | 0.0 |
| | antmaze-large-diverse | 14.9 | 8.3 | 17.3 | **27.0** | 15.9 | **26.2** | 0.0 | 0.0 | 0.0 | 0.0 |
| Gym | cheetah-random | 35.4 | 43.1 | 35.7 | **52.3** | 45.2 | **53.9** | 31.2 | **35.6** | 34.3 | **36.1** |
| | cheetah-medium | 44.4 | 44.8 | 45.6 | **48.8** | 46.2 | 48.6 | 46.3 | **49.1** | 46.1 | 46.0 |
| | cheetah-medium-replay | 42.0 | 36.5 | 41.9 | 51.4 | 46.2 | **51.7** | **47.7** | **47.9** | **47.0** | **47.9** |
| | cheetah-medium-expert | 62.4 | 65.6 | 70.7 | **79.0** | 73.1 | 78.1 | 41.9 | **52.1** | 46.7 | **53.4** |
| | hopper-random | **10.8** | **10.8** | **10.8** | **10.8** | **11.0** | 10.7 | **12.2** | **12.9** | 10.9 | **12.1** |
| | hopper-medium | 58.0 | 61.9 | 66.4 | **78.9** | **79.2** | **81.3** | 31.1 | 45.1 | 41.9 | **48.0** |
| | hopper-medium-replay | 29.5 | 30.1 | **34.8** | **35.4** | **35.6** | **36.8** | **0.6** | **1.1** | **0.9** | **0.4** |
| | hopper-medium-expert | **111.0** | **109.7** | **114.3** | **113.5** | **112.3** | 117.9 | 0.8 | **59.0** | 49.5 | 55.3 |
| | walker-random | 7.0 | 6.6 | 6.5 | **24.9** | 10.2 | **25.1** | 1.9 | 9.5 | 2.4 | **14.1** |
| | walker-medium | 79.2 | 81.3 | 80.8 | **93.6** | 88.6 | **93.1** | 81.1 | **85.4** | 80.9 | **86.7** |
| | walker-medium-replay | 21.1 | 24.5 | 26.4 | 30.3 | 27.6 | **35.0** | **0.9** | **1.1** | **0.9** | **1.1** |
| | walker-medium-expert | 98.7 | 103.0 | 99.6 | **112.2** | 104.3 | 107.1 | 81.6 | 90.4 | 89.7 | **94.5** |
| Adroit | pen-human | 37.5 | 33.4 | 39.7 | **44.4** | 41.6 | **51.2** | 0.6 | **11.3** | 0.8 | **14.1** |
| | pen-cloned | 39.2 | 40.1 | 41.3 | **57.1** | 44.5 | **58.2** | -2.5 | **10.6** | 0.5 | **9.9** |
| | hammer-human | 4.4 | **6.1** | **6.0** | **5.9** | **6.3** | **6.3** | 0.2 | **3.2** | **3.3** | **5.4** |
| | hammer-cloned | 2.1 | 1.9 | 2.1 | **2.7** | **2.4** | **2.9** | 0.3 | 0.1 | 0.3 | **1.2** |
| | door-human | 9.9 | 10.3 | 14.3 | 27.0 | 16.7 | **35.3** | -0.3 | **2.8** | 0.2 | **4.2** |
| | door-cloned | 0.4 | 0.4 | 0.7 | **2.1** | 0.3 | 0.8 | -0.1 | **0.3** | -0.1 | **0.5** |
| | relocate-human | **0.2** | **0.2** | **0.2** | **0.2** | **0.2** | **0.2** | -0.3 | **0.1** | **0.1** | **0.2** |
| | relocate-cloned | **-0.1** | **-0.1** | -0.3 | **-0.1** | **-0.1** | **-0.1** | -0.3 | **-0.1** | **-0.1** | **-0.1** |
| Franka | kitchen-complete | 43.8 | 54.4 | 42.1 | 77.1 | 60.4 | **88.1** | 0.0 | 0.0 | 0.0 | **12.3** |
| | kitchen-partial | 49.8 | 59.6 | 49.9 | 74.8 | 59.0 | **83.6** | 0.0 | 0.0 | 0.0 | **13.9** |




→ CQL+S4RL-Adv outperforms all other baselines.

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. "**MOReL: Model-based offline reinforcement learning**." NeurIPS 2020.

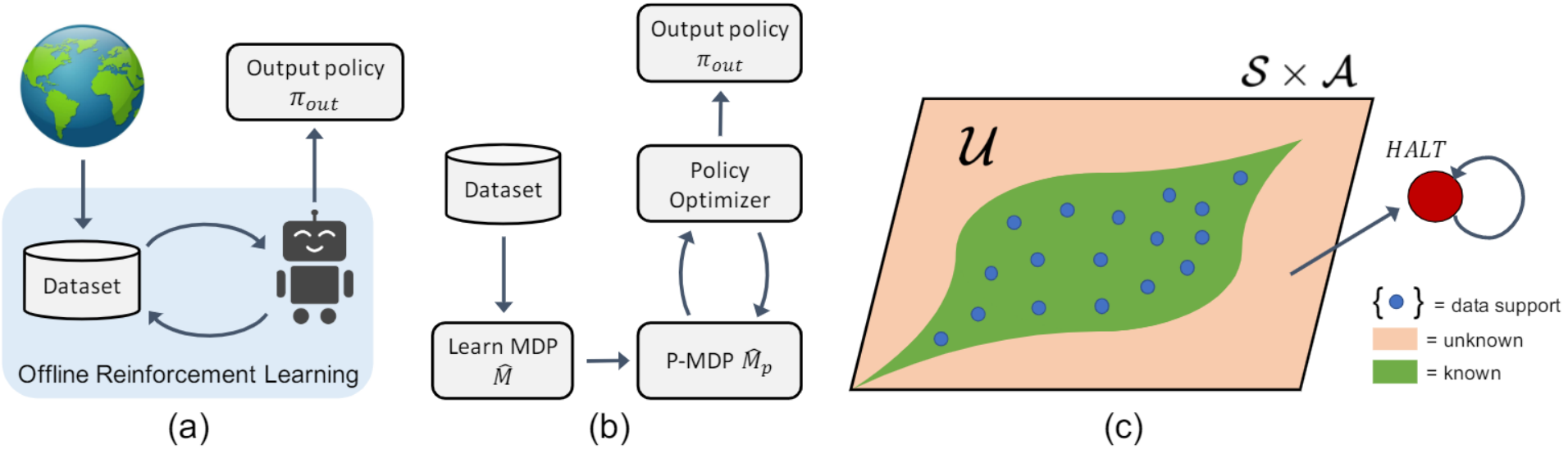
# MOReL: Model-based offline reinforcement learning

# Introduction



- Prior works in offline RL have been confined to model-free RL approaches.
- Model-based RL (MBRL) algorithms are highly sample efficient, but the learned model is unlikely to be globally accurate due to lack of offline data.
- In this paper, a pessimistic MDP (P-MDP) is proposed, where a policy is penalized when entering “unknown” regions.
- Theoretically, a policy trained in P-MDP has the upper bound for the sub-optimality.
- The proposed method, MOReL, obtains SOTA results in 12 out of 20 environments.

# Proposed Method




**Overview of the proposed method** ((a), (b): offline RL, (c): P-MDP)

## Definition 1

- Learned MDP: $\hat{\mathcal{M}} = \{S, A, r, \hat{P}, \hat{\rho}_0, \gamma\}$
- Unknown state-action detector (USAD):

$$U^{\alpha}(s,a) = \begin{cases} \textit{FALSE (i.e. Known)} & \text{if } D_{TV}\left(\hat{P}(\cdot|s,a), P(\cdot|s,a)\right) \leq \alpha \textit{ can be guaranteed} \\ \textit{TRUE (i.e. Unknown)} & \textit{otherwise} \end{cases}$$

# Proposed Method

## Definition 2

- Pessimistic-MDP (P-MDP): $\hat{\mathcal{M}}_p := \{S \cup HALT, A, r_p, \hat{P}_p, \hat{\rho}_0, \gamma\}$

$$\hat{P}_p(s'|s,a) = \begin{cases} \delta(s' = \text{HALT}) & \text{if } U^\alpha(s,a) = \text{TRUE} \\ & \text{or } \ s = \text{HALT} \\ \hat{P}(s'|s,a) & \textit{otherwise} \end{cases} \qquad r_p(s,a) = \begin{cases} -\kappa & \text{if } s = \text{HALT} \\ r(s,a) & \textit{otherwise} \end{cases}$$

## Implementation

- Dynamic model learning:

$\hat{P}(\cdot|s,a) \equiv \mathcal{N}\left(f_\phi(s,a), \Sigma\right)$, with $f_\phi(s,a) = s + \sigma_\Delta \ \text{MLP}_\phi\left((s-\mu_s)/\sigma_s, (a-\mu_a)/\sigma_a\right)$,
$\mu_s, \sigma_s, \mu_a, \sigma_a$ are the mean and standard deviations of states/actions in $\mathcal{D}$,
$\sigma_\Delta$ is the standard deviation of state differences, i.e. $\Delta = s' - s, (s,s') \in \mathcal{D}$.

- Unknown state-action detector (USAD):

Train an ensemble dynamic model and define the disc function as:

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

Then $$U_{\text{practical}}(s,a) = \begin{cases} \text{FALSE (i.e. Known)} & \text{if } \text{disc}(s,a) \le \text{threshold} \\ \text{TRUE (i.e. Unknown)} & \text{if } \text{disc}(s,a) > \text{threshold} \end{cases}$$

- Policy learning: use natural policy gradient (NPG)

## Definition 3

- Hitting time $T^{\pi}_{(s,a)}$ : the first time action a is taken at state s by $\pi$ .

$$\rightarrow \; T^{\pi}_{\mathcal{S}} \overset{def}{=} \min_{(s,a)\in\mathcal{S}} T^{\pi}_{(s,a)}$$

**Theorem 1.** *(Policy value with pessimism) The value of any policy $\pi$ on the original MDP $\mathcal{M}$ and its $(\alpha, R_{\max})$-pessimistic MDP $\hat{\mathcal{M}}_p$ satisfies:*

$$J_{\hat{\rho}_0}(\pi, \hat{\mathcal{M}}_p) \geq J_{\rho_0}(\pi, \mathcal{M}) - \frac{2R_{max}}{1-\gamma} \cdot D_{TV}(\rho_0, \hat{\rho}_0) - \frac{2\gamma R_{max}}{(1-\gamma)^2} \cdot \alpha - \frac{2R_{max}}{1-\gamma} \cdot \mathbb{E}\left[\gamma^{T^{\pi}_{\mathcal{U}}}\right], \text{ and}$$

$$J_{\hat{\rho}_0}(\pi, \hat{\mathcal{M}}_p) \leq J_{\rho_0}(\pi, \mathcal{M}) + \frac{2R_{max}}{1-\gamma} \cdot D_{TV}(\rho_0, \hat{\rho}_0) + \frac{2\gamma R_{max}}{(1-\gamma)^2} \cdot \alpha,$$

*where $T^{\pi}_{\mathcal{U}}$ denotes the hitting time of unknown states $\mathcal{U} \overset{def}{=} \{(s,a) : U^{\alpha}(s,a) = TRUE\}$ by $\pi$ on $\mathcal{M}$.*

**Corollary 3.** *(Upper bound) Suppose the dataset $\mathcal{D}$ is large enough so that $\alpha = D_{TV}(\rho_0, \hat{\rho}_0) = 0$. Then, the output $\pi_{out}$ of Algorithm 1 satisfies:*

$$J_{\rho_0}(\pi^*, \mathcal{M}) - J_{\rho_0}(\pi_{out}, \mathcal{M}) \leq \epsilon_{\pi} + \frac{2R_{max}}{1-\gamma} \cdot \mathbb{E}\left[\gamma^{T^{\pi^*}_{\mathcal{U}}}\right] \leq \epsilon_{\pi} + \frac{2R_{\max}}{(1-\gamma)^2} \cdot d^{\pi^*, \mathcal{M}}(\mathcal{U})$$

→ give an upper bound of the sub-optimality gap between the optimal policy in the true MDP and P-MDP.

# Experiment: environmental settings



- Environments:
  - Hopper, Half-Cheetah, Walker-2d, and Ant in MuJoCo.

- Dataset:
  - First train a policy $\pi_p$ to obtain reward sums 1000, 4000, 1000, and 1000 for the four environments.
  - Pure: collecting data using $\pi_p$ .
  - $\texttt{eps-1}, \texttt{eps-3}, \texttt{gauss-1}, \texttt{gauss-3}$ : collecting data using $\pi_p$ with noise.

- Baselines:
  - Batch-constrained Q learning (BCQ)
  - Bootstrapping error accumulation reduction (BEAR)
  - Behaviour regularized actor-critic (BRAC)

# Experiment: results



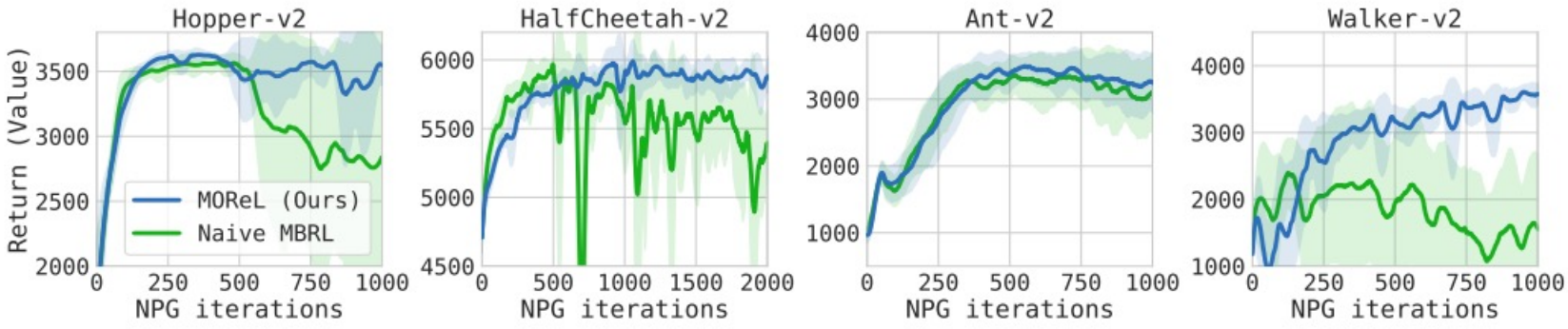
| Environment: Ant-v2 | | | | | |
|---|---|---|---|---|---|
| Algorithm | BCQ [15] | BEAR [16] | BRAC [18] | Best Baseline | MOReL (Ours) |
| Pure | 1921 | 2100 | 2839 | 2839 | **3663±247** |
| Eps-1 | 1864 | 1897 | 2672 | 2672 | **3305±413** |
| Eps-3 | 1504 | 2008 | 2602 | 2602 | **3008±231** |
| Gauss-1 | 1731 | 2054 | 2667 | 2667 | **3329±270** |
| Gauss-3 | 1887 | 2018 | 2640 | 2661 | **3693±33** |

| Environment: Hopper-v2 | | | | | |
|---|---|---|---|---|---|
| Algorithm | BCQ [15] | BEAR [16] | BRAC [18] | Best Baseline | MOReL (Ours) |
| Pure | 1543 | 0 | 2291 | 2774 | **3642±54** |
| Eps-1 | 1652 | 1620 | 2282 | 2360 | **3724±46** |
| Eps-3 | 1632 | 2213 | 1892 | 2892 | **3535±91** |
| Gauss-1 | 1599 | 1825 | 2255 | 2255 | **3653±52** |
| Gauss-3 | 1590 | 1720 | 1458 | 2097 | **3648±148** |

| Environment: HalfCheetah-v2 | | | | | |
|---|---|---|---|---|---|
| Algorithm | BCQ [15] | BEAR [16] | BRAC [18] | Best Baseline | MOReL (Ours) |
| Pure | 5064 | 5325 | 6207 | **6209** | **6028±192** |
| Eps-1 | 5693 | 5435 | **6307** | **6307** | 5861±192 |
| Eps-3 | 5588 | 5149 | 6263 | **6359** | 5869±139 |
| Gauss-1 | 5614 | 5394 | **6323** | **6323** | 6026±74 |
| Gauss-3 | 5837 | 5329 | **6400** | **6400** | 5892±128 |

| Environment: Walker-v2 | | | | | |
|---|---|---|---|---|---|
| Algorithm | BCQ [15] | BEAR [16] | BRAC [18] | Best Baseline | MOReL (Ours) |
| Pure | 2095 | 2646 | 2694 | 2907 | **3709±159** |
| Eps-1 | 1921 | 2695 | 3241 | **3490** | 2899±588 |
| Eps-3 | 1953 | 2608 | **3255** | **3255** | **3186±92** |
| Gauss-1 | 2094 | 2539 | 2893 | 3193 | **4027±314** |
| Gauss-3 | 1734 | 2194 | **3368** | **3368** | **2828±589** |

→ MOReL shows the best performance in 12 of 20 environments and exceeds the performance of the data-collecting policy.




Results on MDP (naive) vs P-MDP (ours)

→ P-MDP is essential to prevent performance degradation.

Chen, Lili, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. "**Decision transformer: Reinforcement learning via sequence modeling**." NeurIPS 2021.

# DECISION TRANSFORMER: REINFORCEMENT LEARNING VIA SEQUENCE MODELING

# Main Idea



- Instead of TD-learning, train transformer models on collected experience using a sequence modeling objective
- Autoregressive transformer model (GPT) is used.

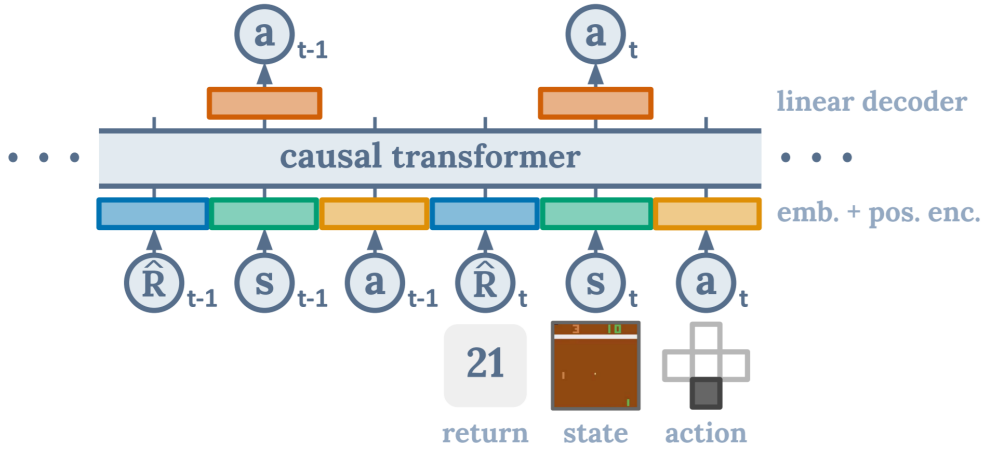
- Advantages
  - Can bypass the need for bootstrapping for long term credit assignment.
  - Avoids the need for discounting future rewards.
  - Perform credit assignment directly via self-attention
  - No other methods are needed to solve the problems of offline RL (error propagation and value overestimation).

# Details

- Trajectory representation
  - Use returns-to-go to allow the model to generate actions based on future desired returns.

$$\text{returns-to-go } \widehat{R}_t = \sum_{t'=t}^{T} r_{t'} \qquad \tau = \left(\widehat{R}_1, s_1, a_1, \widehat{R}_2, s_2, a_2, \ldots, \widehat{R}_T, s_T, a_T\right)$$

- Architecture
  - Put tokens (return-to-go, state, action) of K time steps as input.
  - Use GPT model which predicts future action tokens via autoregressive modeling
  - The action of each time step is the output of the model.
- Training
  - Sample minibatches of sequence length K from the trajectory dataset
  - Use cross entropy (discrete) or mean-squared error (continuous) as loss.



# Algorithm



**Algorithm 1** Decision Transformer Pseudocode (for continuous actions)

```
# R, s, a, t: returns-to-go, states, actions, or timesteps
# transformer: transformer with causal masking (GPT)
# embed_s, embed_a, embed_R: linear embedding layers
# embed_t: learned episode positional embedding
# pred_a: linear action prediction layer

# main model
def DecisionTransformer(R, s, a, t):
    # compute embeddings for tokens
    pos_embedding = embed_t(t)  # per-timestep (note: not per-token)
    s_embedding = embed_s(s) + pos_embedding
    a_embedding = embed_a(a) + pos_embedding
    R_embedding = embed_R(R) + pos_embedding

    # interleave tokens as (R_1, s_1, a_1, ..., R_K, s_K)
    input_embeds = stack(R_embedding, s_embedding, a_embedding)

    # use transformer to get hidden states
    hidden_states = transformer(input_embeds=input_embeds)

    # select hidden states for action prediction tokens
    a_hidden = unstack(hidden_states).actions

    # predict action
    return pred_a(a_hidden)

# training loop
for (R, s, a, t) in dataloader:  # dims: (batch_size, K, dim)
    a_preds = DecisionTransformer(R, s, a, t)
    loss = mean((a_preds - a)**2)  # L2 loss for continuous actions
    optimizer.zero_grad(); loss.backward(); optimizer.step()

# evaluation loop
target_return = 1  # for instance, expert-level return
R, s, a, t, done = [target_return], [env.reset()], [], [1], False
while not done:  # autoregressive generation/sampling
    # sample next action
    action = DecisionTransformer(R, s, a, t)[-1]  # for cts actions
    new_s, r, done, _ = env.step(action)

    # append new tokens to sequence
    R = R + [R[-1] - r]  # decrement returns-to-go with reward
    s, a, t = s + [new_s], a + [action], t + [len(R)]
    R, s, a, t = R[-K:], ...  # only keep context length of K
```

# Experiments

- How well does Decision Transformer model the distribution of returns?

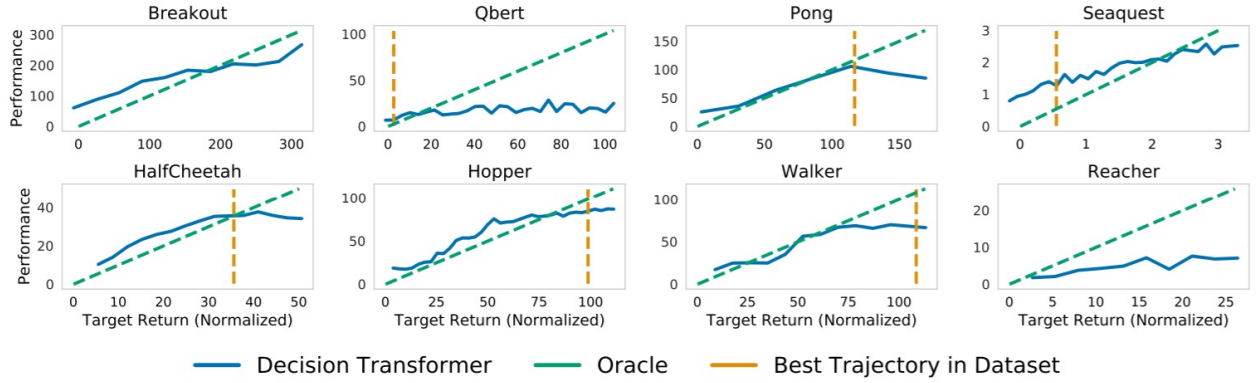



Figure 4: Sampled (evaluation) returns accumulated by Decision Transformer when conditioned on the specified target (desired) returns. **Top:** Atari. **Bottom:** D4RL medium-replay datasets.

It shows that it is possible to generate a trajectory with a desired reward sum.

- Does Decision Transformer perform well in sparse reward settings?

| | | Delayed (Sparse) | | Agnostic | | Original (Dense) | |
|---|---|---|---|---|---|---|---|
| **Dataset** | **Environment** | **DT (Ours)** | **CQL** | **BC** | **%BC** | **DT (Ours)** | **CQL** |
| Medium-Expert | Hopper | **107.3 ± 3.5** | 9.0 | 59.9 | 102.6 | 107.6 | 111.0 |
| Medium | Hopper | 60.7 ± 4.5 | 5.2 | 63.9 | **65.9** | 67.6 | 58.0 |
| Medium-Replay | Hopper | **78.5 ± 3.7** | 2.0 | 27.6 | 70.6 | 82.7 | 48.6 |

Table 7: Results for D4RL datasets with delayed (sparse) reward. Decision Transformer (DT) and imitation learning are minimally affected by the removal of dense rewards, while CQL fails.

Compared to CQL (another offline RL algorithm), it works well in sparse reward setting.

Michael Janner, Qiyang Li, and Sergey Levine, "**Offline Reinforcement Learning as One Big Sequence Modeling Problem**", Advances in Neural Information Processing Systems (NeurIPS), 2021.

# TRAJECTORY TRANSFORMER

# Trajectory Transformer



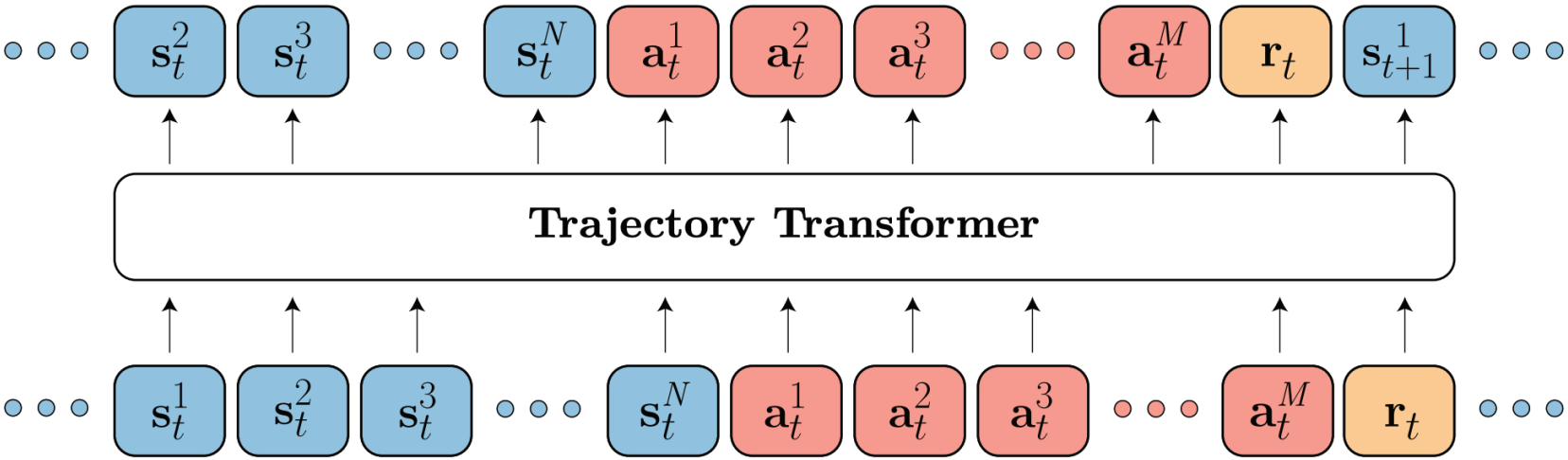
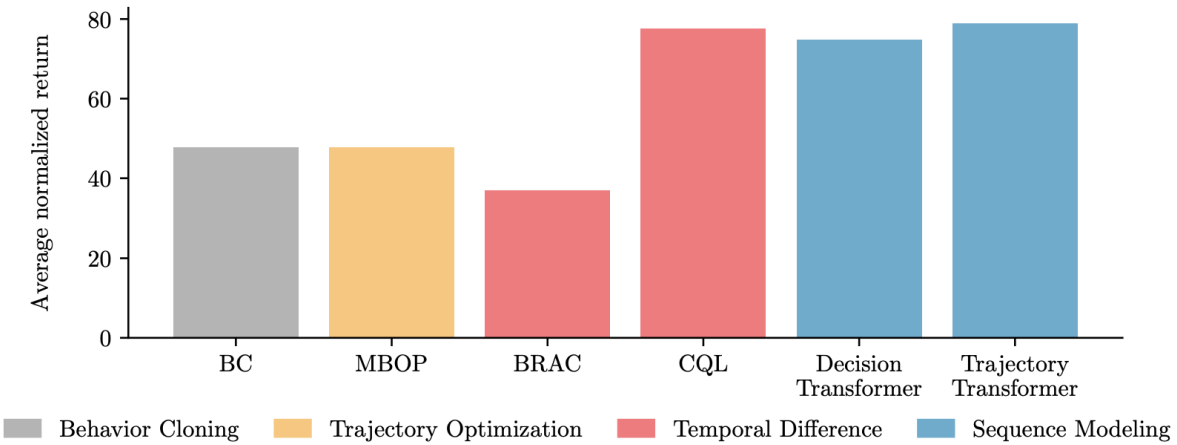



**Figure 1 (Architecture)** The Trajectory Transformer trains on sequences of (autoregressively discretized) states, actions, and rewards. Planning with the Trajectory Transformer mirrors the sampling procedure used to generate sequences from a language model.

# Experimental Results



| Dataset | Environment | BC | MBOP | BRAC | CQL | DT | TT (uniform) | TT (quantile) |
|---|---|---|---|---|---|---|---|---|
| Med-Expert | HalfCheetah | 59.9 | 105.9 | 41.9 | 91.6 | 86.8 | 40.8 ±2.3 | 95.0 ±0.2 |
| Med-Expert | Hopper | 79.6 | 55.1 | 0.9 | 105.4 | 107.6 | 106.0 ±0.28 | 110.0 ±2.7 |
| Med-Expert | Walker2d | 36.6 | 70.2 | 81.6 | 108.8 | 108.1 | 91.0 ±2.8 | 101.9 ±6.8 |
| Medium | HalfCheetah | 43.1 | 44.6 | 46.3 | 44.0 | 42.6 | 44.0 ±0.31 | 46.9 ±0.4 |
| Medium | Hopper | 63.9 | 48.8 | 31.3 | 58.5 | 67.6 | 67.4 ±2.9 | 61.1 ±3.6 |
| Medium | Walker2d | 77.3 | 41.0 | 81.1 | 72.5 | 74.0 | 81.3 ±2.1 | 79.0 ±2.8 |
| Med-Replay | HalfCheetah | 4.3 | 42.3 | 47.7 | 45.5 | 36.6 | 44.1 ±0.9 | 41.9 ±2.5 |
| Med-Replay | Hopper | 27.6 | 12.4 | 0.6 | 95.0 | 82.7 | 99.4 ±3.2 | 91.5 ±3.6 |
| Med-Replay | Walker2d | 36.9 | 9.7 | 0.9 | 77.2 | 66.6 | 79.4 ±3.3 | 82.6 ±6.9 |
| **Average** | | 47.7 | 47.8 | 36.9 | 77.6 | 74.7 | 72.6 | 78.9 |



Mandlekar, Ajay, Danfei Xu, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. "**GTI: Learning to generalize across long-horizon tasks from human demonstrations**." Robotics: Science and Systems (RSS), 2020.

# GTI: Learning to Generalize Across Long-Horizon Tasks from Human Demonstrations

# Introduction



- Learning from offline data is an effective and safe technique but learning policies that generalize beyond the demonstrated behaviors is still an open challenge.

- This paper proposes Generalization Through Imitation (GTI), a two-stage offline imitation learning algorithm.

- GTI enable robots to

  1) learn complex real world manipulation tasks efficiently from a small number of human demonstrations,

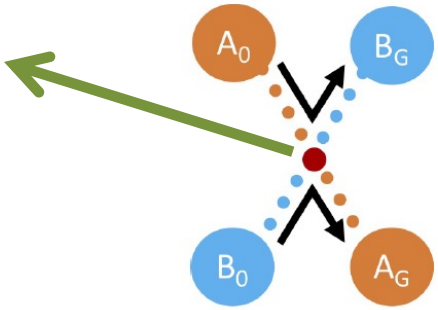
  2) synthesize new behaviors not contained in the offline data.

# GTI: Structure

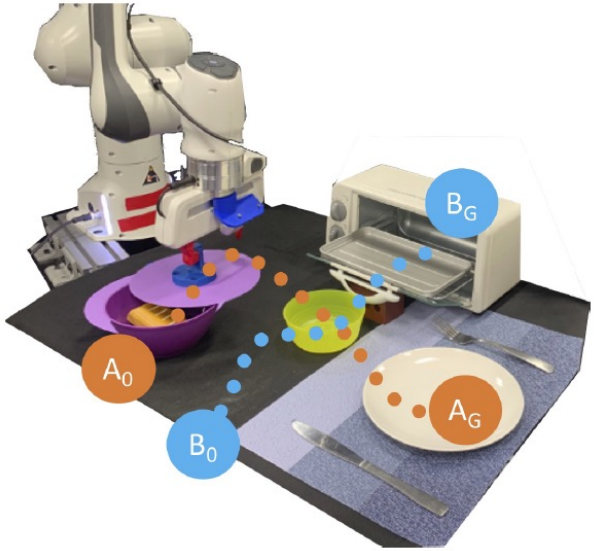


- In the first stage of GTI, they train a stochastic policy that leverages trajectory intersections to have the capacity to compose behaviors from different demonstrations together.

trajectory intersection




black arrows
: unseen behaviors to be generated



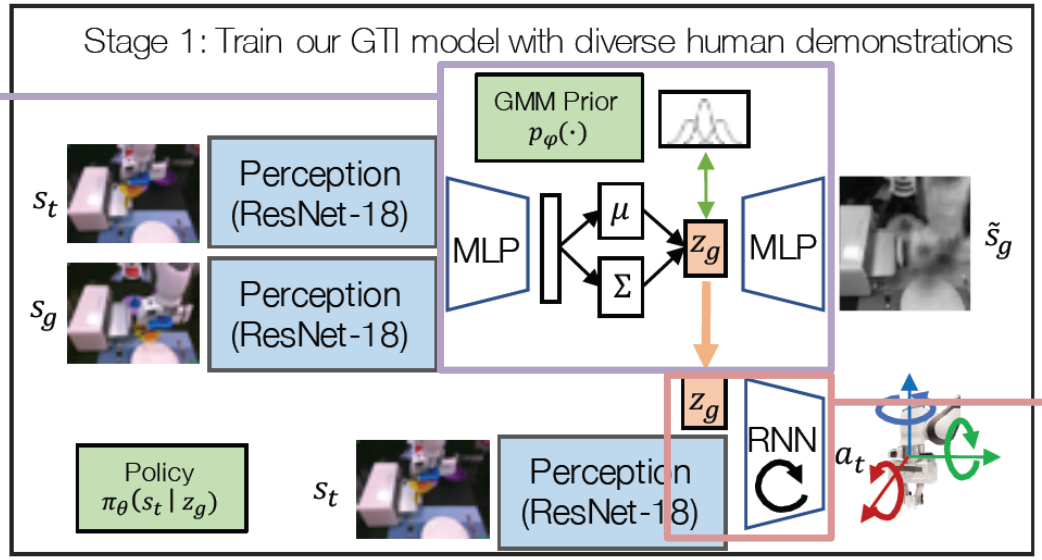

- In the second stage of GTI, they collect a small set of rollouts from the unconditioned stochastic policy of the first stage and train a goal-directed agent to generalize to novel start and goal state.

# GTI: Stage 1

- In Stage 1, GTI decomposes imitation learning into two subproblems.
  1) learn a generative model from demonstrations to predict possible future states conditioned on a current state.
  2) train a goal-conditioned imitation policy from the demonstrations using predicted states as goals.

**Generative model**

**:** Gaussian mixture model (GMM) & conditional VAE (cVAE)




**Goal-Conditioned Policy**

**:**cVAE & RNN

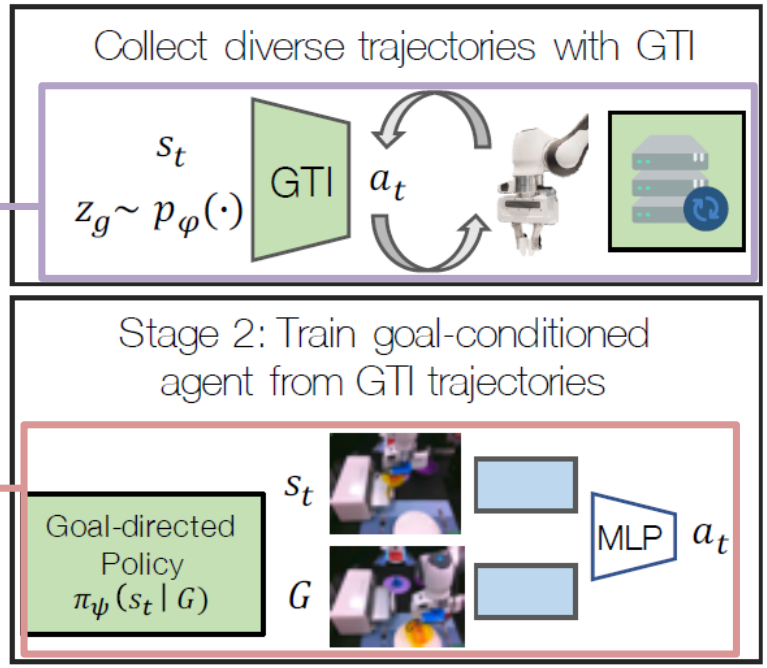
# GTI: Stage 2



- In Stage 2, GTI uses trained models in Stage 1 to generate new demonstrations through compositional generalization.
- Then, GTI trains a new policy using newly collected behaviors in a goal directed manner.

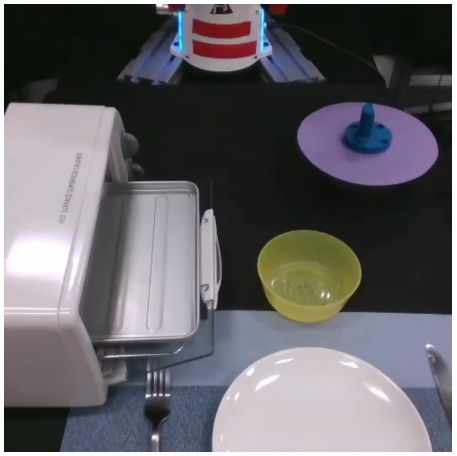
**Generate new demonstrations**
using models
trained in Stage 1

**Goal-Directed Imitation Policy**
: using a single-step
goal-directed
behavior cloning method



# GTI: Real Robot Experiment



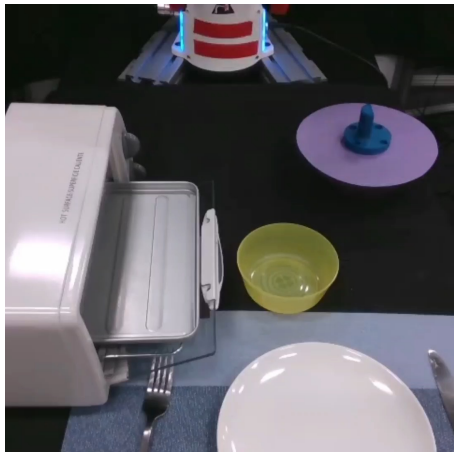


**Demonstration 1**
- Start: Bread in Container
- End: Bread on Plate

**Demonstration 2**
- Start: Bread on Table
- End: Bread in Oven

**Demonstrations**
(offline data)

**Train GTI**



**Unseen Trajectory**
- Start: Bread in Container
- End: Bread in Oven