

ML Meeting

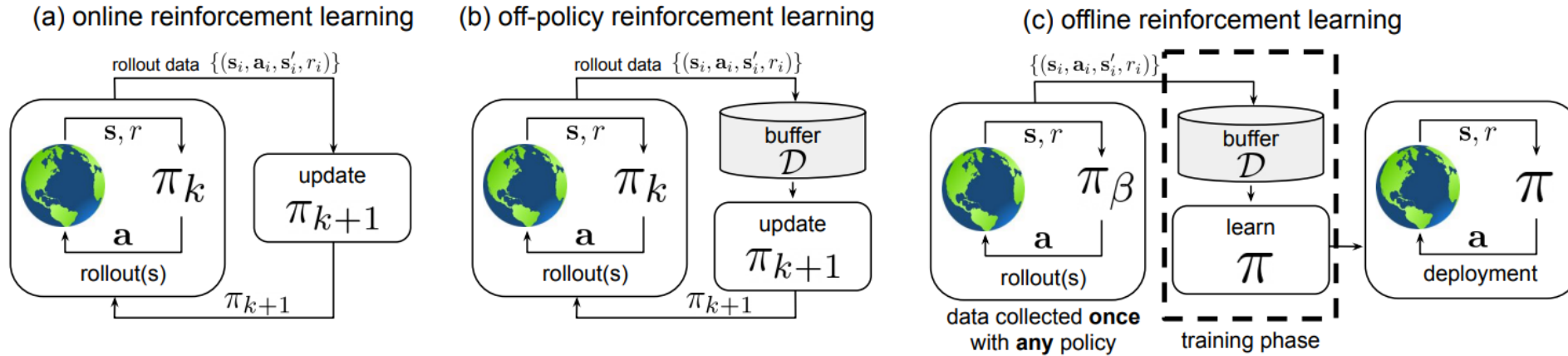
Idea Proposals

12/28

Idea Proposals

- NTK + policy-constrained offline RL
- NTK + uncertainty-based offline RL
- Decision Transformer + NTK
- GA-NTK + GAIL

Offline RL



We aim to learn a policy π from the history of trajectories $\mathcal{D} = \{s_t, a_t, s'_t, r_t\}$ generated by behavioral policy π_β s.t. the performance $\pi \geq \pi_\beta$ (Which means we want to train an agent's policy π only from the history of trajectories \mathcal{D})

$$\hat{Q}_{k+1}^\pi \leftarrow \arg \min_Q \mathbb{E}_{(s, a, s') \sim \mathcal{D}} \left[\left(Q(s, a) - \left(r(s, a) + \gamma \mathbb{E}_{a' \sim \pi_k(a'|s')} [\hat{Q}_k^\pi(s', a')] \right) \right)^2 \right]$$

$$\pi_{k+1} \leftarrow \arg \max_\pi \mathbb{E}_{s \sim \mathcal{D}} \left[\mathbb{E}_{a \sim \pi(a|s)} [\hat{Q}_{k+1}^\pi(s, a)] \right] \text{ s.t. } D(\pi, \pi_\beta) \leq \epsilon.$$

Challenge

- Extrapolation Error
 - Because the agent's policy isn't the same as the behavioral policy. The agent may overestimate unseen $Q^\pi(s, a)$.
 - In the offline setting, the policy cannot correct such over-optimistic Q-values.

Extrapolation Error

- **Final Buffer:** train a DDPG agent for 1 million time steps, adding $\mathcal{N}(0, 0.5)$ Gaussian noise to actions for high exploration.
- **Concurrent:** We concurrently train the off-policy and behavioral DDPG agents, for 1 million time steps. A standard $\mathcal{N}(0, 0.1)$ Gaussian noise is added to action
- **Imitation:** A trained DDPG agent acts as an expert, and is used to collect a dataset of 1 million transitions.

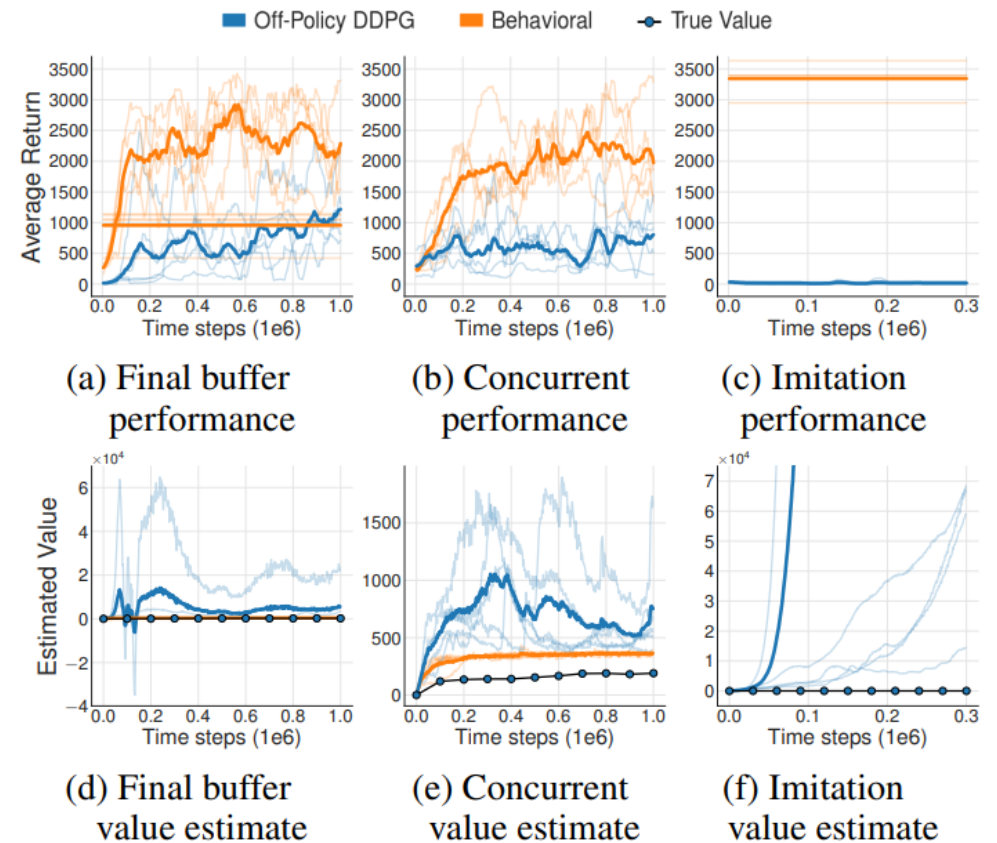


Figure 1. We examine the performance (top row) and corresponding value estimates (bottom row) of DDPG in three batch tasks on Hopper-v1. Each individual trial is plotted with a thin line, with the mean in bold (evaluated without exploration noise). Straight lines represent the average return of episodes contained in the batch (with exploration noise). An estimate of the true value of the off-policy agent, evaluated by Monte Carlo returns, is marked by a dotted line. In all three experiments, we observe a large gap in the performance between the behavioral and off-policy agent, even when learning from the same dataset (*concurrent*). Furthermore, the value estimates are unstable or divergent across all tasks.

Extrapolation Error

- In each task, the off-policy agent performances significantly worse than the behavioral agent, even in the concurrent experiment.

Solutions for Extrapolation Error

- Support Constraints: Penalizes the total amount of probability mass on out-of-distribution actions under the π distribution

$$D_{support,\epsilon}(\pi(\cdot|s), \pi_\beta(\cdot|s)) = \sum_{a \in \mathcal{A}, \pi_\beta(a|s) \leq \epsilon} \pi(a|s)$$

- MMD distance does not depend on the specific densities of the behavior distribution or the policy, and can be computed via a kernel-based distance on samples from each distribution
- However, MMD just provides sufficient enough for constraining supports when finite samples are used.
- SOTA algorithm: BEAR

Policy Constraints Offline RL

Integral probability metrics (IPMs). Another choice of the policy constraint D is an integral probability metric, which is a divergence measure with the following functional form dependent on a function class Φ :

$$D_{\Phi}(\pi(\cdot|\mathbf{s}), \pi_{\beta}(\cdot|\mathbf{s})) = \sup_{\phi \in \Phi, \phi: S \times A \rightarrow \mathbb{R}} \left| \mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s})} [\phi(\mathbf{s}, \mathbf{a})] - \mathbb{E}_{\mathbf{a}' \sim \pi_{\beta}(\cdot|\mathbf{s})} [\phi(\mathbf{s}, \mathbf{a}')] \right|. \quad (21)$$

Distance measured in MMD

$$\begin{aligned} \text{MMD}^2(\pi(\cdot|\mathbf{s}), \pi_{\beta}(\cdot|\mathbf{s})) &= \mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s}), \mathbf{a}' \sim \pi(\cdot|\mathbf{s})} [k(\mathbf{a}, \mathbf{a}')] - \\ &\quad 2\mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s}), \mathbf{a}' \sim \pi_{\beta}(\cdot|\mathbf{s})} [k(\mathbf{a}, \mathbf{a}')] + \mathbb{E}_{\mathbf{a} \sim \pi_{\beta}(\cdot|\mathbf{s}), \mathbf{a}' \sim \pi_{\beta}(\cdot|\mathbf{s})} [k(\mathbf{a}, \mathbf{a}')] , \end{aligned}$$

where $k(\cdot, \cdot)$ represents any radial basis kernel, such as the Gaussian or Laplacian kernel. As another

IDEA 1: NTK instead of MMD

Uncertainty-Based Offline RL

- Uncertainty Estimation: Penalize the unseen action on $Q^\pi(s, a)$
 - Challenges: Hard to measure the uncertainty. The SOTA algorithm EDAC trains 10 ~ 500 Q-networks to estimate the uncertainty.

$$\pi_{k+1} \leftarrow \arg \max_{\pi} \mathbb{E}_{\mathbf{s} \sim \mathcal{D}} \left[\underbrace{\mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} \left[\mathbb{E}_{Q_{k+1}^\pi \sim \mathcal{P}_{\mathcal{D}}(Q^\pi)} [Q_{k+1}^\pi(\mathbf{s}, \mathbf{a})] - \alpha \text{Unc}(\mathcal{P}_{\mathcal{D}}(Q^\pi)) \right]}_{\text{conservative estimate}} \right], \quad (24)$$

IDEA 2: Use NTK to measure uncertainty

However, there is another way to tackle the offline RL problem.

How about we take offline RL problem as a sequence modeling problem?

Decision Transformer(DT)

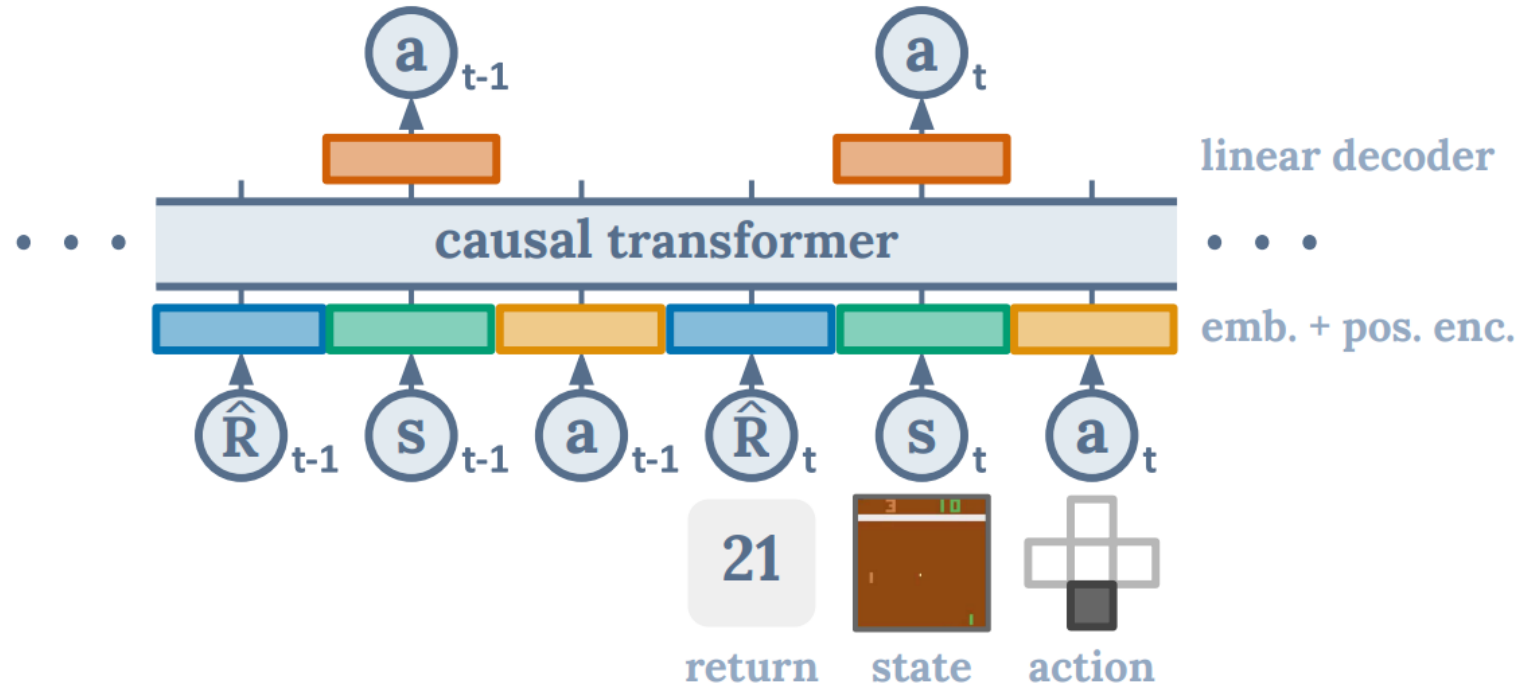


Figure 1: Decision Transformer architecture¹. States, actions, and returns are fed into modality-specific linear embeddings and a positional episodic timestep encoding is added. Tokens are fed into a GPT architecture which predicts actions autoregressively using a causal self-attention mask.

DT use GPT model. Denote s_t is state, a_t is action and, r_t is reward at time step t .

\hat{R}_i is the reward-to-go, which means $\hat{R}_{t-1} = \sum_{i=t}^T r_i$ for a sequence with length T .

But Why It Work?

Actually, Decision Transformer is a Conditional Policy Generator(In My Opinion).

Decision Transformer As Conditional Policy Generator

- The original paper gives an explanation in the view of shortest path. But I don't think it's convincing.
- If we see the Decision Transformer as a **Conditional Policy Generator**. I think the Decision Transformer has 3 assumptions.
 - Assume the **decay of the reward** $\gamma = 1$
 - Assume the environment is **non-stochastic Hidden Markov Chain**
 - Assume **deterministic policy**

Decision Transformer As Conditional Policy Generator

- Denote the actions space as $a \in \mathcal{A}$, state space as $s \in \mathcal{S}$, value function of policy π as $V_\pi(s)$, the decay factor of the reward as γ , and the reward function as $r(s, a)$.
- We consider the **assumption: deterministic policy**

$$V_\pi(s_t) = \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1} | s_t, \pi(s_t)) (r(s_t, \pi(s_t)) + \gamma V_\pi(s_{t+1}))$$

Decision Transformer As Conditional Policy Generator

- While considering the **assumption: non-stochastic MDP** and denote the transition function as $t(s_t, a_t) = s_{t+1}$, the transition probability reduce to a Dirac-Delta distribution which means $\delta(s'|s, \pi(s)) = P(s'|s, \pi(s)) = 1$ if $t(s, a) = s'$, otherwise $= 0$. Thus, we can derive

$$V_{\pi}(s_t) = r(s_t, \pi(s_t)) + \gamma V_{\pi}(s_{t+1})$$

- Finally, consider the **assumption: decay of the reward $\gamma = 1$**

$$V_{\pi}(s_t) = r(s_t, \pi(s_t)) + V_{\pi}(s_{t+1})$$

Trivially, $V_{\pi}(s_t)$ is reward-to-go \hat{R}_t

Decision Transformer As Conditional Policy Generator

- First, we consider to learn a policy generator G

$$G(\hat{R}_0; \theta) = \pi \quad s.t. \quad \hat{R}_0 = V_\pi(s_0)$$

- Next, we can expand the model G as a **Conditional Policy Generator** H

$$H(\hat{R}_t, s_t; \theta) = \hat{a}_t \quad s.t. \quad \hat{R}_t = V_\pi(s_t)$$

$$\arg \min_{\theta} \sum_{t=0}^T ||\hat{a}_t - a_t||_2$$

- Consider the transformer is an auto-regressive model, it models the environment as a **hidden markov chain** corresponding to the assumption.

Disadvantages

- Cannot use the model online, since we need to produce the reward-to-go and feed it into the model.
- Poor performance in stochastic environments.
- The performance during testing time would highly depends on the given value of reward-to-go. If the first reward-to-go \hat{R}_0 we give is too high, DT would behave unexpectedly.

Advantages

- Provide competitive performance compared to SOTA offline RL algorithm in most of environment.

Performance

Dataset	Environment	DT (Ours)	CQL	BEAR	BRAC-v	AWR	BC
Medium-Expert	HalfCheetah	86.8 \pm 1.3	62.4	53.4	41.9	52.7	59.9
Medium-Expert	Hopper	107.6 \pm 1.8	111.0	96.3	0.8	27.1	79.6
Medium-Expert	Walker	108.1 \pm 0.2	98.7	40.1	81.6	53.8	36.6
Medium-Expert	Reacher	89.1 \pm 1.3	30.6	-	-	-	73.3
Medium	HalfCheetah	42.6 \pm 0.1	44.4	41.7	46.3	37.4	43.1
Medium	Hopper	67.6 \pm 1.0	58.0	52.1	31.1	35.9	63.9
Medium	Walker	74.0 \pm 1.4	79.2	59.1	81.1	17.4	77.3
Medium	Reacher	51.2 \pm 3.4	26.0	-	-	-	48.9
Medium-Replay	HalfCheetah	36.6 \pm 0.8	46.2	38.6	47.7	40.3	4.3
Medium-Replay	Hopper	82.7 \pm 7.0	48.6	33.7	0.6	28.4	27.6
Medium-Replay	Walker	66.6 \pm 3.0	26.7	19.2	0.9	15.5	36.9
Medium-Replay	Reacher	18.0 \pm 2.4	19.0	-	-	-	5.4
Average (Without Reacher)		74.7	63.9	48.2	36.9	34.3	46.4
Average (All Settings)		69.2	54.2	-	-	-	47.7

Table 2: Results for D4RL datasets³. We report the mean and variance for three seeds. Decision Transformer (DT) outperforms conventional RL algorithms on almost all tasks.

Performance

Game	DT (Ours)	CQL	QR-DQN	REM	BC
Breakout	267.5 ± 97.5	211.1	17.1	8.9	138.9 ± 61.7
Qbert	15.4 ± 11.4	104.2	0.0	0.0	17.3 ± 14.7
Pong	106.1 ± 8.1	111.9	18.0	0.5	85.2 ± 20.0
Seaquest	2.5 ± 0.4	1.7	0.4	0.7	2.1 ± 0.3

Table 1: **Gamer-normalized scores for the 1% DQN-replay Atari dataset.** We report the mean and **variance across 3 seeds**. Best mean scores are highlighted in bold. Decision Transformer (DT) performs comparably to CQL on 3 out of 4 games, and outperforms other baselines in most games.

- In Qbert, DT is extremely poor, since Qbert has highly stochastic environment.

Idea 3: Kernelize Conditional Policy Generator

- Predict action a_{t-1} and next reward-to-go \hat{R}_t at time step $t - 1$.

$$H(\hat{R}_t, s_t; \theta) = \hat{a}_t, \hat{R}_{t+1} \quad s.t. \quad \hat{R}_{t+1} = V_\pi(s_{t+1})$$

$$\arg \min_{\theta} \sum_{t=0}^T ||\hat{a}_t - a_t||_2$$

- Then kernelize it.

Advantage

- Since the loss is defined as MSE. We can use NTK-GP compute the agent without training.
- Can be applied on both offline RL and real-time situation, since it don't need to train the agent iteratively.
- Don't need to search the next sampling point in the high dimensional space like Bayesian optimization.
- Safe exploration? Avoid over-fitting(Bayesian) naturally. High sample efficiency.
- The next reward-to-go is predicted and we only need to give the first reward-to-go.

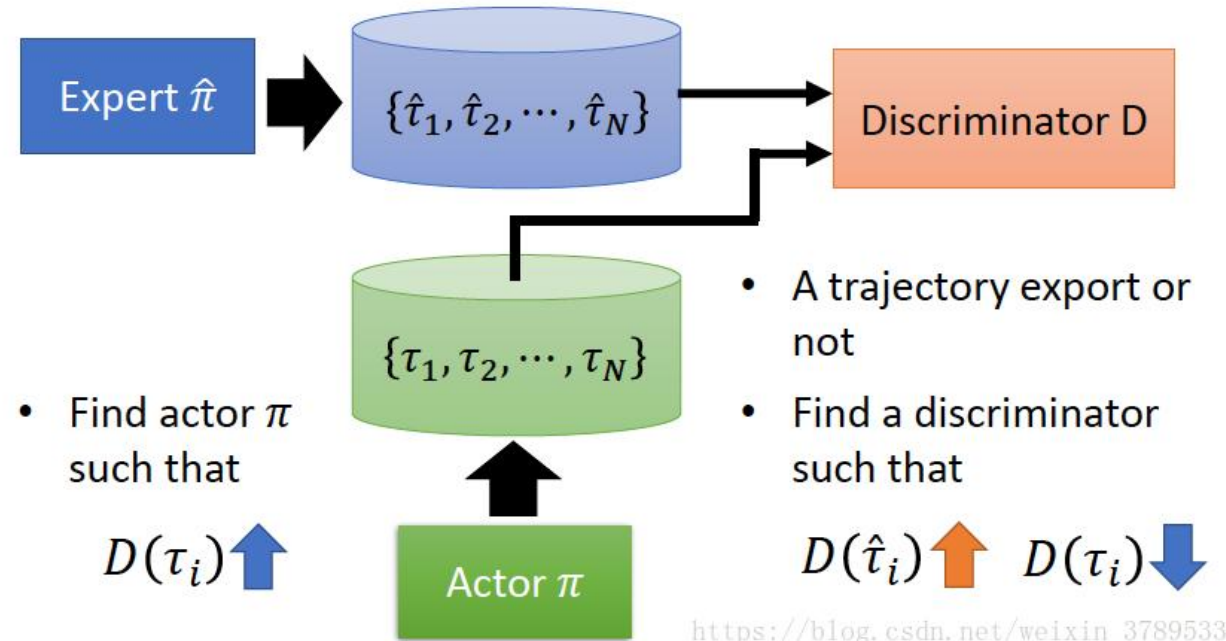
Disadvantage

- If we want to extend the Decision Transformer to stochastic MDP, we need to work around the estimation of \hat{R}_t in stochastic MDP and stochastic policy.

Generative Adversarial Imitation Learning(GAIL)

Given expert trajectory $\hat{\tau}$ generated by expert policy $\hat{\pi}$ as ground truth and the generated trajectory τ by generator(agent) π .

Discriminator D need to distinguish which trajectory is generated by the generator(fake).



Generative Adversarial Imitation Learning(GAIL)

Algorithm 1 Generative adversarial imitation learning

- 1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
- 2: **for** $i = 0, 1, 2, \dots$ **do**
- 3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
- 4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i} [\nabla_w \log(D_w(s, a))] + \hat{\mathbb{E}}_{\tau_E} [\nabla_w \log(1 - D_w(s, a))] \quad (17)$$

- 5: Take a policy step from θ_i to θ_{i+1} , using the TRPO rule with cost function $\log(D_{w_{i+1}}(s, a))$. Specifically, take a KL-constrained natural gradient step with

$$\begin{aligned} & \hat{\mathbb{E}}_{\tau_i} [\nabla_{\theta} \log \pi_{\theta}(a|s) Q(s, a)] - \lambda \nabla_{\theta} H(\pi_{\theta}), \\ & \text{where } Q(\bar{s}, \bar{a}) = \hat{\mathbb{E}}_{\tau_i} [\log(D_{w_{i+1}}(s, a)) \mid s_0 = \bar{s}, a_0 = \bar{a}] \end{aligned} \quad (18)$$

- 6: **end for**
-

Issues of GAIL

Most of them are the same as GANs

- Mode collapse, unstable training, low sample efficiency
- Recent search [What Matters for Adversarial Imitation Learning?](#) argues that
 - GAILs only synthetic demonstrations may lead to algorithms which perform poorly in the more realistic scenarios with human demonstrations, which means that GAILs doesn't generalize well on the reality.
 - But performs well on low-dimension tasks.
- Challenged by Offline RL, which usually can perform well on non-expert dataset

Idea: GA-NTK + GAIL

Advantages

- Avoid mode collapse, unstable training, low sample efficiency

Disadvantages

- Interpolation on small training dataset, same as GAIL

Notes

- Clarify MMD and support constraint
- Argue more about the advantages of DT + NTK
- Fix the notation of DT as sequence prediction