23년 하계 워크샵

연세대학교 산업공학과 경영과학연구실
석사과정 김지원

현재 연구 진행 상황

연구 목적 : DRL을 이용한 리프터 할당 알고리즘

Summary: FAB의 가시적인 정보만을 이용하여 반송 시간 예측하고 이를 최소화하는 리프터를 DQN으로 예측 후 할당

프레임워크

1) 시뮬레이션을 통해 policy 학습 - 환경과 상호작용하는 Online learning / experience buffer에 수집된 데이터로 학습하는 Off policy
2) FAB 과 Lot states를 배열한 matrix에서 local feature를 추출하기 위해 CNN사용 (DQN)

현재 연구 진행 상황

Challenges

1) Online learning의 단점
- Online learning에서는 초기에 네트워크가 학습되기 전까지 사실상 랜덤 리프터로 할당되는 State와 Action 데이터가 수집됨
- 초기에 starting point를 제시해주면 학습 속도를 높일 수 있음

2) Feature extraction
- 다른 형태로 state 표현이 가능한가?

3) 실제 FAB에서의 학습이 불가능함
- 실제 FAB과 시뮬레이션 환경에 차이 존재
- 실제 FAB물류에서는 uncertainty가 높은 action으로 exploration할 경우, 이후 disruptive impact
- 환경 초기화가 거의 불가능하기 때문에 실제 환경과 상호작용하는 online RL 적용이 어려움

현재 연구 진행 상황

Solution

1) Historical data를 이용한 offline learning 후 fine tuning in online learning

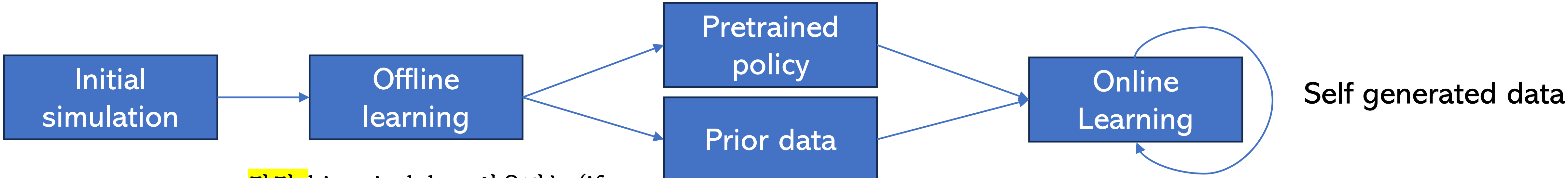
- 선행연구를 통해 historical data를 이용하면 더 좋은 성능을 보임을 증명
- Expert demonstration을 통해 얻은 데이터로 지도학습 후 Pretrained model을 이용해서 초기 네트워크로 이용
- FAB 환경이 변화하더라도 빠른 학습 가능 (이후 transfer learning으로의 확장 가능)

2) 다른 feature extractor이용

- Transformer: State matrix의 단위가 다르기 때문에 position embedding과 multi-head attention을 통해 해결

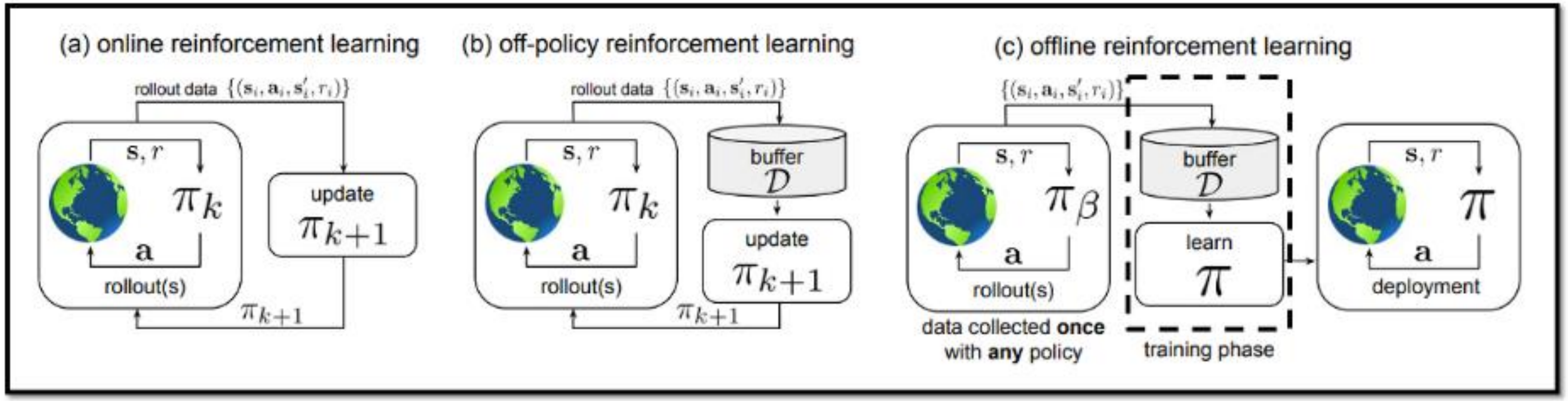
3) Conservative exploration with human feedback
- 전문가가 직접보고 명백하게 불가능하거나 좋지 않은 action을 candidate에서 제거 (Actor)

실험 설계



장점: historical data 사용가능 (if optimal, guarantee better performance)
단점: env와 상호 작용 불가

장점: env와 상호 작용하며 explore가능
단점: If 시뮬레이션을 통한 학습, FAB 실제 환경을 시뮬레이션이 제대로 반영 하지 못하면 결과 무쓸모
If 실제 환경을 통한 학습, 과감한 exploration 불가



Decision Transformer
: Reinforcement Learning via SequenceModeling

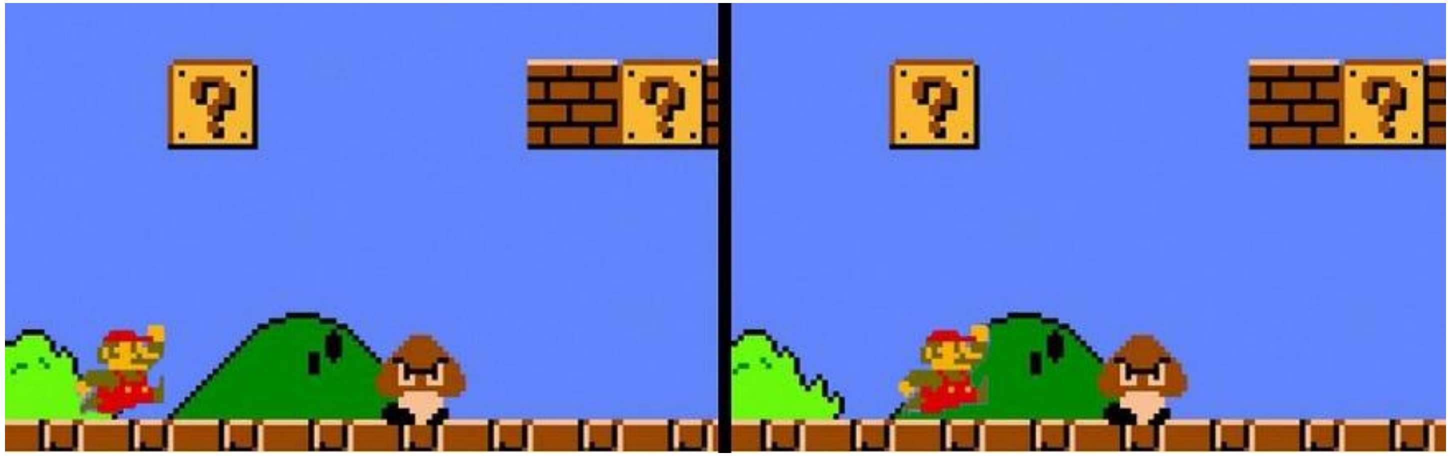
Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, Igor Mordatch

UC Berkeley ,Facebook AIResearch, GoogleBrain

Problem Statement

Deadly triad in RL

- function approximation: 비슷한 State에 대해서 neural network가 혼동
- Bootstrapping: 같은 종류의 추정 값을 업데이트할 때 다른 추정 값을 이용
- Off-policy: Divergence 가능성이 높아짐



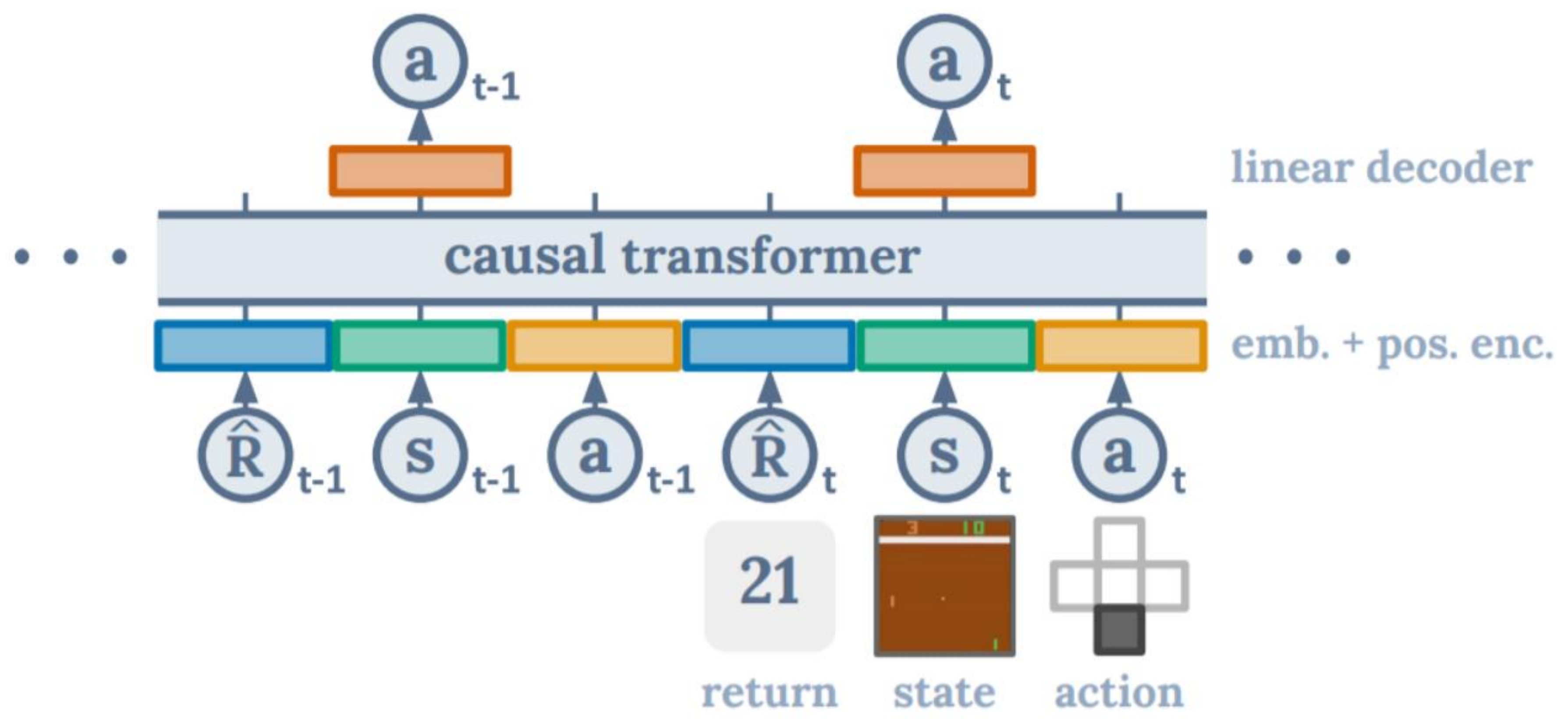
Key Idea

- Offline learning : Sequence modeling objective를 사용하여 수집된 경험에 대해 Transformer 모델을 훈련 (bootstrap하지 않음, short-sighted decision 방지)

- 리워드를 최대화 하는 방식으로 학습하는 것이 아니라, 주어진 past experience가 주어졌을 때, desired reward에 가장 가까운 action을 도출하도록 함

- causally masked Transformer 프레임워크 사용하여 자기회귀적으로 trajectories를 모델링 (기존의 트랜스포머 구조 사용)

Methodology



Methodology

Trajectory representation

$$\tau = \left(\widehat{R}_1, s_1, a_1, \widehat{R}_2, s_2, a_2, \dots, \widehat{R}_T, s_T, a_T\right)$$

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)
```

Linear layer for each modality

Methodology

Training

- K timesteps를 샘플링하고 input token s_t를 보고 a_t를 예측하도록 학습함
이때 discrete action은 cross entropy, continuous action은 mse loss를 사용

Evaluation

- Target return과 environment starting state를 정함
- 생성된 action을 취한 뒤 target return에서 지금 얻은 return을 빼서 return- to – go 계산 하고 next state 구함

```
# 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

Atari Game

- The paper compared Decision transformer with standard TD and imitation learning approaches for offline RL

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.

Experiments

Open Gym

- Medium : 1 million time steps generated by a“medium”policy that achieves approximately one-third the score of an expert policy
- Medium-Replay: Replay buffer of an agent trained to the performance of a medium policy
- Medium-Expert: Medium policy dataset + Expert dataset

Dataset	Environment	DT (Ours)	CQL	BEAR	BRAC-v	AWR	BC
Medium-Expert	HalfCheetah	**86.8 ± 1.3**	62.4	53.4	41.9	52.7	59.9
Medium-Expert	Hopper	107.6 ± 1.8	**111.0**	96.3	0.8	27.1	79.6
Medium-Expert	Walker	**108.1 ± 0.2**	98.7	40.1	81.6	53.8	36.6
Medium-Expert	Reacher	**89.1 ± 1.3**	30.6	-	-	-	73.3
Medium	HalfCheetah	42.6 ± 0.1	44.4	41.7	**46.3**	37.4	43.1
Medium	Hopper	**67.6 ± 1.0**	58.0	52.1	31.1	35.9	63.9
Medium	Walker	74.0 ± 1.4	79.2	59.1	**81.1**	17.4	77.3
Medium	Reacher	**51.2 ± 3.4**	26.0	-	-	-	**48.9**
Medium-Replay	HalfCheetah	36.6 ± 0.8	46.2	38.6	**47.7**	40.3	4.3
Medium-Replay	Hopper	**82.7 ± 7.0**	48.6	33.7	0.6	28.4	27.6
Medium-Replay	Walker	**66.6 ± 3.0**	26.7	19.2	0.9	15.5	36.9
Medium-Replay	Reacher	**18.0 ± 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

Reacher: a goal-conditioned task

AWAC: Accelerating Online Reinforcement Learning with Offline Datasets

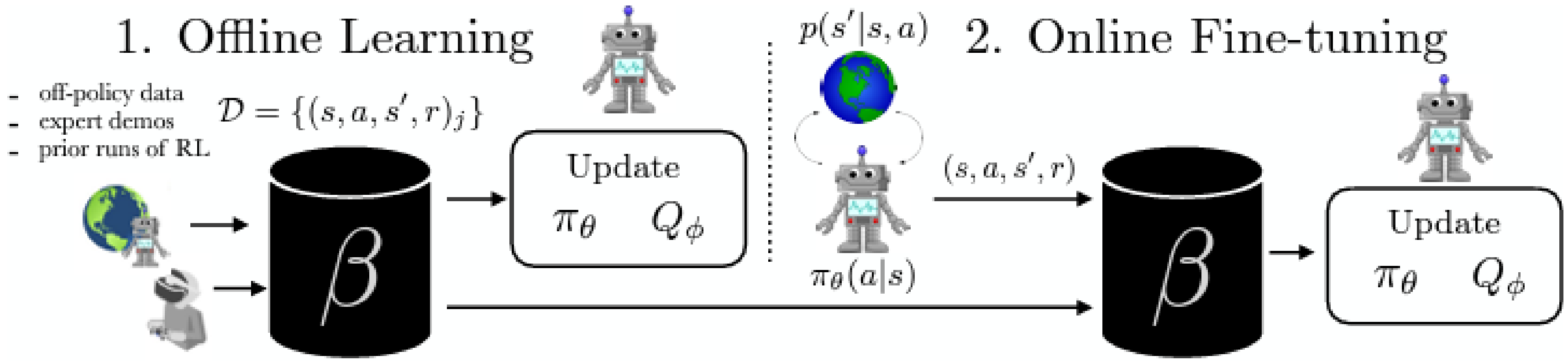
Ashvin Nair, Abhishek Gupta, Murtaza Dalal, Sergey Levine

Department of Electrical Engineering and Computer Science, UC Berkely

Background

Offline learning 후 Online learning을 통해 fine tuning

- Online learning process에 이전에 수집된 데이터를 효과적으로 사용 -> starting point를 제공

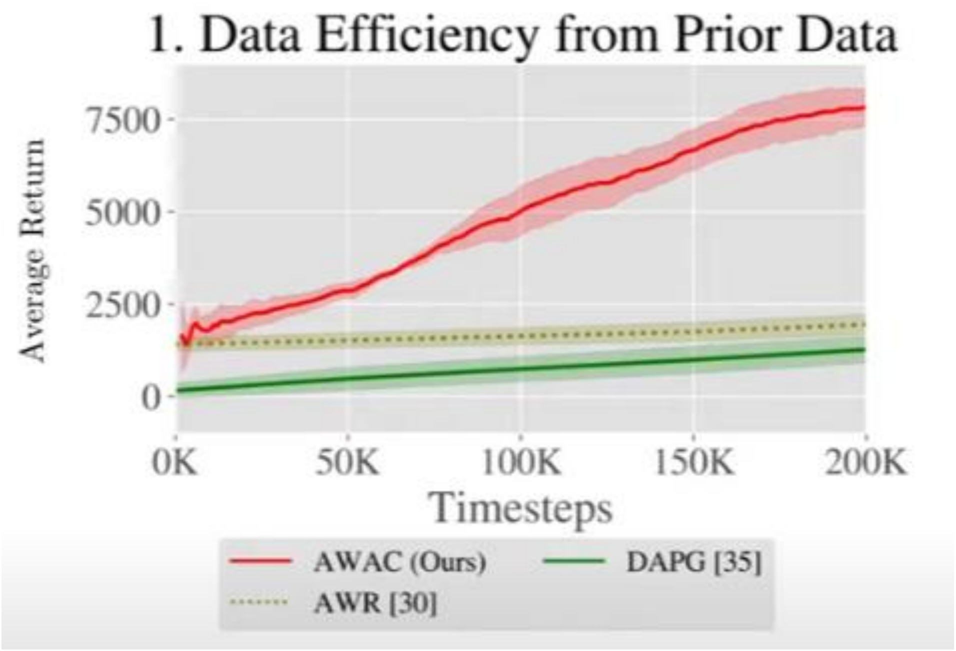


Problem Statement

Challenges in offline RL with online fine-tuning

1. Data Efficiency

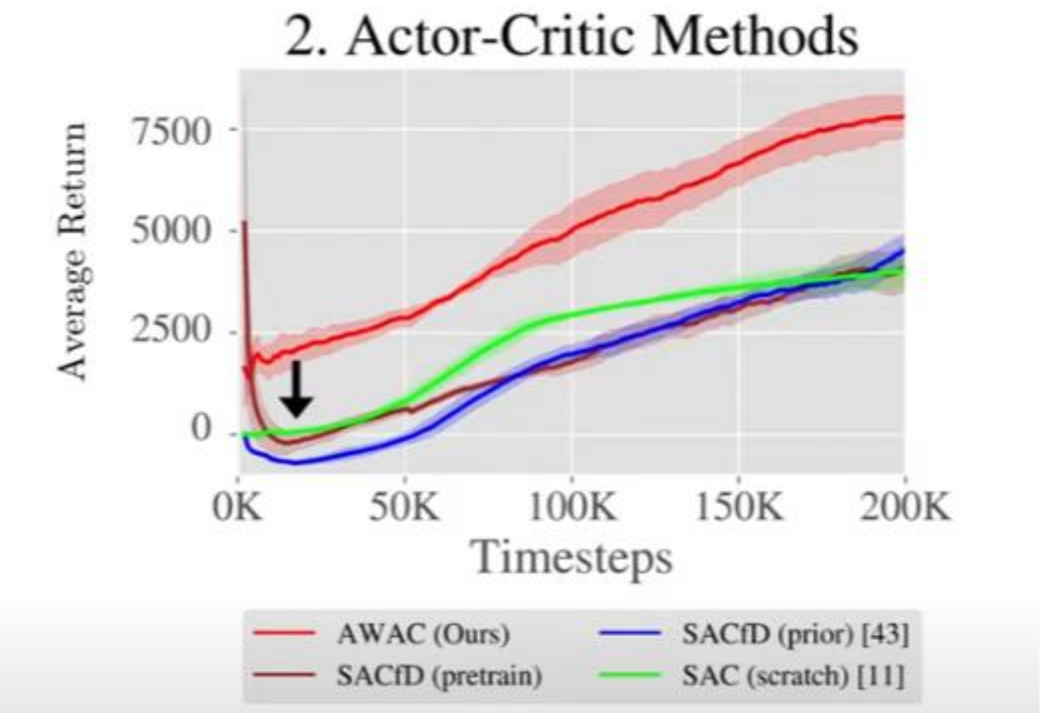
- 이전에 수집된 데이터가 optimal이 아닐 수 있음
- On-policy fine tuning에서는 prior data를 쓰지 못하기 때문에 비효율적임



Problem Statement

2. Actor-critic methods do not take advantage of offline training

Bootstrap error in offline learning with actor critic methods
:Q(s',a')가 A(s,a)로 업데이트 될때 a'가 기존 데이터 분포에서 벗어날 경우 정확하지 않을 수 있음



Problem Statement

3. Policy constraint methods

- Bootstrapping error를 막기 위해 policy constraint를 이용할 경우, offline에서는 잘 작동하나 fine tuning이 잘되지 않는 문제 발생
- Policy constraint: Optimizing the policy to maximize the estimated Q function while restricting the policy distribution to stay close to the data observed so far

$$\pi_{k+1} = \arg\max_{\pi \in \Pi} \mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s}, \mathbf{a})]$$
$$\text{s.t. } D_{\mathrm{KL}}(\boxed{\pi(\cdot|\mathbf{s})} \| \boxed{\pi_\beta(\cdot|\mathbf{s})}) \leq \epsilon.$$

Actor being updated

Behavior model (from supervised learning)
Distribution from which all of the data seen so far

그러나 Prior data로 학습 (지도학습)한 모델은 online 학습에 적합하지 않아 conservative exploration -> Limited improvement

Key Idea

Advantaged Weighted Actor Critic

- Policy evaluation : Off-policy TD learning (for data efficiency)
- Policy improvement update to avoid conservative behavior and bootstrap error accumulation

  : AWAC incorporates a KL constraint into the actor-critic framework implicitly
  : Avoid modeling of the previous observed data with a parametric model

Advantaged Weighted Actor Critic

Advantaged Weighted Actor Critic

- AWAC incorporates a KL constraint into the actor-critic framework implicitly

$$\pi_{k+1} = \arg\max_{\pi \in \Pi} \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s},\mathbf{a})] \quad \text{s.t. } D_{\mathrm{KL}}(\pi(\cdot|\mathbf{s})||\pi_\beta(\cdot|\mathbf{s})) \le \epsilon.$$

K 번째 iteration

$$\downarrow$$

$$\mathcal{L}(\pi,\lambda) = \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[A^{\pi_k}(\mathbf{s},\mathbf{a})] + \lambda(\epsilon - D_{\mathrm{KL}}(\pi(\cdot|\mathbf{s})||\pi_\beta(\cdot|\mathbf{s})))$$

$$\pi^*(\mathbf{a}|\mathbf{s}) = \frac{1}{Z(\mathbf{s})}\pi_\beta(\mathbf{a}|\mathbf{s})\exp\left(\frac{1}{\lambda}A^{\pi_k}(\mathbf{s},\mathbf{a})\right)$$

The targets are obtained by reweighting the state-action pairs observed in the current dataset by the predicted advantages from the learned critic

$$\downarrow$$

$$\theta_{k+1} = \arg\max_{\theta} \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a}\sim\beta}\left[\log \pi_\theta(\mathbf{a}|\mathbf{s})\exp\left(\frac{1}{\lambda}A^{\pi_k}(\mathbf{s},\mathbf{a})\right)\right]$$

policy update by sampling directly from β

Advantaged Weighted Actor Critic

Advantaged Weighted Actor Critic

Algorithm 1 Advantage Weighted Actor Critic (AWAC)

Dataset $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)_j\}$
Initialize buffer $\beta = \mathcal{D}$
Initialize π_θ, Q_ϕ
for iteration $i = 1, 2, \ldots$ **do**
    Sample batch $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r) \sim \beta$
    $y = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}', \mathbf{a}'}[Q_{\phi_{k-1}}(\mathbf{s}', \mathbf{a}')]$
    $\phi \leftarrow \arg\min_\phi \ \mathbb{E}_{\mathcal{D}}[(Q_\phi(\mathbf{s}, \mathbf{a}) - y)^2]$
    $\theta \leftarrow \arg\max_\theta \ \mathbb{E}_{\mathbf{s}, \mathbf{a} \sim \beta}\left[\log \pi_\theta(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s}, \mathbf{a})\right)\right]$
    if $i >$ num_offline_steps **then**
        $\tau_1, \ldots, \tau_K \sim p_{\pi_\theta}(\tau)$
        $\beta \leftarrow \beta \cup \{\tau_1, \ldots, \tau_K\}$
    end if
end for

Deep Q-learning from Demonstrations

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris A., Osband, I., Dulac-Arnold, G., Agapiou,J.

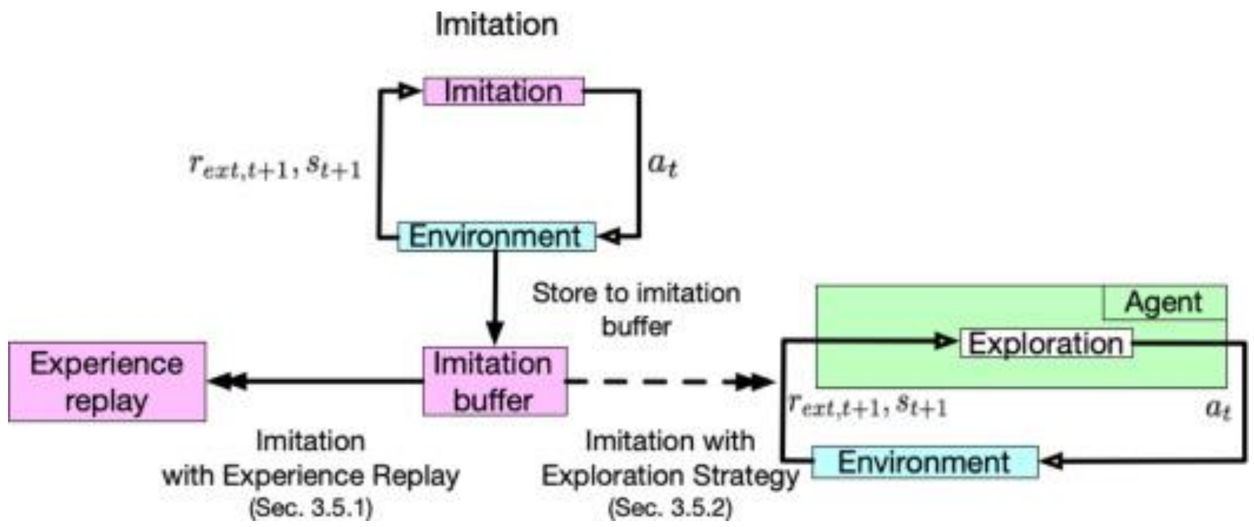
Google Deepmind

The Thirty-Second AAAI Conference on Artificial Intelligence (2017)

Background

Imitation-based reinforcement learning

"Demonstrations do not have to be perfect; rather, they just need to be a good starting point"



Background

Imitation in experience replay methods

- Combining samples from demonstrations with samples collected by an agent in a single experience replay
- The transitions from demonstrations have a higher probability of being selected

Background

Imitation with exploration strategy methods

The agent randomly explores from a state alongside a single demonstration run or can ask for help from the demonstrator
→ the effect of overcoming the initial burden of exploration through demonstrations

Problem Statement

It is difficult to apply RL algorithms in real world problems (autonomous vehicles, data centers etc.)
These algorithms learn good control policies only after many millions of steps

This situation is only acceptable if the simulator is perfectly accurate

-> The agent should have good online performance from the start of learning

Key Idea

1) Utilizing data of the system operating under a previous controller, Deep Q-learning from Demonstrations (DQFD) pretrains solely on the demonstration data

2) After pretraining, the agent starts interacting with the domain and updates its network using a mix of demonstration and self generated data

Methodology

1. Pretraining

Goal of the pretraining phase : to learn to imitate the demonstrator with a value function that satisfies the Bellman equation

Update network by applying four losses : 1-step double Q learning loss, n-step double Q- learning loss, supervised large margin classification loss, L2 regularization loss

$$J(Q) = J_{DQ}(Q) + \lambda_1 J_n(Q) + \lambda_2 J_E(Q) + \lambda_3 J_{L2}(Q).$$

The role of supervised large margin classification loss

: Since the demonstration data is covering a narrow part of the state space, the network would update towards the highest of these ungrounded variables
<u>-> supervised large margin classification loss forces the value of other actions to lower than the value of the demonstrator's action</u>

$$J_E(Q) = \max_{a \in A}[Q(s,a) + \underline{l(a_E, a)]} - Q(s, a_E)$$

aE: Action demonstrated by experts

0 if aE=a
Positive value, otherwise

Methodology

2. Online learning

- The agent never overwrites the demonstration data in replay buffer
- Prioritized experience replay: Relative sampling of demonstration versus self-generated data

The probability of sampling a particular transition i: $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$, $p_i = |\delta_i| + \epsilon$

δ_i: the last TD error calculated for this transition i

Methodology

Algorithm 1 Deep Q-learning from Demonstrations.

Inputs: $\mathcal{D}^{replay}$: initialized with demonstration data set, θ: weights for initial behavior network (random), θ': weights for target network (random), τ: frequency at which to update target net, k: number of pre-training gradient updates

for steps $t \in \{1, 2, \ldots k\}$ **do**

  Sample a mini-batch of n transitions from $\mathcal{D}^{replay}$ with prioritization

  Calculate loss $J(Q)$ using target network

  Perform a gradient descent step to update θ

  if $t \bmod \tau = 0$ **then** $\theta' \leftarrow \theta$ **end if**

end for

for steps $t \in \{1, 2, \ldots\}$ **do**

  Sample action from behavior policy $a \sim \pi^{\epsilon Q_\theta}$

  Play action a and observe (s', r).

  Store (s, a, r, s') into $\mathcal{D}^{replay}$, overwriting oldest self-generated transition if over capacity

  Sample a mini-batch of n transitions from $\mathcal{D}^{replay}$ with prioritization

  Calculate loss $J(Q)$ using target network

  Perform a gradient descent step to update θ

  if $t \bmod \tau = 0$ **then** $\theta' \leftarrow \theta$ **end if**

  $s \leftarrow s'$

end for

Experiment

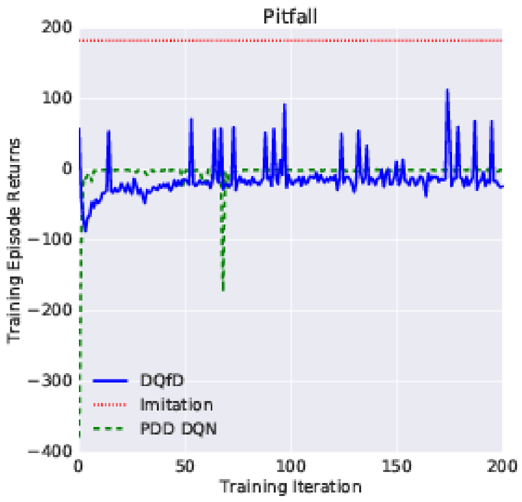
Comparison

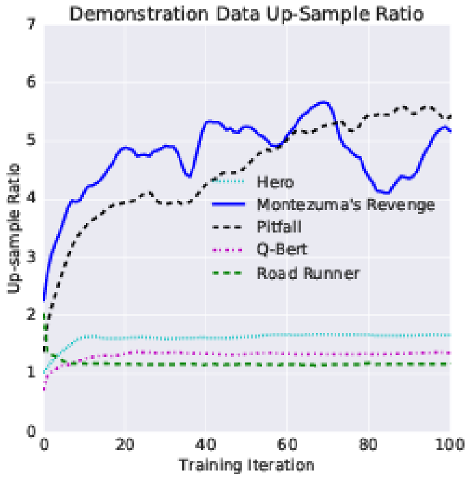
1. Full DQfD algorithm (with human demonstrations)
2. PDD DQN (without any demonstration data)
3. Supervised imitation from demonstration data (without interaction, without TD learning)

Experiment

Results

DQfD leverages the human demonstrations to achieve a higher score than any previously published result





How much more demonstration data was sampled than self-generated data

Experiment

Results
- DQFD with some losses removed -> The agent starts with much lower performance
- Compared with other related algorithms, DQFD outperforms