Reinforcement Learning with Demonstration

Acknowledgement: Most slides were contributed by
施囿維、蔡承倫、郭奎廷、林九州、吳岱霖 etc.,
and organized by 廖唯辰.

Outline

- DQfD
- R2D3
- Decision Transformer

Reference

- "Deep Q-learning from Demonstrations"
  - Published at AAAI 2018
  - Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, Audrunas Gruslys
  - Provided by Google DeepMind
- Slides by Yu-Wei Shih

Reference

- "Making Efficient Use of Demonstrations to Solve Hard Exploration Problems."
  - Published at ICLR 2020
  - Gulcehre, Caglar, Tom Le Paine, Bobak Shahriari, Misha Denil, Matt Hoffman, Hubert Soyer, Richard Tanburn, Steven Kapturowski, Neil Rabinowitz, and Duncan Williams
  - Provided by Google DeepMind
- Slides by 郭奎廷

Reference

- "Decision Transformer: Reinforcement Learning via Sequence Modeling."
  - Published at NeurIPS 2021
  - Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, etc.
  - UC Berkeley, Facebook AI Research, UCLA, OpenAI, Google Brain
- Slides by Tai-Lin Wu

DQfD: Deep Q-learning from Demonstrations

Introduction

- DRL agents ususally have poor performance during the early stage of training
  - This could be problematic for many real world tasks that require fine performance at the beginning
- DQfD leverages small demonstration datasets to greatly speed up early phase training process

Baseline: PDD DQN

- Prioritized Experience Replay + Dueling Network + DDQN
- Based on this structure, DQfD makes several improvements to leverage demonstration data

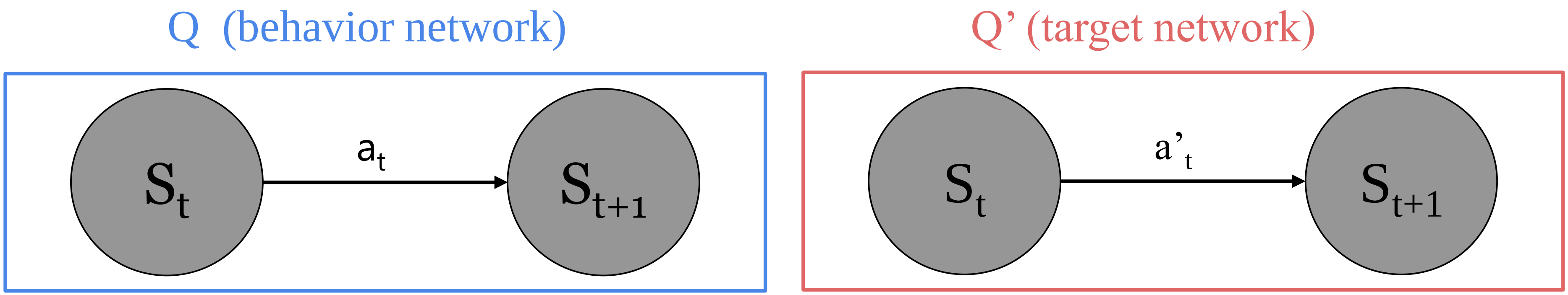
Note:

Dueling Network is used in experiment but the authors didn't mention this as a requirement in DQfD.

DQfD: Four Losses

- DQfD uses 4 losses to update:
  - 1-step double Q-learning loss
  - n-step double Q-learning loss
  - Supervised large margin classification loss
  - L2 regularization loss

1-step Double Q-learning Loss

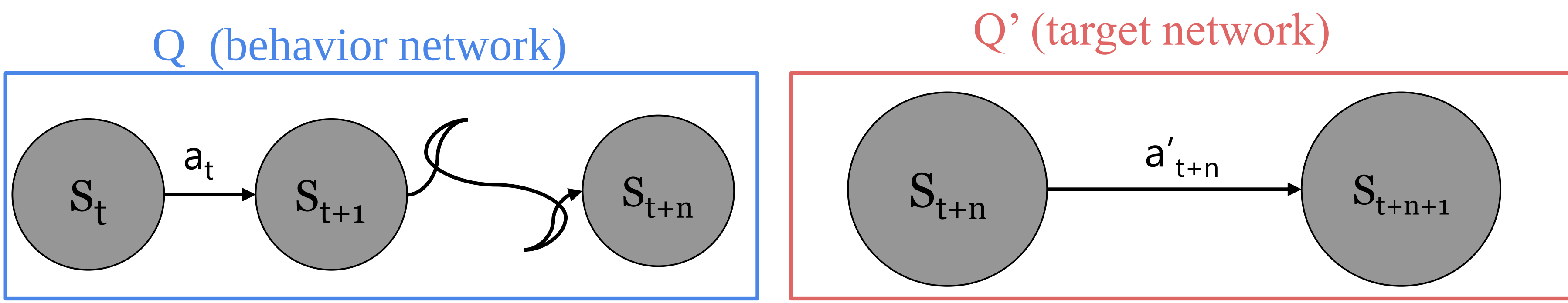


$$y_t = R(s_t, a_t) + \gamma Q'\left(s_{t+1}, \mathrm{argmax}_a Q(s_{t+1}, a)\right)$$

$$J_{DQ}(Q) = \left(y_t - Q(s_t, a_t)\right)^2$$

- Same as the loss in DDQN method.

n-step Double Q-learning Loss



$$y_t = R_t + \gamma R_{t+1} + \ldots + \gamma^{n-1} R_{t+n-1} + \gamma^n Q'\left(s_{t+n}, argmax_a Q\left(s_{t+n}, a\right)\right)$$

$$where\ R_t = R\left(s_t, a_t\right)$$

$$J_n(Q) = \left(y_t - Q\left(s_t, a_t\right)\right)^2$$

- Adding n-step returns helps propagate the values of the expert’s trajectory to all the earlier states, leading to better pre-training.
- This paper uses 10-step loss (n=10)

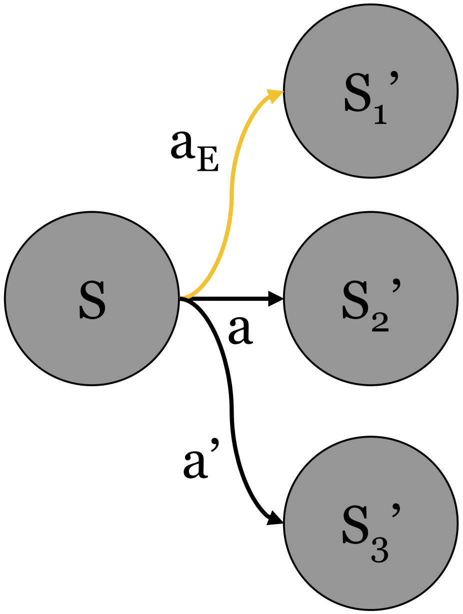
I-Chen Wu

Superviesd Large Margin Classification Loss

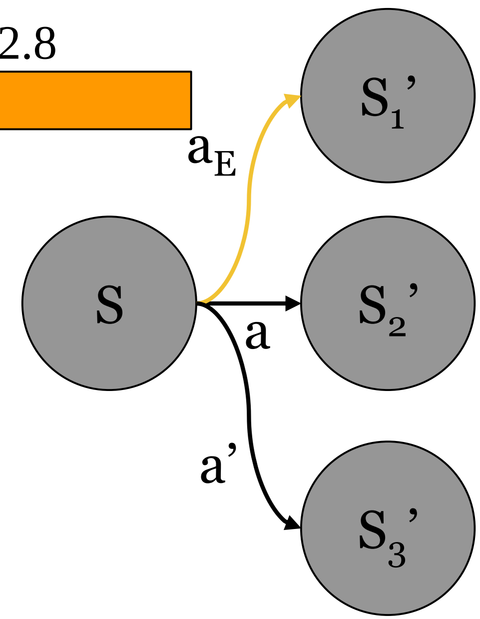
- Purpose: make the Q value of a_E at least a margin larger than Q value of any other actions.
  - a_E is demonstration data
- Encourage the agent to follow demonstration data instead of choosing other values.

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

$$l(a_E, a) = \begin{cases} 0 & if\ a = a_E \\ positive\ value & if\ a \neq a_E \end{cases}$$



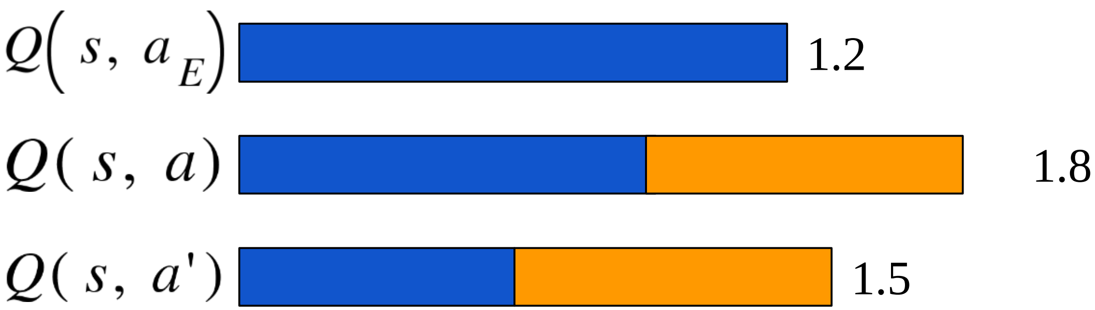
Superviesd Large Margin Classification Loss

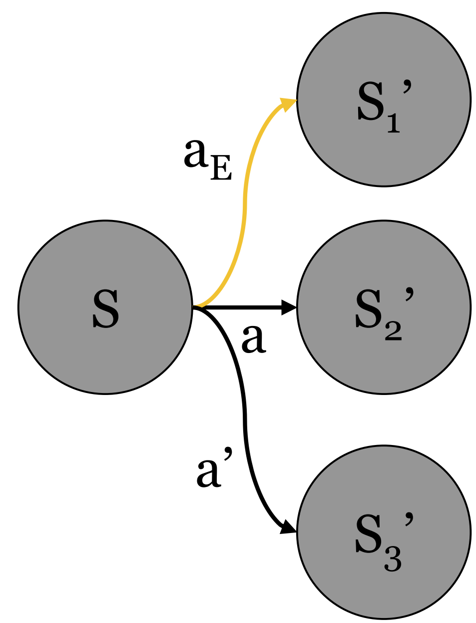


$$J_E(Q) = \boxed{\overbrace{\max_{a \in A}\left[Q(s, a) + l(a_E, a)\right]}^{2.8}} - \boxed{\overbrace{Q(s, a_E)}^{1.0}} > 0$$

$$l(a_E, a) = \begin{cases} 0 & if\ a = a_E \\ 0.8 & if\ a \neq a_E \end{cases}$$

Superviesd Large Margin Classification Loss

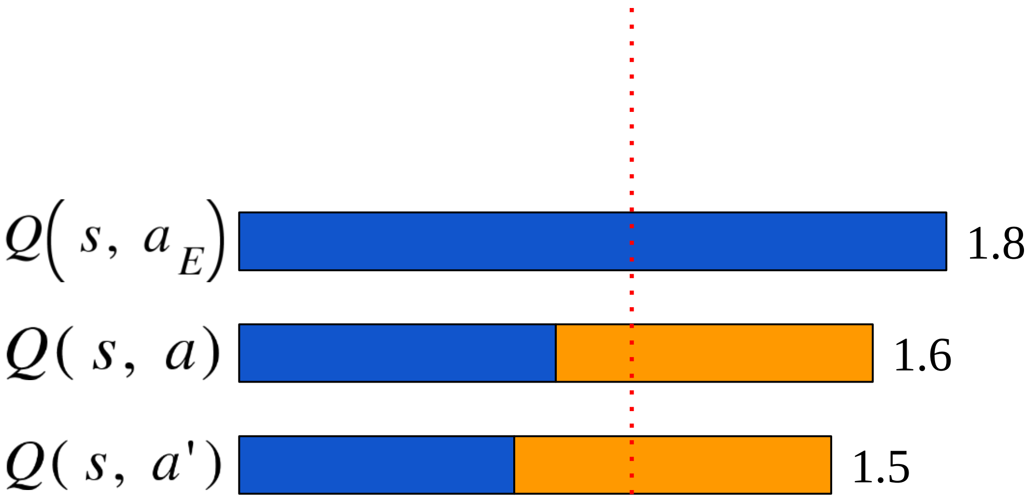




$$J_E(Q) = \overset{1.8}{\boxed{\max_{a \in A}\left[Q(s, a) + l(a_E, a)\right]}} - \overset{1.2}{\boxed{Q(s, a_E)}} > 0$$

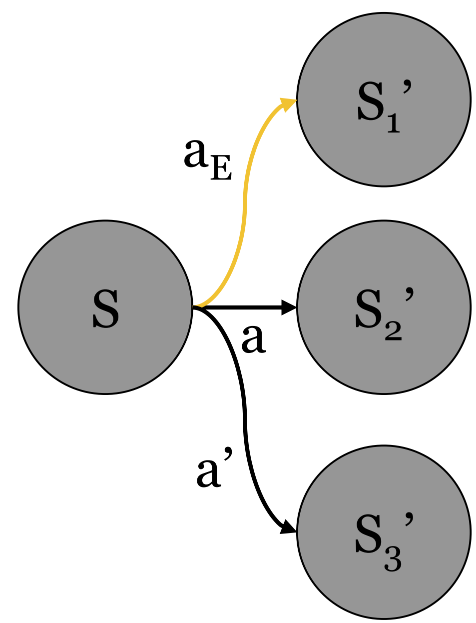
$$l(a_E, a) = \begin{cases} 0 & if\ a = a_E \\ 0.8 & if\ a \neq a_E \end{cases}$$

Superviesd Large Margin Classification Loss



$$J_E(Q) = \overbrace{\boxed{\max_{a \in A}\left[Q(s, a) + l(a_E, a)\right]}}^{1.8} - \overbrace{\boxed{Q(s, a_E)}}^{1.8} = 0$$

$$l(a_E, a) = \begin{cases} 0 & if\ a = a_E \\ 0.8 & if\ a \neq a_E \end{cases}$$



L2 Regularization Loss

- Use L2 Norm of all weights and bias in model as a regularization loss to avoid over-fitting.

DQfD: Four Losses

- Combining all four losses:

$$J(Q) = \underbrace{J_{DQ}(Q) + \lambda_1 J_n(Q) + \lambda_2 J_E(Q) + \lambda_3 J_{L2}(Q)}_{\text{When updating by demonstration data.}}$$

In this paper: $\lambda_1 = 1,\ \lambda_2 = 1,\ \lambda_3 = 10^{-5}$

DQfD: Four Losses

- Combining all four losses:

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

When updating by self-generated data.

In this paper: $\lambda_1 = 1,\ \lambda_2 = 0,\ \lambda_3 = 10^{-5}$

Prioritized Experience Replay (PER)

- Sample important trajectories more frequently.
- Usually importance = |TD error|.

$$P(i) = \frac{p_i^{\alpha}}{\sum_k p_k^{\alpha}}, \; p_i = |\delta_i| + \varepsilon$$

$$w_i = \left(\frac{1}{N} \times \frac{1}{P(i)} \right)^{\beta}$$

P(i) = probability to be sampled for trajectory i

δ = TD error

N = size of replay buffer

α = priority exponent

β = importance sampling exponent

Prioritized Experience Replay (PER)

- Sample important trajectories more frequently.
- Usually importance = |TD error|.

use different constant $\varepsilon_a \varepsilon_d$ for self-play data and demonstration data ($\varepsilon_a = 0.001$, $\varepsilon_d = 1$)

$$P(i) = \frac{p_i^{\alpha}}{\sum_k p_k^{\alpha}}, \quad p_i = |\delta_i| + \varepsilon$$

$$w_i = \left(\frac{1}{N} \times \frac{1}{P(i)} \right)^{\beta}$$

P(i) = probability to be sampled for trajectory i

δ = TD error

N = size of replay buffer

α = priority exponent

β = importance sampling exponent

Pseudocode

k=750, 000 in this paper

- Demonstration data are placed in replay buffer at the beginning of training.
- When replay buffer is full, replace old self-play data but never remove demonstration data.

Pretraining Phase

Interacting with environment

Algorithm 1 Deep Q-learning from Demonstrations.

```
Inputs: 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 ∈ {1, 2, ... k} do
  Sample a mini-batch of n transitions from D^replay
  with prioritization
  Calculate loss J(Q) using target network
  Perform a gradient descent step to update θ
  if t mod τ = 0 then θ′ ← θ end if
end for
for steps t ∈ {1, 2, ...} do
  Sample action from behavior policy a ~ π^{εQ_θ}
  Play action a and observe (s′, r).
  Store (s, a, r, s′) into D^replay, overwriting oldest
  self-generated transition if over capacity
  Sample a mini-batch of n transitions from D^replay
  with prioritization
  Calculate loss J(Q) using target network
  Perform a gradient descent step to update θ
  if t mod τ = 0 then θ′ ← θ end if
  s ← s′
end for
```

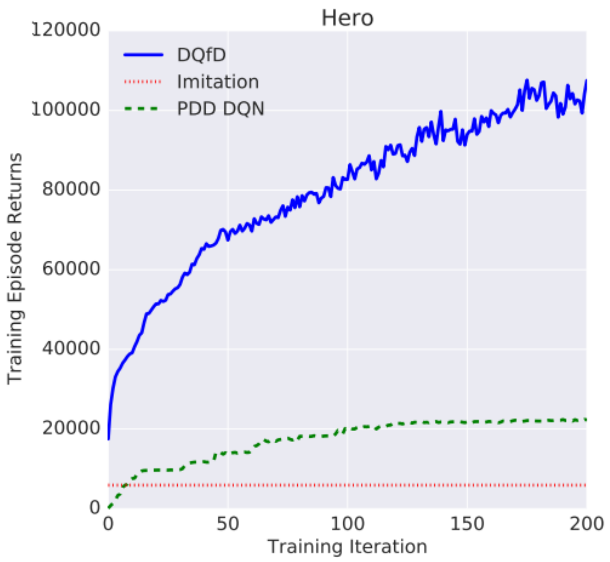
$$J(Q) = J_{DQ}(Q) + \lambda_1 J_n(Q) + \lambda_2 J_E(Q) + \lambda_3 J_{L2}(Q)$$

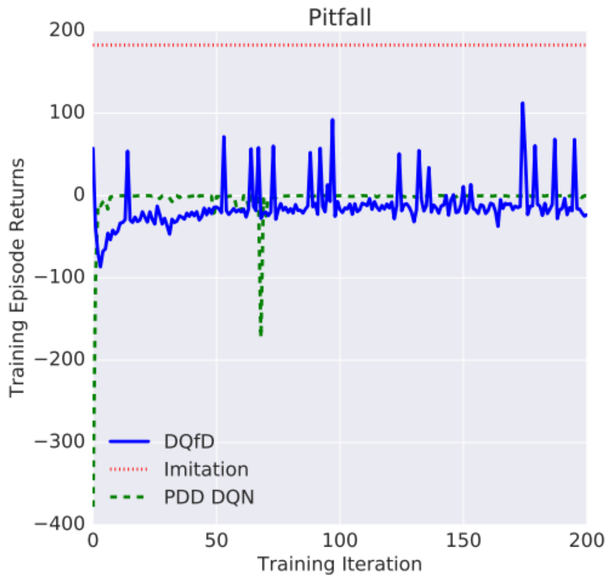
Note: in this paper: $k = 750000$

Compare with Baseline PDD DQN

- Full DQfD with human demonstration data
- PDD DQN without any demonstration data
- Supervised imitation learning from demonstration data without any interaction with environments.

Result


Hero
Training Episode Returns
Training Iteration
DQfD
Imitation
PDD DQN
0
20000
40000
60000
80000
100000
120000
0
50
100
150
200


Pitfall
Training Episode Returns
Training Iteration
DQfD
Imitation
PDD DQN
200
100
0
−100
−200
−300
−400
0
50
100
150
200

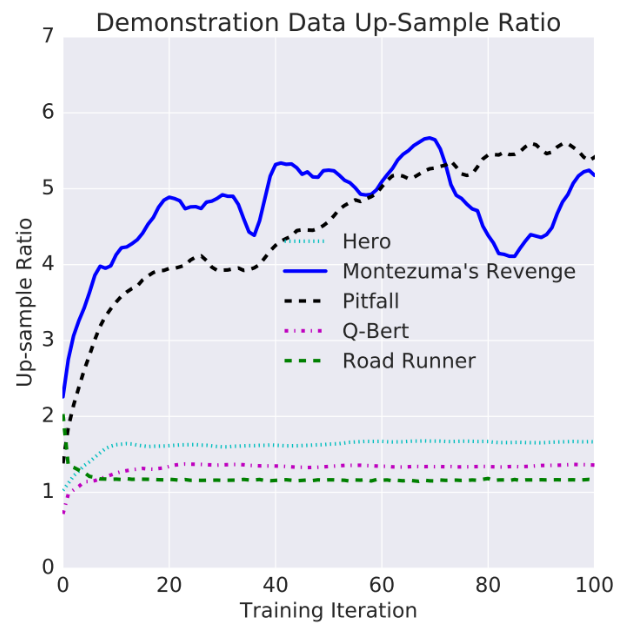

Road Runner
Training Episode Returns
Training Iteration
DQfD
Imitation
PDD DQN
0
10000
20000
30000
40000
50000
60000
0
50
100
150
200

Demonstration Data Ratio in a Mini-batch

- In environments with sparse rewards, the constants ε in PER become specifically important.
- Big constant for demonstration data help DQfD agents overcome environments with such rewards by following demonstration data.

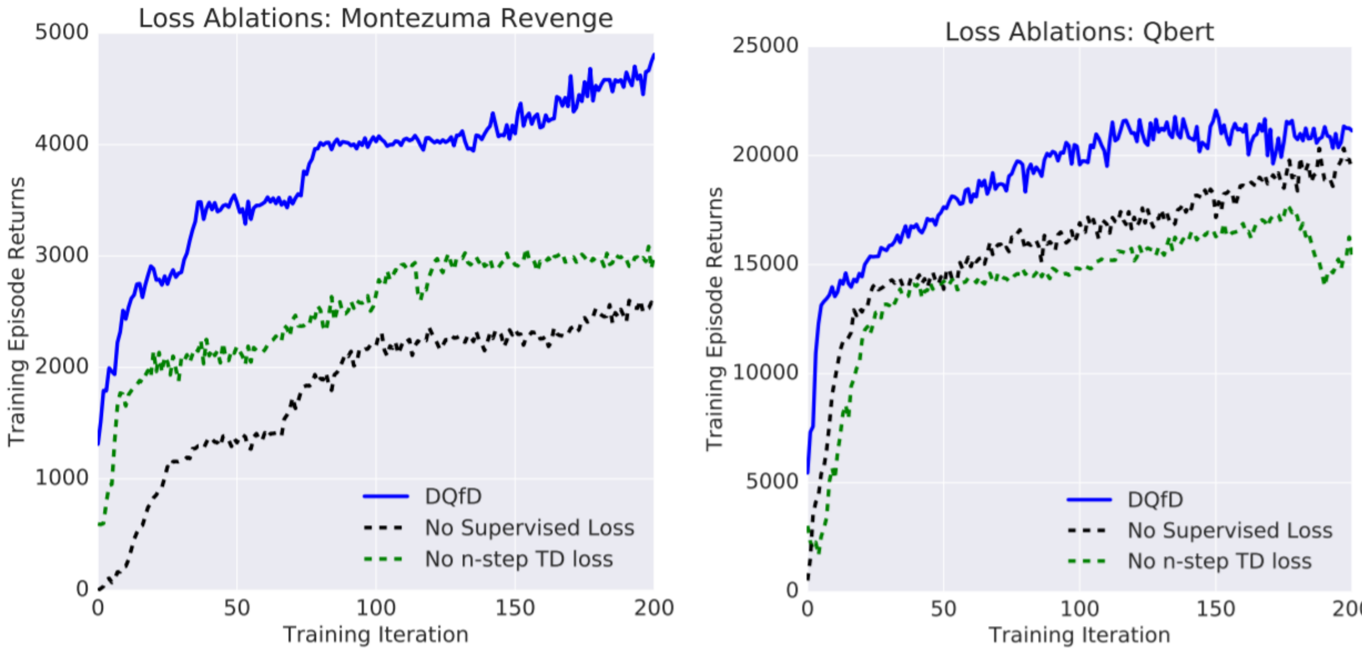
$$p_i = |\delta_i| + \varepsilon$$

($\varepsilon_a = 0.001$, $\varepsilon_d = 1$)



The plot shows how much more frequently the demonstration data was sampled than if data were sampled uniformly, for 5 different games

Ablation Study



Conclusion

- DQfD has good performance from the beginning of the training due to pretraining by demonstration data
- Adjusting the ratio between demonstration data and self-play data in each mini-bach by Prioritized Experience Reply
- Combining supervised loss and TD loss to follow demonstrators' actions

R2D3

Recurrent Replay Distributed DQN from Demonstration

(R2D2: Recurrent Replay Distributed DQN)

Introduction

- Sparse reward problem is a challenge for RL methods
  - Common approaches
    - Intrinsic motivation, reward shaping, or curriculum tasks
  - Drawbacks:
    - Do not scale well
    - Lead to unexpected behavior
    - Difficult to specify
- Learning from demonstrations has proven to be an effective strategy

Introduction

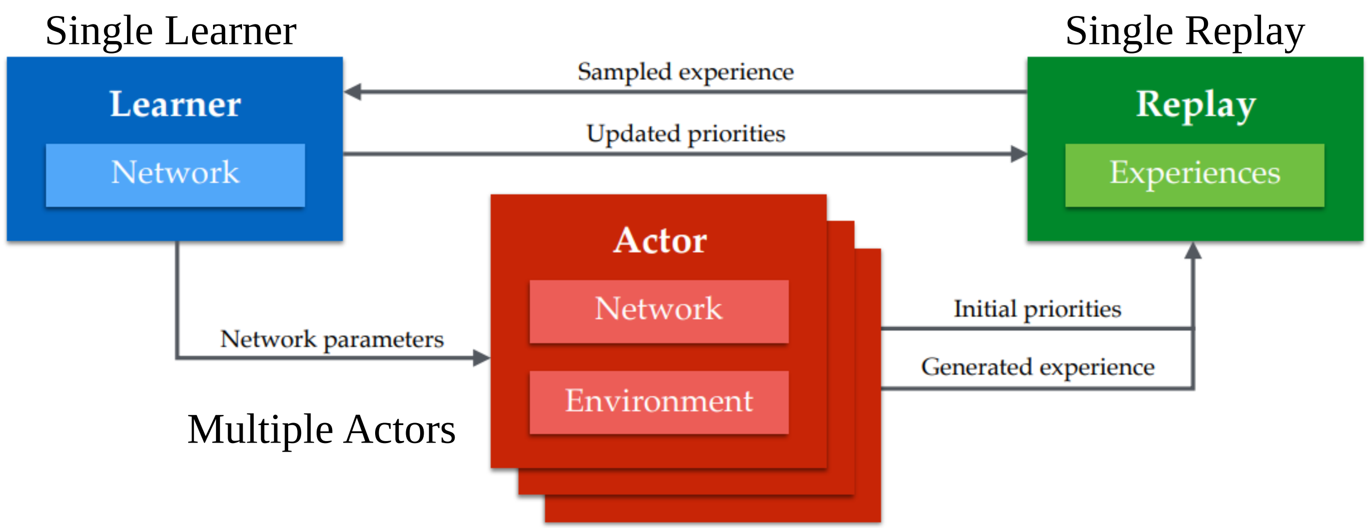
- Three aspects to make learning from demonstrations challenging
  - Sparse rewards
  - Partial observability
  - Highly variable initial conditions
- The demonstrations cannot account for all possible configurations

Background – DQfD

- Leverage expert demonstration datasets to speed up training
- Encourage the agent to follow demonstration data
  - Use supervised imitation loss in training
- Four losses (weighted sum):
  - 1-step double Q-learning loss
  - n-step double Q-learning loss
  - Supervised large margin classification loss
  - L2 regularization loss

Background – Ape-X

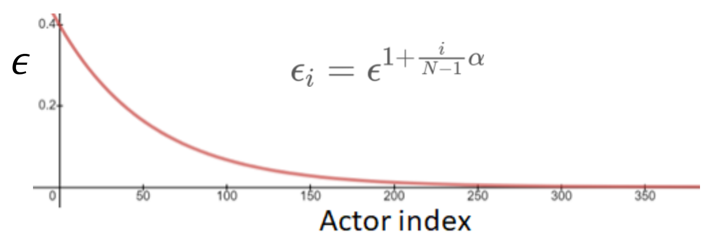
- Leverage prioritized experience replay (PER) on distributed architecture



Horgan, Dan, et al. "Distributed prioritized experience replay." *arXiv preprint arXiv:1803.00933* (2018).

Background – Ape-X

- Problem
  - Different games require a significantly different degree of exploration
- Solution
  - Give the different actors different exploration policies
    - E.g., different ϵ values for ϵ-greedy
  - Sample with priority to get the most useful data



Horgan, Dan, et al. "Distributed prioritized experience replay." *arXiv preprint arXiv:1803.00933* (2018).

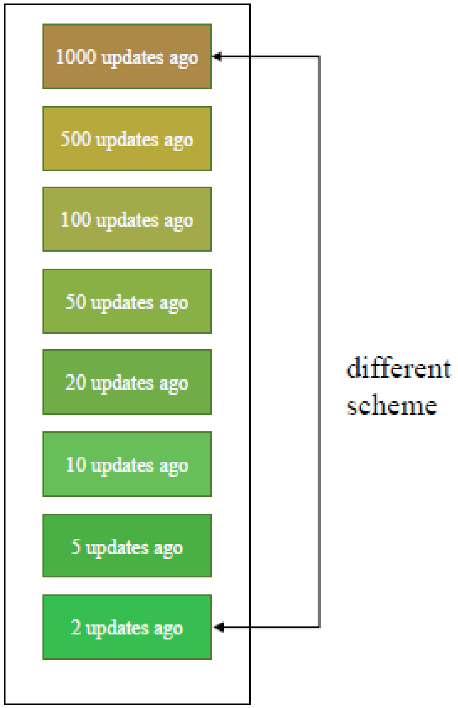
Background – R2D2

- R2D2: Recurrent Replay Distributed DQN
- Build on Ape-X
- Train an RNN from experience replay
  - Partial observability
  - Better representation learning

Steven Kapturowski, et al. "Recurrent Experience Replay in Distributed Reinforcement Learning." *International Conference on Learning Representations*. 2019.

Training Recurrent RL Agent with Experience Replay

- Stored state
  - Storing the recurrent state in replay
    - Use it to initialize the network at training time
  - New problems
    - Representational drift
      - Out of date representation



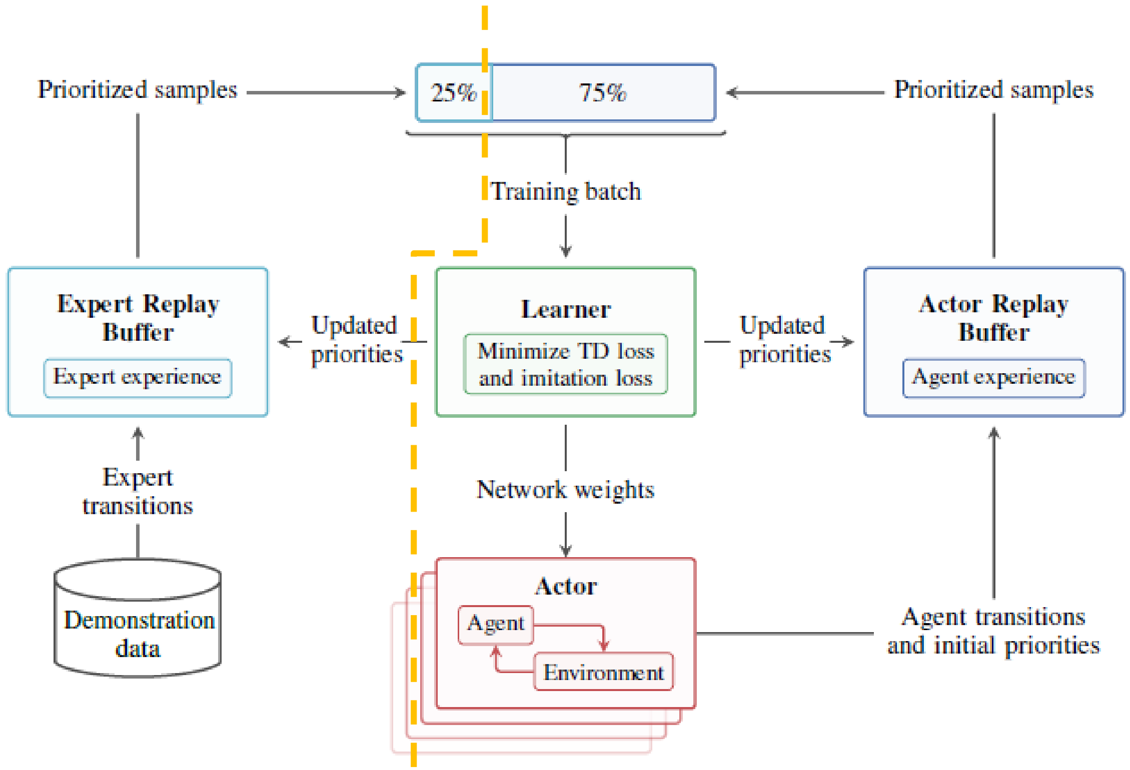
Steven Kapturowski, et al. "Recurrent Experience Replay in Distributed Reinforcement Learning." *International Conference on Learning Representations*. 2019.

Training Recurrent RL Agent with Experience Replay

- Burn in
  - Allow the network a ‘burn in period’ by using a portion of the replay sequence
  - Allow the network to partially recover from a poor start state

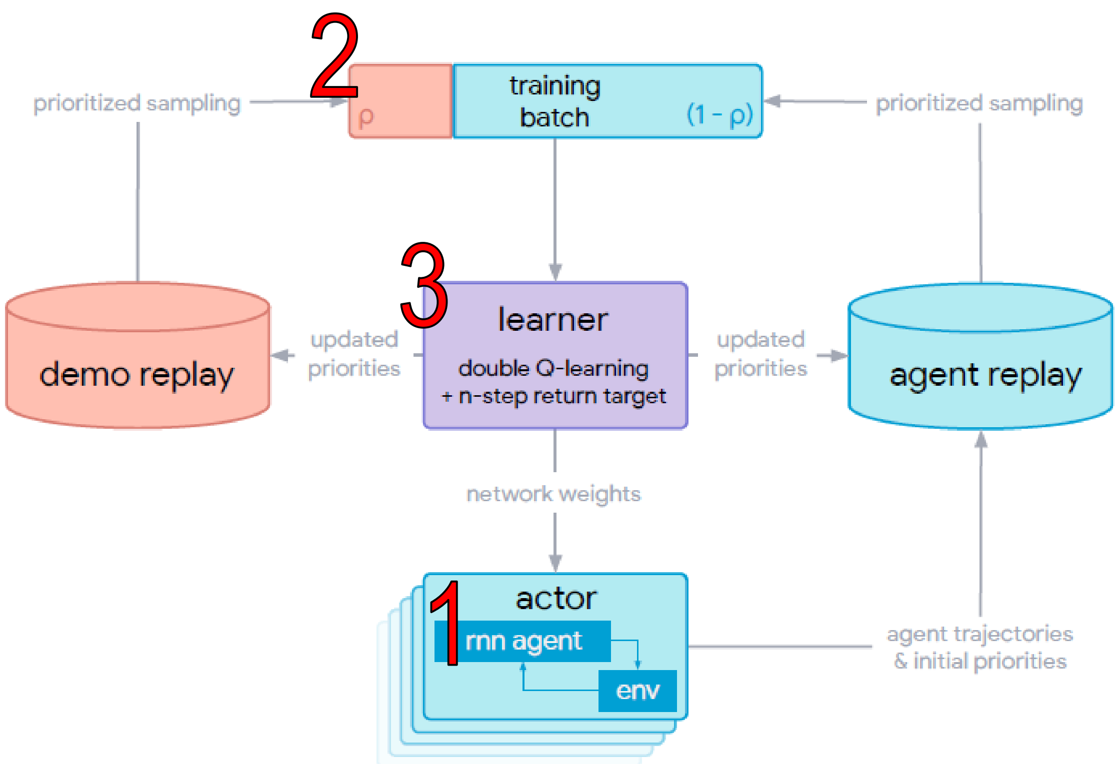
Steven Kapturowski, et al. "Recurrent Experience Replay in Distributed Reinforcement Learning." *International Conference on Learning Representations*. 2019.

Background – Ape-X DQfD



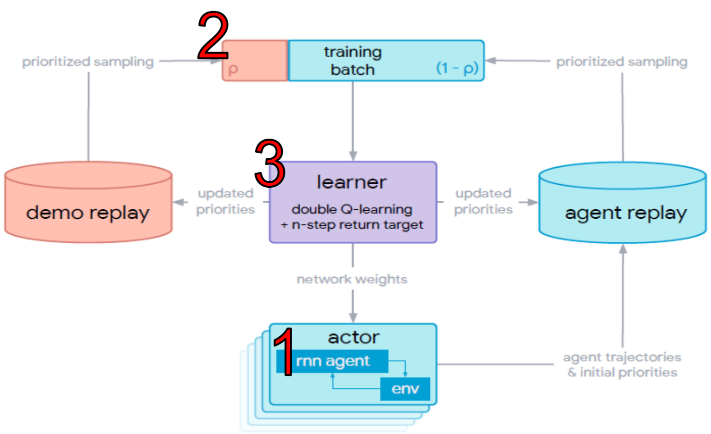
Pohlen, Tobias, et al. "Observe and look further: Achieving consistent performance on atari." *arXiv preprint arXiv:1805.11593* (2018).

R2D3


2
ρ
training
batch
(1 - ρ)
prioritized sampling
prioritized sampling
3
learner
double Q-learning
+ n-step return target
demo replay
agent replay
updated
priorities
updated
priorities
network weights
actor
1
rnn agent
env
agent trajectories
& initial priorities

R2D3

- Three main changes
  1. Use R2D2 to replace Ape-X
     - Deal with partial observable environment
  2. Tune the demo ratio ρ
     - Surprisingly, find that the optimal demo ratio is very small (1/256)
  3. Remove imitation loss
     - Use expert demos to bias the agent's own autonomous exploration



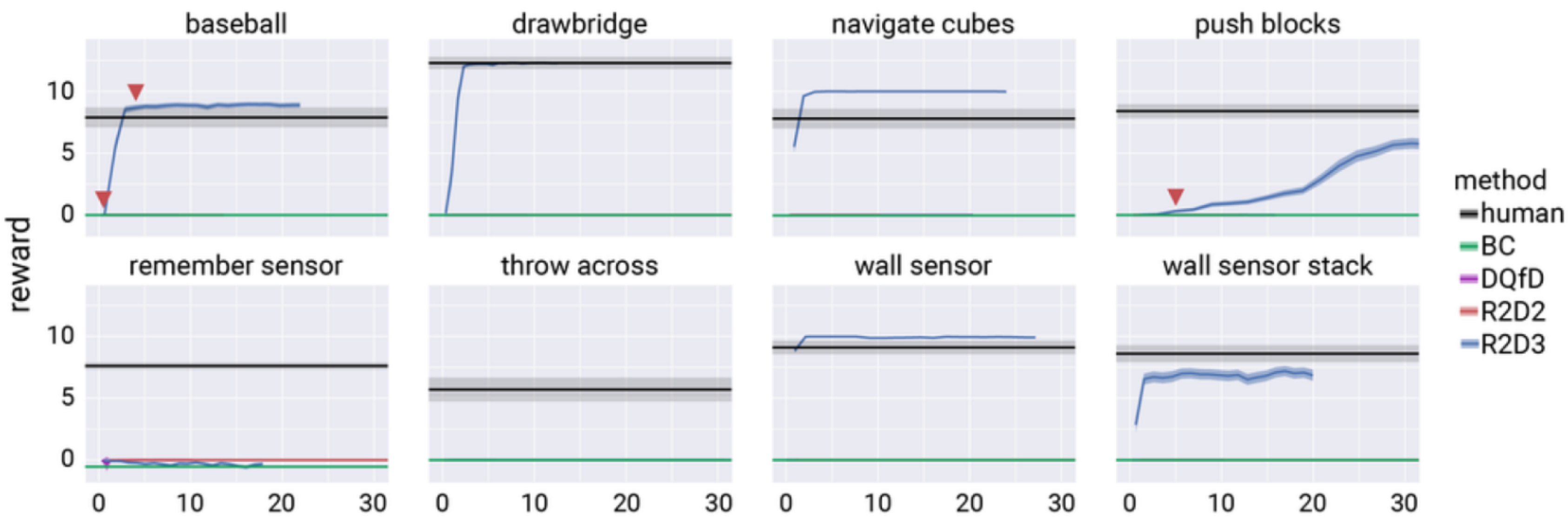
R2D3 – Hard-Eight Task Suite

- A procedurally-generated 3D world
- Common properties
  - Sparse reward
  - Partially observable
  - Highly variable initial conditions

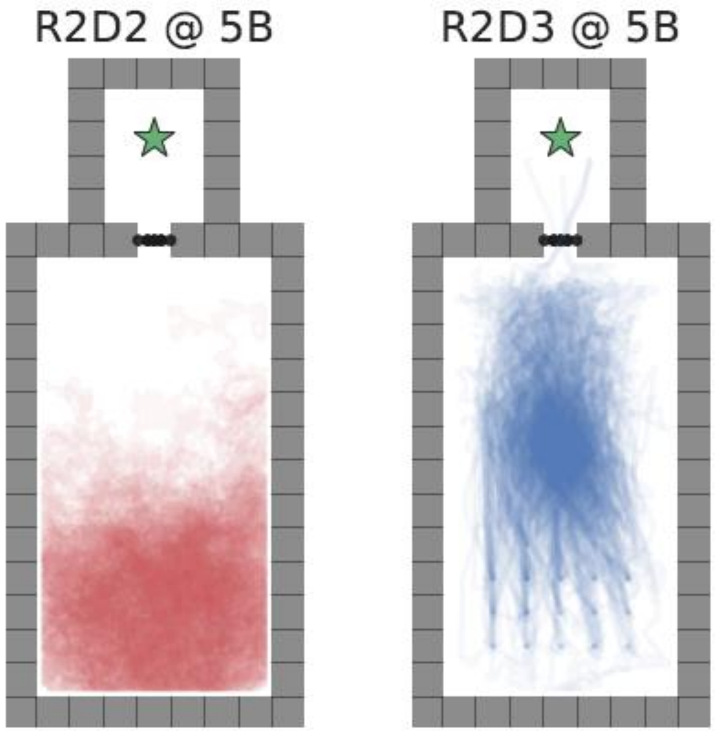
R2D3 – Hard-Eight Task Suite

- Example: Push Blocks
- Step
  1. Check the color of sensor
  2. Push a block whose color matches the sensor into the recess
  3. Collect the apple

Experiment – R2D3 Performance in Hard-Eight suite

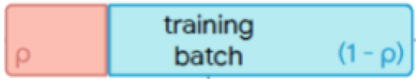


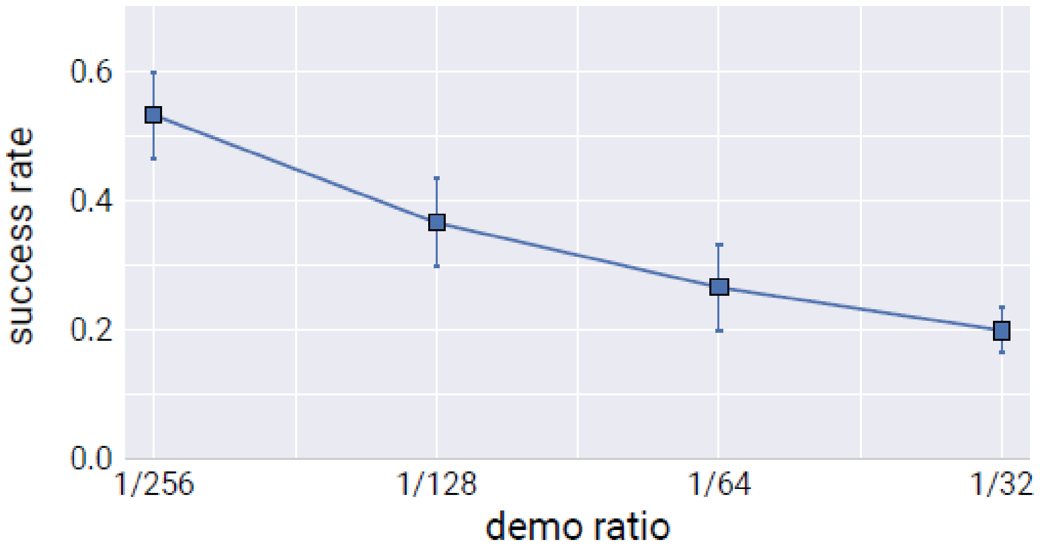
Experiment – R2D3 Guided Exploration Behavior



The red indicators mark analysis points

Experiment – R2D3 Demo Ratio Tuning





Conclusion

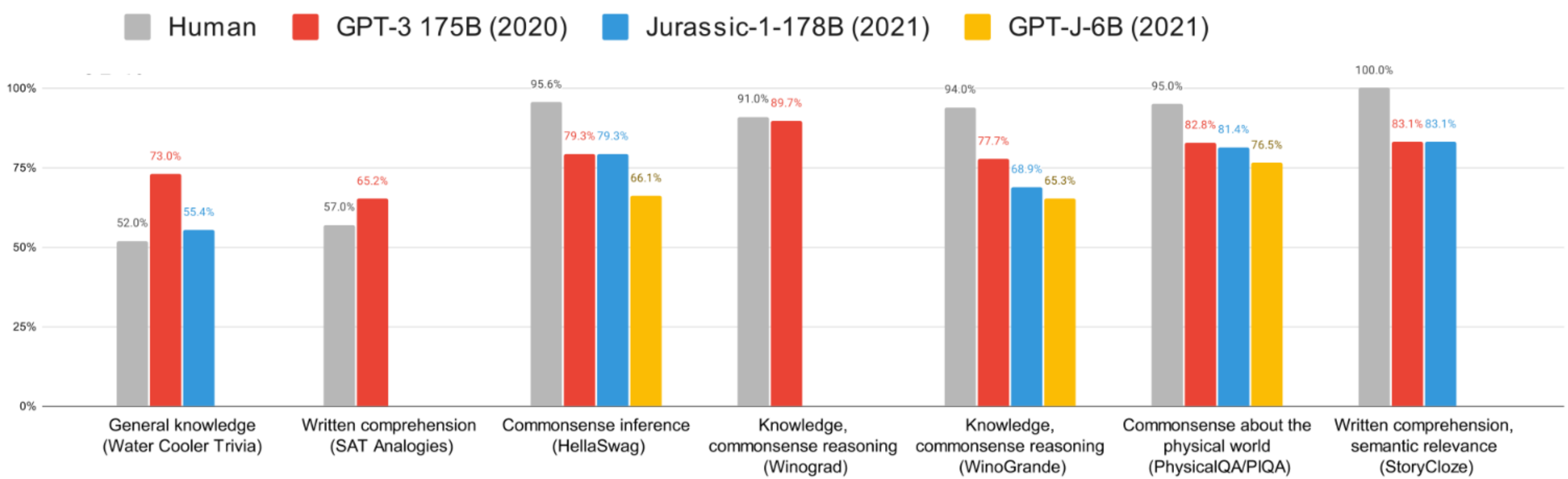
- Design to make efficient use of demonstrations to learn
  - Partially observable environments
  - Sparse rewards
  - Highly variable initial conditions
- Use the expert demonstrations in a way that guides the agent's own autonomous exploration

Decision Transformer

Decision Transformer: Reinforcement Learning
via Sequence Modeling

Introduction

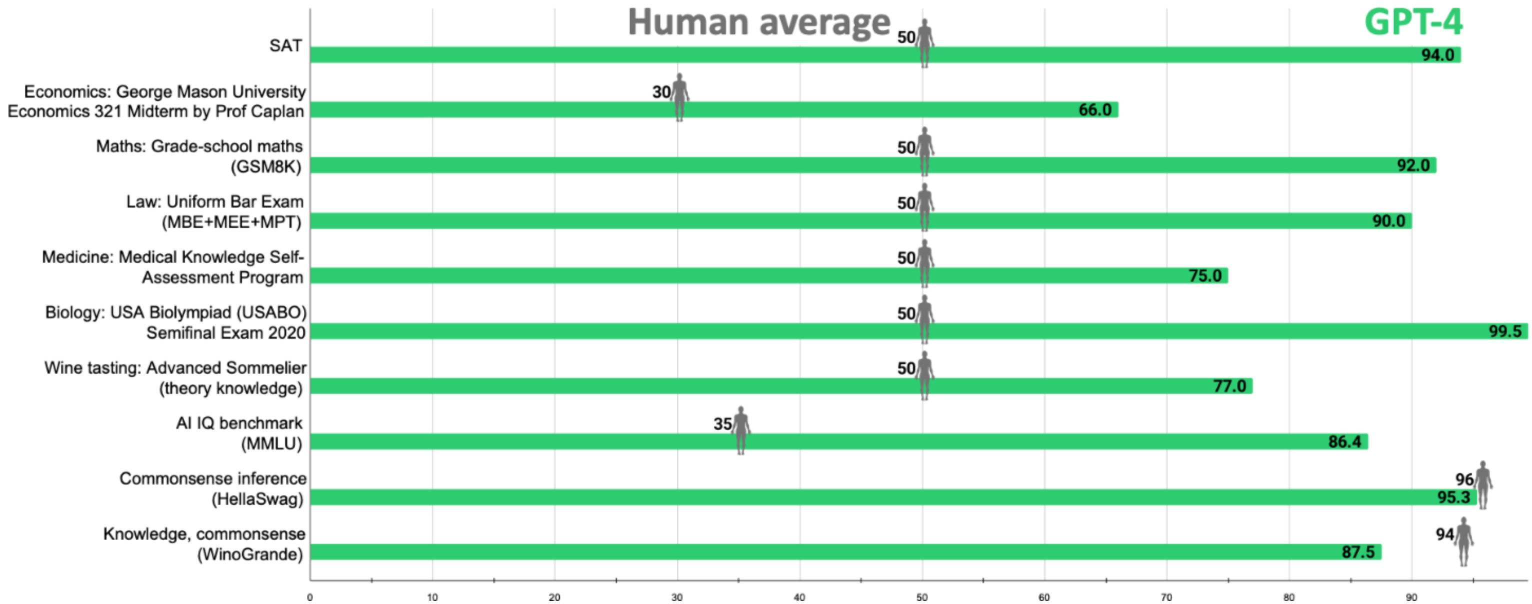
- Recent work has shown transformers can model distributions of semantic concepts, and associated advances in language modeling such as GPT-x and BERT have achieved impressive results. (GPT3/3.5)



https://lifearchitect.ai/iq-testing-ai/

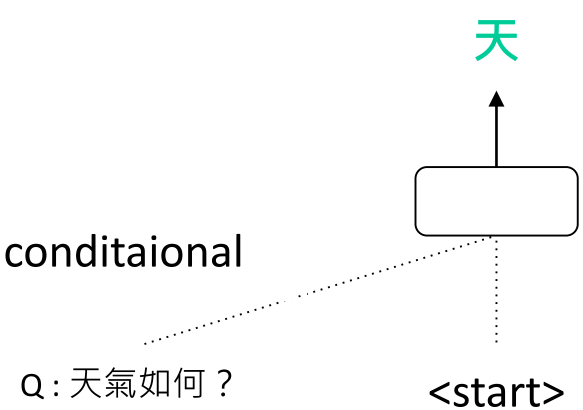
Introduction

- Recent work has shown transformers can model distributions of semantic concepts, and associated advances in language modeling such as GPT-x and BERT have achieved impressive results. (GPT4)



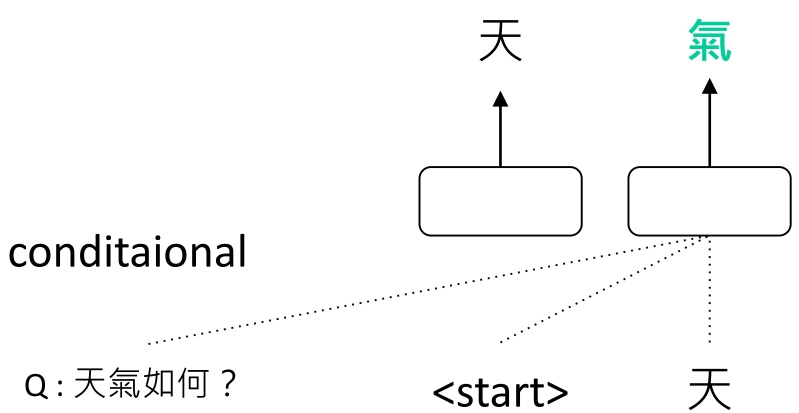
Introduction

- This paper present Decision Transformer as:
  - An architecture that casts the problem of RL as conditional sequence modeling (Generative Transformer).



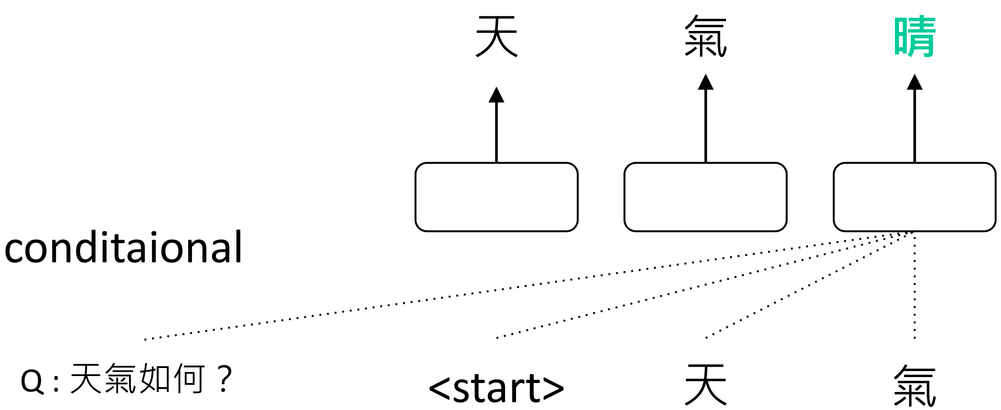
Introduction

- This paper present Decision Transformer as:
  - An architecture that casts the problem of RL as conditional sequence modeling (Generative Transformer).



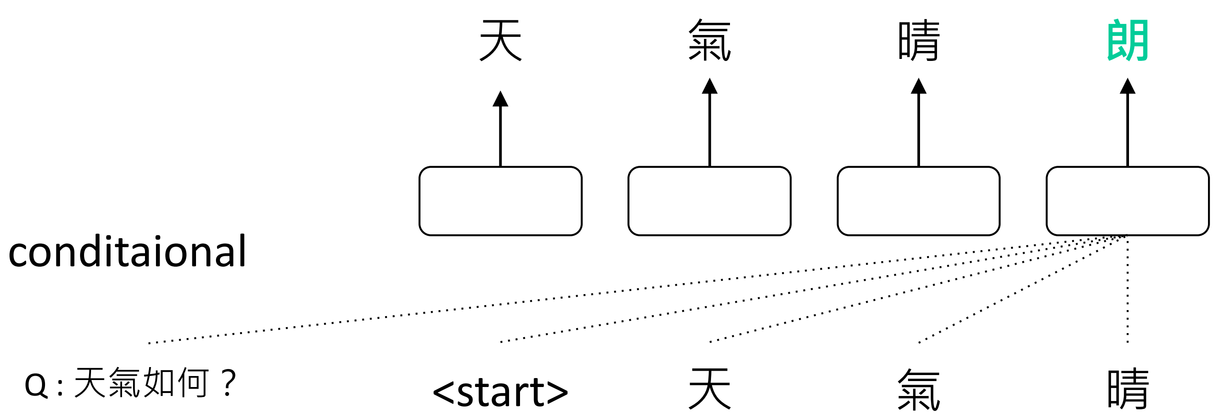
Introduction

- This paper present Decision Transformer as:
  - An architecture that casts the problem of RL as conditional sequence modeling (Generative Transformer).



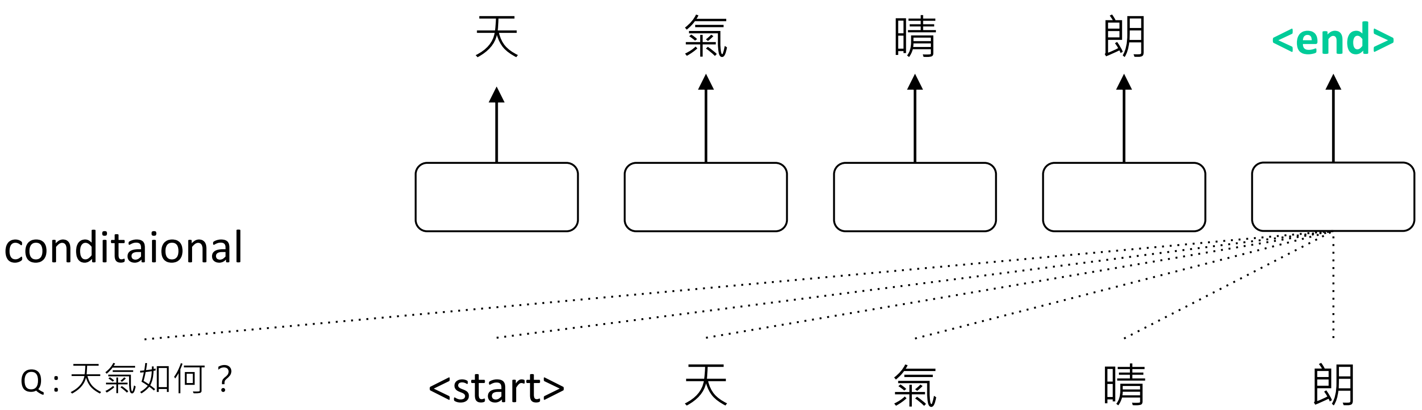
Introduction

- This paper present Decision Transformer as:
  - An architecture that casts the problem of RL as conditional sequence modeling (Generative Transformer).



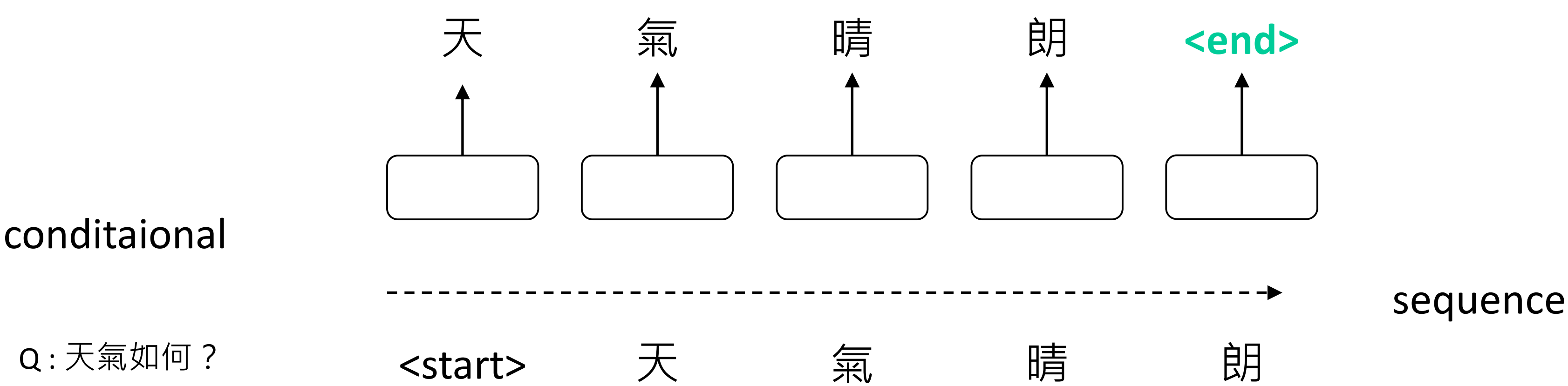
Introduction

- This paper present Decision Transformer as:
  - An architecture that casts the problem of RL as conditional sequence modeling (Generative Transformer).



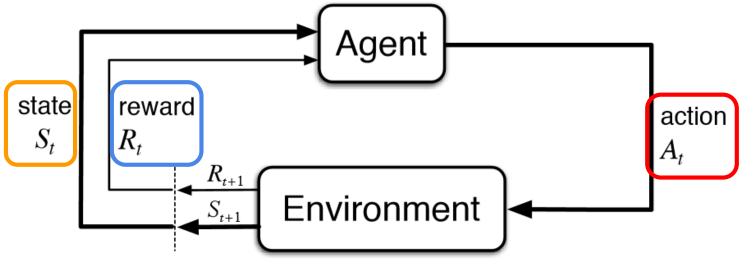
Introduction

- This paper present Decision Transformer as:
  - An architecture that casts the problem of RL as conditional sequence modeling (Generative Transformer).



Introduction

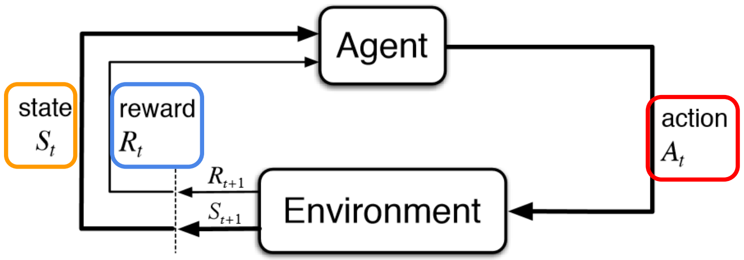
- This paper present Decision Transformer as:
  - An architecture that casts the problem of RL as conditional sequence modeling (Generative Transformer).



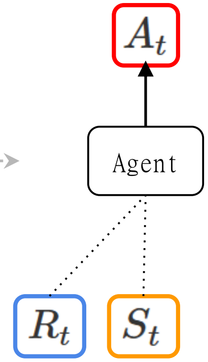
RL Problem

Introduction

- This paper present Decision Transformer as:
  - An architecture that casts the problem of RL as conditional sequence modeling (Generative Transformer).

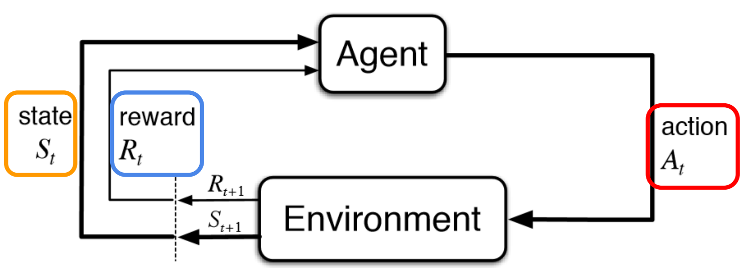


RL Problem

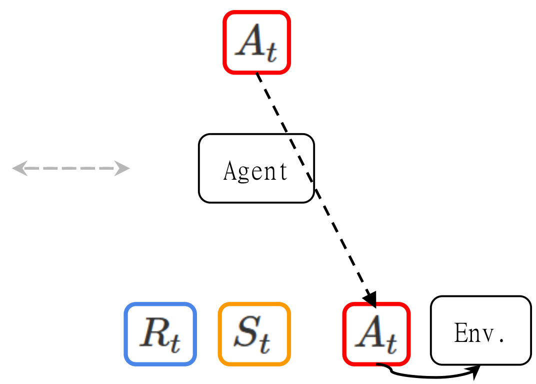


Introduction

- This paper present Decision Transformer as:
  - An architecture that casts the problem of RL as conditional sequence modeling (Generative Transformer).

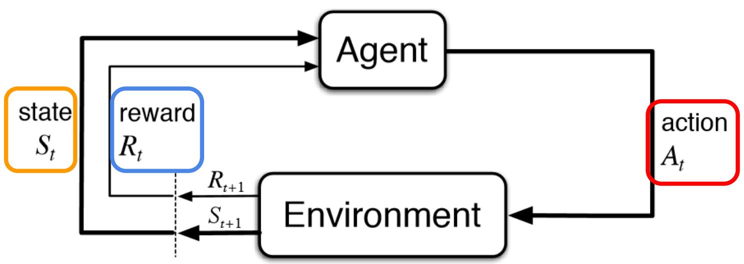


RL Problem

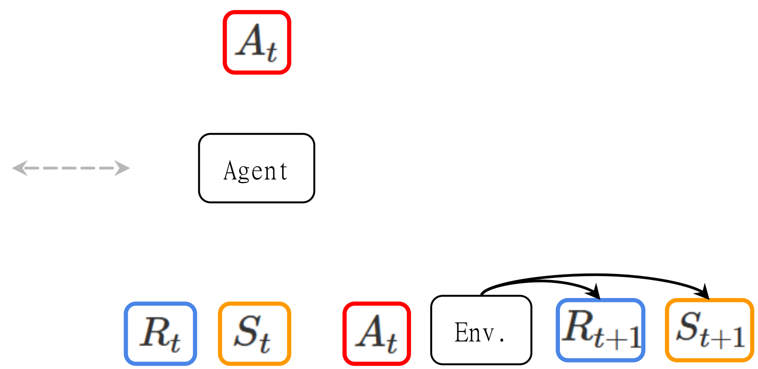


Introduction

- This paper present Decision Transformer as:
  - An architecture that casts the problem of RL as conditional sequence modeling (Generative Transformer).

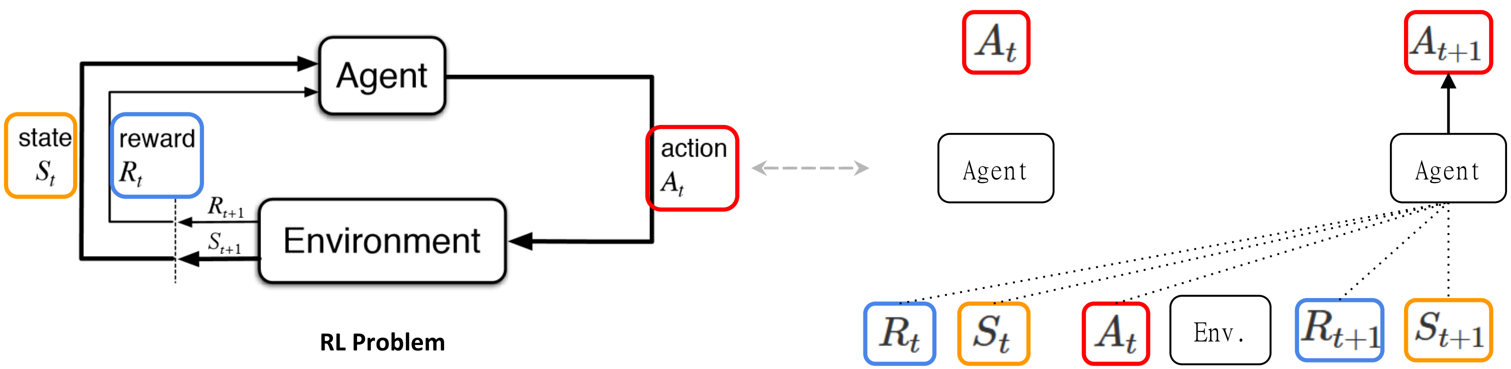


RL Problem



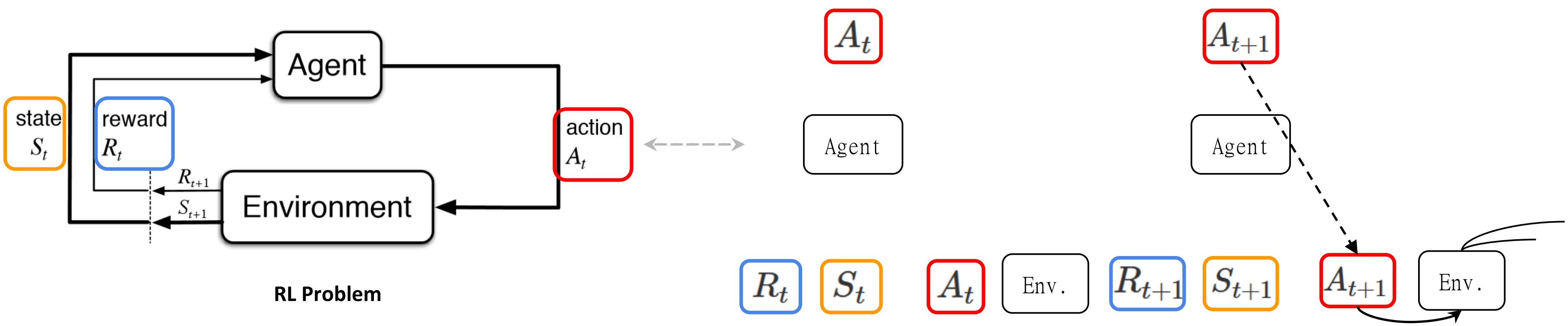
Introduction

- This paper present Decision Transformer as:
  - An architecture that casts the problem of RL as conditional sequence modeling (Generative Transformer).



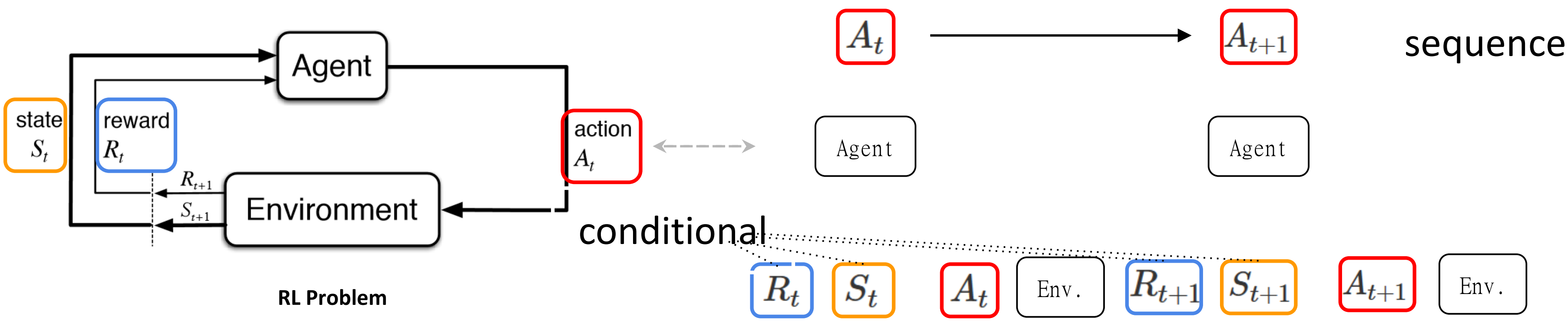
Introduction

- This paper present Decision Transformer as:
  - An architecture that casts the problem of RL as conditional sequence modeling (Generative Transformer).



Introduction

- This paper present Decision Transformer as:
  - An architecture that casts the problem of RL as conditional sequence modeling (Generative Transformer).

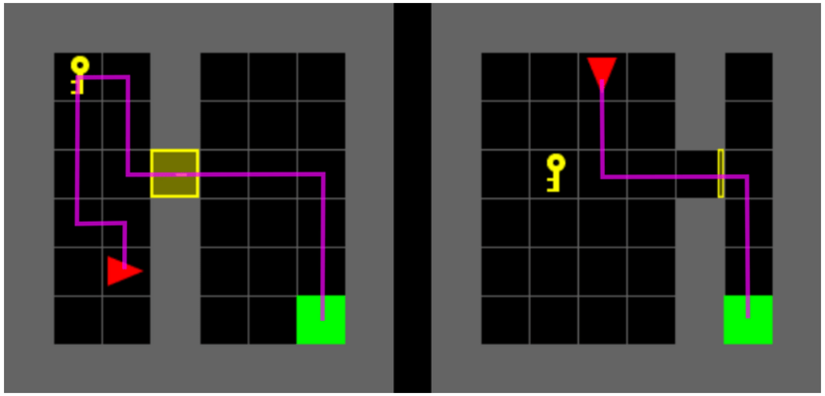


Introduction

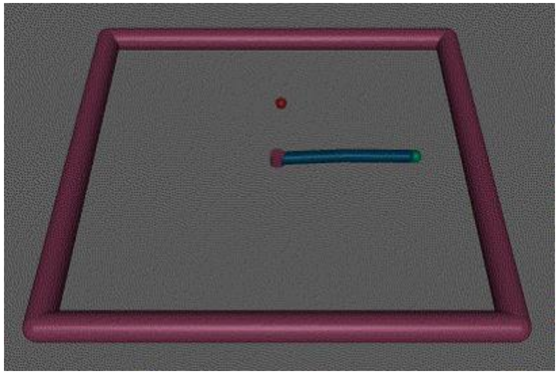
- Decision Transformer **matches** or **exceeds** the performance of state-of-the-art **model-free offline RL baselines** on Atari (2021), OpenAI Gym.

Motivation

- **Sparse rewards**: which refers to infrequent or insufficient reward signals that make it difficult to guide learning
- Common methods used to address the issue:
  - Reward shaping, curriculum learning, etc.



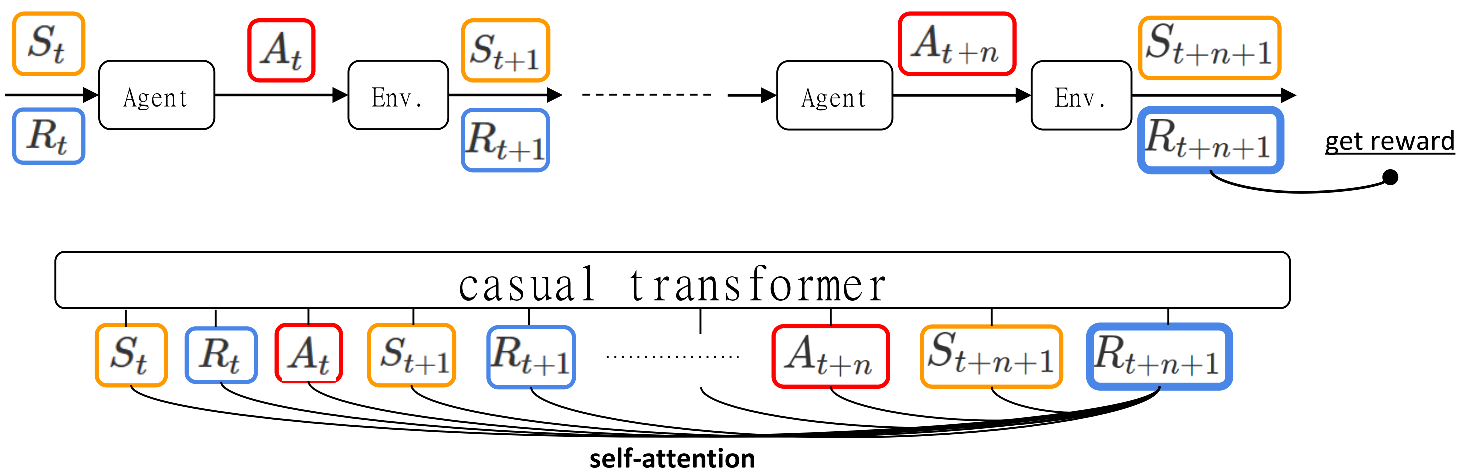
DoorKey task



2D Reacher (OpenAI Gym)

Motivation (sparse reward)

- Transformers can perform credit assignment directly via self-attention, in contrast to Bellman backups which slowly propagate rewards and are prone to “distractor” signals.
  - This can enable transformers to still **work effectively in the presence of sparse or distracting rewards.**



Motivation: (short-sighted behaviors)

- Compared to when we solve RL problems under the **Model-free** condition in the past.
  - Decision Transformer allow us to bypass the need for bootstrapping to propagate returns
  - Avoids the need for discounting future rewards, as typically done in TD-learning, which can induce undesirable short-sighted behaviors.

Simplest temporal-difference learning algorithm: TD(0)

- Update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma \boxed{V(S_{t+1})} - V(S_t))$$

Update each value with the terminal result

Offline Reinforcement Learning

- Learning in a Markov decision process (MDP) described by the tuple (S, A, P, R).
  - S state, A actions, P state transition probability, R reward
- Instead of obtaining data via environment interactions
- The goal in reinforcement learning is to learn a policy which maximizes the **expected return** $\mathbb{E}\left[\sum_{t=1}^{T} r_t\right]$ in an MDP

Trajectory Representation

- The key desiderata in trajectory representation are :
  - should enable transformers to learn meaningful patterns
  - should be able to conditionaly generate action at test time
  - generate actions based on **future desired returns**, rather than past rewards

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

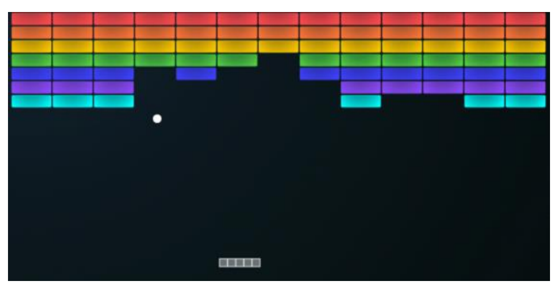
states $s \in \mathcal{S}$,
actions $a \in \mathcal{A}$

$$\widehat{R}_t = \sum_{t'=t}^{T} r_{t'}$$

How to define return-to-go?

1. We **specify** a target return based on our desired performance.
2. After executing the generated action, we decrement the target return by the achieved reward and obtain the next state.

Return-to-go conditioning		
	90 Breakout	6000 HalfCheetah
	2500 Qbert	3600 Hopper
	20 Pong	5000 Walker
	1450 Seaquest	50 Reacher

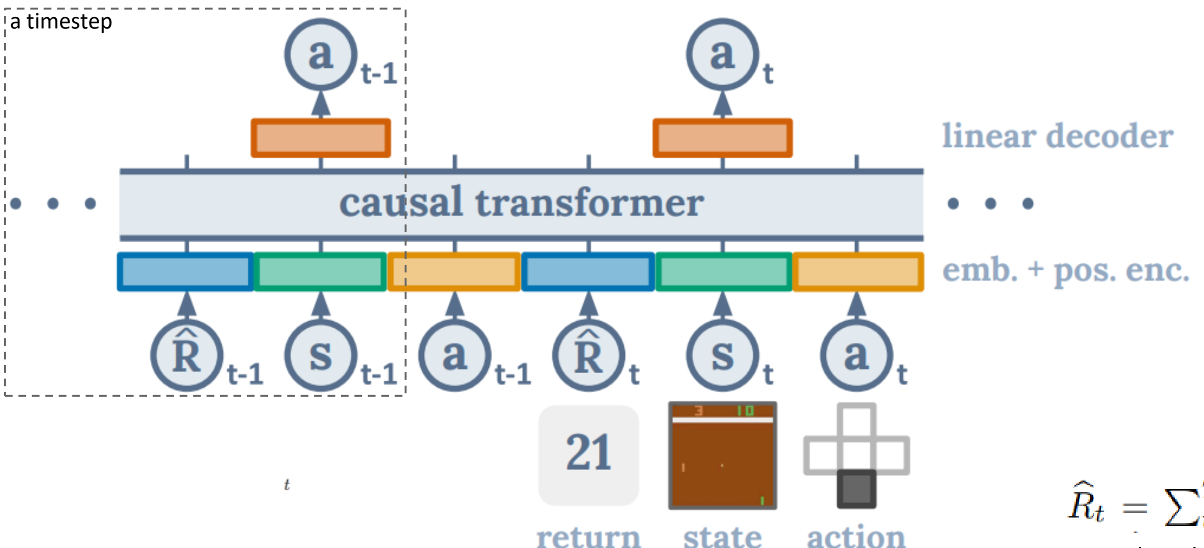
$\widehat{R}_1$ $\longrightarrow$  $\longrightarrow$ $\widehat{R}_2$

90 85

get rewards: 5

Architecture

- Feed the last K timesteps into Decision Transformer, for a total of 3K tokens (for each modality include: return-to-go, state, action)

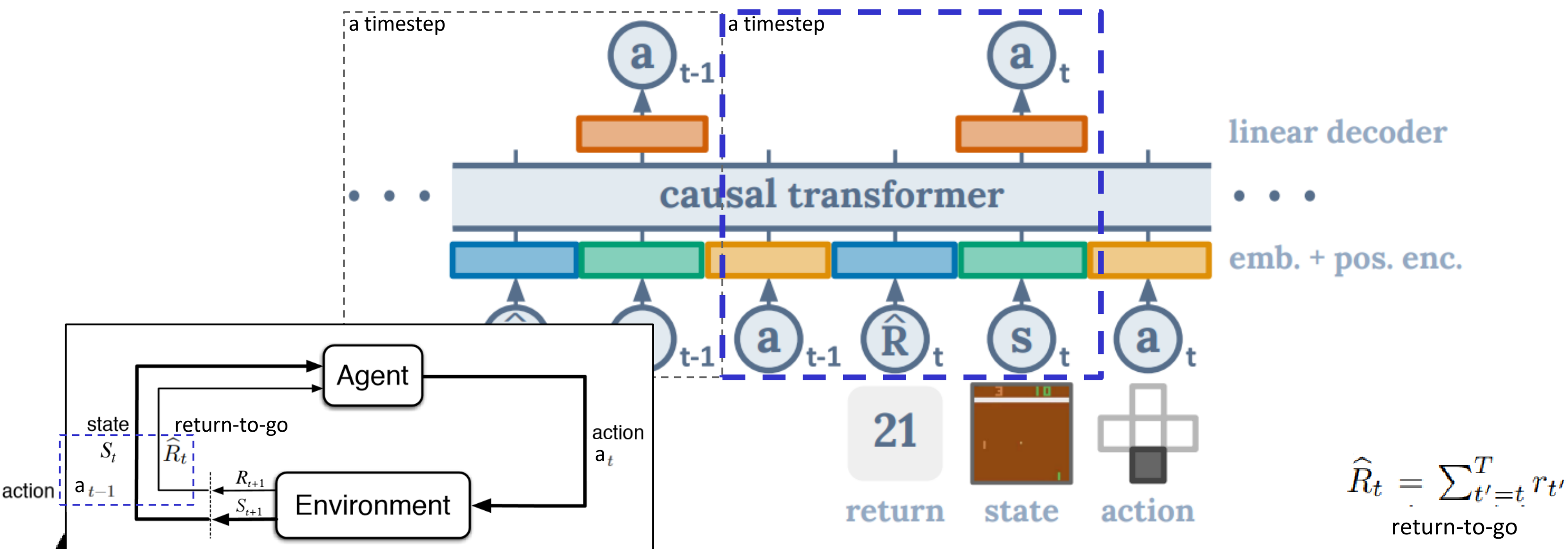


$$\hat{R}_t = \sum_{t'=t}^{T} r_{t'}$$

return-to-go

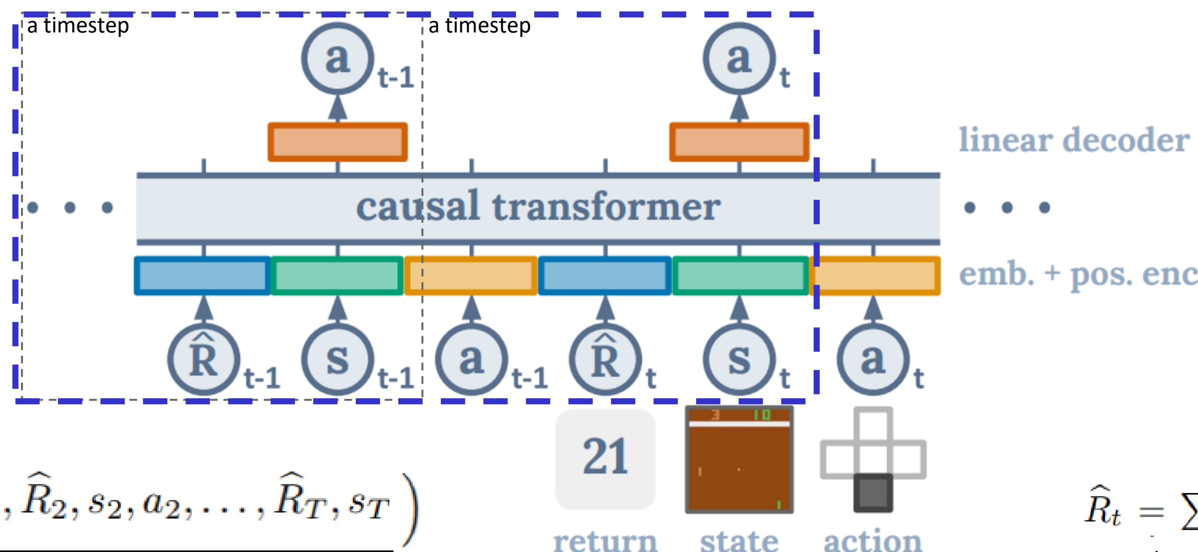
Architecture

- Feed the last K timesteps into Decision Transformer, for a total of 3K tokens (for each modality include: return-to-go, state, action)



Architecture

- Feed the last K timesteps into Decision Transformer, for a total of 3K tokens (for each modality include: return-to-go, state, action)



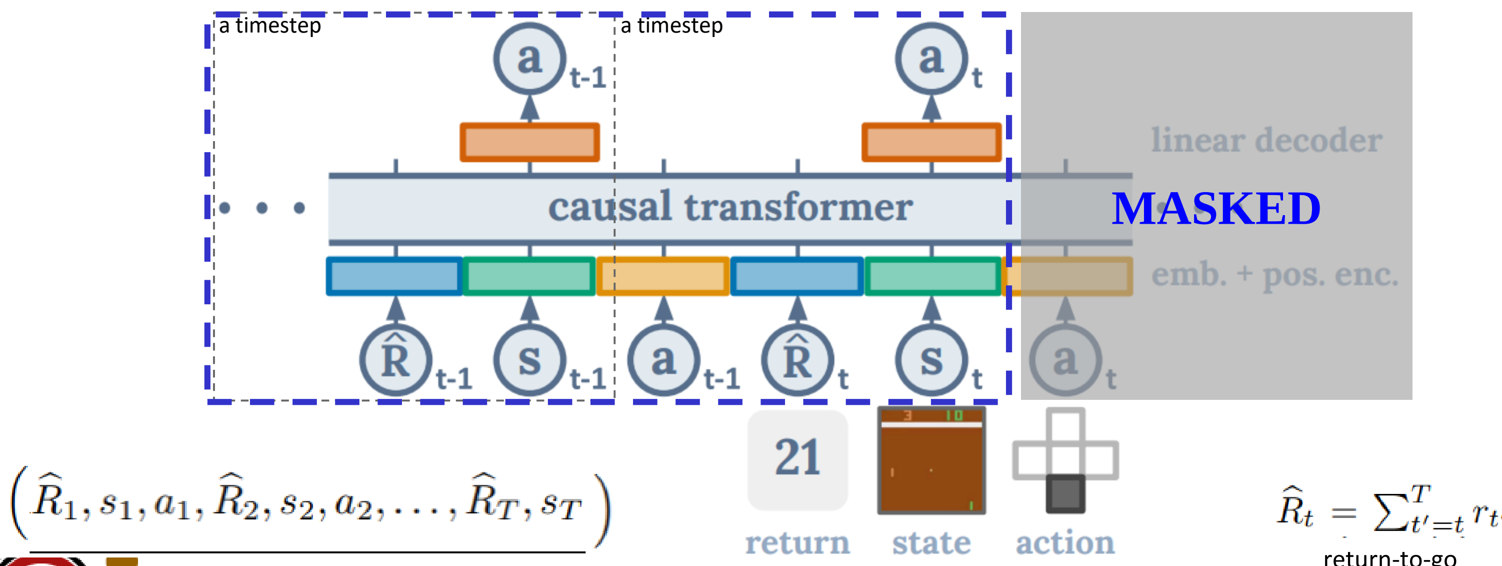
$$\left(\widehat{R}_1, s_1, a_1, \widehat{R}_2, s_2, a_2, \dots, \widehat{R}_T, s_T\right)$$

$$\widehat{R}_t = \sum_{t'=t}^{T} r_{t'}$$

return-to-go

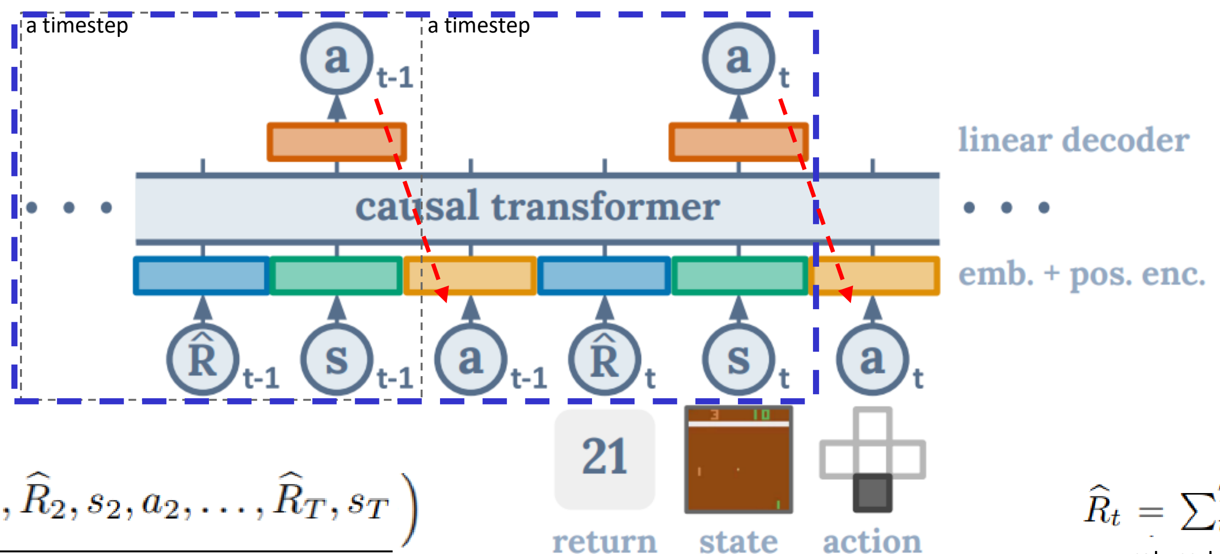
Architecture

- Feed the last K timesteps into Decision Transformer, for a total of 3K tokens (for each modality include: return-to-go, state, action)



Architecture

- Feed the last K timesteps into Decision Transformer, for a total of 3K tokens (for each modality include: return-to-go, state, action)



$$\left(\widehat{R}_1, s_1, a_1, \widehat{R}_2, s_2, a_2, \ldots, \widehat{R}_T, s_T\right)$$

$$\widehat{R}_t = \sum_{t'=t}^{T} r_{t'}$$

return-to-go

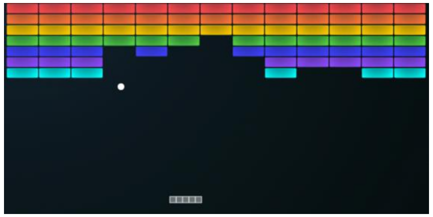
Experiments (Atari)

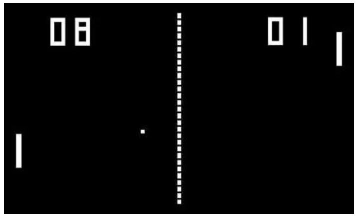
- Training Data: 1% of all samples in the DQN-replay dataset. (500 thousand)

Context lengths of K=30 for Decision Transformer (expect K=50 for Pong)

Game	DT (Ours)	CQL	QR-DQN	REM	BC
Breakout	**267.5 ± 97.5**	211.1	21.1	32.1	138.9 ± 61.7
Qbert	15.1 ± 11.4	**104.2**	1.7	1.4	17.3 ± 14.7
Pong	106.1 ± 8.1	**111.9**	20.0	39.1	85.2 ± 20.0
Seaquest	**2.4 ± 0.7**	1.7	1.4	1.0	2.1 ± 0.3

BC : Behavior Cloning
CQL : Conservative Q-Learning
REM : Random Ensemble Mixture





CQL: https://proceedings.neurips.cc/paper_files/paper/2020/file/0d2b2061826a5df3221116a5085a6052-Paper.pdf

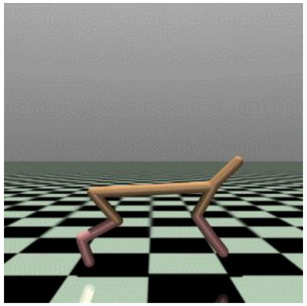
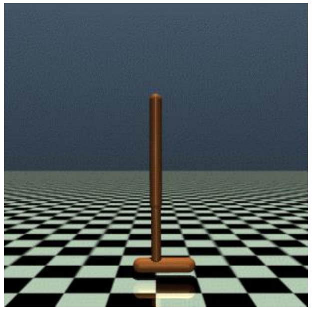
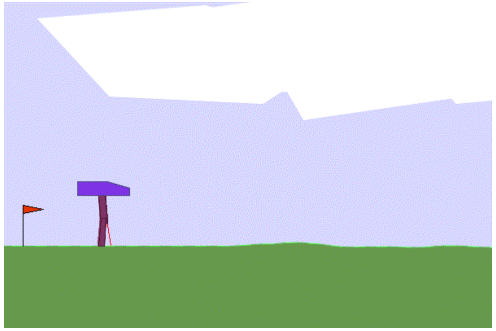
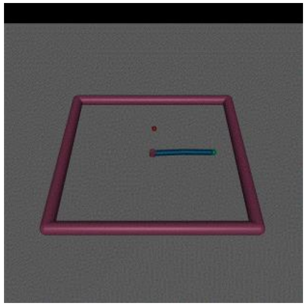
Experiments (OpenAI Gym)

- Training Data: D4RL benchmark

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**

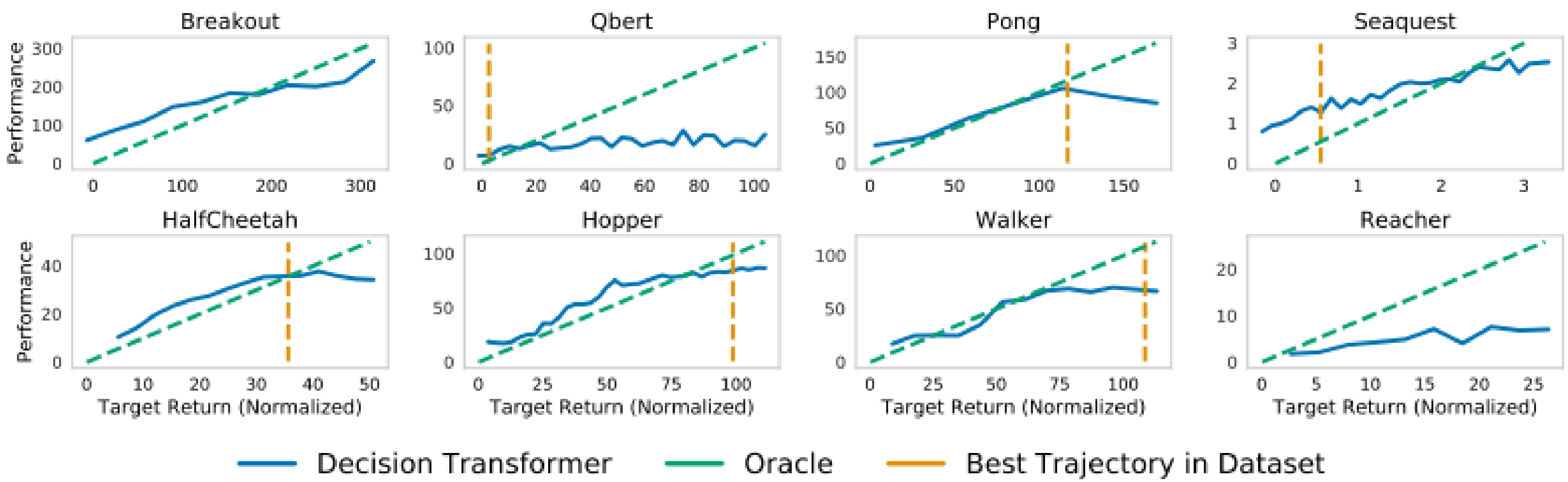
Medium-Expert:
1M: medium policy
1M: expert policy

Medium:
1M: medium policy

Experiments (Return-to-go)

- On every task, the desired target returns and the true observed returns are highly correlated.



Experiments (Behavior Cloning)

- Depends on environment, **Decision Transformer** can outperform **%BC** by using all trajectories in the dataset to improve generalization.
- Percentile Behavior Cloning (%BC) : Run BC on only the top X% of timesteps in the dataset.

Environment	DT (Ours)	10%BC	25%BC	40%BC	100%BC
HalfCheetah	42.6 ± 0.1	42.9	43.0	43.1	43.1
Hopper	$\mathbf{67.6 \pm 1.0}$	65.9	65.2	65.3	63.9
Walker	74.0 ± 1.4	78.8	**80.9**	78.8	77.3
Reacher	51.2 ± 3.4	51.0	48.9	58.2	**58.4**

Game	DT (Ours)	10%BC	25%BC	40%BC	100%BC
Breakout	$\mathbf{267.5 \pm 97.5}$	28.5 ± 8.2	73.5 ± 6.4	108.2 ± 67.5	138.9 ± 61.7
Qbert	15.1 ± 11.4	6.6 ± 1.7	16.0 ± 13.8	11.8 ± 5.8	$\mathbf{17.3 \pm 14.7}$
Pong	$\mathbf{106.1 \pm 8.1}$	2.5 ± 0.2	13.3 ± 2.7	72.7 ± 13.3	85.2 ± 20.0
Seaquest	$\mathbf{2.4 \pm 0.7}$	1.1 ± 0.2	1.1 ± 0.2	1.6 ± 0.4	2.1 ± 0.3

Conclusion

- Decision Transformer can match or outperform strong algorithms (CQL) designed explicitly for offline RL
- Minimal modifications from standard language modeling architectures