Decision Transformer: Reinforcement Learning via Sequence Modeling

Hyeonhoon Lee

RL paper study (9th)

2022.10.31.

Paper

Decision Transformer: Reinforcement Learning via Sequence Modeling

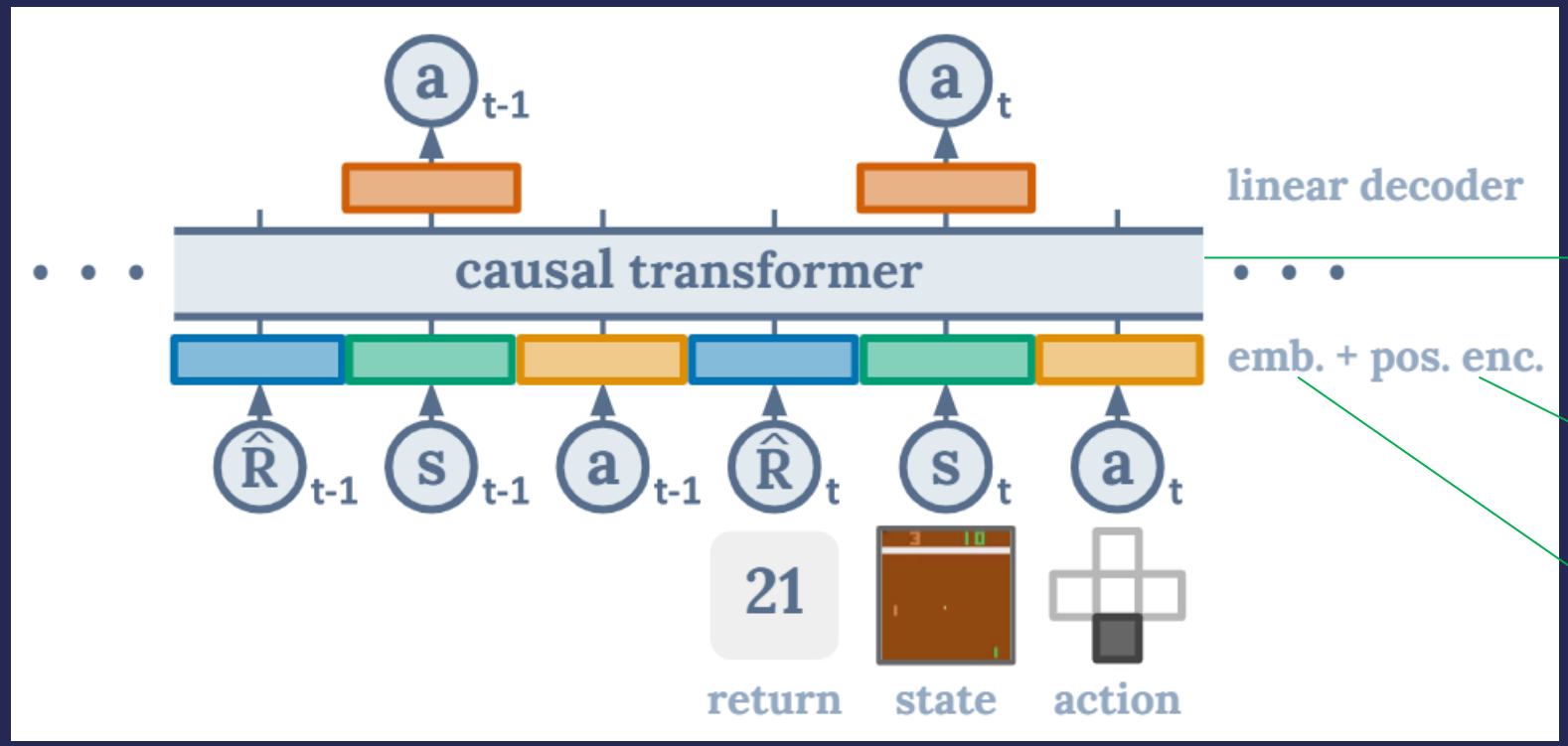
Lili Chen[*,1], **Kevin Lu**[*,1], **Aravind Rajeswaran**[2], **Kimin Lee**[1],
Aditya Grover[2], **Michael Laskin**[1], **Pieter Abbeel**[1], **Aravind Srinivas**[†,1], **Igor Mordatch**[†,3]
[*]equal contribution [†]equal advising
[1]UC Berkeley [2]Facebook AI Research [3]Google Brain
{lilichen, kzl}@berkeley.edu

Contents

Introduction

Main idea

- Reinforcement Learning (RL) -> Sequence modeling problem -> Transformer



Causal self-attention mask

Positional episodic timestep encoding

Modality(R,s,a)-specific linear embeddings

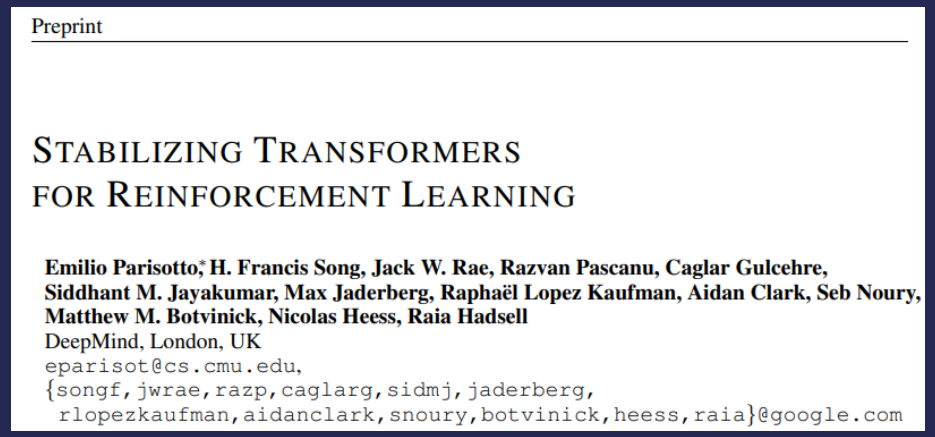
Main idea

- Reinforcement Learning (RL) -> Sequence modeling problem -> Transformer

An autoregressive model conditioned on the desired return, past states, and actions to generate future actions that achieve the desired return.

Prior study

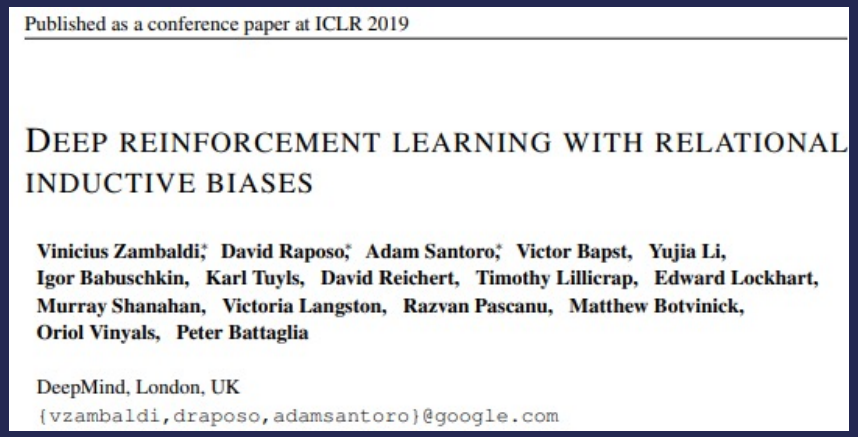
- RL using transformer
  - *As an architectural choice for components within traditional RL algorithms*



Preprint

STABILIZING TRANSFORMERS FOR REINFORCEMENT LEARNING

Emilio Parisotto,* H. Francis Song, Jack W. Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant M. Jayakumar, Max Jaderberg, Raphaël Lopez Kaufman, Aidan Clark, Seb Noury, Matthew M. Botvinick, Nicolas Heess, Raia Hadsell
DeepMind, London, UK
eparisot@cs.cmu.edu,
{songf,jwrae,razp,caglarg,sidmj,jaderberg,
rlopezkaufman,aidanclark,snoury,botvinick,heess,raia}@google.com



Published as a conference paper at ICLR 2019

DEEP REINFORCEMENT LEARNING WITH RELATIONAL INDUCTIVE BIASES

Vinicius Zambaldi,* David Raposo,* Adam Santoro,* Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, Peter Battaglia

DeepMind, London, UK
{vzambaldi,draposo,adamsantoro}@google.com

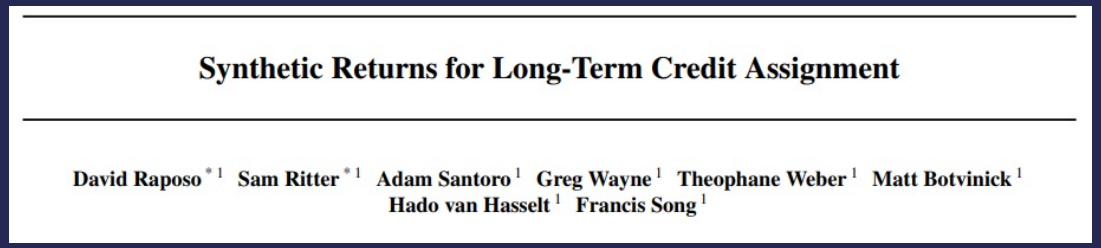
- This study aims to *replacement* for conventional RL algorithms
  - Instead of training a policy through conventional RL algorithms like temporal difference (TD) learning, we will train transformer models on collected experience using a sequence modeling objective

Advantage of Transformer (1)

- Bypassing the need for bootstrapping for long term credit assignment.
  - 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 reward.

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \right]$$

- Long term credit assignment
  - Delayed reward 환경에서 발생하는 문제.
  - 에이전트가 어떠한 적절한 행동을 취하여 desirable한 상태를 얻었음에도, 그에 대한 보상이 한참 후에 주어진다면 적절한 행동을 강화하는 데 장애가 됨.
  - 설령 그 적절한 행동이 작업을 성공하는 정말 결정적인 행동이었다 하더라도, 해당 행동에 높은 기여도를 부여(credit assignment)하기에는 그 행동 이후부터 실제로 보상을 얻기까지의 무관한 행동들도 보상에 기여한 것처럼 고려되기 때문.



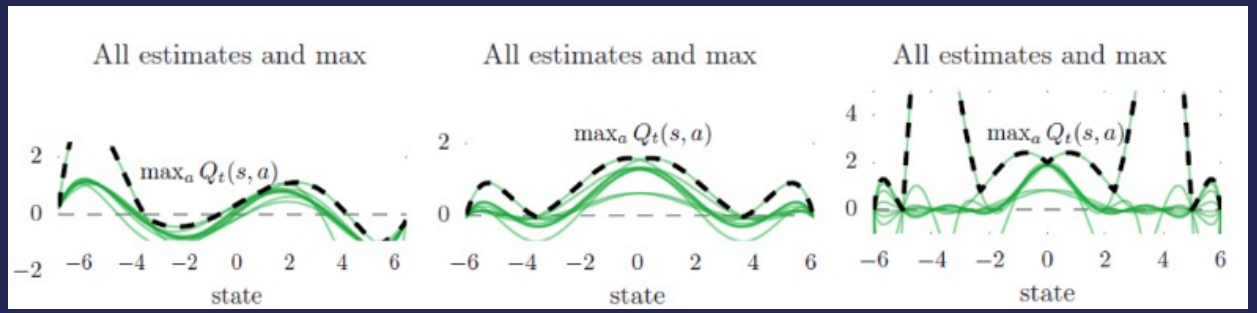
Synthetic Returns for Long-Term Credit Assignment

David Raposo [*1] Sam Ritter [*1] Adam Santoro [1] Greg Wayne [1] Theophane Weber [1] Matt Botvinick [1]
Hado van Hasselt [1] Francis Song [1]

https://connectionism.tistory.com/97

Advantage of Transformer (2)

- Challenges of Offline RL
  - Offline RL: Agents learns policies from suboptimal data – producing maximally effective behavior from fixed, limited experience.
  - Challenges: error propagation, value overestimation



- Reducing policy sampling to autoregressive generative modeling.
  - By training an autoregressive model on sequences of states, actions, and returns. (Tokens)
  - We can specify the expertise of the policy – which "skill" to query – by selecting the desired return tokens, acting as a prompt for generation.

https://ropiens.tistory.com/144

Illustrative example

- The reward is 0 when the agent is at the goal node and −1 otherwise.
- We train a GPT model to predict next token in a sequence of returns-to-go (sum of future rewards), states, and actions.

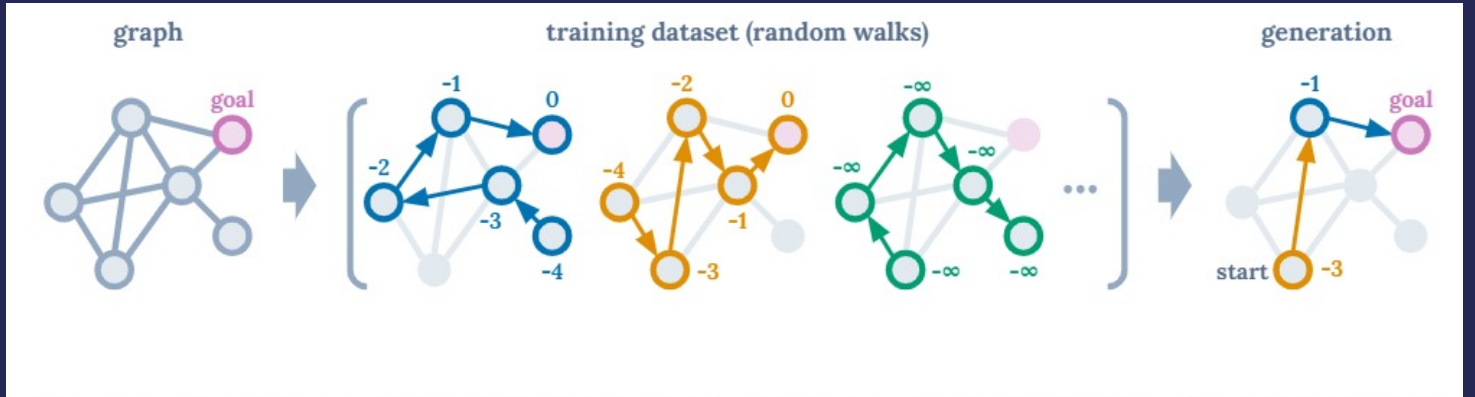


Figure 2: Illustrative example of finding shortest path for a fixed graph (left) posed as reinforcement learning. Training dataset consists of random walk trajectories and their per-node returns-to-go (middle). Conditioned on a starting state and generating largest possible return at each node, Decision Transformer sequences optimal paths.

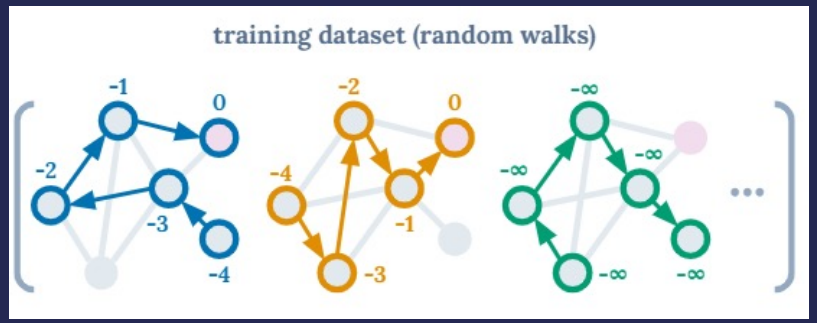
- By combining the tools of sequence modeling with hindsight return information, we achieve policy improvement without the need for dynamic programming.

Methods

Trajectory representation

- Returns-to-go (instead of rewards directly)

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



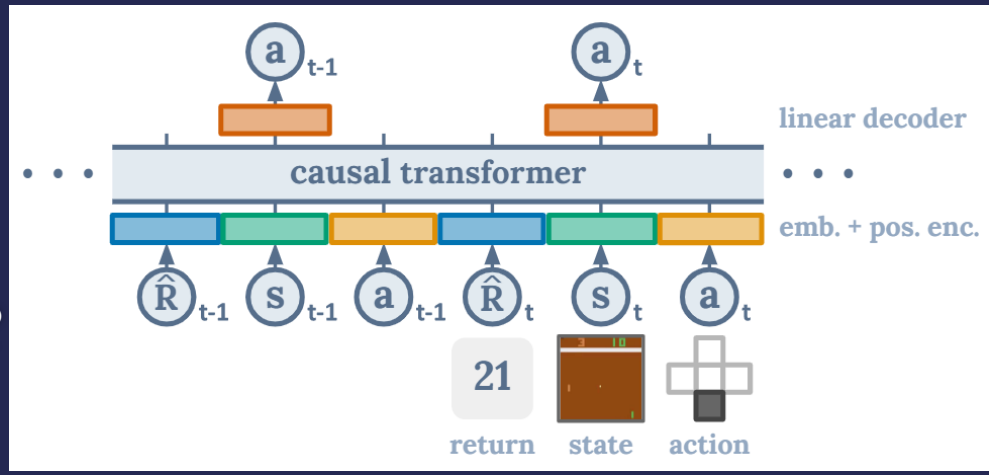
- Trajectory

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

- Test time
  - Specifying the desired performance (e.g. 1 for success or 0 for failure).
    - After executing the generated action for the current state, we decrement the target return by the achieved reward and repeat until episode termination.
  - Specifying Initial state (for conditioning information).

Architecture

- Last *K* timesteps into Decision Transformer (DT): A total of 3*K* tokens (R, s, a)
- Token embedding + Positional embedding (3 tokens per each timestep).
- GPT architecture
  - Self-attention layer: the layer to assign "credit" by implicitly forming state-return associations via similarity of the query and key vectors (maximizing the dot product): certain "important" states comprise most of the credit.
  - Modifying the transformer architecture with a causal self-attention mask to enable autoregressive generation, replacing the summation/softmax over then tokens with only the previous tokens in the sequence
- The tokens are then processed by a GPT model, which predicts future action tokens via autoregressive modeling.
- Either with cross-entropy loss for discrete actions or mean-squared error for continuous actions – and the losses for each timestep are averaged.



Training

- Given a dataset of offline trajectories
- Sampling minibatches of sequence length *K* from dataset.
- Either with cross-entropy loss for discrete actions or mean-squared error for continuous actions – and the losses for each timestep are averaged.

Pseudo code

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
```

https://velog.io/@nawnoes/Decision-Transformer-ReinforcementLearning-via-Sequence-Modeling

Results

Benchmarks

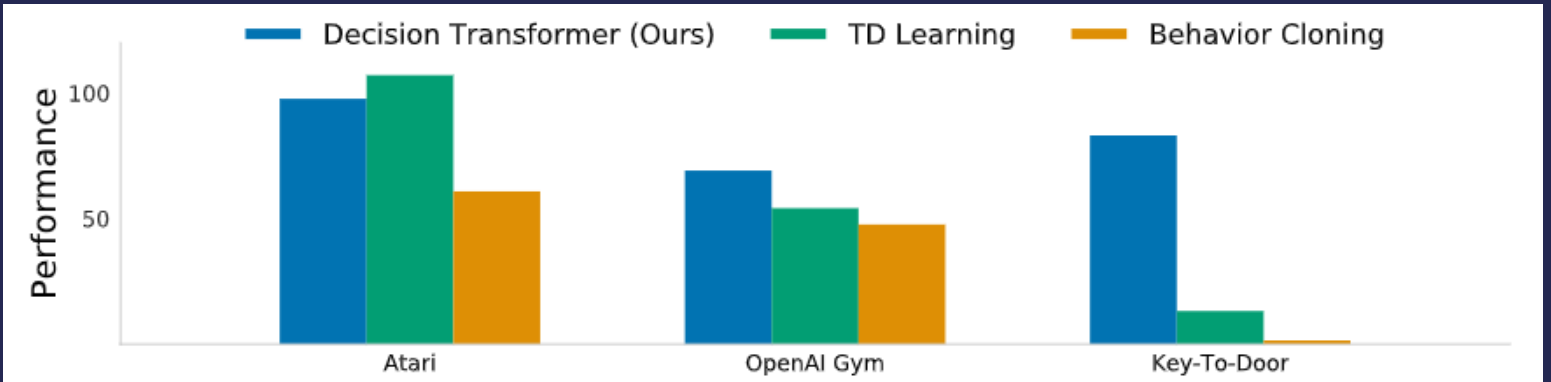


Figure 3: Results comparing Decision Transformer (ours) to TD learning (CQL) and behavior cloning across Atari, OpenAI Gym, and Minigrid. On a diverse set of tasks, Decision Transformer performs comparably or better than traditional approaches. Performance is measured by normalized episode return (see text for details).

- Behavior cloning(BC) utilizes the same network architecture and hyperparameters as Decision Transformer but does not have return-to-go conditioning.

Benchmarks

- Atari

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.

Benchmarks

- OpenAI Gym

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

Table 2: Results for D4RL datasets[3]. We report the mean and variance for three seeds. Decision Transformer (DT) outperforms conventional RL algorithms on almost all tasks.

1. Medium: 1 million timesteps generated by a "medium" policy that achieves approximately one-third the score of an expert policy.
2. Medium-Replay: the replay buffer of an agent trained to the performance of a medium policy(approximately 25k-400k timesteps in our environments).
3. Medium-Expert: 1 million timesteps generated by the medium policy concatenated with 1 million timesteps generated by an expert policy.

Discussion

Discussion

- Does Decision Transformer perform behavior cloning on a subset of the data?

Dataset	Environment	DT (Ours)	10%BC	25%BC	40%BC	100%BC	CQL
Medium	HalfCheetah	42.6 ± 0.1	42.9	43.0	43.1	43.1	**44.4**
Medium	Hopper	$\mathbf{67.6 \pm 1.0}$	65.9	65.2	65.3	63.9	58.0
Medium	Walker	74.0 ± 1.4	78.8	**80.9**	78.8	77.3	79.2
Medium	Reacher	51.2 ± 3.4	51.0	48.9	58.2	**58.4**	26.0
Medium-Replay	HalfCheetah	36.6 ± 0.8	40.8	40.9	41.1	4.3	**46.2**
Medium-Replay	Hopper	$\mathbf{82.7 \pm 7.0}$	70.6	58.6	31.0	27.6	48.6
Medium-Replay	Walker	66.6 ± 3.0	**70.4**	67.8	67.2	36.9	26.7
Medium-Replay	Reacher	18.0 ± 2.4	**33.1**	16.2	10.7	5.4	19.0
Average		56.1	**56.7**	52.7	49.4	39.5	43.5

Table 3: Comparison between Decision Transformer (DT) and Percentile Behavior Cloning (%BC).

- The top X% of timesteps in the dataset, ordered by episode return.
- The percentile X% interpolates between standard BC (X=100%) that trains on the entire dataset and only cloning the best observed trajectory (X→0%).

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	15.4 ± 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	$\mathbf{2.5 \pm 0.4}$	1.1 ± 0.2	1.1 ± 0.2	1.6 ± 0.4	2.1 ± 0.3

Table 4: %BC scores for Atari. We report the mean and variance across 3 seeds. Decision Transformer (DT) outperforms all versions of %BC in most games.

- We use 1% of a replay buffer as the dataset.
- DT can be more effective than simply performing imitation learning on a subset of the dataset

Discussion

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

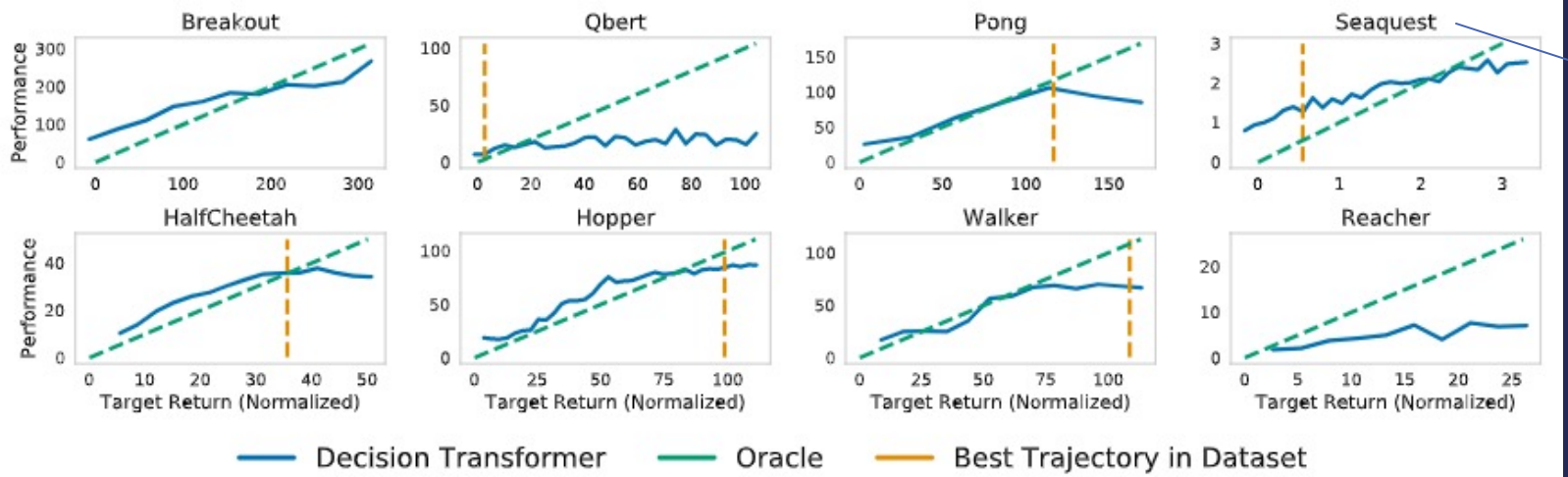


Figure 4: Sampled (evaluation) returns accumulated by Decision Transformer when conditioned on the specified target (desired) returns. **Top:** Atari. **Bottom:** D4RL medium-replay datasets.

DT is sometimes capable of extrapolation

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

Discussion

- What is the benefit of using a longer context length?

Game	DT (Ours)	DT with no context ($K = 1$)
Breakout	$\mathbf{267.5 \pm 97.5}$	73.9 ± 10
Qbert	$\mathbf{15.1 \pm 11.4}$	13.6 ± 11.3
Pong	$\mathbf{106.1 \pm 8.1}$	2.5 ± 0.2
Seaquest	$\mathbf{2.5 \pm 0.4}$	0.6 ± 0.1

Table 5: Ablation on context length. Decision Transformer (DT) performs better when using a longer context length ($K = 50$ for Pong, $K = 30$ for others).

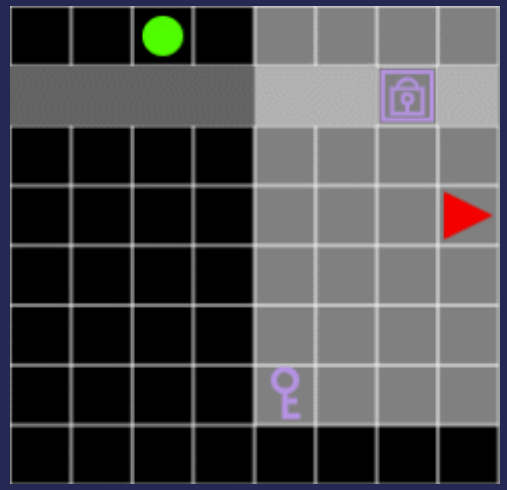
- DT performance is significantly worse when K= 1, indicating that past information is useful for Atari games.

Discussion

- Does Decision Transformer perform effective long-term credit assignment?

Dataset	DT (Ours)	CQL	BC	%BC	Random
1K Random Trajectories	**71.8%**	13.1%	1.4%	69.9%	3.1%
10K Random Trajectories	94.6%	13.3%	1.6%	**95.1%**	3.1%

Table 6: Success rate for Key-to-Door environment. Methods using hindsight (Decision Transformer, %BC) can learn successful policies, while TD learning struggles to perform credit assignment.



- The agent receives a binary reward when reaching the door in the third phase, but only if it picked up the key in the first phase.

Discussion

- Can transformers be accurate critics in sparse reward settings?

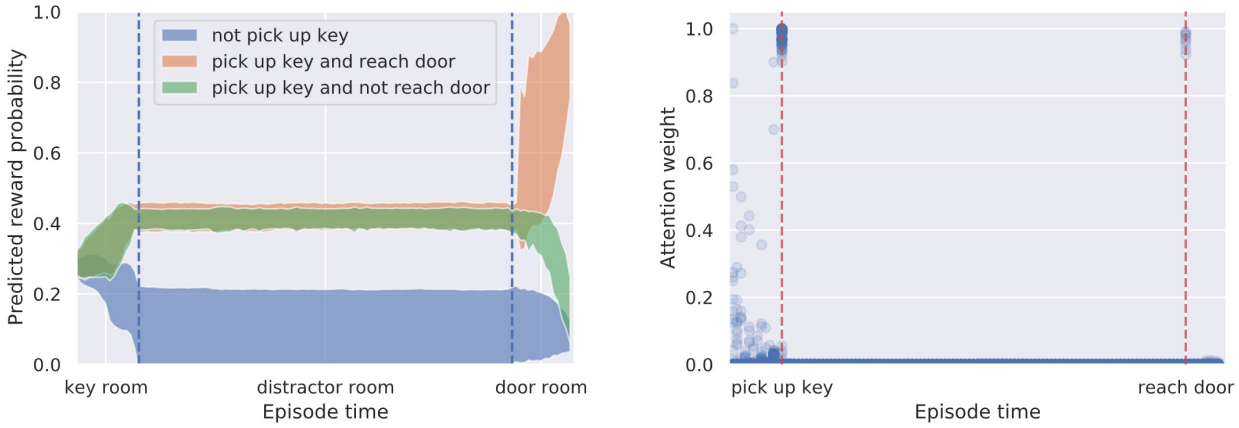


Figure 5: **Left:** Averages of running return probabilities predicted by the transformer model for three types of episode outcomes. **Right:** Transformer attention weights from all timesteps superimposed for a particular successful episode. The model attends to steps near pivotal events in the episode, such as picking up the key and reaching the door.

- DT continuously updates reward probability based on events during the episode.
- DT attends to critical events in the episode (picking up the key or reaching the door, indicating formation of state-reward associations and enabling accurate value prediction.

Discussion

- Does Decision Transformer perform well in sparse reward settings?

Dataset	Environment	Delayed (Sparse) DT (Ours)	Delayed (Sparse) CQL	Agnostic BC	Agnostic %BC	Original (Dense) DT (Ours)	Original (Dense) 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.

where the agent does not receive any rewards along the trajectory, and instead receives the cumulative reward of thetrajectory in the final timestep.

Discussion

- Why does Decision Transformer avoid the need for value pessimism or behavior regularization?
  - TD-learning based algorithms learn an approximate value function and improve the policy by optimizing this value function.
- How can Decision Transformer benefit online RL regimes?
  - Offline RL and the ability to model behaviors has the potential to enable sample-efficient online RL for downstream tasks.
  - Works studying the transition from offline to online generally find that likelihood-based approaches, like our sequence modeling objective, are more successful

Implementation

Introducing Decision Transformers on Hugging Face 🤗

Published March 28, 2022.

Update on GitHub


edbeeching
Edward Beeching


ThomasSimonini
Thomas Simonini

```
from transformers import DecisionTransformerModel

model_name = "edbeeching/decision-transformer-gym-hopper-expert"
model = DecisionTransformerModel.from_pretrained(model_name)
```

https://huggingface.co/blog/decision-transformers


Thank you