Decision Transformer: Reinforcement Learning via Sequence Modeling

Lili Chen*,[1], Kevin Lu*,[1] et al., 2021

Presenter: **Sanjeev Kumar**
Supervisor: Dr. Friedrich Solowjow

Contact : sanjeev.kumar@rwth-aachen.de
: friedrich.solowjow@dsme.rwth-aachen.de

Motivation:

Today's Approach in Reinforcement Learning:

Using trial-error, learn the policy that maximizes the agent's return (cumulative rewards over the time)

- **Value based**
- **Policy based**
- **Actor - Critic**

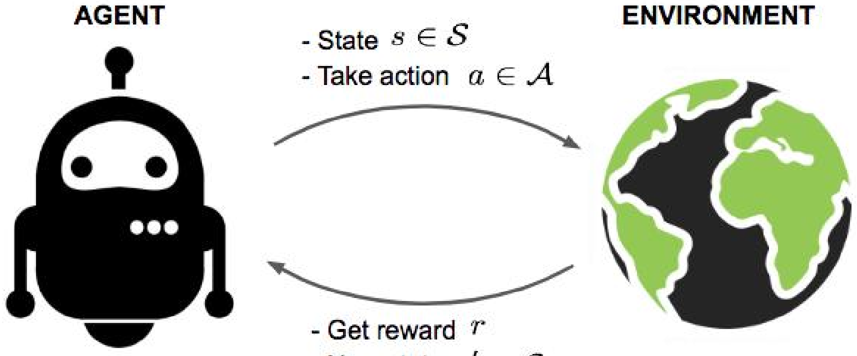


Image source: https://vitalflux.com/reinforcement-learning-real-world-examples/

What is gap between current practices in RL and matured data driven ML regimes (NLP, Vision) ?

• • • $\hat{R}_{t-1}$

Problem Statement:

Why do RL Methods scale poorly ?

- Limited by small-scale, single-task nature of training.
- Poor modelling of large scale distribution, hence catastropic forgetting (w.r.t to past behavior)
- Non-stationary problem, caused by learning both actor and critics network.
- Complex solution (don’t scale well) are used, in **offline RL** to fix the instabilities

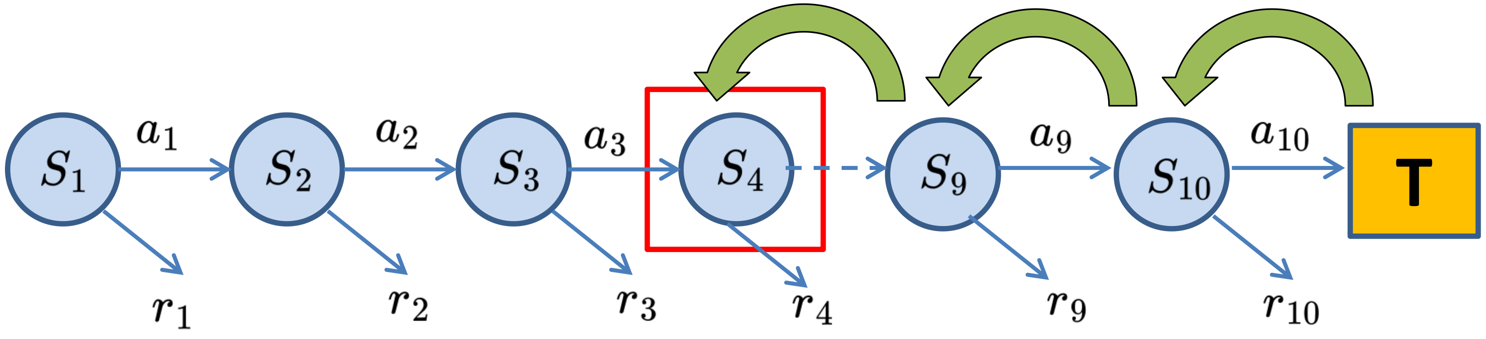
No trial-error, Only logged dataset of interactions

Image source: https://www.deepmind.com/blog/rl-unplugged-benchmarks-for-offline-reinforcement-learning

Goal: learn general data driven behavior using , model free - offline line RL, in same framework that has scaled language and vision successfully

Institute for Data Science in Mechanical Engineering
RWTH AACHEN UNIVERSITY

Problem Statement: forgetting



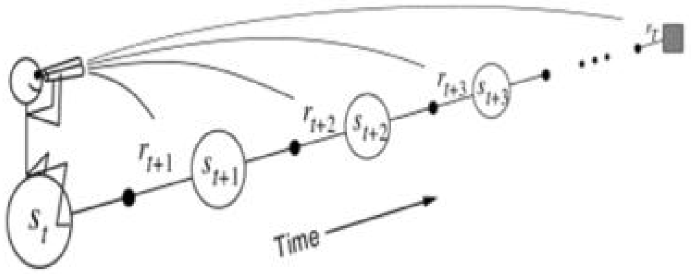


Image Source: http://www.incompleteideas.net/book/ebook/node74.html

- use more than **one temporal step** to update the earlier state values

current TD (n-step) update Rule:

Why it is inefficent ?

- Update at lastest time step, influence **very little** at earlier time step, because of **discount factor.**

What if we can assign credit over the sequence, and predict new sequence of action based on assigned credit ?

Key Idea:

Based on the current advancement in NLP and vision, Transformers seems to work expectionally well **on the sequence Data .**

In RL, we also have a sequence in form of trajectory (s1, a1,r1, s2, a2,r2, ...)

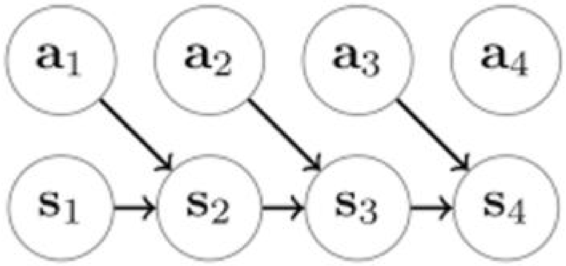


Fig 1. graphical model with states and actions

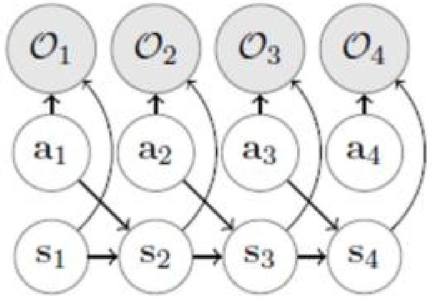


Fig 2. graphical model with optimality variables

Limitation: *Don't tell how to pick a_t*

Advantage: given (s_t, a_t), O_t determine action taken was optimal or not

Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review by Levine 2018

Objective:

Algorithm Idea:

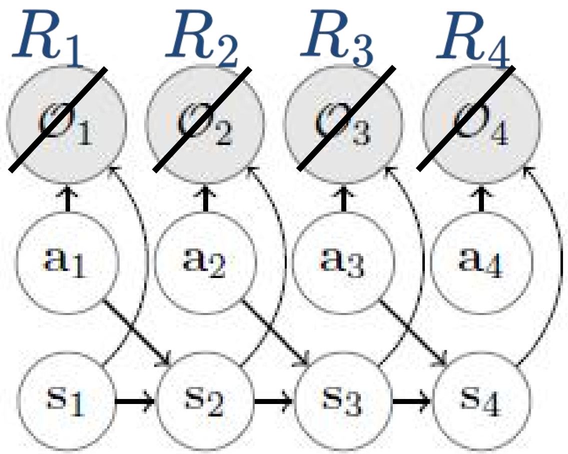
In practise we don't have O_t

$$\rightarrow \textbf{replace } O_t \textbf{ with returns-to-go } R_t$$

$$p(O_t \mid s_t, a_t) \propto p(a_t \mid s_t, O_t) --> (Bayes' Rule)$$

$$\Rightarrow p(a_t \mid s_t, R_t) --> (Markovian\ env)$$

$$\Rightarrow \mathbf{p}(a_t \mid s_t, R_t, a_{t-1}, R_{t-1}, \ldots\ldots) ---> (general\ env)$$



same objective as auto-regressive sequence modeling such as NLP and vision
=> apply transformer model on trajectory sequence

Reinforcement Learning Upside Down: Don't Predict Rewards--Just Map Them to Actions by Schmidhuber et al., 2019

Method : Decision Transformer (DT) for Offline RL

Simple Objective:

$$\max p(a_t \mid trajectory\ upto\ t)$$

Attention Mechanism Idea:

Assign attention weight to each token, to know how much "attention" the model should pay to each token present in the context length, while predicting the next token.

$$token \Rightarrow \langle R_t, s_t, a_t \rangle$$

Main Steps in Attention Mechanism:

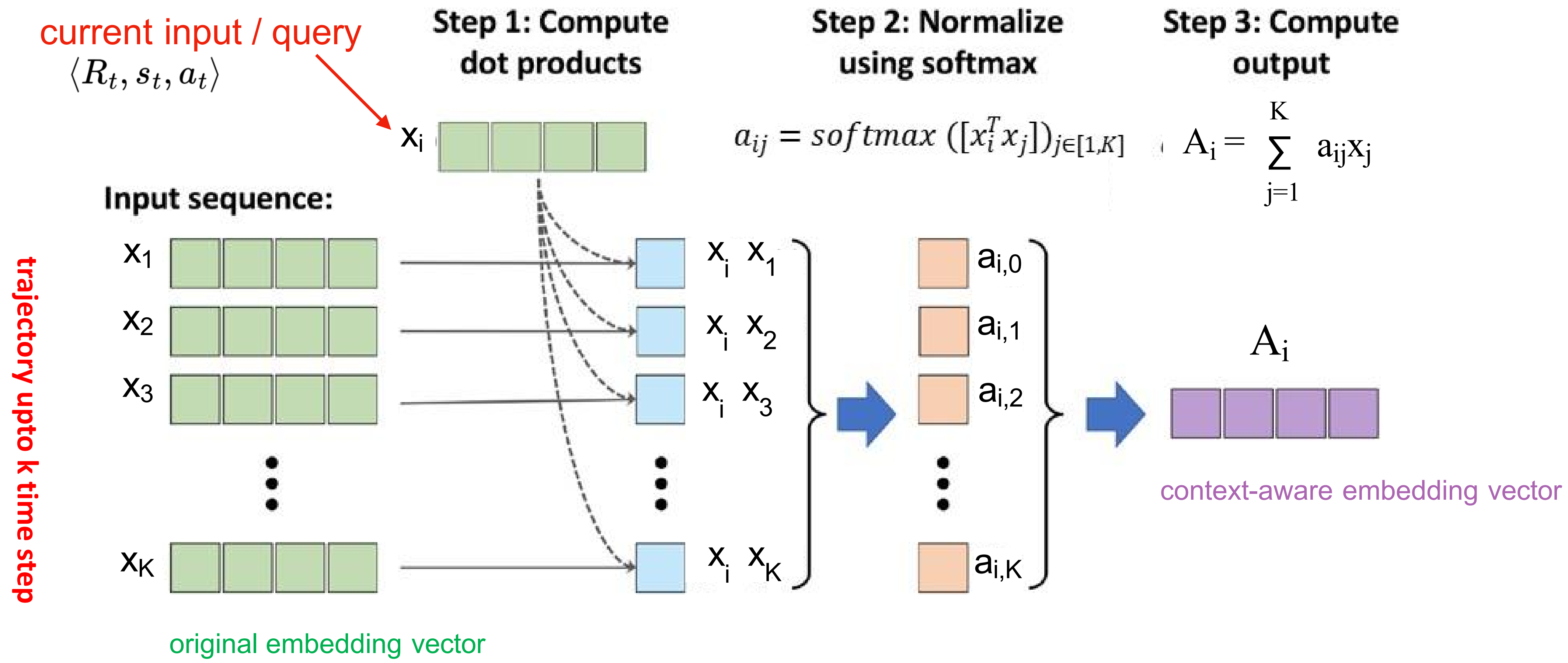
- Compute **attention weight** : similarity betwen current token and all other tokens present in context length
- Normalize the attention weights via **softmax**
- Compute **attention value** from normalized weights and corresponding tokens present in the context length

$$\mathrm{A_i} = \sum_{j=1}^{T} a_{ij} \; x_j$$

attention value of ith input token

attention weight based on similarity btw current ith input token and all other tokens in context length

Self Attention Mechanism - Very Basic Form



Base Image source: Raschka & Mirjalili 2019. Python Machine Learning, 3rd edition

Self Attention Mechanism (Scaled Dot Product Attention)

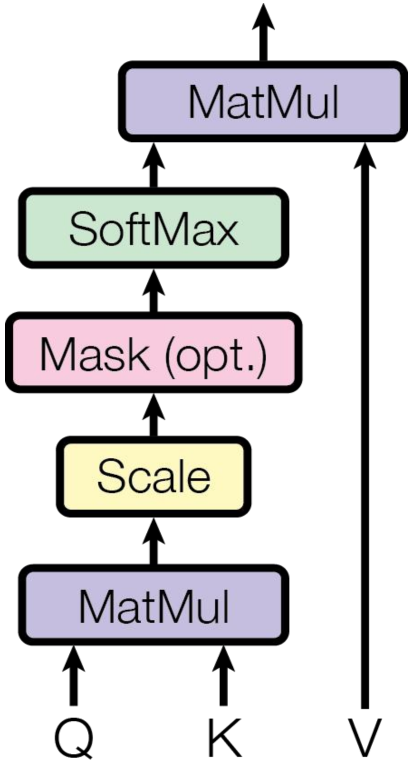
- Previous basic version did not involve any learnable parameters, so not useful for learning a language model
- Added 3 trainable weight matrices that are multiplied with the input sequence embeddings.

$$query = W^q x_i$$

$$key = W^k x_i$$

$$value = W^v x_i$$

$$A(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Decision Transformer Pseudo Code for Continous action

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)
```

Input : sequence of (returns-to-go, states, actions)

Output: sequence of predicated actions

Attention is computed over the **context length K** **(hyperparameter)**

Training and Evaluation Loop

Training loop:

```
# 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()
```

- **Continuous Actions:** Mean Square Error Loss
- **Discrete Action Space:** Cross Entropy Loss

Both loss functions are stable to train and easy to regularize.

Evaluation loop:

```
# 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
```

- For evaluation, set initial **returns-to- go** to as desired target return (1: expert, 0: failure)

Evaluations on Offline RL Benchmarks

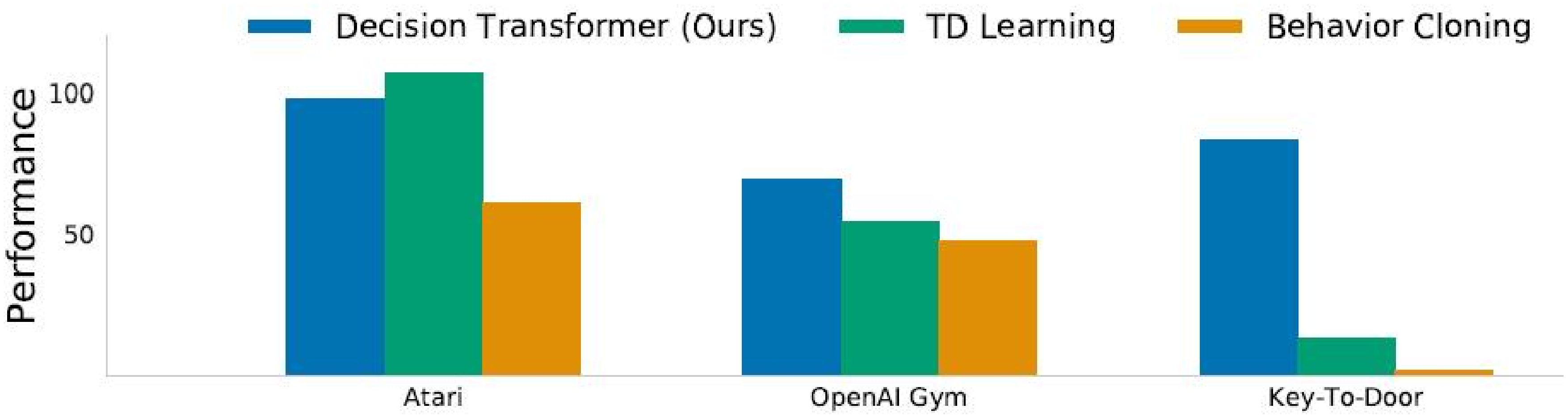


Fig: Results comparing Decision Transformer (ours) to TD learning (CQL) and behavior cloning across Atari, OpenAI Gym, and Minigrid. On a diverse set of tasks, Decision Transformerperforms comparably or better than traditional approaches.

Compare the performance of DT relative to:

- **model-free offline RL based on TD learning** - SOTA is Conservative Q-Learning(CQL)
- **Iimitation learning algorithms:** Behaviour clonning

- DT is competitive with SOTA model free offline RL methods (CQL)
- DT outperforms at long term credit assignment tasks in env with sparse rewards

Experimental Results: Varying Return Conditioning

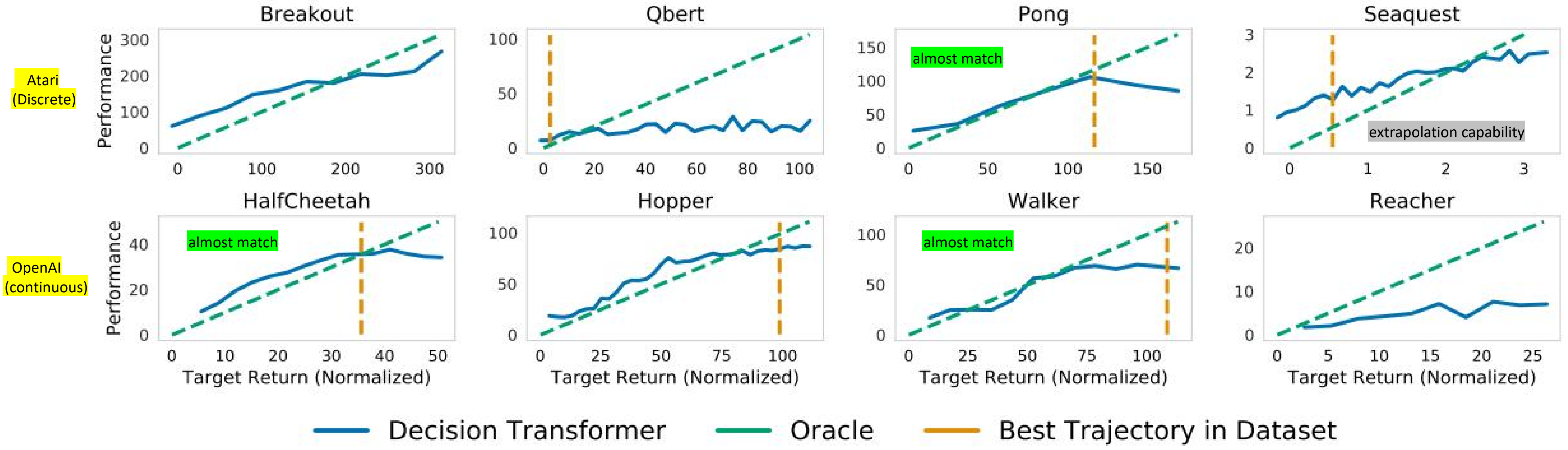


Fig: Sampled (evaluation) returns accumulated by Decision Transformer when conditioned onthe specified target (desired) returns. Top: Atari. Bottom: D4RL medium-replay datasets.

Best Trajectory in Dataset: upper bound on the offline data

- DT can perform multi-task via Return Conditioning.
- Occassionally, these models extrapolate.

future work : seaquest like performance consistent across diff environment

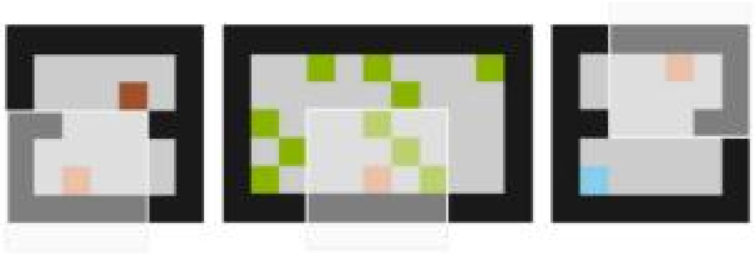
Experimental Results: Long Credit Assignment

Environment Used: variant of Key-to-Door proposed in Mesnard et.al

Description: Grid based environmnet with 3 phases

- **Phase1 :** Agent placed in room with a key
- **Phase2:** Agent placed in empty room
- **Phase3:** Agent placed in room with a door

Key-to-Door



Agent receives a binary reward when open the door in third phase, **but only if it picked up the key in first phase.**

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: 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.

Some Other Discussions in DT

- Effect of Context Length studied on Atari environment. DT performs better when using a longer context length.
- Evaluated the performance of DT in spase reward settings. Very minimal effect by removal of dense rewards.
- DT as critics in sparse reward settings.

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

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

		Delayed (Sparse)		Agnostic		Original (Dense)	
Dataset	Environment	DT (Ours)	CQL	BC	%BC	DT (Ours)	CQL
Medium-Expert	Hopper	$\mathbf{107.3 \pm 3.5}$	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	$\mathbf{78.5 \pm 3.7}$	2.0	27.6	70.6	82.7	48.6

Table: 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.

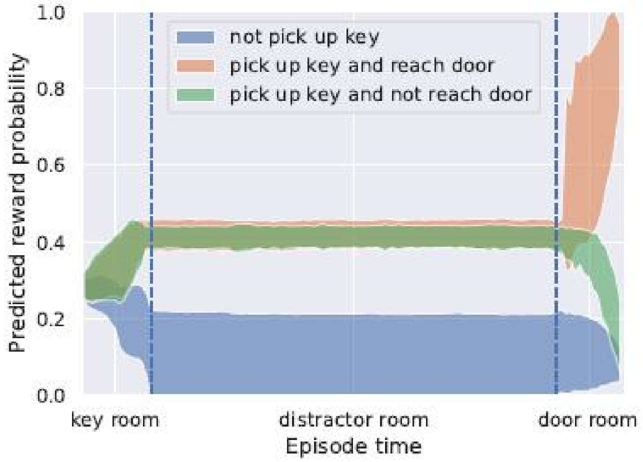


Figure : Left: Averages of running return probabilities predicted by the transformer model for three types of episode outcomes

Summary

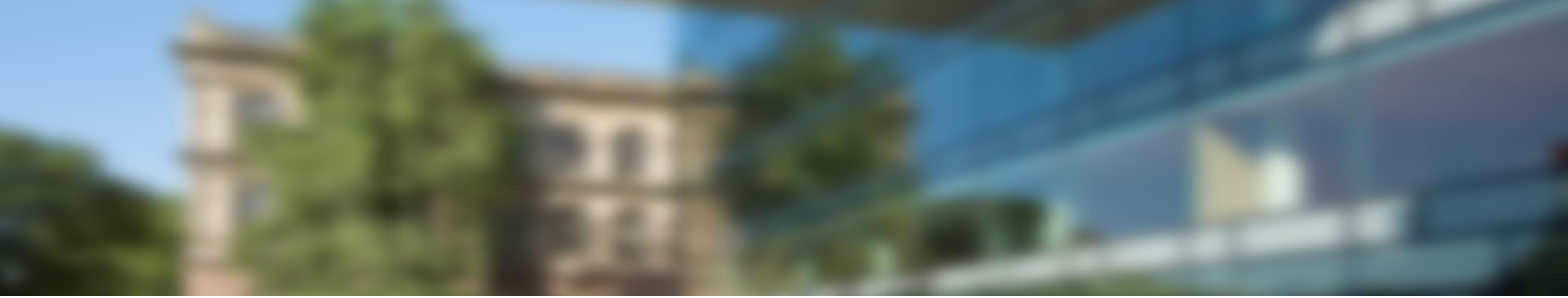
Decision transformers demonstrate a new approach for RL based on sequence modeling

Advantages over previous RL approaches:

- simple, scalable design
- stable training
- easy integration with language and visual inputs

Limitation of DT:

- based on return-to-go conditioning



Thank you for your attention!