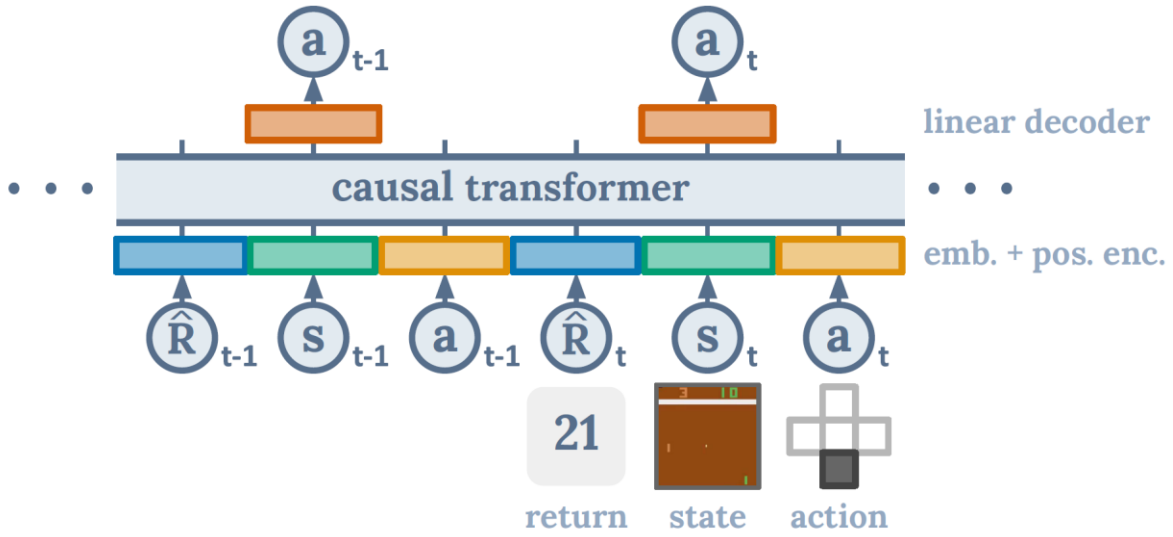
Decision Transformer: Reinforcement Learning via Sequence Modeling

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee,

Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivasy, Igor Mordatchy

UC Berkeley Facebook AI Research Google Brain

Abstract



- Decision Trnasformer는 RL을 조건부 시퀀스 문제를 푸는 프레임워크로 보고. 이러한 관점으로 GPT 또는 BERT와 같은 transformer 구조를 적용함.
- DT 모델은 가장 최신 model free offline RL baselines으로 한 RL모델들 보다 Atari, open AI Gym, key-to-Door task에서 동일하거나 훨씬 뛰어 넘는 성능을 보여준다.

Introduction

- Transformer는 Language, vision 과 같은 다양한 곳에 성공적으로 적용되었고, 여기서 제안 하는것은 Transformer를 sequential decision making problem(RL)에 적용하려 하는거다.
- 이전에도 이와 비슷한 시도가 있었다. Transformer를 전통 RL 구조의 일부 component로 채택하는 시도가 있었지만, 이전 시도들과는 달리 여기서는 generative trajectory modeling(states, actions, reward의 joint distribution을 모델링하는거)이 전통 RL 알고리즘을 대체 할 수 있는지를 알아보고자 한다.
- 이 방법은 long term credit assignment 위해서 boostsrapping 시켜야 하는 필요를 없애준다.
- 또한 미래의 보상을 discount할 필요를 없애준다.
- 추가적으로 우리는 언어, 비젼 쪽으로 널리 사용되는 transformer 프레임워크를 이용할수 있다. Ex) GPT – X, BERT 등등
- Transformer는 bellman backup(slowly propagate rewards and are prone to "distractor" signal)과는 달리 credit assignment 를 self-attention으로 직접적으로 행사할수 잇다.
  이로 인해 Transformer로 하여금 sparse or distracting rewards 같은 상황에서도 꽤 효율적으로 작동될 수 있다.
- 이 가정들을 offline RL에 적용하였다. (policy를 학습할때 suboptimal data를 시용한다)
  * suboptimal - 고정되고 제한된 경험에서 최고의 효율의 행동을 할수 있게 하는거

Introduction

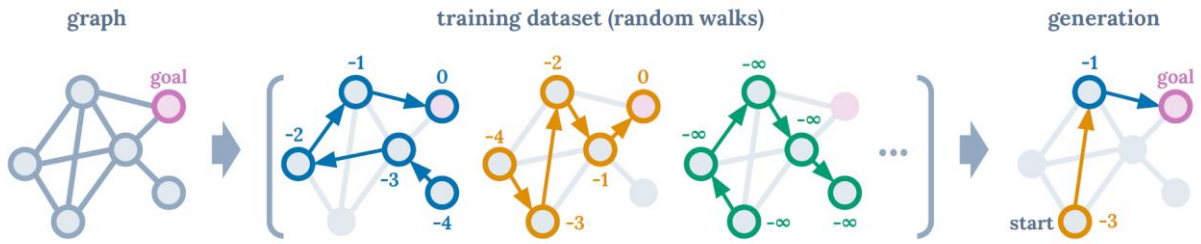


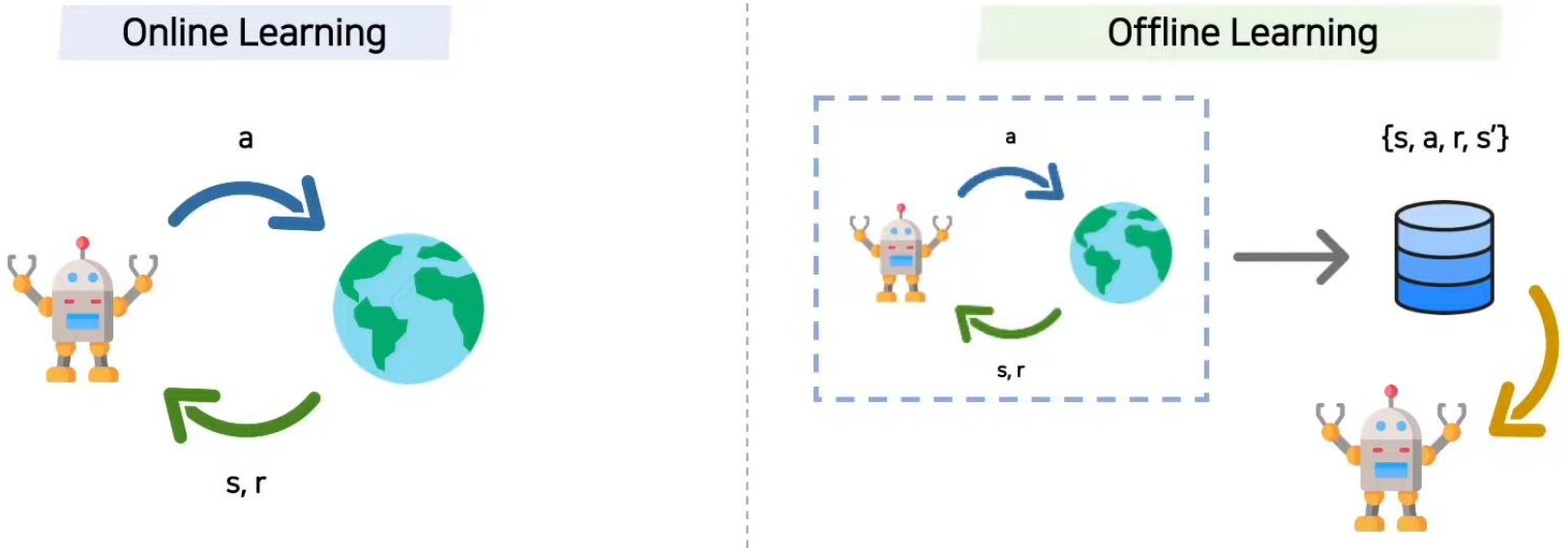
Figure 2: Illustrative example of finding shortest path for a fixed graph (left) posed as reinforcement learning. Training dataset consists of random walk trajectories and their per-node returns-to-go (middle). Conditioned on a starting state and generating largest possible return at each node, Decision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

Illustartive example

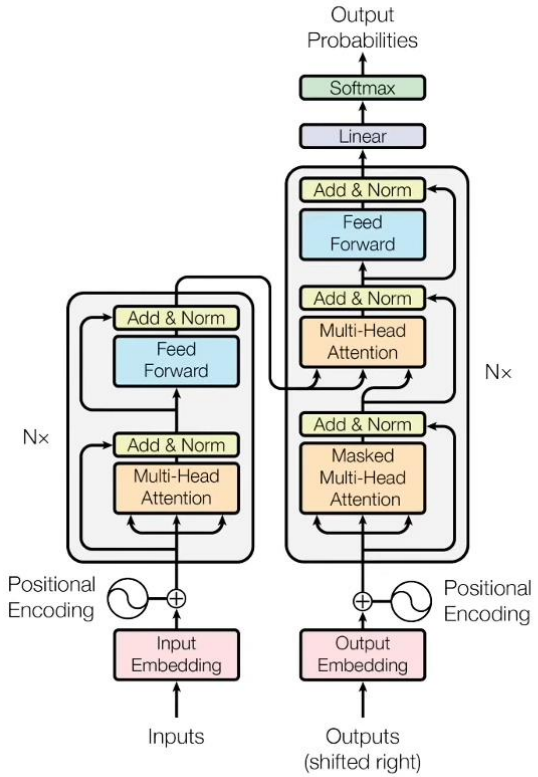
좀 더 직관을 가질수 있게 예를 보여준다. 여기서 보여준 예는 directed graph에서 목적지에 가장 짧은 경로를 찾는거다.
goal 노드에 도착하면 0의 reward가 주어지고, 그렇지 않으면 -1이 주어진다. 여기서 GPT 모델을 학습 시켜 다음 token(return-to-go, states, action)을 예상한다. expert의 demonstrations없이, random walk 데이터로만 학습시켰다.

Offline learning

- Offline Reinforcement learning 이란?
  - Batch Reinforcement Learning(batchRL)이라고도 알려져 있음
  - Exploration 없이도 정해진 batch만큼의 데이터만 가지고 agent를 학습
  - Policy π_β (정답이 되는 행동 등)을 가지고 suboptimal 데이터 집합 D를 수집하고 이로부터 policy를 학습
  - 고정된 데이터셋만 가지고 exploitation을 수행하는 것



Transformer



✓ **Machine translation을 위해 제안된 Sequence-to-sequence 모델**

✓ **Encoder-Decoder 구조**

- 각 Encoder와 Decoder는 N개의 동일한 Block이 Stack된 형태
- Multi-head self-attention
- Position-wise feed-forward network
- Residual connection
- Layer Normalization

✓ **Positional Encoding**

- 문장의 Token들이 순차적으로 모델의 input이 되는 것이 아니라 하나의 matrix로 input 구성
- 따라서 순서 정보를 반영하지 못하여 이를 보완하기 위해 Positional Encoding을 사용

✓ **Self-Attention**

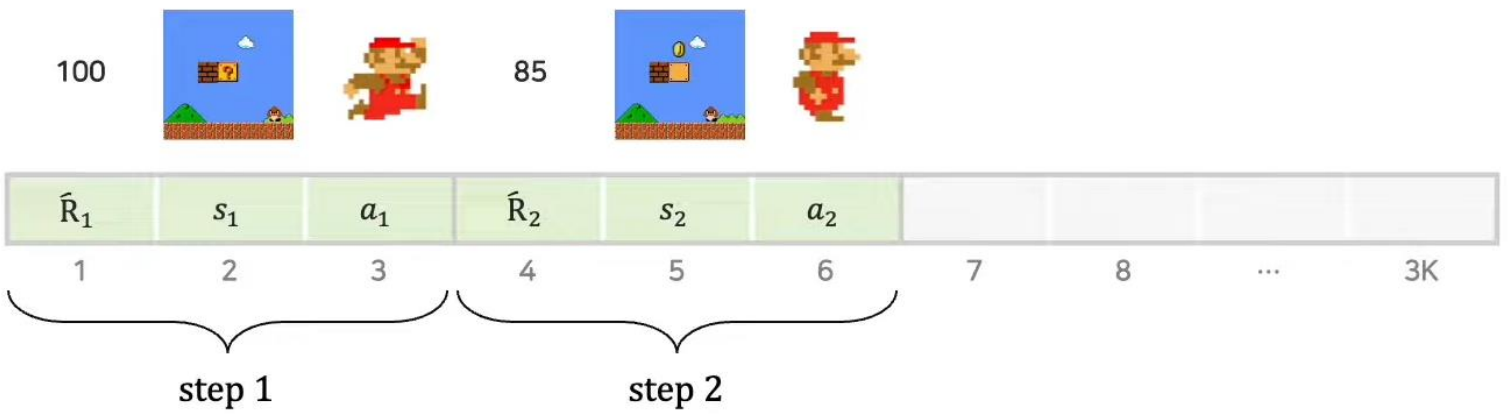
- 입력한 문장 내의 각 단어를 처리해 나가면서 , 문장 내의 다른 위치에 있는 단어들을 보고 힌트를 받아 현재 타겟 위치의 단어를 더 잘 인코딩할 수 있게 하는 과정

Method 설명

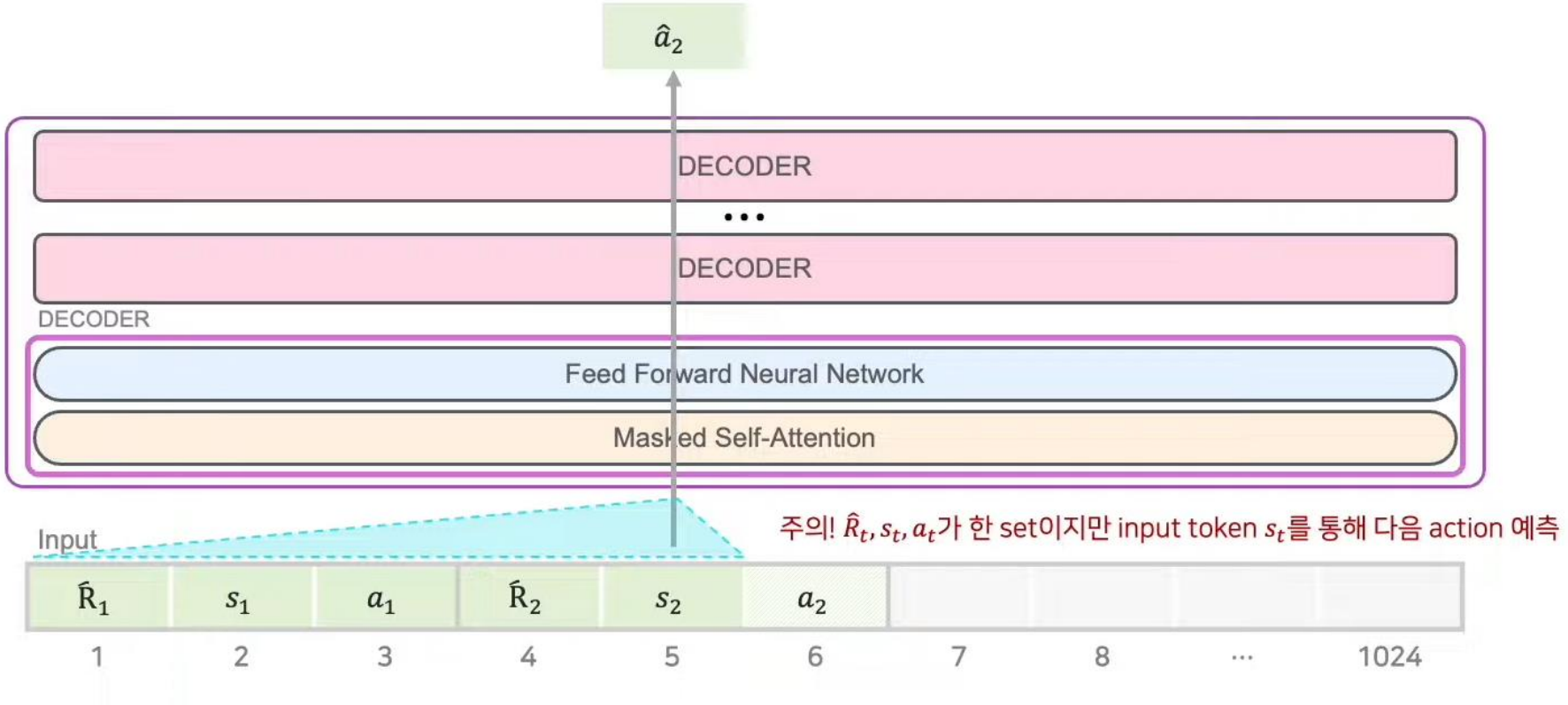
- **Trajectory Representation이란?**
  - Return-to-go, state, action이 한 세트 (time-step 단위)
  - Transformer가 의미 있는 패턴을 학습할 수 있어야 하고 test time에 conditionally action을 생성할 수 있어야 한다.
  - 과거 보상들보다 미래의 desired returns에 기반한 actions을 생성하길 원하기 때문에 rewards 대신에 return-to-go를 feeding

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

$$\hat{R}_t = \sum_{t'=t}^{T} r_{t'}$$ 주의! Discounted Return 아님!



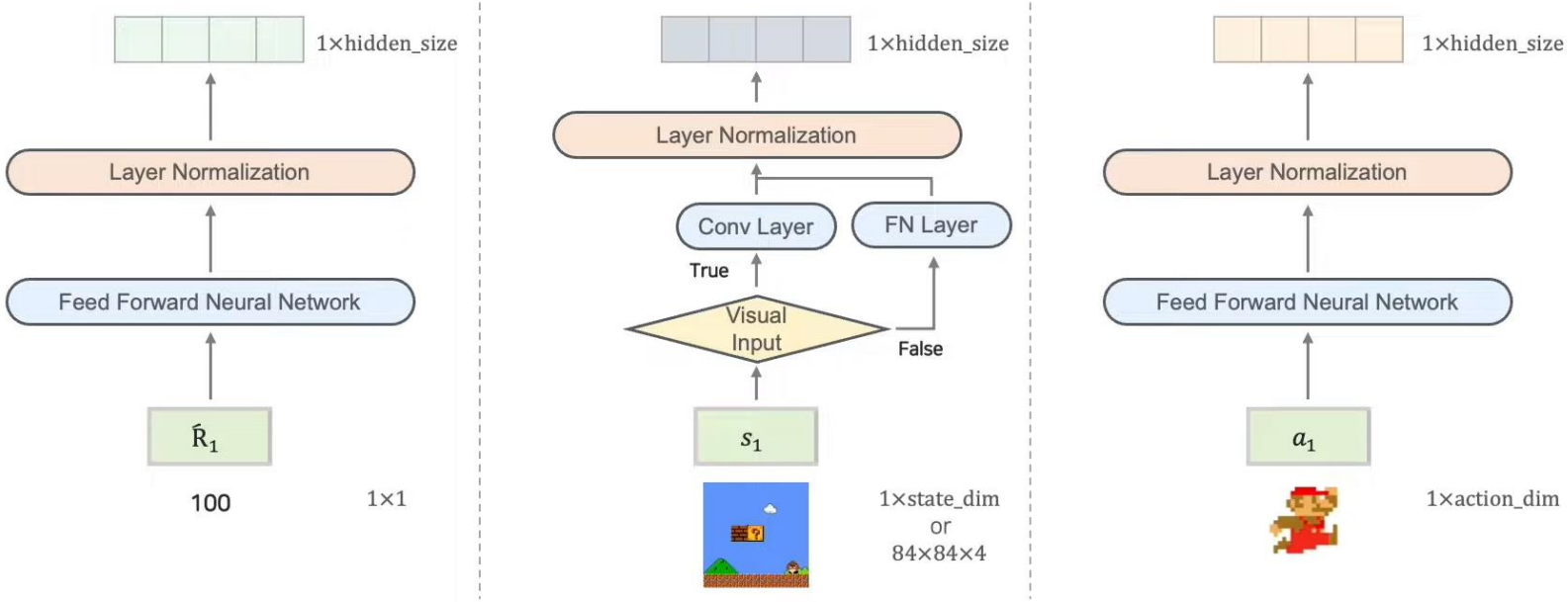
Method 설명



- Trajectory 데이터를 input으로 받아 다음 action을 예측 하는 방식

Method 설명

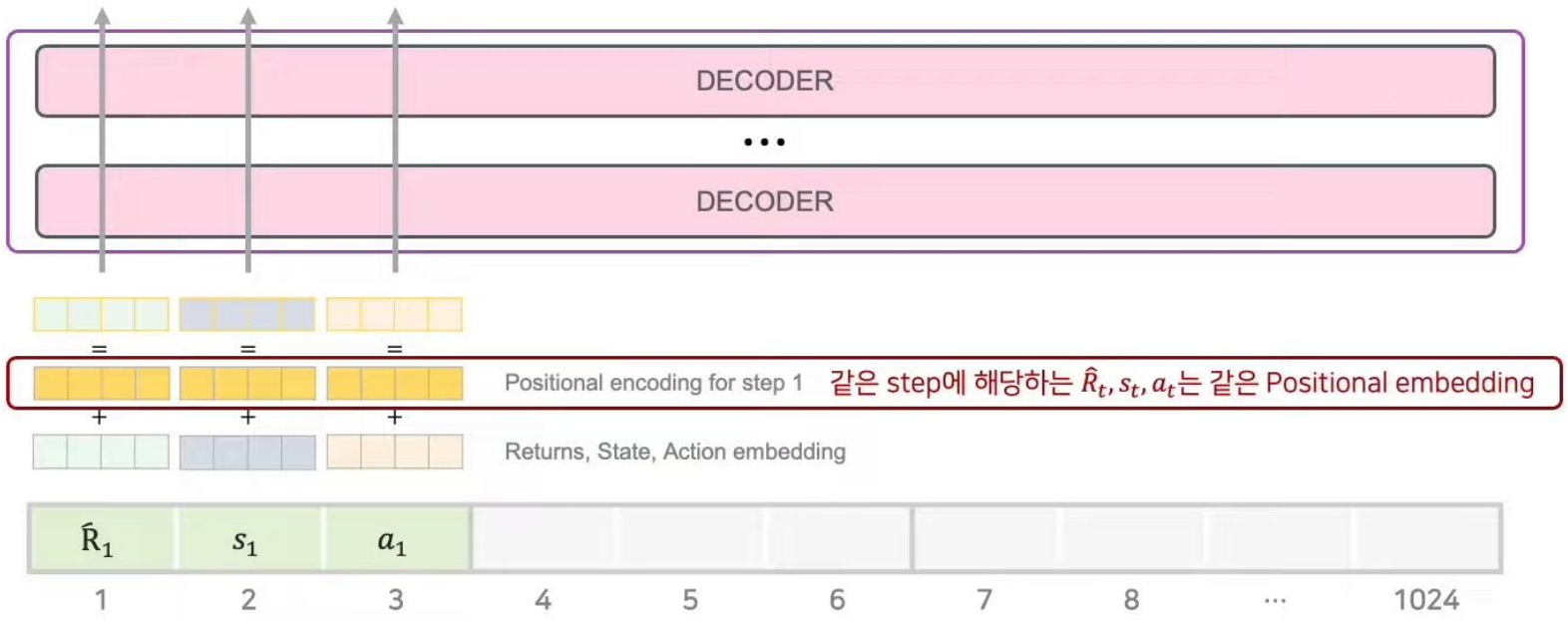
- **Embedding Layer**
  - Token embedding을 얻기 위해 embedding dimension에 raw input을 projection한 다음 layer normalization를 수행
  - Visual input이 있는 environment의 경우 state는 linear layer 대신 convolution encoder에 입력



Method 설명

Training :
길이가 K인 sequence를 dataset에서 샘플링했다.

- **Positional Embedding**
  - 기존 transformer에서의 positional embedding이 아닌 세 개의 tokens에 대응되는 하나의 time-step에 대한 것
  - 같은 time-step에 해당하는 Return-to-go, State, Action Embedding에는 같은 Positional Embedding이 더해진다.



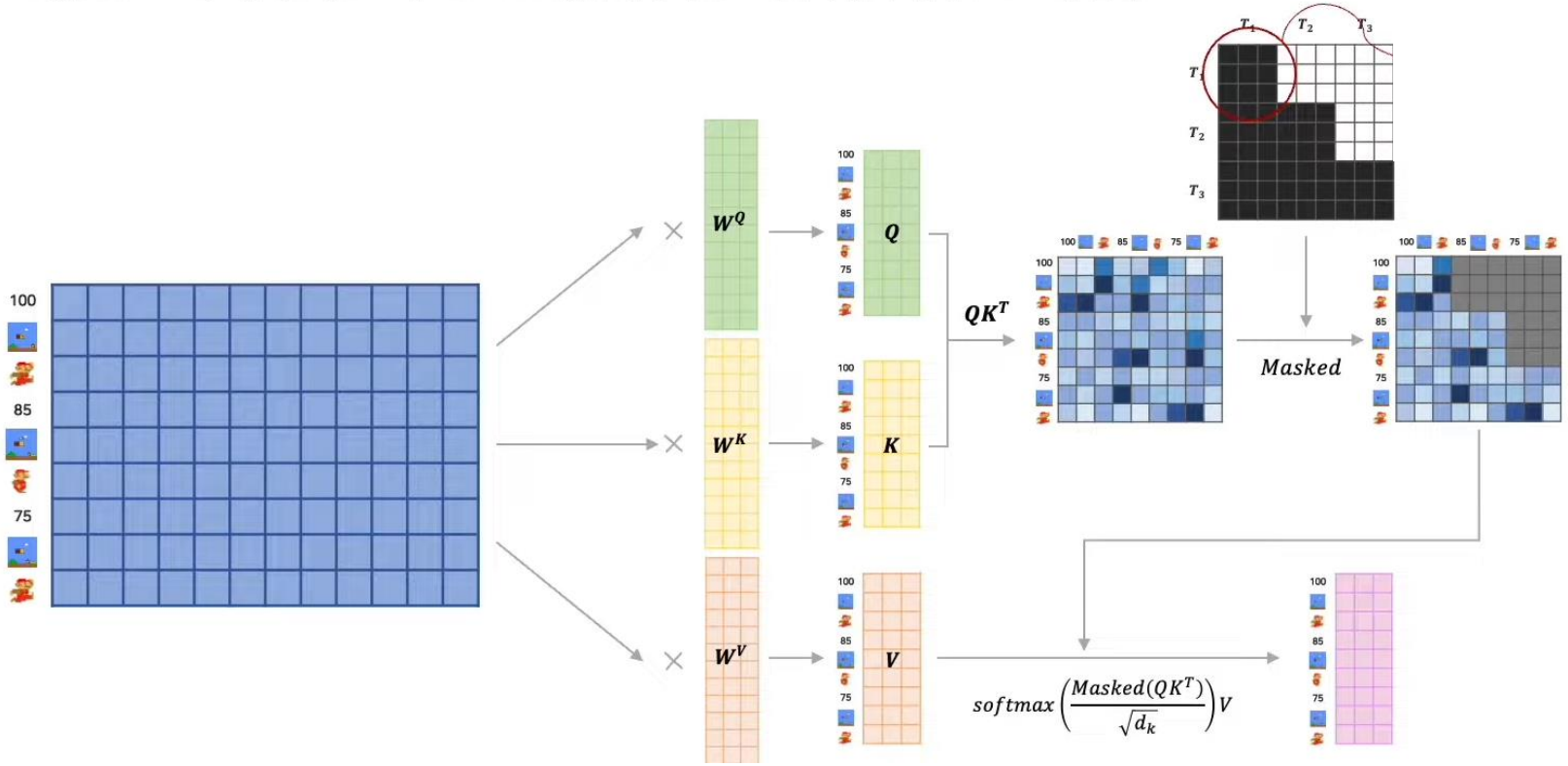
Method 설명

- **Decoder Layer**
  - Decoder Block N개를 stack한 구조 (N : Atari 6개, Gym 3개)

Training :
길이가 K인 sequence를 dataset에서 샘플링했다.

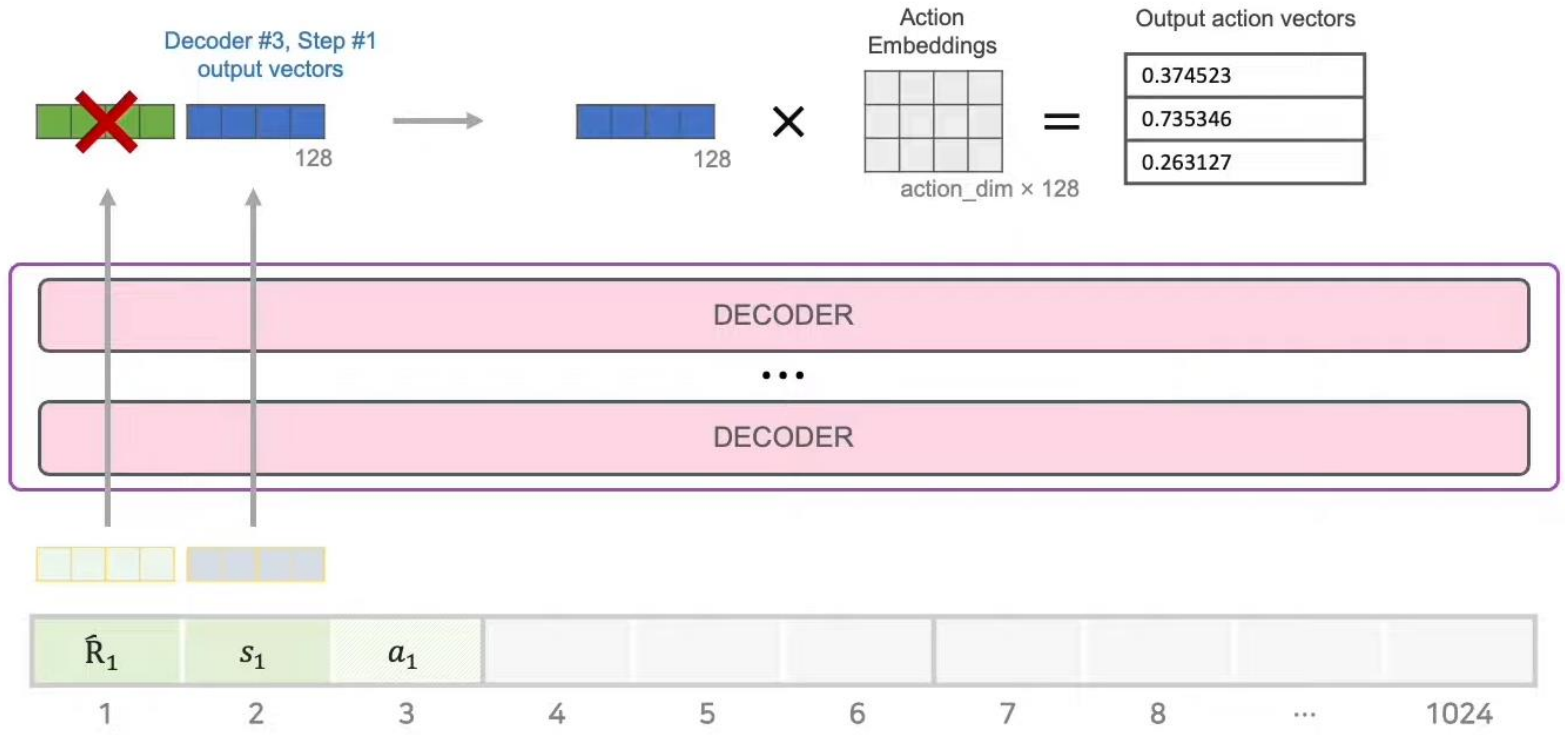
Method 설명

- Masked Self-Attention
  - 해당 time-step 뒤의 trajectory embedding에 대해서는 고려하지 않기 위해 masking 한다.



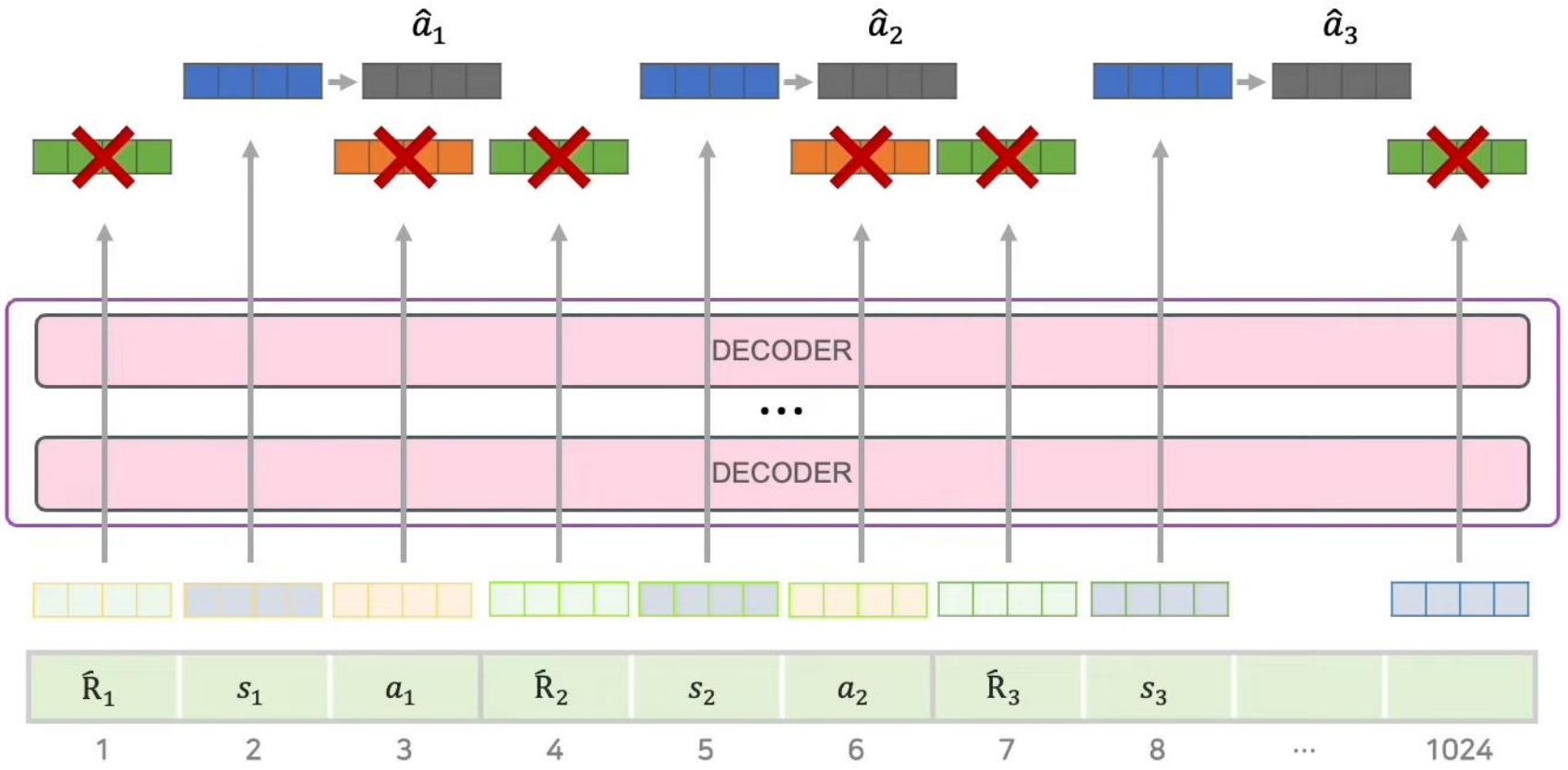
Method 설명

- **Output Layer**
  - Decoder의 output vector 중 state embedding의 output vector에 해당하는 vector들을 이용하여 다음 action을 예측



Method 설명

- Output Layer
  - Decoder의 output vector 중 state embedding의 output vector에 해당하는 vector들을 이용하여 다음 action을 예측



Method 설명

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
```

Evaluations on Offline RL Benchmarks

- **TD learning**: most of these methods use an action-space constraint or value pessimism, and will be the most faithful comparison to Decision Transformer, representing standard RL methods. A state-of-the-art model-free method is Conservative Q-Learning (CQL) [14] which serves as our primary comparison. In addition, we also compare against other prior model-free RL algorithms like BEAR [18] and BRAC [19].
- **Imitation learning**: this regime similarly uses supervised losses for training, rather than Bellman backups. We use behavior cloning here, and include a more detailed discussion in Section 5.1.

  - DT의 비교 대상으로 TD learning과 Imitaion Learning이 있다.
  - 비교 환경은 Atari Game과 Open AI Gym에서 4개의 태스크(D4RL)에서 비교할것이다.

Evaluations on Offline RL Benchmarks

Atari 환경

Game	DT (Ours)	CQL	QR-DQN	REM	BC
Breakout	$\mathbf{267.5 \pm 97.5}$	211.1	17.1	8.9	138.9 ± 61.7
Qbert	15.4 ± 11.4	$\mathbf{104.2}$	0.0	0.0	17.3 ± 14.7
Pong	106.1 ± 8.1	$\mathbf{111.9}$	18.0	0.5	85.2 ± 20.0
Seaquest	$\mathbf{2.5 \pm 0.4}$	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.

※DT의 K 길이는 30이다. (Pong에서만 K =50)

Evaluations on Offline RL Benchmarks

Open AI gym 환경

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 01:

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).

- DT가 데이터 subset의 imitation learning으로 작용하는지를 알아보기위해서 Percentile Behavior Cloning (%BC) 를 제안하고 이와 비교 해보기로 한다.
- Percentile Behavior Cloning (%BC) 는 episode return의 순서에 따라 나열한 dataset의 top X%의 timestpes의 행동만을 cloning 한다.
- 데이터가 충분히 많을때, DT를 BC와 CQL과 비교 했을때 대부분의 환경에서 DT가 BC와 비등한 성능을 보여준다.

Discussion 01:

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

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.

- 반대로 데이터가 적을때는 대부분의 환경에서 DT가 BC보다 압도적으로 뛰어난 성능을 보인다.
- 이는 DT가 단순히 데이터 subset에서 imitation Learning를 시행하는 거보다 더 효과적이다는걸 알 수 있다.

Discussion 02:

How well does Decision Transformer model the distribution of returns?

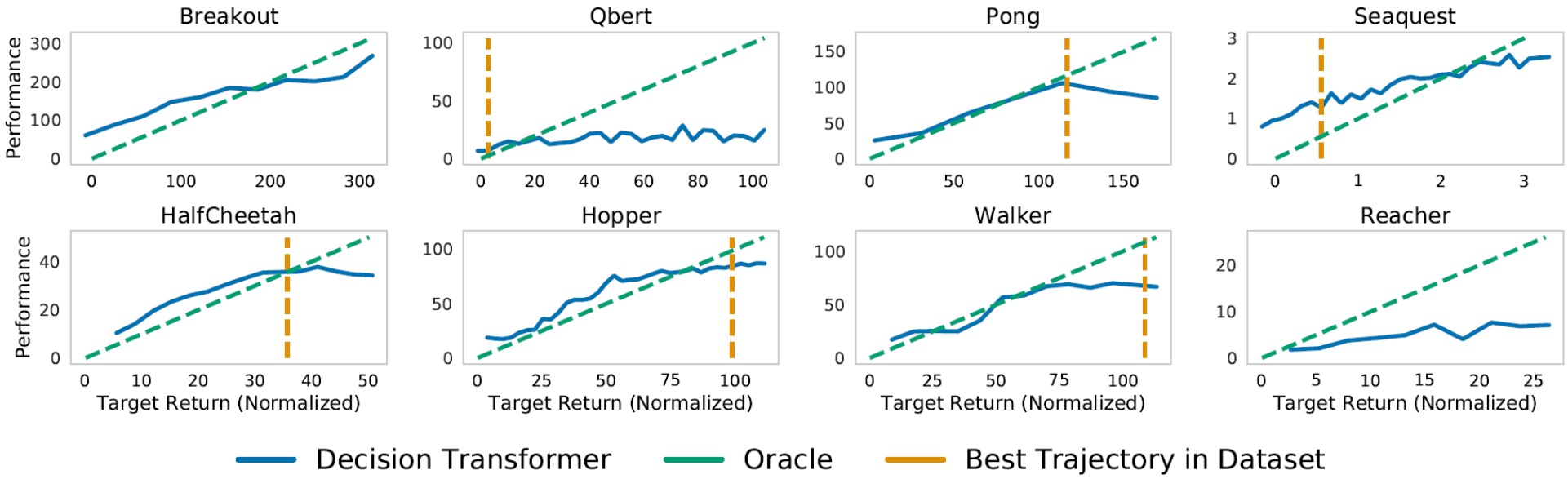


Figure 4: Sampled (evaluation) returns accumulated by Decision Transformer when conditioned on the specified target (desired) returns. **Top:** Atari. **Bottom:** D4RL medium-replay datasets.

- 목표하는 return과 실제의 return은 상당한 연관성을 가지고 있다.
- 대부분의 task에서의 return 값이 목표 return값과 거의 일치한다.

Discussion 03:

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).

- 이전에는 K= 1이여도 충분하다고 판단했었다. (TD-Learning) 하지만 K가 클때 DT에서 보여주듯 많은 양의 과거의 데이터가 있을때 DT는 엄청난 성능을 보여준다.
- 한 가정으로 만약 policy들의 distribution을 representing 할때, contect는 transformer가 어떤 policy 그 action을 생성했는지 알아볼수 있고, dynamics를 더 잘 학습하게 해주는 거다

Discussion 04:

Does Decision Transformer perform effective long-term credit assignment?

Figure 2: Key-To-Door environments visual. The agent is represented by the beige pixel, key by brown, apples by green, and the final door by blue. The agent has a partial field of view, highlighted in white.

1. 키가 같이 놓여져 있는 방에 agent가 등장
2. Agent가 빈방에 위치되어 진다.
3. 마지막으로 문이 있는 방에 놓여진다.

* Agent가 첫번째 페이즈에서 키를 줏었을때만 reward를 받는다.

- 이 문제는 credit assignment에서 너무 어렵다. 그 이유는 credit가 에피소드의 처음부터 끝까지 전파(propagate)되어 져야 한다.

※여기서 credit는 value와 동의어라 보면된다.

Discussion 04:

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.

- 표에서 봤듯이 DT value를 propagate함에 있어 뛰어난 성능을 보이는데에 반해 CQL은 그 성능이 많이 떨어진다.

Discussion 05:

Can transformers be accurate critics in sparse reward settings?

- 이전 질문에서는 transformer가 효율적인 policy를 생성(actor)할수 잇는지를 물어봤다면 이번에는 효율적인 critic이 될수 있는지를 알아보고자 한다.

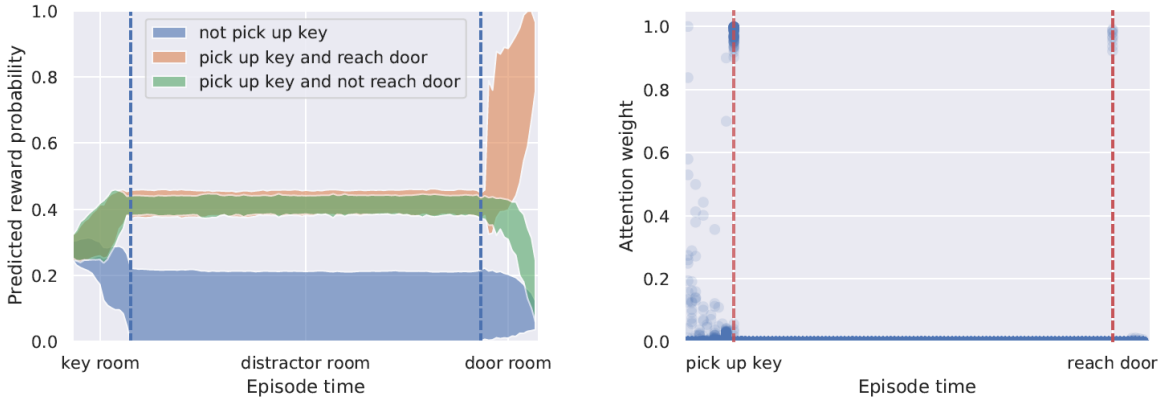


Figure 5: **Left:** Averages of running return probabilities predicted by the transformer model for three types of episode outcomes. **Right:** Transformer attention weights from all timesteps superimposed for a particular successful episode. The model attends to steps near pivotal events in the episode,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

- Key to door 환경에서 action 토큰에 더 나아가 return 토큰도 추가 하였다. 첫번째 return token은 주어지지 않지만 대신 initial distribution으로 예측한다.
- 왼쪽 그림으로 통해서 transformer가 지속적으로 reward probability 을 업데이트 한다는 걸 알수 있다. (에피소드 동안 각각의 이벤트를 따라서)
- 더 나아가 에피소드 중에서 크리티컬한 이벤트에서 엄청 높은 attention weight 가 주어진다는걸 발견된다. 그리고 이게 더 예리한 value prediction을 가능케 한다.

Discussion 06:

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.

- TD learnin의 약점은 sparse reward 환경에 퍼포먼스가 안좋다는 거다
- 이를 평가 하기 위해서 delayed return 버전의 D4RL benchmark을 고려함.
  (trajectory 진행중에 아무 reward 못받다가 마지막 timestep에 가서 축적된 reward를 받는다)
- DT, CQL, BC를 비교했을때 sparse reward 환경에서 DT가 높은 성능을 보인다.

Discussion 07:

Why does Decision Transformer avoid the need for value pessimism or behavior regularization?

- DT와 기존 offline RL의 차이점은 DT는 좋은 퍼포먼스를 위해 policy regularization 또는 conservatism가 필요치 않다
- TD learning을 기반으로 한 알고리즘은 근사 value function을 학습하고 policy를 이에 최적화 시키면서 향상시킨다. 이미 학습한 함수에 최적화 한다는 행동은 exacerbate 하고 부정확한 근사 value funtion을 이용하는게 policy improvement에 악영향을 끼칠수 있다.
- DT는 학습한 함수를 목적함수로 이용하여 최적화 하지 않기에 regularization과 conservatism 할 필요가 없다.

Discussion 08:

How can Decision Transformer benefit online RL regimes?

- Online RL에서 의미 있는 성과를 이룰거라 믿고 있다.
- Go-Explore 같이 memorization engine이 강력하고, conjunction with powerful exploration 을 하는 알고리즘 처럼 serve 할수 있다고 생각한다.

Conclusion

- Decisoin Transformer 라는 RL과 Language/Sequence 모델을 하나로 통합하는 방법을 제안했다.
- Offline RL에서 기존 알고리즘에 비해 성능이 비등하거나 더 뛰어났다.
- 향후에 좀 더 큰 Transformer 모델이 RL에 적용 하는 연구들이 더 활발했으면 한다.
- Real World에 적용 시킬때 MDP에 Transformer를 적용시킬때 발생하는 에러와 Explore가 덜 되는 부정적인 결과를 내는거를 이해하는게 중요하다.

End of the presentation

참고 자료 :
https://www.youtube.com/watch?v=vRg8Tf_B608
https://sites.google.com/berkeley.edu/decision-transformer
https://github.com/kzl/decision-transformer