

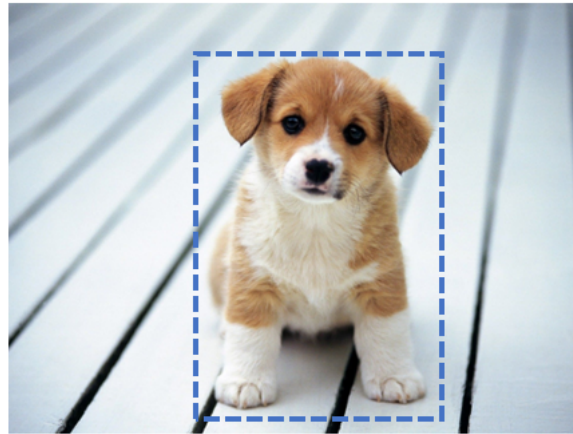
Deep Reinforcement Learning

Qiwen Cui Xinqi Wang Runlong Zhou

Supervised Learning

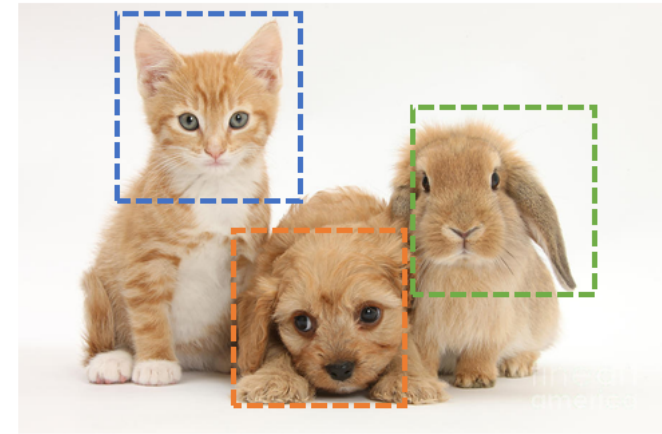
- Data: (x, y)
- Goal: Learn a function $f(x)=y$
- Examples: Classification, Regression, ...

★ Single-label classification



Dog

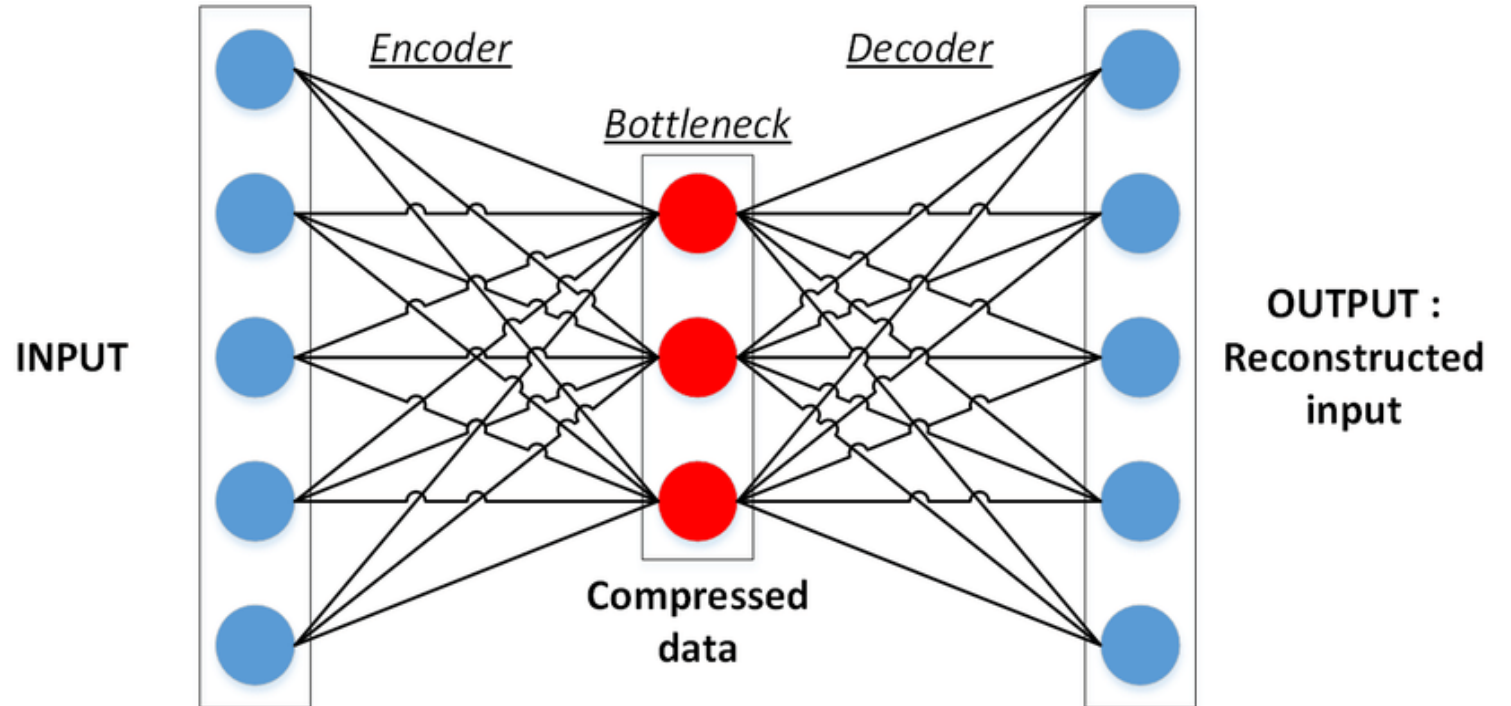
★ Multi-label classification



Cat, Dog, Rabbit

Self-supervised Learning

- Data: x
- Goal: Learn underlying structure of the data
- Examples: Representation Learning, Contrastive Learning, Autoregressive Pretraining

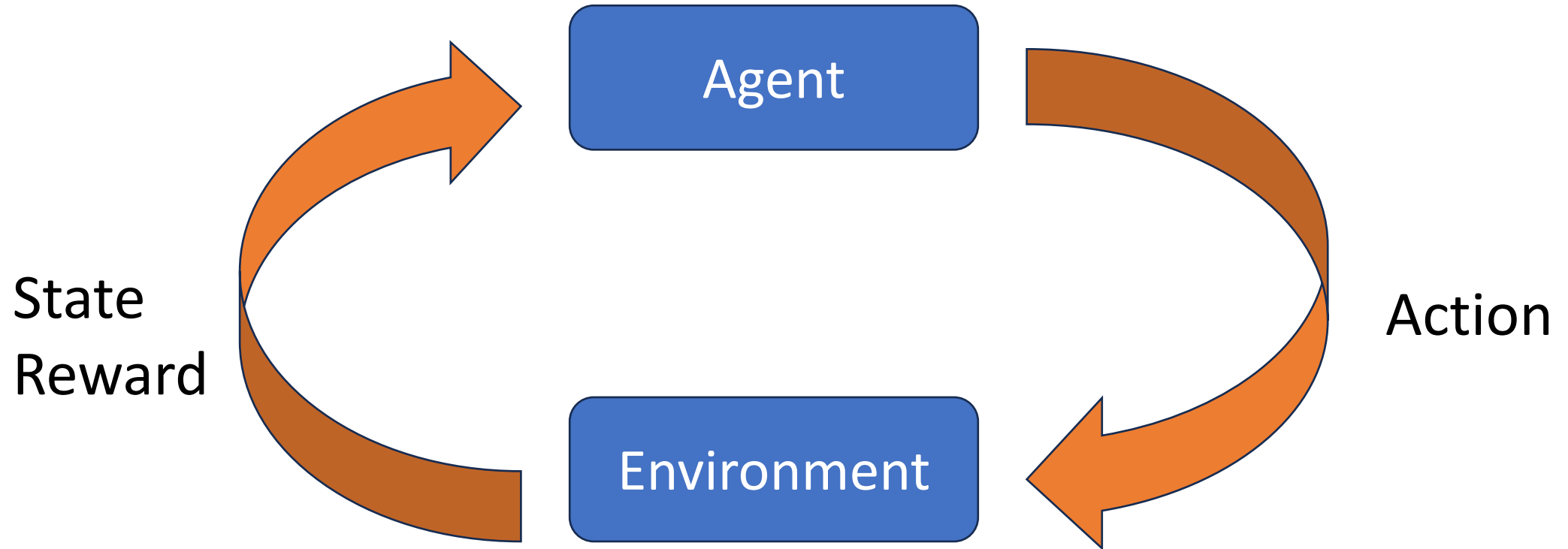


Reinforcement Learning

- Goal: Learn a policy to maximize reward
- Examples: Chess, Go, Poker, Self-driving

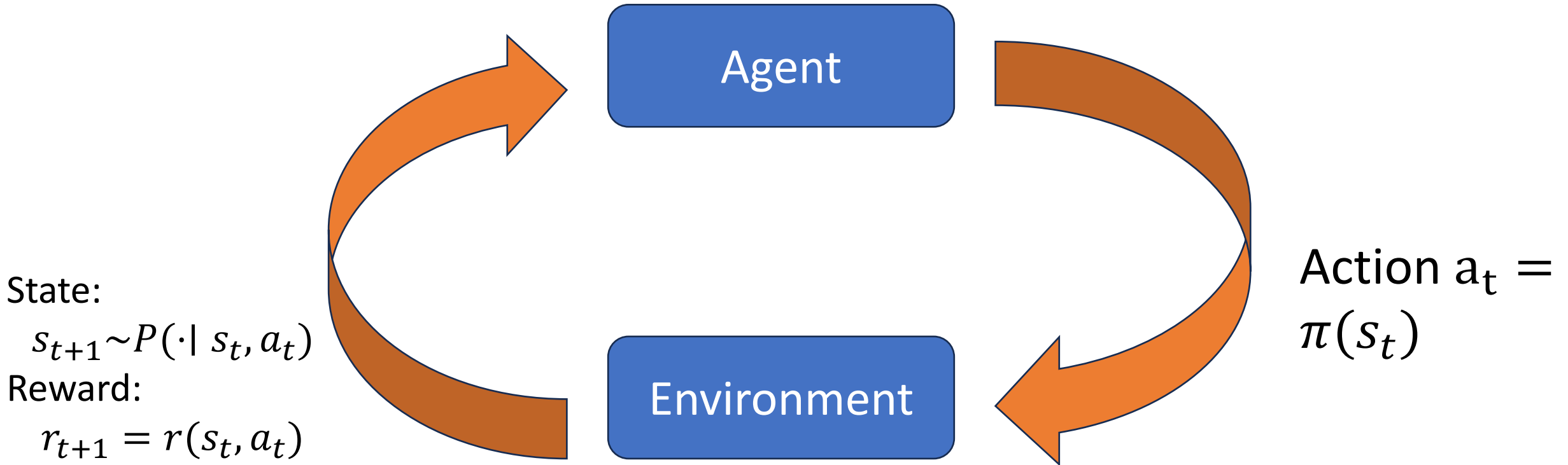


Markov Decision Process



- Goal: Collect as much reward as possible.

Markov Decision Process



Maximize total discounted reward $\sum \gamma^t r_t$.

Markov Decision Process

- Policy: $\pi(s) = a$.
- Discount factor: $\gamma \in (0,1)$.
- Value function: $V^\pi(s_0) = \mathbb{E}_\pi[\sum_t \gamma^t r_t]$, where $s_0, a_0, r_0, s_1, a_1, r_1, \dots$ is a trajectory sampled by using policy π .
- Q function: $Q^\pi(s_0, a_0) = \mathbb{E}_\pi[\sum_t \gamma^t r(s_t, a_t)]$.
- Optimal policy: $\pi^* = \operatorname{argmax}_\pi V^\pi(s)$.
- There exists an optimal policy that achieves the argmax for all s **simultaneously!**

Optimal Q Function

- Optimal Q function: $Q^{\pi^*}(s_0, a_0) = \mathbb{E}_{\pi^*}[\sum_t \gamma^t r(s_t, a_t)]$.
- Property: $\pi^*(s) = \operatorname{argmax}_a Q^{\pi^*}(s, a)$.
- If we know Q^* , we know π^* .

Reinforcement Learning

- If we know $r(s, a)$ and $P(s' | s, a)$, we can use dynamic programming to solve the optimal policy.
- How to learn the optimal policy **without** the knowledge of $r(s, a)$ and $P(s' | s, a)$?
- Collect **samples!**

Challenge: Large State Space

- 3^{361} possible board configurations in Go.
- Impossible to enumerate.
- Theorem: $\Omega(SA)$ samples are necessary for learning MDP without structures, where S is # of states and A is # of actions.

Function Approximation

- Challenge in RL: large state and action space.
- Many states and actions are similar and have similar Q^{π^*} .
- Use a function class $\mathcal{F} = \{f_\theta\}$ to approximate Q function.
- Suppose we have a dataset $\mathcal{D} = \{Q^{\pi^*}(s, a)\}$, then we can fit a f_θ to approximate Q^{π^*} :

$$\theta^* = \operatorname{argmin}_\theta \sum_{(s,a) \in \mathcal{D}} \left(f_\theta(s, a) - Q^{\pi^*}(s, a) \right)^2.$$

Offline Reinforcement Learning

- Dataset: trajectories $s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_T$ sampled from some behavior policy π_b .
- Challenge: unknown $Q^{\pi^*}(s, a)$.

Q-learning

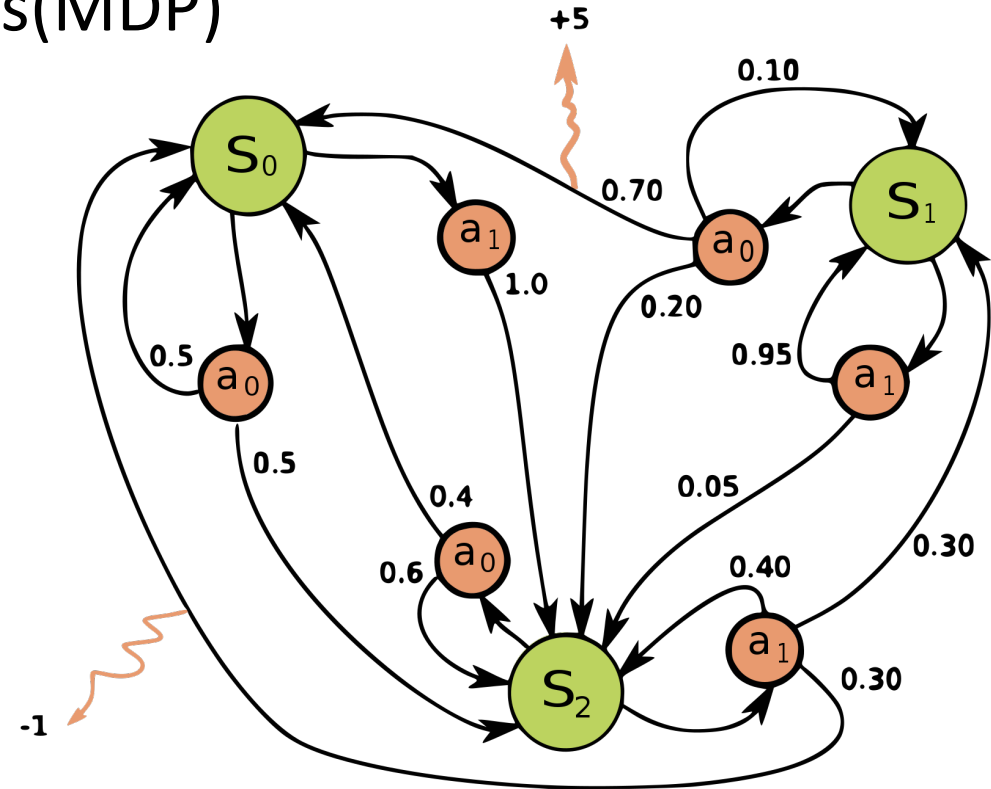
- Reminder: Markov-Decision Process(MDP)

State:

$$s_{h+1} = P(\cdot | s_h, a_h)$$

Reward:

$$r_{h+1} = r(s_h, a_h)$$



Q-learning

- Value-based method:
 - Evaluate all the states, then find the action leading to the best state.
- Reminder: Value function and Q function:

- $V_{\pi}(s) = E_{\pi}[\sum_h \gamma^h r_h \mid s]$

- We need to know which action leads to the given reward:

- $Q_{\pi}(s, a) = E_{\pi}[\sum_h \gamma^h r_h \mid s, a]$

Q-learning

Q-function:

$$Q_{\pi}(s, a) = E_{\pi} \left[\sum_h \gamma^h r_h \mid s, a \right]$$

- Target: derive the Q function for the optimal policy π^* , Q^*
- How to solve this system?

- Of course, we can use Monte Carlo's Method to estimate Q function.
- But it takes $\Omega(SA^h)$ sample trajectories.

- Can we do better?

Q-learning: Tabular learning

Q-function:

$$Q_{\pi}(s, a) = E_{\pi} \left[\sum_h \gamma^h r_h \mid s, a \right]$$

- Notice that Q function should satisfy the successor relationship;
- Bellman's Equation:
 - $Q^*(s, a) = r(s, a) + \gamma E_{\pi^*} [V^*(s') \mid s, a]$
 - $V^*(s) = \max_a Q^*(s, a)$
- Then we can solve it with polynomial samples!

Q-learning: Tabular learning

- First, initialize $Q(\cdot) = 0$;
- Then we do iterative DP:
 - Until convergency, do:

- For $(s, a) \in S \times A$:
 - Update Q: $Q(s, a) \leftarrow \frac{1}{N_{s,a}} \sum_{s_i=s, a_i=a} (r_i + \gamma V(s_{i+1}))$

- For $s \in S$:

Here $N_{s,a}$ is the counter of (s,a) in dataset.

- Update V: $V(s) \leftarrow \max_a Q(s, a)$

Deep Q Network (DQN)

- When we combine Deep Learning with Q-learning, we get DQN.
- Reminder: function approximation
 - Structure/function class: MLP, CNN, Transformer, etc.
 - Solve the Bellman's Equation with gradient descent!
 - $Q^*(s, a) = r(s, a) + \gamma E_{\pi^*} [V^*(s') \mid s, a]$
 - $V^*(s) = \max_a Q^*(s, a)$
 - Loss function:

$$L(\theta) = \mathbf{E}_{\theta} [(Q_{\theta}(s, a) - r(s, a) - \gamma \mathbf{E}[V_{\theta}(s') \mid s, a])^2]$$

Deep Q Network (DQN)

- Loss function:

$$L(\theta) = \mathbf{E}_{\theta}[(Q_{\theta}(s, a) - r(s, a) - \gamma \mathbf{E}[V_{\theta}(s') | s, a])^2]$$

- Estimated loss:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^N [Q_{\theta}(s_i, a_i) - r_i - \gamma \max_{a'} Q_{\theta}(s'_i, a')]^2$$

- Other tricks:
 1. Double network trick for stronger stability;
 2. Replay buffer for higher sample efficiency.

DQN: double network structure

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^N [Q_{\theta}(s_i, a_i) - r_i - \gamma \max_{a'} Q_{\theta'}(s'_i, a')]^2$$

- Evaluate network: trained network θ
 - Updated in each iteration
 - The first Q is the evaluate network
- Target network: temporal copy of evaluate network θ'
 - Updated at regular intervals
 - The second Q is fixed to be target network
- Avoid overfitting problem;
- Don't need to solve a max problem in each iteration;
- Stabilize the training process.

DQN: experience replay

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^N [Q_{\theta}(s_i, a_i) - r_i - \gamma \max_{a'} Q_{\theta}(s'_i, a')]^2$$

- Problem: batch size is very small compared with the dataset
 - Each batch may only contain the transitions from a single trajectory
 - Not mutually independent!
- Notice that we only need transitions $\{s_i, a_i, r_i, s'_i\}$, instead of complete trajectories.
- Solution: In each iteration, we randomly sample data from the replay buffer to form the training batch.
- The replay buffer can be the offline dataset, or the data collected with latest policy, which gives better sample efficiency.

Policy-Gradient

- Sometimes we don't want to estimate the Value function!
 - Value function approximation can be extremely tricky;
 - Empirical experiments tell us simpler algorithm leads to better performance;
 - We need to solve an argmax/max problem for each update, which can be very expensive.

$$\pi(s) \leftarrow \arg \max_a \{ \mathbf{E}_\pi [\sum \gamma^h r_h \mid s, a] \}$$

- Policy-Gradient(PG) directly optimize the policy!
- Directly approximate $\pi^*(\cdot)$ with DNN.
 - Now we use π_θ to denote the policy learnt.

Policy-Gradient

- Denote the probability of getting a certain trajectory τ as $P(\tau, \theta)$, and the corresponding reward as $R(\tau)$.

$$P(\tau, \theta) = \prod_h \pi_\theta(a_h | s_h)$$

$$R(\tau) = \sum_h \gamma^h r_h$$

- Target: maximize $J(\theta) = E_{\pi_\theta}[\sum_h \gamma^h r_h] = \sum_h P(\tau, \theta)R(\tau)$
- Gradient ascent: $\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$

- Great so far!

- The problem lies in the estimation of $\nabla_\theta J(\theta)$.

Policy-Gradient

- Target: maximize $J(\theta) = E_{\pi_{\theta}} [\sum_h \gamma^h r_h] = \sum_h P(\tau, \theta) R(\tau)$
- Gradient ascent: $\theta \leftarrow \theta + \eta \nabla_{\theta} J(\theta)$
- Directly calculation of the gradient of empirical reward gives:

$$J(\theta) \approx \frac{1}{N} \sum_{i=1}^N R(\tau_i),$$

$$\nabla_{\theta} J(\theta) \approx \nabla_{\theta} \left[\frac{1}{N} \sum_{i=1}^N R(\tau_i) \right]?$$

- Remember that $R(\tau)$ doesn't depend on θ directly:
 - $P(\tau, \theta) = \prod_h \pi_{\theta}(a_h | s_h)$
 - $R(\tau) = \sum_h \gamma^h r_h$

Policy-Gradient

- Target: maximize $J(\theta) = E_{\pi_\theta} [\sum_h \gamma^h r_h] = \sum_\tau P(\tau, \theta) R(\tau)$
- Directly calculation of the gradient of empirical reward gives:

$$J(\theta) \approx \frac{1}{N} \sum_{i=1}^N R(\tau_i),$$
$$\nabla_\theta J(\theta) \approx \nabla_\theta \left[\frac{1}{N} \sum_{i=1}^N R(\tau_i) \right]$$

- Problem: We are not calculating the exact reward with probability, but with sampling!
 - Therefore, we cannot backpropagate the gradient to DNN;
 - (Sad news, can't leave differential to `loss.backward()` this time)

Policy-Gradient

- Target: maximize $J(\theta) = E_{\pi_\theta} [\sum_h \gamma^h r_h] = \sum_h P(\tau, \theta) R(\tau)$

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_\tau \nabla_\theta P(\tau, \theta) R(\tau) \\ &= \sum_\tau \frac{P(\tau, \theta)}{P(\tau, \theta)} \nabla_\theta P(\tau, \theta) R(\tau) \\ &= \sum_\tau P(\tau, \theta) \frac{\nabla_\theta P(\tau, \theta)}{P(\tau, \theta)} R(\tau) \\ &= \sum_\tau P(\tau, \theta) \nabla_\theta \log P(\tau, \theta) R(\tau) \\ &= \mathbf{E}_{\pi_\theta} [R(\tau) \nabla_\theta \log P(\tau, \theta)]\end{aligned}$$

- Good! The gradient can be also understood as an expectation!
- Therefore, the empirical update function is:

$$\theta \leftarrow \theta + \frac{\eta}{N} \sum_{i=1}^N R(\tau_i) \nabla_\theta \log P(\tau_i, \theta)$$

Decision Transformers



- Language modeling: autoregressive conditional sequence modeling

- Predict next **token** (\approx word) with some probability

$$P(\text{"you"} | [\text{"How"}, \text{" "}, \text{"are"}, \text{" "}])$$

- **Autoregressive:** sample, and predict next

$$P(\text{"? "} | [\text{"How"}, \text{" "}, \text{"are"}, \text{" "}, \text{"you"}])$$

- Just like **policy** in RL!

$$\pi(a_t | s_1, a_1, r_1, \dots, s_t)$$

Decision Transformers for Offline RL

- Offline dataset:

- Consider **deterministic reward, finite horizon H , and discount $\gamma = 1$**

$$D = \left\{ \tau^i = (s_0^i, a_0^i, r_0^i; s_1^i, a_1^i, r_1^i; \dots; s_H^i, a_H^i, r_H^i) \right\}_{i=1}^N$$

- Decision Transformers:

- **Return-to-go:** $\hat{R}_t = \sum_{h=t}^H r_h$

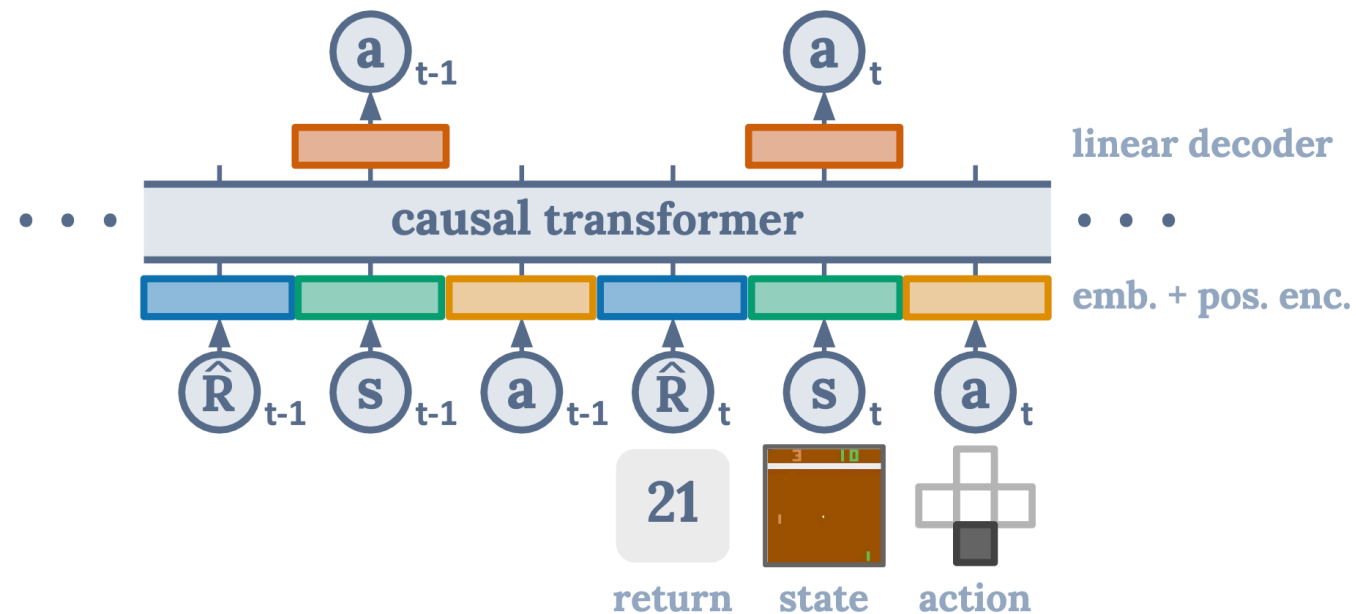
$$D = \left\{ \tau^i = (\hat{R}_0^i, s_0^i, a_0^i; \hat{R}_1^i, s_1^i, a_1^i; \dots; \hat{R}_H^i, s_H^i, a_H^i) \right\}_{i=1}^N$$

Decision Transformers for Offline RL

- Decision Transformers:

- Return-to-go (RTG):** $\hat{R}_t = \sum_{h=t}^H r_h$

$$D = \left\{ \tau^i = (\hat{R}_0^i, s_0^i, a_0^i; \hat{R}_1^i, s_1^i, a_1^i; \dots; \hat{R}_H^i, s_H^i, a_H^i) \right\}_{i=1}^N$$



Decision Transformers for Offline RL

- Decision Transformers:

- Return-to-go (RTG):** $\hat{R}_t = \sum_{h=t}^H r_h$

$$D = \left\{ \tau^i = \left(\hat{R}_0^i, s_0^i, a_0^i; \hat{R}_1^i, s_1^i, a_1^i; \dots; \hat{R}_H^i, s_H^i, a_H^i \right) \right\}_{i=1}^N$$

```
# 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_embs = stack(R_embedding, s_embedding, a_embedding)

    # use transformer to get hidden states
    hidden_states = transformer(input_embs=input_embs)

    # select hidden states for action prediction tokens
    a_hidden = unstack(hidden_states).actions

    # predict action
    return pred_a(a_hidden)
```

```
self.embed_timestep = nn.Embedding(max_ep_len, hidden_size)
self.embed_return = torch.nn.Linear(1, hidden_size)
self.embed_state = torch.nn.Linear(self.state_dim, hidden_size)
self.embed_action = torch.nn.Linear(self.act_dim, hidden_size)
```

Training

- Minibatch of sequence with length K
 - Context length K : use previous K steps to predict next action
 - Slice τ^i into $\tau_{[\max\{j-K+1,1\}:j]}^i$ for $j = 1, 2, \dots, H$

$$\tau_{[l:r]}^i = (\hat{R}_l^i, s_l^i, a_l^i; \dots; \hat{R}_r^i, s_r^i, a_r^i)$$
$$\check{\tau}_{[l:r]}^i = (\hat{R}_l^i, s_l^i, a_l^i; \dots; \hat{R}_r^i, s_r^i)$$

Training

- Loss function
 - **Cross-entropy loss** for discrete action space

$$\mathcal{L}_{\text{decision}} = \sum_{i=1}^N \sum_{j=1}^H -\log \pi(a_j^i | \check{\tau}_{[\max\{j-K+1, 1\}:j]}^i)$$

- **L2 loss** for continuous action space

$$\mathcal{L}_{\text{decision}} = \sum_{i=1}^N \sum_{j=1}^H \mathbb{E}_{a \sim \pi(\cdot | \check{\tau}_{[\max\{j-K+1, 1\}:j]}^i)} (a_j^i - a)^2$$

```
# 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()
```

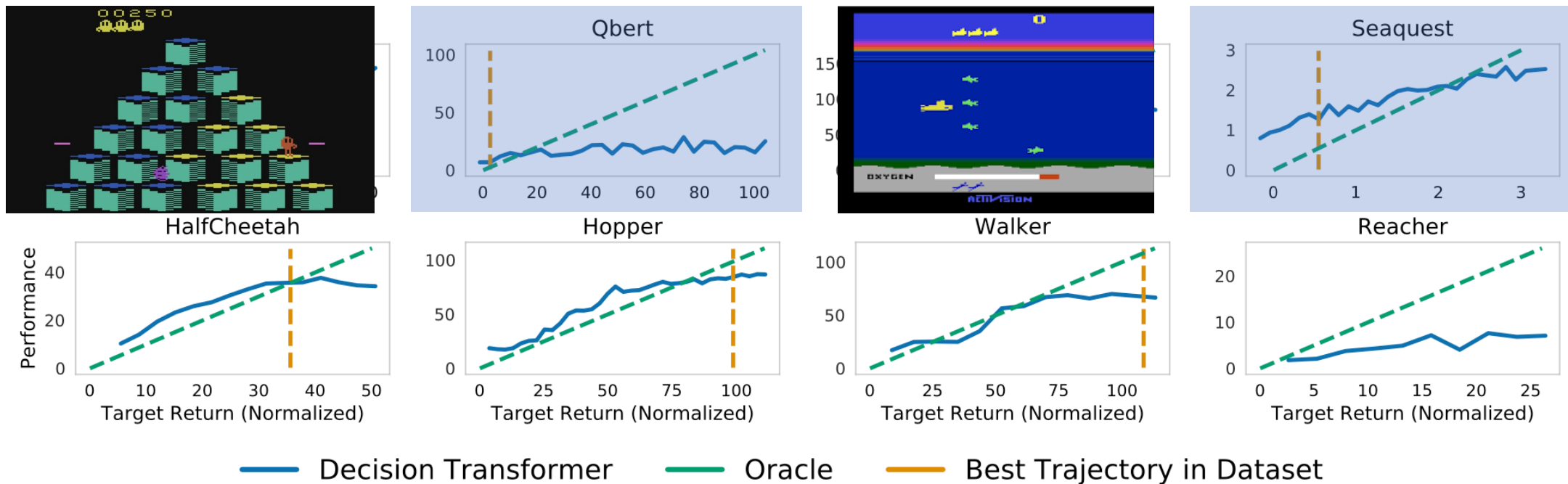
Evaluation

- Set an **initial RTG** (large enough)
- Run the DT and subtract the current return-to-go with the observed reward
- Crop the sequence to length K

```
# 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
```

Results



- **Possible** to outperform the best trajectory in dataset

Results

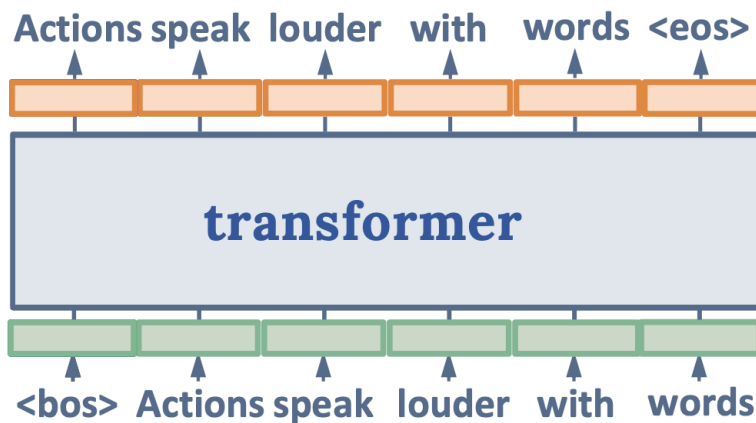
Dataset	Environment	DT (Ours)	CQL	BEAR	BRAC-v	AWR	BC
Medium-Expert	HalfCheetah	86.8 ± 1.3	62.4	53.4	41.9	52.7	59.9
Medium-Expert	Hopper	107.6 ± 1.8	111.0	96.3	0.8	27.1	79.6
Medium-Expert	Walker	108.1 ± 0.2	98.7	40.1	81.6	53.8	36.6
Medium-Expert	Reacher	89.1 ± 1.3	30.6	-	-	-	73.3
Medium	HalfCheetah	42.6 ± 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

- CQL: conservative Q-learning
- BEAR: off-policy Q-learning
- BRAC-v: behavior regularized offline RL
- AWR: advantage-weighted regression
- BC: behavior cloning

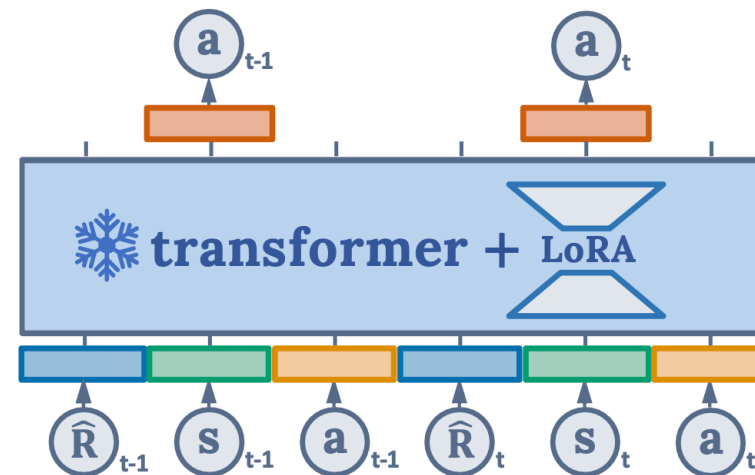
Pretraining DTs on Language Tasks

- Use a pretrained language model (GPT2) as initialization

large language model pre-train



downstream offline RL



Pretraining DTs on Language Tasks

- Use MLP for embedding

```
self.embed_timestep = nn.Embedding(max_ep_len, hidden_size)
self.embed_return = torch.nn.Linear(1, hidden_size)
self.embed_state = torch.nn.Linear(self.state_dim, hidden_size)
self.embed_action = torch.nn.Linear(self.act_dim, hidden_size)
```



$$u(x_t) = W_x^{(1)} \text{GELU}(W_x^{(0)} x_t) + \omega(t), x \in \{\hat{R}, s, a\}$$

$$\text{GELU}(x) = xP(X \leq x) = x\Phi(x) = x \cdot \frac{1}{2} \left[1 + \text{erf}(x/\sqrt{2}) \right]$$

- $X \sim \mathcal{N}(\mu = 0, \sigma^2 = 1)$

Pretraining DTs on Language Tasks

- **Parameter Efficient Finetuning (PEFT): Low-rank Adaptation (LoRA)**
 - High efficiency
 - Avoid overfitting in full finetuning

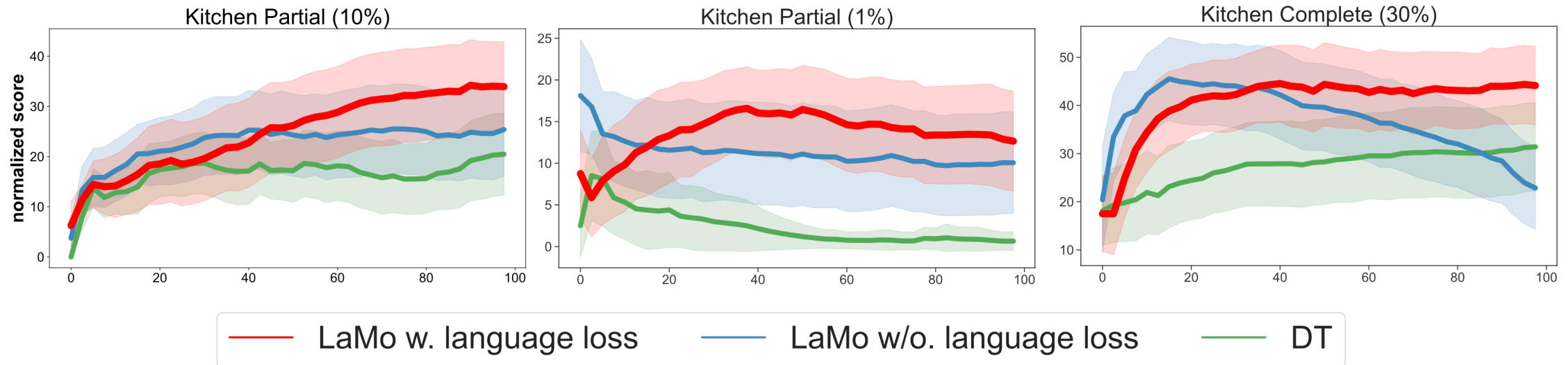
Pretraining DTs on Language Tasks

- Language prediction as an auxiliary objective

- WikiText dataset

- $\mathcal{L}_{\text{language}} = \sum_i -\log T(w_{i+1} | w_1, \dots, w_i)$

$$\mathcal{L} = \mathcal{L}_{\text{decision}} + \lambda \mathcal{L}_{\text{language}}$$



References

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Chen, Lili, et al. "Decision transformer: Reinforcement learning via sequence modeling." *Advances in neural information processing systems* 34 (2021): 15084-15097.

Zheng, Qinqing, Amy Zhang, and Aditya Grover. "Online decision transformer." *international conference on machine learning*. PMLR, 2022.

Shi, Ruizhe, et al. "Unleashing the Power of Pre-trained Language Models for Offline Reinforcement Learning." *arXiv preprint arXiv:2310.20587* (2023).

Brandfonbrener, David, et al. "When does return-conditioned supervised learning work for offline reinforcement learning?." *Advances in Neural Information Processing Systems* 35 (2022): 1542-1553.

Zhou, Zhaoyi, et al. "Free from Bellman Completeness: Trajectory Stitching via Model-based Return-conditioned Supervised Learning." *arXiv preprint arXiv:2310.19308* (2023).