Weighting Online Decision Transformer with Episodic Memory for Offline-to-Online Reinforcement Learning

Xiao Ma[1] and Wu-Jun Li[1]

***Abstract*— Offline reinforcement learning (RL) has been shown to be successfully modeled as a sequence modeling problem, drawing inspiration from the success of Transformers. Offline RL is often limited by the quality of the offline dataset, so offline-to-online RL is a more realistic setting. Online decision transformer (ODT) is an effective and representative sequence modeling-based offline-to-online RL method. Despite its effectiveness, ODT still suffers from the sample inefficiency problem during the online fine-tuning phase. This sample inefficiency problem arises because the agent treats all state-action pairs in the replay buffer equally when trying to learn from the replay buffer. In this paper, we propose a simple yet effective method, called weighting online decision transformer with episodic memory (WODTEM), to improve sample efficiency. We first attempt to introduce an episodic memory (EM) mechanism into the sequence modeling-based RL methods. By utilizing the EM mechanism, we propose a novel training objective with a weighting function, based on ODT, to improve sample efficiency. Experimental results on multiple tasks show that WODTEM can improve sample efficiency.**

I. Introduction

Offline reinforcement learning (RL) [20] has attracted attention in recent years due to its applicability in real-world scenarios where large-scale online data collection might be costly or unsafe. Researchers have explored the potential of training robust agents by leveraging deep neural networks and large-scale offline datasets [17], [18], and have proposed some traditional offline RL methods that try to fit value functions or compute policy gradients [17], [18], [25]. Recently, the Transformer architecture [31] has become a cornerstone in multiple domains, such as natural language processing [6], [4], computer vision [7], and robotics [28], [14]. Motivated by the success of the Transformer architecture, recent works show that the Transformer architecture can be adopted to conduct one whole sequential decision-maker directly [5], [13], [10], [34], [32], [35], [21], [27]. One of the representative methods is decision transformer (DT) [5]. Here, the RL problem is modeled as a sequence modeling task, wherein an agent learns to generate a policy auto-regressively using an offline dataset and yields high returns. This category of methods is called sequence modeling-based offline RL methods.

While these sequence modeling-based offline RL methods have achieved promising performance in various tasks [21], [15], they still suffer the problem that agents trained through these methods might end up with sub-optimal policies due to the limited quality of the offline dataset. To overcome this problem, introducing online fine-tuning is necessary further to enhance agents' performance [25], [38], resulting in a more realistic setting, offline-to-online RL. In offline-to-online RL, agents can train their policies using the offline dataset (offline training phase) and then fine-tune their policies through small-scale online interactions from the environments (online fine-tuning phase). Recently, researchers have proposed some sequence modeling-based offline-to-online RL methods [38], [33]. Although sequence modeling-based offline-to-online RL methods benefit from online fine-tuning, they still suffer from sample inefficiency. This sample inefficiency problem occurs because the training phase of the agent's policy equally learns from all state-action pairs in the replay buffer. In other words, it assigns the same weight to each state-action pair in the replay buffer, regardless of whether some state-action pairs are more valuable or informative than others.

Episodic memory (EM) mechanism [29], [1] can record the historical optimal actions and the corresponding historical optimal discounted returns for states across all experienced trajectories, and it has been well applied in traditional RL methods to improve sample efficiency [22], [19], [23], [37]. To the best of our knowledge, the EM mechanism has not been integrated into sequence modeling-based RL methods, especially sequence modeling-based offline-to-online RL methods. We make the first attempt to introduce the EM mechanism into sequence modeling-based RL methods and propose a simple yet effective method, called weighting online decision transformer with episodic memory (WODTEM), to improve sample efficiency. The contributions of this work are briefly outlined as follows:

- WODTEM is the first work to introduce the EM mechanism into sequence modeling-based RL methods.
- We propose an EM mechanism that enables the agent to access the approximate historical optimal discounted return and approximate historical optimal action for each state.
- Utilizing the EM mechanism, we propose a novel training objective with a weighting function, based on ODT [38], to improve sample efficiency. Here, the weighting function enables the agent to pay attention to the state-action pairs in the replay buffer whose discounted returns are higher than their corresponding approximate historical optimal discounted returns.

*This work was supported by the NSFC Project (No.62192783) and Fundamental Research Funds for the Central Universities (No.020214380108).

[1] Xiao Ma and Wu-Jun Li are with the National Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing 210023, PRC. maxiao@smail.nju.edu.cn, liwujun@nju.edu.cn

Corresponding author: Wu-Jun Li.

- Experimental results on multiple tasks show that our method can improve sample efficiency.

II. Related Work

A. Transformers for Sequential Decision-maker

Recent methods have made much progress when formulating the RL problem as a sequence modeling problem [21], which resembles the supervised learning paradigm [8], [30]. These methods predict action sequences conditioned on some task specifications, like states, desired returns, actions, etc. Most of these methods focus on the offline RL setting, where agents' policies are trained on offline datasets. This category of these methods is referred to as sequence modeling-based offline RL methods. The representative sequence modeling-based offline RL methods include DT [5], TT [13], GDT [10], BootT [32], DoC [35], and QDT [34]. These methods focusing on the offline RL setting might result in sub-optimal policies for agents due to the limited quality of the training dataset.

Introducing online fine-tuning can help the agent alleviate the impact of the quality of the training dataset. Hence, offline-to-online RL is a more realistic setting where agents can train their policies using the offline dataset and then fine-tune them through small-scale online interactions from the environments. Building upon DT [5], researchers proposed sequence modeling-based offline-to-online RL methods [38], [33]. The most representative method is online decision transformer (ODT) [38]. Although ODT can online interact with the environment to overcome the limitation of offline datasets, ODT still suffers from the sample inefficiency problem.

B. Episodic Memory

From the perspective of psychobiology, the reason why humans can quickly exploit the high reward after the discovery is the fact that the hippocampus of humans stores episodic memory (EM) [29], [1]. Motivated by the hippocampus's ability, the EM mechanism has been widely applied in the online RL setting to help the agent remember past valuable experiences and improve sample efficiency [3], [26], [22], [19], [36], [11], [23], [37]. When the action space is discrete, representative works include MFEC [3], NEC [26], and EMDQN [22]. As for the action space being continuous, EMAC [19], CEC [36], and GEM [11] have achieved some progress. Except for these methods for single-agent online RL, the EM mechanism has been extended into multi-agent online RL to improve sample efficiency [23], [37]. For offline RL setting, VEM [24] utilizes EM to accelerate training. While EM has been well utilized in traditional RL methods, EM has not been introduced into the sequence modeling-based RL methods.

III. Preliminaries

A. Notations and Setting

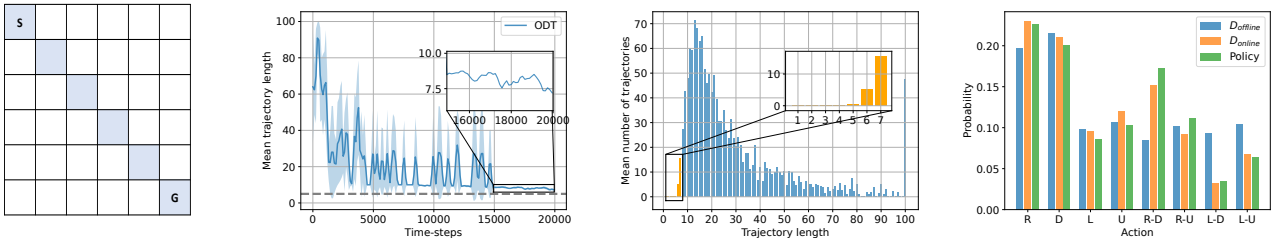
Following ODT [38], we model the environment as a Markov decision process (MDP) [2], which is denoted as a tuple $\langle S, A, P, R, \gamma, \rho \rangle$. Here, $S \in \mathbb{R}^F$ is the state space, and A is the action space. $P(s_{t+1}|s_t, a_t)$ represents the probability distribution over transitions. $R(s_t, a_t)$ is the reward function, and $\gamma \in (0, 1]$ is the discount factor. An agent starts in an initial state $s_1 \in S$, which is sampled from $\rho(s_1)$. At each time-step t, the agent receives a state $s_t \in S$ and then takes an action $a_t \in A$. The environment then yields a reward $r_t = R(s_t, a_t)$ and transits to the next state $s_{t+1} \sim P(\cdot|s_t, a_t)$. Let τ represent a trajectory with a length of $|\tau|$. The return-to-go (RTG) of a trajectory τ at time-step t, denoted as g_t, is computed as the sum of rewards from time-step t until the end of the trajectory: $g_t = \sum_{t'=t}^{|\tau|} r_{t'}$. Please note that the RTG g_1 is equivalent to the total return of the trajectory τ. The discounted return (DR) of a trajectory τ at time-step t, denoted as d_t, is the discounted sum of rewards from time-step t until the end of the trajectory: $d_t = \sum_{t'=t}^{|\tau|} \gamma^{t'-t} r_{t'}$. We use $\boldsymbol{s}_{-C,t} = (s_{\max(1,t-C+1)}, \cdots, s_t)$ to represent the sequence of states with the latest C time-steps at time-step t, respectively. Here, C is denoted as the context length. Similarly, $\boldsymbol{g}_{-C,t}$ represents the sequence of RTGs, $\boldsymbol{a}_{-C,t}$ represents the sequence of actions, and $\boldsymbol{d}_{-C,t}$ represents the sequence of DRs, all with the same context length C.

In this paper, we focus on the offline-to-online RL setting. During the offline training phase, the agent uses an offline dataset (replay buffer) D_{offline}, sampled from the offline data distribution $\mathscr{D}_{\text{offline}}$. During the online fine-tuning phase, the agent utilizes a replay buffer denoted as D_{online}. D_{online} initially contains trajectories from D_{offline} and is then updated using trajectories collected by the agent through online interactions with the environment, following a first-in-first-out manner.

B. Online Decision Transformer

Under the offline-to-online RL setting, online decision transformer (ODT) [38] is one of the representative works that formulates the RL problem as a sequence modeling problem. ODT learns a stochastic policy $\pi_\theta(a_t|\boldsymbol{s}_{-C,t}, \boldsymbol{g}_{-C,t})$ with the policy parameters θ. For the continuous action space, the stochastic policy is represented as a multivariate Gaussian distribution with a mean vector $\mu_\theta(\boldsymbol{s}_{-C,t}, \boldsymbol{g}_{-C,t})$, and a diagonal covariance matrix $\Sigma_\theta(\boldsymbol{s}_{-C,t}, \boldsymbol{g}_{-C,t})$. The original paper of ODT [38] only focuses on the continuous action spaces. In the case of discrete action spaces, we extend ODT by modeling the policy $\pi_\theta(a_t|\boldsymbol{s}_{-C,t}, \boldsymbol{g}_{-C,t})$ using a categorical distribution, a common choice [12]. During the online fine-tuning phase, the agent is given an online RTG $g_{\text{online},1}$ and an initial state s_1. At the time-step t, the agent takes the action a_t based on its stochastic policy, and then receives the next state $s_{t+1} \sim P(\cdot|s_t, a_t)$ and a reward $r_t = R(s_t, a_t)$. The next online RTG $g_{\text{online},t+1}$ is updated as $g_{\text{online},t+1} = g_{\text{online},t} - r_t$. This process is repeated until the trajectory is terminated. Then, we compute the RTGs and store $\tau = (s_1, a_1, g_1, \ldots, s_{|\tau|}, a_{|\tau|}, g_{|\tau|})$ in D_{online}.

Given the replay buffer D (D_{offline}, or D_{online}), ODT samples a sub-trajectory $(\boldsymbol{s}_{-C,t}, \boldsymbol{a}_{-C,t}, \boldsymbol{g}_{-C,t})$, which is abbreviated as $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{g})$ by omitting subscripts. This sub-trajectory contains multiple samples $\{(\boldsymbol{s}^c_{-C,t}, \boldsymbol{a}^c_{-C,t}, \boldsymbol{g}^c_{-C,t}) | \max(1, t - C + 1) \le c \le t\}$ with different lengths. Here, $\boldsymbol{s}^c_{-C,t} = (s_{\max(1,t-C+1)}, \cdots, s_c)$, similarly for $\boldsymbol{a}^c_{-C,t}$, and $\boldsymbol{g}^c_{-C,t}$. ODT



(a) (b) (c) (d)

Fig. 1. (a) The grid-world task. "S" is the start location of the agent, and "G" is the goal location of the agent. The blue blocks mean the optimal trajectory of the agent. (b) Training curve about the mean trajectory length of ODT across 5 random seeds. The grey dotted line represents the optimal policy. (c) Distribution of trajectory lengths, which is summarized from all encountered trajectories after offline training and online fine-tuning across 5 random seeds. (d) Comparison of the action distribution at "S" of learned policy with that in the D_{offline} and D_{online}. Here, D_{online} is obtained after offline training and online fine-tuning. R, L, D and U denote *Right*, *Left*, *Down* and *Up*, respectively. R-D, R-U, L-D and L-U denote *right-down*, *right-up*, *left-down*, and *left-up*, respectively.

solves the following training objective by alternately optimizing θ and λ:

$$\max_{\lambda \geqslant 0} \min_{\theta} J(\theta) + \lambda(\eta - \bar{H}(\theta)), \tag{1}$$

where

$$J(\theta) = \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a},\boldsymbol{g})\sim D}[-\frac{1}{|C^t|}\sum_{c\in C^t} \log \pi_\theta(a_c|\boldsymbol{s}^c_{-C,t}, \boldsymbol{g}^c_{-C,t})], \tag{2}$$

and

$$\bar{H}(\theta) = \mathbb{E}_{(\boldsymbol{s},\boldsymbol{g})\sim D}[\frac{1}{|C^t|}\sum_{c\in C^t} H(\pi_\theta(\cdot|\boldsymbol{s}^c_{-C,t}, \boldsymbol{g}^c_{-C,t}))]. \tag{3}$$

Here, $H(\pi_\theta(\cdot|\boldsymbol{s}^c_{-C,t}, \boldsymbol{g}^c_{-C,t}))$ is the Shannon entropy of the distribution $\pi_\theta(\cdot|\boldsymbol{s}^c_{-C,t}, \boldsymbol{g}^c_{-C,t})$, $C^t = \{c|\max(1, t-C+1) \leq c \leq t\}$ and $(\boldsymbol{s},\boldsymbol{g})$ is the abbreviation of $(\boldsymbol{s}^c_{-C,t}, \boldsymbol{g}^c_{-C,t})$. η is a constant.

During evaluation, the agent is given an evaluation RTG $g_{\text{eval},1}$ and an initial state s_1 at the beginning of a trajectory. At the time-step t, ODT selects the mean vector of the action distribution as its action a_t for the continuous action spaces, and ODT selects the action with the maximum probability as its action a_t for the discrete action spaces. The agent executes the action a_t and then receives the next state $s_{t+1} \sim P(\cdot|s_t, a_t)$ and a reward $r_t = R(s_t, a_t)$. The next evaluation RTG $g_{\text{eval},t+1}$ is updated as $g_{\text{eval},t+1} = g_{\text{eval},t} - r_t$. This process is repeated until the trajectory is terminated.

IV. Method

In this section, we first use a motivating example to illustrate that ODT suffers from the sample inefficiency problem. Then, we propose a simple yet effective method, called weighting online decision transformer with episodic memory (WODTEM), to improve sample efficiency.

A. Motivating Example

We use a simple grid-world task with a 6×6 grid to demonstrate the sample inefficiency problem encountered by ODT [38] and to reveal interesting findings. The agent in this task has 8 actions, including 4 cardinal directions (*right, down, left, up*) and 4 diagonal directions (*right-down, right-up, left-down, left-up*). The agent aims to reach the bottom-right "G" from the top-left "S" within 100 time steps. The agent only receives a reward of 1 upon reaching "G", where the trajectory is terminated. In all other locations, the agent receives a reward of 0. For offline training, we use an offline dataset consisting of 500 trajectories. These trajectories were collected using a fixed sub-optimal policy, with a mean trajectory length of 47.50 and a mean return of 0.91. This fixed sub-optimal policy operates as follows: at each state, the agent's policy assigns a probability of 20% to the *left* action, a probability of 20% to the *down* action, and a probability of 10% each to any other available action. Fig. 1a shows the snapshot of the grid-world task, and the blue blocks represent the optimal trajectory of the agent, which has a length of 5.

We evaluate ODT on this task with a sample budget of 20K online interactions, and the hyper-parameters settings of ODT are detailed in the following experimental section. After offline training and online fine-tuning, ODT converges to a sub-optimal policy with a mean trajectory length of approximately seven and a mean return of 1, which is an improvement over the policy adopted by the offline dataset. However, ODT still suffers from sample inefficiency since it fails to capture the optimal policy corresponding to the optimal trajectory. Considering five random seeds, Fig. 1b demonstrates the training curve about the mean trajectory length of ODT during the online fine-tuning phase, and Fig. 1c summarizes the mean number of trajectories with varying lengths across all encountered trajectories after offline training and online fine-tuning. We observe that the agent has collected some valuable trajectories with lengths shorter than seven, even the optimal trajectory. This finding illustrates that ODT ignores these valuable trajectories. We also compare the action distribution at "S" of the learned policy with that in D_{offline} and D_{online}, as shown in Fig. 1d. Here, D_{online} is obtained after offline training and online fine-tuning. Please note that the optimal action at "S" is *right-down* (R-D). We can find that although the probability of R-D in the learned policy is higher than that in the offline dataset, these three action distributions still share similarities. This finding illustrates that the policy only matches the action distribution in the replay buffer, and this might be caused by equally weighing over all state-action pairs, which causes sample inefficiency. Hence, this example motivates us to put

different weights for state-action pairs in the replay buffer to improve sample efficiency.

B. Weighting ODT with Episodic Memory

To address the above problem, we first propose an EM mechanism, which enables the agent to access the approximate historical optimal action and approximate historical optimal DR for each state. Then, we propose a novel training objective with a weighting function based on ODT [38] to improve sample efficiency.

1) EM Mechanism: The EM mechanism can help the agent to remember past valuable experiences, for example, historical optimal actions of states and historical optimal DRs of states. The key to constructing the EM mechanism is establishing a lookup table, denoted as $\boldsymbol{Q}$, indexed by states. Each entry in $\boldsymbol{Q}$, corresponding to a state s, stores a tuple $Q(s) = (s, a^e, e)$. Here, e represents the highest DR ever obtained in state s and is called historical optimal DR. a^e is the action for achieving e in state s and is called historical optimal action. Given a collected trajectory τ from either an offline dataset or online interactions from the environment, we calculate the DR for each state-action pair and use DRs to refresh the trajectory. Then we get $\tau = (s_1, a_1, g_1, d_1, \ldots, s_{|\tau|}, a_{|\tau|}, g_{|\tau|}, d_{|\tau|})$. The lookup table $\boldsymbol{Q}$ is updated by τ based on the following update rules:

$$Q(s_t) \leftarrow \begin{cases} Q(s_t), \text{ if } Q(s_t) \text{ is stored in } \boldsymbol{Q} \text{ and } e \geq d_t; \\ (s_t, a_t, d_t), \text{ if } Q(s_t) \text{ is stored in } \boldsymbol{Q} \text{ and } e < d_t; \\ (s_t, a_t, d_t), \text{ if } Q(s_t) \text{ is not stored in } \boldsymbol{Q}; \end{cases} \tag{4}$$

where $1 \leq t \leq |\tau|$. Here, $e \geq d_t$ signifies that the highest DR obtained in state s_t recorded in $\boldsymbol{Q}$ is larger than or equal to the DR obtained in state s_t during this trajectory. The update rules mean that for each state, we update its historical optimal DR with the maximum DR and its historical optimal action with the action corresponding to the maximum DR. The lookup table $\boldsymbol{Q}$ serves as a buffer for storing the most valuable experiences within historical trajectories. The historical optimal DR stored in each entry is updated non-decreasingly. Following prior works [3], [22], [23], the number of entries in the lookup table $\boldsymbol{Q}$ is increased until the maximum size constraint is satisfied. When $\boldsymbol{Q}$ is filled, we remove the least frequently accessed entry. Moreover, we also employ random projection as a dimensionality reduction technique [16] to project a state from the original state space $S \in \mathbb{R}^F$ into a lower-dimensional space with dimension $F_1 \ll F$.

The most similar EM mechanism to our work is the EM mechanism in CEC [36], both implemented by maintaining a table with state-action-DR tuples. However, there is a notable difference in updating the lookup table. CEC uses a nearest neighbor search method to update the table, while our work focuses on exact state matching and then updates the corresponding action and DR. The exact state matching adopted by our work can avoid imprecise updates and information loss, which can occur when using the nearest neighbor search method.

2) Approximate Historical Optimal DR and Approximate Historical Optimal Action: The EM mechanism adopts a lookup table $\boldsymbol{Q}$ to record the most valuable experiences from the collected trajectories. We expect that given a query state s_q, we can look up from $\boldsymbol{Q}$ and get $Q(s_q) = (s_q, a^e_q, e_q)$. Then, we can utilize the historical optimal action a^e_q and the historical optimal DR e_q for the state s_q to assist the agent training. However, in many applications, the above expectation suffers from a problem that many states have been visited only once or zero times by the agent, especially in an environment with a high-dimensional and continuous state space. This leads to two possible cases when considering a query state s_q: either the state s_q is not found in $\boldsymbol{Q}$, or when found, the corresponding historical optimal action a^e_q and historical optimal DR e_q have not been sufficiently updated because the state s_q is visited infrequently. Consequently, the challenge lies in efficiently approximating a historical optimal DR and a historical optimal action to assist the agent training in such cases.

As assumed in prior works like [36], [19], neighboring states can be considered similar to the query state and then can contribute valuable information. We adopt a strategy of looking up the K closest states (neighboring states) $\{s_k | k = 1, \ldots, K\}$ to a query state s_q within the lookup table $\boldsymbol{Q}$ and providing an approximate historical optimal DR and an approximate historical optimal action for the agent. However, it is essential to note that even the closest state retrieved might not always be similar to the query state. Hence, we need to filter out those dissimilar states. Following [19], the similarity between two states s_q and s_k can be measured by the $l2$ distance metric, expressed as $q(s_q, s_k) = \|s_q - s_k\|_2^2$. Then, we filter out those dissimilar states to the query state by using a distance threshold ε. Specifically, given a query state s_q and one of K closet states in $\boldsymbol{Q}$, s_k, if $q(s_q, s_k) \geq \varepsilon$, the states s_q and s_k are dissimilar and s_k should be filtered out. Hence, we can get a set $\boldsymbol{K}(s_q)$ that comprises similar states with a maximum number of K, their corresponding historical optimal DRs, and historical optimal actions, shown as follows:

$$\boldsymbol{K}(s_q) = \{(s_k, a^e_k, e_k) | q(s_q, s_k) < \varepsilon, 1 \leq k \leq K\}.$$

Given a query state s_q and $\boldsymbol{K}(s_q)$, the approximate historical optimal DR for s_q is denoted as $\hat{e}(s_q)$ and is calculated as a weighted sum of the corresponding historical optimal DRs in $\boldsymbol{K}(s_q)$, shown as follows:

$$\hat{e}(s_q) = \sum_{(s_k, a^e_k, e_k) \in \boldsymbol{K}(s_q)} \frac{\exp(-q(s_q, s_k)) e_k}{\sum_{(s_{k_1}, a^e_{k_1}, e_{k_1}) \in \boldsymbol{K}(s_q)} \exp(-q(s_q, s_{k_1}))}. \tag{5}$$

And the approximate historical optimal action for s_q is denoted as $\hat{a}(s_q)$, which is chosen from $\{a^e_k | (s_k, a^e_k, e_k) \in \boldsymbol{K}(s_q)\}$. Among these similar states, a higher value of the historical optimal DR indicates a better action. Hence, we sample the action $\hat{a}(s_q)$ from the set $\{a^e_k | (s_k, a^e_k, e_k) \in \boldsymbol{K}(s_q)\}$ based on the probability, which is proportional to their

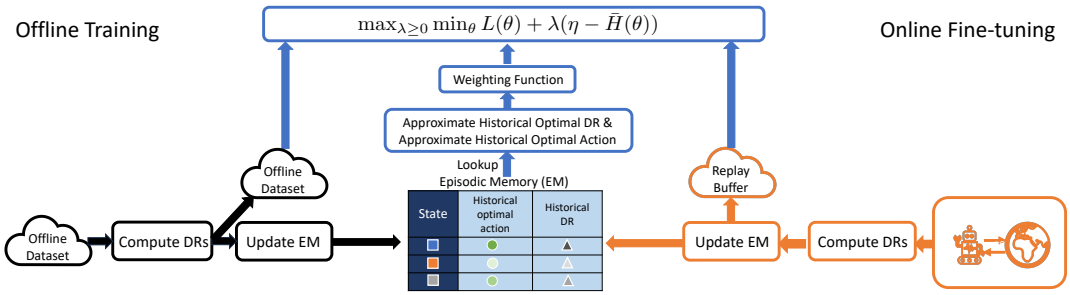


Fig. 2. Architecture of WODTEM during the offline training phase and the online fine-tuning phase.

historical optimal DRs, shown as follows:

$$p(a_k^e|s_q,s_k,e_k)=\frac{\exp(e_k)}{\sum_{(s_k,a_k^e,e_k)\in \boldsymbol{K}(s_q)}\exp(e_k)}. \tag{6}$$

3) Training Procedure: As discussed in Section IV-A, the agent needs to assign appropriate attention to different state-action pairs in the replay buffer. We leverage approximate historical optimal DRs and actions to help the agent allocate varying weights to state-action pairs in the replay buffer. Given a sub-trajectory $(\boldsymbol{s}_{-C,t},\boldsymbol{a}_{-C,t},\boldsymbol{g}_{-C,t},\boldsymbol{d}_{-C,t})$ sampled from the replay buffer, we get multiple samples $\{(\boldsymbol{s}^c_{-C,t},\boldsymbol{a}^c_{-C,t},\boldsymbol{g}^c_{-C,t},\boldsymbol{d}^c_{-C,t})|\max(1,t-C+1)\leq c\leq t\}$ with different lengths. For $(\boldsymbol{s}^c_{-C,t},\boldsymbol{a}^c_{-C,t},\boldsymbol{g}^c_{-C,t},\boldsymbol{d}^c_{-C,t})$, we look up s_c in $\boldsymbol{Q}$ and obtain $\boldsymbol{K}(s_c)$. Based on (5), we obtain the approximate historical optimal DR $\hat{e}(s_c)$. If $d_c\geq\hat{e}(s_c)$, a_c is the historical optimal action, and the agent has reason to pay more attention to this state-action pair from the replay buffer. If $d_c<\hat{e}(s_c)$, it means that a_c, which is sampled from the replay buffer, is not the historical optimal action, and the agent needs to pay less attention to this state-action pair. This is summarized as a weighting function, shown as follows:

$$w(s_c,a_c)=\begin{cases}1, & \text{if } d_c\geq\hat{e}(s_c);\\ \alpha, & \text{otherwise;}\end{cases} \tag{7}$$

where $\alpha\in[0,1]$ is a constant. Moreover, when $d_c<\hat{e}(s_c)$, we can use the approximate historical optimal action $\hat{a}(s_c)$ to assist the agent in finding its potential optimal action at state s_c. Hence, based on ODT, we propose a novel training objective with the weighting function (7), shown as follows:

$$\max_{\lambda\geqslant 0}\min_{\theta} L(\theta)+\lambda(\eta-\bar{H}(\theta)), \tag{8}$$

where

$$L(\theta)=\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a},\boldsymbol{g},\boldsymbol{d})\sim D}[-\frac{1}{|C^t|}\sum_{c\in C^t}w(s_c,a_c)\log\pi_\theta(a_c|\boldsymbol{s}^c_{-C,t},\boldsymbol{g}^c_{-C,t})$$
$$-\frac{1}{|C^t|}\sum_{c\in C^t}(1-w(s_c,a_c))\log\pi_\theta(\hat{a}(s_c)|\boldsymbol{s}^c_{-C,t},\boldsymbol{g}^c_{-C,t})], \tag{9}$$

and

$$\bar{H}(\theta)=\mathbb{E}_{(\boldsymbol{s},\boldsymbol{g})\sim D}[\frac{1}{|C^t|}\sum_{c\in C^t}H(\pi_\theta(\cdot|\boldsymbol{s}^c_{-C,t},\boldsymbol{g}^c_{-C,t}))]. \tag{10}$$

Here, $C^t=\{c|\max(1,t-C+1)\leq c\leq t\}$. $\hat{a}(s_c)$ is the approximate historical optimal action of the state s_c. $H(\pi_\theta(\cdot|\boldsymbol{s}^c_{-C,t},\boldsymbol{g}^c_{-C,t}))$ is the Shannon entropy of the distribution $\pi_\theta(\cdot|\boldsymbol{s}^c_{-C,t},\boldsymbol{g}^c_{-C,t})$. η is a constant. The first term in (9) aims at training the model π_θ to selectively match the observed action distribution from the replay buffer by applying different weights to different state-action pairs. The second term in (9) aims to use approximate historical optimal actions to help train the model π_θ. Moreover, the training objective (10) is the same as that in ODT. This training objective (10) controls the degree of action distribution mismatch during the offline training phase and encourages the agent to explore during the online fine-tuning phase. The whole training objective (8) is solved by alternately optimizing θ and λ. Please note that when $\alpha=1$, our method degenerates to ODT. Fig. 2 shows the architecture of WODTEM during the offline training and online fine-tuning phase.

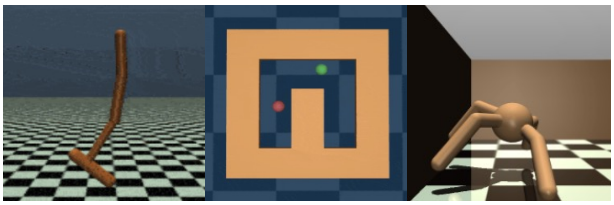


Fig. 3. (a) A snapshot of the hopper task. (b) A snapshot of the maze2d-umaze task. (c) A snap of the antmaze task.

V. Experiments

A. Tasks and Datasets

We focus on evaluating our method on three types of tasks using offline datasets provided by the D4RL benchmark [9], including gym-mujoco tasks, maze2d tasks, and antmaze tasks. For gym-mujoco, we choose hopper, walker2d, halfcheetah, and ant as tasks and use the medium and medium-replay datasets to evaluate our method. For maze2d, a 2D agent is required to reach a fixed goal location. We choose three tasks, including umaze, medium, and large. The datasets are generated by randomly selecting goal locations and a planner. For antmaze, an 8-DoF ant quadruped robot is required to reach a fixed goal location and receives a sparse 0-1 reward. We choose the umaze as a task. We use umaze and umaze-diverse as the datasets. Fig. 3 shows the snapshots of the hopper, maze2d-umaze, and antmaze tasks. We also evaluate all methods on the grid-world task described in Section IV-A, and the way of collecting the dataset has been described in Section IV-A.

B. Baselines

We choose DT [5] and ODT [38] for baselines. DT is the basic architecture adopted by ODT and our method. In the

TABLE I

HYPER-PARAMETER SETTINGS OF ODT ON THE GRID-WORLD TASK.

Hyper-parameter	Value	Hyper-parameter	Value
Number of layers	1	Number of attention heads	1
Embedding dimension	512	Context length	1
Weight decay	0.0	Learning rate	0.001
Offline training updates	1e3	Online training updates per iteration	50
Buffer size	500	Position	no
g_{online}	2.0	g_{eval}	1.0

TABLE II

HYPER-PARAMETER SETTINGS OF ODT ON THE MAZE2D TASKS.

Hyper-parameter	Value	Hyper-parameter	Value
Number of layers	4	Number of attention heads	4
Embedding dimension	512	Context length	20
Weight decay	0.0	Learning rate	0.001
Offline training updates	2e4	Online training updates per iteration	300
Buffer size	1500	Position	yes
g_{online} (maze2d-umaze)	340.0	g_{eval} (maze2d-umaze)	170.0
g_{online} (maze2d-medium)	560.0	g_{eval} (maze2d-medium)	280.0
g_{online} (maze2d-large)	560.0	g_{eval} (maze2d-large)	280.0

offline-to-online RL setting, ODT is the most representative method that models the RL problem as a sequence modeling problem. For gym-mujoco and antmaze tasks, we use the same hyper-parameter settings as those in the original papers. We summarize the specific hyper-parameter settings for the grid-world task and all maze2d tasks in Table I and Table II, respectively. Other hyper-parameter settings are the same as that in [38].

C. Results on Tasks from D4RL

In this section, we evaluate our method and baselines with five random seeds on the gym-mujoco, maze2d, and antmaze tasks from D4RL [9]. ODT and our method both have a sample budget of 200K online interactions. Following the prior work [38], we also report the average normalized scores. Table III summarizes the average normalized scores of all methods on gym-mujoco, maze2d, and antmaze tasks. Here, "m", "mr" and "d" represent "medium", "medium-replay" and "diverse", respectively. In our method, F_1 is set to 4 for the random projection technique. The lookup table size is set to 4e6 for the maze2d-large task and is set to 2e6 for others. The discount factor γ is set to 0.99. α is set to 0.9 and K is set to 2. ε is set to 0.2 and 0.6 for walker2d and halfcheetah tasks, respectively. For ant and antmaze tasks, $\varepsilon = 0.3$. ε is set to 0.1 for other tasks. Other hyper-parameter settings are the same as those in ODT [38]. From Table III, we can find that our method can perform better than ODT and DT in most tasks, demonstrating that our method is effective and improves sample efficiency.

D. Results on Grid-world

For the grid-world task, we do not use the random projection technique, and the lookup table size is set to 36 since the number of all states is 36. The discount factor γ is set to 0.99. K is set to 1 and ε is set to 0.1. α is set to 0.75. ODT and our method both have a sample budget of 20K online interactions. After offline training and online fine-tuning, our method converges to the optimal policy with a mean trajectory length of 5 and a mean return of 1. Fig 4a shows the training curves about the mean trajectory length of all methods. We find that after offline training, our method performs better than other methods, which illustrates that our method can learn better from the offline dataset compared with other methods. During the online fine-tuning phase, our method can collect the optimal trajectory by exploration. We also find that our method quickly overlaps with the grey dotted line, which represents the optimal policy. This observation verifies again that our method can improve sample efficiency. Furthermore, we compare the action distribution at "S" of the learned policy with that in D_{offline} and D_{online}, as shown in Fig 4b. Here, D_{online} is obtained after offline training and online fine-tuning. We can find that at "S", the action distribution of the learned policy is different from that in the offline dataset D_{offline}. This observation illustrates that our method can help the policy avoid blindly matching the action distribution in the replay buffer.

TABLE III

COMPARISON OF THE AVERAGE NORMALIZED SCORES ON TASKS FROM D4RL. WE REPORT THE MEAN AND STANDARD DEVIATION ACROSS FIVE RANDOM SEEDS.

Dataset	DT	ODT	WODTEM
hopper-m	61.03 ± 5.11	95.56 ± 3.53	$\mathbf{96.54 \pm 4.60}$
hopper-mr	62.75 ± 15.05	85.52 ± 3.26	$\mathbf{88.78 \pm 2.16}$
walker2d-m	72.03 ± 4.32	72.58 ± 1.42	$\mathbf{73.99 \pm 2.26}$
walker2d-mr	42.53 ± 15.36	73.11 ± 4.54	$\mathbf{76.26 \pm 3.97}$
halfcheetah-m	$\mathbf{42.43 \pm 0.30}$	42.38 ± 0.25	42.17 ± 0.11
halfcheetah-mr	35.92 ± 1.56	40.08 ± 0.46	$\mathbf{40.09 \pm 0.51}$
ant-m	$\mathbf{93.56 \pm 4.94}$	87.51 ± 2.69	88.96 ± 2.45
ant-mr	89.08 ± 5.33	88.92 ± 1.10	$\mathbf{91.31 \pm 2.25}$
sum	499.33	585.66	**598.10**
antmaze-umaze	53.30 ± 5.52	$\mathbf{82.60 \pm 5.57}$	80.20 ± 1.33
antmaze-umaze-d	52.50 ± 9.89	53.60 ± 1.47	$\mathbf{58.60 \pm 3.72}$
sum	105.80	136.20	**138.80**
maze2d-umaze	39.95 ± 12.17	31.85 ± 7.46	$\mathbf{75.18 \pm 10.79}$
maze2d-medium	19.11 ± 4.17	82.88 ± 7.88	$\mathbf{96.23 \pm 3.24}$
maze2d-large	38.93 ± 4.32	69.23 ± 12.35	$\mathbf{84.26 \pm 16.54}$
sum	97.99	183.96	**255.67**

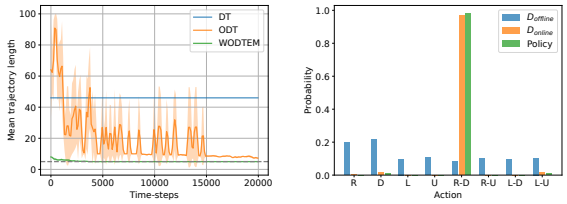


Fig. 4. (a) Training curves of ODT, and WODTEM across 5 random seeds. Due to DT does not have online fine-tuning, we use a reference line to show DT. (b) Comparison of the action distribution at "S" of learned policy with that in the D_{offline} and D_{online} after offline training and online fine-tuning.

VI. CONCLUSION

In this paper, we have proposed a novel and effective method, WODTEM, to improve sample efficiency when modeling the RL problem as a sequence modeling problem in the offline-to-online RL setting. To the best of our knowledge, WODTEM is the first work to introduce the EM mechanism into sequence modeling-based RL methods. Utilizing the EM mechanism, we propose a novel training objective with a weighting function based on ODT to improve sample efficiency. Experimental results on multiple tasks have verified that WODTEM can improve sample efficiency.

REFERENCES

[1] P. Andersen, R. Morris, D. Amaral, T. Bliss, and J. O'Keefe. *The hippocampus Book*. Oxford university press, Oxford, 2006.

[2] R. Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[3] C. Blundell, B. Uria, A. Pritzel, Y. Li, A. Ruderman, J. Z. Leibo, J. W. Rae, D. Wierstra, and D. Hassabis. Model-free episodic control. *CoRR*, abs/1606.04460, 2016.

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *NeurIPS*, 2020.

[5] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *NeurIPS*, 2021.

[6] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[8] S. Emmons, B. Eysenbach, I. Kostrikov, and S. Levine. Rvs: What is essential for offline RL via supervised learning? In *ICLR*, 2022.

[9] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4RL: datasets for deep data-driven reinforcement learning. *CoRR*, abs/2004.07219, 2020.

[10] H. Furuta, Y. Matsuo, and S. S. Gu. Generalized decision transformer for offline hindsight information matching. In *ICLR*, 2022.

[11] H. Hu, J. Ye, G. Zhu, Z. Ren, and C. Zhang. Generalizable episodic memory for deep reinforcement learning. In *ICML*, 2021.

[12] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

[13] M. Janner, Q. Li, and S. Levine. Offline reinforcement learning as one big sequence modeling problem. In *NeurIPS*, 2021.

[14] H. Kim, Y. Ohmura, and Y. Kuniyoshi. Transformer-based deep imitation learning for dual-arm robot manipulation. In *IROS*, 2021.

[15] S. G. Konan, E. Seraj, and M. C. Gombolay. Contrastive decision transformers. In *CoRL*, 2022.

[16] I. Kononenko and M. Kukar. *Machine learning and data mining*. Horwood Publishing, 2007.

[17] I. Kostrikov, A. Nair, and S. Levine. Offline reinforcement learning with implicit q-learning. In *ICLR*, 2022.

[18] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. In *NeurIPS*, 2020.

[19] I. Kuznetsov and A. Filchenkov. Solving continuous control with episodic memory. In *IJCAI*, 2021.

[20] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *CoRR*, abs/2005.01643, 2020.

[21] W. Li, H. Luo, Z. Lin, C. Zhang, Z. Lu, and D. Ye. A survey on transformers in reinforcement learning. *CoRR*, abs/2301.03044, 2023.

[22] Z. Lin, T. Zhao, G. Yang, and L. Zhang. Episodic memory deep q-networks. In *IJCAI*, 2018.

[23] X. Ma and W.-J. Li. State-based episodic memory for multi-agent reinforcement learning. *Machine Learning*, 112(12):5163–5190, 2023.

[24] X. Ma, Y. Yang, H. Hu, J. Yang, C. Zhang, Q. Zhao, B. Liang, and Q. Liu. Offline reinforcement learning with value-based episodic memory. In *ICLR*, 2022.

[25] A. Nair, M. Dalal, A. Gupta, and S. Levine. Accelerating online reinforcement learning with offline datasets. *CoRR*, abs/2006.09359, 2020.

[26] A. Pritzel, B. Uria, S. Srinivasan, A. P. Badia, O. Vinyals, D. Hassabis, D. Wierstra, and C. Blundell. Neural episodic control. In *ICML*, 2017.

[27] J. Shang, X. Li, K. Kahatapitiya, Y. Lee, and M. S. Ryoo. Starformer: Transformer with state-action-reward representations for robot learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(11):12862–12877, 2023.

[28] M. Shridhar, L. Manuelli, and D. Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *CoRL*, 2022.

[29] L. R. Squire. Memory systems of the brain: a brief history and current perspective. *Neurobiology of Learning and Memory*, 82(3):171–177, 2004.

[30] R. K. Srivastava, P. Shyam, F. Mutz, W. Jaskowski, and J. Schmidhuber. Training agents using upside-down reinforcement learning. *CoRR*, abs/1912.02877, 2019.

[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[32] K. Wang, H. Zhao, X. Luo, K. Ren, W. Zhang, and D. Li. Bootstrapped transformer for offline reinforcement learning. In *NeurIPS*, 2022.

[33] Z. Xie, Z. Lin, D. Ye, Q. Fu, Y. Wei, and S. Li. Future-conditioned unsupervised pretraining for decision transformer. In *ICML*, 2023.

[34] T. Yamagata, A. Khalil, and R. Santos-Rodríguez. Q-learning decision transformer: Leveraging dynamic programming for conditional sequence modelling in offline RL. In *ICML*, 2023.

[35] S. Yang, D. Schuurmans, P. Abbeel, and O. Nachum. Dichotomy of control: Separating what you can control from what you cannot. In *ICLR*, 2023.

[36] Z. Yang, T. M. Moerland, M. Preuss, and A. Plaat. Continuous episodic control. *CoRR*, abs/2211.15183, 2022.

[37] L. Zheng, J. Chen, J. Wang, J. He, Y. Hu, Y. Chen, C. Fan, Y. Gao, and C. Zhang. Episodic multi-agent reinforcement learning with curiosity-driven exploration. In *NeurIPS*, pages 3757–3769, 2021.

[38] Q. Zheng, A. Zhang, and A. Grover. Online decision transformer. In *ICML*, 2022.