Online Tuning for Offline Decentralized Multi-Agent Reinforcement Learning

Jiechuan Jiang, Zongqing Lu[†]

School of Computer Science, Peking University
{jiechuan.jiang, zongqing.lu}@pku.edu.cn

Abstract

Offline reinforcement learning could learn effective policies from a fixed dataset, which is promising for real-world applications. However, in offline decentralized multi-agent reinforcement learning, due to the discrepancy between the behavior policy and learned policy, the transition dynamics in offline experiences do not accord with the transition dynamics in online execution, which creates severe errors in value estimates, leading to uncoordinated low-performing policies. One way to overcome this problem is to bridge offline training and online tuning. However, considering both deployment efficiency and sample efficiency, we could only collect very limited online experiences, making it insufficient to use merely online data for updating the agent policy. To utilize both offline and online experiences to tune the policies of agents, we introduce *online transition correction* (OTC) to implicitly correct the offline transition dynamics by modifying sampling probabilities. We design two types of distances, *i.e.*, embedding-based and value-based distance, to measure the similarity between transitions, and further propose an adaptive rank-based prioritization to sample transitions according to the transition similarity. OTC is simple yet effective to increase data efficiency and improve agent policies in online tuning. Empirically, OTC outperforms baselines in a variety of tasks.

1 Introduction

In fully decentralized multi-agent reinforcement learning (MARL) (de Witt et al. 2020a; Su et al. 2022; Jiang and Lu 2022; Su and Lu 2022), agents interact with the environment to obtain individual experiences and independently improve the policies to maximize the shared rewards. Due to the scalability and robustness, fully decentralized learning shows great potential in solving real-world cooperative tasks. However, in many industrial applications, continuously interacting with the environment to collect the experiences for learning is costly and risky, *e.g.*, autonomous driving. To overcome this challenge, offline decentralized MARL (Jiang and Lu 2021) lets each agent learn its policy from a fixed dataset of experiences without interacting with the environment. The dataset of each agent contains the individual action instead of the joint action of all agents. There is no assumption on the data collection policies and the relationship between the datasets of agents.

However, from the perspective of individual agent, the transition dynamics depend on the policies of other agents and will change as other agents improve their policies (Foerster et al. 2017). Since the learned policies of other agents are inconsistent with their behavior policies for data collection, the transition dynamics in execution are different from the transition dynamics in the dataset, which will cause extrapolation error, *i.e.*, the error in value estimate incurred by the mismatch between the experience distribution of the learned policy and the dataset (Fujimoto, Meger, and Precup 2019). The extrapolation error makes the agent underestimate or overestimate state values, which leads to uncoordinated low-performing policies (Jiang and Lu 2021).

One way to reduce the extrapolation error caused by the mismatch of transition dynamics is to bridge offline training and online tuning. However, since both deploying new policies and interacting with the environment are costly and risky, we should consider the deployment efficiency (the number of deployments) and sample efficiency (the number of interactions) in the collection of online experiences (Matsushima et al. 2021). Due to the efficiency requirement, the collected online experiences can be very limited. Thus, it is insufficient to tune the policies of agents merely using the online data, and the small online dataset may also cause overfitting. To increase data efficiency, it is better to additionally exploit the offline data for online tuning. However, uniformly sampling from the merged offline and online experiences (Nair et al. 2020) cannot address the transition mismatch problem. Therefore, it is necessary to correct the transition dynamics in the offline data to make it close to the online transition dynamics. Nevertheless, the requirement of efficiency also means it is infeasible to accurately estimate the real transition dynamics from the limited online experiences, thus explicit correction is impractical.

In this paper, we introduce a simple yet effective method to correct the transition dynamics of offline data for online tuning, without explicitly modeling the transition dynamics. When sampling a transition from the offline experiences, we first search for the best-matched transition in the online experiences, which has the minimum state-action distance to the sampled transition. Then, we compute the next-state distance between the sampled transition and the best-matched

[†]Corresponding Author

online transition to represent the transition similarity. After that, a probability function maps the transition similarity to the probability of accepting the sampled transition for update, which is equivalent to modifying the transition probability. Therefore, the objective is to find the optimal probability function which minimizes the KL-divergence between the online transition distribution and the modified transition distribution, given the distance measure. We design two distance measures based on the embedding and Q-value of transitions, respectively. The embedding-based distance captures the similarity in feature space, and the value-based distance measures the isomorphism. Due to the limited online experiences, it is hard to find the optimal probability function by gradient-based optimization, but we empirically find that the rank-based prioritization in PER (Schaul et al. 2016) is a proper choice of the probability function. Moreover, we propose an adaptive rank-based prioritization to adjust the degree of the correction according to the difference between offline and online transition distributions.

The proposed method, termed **OTC** (**O**nline **T**ransition **C**orrection), can be applied to many offline RL methods for decentralized MARL. We construct decentralized datasets from a variety of tasks, including D4RL (Fu et al. 2020), MPE (Lowe et al. 2017), and SMAC (Samvelyan et al. 2019). Empirically, OTC outperforms baselines, and ablation studies demonstrate the effectiveness of the two distance measures, the practicability of rank-based prioritization, and the improvement of adaptive prioritization. We additionally extend OTC to non-stationary single-agent settings and verify the scalability. *To the best of our knowledge, OTC is the first method for bridging offline training and online tuning in fully decentralized MARL.*

2 Related Work

Offline RL. In offline RL, the agent could only access to a fixed dateset of single-step transitions collected by a behavior policy, and no interactive experience collection is allowed during learning. Offline RL easily suffers from the extrapolation error, which is mainly caused by out-of-distribution actions in single-agent environments. Constraint-based methods introduce policy constraints to enforce the learned policy to be close to the behavior policy, *e.g.*, direct action constraint (Fujimoto, Meger, and Precup 2019), kernel MMD (Kumar et al. 2019), Wasserstein distance (Wu, Tucker, and Nachum 2019), and $l2$ distance (Pan et al. 2021; Fujimoto and Gu 2021). Conservative methods (Kumar et al. 2020; Yu et al. 2021) train a Q-function pessimistic to out-of-distribution actions. Uncertainty-based methods quantify the uncertainty by learned environment model (Yu et al. 2020) or by Monte Carlo dropout (Wu et al. 2021) of Q-function, and use it as a penalty or to weight the update of Q-function, so as to avoid the overestimation of out-of-distribution actions. Recent work (Yang et al. 2021) studies offline MARL in the centralized learning and decentralized execution setting, which requires the joint actions of all agents and cannot be applied to decentralized datasets that contains only individual actions (Jiang and Lu 2021).

In offline decentralized MARL, besides out-of-distribution actions, the extrapolation error is also caused by the mismatch of transition dynamics. For each individual agent, since the transition dynamics depend on other agents' policies which are also updating, there will be a difference between the transition dynamics in the offline dataset and the real transition dynamics during online deployment (Jiang and Lu 2021). To overcome this, MABCQ (Jiang and Lu 2021) focuses on offline training and uses two importance weights to modify the offline transition dynamics by normalizing the transition probabilities and increasing the transition probabilities of high-value next states. However, the modified transition dynamics in MABCQ are not guaranteed to be close to the real ones. Unlike MABCQ, OTC focuses on online tuning and exploits online experiences to quickly correct the bias of transition dynamics.

Bridging Offline Learning and Online Tuning. Since the offline dataset is usually insufficient to cover the entire transition space, the extrapolation error cannot be eliminated entirely in fully offline learning. It is crucial to improve the policy trained using offline data further with online reinforcement learning. A theoretical work (Xie et al. 2021) analyzes a sample complexity lower bound for policy finetuning algorithm. Since the online interaction is expensive, we must consider both the deployment efficiency (the number of policy deployments) and sample efficiency (the number of interactions) in online tuning. The concept of deployment efficiency is adopted in BREMEN (Matsushima et al. 2021) and MUSBO (Su et al. 2021), which, however, do not aim to finetune the pre-trained policy but instead train the policy from scratch with limited deployments. AWAC (Nair et al. 2020) employs an implicit constraint that can mitigate the extrapolation error while avoiding overly conservative updates in offline learning and thus quickly performs online finetuning. Balanced Replay (Lee et al. 2021) adopts prioritized sampling to encourage the use of near-on-policy samples from the offline dataset. However, they deploy the policy frequently, ignoring the deployment efficiency. Moreover, these methods are designed for single-agent environments, where offline and online data follow the same transition dynamics, thus they cannot deal with the mismatch of transition dynamics. Abiding by both deployment and sample efficiency, OTC uses prioritized sampling to reduce the bias of transition dynamics in decentralized MARL.

3 Method

3.1 Preliminaries

In offline and decentralized cooperative settings, each agent i can only access to an offline dataset $\mathcal{B}_i$, which is collected by a behavior policy and contains the tuples $\langle s, a_i, r, s' \rangle$, where s is the state[1], a_i is the individual action of agent i, r is the shared reward, and s' is the next state. Note that $\mathcal{B}_i$ does not contain the joint actions of all agents. Each agent i independently learns its policy π_i using offline RL algorithm, without information sharing among agents. The goal of all agents is to maximize the expected return $\mathbb{E} \sum_{t=0}^{\infty} \gamma^t r_t$ when deploying their learned policies in the environment, where γ

[1] Global state is only for the convenience of theoretical analysis, the agents could learn based on partial observation in practice.

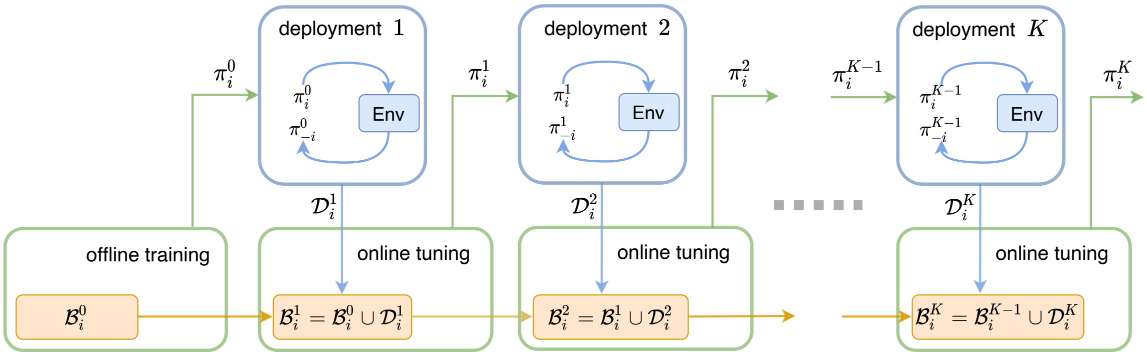


Figure 1: Overview of offline training and online tuning. After each agent i learning its policy π_i^0 from offline dataset $\mathcal{B}_i^0$, their policies are deployed in the environment to get the online dataset $\mathcal{D}_i^1$. Then, π_i^0 is finetuned to obtain π_i^1 using the merged dataset $\mathcal{B}_i^1$. The online tuning is repeated for K times.

is the discount factor. From the perspective of each agent i, the transition probability in $\mathcal{B}_i$, denoted by $P_{\mathcal{B}_i}(s'|s, a_i)$, depends on other agents' behavior policies during the collection of $\mathcal{B}_i$, while the real transition probability in execution, denoted by $P_{\mathcal{E}_i}(s'|s, a_i)$, depends on other agents' learned policies. The difference between $P_{\mathcal{B}_i}$ and $P_{\mathcal{E}_i}$ causes severe extrapolation errors, which eventually lead to uncoordinated low-performing policies (Jiang and Lu 2021).

To finetune the policies learned from offline datasets, we allow the agents to interact with the environment to collect online experiences. As illustrated in Figure 1, after offline learning using initial dataset $\mathcal{B}_i^0$ for each agent i, their learned policies $\langle \pi_i^0, \pi_{-i}^0 \rangle$, where π_{-i}^0 denotes the policies of all agents except i, are deployed in the environment and interact with each other for M timesteps. Each agent i obtains an online dataset $\mathcal{D}_i^1$ with M transitions. $\mathcal{D}_i^1$ still only contains the individual actions of agent i rather than the joint actions. We merge $\mathcal{D}_i^1$ and $\mathcal{B}_i^0$ to get $\mathcal{B}_i^1$, and finetune the policy π_i^0 to obtain π_i^1 using $\mathcal{B}_i^1$. Then, we deploy the updated policies $\langle \pi_i^1, \pi_{-i}^1 \rangle$ in the environment and repeat the procedures until K deployments. K represents the deployment efficiency, and $K \times M$ represents the sample efficiency. Note that for presentation simplicity, we denote π_{-i} as also updating, but other agents may or may not be learning online, which however does not affect the following problem formulation and our method. That said OTC does not have any assumptions on other agents.

3.2 Problem Formulation

In general, for two MDPs with *only* different transition dynamics, we have the following theorem.

Theorem 1. *Under different transition dynamics P_1 and P_2, the corresponding optimal value functions V_1^* and V_2^* satisfy $\|V_1^* - V_2^*\|_\infty \leq \frac{\gamma r_{\max}}{(1-\gamma)^2} \|P_1 - P_2\|_1$, where $\|P_1 - P_2\|_1 = \max_{s,a} \sum_{s'} |P_1(s'|s,a) - P_2(s'|s,a)|$.*

The proof is given in Section Proof. Based on this theorem, since the offline transition dynamics $P_{\mathcal{B}_i^k}$ are generally much different from the ones in the online dataset $P_{\mathcal{D}_i^k}$, the value function learned from $\mathcal{B}_i^k$ is inconsistent with the optimal one in online execution. In order to use the merged dataset to finetune the policy, we need to modify $P_{\mathcal{B}_i^k}$ as $\tilde{P}_{\mathcal{B}_i^k}$ to *minimize* the difference between $P_{\mathcal{D}_i^k}$ and $\tilde{P}_{\mathcal{B}_i^k}$.

Since the difference of transition dynamics means the difference of next-state distributions given the same state-action pair, we first define two distance functions. For agent i's two transitions $\langle s_1, a_{i_1}, s_1', r_1 \rangle$ and $\langle s_2, a_{i_2}, s_2', r_2 \rangle$, $d(s_1, a_{i_1}, s_2, a_{i_2})$ measures the similarity of state-action pairs, and $d(s_1', s_2')$ measures the similarity of next states. Once sampling a transition $\langle s, a_i, s', r \rangle$ from $\mathcal{B}_i^k$, we select the best-matched transition $\langle s^*, a_i^*, s'^*, r^* \rangle$ from $\mathcal{D}_i^k$, which has the minimum state-action distance to $\langle s, a_i, s', r \rangle$,

$$\langle s^*, a_i^*, s'^*, r^* \rangle = \arg\min\nolimits_{\mathcal{D}_i^k} d(s, a_i, s^*, a_i^*). \tag{1}$$

The best-matched transition approximately has the same state-action pair with the sampled offline transition. If there is more than one best-matched transition, we uniformly select one. Then, we adopt a probability function f mapping the distance $d(s', s'^*)$ to the probability of accepting the sampled transition $\langle s, a_i, s', r \rangle$ for update, *i.e.*, $f(d(s', s'^*))$. Therefore, the transition probability can be modified as

$$\tilde{P}_{\mathcal{B}_i^k}(s'|s, a_i) = P_{\mathcal{B}_i^k}(s'|s, a_i) \frac{\sum_{\hat{s}'} P_{\mathcal{D}_i^k}(\hat{s}'|s, a_i) f(d(s', \hat{s}'))}{Z(s, a_i)},$$

where $Z(s, a_i)$ is a normalization term to make sure $\sum_{s'} \tilde{P}_{\mathcal{B}_i^k}(s'|s, a_i) = 1$. Thus the KL-divergence between $P_{\mathcal{D}_i^k}$ and $\tilde{P}_{\mathcal{B}_i^k}$ is

$$\begin{aligned} &\mathrm{KL}(P_{\mathcal{D}_i^k} \| \tilde{P}_{\mathcal{B}_i^k}) = \mathrm{KL}(P_{\mathcal{D}_i^k} \| P_{\mathcal{B}_i^k}) \qquad (2) \\ &- \sum_{s'} P_{\mathcal{D}_i^k}(s'|s, a_i) \log \frac{\sum_{\hat{s}'} P_{\mathcal{D}_i^k}(\hat{s}'|s, a_i) f(d(s', \hat{s}'))}{Z(s, a_i)}. \end{aligned}$$

To minimize the KL-divergence, we need to design appropriate d-functions to accurately measure the distances between transitions and find the optimal f-function which properly maximizes the second term of RHS of (2).

3.3 d-Functions

Due to the limitations of computational complexity and representation ability, directly measuring the distances

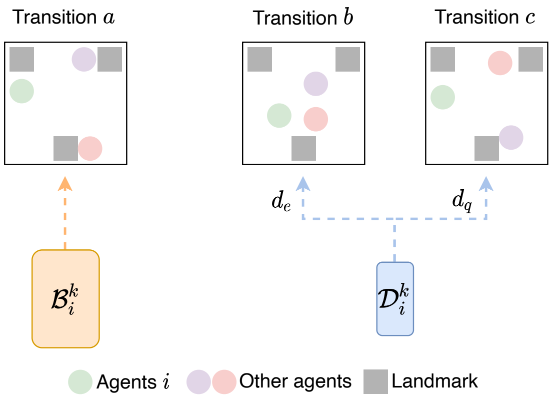


Figure 2: In navigation task, the agents are learning to cover all the landmarks. For the transition a from $\mathcal{B}_i^k$, the best-matched transition b selected from $\mathcal{D}_i^k$ using d_e is still much different from transition a. However, the value-based distance d_q will select the nearly isomorphic transition c, which is more helpful for evaluating the trainsition similarity.

in the original space is impractical, especially in high-dimensional environments. Therefore, we design two types of d-functions. The first type is the distance in the embedding space. We employ VAE (Kingma and Welling 2013) to encode the state-action pair and next state into $e(s, a_i)$ and $e(s')$, and define the d-functions as $l1$ distance in the embedding space,

$$\begin{aligned} d_e(s, a_i, s^*, a_i^*) &= \|e(s, a_i) - e(s^*, a_i^*)\|, \\ d_e(s', s'^*) &= \|e(s') - e(s'^*)\|. \end{aligned}$$

Relying on the generalization ability of the encoder, similar inputs will be encoded into similar embeddings. We can search for the best-matched transition with the most similar state-action pair in terms of $d_e(s, a_i, s^*, a_i^*)$ and then evaluate the transition similarity using $d_e(s', s'^*)$.

The embedding-based distance measures the similarity in feature space. However, since the limited online experiences cannot cover all state-action pairs in offline dataset, there must be some transitions in the offline dataset which are still much different from the transition with the most similar state-action feature, *e.g.*, transition a and b in a navigation task in Figure 2. In such cases, we cannot accurately evaluate the transition similarity using d_e. Inspired by state representation learning (Gelada et al. 2019; Zhang et al. 2020), we notice that some state-action pairs may have common latent structure, *e.g.*, transition a and c in Figure 2. Although they are different in feature space, the topologies of agents are isomorphic. As pointed by DeepMDP (Gelada et al. 2019), nearly isomorphic state-action pairs would have similar Q-values. Thus, we use the difference in Q-values to evaluate the isomorphism in state-action pairs and select the best-matched transition from the online dataset,

$$d_q(s, a_i, s^*, a_i^*) = \|Q(s, a_i) - Q(s^*, a_i^*)\|.$$

On the other hand, we use the expected Q-value to measure

Algorithm 1: OTC for Agent i

```
Initialize the RL model and the modification degree $\alpha_i^0$
Train the RL model using $\mathcal{B}_i^0$ to obtain the policy $\pi_i^0$
for $k = 1, \dots, K$ do
    Deploy the policy $\pi_i^{k-1}$ in the environment and collect online dataset $\mathcal{D}_i^k$ with $M$ transitions
    Merge the experiences $\mathcal{B}_i^k = \mathcal{B}_i^{k-1} \cup \mathcal{D}_i^k$
    if $k > 1$ then
        Adjust $\alpha_i^k$ by (4)
    end if
    for $t = 1, \dots, max\_update$ do
        Sample a minibatch B from $\mathcal{B}_i^k$ and a minibatch D from $\mathcal{D}_i^k$
        for each transition in B do
            Find the best-matched transition in D (1) and compute the transition similarity
        end for
        Sample transitions from B by prioritization (3)
        Update the RL model using the sampled transitions
    end for
end for
```

the distance of next states,

$$d_q(s', s'^*) = \left\| \frac{V(s')}{\mathbb{E}_{s'} V(s')} - \frac{V(s'^*)}{\mathbb{E}_{s'^*} V(s'^*)}) \right\|,$$

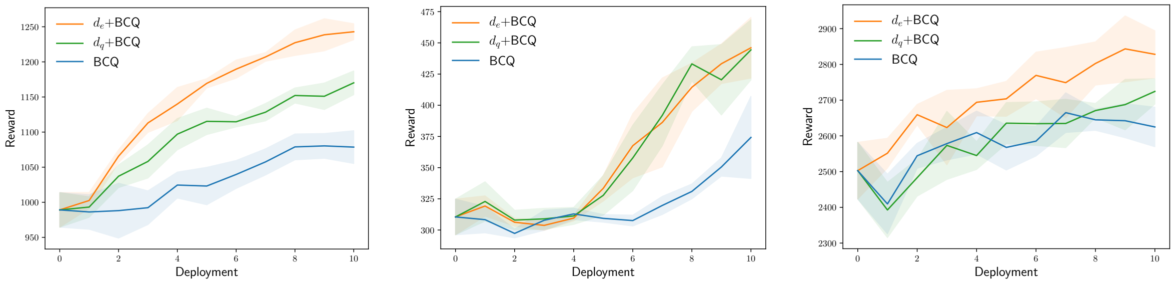
where $V(s') = \mathbb{E}_{a_i} Q(s', a_i)$. $\frac{V(s')}{\mathbb{E}_{s'} V(s')}$ is the deviation from the expected value, which can mitigate the influence of absolute value. The value-based distance d_q has stronger representation ability than the embedding-based distance d_e. The transitions that are close in embedding space will also have small value difference, and the value-based distance also represents the isomorphism of transitions.

3.4 f-Function

The f-function is to maximize the second term of RHS of (2). However, since the online experiences are limited, it is hard to solve the optimal f-function by gradient-based optimization. Nevertheless, there must exist such f-functions which do not increase $\mathrm{KL}(P_{\mathcal{D}_i^k} \| \tilde{P}_{\mathcal{B}_i^k})$, and a trivial example is the constant function. Therefore, we try to find a heuristic and practical f-function which is able to reduce the KL-divergence and thus the extrapolation error. An important prior is that f-function should be monotonic, which will produce a larger acceptance probability when fed with a smaller distance of next states. The intuition is that if the next state of the transition from $\mathcal{B}_i^k$ is more similar to the next state of the online experience with the same state-action pair, the transition is more likely to follow the transition dynamics in $\mathcal{D}_i^k$. Therefore, we should give it a larger acceptance probability, which eventually leads to a larger transition probability. Empirically, we find the rank-based prioritized sampling in PER (Schaul et al. 2016) is a good solution. Concretely, the probability of accepting transition j is

$$P(j) = \frac{p_j^\alpha}{\sum_m p_m^\alpha}, \tag{3}$$

where the priority $p_j = \frac{1}{\mathrm{rank}(j)}$, and $\mathrm{rank}(j)$ is the rank of transition j when the transitions are sorted according to



(a) halfcheetah-random

(b) walker2d-random

(c) halfcheetah-replay

Figure 3: Learning curves of OTC on BCQ.

$d(s', s'^*)$. The exponent α determines the degree of modifying the transition probability, with $\alpha = 0$ meaning the f-function degrades into a constant function. The rank-based prioritization ensures that the probability of being accepted is monotonic in terms of transition similarity, and is robust as it is insensitive to outliers (Schaul et al. 2016).

Another prior is that the modification degree depends on the difference between $\mathcal{D}_i^k$ and $\mathcal{B}_i^k$. When transition dynamics in $\mathcal{B}_i^k$ are more similar to that in $\mathcal{D}_i^k$, the modification degree should be weaker, and vice versa. Thus, we propose to adaptively adjust α at each deployment k ($k > 1$) as

$$\alpha_i^k = \alpha_i^{k-1} \times \frac{\mathbb{D}_i^k}{\mathbb{D}_i^{k-1}}, \tag{4}$$

where $\mathbb{D}_i^k = \mathbb{E}_{s,a,s' \sim \mathcal{B}_i^k}\, d(s', s'^* \sim \arg\min_{\mathcal{D}_i^k} d(s, a_i, s^*, a_i^*))$.

$\mathbb{D}_i^k$ is the expected transition similarity for agent i at deployment k. As the difference between offline and online transition distributions will change along with the update of agents, a fixed α is not an optimal solution. On the contrary, for example, if the distribution difference grows, adaptive α is likely to take on large value, and thus more aggressively modifies the transition dynamics.

3.5 Implementation Details

For the embedding-based distance d_e, we do not maintain two embeddings $e(s, a_i)$ and $e(s')$, but train a conditional VAE $G_i = \{E_i(\mu, \sigma | s, a_i, s'), D_i(a_i | s, s', z \sim (\mu, \sigma))\}$ which encodes $\langle s, a_i, s' \rangle$ into the embedding $\mu(s, a_i, s')$. We take the embedding $\mu(s, a_i, s')$ as a substitute for both $e(s, a_i)$ and $e(s')$, which is more effective and computationally efficient in practice. Moreover, it is costly to sample transitions from $\mathcal{B}_i^k$ and then search the best-match transitions from $\mathcal{D}_i^k$ for every update. To reduce the complexity, for each update we uniformly sample two minibatches, B and D respectively from $\mathcal{B}_i^k$ and $\mathcal{D}_i^k$, and perform the rank-based prioritized sampling on the two minibatches. The training of OTC is summarized in Algorithm 1.

4 Experiments

The experiments could be divided into two parts. First, we fully verify the effectiveness of each module including distance metrics d_q and d_e, on standard benchmarks D4RL (Fu et al. 2020) and on the two representative base algorithms: BCQ (Fujimoto, Meger, and Precup 2019), which adopts standard Bellman update, and AWAC (Nair et al. 2020) which is an important offline-to-online baseline. Second, we verify OTC in more diverse settings, including different tasks (SMAC (Samvelyan et al. 2019), MPE (Lowe et al. 2017)) and more base algorithms (BREMEN (Matsushima et al. 2021), CQL (Kumar et al. 2020), and TD3-BC (Fujimoto and Gu 2021)), so we only choose one distance.

4.1 Settings

We evaluate OTC in D4RL (Fu et al. 2020) datasets with three types: random, medium, and medium-replay. Following the settings in multi-agent mujoco (de Witt et al. 2020b; Jiang and Lu 2021), we split the original action space of three mujoco tasks (Todorov, Erez, and Tassa 2012) into several sub-spaces, *i.e.*, 2-agent HalfCheetah, 2-agent Walker2d, and 3-agent Hopper. Each agent obtains the state and reward of the robot and independently controls one or some joints of the robot. For each agent i, we delete the actions of other agents from the original dataset and take the modified dataset as $\mathcal{B}_i^0$. During online tuning, we perform $K = 10$ deployments. For each deployment k, the agent collects a very limited online dataset $\mathcal{D}_i^k$, of which the size is only one percent of the initial offline dataset ($|\mathcal{D}_i^k| = 1\% |\mathcal{B}_i^0|$). After online data collection, we finetune the agents by L updates and deploy the updated policies in the environment for next deployment.

We instantiate OTC respectively on two offline RL algorithms, BCQ (Fujimoto, Meger, and Precup 2019), and AWAC (Nair et al. 2020), and also take them as the baselines. During online tuning, the baselines uniformly and randomly sample transitions from the merged offline and online dataset, and they also have the same neural network architectures and hyperparameters as OTC. All the models are trained for five runs with different random seeds, and results are presented using mean and standard deviation.

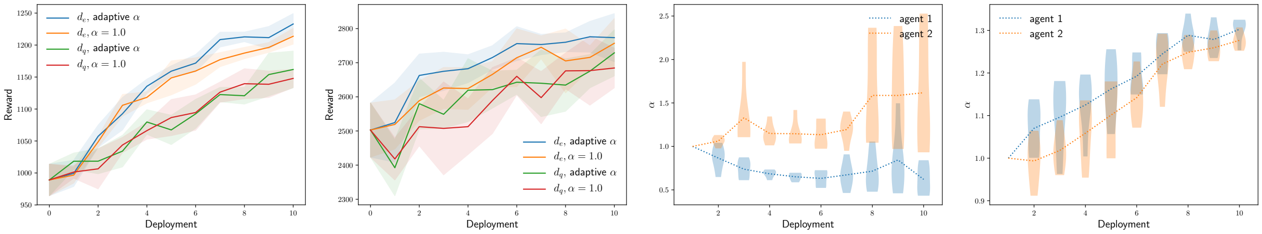
4.2 Evaluating d-Functions

We plot the learning curves along with the number of deployments for a part of tasks in Figure 3 and summarize the performance improvement (%) at the last deployment compared with the offline-trained policies for all the tasks

		d_e + BCQ	d_q + BCQ	BCQ	d_e + AWAC	d_q + AWAC	AWAC
random	halfcheetah	**25.7** ± 1.2	18.3 ± 1.7	9.0 ± 2.4	-	-	-[†]
	walker2d	**43.7** ± 7.8	43.1 ± 7.8	20.5 ± 10.6	**102.4** ± 83.0	−14.8 ± 6.7	−19.5 ± 2.7
	hopper	**7.3** ± 1.3	6.4 ± 2.8	1.2 ± 12.8	**41.5** ± 20.3	17.8 ± 16.9	−33.0 ± 15.8
replay	halfcheetah	**13.0** ± 2.6	8.9 ± 1.4	4.9 ± 2.2	16.7 ± 8.4	**27.4** ± 3.8	0.6 ± 7.3
	walker2d	27.8 ± 8.1	**47.2** ± 14.6	17.3 ± 25.8	−8.0 ± 15.1	3.2 ± 8.4	−17.2 ± 13.5
	hopper	**44.6** ± 12.3	49.2 ± 52.1	9.2 ± 4.3	**203.0** ± 13.5	93.3 ± 72.4	79.2 ± 46.9
medium	halfcheetah	14.8 ± 4.1	**15.0** ± 1.1	12.1 ± 3.0	**31.0** ± 1.7	28.7 ± 3.6	26.0 ± 2.3
	walker2d	27.1 ± 21.7	**23.3** ± 7.9	−13.9 ± 4.4	**77.7** ± 27.3	84.0 ± 51.6	−7.2 ± 59.0
	hopper	2.6 ± 3.8	**12.0** ± 2.6	−6.8 ± 7.0	**104.2** ± 42.5	81.3 ± 62.6	11.8 ± 30.6

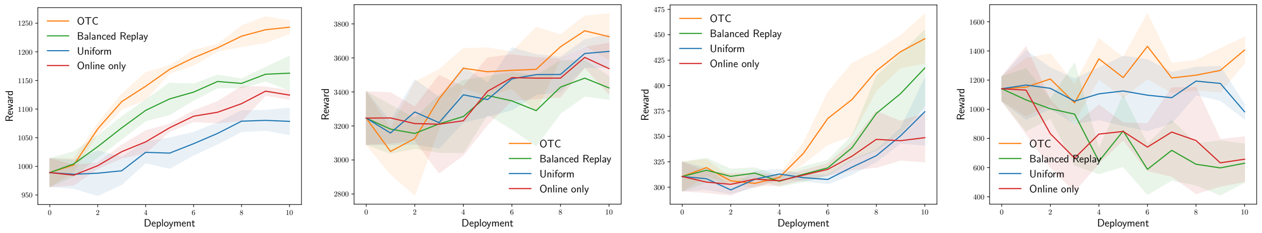
[†] AWAC does not work on halfcheetah-random, which is consistent with the result in AWAC paper (Nair et al. 2020).

Table 1: Performance improvement (%) at the last deployment compared with the offline-trained policies.



(a) halfcheetah-random (b) halfcheetah-replay (c) halfcheetah-random (d) halfcheetah-replay

Figure 4: *Left:* Learning curves of OTC (d_e) on BCQ with adaptive α and fixed α (1.0). *Right:* Curves of adaptive α. Dotted lines show mean values, and violin plots show distributions over seeds



(a) halfcheetah-random (b) halfcheetah-medium (c) walker2d-random (d) walker2d-medium

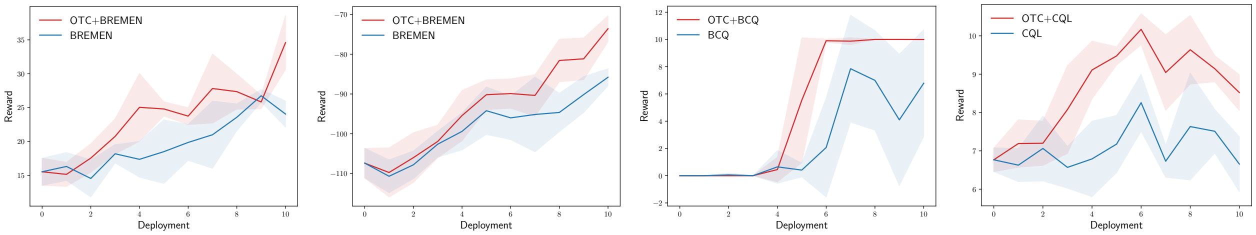
Figure 5: Learning curves of OTC, Balanced Replay, uniformly sampling from the merged dataset, and only using online data.

in Table 1. The empirical results show that OTC with d_e (embedding-based distance) or d_q (value-based distance) substantially outperforms the baselines, which verifies that d_e and d_q are capable of properly measuring the transition similarity. Since the online experiences are limited, uniformly sampling from the merged dataset is not effective at correcting the difference between transition dynamics. The value-based distance d_q essentially has a stronger representation ability than the embedding-based distance d_e, since it can represent similarities in both feature and isomorphism. But d_q does not commonly outperform d_e. The reason might be that d_q would mistakenly judge the state-action pairs, which are different in feature and isomorphism but have similar values, as best-matched pairs. Moreover, since the Q-values are updating, d_q is inconsistent during the tuning process. However, by reusing the learned Q values, d_q does not introduce additional modules, which follows Occam's razor (Blumer et al. 1987) and makes OTC more robust.

The computation of transition similarity is on GPU, which is fully parallelized and very efficient. In Table 2, we record the average time taken by one update. The computation of transition similarity in OTC costs less than 25% of the BCQ training time. The experiments are carried out on Intel i7-8700 CPU and NVIDIA GTX 1080Ti GPU.

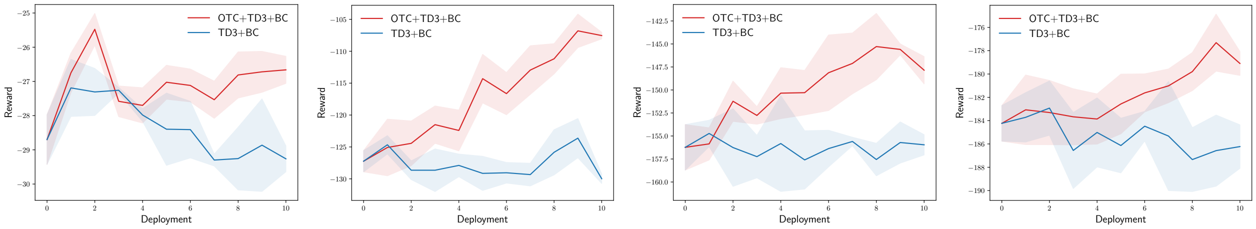
d_e+BCQ	d_q+BCQ	BCQ
14.2ms	13.8ms	11.6ms

Table 2: Average time taken by one update.



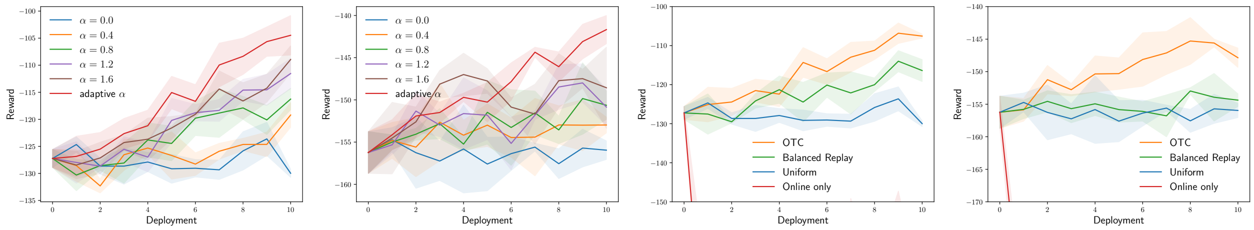
(a) Predator-Prey (b) Cooperative Navigation (c) Single Navigation (d) 5m_vs_6m

Figure 6: Learning curves on MPE (Lowe et al. 2017) tasks and SMAC (Samvelyan et al. 2019) task.



(a) 2 agents (b) 4 agents (c) 6 agents (d) 8 agents

Figure 7: Learning curves of OTC+TD3+BC with different numbers of agents on Cooperative Navigation.



(a) α, 4 agents (b) α, 6 agents (c) sampling, 4 agents (d) sampling, 6 agents

Figure 8: Ablation on Cooperative Navigation. *Left:* Effectiveness of α. *Right:* Different sampling strategies.

4.3 Evaluating f-Function

The hyperparameter α controls the strength of modifying the transition dynamics. It is observed that if α is too small, OTC has weak effects on correcting the transition dynamics. Otherwise, the overly modified transition dynamics will deviate from the real transition dynamics and degrade the performance. Since the agents are continuously updated, the difference between the transition dynamics in $\mathcal{B}_i^k$ and $\mathcal{D}_i^k$ will also change every deployment, so a fixed α cannot properly deal with the non-stationarity. We test the fixed $\alpha = 1.0$ and our method for adaptive α where initial $\alpha^0 = 1.0$. Figure 4 (*left*) shows that adaptive α can improve the performance of online tuning. Figure 4 (*right*) shows the curves of α during the online tuning of OTC (d_e). In halfcheetah-replay, α of both two agents grows, indicating the difference between $\mathcal{B}_i^k$ and $\mathcal{D}_i^k$ increases as the agents are further improved during online tuning, and α becomes stable in the latter deployments, meaning the convergence of online tuning. In halfcheetah-random, two agents have different trends of α, which means that the agents are influenced differently by the update of other agent. Our adaptive α method can discriminate the difference between $P_{\mathcal{B}_i^k}$ and $P_{\mathcal{D}_i^k}$ of each agent rather than treat them equally.

We additionally provide ablation studies of different sampling strategies in Figure 5. Although there is no transition bias in online only, Since the online dataset is very limited, finetuning the agents with online samples only is insufficient, and the small dataset can cause overfitting, *e.g.*, Figure 5d. Balanced Replay (Lee et al. 2021) and uniformly sampling are susceptible to the difference between $P_{\mathcal{B}_i^k}$ and $P_{\mathcal{D}_i^k}$. By considering the transition bias, OTC obtains the performance gain over the other three sampling strategies.

4.4 Deployment-Efficiency Baseline

Although deployment efficiency has been considered in BREMEN (Matsushima et al. 2021) and MUSBO (Su et al. 2021), the settings are different from ours. In their deployment-efficiency settings, BREMEN and MUSBO do not use the offline data but only collect online data, and they do not finetune the policies but retrain the policy from the scratch at each deployment. However, our method can also be built on these methods. We adapt BREMEN to our setting, training the policies from the offline data and finetuning the policies using limited online data at each deployment, and test OTC (d_e) with BREMEN on two MPE (Lowe et al. 2017) tasks: Predator-Prey and Cooperative Navigation. The offline datasets are collected by uniform random policies, containing 1×10^6 transitions. As shown in Figures 6a and 6b, OTC also outperforms the deployment-efficiency baseline in online tuning.

4.5 Single-Agent Case

Since OTC deals with the mismatches between the offline transition dynamics and online transition dynamics, it can be extended to non-stationary single-agent settings, where the transition dynamics will change between deployments. We test OTC (d_e) on a non-stationary single-agent navigation task, where an agent learns to reach a landmark to get a reward 10. The speed of the agent is βa, where a is the action and β is a random variable that controls the transition dynamics. During the data collection, the agent takes uniformly random policy and β is also uniformly and randomly sampled from $[-0.2, 0.2]$, which makes the expected movement of all actions be 0. Thus Q-values learned from the offline dataset are all equal to 0, and the agent cannot solve this task after offline training. During online finetuning, β gradually changes from 0.1 to 0.2. The learning curves are shown in Figure 6c. OTC modifies the offline translation probabilities to be close to the online ones, and thus outperforms BCQ, which shows OTC helps the finetuning in non-stationary single-agent settings. However, non-stationarity is unusual in single-agent environments but is an inherent issue in all decentralized multi-agent tasks (Mao et al. 2021). That is the reason we mainly focus on multi-agent settings.

4.6 Online Exploration and Exploitation

Since OTC requires experiences that follow online transition dynamics, when deployed in the environment, the agents have to only exploit the learned policies, without any exploration, which however will limit the exploration of out-of-distribution actions. A feasible way to explore novel experiences in online interaction is that the agents collect online data sequentially. When one agent collects online data, this agent takes the exploration policy, and other agents take the learned policies. We collect offline decentralized datasets on the SMAC 5m_vs_6m task (Samvelyan et al. 2019) using medium policies, which contain 1×10^4 episodes. Since the behavior policies are very deterministic, exploration in online interaction is necessary. We build OTC (d_q) on CQL (Kumar et al. 2020) and let the agents sequentially collect online experiences at each deployment. As shown in Figure 6d, OTC outperforms the baseline. However, the cost of sequential collection grows linearly with the number of agents. We leave the balance of exploration and exploitation in online interaction to future work.

4.7 Scalability

OTC is a decentralized method that takes other agents as a part of the environment regardless of the number of agents and does not require any information of other agents. We built OTC (d_q) on TD3+BC (Fujimoto and Gu 2021) in Cooperative Navigation with different numbers of agents. The offline datasets are collected by uniform random policies, containing 1×10^6 transitions. Figure 7 shows that OTC scales well with agent number. We also perform ablation studies on 4-agent and 6-agent Cooperative Navigation in Figure 8. The effectiveness of adaptive α is obvious since the performance is sensitive to α. By considering the transition bias, OTC obtains the significant performance gain over the other sampling strategies.

5 Proof

Theorem 1. *Under different transition dynamics P_1 and P_2, the corresponding optimal value functions V_1^* and V_2^* satisfy $\|V_1^* - V_2^*\|_\infty \leq \frac{\gamma r_{\max}}{(1-\gamma)^2} \|P_1 - P_2\|_1$, where $\|P_1 - P_2\|_1 = \max_{s,a} \sum_{s'} |P_1(s'|s,a) - P_2(s'|s,a)|$.*

Proof.

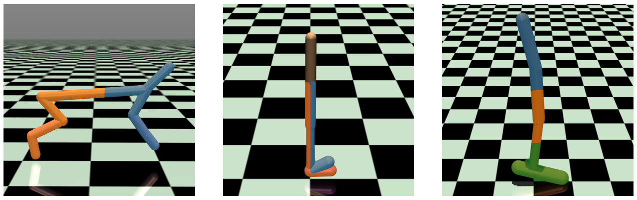
$$
\begin{aligned}
&|V_1^* - V_2^*|_\infty \\
&= \max_s \left| \max_a \left[r(s,a) + \gamma \sum_{s'} P_1(s'|s,a) V_1^*(s') \right] \right. \\
&\qquad \left. - \max_a \left[r(s,a) + \gamma \sum_{s'} P_2(s'|s,a) V_2^*(s') \right] \right| \\
&\leq \max_{s,a} \left| r(s,a) + \gamma \sum_{s'} P_1(s'|s,a) V_1^*(s') \right. \\
&\qquad \left. - r(s,a) - \gamma \sum_{s'} P_2(s'|s,a) V_2^*(s') \right| \\
&= \max_{s,a} \gamma \left| \sum_{s'} P_1(s'|s,a) V_1^*(s') - P_2(s'|s,a) V_2^*(s') \right| \\
&= \max_{s,a_i} \gamma \left| \sum_{s'} (P_1(s'|s,a) - P_2(s'|s,a)) V_1^*(s') \right. \\
&\qquad \left. + \sum_{s'} P_2(s'|s,a)(V_1^*(s') - V_2^*(s')) \right| \\
&\leq \max_{s'} \gamma \left| V_1^*(s') \right| \max_{s,a} \sum_{s'} \left| P_1(s'|s,a) - P_2(s'|s,a) \right| \\
&\qquad + \max_{s'} \gamma \left| V_1^*(s') - V_2^*(s') \right| \max_{s,a} \sum_{s'} P_2(s'|s,a) \\
&= \gamma \frac{r_{\max}}{1-\gamma} \|P_1 - P_2\|_1 + \gamma |V_1^* - V_2^*|_\infty
\end{aligned}
$$

Therefore,

$$
\|V_1^* - V_2^*\|_\infty \leq \frac{\gamma r_{\max}}{(1-\gamma)^2} \|P_1 - P_2\|_1
$$

□

Based on the theorem, OTC minimizes the difference between the offline transition dynamics and the online transition dynamics, making the learned values close to the optimal values in online execution.



(a) HalfCheetah (b) Walker2d (c) Hopper

Figure 9: Illustrations of multi-agent mujoco tasks. Different colors mean different agents (Jiang and Lu 2021).

6 Experiment Settings and Hyperparameters

Multi-Agent MuJoCo tasks are illustrated in Figure 9. In MPE experiments, the action space of agents is $[-0.1, 0.1]$, meaning the agent speed.

- Predator-Prey. Three slower agents learn to chase a faster rule-based prey. Each time one of the agents collide with the prey (distance < 0.1), the agents get a reward $+1$.
- Cooperative Navigation. N agents learn to cover N landmarks. The reward is $-\text{sum}(\text{distance}_j)$, where distance_j is the distance from landmark j to the closest agent. In BREMEN, we set $N = 4$

The hyperparameters are summarized in Table 3.

Hyperparameter	BCQ	AWAC	BREMEN	CQL	TD3+BC
discount (γ)	0.99				
|B|	512				
|D|	2000				
batch size	128				
hidden sizes	$(64, 64)$	$(256, 256)$			
activation	ReLU				
actor lr ($\times 10^{-4}$)	1	1	1	1	3
critic lr ($\times 10^{-4}$)	1	5	5	1	3
embedding dimension	10				
finetuning updates (L)	4000	2000			

Table 3: Hyperparameters

7 Conclusion

We propose OTC to effectively correct the transition dynamics during online interaction for tuning decentralized multi-agent policies learned from offline datasets, given limited online experiences. OTC consists of two types of distances to measure the transition similarity and an adaptive rank-based prioritization to sample transitions for updating the agent policy according to the transition similarity. Experimental results show that OTC outperforms baselines for online tuning in a variety of tasks.

Acknowledgments

This work was supported in part by NSF China under grant 61872009.

References

Blumer, A.; Ehrenfeucht, A.; Haussler, D.; and Warmuth, M. K. 1987. Occam's razor. *Information processing letters*, 24(6): 377–380.

de Witt, C. S.; Gupta, T.; Makoviichuk, D.; Makoviychuk, V.; Torr, P. H.; Sun, M.; and Whiteson, S. 2020a. Is Independent Learning All You Need in The StarCraft Multi-Agent Challenge? *arXiv preprint arXiv:2011.09533*.

de Witt, C. S.; Peng, B.; Kamienny, P.-A.; Torr, P.; Böhmer, W.; and Whiteson, S. 2020b. Deep Multi-Agent Reinforcement Learning for Decentralized Continuous Cooperative Control. *arXiv preprint arXiv:2003.06709*.

Foerster, J.; Nardelli, N.; Farquhar, G.; Afouras, T.; Torr, P. H.; Kohli, P.; and Whiteson, S. 2017. Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning. In *International Conference on Machine Learning (ICML)*.

Fu, J.; Kumar, A.; Nachum, O.; Tucker, G.; and Levine, S. 2020. D4rl: Datasets for Deep Data-Driven Reinforcement Learning. *arXiv preprint arXiv:2004.07219*.

Fujimoto, S.; and Gu, S. S. 2021. A Minimalist Approach To Offline Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Fujimoto, S.; Meger, D.; and Precup, D. 2019. Off-Policy Deep Reinforcement Learning Without Exploration. In *International Conference on Machine Learning (ICML)*.

Gelada, C.; Kumar, S.; Buckman, J.; Nachum, O.; and Bellemare, M. G. 2019. Deepmdp: Learning Continuous Latent Space Models for Representation Learning. In *International Conference on Machine Learning (ICML)*.

Jiang, J.; and Lu, Z. 2021. Offline Decentralized Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2108.01832*.

Jiang, J.; and Lu, Z. 2022. I2Q : A Fully-Decentralized Q-Learning Algorithm. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Kingma, D. P.; and Welling, M. 2013. Auto-Encoding Variational Bayes. *arXiv preprint arXiv:1312.6114*.

Kumar, A.; Fu, J.; Soh, M.; Tucker, G.; and Levine, S. 2019. Stabilizing Off-Policy Q-Learning Via Bootstrapping Error Reduction. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Kumar, A.; Zhou, A.; Tucker, G.; and Levine, S. 2020. Conservative Q-Learning for Offline Reinforcement Learning. *Advances in Neural Information Processing Systems (NeurIPS)*.

Lee, S.; Seo, Y.; Lee, K.; Abbeel, P.; and Shin, J. 2021. Offline-To-Online Reinforcement Learning Via Balanced Replay and Pessimistic Q-Ensemble. *Annual Conference on Robot Learning (CoRL)*.

Lowe, R.; Wu, Y.; Tamar, A.; Harb, J.; Abbeel, P.; and Mordatch, I. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *Advances in Neural Information Processing Systems (NeurIPS)*.

Mao, W.; Zhang, K.; Zhu, R.; Simchi-Levi, D.; and Basar, T. 2021. Near-Optimal Model-Free Reinforcement Learning in Non-Stationary Episodic MDPs. In *International Conference on Machine Learning (ICML)*.

Matsushima, T.; Furuta, H.; Matsuo, Y.; Nachum, O.; and Gu, S. 2021. Deployment-Efficient Reinforcement Learning Via Model-Based Offline Optimization. In *International Conference on Learning Representations (ICLR)*.

Nair, A.; Dalal, M.; Gupta, A.; and Levine, S. 2020. Accelerating Online Reinforcement Learning with Offline Datasets. *arXiv preprint arXiv:2006.09359*.

Pan, L.; Huang, L.; Ma, T.; and Xu, H. 2021. Plan Better Amid Conservatism: Offline Multi-Agent Reinforcement Learning with Actor Rectification. *arXiv preprint arXiv:2111.11188*.

Samvelyan, M.; Rashid, T.; de Witt, C. S.; Farquhar, G.; Nardelli, N.; Rudner, T. G. J.; Hung, C.-M.; Torr, P. H. S.; Foerster, J.; and Whiteson, S. 2019. The StarCraft Multi-Agent Challenge. *arXiv preprint arXiv:1902.04043*.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2016. Prioritized Experience Replay. In *International Conference on Learning Representations (ICLR)*.

Su, D.; Lee, J. D.; Mulvey, J. M.; and Poor, H. V. 2021. MUSBO: Model-Based Uncertainty Regularized and Sample Efficient Batch Optimization for Deployment Constrained Reinforcement Learning. *International Conference on Machine Learning (ICML)*.

Su, K.; and Lu, Z. 2022. Decentralized Policy Optimization. *arXiv preprint arXiv:2211.03032*.

Su, K.; Zhou, S.; Gan, C.; Wang, X.; and Lu, Z. 2022. MA2QL: A Minimalist Approach to Fully Decentralized Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2209.08244*.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A Physics Engine for Model-Based Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Wu, Y.; Tucker, G.; and Nachum, O. 2019. Behavior Regularized Offline Reinforcement Learning. *arXiv preprint arXiv:1911.11361*.

Wu, Y.; Zhai, S.; Srivastava, N.; Susskind, J.; Zhang, J.; Salakhutdinov, R.; and Goh, H. 2021. Uncertainty Weighted Actor-Critic for Offline Reinforcement Learning. *International Conference on Machine Learning (ICML)*.

Xie, T.; Jiang, N.; Wang, H.; Xiong, C.; and Bai, Y. 2021. Policy Finetuning: Bridging Sample-Efficient Offline and Online Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Yang, Y.; Ma, X.; Li, C.; Zheng, Z.; Zhang, Q.; Huang, G.; Yang, J.; and Zhao, Q. 2021. Believe What You See: Implicit Constraint Approach for Offline Multi-Agent Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Yu, T.; Kumar, A.; Rafailov, R.; Rajeswaran, A.; Levine, S.; and Finn, C. 2021. Combo: Conservative Offline Model-Based Policy Optimization. *Advances in Neural Information Processing Systems (NeurIPS)*.

Yu, T.; Thomas, G.; Yu, L.; Ermon, S.; Zou, J. Y.; Levine, S.; Finn, C.; and Ma, T. 2020. MOPO: Model-Based Offline Policy Optimization. *Advances in Neural Information Processing Systems (NeurIPS)*.

Zhang, A.; Sodhani, S.; Khetarpal, K.; and Pineau, J. 2020. Learning Robust State Abstractions for Hidden-Parameter Block MDPs. In *International Conference on Learning Representations (ICLR)*.