VNU Journal of Science: Comp. Science & Com. Eng., Vol. 41, No. 1 (2025) 69–83

VNU Journal of Science:
Computer Science and Communication Engineering

Journal homepage: http://www.jcsce.vnu.edu.vn/index.php/jcsce

Original Article

Adaptive Weighting by Sinkhorn Distance for Sharing Experiences between Multi-Task Reinforcement Learning in Sparse-reward Environments

Viet Cuong Ta*

VNU University of Engineering and Technology, Hanoi, 144 Xuan Thuy, Cau Giay, Hanoi, Vietnam

Received 30th October 2024
Revised 22nd March 2025; Accepted 10th June 2025

Abstract: In multi-task reinforcement learning, an agent using off-policy learning can leverage samples from other tasks to improve its learning process. When the reward signal from the environment is sparse, the agent in each task spends most of its training time exploring the environment. Therefore, the shared experiences between tasks can generally be considered as samples derived from an exploration policy. However, when the exploitation phase begins, the shared experience framework must account for the divergence of policies across different learning tasks. However, when the exploitation phase starts, the sharing experience framework has to take into account the policies' divergence issue of different learning tasks. Our work addresses this issue by employing an adaptive weight for shared experiences. First, a central buffer collects and shares the experiences from each individual task. To mitigate the effects of policy divergence among multiple tasks, we propose an algorithm that measures policy distances using the Sinkhorn distance. The computed distances are used to assign a specific weight to each shared sample, controlling the amount of knowledge shared as the policies begin to diverge during the exploitation phase. We conduct experiments in two goal-based multi-task learning environments to evaluate the effectiveness of our approach. The results show that our proposed method can improve from 8%-10% in average rewards in comparison with other baselines.

Keywords: multi-task reinforcement learning, off-policy, experience sharing

1. Introduction

In reinforcement learning (RL), an agent is trained to achieve its goals through interactions with the environment. The agent's objectives are represented by the reward signals provided for each action. By maximizing discounted cumulative rewards over a sequence of actions,

*Corresponding Author.
E-mail address: cuongtv@vnu.edu.vn

https://doi.org/10.25073/2588-1086/vnucsce.3918

the agent can reach its goal. RL has been applied to various real-world problems and has achieved promising results [1, 2]. Among different RL problem settings, the sparse-reward setting is defined by rewards that are only relevant for specific state-action pairs. Therefore, the agent must employ an efficient exploration and exploitation strategy to discover these state-action pairs and shape its policy [3]. The multi-task reinforcement learning (MTRL) framework employs different goal settings within the same environment [4, 5]. By allowing each agent to access the experiences of others, the shared experience framework in MTRL is designed to improve data efficiency by reducing the total number of samples required to train all tasks [6]. It is expected that an agent optimized for a specific task can extract relevant information from experiences derived from other tasks. If these experiences come from an exploration policy, it is straightforward to rely on that exploration to learn the optimal policy, as in standard off-policy learning [7]. In a sparse reward environment, an agent can use other agents' experiences as additional exploration samples to accelerate its exploration phase. Roughly speaking, the sparse condition alleviates the exploitation-divergence issue in the shared experience framework within the MTRL setting.

In general, approaches for training agents in multi-task reinforcement learning (MTRL) can be divided into several subcategories. Feature-based approaches focus on learning an efficient representation that enables the agent to optimize the reward function. Gradient-based approaches exploit gradient information to maximize training efficiency in multi-task RL. For instance, [8] accumulates shared gradients through a central model from each downstream task, while more advanced methods employ normalization procedures to eliminate gradient disagreement among tasks [9]. In a sparse-reward environment, [10] extends the standard fitted Q-iteration algorithm to improve the shared feature representation.

For experience-sharing in MTRL training, off-policy learning algorithms such as DQN [11] and SAC [12] can be used to train on experiences from other tasks. The work in [6] discusses the sharing capability among multiple tasks by employing the V-trace correction. In [13], the authors use a filtered agent to select the most relevant experiences for a specific task from all available agents' experiences. These selected experiences are expected to reduce the effects of distribution shift. Collectively, these studies suggest that if the policies are similar enough, directly using experiences from other tasks can improve the overall training process. From a domain adaptation perspective, Huang et al. [14] generate a sequence of tasks that can be used to improve learning efficiency based on the Wasserstein barycenter. Several works [15, 16] rely on optimal transport to establish policy distances in imitation learning or offline RL.

In this paper, we explore the idea of directly sharing training experiences among multiple tasks in a sparse-reward environment. We propose the Weighted Experience Sharing using Sinkhorn Distances (WES-SD) method, which is based on Soft Actor-Critic off-policy learning. In WES-SD, a common replay buffer is used to collect training samples from individual tasks. To reduce sample distribution mismatch between tasks, an adaptive weight is assigned to each agent's collected experiences. When an experience is shared with another agent, this weight is incorporated into the actor and critic update according to the loss function. This procedure allows experiences from the initial exploration phase to improve sample efficiency by assigning them a high weight. Once each agent begins to optimize its specific task, WES-SD employs Sinkhorn distances to measure policy divergence. By detecting divergence, the shared experiences are reweighted based on the degree of policy difference between tasks. This updated weight provides a flexible way to

control how much information is shared across different tasks. To evaluate the effectiveness of our method, we conducted experiments on two goal-based navigation environments using 2, 3, and 4-task settings. WES-SD outperformed the baseline methods in both environments, improving average rewards by approximately 8% to 10%. Compared to other baselines, our approach reduces the effects of policy divergence while maintaining effective sharing ratios during exploration.

2. Related Works

Multitask RL: While it is possible to train an individual policy to optimize each targeted reward, MTRL methods aim to improve training efficiency by optimizing all environments simultaneously and reusing knowledge from other tasks [8, 17]. Possible approaches include gradient-based regularization and experience sharing. For gradient-based methods, MAML [8] attempts to find a direction that accounts for the optimized direction of each policy. Meanwhile, PCGrad employs an operator to resolve conflicts in the gradient update directions across multiple tasks [9]. Distral [18] proposes a framework that maximizes task-based objectives while minimizing the distance to a central policy. From an optimization perspective, Distral can be categorized as a gradient-based method in which regularization constrains the update direction of the policy. Instead of using a central policy, the work in[19] proposes a guidance policy for efficiently collecting experiences in MTRL. On the other hand, the work of SACMT [20] successfully extends SAC to the multi-task RL domain. The main drawback of the gradient-based approach is filtering out the noisy gradients from different tasks.

From an experience-sharing perspective, shared samples act as exploration experiences that can be used to optimize a specific reward function. Unlike a standard exploration policy, samples coming from a different task must be adjusted or selected to remove bias toward the shared reward functions [6, 13, 21]. However, directly using data from other tasks introduces bias and can negatively interfere with the learning of the current tasks. Importance sampling (IS) is a popular approach that reweights each sample to correct the bias resulting from mismatches between task distributions [22, 23]. Although IS has a solid theoretical foundation [22], empirical results indicate that its estimates suffer from high variance and are only useful in certain cases.

Policy Distance: Quantifying the discrepancy between tasks is critical for modeling the connections between different policies in many RL problems. One popular approach relies on various information-theoretic metrics, such as Kullback–Leibler (KL) divergence [24] or mutual information [25]. These metrics have the practical advantage of providing closed-form expressions for some widely used parametric policy distributions or being easily estimated from samples. Alternatively, optimal transport distances and their entropy-regularized versions [26] have recently been applied to several RL problems, including imitation learning [27, 28], unsupervised RL [29], and offline RL [24]. In addition to KL divergence, optimal transport distances offer a practical means of measuring policy differences. For example, the authors in [30] propose an auxiliary reward built upon the Wasserstein distance between two trajectories to distill knowledge between policies. Recently, optimal transport has emerged as a robust framework for modeling policy distances across various RL domains, such as imitation learning and offline training [15, 16].

3. Backgrounds

Our WES-SD employs the setting of a multi-task environment with spare-reward of N tasks. It can be formulated as a tuple of $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R_i, \mathcal{E}_i, \gamma \rangle$ with $i = 1 \dots N$ as follows:

- Let $\mathcal{S}$ be the set of states in the MDP, $\mathcal{A}$ is the set of actions with the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. An initial probability distribution function is provided if necessary. In general, we can assume the initial distribution is the uniform distribution over $\mathcal{S}$.

- For each task i, given the discount factor γ, a policy π_i is trained to maximize the reward function R_i with respect to a specific set of terminal state $\mathcal{E}_i$. The reward function R_i is defined as a mapping:

$$R_i : \mathcal{S} \rightarrow \mathbb{R} \tag{1}$$

  Additionally, we assume that the reward function R_i is sparse, meaning that for most of states, the reward signal is 0

$$R_i(s) = \{0 | \forall s \in \mathcal{S}, s \notin \mathcal{E}_i\}.$$

  Only a limited set of terminated states $s_e \in \mathcal{E}_i$ provide a positive rewards $R_i(s_e) > 0$. An episode terminates when the agent reaches one of these terminal states. We further assume that $|\mathcal{E}_i| \ll \mathcal{S}$.

- The objective function of task i is defined as the expected discounted cumulative reward when starting from an initial state s_0 and following the policy π_i:

$$J(\pi_i) = \mathbb{E}_{\pi_i}\left[\sum_{t \geq 0} \gamma^t R_i(s_{t+1})\right] \tag{2}$$

  where $s_{t+1} \sim \mathcal{T}(s_t, a \sim \pi_i(.|s_t))$ and $s_0 \sim \mathcal{P}_0$

- The overall objective across N tasks is defined as the sum of individual objective functions:

$$J(\{\pi_{i=1}^N\}) = \sum_i J(\pi_i) \tag{3}$$

In other words, each task i differs only in its the specific reward function R_i and terminal state $\mathcal{E}_i$. To maximize the total objective function in Equation 3, one can choose to train each task i separately to maximize its individual objective $J(\pi_i)$.

In our work, we consider each task i as a subtask which can be used to construct a central policy π_0. The π_0 is trained to maximize the following new reward function $R_0 : \mathcal{S} \rightarrow \mathbb{R}$ as follow:

$$R_0(s) = \begin{cases} 1, & \text{if } \sum_{i=1}^N R_i(s) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

We further assume that each subtask reward functions R_i are normalized to the range $[0, 1]$ and R_0 can be interpreted as an indicator function. The terminated set of states of the central policy $\mathcal{E}_0 = \bigcup_{i=1}^N \mathcal{E}_i$, which is a combination of ending set of each task i.

Our optimization framework for multi-task learning is built on an off-policy learning method that allows experiences can be shared across different tasks. We choose the popular Soft Actor-Critics (SAC) [12] algorithm as the base learning method. In SAC, the policy is trained to maximize a maximum entropy objective, which augments the expected discounted cumulative rewards with expected entropy $\mathcal{H}(\pi(\cdot|s_t))$ of the policy at each timestep:

$$J(\pi) = \sum_{t=0} \mathbb{E}_{s_t, a_t} \gamma^t [r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))]. \tag{5}$$

To maximize the objective in Equation 5, the SAC framework alternates between evaluating the soft-critic value function $Q_(s, a)$ and updating the actor to minimize the Kullback-Leibler (KL) divergence between the learning policy and the policy derived from the soft Q-value function. The entropy coefficient term α plays an important role to control the exploration behaviour of the learned SAC policy. For a given environment, one can set a target entropy $\mathcal{H}_{target}$ and update the value of α to match the target value [31]. Although, other optimization algorithm, such as

DQN, can serve as the underlying learning policy, SAC is chosen as it is more flexible in exploration by controlling the regularized entropy in Equation 5. In the sparse settings of our work, a high target entropy $\mathcal{H}_{target}$ helps stabilize the exploration phase.

4. Method

4.1. Sharing Experiences Framework through Policy Divergence

To share experiences among multi-task agent training, we propose the Weighted Experience Sharing with Sinkhorn Distance (WES-SD) method. Our approach utilizes a central buffer that aggregates and directly shares experiences. To control the effects of policy divergence across tasks, an adaptive weighting mechanism is applied to each shared experience. These weights are designed to enhance experience sharing during the exploration phase and mitigate the effects of policy divergence during the exploitation phase. An overview of our framework is shown in Figure 1. For a specific objective $\mathcal{T}_i$, each agent interacts with its environment to maximize its own reward function. The experiences of the i^{th} agent are collected and stored in the buffer replay B_i as a tuple of state s, action a, next state s' and reward r. An additional weight w is added, initially set to 1, and is later used for training the actor and critic of SAC.

In order to use experiences from other tasks, a sharing buffer B_0 is used to collect samples from all the tasks $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_N$. Each agent's buffer B_i is then combined with its own experiences and samples from common buffer B_0. The buffer B_i is used to train its policy π_i using the SAC algorithm. At each SAC updating step of the i^{th} agent, additional samples from the central buffer B_0 are added to buffer B_i. However, the weight of these additional samples is adjusted to control the effects of policy divergence. In contrast, samples originating from task i always have a weight of 1.

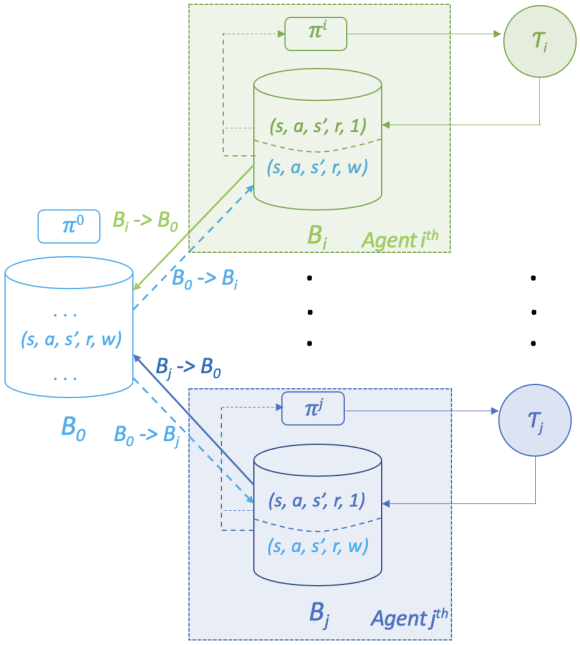

Figure 1. An overview of our Weighted Experience Sharing with Sinkhorn Distance framework. The central buffer B_0 aggregates samples from individual tasks, such as, B_i and B_j in this example. When a sample is drawn from B_0 to B_i, it is assigned an additional weight to account for the policy divergence.

As illustrated in Figure 1, B_0 serves as the buffer of the central policy π_0, which is trained to maximize the reward function R_0 in Equation 4. The purpose of buffer B_0 is to act as a temporal shared memory among the policies. Each sample in B_0 is assigned a specific weight to estimate the divergence between policy π_i to the central π_0:

$$w_{B_i \to B_0} = \frac{d(\pi_i, \pi_0)}{C(\pi_0)}, \tag{6}$$

where $d(\pi_i, \pi_0)$ is the measured divergence of policy π_i to π_0 and $C(\pi_0)$ is the normalized factor for the central policy π_0. Based on the weight in Equation 6, when task j draws samples from B_0, a reversed sample weight is calculated as follows:

$$w_{B_0 \to B_j} = \frac{d(\pi_j, \pi_0)}{C(\pi_j)} \tag{7}$$

To avoid training an additional central policy π_0, we further combine Equation 6 and Equation 7 by incorporating the normalization constant $C(\pi_0)$ to $C(\pi_j)$. Specifically, an experience from buffer B_i can be shared to buffer B_j with a new weight:

$$w_{B_i \to B_j} = C(\pi_j)\frac{d(\pi_i, \pi_0)}{d(\pi_j, \pi_0)} \tag{8}$$

At the early stage of training, when the two policies π_i and π_j are in the exploration phase and exhibit similar performance, the ratio $\frac{d(\pi_i,\pi_0)}{d(\pi_j,\pi_0)}$ is closed to 1. However, at the later stage, when the two policies π_i and π_j begin to optimize their targeted reward function R_i, they start to diverge. In this case, an experience collected from task i may introduce noise to the task j. Our WES-SD is designed to detect such changes during training process and to mitigate the effect of distribution drifting to the policy π_j when its training buffer B_j incorporates samples from other policies π_i.

4.2. Estimated the Policy Divergence with Sinkhorn Distance

To estimate the transfer weight in Equation 8, it is required to compute the policy divergence between a specific task π_i to the central policy π_0. There are several methods to compute the policy divergence such as KL-divergence [18]. However, this approach requires direct access to the central policy π_0, which could be very noisy because the reward function R_0 is combination of task-specific rewards.

In our work, we propose an alternative approach that measures the policy distance $d(\pi_i, \pi_j)$, or $d_{i,j}$ for short, between each task pair i, j and uses this measurement to estimate the divergence measurement $d_{i,0}$. As the state and actions distributions in a sparse reward environment change slowly during the exploration phase, we can rely on the buffer replay B_i to represent the distribution of the policy π_i. As our sharing framework is required frequent computation the pairwise distance $d(\pi_i, \pi_j)$, we employ the Sinkhorn Distance [26], which is widely used for comparing distributions based on data samples. For each task π_i and π_j, we take the last k trajectories from their respective replay buffers B_i and B_j. The sample representation from each buffer is a tuple of $< s, a, s', r >$. Without the lost of generality, we denote the samples from the two buffer replays as $\{\tau_i\}$ and $\{\tau_j\}$, respectively.

The Sinkhorn Distance modifies the transport cost of the Wasserstein Distance by subtracting an entropy regularize term. The introduced entropy term makes the computation of the Wasserstein Distance more tractable by smoothing the transport plan. By leveraging this smooth property, the original optimization is converted into an convex one that can be solved by iterative methods with quadratic time complextiy. Specifically, the Sinkhorn Distance finds the transport matrix P which maximizes the following objective function W_ϵ:

$$W_\epsilon = \min_{P \in U(\{\tau_i\},\{\tau_j\})} \langle C, P \rangle - \epsilon H(P) \tag{9}$$

where $U(\{\tau_i\}, \{\tau_j\})$ denotes the set of transport mappings from $\{\tau_i\}$ to $\{\tau_j\}$, $H(P)$ denotes the entropy of the mapping P and ϵ is the weight of the entropy regularizer. The transportation cost matrix C defines the cost for moving a sample from one distribution to another. Since each distribution is represented by a set of observation from the underlying task, we employ a L1-distance between observation as follows:

$$C(x, y) = \|x - y\| \tag{10}$$

with $x \in \{\tau_i\}$, $y \in \{\tau_j\}$.

In [26], the author proposes to find the solution of Equation 9 in the following form:

$$P^* = \text{diag}(u^*) \odot K \odot \text{diag}(v^*) \tag{11}$$

where u^* and v^* are the two diagonal parameterized of P^*, K is the kernel matrix, and $\odot$ denotes element-wise multiplication

between two matrixes. The kernel matrix K is the smooth version of the cost matrix C obtained using the regularized term:

$$K = \exp(-\frac{C}{\epsilon}) \tag{12}$$

The optimal value u^*, v^* can be computed by an iterative process [26]. The Algorithm 1 describes the computing of Sinkhorn Distance between two set of trajectories.

Algorithm 1 Sinkhorn Distance (SD) for computing $W_\epsilon(\{\tau_i\}, \{\tau_j\})$ with ϵ is the regularized entropy weight

SD($\{\tau_i\}, \{\tau_j\}$)
  Initialize C as in Equation 10
  $K \leftarrow \exp(-\frac{C}{\epsilon})$
  Initialize $u \leftarrow \mathbf{1}$, $v_0 \leftarrow \mathbf{1}$
  for $t = 1, 2, \ldots,$ max_iter:
    $u_t \leftarrow \frac{\mathbf{1}}{Kv_{t-1}}$
    $v_t \leftarrow \frac{\mathbf{1}}{Ku_{t-1}}$
  end for
  $P \leftarrow \text{diag}(u_t) \odot K \odot \text{diag}(v_t)$
  Return TRACE($C^T P$)

Given the Algorithm 1, the distance can be computed for any pair of trajectory set π_i and π_j. Based on the computed distance, we derive Algorithm 2, which outputs the pairwise distance as defined in Equation 8. Firstly, if B_i and B_j come from the two different tasks, it is straightforward to compute the distance $l_{i,j}$ by relying on the last k trajectories. We rely on the last k trajectory in Line 5 because we want to focus on the divergence of the two policies based on their most recent rollouts. The number of trajectory k is chosen as a hyper-parameter. Secondly, to stabilize the distance measuremen, we employ a running average with coefficient λ and the previous pairwise distance $\hat{d}_{i,j}$. If the λ is set to 1, the new distance is used directly for computing the sharing weight.

When the two buffers B_i and B_j come from the same task, the distance between the last k trajectories and the last $2k$ to $k+1$ trajectories are used as stated in Line 7 of Algorithm 2. This value measures the divergence between the policy i with itself. The self-distance $d_{i,i}$ is later used as the normalized constant $C(\pi_i)$ as in the Equation 8. Note that, for each of the two cases $i = j$ or $i \neq j$, the two buffers, B_i and B_j, are assumed to contain enough samples to form two trajectory sets $\{\tau_i\}, \{\tau_j\}$, respectively.

In our approach, we do not train directly π_0 because training π_0 requires a combination of experiences from each individual task. Instead, we rely on the pairwise distance of between each pair individual task i to approximate the distance $d_{i,0}$. Because the central policy is a mixture of other tasks, we propose to estimate $d_{i,0}$ by averaging the distance from task i to the other N tasks as follows:

$$d_{i,0} = \frac{\sum_{j=1}^{N} d_{i,j}}{N}. \tag{13}$$

Algorithm 2 Computing the policy distance between two policy π_i and π_j with running average parameter λ

1: DISTANCE($B_i, B_j, \hat{d}_{i,j}, k, \lambda$)
2: Let $B_i = \{\tau_i^0, \ldots \tau_i^h\}$
3: Let $B_j = \{\tau_j^0, \ldots \tau_j^s\}$
4: **if** $i = j$:
5: $l_{i,j} = \text{SD}(\{\tau_i^{h-2k+1 \to h-k}\}, \{\tau_i^{h-k+1 \to h}\})$
6: **if** $i \neq j$:
7: $l_{i,j} = \text{SD}(\{\tau_i^{h-k+1 \to h}\}, \{\tau_j^{s-k+1 \to s}\})$
8: $d_{i,j} = (1-\lambda)\hat{d}_{i,j} + \lambda l_{i,j}$
9: Return $d_{i,j}$

By combining the Algorithm 2 and Equation 13, it is possible to estimate the pairwise distance for all the policies $\pi_0, \pi_1, \ldots, \pi_N$. Since the pairwise distance could be noisy, the variance can be controlled by adjusting the number of trajectory k and the running average value λ. However, in the sparse signal reward environment, the policy focuses on the

exploration, so the pairwise distance is expected to be estimated stably.

4.3. Training Pipeline by Weighting Experiences with Task Distance

Using the proposed distance estimation in Algorithm 2, we arrive at the following training procedure in Algorithm 3. The algorithm receives a list of policies π_i and trains them by the SAC algorithm. The main difference between WES-SD and the SAC baseline is the introduction of the shared replay buffer B_0 and the pairwise distance $d_{i,j}$. Before training, all the buffers are empty and the distances $d_{i,j}$ are initialized to 1. During, each buffer B_i collects the experiences as the policy π_i interact swith its environment $\mathcal{T}_i$. Each collect sample is assigned a training weight w according to the Equation 8 for subsequent SAC updates. The weight of samples of task i in the buffer B_i are set to the default value 1. However, in the central buffer B_0, the weight is the reverse distance from task i^{th} to the central policy, which is $1/d_{i,0}$, specifically.

Algorithm 3 Experience sharing between multi-task in with sharing buffer replay

1: TRAIN($\pi_1, \pi_2, ..., \pi_N, M, \delta$)
2: Initialize $B_i = \{\}, \forall i \in [0, 1, \ldots, N]$
3: Initialize $d_{i,j} = 1, \forall i, j \in [0, 1, \ldots, N]$
4: **for**($i = 1, \ldots, N$)**:**
5: Let π_i interact with environment $\mathcal{T}_i$
6: $B_i \leftarrow (s, a, s', r, w = 1)$
7: $B_0 \leftarrow (s, a, s', r, w = 1/d_{i,0})$
8: **for**($j = 1, \ldots, N$)**:**
9: **for**($i = 1, \ldots, M$)**:**
10: Draw $(s, a, s', r, w) \sim B_0$
11: $B_j \leftarrow (s, a, s', r, w' = \delta * w * d_{j,j}/d_{j,0})$
12: RUN_SAC_UPDATE(π_j, B_j)
13: **if** (WARM_UP): Continue
14: **for**($i, j = 1, \ldots, N$)**:**
15: $d_{i,j} \leftarrow$ DISTANCE($B_i, B_j, d_{i,j}$)
16: $d_{i,0} \leftarrow \sum_{j=1}^{N} d_{i,j}/N$

When a sample is drawn from the central buffer B_0 and added to B_i, its weight is updated as in Line number 11 and is added to the shared experiences. The normalized constant to each task j, $C(\pi_j)$, is computed as a divergence from task j^{th} to itself. We further employ a sharing constant δ, which serves as a hyperparameter to control the sharing weight. The value of δ depends on the nature of the environment and the combined tas $\mathcal{T}_0$. Moreover, the number of sharing experiences M is introduced to balance intra-task and inter-task samples. In practice, we select M as a percentage of the newly added samples during the sampling steps in Line 5. After new samples from the central buffer B_0 are added to B_j, each policy π_j is trained with the SAC objective in Equation 5. We update the pair-wise distance matrix $d_{i,j}$ using the procedure from Line number 14-16. To prevent an empty buffer when computing the distance, a warm-up phase is implemented at the beginning of the training stage, as shown in Line 13.

Complexity Analysis: Our WES-SD has a complexity of $O(NM)$ for drawing from the central buffer replay B_0. The training time of the each π_i would be the same as SAC, with more training samples. Additionally, tWES-SD must update the pair-wise distance d_{ij} with a complexity if $O(N^2)$. The computation of d_{ij} can be approximated by the runtime complexity of Sinkhorn Distance between a two sets of k trajectories, each with T_{max} timesteps. The complexity of Sinkhorn Distance can be estimated by $O(cS^2)$, where S is the maximum total number of samples in each set $\{\tau_i\}, \{\tau_j\}$ and c is a constant determined by the regularized entropy weighted ϵ. Therefore, at each training steps, the time complexity of our WES-SD is $O(NM + cN^2S^2)$, compared to the standard SAC algorithm. The memory complexity is $O(N|B| + N^2)$ with $|B|$ denoted the maximum capacity of the replay buffer.

5. Experiment Results and Discussions

In this section, we design our experiment to asnwer the following research questions (RQ):

- RQ1: The performance of our sharing framework in comparison with other popular sharing frameworks and other distance-based approaches?
- RQ2: How sensitive of the sharing constant δ in controlling the policies' divergence in WES-SD?
- RQ3: How the state representation could affect the performance of WES-SD?
- RQ4: Ablation study on other hyperparameters and time comparision of the proposed WES-SD.

5.1. Experiment Settings

Environments. We evaluate the proposed algorithm using two navigation environments. During the training phase, each agent starts at a random position and moves within the environments. Agents receive position reward when they reach the target and the episode is terminated. In the testing phase, the agents' task is to move to target position. Different tasks have different goal positions, which are fixed between training and testing phases. The details of each environment are described as follows:

Corridor-TwoRooms: We reuse the simple set up of multi-tasks with two rooms and a connecting corridor between two rooms in [18]. The environments are tested with 2, 3 and 4 tasks, which are illustrated in the left image of Fig. 2. The agent is started at a random position and can move into four direction: up, down, left, right. An episode is terminated if the agent can reach the goal position or its number of interactions reaches to a maximum timestep. When the agent can reach the goal, agents receive a reward maximum at 1 and reduce linearly with the length of the episode. The input observation is a tuple of 3×3 squares around the agent. The value of each element is one of 0-empty, 1-wall or 2-target.

MiniGrid-FourRooms: The environment is a discrete, grid-based spare-reward environment where the agent (the red triangle) navigates in a maze with partial observations of the environment. In our experiment, we use the custom FourRooms environment with different goals in each task, which is illustrated in the right image of Fig. 2. There are walls which divided each room. The walls are fixed in the environment and do not change between episodes. In this setting, the starting position and direction of the agent is randomized at a valid square. The agent can carry one of three actions: moving forward, turn left or turn right. Similarly to the Corridor-TwoRooms environment, the reward has a maximum value at 1 and reduce linearly with the length of the episode. The agent receives zero reward and is terminated if it performs more than a maximum number of actions.

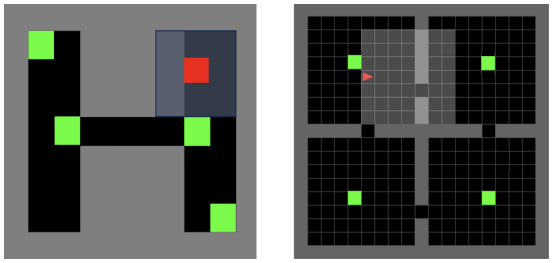

Figure 2. The two testing environments: Corridor-TwoRooms on the left image and Minigrid-FourRooms on the right image. The agent is denote with red with its observation distance and the target positions are denoted with green square.

Baselines. We compare our WES-SD with several baselines, which can be divided into three groups:

- *SAC*: We train a standard SAC agent for each tasks. As our environment is discrete action, we follow the work in [32], which adapts the SAC framework for discrete action environments.

Table 1. Hyper-parameters for models and training settings

Params	**Two Room Corridor**	**Minigrid-FourRoom**
Observation	1x3x3	3x7x7
No. of Actions	4	3
Feature Type	MLP	2xConv2D
Buffer size	25K	100K
Warmup buffer	5K	5K
Minibatch size	128	128
Entropy	0.6	0.6
Training Steps	50K	200K

- *SAC-Sharing*: We employ a simple version of directly experience sharing based on SAC agents. Each agent can use experiences from other tasks for optimizing its training update. The sharing mechanism is follow the Alg. 1, except the updated weight at line 8. In the SAC-Sharing method, all the pair-wise distance is set to 1. Thus, the sharing weight now depends only on the sharing constant δ. We vary the sharing constant δ in the set of {0.2, 0.5, 1.0}, which aims to control the sharing information with different degrees. Note that, a sharing constant $\delta = 1.0$ means that the samples from B_0 have equals weight to samples in each of B_i.
- *SAC-MT* [20], *PCGrad* [9], *Distral* [18]: we employ the three multi-task sharing baselines from the literature to compare its effectiveness with our method in sparse multi-task environments.

All sharing methods are implemented based on SAC algorithm for a fair comparison. Network architecture and common hyper-parameters are kept the same for all methods of the same types. We give a summarize of key hyperparameters in the Table 1. For our WES-SD method, the sharing constant is chosen at $\delta = 0.2$, the number of trajectory $k = 3$, and the running weights $\lambda = 0.8$. The number of sharing samples M is selected to equal with the new collected samples of each buffer B_i in each training iteration.

Given a multi-task training set up, each tasks is trained parallel given the maximum number of interactions steps. For the number of training steps, as the Corridor-TwoRooms environment has a low complexity, the maximum number of steps is set with 50K samples. For the Minigrid-FourRooms environment, we set the maximum interaction steps for 200K steps. For each environment and number of tasks, we train with 3 different seeds and report the average testing rewards on all goals. The testing rewards are average across 100 trajectory for each tasks.

5.2. RQ1: Main Results

We provide the testing rewards for the Corridor-TwoRooms settings in Table 2 and the Minigrid-FourRooms in Table 3. In the first environment, our WES-SD achieves the best performance in 2 of 3 testing configs. The SACMT method has the best performance on the remaining 2-tasks setting while our method ranks third with a slightly lower performance. Our method has the best rank of 1.7 and the best reward of 0.42 in average. Surprisingly, the SAC baseline without sharing has the second-best performance. The third performance method is the SAC-Sharing baseline with the sharing weight $\delta = 0.5$. The two sharing-based approaches, SACMT and Distral, have lower performance than our WES-SD and only have an average rank of 4.3.

Similarly, in the Minigrid-Fourooms environment, our WES-SD has the best performance in all the three settings. The SAC-Sharing baseline with $\delta = 0.5$ and $\delta = 1.0$ rank second and third place in average ranking, respectively. Our method achieves the highest performance. As our distance can be adapted to different phase of each individual policy, it outperforms the SAC baseline with constant weight. For other sharing-based methods, the SACMT has the most competitive performance

Table 2. Results with average and the standard deviation of testing rewards on Corridor-TwoRooms environment in the configs of 2, 3 and 4 multi-tasks. The **best**, second and *third* performance for each columns are highlighted with bold, underline and italics, respectively.

Method	2 tasks	3 tasks	4 tasks	Avg. Rewards	Avg. Rank
SAC	0.41 ± 0.04	0.36 ± 0.13	0.36 ± 0.07	0.38	4.0
SAC-Sharing (δ = 0.2)	0.45 ± 0.13	0.23 ± 0.06	*0.32 ± 0.07*	0.33	4.0
SAC-Sharing (δ = 0.5)	0.42 ± 0.06	*0.38 ± 0.08*	0.29 ± 0.08	*0.36*	*4.0*
SAC-Sharing (δ = 1.0)	0.34 ± 0.05	0.22 ± 0.05	0.24 ± 0.12	0.27	7.3
SACMT	**0.46 ± 0.11**	0.31 ± 0.12	0.28 ± 0.12	0.35	4.3
PCGrad	0.32 ± 0.24	0.40 ± 0.07	0.23 ± 0.11	0.29	6.0
Distral	0.36 ± 0.03	*0.38 ± 0.05*	0.31 ± 0.14	0.35	4.3
WES-SD	*0.44 ± 0.14*	**0.43 ± 0.09**	**0.39 ± 0.04**	**0.42**	**1.7**

Table 3. Results with average and the standard deviation of testing rewards on Minigrid-FourRooms environment in the configs of 2, 3 and 4 multi-tasks. The **best**, second and *third* performance for each columns are highlighted with bold, underline and italics, respectively

Method	2 tasks	3 tasks	4 tasks	Avg. Rewards	Avg. Rank
SAC	0.23 ± 0.10	*0.24 ± 0.02*	0.18 ± 0.04	0.22	5.3
SAC-Sharing (δ = 0.2)	0.25 ± 0.08	0.22 ± 0.02	0.23 ± 0.03	0.23	4.7
SAC-Sharing (δ = 0.5)	*0.26 ± 0.01*	0.25 ± 0.01	*0.21 ± 0.04*	0.24	3.0
SAC-Sharing (δ = 1.0)	0.27 ± 0.05	0.23 ± 0.05	0.23 ± 0.05	*0.24*	*3.3*
SACMT	*0.26 ± 0.05*	*0.24 ± 0.04*	0.20 ± 0.04	0.23	3.7
PCGrad	0.23 ± 0.01	*0.24 ± 0.02*	0.20 ± 0.03	0.22	4.7
Distral	0.16 ± 0.07	0.19 ± 0.05	0.14 ± 0.07	0.16	8.0
WES-SD	**0.29 ± 0.03**	**0.26 ± 0.01**	**0.24 ± 0.02**	**0.26**	**1.0**

which is slightly lower than the constant weight baselines.

Note that, in the two settings, the Distral method which employs the KL-divergence for adjusting the central policy fails to achieve a good performance. It can be explained that the KL-divergence is more sensitive to noise. In a sparse-reward environment, each task policy is mostly in the exploration mode with high amount of noise signal from environment. Hence, the KL-divergence makes the central policy unstable. On the other hand, our WES-SD is designed based on the Sinkhorn distance which is more robust to noise and can improve the average rewards 20% in comparison to Distral.

Table 4. Results with average and the standard deviation of testing rewards on the 4-tasks setting of Corridor-TwoRooms and Minigrid-FourRooms with different sharing constant values δ

Sharing Constant	Corridor-TwoRooms	Minigrid-FourRooms
δ = 0.1	0.36 ± 0.15	0.21 ± 0.03
δ = 0.2	**0.39 ± 0.04**	**0.24 ± 0.02**
δ = 0.5	0.29 ± 0.07	0.21 ± 0.04
δ = 1.0	0.34 ± 0.03	0.22 ± 0.03

5.3. RQ2: The Effects of Sharing Constant δ

From the results in the Table 2 and 3, it is clear that the sharing constant δ has a significant impart on the weight sharing performance. In this section, we further vary the sharing constant δ as

Table 5. Results with different observation choices on the 4-tasks setting of Corridor-TwoRooms and Minigrid-FourRoom for weight computing

Input Observation	Corridor-TwoRooms	Minigrid-FourRooms
$\langle s \rangle$	0.35 ± 0.03	0.22 ± 0.03
$\langle s, a \rangle$	0.30 ± 0.06	0.24 ± 0.03
$\langle s, a, s' \rangle$	0.37 ± 0.04	0.21 ± 0.03
$\langle s, a, s', r \rangle$	**0.39 ± 0.04**	**0.24 ± 0.02**

it controls the effects of the experiences which are sampled from the buffer B_0. The result on the two environments with four task setting is presented in the Table 4. From the Table, the best result can be achieved with $\delta = 0.2$. With the value of $\delta = 0.1$, the performance is slightly lower and achieves the second best performance. In all 4 tested values of δ, a lower value is more prefer with more stable value than high values of delta. On the other hand, the standard SAC-Sharing baseline with $\delta = 0.5$ has the best performance among the other values across the two tested environments. For comparison between our WES-SD and the SAC-Sharing, the interaction between tasks in our framework is more flexible than the latter.

5.4. RQ3: Analysis on Different Input Observations

The input observation plays a central role to compute the distance between two policy. It is used to construct the cost matrix C in Equation 9 and Equation 10. In the standard setting, the full observation $\langle s, a, s', r \rangle$, which includes state, action, next state and reward is used. We alternate the input observation for verifying its effectiveness in the WES-SD method From the full observation $\langle s, a, s', r \rangle$, we remove each of the component and construct the cost matrix with: *state only* $\langle s \rangle$; *state and action* $\langle s, a \rangle$; *state, action and next state* $\langle s, a, s' \rangle$. The Table 5 illustrates the result of different input observations.

From the Table 5, it can be seen that the full observation has the highest performance for the experience sharing framework. When the information is removed from the full observation, the performance is reduced in general. In the Corridor-TwoRooms environment, there are reductions from 5%-10% of the average rewards. However, in the Minigrid-FourRooms environment, the state and action config has a performance as good as the full observation. Note that, the input representation is only involved in cost matrix computation, which does not contribute much to the computation time of the Sinkhorn algorithm. Therefore, it is reasonable to use the full observation as the default setting in our proposed WES-SD.

5.5. RQ4: Ablation Study

In this section, we evaluate the performance of our WES-SD with different values of k and λ as desribed in the Algorithm 2, and we compare the running time of our baseline with other methods. The two settings used in this section are the 4-task configurations of Corridor-TwoRooms and Minigrid-FourRooms. Each experiment is run with three different seeds, and the mean testing rewards are reported.

- Number of trajectories k: We iterate the value of k over the set $\{3, 4, 5\}$. The results are reported in the Table 6. In general, as k is increases, the sharing effect diminishes in both environment settings. The main reason is that with a larger number of samples, the estimated Sinkhorn Distance between two tasks becomes more stable. Consequently, our WES-SD behaves more similarly to SAC-Sharing with a constant δ.

- Running weight λ: We vary the value of λ in the range of $[0.8, 1.0]$. The Table 7 highlights the results. When the value of λ is increased from 0.8 to 1.0, the pair-wise distance becomes more noisier, which reduces the overall performance.

Table 6. Results with different values of k on the 4-tasks setting of Corridor-TwoRooms and Minigrid-FourRoom

Number of Trajectory	Corridor-TwoRooms	Minigrid-FourRooms
$k = 3$	**0.39 ± 0.04**	**0.24 ± 0.02**
$k = 4$	0.31 ± 0.11	0.19 ± 0.05
$k = 5$	0.29 ± 0.07	0.23 ± 0.03

Table 7. Results with different values of λ on the 4-tasks setting of Corridor-TwoRooms and Minigrid-FourRoom

Running Weight	Corridor-TwoRooms	Minigrid-FourRooms
$\lambda = 0.8$	**0.39 ± 0.04**	**0.24 ± 0.02**
$\lambda = 0.9$	0.32 ± 0.04	0.21 ± 0.02
$\lambda = 1.0$	0.25 ± 0.04	0.21 ± 0.04

- Training time comparision: Table 8 presents the training time comparisons between our WES-SD and other baselines. Since the running time for the SAC-Sharing baseline is similar across different settings, we only report the results for $\delta = 0.2$. The table shows that our WES-SD requires 2.5 times more training time than standard SAC. This overhead is primarily due to the increased number of training samples. The computation gap can be reduced by lowering the value of M. Compared to the SAC-Sharing baseline, the WES-SD training time is is 10% longer due to the additional computation of the pairwise distance matrix. Among the other baselines, PCGRAD has a similar performance to our WES-SD, while both SACMT and Distral run efficiently and achieve performance comparable to the SAC baseline.

6. Conclusions

In this paper, we present the WES-SD, a distance-based metric for experience sharing in multi-task RL. Our method is designed for the spare-reward environments where the agent has to balance between exploration and exploitation phase. The WES-SD is built upon the SAC algorithm and employs a central sharing buffer which aggregates all of the experiences from the other tasks and is also used to share the experiences. To control the policy divergence in shared samples, we introduce a Sinkhorn-based distance for computing the pairwise distance from two individual policies. The distance is then used to compute the sharing weight for samples. We evaluate our methods on two goal-based navigation environments with 2, 3 and 4 tasks settings. The results show that WES-SD achieves the best average reward and ranking in both environments. We also conduct experiments with several key parameters and illustrate their impact on the overall performance of the method

Table 8. Training time in seconds for 10K samples on Corridor-TwoRooms and Minigrid-FourRooms with the setting of 4 tasks

Methods	Corridor-TwoRooms	Minigrid-FourRooms
SAC	14.1 ± 0.2	22.0 ± 0.2
SAC-Sharing	35.9 ± 0.1	45.1 ± 0.2
SACMT	18.5 ± 0.2	26.1 ± 0.1
PCGrad	40.9 ± 0.4	49.2 ± 0.3
Distral	14.3 ± 0.3	21.8 ± 0.3
WES-SD	41.3 ± 0.4	50.0 ± 0.4

From the results of our paper, one possible extension is to replace the Sinkhorn distance with other measures of policy divergence such as Slice Wasserstein Distance or Total Variation Distance. In addition, because sharing samples can substantially increase training complexity, a more advanced approach would be to balance training efficiency and the exploration signal by employing an adaptive sharing weight δ. The appropriate value of δ could be estimated as a function of the entropy of the sharing policy. Given the flexibility of our method, it

can be extended to other types of environments, such as dense-reward settings or continuous-action domains. We also plan to develop a complexity analysis framework to better balance the exploration and exploitation trade-off in multi-task experience-sharing settings.

Acknowledgements

This work has been supported by VNU University of Engineering and Technology under project number CN24.13

References

[1] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, F. L. Lewis, Optimal and Autonomous Control Using Reinforcement learning: A Survey, IEEE Transactions on Neural Networks and Learning Systems, Vol. 29, No. 6, 2017, pp. 2042–2062. doi:10.1109/TNNLS.2017.2773458.

[2] K. Arulkumaran, M. P. Deisenroth, M. Brundage, A. A. Bharath, Deep Reinforcement Learning: A Brief Survey, IEEE Signal Processing Magazine, Vol. 34, No. 6, 2017, pp. 26–38. doi:10.1109/MSP.2017.2743240.

[3] D. Pathak, P. Agrawal, A. A. Efros, T. Darrell, Curiosity-driven Exploration by Self-supervised Prediction, in: International Conference on Machine Learning, PMLR, 2017, pp. 2778–2787.

[4] A. Wilson, A. Fern, S. Ray, P. Tadepalli, Multi-task Reinforcement Learning: A Hierarchical Bayesian Approach, in: Proceedings of the 24th International Conference on Machine learning, 2007, pp. 1015–1022.

[5] H. Li, X. Liao, L. Carin, Multi-task Reinforcement Learning in Partially Observable Stochastic Environments, Journal of Machine Learning Research, Vol. 10, No. 5, (2009).

[6] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al., Impala: Scalable Distributed Deep-rl with Importance Weighted Actor-learner Architectures, in: International Conference on Machine Learning, PMLR, 2018, pp. 1407–1416.

[7] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, MIT press, 2018.

[8] C. Finn, P. Abbeel, S. Levine, Model-agnostic Meta-learning for Fast Adaptation of Deep Networks, in: International Conference on Machine Learning, PMLR, 2017, pp. 1126–1135.

[9] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, C. Finn, Gradient Surgery for Multi-task Learning, Advances in Neural Information Processing Systems, Vol. 33, 2020, pp. 5824–5836.

[10] D. Calandriello, A. Lazaric, M. Restelli, Sparse Multi-task Reinforcement Learning, Advances in Neural Information Processing Systems, Vol. 27, (2014).

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level Control through Deep Reinforcement Learning, Nature, Vol. 518, No. 7540, 2015, pp. 529–533. doi:10.1038/nature14236.

[12] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft Actor-critic: Off-policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, in: International Conference on Machine Learning, PMLR, 2018, pp. 1861–1870.

[13] T.-L. Vuong, D.-V. Nguyen, T.-L. Nguyen, C.-M. Bui, H.-D. Kieu, V.-C. Ta, Q.-L. Tran, T.-H. Le, Sharing Experience in Multitask Reinforcement Learning, in: Proceedings of the 28th International Joint Conference on Artificial Intelligence, 2019, pp. 3642–3648.

[14] P. Huang, M. Xu, J. Zhu, L. Shi, F. Fang, D. Zhao, Curriculum Reinforcement Learning Using Optimal Transport via Gradual Domain Adaptation, Advances in Neural Information Processing Systems, Vol. 35, 2022, pp. 10656–10670.

[15] Y. Luo, Z. Jiang, S. Cohen, E. Grefenstette, M. P. Deisenroth, Optimal Transport for Offline Imitation Learning, ArXiv Preprint arXiv:2303.13971 (2023).

[16] A. Asadulaev, R. Korst, A. Korotin, V. Egiazarian, A. Filchenkov, E. Burnaev, Rethinking Optimal Transport in Offline Reinforcement Learning, Advances in Neural Information Processing Systems, Vol. 37, 2024, pp. 123592–123607.

[17] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, R. Hadsell, Policy Distillation, ArXiv Preprint arXiv:1511.06295 (2015).

[18] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, R. Pascanu, Distral: Robust Multitask Reinforcement Learning, Advances in Neural Information Processing Systems, Vol. 30, (2017).

[19] J. He, K. Li, Y. Zang, H. Fu, Q. Fu, J. Xing, J. Cheng, Efficient Multi-task Reinforcement Learning with Cross-Task Policy Guidance, Advances in Neural Information Processing Systems, Vol. 37, 2024, pp. 117997–118024.

[20] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, S. Levine, Meta-world: A Benchmark and Evaluation for Multi-task and Meta Reinforcement Learning, in: Conference on Robot Learning, PMLR, 2020, pp. 1094–1100.

[21] H. Ma, Z. Luo, T. V. Vo, K. Sima, T.-Y. Leong, Knowledge Sharing and Transfer via Centralized Reward Agent for Multi-Task Reinforcement Learning, ArXiv Preprint arXiv:2408.10858 (2024).

[22] A. Tirinzoni, A. Sessa, M. Pirotta, M. Restelli, Importance Weighted Transfer of Samples in Reinforcement Learning, in: International Conference on Machine Learning, PMLR, 2018, pp. 4936–4945.

[23] A. Xie, C. Finn, Lifelong Robotic Reinforcement Learning by Retaining Experiences, in: Conference on Lifelong Learning Agents, PMLR, 2022, pp. 838–855.

[24] Y. Wu, G. Tucker, O. Nachum, Behavior Regularized Offline Reinforcement Learning, ArXiv Preprint arXiv:1911.11361 (2019).

[25] B. Eysenbach, A. Gupta, J. Ibarz, S. Levine, Diversity Is All You Need: Learning Skills without a Reward Function, ArXiv Preprint arXiv:1802.06070 (2018).

[26] M. Cuturi, Sinkhorn Distances: Lightspeed Computation of Optimal Transport, Advances in Neural Information Processing Systems, Vol. 26, (2013).

[27] R. Dadashi, L. Hussenot, M. Geist, O. Pietquin, Primal Wasserstein Imitation Learning, ArXiv Preprint arXiv:2006.04678 (2020).

[28] G. Papagiannis, Y. Li, Imitation Learning with Sinkhorn Distances, ArXiv Preprint arXiv:2008.09167 (2020).

[29] S. He, Y. Jiang, H. Zhang, J. Shao, X. Ji, Wasserstein Unsupervised Reinforcement Learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 36, 2022, pp. 6884–6892. doi:10.1609/aaai.v36i6.20645.

[30] B. G. Le, V. C. Ta, Distill Knowledge in Multi-task Reinforcement Learning with Optimal-Transport Regularization, in: 2022 14th International Conference on Knowledge and Systems Engineering (KSE), IEEE, 2022, pp. 1–6. doi:10.1109/KSE56063.2022.9953750.

[31] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, Others, Soft Actor-critic Algorithms and Applications, ArXiv Preprint arXiv:1812.05905 (2018).

[32] P. Christodoulou, Soft Actor-critic for Discrete Action Settings, ArXiv Preprint arXiv:1910.07207 (2019).