VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Le Bang Giang - 18020428

Student Thesis

SHARING EXPERIENCE MULTI-TASK REINFORCEMENT LEARNING VIA OPTIMAL TRANSPORT

HA NOI - 2022

VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Student Thesis

SHARING EXPERIENCE MULTI-TASK REINFORCEMENT LEARNING VIA OPTIMAL TRANSPORT

Supervisor: Dr. Ta Viet Cuong

HA NOI - 2022

ABSTRACT

Contents

List of figures

List of tables

CHAPTER 1. INTRODUCTION

1.1 Sharing samples reinforcement learning

1.2 Contributions

CHAPTER 2. RELATED WORK

2.1 Background

2.1.1 Reinforcement learning (RL) and deep reinforcement learning

Reinforcement learning (RL) problem is a decision making problem in which an agent learns to make optimal decisions through interactions with an environment. The environment in reinforcement learning problems is usually modeled as a black box that emits new states as well as reward signals upon receiving actions. This section starts by introducing the mathematical framework to model any reinforcement learning problems: Markov Decision Process.

A Markov Decision Process (MDP) is (usually) defined as $M := (S, A, p, P_0, r, \gamma)$, where S is the state space, A is the action space, p is the dynamic transition, or transition kernel, which dictates the distribution of the next state conditioned on current state and action taken, P_0 is the initial state distribution , r is the reward function and γ is discount factor. The standard objective of RL is to find a policy $\pi_\theta(a|s)$ parameterized by θ that maximize the cumulative reward $J(\pi) = \mathbb{E}_\pi \sum_{t=0}^{T} \gamma^t r_t$.

The purpose of Reinforcement learning is to find the best course of actions or strategies that achieve the maximum rewards collected throughout the entire trajectories of the agent in the environment. This is sometimes nontrivial as the agent needs to uncover the entanglement of the reward collected in the future and current actions: it needs to learn how to reach high reward regions that bring abouts the long-term benefit, even though that means it might sacrifice short term immediate high reward actions. Since both the policy and environment are stochastic, RL algorithms attempt to find the best policies that attain high cumulative reward in expectation.

Reinforcement learning makes use of function approximation for their learning representation, traditional RL does not assume any postulation about the functions used, they can be a perfect approximator function where data is saved in a table or they can belong to a hypothesis class of function. Deep reinforcement learning is when neural networks are used as function approximators and deep learning techniques are incorporated in RL. Deep RL can effectively solve complex tasks with high dimensional input data, e.g images of observation of the environment or sensor stream from robots.

There are many Reinforcement learning algorithms that have been proposed in the last several decades. They are all diverse in approach and methodology that makes it hard

to draw an accurate, all-encompassing taxonomy of RL algorithms as different perspectives of RL algorithms emphasize on some specific dimensionality of the algorithms design and blur out other irrelevant aspects. Some of the most important criteria to consider in the classification of RL algorithms are model-based and model-free methods, value-based and policy-based methods, Monte Carlo methods and temporal-difference methods, on-policy and off-policy methods, and so on. These criteria are not necessarily exclusive as one RL algorithm can fall under multiple categories, and considering only some of the criteria will never be sufficient since they are not enough to describe all characteristics of RL algorithms. In the following sections, we discuss some of the perspectives that we think are relevant in our work.

2.1.2 On-policy and off-policy algorithms

There are various strategies to approach a reinforcement learning problem, but they can be generally subsumed into two big categories: model-based and model-free RL. In model-based RL, agents first learn the dynamics of the environment, i.e how the world works, and then plan on the learned knowledge. The planning step happens after thorough exploration and interactions with the environment, the performance of model-based RL relies heavily on how well it models the environment internally. On the other hand, with model free RL, agents directly learn a policy, and improve it by trying to act according to that policy in the environment, the learning process interleaves with the interactions phase. This section focuses on the model-free side of RL algorithms.

Many model-free RL algorithms pivot around the core idea of how to estimate how good an action is in a given state. This "goodness" of a state-action pair is represented as a scalar value $Q(s, a)$ denoted by the expected total returns starting from this state-action pair and following the given policy onward, and the higher this value is, the better the action in that state. Usually, estimating this Q function is challenging as the actions taken in the moment can affect the state that the agent ends up in later, and some actions can have more influence than others. Moreover, this influence might be obscure and not certain as the effect can be delayed to the far future. Based on how agents estimate the quality of state-action pairs, model-free RL algorithms can be further categorized into two different lines of work: on-policy and off-policy algorithms.

Off-Policy Reinforcement Learning (RL) techniques improve a target policy by using transitions collected by a behavioral policy that is not the same as the target policy. This differs from on-policy RL, in which a policy π is updated via data collected by itself. Generally speaking, many current state-of-the-art on-policy RL approaches, such as REINFORCE, PPO [20] and TRPO [19], usually directly estimates the quality of state-action pairs by using Monte Carlo method: by trying out playing the current policy in the environment for a number of rollouts, averaging the total rewards received and taking this empirical mean as estimation for $J(\pi)$. Some other on-policy algorithms apply the Bellman update rule with on-policy data, forming an online version of the Bellman equation; examples of this approach are SARSA and A3C [14]. While in off-policy RL, agents generally use a method called bootstrap to evaluate the value of actions: it employs

the recurrent formula of the Q function (the Bellman equation) to cut-off the lengthy summation of rewards from all time steps as in on-policy formula.

Off-policy algorithms are usually better than on-policy algorithms in terms of sample efficiency as they can utilize transitions collected from the past, while on-policy algorithms require fresh samples every policy update step. However, on-policy algorithms can in general achieve better performance than off-policy algorithms, as their value estimations are based on Monte Carlo method and therefore they are unbiased, while the value estimations of off-policy algorithms are heavily depend on the function approximator to find solutions for Bellman equation, which is not guaranteed to converge in theory, thus with large sample size, on-policy methods can perform better than other RL methods.

On-policy RL algorithms

Generally speaking, on-policy RL algorithms are any algorithms that require fresh samples from the environment on every update step of the algorithm. It is difficult to draw a general framework of on-policy algorithms aside from the fact that they use online data as opposed to other approaches as they can be very different. The first classical on-policy algorithm that has to be mentioned is REINFORCE, this algorithm belongs to the gradient policy methods that directly optimize the cumulative reward by gradient ascend with respect to policy parameters. Using the result from *Policy Gradient Theorem* [21], the gradient of the RL objective $J(\pi_\theta)$ has the form:

$$\nabla J(\pi_\theta) \propto \mathbb{E}_\pi[Q^\pi(s,a)\nabla_\theta \log \pi_\theta(a|s)] \tag{2.1}$$

The expected return term $Q^\pi(s,a)$ is estimated by the empirical mean of discounted rewards collected from complete episodes or rollouts. The gradient estimate of REINFORCE is an unbiased but high-variance estimator, so with large enough samples size, this algorithm can perform reasonably well. Other variants are inspired from REINFORCE to reduce this high variance by replacing the total discounted episode reward with baselined version of total trajectory reward, Q function, reward-to-go, advantage function and temporal difference residual. TRPO [19] introduces the idea of trusted region optimization to the policy gradient step by restricting the policy update to be small: the updated policy stays close to the old policy in terms of KL divergence.

$$\begin{aligned} \text{maximize} \quad & J(\pi_\theta) = \mathbb{E}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} Q_{\pi_{\text{old}}}(s,a)\right] \\ \text{subject to} \quad & \mathbb{E}[D_{KL}(\pi_{\theta_{\text{old}}}(\cdot|s) \| \pi_\theta(\cdot|s))] \leq \delta \end{aligned}$$

As the computation of the KL constraint is relatively complicated, PPO [20] introduced the clipped surrogate objective, which simplifies the procedure while retaining similar performance of the algorithm. The new objective function is essentially the original objective, aided with a clipping threshold $[1-\epsilon, 1+\epsilon]$ of the importance weights. The use of ratio clipping acts as a "lock" to eliminate the gradient from the objective function when importance weights of the two policies get outside of a certain range based on the hyperparameter ϵ, thus preventing the new policy to stray further away from the original

policy.

$$J^{\text{CLIP}}(\theta) = \mathbb{E}[\min(r(\theta)\hat{A}_{\text{old}}, \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\text{old}})]$$

Where $\hat{A}$ is an estimator of the advantage function, and the hyperparameter ϵ controls how conservative each policy update is. Both REINFORCE, PPO and TRPO use a similar method to estimate the cumulative reward/Q function/advantage function: they wait until the end of each episode, sum up all the reward collected along the trajectories and use this as a rough approximation of the true expected returns. This approach, which is called Monte Carlo method, has a disadvantage that each policy update step needs to happen at every integral number of rollouts.

Other on-policy RL algorithms use a rather different approach to the previously mentioned classical algorithms. SARSA applies a value-based method to learn the Q function in a similar way as TD(0) and Q learning, but instead of using off-line data as in Q learning, it uses a single online sample for each function update. The update rule of SARSA is motivated from the recursive form of the Q function as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

This update function is usually called *bootstrapping* or *one-step learning* as the update step moves the estimated function toward the true Q function by one-step looking ahead of the next state-action returns. In this case, we are assuming that the estimated Q value of the next state is a good proxy to represent all rewards coming after the time step $t+1$. Since it only consider one step ahead, the acquisition of reward r_t at some time step t only has a direct impact on the Q value of state action pair at that time step and has little effect on the value of other state action, as the rewards would need many more update steps to propagate to the relevant preceding states and actions, which can be a slow and inefficient process. A3C [14] considers *n-steps returns* for faster convergence of the Q function as well as uses multiple parallel actor-learners to boost the exploration of different parts of the environment. A3C also specifically makes use of neural networks as their function approximation and employs gradient ascent on the regression problem of learning Q function as opposed to the tabular update rule as in SARSA. The target Q learning in A3C is then

$$\sum_{i=0}^{n-1} \gamma^i r_{i+t} + \gamma^n V(s_{t+n})$$

Using this as the approximation of the Q function, A3C proceeds to update the policy as other policy gradient methods.

Off-policy RL algorithms

We define the action state value of a policy π as $Q_\pi(s, a) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t | s, a]$, the expected return when following the policy π after taking action a at state s. Given a policy, the Q function can also be expressed in a recursive form:

$$Q_\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1}}[r_t + \gamma \, \mathbb{E}_\pi \, Q(s_{t+1}, a_{t+1})] \tag{2.2}$$

Equation 2.2 is known as the *Bellman equation* for Q_π. Bellman equation describes the connection between a state's value and the values of its succeeding states. Any function that satisfy the equation 2.2 for all state-action pairs of some policy is the unique Q function to that policy. By using Bellman equation, one can approximate the value of a state-action pair as long as the estimate of the value of its successor states are reasonable. The operation that updates the value of current state-action pairs based on 2.2 is called the *Bellman operator*. The Bellman operator is a contraction for $\gamma < 1$ with a unique fixed point Q_π, which means that by starting from an initial guess of the Q function and repeatedly applying the Bellman operator, it is guaranteed that the function converges to the true Q function at infinity.

A special case of the Bellman equation is the Bellman equation of the optimal policy, called the *Bellman optimality equation*. A policy is defined to be better than another policy if its value function is greater than or equal to that of the other policy for all states. The optimal policy is the policy that is better than any other policies. There is at least one such optimal policy for any MDP. The Bellman optimality equation has a property that the values of any state under the optimal policy is equal to the expected returns of best action in the next states.

$$Q^*(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1}}[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})]$$

One we have the optimal Q function, it is relatively trivial to derive the optimal policies: Given the optimal Q* function, any policies that assign non zero probability to only the actions that maximize the Q* are optimal policies. In other words, the optimal policies are greedy policies with respect to the optimal Q* function. Therefore, many methods aim at finding the optimal Q function instead of directly solving for the optimal policy, since the policy can be recovered by simply finding the argmax of the Q function at the current state. This gives rise to value-based methods, whose main motivation is how to estimate the optimal value function.

The Bellman equation inspires ideas for many of the modern off-policy algorithms, off policy RL today are basically using some variants of the Bellman equation or optimal bellman: Traditional Q learning and DQN straightforwardly use the Bellman optimality equation to learn Q function in the discrete spaces. In the continuous domain, however, the Bellman equation cannot be applied easily as finding the argmax of a highly non-convex and continuous value function is nearly impossible. Many works attempted to develop methods to approximately solve this problem: DDPG is a direct extension of Q learning to the continuous action domain, with the aid of another high-capacity neural network to estimate the maximum of value functions, TD3 improves the stability of DDPG by using various strategies such as twin-delay Q function and clipping Q tricks, Soft-Q learning replace the max operator in the Bellman optimality equation with Soft-max and optimize the entropy RL objective. SAC extends the Soft-Q learning to the settings of the Actor-critic algorithm to update the new entropy regularized objective RL.

2.2 Multi-task RL and transfer in RL

2.3 Optimal transport as a distance metric in RL

Optimal transport (OT) has seen growing attention over the last few years in the machine learning community and researchers. Its applications are ubiquitous and can be found in various fields such as computer vision, image retrieval, domain adaptation, natural language processing, generative modeling and many more. Recently, OT has also found its way to reinforcement learning literature and is repeatedly proven to be effective in various conditions. The potency of OT lies in its capability to measure the discrepancy between probability distributions, taking into account the geometric structures of the underlying problems. In the later chapters, we will begin by giving a brief introduction to OT in the context of distance metrics. We then review some of the most prominent OT metrics used in machine learning, namely Wasserstein distance, entropy regularized Wasserstein distance and Sliced Wasserstein distance. Since OT is a computationally expensive problem, we will also give some overviews on the computational aspects of OT, including algorithms and tricks that are used to overcome the challenges of computing OT in practice. Finally, we present experiments to illustrate the behaviour of different OT metrics in reinforcement learning and apply it to the setting of sharing experience in RL.

Optimal transport is essentially a metric distance, a metric distance is basically a means by which we measure how different two things are. The idea of measuring distance between policies, implicitly or explicitly, has been frequently picked up in reinforcement learning. For probability distance, before optimal transport, multiple distance metrics have been investigated in reinforcement learning, most of the time they are information based metrics: mutual information, Kullback Leibler divergence, total variation divergence and Jensen Shanon divergence are the names that frequently pop up in the field. One of the advantages of these information distances is that they have well-developed theories that can bring about great theoretical insight and in many cases, information distances are applied first before any other kind of distances being explored. In this section, we discuss two research directions in RL that have succeeded in applying Optimal transport to improve RL performance: unsupervised learning RL and Imitation learning.

2.3.1 OT in Unsupervised RL

Traditional reinforcement learning algorithms aim at solving the RL problems by finding an optimal policy that maximizes the expected total rewards of a given reward function of an MDP. This reward function is very crucial in RL as it somehow represents the purposes of the agent in that environment. In other words, the objective of the agent is formalized mathematically by reward signals that the agent receives upon interacting with the environment, which is simply a scalar at each time step. This idea of reward maximization is a distinctive feature of reinforcement learning: the *reward hypothesis*,

according to Sutton and Barto [21], states that: "*all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward)*". As a result, the behavior of the agent is deeply connected to the reward signals, and the reward function has to be reflective so that it truly indicates what we intend the agent to achieve. This illustrates how important the role reward signals play in reinforcement learning, as they are the tool for us to communicate to the RL agent what you want it to achieve, and it also suggests that designing a good reward function is a challenging task since a reward function that can elicit desired behaviors from agent requires great deal of engineering. A simple example of faulty reward functions is described in [4].

Traditional reinforcement learning algorithms like PPO [20], TRPO [19] and REINFORCE work best when useful reward signals from environments are abundant. However, In many cases, the reward signals are sparse and only visible to the agent at the end of each episode, one example of this is in Chess or Go, reward signal is given to agent only when the game finishes, which might be hard to infer which moves are bad or which moves actually contribute to the final winnings. In the most extreme cases, the reward signal is so sparse that the agent could not even find good reward states and thus has nothing to learn from. Thus, the sparsity of reward signals is also a factor that can greatly hinder the performance of RL algorithms.

A natural question that might arise: Can an agent learn without reward feedback? Sure enough, intelligent creatures can learn about their environments without supervision to acquire basic knowledge so that it can be adapted and applied to solve new upcoming tasks later. Unsupervised learning RL addresses the problem of reinforcement learning in the absence of reward function. The general idea of unsupervised RL is as follows: they learn a set of useful skills, or set of policies, that can serve later as primitives for other downstream tasks. This set of policies are expected to be as diverse as possible and collectively explore a large portion of the state spaces. The usefulness of a skill or policy is hard to define, as we do not have any prior knowledge about the reward function of the tasks at hand. According to [8], a skill learned by Unsupervised RL is "*a latent-conditioned policy that alters that state of the environment in a consistent way*". Unsupervised RL has a number of practical applications: by learning a good set of skills, the agent can acquire a repertoire of tools that can be used for other complex tasks for a better parameter initialization, primary skills in hierarchical RL or in environments where evaluating the reward signals is expensive.

DIAYN (Diversity is All You Need) [8] approaches the problem of unsupervised RL by an assumption that it is sufficient for the agent to learn a diverse set of skills that collectively cover large parts of the state space, arguing that complex behaviors can arise by directly maximizing diversity. The intuition of DIAYN is that while some skills might act randomly and are of little use, other skills should perform differently from the random skills, they can reach state regions that other skills cannot reach and therefore might be more useful.

The framework that DIAYN uses to define the diversity is mutual information. More

specifically, DIAYN's approach is based on three ideas: first, the skills should dictate the states the agents visit, and different skills should visit different states, for that reason, state distribution of a skill is the hallmark of that skill and should be used to distinguish it from other skills, not the actions the skill take because actions that do not have explicit effect on the states of the environment is not visible to an outside observer, and this is the second idea. Finally, skills should act as randomly as possible, as randomness of policy has been proved by many previous works that it improves the exploration of the agent. Summarizing all of these ideas, the objective of DIAYN can be expressed in three terms, each equivalent to one of the aforementioned ideas.

$$\begin{aligned} F(\theta) &\triangleq I(S, Z) + H[A|S] - I(A, Z|S) \\ &= H[Z] - H[Z|S] + H[A|S, Z] \end{aligned}$$

In the above formula, Z is the latent variable of skills, the mutual information term $I(S; Z)$ reflects the idea that the latent-skill random variable Z of a skill should encode information of the states visited by that skill, which corresponds to the first idea. The entropy term defines the randomness of policy and represents the third idea. While the negative mutual information term $-I(A; Z|S)$ indicates that it is the states and not actions that bring useful information to differentiate skills. DIYAN implements this optimization problem by maximizing a lower bound of the above objective function, with the help of a learned discriminator $q_\phi(z|s)$.

$$F(\theta) \geq H[A|S, Z] + \mathbb{E}_{z,s}[\log q_\phi(z|s) - \log p(z)]$$

With p is the probability density function of the latent variable Z and $q_\phi(z|s)$ is approximated by a neural network. It has been proved through experiments that skills learned by mutual information maximization achieve a wide range of total reward, and some skills actually achieve high performance as if they are learned with reward feedback. This indicates that diversity of the skills indeed helps explore meaningful behaviors.

The effectiveness of the mutual information maximization has been empirically demonstrated, which suggests that diversity might be the right direction to approach unsupervised reinforcement learning. But does mutual information objective actually help discover useful skills? Or more concretely, how do we define the usefulness of a policy? And if so, are the learned skills optimal? [9] helps address these questions by showing that the skills learned by mutual information are optimal for some reward functions, and that the distribution over skills learned by mutual information induces marginal state distribution that is optimal under an adversarially-chosen reward function. Analysis from [9] also provides a geometric points of view on skills learning RL problems.

According to [9], the state distribution of the skill distribution is optimal with respect to a information regularized regret objective under the worst case reward functions. This can be succinctly stated in the followings equation:

$$\min_{\rho(s) \in C} \max_{r(s)} \text{AdaptationObjective}(\rho(s), r(s)) = \max_{p(z)} I(s; z)$$

With the adaptation objective is defined as

$$\text{AdaptationObjective}(\rho(s), r(s)) \triangleq \min_{\rho^*(s)\in C} \max_{\rho^+(s)\in C} \mathbb{E}_{\rho^+(s)}[r(s)] - \mathbb{E}_{\rho^*(s)}[r(s)] + D_{KL}(\rho^*(s)\|\rho(s))$$

Where C is the set of all feasible state marginal distributions of the environment, $r(s)$ is the reward function of new unknown tasks and $\max_{p(z)} I(s, z)$ is the maximization problem of mutual information. The *AdaptationObjective* can be understood as an information regularized regret objective, with $\max_{\rho^+(s)\in C} \mathbb{E}_{\rho^+(s)}[r(s)]$ is the optimal policy under $r(s)$. The term $\max_{\rho^+(s)\in C} \mathbb{E}_{\rho^+(s)}[r(s)] - \mathbb{E}_{\rho^*(s)}[r(s)]$ is the *regret* of the policy that induces the state distribution $\rho^*(s)$, regret is the gap of the expected return between the current policy and the optimal policy, it is the difference in performance of a policy compared with the best policy in hindsight. An information regularization term $D_{KL}(\rho^*(s)\|\rho(s))$ control how far the new state distribution $\rho^*(s)$ is allowed to some given state distribution $\rho(s)$, this restriction in the state distribution space indirectly limits the set of new possible policies in the policy space. One can intuit the Adaptation Objective as the problem that: given a prior state distribution $\rho(s)$ and the reward function of a task $r(s)$, find a new state distribution $\rho^*(s)$ that minimizes the regret with respect to the optimal state distribution $\rho^+(s)$ of the reward $r(s)$, with a constraint that this new state distribution must stay close enough to the prior state distribution $\rho(s)$. The constraint ensures that the search of new distributions is conservative: it prevents new policy from overfitting to a limited number of samples of a (noisy) reward function or it can be thought of as the cost to adapt the old state distribution $\rho(s)$ to a new better state distribution $\rho^*(s)$, given that new information of the task is revealed to us in the form of reward function $r(s)$. Thus the Adaptation Objective is a fine tuning problem (or adaptation problem), it happens after the unsupervised stage, when a downstream task is assigned to the agent. The optimization that occurs outside of the Adaptation Objective is where our unsupervised learning stage lies: it can be interpreted as finding an optimal prior state distribution $\rho(s)$ that minimize the cost of fine tuning to a new state distribution under the worst-case reward function $\max_{r(s)} \text{AdaptationObjective}(\rho(s), r(s))$. The prior state distribution learned by mutual information is thus optimal in a sense that it provides the best policy initialization that is optimal to adapt to downstream tasks.

Additionally, [9] also dissects the skill learning problem from a geometric point of view, which can be summarized as follows: By showing that the set of all possible state distribution of a policy is a convex set, where each of the element in this set is represented as the point on a plane in $R^{|S|}$, and a reward function of any task can be represented as a vector that, in order to compute the expected return of a policy with that reward, we project the state distribution of that policy onto the reward vector by calculating a dot product. And since the set of all feasible state distributions is convex, it is necessary that optimal policy must be the one that lies at a vertex of the state marginal polytope. From experiments, [9] shows that mutual information maximization recovers some of the policies at these vertices, which means that they are optimal for some state-dependent reward functions.

Conventional approaches in Unsupervised learning RL focus on using mutual information maximization between skill and state distributions: latent skill variable carries

important information to infer what states the agent visits. WURL (Wasserstein unsupervised reinforcement learning) [12] consider using Optimal transport distances in place of mutual information, arguing that Optimal transport metrics are more effective in capturing the discrepancy between policies. The motivation of WURL is to incentivise learned policies to discover a more diverse and broad set of state distributions with geometric-aware distances. Using OT distances, the pseudo rewards derived from the distance between policies are used to train Unsupervised Rl agents. To learn multiple skills, WURL proposes to use the minimum OT distances of policies, meaning that a skill should keep distance as far as possible to its nearest neighbor skill. Amortized reward is used to combat the reward sparsity of the problem, the amortized reward of each state is calculated as the contribution of that state to the overall distance, as more novel states are weighted more to encourage agents to explore. Experiments of WURL show that skills learned by optimal transport are evenly distributed across a wide region of state space, and they generally perform better than that of mutual information approach.

2.3.2 OT in Imitation learning RL

CHAPTER 3. SHARING EXPERIENCE VIA OPTIMAL TRANSPORT DISTANCE

3.1 Introduction

Give some overview

3.2 Estimate Policy Distance in Multi-task Reinforcement Learning from State Distribution

In this section, we provide the method to calculate the Optimal transport distances between policies, given the exact state distributions. We explore three different Optimal transport metrics that are widely used and discuss their characteristics of each distance. Since Optimal transport distance is intractable to compute in its exact form, we also review the methods to approximate them in practice, we also provide the pseudocode of the methods that we tried out in our experiments.

The general motivation of optimal transport can be derived from the question of how to move a pile of earth (some natural resource) to a target location with the least amount of effort? More specifically, suppose there are several piles of dirt spread over the ground and there are some holes that need to be filled, one wishes to find an optimal transport plan that requires a minimum amount of work to move piles of earth to holes, resulting in a state where all piles and holes disappear. The problem is thus sometimes called Earth mover's distance [17] and can be solved by linear programming. The cost, or work, is usually defined to be the amount of weight that needs to be moved times the distance by which it is moved.

The first metric that we examine in this section is the general formula of the primal form of Optimal Transport, called the Wasserstein Distance (WD) in mathematics. The intuition of the Wasserstein metric closely relates to the above intuition of Optimal transport. As we will see, solving the exact solution of the primal Wasserstein distance has a number of disadvantages: it requires the probability spaces to live in the discrete domain and an expensive linear optimization problem must be solved, which might be prohibitive in large problems with thousands of variables. The dual form of Wasserstein distance provides a way to circumvent these challenges with the aid of an auxiliary function, often

called testing function or probing function; this function is generally implemented in deep learning as a neural network. Another approach in optimal transport that is inspired from the simplicity of the Wasserstein distance in 1-dimensional space; the slicing approach, which calculates the expectation of the WD over a distribution on a projected sphere. The final approach that we discuss is an entropic regularization problem of the primal WD, which turns the original problem into a strict convex optimization problem and can be approximated by an iterative process.

In all of the following sections that we will discuss about optimal transport metrics, we consider the general methods for calculating the distances between *any* probability distributions on a metric space and not only specifically for policy distances. For that reason, we can easily extend the idea to apply to measure policy distances. Since a policy is the action distributions conditioned on states, given an MDP, one can obtain either marginal state distribution or state-action distribution. Both options have been proved to be good to represent distances between policies by a number of prior works: [12] successfully use state distributions in Unsupervised RL, borrowing idea from [8] that whatever actions which do not have explicit influences in the form of changes on the state distribution are not important, [16] experimented using state-action distribution in calculating the difference between expert policy and learner policy with Sinkhorn, or in [6] examined both state and state-action pair distribution in Imitation learning, which subsequently leads to different learning problem settings of vanilla Imitation learning and Learning from Observation (LfO).

3.2.1 Wasserstein distance

Wasserstein distance (WD) or Kantorovich–Rubinstein metric is a distance function defined between probability distributions on a given metric space. Let p and q be two probability distributions on domains $X, Y \subseteq R^n$ and $\Pi(p, q)$ be the set of all distributions on product space $X \times Y$ whose marginal distributions on X and Y being p, q respectively. Therefore, given a proper distance function $d(x, y) : X \times Y \to R$ for moving mass from x to y, the Wasserstein distance is defined as

$$W_m(p, q) = \left(\inf_{\pi \in \Pi(p,q)} \int_{X \times Y} d(x, y)^m \mathrm{d}\pi \right)^{1/m} \tag{3.3}$$

In a sense, the Wasserstein distance is closely related to the Optimal transport problem. Informally, each distribution can be thought of as piles of earth and holes whose total mass equals one, $\Pi(p, q)$ is then the set of all possible ways to move all piles into holes, the marginal constraints on this joint distribution set Π enforces the ideas that 1) the total amount of earth move out of a pile must be equal to the initial amount that was there to begin with and 2) the total amount of earth go into a hole must be equal to the capacity of the hole at the beginning. The Wasserstein distance is then the minimum transportation cost needed to transform one distribution to the other such that at the end, both the piles of earth and the holes in the ground completely vanish. If the cost of a move is simply the distance between the two points, then the optimal cost is identical to

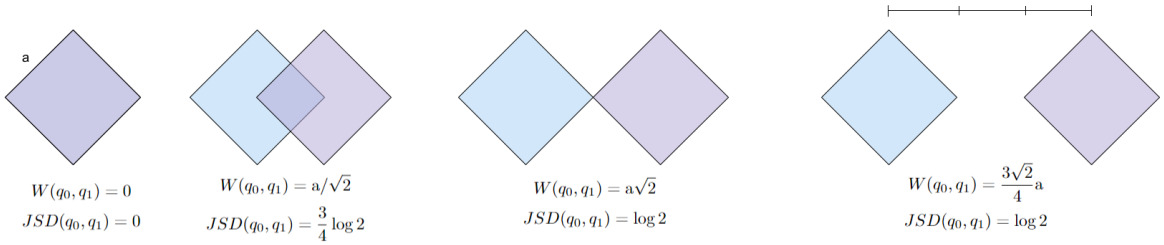


Figure 3.1: Examples of Jensen-Shannon divergence and Wasserstein distance between q_0 and q_1. Jensen divergence is always bounded at log(2) while Wassertein distance increases linearly as the two distributions move away from each other.

the definition of the W_1 distance. Normally, we consider the 1-Wasserstein, also called the Earth mover's distance, since other higher order distances can be handled by absorbing the power term into the distance function.

One possibly interesting characteristic of Wasserstein distance is that it takes into consideration the geometric topology of the problem, which is embedded in the distance function term d. To better illustrate this idea, consider a simple example of two random distributions q_0 and q_1 on a 2-D Cartesian plane, each distribution is a rhombic uniform distribution, i.e uniform distribution on some rhombus of size a for all edges. The KL divergence between q_0 and q_1 equals zero if they are identical but jumps to infinity as one distribution expands beyond the supported region of the other distribution. The Jensen Shanon divergence ameliorates the infinity nuisance of KL by taking the average of the KL distance between the two distributions with an intermediate mixture distribution. More specifically, the Jensen Shanon divergence between q_0 and q_1 equals the mutual information between a mixture distribution X between q_0 and q_1, $X = (q_0 + q_1)/2$, and a binary indicator variable Z to indicate which distribution in q_0 and p_1 produces the mixture. Thus $I(Z, X) = H(Z) - H(Z|X)$ is upper bounded by $H(Z)$, which in this case is the maximum entropy of a Bernoulli random variable. The conditional entropy term $H(Z|X)$ reaches its minimum when X is produced by two disjoint distributions, i.e q_0 and q_1 do not overlap, in which case observing X would give all the information needed to infer the value of Z. The conditional variable Z thus has no uncertainty and therefore its entropy is zero. For that reason, no matter how far away the two distributions q_0 and q_1 are, their Jensen divergence is always bounded at log(2) at max. On the other hand, the 1-Wasserstein distance in case of a simple affine operation that translates the entire distribution scales linearly with the translated distance, which in some situations might give a more faithful description of the discrepancy between distributions.

To calculate WD, we consider the discrete state spaces, the Wasserstein distance (Earth mover's distance) between two discrete distributions is the minimum cost to transport weight (dirt) from one distribution to the other. The Wasserstein distance between two distributions $u \in R^n$ and $v \in R^m$ in this case is $\min_P \sum_{ij} P_{ij} C_{ij}$, where $C \geq 0$ is the $n \times m$ cost matrix of transporting a single unit of weight from i in distribution u to j in distribution v (which corresponds to the distance function in equation 3.3) and $P \geq 0$ is the $n \times m$ transportation plan matrix denoting how much weight is transported from i to

j. Format the problem as a linear progamming problem:

$$\begin{array}{lllll} \text{minimize} & W = \sum_{ij} P_{ij} C_{ij} & & & \\ \text{subject to} & \sum_j P_{ij} = u_i & i = 1, \ldots, n & \text{constraints on marginal distribution } u \\ & \sum_i P_{ij} = v_j, & j = 1, \ldots, m & \text{constraints on marginal distribution } v \\ & 0 \leq P_{ij} \leq 1, & \forall i, j & \end{array}$$

This linear problem can be solved by any LP solver, the resulting optimization gives us the exact solution to the WD between u and v, a byproduct of this process is the optimal transportation mapping matrix, which can be then combined with the cost matrix to find how much each samples in each distributions contributes to the distance. The constraints on rows and columns of the matrix P is to ensure that the overall weights to be transported is preserved: total amount of weight before and after the transportation should be the same, other variants of the distance can take into consideration distributions with different total masses.

Unfortunately, directly solving the optimal transport problem from the primal formulation is hard. Indeed, no matter what the algorithms used, network simplex or interior points, the time complexity is scaled at least in $O(d^3 \log(d))$ (supercubic complexity [5]) where d is the number of bins in the histograms of two distributions. Therefore, computing Wasserstein distance suffers from expensive computational complexity, which hinders the applicability of OT in practice, especially in machine learning where large-scale data are abundant and prevalent. There are a number of works that seek to overcome these computation challenges, among which there are two main notable approaches [15]: the first one tries to approximate the value of OT by some means. Examples of this are approximating OT by optimizing a dual problem of the primal OT formula, e.g Kantorovich-Rubinstein duality, or applying entropic constraints to speed up the computation, e.g Sinkhorn distance. The second approach is to project the original distributions onto various 1-dimensional subspaces along random directions which are drawn from some distribution on the unit sphere and compute many 1-D OT subproblems. A popular example in this direction is Sliced Wasserstein distance [13], in which the random projection directions are sampled from a uniform distribution over a unit sphere, and the result is the expectation of 1-dimensional OT distances over all directions. Both approaches have been successfully applied to a variety of tasks and proven to be comparable in performance with respect to other distances and divergences between probability distributions. We will discuss more about Sinkhorn distance and Sliced Wasserstein distance in later sections.

One final note is that in some particular cases when the interested distributions are in low dimensional spaces, computing the Wasserstein distance is reasonably cheap. In 1 dimensional space for example, the 1-Wasserstein distance has a closed-form expression[1]:

$$W_1(p, q) = \int_{-\infty}^{+\infty} |P - Q|$$

Where P and Q are the respective CDF of p and q. Therefore, computing the Wasserstein distance in 1 dimensional spaces only requires $O(d \log d)$ operations. This formula might come in handy in computing Sliced Wasserstein distance since SWD specifically computes many 1D OT subproblems in the projected spaces.

Some researches have focused on approaches to mitigate the computation aspect of the Wassterstein distance. For example, [6] proposed an alternative to compute the upper bound of WD through a method called greedy coupling; the pseudocode of this method is included in Algorithm 1 for completeness. This algorithm is well-suited to the settings where we have access to an online stream of data and the coupling decision needs to be decided immediately. In this method, online samples that come first have higher priority to be coupled with nearest data points from the other distribution: the knowledge of samples arrives sequentially, and the transportation plan for that sample is determined greedily right when it appears - to the closest remaining available holes. This coupling strategy however is suboptimal as it can be largely off the true distance.

Algorithm 1 Greedy coupling WD

Input: u, v: probability distributions, D, F: Sample sets, m, n: length of D, F
d: distance function
$c \leftarrow 0$
for i=1..m **do**
  while $u_i > 0$ **do**
    $j \leftarrow \arg\min_{j=1..n}(d(F_j, D_i))$
    $w \leftarrow \min(v_j, u_i)$
    $c \leftarrow c + w * d(F_j, D_i)$
    $u_i \leftarrow u_i - w$
    $v_j \leftarrow v_j - w$
    if $v_j = 0$ **then**
      F.pop(F_j)
return c

3.2.2 Dual form estimation

Dual form formulation gives us another way to compute the Wasserstein distance by solving a different (and hopefully easier) optimization problem. Concretely, the Kantorovich-Rubinstein duality [2] of the Wasserstein distance is

$$W_1(p, q) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim p(x), y \sim q(y)} [f(x) - f(y)]$$

The primal form of Wasserstein distance is lower bounded by the difference of $f(x)$ and $f(y)$ where f is any 1-Lipschitz function, solving the dual problem of maximizing the right hand side yields the same solution as solving the original problem of minimizing transportation cost. When one replaces the 1-Lipschitz by some other K-Lipschitz constraint, the left hand side becomes KW. Therefore in general, we only need to optimize $E[f(x) - f(y)]$ with f belongs to a parametrized set of functions with some Lipschitz constraint which we do not need to know exactly, the result of the optimization process would be an estimate of $W(p, q)$ up to a multiplicative factor. In the deep learning context, the function f is usually represented as a neural network, with some mechanism to enforce the Lipschitz constraint, for example weight clipping [2] and gradient penalty [10]. The resulting objective resemble that of the GAN formulation.

Weight clipping is the method that was first proposed in GAN literature, in which the neural network function f is constrained in a compact space by clamping the neural weights in a fixed box. This is a quite simple method but it introduces a new clipping threshold hyper-parameter which might be hard to fine-tune. Besides, with deep neural nets, a suitable clipping parameter can vary greatly between shallow and deep layers, and using a single clipping constant throughout the network can cause gradient vanishing or exploding as shown in [10]. An improvement of the method is to directly constraint the Lipschitz by adding a gradient norm to the loss function, hence the name gradient penalty. Gradient penalty is shown to be better than weight clipping and has been successfully applied to a variety of problems in generative modeling.

Algorithm 2 Dual form WD-GP

Input: D, F: Sample sets of same size, n: sample sets' size, f_θ: test function, m: Batch size
γ: learning rate, λ: Lipchitz coefficient
while θ not converge **do**
  for i=1..m **do**
    Sample u from D, v from F
    Sample random ϵ from U[0, 1]
    $x \leftarrow \epsilon u + (1-\epsilon)v$
    $L_i \leftarrow f_\theta(u) - f_\theta(v) + \lambda(\|\nabla_x f_\theta(x)\|_2 - 1)^2$
  $\theta \leftarrow \theta - \gamma \nabla_\theta \sum_i^m L_i/m$
for i=1..n **do**
  $d_i \leftarrow f_\theta(F_i) - f_\theta(D_i)$
return $\sum_i^n d_i/n$

The Algorithm for calculating the dual WD with Gradient penalty is provided in 2. This algorithm is derived from a statement that any optimal function that satisfies the 1 Lipchitz constraints has a gradient norm of 1 along the straight line which connects the two distributions (see [10] for the derivation). Thus [10] implemented this idea through a Sampling distribution that uniformly samples data points on straight lines between pairs of data points and enforces the gradient constraints in these samples, which is realized in the pseudocode as the gradient norm of the test function with respect to a convex sum of the two samples from each distribution.

3.2.3 Sliced Wasserstein distance

Sliced Wasserstein distance (SWD) [13] takes a rather different perspective compared to other approaches in OT. The motivation of Sliced Wasserstein distance is that we can estimate the Wasserstein distance in high dimensional spaces by projecting distributions to R, efficiently compute the OT distances in these projected spaces and then aggregate the distances to a final estimation. Continue on the previous section, let p and q are probability on R^d, S^{d-1} is the unit sphere, $U(S^{d-1})$ is the uniform distribution on S^{d-1},

the Sliced Wasserstein distance is

$$SW_m(p,q) = \left(\int_{S^{d-1}} W_m^m(R_v(p), R_v(q))\mathrm{d}v\right)^{\frac{1}{m}} = \left(\mathbb{E}_{v\sim U(S^{d-1})}\left[W_m^m(R_v(p), R_v(q))\right]\right)^{\frac{1}{m}}$$

Where $R_v(.)$ is the linear projection operation onto a 1 dimensional space along the directions v drawn from S^{d-1}. Essentially, the above formula says that computing the SWD is equivalent to solving many 1-D Wasserstein distances, which have a closed-form solution, in the projected spaces, then averaging the results over all projection directions.

Algorithm 3 Sliced Wasserstein

Input: u, v: probability distributions, D, F: Sample sets
n_proj: number of projections, d: dimensions of samples
$d \leftarrow 0$
for n_proj **do**
  Sample random projection $w \sim U(S^{d-1})$
  $D^w \leftarrow w^T D, F^w \leftarrow w^T F$
  $D^w_\sigma \leftarrow$ sorted $D^w, F^w_\sigma \leftarrow$ sorted F^w
  Initialize lists A= $[(u_i, D^w_{\sigma i})]_i$ and B= $[(v_j, F^w_{\sigma j})]_j$
  $u' \leftarrow 0, v' \leftarrow 0$
  while A $\neq \emptyset$ and B $\neq \emptyset$ **do**
    $u', k \leftarrow$ A.pop() if $u' = 0$
    $v', l \leftarrow$ B.pop() if $v' = 0$
    $t \leftarrow \min(u', v')$
    $d \leftarrow d + t * \|l - k\|$
    $u' -= w$
    $v' -= w$
return d/n_proj

In practice, since it is not possible to calculate the integral in SWD, we generally approximate the SWD using Monte Carlo method: by randomly drawing a number project directions, computing the WD in the projected line and taking the empirical mean of these projected WD as the estimate of SWD. The pseudocode of the procedure is given in Algorithm 3. The basic idea of the pseudocode 3 is that, we can compute the coupling matrix of two 1-dimensional distributions by utilizing a greedy approach as follows: since each sample in the distributions is represented by a scalar in 1D space, the lowest sample should prioritize transporting as much as possible all of its weights to the nearest sample in the other distribution, which is also the lowest sample in that distribution, until one of the two sample run out of weight or capacity, then the process keep continuing to the rests, thus a simple sort would do the tricks. A similar procedure can be found in [12].

One drawback of the Sliced Wasserstein distance is that it estimates the expectation by using Monte-Carlo method that can be inefficient in high dimensional spaces. SWD therefore usually needs a large number of slices to get an accurate estimate as irrelevant directions that do not contribute much to distinguishing distributions overwhelm other relevant ones. Recently, there are several other proposals to improve SWD by trying

to focus on more meaningful projection directions, for example Max Sliced Wasserstein [7] that searches for the best Dirac distribution over the unit sphere and Distributional Sliced Wasserstein distance [15] that finds the optimal projection distribution with a cosine similarity regularization constraint.

3.2.4 Entropic Regularization

We've known from the previous sections that optimal transport can be formulated as a linear programming problem (which will be illustrated through an example in the next section) and that it can be solved to get the exact solution. But sometimes the exact solution does not mean much in practice. One example of this is that in image retrieval and color transfer, the OT distance of transforming the signature of one image to others' [17] has little practical interpretation. In addition, function approximators like neural networks lie at the heart of machine learning and in many cases, in order for a neural net to work, we don't really need an exact answer. For that reason, perhaps it will be fine with an approximate solution as long as it is much faster and has other interesting properties, e.g differentiability and smoothness.

Sinkhorn distance [5] is an entropic constraint optimal transport metric that takes a novel strategy to speed-up the computation process. In Sinkhorn distance, the set of all available transportation plans, or the search space of the optimization, is restricted to a subset of joint distributions that are close to a distribution pq in terms of KL divergence. The newly added regularization term turns the transport problem into a strictly convex problem and can be approximated using Sinkhorn's fixed point iteration. Experiments show that the running time of Sinkhorn distance is several times faster than that of the original optimal transport.

We will start by defining a convex set of joint distributions with entropic constraint

$$\begin{aligned}\Pi_\alpha(p,q) &= \{\pi \in \Pi(p,q) | KL(\pi \| pq) \leq \alpha\} \\ &= \{\pi \in \Pi(p,q) | H(\pi) \geq H(p) + H(q) - \alpha\}\end{aligned}$$

Then the Sinkhorn distance of p and q is

$$d_\alpha = \inf_{\pi \in \Pi_\alpha(p,q)} \int_{X \times Y} d(x,y) \mathrm{d}\pi$$

Which is equivalent to the unconstrained optimization problem by Lagrange Multiplier,

$$\tilde{W}_{1,\lambda} = \inf_{\pi \in \Pi(p,q)} \left[\int_{X \times Y} d(x,y) \mathrm{d}\pi - \lambda H(\pi) \right] \tag{3.4}$$

for some positive λ that is dependent only on α. Usually the formula 3.4 is referred to as the definition of Sinkhorn distance, with the λ being some user-defined hyper-parameter that controls the degree to which the constraint contributes to the optimization. Sinkhorn distance is essentially the original OT distance plus an entropic regularization term. When α is large enough, the Sinkhorn distance returns to the ordinary Wasserstein distance. Indeed, we can see that when $\alpha \geq H(p) + H(q)$, the constraint on the joint

distribution $H(\pi) \geq H(p) + H(q) - \alpha$ becomes true for all $\pi \in \Pi$. In somes cases, the Sinkhorn distance is defined as $\tilde{W}_\lambda = \inf_{\pi \in \Pi(p,q)} \left[\int_{X \times Y} d(x,y) \mathrm{d}\pi + \lambda KL(\pi \| pq) \right]$ which is the Lagrange multiplier based on the KL constraint in the first definition of Π_α and is still equivalent to 3.4.

Sinkhorn distance satisfies two in three axioms of distance: symmetry and triangle inequality. Since when α is small enough, $d_\alpha(r,r) > 0$ for any r such that $H(r) > 0$, Sinkhorn distance cannot satisfy the coincidence axiom.

In case of discrete probability distributions, equation (1) becomes

$$\tilde{W} = \min_P \langle P, M \rangle - \lambda H(P) = \min_{p_{ij}} \sum_{ij} p_{ij} m_{ij} + \lambda p_{ij} \log p_{ij}$$

here $\langle \cdot, \cdot \rangle$ stands for the Frobenius dot-product, $P \in R^{d \times d}$ is the transportation plan matrix, $P 1_d = p$ and $P^T 1_d = q$ are the marginal distribution constraints and $M \in R^{d \times d}$ is the cost (distance) matrix. Then let $L(P, \alpha, \beta)$ be the Lagrangian of the above equation with dual variable α, β

$$L(P, \alpha, \beta) = \lambda \sum_{ij} (p_{ij} \log p_{ij} + p_{ij} m_{ij}) + \alpha^T (P 1_d - p) + \beta (P^T 1_d - q).$$

Differentiate the Lagrange with respect to the variable p_{ij}

$$\begin{aligned} 0 = \delta L / \delta p_{ij} &= \lambda \log p_{ij} + \lambda + m_{ij} + \alpha_i + \beta_j \\ \implies p_{ij} &= e^{-1/2 - \alpha_i/\lambda} e^{-m_{ij}/\lambda} e^{-1/2 - \beta_j/\lambda} \end{aligned}$$

From that, writing it in terms of vector-vector product, let $u = e^{-1/2-\alpha/\lambda}$, $v = e^{-1/2-\beta/\lambda}$ and $K = e^{-M/\lambda}$, then $P^\lambda := \mathrm{diag}(u) K \mathrm{diag}(v)$ is the optimal plan matrix, which depends on the unknown Lagrange multipliers α and β. Substitute P to the marginal constraints, we have that

$$\begin{cases} P 1_d = \mathrm{diag}(u) K v = u \odot (Kv) = p \\ P^T 1_d = \mathrm{diag}(v) K^T u = v \odot (K^T u) = q \end{cases} \implies \begin{cases} u = p \oslash (Kv) \\ v = q \oslash (K^T u) \end{cases} \tag{3.5}$$

where $\odot$ and $\oslash$ are element-wise multiplication and division operators, respectively. Equations in (3.5) tell us a way to compute u and v; knowing u gives v and vice versa. This is the key idea in Sinkhorn fixed point iteration algorithm: starting from some initial guess of u (or v), we then plug the guess into the update equation to get an estimate of v, and again plug this new v into the update equation to get a new estimate of u, the process repeats until u and v converge to some stationary points.

The pseudocode for computing Sinkhorn distance is illustrated in the Algorithm 4, which is a simplification version of the pseudocode from the original paper [5]. The computation process closely follows the aforementioned rationale. Actual implementation of this algorithm can be very suitable for any vector-matrix library for fast execution on both CPU and GPU, which is especially well-suited with many deep learning frameworks where these operations are thoroughly optimized and extensively used.

The characteristics of Sinkhorn distance change when varying λ. As alluded earlier, with small λ, Sinkhorn divergence converges to the exact OT divergence, meanwhile when

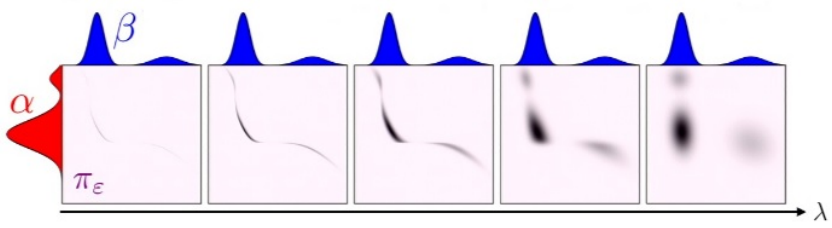


Figure 3.2: Illustration of the behaviour of Sinkhorn distance when varying λ. When λ is small, Sinkhorn distance is an approximate of the exact OT distance, the transportation matrix in this case is sparse with most of the mass concentrates on some small number of entries. With large λ, the transportation mass disperses all over other entries, giving a smoother transport function. Image from Dr. Gabriel Peyré's slide.

Algorithm 4 Sinkhorn distance

Input: p, q: probability distributions, C: Cost matrix, λ: Entropy coefficient
$K \leftarrow e^{-\lambda M}$
$v \leftarrow q$
while not converge **do**
  $u \leftarrow p/Kv$
  $v \leftarrow q/K^T u$
$P^\lambda \leftarrow \text{diag}(u)K\text{diag}(v)$
return $\text{sum}(P^\lambda * M)$

λ goes to infinity, the search space of the optimization problem collapses to the distribution pq^T, the outer product of p and q. In this case, the mass in a bin of one distribution is distributed to all the bins of the other distribution according to those bins' probability mass. Beyond these two extreme cases of λ, the entropy regularization term has the effect that smoothly distributes the transported weights in the coupling matrix, as described in Figure 3.2.

3.3 Approximate Policy Distance with Slice-Wasserstein Distance

In the previous section, we introduce the foundation of some well-known Optimal Transport metrics as well as how to implement them in general. In this section, we discuss their practicality and briefly discuss the advantages and disadvantages of each choice of algorithm in our problem.

In reality, especially in deep learning, we often deal with huge and complex problems with hundreds of thousands of sample points and high dimensional data which makes it hard or even impossible for the applicability of many methods. As mentioned earlier, computing the exact formula of Wasserstein distance requires solving a linear programming problem, which grows super-cubically with the number of data points. Additionally, the computation of Wasserstein distance relies heavily on another third-party Linear Programming solver, which can become cumbersome and hard to maintain in some situations.

In fact, it is rarely seen the case where pure Wasserstein distance is directly applied in the deep learning community.

Alternative approach to calculate the WD is to use the dual formulation as presented in the previous section. This approach has been very successfully applied in Generative Adverserial Network (GAN) as using the testing function equivalent to the discriminator in traditional GAN. The dual form estimation has several advantages compared to other methods as they are fast to compute and easy to integrate to any existing deep learning framework. However, as we mentioned before, the expectation of the difference between the outputs of the testing function of the two distribution equals to the Wasserstein distance up to a multiplicative factor, and since we in general do not know what this Lipchitz constant of the neural network is, the absolute values calculated by this method do not signify any significant connection to the true Wassteint Distance, as there always a hidden factor lurking in the background; we only have a glimpse of the relative distances between different samples of the two distributions, as higher value means the more different, or further they are. Moreover, when one wants to calculate the distances of multiple distributions by this dual formulation, she will need a separate neural network for each pair of distributions, which means that the total number of neural nets grows quadratically with the number of distributions. As a result, maintaining these networks might be very difficult and painful, and as these calculated distances are only meaningful locally with in-distribution samples, comparing distances among different pairs of distributions is of little help because they might have different Lipchitz constraints, not to mention the approximation error and the gradient norms are only enforced temporarily in the training process in the form of loss function but not inherently as a restricted subset of the hypothesis class, thus data points that are out of distribution might fall into regions where the gradient constraints are poorly imposed, making the estimation distance of these samples inaccurate.

The slicing approach is a good candidate to apply to problems that have small to medium size input dimension, as Sliced Wasserstein estimates the expectations by sampling uniformly random projection directions on a unit sphere, the number of directions grows exponentially with the number of dimensions. As a result, Wasserstein distance is not a good metric when the data are images. In the case where the two sample sets have the same size, computing the coupling in sliced Wasserstein distance can be implemented efficiently; when estimating 1-D WD from samples, and when the empirical probability of each sample is presumed to be identical and equal $1/N$ where N is the number of samples drawn from each distribution, we can employ a simple observation that each sample from one distribution should be paired with its corresponding from the other distribution, given that the samples are sorted in ascending (or descending) order (which is easily obtainable by just an ordinary sort in 1 dimensional case). An implementation of SWD in such cases can take advantage of any matrix algebra library for faster computing, the pseudocode is provided in Algorithm 5.

Sinkhorn distance is a regularized version of the Wasserstein distance that can be approximated by an iterative process, and as any other iterative procedure, there is a trade off between the accuracy and speed of the algorithm. Depending on the requirements of

Algorithm 5 SWD

Input: D: samples from first distribution, F: samples from second distribution,
n_proj: number of projections, n: dimensions of samples
$dist \leftarrow 0$
for n_proj **do**
  Sample random projection w
  $D^w \leftarrow w^T D, F^w \leftarrow w^T F$
  $D^w_\sigma \leftarrow$ sorted $D^w, F^w_\sigma \leftarrow$ sorted F^w
  $dist \leftarrow dist + \frac{1}{n}\|D^w_\sigma - F^w_\sigma\|^2$
return $dist/n\_proj$

the problem, we can adjust the hyper parameters to control the accuracy and computing speed of the process: the Sinkhorn iteration converges faster when the entropy weight is large and vice versa, and stopping criteria decide on when to halt the algorithm. One critical condition for the guarantee of the convergence of the Sinkhorn fixed point iteration is that the matrix K must be strictly positive. Therefore the stability of the Sinkhorn algorithm can be an issue when entries of the distance matrix M become large due to the exponentiation step that makes $e^{-\lambda M}$ close to zero.

Also, similar to other metrics, the Sinkhorn distance performs poorly when working with high dimensional data like images as metric distances like Euclidean distance are not statistically efficient in this regime. Some previous works applied methods stabilize and improve the efficiency of the method by finding appropriate cost functions: [6] applied the standardized L2 norm before computing the OT distance: the cost function used is the L2 norm of the normalized input with mean and standard deviation of the expert policy in the context of Imitation learning, [18] use the idea of adversarial learning a latent space that capture important features of the input data by minimizing the cosine similarity of the latent vectors, then the Sinkhorn distance is applied on this latent space, coupled with an energy distance function to ensure that the estimate distance is unbiased when trained with mini-batch algorithms.

All in all, we choose to use the Sliced Wasserstein distance in our transfer experience algorithm due to its simplicity: SWD does not require any auxiliary network; it is easy to implement and fast to compute with our problem size, also we can speed up the computation time by taking advantage of GPU libraries for deep learning like Pytorch and Tensorflow. The SWD is also computationally stable and easy to debug.

3.4 Experience Transfer via Policy Distance

Transfer learning in reinforcement learning is the use of external knowledge, possibly through previously learned tasks, to speed up the Reinforcement learning algorithm. The motivation behind transfer learning is that the experience obtained from source tasks can be effectively used to solve some related but different target tasks. Transfer knowledge through experience is one approach in transfer learning Reinforcement learning. In this

section, we consider the problem when we want to transfer experience in the form of transition dynamics collected by other tasks and use that to help the training of the target task.

The intuition of our method is that when two tasks are similar, they should share more of their experience and vice versa: when tasks are different, they should have a mechanism to constrain the sharing process if the sharing causes harm to the training. Our problem then boils down to the question of how to estimate the similarity between tasks. In this work, we consider using Optimal transport as a measure for comparing tasks, and take this OT distance to weight the relevance of transitions from other tasks. As a result, source tasks which have smaller OT distance to the target tasks will have a higher impact on the training of target agents, and when the two tasks diverge in OT distance, they should stop sharing experience.

One of the causes of the inefficiency of the on-policy RL approaches is their incapability to reuse past experience; most popular on-policy deep RL algorithms, including TRPO [19], PPO [20] and A3C [14] require fresh samples to be gathered for each gradient step. Off-policy algorithms on the other hand aim at reusing transitions in the past by using the Q learning based methods, which assesses the value of action state pairs by using Bellman update rule for transitions that do not necessarily come from the current policy. Therefore, we used an off-policy algorithm as our backbone RL algorithm for solving tasks since off-policy can reuse transitions from other policies, which in a sense is very similar to the sharing experience settings (they are indeed the same problem when the two tasks share the same transition dynamic and reward function).

The algorithm we chose is Soft Actor-Critic (SAC), which was proposed in [11]. Unlike other classical RL methods that maximize the expectation of the accumulated reward over entire trajectories, SAC optimizes an alternative RL objective which augments the standard RL reward function with an entropy term. The update rule in SAC is inspired from the soft Q learning update rule that employs a softmax operator instead of the max as in traditional Q learning. The behavior of SAC allows it to learn multiple near optimal policies while maintaining a good exploration noise.

The RL objective that Soft Actor-Critic considers is a maximum entropy objective: $J(\theta) = \mathbb{E}[\sum_t^T r_t + \alpha H(\cdot|s_t)]$. The optimization of SAC uses the soft policy iteration, which can be derived from the policy iteration algorithm: it involves two alternating steps of policy evaluation and policy improvement. In the policy evaluation step, the value function of the current policy is estimated, i.e to evaluate how good an action is in a specific state, given that the actions taken afterward are followed as the policy. Different approaches in RL tackle this evaluation step differently: for example, on-policy algorithms approximate value function directly via Monte Carlo method, which yields an unbiased estimation of the true value function but suffers from high variance, whereas off-policy methods use Bellman update rule to update an approximator function of value function, resulting in a lower variance but biased estimate. The learned value function in policy evaluation provides a mean to organize and guide the search for good policy in the policy improvement step. In SAC, the policy evaluation step is called the *soft policy*

evaluation which differs from the original evaluation procedure in the use Value function that already includes the entropy term, as if the entropy of an action is subsumed into the reward function. The soft policy update used in SAC is then

$$T^{\pi}Q(s_t, a_t) \triangleq r(s_t, a_t) + \gamma \, \mathbb{E}_{\pi} \, V(s_{t+1})$$

Where $V(s_t) = \mathbb{E}_{\pi}[Q(s_t, a_t) - \log \pi(a_t|s_t)]$ is the value function of a state, which resembles the vanilla value function in that it is the expectation of Q value over actions given state but plus the entropy of action distribution according to the current policy. The policy improvement of SAC is inspired from Soft Q learning, called the *Soft policy improvement*, in which instead of using the max operation as the best Dirac distribution in Q learning, an exponential distribution of the Q function is used as the target to update the policy. Furthermore, SAC assumes that the policy is restricted to some set of possible policies Π: we only have access to a limited capacity of a parameterized family of distributions, for example a neural network that returns the means and stds of a Multivariate normal distribution. The output of the soft policy improvement step is then the policy closest to the exponential policy in this desired set. An information projection in term of the Kullback-Leibler divergence onto the acceptable policy space turns out to provide a convenient objective function for the policy optimization as:

$$\pi_{\text{new}} = \arg\min_{\pi \in \Pi} D_{KL}\left(\pi(\cdot|s_t) \,\middle\|\, \frac{\exp(Q^{\pi_{\text{old}}}(s_t, \cdot))}{Z^{\pi_{\text{old}}}(s_t)} \right) \tag{3.6}$$

$$= \arg\min_{\pi \in \Pi} \mathbb{E}_{\pi} \left[\log \pi(\cdot|s_t) - Q^{\pi_{\text{old}}}(s_t, \cdot) + \log Z^{\pi_{\text{old}}}(s_t) \right] \tag{3.7}$$

$$= \arg\min_{\pi \in \Pi} \mathbb{E}_{\pi} \left[\log \pi(\cdot|s_t) - Q^{\pi_{\text{old}}}(s_t, \cdot) \right] \tag{3.8}$$

The distribution $\frac{\exp(Q^{\pi_{\text{old}}}(s_t, \cdot))}{Z^{\pi_{\text{old}}}(s_t)}$ is known as the *Boltzmann distribution* in statistical mechanics which gives the probability distribution of states as a function of energy and temperature of the system, the $Z^{\pi_{\text{old}}}(s_t) = \int \exp(Q^{\pi_{\text{old}}}(s_t, a))da$ is called the *partition function* that normalizes the distribution to appropriate range, however this partition function does not contribute to the solution of the optimization in 3.7 so it can be ignored. In case that the distribution is discrete, we have a function that is ubiquitously known as the *Softmax* operator in machine learning. In some sense, this soft update operator is a "soften" version of the Bellman optimality equation that replaces the max operator over action for state value with a softmax that converges to the argmax function as the temperature tends to infinity. SAC approximates the optimal policy defined in 3.8 by using the right hand side as a loss function and optimizes it with respect to the policy with gradient descent algorithms. One thing to note about equation 3.8 is that the expectation is taken over the current policy and does not depend on the transition distribution, which means that we can use transitions collected from any policies, as long as they operate on the same MDP. Thus SAC is an off-policy algorithm.

The algorithm for sharing samples between tasks is presented in 6. The optimal transport distances presented in the previous section are used to estimate the distance between tasks. Samples from source tasks can be transferred to the target task, weighted by the distance between two tasks. The training process of each agent consists of two

stages, the first stage is when the agent is trained from the samples collected by itself, with an off-policy algorithm; we used SAC for our experiment but it can be replaced by any other similar algorithms. In the second stage, the agent uses samples of other agents, trains as if samples were collected by itself, update steps are rescaled by the inverse distance between tasks.

Algorithm 6 Sharing experience pseudocode

```
envs[1..n]: set of all environments
agents[1..n]: set of all agents
buffers[1..n]: set of all buffers
n: number of tasks
while not converge do
    for n_step_interact_env do
        for i in 1..n do
            Collect (s_i, a_i, r_i, s'_i) from envs[i] by agents[i], append to buffers[i]
    for n_step_update_self do
        for i in 1..n do
            Train agents[i] on transitions sampled from buffers[i] with any offpolicy algorithm
    for n_step_update_share do
        for i in 1..n do
            B_anchor ← samples from buffers[i]
            B_positive ← samples from buffers[i]
            B_negative ← samples from buffers[j]; j ≠ i
            D_self ← Compute distance D(B_anchor, B_positive)
            D_share ← Compute distance D(B_anchor, B_negative)
            w ← D_self/D_share
            Train agents[i] on B_share with any offpolicy algorithm, weighted by w
return agents[1..n]
```

Sharing by samples was considered in great detail in [22], where the source and target tasks are assumed to have possibly different MDPs and reward functions. The transferring of samples are performed in the back of a batch RL framework *fitted Q learning*. [22] treated each of the optimization processes in fitted Q learning as an instance of empirical risk minimization, with samples from other tasks weighted with importance sampling weights according to the estimated distribution of each task, both for reward function distributions and transition distributions. Our method differs with [22] in a way that we consider using optimal transport to calculate the weight of samples from external tasks in place of importance weights. Importance sampling weight is also considered in [23] in robotics for life-long learning settings, however, the importance weight is only used for filtering out important samples to be saved for later task training but is not utilized any further than that.

3.5 Summary

In this chapter, we have covered the basic and general ideas of OT distances. We started by looking at the primal OT distance: Wasserstein distance, the difference of the characteristics of Wasserstein distance compared to other information distance metrics through a small experiment, we then discussed the difficulties encountered in computing the original OT problem through linear programming, and an alternative approach for fast approximation of the primal Wasserstein distance. Furthermore, this chapter presented two additional approaches to overcome these computational challenges: Slicing and entropic regularization, among which two of the most well-known metrics of each type are discussed, namely Sliced Wasserstein distance and Sinkhorn distance. We have also provided their pseudocode with a quick glance through their mathematical derivations for various OT algorithms and explained their strengths as well as weaknesses. The successive section discusses the backbone RL algorithms called Soft Actor-Critic that we will use in the later section and illustrate our sharing mechanism. In the next chapter, we will illustrate how to apply OT metrics to reinforcement learning and the computation and characteristics of the aforementioned OT distances through experiments. We will also focus on applying OT distances on a sharing experience basis to improve the training process and generalization of reinforcement learning agents.

CHAPTER 4. EXPERIMENTS AND RESULTS

4.1 Compute policy distance through state distribution

4.1.1 Toy example

This section illustrates how to compute discrepancy between policies in a simple example through OT distances of state distributions. We compute the distances in the state spaces, not in action spaces because of two reasons: 1) states are visible to outside observers, actions are not sometimes, this is inspired from DIAYN and WURL, and 2) state spaces are often metric spaces, which is necessary to apply OT distances. Other variants, for example concatenation of state and action are briefly discussed in the previous section.

Consider a grid world where there are no obstacles. In each step, agents can move freely on the grid one cell in 4 different directions up, down, left and right. Assume that there are three simple agents in this environment that:

- Agent 1 always moves up with 90% and right with 10%
- Agent 2 always moves right with 90% and up with 10%
- Agent 3 always moves down with 90% and left with 10%

For simplicity, let the horizon be 3; all the agents will stop after making three steps to the environment. Policies of these agents are constructed in a way that their trajectories form directed acyclic graphs, i.e they do not revisit visited cells, for the ease of calculation. We want to find the Wasserstein distance between the distributions of state induced by these policies.

In this example, we will find the Earth mover's distance between Policies of agents 1 and 2, 1 and 3, we expect that agent 1's policy should be closer to agent 2's than to agent 3's since the agent 3 goes to the opposite direction of agent 1. The states' probability distributions of the three agents are illustrated in Figure 4.1, the blue cell is the origin where all agents start, from left to right are the probability mass functions of state distribution of agents from 1 to 3, respectively.

The cost metric used in this example is Manhattan distance (see Fig 4.2), which makes sense in the grid environment. Solving the optimization problems with respect to P, we yield the Wasserstein distance between agent 1 and 2 is 3.2 and distance between

	WD	SWD	Sinkhorn $\lambda = .1$	Sinkhorn $\lambda = 1$	Sinkhorn $\lambda = 10$
agent 1 and 2	3.2	1.439854379	3.2172666	3.214429	3.2561991
agent 1 and 3	4.0	2.328088798	3.9999998	4.0	4.0
agent 1 and 1	0.0	0.004472817	0.0103338	0.62846744	1.1915964

Table 4.1: Comparing different OT distances. The distances are measured between agent 1 and 2, 1 and 3 and 1 and itself.

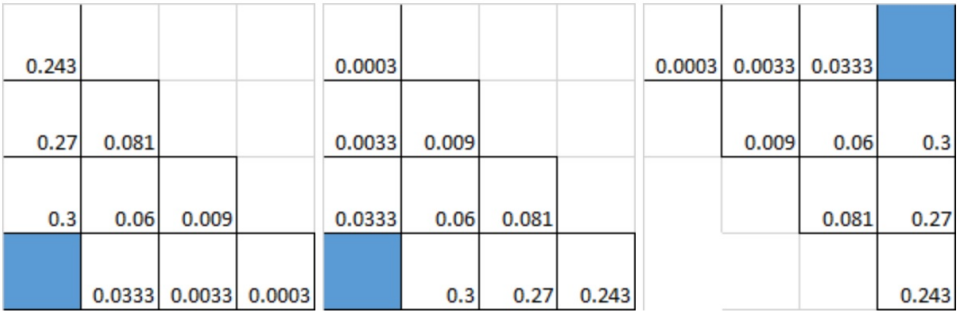


Figure 4.1: From left to right: probability mass functions of the state distributions visited by agent 1, 2 and 3 respectively.

agent 1 and 3 is 4, which verifies the initial conjecture. Other OT distances are described in details in table 4.1. With Sinkhorn distance, three different settings of λ are computed, ranging from 0.1 to 10. As can be seen, Sinkhorn distance between agent 1 and itself stays near 0 when λ is small but increases significantly as λ increases, Sinkhorn distances therefore do not satisfy the coincidence axiom with large λ. For other agents, Sinkhorn distance slightly increases but generally stays approximately around the Wasserstein distance. Sliced Wasserstein distance, on the other hand, lands on a completely different regime because of its novel slicing approach. Nevertheless, the relative relationship between agents still correlates with other OT distances. SWD of agent 1 and itself is not exactly equal to 0 as SWD uses Monte-Carlo method to estimate expectations.

0	2	4	6	1	3	5	2	4
2	0	2	4	1	1	3	2	2
4	2	0	2	3	1	1	2	2
6	4	2	0	5	3	1	4	2
1	1	3	5	0	2	4	1	3
3	1	1	3	2	0	2	1	1
5	3	1	1	4	2	0	3	1
2	2	2	4	1	1	3	0	2
4	2	2	2	3	1	1	2	0

Distance matrix of agent 1 and 2

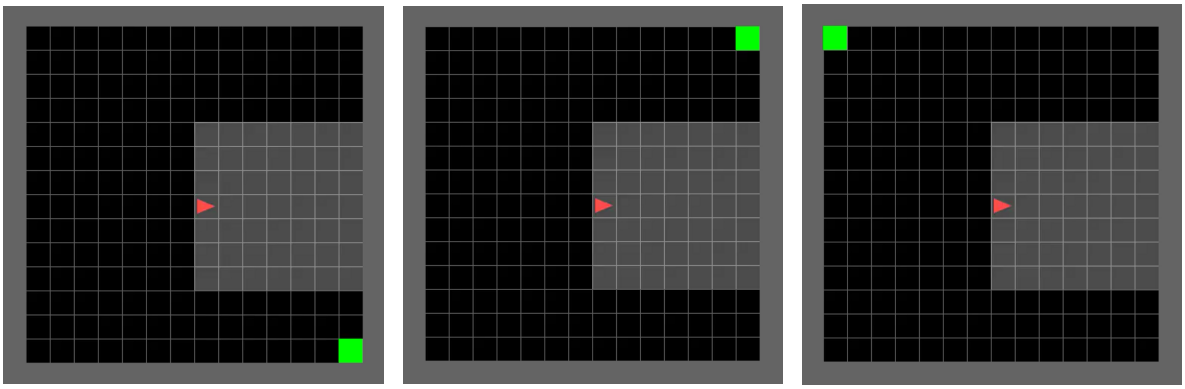
4	4	5	5	5	6	6	6	6
4	4	5	5	5	6	6	6	6
4	4	5	5	5	6	6	6	6
4	4	5	5	5	6	6	6	6
3	3	4	4	4	5	5	5	5
3	3	4	4	4	5	5	5	5
3	3	4	4	4	5	5	5	5
2	2	3	3	3	4	4	4	4
2	2	3	3	3	4	4	4	4

Distance matrix of agent 1 and 3

Figure 4.2: Cost matrices.

4.1.2 Experiments with Minigrid world

This section considers a real reinforcement learning example in more sophisticated environments: Minigird world [3]. Similar to the environment presented in the previous section, in the minigrid world, agents also navigate with the environment through four different directions. There are obstacles that block the paths of the agents like doors, walls, lava, balls etc. Minigird world is partially observable: agents can only see a small window of size 7×7 in front of them and agents' views are occluded by obstacles. The goal of the agents is to reach a specific location that is marked with green color. Upon reaching the goal positions, agents receive a reward $r \in [0.1, 1]$ with a small penalty being subtracted for the number of steps to reach the goal. If the agent cannot find the goal positions, then the reward equals zero. Minigrid world is sparse-reward environments



(a) Bottom Right room **(b)** Top Right room **(c)** Top Left room

Figure 4.3: Frames from three different custom Empty Room environments, with different goal settings. Starting position of the agent is always the same in all experiments, the green squares are goal positions, agents are represented by red triangles with pointed directions corresponding to the directions that the agents are currently facing. Shaded regions in front of agents are agents' views in the minigrid world.

since reward signals are only granted to agents at the end of every episode and only when agents hit goal. In some complex minigrid world environments, it is very hard for traditional reinforcement learning algorithms with vanilla random exploration strategies to find high reward states. For example, in MiniGrid-KeyCorridor, the agent is locked in a corridor with keys to open the doors hidden in some room. The agent has to explore the environment and find the keys in order to unlock doors without any reward feedback.

In this example, we experiment with three different settings of Empty Room world environments where the goal positions are placed in one room for each setting, we name these settings Bottom-Left-Room, Top-Right-Room and Top-Left-Room, each corresponds to different placements of the goal positions and is illustrated in figure 4.6. These environments are relatively easy to solve as there is no walls or obstacles that can block agents' trajectories, and there is no penalty when agents make mistakes. In all of the experiments, we use PPO [20] as a base training algorithm and train five different instances of the algorithm with different random seeds. We then plot the state visitation heatmap of each trained agent and compute OT distances between them. In all experiments, we use

Table 4.2: Hyperparameters used in PPO

Hyperparameters	**Values**
Num.timesteps	100k
Horizon (T)	600
Num.epochs	5
Minibatch size	128
Discount (γ)	0.99
Adam stepsize	10^{-3}
Clipping parameter (η)	0.2
VF coeff. c_1	1
Entropy coeff. c_2	0.01

	seed = 456	seed = 579	seed = 702	seed = 825	seed = 948
MiniGrid-TopLeft	0.967	0.974	0.964	0.976	0.964
MiniGrid-BottomRight	0.957	0.984	0.008	0.985	0.930
MiniGrid-TopRight	0.983	0.962	0.987	0.955	0.000

Table 4.3: Average reward obtained by agents trained on different seeds.

the same set of hyper-parameters which are described in table 4.4.

The heatmap of state visitation distribution of all experiments with different seeds of custom Empty Room are shown in figure 4.4. States that are visited more frequently have greater heat intensity. We average these state distributions as one marginal state distribution over seeds and compare the policies inducing them by optimal transport metrics. From the heatmap distribution, we can see that in some experiments, the agents have found optimal or near optimal paths to goals, but some have not. The expected reward in all experiments are displayed in table 4.3. Table 4.2 shows hyperparameters used in all experiments.

Table 4.3 shows the performance of the learned policies. We can see that these custom Empty-Rooms are easy environments, so the reward achieved is extremely high: approximate one, the highest possible reward, across seeds. In some experiments, the agent finds the optimal policy, i.e the shortest path to the goal. However, in some seeds the agents fail to solve the environment: they fail to reach their goal even once in 100 simulations conducted to produce this table. Investigating the heatmap distributions of these failed agents shows that their state trajectories concentrate mainly on the center of the maze, which might suggest that they are still exploring the environments.

Policies		WD	SWD	Sinkhorn $\lambda = .1$	Sinkhorn $\lambda = 1$	Sinkhorn $\lambda = 10$
TopRight	MiniGrid-BottomRight	3.012	1.650	nan	3.577	7.294
	MiniGrid-TopLeft	4.582	2.301	nan	4.943	8.673
	MiniGrid-TopRight	0.000	0.000	0.000	1.175	6.233
BottomRight	MiniGrid-BottomRight	0.000	0.000	0.000	1.265	6.703
	MiniGrid-TopLeft	3.897	2.044	nan	4.322	8.474
	MiniGrid-TopRight	3.012	1.649	nan	3.577	7.294

Table 4.4: OT distances of agents trained on Minigrid-TOpRight and and other settings. State visitation probabilities are averaged over all seeds. The motivation for designing these environments is similar to the toy example presented in the previous section: we expect that the agent that goes to the bottom-right would be further way to the top-left agent than the top-right agent.

Next, the Optimal transport distances of different metrics are computed between the learned policies with different tasks. The instability in the computation of Sinkhorn distance is visible from this experiment. From 4.4, it is apparent that with lambda equals 0.1 or lower, $e^{-M/\lambda}$ approximately becomes zero when M is large, which makes the computation in Eq 3.5 unstable. Actually, not all matrices K will make the Sinkhorn distances converge, a quick example is that when K is a diagonal matrix, then no matter how many iterations we take, Eq 3.5 will never converge since there is no such diagonal matrix P^{λ} that belongs to $\Pi(p, q)$. Also, in order for the Sinkhorn's theorem to hold, matrix K must be strictly positive, which is implicitly satisfied in the exponentiation steps but due to machine-precision limits in the floating-point representation that might round off these quantities to zeros. Possible ways to circumvent this are either to rescale the distance matrix M to an appropriate range or to use other more appropriate metrics, as briefly discussed in the section 3.2.4, however this is outside the scope of this thesis and is left for future discussions.

4.2 Sharing transitions via policy distance weights

In this section, we consider another custom multi-task Mini Grid environment where there are multiple goal squares, each goal square has a different color, the environment terminates when the agent reaches the goal with correct color. Each agent will have a different goal color to be reached and the reward is granted to agents only at the end of the episode. A sample frame from the multi-color Room is illustrated in figure 4.5. All goal positions and agents' starting positions are placed at random. The horizon of the environment is set to be at 200, meaning that agents will end an episode of interacting with the environment after 200 time steps at max. We test the sharing algorithm presented in 3.4 in three settings of 2 agents, 4 agents and 6 agents.

We report the progress of the agents at every fixed number of environment steps, the progress is measured by the empirical expected returns collected by running several episodes with separate evaluation environments. Each experiment is carried out multiple times with different seeds. The performance of all agents is average over all tasks and

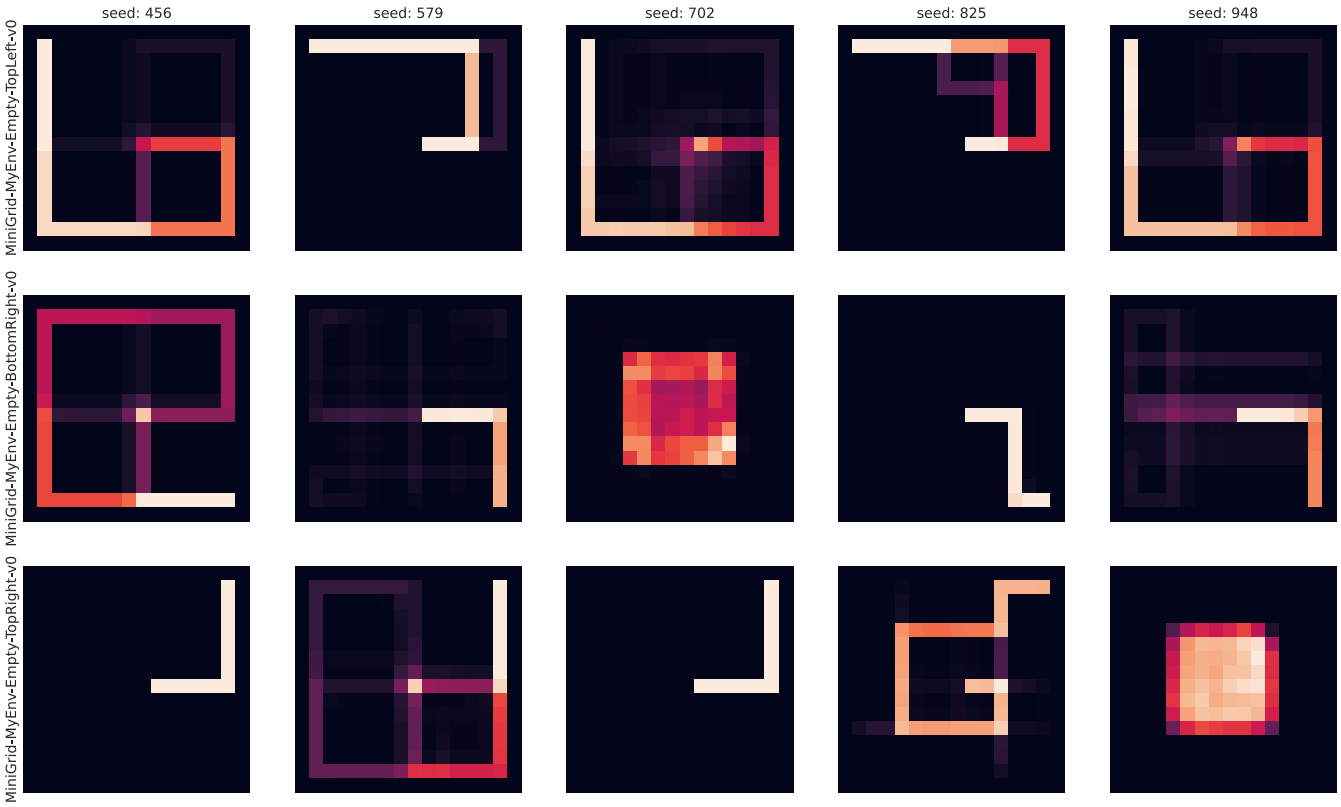


Figure 4.4: State visitation heatmaps for different settings, trained with five different seeds. The result is averaged over 100 simulations. With MiniGrid-Same, agents can easily find goal position, therefore their trajectories quickly converge to the optimal paths. For other settings, most of their time agents spend wandering at the states in the room from which they start, indicating that they might not sufficiently explore the environment enough to see other interesting high-reward states.

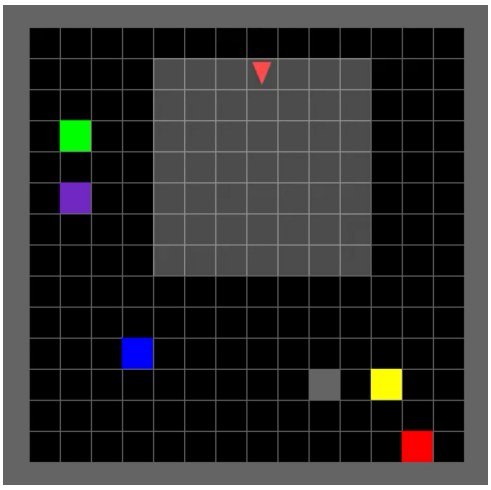
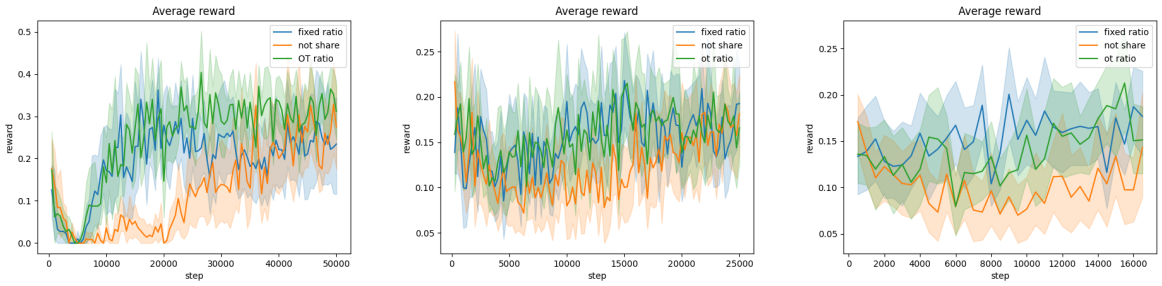


Figure 4.5: Multi-color goal Environment.

seeds. In each experiment, three different setups are implemented: one setup where all agents do not share experience, they are all trained independently to each other, this is similar to the case of training different tasks separately, the second setup where experience are always shared between agents, and shared samples from other agents are treated equally important as samples collected by the agent itself, in the final setup, shared experience is weighted with ratio according to the distance between the transitions of environments from which shared samples are collected and the environment of the task that agent are solving. For all experiments, we keep the hyperparameters of learning algorithms the same across all setups, which are provided in table **??**.



(a) Sharing with two agents **(b)** Sharing with four agents **(c)** Sharing with six agents

Figure 4.6: Results from sharing experience, with different number of agents. Results are averaged over 4 seeds in each experiment.

Table 4.5: Hyperparameters used in SAC

Hyperparameters	**Values**
Num.timesteps	100k
Buffer size	10k
Horizon (T)	500
Update every env step	1
Minibatch size	256
Discount (γ)	0.99
Adam stepsize	2.10^{-3} or 10^{-2}
target Entropy	0.4
Start step	400
Polyak	0.95

CHAPTER 5. CONCLUSION

Bibliography

[1] scipy stats wasserstein distance manual. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wasserstein_distance.html`. Accessed: 2021-17-12.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[3] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[4] Jack Clark. Faulty reward functions in the wild, Mar 2019.

[5] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transportation distances. *arXiv preprint arXiv:1306.0895*, 2013.

[6] Robert Dadashi, Léonard Hussenot, Matthieu Geist, and Olivier Pietquin. Primal wasserstein imitation learning. *arXiv preprint arXiv:2006.04678*, 2020.

[7] Ishan Deshpande, Yuan-Ting Hu, Ruoyu Sun, Ayis Pyrros, Nasir Siddiqui, Sanmi Koyejo, Zhizhen Zhao, David Forsyth, and Alexander G Schwing. Max-sliced wasserstein distance and its use for gans. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10648–10656, 2019.

[8] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

[9] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. The information geometry of unsupervised reinforcement learning. *arXiv preprint arXiv:2110.02719*, 2021.

[10] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[11] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[12] Shuncheng He, Yuhang Jiang, Hongchang Zhang, Jianzhun Shao, and Xiangyang Ji. Wasserstein unsupervised reinforcement learning. *arXiv preprint arXiv:2110.07940*, 2021.

[13] Soheil Kolouri, Kimia Nadjahi, Umut Simsekli, Roland Badeau, and Gustavo K Rohde. Generalized sliced wasserstein distances. *arXiv preprint arXiv:1902.00434*, 2019.

[14] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[15] Khai Nguyen, Nhat Ho, Tung Pham, and Hung Bui. Distributional sliced-wasserstein and applications to generative modeling. *arXiv preprint arXiv:2002.07367*, 2020.

[16] Georgios Papagiannis and Yunpeng Li. Imitation learning with sinkhorn distances. *arXiv preprint arXiv:2008.09167*, 2020.

[17] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pages 59–66. IEEE, 1998.

[18] Tim Salimans, Han Zhang, Alec Radford, and Dimitris Metaxas. Improving gans using optimal transport. *arXiv preprint arXiv:1803.05573*, 2018.

[19] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

[21] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[22] Andrea Tirinzoni, Andrea Sessa, Matteo Pirotta, and Marcello Restelli. Importance weighted transfer of samples in reinforcement learning. In *International Conference on Machine Learning*, pages 4936–4945. PMLR, 2018.

[23] Annie Xie and Chelsea Finn. Lifelong robotic reinforcement learning by retaining experiences. *arXiv preprint arXiv:2109.09180*, 2021.