Imperial College London

Department of Earth Science and Engineering

Msc in Applied Computational Science and Engineering

Independent Research Project

Final Report

Applications of Wasserstein Barycenter in Reinforcement Learning

Github: acse-yl27218

Candidate Name: Yihang Liao

Email: yihang.liao18@imperial.ac.uk

Abstract:

Continuous control of high-dimensional systems, including Go, video games, automating driving has been solved by state-of-the-art reinforcement learning algorithms. However, in most cases of these algorithms require a significant amount of data samples, which in real world cases the data samples are limited. Besides, in some cases, for example Atari Montezuma's revenge, even though the data samples are sufficient, the state-of-the-art algorithms get stuck and fails to learn to make a series of proper actions. This is the well-known exploration and exploitation dilemma in reinforcement learning. Though, a variety approaches of deep exploration has been proposed, most of them are model based. Recently, Metelli proposed a new direction to balance the exploration and exploitation in reinforcement learning by introducing the Wasserstein Barycenter to model the uncertainty of the value function. My work regenerates the Wasserstein Q learning algorithm into Pytorch framework and compare to several algorithms in OpenAI gym (Brockman *et al.*, 2017) environments, and the Deep Wasserstein Q learning shows its effectiveness in different environments, outperforms the state-of-the-art Deep Q Learning algorithm, especially in sparse reward environments.

1. Table of Contents

2. Introduction:

Reinforcement learning (RL) has shown its capability to solve decision-making problems in many fields (Mnih *et al.*, 2015; Silver *et al.*, 2016; Jaderberg *et al.*, 2018; Wang, Jia and Weng, 2018). During the training process, an agent needs to learn to take a sequence of actions to maximize its expected cumulative reward, while repeatedly interacting with the unknown environment. These interactions affect both rewards and the next state observations of the agents, which leads to the well-known exploration and exploitation dilemma (March, 1991). Balancing the exploration and exploitation has become the fundamental issues in RL: insufficient exploration may cause the agent stick at sub-optimal strategy while excessive exploration tends to incur a huge exploration cost. How to develop an optimal exploration strategy has been proposed by researchers in recent years and has been divided to several sub-fields.

Random exploration strategies, such as ϵ-greedy and Boltzmann exploration (Landau and Lifshitz, 1980), make use of the random noise during the action selection process to guarantee enough actions to be tried during the training following a given probability. Although these methods work well in some of the applications, they are not efficient since the exploration is random and not driven by any information on the estimation of the known experience during training. Especially in sparse reward environment where agent must make a sequence of correct actions to gain the rewards, random exploration strategy might converge towards the optimal behavior after an exponential number of steps or might not contributing to anything towards the agent's approximations to optimal solution.

Upper confidence bound (UCB) has been successful in multi-armed bandits' problem (Auer, 2002) and it is further developed to count-base exploration (Bellemare *et al.*, 2016) for complicated environments. UCB type of exploration methods directly use visit counts to guide an agent's behavior towards reducing uncertainty. Moreover, Thompson sample (TS) is another widely used exploration method with good practical performance and theoretical guarantees (Thompson, 1933; Russo *et al.*, 2018). UCB and TS, though are different approaches to solve the exploration-exploitation problems in RL, both quantifies the uncertainty of the value functions, which is a fundamental step towards efficient exploration. The notion of uncertainty is formalized in Bayesian statistics by means of a posterior distribution.

However, most of these exploration methods are model-based (Bellemare *et al.*, 2016; Pathak *et al.*, 2017) when applying to deep neural network, and it becomes computationally expensive when the environment is complicated, eg. Training Atari games by taking the graphical data as input. Model-free exploration strategies has become more and more popular recently in RL community. In Bayesian model-free RL, the notion of uncertainty is formalized in Bayesian statistics by means of a posterior

distribution of actions over the states (Dearden, Friedman and Russell, 1998). Bayesian RL make use of Bayesian inference, giving a principled approach to solve the exploration-exploitation dilemma. However, these methods rarely exploit the specific way in which the uncertainty propagates through the Bellman equation. In Metelli ’s work, Wasserstein-Q-learning (WQL), a Bayesian framework is proposed to address the problem of exploration using posterior distributions over the value function(Metelli, Likmeta and Restelli, 2019). Different from standard Bayesian update, Metelli proposed a new algorithm based on Wasserstein barycenter, which encodes the uncertainties into value function by computing the barycenter of state-action pair given the approximate posterior distribution, hence improved the learning abilities of algorithms. In this project, the WQL is reconstructed into Pytorch framework and an experimental evaluation on OpenAI gym environments is presented, comparing WQL with not only baseline DQN algorithm, but also a model-based exploration strategy, Curiosity-Driven exploration. These results show the effectiveness of WQL in balancing the exploration and exploitation during the training process compared to the novel reinforcement learning algorithm, while itself propagates the uncertainty without using the environment dynamics compare to typical exploration strategies.

3. Preliminaries

3.1 Markov Decision Process (MDP):

Markov Decision Process (Thomas, White and Puterman, 1995; Ng, Harada and Russell, 1999) is a fundamental model to describe the environment in reinforcement learning. At each discrete time, MDP is defined by a 5-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$, where $\mathcal{S}$ is the state space describing the possible configuration of the environment, $\mathcal{A}$ is the finite action space, $\mathcal{R}$ and $\mathcal{P}$, are the reward function and state transition function respectively and $\gamma \in [0,1)$ is the discount factor. The policy, π, defines the behavior of the agent given the current state $s_t \in \mathcal{S}$, that the action $a_t \in \mathcal{A}$,$a \sim \pi_\theta(\cdot|s_t)$. The state transition function $\mathcal{P}$ mapping the next state from current state and the action, $\mathcal{S}' \sim \mathcal{P}(\cdot|s_t, a_t)$, whereas the reward function $\mathcal{R}$ mapping the rewards from current state and action, $r_t \sim \mathcal{R}(\cdot|s_t, a_t)$.

3.2 Reinforcement Learning:

Reinforcement learning is an optimization process in which the agent learns a policy π by maximizing its expected reward during its interactions with an environment. During the training, agents seek to approximates its policy to optimal, that $\pi \approx \pi^*$, which the expected cumulative reward with discount is maximized. At time step t, the reward $R_t = \sum_{k=0}^{\infty} r_{t+k}$, which is the expected future reward from current state onwards in continuous tasks or until the final state. Typically, in reinforcement learning, the discount factor $\gamma \in [0,1)$ is used to avoid the reward from being infinite. The agents approximating the expected reward value functions, V^π and Q^π, indicates value functions and state-action value functions respectively.

$$V^{\boldsymbol{\pi}}(s,a) = E_\pi[R_t|s^t = s] = E_\pi[\textstyle\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,|s^t = s] \tag{1}$$

$$Q^{\boldsymbol{\pi}}(s,a) = E_\pi[R_t|s^t = s, a^t = a] = E_\pi[\textstyle\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,|s^t = s, a^t = a] \tag{2}$$

3.3 Bellman Equation:

The Bellman equation (Bellman Re and Zadeh La, 1970) for state-value and value function is denoted by:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s_,} P^a_{ss'}(\mathcal{R}(s,a) + \gamma V_\pi(s')) \tag{3}$$

$$Q^{\boldsymbol{\pi}}(s,a) = \sum_{s_,} P^a_{ss'}(\mathcal{R}(s,a) + \gamma \sum_{a'} \pi(a'|s')\, Q_\pi(s',a')) \tag{4}$$

Bellman equation decomposes the value function into instantaneous reward and the discounted future values, taking the advantage in the calculations of the reward value, that rather summing over entire time steps, it is calculated recursively at each discrete time-step.

3.4 Model-based and model-free RL:

The difference between model-based and model free RL is whether the agent learns the model of the environment. Without using the Bellman equations, which approximating the overall value function recursively, agent must learn the entire model of the environment, more specifically, the state transition function $\mathcal{P}$, that $\mathcal{S}' \sim \mathcal{P}(\cdot\,|s_t, a_t)$, to gives value function, as shown in Figure 1.

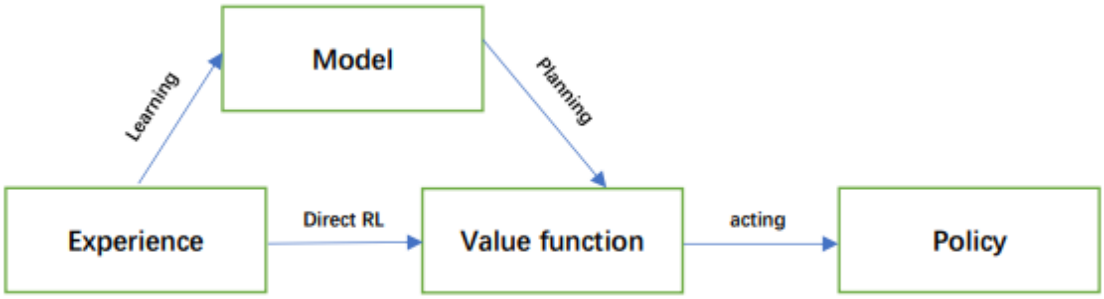

Figure 1: explanation of model-base and model-free RL

Through learning the value function directly from the experience, without knowing the overall MDP, model free reinforcement learning avoids having a computational expensive model. However, in some cases, the environment dynamics model is indeed required to training an agent to a human level. A good example is AlphaGo, of which based on the tree search mastering the game of GOs by visiting an exponential number of possible configurations, outperforms in the game of GO than professions.

Model-based and model-free RL have their own advantages and disadvantages. In model-based RL, the quantification of the exploration and exploitation is easier through modelling the environment, which prevents the agent from being stuck at local optimize policy. However, learning an extra model of the environment would extending the learning process, especially in those with high complexities, whereas in model-free RL, an agent directly learns from the value function, it is more computational efficient.

4. Theoretical Background:

4.1 Deep Q Learning (DQN):

DQN has a framework of deep neural networks, at each time step, DQN computes the distribution of state-action values corresponds to finite actions, given the current state values. Given the state-action value function, the agent selects the action that maximizes its expected reward, that:

$$a^t = argmax Q_\pi(s^t, a),\ s^t \in \mathcal{S}$$

$$r^t_{expect} = max V_\pi(s^t) = max Q_\pi(s^t, a) \tag{5}$$

Recalling the Bellman equation (4), the state-action value function of current time step is:

$$Q^{\boldsymbol{\pi}}(s, a) = E[R_t | s^t = s, a^t = a]$$

could be rewritten into:

$$Q^{\boldsymbol{\pi}}(s, a) = E_{s^{t+1}}\left[\mathcal{R}(s^t, a^t) + \gamma E_{a^{t+1}\sim\pi}[Q^{\boldsymbol{\pi}}(s^{t+1}, a^{t+1})]\right] \tag{6}$$

where s^{t+1}, a^{t+1} indicates the state and actions of the next time step. Besides, given the current step rewards R_t, the actual discounted cumulative reward is calculated:

$$\mathcal{y} = \mathcal{R}(s^t, a^t) + \gamma max_{a^{t+1}} Q^{\boldsymbol{\pi}}(s^{t+1}, a^{t+1}) \tag{7}$$

Given the approximation of the actual rewards, the loss function is defined by the temporal difference between the current Q value and the approximates actual reward.

$$\boldsymbol{\mathcal{L}}(\theta) = E_{s^t, a^t, r^t, s^{t+1}}[(Q^{\boldsymbol{\pi}}(s^t, a^t|\theta) - \mathcal{y})^2] \tag{8}$$

Finally, the gradient computation given the loss function is:

$$\nabla_\theta \boldsymbol{\mathcal{L}}(\theta) = E_{\boldsymbol{\pi}}(||Q^{\boldsymbol{\pi}}(s^t, a^t|\theta) - \mathcal{y}||_2^2) \nabla_\theta Q^{\boldsymbol{\pi}}(s^t, a^t|\theta) | s^t, a^t, s^{t+1}] \tag{9}$$

By the mean squared loss given the approximated actual returns and the values.

4.2 Deep Deterministic Policy Gradient (DDPG):

Policy Gradient (PG) method is the other baseline algorithm in reinforcement learning which differs to DQN, is available for continuous action domain (Lillicrap *et al.*, 2015). Rather than maximizing the action-value function approximations, PG differentiates the action-value function into two, actor function $\mu(s|\boldsymbol{\theta}_\mu)$ and the critic function $Q^{\boldsymbol{\pi}}(s, a|\boldsymbol{\theta}_Q)$. The actor function $\mu(s|\boldsymbol{\theta}_\mu)$, deterministically mapping states to the action, then given the action, the critic function directly calculates the expected return.

$$a^t = \mu\left(s^t|\boldsymbol{\theta}_\mu\right),\ s^t \in \mathcal{S}$$

$$r^t_{expect} = Q^{\boldsymbol{\pi}}\left(s^t, a^t|\boldsymbol{\theta}_Q\right), s^t \in \mathcal{S}, a^t \in \mathcal{A}$$

Given the distribution of action denoted by J, and $\mathcal{S}$ the actor function updated by following chain rule:

$$\begin{aligned}\nabla_{\theta^\mu} J(\theta) &= E_{s\sim\mathcal{S}, a\sim\mathcal{A}}\left[\nabla_{\boldsymbol{\theta}_\mu} Q^{\boldsymbol{\pi}}\left(s^t, a^t|\boldsymbol{\theta}_Q\right)\middle| a^t = \mu\left(s^t|\boldsymbol{\theta}_\mu\right),\ s^t \in \mathcal{S}\right] \\ &= E_{s\sim\mathcal{S}, a\sim\mathcal{A}}\left[\nabla_{a^t} Q^{\boldsymbol{\pi}}\left(s^t, a^t|\boldsymbol{\theta}_Q\right)\middle| a^t = \mu(s^t)\nabla_{\theta^\mu}\mu\left(s^t|\boldsymbol{\theta}_\mu\right),\ s^t \in \mathcal{S}\right] \qquad (10)\end{aligned}$$

whereas the value function is the same as DQN describing by equation (8). Hence, during the training, the critic is trained by approximating the actual reward, as shown:

$$\boldsymbol{\mathcal{L}}\left(\boldsymbol{\theta}_Q\right) = ||Q^{\boldsymbol{\pi}}\left(s^t, a^t|\boldsymbol{\theta}_Q\right) - \mathcal{R}(s^t, a^t)||_2^2 \qquad (11)$$

whereas, actor always maximizes the expected rewards gain, that:

$$\boldsymbol{\mathcal{L}}\left(\boldsymbol{\theta}_\mu\right) = -r^t_{expect} = Q^{\boldsymbol{\pi}}\left(s^t, a^t|\boldsymbol{\theta}_Q\right) \qquad (12)$$

4.3 ϵ-greedy exploration:

Greedy policy leads to poor exploration during the training in reinforcement learning, as an agent chooses an action directly from its policy, which limits the exploration to states of known rewards. Rather, the ϵ-greedy policy specifying a fixed or decaying random exploration rate, enabling the agent explores sufficiently.

$$\epsilon - greedy\begin{cases} a^t = random\ a \in \mathcal{A}, & with\ probability\ \epsilon \\ a^t = argmax Q_\pi(s^t, a) \in \mathcal{A}, & with\ probability\ \epsilon \end{cases} \qquad (13)$$

4.4 Curiosity driven exploration:

The curiosity driven exploration is a model-based reinforcement learning algorithm by introducing intrinsic reward function by intrinsic curiosity model (ICM), based on DDPG algorithm.

ICM has a framework of forward model and inverse model, and state encoders. The state encoder directly encodes the state s^t into $\phi(s^t)$, s^{t+1} into $\phi(s^{t+1})$, given the action a^t, forward model generates $\widehat{\phi}(s^{t+1})$ given $\phi(s^t)$ and a^t, the inverse model generates $\hat{a}^t$ given $\phi(s^t)$ and $\phi(s^{t+1})$.

The ICM model is trained via two steps:

1. Forward model is trained by minimizing difference between $\hat{\phi}(s^{t+1})$ and $\phi(s^{t+1})$

2. Inverse and encoder models are trained by minimizing difference the prediction action $\hat{a}^t$ and a^t

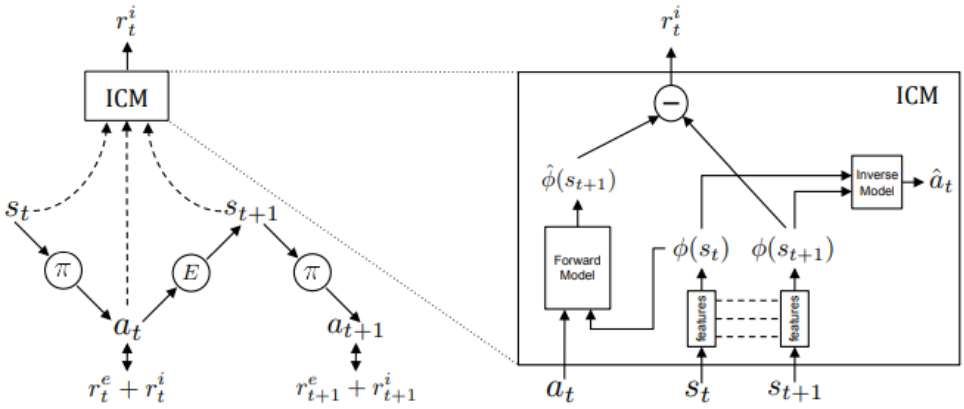

Figure 2: The curiosity driven exploration network(Pathak et al., 2017)

Given the encoded state $\phi(s^{t+1})$ and its prediction by the forward model, $\hat{\phi}(s^{t+1})$, the intrinsic reward is calculated by the 2-norm of encoded state and its prediction, $r_{intrinsic} = \left|\left|\hat{\phi}(s^{t+1}) - \phi(s^{t+1})\right|\right|_2^2$, and the expected return of the critic becomes $\mathcal{R}(s^t, a^t) + r_{intrinsic}$. Hence, the loss of the critic in curiosity driven network is:

$$\mathcal{L}(\boldsymbol{\theta}_Q) = ||Q^{\boldsymbol{\pi}}(s^t, a^t|\boldsymbol{\theta}_Q) - \mathcal{R}(s^t, a^t) + r_{intrinsic}||_2^2 \quad (14)$$

4.5 Wasserstein Barycenter:

Let $(\mathcal{X}, d)$ be a complete separable metric space and $x_0 \in \chi$ be an arbitrary point. For each $p \in [1, +\infty]$ we define $\mathcal{P}_p(\mathcal{X})$ as the set of all probability measures μ over $(\mathcal{X}, \mathcal{F})$ such that $E_{X\sim\mu}[d(X, x_0)^p] < +\infty$. Let $\mu, \nu \in \mathcal{P}_p(\mathcal{X})$, the L^p-Wasserstein distance between μ and ν is calculated as (Metelli, Likmeta and Restelli, 2019):

$$W_p(\mu, \nu) = \left(\inf_{\rho\in\Gamma(\mu,\nu)} E_{X,Y\sim\rho}[d(X, x_0)^p]\right)^{1/p} \quad (15)$$

4.6 Wasserstein DQN:

Recalling DQN trained by updating the state-action value function during the interactions with the environments. The state-action value function $Q^{\boldsymbol{\pi}}(s, \mathcal{A})$, maintains a probability distribution of actions, given the states. As the model-free updates the probability distribution could not be represented exactly, but an approximating of probability distributions of actions $\wp \subseteq \mathcal{P}\left(Q^{\boldsymbol{\pi}}(s, \mathcal{A})\right)$ could be obtained given the sampling of past experiences. The Wasserstein DQN, introducing an estimation of V-

posterior $V(s)$, being the Wasserstein barycenter of the current action value function $Q^{\boldsymbol{\pi}}(s,\mathcal{A})$, over the entire action space.

$$V(s) \in argin f_{V\in\wp}\{E_{a\sim\boldsymbol{\pi}}[W_2\big(V, Q^{\boldsymbol{\pi}}(s,\mathcal{A})\big)^2] \tag{16}$$

when the policy $\boldsymbol{\pi}$ is known, the expectation over the action space can be computed if action space $\mathcal{A}$ is finite. Moreover, when the state-action function $Q^{\boldsymbol{\pi}}(s,a)$ are deterministic distributions, $V(s)$ is a deterministic distribution too centered in the mean of $Q^{\boldsymbol{\pi}}(s,a)$.

Specifically, $(V(s), Q^{\boldsymbol{\pi}}(s,\mathcal{A}))$ is a measurable space with the probability $\wp$ distributions of values measure over $Q^{\boldsymbol{\pi}}(s,\mathcal{A})$. The Closed-Formed Wasserstein Barycenter is approximating by the expectation of the values given the measure of the actions' distributions.

$$W_2\big(V(s), Q^{\boldsymbol{\pi}}(s,\mathcal{A})\big) \approx E(Q^{\boldsymbol{\pi}}(s,\mathcal{A})|\wp) \tag{17}$$

Given the value function is a Wasserstein Barycenter of $Q^{\boldsymbol{\pi}}(s,\mathcal{A})$ posterior distributions, the loss function:

$$\boldsymbol{\mathcal{L}}(\theta) = E_{s^t,a^t,r^t,s^{t+1}}\left(\left(W_2(Q, Q_t^{\pi}(s,\mathcal{A})\right) - \mathcal{y}\right)^2 \tag{18}$$

where,

$$\mathcal{y} = \mathcal{R}(s^t,a^t) + \gamma W_2\big(V(s^{t+1}), Q_t^{\pi}(s^{t+1},\mathcal{A})\big) \tag{19}$$

5. Implementations:

5.1 Double Deep Q Learning (DDQN):

DDQN is an improvement of DQN algorithm, with asynchronous updates of the value and the action-value network, which prevents the model from overestimating the actual cumulative rewards hence stabilizing the training process.

Recalling the equation (7), of which the action and the expected reward of current time step defined by the action-value function $Q^{\boldsymbol{\pi}}$, the DDQN algorithm has two separates action-value function $Q^{\boldsymbol{\pi}}$, $Q^{\boldsymbol{\pi}}_{target}$, computes the action and expected rewards respectively. Equation (7) becomes:

$$\mathcal{y} = \mathcal{R}(s^t, a^t) + \gamma max_{a^{t+1}} Q^{\boldsymbol{\pi}}_{target}(s^{t+1}, a^{t+1}) \tag{20}$$

The pseudo code for DDQN is shown below:

ϵ-greedy DDQN pseudo code:

Initialize the environment, replay memory $\mathcal{D}$ to capacity N,action-value function $Q^{\boldsymbol{\pi}}(\mathcal{S}, \mathcal{A})$, $Q^{\boldsymbol{\pi}}_{target}(\mathcal{S}, \mathcal{A})$

for** episodes in [0,1, 2, … ,Maxepisodes], **do

for** time $t, \dots, t_{terminal}$, **do

select actions according to ϵ-greedy policy

observe s^t, a^t, s^{t+1}, r^t

add the observed information to replay buffer $\mathcal{D} \leftarrow (s^t, a^t, s^{t+1}, r^t)$

sample minibatch of tuple $(s^j, a^j, s^{j+1}, r^j) \sim \mathcal{D}$

$$\mathcal{y}^j = \begin{cases} \mathcal{R}(s^j, a^j), if\ \ s^{j+1}\ is\ terminal \\ \mathcal{R}(s^j, a^j) + \gamma max_{a^{t+1}} Q^{\boldsymbol{\pi}}_{target}(s^{t+1}, a^{t+1}), else \end{cases}$$

Computes MSE loss $\boldsymbol{\mathcal{L}}(\theta)$ according to equation (11).

Update $Q^{\boldsymbol{\pi}}(\mathcal{S}, \mathcal{A})$

Every l_{target} steps Update targets $Q^{\boldsymbol{\pi}}_{target}$ by copying $Q^{\boldsymbol{\pi}}(\mathcal{S}, \mathcal{A})$

end for

end for

5.2 Curiosity Driven DQN:

Curiosity driven exploration algorithm was designed based on DDPG algorithm in continuous action domain environments. This project rewrites the curiosity driven exploration based on DQN algorithm. Rather than predicting the actual actions given the current state s^t and next state s^{t+1}, the intrinsic curiosity model approximating the actual rewards $\mathcal{R}(s^t, a^t)$, denoted by $V(s^t, s^{t+1})$.

With the additional environment dynamics, the curiosity driven network generates the intrinsic rewards forcing itself to try sufficient actions at different time step. However, compared to the state-of-the-art DQN network, as described in

Curiosity driven DQN:

Initialize the environment, replay memory $\mathcal{D}$ *to capacity* N*,action-value function* $Q^{\boldsymbol{\pi}}(\mathcal{S},\mathcal{A})$ *,* $Q^{\boldsymbol{\pi}}_{target}(\mathcal{S},\mathcal{A})$

Initialize dynamic model and target dynamic model, ICM, ICM_{target}

for *steps < Max_Steps,* ***do***

for *time* $t, \dots, t_{terminal}$*,* ***do***

select actions according to ϵ*-greedy policy*

observe s^t, a^t, s^{t+1}, r^t

add the observed information to replay buffer $\mathcal{D} \leftarrow (s^t, a^t, s^{t+1}, r^t)$

sample minibatch of tuple $(s^j, a^j, s^{j+1}, r^j) \sim \mathcal{D}$

$\widehat{\phi}(s^{t+1}), \widehat{\phi}(s^t)$ *is computed by forward model of* ICM_{target}

$r^i_{intrinsic} = ||\widehat{\phi}_{target}(s^{t+1}), \widehat{\phi}_{target}(s^t)||_2^2$

$$\mathcal{y}^j = \begin{cases} \mathcal{R}(s^j, a^j) + r^i_{intrinsic}, if\ \ s^{j+1}\ is\ terminal \\ \mathcal{R}(s^j, a^j) + r^i_{intrinsic} + \gamma max_{a^{t+1}} Q^{\boldsymbol{\pi}}_{target}(s^{t+1}, a^{t+1}), else \end{cases}$$

Computes $\boldsymbol{\mathcal{L}}(\theta_\pi)$ *according to equation (11)*

$\boldsymbol{\mathcal{L}}(\theta_{\text{forward}}) = ||\widehat{\phi}(s^{t+1}), \widehat{\phi}(s^t)||_2^2$

$\boldsymbol{\mathcal{L}}(\theta_{\text{inverse}}) = ||V(s^t, s^{t+1}), \mathcal{R}(s^t, a^t)||_2^2$

Update $Q^{\boldsymbol{\pi}}(\mathcal{S},\mathcal{A})$

Update ICM *model*

Every l_{target} *episodes Update targets* $Q^{\boldsymbol{\pi}}_{target}, ICM_{target}$

end for

end for

5.3 Closed Form Wasserstein Barycenter:

The Wasserstein metric has a closed form expression given the probability measures on the sample, conditioning in the probability space is convex. The 2-wasserstein barycenter could be expressed as:

$$W_2(\mu, \nu) = \int_0^1 \left(F^-_{\mu}(u) - F^-_{\nu}(u)\right)^2 du \tag{21}$$

where F^- is the quantile function, this is the Kantorovich dual form of the W_2 - barycenter. In the training step of reinforcement learning, the batch sample of $Q^{\boldsymbol{\pi}}_{batch}(\mathcal{S},\mathcal{A})$, for each component in the batch samples, the weighting $w_k, k \in [1,2,3,\dots,K_a]$, indicating the probability measures of the action given batch, having $f_{\mu}(x) = \sum_1^K w_k\, \delta(x - u_j)\ and\ f_{\mu}(x) = \sum_1^K w_k\, \delta(x - v_j)$ as pdfs. Then the 2-Wasserstein distance between the space:

$$\begin{aligned} W_2(\mu, \nu)^2 &= \int_0^1 \left(F^-_{\mu}(u) - F^-_{\nu}(u)\right)^2 du \\ &= \sum_1^K \int_{I_j} \left(F^-_{\mu}(u) - F^-_{\nu}(u)\right)^2 du \end{aligned}$$

$$= \sum_{1}^{K} \int_{I_j} (u_j - v_j)^2 du$$

$$= \sum_{1}^{K} (u_j - v_j)^2 \int_{I_j} du$$

$$= \sum_{1}^{K} w_k \, (u_j - v_j)\text{^}2$$

where I_j is the quantile function of cumulative probability distribution. In reinforcement learning, the update of the state-action pairs does not encode the values of uncertainty.

However, the probability distribution of actions approximated by a batch sample from the experience replay, if the sampling batch size is small, the probability distribution would be inaccurate, which may lead to the calculation of values inaccurate. The unexpected sample errors may lead to unconvergence of training.

5.4 Wasserstein Deep Q Learning:

Wasserstein DQN Pesudo Code:

Initialize the environment, replay memory $\mathcal{D}$ *to capacity* N*,action-value function* $Q^{\pi}(\mathcal{S}, \mathcal{A})$ *,* $Q^{\pi}_{target}(\mathcal{S}, \mathcal{A})$

for *steps < Max_Steps,* ***do***

for *time* $t, \ldots, t_{terminal}$*,* ***do***

select actions by $Q^{\pi}(\mathcal{S}, \mathcal{A})$

observe s^t, a^t, s^{t+1}, r^t

add the observed information to replay buffer $\mathcal{D} \leftarrow (s^t, a^t, s^{t+1}, r^t)$

sample minibatch of tuple $(s^j, a^j, s^{j+1}, r^j) \sim \mathcal{D}$

$$y^j = \begin{cases} \mathcal{R}(s^j, a^j), if \;\; s^{j+1} \; is \; terminal \\ \mathcal{R}(s^t, a^t) + \gamma W_2 \left(Q^{\pi}_{target}(s^{t+1}, \mathcal{A}), Q^{\pi}_{t}(s^{t+1}, \mathcal{A}) \right), else \end{cases}$$

Computes MSE loss $\mathcal{L}(\theta)$ *by equation (18,19)*

Update $Q^{\pi}(\mathcal{S}, \mathcal{A})$ *,*

Every l_{target} *steps Update targets* Q^{π}_{target}

end for

end for

The Wasserstein DQN algorithm has a similar framework as DDQN, instead updating the Value function encodes with uncertainty through computations of the Wasserstein Barycenter of value functions in batch samples

6. Result & Discussion:

6.1 Experiment Setup:

In this project, other than the state-of-the-art DQN with greedy and ϵ-greedy exploration, curiosity-driven-exploration is implemented as well with slightly modifications into DQN (base algorithm) instead of DDPG. The experiments are conducted in OpenAI gym classic control, and was expected to add experiments of Atari games, with both ram and graphics input. However, the training for Atari games haven't completed due to the computational complexity and limited time of this project. All the experiments ran on a computer with Intel i9-9900k CPU, NVIDIA GeForce RTX2070S GPU and 64GB of DDR4 RAM.

The algorithms could be trained by CPU or with GPU acceleration, with CUDA version 11.1 and Cudnn 0.8.5, written in pytorch framework with pytorch version 1.8.0 and torchvision 0.9.0.

Wasserstein-DQN, Double-DQN, ϵ-greedy Double-DQN, Curiosity-driven-DQN are the 4 algorithms applied to different OpenAI baselines. The environments of classic control include, Cartpole, Pendulum, Acrobot, MountainCar.

Table of environments:

Environment	Observation	Action-type	Action-range	Reward-type	Episode's length:
Cartpole	4	Discrete	[0, 1]	dense	[0,500]
Pendulum	2	Continuous	[-2,2]	dense	200
Acrobot	6	Discrete	[-1,0,1]	dense	[0,500]
MountainCar	2	Discrete	[-1,0,1]	Sparse	[0,200]

In OpenAI gym classic controls, the action space could be either discrete or continuous. If the environment has continuous action space, action discretization function would be used to discretize the infinite number of actions into a line space into finite number of actions so that the DQN network could be adapted into continuous action environment.

In CartPole, Pendulum and Acrobot environment, the rewards are dense (Appendix 10.1). The MountainCar environments (Moore, 1990), is different from other environments, only generating reward at the terminal steps of an episodes. This environment is initialized with a car controlled by the agent, which is initialized at the bottom of a valley. Agent may choose to accelerate to the left, right or cease any

acceleration at each time step. The environment would either terminate at maximum 200 steps or the car position reach to the top, with reward of 0 if agent successfully complete the task else reward of -1. In this environment, agent should take a series of good action to obtain the rewards (hard for taking random actions).

Table of hyperparameters:

Optimizer	Adams
Length/episodes	1000
Update Period/steps	1
Target Update Period/steps	1000
Update Start/episodes	50
Reward Discount	0.99
Hidden Size	16
Batch Size	512
Learning Rate	0.005

An efficient exploration strategy should not only mastering environments with dense rewards, but also in those with sparse rewards. In order to test the efficiency of the algorithm, the above four environment are used with same hyperparameters, as shown in table 2.

The number of episodes to train is 1000, considering the classic control environments are small and easier to train. Hence, the range of steps taken by agent in CartPole and Acrobot is [0,500000], in MountainCar is [0,200000] and in pendulum is fixed 200000. At the start of the training, agents make random actions, not until the episodes number > Update Start (10) agents start to choose actions according to its policy.

6.2 StatsWrapper:

Based on *cherry-rl* package (*learnables/cherry: A PyTorch Library for Reinforcement Learning Research, 2019*), the stats wrapper is a wrapper to OpenAI gym environment which would not only generating the log information but also records the episodic rewards and steps rewards along with standard deviations of rewards given the record interval, which make the plotting of the statistic easier.

The Logs message is generated periodically in steps or episodes, clarifying by a bool parameter record_episodic and an int parameter interval l_{log}. The logger generating logs every l_{log} episodes or every l_{log} steps.

The episodic rewards and its standard deviations are calculated and stored in list, given by the parameter: l_{ep}, the episode interval with default value of 5. Whenever the logger updates, the mean reward of l_{ep} episodes and its standard deviation is recorded.

Similarly, the steps rewards and its standard deviations are calculated and stored in list, given by the parameter: l_{steps}, the steps interval with default value of 1000. Whenever the logger updates, the mean reward of l_{steps} steps and its standard deviation is recorded.

6.3 Results & Discussion:

The results shown in Figure 3 is generated by recording the average rewards gain of 5 episodes (l_{ep}), and plots with rolling mean of 4, which means the points in the figure are the average rewards gain of 20 episodes, recorded by the stats wrapper.

As showed in Figure3, due to the limited time of this project, a maximum number 1000 is used to train and for comparing the result. Despite the MountainCar environments, both 4 algorithm seems to converge within the limit training steps. As described the task of these 3 environments generates valuable rewards instantly after the agent taking the actions. Hence, in these environments does, a good exploration and exploitation strategies is not required to train the agent.

In Acrobot result, the ICM and Wasserstein network converges slower than Double DQN and ϵ-greedy DQN. Firstly, for ICM network an environment dynamic is required to train to generate the intrinsic rewards, this will extend the learning stages. Second, for Wasserstein DQN, since the uncertainties if backpropagates towards the network, will slowing down the training.

In MountainCar environments, both DQN and DQN with ϵ-greedy exploration fails to solve the mountain car problems, shows that either greedy or ϵ-greedy exploration is not sufficient to train in sparse rewards environment. In contrast, both ICM and Wasserstein-DQN converges within the limiting training episodes. However, it is hard to justify which algorithm is better because the number of training episodes is insufficient. While the rewards of ICM increases at approximately episode 50, the Wasserstein DQN starts at approximately episode 300, with average rewards increases steeper than ICM.

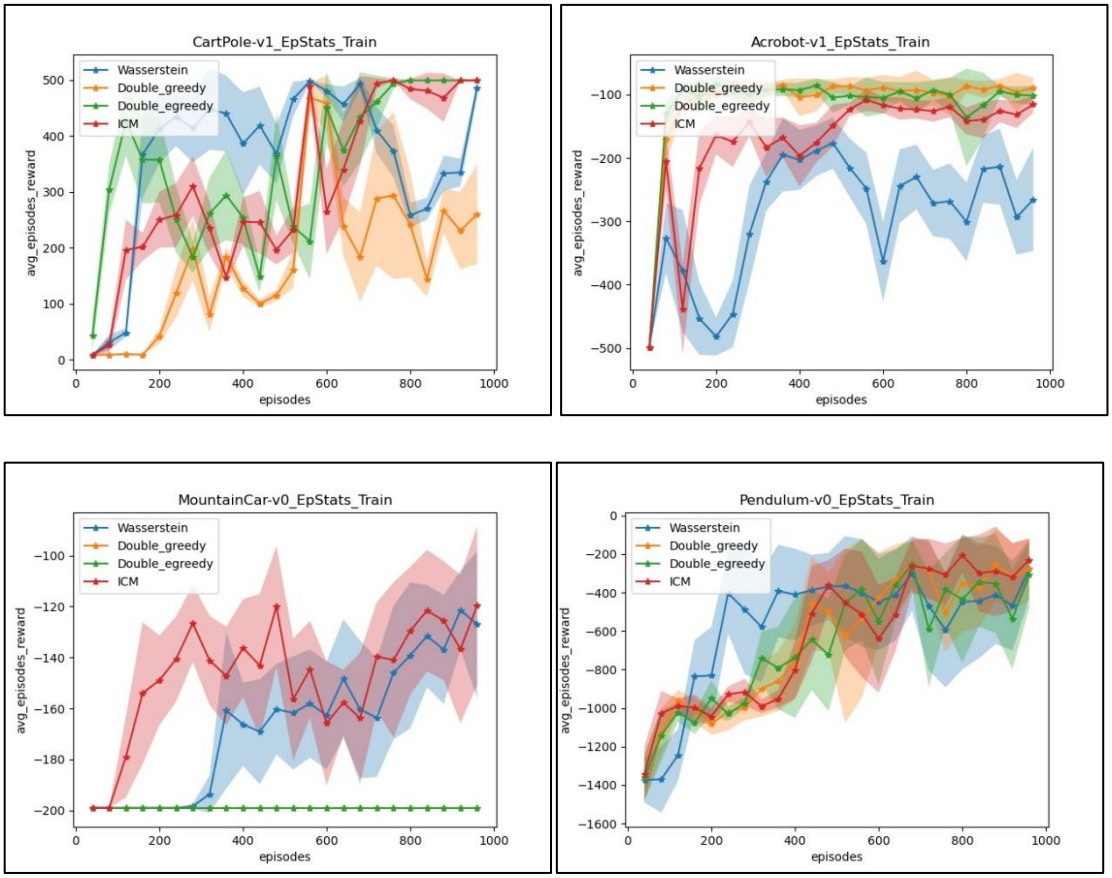

Figure 3: The average rewards every 5 episodes plots during the training of four algorithms in environments of OpenAI gym classic control problems. Wasserstein DQN, Double DQN, Double DQN with ϵ-greedy exploration policy, Curiosity-driven exploration respectively.

Comparing the Wasserstein DQN to ICM model, firstly, it is a model-free reinforcement algorithm with the same structure as DQN, avoids learning the environment dynamic model. This may be more efficient compared to ICM network if an agent is trained in a complex environment, eg. Atari games with graphical input. However, this could not be verified without the experiments.

7. Conclusion & Future Work

In this project, we compare the Wasserstein Q learning with state-of-the-art DQN learning and the curiosity driven exploration DQN. The results shows that Wasserstein Q learning could not only learns in dense reward environments but also in more complex environment, suppress the state-of-the-art algorithm who get stuck at local minima because compared to the large network agent, the reward is sparse and too small, that the gradient generates by the reward is too small for agent to escape from the local minima.

The experiments of Atari games on both ram and graphical inputs have been constructed. The agent playing Atari game is the same as classic control if using ram input, but the experiment has not completed since the initialization is required for the agent. If use RGB inputs, a convolutional neural network is required at the top of the agents.

Due to the limit of time, the number of experiments is not enough, the future work is to implement these algorithms in Atari games. However, it is confidence to have conclusion from the classic control experiments that extensive experiments in extensive environments the performance of Wasserstein DQN will outperforms to DQN. Besides, in complex environments the training time of curiosity driven network with no double longer than Wasserstein DQN given the same number of steps.

Overall, Wasserstein Barycenter is a good way to generate a more confidential values for updates in DQN, which encodes the uncertainty and propagate backward to the deep neural network, enables the state-of-the-art DQN to have a good exploration strategy in RL fields. At the same time, it is model-free which require less computational resource compared to most of the good exploration network.

8. Bibliography:

Auer, P. (2002) *Using Confidence Bounds for Exploitation-Exploration Trade-offs*, *Journal of Machine Learning Research*.

Barto, A. G., Sutton, R. S. and Anderson, C. W. (1983) 'Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems', *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13(5), pp. 834–846. doi: 10.1109/TSMC.1983.6313077.

Bellemare, M. G. *et al.* (2016) 'Unifying Count-Based Exploration and Intrinsic Motivation', *Advances in Neural Information Processing Systems*. Neural information processing systems foundation, pp. 1479–1487. Available at: http://arxiv.org/abs/1606.01868 (Accessed: 2 July 2021).

BELLMAN RE and ZADEH LA (1970) 'DECISION-MAKING IN A FUZZY ENVIRONMENT', *Management Science*, 17(4). doi: 10.1142/9789812819789_0004.

Brockman, G. *et al.* (2017) *OpenAI Gym*.

Dearden, R., Friedman, N. and Russell, S. (1998) 'Bayesian Q-learning'. Available at: www.aaai.org (Accessed: 26 August 2021).

Jaderberg, M. *et al.* (2018) *Human-level performance in first-person multiplayer games with population-based deep reinforcement learning*. Available at: https://youtu.be/dltN4MxV1RI. (Accessed: 13 October 2019).

Landau, L. D. and Lifshitz, E. M. (1980) 'Landau L.D. & Lifschitz E.M.- Vol. 9 - Statistical Physics part 2.pdf', *ZAMM Zeitschrift f*, pp. 603–603.

learnables/cherry: A PyTorch Library for Reinforcement Learning Research (2019). Available at: https://github.com/learnables/cherry (Accessed: 26 August 2021).

Lillicrap, T. P. *et al.* (2015) 'Continuous control with deep reinforcement learning', *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR. Available at: https://arxiv.org/abs/1509.02971v6 (Accessed: 25 August 2021).

March, J. G. (1991) 'Exploration and Exploitation in Organizational Learning', 2(1), pp. 71–87. Available at: https://www.jstor.org/stable/2634940?seq=1&cid=pdf- (Accessed: 26 August 2021).

Metelli, A. M., Likmeta, A. and Restelli, M. (2019) 'Propagating uncertainty in reinforcement learning via wasserstein barycenters', *Advances in Neural Information Processing Systems*, 32(NeurIPS).

Mnih, V. *et al.* (2015) 'Human-level control through deep reinforcement learning', *Nature 2015 518:7540*. Nature Publishing Group, 518(7540), pp. 529–533. doi: 10.1038/nature14236.

Moore, A. W. and Moore, A. W. (1990) 'Efficient Memory-based Learning for Robot Control'. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.2654 (Accessed: 26 August 2021).

Ng, A. Y., Harada, D. and Russell, S. (1999) 'Policy invariance under reward transformations : Theory and application to reward shaping', *Sixteenth International Conference on Machine Learning*. doi: 10.1.1.48.345.

Pathak, D. *et al.* (2017) 'Curiosity-driven Exploration by Self-supervised Prediction'.

Russo, D. J. *et al.* (2018) 'A Tutorial on Thompson Sampling', *A Tutorial on Thompson Sampling*, 11(1), pp. 1–96. doi: 10.1561/2200000070.

Silver, D. *et al.* (2016) 'Mastering the game of Go with deep neural networks and tree search', *Nature 2016 529:7587*. Nature Publishing Group, 529(7587), pp. 484–489. doi: 10.1038/nature16961.

Sutton, R. S. (1996) 'Generalization in Reinforcement Learning', *Advances in Neural Information Processing Systems 8*, 8, pp. 1038–1044.

Thomas, L. C., White, D. J. and Puterman, M. L. (1995) 'Markov Decision Processes.Markov Decision Processes: Discrete Stochastic Dynamic Programming.', *The Journal of the Operational Research Society*. doi: 10.2307/2584317.

Thompson, W. R. (1933) 'On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples', *Biometrika*. JSTOR, 25(3/4), p. 285. doi: 10.2307/2332286.

Wang, S., Jia, D. and Weng, X. (2018) 'Deep Reinforcement Learning for Autonomous Driving', *[RecSys2018]Proceedings of the 12th ACM conference on Recommender systems*, pp. 95–103. Available at: https://arxiv.org/abs/1811.11329v3 (Accessed: 25 August 2021).

9. Appendix:

9.1 Classic control environments:

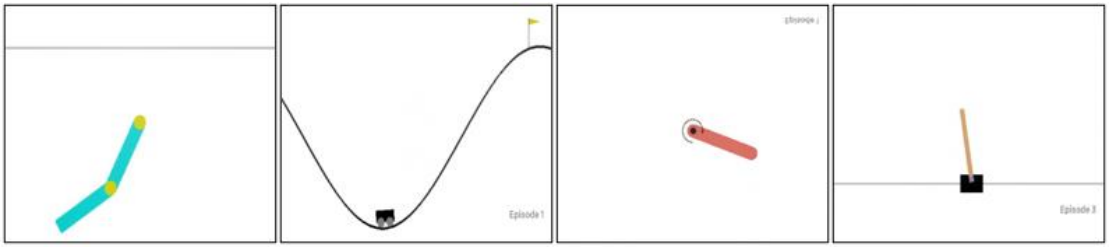

*Figure 4: the screenshots of Acrobot, MountainCar, Pendulum and Cartpole respectively (**Brockman et al., 2017**).*

1. Acrobot:

   The acrobot system includes two joints and two links, where the joint between the two link is actuated. Initially, the links are hanging downwards, and the goal is to swing the end of the lower link up to a given height of the grey line in the picture (Sutton, 1996).

2. MountainCar:

   A car is on a one-dimensional track, positioned between two “hills”. The goal is to drive up the mountain on the right to the yellow flag. However, the car’s engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum (Moore and Moore, 1990).

3. Pendulum:

   The inverted pendulum swings up problem. The goal is to swing the pendulum to stay at upright corner of the picture.

4. Cartpole:

   A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1, 0, -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over, ends if the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center (Barto, Sutton and Anderson, 1983).

9.2 Steps stats of classic control training:

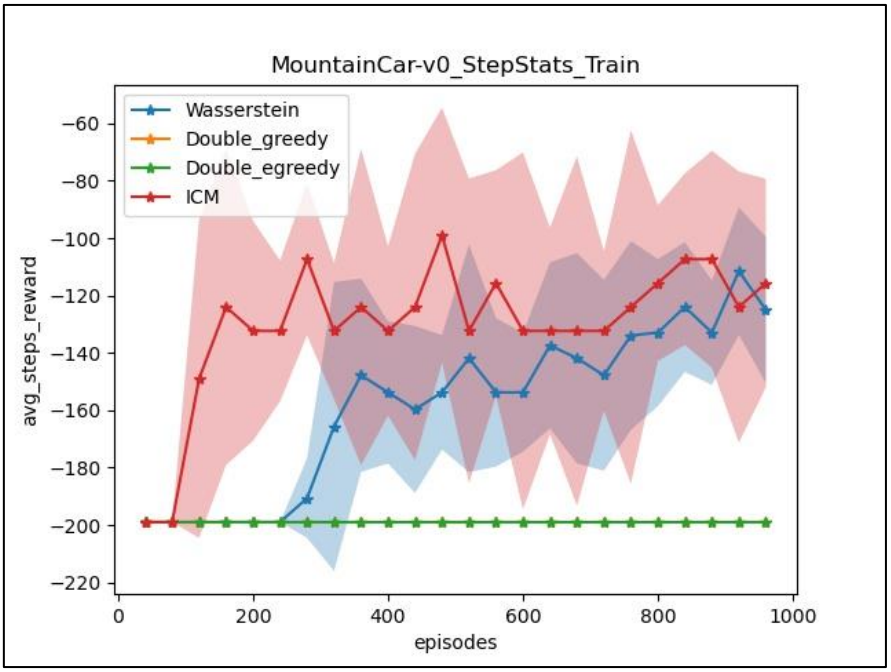

Figure 5

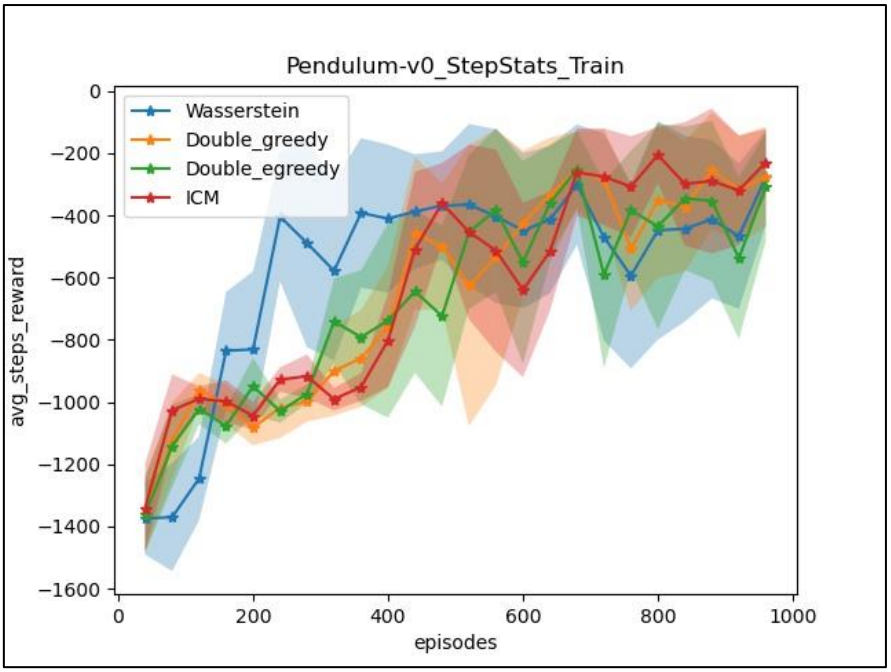

Figure 6

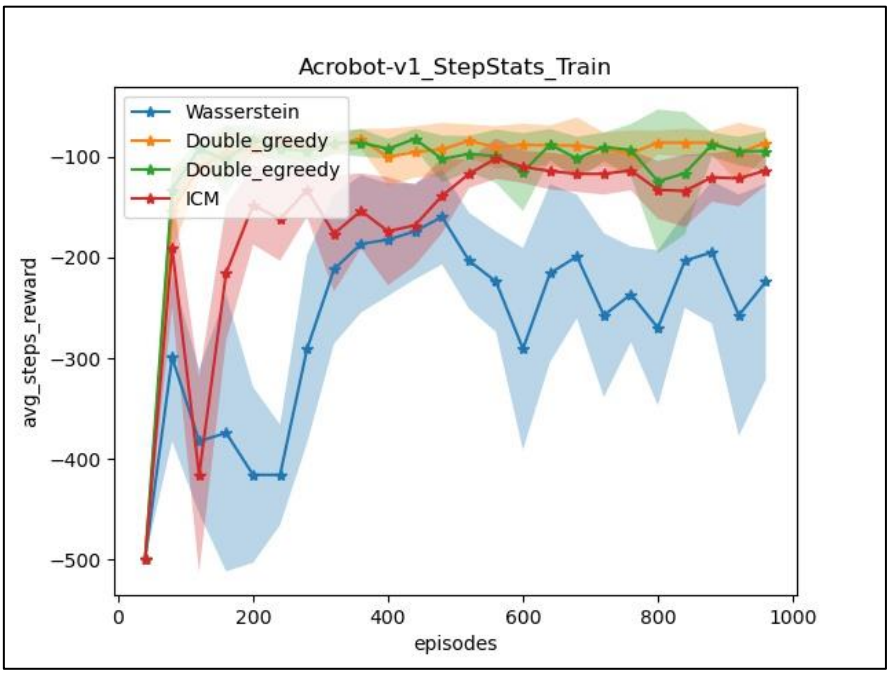

Figure 7

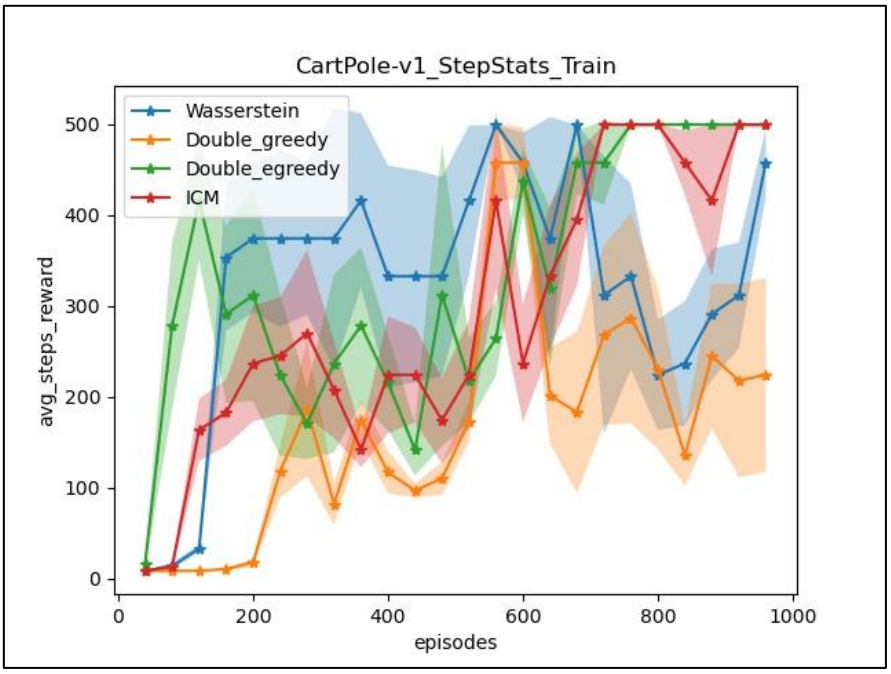

Figure 8