PROJECT REPORT

PROJECT: APPLYING DEEP REINFORCEMENT LEARNING

Student: Nguyen Phuoc Gia Huy

Supervisor: IIMA Hitoshi sensei

I. Introduction

Firstly, I would like to express my sincere gratitude for Professor Iima's enthusiastic guidance and the dedication of the lab members at the Kyoto Institute of Technology. I am also grateful to Ho Chi Minh City University of Science (HCMUS) and KIT for providing the opportunity for an exchange study at KIT.

This project is my work in the internship term at KIT – Kyoto Institute of Technology. There are two tasks in this project:

- Task 1: Practice deep Reinforcement learning. Apply deep reinforcement learning methods such as DQN, A3C, Rainbow, PPO, and TRPO to an environment. Use a library such as PFRL and TensorForce.
- Task 2: Apply deep reinforcement learning methods to the traveling salesman problem.

For the first task, I apply basic methods of DRL by PFRL- Pytorch Framework for Reinforcement Learning to an environment. The environment chosen for learning is Pong - one of the Atari games.

II. Description

1. DRL – Deep Reinforcement Learning

Reinforcement Learning lets the agent learn how to act based on the environment state to maximize the expected long-term rewards.

This learning method is often used in various fields such as game theory control optimization, multi-agent systems, swarm intelligence, and statistics.

Combine Reinforcement Learning and Deep Learning we have a new research field - Deep Reinforcement Learning. In Reinforcement Learning, we have a problem of computational agent learning (policy) to make decisions. To solve the problem, we incorporate deep learning that allows agents to make decisions from unstructured input data without manual engineering of the state space. Now we can take in very large inputs (e.g.: pixels of each frame in a video game) and decide what actions to perform to optimize an objective (gain game score).

In this project, I implement 5 Deep Reinforcement Learning methods:

- DQN: Deep Q-Learning Network
- A2C: Advantage Actor-Critic
- Rainbow: Combining Improvements in Deep Reinforcement Learning
- PPO: Proximal Policy Optimization Algorithms
- TRPO: Trust Region Policy Optimization

2. *Applied Environment*

The Environment applied is Pong - one of Atari games, from [2] gym atari:

- Number Action: 6 (NOOP – FIRE – RIGHT – LEFT – RIGHTFIRE - LEFTFIRE)
- Observation Space: (210,160,3) - 210x160 RBG pixels
- observation value: 0-255
- Rewards: Get a point for getting the ball passes the opponent's paddle, and lose a point if the ball passes your paddle. Who gains 21 points first wins the game (return done=True).

In this project, I use Atari environments downloading from Atari Roms and use ALE to set up and emulate the environment [1].

3. *Framework PFRL*

PFRL is a PyTorch-based deep reinforcement learning library that implements various DRL algorithms in Python.

Install using pip python: ***pip install pfrl***

Github: https://github.com/pfnet/pfrl

4. *Source code:*

My source code refers to pfrl library examples [3].

Folder tree:

- **Atari**: consists of methods code, model saving files, and results for the Pong game (task 1).
  - **Code**: consists of methods code.
  - **Checkpoint**: consists of model-saving files.
  - **Results**: consists of results when training files.
- **TSP**: consists of applied code to solve the Traveling Salesman Problem (task 2).
  - **Code**: consists of methods code.
  - **Checkpoint**: consists of model saving files.
  - **Results**: consists of results when training files.
  - **Dataset**: consists if TSP dataset.

I have been uploading source code, results, and project reports to GitHub as follows link: https://github.com/npghuy/DRL

III. Task 1 - Practice

Task 1 requires to implementation of 5 methods: DQN, A3C, Rainbow, PPO, and TRPO. But there are some errors with Asynchronous in the PFRL library, so I change from A3C to A2C. In other to easily compare methods, I set programs that will stop training the processes after approximately 60,000 seconds.

1. *DQN method*

DQN (Deep Q-learning Network) [4] is the most famous and classical method of Deep Reinforcement Learning. This combines Q-learning of Reinforcement Learning with Deep Learning to compute Q-values.

With 60073 seconds, DQN had trained 2030 episodes:

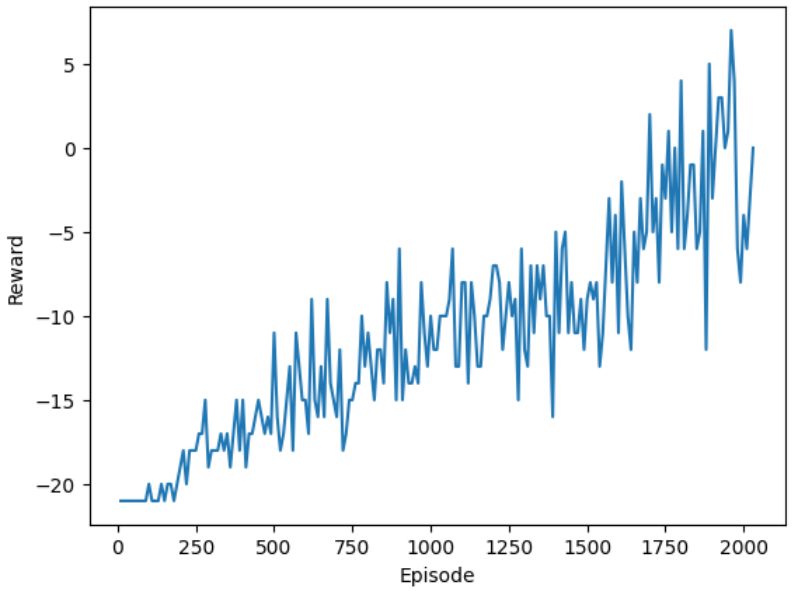

Figure 1: The chart of the Total Reward after each 10 training episodes using the DQN method.

The DQN method runs quite slow but the learning rate is quite good, and the fluctuation range is quite small. There's nothing more to expect from one of the most basic methods.

2. *A2C method*

A2C, Advantage Actor Critic (A2C) [6], combines 2 types of Reinforcement Learning: Value-Based and Policy-Based. And instead of learning from Q-values, it learns from Advantage values.

With 60002 seconds, A2C had trained 14480 episodes:

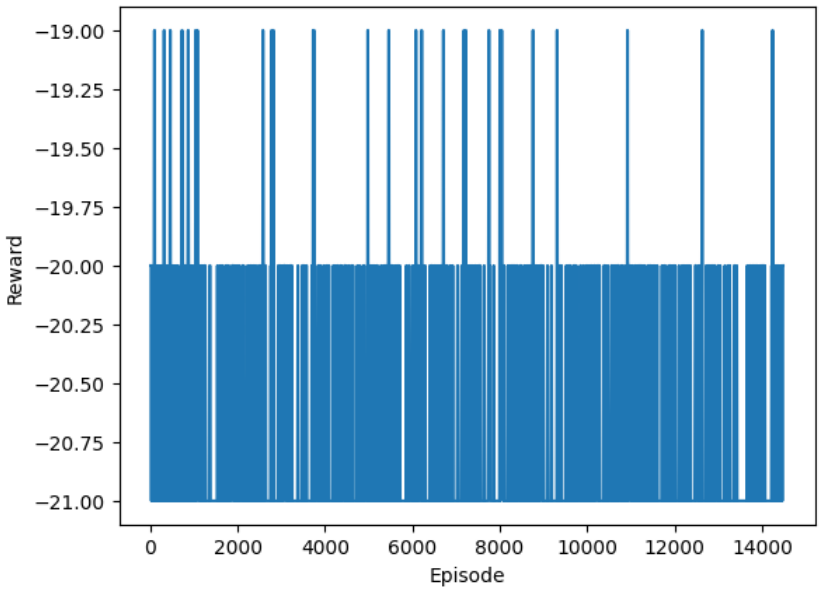

Figure 2: The chart of the Total Reward after each 10 training episodes using the A2C method.

The A2C is quite not suitable with "Pong". It nearly learns nothing from the environment. The good thing only that it runs each episode very fast. I had tried applying A2C to "CartPole" - the simpler game and it ran well, but it seems that it is not good in Pong.

3. Rainbow method

Rainbow [7] is an extension of DQN that combines several improvements into one single agent. It uses Double Q-learning, Prioritized Experience Replay, dueling networks, multi-step learning, distributional reinforcement learning, and noisy linear layers for exploration. Rainbow might be the most powerful method of the 5 methods. I have tried to implement Rainbow to 'CartPole', but it is too simple to apply. Rainbow needs some environment more complex with a big input value size.

With 60037 seconds, Rainbow had trained 6280 episodes:

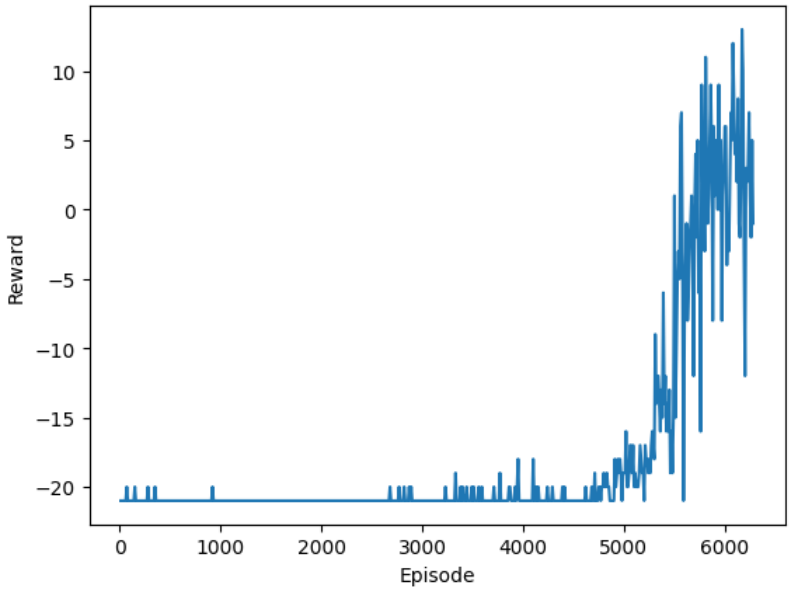

Figure 3: The chart of the Total Reward after each 10 training episodes using the Rainbow method.

Rainbow takes much less time to run each episode than DQN. In early episodes, it lets us think it learns nothing but instantly increases so much, and the fluctuation range after that is not large.

4. PPO method

PPO, Proximal Policy Optimization [8], is a policy gradient method that alternates between sampling data and optimizing a "surrogate" objective function. It has some advantages of TRPO (Trust Region Policy Optimization), but it is much simpler to implement and has better sample complexity.

With 60042 seconds, PPO had trained 4760 episodes:

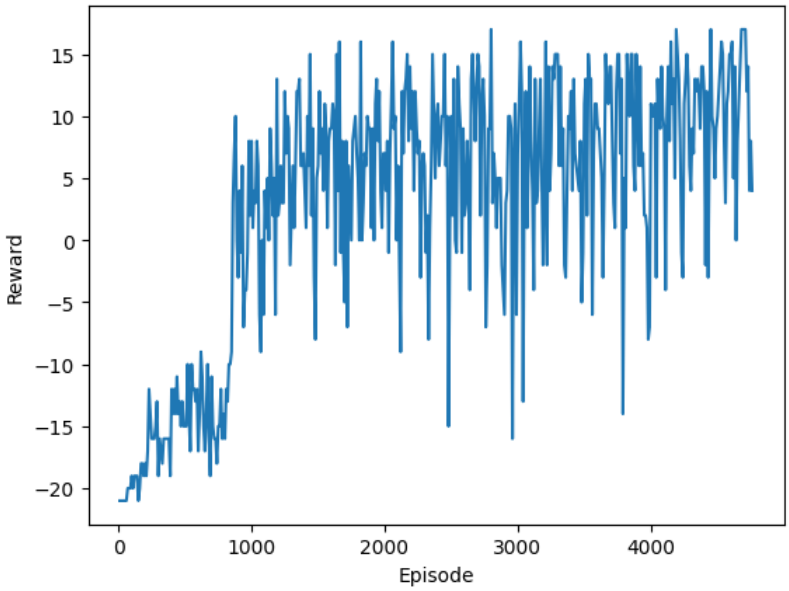

Figure 4: The chart of the Total Reward after each 10 training episodes using the PPO method.

With the same training time, the number of episodes of PPO is smaller than Rainbow. But it learns well very early but the fluctuation range is quite large

5. *TRPO method*

TRPO, Trust Region Policy Optimization [9], is a policy gradient method that avoids parameter updates making the policy change too much with a KL divergence constraint on the size of the policy update at each iteration.

With 60041 seconds, TRPO had trained 4670 episodes:

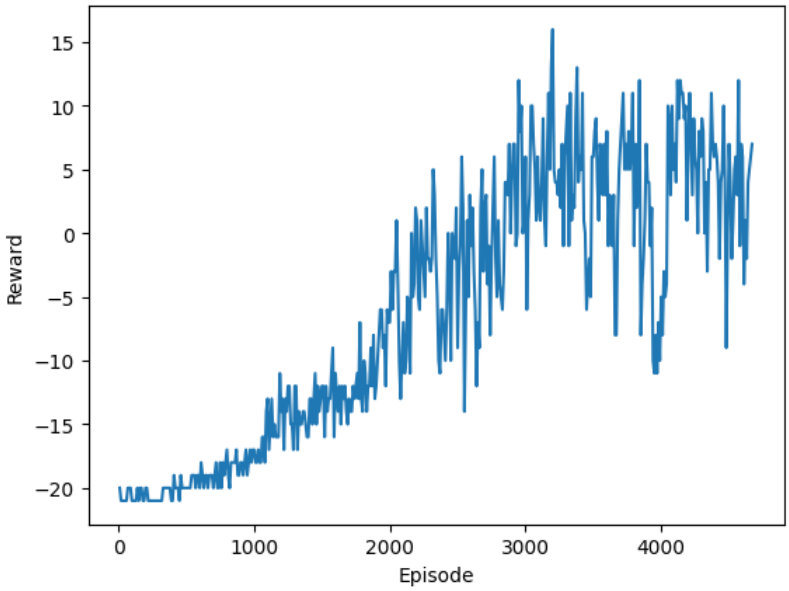

Figure 5: The chart of the Total Reward after each 10 training episodes using the TRPO method.

The learning rate of TRPO is quite the same as PPO, but it increases more stably and the fluctuation range seems smaller than PPO.

6. *Summary*

For the Pong game, if there have short time to train, we could pick TRPO or PPO and the agent can play well (same as human). But there is enough long time for training, we should try the rainbow, it is such a powerful method. After instantly strongly increasing the reward, it becomes a good learner that makes the agent play better the human.

DQN is a such basic method that, we can see DQN in the other 4 methods. And Rainbow is made from several DQN improvements.

I had trained 10000 episodes of Rainbow. It took me 118473 seconds (~1 day 9 hours).

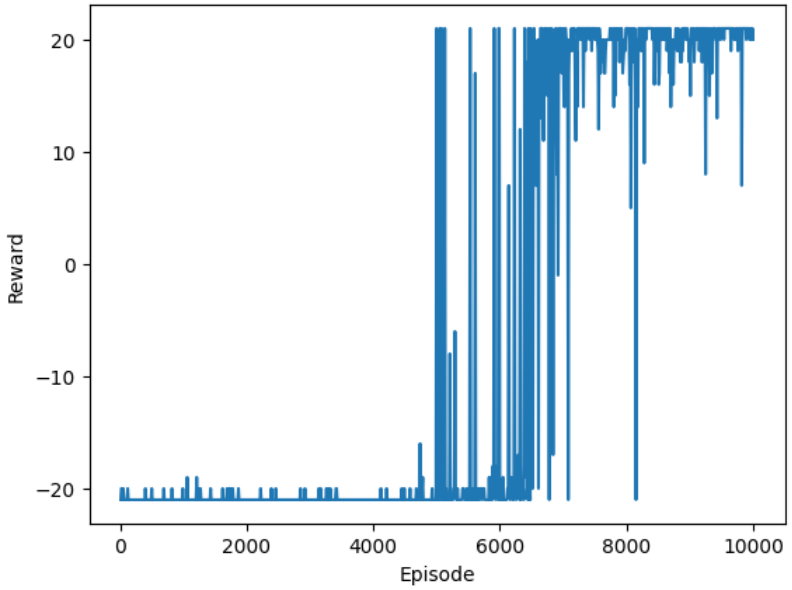

Figure 6: The chart of the Total Reward after each 10 training episodes using the Rainbow method (10,000 ep).

IV. Task 2 – Application

1. *Describe the problem*

The Traveling Salesman Problem, also known as the shortest Hamilton cycle, gives n cities positions and we need to find the shortest path for the Salesman to travel all cities and come back.

The dataset of the problem, I took from this link. To not need to compute the distance between each pair of cities, I use 'the intercity distance table'. I chose dataset 'p01_d.txt', which has 15 cities.

2. *Implement*

For solving Traveling Salesman Problems (TSP), I try to apply the DQN method. The idea is I look at the problem like a game, where the Salesman can start at any (random) city and the target is going as much as different cities with the lowest distance. That means when the agent moves

to a different city it gains points and the shorter the distance is, the more points the agent could get.

Which the dataset 'p01_d.txt', which has 15 cities, I set 1,000,000 episodes for training and it took 32,659 seconds.

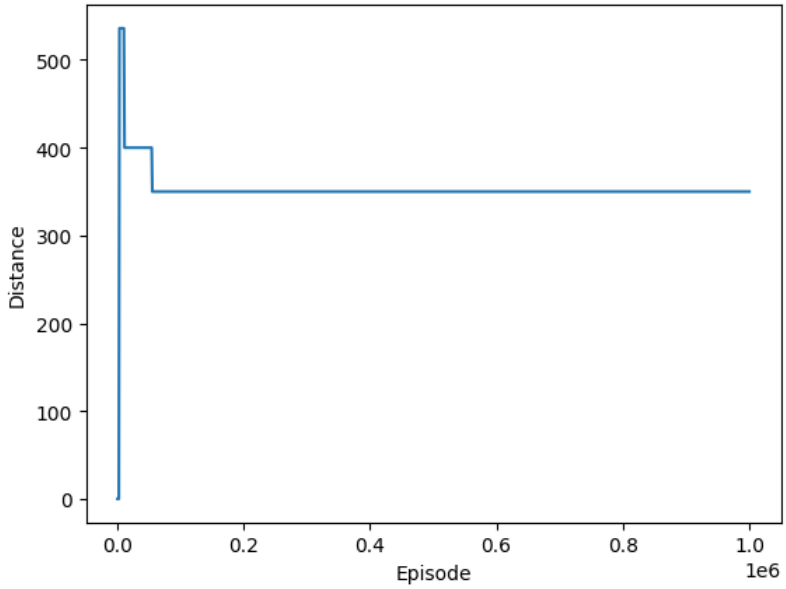

Figure 7: The chart of the shortest distance after each 1000 training episodes using the DQN method.

It is easy to get the short distance but hard to optimize the result after that. The shortest distance of the dataset is 291, and DQN returns 350 after training. It only can find local mins, and do not know if that min is global or not.

V. Summary

Besides I have still not really finished the application task, Deep Reinforcement Learning is a promising research field, especially in games and robotics. It can be learned by itself from environment interaction and can compute large input sizes with deep learning.

In the future, I might improve algorithms in application tasks by changing methods or ways to solve them.

PFRL is a good library for doing Reinforcement Learning tasks, but there are still some errors so I could not use some functions or I do not know the right way to use it.

VI. Reference

[1]. ALE: https://github.com/Farama-Foundation/Arcade-Learning-Environment

[2]. Pong environment: https://www.gymlibrary.dev/environments/atari/pong/

[3]. PFRL: https://github.com/pfnet/pfrl

[4]. DQN: Playing Atari with Deep Reinforcement Learning: arXiv:1312.5602

[5]. A3C: Asynchronous Methods for Deep Reinforcement Learning: arXiv:1602.01783
[6]. A2C: https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f
[7]. Rainbow: Combining Improvements in Deep Reinforcement Learning: arXiv:1710.02298
[8]. PPO: Proximal Policy Optimization Algorithms: arXiv:1707.06347
[9]. TRPO: Trust Region Policy Optimization: arXiv:1502.05477