Reinforcement Learning for Adaptive Traffic Light Control: Progress Report

Yu Yanlin 2140365

November 2023

1 Introduction

With the widespread use of automotive, traffic flows are getting heavy among the world, traffic congestion has become increasingly severe. Faced with this challenge, Transport management center needs innovative and dynamic traffic control systems to reduce the impact of congestion between regions. Despite various proposed solutions, including machine learning and urban planning, traffic congestion remains a significant problem.

This project aims to utilize reinforcement learning algorithms to regulate multiple traffic signal phases, optimizing traffic flow within regions. Our system can react in real-time, alleviating traffic congestion. To enhance the system's intelligence, the system incorporates technologies such as Memory Palace and Phase Gate[7]. This comprehensive approach, considering various traffic scenarios, makes our traffic control system more flexible and efficient.

2 Current Progress

2.1 Methodology

To achieve the main goal of this project, plan-driven and agile methods will be used, this can easily help the project is always on progress and complete objectives step by step. This also performs well to Supervisor or other support member understand the progression. Due to some unpredictable situation, plan-driven may not fit that well, agile then can help with catch up with the plan.

Q-learning: $Q(s,a) = Q(s,a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a))$[3]

To build the control system visualisable and self-adaptive traffic light, a

lot of tools might be used, reinforcement learning algorithm, python and GitHub is mainly used in this project. Research also part of this project, before producing the control system, research provides information about reinforcement learning for adaptive traffic lights, involve its validation. SUMO may also be used as a implementation for the Simulator and Environment.

Algorithm, Deep Q-Network(DQN) is applied, which is one approach to approximate the Q-learning algorithm using neural networks to estimate the value function[1], used to help traffic light make decisions. In DQN, there is a replay buffer with several stored experiences, which will be selected randomly and used for training evaluate network parameters[2]. This empirical replay mechanism is very effective in reducing the correlation between the data, thus making the training more stable. For some basic part in this model, Q-learning is also feasible here. This Algorithm part is the main part of the model, metrics, value function and reward will be designed here. Algorithm will come out come out value function by using value iteration method. The reward of Value function was mixed by several features, Waiting Time of Vehicles, Queueing length of vehicles, Number of Vehicles passed, Time allow Vehicle pass and action of Traffic Light. However due to the time of each phase of traffic light is fixed in Simulator, Time related part was removed from consideration, Reward function now consist with Queueing length of vehicles and Number of Vehicles is waiting.

Python is the mainly programming language used in this project. It uses a simplified syntax with an emphasis on natural language. Algorithms are easier to be implemented by python since libraries in python are powerful enough. Pytorch is used, it has powerful capabilities in building and training deep learning models. Computer-generated simulations are simply imitation of a situation or a process[6], it performs what decisions made by agents and helps to improve the control system. In case, this will also be developed in python base with SUMO by library traci.

SUMO (Simulation of Urban Mobility) is an open-source traffic simulation tool used to simulate urban traffic flow. It is employed for researching and evaluating traffic planning and control strategies. The simulator will be built based on SUMO packages.

Research. To successfully build up the control system with visualisation, research is necessary. GitHub and Google Scholar will provide a lot of information relates project, both code and publication. From GitHub, code related potential issue can be found a solution. Code reading helps code more readable to people other than me. Publication reading provides general advice to the whole project, idea can be raised after reading and analysis.

2.2 Objectives

The objective of this project is to optimize the urban traffic system based on different time flows, and consist of several sub-goals. The primary goal was to create an adaptive traffic light control system that responds in real-time to varying traffic conditions, effectively reducing congestion, and improving overall traffic flow. The real-time Adaptive Traffic Control System will be implemented by MultiAgent System (*known as M.A.S*), since on real life situation, traffic system in city is controlled by many different traffic lights, adding M.A.S can make them collaborate with each other, therefore optimization of the traffic flows between regions can be reached. Now the primary goal has slightly changed through consideration. Primary goal now becomes create Adaptive Traffic Light in single crossroads with 1 traffic light, control with Deep Q-Network(DQN) algorithm. Since this project focuses more on apply reinforcement learning algorithm, the Computer vision part will be ignored here.

- The first milestone of the project is building up a simulator and environment of road situation which will be used in algorithm test later. The simulator will first be 1 intersection, then later it can generate cars and with more intersections. (ACHIEVED)
- The second goal, come out the algorithm of the control system and train with apply them on the simulator. This section will address the decisions made by single traffic light in response to real-time traffic flow. (ACHIEVED)
- Apply Multi-Agent System (M.A.S) to the control system, enhancing the Inter-Connection of Multiple Traffic Lights, make control to different traffic light.
- Add more complex road situation to the simulator and M.A.S, such as traffic light of footpath and bend.

2.3 Progress

Sub-goal 1 and 2 is roughly achieved. A simulator is built with SUMO, which can contains 1 traffic light with 6 different phases and 12 vehicle lanes, controlled by states in each phase, as the following figure1. Algorithm is designed and implemented with code. With code, Traffic light can make decision on which phase is applied at this moment.To achieve these, tutorials and extra researches have done.

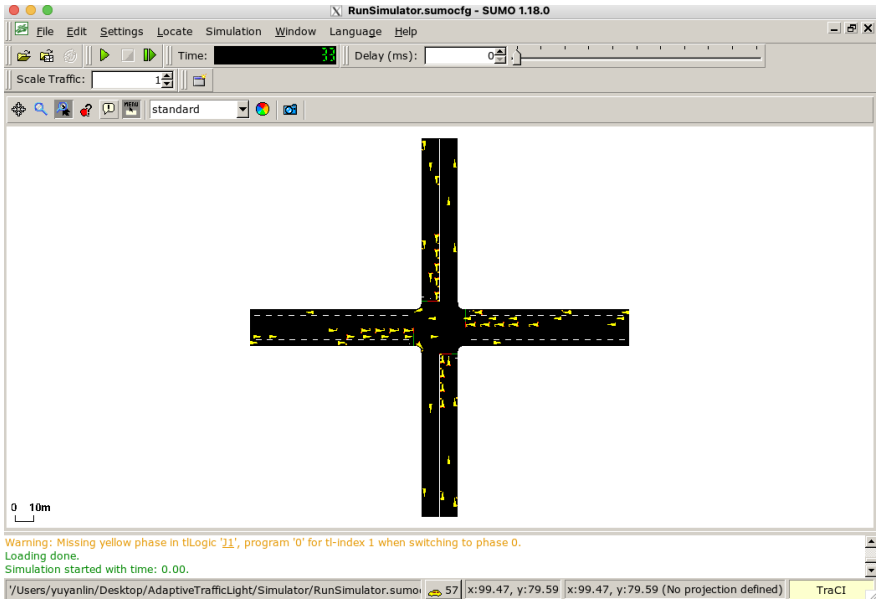


Figure 1: SUMO

Cartpole, a traditional tutorial for reinforcement learning in coding; SUMO document, provide information on how to build route through SUMO and how to link SUMO simulator with python code.In Cartpole tutorial, it aims to stabilise the sticks on the flatbed through reinforcement learning method, as following figure2. A Actor-Critic model was built for Cartpole task and got a nice reward through the model, as figure. Act0r-Critic model is also used to test if the new environment for Adaptive Traffic Light works properly. With SUMO simulator, traffic light will change automatically once the setting time of a phase is reached; Vehicle in SUMO can also be generated automatically once then route of traffic flow is set.

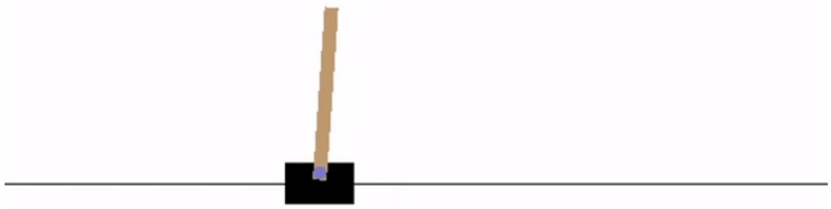
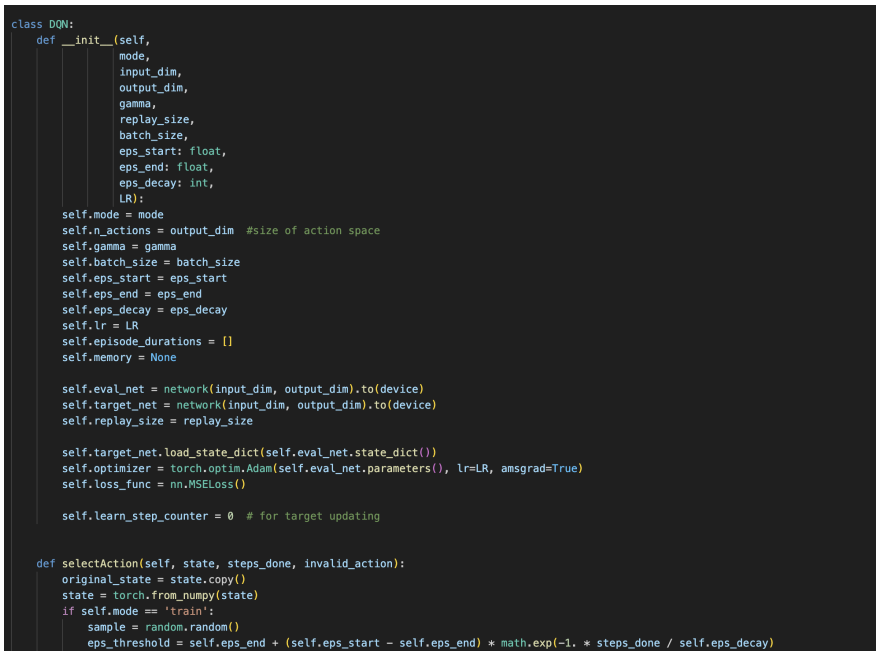


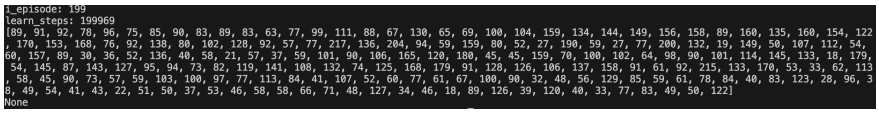
Figure 2: Cartpole

In order to understand how DQN works in application, a paper applying DQN provides ideas for it. Playing Atari with Deep Reinforcement Learning, a paper presented deep learning model to learn control policies on Atari games[4]. Environment and Pseudocode for DQN of Adaptive-Traffic-Light is created after doing research on paper. The validation of environment is test through a rough self-made test to show the environment is feasible and able to give observations and reward of each state. Bad reward is shown since the policy chose is fixed in test but the environment works properly. To apply DQN properly on the environment, understand how this control system work is necessary, pseudocode and case diagram should be designed. Both part of DQN code code and its output are shown in the below figures.



```
class DQN:
    def __init__(self,
                 mode,
                 input_dim,
                 output_dim,
                 gamma,
                 replay_size,
                 batch_size,
                 eps_start: float,
                 eps_end: float,
                 eps_decay: int,
                 LR):
        self.mode = mode
        self.n_actions = output_dim  #size of action space
        self.gamma = gamma
        self.batch_size = batch_size
        self.eps_start = eps_start
        self.eps_end = eps_end
        self.eps_decay = eps_decay
        self.lr = LR
        self.episode_durations = []
        self.memory = None

        self.eval_net = network(input_dim, output_dim).to(device)
        self.target_net = network(input_dim, output_dim).to(device)
        self.replay_size = replay_size

        self.target_net.load_state_dict(self.eval_net.state_dict())
        self.optimizer = torch.optim.Adam(self.eval_net.parameters(), lr=LR, amsgrad=True)
        self.loss_func = nn.MSELoss()

        self.learn_step_counter = 0  # for target updating


    def selectAction(self, state, steps_done, invalid_action):
        original_state = state.copy()
        state = torch.from_numpy(state)
        if self.mode == 'train':
            sample = random.random()
            eps_threshold = self.eps_end + (self.eps_start - self.eps_end) * math.exp(-1. * steps_done / self.eps_decay)
```

Figure 3: DQN code



```
i_episode: 199
learn_steps: 199969
[89, 91, 92, 78, 96, 75, 85, 90, 83, 89, 83, 63, 77, 99, 111, 88, 67, 130, 65, 69, 100, 104, 159, 134, 144, 149, 156, 158, 89, 160, 135, 160, 154, 122
, 170, 153, 168, 76, 92, 138, 80, 102, 128, 92, 57, 77, 217, 136, 204, 94, 59, 159, 80, 52, 27, 190, 59, 27, 77, 200, 132, 19, 149, 50, 107, 112, 54,
60, 157, 89, 30, 36, 52, 136, 40, 58, 21, 57, 37, 59, 101, 90, 106, 165, 120, 180, 45, 45, 159, 70, 100, 102, 64, 98, 90, 101, 114, 145, 133, 18, 179,
 54, 145, 87, 143, 127, 95, 94, 73, 82, 119, 141, 108, 132, 74, 125, 168, 179, 91, 128, 126, 106, 137, 158, 91, 61, 92, 215, 133, 170, 53, 33, 62, 113
, 58, 45, 90, 73, 57, 59, 103, 100, 97, 77, 113, 84, 41, 107, 52, 60, 77, 61, 67, 100, 90, 32, 48, 56, 129, 85, 59, 61, 78, 84, 40, 83, 123, 28, 96, 3
8, 49, 54, 41, 43, 22, 51, 50, 37, 53, 46, 58, 58, 66, 71, 48, 127, 34, 46, 18, 89, 126, 39, 120, 40, 33, 77, 83, 49, 50, 122]
None
```

Figure 4: Returns for each episode in first run

There were difficulties in applying pseudo code to real code during coding. Some useful DQN notations and example code are applied in the Cartpole

exercise on torch tutorials [5]. This provides lots of ideas for building DQN in Python. Debugging is also large part of progress. Since there are 5 different classes doing different in for the control system. 2 for environments, 2 for DQN agent and 1 main class to run environment and agent. Making data structure consistent is also important, if part of code fail, control system will not be runnable.

Algorithm 1 DQN

procedure DQN PROCEDURE
    Initialise replay memory D to capacity N
    Initialise action-value function Q
    $episode \leftarrow 0$
loop:
    if $episode < EndTime$ **then** do
        Initialise the state sequence
        loop:
        if $string(i) = path(j)$ **then**
            With ϵ probability select a random action α_t
            Other selection $\alpha_t \leftarrow \max_a Q^*(S_t, \alpha)$
            Execute action α_t and observe reward r_t
            set next step of sequence, store it in sequence
            store (s, s', a, r) in memory of replay buffer
            Perform gradient descent.

Figure 5 shows how adaptive traffic light works in a single intersection[7]. The offline part is used to collect data at first, this is used to ensure there will be enough data for training in replay buffer. In time $t_1 and t_2$, Traffic light follows policies in policy-network ($evaluate - network$), which is the network update parameters, and make decisions to control traffic light. These (s, a, r, s') data will also be stored in the memory of replay buffer to train network again with random experiences($data\ performed\ previously$)[2]. In the final stage t_3, with the updated policy-network, traffic light will follow its policies to make new decisions. These process stages will run again until the end condition is met, which is simulation time of SUMO in this project.

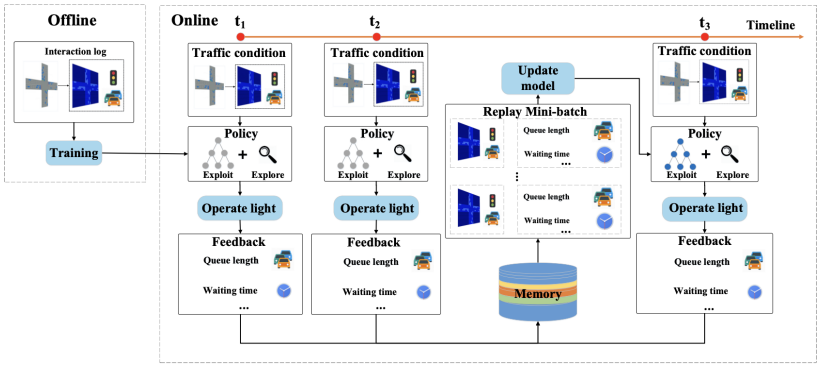


Figure 5: Adaptive Traffic Light System

3 Objective & Further Plan

3.1 Timetable & Plan

Majority of project will base the timetable and goal driven; goals should be accomplished in given time. Weekly job will follow the framework of weekday timetable.

Meeting with supervisor will be on Wednesday Noon every two week and will have meeting with a Phd.Student every week. Meeting will give advice on current state of project, help solve difficulties and push on project progression.

- Goals Best End Time DDL

Goals	Best End Time	DDL
Build Simulator and RL tutorial	End of October	First week of November
RL model	End of November	Mid of December
Add M.A.S to model	End of February	End of February
Apply to Real Road	Not necessary, do it if have enough time	

Generally first 2 goals performs fairly well, Reinforcement learning(RL) model goal meet at best End Time, which is end of November. However, the first goal did not achieve as expected at first. Simulator of the model performs well independently, traffic flows and traffic light control can be seen as expected. However, this can not be seen when applying DQN on simulator, which was unexpected. Later this have been solved. Python code was connected to SUMO but did not use 'sumo-gui' command so SUMO will disconnected by itself and no visualisation when running episode. Then this is fixed, a new variable 'simulation-time' was added into 'reset' in environment for DQN, without this variable, simulator will only run in the first episode, and the following episode can only run without visualisation. In each episode, simulator will run from

step 1 again to show how DQN control the traffic light.

- The table below will contain the information about weekly job.

Monday	Plan this week project state
Tuesday	Research, Tutorial
Wednesday	Weekly meeting, slightly replan
Thursday	Coding follows the plan
Friday	Coding follows the plan
Saturday	Ending weekly plan

3.2 Difficulties

In order to make this control system more general and adaptable, there are some difficulties that have met and need to be implemented in the next phase of the project by the time the project progresses.

During the research, A more general control method for traffic light can be used by using the library Traci(in python). Traffic light phases and states can be setting a more general way by using this method instead of setting phases manually in advance. This method will be implement into the Adaptive Traffic Light control system in second part.

Data for real-life traffic flow may be hard to get, there is limited data presented online, more budget is need if real-life data is needed in the later project. There will be issues on data cleaning as well, since data collected from different sources may have different format, cleaning for making consistency is necessary when doing data analysis and learning.

3.3 Further Plan

In the next step of Adaptive Traffic Light, difficulties met in the first model need be solved as soon as possible. With the progression so far, multi-agent system(M.A.S) will be implemented to the Adaptive Traffic Light as expected. Applying M.A.S can make model more general, suitable to more area instead of single intersection. More algorithms will be developed and apply on the the DQN to build up the M.A.S Adaptive Traffic Light. With M.A.S, Adaptive Traffic Light will connect multiple traffic light in different intersections, which aims to deal with the behavior of each agent and their intersection optimally[8].

More research on M.A.S need to be done in next phase of Adaptive Traffic

Light, understand how MultiAgent collaborate with each other theoretically. Further, more researches on algorithms about MultiAgent, know about how that work and how they can be implemented to the existing Reinforcement learning environment, a pseudocode for M.A.S would be better.

3.4 Ethical Consent

There are some Ethical consents need to be considered in Adaptive Traffic light, such as Data related information, security of self control system and transparency of Adaptive Traffic Light.

For data related issue. Training control system needs large amount of data in real life. Vehicle information and pedestrian information might be recorded. Exposure of location information and behavioral habits needs to be considered. However in Adaptive Traffic Light, all data are got from the SUMO simulator. In this case, data related issue might not be consider at this moment.

For security of this control system. Control system should care about the cybersecurity related issue to avoid hacking by individual citizens. This can lead to traffic jam and even traffic accidents.

For transparency of Adaptive Traffic light. Adaptive Traffic Light is based on reinforcement learning, so there will be some decisions made by control system which may not be explained well by human beings because of the Opaqueness of reinforcement learning. This could lead to lots concerns.

References

[1] Ankit Choudhary. *A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python*. 2023. URL: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/#h-what-is-deep-q-learning.

[2] William Fedus et al. "Revisiting fundamentals of experience replay". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3061–3071.

[3] Sean Michael Kerner. *Q-learning*. 2023. URL: https://www.techtarget.com/searchenterpriseai/definition/Q-learning.

[4] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[5] Adam Paszke. *REINFORCEMENT LEARNING (DQN) TUTORIAL)*. 2023. URL: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html?highlight=dqn.

[6] Terence Shin. *Building Simulations in Python — A Step by Step Walkthrough*. 2020. URL: https://towardsdatascience.com/building-simulations-in-python-a-complete-walkthrough-3965b2d3ede0.

[7] Hua Wei et al. "Intellilight: A reinforcement learning approach for intelligent traffic light control". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 2496–2505.

[8] Marco A Wiering et al. "Multi-agent reinforcement learning for traffic light control". In: *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*. 2000, pp. 1151–1158.