Reinforcement Learning

COMP9417, 2019s2, Assignment

UNSW
SYDNEY

z5096599 Zilu Dong
z5165957 Dong Zhu

1. Introduction

With the development of modern cities, the number of motor vehicles is also increasing rapidly. An important reason for the rapid passage of urban roads is the control of traffic lights. However, traditional traffic lights usually use a method of fixed-time switching to achieve control of the intersection. When there is a significant difference in the traffic volume at the intersection, the signal light may interrupt the heavy flow and allow the minor flow to pass.

In order to reduce the delay of waiting for a vehicle intersection effectively, intelligent traffic light control is essential and a future development direction. At present, many cities have installed weight sensors under the road surface to judge the traffic flow and realize the on-demand switching of the signal lights. But this method is not only complicated to install but also time consuming.

The purpose of this project is to design an intelligent traffic light control system to reduce the delay at lights. In order to simulate the intersection traffic lights, the program will display simulated traffic generation, flow and traffic lights on the screen, and simulate the lights controlling using a fixed change time of 10 time-steps and other two reinforcement learning algorithm to improve them.

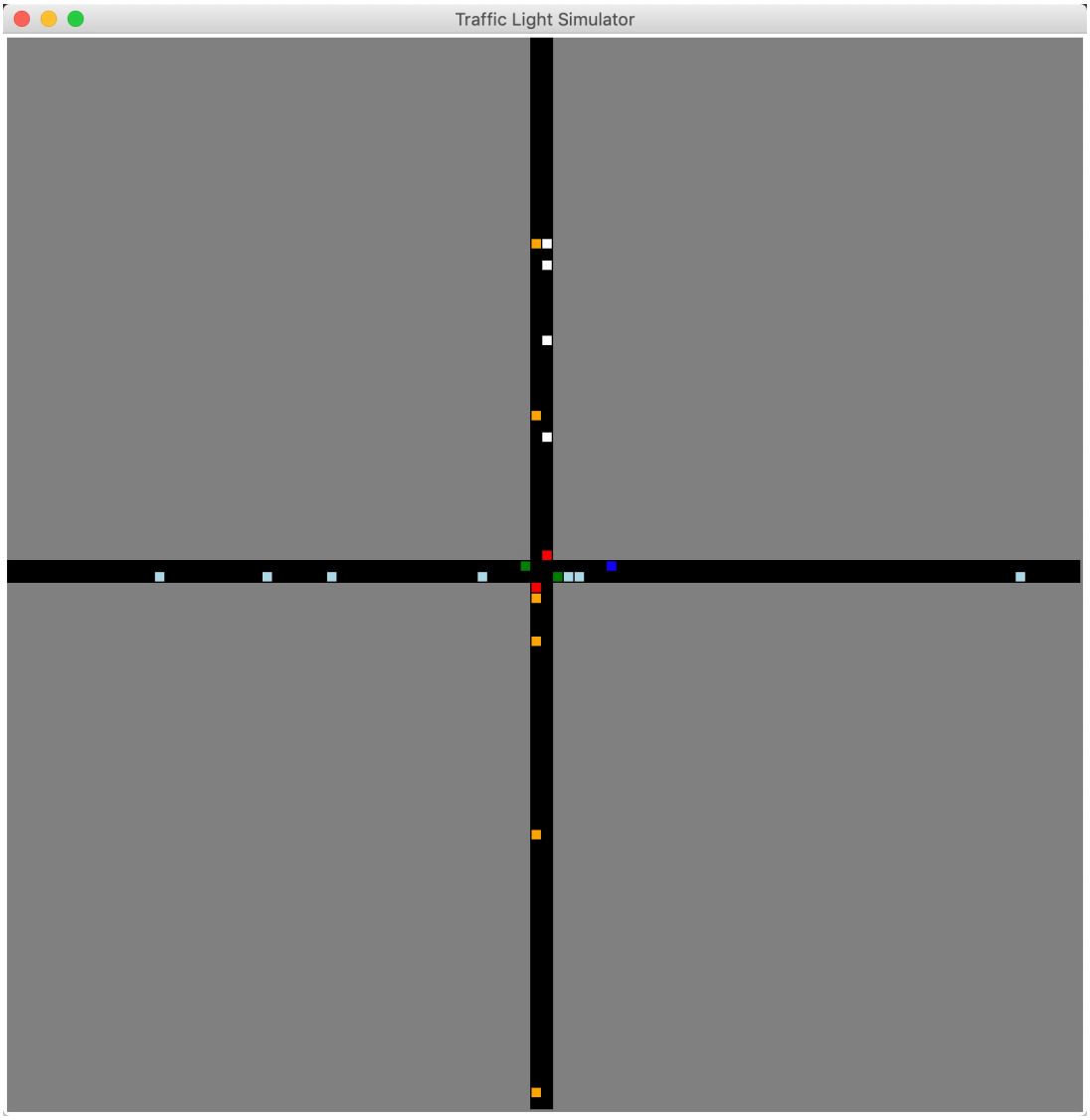


Figure 1. Traffic Light Simulator

2. Related work

In recent years, more and more have begun to work on the research of intelligent traffic lights, and the research of intensive learning algorithms is also widely used. Marco Wiering studied the use of the Reinforcement Learning (RL) algorithm to learn traffic light

controllers to minimize the waiting time of cars in the city (Wiering

2000). If the vehicle begins to saturate at load, the RL system is significantly better than the fixed controller.

For this project, in order to simulate the traffic lights, we used the method of Q-learning to conduct research and analyse its performance by varying the RL parameters and some other features.

2.1 Q-learning

Q-learning (Watkins, 1989) is a learning algorithm f in a controlled Markov domain. It improves the assessment of the specific actions by exploring states, actions, and rewards, and help agents to learn how to act optimally.

In a non-stationary environment, Q-learning is a very effective signal control algorithm (Abdoos 2011). Since different Parametric representations of the action space can implement multiple scenarios, it can be applied to real traffic networks with unpredictable traffic demands. Abdulhai (Abdulhai 2003) pointed out that the key factor in the period is real-time, adaptive control. In the reinforcement learning algorithm, the Q-learning algorithm performs very well on traffic signal control.

QLearning is a value-based algorithm in the reinforcement learning algorithm. Q is Q(s, a) is the expectation that the action ‘a’ ($a \in A$) action can be obtained in the ‘s’ state ($s \in S$) at a certain moment. The system will feedback the corresponding reward ‘r’ according to the action of the agent. Constructing the State and Action into a Q-table to store the Q value, and then select the action that can obtain the maximum benefit according to the Q value.

$$Q(s,a) = Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

The learning process is the update process of Q-table, where α is the learning rate, γ is the discount factor, and r is the future calculated reward. MaxQ(s',a') is the maximum value of all possible actions a' in the next state s'. The selection of action a follows the e-greedy strategy, and the calculation of the target Q value is also calculated according to the action a' obtained by the strategy.
The above is the formula of Q-learning update.

3. Implementation

3.1 The map

The road length can be about 100 units, one unit of road holds one car. The simulator moves the cars one space along the road at each time-step. The cars only travel in only one direction and are removed from the road when they reach the boundary of the simulated region. If a car is stopped by a red light, it cannot move forward. The cars cannot occupy the same space. Other cars therefore queue behind stopped cars.

To start, traffic is flowing two ways on two intersecting roads controlled by a set of traffic lights. Cars enter the road some way from the intersection when current_time % (rnd.randint(1, 5) + 5)==0. Time is just a clock counter that is incremented at each step.

3.2 The parameter

The traffic control simulator program is using Python 3.6.

Default MDP specification and Q-Learning parameters are:
Reward = -1.0 if a car is stopped at a red light on either road, zero otherwise.
Discount factor: γ = 0.9
Learning rate: α= 0.1
Epsilon-greedy exploration = 0.1
Actions: switch or not switch (0 for not switch, 1 for switch)
State = closest car position from intersection for road 1 (0-8, 9 if no cars) X
closest car position from intersection for road 2 (0-8, 9 if no cars X
closest car position from intersection for road 3 (0-8, 9 if no cars X
closest car position from intersection for road 4 (0-8, 9 if no cars X
light setting (0-green, 1 red for one of the roads) X
light delay (0-3)

4. Extension

4.1 Include amber phase in lights

In the original traffic light, only the red and green lights are included. In this case, the switching of the signal is likely to cause the car crash. In order to prevent this, we added Amber signal light. The amber light will illuminate when the signal changes between green and red, and the vehicle in front of the signal will not be able to pass until the other vehicles pass completely through the intersection, then the red light at the other intersection turns green and the vehicle on another road begin to pass. The amber light will lasts for 3 seconds before switching to red.

4.2 Include 2-way traffic

In this expansion we have added two lanes in the opposite direction, and actually, most of the roads will be the same. This expansion will make the simulation closer to real life traffic.

4.3 Test different traffic intensities and generating functions

The parameters generated by the vehicle are set before the simulation starts. The default car generation parameter is current_time % (rnd.randint(1, 5) + 5)==0, and we implement this extension by changing the parameter of rnd.randint(1, 5). This will simulate vehicle congestion at different traffic intensities.

4.4 Vary RL parameters

For reinforcement learning, using different RL parameters will produce different learning effects. The default RL parameter is: discount factor: γ = 0.9, learning rate: α= 0.1, epsilon-greedy exploration = 0.1. In this extension we will test different RL parameters to evaluate its learning effects to find the optimal set of parameters for learning.

4.5 Try different reward functions

The original incentive policy is that if a car is found to stop on the road then reward = reward – 1. The goal of this policy is to punish the occurrence of car stopping. In this expansion we have changed

the reward functions, which is reward = reward - 1 whether the vehicle is stopped or not.

4.6 Try different exploration strategies

There are many algorithms for reinforcement learning, and Q-learning is just one of the off-policy algorithms. In this extension we have added another on-policy algorithm, Sarsa. The comparison can be used to learn the effects of different learning algorithms.

5. Results and analysis

By default, there is one intersection and cars will be generated in 4 directions. The default car generating function is :
generate a car if current_time % (rnd.randint(1, 5) + 5) == 0

The Default parameters used in Q learning is showed in parameter section:

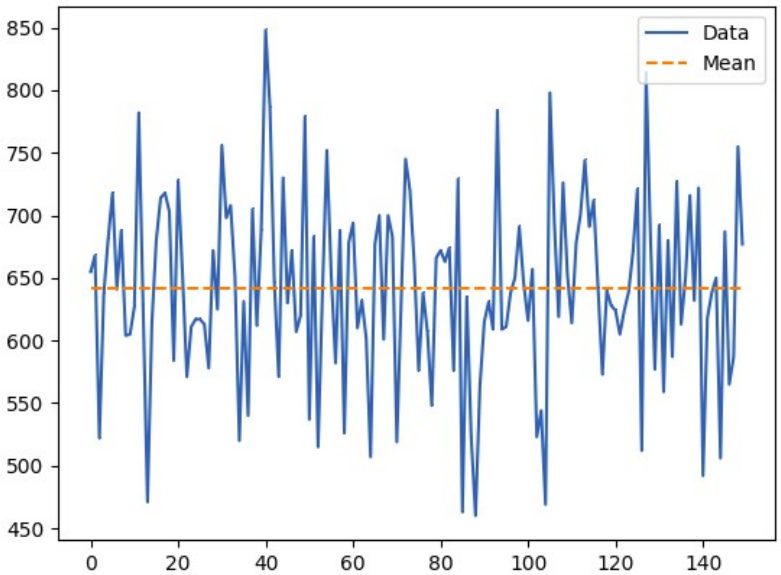
Performance is measured by sum of the number of cars stopped at intersection at one episode, where one episode contains 1000 time-steps.

While varying one parameter, other parameters will stay the same as default parameters.
By default, the Q learning will trained for 150 episodes and then do the testing with 50 episodes.
The results showed in following analysis are derived from the testing data, where the *average car stopped* means the average number in 50 episodes of sum of car stopped in one episode.

5.1 Compare performance measures between fixed switching and Q learning

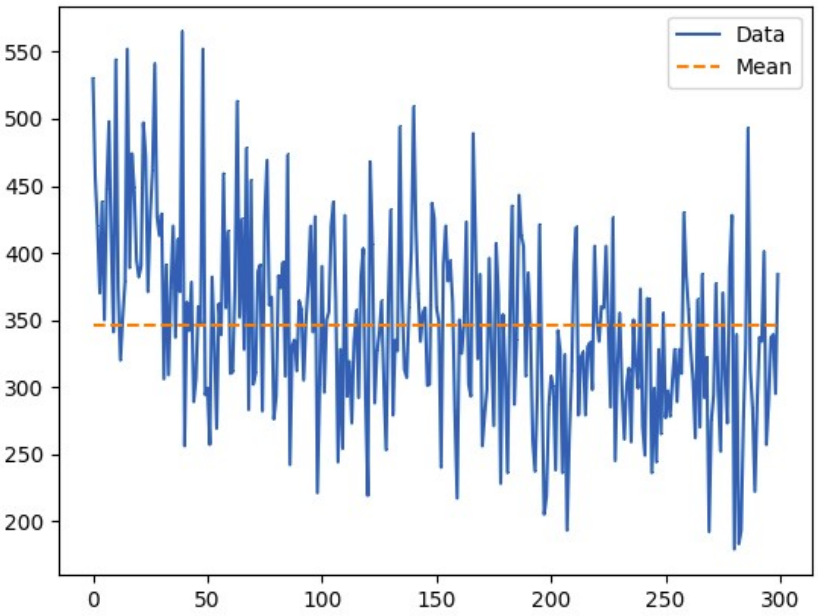
5.1.1 Fixed switching performance

Controls the lights using a fixed change time of 10 time-steps
No learning algorithm is applied in this strategy, therefore the performance is poor.



5.1.2 Q-learning performance

This is the performance of Q learning using Default settings. This the testing of 300 episodes without any training ahead of time. It shows a decreasing tendency of car stopped from episode 0 to episode 100. This is because Q learning algorithm is updating its Q table in the process of testing. Then the performance tends to become more stable since Q table has been updated to near optimal.



5.2 Vary RL parameters

5.2.1 The learning rate alpha α

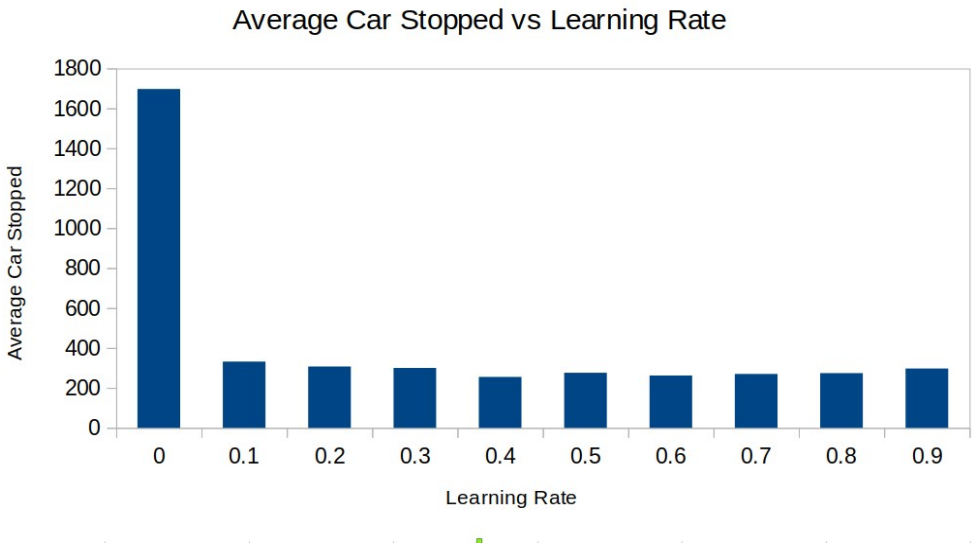


Figure 5.2.1

α determines to what extent newly acquired information overrides old information. According to the figure above, it can be seen that setting α too low will cause the algorithm to care only about the previous result, instead of the new rewards. When we set the α to 0, the performance of the algorithm is the worst, it has hardly learn anything. When the parameter α is set to 0.9, the agent will consider the new reward too much, therefore the performance at this time is not optimal. From the above figure, when the learning rate is 0.4, the result at this time is optimal.

5.2.2 The discount factor gamma γ

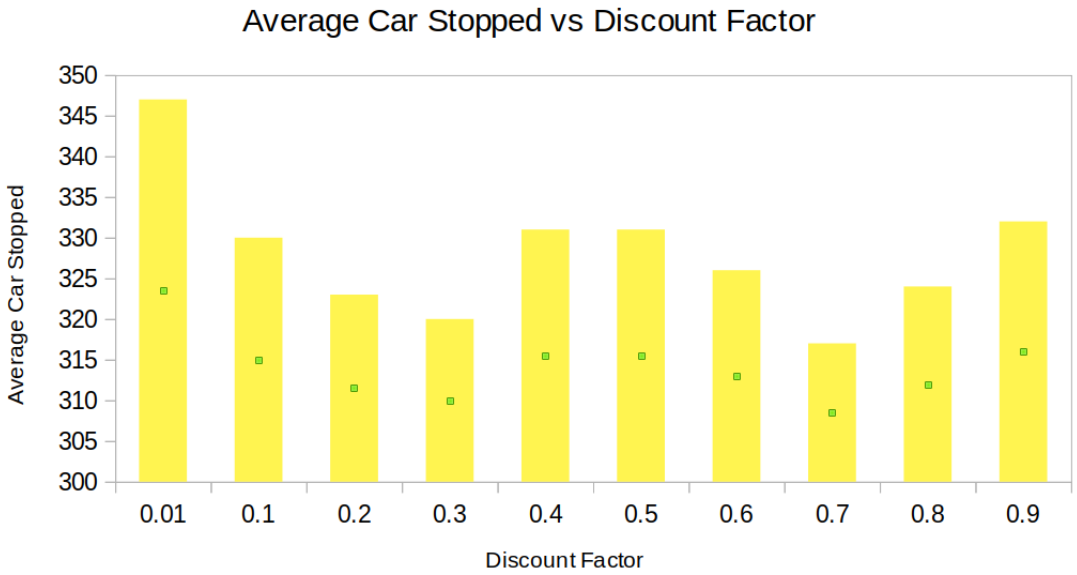


Figure 5.2.2

γ is a factor that considers future rewards and is a value between (0, 1). Generally, we set a little bigger and can fully consider external rewards. When γ is set to 0, the agent will only consider the current reward and ignore the future reward. When γ is set to near 1, the agent will only consider the future reward and ignore the current reward. In both extreme cases, the performance of the algorithm will deteriorate. Therefore, a reasonable discount factor γ could make it optimal. From the figure above, it could be seen that the value of the chart reaches its lowest point when $\gamma = 0.7$.

5.2.3 The exploration factor epsilon

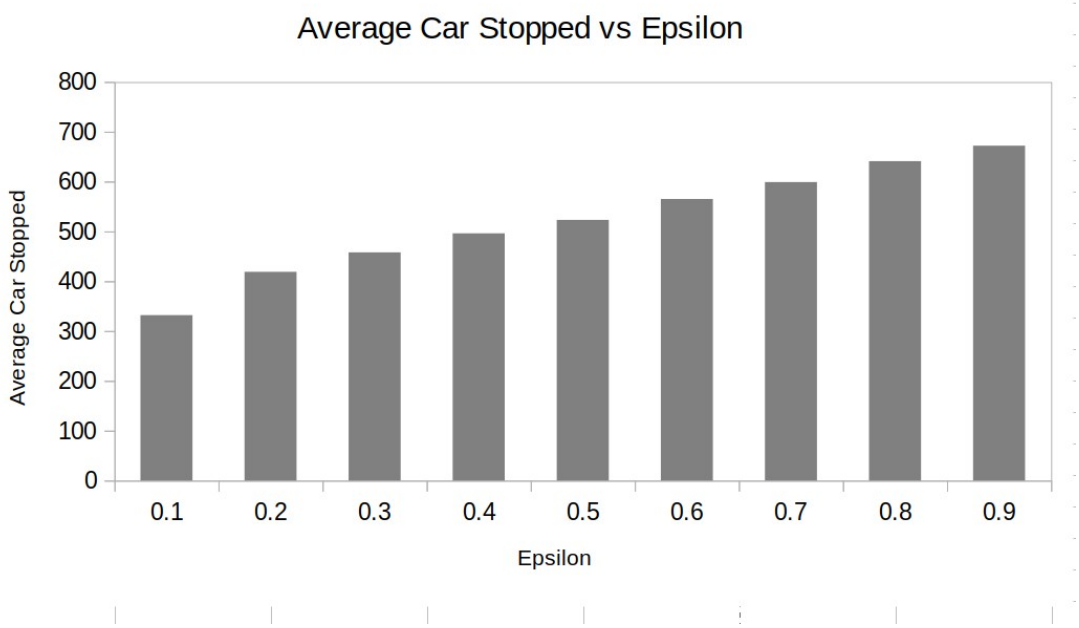


Figure 5.2.3

The exploration factor allows the agent to have a certain probability of randomly selecting the next action instead of the maximum based on the q-table. The discovery factor allows the agent to have a certain probability of randomly selecting the next action instead of choosing the maximum value based on the q-table. When the exploration factor is large, the next choice of the agent is more likely to be random and reduce the performace. From the figure above, it could be seen that the value of the chart reaches its lowest point when epsilon $= 0.1$.

5.3 Test different traffic intensities and generating functions

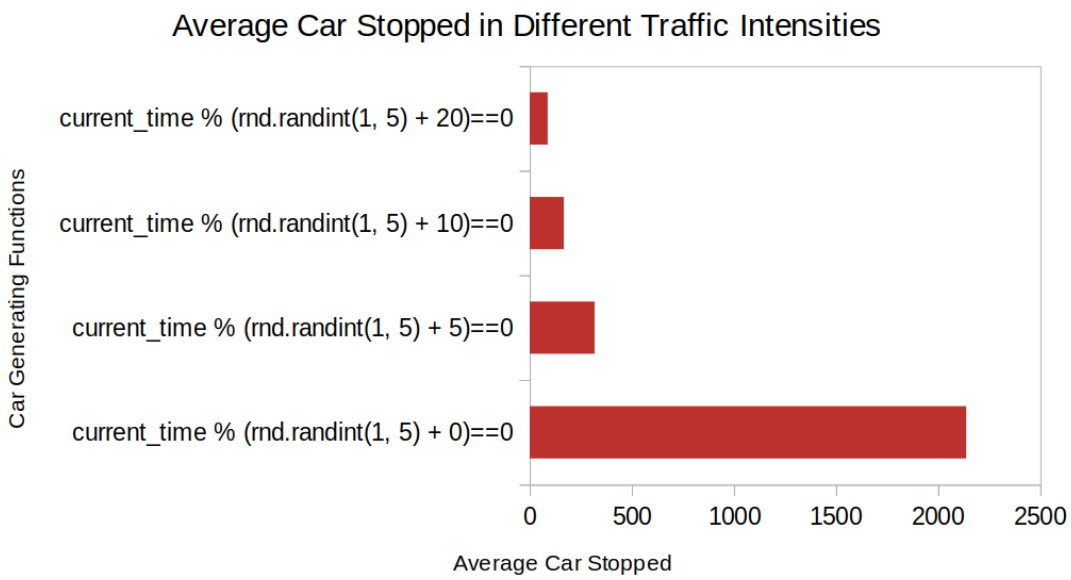


Figure 5.3.1 current_time % (rnd.randint(1, 5) + 0)==0

We tried the following 4 different vehicle generation rates, which starting with the initial current_time % (rnd.randint(1, 5) + 0)==0 to current_time % (rnd.randint(1, 5) + 20)==0. The generation rates is gradually decreasing and we generate the following table.

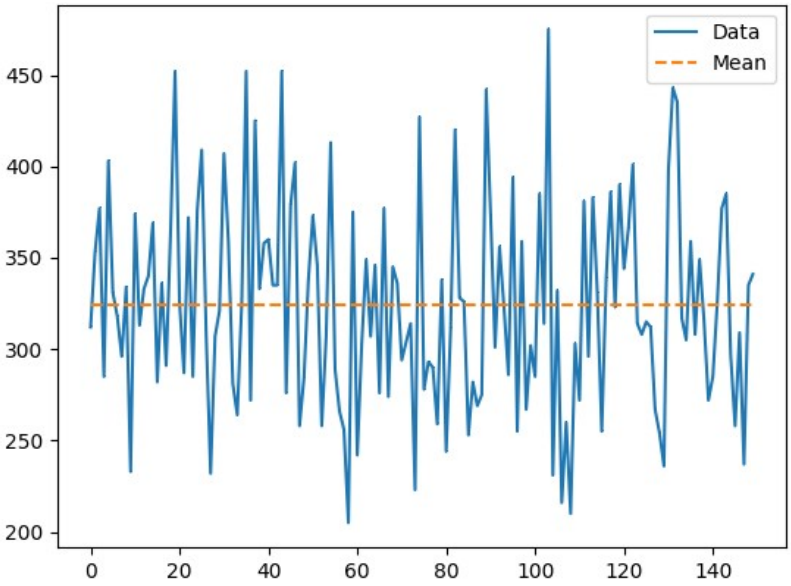
The above chart shows that with the increasing of traffic intensity, the average number of sum of car stopped will also significantly increase.

5.4 Try different reward functions

Figures in this section is made using Q learning with Default parameters. This is the testing data in 150 episodes after trained by 150 episodes. The x-axis is episode and y-axis is the sum of car stopped in one episode.

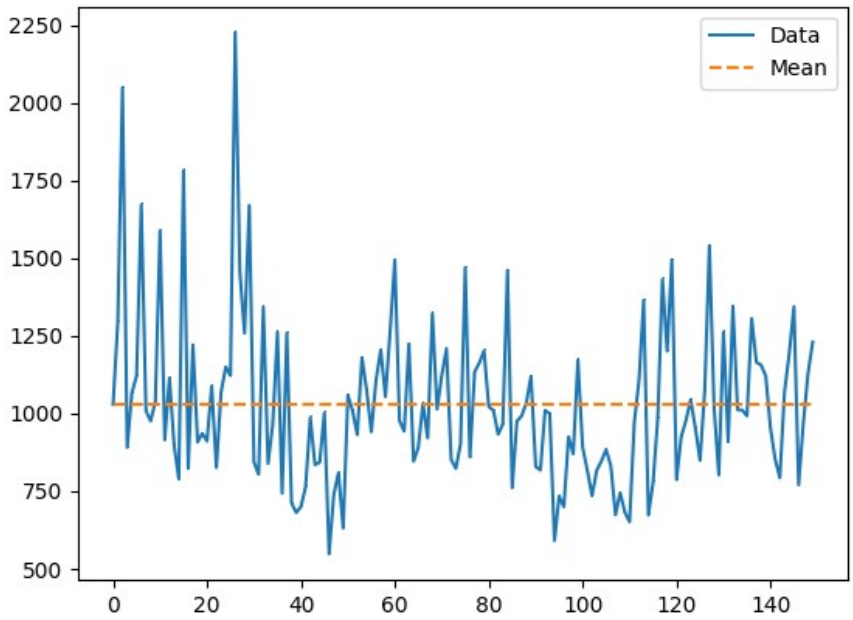
5.4.1 reward = negative sum of number of cars stopped

The reward will decrease by the number of cars stopped at intersection at each timestep.



5.4.2 reward = – 1 if there is any car stopped

The reward will only decrease by 1 if there is any cars stopped at intersection at each timestep.



It shows that using the negative sum of number of cars stopped as reward leads to better performance, since that reward function has a one to one relationship to the performance function and therefore

reflect the performance more efficiently than reward =-1 if any car stopped.

6. Conclusion

In conclusion, the application of Q learning in the traffic light controlling problem can be helpful to solve traffic congestion issue. Although with the increasing of traffic flow the performance of Q

learning will decrease, it can be concluded that **5.2.3 The**

exploration factor epsilonQ learning can reduce the total number of

car stopped at some levels of traffic intensities. Apart from this, the parameters applied in Q learning, such as discount factor and learning rate, will significantly affect the final performance. Therefore, it is important to choose the best parameter combination while applying the algorithm to solve real life problems.

7. References

Wiering M A. Multi-agent reinforcement learning for traffic light control[C]//Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000). 2000: 1151-1158.

Abdulhai B, Pringle R, Karakoulas G J. Reinforcement learning for true adaptive traffic signal control[J]. Journal of Transportation Engineering, 2003, 129(3): 278-285.

Abdoos M, Mozayani N, Bazzan A L C. Traffic light control in non-stationary environments based on multi agent Q-learning[C]//2011 14th International IEEE conference on intelligent transportation systems (ITSC). IEEE, 2011: 1580-1585.

Watkins C J C H, Dayan P. Q-learning[J]. Machine learning, 1992, 8(3-4): 279-292.

Watkins, C.J.C.H. 1989, ‘Learning from delayed rewards’, PhD thesis, University of Cambridge England.

Mnih V, Badia A P, Mirza M, et al. Asynchronous methods for deep reinforcement learning[C]//International conference on machine learning. 2016: 1928-1937.

Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. MIT press, 2018.