Traffic Light Control using Deep Reinforcement Learning

Akshat Gupta (2017014), Prince Sachdeva (2017080), Rudraroop Ray (2017311)

Source Code

This is the link to the github repository that contains our source code

https://github.com/prince17080/RL-Project-2019

Goal of the Project

Traffic is a major problem in our city. We aim to use Deep Q Learning methods to automate traffic signals in such a way that the flow of traffic across intersections becomes more efficient.

Given a state representation of the traffic situation of an intersection, what signal should the traffic light display (for a fixed duration until the state is provided again) in order to ensure traffic movement with lowest overall waiting time?

Reference Papers

Primary reference

- Simulation of traffic light scenario controlled by a Deep Reinforcement Learning Agent - Andrea Vidali

  https://github.com/AndreaVidali/Deep-QLearning-Agent-for-Traffic-Signal-Control/tree/master/TLCS

Secondary

- Traffic Light control using deep policy gradient and value function based reinforcement learning

Novel Contribution by the Team

While the reference paper uses Deep Q Learning to solve the traffic efficiency problem, we employ the Double DQN algorithm. Deep Q Learning is prone to overestimation of action values based on optimistic samples. The Double DQN algorithm succeeds in removing overly optimistic value estimates and should therefore help us find a better policy.

Environment, Agents, Rewards, etc.

Environment: A 4-way intersection with each of the 4 roads consisting of 4 lanes that approach the intersection and 4 that leave it. The agent interacts with the environment using SUMO (Simulation of Urban Mobility) software

Agent: The state for the agent indicates the number of vehicles currently present at the intersection. The action space for the agent is simpler i.e. traffic light configuration: green light activated for some lanes, red light for some others, etc.

Reward: The reward for any action is the difference between waiting times i.e. *Previous Total Waiting Time - Current Total Waiting Time*

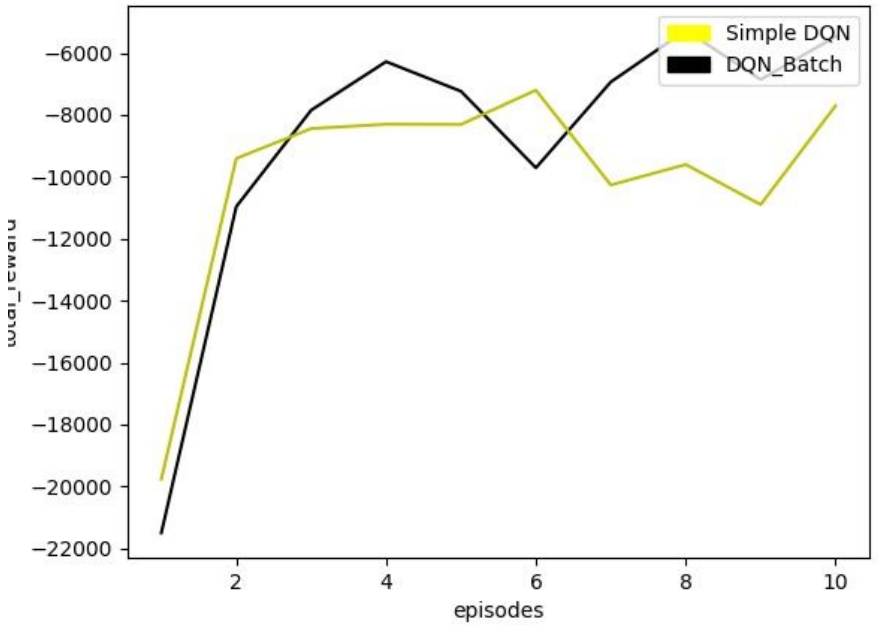
RL Algorithm

Here we will use the Double DQN algorithm to figure out near-optimal state-action values and consequently, near-optimal policies. This means that we will maintain two DQN’s and for each episode, one of these will be randomly selected for *epsilon greedy action selection while the other will be used for updation of values.

*We will employ an epsilon greedy policy to balance exploration with exploitation.

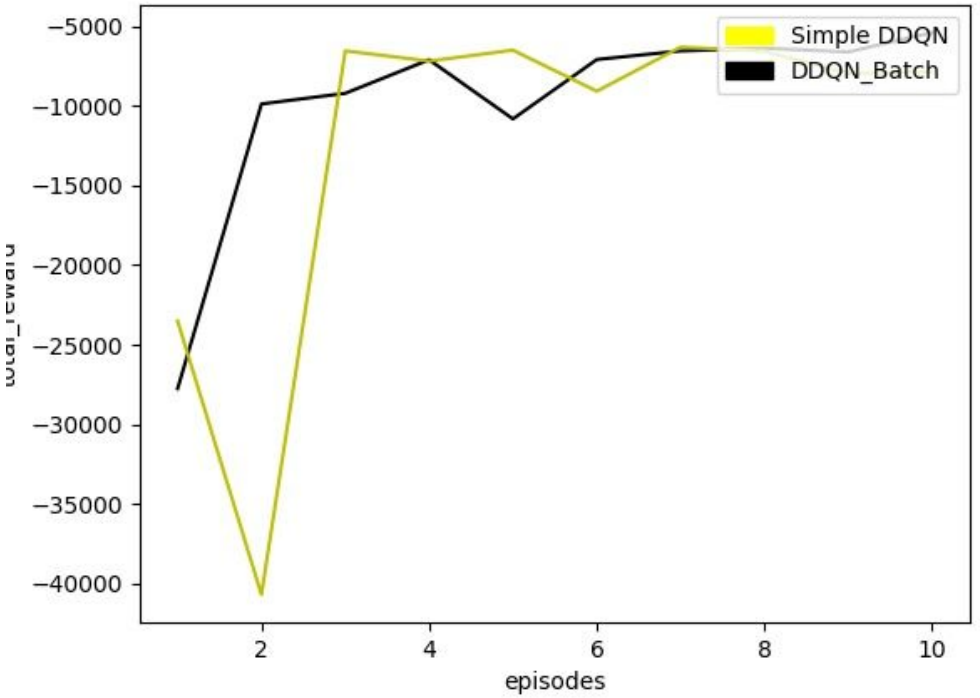
Results - 1

It is seen that cumulative reward for DQN with experience replay is higher than simple online Deep Q learning with no batch training. This is due to higher sample efficiency of batch DQN



Results - 2

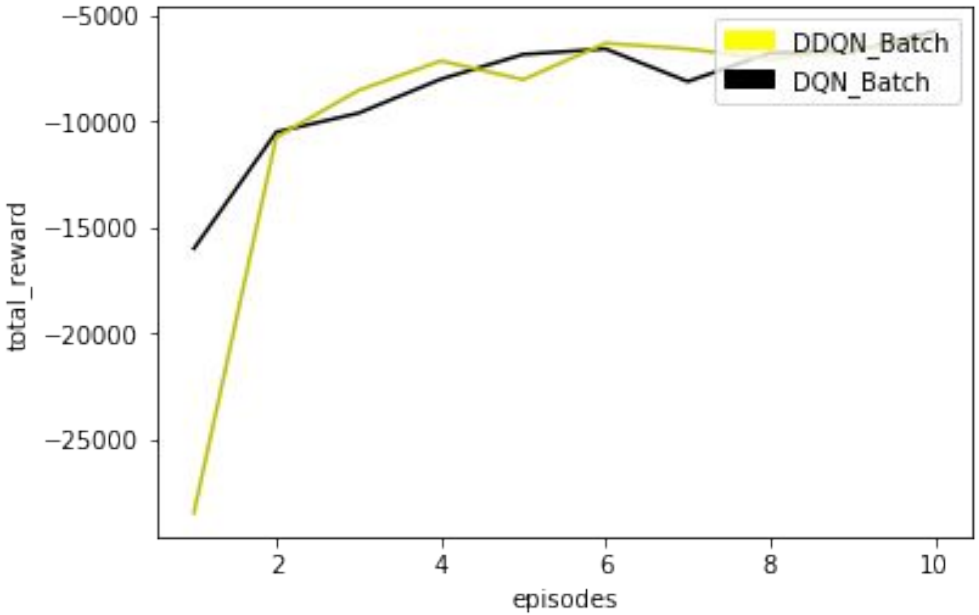
It is seen that Double DQN with experience replay starts improving on its cumulative reward quicker than the Double DQN algorithm **without** experience replay



Results - 3

Both key algorithms show a great initial increase in reward followed by a plateau.

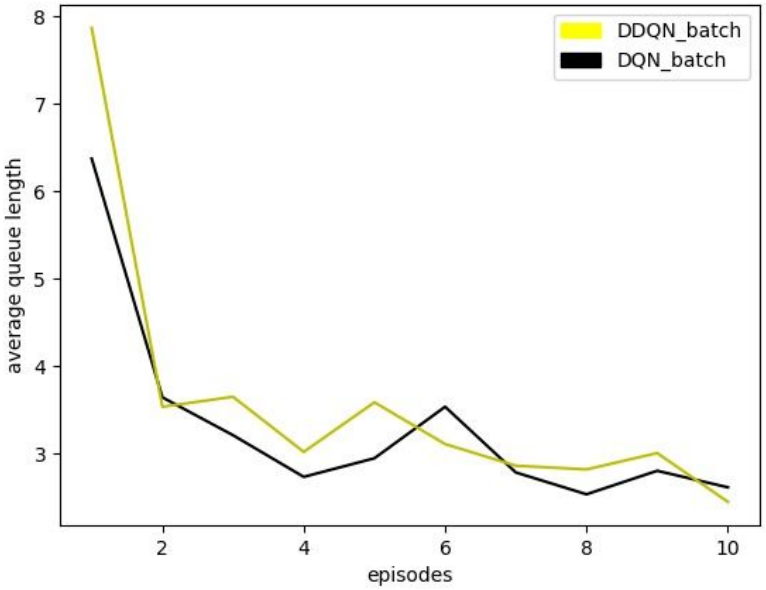
The behaviour of DDQN and regular DQN algorithms is quite similar in this regard. (**Double** DQN algorithm closes the gap quite quickly despite a lagging start.)



Results - 4

The average queue length reduces quickly using both key algorithms.

Queue length is seen to reduce more from the starting episode when we apply Double DQN with experience replay as compared to traditional DQN with experience replay



Thank You