Reinforcement Learning Algorithms With Applications to Traffic Flow Control

CS221 Artifical Intelligence Project, Spring 2020 Final Report

Akshita Agarwal
akshita@stanford.edu

Khalid El-Awady
kae@stanford.edu

Mayank Gupta
gmayank@stanford.edu

I. Introduction

Autonomous systems are a very active area of industrial application, such as in manufacturing, exploration, automation and others. These systems are characterized by a need for decision making under uncertainty. Reinforcement learning (RL) is a very general framework for learning sequential decision making tasks.

In this project we study the class of RL algorithms and their applicability to a real world problem, namely **traffic flow control**. This represents a non-trivial example where RL has been shown to yield results on a real-world complex system [8], [9]. Further, a simulator has been identified that can readily provide input/output data [10].

II. Related Works

Transportation accounts for 28% of energy consumption in the US. Workers spent on aggregate over three million driver years commuting to their jobs [1], with significant impact on nation-wide congestion. Based on 2012 estimates, U.S. commuters experienced an average of 52 hours of delay per year, causing \$121 billion of delay and fuel costs annually [2].

This extreme expenditure in cost, lost time, pollution, and quality of life, has led to a large investment in the study of traffic and its control and creating simulators to model traffic flow and allow the testing of new control schemes. One such simulator that has received acclaim and attention is SUMO – Simulation of Urban MObility, an open source, highly portable, microscopic and continuous road traffic simulation package designed to handle large road networks [3], [4]. We will utilize SUMO as the simulation engine behind our data generation. In addition to SUMO, we will be using Flow, an open-source computational framework for RL and control experiments [5].

The works of [7], [8] were instrumental in demonstrating the applicability of RL to improving traffic flow through agent-controlled traffic lights. Much of the current research in RL control of traffic lights revolves around implementations of deep learning algorithms [9]–[11].

In this project our objectives are to:

- Setup a simple traffic light environment with 2-way traffic flow in SUMO with a typical model of vehicle arrivals,
- Simulate the flow of the traffic through the intersection using a typical static control agent and compute statistics around vehicle waiting time and speed – this will be our baseline,
- Implement a basic RL algoritm to learn an alternative policy for control of the light and compare the resulting vehicle statistics to those of the baseline,
- Explore one or more advanced RL algorithm to attempt to improve performance beyond what was achieved in our first attempt.

In this progress report we present and compare the results of the baseline and initial RL-based approach, and propose our next steps for the modifications we will look at in the balance of this project.

III. Dataset

In SUMO, a user defines a network which represents the elements of the traffic environment to be simulated. In our case the environment is made up of a north-south road met by an east-west road at an intersection controlled by a traffic light. A screen shot of the environment is shown in Figure 1.

Our network represents a pair of local city roads with a speed limit of 35 mph and an enforced minimum gap between cars of 2.5 secs of travel (subject to an assumed car deceleration rate of 7.5 m/h^2). Cars can enter the network from the endpoints of any of the respective four arms of the network at a distance of 300 yards away from the intersection. The car proceeds to the intersection and either passes through it if the light is green in that direction, or waits if the light is red. Once the car passes through the intersection it continues for 100 yards until exiting the network and the simulation.

SUMO injects new car traffic into the network at the beginning of each edge using a uniform distribution with a given mean inter-arrival time while also enforcing our prescribed minimum distance between cars. To create some variation in the flows we have chosen to vary the rate of cars entering each of the arms. The reader might be able to visually discern the difference in traffic flows from the figure (the bottom arm shows much higher density than the top arm for example).

The traffic light experiences so-called phases which represent the light's color in each of the four directions. A typical static light might cycle through phases of red and green, each of length 30 seconds each with an optional short transition period when the light that was green turns yellow. A typical simulation might last for 200 seconds and involve around 100

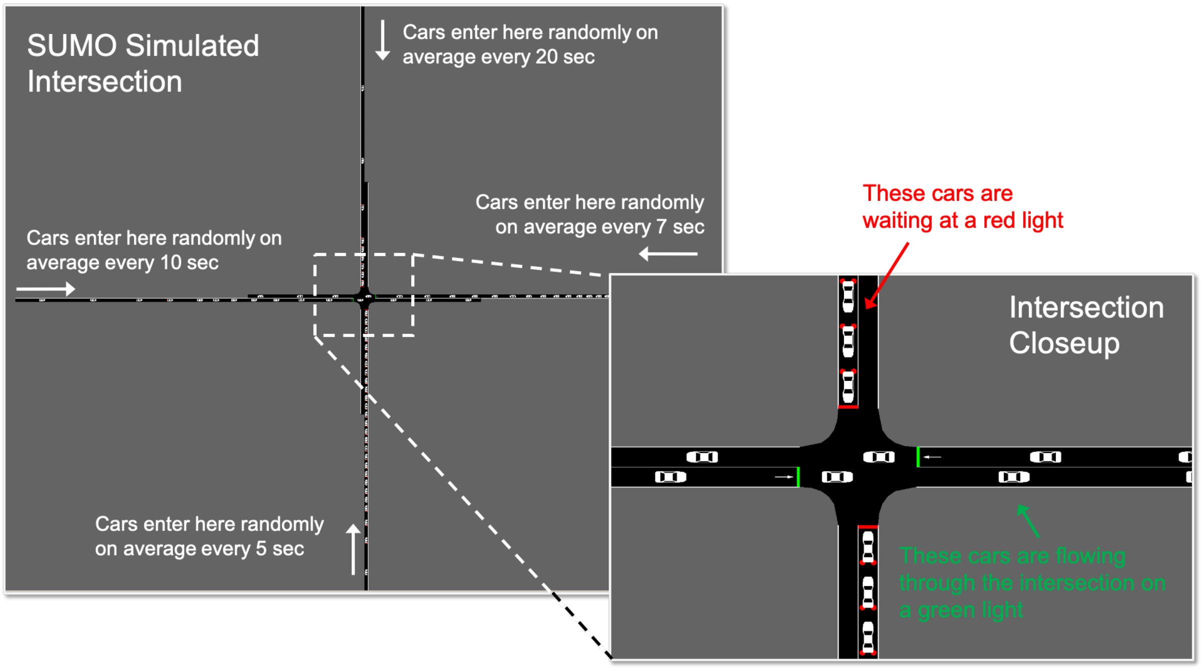

SUMO Simulated Intersection
Cars enter here randomly on average every 20 sec
Cars enter here randomly on average every 7 sec
Cars enter here randomly on average every 10 sec
Cars enter here randomly on average every 5 sec
These cars are waiting at a red light
Intersection Closeup
These cars are flowing through the intersection on a green light

Fig. 1. Screen shot of the SUMO simulation environment of a simple intersection with a traffic light.

cars. A car traveling at around 10 mph that does not stop at the intersection will pass through the network in about 70 sec.

The output of the simulator is a file of data called an "emissions" file that comprises a set of time samples for each vehicle in the simulation with a set of vehicle parameters at the designated simulation sample output interval. The main data points of interest to us in the emissions file are the 'waiting' time and 'speed' parameters. The waiting time represents the cumulative time (in secs) the vehicle has been waiting as of that time sample. Speed is a measure of the instantaneous speed of the car at the time sample.

Figure 2 shows a sample of the vehicle data for one vehicle (id: flow_00.10). In this vehicle's case, it enters the simulation at time 40 with a speed of 10 mph heading west to east. The vehicle eventually decelerates to a stop at the traffic light by time 65. It then waits a cumulative 31 seconds at the intersection before accelerating and leaving the simulation at time 113 going at 13.99 mph.

IV. Baseline

Our baseline will be a traffic light that employs static phases. There are three main phases that our light assumes. These are:

1) green in the north-south direction, red in east-west direction for a duration of 30 sec,
2) yellow in the north-south direction, red in east-west direction for a duration of 4 sec,
3) red in north-south direction, green in east-west direction for a duration of 30 sec, and
4) red in north-south direction, yellow in east-west direction for a duration of 4 sec.

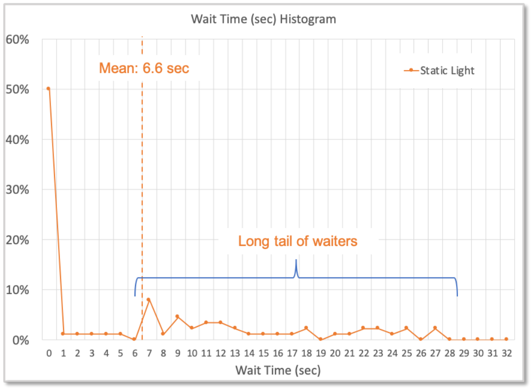
The durations were chosen based on recommended 'typical' values suggested by Flow.

time	id	waiting	speed
40	flow_00.10	0	10
45	flow_00.10	0	13.63
50	flow_00.10	0	14.02
55	flow_00.10	0	13.7
60	flow_00.10	0	2.91
65	flow_00.10	1	0.1
70	flow_00.10	6	0
75	flow_00.10	11	0
80	flow_00.10	16	0
85	flow_00.10	21	0
90	flow_00.10	26	0
95	flow_00.10	31	0
96	flow_00.10	0	0.6
100	flow_00.10	0	5.28
105	flow_00.10	0	11.1
110	flow_00.10	0	12.61
113	flow_00.10	0	13.99

Fig. 2. Sample data from one vehicle in the SUMO simulation.

Figure 3 shows the results of the static traffic light simulation on uniform traffic flow of 10 sec average inter-arrival time on all edges. The top plot shows the histogram of wait times for cars. From the plot we can see two roughly equal cohorts: a group of cars who generally breeze through the intersection and wait little (the spike at 0 wait time), if at all, and a group who end up at the traffic light with wait times that vary from 1 to 28 seconds. The mean for the set is 6.6 seconds. These results match intuition.

In addition we show the histogram of average car speed through the network. Again we see two distinct groups. One group flows through at near the entrance speed of 10 mph



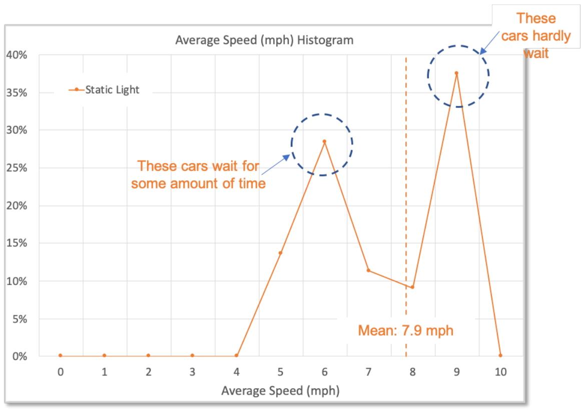


Fig. 3. Baseline metrics: distribution of wait times at the intersection and average speed through the network for a static 20 second traffic light.

(these are the cars that don't wait), and the other who wait some amount of time resulting in a lower average speed. The distribution shows a median of 7.8 mph (and mean of 7.9 mph), which will be another importance baseline metric for comparison with our RL agent.

V. Implementation

For comparison with out baseline static agent we implemented a deep Q-learning approach described below.

A. Experimental Setup

1) The Environment: We utilize the environment of a TrafficLightGridEnv that is available in Flow. This environment acts as a simulator and performs the change of state of the system given the action returned by the RL agent. It also gives the reward to the agent based on the action and the current observation space.

2) The State Space: The state space includes key parameters for each vehicles currently present in the network:

- the distance from the vehicle to the intersection,
- the speed of the vehicle, and
- the edge which the vehicle is on.

3) Allowable Actions: The action space is discrete with values of 'green', 'yellow', or 'red' for the traffic light in each of the direction. Note that the actions of the lights in the two directions are dependent with the 'red' phase of one light being the complement of the 'green'/'yellow' state of the other.

4) The Reward Function: The reward is the negative of the cumulative vehicle delay less a penalty for vehicles standing still.

B. Algorithms Implemented

1) Deep-Q Learning: The Q-learning algorithm learns the Q values for a given state and action based on the Bellman update equation:

$$Q^{\star}_{(}s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|s,a) \max_{a'} Q^{\star}_{(}s',a'). \quad (1)$$

Q-learning is a tabular algorithm, meaning it needs to compute the Q values corresponding to all states and actions. In our case, the state space is very large, with up to 100 cars present in the network at a given time, positions in the range of 0 to the length of the road under observation, and speeds between 0 and a maximum of 10 mph. A tabular approach would not perform well for such a large state space. Instead we utilize a **Deep Q-Learning** algorithm that takes in a state and gives the action based on a multilayer perceptron that essentially performs function approximation on the Q values. Figure 4 below shows the learning over episodes for the Simple Deep Q Learning Algorithm:

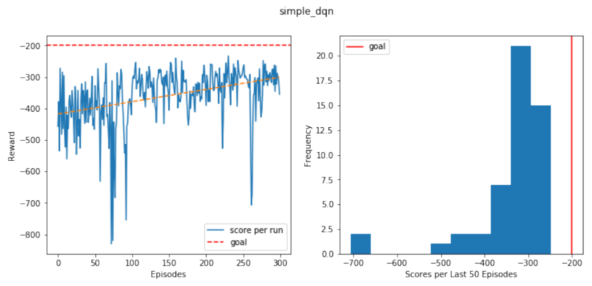


Fig. 4. Summary of training for the Deep Q Learning Algorithm

The algorithm for this is given below:

Algorithm 1 Deep Q Learning for Traffic Flow Control

```
$Q \leftarrow 0$
$s \leftarrow InitialState$
$done \leftarrow False$
**while** $done \neq True$ **do**
    $a \leftarrow Agent.GetAction$
    $s', r, done \leftarrow Simulator.Step(a)$
    $tdError = Q(s,a) - (r + \gamma max'_a Q(s',a'))$
    Backpropagate the error to update the neural net
**end while**
```

2) Deep Q Learning With Experience Replay: With deep Q-networks, we often utilize this technique called experience replay during training. With experience replay, we store the agent's experiences at each time step in a data set called the replay memory. We represent the agent's experience at time t as

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1})$$

All of the agent's experiences at each time step over all episodes played by the agent are stored in the *replay memory*. Then at each time step apart from the current experience, batches of experiences are randomly sampled from memory and are used to train the neural network. Such learning consists of two phases–gaining experience and updating the model. The size of the replay controls the number of experiences that are used for the network update. We do this because if the network learned only from consecutive samples of experience as they occurred sequentially in the environment, the samples would be highly correlated and would therefore lead to inefficient learning. Taking random samples from replay memory helps to break this correlation and achieve better performance. For our case, we took the Replay Buffer size of 20 experiences. Figure 5 shows the rewards summary over episodes for the training phase of the algorithm.

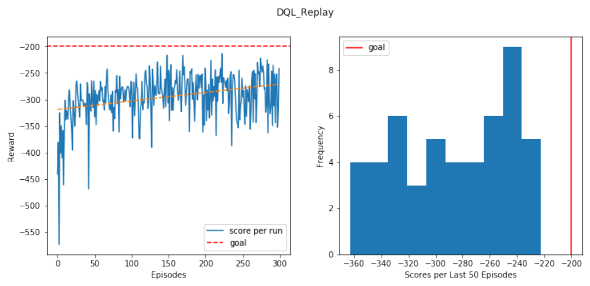


Fig. 5. Summary of training for the Deep Q Learning Algorithm

Algorithm 2 Deep Q Learning with Experience Replay

```
$Q \leftarrow 0$
$s \leftarrow InitialState$
$done \leftarrow False$
**while** $done \neq True$ **do**
    $a \leftarrow Agent.GetAction$
    $s', r, done \leftarrow Simulator.Step(a)$
    $sample \leftarrow randomSample(replayMemory, N) + currentExperience$
    **for** experience(s,a,r,s') in sample **do**
        $tdError = Q(s,a) - (r + \gamma max'_a Q(s', a'))$
        Backpropagate the error and update the neural net
    **end for**
**end while**
```

3) Double Deep Q Learning: Consider the Q-value function:

$$Q^*(s,a) = \mathbb{E}(r + \gamma max'_a Q(s', a'))$$

Taking the maximum overestimated values as such is implicitly taking the estimate of the maximum value. This systematic overestimation introduces a maximization bias in learning. And since Q-learning involves bootstrapping — learning estimates from estimates — such overestimation can be problematic.

To solve this, Double Deep Q Learning [17], uses two separate Q-value estimators(one is the actual network and the other is the Target Network), each of which is used to update the other. Using these independent estimators, we can unbiased Q-value estimates of the actions selected using the opposite estimator. We can thus avoid maximization bias by disentangling our updates from biased estimates. We update the *TargetNetwork* after some N iterations of the *Q-network* and to update the Q network we take the estimates from the *TargetNetwork*. Figure 6 shows the rewards summary over episodes for the training phase of the algorithm.

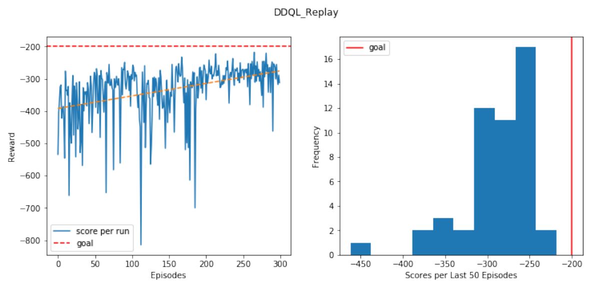
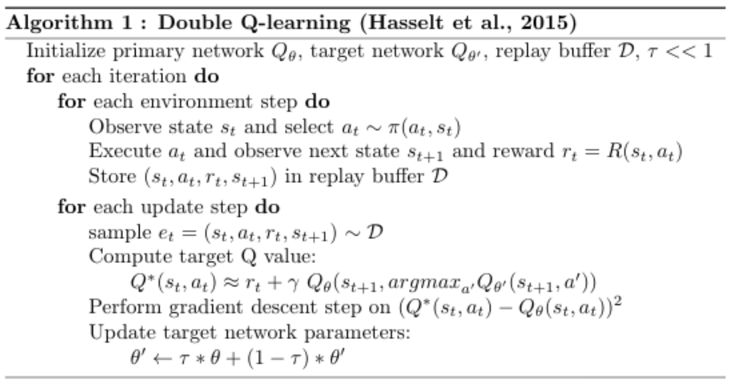


Fig. 6. Summary of training for the Deep Q Learning Algorithm

Figure 7 shows the algorithm.



Algorithm 1 : Double Q-learning (Hasselt et al., 2015)

```
Initialize primary network $Q_\theta$, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau << 1$
**for** each iteration **do**
    **for** each environment step **do**
        Observe state $s_t$ and select $a_t \sim \pi(a_t, s_t)$
        Execute $a_t$ and observe next state $s_{t+1}$ and reward $r_t = R(s_t, a_t)$
        Store $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $\mathcal{D}$
    **for** each update step **do**
        sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
        Compute target Q value:
            $Q^*(s_t, a_t) \approx r_t + \gamma\ Q_\theta(s_{t+1}, argmax_{a'} Q_{\theta'}(s_{t+1}, a'))$
        Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$
        Update target network parameters:
            $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$
```

Fig. 7. Double Deep Q Learning Algorithm

C. Training

We implemented the above algorithm using Pytorch Sequential Network [16] with a mlp (multi-layer perceptron) network which had 2 hidden layers with 32 and 64 units respectively. We used following hyper-parameters for learning:

1) Hyper Parameters:

- Learning rate (η): e^{-3}.
- Exploration fraction (ϵ): start with 0.3, decreasing to 0.01 as the learning progresses with decay rate of 0.99
- gama (reward decay): 0.9
- Number of episodes: 300.

- Size of Replay buffer: 10,000
- Replay Size: 64 samples in each update
- No. of iterations after which Target Network Updates: 10

D. Evaluation

The policy resulting from the training phase is stored in a pickle file that is loaded and applied during the evaluation phase. The experiment was setup to have the state action loop. The policy is then evaluated by running a simulation on the TrafficLightGridEnv for 10 episodes and reporting the average results.

VI. Evaluation Metric

Our evaluation metrics will mirror those used in the baseline case: namely the distribution of wait times and average speed through the network. See the following section for a description and analysis of the results.

VII. Analysis of Optimal Policies for Different Traffic Patterns

To explore the behavior of the learning algorithm we consider three different sets of car traffic behavior:

1) Uniform traffic: expected inter-arrival times on all edges of the network are equal and set to 10 seconds.
2) Biased traffic: the traffic is heavier in the east-west direction (expected inter-arrival time of 6.7 seconds) and lighter in the north-south direction (expected inter-arrival time of 20 seconds).
3) Dense biased traffic: we double the traffic of the prior scenario with east-west expected inter-arrival time halved to 3.35 seconds and that in the north-south direction halved to 10 seconds. This scenario is intended to test the nonlinear character of traffic flow, which are well-known [14].

A. Uniform Traffic

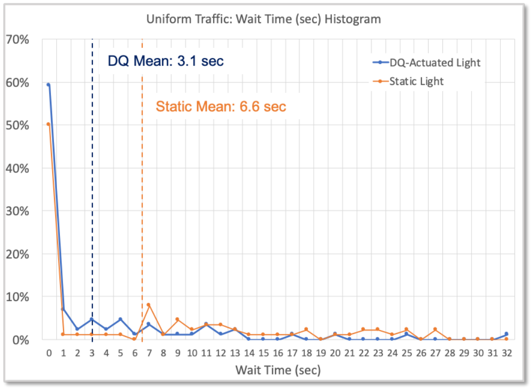
Figure 8 shows the performance of the DQ agent in comparison to that of our baseline static light on uniform traffic. For the wait times, we notice a dramatic improvement in the wait times. The mean wait time drops to 3.1 seconds, less than half that of the static light.

The improvement in average speed though is not as dramatic. Here we see an improvement of 10% or so in mean, though a larger portion of the cars see higher speeds.

It is instructive to observe the policy generated by the DQ agent for one of the episodes. This is shown in Figure 9. The plot shows the phases of one of the lights (with the other being the complement of this). Comparing this to the static light policy of 30 seconds green (+ 4 seconds yellow) - 34 seconds red, we observe:

- The DQ policy results in a total of 109 seconds of green/yellow time, compared to 119 seconds of the static policy.
- In the episode plotted, the traffic light presents a relatively long initial period, which is assumed in response to the traffic conditions initially perceived. But because the traffic is uniform, later in the episode it must compensate by leaving the orthogonal light on longer.

Although the total red/yellow time would appear to be the same order of magnitude between the two policies, we see that the re-distribution over the course of the episode does result in significant improvement in average weight times. This seems to show that adapting to the local conditions of the network is very powerful.



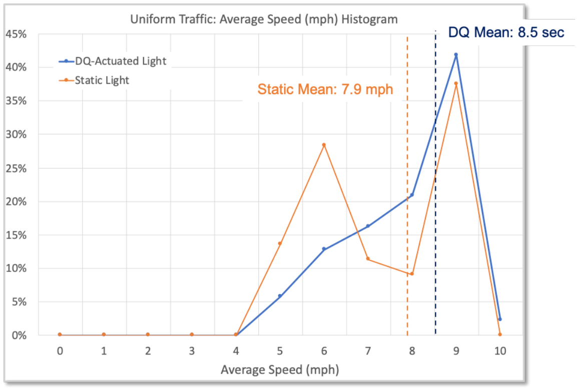


Fig. 8. DQ agent performance: distribution of wait times at the intersection and average speed through the network with comparison to the baseline static traffic light for uniform traffic.

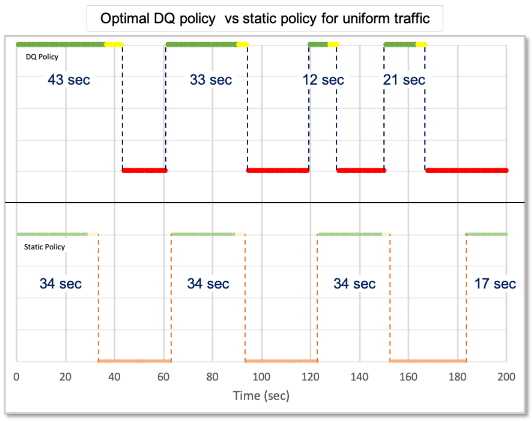
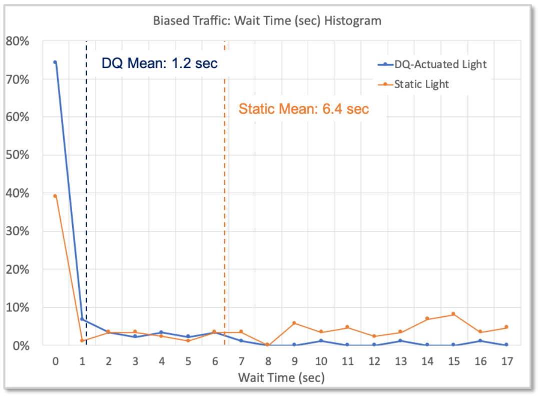


Fig. 9. DQ agent policy (for one of the lights): the DQ agent adapts to local (in time) traffic conditions while maintaining similar average behavior to the static agent.



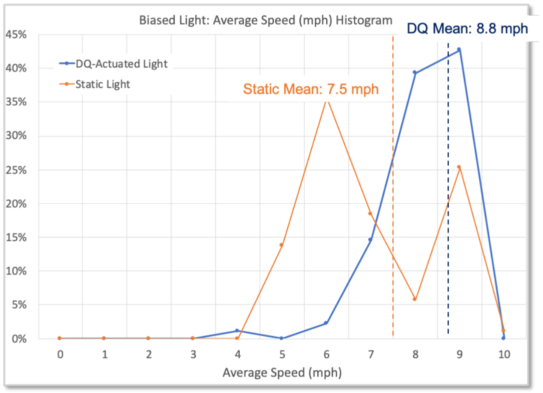


Fig. 10. DQ agent performance: distribution of wait times at the intersection and average speed through the network with comparison to the baseline static traffic light for biased traffic.

B. Biased Traffic

In this simulation the network has the same overall traffic as in the uniform case, but half the traffic of one direction is shifted to the other (meaning east-west traffic is 3× that of north-south.

Figure 10 shows the performance of the DQ agent in comparison to that of our baseline static light on biased traffic. For the wait times, the improvement is even more significant than for the uniform case. The mean wait time drops to 1.2 seconds, less than 20% that of the static light. The improvement in average speed is also higher with the DQ mean speed rising to 8.8 seconds.

Figure 11 shows the resultant policy for the DQ agent for a specific episode of the biased traffic. Here the total green/yellow time is 92 seconds, compared to 115 for the static agent, a ratio of 0.8. After an initial relatively lengthy green/yellow phase, the light does adapt to much shorter bursts of green/yellow on the lower trafficked edge.

The results match our intuition that the learned policy adapts to local conditions far better than a static policy and shows the potential for improvement when intelligent agents are employed.

C. Dense Biased Traffic

Figure 12 shows the performance of the DQ agent in comparison to that of our baseline static light on dense biased

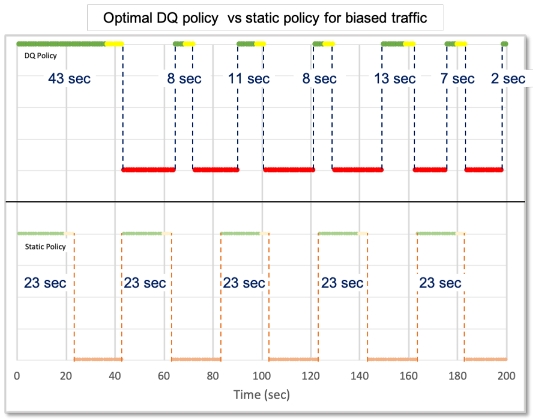


Fig. 11. DQ agent policy (for one of the lights): The DQ agent results in shorter duration green/yellow phases for the lesser-trafficked route compared to the static agent.

traffic. As expected, the denser traffic leads to a significant increase in average wait times with the static policy increasing to 8.8 seconds and the DQ agent increasing to 5.0 seconds. It's interesting to note that the degradation in performance for the DQ agent is quite severe compared to the static agent (increase in mean wait time from 1.2 seconds to 5.0 seconds, versus the change from 6.4 to 8.8 seconds for the static light). The mean average speed throug the network is now quite close for both the static and DQ agents with static at 5.7 mph and DQ at 6.3 mph.

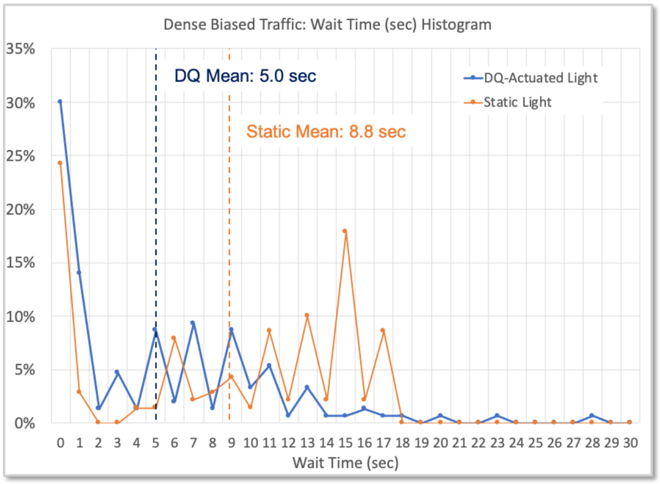
Figure 13 shows the resulting policy under dense traffic. If the traffic light system were linear, we would have expected the behavior of the DQ agent to mimic that of the system with half the traffic levels. The DQ agent starts similar to the light traffic case with a long duration green light. But instead of subsequent numerous short light bursts as we saw in the previous section, we see here fewer longer greens. The total green/yellow time is 101 seconds, 11 seconds longer than the light traffic case.

D. Left Turns in Network with 2 lanes

To bring the problem closer to the real world scenario, We performed experiment 14 with 2 lanes in each edge and added routes and connection for left turns. Agent still needed to take just 2 actions, either 0 or 1, with 1 means, switch the traffic light. This was more complex problem with more complex state space. We tried static agent, DQN with experience replay and DDQN with experience replay and found DQN ourperformed static agent and DDQN outperformed DQN by good margin. we saw the best gain by DDQN on this setup, this can be attributed to the fact, that this was mroe complex problem, and DDQN is well suited for complex state space problems and it's benefit is clearly visible on this experiment.

We saw Average velocities of 6.5 (static), 8.3 (DQN) and 9.3 (DDQN) after training for 500 episodes.

Also from waiting time graph 16, it can be seen that there is not much improvement for 0 waiting time vehicles in DDQN compare to baseline, but lot of increase in other low waiting



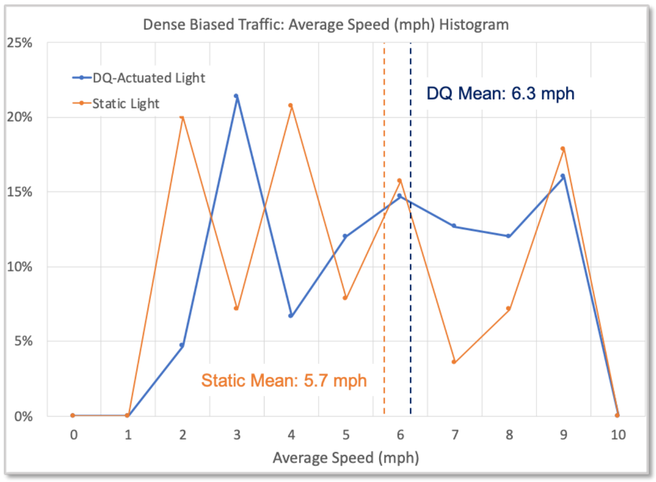


Fig. 12. DQ agent performance: distribution of wait times at the intersection and average speed through the network with comparison to the baseline static traffic light for dense biased traffic.

times. This is happening because vehicles which are going straight and not taking any turns, don't have to wait in any of scenario and so they both are taking roughly same time. But vehicles taking left turn has improved wait times with DDQN policy, and RL agent is able to reduce the wait times and increase traffic velocity.

VIII. Deep Q-Learning with Experience Replay

Figure 17 shows the improvement in performance achieved with the double deep Q-learning agent for the case of dense biased traffic compared to the deep Q-Learning agent and the static light. The improvement is significant: the mean wait time is reduced to 3.4 seconds, which is over 30% reduction from the performance of the deep Q-Learning agent. Further the mean average speed for the double deep Q-learning agent rises to 6.8 mph compared to 6.3 for the deep Q-learning agent.

Figure 18 compares the resultant policy of the double deep Q-learning agent and compares it with thos of the deep Q-learning and the static light. The double deep-Q does show a

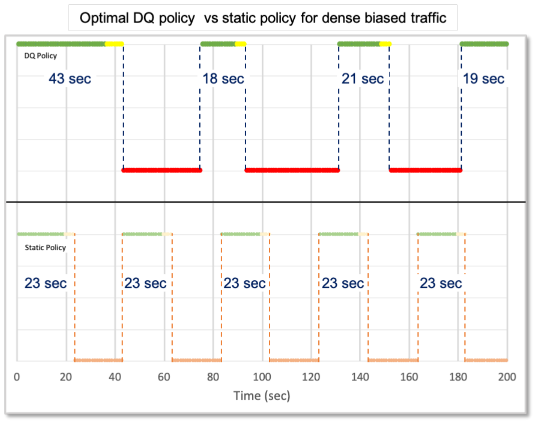


Fig. 13. DQ agent policy (for one of the lights): The DQ agent results in shorter duration green/yellow phases for the lesser-trafficked route compared to the static agent.

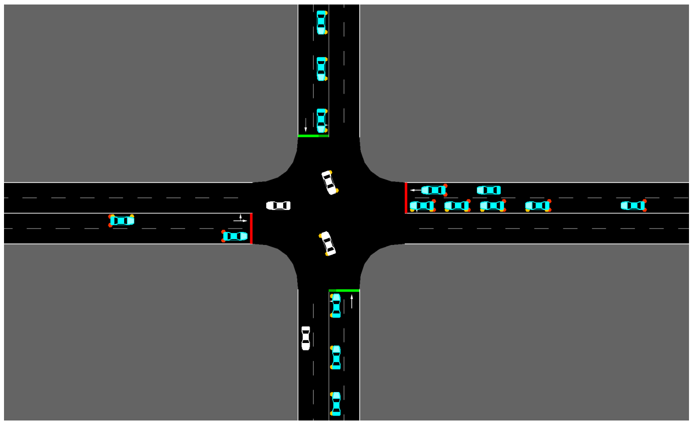


Fig. 14. A 2 lane traffic light signal with Left Turns

significantly different policy with more frequent and shorter green/yellow phases. Also the overall green/yellow time is reduced to 82 seconds compared with 101 seconds for deep Q.

IX. Code

The implementation can be found here

X. Future Work

There is lot of room for improvement on the current RL policy as well as to extend the problem further and consider taking multi agents system. A number of areas one can decide to work further on are:

1) Rewards tuning: There is certainly possibility of figuring out more reward functions, either handcrafted or maybe learned automatically by some algorithm.
2) Extending this to have more traffic light intersection and considering solving traffic light assignment over a set of traffic lights agents. So it will involve trying to optimize network flow and all traffic lights will have to coordinate together to optimize things globally. This will certainly

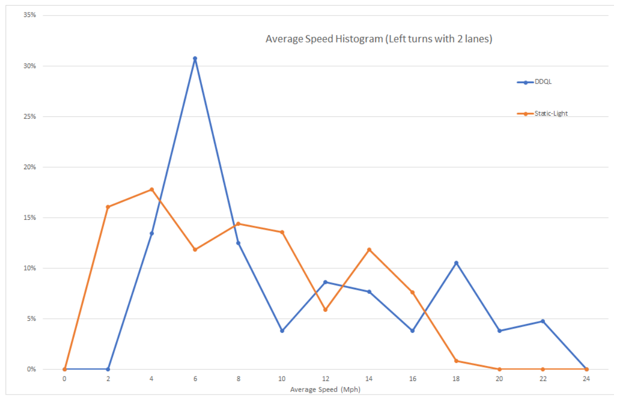


Fig. 15. Average speed for cars in a 2 lane network with left turns using DDQN

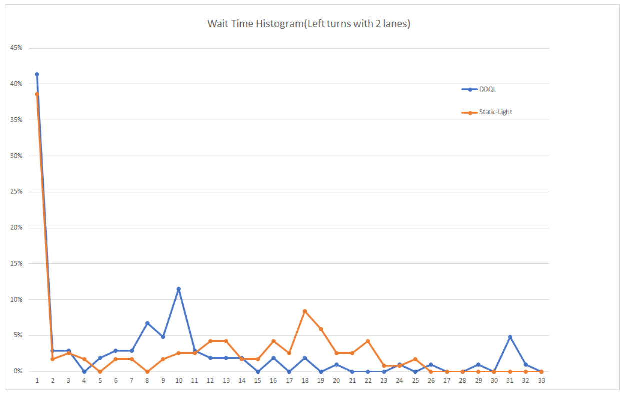
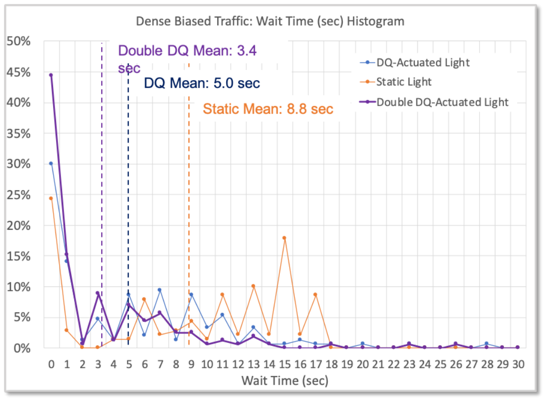


Fig. 16. Average waiting time for cars in a 2 lane network with left turns using DDQN

make problem a lot more complex and will have more use cases in real life situations.

3) We tried out Experience replay in this project and it certainly improved sample efficieny and improved the performance. We can try out prioritize experience replay and see if it is helping in converging algorithm faster, as we found in literature review, that it has helped in other problems.
4) Experimenting and tuning the right observation space for best performance. we tried with k-nearest vehicles from intersection in each edge. and tried various values of k (2, 5, 10). It will be interesting to explore what's the best observation space to be used for this problem. From our experience k=10 seems to be working well. Though number of parameters increases in the neural net and it takes more time to converge.
5) Trying out Policy gradient algorithm like PPO. we studied about these algorithms, and built some understanding, but due to time constraint, couldn't implement those algorithms. Given that these algorithms produce very good results on other problems like cartpole, it is a promising area of further exploration.
6) Train for longer period of time, this should result in more stable policies and maybe better aggregate rewards. Due to compute constraint we just tried 500-1000 episodes of training for each experiment.



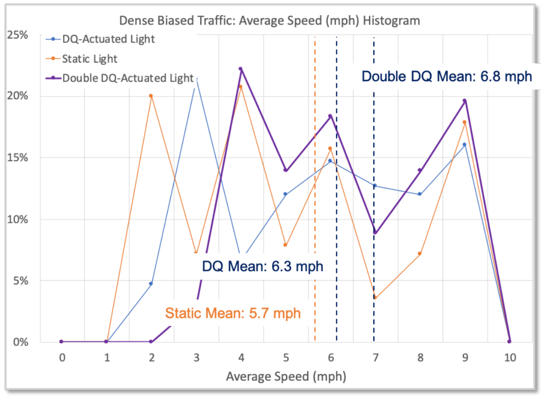


Fig. 17. Double DQ agent performance: distribution of wait times at the intersection and average speed through the network with comparison to the baseline static traffic light and the Deep Q-Learning agent for dense biased traffic.

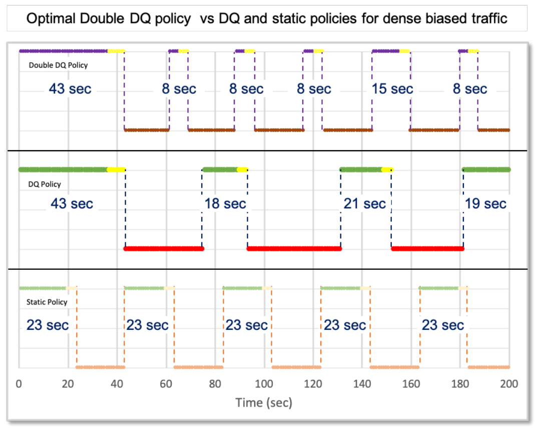


Fig. 18. Double DQ agent policy (for one of the lights): The double DQ agent results in shorter duration green/yellow phases and less overall green/yellow than the deep Q-learning agent for the same dense biased traffic.

References

[1] U. DOT, "National transportation statistics." Bureau of Transportation Statistics, Washington, DC, 2016.

[2] D. Schrank, B. Eisele, and T. Lomax, “Ttis 2012 urban mobility report,” Texas AM Transportation Inst. The Texas AM Univ. System, 2012.

[3] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, International Journal On Advances in Systems and Measurementspenalize standstill of traffic, vol. 5, no. 34, 2012.

[4] https://sumo.dlr.de/docs/index.html.

[5] https://flow-project.github.io/index.html.

[6] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, “Benchmarking deep reinforcement learning for continuous control

[7] V. Joshi. “How reinforcement learning can help in solving real-world problems?

[8] I. Arel, C. Liu, T. Urbanik, and A.G. Kohls, ”Reinforcement learning-based multi-agent system for network traffic signal control”, IET Intelligent Transport Systems, 2010, Vol. 4, Iss. 2, pp. 128– 135.

[9] G. Marion. “Deep reinforcement learning on a traffic light system,” https://arxiv.org/pdf/1803.11115.pdf.

[10] Flow: A Deep Reinforcement Learning Framework for Mixed Autonomy Traffic. Mobile Sensing Lab, University of California Berkeley. https://flow-project.github.io/index.html

[11] C. Wu, A. Kreidieh, K. Parvate, E. Vinitsky, A. Bayen, ”Flow: Architecture and Benchmarking for Reinforcement Learning in Traffic Control,” CoRR, vol. abs/1710.05465, 2017.

[12] W. Genders and S. Razavi, “Using a deep reinforcement learning agent for traffic signal control,” arXiv preprint arXiv:1611.01142, November

[13] OpenAI Baselines, https://github.com/openai/baselines

[14] Hui, Meng and Bai, Lin and Li, YanBo and Wu, QiSheng,”Highway traffic flow nonlinear character analysis and prediction,” Mathematical Problems in Engineering, volume 2015, doi = 10.1155/2015/902191.

[15] S. Adam, L. Busoniu and R. Babuska, ”Experience Replay for Real-Time Reinforcement Learning Control,” in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 42, no. 2, pp. 201-212, March 2012, doi: 10.1109/TSMCC.2011.2106494.

[16] https://pytorch.org/tutorials/beginner/examples_nn/two_layer_net_nn.html

[17] Van Hasselt, Hado Guez, Arthur Silver, David. (2015). Deep Reinforcement Learning with Double Q-learning.