Deep Q Traffic Light Control

Zahra Aminiranjbar
Department of Electrical and Computer Engineering
University of California, Davis
Davis, CA, 95616
zaminiranjbar@ucdavis.edu

Dylan Shadduck
Department of Electrical and Computer Engineering
University of California, Davis
Davis, CA, 95616
dmshadduck@ucdavis.edu

Greesan Gurumurthy
Department of Computer Science
University of California, Davis
Davis, CA, 95616

Abstract

Traffic congestion at four way intersections commonly arises from the poor adaptive capabilities of the scheduling policy executed on traffic lights. Reinforcement learning has been used to allow such systems to adapt to varying parameters that define the state of the intersection at any given moment. Implementations of reinforcement learning towards this problem have shown that traffic congestion can be decreased in various dynamic simulation environments. Our work focuses on a further introspection and expansion of both the state space and the reward function of these reinforcement learning agents. In particular, we include the current light phase, the summative queue length of lanes in the same direction, and the position and velocities of each lane into our state space. We include both queue length and summative delay over all current cars in the intersection as part of our reward function. The inclusion of particularly the position and velocities of each lane forces the use of convolutional neural networks and deep neural networks to assist in their processing, as the states space that arises from these particular elements is very large therefore hard to decipher. Through the addition of such measures, we were able to decrease traffic congestion, as shown by our simulation's decrease in average time delay by about 20-63% from common baselines.

1 Introduction

Increased traffic congestion can lead to direct consequences including increased traveling time and increased fuel consumption. These problems have further indirect consequences, including increased driving infractions in attempts to reach destinations quicker, more common car crashes, and increased CO2 emissions. As the number of cars on the road continues to increase, such issues will be further accelerated, as we cannot alleviate such rapidly growing pressures through gradual changes in road infrastructure. As traffic lights at four way intersections are a common bottleneck with the goal of ensuring citizen safety at the cost of increased traffic congestion, we focus on how traffic light schedules controlling such junctions operate, and see if we can alleviate traffic congestion by optimizing such schedules without a resultant decrease in safety. In most instances, such traffic lights are kept on a rigid schedule that can slightly adapt in a fixed manner based on the small number of cars queued that can be detected by inductive loop traffic detectors, which can be used to surmise a limited queue length. Improvements in camera vision also provide ways to gain more data that can help create a more accurate representation of intersection environments, including position and

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

velocities of cars approaching an intersection but not queued. Such data parameters along with current traffic phase data, can be fed into dynamically evolving reinforcement learning algorithms in order to provide vast decreases to traffic congestion. Authors in [2] proposed a reinforcement learning system which includes position and velocity matrices as part of the state space, and we believe that using this data in addition to queue length and phase data, used in [7] will provide a more accurately optimizing reinforcement learning system. In section 2 we elaborate on related work that assisted us while implementing our project. In section 3 we discuss our reinforcement learning system model in more depth. In section 4 we focus on the Deep Neural Network that we use to survey our large state space for Q values associated with the current state of the intersection. In section 5 we detail and explain the significance of our results accumulated from running simulations on the Simulation of Urban MObility (SUMO). In section 6 we discuss possible further improvements to our system.

2 Topic Review

As mentioned earlier, reinforcement learning can be used to optimize traffic light policies at 4 way intersections. Diving a bit deeper, much of the associated groundwork related to our project has been set by other published works[1][3][10][5][6][4][9]. [8] provides a review of many related papers, from which it can be noticed that reinforcement learning agents that particularly used CNNs worked well, as this allows the incorporation of vehicle position and vehicle speed as state information. Furthermore, the most commonly used simulation environment used for such CNNs was SUMO, prompting our use of the SUMO simulation. While most papers mentioned in the review paper were similar, there were minor differences that led to large discrepancies in final optimization outcomes. For example, [7] used only the queue length and the current traffic phase as state inputs and resulted in a 68% reduction in the average delay of cars compared to a rigid light schedule baseline, while [2] used the current traffic light phase and the vehicle position and speed as state attributes, resulting in an 86% reduction instead.
We closely scripted our implementation to the Gao paper [2], as this provided the best improvement over the rigid baseline. [2] used CNN layers to input velocity and position matrices into two DNNs: a target and train network, which would output Q values associated with the actions that can be taken. Depending on an epsilon greedy implementation, either the action corresponding to the highest Q value or a random action would be taken. As the actions in this scenario included only letting North-South traffic pass and letting East-West traffic pass, this results in a 1x2 output layer. The weights of these networks were trained through experience replays of past state,action,reward sets, with slightly different variations in how the training and target network were trained. While this implementation only used the position and speed matrices and current light phase as states, and used only average vehicle delay, our implementation also takes into account the queue length as both a state and factor in the reward function.

3 Simulation and problem formulation

In this section we describe how we have formulated the traffic light control as a reinforcement learning (RL) problem. In RL the agent learns by interacting with an environment while receiving rewards. The environment we picked for this TLC problem is a four-way intersection (figure 1).

Each road in this intersection has 3 lanes. The vehicles in the inner-most lane are only allowed to make left turns and the vehicles in the outer-most lane can go straight or make right turns only when their light is green. The junction simulation was done using NetEdit . NetEdit allows us to set the road length, number of lanes per road, the allowed turns at the junction and much more that allowed us to simulate our junction as close to a real world problem as possible. The cars in our simulation follow traffic rules which means, they go through the intersection when the light is green and decelerate and stop when the light is yellow and come to full stop when the light is red.

3.1 State

The RL agent interacts with the environment by controlling the traffic light policy and the goal of the agent is to reduce the cumulative staying time of the vehicles as well as the queue length of the cars waiting behind the light. The agent does this by getting state information from the environment. In this problem we define the states as vehicle position, vehicle speed, queue length and traffic light

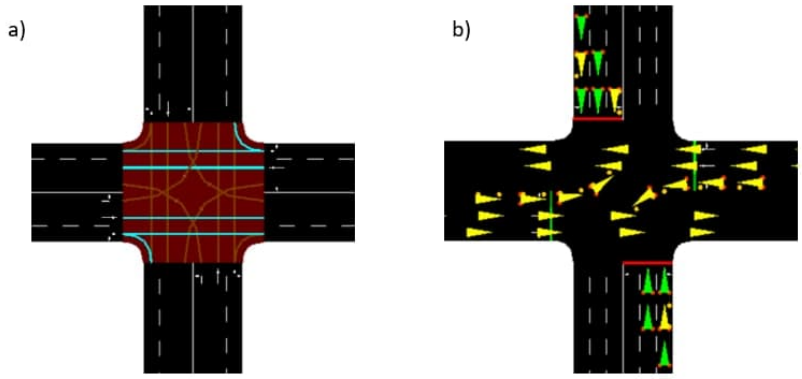

Figure 1: (a) NetEdit simulation of junction and allowed routes. (b) snapshot of traffic at the junction.

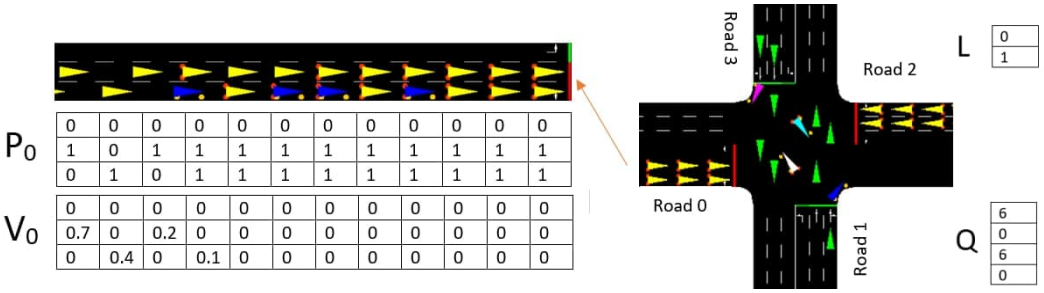

Figure 2: Snapshot of traffic at the junction and the position(P), velocity(V) matrices of road 0 and light state (L) and queue length (Q) matrices of the junction

phase. The agent then selects an action and actuates traffic signals. This actuation traverses the environment to a new state and would result in a reward which the agent uses later to make its future decisions. We store these experiences as state, action, reward, and next_state in the memory to use and train our agent. I mentioned that the states include vehicle position, which we define as a 12 by 12 sparse matrix, P. We divide the lanes of each incoming road into 12 segments of length 7 (assuming that each segment would include a single car that has length 5) starting from the stop line. Which gives us four 3 by 12 matrices which build the position matrix. Another state feature is the speed matrix, V, which has the same dimension as the position matrix and corresponds to the normalized speed of the vehicle at that position. Our agent's third input data from the environment is the queue length, Q. This is a 4 by 1 matrix that has the sum of all the stopping vehicles at the intersection of each road. The final data that we provide for our agent is the state of the traffic light, L. It's a 2 by 1 matrix. L=[1;0] when the green light is on for west-east traffic and L=[0;1] when the green light is on for north-south. Therefore at each time step t the agent observes the current state in the form of S_t=(P,V,Q,L), (figure 2) .

3.2 Action

The agent of our problem is capable of taking two actions 0 or 1. The agent takes this action by epsilon greedy policy in which 90 of the time, the agent takes the action with maximum Q value and other times it picks an action randomly between 0 and 1. Action 0 corresponds to turning the East-West (EW) light green and action 1 is when the agent turns the North-South (NS) light green. It is important to note that to allow for left turns to also take place in our simulation , we have transition states between agent's actions. For example , if our agent takes action 1 (turning NS light green) , we first check the light phase. If the NS light is already green , then action 1 will keep it green for another tg=10s otherwise , we first turn the EW light yellow for all EW lanes EXCEPT the left turns for ty=6s. This would allow cars waiting to turn left finish their turn successfully , otherwise the cars end up crashing or disrupting traffic rules (making turns when the light is red). Then we turn the EW road's left turn light yellow for another ty=6 seconds while turning other EW lanes light red. Finally

Time line			Road 0			Road 1			Road 2			Road 3		
			L0	L1	L2	L0	L1	L2	L0	L1	L2	L0	L1	L2
Step t-1 Green tg			g	G	G	r	r	r	g	G	G	r	r	r
Step t Green tg			g	G	G	r	r	r	g	G	G	r	r	r
Step t+1	**transition**	ty	g	y	y	r	r	r	g	y	y	r	r	r
		tg	G	r	r	r	r	r	G	r	r	r	r	r
		ty	Y	r	r	r	r	r	y	r	r	r	r	r
	green	tg	r	r	r	g	G	G	r	r	r	g	G	G

r: red light
G:green light
g: is green light for vehicles turning left , letting vehicles going straight pass first
y: yellow light
L_i : lane i

Figure 3: Transition of light phase from EW green to NS green

we turn all the EW lane's light red and turn the NS light green. The same goes if action 0 is picked (figure 3).

3.3 Reward

The final element of our model after the agent observing the current state S_t , taking action A_t and observing future state S_{t+1}, is for it to receive a reward. This is a very important part of the reinforcement learning. It allows the agent to interact with the environment and adopt new policies according to the current state of the environment. This part allows us to deviate from traditional traffic light control methods and brings AI agents to life that can observe the intersection and make the best decision based on the current status of the intersection. In our problem, our aim is to reduce the overall waiting time of the cars behind the traffic light. To capture this, we set our reward as the difference between the cumulative delays of all the cars stopping on the roads before the action takes place and after the action is executed. For our reward function to have a more accurate representation of the intersection, we add the queue length of all the roads to the waiting time before and after each action.

$$R_t = (W_t + qlen_t) - (W'_t + qlen'_t) \tag{1}$$

W_t the is sum of the waiting time of all the vehicles at the beginning of the green light interval at time step t and $qlen_t$ is the sum of all the roads queue length at time t. W'_t and $qlen'_t$ are the sum of waiting time of all the vehicles and sum of all the roads queue length respectively at the end of the green light interval at time step t.

4 Deep Neural Network

To train our agent on the complex state space outlined above, a Deep Neural Network (DNN) is implemented to find associations between the state space and the expected reward or Q value. To allow for our model to converge, we followed a similar approach to [2] by making use of both a target network and a train network. The structure of both these networks is identical, but the method of updating the weights is altered so that the two models can communicate and find an optimal policy.

4.1 Network Structure

The DNN network takes in four inputs: position matrix, speed matrix, light phase, and queue length. Of these four inputs, the position matrix and the speed matrix have the highest dimension and thus are treated differently. Both of these inputs are subjected to two 1D convolution layers to reduce the dimension. After convolution, these tensors are flattened and appended together along with flattened versions of the other two inputs, the light phase and queue length. Once all the inputs are together, the following layers are all fully connected dense layers with 128 and 64 nodes. Finally, the output of

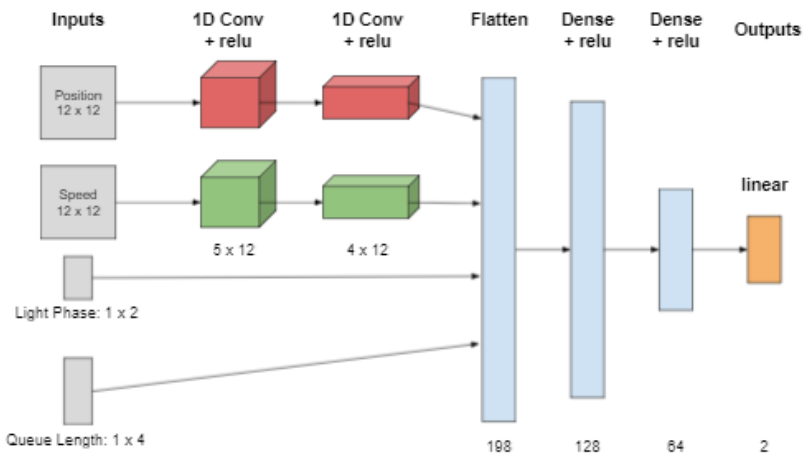

Figure 4: Deep neural network structure

this network is the estimated Q value for all possible actions from the given state information. The full network structure can be seen in figure 4.

4.2 Updating Model Weights

To update the weights of these networks, the RMSProp optimizer is used externally. This needs to be done in this manner since the gradient descent method is to be applied based on the estimated Q values from the train network. The value we need to extract is the Q value associated with the action taken in our experience. We then compare this Q value with the true value as predicted by the target network. This true value is based on the reward observed and the predicted Q value and is defined to be $Q(S_t, A_t, \theta)$. We then use the mean squared error between these two results as the loss for our RMSProp algorithm and update the weights of the training network. The loss function of the combined network structure is thus defined as

$$\frac{1}{m}\sum_{t=1}^{m}((R + \gamma \max_{a} Q(S_{t+1}, a, \theta')) - Q(S_t, A_t, \theta))^2 \tag{2}$$

Here, m represents the number of experiences in our batch which we have defined to be 32. After the train model weights, θ, have been updated, we can update the target model weights θ' using equation 2

$$\theta' = \beta\theta + (1 - \beta)\theta' \tag{3}$$

where β is the update rate of the target network.

5 Agent Training

The training of our agent is done using python and the traffic simulation SUMO. We start the training by first initializing the target and training DNN models and ensuring that their model weights are the same. We then begin to train our agent over a total number of episodes that we have set to 2000. Inside each training episode, a traffic simulation is initialized with random vehicle routes. This ensures that the models learn to understand optimal traffic signal phase for any circumstance, not a specific vehicle distribution.

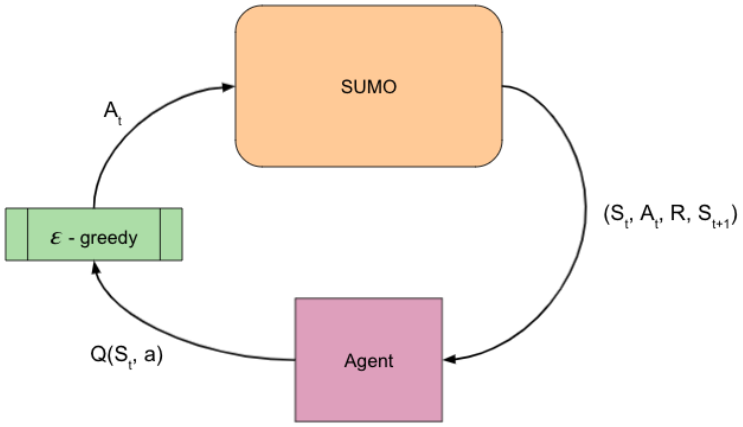

Figure 5: Agent simulation interaction

5.1 Action Selection

We have a set minimum amount of time that each light phase must last for, so while the light remains in the same phase, our agent does not train or learn the state of the simulation. Only when a change of phase may occur, does the agent train. The training process begins by estimating the Q value for the given state. This is done by applying the "predict" method to our target DNN. Next we decide the optimal action based on the maximum Q value and the epsilon greedy algorithm. Once an action is chosen, we verify if this action is the same or different from the previous action. If it is the same, we don't need to set the phase, otherwise, we allow the light to cycle from one phase to the next. A simplified model of this interaction can be seen in figure 5.

5.2 Data Collection

After the light phase has been set we can update our experience memory mentioned earlier. This memory is a collection of simulation experiences that can date back up to 200 episodes prior. Each experience consists of a state, an action, a reward, and the next state. To get this information we first allow the agent to run once and save a state to a variable called previous state. When we take another simulation step and observe another state, we can update our experience by looking at the previous state we set earlier, the difference in wait time between the two states (reward), and our current state. The difference between our two states tells us what the action was, so we have all of our experience information. We append this experience to our memory. The full algorithm used for training can be found in algorithm 1.

6 Results

While the running the simulation with the initial configuration of weights can lead to wait times as high as $3.5 * 10^6$ s in heavy traffic situations, Figure 7 emphasizes that the reinforcement policy does actively succeed in starting to optimize the phase schedule to result in lower wait times, as shown by the downward trend in average wait time as the number of episodes increases. Through the training and iteration of our DNN assisted Reinforcement Learning agent, our optimized policy leads to an average wait time of about 420s.

When testing our RL model with lower amounts of traffic congestion, done by changing the code creating the Sumo route file, we noticed that our average wait starts at a lower initial average wait time, and drops to a lower optimal value even when trained for fewer episodes, both of which are expected (Figure 9). As iterations over episodes can take anywhere from a minute to 8 minutes, depending on the memory and compute capabilities we have on hand, taking averages over hundreds

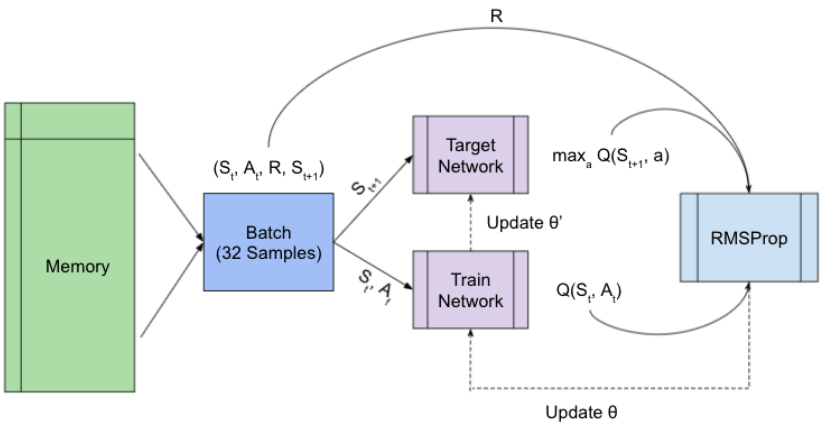

Figure 6: Neural network optimization

Algorithm 1

1. Initialize train DNN with weights θ
2. Initialize target DNN with weights $\theta' = \theta$
3. Initialize $\gamma, \epsilon, \beta, N$
4. Initialize memory bank M
5. For step in range(N):
6. Initialize simulation
7. for t in range(T):
8. Take one time step
9. if t is light transition time
10. Get state information S_t
11. Set action A_t according to $\max_a Q(S_t, a, \theta)$ with probability of 1 - ϵ
12. Take random action with probability ϵ
13. if A_t == $A_t - 1$:
14. keep light phase the same
15. else:
16. Transition to yellow light phase
17. Take action A_t
18. Generate experience S_t, A_t, R, S_{t+1}
19. Add experience to memory M
20. Train DNN on random batch of 32 from M
21. Update θ and θ' according to (3)
22. Set S_{t+1} to S_t

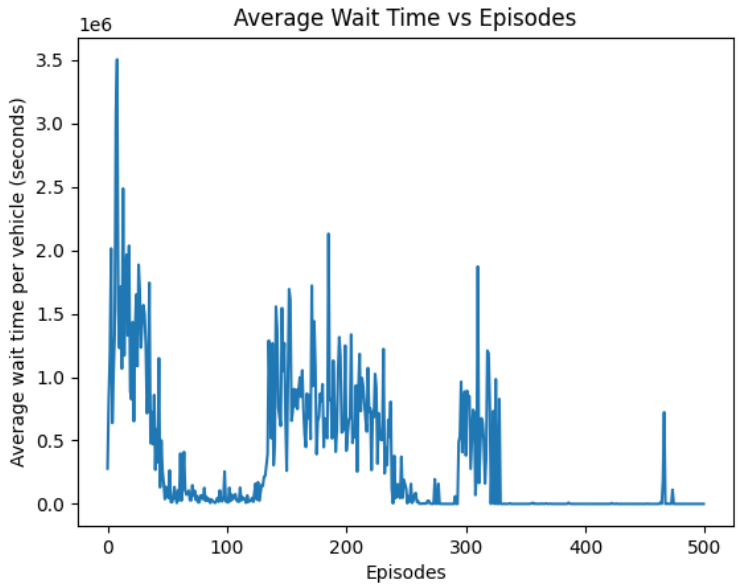

Figure 7: Result of our RL model when trained in heavy traffic simulation

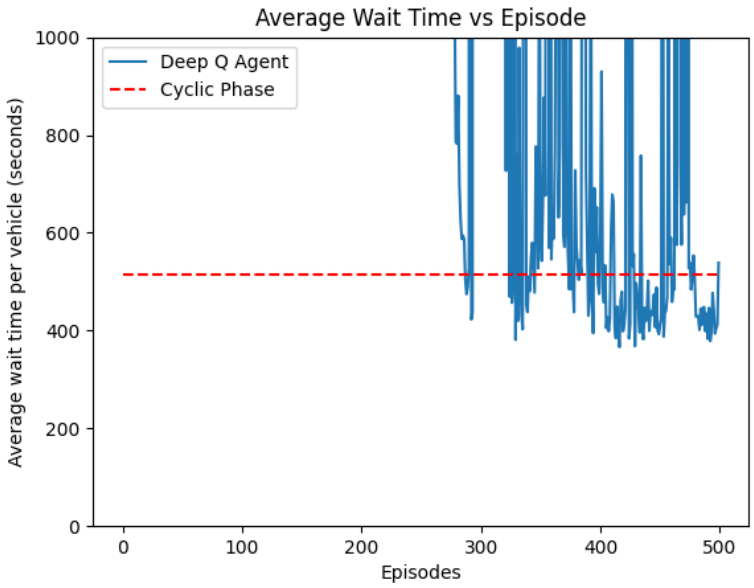

Figure 8: Comparison between RL model and optimal rigid schedule baseline when tested in heavy traffic. As shown by the diagram our model does beat the optimal rigid scheduler, showing that our model adjusts the light phase schedule more dynamically than the optimal rigid schedule, leading to the deep Q agent beating the rigid schedule as the number of episodes trained for increases. The associated wait time speedup of the RL implementation is about 20%.

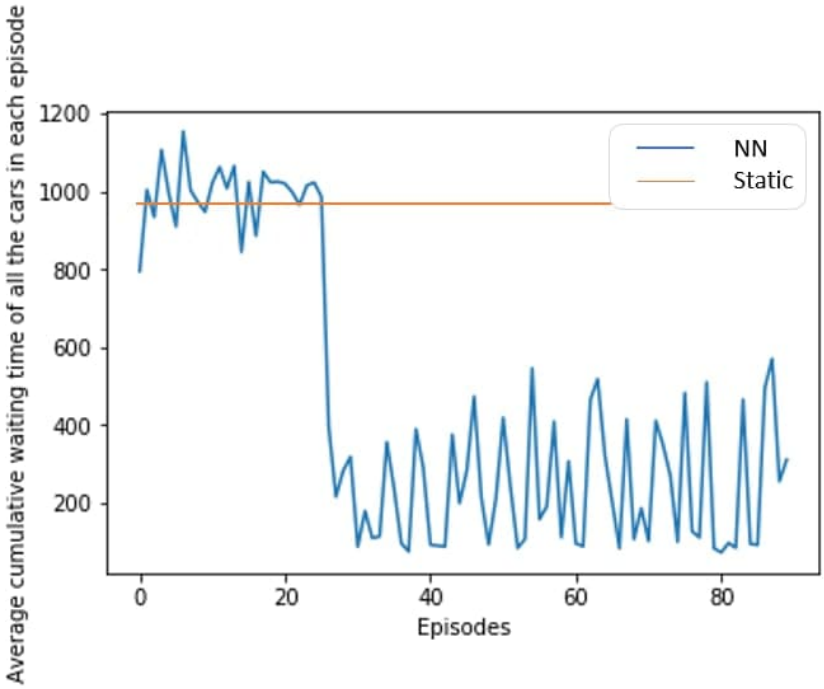

Figure 9: Result of our RL model when trained in light traffic simulation, compared with an optimal rigid schedule. Again the our model beats the optimal rigid scheduler. The associated wait time speed up of the RL implementation in this case is about 63%.

of iterations over 5000 episodes was not possible given our limited time frame for project submission. By incorporating the use of large compute services, possibly including EC2 instances or on-campus HPC systems, we can get more generalized data on how well our reinforcement learning policy performs on average, while also without spending days for simulation. The large variance in our average wait times is similar to the Gao paper [2], which shows a similar random dispersal of average wait times over each episode to begin with, before converging to a low wait time.

7 Insight on further implementations

While the implementation we produced shows a working proof of concept that reinforcement learning based traffic light control can be optimized past normal traffic light control schedules, we can further implement more features so that the simulation can more accurately model real life intersections. The most important aspect to focus on immediately is the fact that our simulation sometimes results in 2 cars in the left turn lane getting stranded in the middle of the intersection, which would require more strict restrictions to be placed to ensure vehicle safety. Ways to bring together the sim2real gap include adding right turn on red capabilities, adding pedestrians to the intersection, and implementing an agent per intersection in a multi-intersection environment, where each intersection can have the ability to communicate with connected intersections.

8 Contributions

All members of this project contributed equally to its completion. Zahra worked on the set up of our environment in SUMO by designing the Netedit file and the route file. She also developed methods for extracting the state, action, and reward information from SUMO and the RL agnet's reward function implementation. Dylan and Greesan worked together on the development of the deep neural network and the agent training process.

References

[1] Itamar Arel, Cong Liu, Tom Urbanik, and Airton G Kohls. Reinforcement learning-based multi-agent system for network traffic signal control. *IET Intelligent Transport Systems*, 4(2):128–135, 2010.

[2] Juntao Gao, Yulong Shen, Jia Liu, Minoru Ito, and Norio Shiratori. Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network. 05 2017.

[3] Deepeka Garg, Maria Chli, and George Vogiatzis. Deep reinforcement learning for autonomous traffic light control. In *2018 3rd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*, pages 214–218. IEEE, 2018.

[4] Hongwei Ge, Yumei Song, Chunguo Wu, Jiankang Ren, and Guozhen Tan. Cooperative deep q-learning with q-value transfer for multi-intersection signal control. *IEEE Access*, 7:40797–40809, 2019.

[5] Wade Genders and Saiedeh Razavi. Using a deep reinforcement learning agent for traffic signal control. *arXiv preprint arXiv:1611.01142*, 2016.

[6] Xiaoyuan Liang, Xunsheng Du, Guiling Wang, and Zhu Han. A deep reinforcement learning network for traffic light cycle control. *IEEE Transactions on Vehicular Technology*, 68(2):1243–1253, 2019.

[7] Seyed Sajad Mousavi, Michael Schukat, Peter Corcoran, and Enda Howley. Traffic light control using deep policy-gradient and value-function based reinforcement learning. *CoRR*, abs/1704.08883, 2017.

[8] Faizan Rasheed, Kok-Lim Alvin Yau, Rafidah Md. Noor, Celimuge Wu, and Yeh-Ching Low. Deep reinforcement learning for traffic signal control: A review. *IEEE Access*, 8:208016–208044, 2020.

[9] Remi Tachet, Paolo Santi, Stanislav Sobolevsky, Luis Ignacio Reyes-Castro, Emilio Frazzoli, Dirk Helbing, and Carlo Ratti. Revisiting street intersections using slot-based systems. *PloS one*, 11(3):e0149607, 2016.

[10] Hua Wei, Guanjie Zheng, Vikash Gayah, and Zhenhui Li. Recent advances in reinforcement learning for traffic signal control: A survey of models and evaluation. *ACM SIGKDD Explorations Newsletter*, 22(2):12–18, 2021.