TRAFFIC LIGHT SIGNAL CONTROL USING DEEP REINFORCEMENT LEARNING

Project report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

Submitted by

Sayantika Mandal (1810110224)

And

Sanskar Tewatia (1810110215)

Under supervision

of

Prof. Madan Gopal
Department of Electrical Engineering

DEPARTMENT OF ELECTRICAL ENGINEERING

SCHOOL OF ENGINEERING

SHIV NADAR UNIVERSITY

(December 2021)

Candidate Declaration

We hereby declare that the thesis entitled "Traffic Light Signal Control Using Deep Reinforcement Learning" submitted for the B. Tech. degree program. This thesis has been written in our own words. We have adequately cited and referenced the original sources.

[illegible]

(Signature)
Sayantika Mandal
(1810110224)
Date: 27/11/2021

[illegible]

(Signature)
Sanskar Tewatia
(1810110215)
Date: 27/11/2021

CERTIFICATE

It is certified that the work contained in the project report titled “Traffic Light Signal Control Using Deep Reinforcement Learning,” by “Sayantika Mandal” has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Signature)
Prof. Madan Gopal
Dept. of Electrical Engineering
School of Engineering
Shiv Nadar University
Date: 27/11/2021

CERTIFICATE

It is certified that the work contained in the project report titled "Traffic Light Signal Control Using Deep Reinforcement Learning," by "Sanskar Tewatia" has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Signature)
Prof. Madan Gopal
Dept. of Electrical Engineering
School of Engineering
Shiv Nadar University
Date: 27/11/2021

Abstract

An efficient transportation system is the need of the hour in developing countries, where incessant traffic congestions have become increasing resource-consuming, and pose an obstacle towards unceasing mobility. Traditional controllers such as Adaptive Traffic Signal Control (ATSC), based on sensors placed around traffic intersections and pre-defined fixed-time controllers optimized by traffic engineers, are unable to sufficiently resolve the traffic congestions since they do not account for the unexpected changes in traffic patterns. Though they are capable of regulating nominal traffic flow, they are vulnerable abnormal circumstances such as accidents, construction, and rush hours. Hence, an intelligent traffic light control is indispensable for alleviating the pressure caused due to traffic congestions. We propose an intelligent traffic light control agent, which will consider a four-way intersection as the environment, and use Deep Neural Networks to analyze the state representation of traffic such as queue length on the lanes, position of vehicles along with current and the next phase of traffic light via image representation from a simulator. The agent uses deep reinforcement learning techniques such as Deep Q-Networks (value-based method), REINFORCE (policy-based method) and Advantage Actor Critic method to suggest the optimal action of either changing or maintaining the current traffic light phase, so as to maximize the reward, which in our case is the average waiting time of vehicles and the queue length. In order to gather data to train and test the agent, SUMO (Simulation of Urban Mobility) simulator package is used, which helps emulate real-life traffic scenarios.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Motivation:

With a population of over 7 billion and increasing, the vehicular traffic has always been on the rise. As more manufacturers are competing with each other to reduce the expenses of purchasing or owning vehicles, people today are in a better capacity of being able to own their own cars. Even in the presence of public transport such as underground metros, people prefer to travel in their own personal vehicles. This has led to a massive increase in the demand of efficient traffic management, be it traffic light control, or building efficient architectures that completely alleviate the need of traffic lights [1]. However, such construction activities require immense planning, budget allocation and might not be feasible in crowded areas where the space is also limited. Hence, there is a scope of optimizing traffic signal control, in order to better manage the traffic flow and reduce the waiting time as well as the queue length at various intersections throughout the world.

At present, traffic signals are controlled by either setting a fixed duration for each state of the traffic light [2, 3], or by using the vehicle-activated inductive sensors underneath the road, and using these inputs to estimate the traffic signal duration. However, both of these methods, though simple to implement, are unable to adapt to emergency, accidental or rush hour scenarios. In such tricky cases, usually traffic policemen have to step in, and manually control the flow of traffic, on the basis of the congestions they witness, based on their previous experiences of controlling such crowded junctions. Thus, an adaptive traffic light control technique is the need of the hour as such a technique would be able to accurately process the current state of traffic, and take optimal actions in regard to the reducing the congestions, and thereby decreasing the waiting time and queue length [4, 5, 6].

Reinforcement Learning:

Reinforcement Learning (RL), a feedback-based Machine Learning technique, is the most-used methodology for traffic signal control and has been shown to have the potential to dynamically adjust traffic lights according to real-time traffic in recent studies. In a Reinforcement Learning problem, an agent observes a stochastic environment, perceives its state and learns automatically by exploring the environment and performing actions on a timely basis which results in a change in the environment to another state [7]. A policy function governs the transitions between states in a given time step as defined by the Markov Decision Process (MDP) which describes an environment in RL [8]. The agent obtains feedbacks on the basis of the actions taken, known as the reward which acts as performance measure. A positive reward is the result of a good action, whereas a bad action leads to a negative reward or penalty.

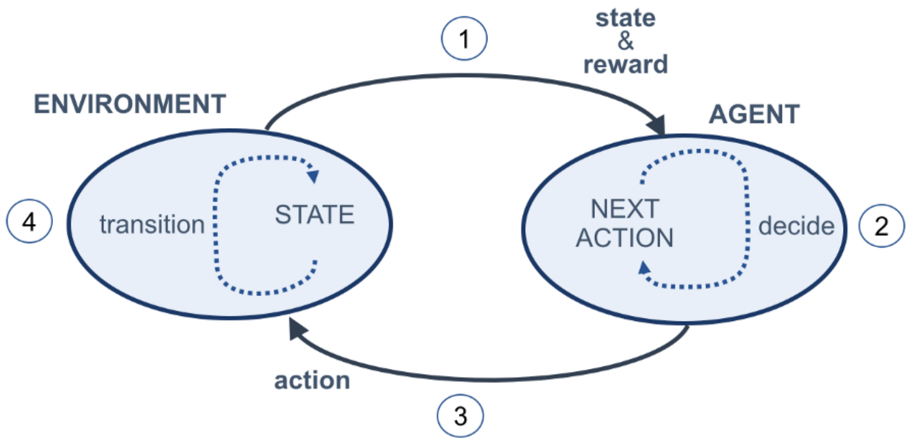


Fig. 1.1: Reinforcement learning cycle

The agent's main goal is to learn the optimal policy defining the behavior of the agent in order to maximize the cumulative reward. At the start of the learning process, the agent performs random actions, indication that the control policy is not known. However, as the agent explores the state-action space extensively and learns through time, it starts exploiting the learned policy and the tendency of choosing random actions declines. The Exploration vs. Exploitation trade-off is the term coined for the dilemma faced by the agent in the exploration of the new states while maximizing its overall reward simultaneously. The Epsilon-Greedy algorithm is the most used method to balance exploration and exploitation and performs actions by choosing between exploration and exploitation randomly [9]. Epsilon refers to the probability of choosing to explore and the algorithm chooses to exploit most of the time with a small chance of exploring.

1. **Types of Reinforcement Learning:** There are three types of RL methods: value-based, policy-based and actor-critic methods [10].
    a. ***Value-based methods:*** Value-based approach has the goal of finding an optimal

value function, which is the maximum value at a state under any policy and maximizing the expected future rewards. Q-learning is a value-based RL method [11].

   b. ***Policy-based methods:*** Policy-based approaches such as REINFORCE, directly approximate the optimal policy for the maximum future rewards without using the value function, through policy iteration. [12].

   c. ***Actor-Critic methods:*** Actor-critic integrates both value and policy-based approaches in a single RL framework, where the critic estimates the value function and the actor estimates the best policy [13] This approach uses two separate models for each of its parts.

2. **Q-Learning:** Q-learning [14] is an off-policy method in which the value learnt by the agent is based on an action derived from another policy. It is the most used model-free RL algorithm for traffic control [15]. The method updates the optimal Quality (Q)-value of each state-action pair iteratively according to the Bellman equation and stores in a Q-table in agreement with the respective state-action pairs, as shown in Figure 1.2. These Q-values are representative of the expected future rewards for each state-action pair.

3. **Deep Reinforcement Learning:** Deep Reinforcement Learning (DRL) is the combination of RL and Deep Learning (DL). In the case of traffic control systems, the state-action space is extremely huge as the environment is very complex and hence the Q-table may suffer from high-dimensionality. Thus, as the number of states increases, the amount of memory required to save and update the Q-table would increase and the amount of time required to explore each state to create the required Q-table would be unrealistic. As a solution, DL models such as Deep Neural Networks (DNN) and Convolutional Neural Networks (CNN) are used to approximate the Q-value efficiently, which eradicates the need for maintaining a Q-table in the RL framework. This concept is also known as Deep Q-Learning (DQL). Deep Q-Network (DQN) algorithm as shown in Figure 1.2, modifies Q-learning and uses a technique called Experience Replay (ER), in which the agent's experiences are stored at each time step in a data set combined over many episodes into a replay buffer memory [16].

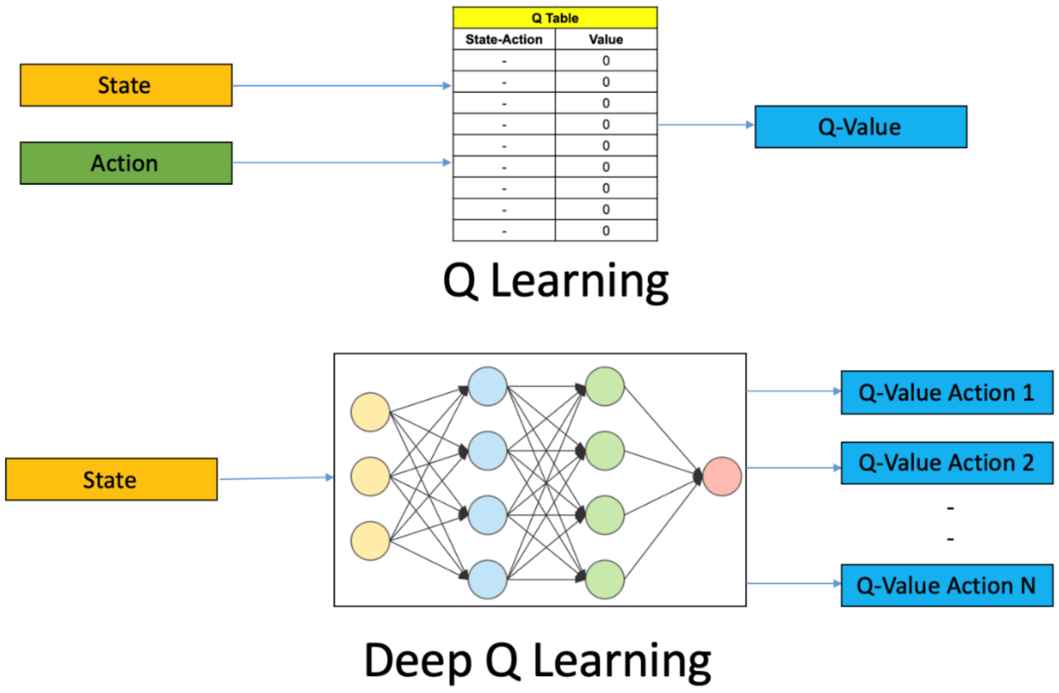


Fig. 1.2: Q-Learning vs. Deep Q-Learning

Problem Statement:

The objective of this project is to build a Deep Reinforcement Learning traffic light control agent to manage a traffic-lights regulated single four-way intersection as the environment provided by the Simulation of Urban Mobility (SUMO) simulator. The agent learns DRL algorithms to estimate the reward using Deep Neural Networks to perceive the current state of the traffic via images from the simulator. It then chooses the optimal action regarding the current traffic light phase from a fixed set of predetermined actions, based on the exploitation of its learning experience in order to maximize the reward and as a result, optimize the intersection's traffic efficiency.

Chapter 2
Literature Survey

Reinforcement learning has been used to successfully fulfil several challenging tasks in a variety of disciplines, including games, robotics [26], and traffic signal management. Various possible approaches using different algorithms and neural network structures have been proposed in the recent works on Reinforcement Learning to solve the traffic signal control problem.

IntelliLight, an intelligent traffic control system implemented by H. Wei et al [1], used a Deep Q-network framework to train the DRL agent. To resolve the difficulties in distinguishing the decision process for different traffic light phases, the study proposed a special sub-structure known as Phase Gate and also utilized the Memory Palace theory to improve the fitting capability of the network to predict the reward accurately. On real-world data, their approach IntelliLight is able to obtain the best reward, queue length, delay, and duration among baseline methods. K. Tan et al [17] also utilized the Deep Q-network algorithm to test their DRL-based adaptive traffic signal control framework that explicitly considers realistic traffic scenarios, sensors, and physical constraints. In this framework, they also proposed a novel reward function that shows significantly improved traffic performance compared to the typical baseline pretimed and fully-actuated traffic signals controllers. Their DRL agent's performance was tested on real traffic data during high traffic demand periods which outperformed both the baseline controllers. D. Li et al [16] built a truly adaptive traffic signal control model in a traffic simulator using the technology of modern deep reinforcement learning. The model proposed was based on Deep Q-network algorithm and both single-agent (for single 4-way intersection) and multiagent (for multiple intersections) cases were demonstrated. They tested with data sets pertaining to three different traffic conditions, and proved that their proposed model is better than other methods (e.g., Q-learning method, longest queue first method, and Webster fixed timing control method) for all cases. The proposed model reduces both the average waiting time and travel time, and it becomes more advantageous as the traffic environment becomes more complex.

A. Vidali et al [18] in their research work, adopted the Deep Q-learning mechanism to train two

DRL agents using two different reward functions and discussed the performance of both agents with a static traffic light benchmark. The cumulative wait time and the average wait time per vehicle were the metrics used to assess the difference in the performance of the two agents. Sajad et al [7] proposed two kinds of reinforcement learning algorithms: deep policy-gradient and value-function based agents to predict the best possible traffic signal. In their policy-gradient agent, there is mapping of observations to the control signal whereas in the value-functioned agent, first the values are estimated for all legal signals, and the optimal one is selected. They performed gradient descent on the policy parameters. In the value-functioned based approach, they used deep neural networks to estimate the action-value function, which maps the input state to the action values (i.e., the future reward that can be achieved for that given state and action). Liang et al [9] used data collected from different sensors and vehicular networks and proposed a model with prioritized experience replay. The general idea is that they extend the duration for the phase that has more vehicles in that direction, and hence this adaptive phase duration optimization is their key problem. They use state of the art techniques along with CNN and propose a network termed Double Dueling Deep Q Network.

Though all the above-mentioned studies learn a Q-function to map state and action to reward, they however differ in the state representation including features such as queue length [1, 19, 20], number of vehicles [1], average delay [1, 5, 21], average travel time [17] and features based on image representation of vehicles' position [1, 5, 9, 18, 22, 23]. The reward functions chosen also vary, containing average delay [1, 4, 7, 17, 22, 24], average travel time [5, 22], queue length [1, 17, 19], total waiting time [18].

Chapter 3

Work Done

Experimental Setting:

SUMO was used as the traffic micro-simulator for this project [25]. Simulation of Urban Mobility (SUMO) is a free and open-source traffic simulation software which includes an infrastructure editor and an application programming interface (API) and offers advanced features such as multimodal traffic, route generation, traffic light control, and a multitude of customizable road network maps. This software has been widely used to simulate all sorts of traffic conditions, and is used to optimize road networks in order to improve the flow of traffic, and reduce the waiting times. Results gathered from the simulations conducted in this software could be used to construct efficient traffic networks.

In particular this software was chosen because it can be interfaced directly from the command line and through python by using the TraCI library. Using this library, one can change all the parameters of the network, control traffic lights, and retrieve various observations in regard to the flow of traffic, and using these values to tune and manipulate the parameters instantly. In our project, first we created an empty road network by specifying all the nodes, junctions, traffic lanes, and the traffic light. Then we created a separate python file to generate traffic on this road, specifying the total number of cars and then bonded them together to create what is called a SUMO config file, which has moving traffic, and can be observed in real time.

In our earlier work, the easiest junction possible was chosen in order to create a baseline model. This junction consists of only two types of traffic – either East-West traffic or North-South traffic as shown in Figure 3.1. Each road has 2 lanes, and these roads have lengths equal to 750 meters, with the junction in the middle. The duration of green signal is chosen to be 8 seconds, and the duration of yellow signal is 3 seconds. These values were chosen after extensive testing and tuning, and the TraCI library can control whether to change the traffic signals or not.

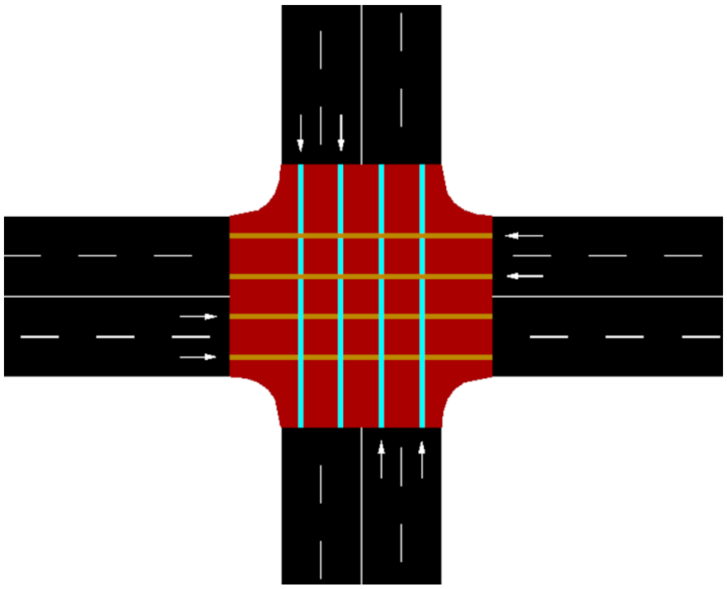


Fig. 3.1: The SUMO environment junction representing 2 types of traffic

Afterwards, this 4-way junction was extended for incorporating all types of right-hand side traffic instead of just unidirectional traffic (East-West traffic or North-South traffic) in order to model an environment close to a realistic traffic environment. The intersection modelled in SUMO now contains four lanes per road, with the traffic in the rightmost lane going either straight or to the right, cars in the middle two lanes only allowed to go straight and the vehicles in the leftmost lane are bound to take a left turn only as shown in Figure 3.2.

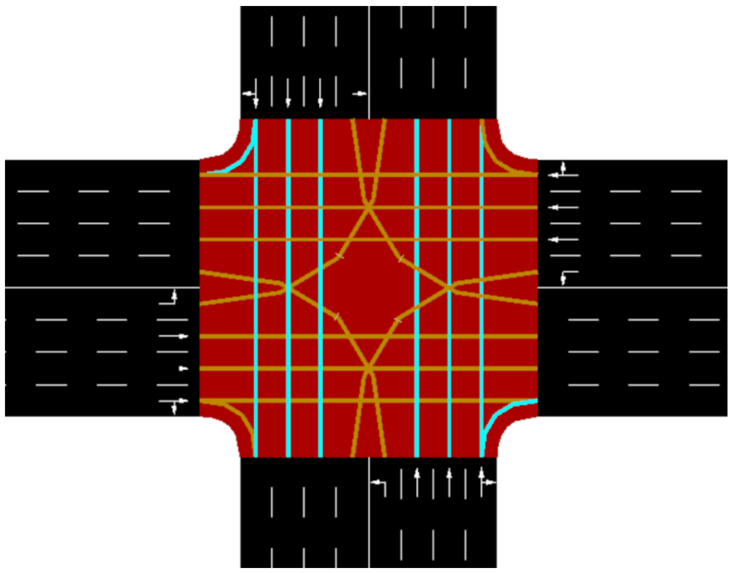


Fig. 3.2: The SUMO environment junction representing all types of traffic

The moving traffic at the junction is managed by a traffic light system in the center of the intersection controlled by the DRL agent. The three rightmost lanes on every arm have a shared traffic light whereas the fourth or the leftmost lane has a dedicated traffic light as seen in Figure 3.3. The duration of green signal and yellow signal is kept unchanged.

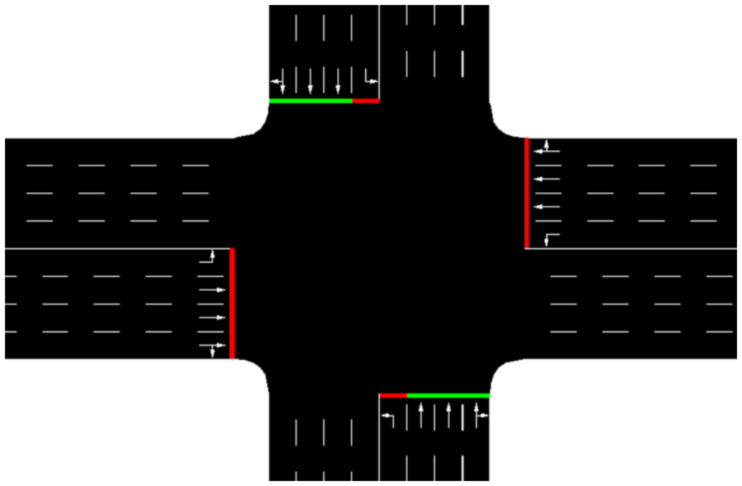


Fig. 3.3: Traffic lights in the SUMO realistic environment junction

We will use the term *"baseline environment"* for the SUMO environment junction representing two types of traffic and will refer to the environment represented in Figure 3.2 as *"realistic environment"*.

DRL approach description:

The reinforcement learning framework we used is defined as follows:

1. **State representation:** The representation of the traffic situation in the environment in a given action step t is denoted by s_t. The spatial information of the vehicles inside both the environment was used as the state representation as seen in Figure 3.4 and 3.5.

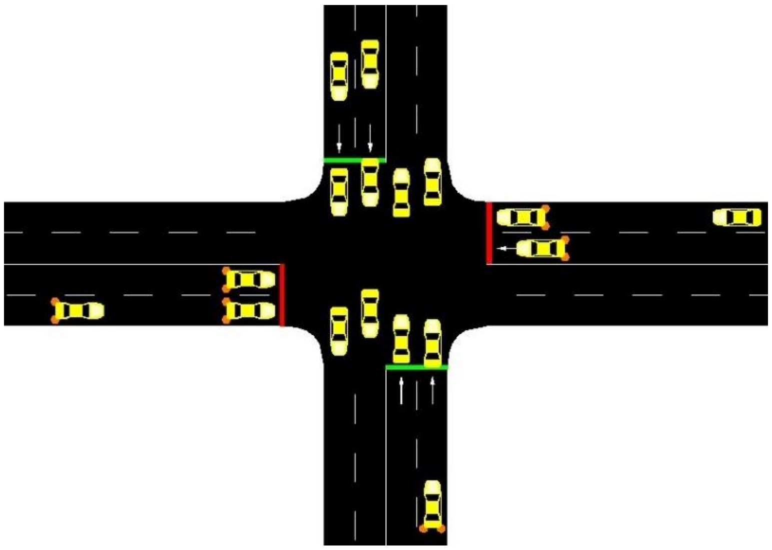


Fig. 3.4: Image representation of a state of the baseline environment

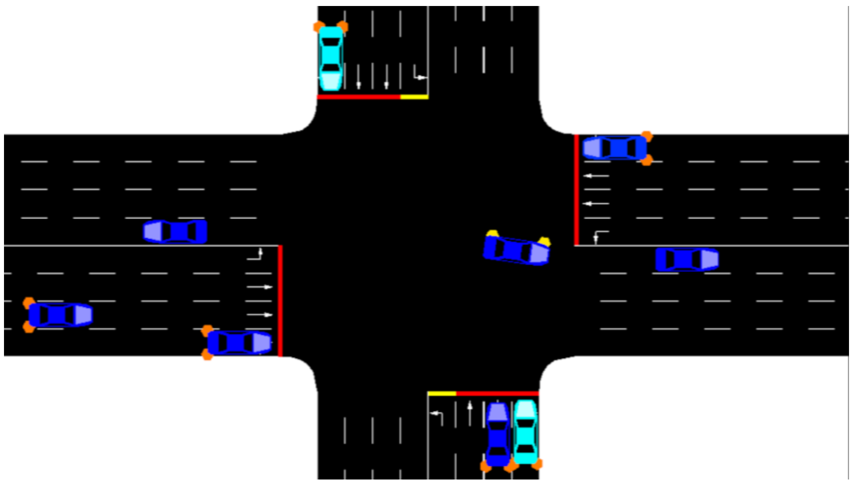


Fig. 3.5: Image representation of a state of the realistic environment

The continuous environment was made discrete by using cells to represent the presence and position of the vehicles on the lanes. As shown in Figure 3.6, each arm of the intersection in the baseline environment was discretized into ten cells and the state is represented as a binary matrix wherein the presence of a vehicle in a cell is denoted by a ‘1’ and the absence denoted by a ‘0’. Since movement direction of vehicles in both the lanes is straight, the ten cells cover both the lanes. Therefore, there are 40 cells in the entire intersection and hence the number of states is 40. Similarly, each arm of the intersection in the realistic environment was discretized into 20 cells, 10 of which cover the 3 lanes sharing the same traffic light and the remaining 10 cells contained in the leftmost lane as depicted in Figure 3.7. Hence, there are 80 cells in the whole intersection and 80 states in total for the realistic environment.

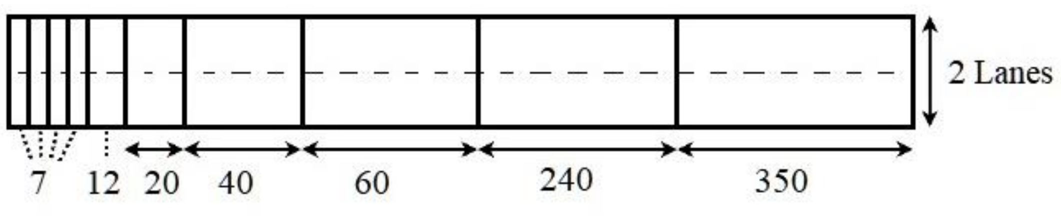


Fig. 3.6: State representation design of the east arm of the intersection in the baseline environment

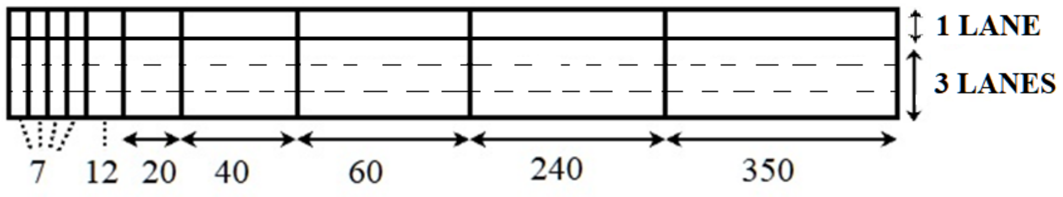


Fig. 3.7: State representation design of the east arm of the intersection in the realistic environment

However, the cells aren't the same size: the cell length increases as its distance from the intersection increases because the main motive is to detect standing vehicles near to the intersection. This also decreases the computational complexity which would have arisen in the case of equal and short sized cells. The length of the shortest cells is 7 meters which is 2 meters extra than the length of the cars (5 meters). In conclusion, on observing the status of the environment, the agent will acquire a set of cells describing the presence or absence of cars in each section of the approaching roads.

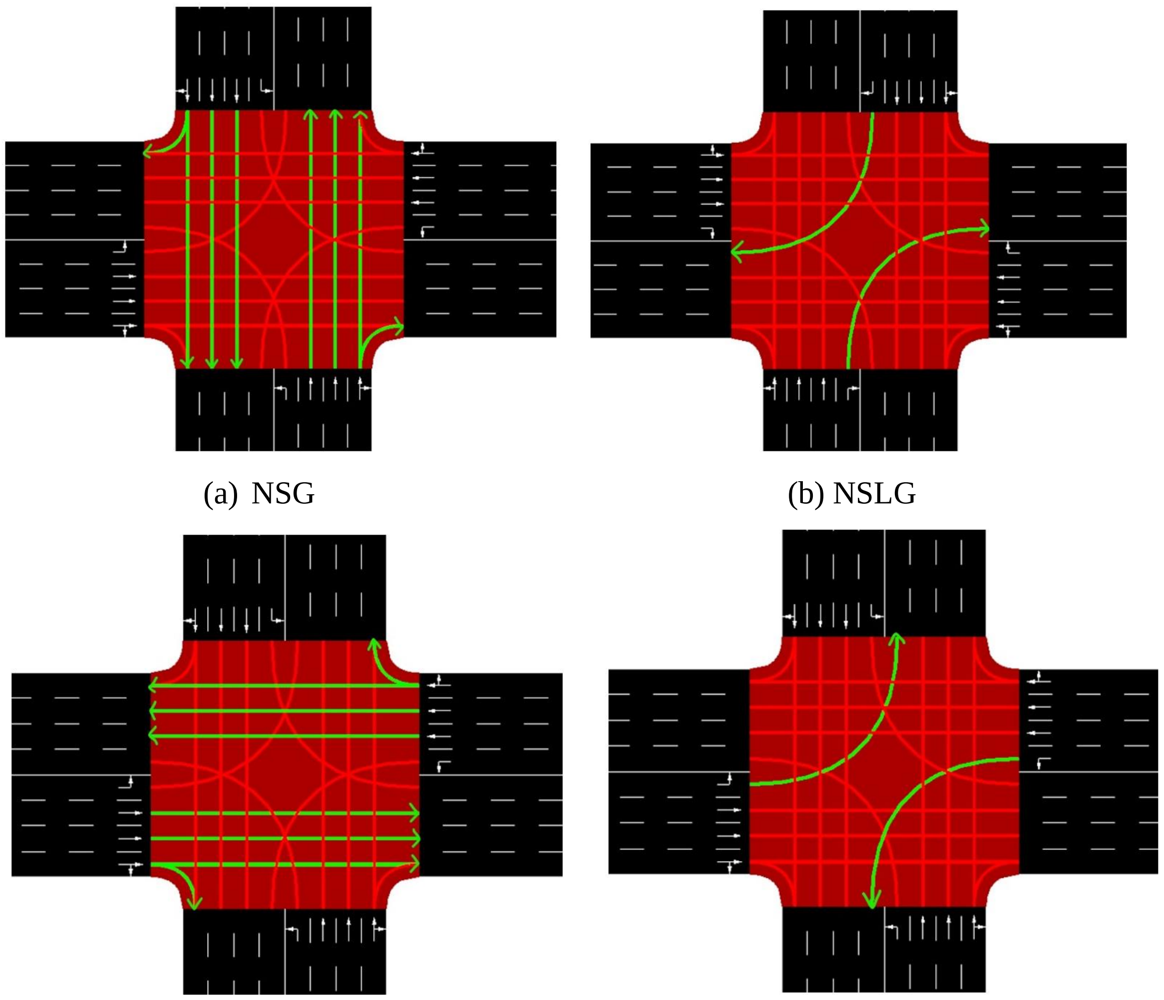
2. **Action Set:** The action space is different for the both the environments. For the baseline environment, since there is only straight traffic allowed, the set of 2 possible actions that the agent can choose from is defined in (1).

$$A = \{NSG, EWG\} \qquad (1)$$

On the other hand, the action space for the realistic environment consists of 4 possible actions and is defined in the set (2).

$$A = \{NSG, NSLG, EWG, EWLG\} \qquad (2)$$

North-South Green (NSG) means that the green traffic phase is active for vehicles in the north-south intersection arm which want to go straight or turn right whereas in East-West Green (EWG), the green traffic phase is active for vehicles in the east-west intersection arm which want to go straight or turn right. In North-South Left Green (NSLG), the green traffic phase is active for vehicles in the north-south intersection arm which want to turn left whereas in East-West Left Green (EWLG), the green traffic phase is active for vehicles in the east-west intersection arm which want to turn left. However, if the two actions chosen at consecutive action steps are not the same, then a yellow phase is introduced in between. Figure 3.8 depicts the 4 possible actions pictorially.


(a) NSG

(b) NSLG

(c) EWG

(d) EWLG

Fig. 3.8: Pictorial representation of possible actions

3. **Reward Function:** The reward function uses the accumulated total waiting time as the metric as defined in equations (3) and (4).

$$atwt(t) = \sum_{v=0}^{n} awt(v,t) \qquad (3)$$

$$r(t) = \mathrm{atwt(t-1)} - atwt(t) \qquad (4)$$

Where v represents a vehicle, n is the total number of vehicles in the environment at action step t, and awt(v, t) is the amount of time (in seconds) a vehicle v has a speed of less than 0.1m/s at action step t, since its arrival in the environment. In equation (4), r(t) is the reward at action step t, and atwt(t) represents the accumulated total waiting time of all the vehicles in the intersection at action step t. The same reward function is used for both the environments.

Deep Q-Networks:

The agent learned a Deep Q-Network (DQN) to predict the reward. In other words, the learning mechanism used here was Deep Q-Learning (DQL), which is an augmentation of the value-based Q-learning algorithm. In Q-learning, the Q-value is updated by using the newly received sample (s_t, a_t, s_{t+1}, r_t) according to the Bellman equation as described in equation (5).

$$Q^{new}(s_t, a_t) = \mathrm{Q}(s_t, a_t) + \alpha(r_t + \gamma . \max_a \mathrm{Q}(s_{t+1}, a) - \mathrm{Q}(s_t, a_t)) \quad (5)$$

where $\mathrm{Q}(s_t, a_t)$ is the action a_t taken from state s_t at action step t or the current Q-value of state–action pair, $Q^{new}(s_t, a_t)$ is a new Q-value, α is learning rate, r_t is the reward associated to taking action a_t from state s_t, received from the environment, γ is discount factor ($0 < \gamma < 1$) which can be static or change over time in order to model a process in which the earlier rewards are worthier than the future rewards, $\max_a \mathrm{Q}(s_{t+1}, a)$ is the estimate of optimal future value and a is any action from the possible action space. The term $(r_t + \gamma . \max_a \mathrm{Q}(s_{t+1}, a))$ is known as the learned value.

Since we used DQL, the new Q value was approximated by the DNN model implemented with weights θ and is represented as $\mathrm{Q}(s_t, a_t;\ \theta)$. The DNN model is learned to minimize the expected squared error (loss) between the predicted Q-value and the target Q-value, as described by the following equation (6):

$$L = \mathrm{E}\left[\left(r_{t+1} + \gamma . \max_{a} \mathrm{Q}(s_{t+1}, a) - \mathrm{Q}(s_t, a_t; \theta)\right)^2\right] \qquad (6)$$

A fully-connected DNN for the baseline environment was built with an input layer of 40 neurons (as there are 40 state representations), 5 hidden layers with varying number of neurons each using Rectified Linear Unit (ReLU) as the activation function and an output layer with 2 neurons with linear activation function as there are 2 possible actions given a state. The DNN model weight-updating was done based on the Adam optimization method in every learning iteration to induce the minimization of equation (6). This DNN model structure which we will call as model 1, is shown in Figure 3.9.

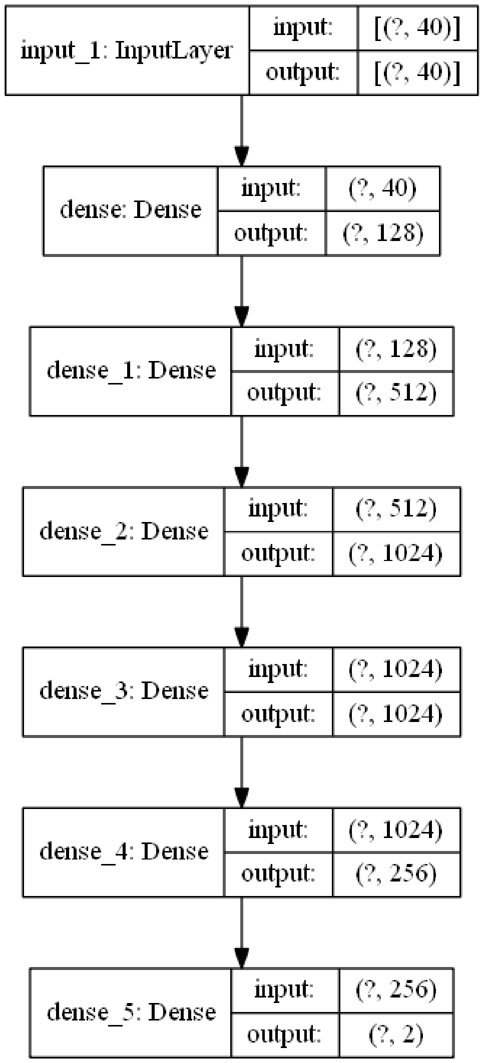


Fig. 3.9: DNN model 1 structure for baseline environment

For the realistic environment, a baseline DNN model which we will refer to as model 2 and another DNN model (model 3) was built with an input layer of 80 neurons (as there are 80 state representations), 5 hidden layers and an output layer with 4 neurons as there are 4 possible actions given a state. ReLU activation function was used in all the hidden layers whereas linear activation function was used in the output layer as before. Again, Adam optimization method was deployed for updating the model weights. The only difference between the two models is the number of neurons in each hidden layer. Figure 3.10 and 3.11 show the DNN model 2 and model 3 structures respectively.

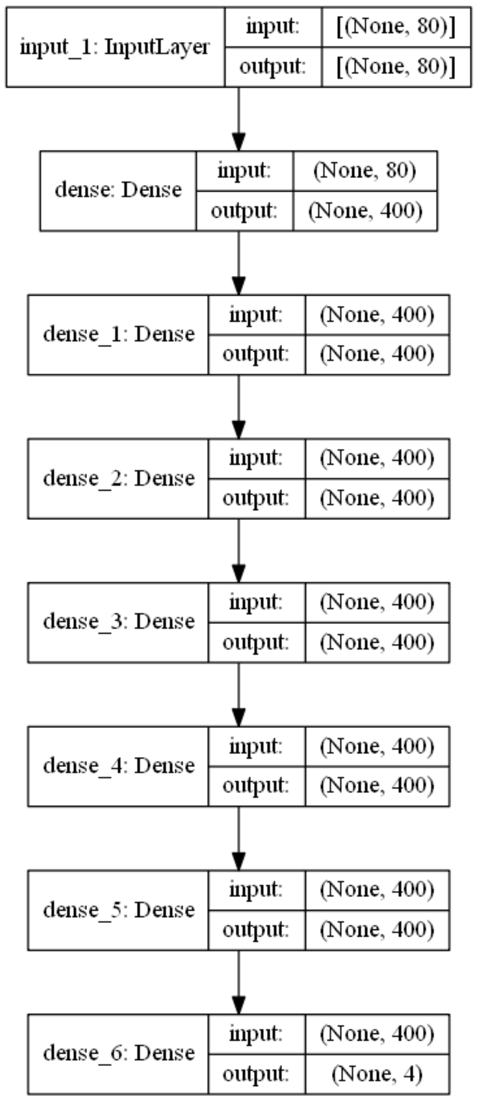


Fig. 3.10: DNN model 2 structure for realistic environment

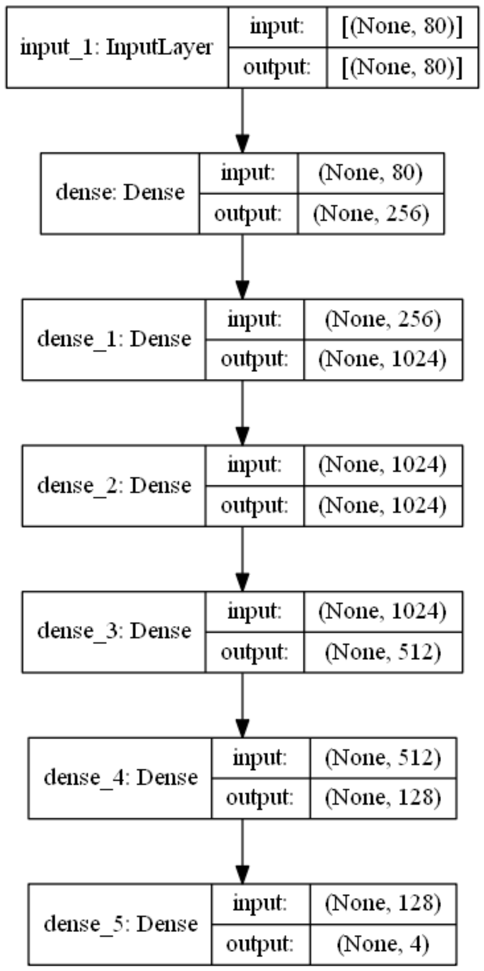


Fig. 3.11: DNN model 3 structure for realistic environment

Table 3.1 summarizes the hyperparameter values of the DNN model as used in this study.

Table 3.1: DNN model hyper-parameter details

S. No.	**Parameter**	**Value**
1	Depth	5
2	Batch size	100
3	Number of epochs	800
4	Learning rate	0.0001

Training Process:

The entire training was done across 100 or 150 episodes and the period of each episode was set at 2 hours. Since the time frequency in SUMO is 1 second per step, 1 simulation step corresponds to 1 second and hence the maximum number of steps per episode is 7200. We arrived at these values by trial-error method by starting the training from 100 episodes of half hour each and then 125 episodes of 1 hour each and comparing the results at the end. In each episode the traffic was generated according to a Weibull distribution with a shape of 2 in order to maintain a high degree of reality. Every generated vehicle had similar physical dimensions and performance in terms of acceleration and speed, and a total of 1250 or 1500 vehicles were generated for the training purpose and distributed evenly on each intersection arm. In order to mimic the conditions of real-life driving, each car has an acceleration of 1 m/s^2, a deceleration of 4.5 m/s^2, and a top speed of 25 m/s. During each traffic scenario, 75% of the total vehicles go straight and the remaining 25% take a left or right turn.

The technique of Experience Replay was adopted during the training phase where the agent gets randomized samples in the form of batches from a buffer memory for the purpose of learning as demonstrated in Figure 3.12 below which shows the workflow of the DRL agent in the form of a flow chart. This improves the learning efficiency and performance of the agent. The memory size which represents the maximum number of samples that can be stored, was set at 50000 samples. Further, the most used ε-greedy exploration policy was employed for action-selection where the value of ε was set according to equation (7).

$$\varepsilon_n = 1 - \frac{n}{N} \tag{7}$$

where n is the current episode of training and N in the total number of episodes. The algorithm explores with a probability ε and initially ε = 1, indicative of the agent exploring. However, with the progress in training, ε keeps decreasing and the agent exploits its learning experience with a probability 1- ε.

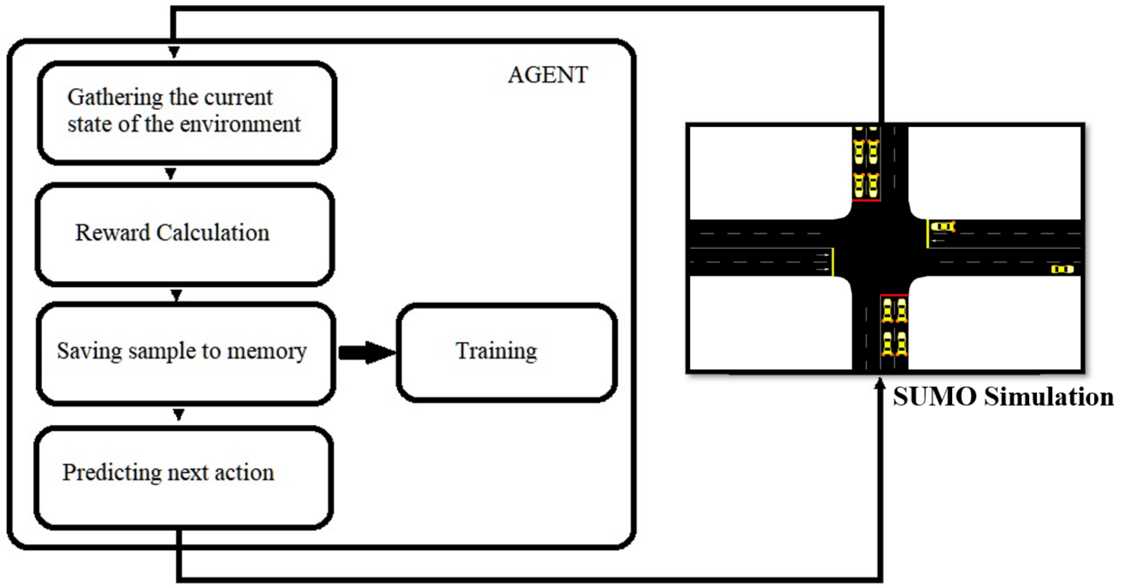


Fig. 3.12: DRL Agent’s workflow

Training Results:

The performance of the agent was analyzed on the basis of the reward curves during the training and the average queue length (sum of the number of waiting cars) across all episodes.

Table 3.2 and 3.3 summarize the hyperparameter values of the DQN agents 1 and 2 and the SUMO simulation as used in this project. DQN Agent 1 operates in the baseline environment and it approximates the Q values using the DNN model 1 as defined in Figure 3.9. On the contrary, DQN Agent 2 uses the DNN model 2 defined in Figure 3.10 to approximate the Q values and it functions in the realistic environment.

Figure 3.13 and 3.16 show the learning improvement during the training in terms of cumulative negative reward which is the magnitude of the negative outcomes of the actions during each episode for DQN Agents 1 and 2 respectively. The reward curves are not significantly unstable, indicating that the agents were able to learn a sufficiently correct policy in both the environments.

Figures 3.14 and 3.17 display the curve of average queue length of vehicles as the training proceeds and the decreasing nature of the curves indicates that the agents achieved success in clearing traffic congestion by minimizing the number of waiting cars at the junction.

Figures 3.15 and 3.18 display the cumulative delay curves of vehicles in seconds across the entire training process and it can be inferred from the decreasing nature of the curves that the agents succeeded in the cumulative delay or cumulative waiting time per vehicles as well. The cumulative waiting time is defined as the sum of all waiting times of every vehicle during the episode.

Table 3.2: Hyper-parameter details of DQN Agent 1 and SUMO simulation

	Parameter	**Value**
DQN Agent 1 (Uses DNN model 1)	Number of states	40
	Number of actions	2
	Discount factor (γ)	0.95
	Learning rate (α)	0.0001
SUMO	Episodes	150
	Steps per episode	7200
	Cars generated	1250

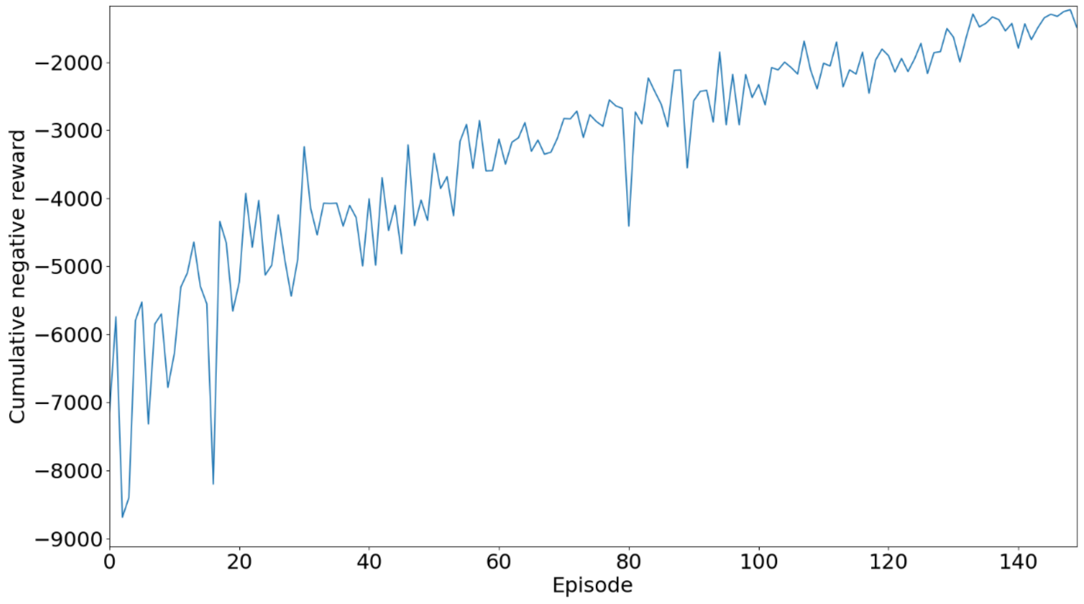


Fig. 3.13: Cumulative negative reward per episode during training for DQN Agent 1

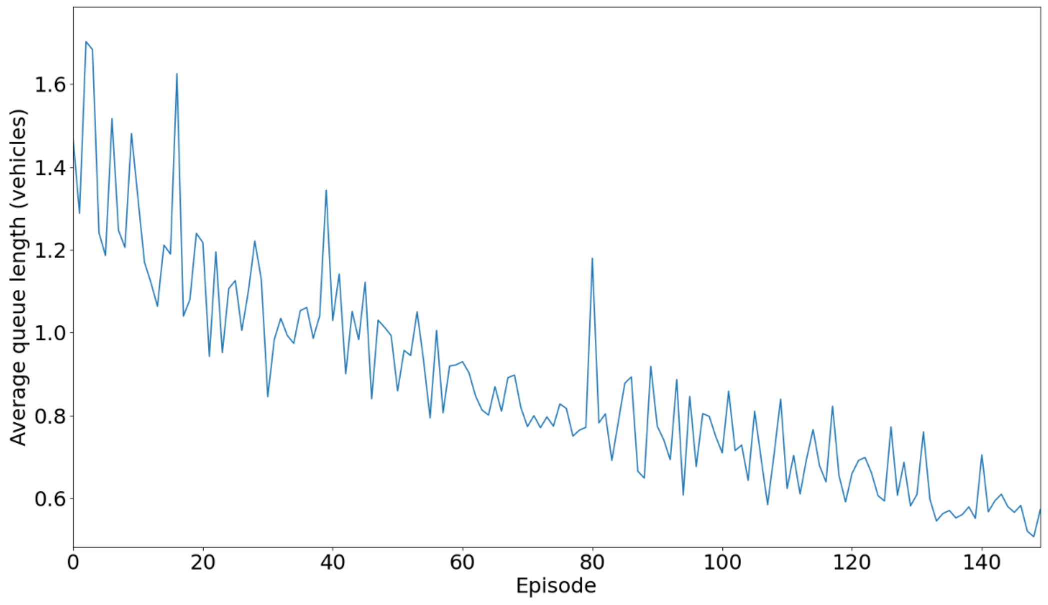


Fig. 3.14: Average queue length of vehicles per episode during training for DQN Agent 1

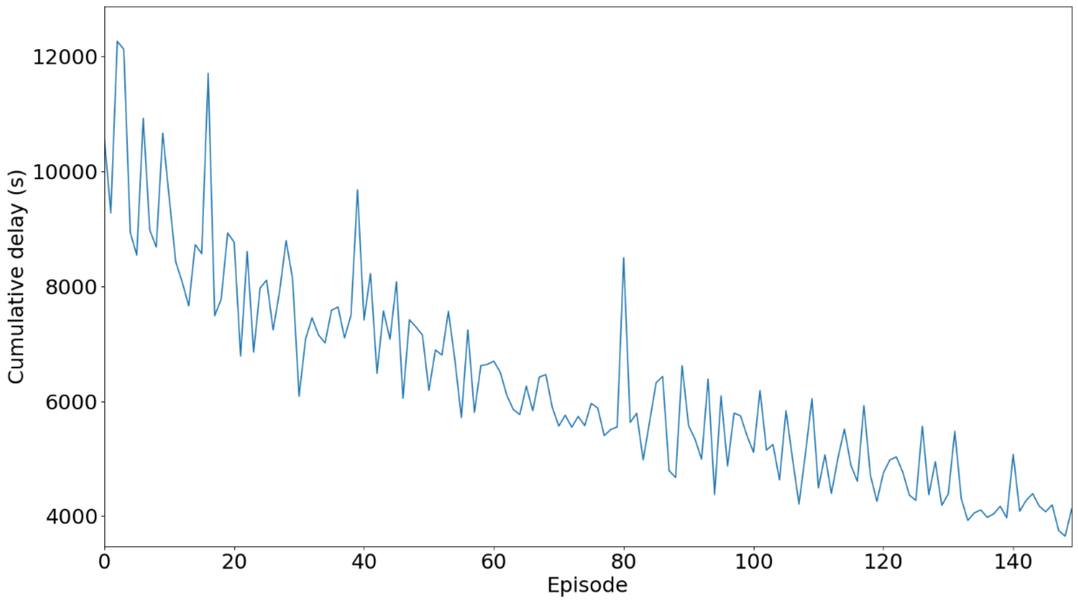


Fig. 3.15: Cumulative delay of vehicles in seconds per episode during training for DQN Agent 1

Table 3.3: Hyper-parameter details of DQN Agent 2 and SUMO simulation

	Parameter	**Value**
DQN Agent 2 (Uses DNN model 2)	Number of states	80
	Number of actions	4
	Discount factor (γ)	0.75
	Learning rate (α)	0.0001
SUMO	Episodes	100
	Steps per episode	7200
	Cars generated	1000

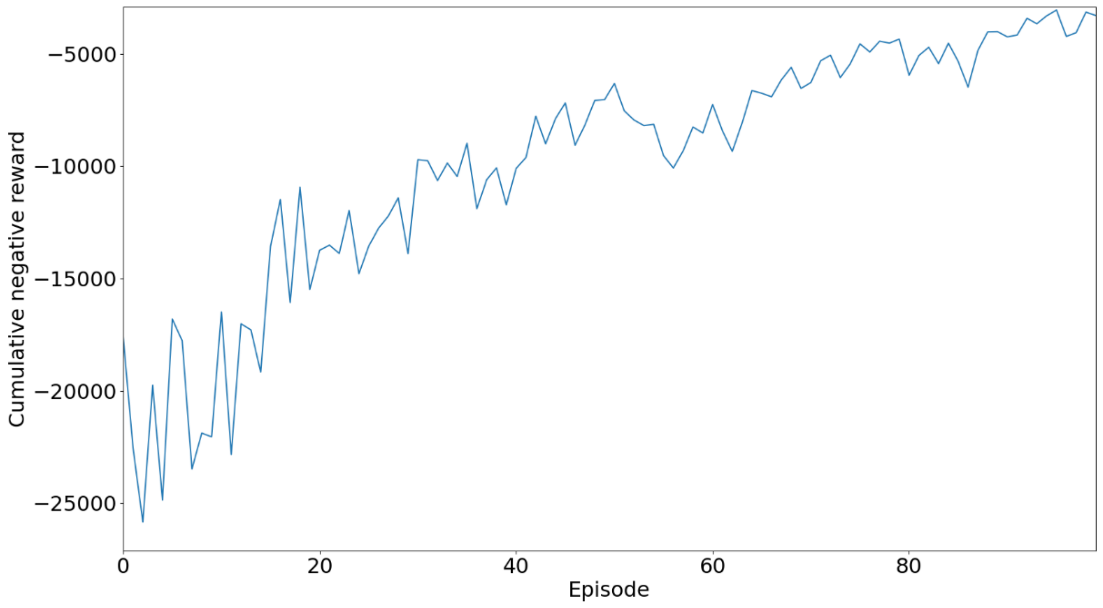


Fig. 3.16: Cumulative negative reward per episode during training for DQN Agent 2

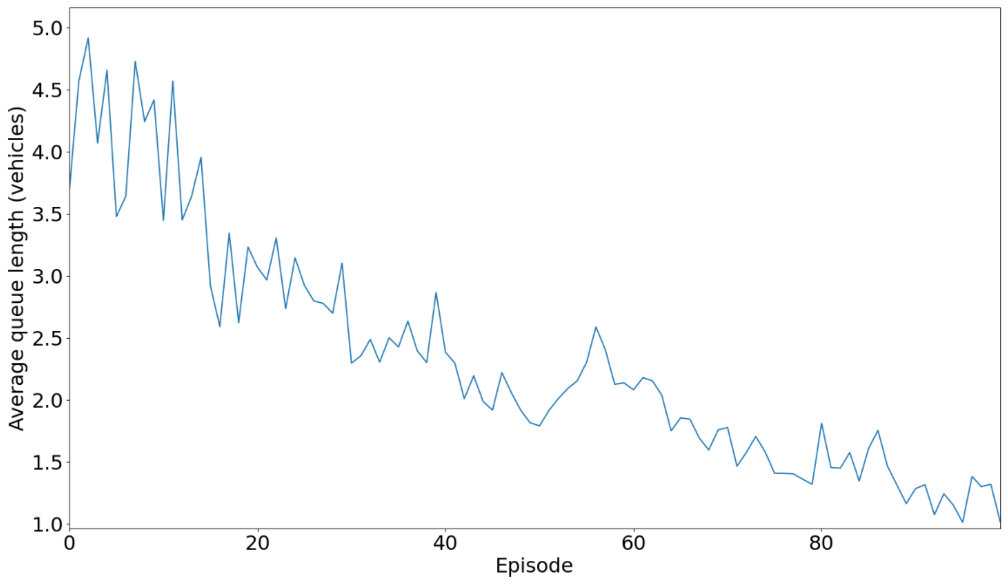


Fig. 3.17: Average queue length of vehicles per episode during training for DQN Agent 2

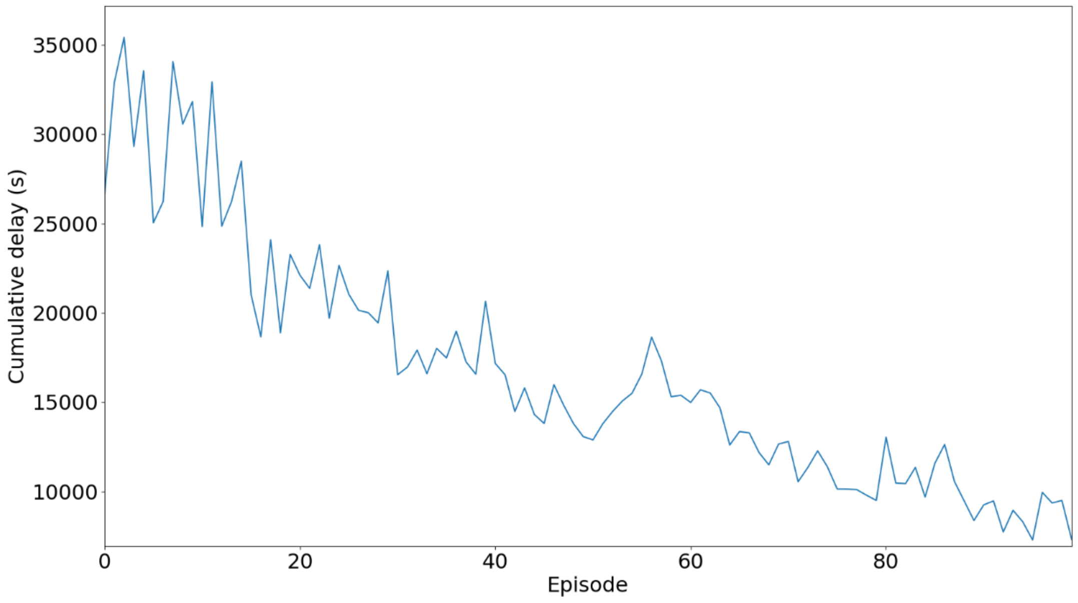


Fig. 3.18: Cumulative delay of vehicles in seconds per episode during training for DQN Agent 2

Testing Results:

The results obtained after testing the agent are shown in Figures 3.19 - 3.22 representing the reward across all action steps and queue lengths at each simulation step for both the agents.

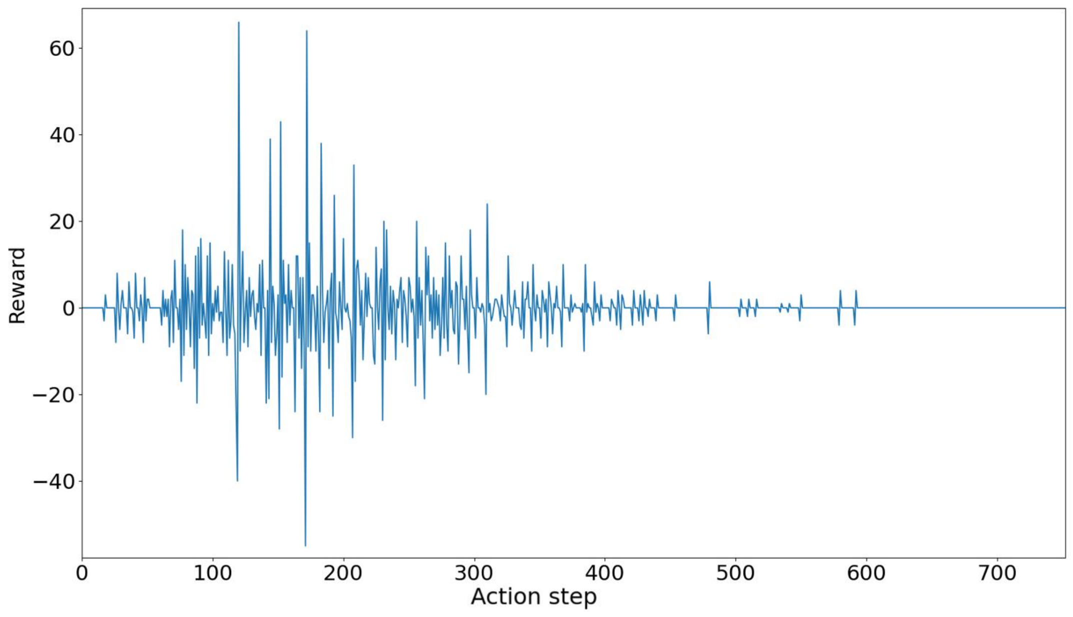


Fig. 3.19: Reward of agent per action step during testing for DQN Agent 1

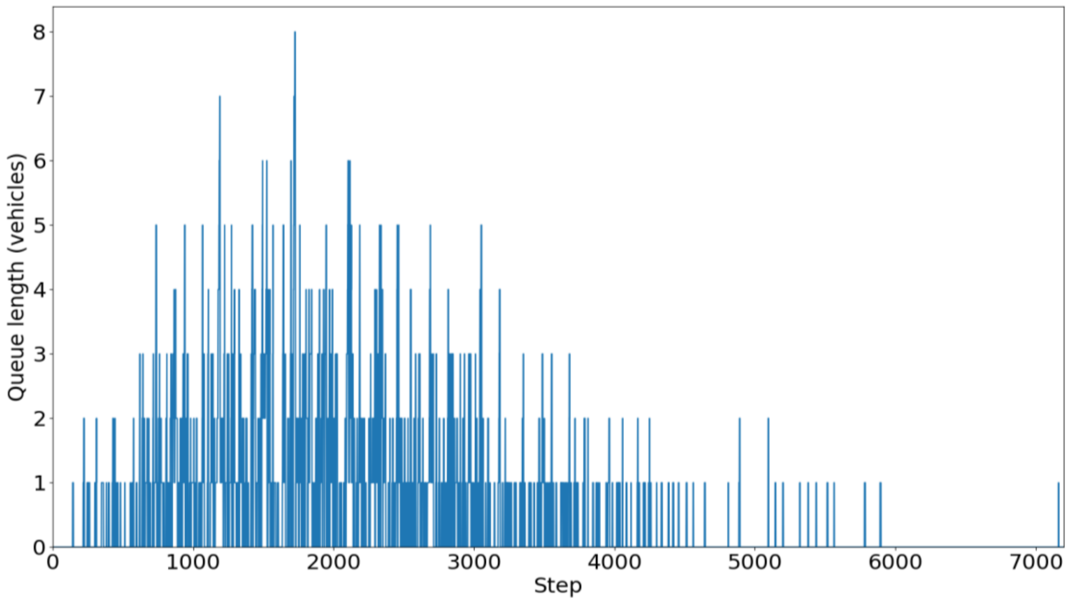


Fig. 3.20: Queue length of vehicles per simulation step during testing for DQN Agent 1

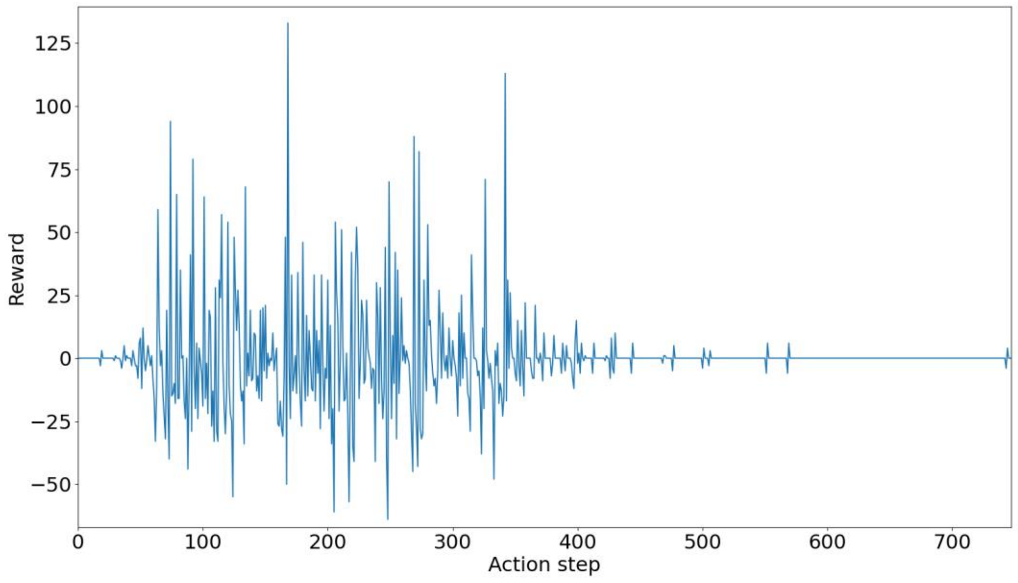


Fig. 3.21: Reward of agent per action step during testing for DQN Agent 2

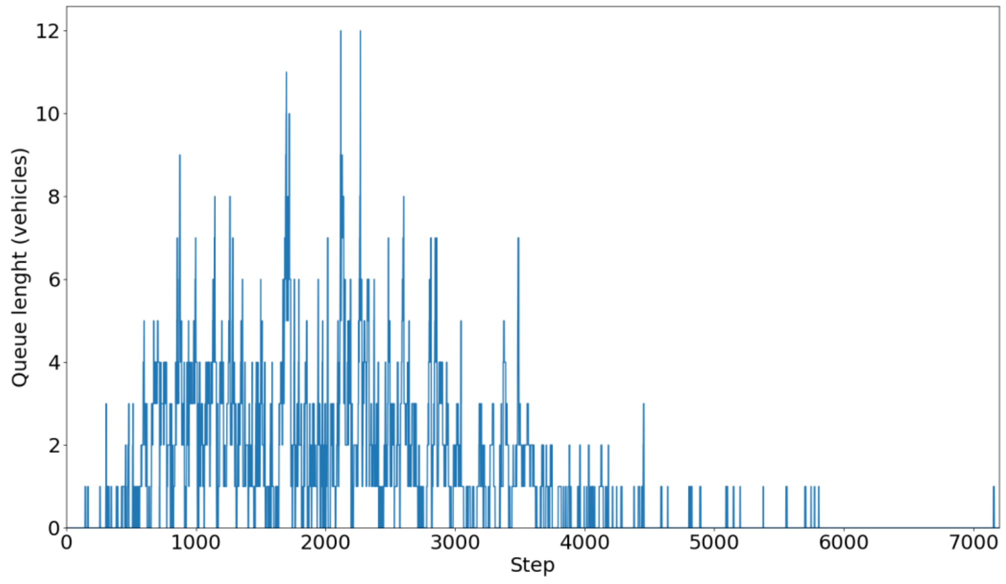


Fig. 3.22: Queue length of vehicles per simulation step during testing for DQN Agent 2

The test results illustrate that both the DQN agents are able to attain sufficiently higher rewards and also manage to queues of vehicles below 8 and 12 queues in the baseline and realistic environments respectively throughout the whole duration of traffic simulation.

Hyper-parameter Tuning:

We further tried investigating the effect of different hyperparameter values for discount factor and learning rate on the DQN agent's performance in the realistic environment. In order to draw a comparison between the performance of the agents with different hyper-parameter values, the same parameter settings were kept for the simulation. The training was divided into 150 episodes of 2 hour each for each of the agents and 1500 vehicles were generated in each episode. The DQN agents estimate the Q value using the DNN model 3 whose structure is depicted in Figure 3.11. Table 3.4 lists the hyper-parameter values for all the agents.

Table 3.4: Hyper-parameter details of DQN Agents for performance comparison

	Discount factor **(γ)**	**Learning rate (α)**
DQN Agent 3	0.75	0.001
DQN Agent 4	0.75	0.0001
DQN Agent 5	0.99	0.0001

Figures 3.23 and 3.24 show the learning improvement during training of the agents. Figure 3.23 compares the reward curve for the agents with different discount factors whereas Figure 3.24 compares the reward curve for the agents with different learning rates.

The discount factor helps in making the cumulative sum of rewards bounded as it scales down the rewards more and more after each step and can also adjust the agent's behavior to achieve long term goals. A discount factor value close to 0 causes the agent to base its decisions only on selecting actions by taking care of an immediate reward and this is known as a myopic case. This behavior can badly influence further future rewards. On the contrary, a discount factor value close to 1 lets the agent weight future rewards stronger and thus is called the for-sight case. Hence, generally a value greater than 0.75 is chosen as the discount factor. However, it has shown that applying Reinforcement learning algorithms with a lower discount factor can act as regularizer and improve performance [27]. This method is referred to as discount regularization. However, the performance comparison in Figure 3.23 shows that both the discount factors are able to produce a stable policy and both the agents perform well.

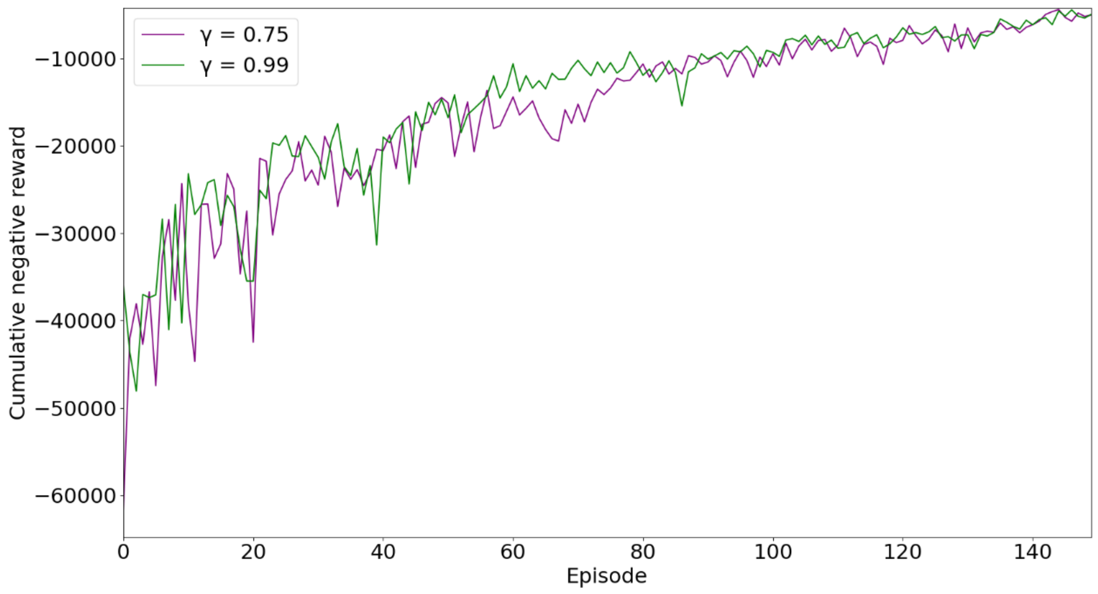


Fig. 3.23: Cumulative negative reward of vehicles per episode during training for DQN Agents 3 and 5

The performance comparison in Figure 3.24 for a lower and higher learning rate value for a constant discount factor of 0.75 also indicates that both the agents are able to learn a more or less stable policy and perform at par with each other.

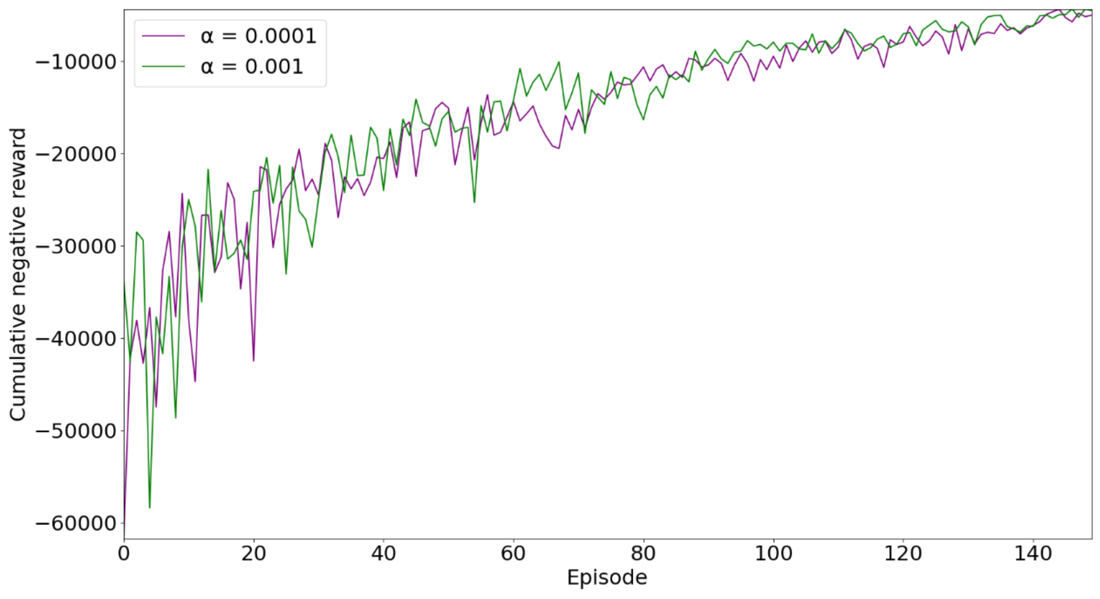


Fig. 3.24: Cumulative negative reward of vehicles per episode during training for DQN Agents 3 and 4

Chapter 4

Conclusion

In this study, we simulated road traffic conditions of a high-density traffic scenario, in which we considered 2 types of 4-way intersections – single direction of movement of vehicles in which the cars are not allowed to change direction by taking right or left turns, and a multidirectional intersection where the vehicles are free to take right or left turns, given that they are in the appropriate lanes before turning. The simulation was performed using an open-source program called Simulation of Urban Mobility (SUMO), and the code for the deep reinforcement learning was written using python.

Table 3.5: Performance of Agent w.r.t Delay, Queue Length and Reward

	Cumulative Wait Time (s)	**Average Queue Length of all Lanes**	**Reward**
DQN Agent 1	3654	0.5075	-1229
DQN Agent 2	7286	1.0119	-3051
DQN Agent 3	13827	1.9204	-4387
DQN Agent 4	13299	1.8470	-4427
DQN Agent 5	13875	1.9270	-4337

Table 3.5 lists the final results of the various DQN agents trained using different settings and different hyperparameters. For a simple 4 lane intersection with only single direction traffic, our agent was able to significantly reduce the cumulative waiting time and the queue length for all the 1000 cars in that setting. For the 4-way intersection with 4 lanes in each direction, 2 hyperparameters that were tested were the learning rate and the discount factor. DQN Agent 4 with lesser learning rate always had an average queue length and cumulative wait time lower than the other DQN agents 3 and 5, where 150 episodes with 1500 cars were simulated. On analyzing the reward curves, it is clearly illustrated that all the DRL agents were able to learn a stable policy and perform well. In essence, it is evident that all the DRL agents were able to outperform the classical pre-timed traffic control system and can further be improved upon.

Chapter 5
Future Prospects

So far, the experimentation was performed on a multi-lane 4-way intersection. In the real world, there are different types of intersections, with different types of priorities, or emergency situations that require specific lanes to be cleared. So in the future, such miscellaneous cases of different structures of intersections could be taken, be it 3-way intersections, round-abouts or intersections with flyovers, and the experimentation and testing could be performed on such complex structures too.

Also, we have only considered a single intersection in our study, but this task of using deep reinforcement learning could be extended to small colonies, or cities, where a network of such interconnected junctions are present, and the same set of vehicles travel among these intertwined networks. For this purpose, there is also a requirement for very strong computing hardware, but it indeed is possible to train an agent to control an array of traffic lights, in order to better manage the traffic.

Lastly, we have used the value-based reinforcement learning technique where a Q Network is trained by updating the Q values of action - state pairs, for each episode. However, apart from improving the results in the study using the same methods, there are multiple other reinforcement learning algorithms such as Actor Critique methods, policy-based methods, etc., that if investigated and experimented properly, would provide much better results.

In order to find out the real-world advantages of such an advanced interconnected deep reinforcement learning based traffic light control system, there is a need to experiment on this problem in regards to the three points mentioned above. Based on the data collected, the agent would then continue to learn more patterns and accordingly, one could alter, innovate or optimize the architecture to further improve the results.

References

[1] H. Wei et al, "IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control," Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2496–2505, July 2018.

[2] Alan J Miller., "Settings for fixed-cycle traffic signals.", Journal of the Operational Research Society 14, vol. 4, pp. 373–386, 1963.

[3] F. V Webster. 1958. Traffic signal settings. Road Research Technical Paper 39, 1958.

[4] Lior Kuyer, Shimon Whiteson, Bram Bakker, and Nikos Vlassis, "Multiagent reinforcement learning for urban traffic control using coordination graphs," Machine learning and knowledge discovery in databases, pp. 656–671, 2008.

[5] Elise van der Pol and Frans A Oliehoek., Coordinated Deep Reinforcement Learners for Traffic Light Control. NIPS, 2016.

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

[7] Sajad Mousavi, S., Schukat, M. and Howley, E., "Traffic Light Control Using Deep Policy-Gradient and Value-Function Based Reinforcement Learning," arXiv e-prints, pp. arXiv-1704, 2017.

[8] Martin et al., "Application of Deep Reinforcement Learning in Traffic Signal Control: An Overview and Impact of Open Traffic Data", Journal of Applied Sciences, vol. 10, issue 4011, 2020.

[9] Liang, X., Du, X., Wang, G. and Han, Z., "Deep reinforcement learning for traffic light control in vehicular networks," arXiv preprint, arXiv:1803.11115, 2018.

[10] Sutton, R. S., and Barto, A. G., 2018. "Reinforcement learning: An introduction".

[11] Watkins, C. J., and Dayan, P., "Q-learning," Machine learning, vol. 8, issue 3-4, pp. 279–292, 1992.

[12] Williams, R. J., "Simple statistical gradient following algorithms for connectionist reinforcement learning". Machine learning, vol. 8, issue 3-4, pp. 229–256, 1992.

[13] Konda, V. R., and Tsitsiklis, J. N., "Actor-critic algorithms," In Advances in neural information processing systems, pp. 1008–1014, 2000.

[14] C. J. Watkins and P. Dayan, "Q-learning," Machine learning, vol. 8, issue 3-4, pp. 279–292, 1992.

[15] Chin, Y.K.; Kow, W.Y.; Khong, W.L.; Tan, M.K.; Teo, K.T.K., "Q-Learning Traffic Signal Optimization within Multiple Intersections Traffic Network," In Proceedings of the 2012 Sixth UKSim/AMSS European Symposium on Computer Modeling and Simulation, Valetta, Malta, pp. 343–348, November 2012.

[16] Li, D., Wu, J., Xu, M., Wang, Z., Hu, K., "Adaptive Traffic Signal Control Model on Intersections Based on Deep Reinforcement Learning," J. Adv. Transp. 2020, 2020.

[17] K. Tan et al, "Deep Reinforcement Learning for Adaptive Traffic Signal Control," Proceedings of the ASME 2019, Dynamic Systems and Control Conference DSCC2019, November 2019.

[18] A. Vidali, L. Crociani, G. Vizzari, and S. Bandini, "A deep reinforcement learning approach to adaptive traffic lights management," in Proc. Workshop 'From Objects Agents', pp. 42–50, 2019.

[19] L. Li, Y. Lv, and F.-Y. Wang, "Traffic signal timing via deep reinforcement learning," IEEE/CAA Journal of Automatica Sinica, vol. 3, issue 3, pp. 247–254, 2016.

[20] Patrick Mannion, Jim Duggan, and Enda Howley, "An experimental review of reinforcement learning algorithms for adaptive traffic signal control," In Autonomic Road Transport Support Systems, Springer, pp. 47–66.

[21] W. Genders and S. Razavi, "Using a deep reinforcement learning agent for traffic signal control," arXiv preprint arXiv:1611.01142, 2016.

[22] Mengqi Liu, Jiachuan Deng, Ming Xu, Xianbo Zhang, and Wei Wang, "Cooperative Deep Reinforcement Learning for Tra ic Signal Control.", 2017.

[23] Gao, J., Shen, Y., Liu, J., Ito, M., and Shiratori, N., "Adaptive traffic signal control: Deep reinforcement learn learning algorithm with experience replay and target network,". arXiv preprint arXiv:1705.02755, 2017.

[24] Itamar Arel, Cong Liu, T Urbanik, and AG Kohls. 2010. Reinforcement learning-based multi-agent system for network traffic signal control. IET Intelligent Transport Systems, vol. 4, issue 2, pp. 128–135, 2010.
[25] D. Krajzewicz, G. Hertkorn, C. R¨ossel, and P. Wagner, "Sumo (simulation of urban mobility)-an open-source traffic simulation," in Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002), pp. 183–187, 2002.
[26] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke et al., "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," arXiv preprint arXiv: 1806.10293, 2018.
[27] R. Amit et. al., "Discount Factor as a Regularizer in Reinforcement Learning", Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.