Traffic Control System

A Project Report
Presented to
Dr Jahan Ghofraniha

San Jose State University
for coursework
Reinforcement Learning

By

Shreyus Puthiyapurail
Atul Shah

Table of Contents

Executive Summary

The goal of this project is to create an efficient traffic control system which uses state of the art Deep Reinforcement Learning models. The outcome of such a system will help reduce traffic bottlenecks and reduce wait times for the cars at the intersection.

Background/Introduction

As the population of cities across the globe increases, we are observing more traffic bottlenecks in urban areas. In the coming years, this problem is going to get worse. There is an immediate need to make the traffic control system more streamlined and efficient. The modern society requires an efficient transportation system ensuring smooth vehicle movement. However, increase in population and vehicle density has worsened the traffic situation with the current infrastructure.

Problem Statement

Develop a learning agent using artificial intelligence which can learn the traffic patterns and efficiently control the traffic signal to reduce traffic bottlenecks. The agent should be able to choose the traffic phase and optimize the traffic efficiency, given the state of the intersection.

Purpose/Motivation

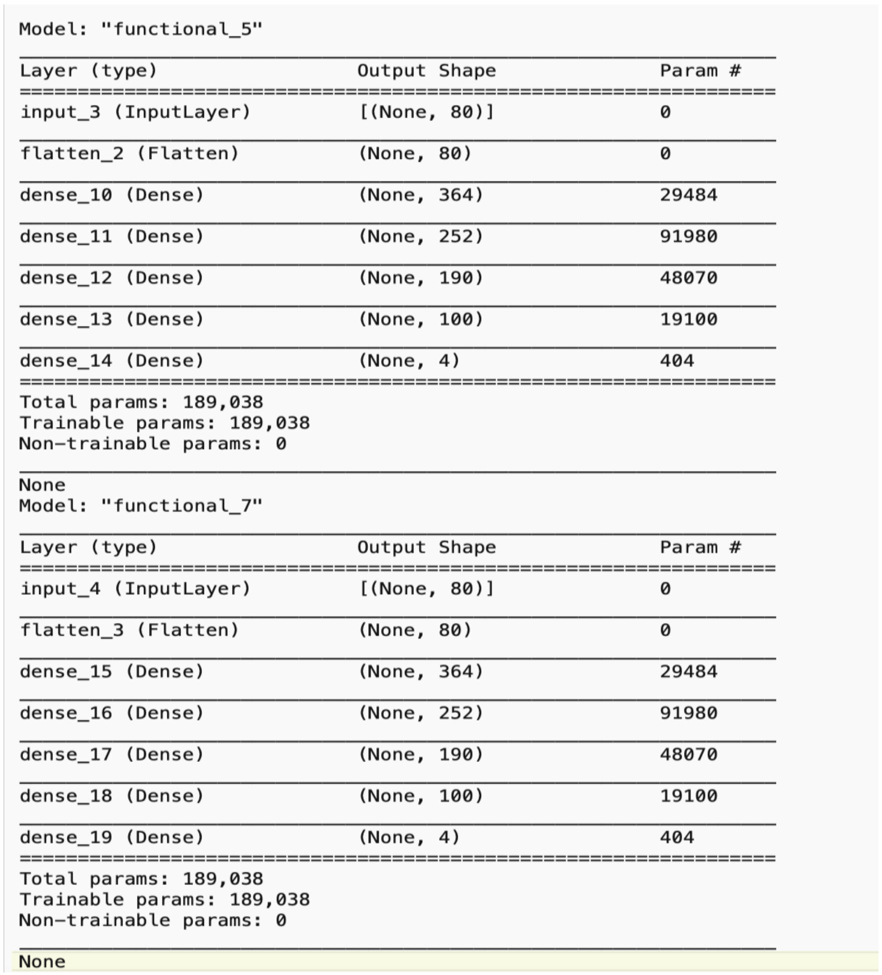
We can think of two solutions to this problem. First, expand transportation capacity and build high quality roads which may require a significant amount of resources and time. Second, use the existing infrastructure by improving existing traffic control systems. We choose the latter.

Differentiator/Contribution

We have utilized the traffic generation and simulation part from the currently available literature. The environment uses SUMO(Simulation of Urban Mobility) to generate traffic and create an environment required for the project. The environment, action, state and reward is kept same as the current literature.

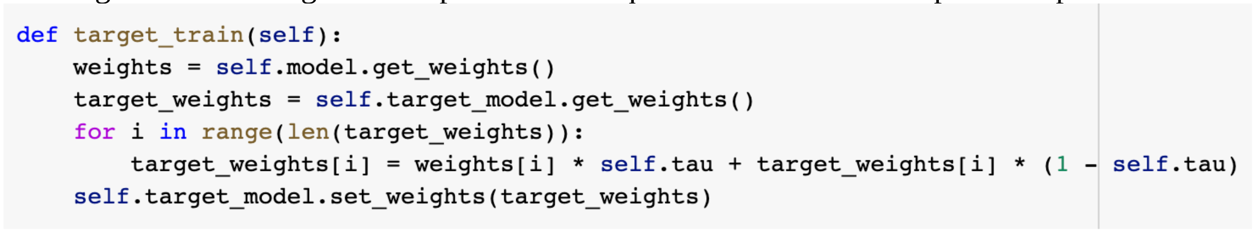
Below are the changes we have implemented in our project which is different from the current literature.

1. Implemented neural network architecture which is a simple multi-layer perceptron with 4 hidden layers using keras.



```
Model: "functional_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 80)]              0
_________________________________________________________________
flatten_2 (Flatten)          (None, 80)                0
_________________________________________________________________
dense_10 (Dense)             (None, 364)               29484
_________________________________________________________________
dense_11 (Dense)             (None, 252)               91980
_________________________________________________________________
dense_12 (Dense)             (None, 190)               48070
_________________________________________________________________
dense_13 (Dense)             (None, 100)               19100
_________________________________________________________________
dense_14 (Dense)             (None, 4)                 404
=================================================================
Total params: 189,038
Trainable params: 189,038
Non-trainable params: 0
_________________________________________________________________
None
Model: "functional_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 80)]              0
_________________________________________________________________
flatten_3 (Flatten)          (None, 80)                0
_________________________________________________________________
dense_15 (Dense)             (None, 364)               29484
_________________________________________________________________
dense_16 (Dense)             (None, 252)               91980
_________________________________________________________________
dense_17 (Dense)             (None, 190)               48070
_________________________________________________________________
dense_18 (Dense)             (None, 100)               19100
_________________________________________________________________
dense_19 (Dense)             (None, 4)                 404
=================================================================
Total params: 189,038
Trainable params: 189,038
Non-trainable params: 0
_________________________________________________________________
None
```

2. The current literature does not use the target network and the current and next actions both are predicted using the same network, we have added the target network in our implementation for q-learning.
3. The q-learning algorithm learnt in the class is implemented using a q-network and target network.
4. The target network weights are copied from the q-network after 5000 steps in an episode.



```
def target_train(self):
    weights = self.model.get_weights()
    target_weights = self.target_model.get_weights()
    for i in range(len(target_weights)):
        target_weights[i] = weights[i] * self.tau + target_weights[i] * (1 - self.tau)
    self.target_model.set_weights(target_weights)
```

5. The replay buffer implementation is similar to the current literature with slight changes in the min and max buffer size, as we have tried changing the limits for testing the model.
6. Different batch size I was tested with our model, some yielding worse results with higher batch size.
7. Different gamma and learning rate values were tested for our model.

Methodology

The agent(Traffic Light) interacts with the environment to get the state of the traffic and calculate the reward. The reward is represented by cumulative waiting time of all cars.

$$twt_t = \sum_{veh=1}^{n} wt_{(veh,t)}$$

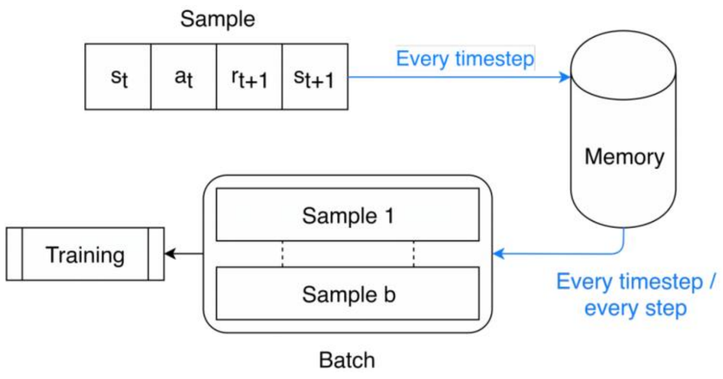
$$r_t = twt_{t-1} - twt_t$$

Where r_t represents the reward at timestep t. tw_t and twt_{t-1} represent the total waiting time of all the cars in the intersection captured respectively at timestep t and $t-1$.

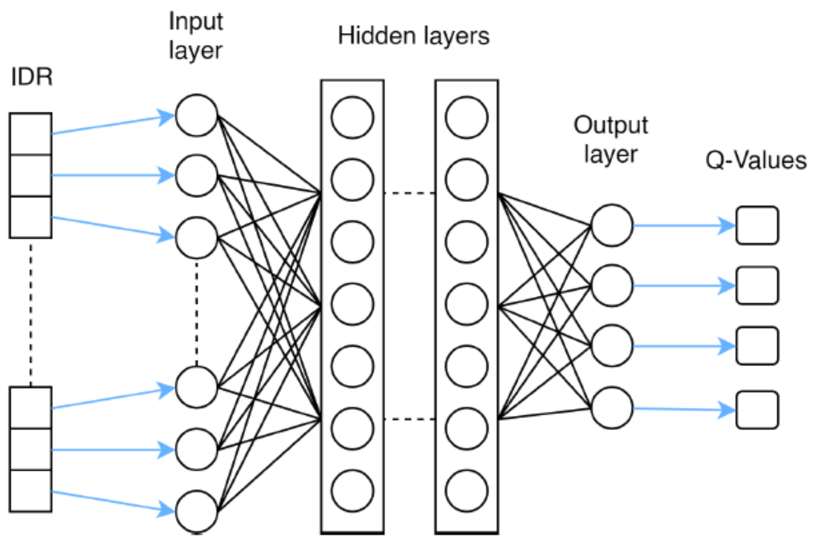
With Q-learning function, we calculate the following:

$$Q(s_t, a_t) = r_{t+1} + \gamma \cdot max_A Q'(s_{t+1}, a_{t+1})$$

The values (s_t, a_t, r_{t+1}, s_{t+1}) are passed to the replay buffer



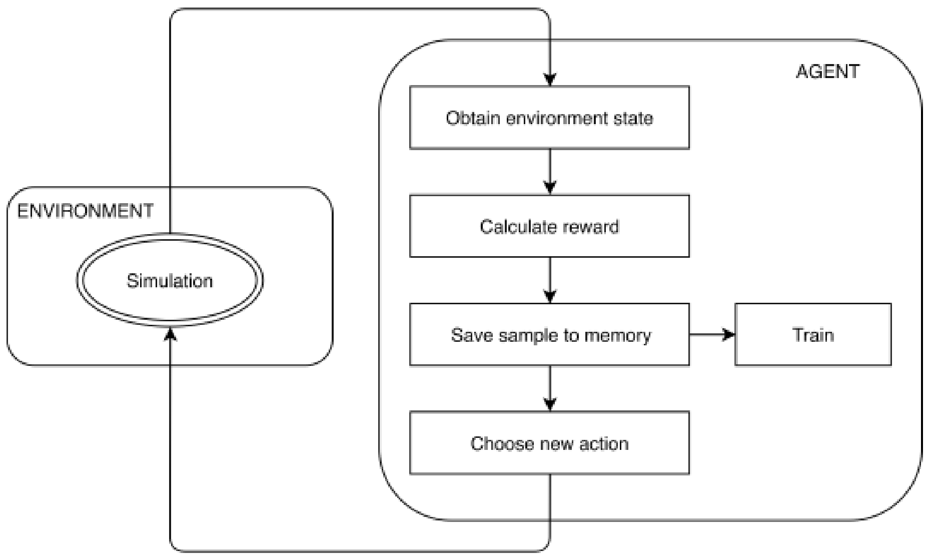
Random samples from the replay buffer in a mini batch of 100 are passed to neural network for training.



The loss is calculated as

$$\mathcal{L} = (Q_{s,a} - (r + \gamma \max_{a' \in A} Q_{s',a'}))^2$$

The weights of the neural network are updated using stochastic gradient descent (SGD) to minimize the loss.
The Target network keeps a copy of the base network and calculates Q (s_{t+1}, a_{t+1}) value in the Bellman equation. The parameters from source network are copied over to the target network every 2000 t steps.

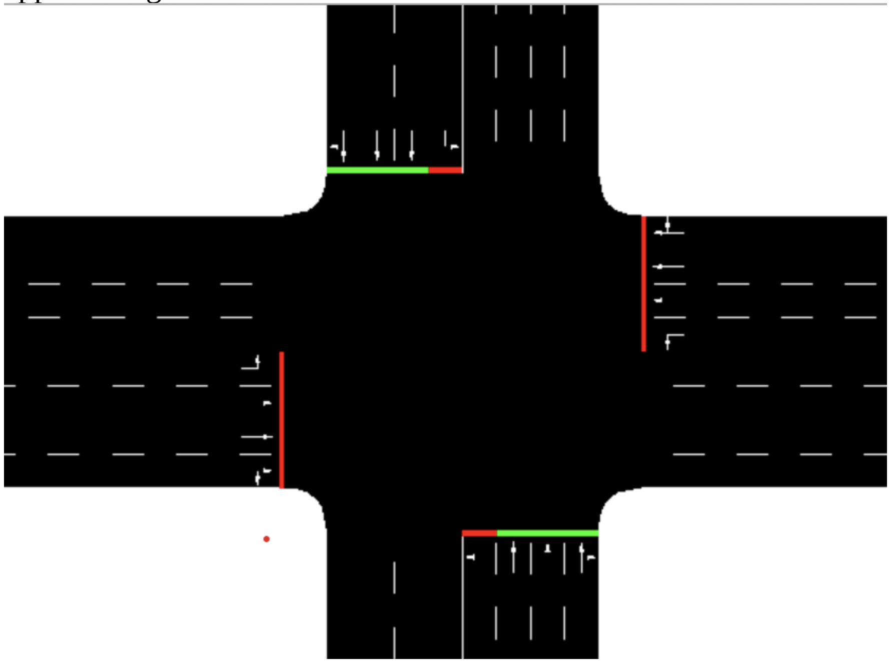


Implementation and Results

Traffic Control System implemented is an approach to manage the traffic signal at a 4-lane intersection. DQ learning is used for this approach with traffic simulation using SUMO (Simulation of Urban Mobility).
The traffic is simulated by the below env.

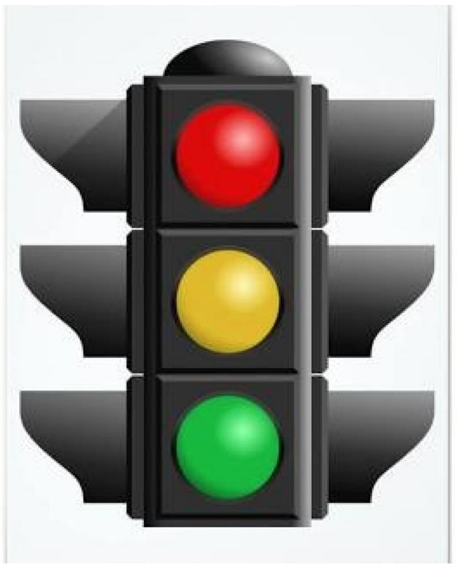
Environment

The environment in which the agent acts is a 4-way intersection. 4 lanes per arm approach the intersection from the compass directions and lead to 4 lanes per arm leaving the intersection. In the center of the intersection, a traffic light system, controlled by the agent, manages the approaching traffic.

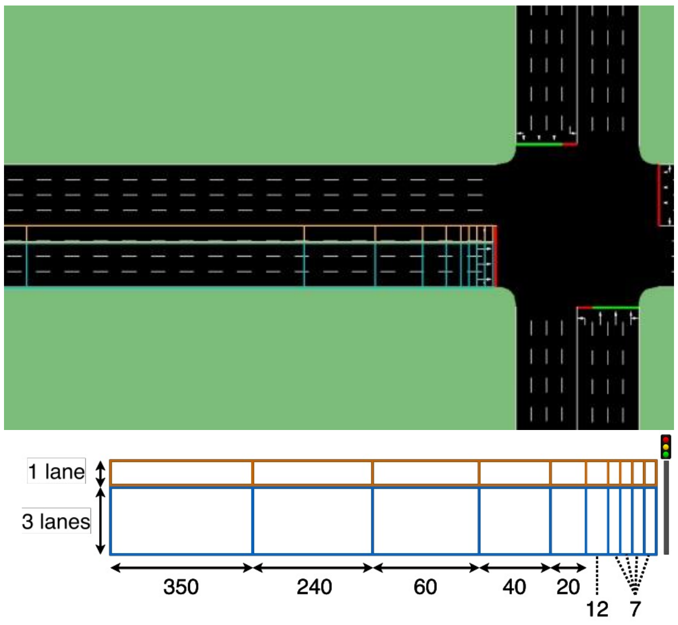


Agent

Agent is defined as a set of traffic lights that manages the incoming traffic into the intersection. Traffic Control System is composed of agents that interacts with the environment using a state s, an action a and a reward r.



State:
The lanes of the intersection are discretized in cells along the length of the lane. The state of the traffic is represented by cells.



Reward
Defined as a function of performance indicator of the intersection such as vehicle delay, queue length, waiting time or throughput.

Action
Given the state of s, the agent should choose a traffic light phase a, from a set of actions A, and maximize the rewards r.
A = {NSA; NSLA;EWA;EWLA}

A simple multilayer perceptron is used a neural network to predict the q-values. The same network is used as target network to predict the q- values with next states. The weights of the target is copied from the q-network once in after 5000 steps in an episode.

Experience replay
The randomized group of samples called batch is taken from a data structure, memory which stores every sample collected during the training phase. A training instance involves the gathering of a group of samples from the memory and the neural network training phase.

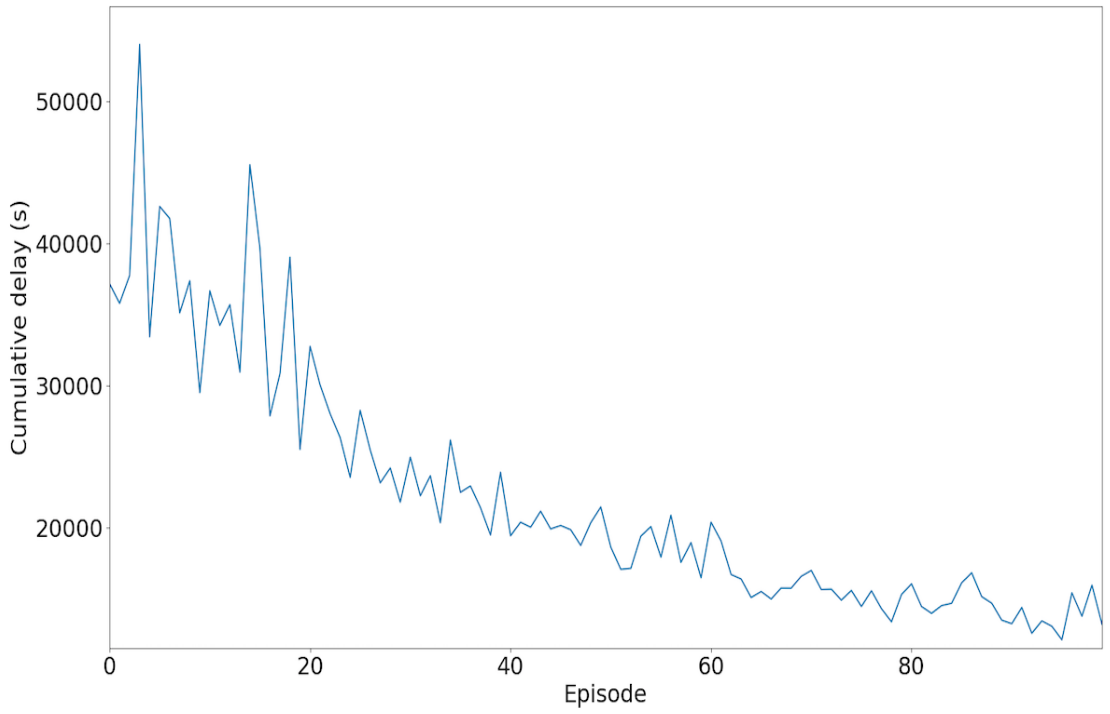
Training

The configuration values are initialized, the q-network and target network are created calling the implemented DQN class. The memory buffer size is initialized with a min of 600 and max buffer size of 5k. The training is done for 100 episodes, each episode is configured to run for 5400 steps and the neural network is trained for 500 epochs in each episode. The training time for this network on google colab is around 5 hours for 1 complete cycle. The aim of the model is to reduce the wait time of the vehicles in the intersection by taking proper action which reduces the queue length of the vehicles in the intersection.

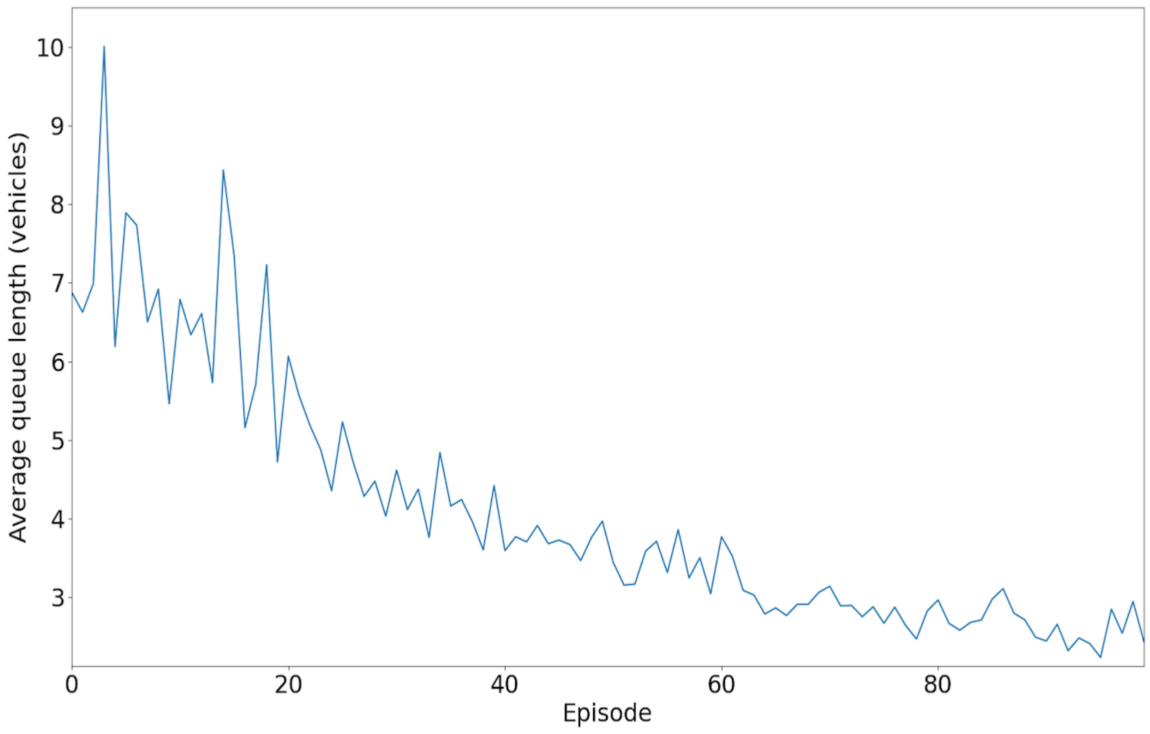
Results

With Gamma = 0.75, learning rate = 0.001

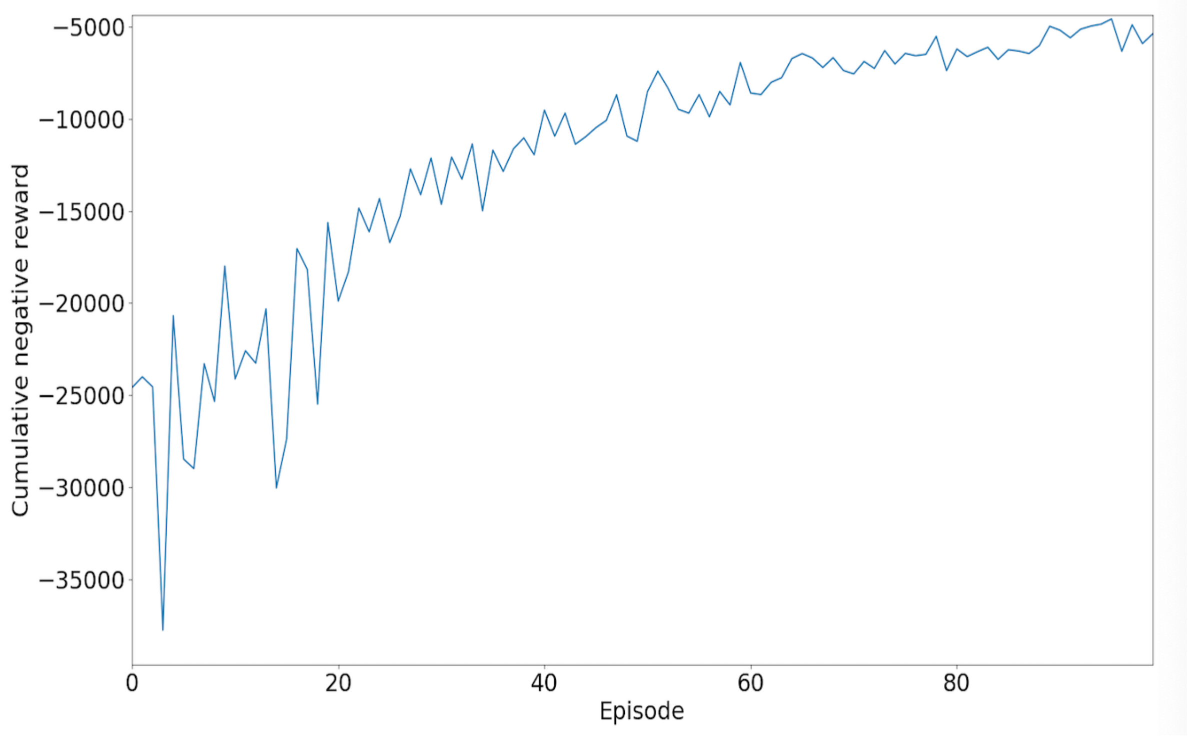
Cumulative delay



Average queue length in the intersection

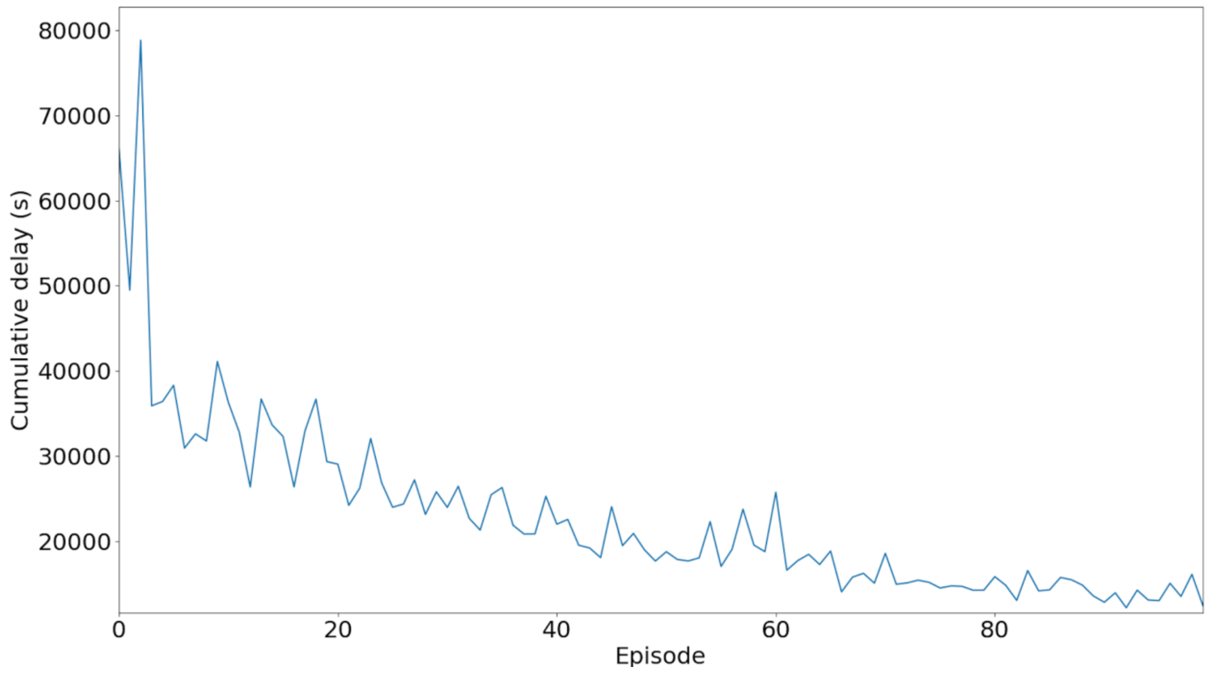


Rewards

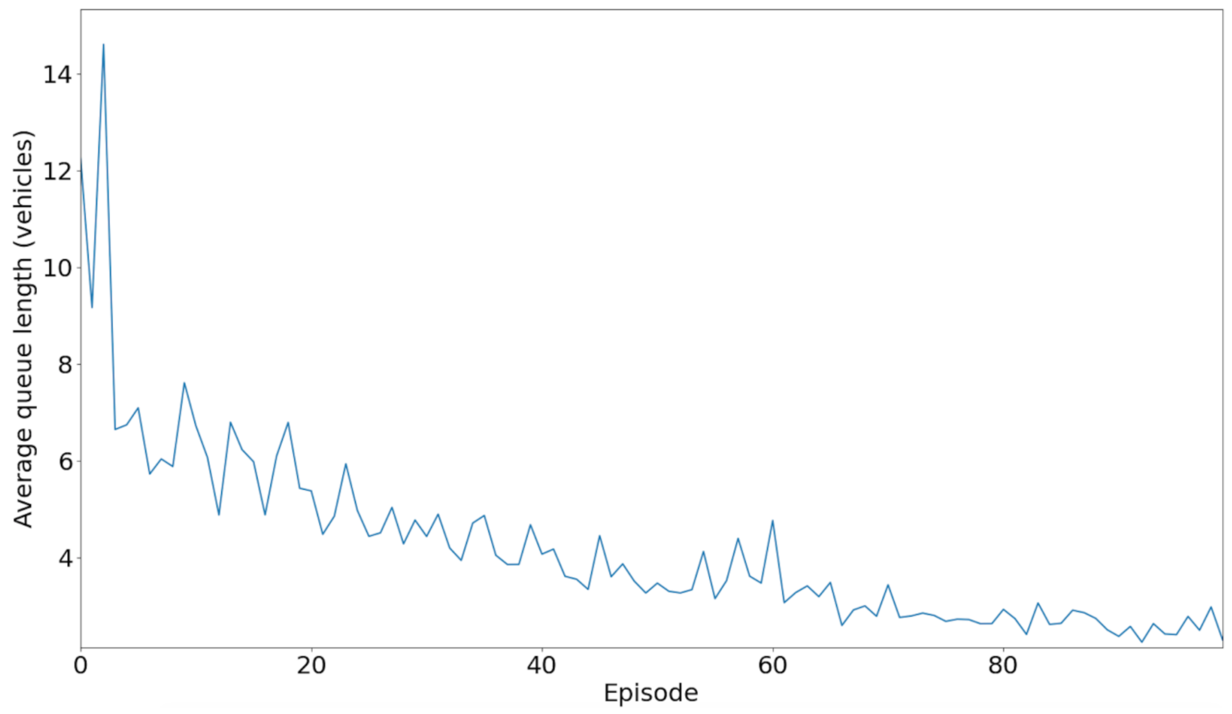


With Gamma = 0.95, learning rate = 0.0015

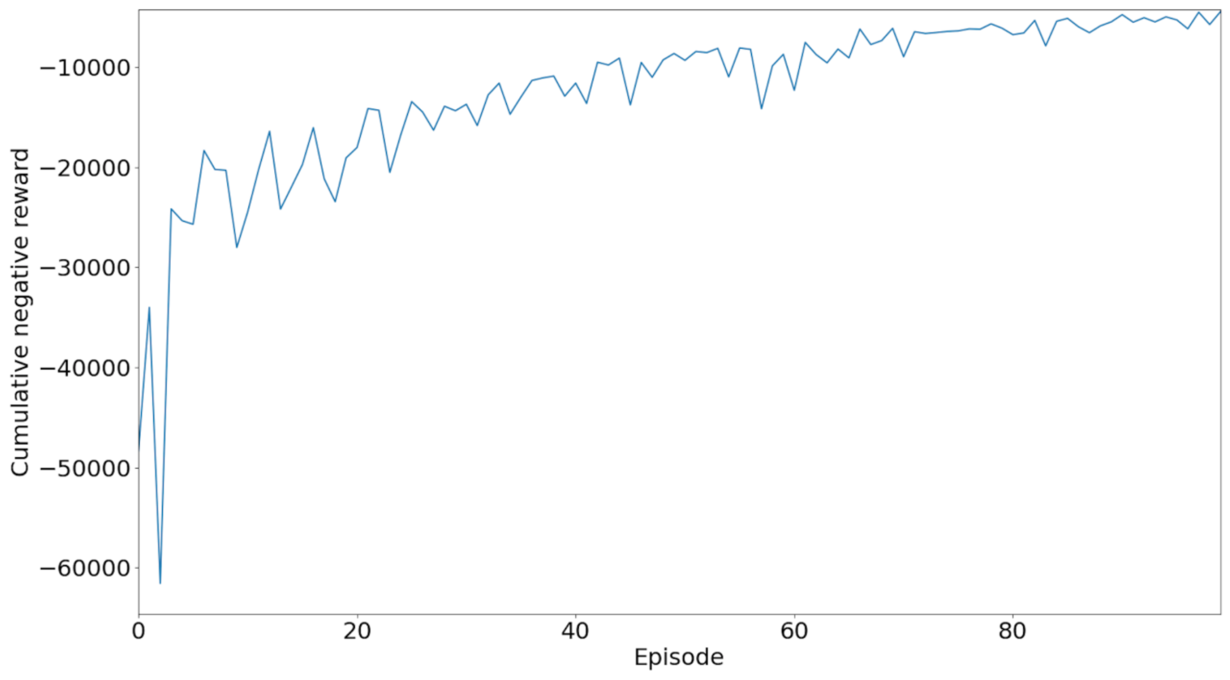
Cumulative delay



Average queue length in the intersection



Rewards



Contributions

Team Member	Contribution
Atul Shah	1. Understanding the simulation of environment and traffic generation. 2. Implementing Q-learning logic including the copy of target network weights. Deciding on the number of steps to copy the Q-network weights to target network, after several runs of training episode. 3. Testing the performance of the model with different values of gamma and learning rate.
Shreyus Puthiyapurail	1. Understanding the parameter values for the network and memory buffer. 2. Implementing the DQN neural network with required hidden layers and neurons. Deciding on the best parameters after several run of training epochs. 3. Testing the model performance with different values of batch size, memory buffer size and changing the number episodes and epochs for training.

Conclusion

The agent trained with Deep Q Learning algorithm clearly outperforms conventional traffic controller system with a static traffic light sequence and duration. However, as the traffic density increases, the agent can take wrong action because of delayed availability of reward, thus leading to longer wait times as compared to conventional traffic controller system.
Future work may include making state representations more comprehensive. In addition to representing the road traffic as discretized cells, the state can also include relative velocity of the traffic which can influence agent's training and corresponding sequence of traffic lights. Also, as the self driving car technology advances, the Deep Q-Learning based traffic controller system can offer a communication protocol between cars and traffic infrastructure and can improve overall efficiency.

Appendix

Link to the code
https://github.com/TrafficControllerRL/TrafficControllerRL

References

[1] AndreaVidali. (n.d.). AndreaVidali/Deep-QLearning-Agent-for-Traffic-Signal-Control. Retrieved December 13, 2020, from https://github.com/AndreaVidali/Deep-QLearning-Agent-for-Traffic-Signal-Control

[2] Kok-Lim Alvin Yau et al. “A survey on reinforcement learning models and algorithms for traffic signal control”. In: ACM Computing Surveys (CSUR) 50.3 (2017), p. 34 (cit. on p. 9).

[3] Reinforcement Learning for Traffic Signal Control. (n.d.). Retrieved December 13, 2020, from https://traffic-signal-control.github.io/