# Traffic Signal Optimization for Smart Cities using Deep Reinforcement Learning

Aryan Patil, Harsh Ramani, Saiteja Kalam, Chandra Mourya

*Stony Brook University*

**Abstract**

As urban populations grow, traffic congestion becomes an increasingly critical issue, affecting both economic productivity and quality of life. Traditional traffic management systems often fall short in adapting to dynamic traffic conditions. This project explores the application of Deep Q-Learning (DQL) to optimize traffic signal control in smart cities. By treating each traffic light as an autonomous agent, DQL enables real-time decision-making based on current traffic data.

Our approach leverages the power of deep neural networks to approximate optimal policies, aiming to reduce congestion, minimize travel times, and improve road safety. The integration of emergency vehicle prioritization further enhances the system's responsiveness during critical situations. Using the SUMO traffic simulator and real-world traffic datasets, we demonstrate significant improvements in traffic flow efficiency.

This study highlights the potential of DQL to transform urban mobility, offering a scalable and adaptive solution for modern cities seeking to enhance their infrastructure through intelligent systems.

## Contents

## 1 Introduction

Transportation is a fundamental pillar of human civilization, serving as the backbone of economic activities and connecting people across various geographies. Traffic signal systems play a critical role in regulating vehicular movement, ensuring safety, and enhancing the efficiency of transportation networks. However, with urbanization, the challenges associated with traffic congestion have escalated dramatically.

Studies reveal that the average human spends significant hours annually in traffic jams, with urban dwellers experiencing up to 54 hours of delay per year due to congestion[1]. These delays are more than just inconveniences; they have far-reaching implications. Economically, traffic congestion costs billions in lost productivity and fuel wastage. In terms of environmental benefits, idling vehicles at intersections increases carbon emissions, contributing to air pollution and climate change. Socially, prolonged commute times reduce quality of life, leaving less time for family and personal development. Intersections, the focal points of traffic networks, are particularly vulnerable to congestion. The lack of efficient and adaptive traffic signal systems exacerbates this problem, especially during peak hours and emergencies. Conventional traffic management methods, relying on static rules or basic optimization, struggle to cope with dynamic traffic conditions and complex road networks. The need for innovative solutions in traffic management is therefore urgent and clear. Deep Reinforcement Learning (DRL) has emerged as a promising technology to address these challenges. By enabling traffic signals to act as intelligent agents, DRL allows dynamic and adaptive decision making. Unlike traditional methods, DRL optimizes signal timings to reduce waiting times, minimize fuel consumption, and prioritize safety.

## 1.1 Our Approach

In this project, we leverage Deep Q-Networks (DQN) to optimize traffic signal control in smart cities, aiming to address key challenges such as traffic congestion, vehicle delays, and the need for emergency vehicle prioritization. The DQN framework, a powerful variant of deep reinforcement learning, combines the decision-making prowess of Q-Learning with the feature extraction capabilities of deep neural networks (DNNs). This synergy allows DQNs to efficiently handle the complexities of large and dynamic state spaces inherent in urban traffic systems. Unlike traditional traffic management methods that rely on static schedules or manually tuned adaptive systems, DQNs enable real-time learning and adaptability by continuously interacting with the traffic environment and optimizing signal timings based on observed patterns and outcomes. By modeling traffic intersections as reinforcement learning environments, the DQN agent learns an optimal policy to minimize cumulative travel time, reduce vehicle idling, and ensure the smooth flow of traffic. The integration of reward structures that prioritize emergency vehicle passage further enhances the system's responsiveness to critical scenarios, contributing to public safety. The use of DQNs also allows for scalable deployment across multiple intersections, enabling coordinated traffic control in complex urban networks. This approach not only improves traffic efficiency but also supports smart city objectives by reducing fuel consumption and emissions, aligning with sustainability goals. Through extensive simulations and evaluations, this project demonstrates the potential of DQNs to transform urban traffic management systems.

## 1.2 Related Works

- **Early Applications of Reinforcement Learning:**
  Early work applied basic RL methods like Q-Learning to traffic signal optimization. These approaches were limited to small-scale, simplified traffic networks due to scalability issues. For example, Wiering (2000)[2] explored RL for traffic light control, but faced challenges in handling large state spaces.

- **Introduction of Deep Q-Networks:**
  The integration of deep neural networks with Q-Learning, as introduced by Mnih et al. (2015)[3], allowed RL to scale to more complex problems. DQNs approximate Q-values for state-action pairs using neural networks. in our project enabling traffic systems to handle diverse and high-dimensional traffic states (e.g., vehicle counts, queue lengths, average speeds).

- **Traffic Signal Control in Single Intersection with DQN:**
  Recent studies have applied DQNs to

optimize traffic signals in single intersection scenarios. For example, Liang et al. (2019) demonstrated the use of DQNs to dynamically adjust traffic light timings[4], achieving significant reductions in vehicle waiting times and congestion.

- **Simulation Platforms:** Studies frequently utilize traffic simulators like SUMO (Simulation of Urban Mobility) to evaluate the performance of DQN-based traffic signal controllers[5]. Simulators provide realistic environments for training and testing RL algorithms.

## 2 Methodology

**In this section, we will setup the simulations and technically define the problem in RL.**

### 2.1 Environment Setup

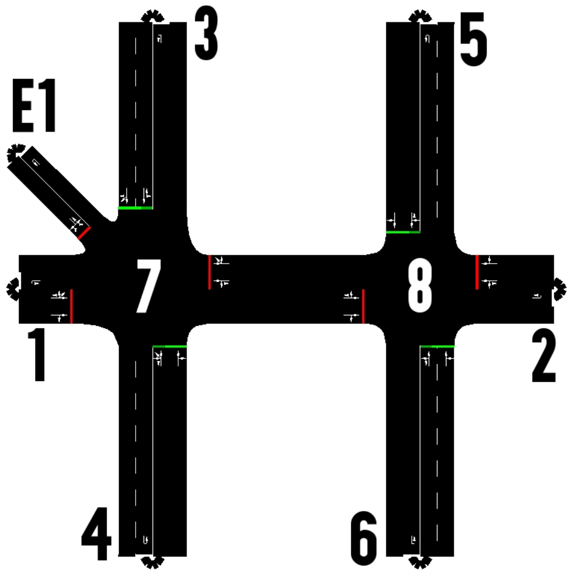

Figure 1: Environment setup for simulations.

The environment in Figure: 2 is a simulated road network designed using the SUMO (Simulation of Urban Mobility) traffic simulation framework. This environment consists of multiple intersections, including traffic-light-controlled junctions and priority junctions, to model complex urban traffic scenarios. Here's a breakdown of the features and functionality:

- **Traffic Lights:** Two intersections (nodes 7 and 8) are controlled by RL agent, ensuring smoother vehicle flow and collision prevention.
- **Regular Nodes:** Other nodes are configured as priority intersections where vehicles follow predefined right-of-way rules. The traffic either originate or finish at these nodes.
- **Emergency Access Road:** Specially designed lanes (e1-to-7, 7-to-e1) provide dedicated routes for emergency vehicles, allowing rapid traversal through the network.
- **Emergency Vehicles:** Faster vehicles with higher acceleration and speed, given priority on predefined routes.
- **Standard Vehicles:** Regular cars with typical acceleration, deceleration, and speed characteristics.
- **Traffic Flows:** The network includes both regular traffic flows (e.g., flow0, flow1) and specialized emergency vehicle flows (e.g., emergency_flow1). These flows simulate varying traffic intensities and scenarios, such as peak-hour congestion and emergency response.

**Code Functionality to Generate the Network:**

- **Node and Edge Files:**
  - The `my_nodes.nod.xml` file de-

fines the network nodes, including their types (e.g., `traffic_light` or `priority`).

- The `my_edges.edg.xml` file specifies the edges connecting the nodes, along with properties such as speed, number of lanes, and priority.

- **Route and Flow Files:**
  - The `my_routes.rou.xml` file defines the routes and vehicle flows within the network, ensuring a mix of standard and emergency traffic.
- **Network File Generation:**
  - The `netconvert` tool converts the node and edge definitions into a complete network file (`my_network.net.xml`).
- **Simulation Configuration:**
  - The `my_simulation.sumocfg` file configures the simulation, specifying the network and route files, simulation time, and logging options.
- **Execution:**
  - The script automates the simulation using the SUMO command-line interface (`sumo -c my_simulation.sumocfg`), enabling easy experimentation and testing.

## 2.2 Problem Definition

**Markov property of intelligent traffic signal control:** An effective Intelligent Traffic Signal Control system adjusts its signal dynamically based on real-time traffic conditions at intersections. This behavior aligns with Markov's property, as the decision to change the traffic signal phase depends solely on the current state of traffic, without considering past traffic conditions. This characteristic makes reinforcement learning particularly suitable for managing traffic signals.
Let's define the problem in RL terminology:

**State:** The state represents the current status of the traffic network, capturing important traffic conditions and system variables.

- **State Components:**
  - `traffic_light_phases`: The current phase of the traffic lights at intersections 7 and 8.
  - `vehicle_count`: Total number of vehicles in the network at the current timestep.
  - `average_speed`: The average speed of vehicles in the network.
  - `queue_lengths`: A dictionary containing the number of vehicles halted (queue length) on specific road edges.
- The state is discretized into categorical values for efficient processing, where:
  - Vehicle counts, average speed, and queue lengths are mapped into predefined ranges.
- **State Representation:**
  - The state is represented as a tuple of discrete values:

```
('traffic_light_phases',
'vehicle_count',
'average_speed',
'queue_lengths')
```

**Action:** The action represents the decisions made by the RL agent, which in this case, involves changing the traffic light phases at intersections.

- **Action Components:**
  - The phases of traffic lights at intersections 7 and 8 are controlled.

  - For each traffic light, the action is represented as an integer (`0, 1, 2, 3`) corresponding to the phase index.

- **Action Representation:**
  - An action is a dictionary mapping intersection IDs to phase indices:

    ```
    {'7': phase_index,
    '8': phase_index}
    ```

  - The RL agent selects these phase indices to maximize traffic flow and minimize congestion.

**Reward:**The reward function quantifies the effectiveness of an action in achieving the desired traffic optimization objectives.

- **Reward Calculation:**
  - The reward is proportional to the `average_speed` of vehicles, incentivizing smoother traffic flow.
  - A penalty is applied for higher `queue_lengths`, discouraging congestion:

    ```
    reward = avg_speed -
    0.1 * total_queue
    ```

- **Objective:**
  - Maximize average vehicle speed.
  - Minimize the total number of halted vehicles (queues).

**Interaction with the Environment:** The RL agent interacts with the SUMO environment as follows:

1. **Observe the Current State:**
   - The agent retrieves the current traffic conditions (e.g., `traffic_light_phases`, `vehicle_count`, `average_speed`, and `queue_lengths`) using the `traci` API.
2. **Take an Action:**
   - The agent selects an action (traffic light phase for intersections `7` and `8`) and applies it using:

     ```
     traci.trafficlight.
      setPhase('7', action['7'])
     traci.trafficlight.
      setPhase('8', action['8'])
     ```

3. **Environment Response:**
   - The environment updates traffic conditions based on the action taken (e.g., vehicles move, queues form).
4. **Receive a Reward:**
   - The reward is calculated based on the updated state using:

     ```
     calculate_reward(next_state)
     ```

5. **Transition to the Next State:**
   - The agent transitions to the next state by retrieving the updated traffic conditions.
6. **Record Interaction:**
   - The interaction (`current_state`, `action`, `reward`, and `next_state`) is stored in the history for training or evaluation:

     ```
     history.append((current_state,
     action, reward, next_state))
     ```

# 3 Algorithms

**In this section, we will discuss how we implemented Q-Learning and Deep Q-Network.**

## 3.1 Q-Learning for Traffic Light Control

**Algorithm 1** Q-Learning for Traffic Light Control

```
Require: Number of episodes $N$, Steps per episode $T$, Learning rate $\alpha$, Discount factor $\gamma$,
    Exploration rate $\epsilon$
 1: Initialize $Q(s,a) \leftarrow 0$ for all states $s$ and actions $a$
 2: for each episode $e = 1$ to $N$ do
 3:     Start simulation and observe initial state $s$
 4:     Discretize state: $s \leftarrow (\text{Traffic Phases}, \text{Vehicle Count}, \text{Average Speed}, \text{Queue Lengths})$
 5:     Initialize $R_{\text{total}} \leftarrow 0$
 6:     for $t = 1$ to $T$ do
 7:         With probability $\epsilon$, select random action $a$
 8:         Otherwise, select $a \leftarrow \arg\max_{a'} Q(s,a')$
 9:         Execute action $a$, observe reward $R$ and next state $s'$
10:         Calculate reward:
                $R = \text{Average Speed} - 0.01 \times \text{Total Queue Length} - 0.2 \times \text{Halting Vehicles}$
11:         Discretize next state:
                $s' \leftarrow (\text{Traffic Phases}, \text{Vehicle Count}, \text{Average Speed}, \text{Queue Lengths})$
12:         Update Q-table:
                $Q(s,a) \leftarrow Q(s,a) + \alpha \left(R + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$
13:         $s \leftarrow s'$, $R_{\text{total}} \leftarrow R_{\text{total}} + R$
14:     end for
15:     Log metrics for episode: Average Waiting Time, Average Speed, Total Queue Length,
    $R_{\text{total}}$
16: end for
17: Output: Optimized Q-table $Q(s,a)$
```

## 3.2 Q-Learning Explanation

This Q-Learning algorithm aims to optimize traffic light control by adjusting the signal timings to improve overall traffic flow. It uses a reinforcement learning approach to learn a policy that maximizes a cumulative reward associated with desirable traffic conditions—such as higher average speeds and shorter queue lengths—while minimizing the number of halting vehicles.

**State Representation:** The environment is described by key traffic parameters:

- *Traffic Phases:* Which traffic signal phase is currently active.
- *Vehicle Count:* The number of vehicles approaching or waiting at the intersection.
- *Average Speed:* The average speed of all vehicles passing through.
- *Queue Lengths:* The length of vehicle queues in each direction.

**Action Selection:** At each time step, the algorithm selects an action corresponding to changing or maintaining the current traffic signal configuration. It uses an $\epsilon$-greedy policy:

- With probability $\epsilon$, choose an action at random (exploration).
- With probability $1-\epsilon$, choose the action that maximizes the current Q-value (exploitation).

**Reward Signal:** After each action, the algorithm receives a reward designed to reflect traffic quality:

$$R = \text{Average Speed}$$

$$-0.01 \times \text{Total Queue Length}$$

$$-0.2 \times \text{Halting Vehicles}$$

This encourages higher speeds, shorter queues, and fewer stops.

**Q-Table Update:** The Q-values, $Q(s,a)$, represent the quality of taking action $a$ in state $s$. After observing the next state $s'$ and reward $R$, the Q-value is updated as:

$$Q(s,a) \leftarrow Q(s,a)+$$

$$\alpha\big(R + \gamma \max_{a'} Q(s',a') - Q(s,a)\big)$$

Here, $\alpha$ is the learning rate and $\gamma$ is the discount factor.

**Learning and Convergence:** Over multiple episodes, the algorithm refines its Q-values and reduces exploration. Eventually, it converges to a policy that optimally controls the traffic signals, reducing congestion and improving overall travel efficiency at the intersection.

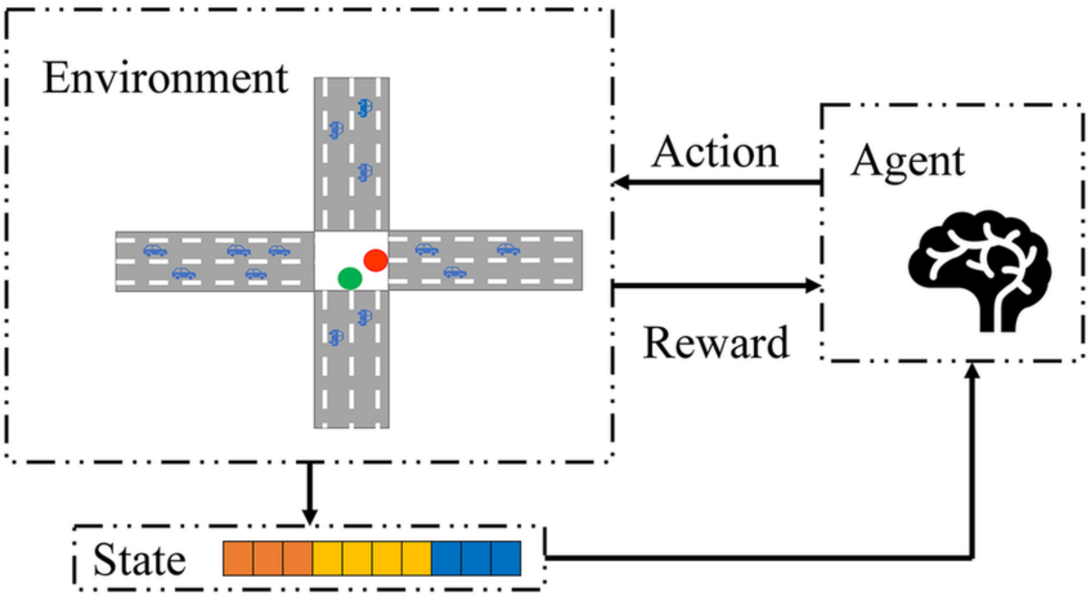

Figure 2: How RL agents interact with environment.

## 3.3 Building Deep Q-Network for Traffic Signal Control

**Algorithm 2** Deep Q-Learning Algorithm

```
Require: Maximum replay buffer size $N$, Number of episodes NUM_EPISODES, Batch size BATCH_SIZE, Target update frequency TARGET_UPDATE_FREQ, Epsilon decay EPSILON_DECAY, Minimum epsilon EPSILON_MIN
 1: Initialize replay buffer $\mathcal{B}$ with maximum size $N$
 2: Initialize policy network $\pi_\theta$ with parameters $\theta$
 3: Initialize target network $\pi_{\theta'}$ and set $\theta' \leftarrow \theta$
 4: Set exploration rate $\epsilon \leftarrow 1.0$
 5: for each episode $e \in \{1, \dots, \text{NUM\_EPISODES}\}$ do
 6:     Start the simulation environment
 7:     Observe initial state $s_0$ and discretize it to obtain $s$
 8:     Set total reward $R \leftarrow 0$
 9:     for each step $t$ in the episode do
10:         With probability $\epsilon$, select random action $a_t$
11:         Otherwise, select $a_t \leftarrow \arg\max_a Q(s, a; \theta)$
12:         Execute action $a_t$ in the environment
13:         Observe reward $r_t$ and next state $s_{t+1}$, then discretize $s_{t+1}$
14:         Store transition $(s, a_t, r_t, s_{t+1})$ in $\mathcal{B}$
15:         Update state $s \leftarrow s_{t+1}$
16:         $R \leftarrow R + r_t$
17:         if $\mathcal{B}$ has at least BATCH_SIZE samples then
18:             Sample minibatch of transitions $(s_i, a_i, r_i, s'_i)$ from $\mathcal{B}$
19:             Compute target $y_i \leftarrow r_i + \gamma \max_{a'} Q(s'_i, a'; \theta')$
20:             Compute loss $\mathcal{L} \leftarrow \frac{1}{\text{BATCH\_SIZE}} \sum_i \big(y_i - Q(s_i, a_i; \theta)\big)^2$
21:             Perform gradient descent step on $\mathcal{L}$ with respect to $\theta$
22:         end if
23:         if $t \mod \text{TARGET\_UPDATE\_FREQ} = 0$ then
24:             Update target network $\theta' \leftarrow \theta$
25:         end if
26:     end for
27:     Decay exploration rate $\epsilon \leftarrow \max(\epsilon \cdot \text{EPSILON\_DECAY}, \text{EPSILON\_MIN})$
28:     Log total reward $R$
29:     if convergence criteria are met then
30:         break
31:     end if
32: end for
33: return trained policy network $\pi_\theta$
```

## 3.4 Explanation of Deep Q-Learning Algorithm

Deep Q-Learning extends traditional Q-Learning by using a neural network (the "policy network") to approximate the Q-function. Instead of storing and updating a large Q-table, the agent learns parameters $\theta$ of a network that maps states and actions to Q-values.

**Replay Buffer:** A replay buffer $\mathcal{B}$ is used to store transitions $(s, a, r, s')$ encountered during training. By sampling random minibatches from this buffer, the learning process decorrelates observations and stabilizes training.

**Target Network:** A separate target network with parameters $\theta'$ is maintained to generate stable target values $y_i$. This network is periodically updated with the parameters of the primary policy network, reducing oscillations and divergence during training.

**Exploration and Exploitation:** An $\epsilon$-greedy policy is employed to balance exploration and exploitation:

- With probability $\epsilon$, choose a random action.
- With probability $1 - \epsilon$, choose the action that maximizes the Q-value given by $Q(s, a; \theta)$.

Over time, $\epsilon$ decays to a minimum value, ensuring that the agent explores more at the beginning and exploits learned knowledge later.

**Network Updates:** At each step, a batch of transitions is sampled from the replay buffer. The target for each transition is computed as:

$$y_i = r_i + \gamma \max_{a'} Q(s'_i, a'; \theta')$$

The loss function is the mean squared error between $y_i$ and $Q(s_i, a_i; \theta)$:

$$\mathcal{L} = \frac{1}{\text{BATCH\_SIZE}} \sum_i (y_i - Q(s_i, a_i; \theta))^2$$

Gradient descent is performed on this loss to update $\theta$, improving the policy network.

**Convergence:** As training progresses, the Q-network's estimates become more accurate, and $Q(s, a; \theta)$ approaches the optimal Q-values. When convergence criteria are met (e.g., stable average rewards), training stops, and the final policy network $\pi_\theta$ can be used for decision-making in the environment.

## 4 Results

### 4.1 Q Learning

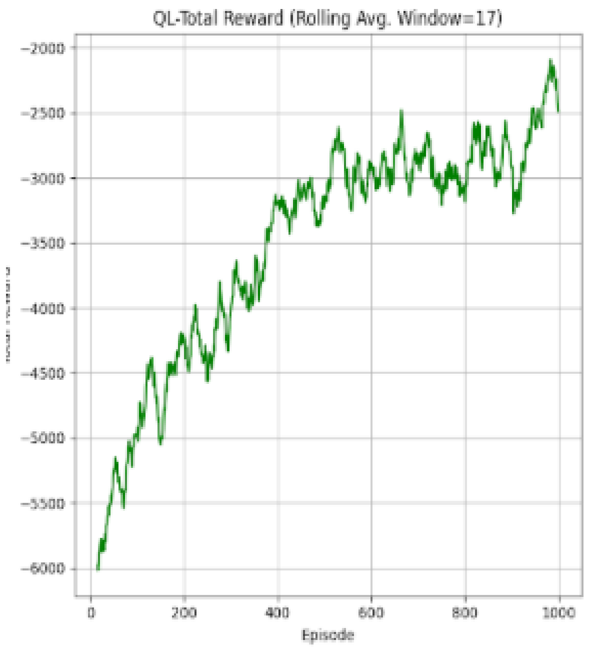

Figure 3: Over reduced space

Q-Learning demonstrated its potential to manage traffic scenarios by improving rewards and reducing waiting times compared to the baseline simulation. However, its limitations were pronounced when handling large state spaces. The algorithm faced challenges such as slower learning rates and inefficiency in exploring high-dimensional environments. These issues hindered its ability to identify optimal actions effectively, resulting in delayed convergence to satisfactory solutions.

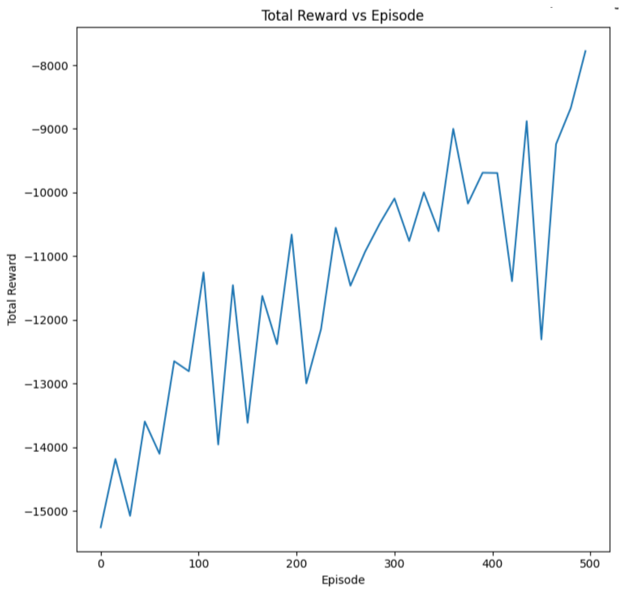

Figure 4: Over large state space

When the state space was reduced, Q-Learning showed moderate improvement in performance. This reduction simplified the learning process, allowing the algorithm to identify better policies for certain scenarios. However, even with a reduced state space, Q-Learning struggled to adapt efficiently to dynamic and complex traffic conditions, particularly when responding to real-time changes. Additionally, the concentration of many Q-values around zero created further inefficiencies, as it caused difficulties in distinguishing between state-action pairs and selecting the optimal action consistently.

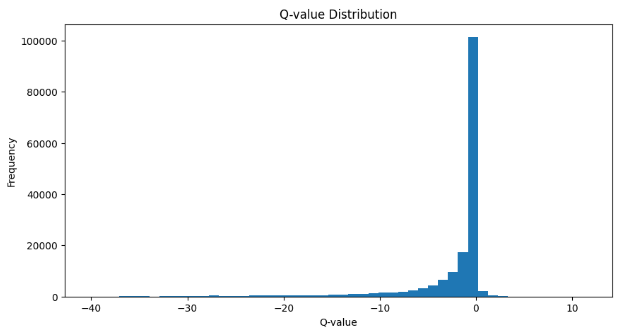

Figure 5: Issue in Q-Learning.

A significant issue in Q-Learning was the concentration of many Q-values around zero. This lack of differentiation between state-action pairs caused difficulties in prioritizing optimal actions, leading to suboptimal decision-making in certain scenarios. These limitations highlight the need for more advanced techniques, such as Deep Q-Learning, to overcome these challenges and ensure robust traffic signal optimization in large-scale environments.

## 4.2 Deep Q-Learning

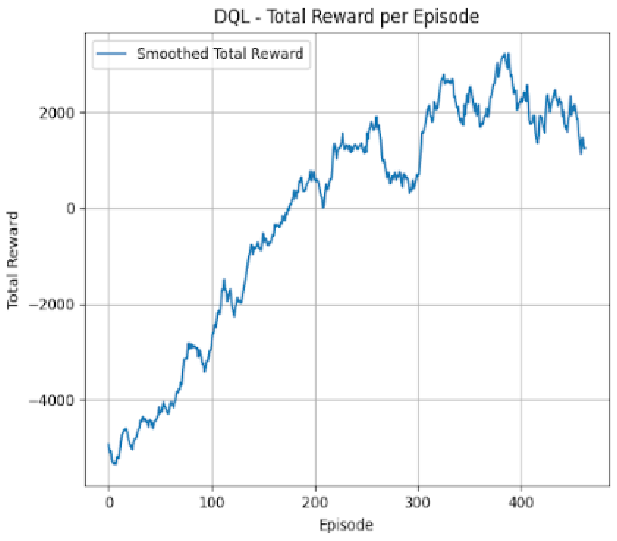

Figure 6: DQL - Total Reward per Episode

DQL addressed the limitations of Q-Learning by utilizing neural networks for function approximation, which significantly enhanced reward optimization. By generalizing over high-dimensional state-action spaces, DQL demonstrated better performance in handling complex scenarios. It displayed faster convergence and greater stability compared to Q-Learning, making it more reliable for learning efficient policies in dynamic environments.

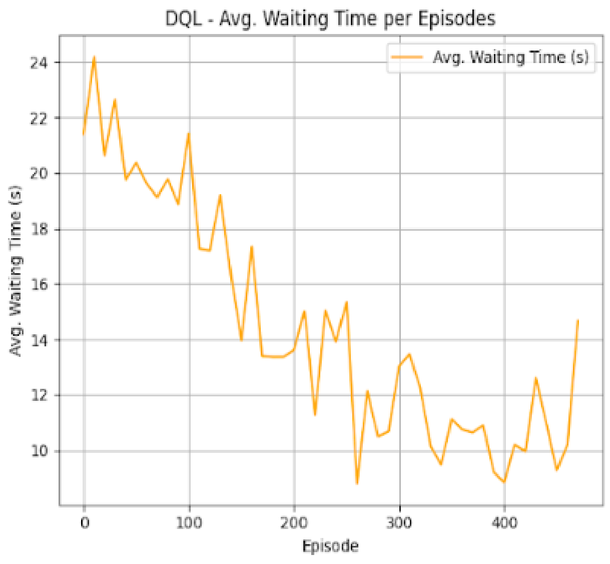

Figure 7: DQL - avg. Waiting Time per episode

In addition to reward optimization, DQL excelled in reducing waiting times and improving overall traffic flow. By leveraging its ability to process and adapt to real-time data, it minimized congestion effectively. This capability made DQL particularly suitable for real-world applications in traffic control, where responsiveness and efficiency are critical for addressing dynamic and fluctuating traffic conditions.

| Metric | Simulation | QL | Deep QL |
|---|---|---|---|
| **Reward** | -15255.36 | -2327.62 | 2774.90 |
| **Vehicles Inserted** | 1086 / 1491 | 1106 / 1491 | 1491 / 1491 |
| **Vehicles Waiting** | 405 | 5 | 0 |
| **Teleport** | 11 | 1 | 0 |
| **Avg Speed (m/s)** | 4.2 | 4.6 | 6.5 |
| **Avg Waiting Time (s)** | 3493.33 | 79.22 | 15.66 |

Table 1: Comparison of metrics across all algorithms

The table demonstrates significant differences in traffic signal optimization between the three approaches: simulation, Q-learning, and deep Q-learning (DQL). The Simulation approach, which lacks reinforcement learning, suffers from inefficiencies with a high negative reward (-15255.36), long vehicle waiting times (3493.33 seconds), and considerable congestion (405 vehicles waiting). Q-Learning improves system performance with a moderate reduction in waiting times (79.22 seconds) and higher vehicle processing rates. However, it still leaves room for optimization.

DQL outperforms both methods, achieving a positive reward (2774.90), completely eliminating vehicle waiting and teleportation events, and maximizing traffic throughput (1491 vehicles processed). Additionally, DQL significantly increases average vehicle speed (6.5 m/s) and minimizes average waiting times to just 15.66 seconds, showcasing its superior capability to handle large-scale traffic environments efficiently.

## 5 Conclusions

The project highlights the effectiveness of reinforcement learning techniques in optimizing traffic signal systems. Through a comparison of Q-Learning and Deep Q-Learning (DQL), it is evident that DQL significantly outperforms traditional Q-Learning, addressing key limitations such as scalability and inefficiency in large state spaces. While Q-Learning struggles with convergence and performance in complex environments, DQL leverages neural networks for function approximation, enabling faster learning and more accurate decision-making.

DQL's ability to minimize average waiting times and achieve higher rewards demonstrates its potential for real-world traffic management applications. Furthermore, incorporating considerations for emergency vehicle prioritization would enhance its utility and societal impact. This project underscores the transformative role of advanced reinforcement learning methods in creating smart, adaptive, and efficient traffic systems to address modern urban challenges.

## References

[1] CNN Staff. Traffic gridlock costs americans billions of dollars and countless hours, study shows, 2019. Accessed: 2024-12-10.

[2] Marco Wiering. Multi-agent reinforcement learning for traffic light control. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 1151–1158. Morgan Kaufmann, 2000.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[4] Xiaoyuan Liang, Xiaojing Du, Guoliang Wang, and Zhu Han. Deep reinforcement learning for traffic light control in vehicular networks. *IEEE Transactions on Vehicular Technology*, 68(2):1243–1253, 2019.

[5] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent developments and applications of sumo – simulation of urban mobility. *International Journal on Advances in Systems and Measurements*, 5(3&4):128–138, 2012.