Report

Project: Reinforcement learning for ramp metering on highways.

AMIRA ramzi (3cs2)

DRAA Abdel-Illah (3cs1)

1) Q-learning and DQN algorithms :

Q-learning:

Q learning is a model-free, off-policy reinforcement learning (RL) algorithm designed to learn the optimal action-value function, which is used to determine the best possible actions in a given state of an environment. It does so by estimating the expected future rewards (Q-values) for each state-action pair and iteratively refining these estimates based on experiences from interactions with the environment.

How Q-learning Works:

In Q-learning, the agent starts with a Q-table where the initial Q-values for all state-action pairs are set to zero or some arbitrary value. Over time, the agent updates this table by observing the reward for taking specific actions in certain states and adjusting its estimates according to the Q-learning update rule.

The Q-value update rule is as follows:

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

Where:

- Q(s,a) represents the expected reward for taking action a in state s.
- α is the learning rate that controls how much the Q-values are updated.
- γ is the discount factor that represents how much future rewards are considered relative to immediate rewards.

- r is the immediate reward obtained by the agent after taking action a in state s.
- s′ is the state the agent transitions to after taking action a.
- $\max_{a'} Q(s',a')$ is the maximum Q-value for the next state s′, representing the optimal future reward from that state.

The goal of Q-learning is to learn an optimal policy that maximizes the cumulative rewards over time. The agent does this by following an epsilon-greedy policy, where it explores random actions with probability ϵ and exploits the best-known action (the one with the highest Q-value) with probability $1-\epsilon$. As the agent learns, the exploration rate ϵ decays over time to favor exploitation.

Limitations:

While Q-learning is a powerful algorithm, it faces limitations when applied to environments with large or continuous state spaces. Storing Q-values for every possible state-action pair becomes impractical, and the Q-table can grow exponentially with the number of possible states.

Deep Q-Network (DQN):

Deep Q-Network (DQN) is an extension of Q-learning designed to overcome the limitations of traditional Q-learning in high-dimensional and continuous state spaces. Instead of using a Q-table, DQN uses a deep neural network to approximate the Q-value function. This allows DQN to handle much more complex environments, such as playing Atari games or controlling robots, where the state space is too large to represent explicitly.

How DQN Works:

DQN replaces the Q-table with a neural network that takes the current state as input and outputs Q-values for each possible action. The neural network is trained to predict the Q-values and improve its

estimates by minimizing the difference between predicted Q-values and the target Q-values.

The Q-value update rule in DQN is similar to that of Q-learning, but it involves using the neural network to approximate the Q-values. The target Q-value y is calculated as:

$$y=r+\gamma \max_{a'} Q(s',a';\theta^{-})$$

Where:

- $Q(s',a';\theta^{-})$ is the predicted Q-value from the target network (a copy of the original network that is updated less frequently).
- θ^{-}represents the weights of the target network.

The neural network is trained using a loss function that minimizes the difference between the predicted Q-values and the target Q-values:

$$L(\theta)=(y - Q(s,a;\theta))^2$$

Where θ represents the weights of the main Q-network.

Key Techniques in DQN:

Experience Replay: DQN uses a technique called experience replay, where the agent stores past experiences (state, action, reward, next state) in a memory buffer. During training, the agent randomly samples mini-batches from this buffer to break the correlation between consecutive experiences. This helps improve the stability and efficiency of learning.

Target Network: DQN employs a target network, which is a copy of the Q-network. The target network's weights are updated less frequently, which stabilizes training by reducing the effect of rapidly changing Q-values. This approach prevents the network from oscillating or diverging during training.

Epsilon-Greedy Policy: DQN still follows the epsilon-greedy policy, where the agent explores random actions with probability ϵ and exploits the best-known action with probability 1−ϵ. As the agent

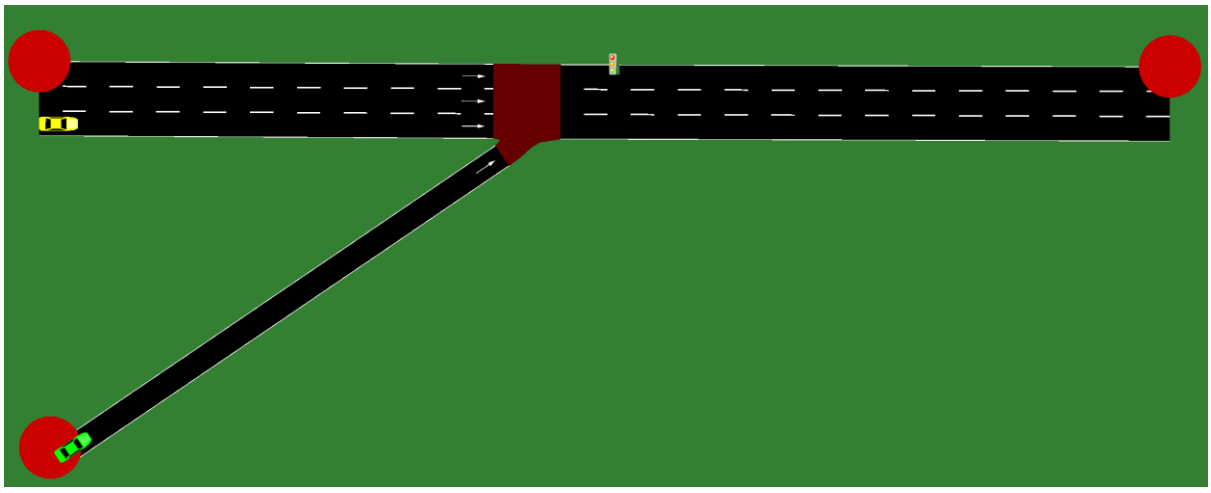
learns, ϵ decays over time, encouraging exploitation of the learned policy.

Advantages of DQN:

- Scalability: DQN can handle large and continuous state spaces that are impractical for traditional Q-learning.
- Stability: Techniques like experience replay and target networks help stabilize learning and improve performance.
- Flexibility: The ability to apply deep learning models enables DQN to work with complex environments like video games, robotics, and traffic management.

2) Description of simulation:

The simulation represents a traffic network with highway and ramp flows, junctions, and vehicle routes. It models vehicle movement, traffic demand, and road geometry using SUMO (Simulation of Urban MObility).



Road Network Configuration

Number of Lanes:

- The freeway section (E0, E1) consists of 3 lanes in each direction (total 3 lanes per edge). Each lane has a speed limit of 13.89 m/s (50 km/h).
- The ramp (E2) has a single lane entering from junction J3 to junction J1.

Road Geometry: The road network is designed with specific edge lengths, such as:

- Edge E0 (connecting J0 to J1) has a total length of 58.13 meters.
- Edge E1 (from J1 to J2) is 78.01 meters long, representing a continuation of the highway.
- Edge E2 (from J3 to J1) is 68.84 meters in length.

Junctions:

- J0: A dead-end junction where no traffic enters or exits.
- J1: A traffic light-controlled junction, allowing traffic from the highway (E0, E1) and ramp (E2) to interact. The traffic light has a cycle of green (42 seconds), yellow (3 seconds), and red phases (22 seconds for green, and 3 seconds for yellow).
- J2 and J3: Dead-end junctions where no incoming traffic exists.

Traffic Demand

- Freeway Demand: The highway experiences a high traffic flow with 2400 vehicles per hour (vph) on the three-lane road (E0 and E1).
- Ramp Demand: A ramp flow of 200 vehicles per hour enters from the ramp (E2) onto the freeway at junction J1, representing a lower volume of traffic compared to the highway flow.

Car-Following Model

The simulation uses standard car-following models (not explicitly described in the XML files, but SUMO typically utilizes such models for vehicle movement dynamics). These models control the acceleration and deceleration behavior of

vehicles depending on the distance to the vehicle in front, vehicle type, and traffic flow.

Traffic Light Control

The simulation uses a traffic light (controlled by junction J1) to manage the intersection of the highway (E0, E1) and ramp (E2). Initially, before the application of the reinforcement learning (RL) model, the traffic light operates based on a fixed program with specific durations for each light phase:

- Green light for 42 seconds.
- Yellow light for 3 seconds.
- Red light for 22 seconds.

Later in the simulation, the actions taken at the traffic light will change dynamically depending on the states and decisions made by the RL model, optimizing traffic flow based on real-time traffic conditions.

Simulation Duration and Setup

The simulation is set to run for 3600 seconds (1 hour), with continuous vehicle flow along the highway and ramp during this period. The ramp traffic flow is lower, reflecting the actual flow conditions of typical urban ramps.

3) RL problem reformulation:

In this reinforcement learning (RL) framework, the goal is to optimize the ramp metering control and highway traffic flow at the intersection controlled by the traffic light. The agent learns to take actions that minimize waiting times on the ramp while maintaining an optimal flow on the highway. The following provides the detailed reformulation of the RL problem, including the state definition, reward function, and action space.

a. State Definition:

The state in this problem encapsulates the traffic conditions on both the highway and the entering ramp. It is a vector that contains the following key elements:

- Highway Traffic Density: This represents the number of vehicles per unit of time or the level of congestion on the highway. A higher value indicates a higher density of vehicles, which might require careful management to maintain smooth flow.
- Ramp Queue Length: This indicates the number of vehicles waiting on the ramp. A longer queue suggests that the traffic light's green phase should be prolonged to allow more vehicles to enter the highway, reducing the ramp waiting time.
- Traffic Light Phase: This is a categorical variable representing the current phase of the traffic light (e.g., Green, Yellow, Red). The phase directly impacts the rate at which vehicles can enter the highway from the ramp.

The state is thus represented as:

State=(highway_traffic , ramp_queue , traffic_light_phase)

- highway_traffic: Continuous value between [0, 1] representing the density of traffic on the highway.
- ramp_queue: Continuous value between [0, 1] representing the number of vehicles in the ramp queue.
- traffic_light_phase: Discrete value (0, 1, or 2) representing the traffic light phase (Green, Yellow, or Red).

b. Reward Function

The reward function is designed to balance the goals of minimizing ramp waiting times while maintaining smooth traffic flow on the highway. The reward is calculated by combining the following factors:

- Highway Flow Reward (**highway_reward** or **highway_flow**): A positive reward for maintaining smooth traffic flow on the highway.
- Average Speed Reward (**vehicle_speed** or **speed_reward**): A positive reward for maintaining a high average speed on the highway. Higher speeds generally indicate good traffic flow, especially during green phases.

- Ramp Queue Penalty (**ramp_queue** or **queue_penalty**): A negative penalty for high ramp queue lengths. Longer queues indicate that vehicles are waiting too long to enter the highway, which we want to minimize.

The reward formula is structured as a weighted sum of these components. However, the exact formula varies throughout the tests we performed, with different weights applied to each factor depending on the test conditions. These weights are adjusted during experimentation to explore different optimization objectives, and we will explain the specific configurations in detail later.

c. Action Space

The action space is discrete and consists of three possible actions related to the traffic light phases:

- Action 0: Green Light (Ramp) – The traffic light phase for allowing vehicles to enter the highway from the ramp.
- Action 1: Yellow Light – Typically indicates a transition phase between green and red.
- Action 2: Red Light – The traffic light phase for stopping the flow of vehicles from the ramp to the highway.

4) Hyperparameters of each algorithm, code description and Evaluation and results:

Q-Learning:

Required Packages

The following Python packages are used in the simulation and reinforcement learning process:

- traci: Interface to control and interact with the SUMO traffic simulation.
- numpy: For numerical operations and managing arrays.
- matplotlib.pyplot: For creating visualizations (plots) of the results.
- random: For generating random numbers, particularly for exploration in the Q-learning algorithm.
- deque: For experience replay buffer management in Q-learning.

- xml.etree.ElementTree: For parsing and extracting data from XML files related to the SUMO network.

I have conducted five tests, each labeled as Q Learning Test (1 to 5), with progressively better results. The results for each test, including visualizations (cumulative reward, highway flow, ramp queue, and average speed over episodes), are presented at the end of each respective test file.

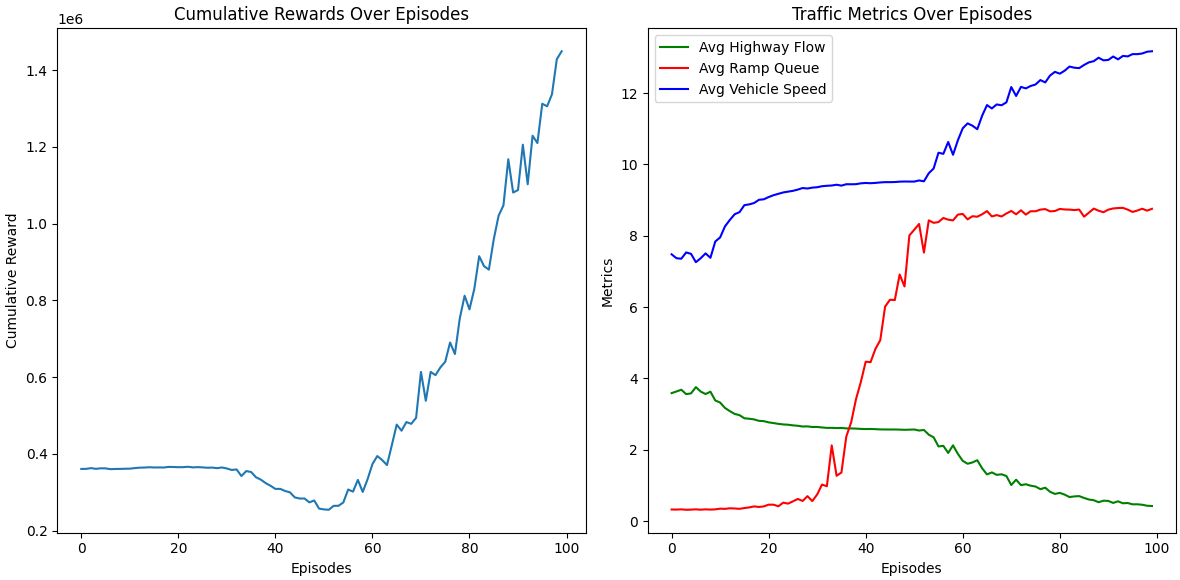
1) Q Learning test 1 :

The reward function used in this file is designed to optimize multiple factors, including highway flow, ramp queue, and vehicle speed. The formula for the reward function is as follows:

Reward Function:

$$R=(highway\_flow \cdot w1)+(ramp\_queue \cdot w2)+(vehicle\_speed \cdot w3)$$

w1=6.4, w2 =−2, and w3=3.7 are the weights assigned to each factor.



Hyper parameters:

- Epsilon (EPSILON): Initial exploration rate of 1.5, decaying over time to encourage more exploitation of the learned actions.
- Number of Episodes (NUM_EPISODES): The simulation runs for 100 episodes to allow the model to learn the optimal traffic light behavior.
- Learning Rate (ALPHA): 0.1, which controls how much new experiences affect the Q-values.

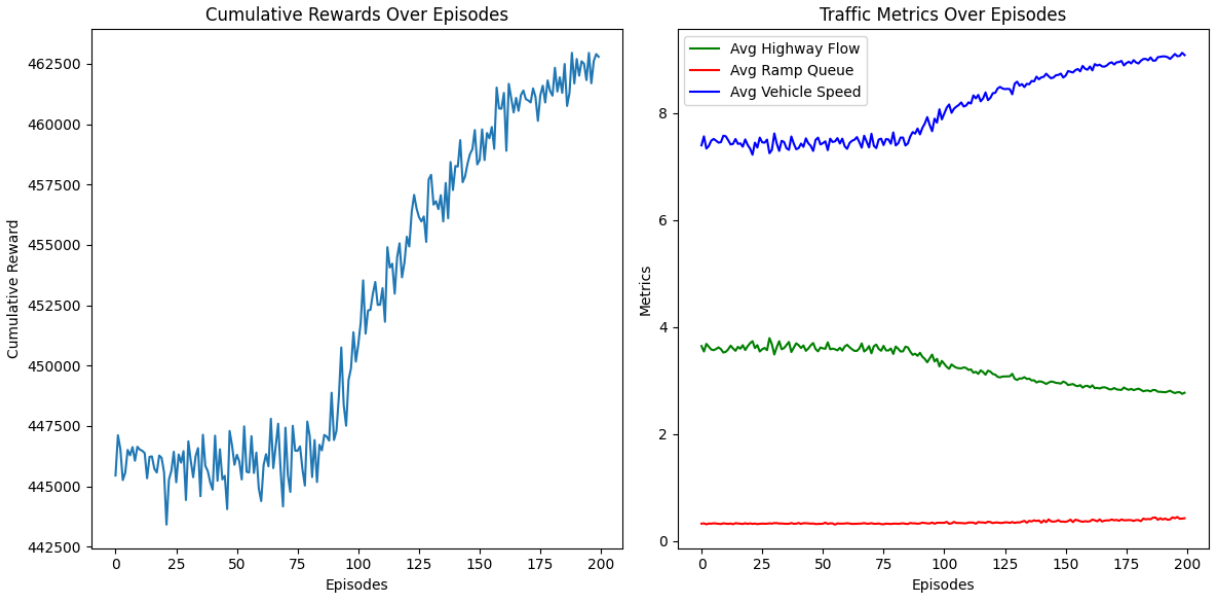
- Discount Factor (GAMMA): 0.95, which determines the importance of future rewards in the learning process.
- Epsilon Decay (EPSILON_DECAY): 0.05, causing epsilon to gradually decrease, shifting the model from exploration to exploitation.

2) Q Learning test 2 :

The updated version of the code we doubled the number of episodes from 100 to 200. This results in smoother reward curves and more stable traffic metrics. However, the overall trends in highway flow, ramp queue, and vehicle speed remain similar, suggesting that while the total reward improved slightly, the underlying traffic dynamics did not change significantly.

$R=(\text{highway\_flow}\cdot w1)+(\text{ramp\_queue}\cdot w2)+(\text{vehicle\_speed}\cdot w3)$

$w1=7$, $w2=-2.1$, and $w3=5$ are the weights assigned to each factor.



Hyper parameters:

- Epsilon (EPSILON): Initial exploration rate of 1.5, decaying over time to encourage more exploitation of the learned actions.
- Number of Episodes (NUM_EPISODES): The simulation runs for 200 episodes to allow the model to learn the optimal traffic light behavior.
- Learning Rate (ALPHA): 0.1, which controls how much new experiences affect the Q-values.

- Discount Factor (GAMMA): 0.95, which determines the importance of future rewards in the learning process.
- Epsilon Decay (EPSILON_DECAY): 0.005, causing epsilon to gradually decrease, shifting the model from exploration to exploitation.
- Minimum Epsilon (MIN_EPSILON): 0.1, which ensures that the epsilon value doesn't drop below this threshold, allowing for some exploration later in training.

3) Q Learning test 3 :

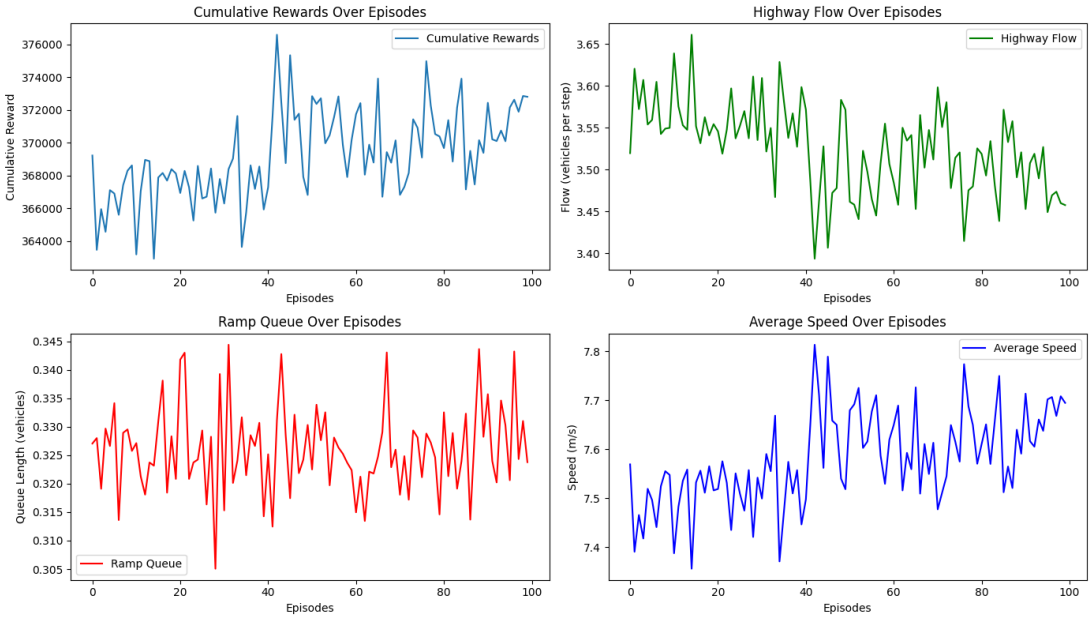
The updated version of the code includes a refined reward function. While highway flow and ramp queue metrics have stabilized, the reward curves still show fluctuations without significant improvement. The overall trends are consistent, but the fluctuations indicate the need for further tuning in future files to ensure more stable results.

Reward Function:

$$R=(10.0 \cdot \log(1+\text{highway\_flow}))+(5.0 \cdot \text{avg\_speed})-(5.0 \cdot \text{ramp\_queue})$$

Where:

- Highway Flow: The reward is calculated using a logarithmic function to moderate the effect of increased flow as traffic becomes denser.
- Average Speed: A moderate weight of 5.0 is applied to encourage higher average speeds.
- Ramp Queue: A heavier penalty (-5.0) is applied to ramp queue to reduce congestion.



(this plot is different from the other ones because we ran multiple tests and retrieved the best result from each version)

Hyper parameters:

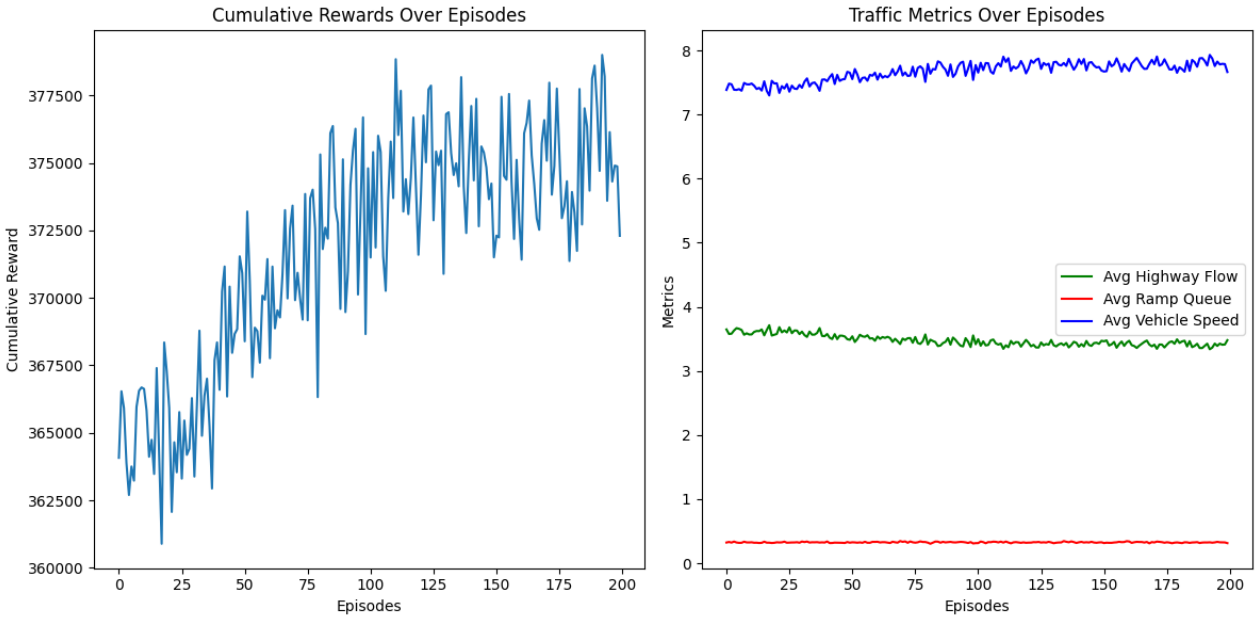
- Epsilon (EPSILON): Initial exploration rate of 1.0, decaying over time to encourage more exploitation of the learned actions.
- Number of Episodes (NUM_EPISODES): The simulation runs for 100 episodes to allow the model to learn the optimal behavior.
- Learning Rate (ALPHA): 0.1, controlling how much new experiences affect the Q-values.
- Discount Factor (GAMMA): 0.95, determining the importance of future rewards in the learning process.
- Epsilon Decay (EPSILON_DECAY): 0.01, causing epsilon to gradually decrease, shifting the model from exploration to exploitation.
- Minimum Epsilon (MIN_EPSILON): 0.001, ensuring that the epsilon value doesn't drop below this threshold, allowing for some exploration later in training.

4) Q Learning test 4 :

This version of the simulation successfully improves cumulative rewards without significantly compromising highway flow or ramp queue metrics. The reward curves show notable enhancements while retaining stability across traffic metrics. However, further tuning is necessary to optimize the balance between highway flow and ramp queue congestion fully. Adjustments to the reward system and hyper parameters played a crucial role in achieving this outcome.

Reward Function:

$$R=(10.0\cdot\log(1+\text{highway\_flow}))+(5.0\cdot\text{avg\_speed})-(5.0\cdot\text{ramp\_queue})$$



the key difference between this version and the previous one is in the hyper parameters :

Learning Rate (α): 0.1 - Controls the extent of Q-value updates.

Discount Factor (γ): 0.95 - Balances immediate and future rewards.

Initial Exploration Rate (ϵ): 2.0 - High initial exploration rate.

Epsilon Decay (ϵ decay): 0.025 - Allows gradual reduction of exploration.

Minimum Epsilon (ϵ min): 0.05 - Ensures ongoing exploration during training.

Number of Episodes: 200 - Length of the simulation.

5) Q Learning test 5 Final :

In this final implementation, we achieved significant improvements in highway traffic flow while maintaining a balanced ramp queue. The average speed remained good, ensuring vehicles avoided congestion, thanks to the optimized ramp metering strategy.

Key Observations:

- Improved Highway Flow: The flow on the highway increased significantly, indicating efficient utilization of the available lanes and traffic throughout.
- Controlled Ramp Queue: Despite the improvement in highway flow, the ramp queue remained well-managed due to the reward function's design, which penalized excessive queue length.

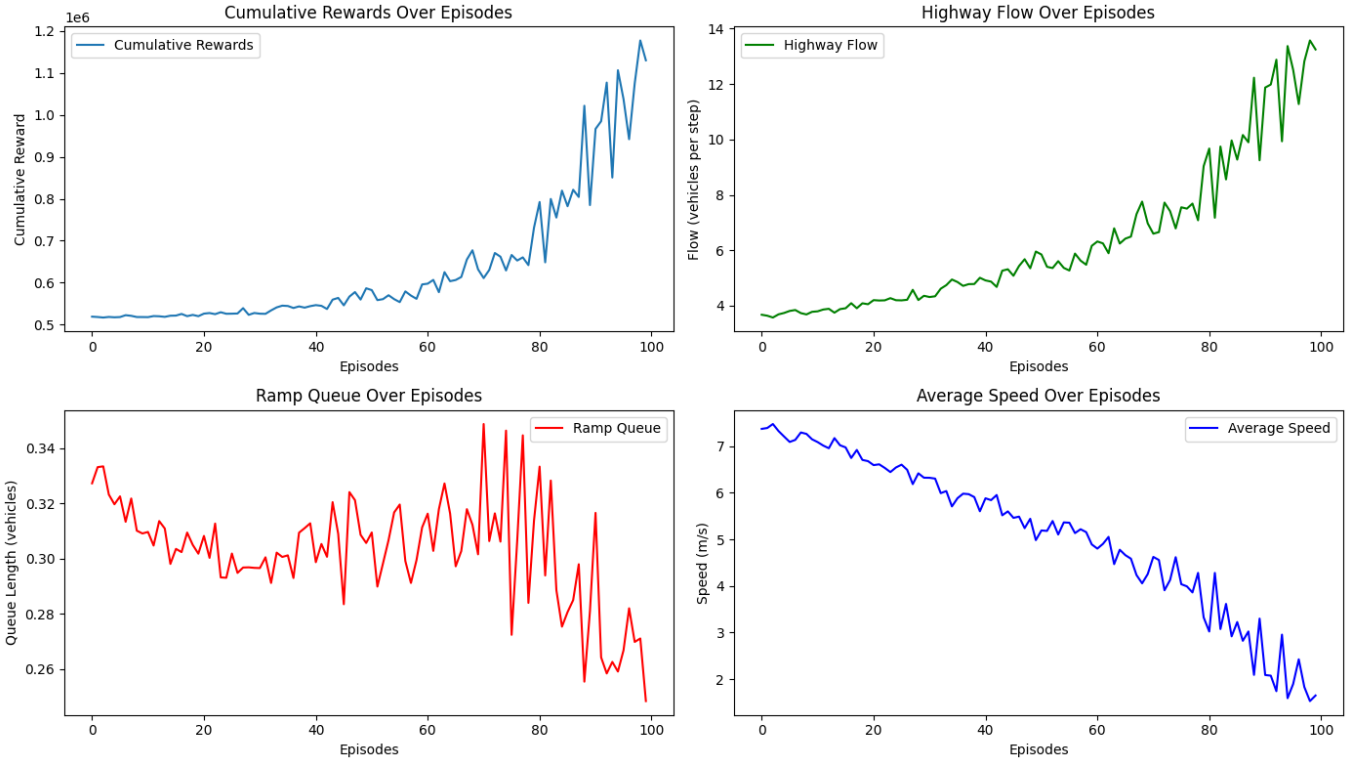
Reward Function:

The reward function was pivotal to these results. It incorporated three components:

- Highway Flow Reward: Encouraged maximizing the flow on the highway with a linear scaling based on the normalized flow.
- Ramp Queue Penalty: Discouraged long queues on the ramp to avoid traffic spillover and maintain balance.

Mathematically:

$$\mathbf{R=10\cdot \frac{highway\_flow}{\max\_highway\_flow} +5\cdot avg\_speed -5\cdot \frac{ramp\_queue}{\max\_ramp\_queue}}$$



Hyper parameters Used:

- Learning Rate (α): 0.1
- Discount Factor (γ): 0.95
- Initial Exploration Rate (ϵ): 1.0
- Epsilon Decay: 0.01
- Minimum Exploration Rate (min ϵ): 0.001
- Number of Episodes: 100

This configuration ensured stable learning and convergence of the Q-learning algorithm while balancing exploration and exploitation effectively. The results validated the approach, showcasing an efficient ramp metering control system optimized for real-world highway traffic scenarios.

DQN:

1. Technologies and Frameworks Used

- tensorflow and keras: For constructing and training the DQN model.
- matplotlib: For visualizing results like cumulative rewards and traffic metrics.
- collections.deque: For implementing an efficient experience replay buffer.

2. Neural Network Model for Q-Learning

The DQN model approximates the Q-value function Q(s,a), mapping states (s) to expected rewards for actions (a).

Model Architecture:

- Input layer: state_size=3 (highway traffic, ramp queue, traffic light phase).
- Two hidden layers: Each with 24 neurons and ReLU activation for non-linearity.
- Output layer: action_size=3 (representing different traffic light phases for ramp control).
- Loss Function: Mean Squared Error (MSE).
- Optimizer: Adam with a learning rate of 0.001.

3. Reward System

$$\textbf{Reward} = 2 \times \textbf{Avg Speed} + 7 \times \textbf{Highway Flow} - 5 \times \textbf{Ramp Queue}$$

we only need one version to achieve a good result, so we didn't need to tweak the weight of the reward function too much, we only had to chance the weights one time, which was good because it took too long (10 hours) to fully train the model (cost too much computer energy, it had to be on executing the code for a long time none stop).

4. Exploration-Exploitation Tradeoff, Experience Replay and Target Network:

Epsilon-Greedy Policy: Balances exploration and exploitation:

- Starts with $\epsilon = 1.0$ (exploration) and decays to $\epsilon_{min} = 0.01$ with a decay rate of 0.995.
- Ensures sufficient exploration during early episodes and gradual exploitation as training progresses.

Experience Replay: Stores past experiences in a buffer for sampling and training. This breaks the correlation between consecutive experiences, stabilizing training.

Target Network: A separate model periodically updated to mirror the main network. It provides a stable target for Q-value updates.