Reinforcement Learning for Ramp Metering on Highways

SAIDA Haithem, MARHOUM Zineddine, OUKID Anis, FERHATI Khalil

Module: Reinforcement Learning and Optimal Control

Contents

1 Introduction

The objective of this project is to apply reinforcement learning (RL) algorithms, specifically Q-learning and Deep Q-Networks (DQN), to control ramp metering on highways. Ramp metering involves managing the flow of vehicles entering a highway via a controlled traffic light on the ramp. The goal is to optimize traffic flow on both the highway and the ramp, minimizing congestion and waiting times.

To achieve this, the project uses the Simulation of Urban Mobility (SUMO) simulator to model a realistic traffic environment. SUMO provides the ability to simulate car-following behaviors, lane-changing dynamics, and traffic demand under various conditions. The RL agent is trained within this simulated environment to learn optimal policies for controlling the ramp traffic light.

The project involves implementing the Q-learning and DQN algorithms from scratch, analyzing their performance, and comparing the results under different scenarios. The trained policies are evaluated based on total rewards and specific traffic performance metrics. This report details the simulation setup, problem formulation, implementation steps, and evaluation results, highlighting the effectiveness of reinforcement learning in solving real-world traffic optimization problems.

2 Description of Simulations

The simulation for ramp metering control on the highway was conducted using the SUMO (Simulation of Urban Mobility) traffic simulator. The purpose of this simulation is to analyze the effects of reinforcement learning-based traffic control strategies, specifically Q-learning and Deep Q-learning, on optimizing traffic conditions.

2.1 Highway Configuration

- **Number of Lanes:** The highway consists of three lanes, while the ramp has a single dedicated lane for merging.
- **Road Length:**
  - The main highway stretches 1 kilometer, with the merge point dividing it into two equal segments: 500 meters before and 500 meters after the ramp junction.
  - The ramp itself has a total length of 136.15 meters, allowing for acceleration and merging.
- **Speed Limits:**
  - Highway: Vehicles are permitted to travel at a maximum speed of 33.33 m/s (120 km/h).
  - Ramp: The maximum speed is restricted to 16.67 m/s (60 km/h) to account for merging conditions.

2.2 Traffic Control and Intersection

- **Traffic Light Configuration:**

- The ramp is regulated by a traffic light situated at the merge point.
- The default static traffic light program consists of:
  * **Highway green phase:** Lasting for 30 seconds.
  * **Yellow transition phases:** Lasting for 3 seconds each.
  * **Ramp green phase:** Lasting for 20 seconds.

2.3 Vehicle Behavior Models

- **Car-Following Model:** SUMO's built-in car-following model governs vehicle acceleration, deceleration, and gap maintenance.
- **Lane-Changing Model:** Vehicles adaptively change lanes using SUMO's lane-changing logic.

2.4 Vehicle Types

- **Passenger Cars:** Represented with the following characteristics:
  - Length: 5 meters
  - Maximum Speed: 40 m/s (144 km/h)
  - Acceleration: 2.6 m/s^2
  - Deceleration: 4.5 m/s^2

2.5 Traffic Demand

- **Highway Traffic:**
  - Flow rate: 1800 vehicles per hour
  - Distributed randomly across three lanes
- **Ramp Traffic:**
  - Flow rate: 600 vehicles per hour
  - Single ramp lane

3 Reinforcement Learning Problem Reformulation

3.1 Problem Formulation Overview

MDP Formulation

The ramp metering control problem is formulated as a Markov Decision Process (MDP) with the following components:

- **State Space** $\mathcal{S}$: Traffic conditions on highway and ramp
- **Action Space** $\mathcal{A}$: Traffic light control decisions
- **Transition Function** $P(s_{t+1}|s_t, a_t)$: Traffic evolution dynamics
- **Reward Function** $R(s_t, a_t)$: Traffic optimization objectives

3.2 State Space Definition

State Representation

The state s_t at time t is defined as a 5-dimensional vector:

$$s_t = (q_h, v_h, q_r, v_r, w_r)$$

Where each component represents:

- **Highway Density** (q_h):
  - Units: vehicles/km
  - Range: [0, 120] vehicles/km
  - Measurement: Average over 500m highway section
- **Highway Speed** (v_h):
  - Units: km/h
  - Range: [0, 120] km/h
  - Measurement: Average of all vehicles on highway
- **Ramp Queue** (q_r):
  - Units: number of vehicles
  - Range: [0, 20] vehicles
  - Measurement: Count of waiting vehicles
- **Ramp Speed** (v_r):
  - Units: km/h
  - Range: [0, 60] km/h
  - Measurement: Average speed on ramp
- **Waiting Time** (w_r):
  - Units: seconds
  - Range: [0, 300] seconds
  - Measurement: Average waiting time per vehicle

3.3 Action Space Definition

Control Actions

The action space $\mathcal{A}$ is binary:

$$a_t \in \{0, 1\}$$

With the following interpretations:

- **Red Light** ($a_t = 0$):
  - Stops vehicles from entering highway
  - Allows highway traffic to flow freely
  - Increases ramp queue length
- **Green Light** ($a_t = 1$):
  - Allows vehicles to enter highway
  - May cause temporary highway slowdown
  - Reduces ramp queue length

Action Duration: Each action is maintained for a minimum of 5 seconds to prevent rapid switching.

3.4 Reward Function Design

Multi-objective Reward

The reward function balances multiple traffic objectives:

$$R(s_t, a_t) = -\left(\alpha \cdot w_r + \beta \cdot \max(0, q_h - q_{\max}) + \gamma \cdot \max(0, v_{\min} - v_h)\right)$$

Component Weights:

- $\alpha = 0.4$: Waiting time penalty
- $\beta = 0.3$: Congestion penalty
- $\gamma = 0.3$: Speed penalty

Threshold Values:

- $q_{\max} = 80$ vehicles/km: Maximum desired highway density
- $v_{\min} = 60$ km/h: Minimum acceptable highway speed

Reward Components Analysis:

1. **Waiting Time Term** ($\alpha \cdot w_r$):
   - Penalizes long queues on ramp
   - Linear relationship with waiting time
   - Range: [0, 120] (normalized)
2. **Congestion Term** ($\beta \cdot \max(0, q_h - q_{\max})$):
   - Activates only when density exceeds threshold
   - Quadratic penalty for severe congestion
   - Range: [0, 100] (normalized)
3. **Speed Term** ($\gamma \cdot \max(0, v_{\min} - v_h)$):
   - Ensures minimum flow speed
   - Linear penalty for slow traffic
   - Range: [0, 60] (normalized)

3.5 Optimization Objective

Policy Optimization

The goal is to find an optimal policy $\pi^*(s_t)$ that maximizes:

$$\max_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$$

Where:

- $\gamma = 0.95$: Discount factor
- Time horizon: 3600 steps (1-hour simulation)
- Episode termination:
  - Maximum time reached
  - Critical congestion ($q_h > 100$ vehicles/km)
  - Excessive waiting time ($w_r > 300$ seconds)

4 Q-Learning and DQN Algorithms

4.1 Q-Learning Algorithm

Q-Learning is a model-free reinforcement learning algorithm that learns the optimal action-selection policy in a Markov Decision Process (MDP). The algorithm maintains a **Q-table**, where each entry $Q(s, a)$ represents the expected cumulative reward for taking action a in state s and following the optimal policy thereafter.

4.1.1 Algorithm Description

Q-Learning Core Components

1. **Q-Table Initialization:**
   - Initialize Q(s,a) arbitrarily for all state-action pairs
   - Common initialization: zeros or small random values
2. **Action Selection (ϵ-greedy):**
   - With probability ϵ: select random action (exploration)
   - With probability $1 - \epsilon$: select $a = \arg\max_a Q(s,a)$ (exploitation)
3. **Q-Value Update Rule:** The Q-values are updated iteratively using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_a Q(s',a) - Q(s,a) \right]$$

   where:
   - α: Learning rate (0.1 in our implementation)
   - γ: Discount factor (0.95 in our implementation)
   - r: Immediate reward
   - s': Next state
   - $\max_a Q(s',a)$: Maximum future Q-value

4.1.2 Implementation Details

Q-Learning Implementation

- **State Discretization:**
  - Traffic density: 3 levels (low, medium, high)
  - Average speed: 3 levels (slow, medium, fast)
  - Queue length: 3 levels (short, medium, long)
  - Results in 27 possible states (3×3×3)
- **Action Space:**
  - Binary actions: {0: Red light, 1: Green light}
  - Q-table dimensions: 27×2 (states × actions)
- **Exploration Strategy:**
  - Initial $\epsilon = 1.0$ (pure exploration)
  - Decay rate = 0.995 per step
  - Minimum $\epsilon = 0.01$
  - Ensures thorough state-space exploration

4.2 Deep Q-Network (DQN) Algorithm

DQN extends Q-Learning by using a neural network to approximate the Q-function, making it suitable for problems with large or continuous state spaces. The Q-function is represented by a neural network $Q_\theta(s, a)$, where θ are the network parameters.

4.2.1 Key Innovations

DQN Improvements over Q-Learning

1. **Experience Replay Buffer:**
   - Stores transitions (s, a, r, s')
   - Buffer size: 10,000 experiences
   - Randomly samples minibatches of size 64
   - Breaks correlations in sequential data
   - Improves sample efficiency
2. **Target Network:**
   - Separate network $Q_{\theta'}$ for computing targets
   - Updated every 1,000 steps
   - Stabilizes training by reducing moving target problem
   - Parameters copied from main network
3. **Loss Function:**

$$\mathcal{L}(\theta) = E\left[\left(r + \gamma \max_a Q_{\theta'}(s', a) - Q_\theta(s, a)\right)^2\right]$$

4.2.2 Neural Network Architecture

4.2.3 Training Process

DQN Training Workflow

1. **State Processing:**
   - Normalize state values to [0,1]
   - Convert to PyTorch tensors
2. **Action Selection:**
   - ϵ-greedy strategy
   - Decaying exploration rate
   - Action selection based on Q-values
3. **Learning Update:**
   - Sample minibatch from replay buffer
   - Compute target Q-values
   - Update network parameters using Adam
   - Periodically update target network
4. **Optimization Details:**
   - Learning rate: 0.001
   - Gradient clipping: [-1, 1]
   - L2 regularization: 0.01

5 Implementation Details

5.1 Project Structure

```
RL-Project/
          code/
                training/
                      qlearning.py          # Q-Learning implementation
                      dqn.py               # Deep Q-Network implementation
                utils/
                    env.py                 # SUMO environment wrapper
          config/
                highway.net.xml            # Highway network definition
                highway.rou.xml            # Traffic route definition
          models/
              dqn_model.pth             # Trained DQN model weights
              qlearning_table.pkl       # Q-Learning table
```

5.2 Core Dependencies

Required Packages

- **torch** $\geq$ 1.9.0: Deep Learning framework for DQN implementation
- **numpy** $\geq$ 1.19.5: Numerical computations and array operations
- **sumo** $\geq$ 1.8.0: Traffic simulation environment
- **matplotlib** $\geq$ 3.3.4: Visualization of training results
- **pickle**: Model serialization and deserialization

5.3 Model Parameters

5.3.1 Q-Learning Configuration

Q-Learning Parameters

- **Learning rate** (α): 0.1
  - Controls how much new information overrides old Q-values
  - Chosen to balance learning speed and stability
- **Discount factor** (γ): 0.95
  - Balances immediate vs. future rewards
  - Higher value emphasizes long-term planning
- **Exploration rate** (ϵ):
  - Initial value: 1.0 (pure exploration)
  - Minimum value: 0.01
  - Decay rate: 0.995 per step
- **State discretization**: 6 dimensions
  - Traffic density (3 levels)
  - Average speed (3 levels)
  - Queue length (3 levels)

5.3.2 DQN Architecture and Parameters

DQN Configuration

- **Neural Network Architecture**:
  - Input Layer: 6 neurons (state dimensions)
  - Hidden Layer 1: 64 neurons with ReLU
  - Hidden Layer 2: 64 neurons with ReLU
  - Output Layer: 2 neurons (Q-values for actions)
- **Training Parameters**:
  - Learning rate: 0.001 (Adam optimizer)
  - Batch size: 64 experiences
  - Memory size: 10000 transitions
  - Target network update: Every 100 steps
  - Exploration decay: 0.995
- **Experience Replay**:
  - Buffer size: 10000 transitions
  - Sampling strategy: Uniform random

– Batch composition: (state, action, reward, next$_state$)

5.4 Training Process

Training Workflow

1. **Environment Initialization**:
   - Setup SUMO simulation
   - Configure traffic parameters
   - Initialize state variables
2. **Training Loop**:
   - Episodes: 100 training episodes
   - Steps per episode: 3600 (1 hour simulation)
   - State observation and preprocessing
   - Action selection using ϵ-greedy policy
   - Environment interaction and reward collection
   - Model updates (Q-table or DQN)
3. **Model Evaluation**:
   - Periodic performance assessment
   - Metrics tracking and visualization
   - Model checkpointing

5.5 Execution Instructions

Running the Project

5.5.1 Training New Models

```
# Train both models
python train_and_evaluate.py

# Parameters in train_and_evaluate.py:
# - EPISODES: Number of training episodes
# - MAX_STEPS: Maximum steps per episode
```

5.5.2 Running Trained Models

```
# Run DQN model
python run_trained_model.py --model dqn

# Run Q-Learning model
python run_trained_model.py --model qlearning

# Optional parameters:
# --simulation_time: Duration (default: 3600)
# --delay: Visualization delay (default: 0.1s)
```

6 Results and Analysis

6.1 Training Performance

- Q-Learning showed stable learning but limited scalability
- DQN demonstrated better generalization in continuous state spaces

6.2 Real-time Metrics

- Average vehicle speed
- Waiting times at ramps
- Number of vehicles served
- Cumulative rewards

7 Future Improvements

1. Implement Prioritized Experience Replay
2. Add A3C algorithm implementation
3. Include more complex traffic scenarios
4. Optimize hyperparameters