Deep Reinforcement Learning for Traffic Control

Antoine Chosson Sasha Hakim William Barthélémy Raphaël Keghian

Under the supervision of :

Nadir Fahri - Grettia Laboratory

Zoi Christophorou - Patras University

May 26, 2025

Contents

Introduction

General Problem Statement

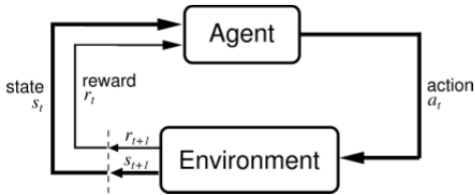
Increased ownership of private cars cause congestion in cities and poses economic, environmental and social disruptions

- Up to **500.000 vehicles** pass through paris each day
- Drivers in Paris spend an average of **138 hours** per year in traffic
- Urban congestion cost the US economy over **$120 billion** in lost time and fuel in 2020
- **91% of cars** sold in the US are "connected"
- **1 out of 5** traffic lights worldwide uses a form of adaptive control

Project objective :

How to use machine learning to optimize traffic flows at an intersection ?

Deep Reinforcement Learning (DRL)



Policy-based algorithms

Directly learn and optimize a policy that maps states to action probabilities to maximize long-term rewards

Value-based algorithms

Estimate the expected return of each action and select the one with the highest value in each state

REINFORCE algorithm - introduction

- First introduced in 1992 by Professor Ronald J. Williams
- Policy-based algorithm
- Learns from trial and error
- Effective for problems involving unknown or complex environments that are difficult to model analytically
- No explicit model of the environment's dynamics
- No knowledge of the driver's behaviors or the previous car flows

Two pillars : **Monte Carlo Sampling** and the **Policy gradient theorem**

Monte Carlo sampling method

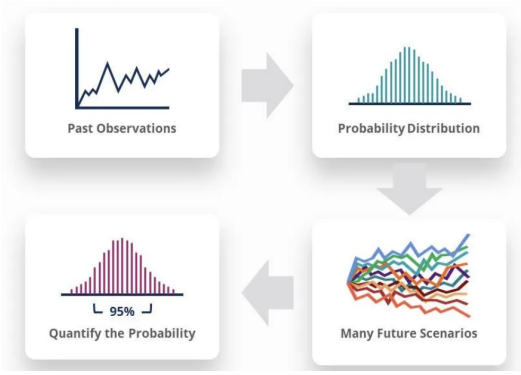


Figure: Principle of Monte Carlo sampling

Monte Carlo - Principle

The Monte Carlo Method is used to estimate the expected return by sampling episodes of interaction with the environment. By observing sequences of states, actions, and rewards until termination, it provides an unbiased estimate of the total reward associated with each trajectory. These sampled returns are then used to guide the policy updates

Policy Gradient theorem

Theorem

The gradient of the expected cumulative reward with respect to the policy parameters can be expressed as the expectation of the product of the return and the gradient of the log-probability of the taken actions

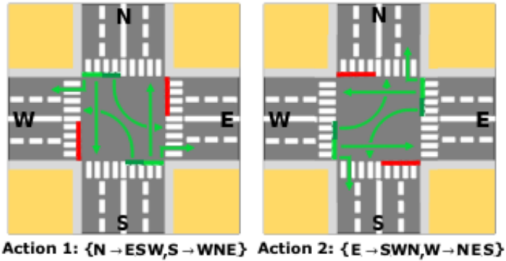
REINFORCE algorithm - theory

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Traffic scenario



- two-phase, double-lane intersection
- permitted conflicting left turns

Traffic scenario

Each lane benefits from two permissive green intervals, corresponding to each primary axis, allowing left turns, through movements, and right turns. **The agent's role is to alternate between the following two traffic signal phases**:

Phase 1: Controls the north-south axis, enabling movements from north to east, south, and west, as well as from south to west, north, and east.
Phase 2: Controls the east-west axis, allowing movements from east to south, west, and north, as well as from west to north, east, and south.

Action space

$$A = \{(n \to esw,\ s \to wne),\ (e \to swn,\ w \to nes)\},$$

Materials and Methods

Agent – Working Tools

PyTorch

Library for building and training neural networks. Used to implement the policy network (MLP) and perform backpropagation using stochastic gradient ascent.

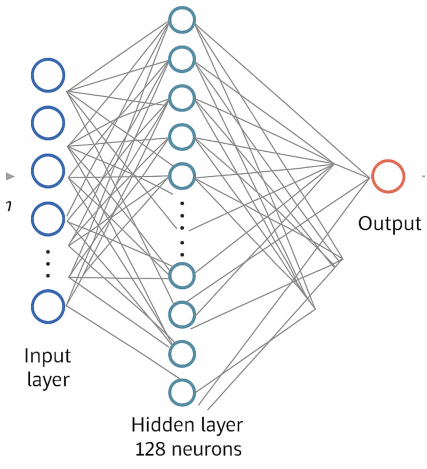
SUMO and TraCI

- **SUMO:** Microscopic, open-source traffic simulator used to create realistic road networks and vehicle flows.
- **TraCI:** Traffic Control Interface that allows real-time communication between SUMO and the Python agent.

TensorBoard

Visualization toolkit used to monitor training metrics (reward, KPIs) in real time.

Agent – Implementation - Policy




```python
class PolicyNetwork(nn.Module):
    def __init__(self,
        state_dim, action_dim):
        super(PolicyNetwork,
            self).__init__()
        self.fc1 = nn.Linear(
            state_dim, 128)
        self.fc2 = nn.Linear
            (128, action_dim)

    def forward(self, state):
        x = F.relu(self.fc1(
            state))
        action_probs = F.
            softmax(self.fc2(x),
             dim=-1)
        return action_probs
```

Agent - Preliminary tests on a gym environment

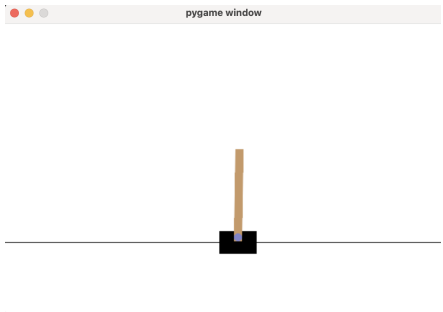
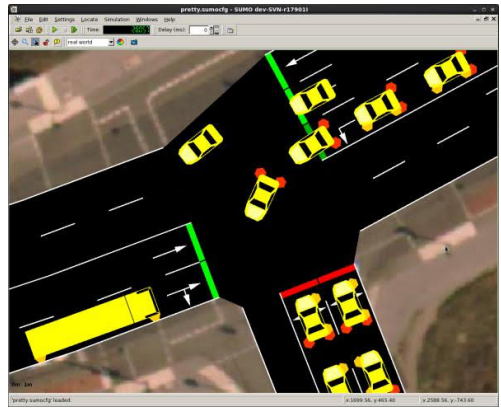


Figure: Testing our agent on CartPole-v1

Environment – Simulation Of Urban Mobility (SUMO)

- **SUMO** is an open-source, microscopic traffic simulator developed by DLR
- Used to model and simulate vehicle movement, traffic lights, and urban layouts
- Interacts in real time with external agents using `TraCI`
- Supports reinforcement learning by enabling custom control of signal phases
- Provides rich traffic data (waiting time, queue length, emissions, etc.)

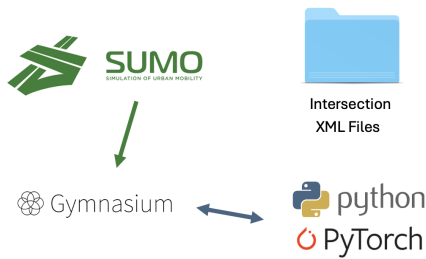


Environment - Relevant metrics (KPIs)

- **Episode mean accumulated waiting time (s)**: Total time vehicles spend waiting at red lights over the course of an episode.
- **Episode mean total delay (s)**: Average reduction in vehicle speed compared to free-flow conditions (i.e., traveling at maximum allowed speed).
- **Episode mean total queue length (number of vehicles)**: Average number of vehicles waiting per timestep across all lanes.
- **Episode mean total volume (number of vehicles)**: Average number of vehicles present within the intersection during the episode.

Integration – Project Architecture

```
> running_scripts/
      >   train.py
      >   observe.py
> results/
      >   kpis.csv
      >   kpis.py
> environment/
      >   sumo_env.py
      >   scheduler.py
> agent/
      >   reinforce_agent.py
      >   reinforce_agent.pth
```



Integration - Scheduler

Enhancing the realism of the situation with a `scheduler.py` file which defines a set of rules for governing traffic light phases.

`set_cooldown()`

Enforces a minimum delay between two consecutive phase changes to prevent flickering of the traffic lights.

`is_congested()`

Detects congestion levels and forces the agent to take action when excessive queuing is observed, making the model more responsive to real-time traffic conditions.

`can_act()`

Determines whether the agent is currently allowed to modify the traffic signals based on cooldown constraints and congestion status.

Integration – Computing the Reward

- **Objective:** Minimize total travel time at the intersection
- **Metric: Total Squared Delay (TSD)**

$$TSD(t) = \sum_i \left(1 - \frac{v_i(t)}{v_{\text{max}}}\right)^2$$

- **Normalized reward:**

$$R(t) = 1 - \frac{TSD(t)}{\max(TSD(t), TSD(t-1))}$$

- **Properties:**
  - Penalizes large delays more than short ones (fairness)
  - Encourages flow over blockage
- **Teleportation penalty:** A large negative reward is applied if vehicles are forcefully removed from the simulation.

Integration – Training Phase

- **Episodes:** 2000 total episodes, 500 steps per episode
- **Policy update:** Every 5 episodes
- **Learning rate:** 10^{-4}, $\gamma = 0.99$
- **Optimization:**
  - Based on Policy Gradient Theorem
  - Updates parameters to maximize expected return
- **Model saving:**
  - Best-performing model saved as `reinforce_agent.pth`
- **Training Time:** $\approx$ 15 mins on local GPU

Integration - Dealing with vehicle teleportations

Problem : When lanes get fully filled with cars, SUMO "teleports" i.e. removes the cars from the lane which wrongly contributes to the improvement of the traffic flow.

Solution : Introduce a high penalty in case of car teleportation in order to prevent the wrong model from being saved.

```
if teleport_count > 0:
    self.teleport_flag = True
    return -1000.0
else:
    self.teleport_flag = False
    return -tsd
```

NB : What seems like the best reward might not have the best effect when applied to a concrete scenario !

Results

Training logs - Reward

Figure: Logging reward while training in tensorboard

Training logs - KPIs

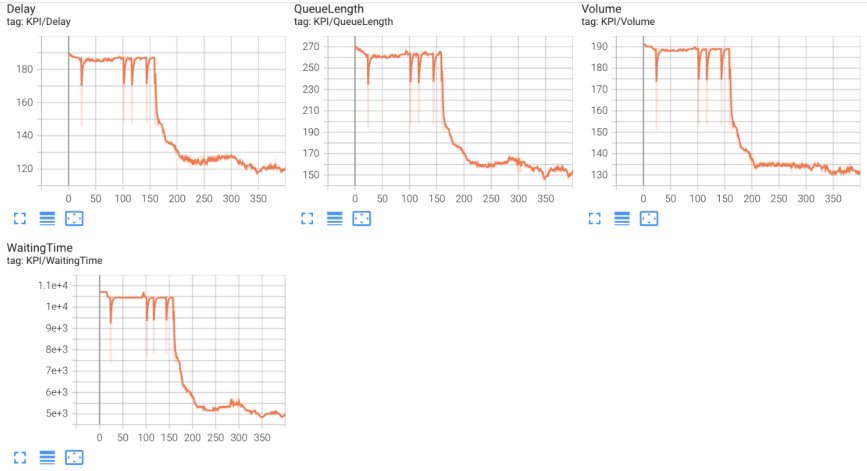


Figure: Logging KPIs while training in tensorboard

Observation

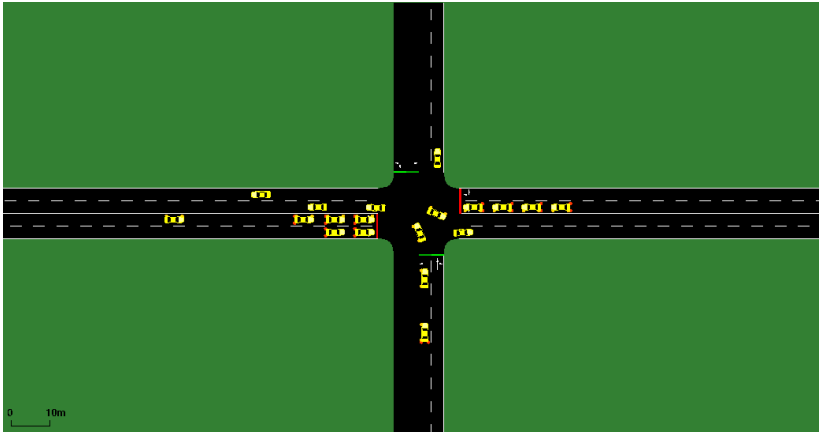
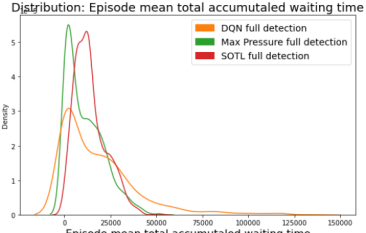


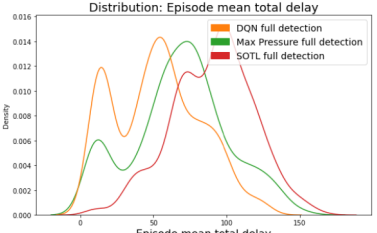
Figure: Observation with SUMO-Gui interface

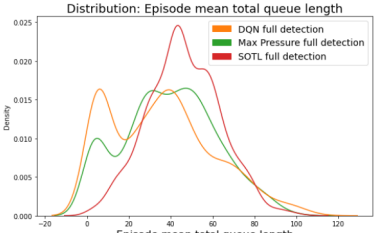
Statistics

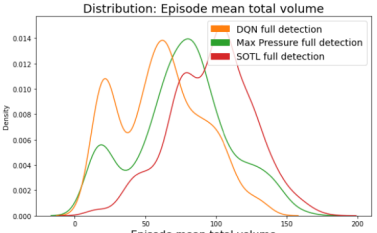
Indicator	KPI	DQN	MP	SOTL	REINFORCE
Mean	Waiting time	18280,2	11130,9	13864,7	14500
	Delay	52,8	68,1	90,5	62
	Queue Length	35,7	39,3	46,2	37
	Volume	60,6	76	96,8	70
Median	Waiting time	12542	8528,5	12433,5	14200
	Delay	53,3	69,8	94	61
	Queue Length	35,7	40	44,6	36,5
	Volume	61,1	77,4	100	69
First Quartile	Waiting time	475,1	2543	7325,2	11500
	Delay	24,6	50,4	71,3	50
	Queue Length	10,6	24,5	34,4	30
	Volume	33,4	57,6	77,1	60
Third Quartile	Waiting time	27001,3	17845,1	18521,8	17500
	Delay	74	87,4	108,8	74
	Queue Length	50,9	54,3	58	44
	Volume	82,9	95,5	115,3	80
Variance	Waiting time	452039780,8	103870420,5	74104235,1	520000000
	Delay	873,3	1088,5	771,7	450
	Queue Length	611,7	489,4	295,4	320
	Volume	904,3	1138,4	817,3	95000

Density distributions - DQN,MP,SOTL

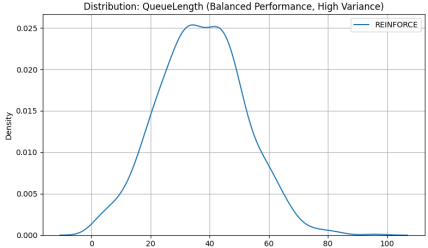


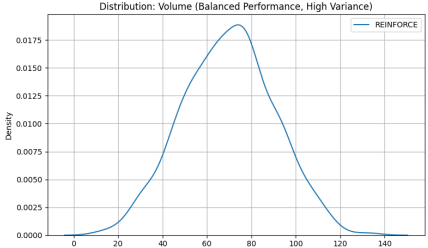


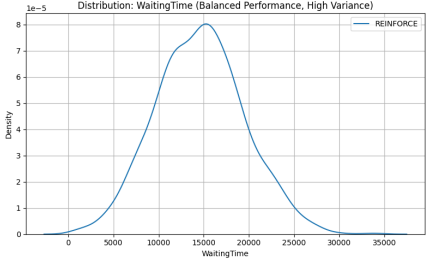


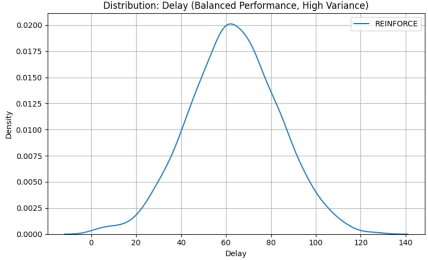


Density distributions - REINFORCE









Conclusions

Comparing with other Algorithms

Deep Q-Network (DQN)

Model-free, value-based reinforcement learning method that approximates the optimal action-value function using deep neural networks. It learns by estimating the long-term expected rewards for different traffic light actions at each intersection state

Max Pressure (MP)

Rule-based algorithm that selects traffic light phases by minimizing queue pressure differences between incoming and outgoing lanes. It is based on real-time queue observations without any learning component

Self-Organizing Traffic Light Control (SOTL)

Decentralized heuristic algorithm where each traffic light adjusts autonomously based on local traffic density, following predefined rules rather than optimization through learning

Comparing with other Algorithms

KPI	REINFORCE vs DQN	REINFORCE vs MP	REINFORCE vs SOTL
Waiting Time	20.3% Worse	30.3% Worse	4.6% Better
Delay	17.4% Worse	8.9% Better	31.4% Better
Queue Length	3.6% Worse	5.9% Better	19.9% Better
Volume	15.5% Better	7.9% Worse	27.7% Better

Comparative performance discussion

- **REINFORCE vs DQN:**
  - 20.3% worse in waiting time
  - 15.5% better in volume
  - Slightly worse on delay and queue length
- **REINFORCE vs MaxPressure (MP):**
  - 8.9% better on delay
  - Better queue length, worse waiting time and volume
- **REINFORCE vs SOTL:**
  - Outperforms on all KPIs
  - 31.4% better in delay
  - Slightly less efficient in volume

Conclusion:

- Strong overall compromise
- Especially effective on delay and queue reduction
- Stability remains an issue

Is REINFORCE efficient for traffic control ?

Strengths

- Simple to implement for traffic intersections — directly optimizes signal phase selection without value estimation
- Supports stochastic policies $\Rightarrow$ promotes diverse phase patterns and adaptive signal behavior
- Naturally responsive to dynamic and uncertain traffic conditions, e.g., varying flows and random arrivals

Limitations

- High performance variance between episodes $\Rightarrow$ unpredictable signal timings across different simulations
- Slow learning due to delayed policy updates after full episodes of traffic simulation
- Difficult to stabilize in complex networks with multiple intersections or varying densities

Further improvements

- **Use of Baselines:**
  - Reduce variance without introducing bias
  - Commonly use state-value function as baseline
- **Advantage Function:**
  - Focus learning on actions better than average
  - Speeds up convergence
- **Actor-Critic Methods:**
  - Actor chooses actions, critic evaluates them
  - Combines benefits of policy and value learning
- **Entropy Regularization:**
  - Encourages continued exploration
  - Prevents premature convergence to suboptimal policies

Thank you

We would like to express our sincere gratitude to Nadir Farhi and Zoi Christoforou, researchers at the CERMICS laboratory of École des Ponts et Chaussées, for their invaluable support and guidance throughout this project.

We also extend our thanks to Romain Ducrocq for his prior work on implementing a DQN agent within a SUMO environment. His project provided us with valuable insights that greatly informed the development of our own REINFORCE-based environment.

References (1)

- R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 2018 (accessed February 3, 2025).
- R. Ducrocq, *Deep Reinforcement Q-Learning for Intelligent Traffic Signal Control with Partial Detection*, Research Internship Report, IFSTTAR-COSYS-GRETTIA, Université Gustave Eiffel, 2021 (accessed February 3, 2025).
- R. Ducrocq, *DQN-ITSCwPD GitHub Repository*, [online]. Available at: `https://github.com/romainducrocq/DQN-ITSCwPD` (accessed February 3, 2025).
- E. Vinitsky, K. Parvate, A. Kreidieh, C. Wu, and A. Bayen, "Grid Formation of Traffic and Autonomous Vehicles in Mixed Autonomy," *HAL-Inria*, 2017. [online]. Available at: `https://inria.hal.science/hal-01656255v1/document` (accessed April 2, 2025).

References (2)

- M. Mujahed, "Policy Gradient Methods (Reinforcement Learning Lecture)," YouTube, 2022. [online video]. Available at: https://www.youtube.com/watch?v=8JVRbHAVCws (accessed January 23, 2025).
- I. Sofeikov, "REINFORCE Algorithm: Reinforcement Learning from Scratch in PyTorch," Medium, 2021. [online]. Available at: https://medium.com/@sofeikov/reinforce-algorithm-reinforcement-learning-from-scratch-in-pytorch-41fc (accessed February 10, 2025).
- M. Cholodovskis, "The True Impact of Baselines in Policy Gradient Methods," Medium, 2022. [online]. Available at: https://medium.com/@marlos.cholodovskis/the-true-impact-of-baselines-in-policy-gradient-methods-aa4cb50c4f8c (accessed May 12, 2025).