RL Exam
Smart traffic lights automation via RL

Soccol Marco, Zinna Isacco

University of Trieste

November 21, 2022

Problem statement

The aim of the project is:

to improve the traffic flow on a simple intersection
regulated by a *smart* traffic light.

Problem statement

The aim of the project is:

to improve the traffic flow on a simple intersection
regulated by a *smart* traffic light.

The environment is composed by:

- A traffic light (controlled by the agent)
- The vehicles of the simulation generated randomly via some distributions.
- Traffic lanes observed by the agent

Metrics:

- **total waiting time** of the vehicles present at the intersection
- **mean speed** of the traffic flow

SUMO (Simulation of Urban MObility)
A traffic simulation environment

- Open source package designed to perform traffic simulations
- Continuous traffic simulations
- Fully customizable scenarios (roads, intersections, traffic lights, vehicles...)
- Lots of metrics for performance evaluation
- *De facto* standard for traffic simulations (see [2])

- SUMO-RL library (see [1]) as reinforcement learning interface

States

States are **continuous** and depend on the number of lanes in the environment. Formally:

$$s = (\texttt{phase}, \texttt{min-green}, [\texttt{d}_1, \ldots, \texttt{d}_n], [\texttt{q}_1, \ldots, \texttt{q}_n])$$

where:

- **phase**: one-hot encoding of traffic light phases
- **min-green**: binary variable indicating whether the minimum seconds of green for the current phase have already passed
- $\mathbf{d}_i$: density inside lane i, the number of incoming vehicles divided by the total lane capacity
- $\mathbf{q}_i$: queue density in i lane, the number of stopped vehicles in the lane divided by total capacity

Actions

The action space is **discrete**
each action corresponds to one of the the possible *green-phases*

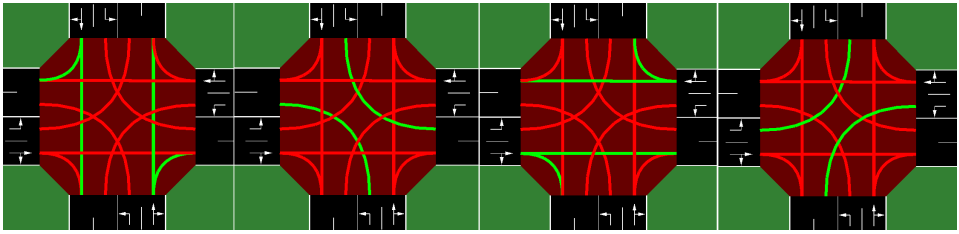


Figure: In this intersection **there are 4 actions available,** corresponding to the 4 green phases shown in the figure.

Every time a change of phase occurs then the next green phase is preceded by the corresponding yellow phase.

Rewards

The reward function, at time t, is given by the variation of the cumulative vehicle waiting time:

$$r_t = \left(\sum_j w_j^{(t-1)} \right) - \left(\sum_j w_j^{(t)} \right) = W^{t-1} - W^t$$

where the sums are taken over all the vehicles that are present at the intersection at time step t.

In other words, the reward is how much the total waiting time changed in relation to the previous time-step.

Algorithms

Model free problem with **continuous states**

Discretization

- SARSA
- Q-learning

Approximation

- SARSA-ANN
- Deep Q-learn

SARSA and Q-learning

- **State-space has been discretized** in order to manage the continuous traffic density information
- Q-table with 10 bins for each continuous variable

Size of the Q-table increases with the number of incoming lanes!

$$|Q\text{-table}| = (\#\text{ of phases}) \cdot 2 \cdot (\#\text{ of bins})^2 \cdot (\#\text{ of lanes})^2$$

In this project:

First scenario Four incoming lanes $\Rightarrow |Q\text{-table}| = 6400$

Second scenario Eight incoming lanes $\Rightarrow |Q\text{-table}| = 51200$

SARSA algorithm - an overview (see [4])

Algorithm 1: SARSA

Input: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$;

initialize $Q(s, a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$;
for each *episode* **do**
    initialize S ;
    choose A from S using policy derived from Q (e.g., ε-greedy) ;
    for each *step of episode* **do**
        take action A, observe R, S' ;
        choose A' from S' using policy derived from Q (e.g., ε-greedy) ;
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$;
        $S, A \leftarrow S', A'$;
    end for
    until S is terminal ;
end for

Q-learning algorithm - an overview (see [4])

Algorithm 2: Q-learning

Input: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$;

initialize $Q(s, a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$;

for each *episode* **do**

  initialize S ;

  for each *step of episode* **do**

    choose A from S using policy derived from Q (e.g., ε-greedy) ;

    take action A, observe R, S' ;

    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$;

    $S \leftarrow S'$

  end for

end for

SARSA ANN

Instead of a Q-table there is a **function approximation** given by a **neural network**.

Given a present state s, **for each action** is implemented a neural network that gives back a q-value approximation

$$q(s, a) \simeq \mathrm{ANN}_a(s), \quad \forall a \in A.$$

The next action is then chosen according to an ε-greedy algorithm using the $q(s, a)$ calculated.

The Neural Networks are updated using only the rewards generated by the corresponding actions.

Deep Q-Learning

A deep reinforcement learning method that **uses a neural network to approximate the policy function** $\pi(a|s)$.
Based on a Q-learning algorithm, it uses an ANN instead of a Q-table.

The algorithm exploits some techniques to improve and stabilize the learning process:

- ε-greedy policy
- Target Network + Model Network
- Experience replay

DQN
Model Network + Target Network

- A 2-steps update procedure using two coupled neural networks to approximate the q-values
- The **Model Network** is the function approximation for the q-values
  - It is updated every four simulation-steps
  - There could be a **big variation of the parameters** of the network; this would lead to **a very unstable learning process**
- The **Target Network** is a temporary copy of the MN updated every 100 simulation-steps
  - It is used to compute the q-values for the update of the MN
  - Its parameters have **smaller variation during training** so using it to compute q-values to update the MN should results in **a more stable learning process**

Experience replay

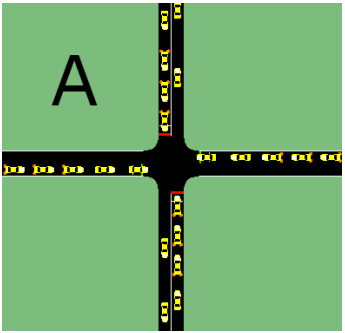
1. Store the agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$
2. Sample mini-batches of experiences from memory to train the model

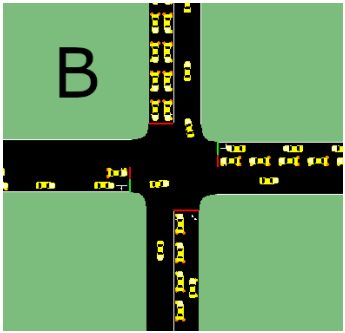
---

- It reduces auto-correlation phenomena
- More general batches for learning
- More efficient use of previous experience

Our environments scenarios

We simulated an intersection between a main street (horizontal) and a secondary one (vertical).





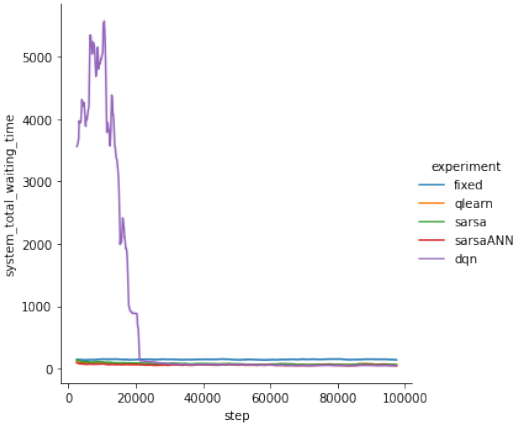
Case *A*:

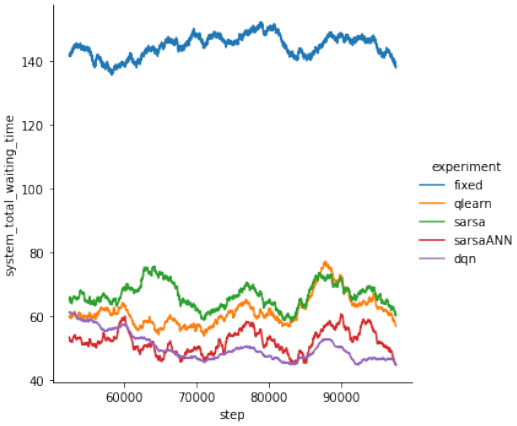
- most simple situation
- 4 different incoming lanes with only one possible trajectory for each (going straight across the intersection)

Case *B*:

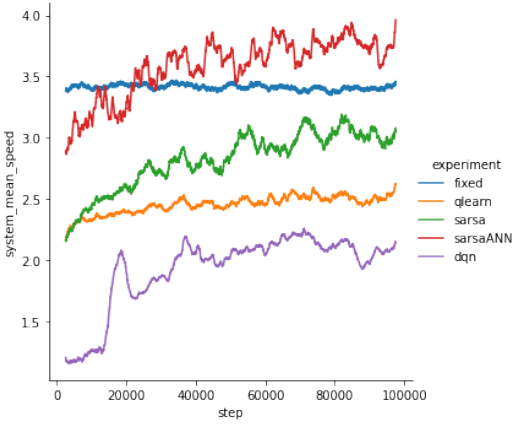
- a bit more complex situation
- 8 different incoming lanes with different trajectories

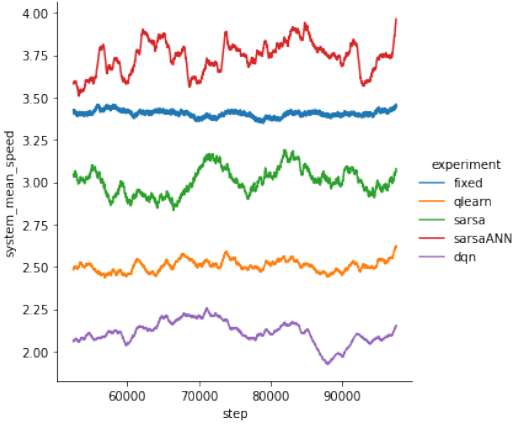
Case A - Total waiting time



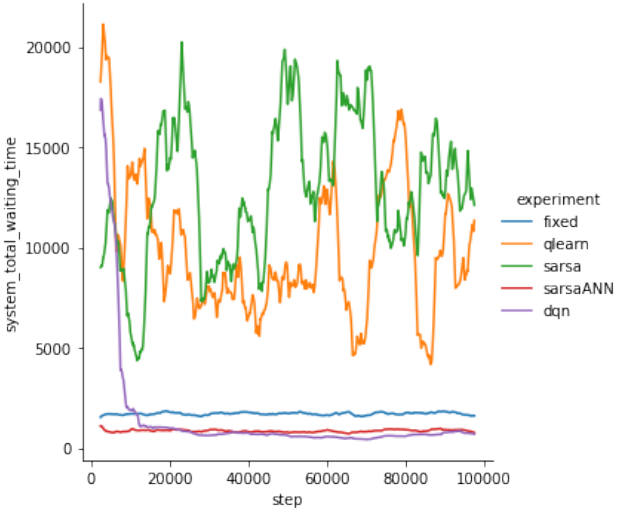


Case A - Mean speed



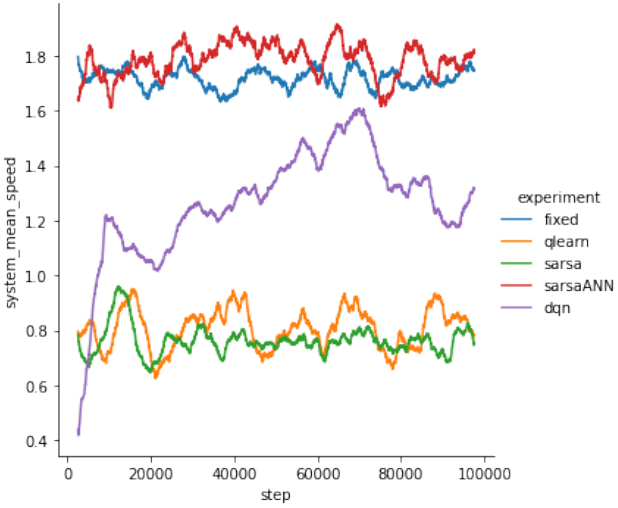


Case B - Total waiting time



- Poor performance for Q-learn and SARSA
- DQN starts with bad performance but rapidly converges to an optimal solution
- SarsaANN converges almost instantly to the optimal solution

Case B - Mean speed

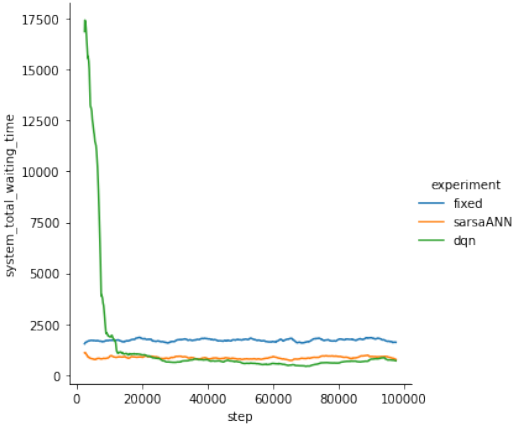


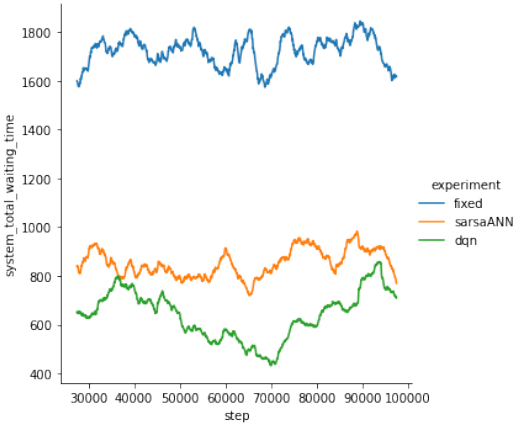
Our reward function evaluates only the waiting time.

The models therefore are not able to converge to an high speed solution.

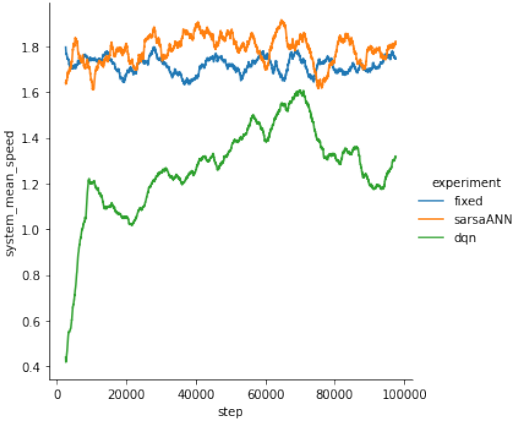
SarsaANN reaches good performances for both the metrics we analyzed.

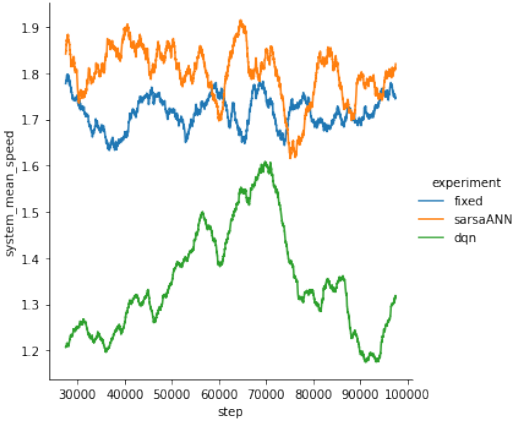
Case B - Focus on SarsaANN / DQN - Time





Case B - Focus on SarsaANN / DQN - Speed





Conclusions

- The use of RL techniques can lead to a considerable improvement of traffic. This should help to reduce the pollution caused by cars and other vehicles.
- **SARSA** and **Q-learning** have good performances only on very simple environments. More complex scenarios generate too much states (generated after the discretization) and these algorithms are not able to find good policies.
- **SarsaANN** has the better performances in terms of reward, metrics evaluation and time of convergence of the model.
- **DQN** has the better performance in *waiting time* but its solution obtains the slowest *mean speed* of vehicles. It is also slower than SarsaANN to converge.

Vulnerabilities / Issues

- A situation of the type *full-empty* on the two directions of the same line will be penalized respect to a *full-full* traffic situation on the other line.
- The implementation of this solutions on a real intersection could have drawbacks. **The traffic light policies obtained via RL could be really unpredictable for drivers.** The fast change in semaphore phases and the absence of periodicity will lead to accidents.
  It seems that SarsaANN model prefers longer traffic light phases.
  This suggests that SarsaANN would be preferable to use in a human environment.

Future improvements

- Explore further **parameters tuning** for all the models especially focusing on neural network architectures.
- **Try different reward functions** that combines together a broader set of metrics, such as mean velocity or pollution/CO_2 emissions.
- **Work on the scalability** of the models: more precisely try to deal with more complex single intersection or to apply the models on a street network with more than one intersection, implementing a multi-agent system.
- **Robustness:** integrate in the simulations traffic anomalies (crashes, bad drivers,...) and broken or malfunctioning sensors.

Thank you for the attention!

References

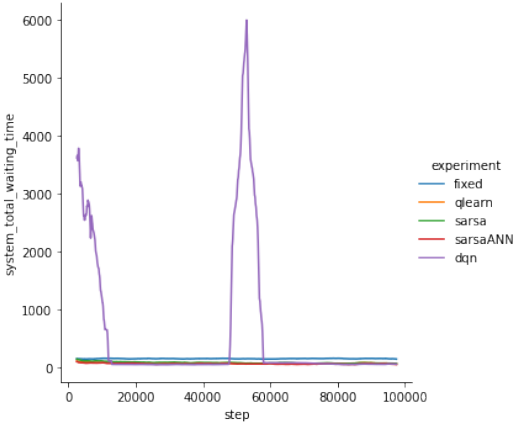
[1] L. N. Alegre.
SUMO-RL.
https://github.com/LucasAlegre/sumo-rl, 2019.

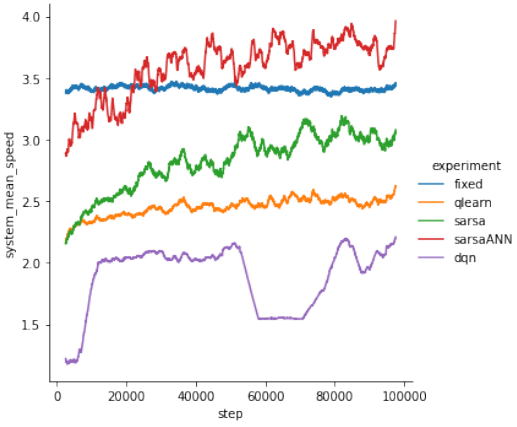
[2] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner.
Microscopic traffic simulation using sumo.
In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller.
Playing atari with deep reinforcement learning, 2013.

[4] R. S. Sutton and A. G. Barto.
Reinforcement Learning: An Introduction.
The MIT Press, second edition, 2018.

Case A - A strange behaviour





Trying to deal with a more complex scenario