UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

https://www.snap4City.org SNAP4CITY KM4CITY

NeuroSymbolic Artificial Intelligence at Scale

Paolo Nesi, paolo.nesi@unifi.it

Marco Fanfani, marco.fanfani@unifi.it

https://www.disit.org/

Parte: 4 (2025-26)

TOP

Neuro-Symbolic Artificial Intelligence 16/06/2026

P4: Deep Reinforced Learning and Symbolic at Scale

- multi agent Deep reinforced learning
- RL and simulation

Corso: **Neuro-Symbolic Artificial Intelligence at Scale**
P4: Deep Reinforced Learning and Symbolic at Scale
multi agent Deep reinforced learning
RL and simulation

[Link: Neuro-Symbolic Artificial Intelligence at Scale | Disit]

Enrico Collini
enrico.collini@unifi.it

Enrico Collini
University of Florence
Email verificata su unifi.it
Information Engineering

SEGUI

TITOLO	CITATA DA	ANNO
Predicting and understanding landslide events with explainable AI E Collini, LAI Palesi, P Nesi, G Pantaleo, N Nocentini, A Rosi IEEE Access 10, 31175-31189	82	2022
Deep learning for short-term prediction of available bikes on bike-sharing stations E Collini, P Nesi, G Pantaleo IEEE Access 9, 124337-124347	49	2021
Short-term prediction of city traffic flow via convolutional deep learning S Bilotta, E Collini, P Nesi, G Pantaleo IEEE Access 10, 113086-113099	40	2022
Data sources and models for integrated mobility and transport solutions P Bellini, S Bilotta, E Collini, M Fanfani, P Nesi Sensors 24 (2), 441	30	2024
Flexible thermal camera solution for Smart city people detection and counting E Collini, LAI Palesi, P Nesi, G Pantaleo, W Zhao	29	2024

TESTO CONSIGLIATO

Reinforcement Learning: An Introduction di Sutton e Barto (https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf)

Lecture Objectives

By the end of this course, you will be equipped to design, analyze, and implement reinforcement learning systems.

1

The RL Framework

Understand the agent–environment interaction loop and how reward signals drive learning

2

Value Functions

Study Bellman equations and how they underpin every major RL algorithm

3

Fundamental Algorithms

Analyze Dynamic Programming, Monte Carlo Methods, and **Temporal Difference Learning**

4

Deep RL

Combine deep neural networks with RL to tackle large-scale, real-world problems

Lecture Structure

PART I

Theoretical Foundations

Building rigorous mathematical intuition from the ground up.

- Markov Decision Processes
- Value functions & Bellman equations
- Dynamic Programming
- Monte Carlo Methods

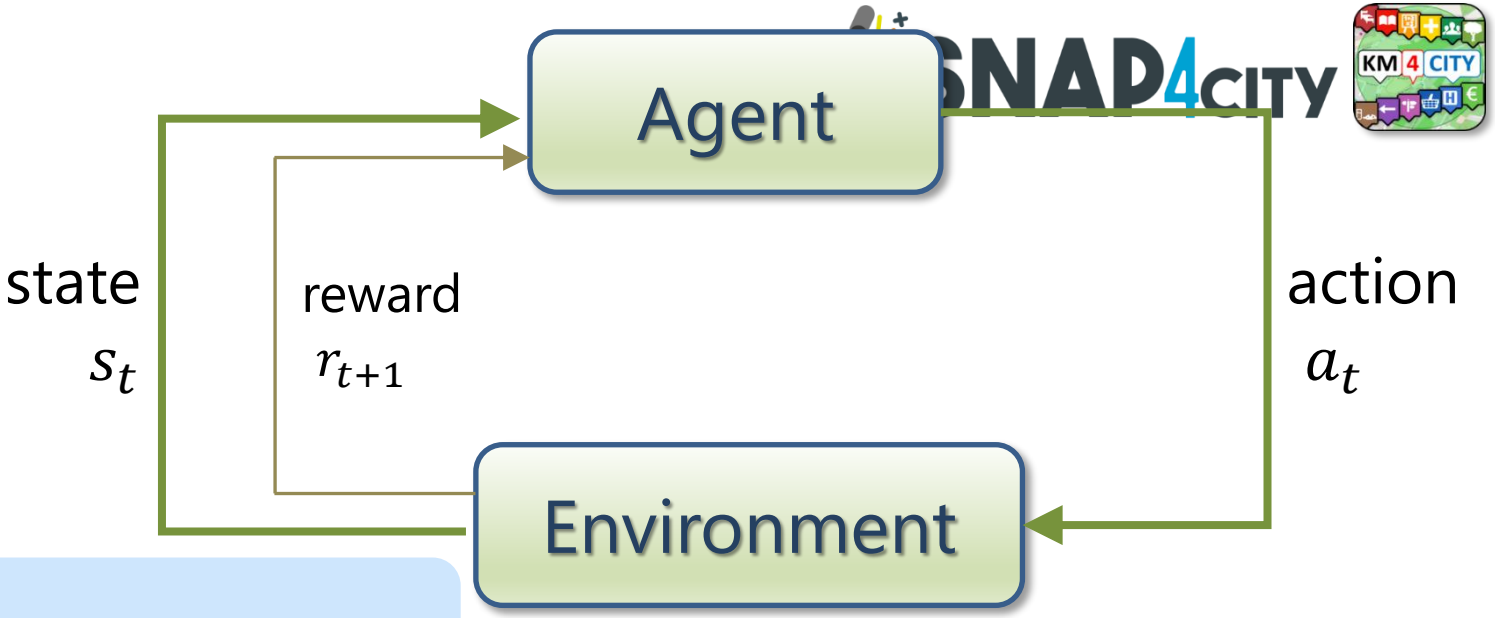
PART II

Deep Reinforcement Learning

Scaling RL to complex, high-dimensional environments.

- Neural value function approximation
- Temporal Difference Learning
- Deep Q-Networks (DQN)
- Modern applications & case studies

- The policy is the function that fully defines how the agent behaves. It maps every state to an action
- Reward signals the quality of the actions just taken
- The value of a state is the total expected reward the agent can accumulate in the future



Model-Based

The agent builds or uses a model to **simulate future scenarios** and plan ahead before acting. Example: AlphaGo combines learning with model-based planning.

Model-Free

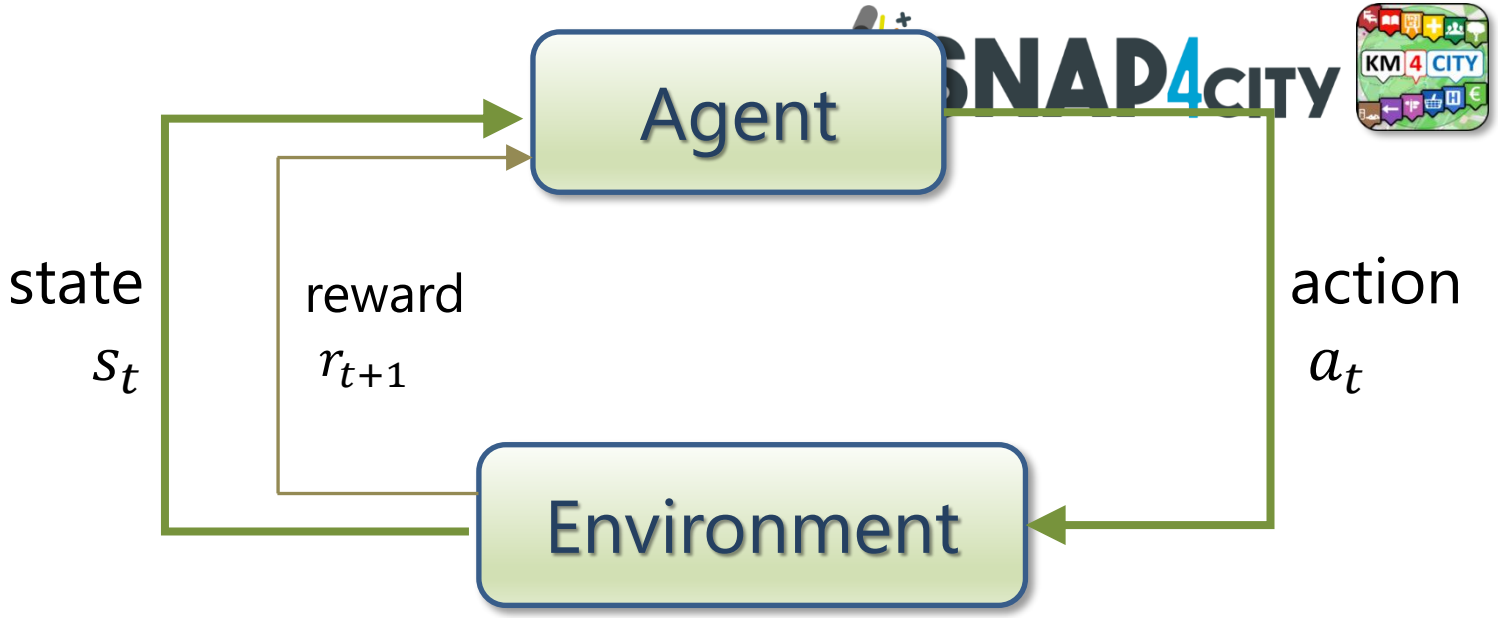
No model is used. The agent learns directly from **trial-and-error experience**. Simpler and widely used. Examples: Q-learning, SARSA, TD learning.

Temporal Difference Learning Methodologies

Objective: optimize cumulative reward

At each timestep, the agent receives a reward signal. The goal is not to maximize any single reward, but the **sum of all future rewards** — the return.

- Find the optimal policy that maximizes the
- expected Reward from every state
- Optimal state value: maximum expected cumulative reward from state *s*, following the best possible policy



Episodes

finite sequences of interactions that end in a **terminal state**.

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-t-1} R_T$$

Continuing Taks

Some problems have **no natural endpoint**. The agent interacts with the environment indefinitely.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$$

Optimal Action Value $Q^*(s,a)$
The expected return when taking action *a* in state *s* and thereafter following π^*

Temporal Difference Learning Methodologies

→ aim to estimate Q^* directly → Once Q^* is known, the optimal policy is simply:

$$\pi^*(s) = \arg\max_a q^*(s,a)$$

- Find the optimal policy that maximizes the
- expected Reward from every state
- Optimal state value: maximum expected cumulative reward from state *s*, following the best possible policy

Learning Process

When an agent is in a particular state, it makes a prediction about the value of that state.

The prediction is performed via a value function v_π for a given policy π

Recall DP MC and TD as well use some variation of the Generalized Policy Iteration (**GPI**)

policy evaluation
↓
policy improvement
↓
new policy
↓
evaluation

Make the policy ϵ **greedy** with respect to the evaluation: now that I know better how much states are worth, I always choose the action that takes me to the best place (with $p = 1 - \epsilon$)

Exploration vs Exploitation

the agent takes an action, receives a reward and moves to the next state. It then makes a new prediction about the value of the next state. $v^\pi(s)$ $q^\pi(s,a)$

Bellman expectation equations

DP	$v^\pi(s) \leftarrow \sum_a \pi(a\|s) \sum_a P(s'\|s,a)[R + \gamma V(s')]$
MC	$v^\pi(s) \leftarrow V(s) + \alpha[G_t - V(s)]$
TD	$q^\pi(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s'.a') - Q(s,a)]$

TEMPORAL DIFFERNECE LEARNING

Temporal Difference (TD) learning is a type of reinforcement learning method used for predicting future events or optimizing decision-making in uncertain environments. It combines ideas from two other methods: Monte Carlo methods and Dynamic Programming.

- The primary goal of TD learning is to estimate the value of a state (or action) in a given environment. The value represents how good it is to be in a state, considering future rewards.
- TD learning involves updating value estimates based on the difference or "error" between predicted rewards and actually observed rewards over time. This difference is called temporal difference error.

TD error δ_t

The difference between what we believed a state was worth and what we discovered after just one step.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[\underbrace{r + \gamma \max_{a'} Q(s'.a')}_{TD\ target} - Q(s,a)]$$

$$\delta_t$$

(choosing the action-value with best expected reward)

SARSA (On-policy TD Control)

The main function for updating the Q-value depends on

the current state of the agent "St",

the action the agent chooses "At",

the reward "Rt+1" the agent gets for choosing this action,

the state "St+1" that the agent enters after taking that action,

and the next action "At+1" the agent chooses in its new state.

The acronym for the quintuple (**S**t, **A**t, **R**t, **S**t+1, **A**t+1) is SARSA.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[\underbrace{r + \boldsymbol{\gamma Q}(\boldsymbol{s'}, \boldsymbol{a'})}_{TD\ target} - Q(s,a)]$$

δ_t

SARSA (On-policy TD Control)

```
def sarsa(env, episodes, alpha, gamma, epsilon):
    # Q-table: (righe, colonne, azioni)
    Q = np.zeros((env.height, env.width, 4))

    for episode in range(episodes):
        state = env.reset()
        action = epsilon_greedy_policy(Q, state, epsilon)
        done = False

        while not done:
            next_state, reward, done = env.step(action)
            next_action = epsilon_greedy_policy(Q, next_state, epsilon)

            x, y = state
            nx, ny = next_state

            # update SARSA
            td_target = reward + gamma * Q[nx, ny, next_action]
            td_error = td_target - Q[x, y, action]

            Q[x, y, action] += alpha * td_error

            state = next_state
            action = next_action

    return Q
```

$$Q(s,a) \leftarrow Q(s,a) + \alpha[\underbrace{r + \gamma Q(s',a')}_{TD\ target} - Q(s,a)]$$

$$\delta_t$$

Q-learning (Off-policy TD Control)

$$Q(s,a) \leftarrow Q(s,a) + \alpha[\underbrace{r + \gamma \max_{a'} Q(s'.a')}_{TD\ target} - Q(s,a)]$$

$$\delta_t$$

(choosing the action-value with best expected reward)

Q-learning (Off-policy TD Control)

```
def q_learning(env, episodes, alpha, gamma, epsilon):
    # Q-table: (righe, colonne, azioni)
    Q = np.zeros((env.height, env.width, 4))

    for episode in range(episodes):
        state = env.reset()
        done = False

        while not done:
            # scegli azione con epsilon-greedy
            action = epsilon_greedy_policy(Q, state, epsilon)

            next_state, reward, done = env.step(action)

            x, y = state
            nx, ny = next_state

            # Q-learning: usa max sulle azioni future
            best_next_action_value = np.max(Q[nx, ny])

            td_target = reward + gamma * best_next_action_value
            td_error = td_target - Q[x, y, action]

            Q[x, y, action] += alpha * td_error

            state = next_state

    return Q
```

$$Q(s,a) \leftarrow Q(s,a) + \alpha[\underbrace{r + \max_{a'} Q(s'.a')}_{TD\ target} - Q(s,a)]$$

$$\delta_t$$

Example Windy Gridworld

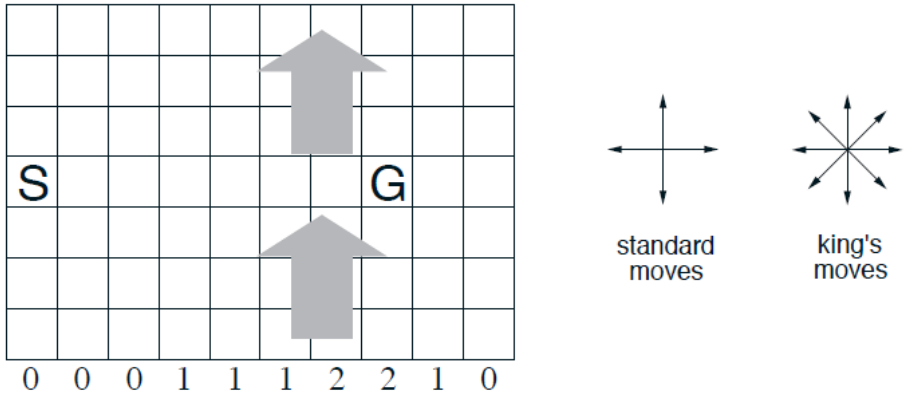
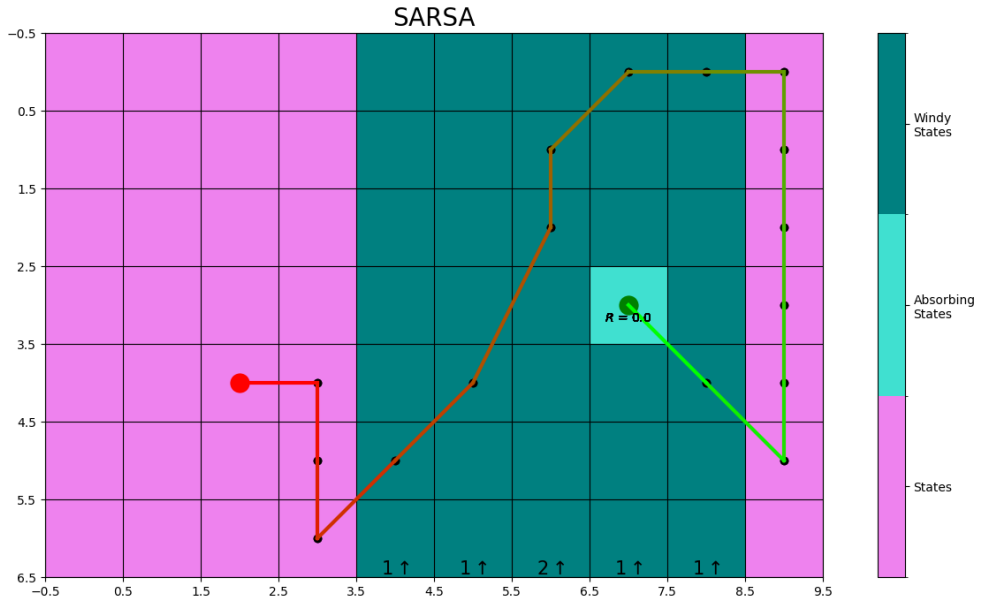


Figure 6.10: Gridworld in which movement is altered by a location-dependent, upward "wind."



The actions are the standard four|up, down,right, and left|but in the middle region the resultant next states are shifted upward by a "wind" the strength of which varies from column to column.

The strength of the wind is given below each column, in number of cells shifted upward.

Example Windy Gridworld

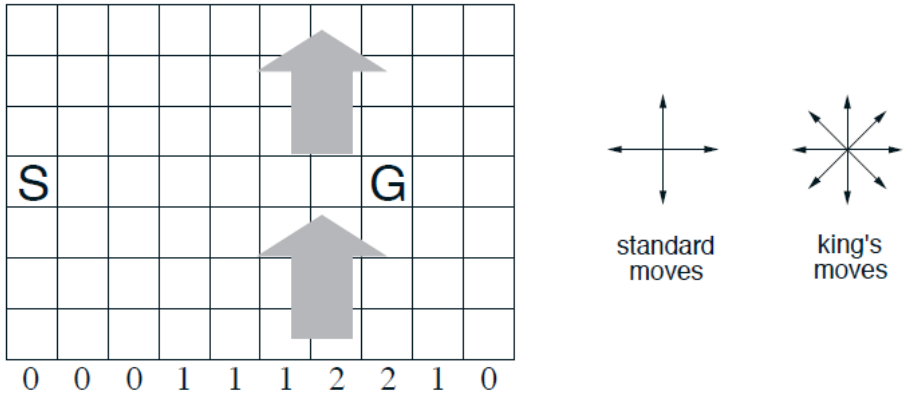
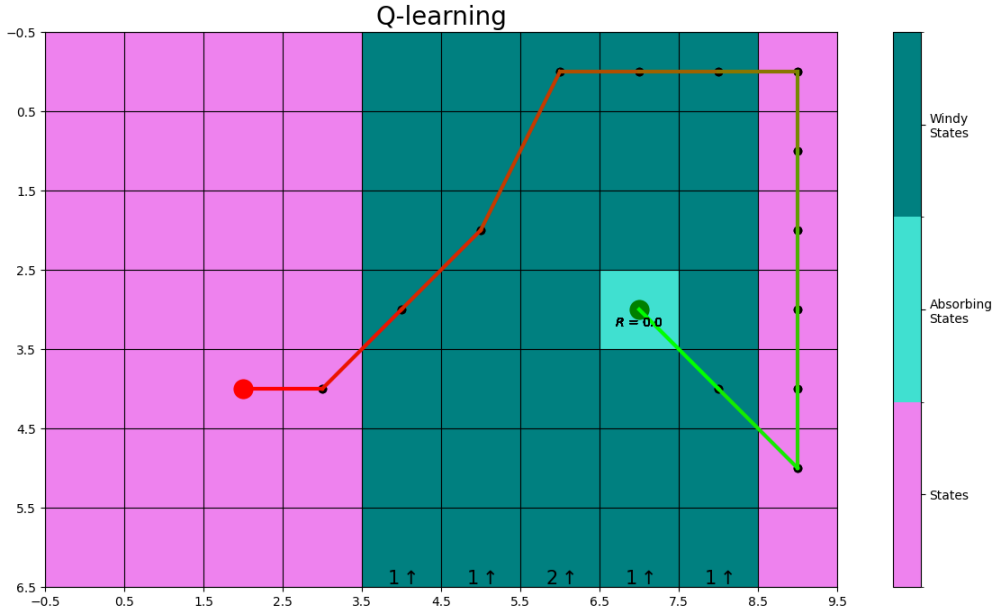
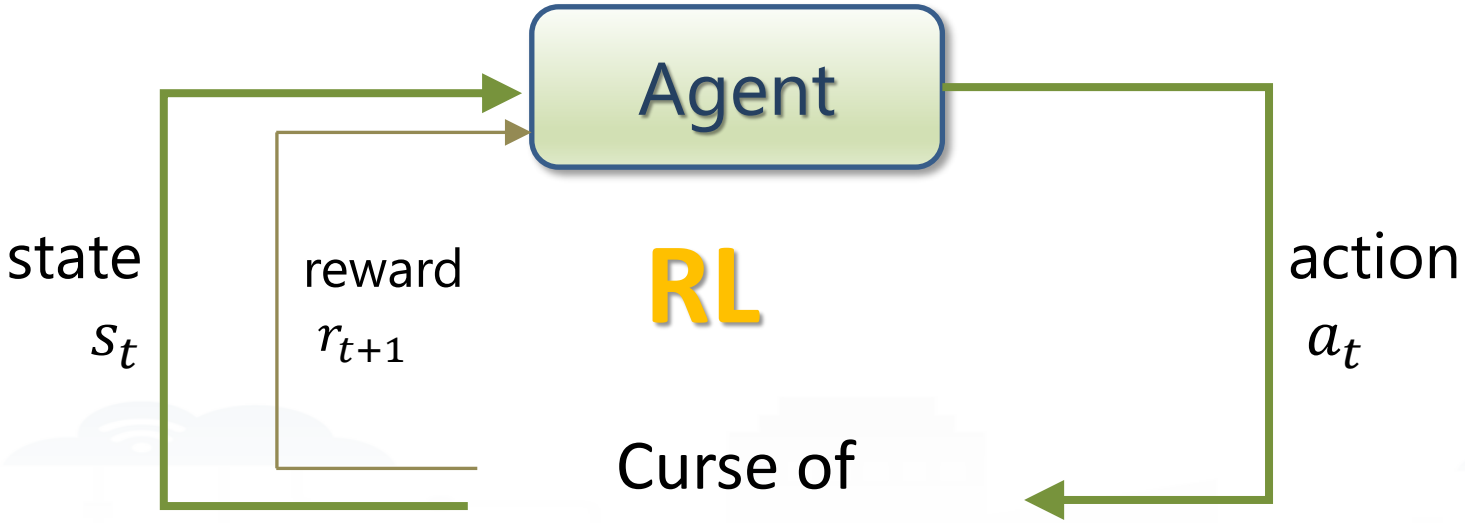


Figure 6.10: Gridworld in which movement is altered by a location-depend upward "wind."



The actions are the standard four|up, down,right, and left|but in the middle region the resultant next states are shifted upward by a "wind" the strength of which varies from column to column.

The strength of the wind is given below each column, in number of cells shifted upward.



(Model-Free)

RL

Curse of dimensionality

S	a	…
s1	Q(s1,a1)	…
Sn		…

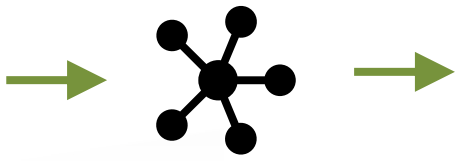
DRL

Neural Value Function Approximation

Deep Q-Network

Idea: use a neural network to estimate Q-values

Input
State (urban road graph)



output
Q action values

Deep Q-Network (DQN) is a reinforcement learning approach that combines Q-learning with deep neural networks to handle environments with large or continuous state spaces.

Instead of using a table to store Q-values for each state–action pair, DQN uses a neural network that takes the current state as input and directly **outputs the estimated Q-values for all possible actions.**

The agent then selects actions based on these values, typically choosing the one with the highest predicted reward.

By **training** the network on experiences collected during interaction with the **environment DQN learns to approximate the optimal action-value function**.

Deep Q-Learning

Q-learning tries to approximate the optimal function:

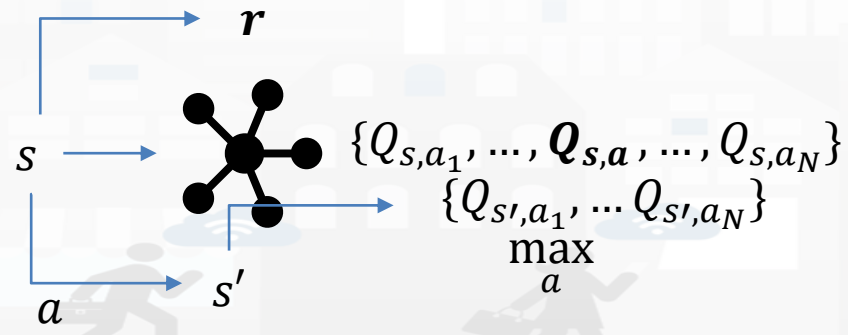
$$Q^*(s,a) = \max_{\pi} E[\sum_t \gamma^t r_t | s, a]$$

Bellman expectation equation

$$Q^*(s,a) = E[r + \gamma \max_{a'} Q^*(s', a') | s, a]$$

Q-learning update

$$Q(s,a) \leftarrow Q(s,a) + \alpha[\underbrace{r + \gamma \max_{a} Q(s', a') - Q(s,a)}_{\delta_t}]$$



$$loss = \delta(s,a) = \underbrace{r + \gamma \max_{a} Q(s', a')}_{\text{estimated } TD\ target} - \boldsymbol{Q(s,a)}$$

Experience Replay Buffer

The experience replay buffer is a memory structure used in Deep Q-Networks to store past interactions between the agent and the environment in the form of transitions (s,a,r,s')(s, a, r, s')(s,a,r,s').

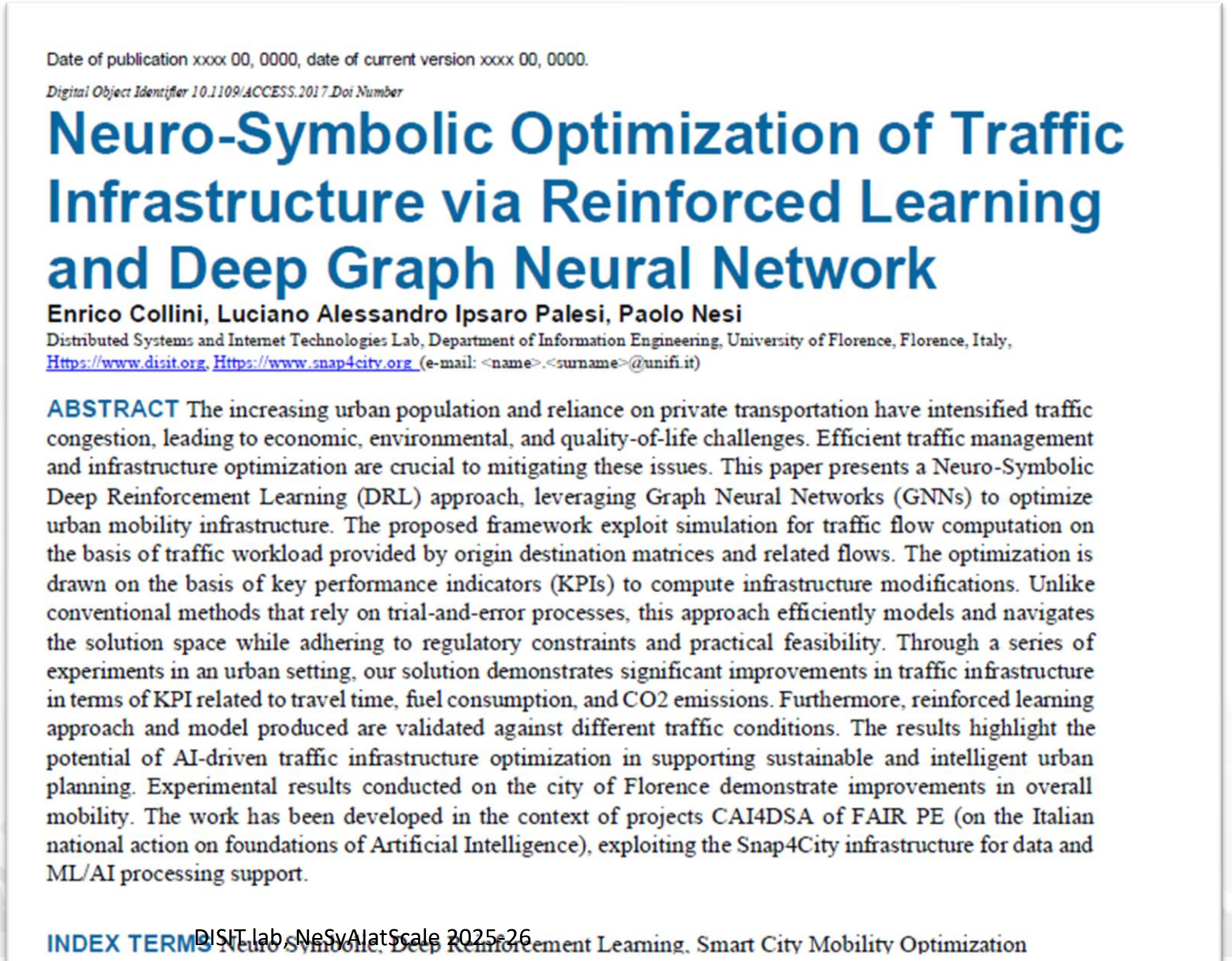
Instead of learning only from the most recent experience, the agent samples random mini-batches of past transitions from this buffer to train the neural network.

This process breaks the strong temporal correlations present in sequential data and improves the stability and efficiency of learning.

By reusing past experiences multiple times, the replay buffer also increases data efficiency and allows the model to better approximate the underlying value function.

- Next

DRL modern applications & case studies



Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Neuro-Symbolic Optimization of Traffic Infrastructure via Reinforced Learning and Deep Graph Neural Network

Enrico Collini, Luciano Alessandro Ipsaro Palesi, Paolo Nesi

Distributed Systems and Internet Technologies Lab, Department of Information Engineering, University of Florence, Florence, Italy,
Https://www.disit.org, Https://www.snap4city.org (e-mail: <name>.<surname>@unifi.it)

ABSTRACT The increasing urban population and reliance on private transportation have intensified traffic congestion, leading to economic, environmental, and quality-of-life challenges. Efficient traffic management and infrastructure optimization are crucial to mitigating these issues. This paper presents a Neuro-Symbolic Deep Reinforcement Learning (DRL) approach, leveraging Graph Neural Networks (GNNs) to optimize urban mobility infrastructure. The proposed framework exploit simulation for traffic flow computation on the basis of traffic workload provided by origin destination matrices and related flows. The optimization is drawn on the basis of key performance indicators (KPIs) to compute infrastructure modifications. Unlike conventional methods that rely on trial-and-error processes, this approach efficiently models and navigates the solution space while adhering to regulatory constraints and practical feasibility. Through a series of experiments in an urban setting, our solution demonstrates significant improvements in traffic infrastructure in terms of KPI related to travel time, fuel consumption, and CO2 emissions. Furthermore, reinforced learning approach and model produced are validated against different traffic conditions. The results highlight the potential of AI-driven traffic infrastructure optimization in supporting sustainable and intelligent urban planning. Experimental results conducted on the city of Florence demonstrate improvements in overall mobility. The work has been developed in the context of projects CAI4DSA of FAIR PE (on the Italian national action on foundations of Artificial Intelligence), exploiting the Snap4City infrastructure for data and ML/AI processing support.

INDEX TERMS Neuro-Symbolic, Deep Reinforcement Learning, Smart City Mobility Optimization