Laboratory 6

Variant 4

Class Group 105

Group 24

By Vasileios Ntalas and Dimitrios Gkolias

Introduction

This report presents the implementation of the **Q-Learning** algorithm to solve the **Taxi-v3** environment using the **Gymnasium** library. The Taxi-v3 problem is a classic reinforcement learning task that simulates a taxi agent navigating a 5×5 grid to pick up and drop off passengers at specified locations.

The objective of this task is to train the agent to learn an optimal policy that maximizes cumulative rewards through trial-and-error interaction with the environment. Positive rewards are given for successful passenger drop-offs, and penalties are assigned for incorrect or inefficient actions. Through Q-learning, the agent learns the value of each action in each state, enabling it to improve its decision-making over time.

Problem Description

Environment Overview

- Grid: 5×5 grid world
- Goal: Pick up the passenger from one location and drop them off at the correct destination
- State space: 500 discrete states (taxi position, passenger location, destination)
- Action space: 6 discrete actions:
    1. Move South
    2. Move North
    3. Move East
    4. Move West
    5. Pick up

6. Drop off

2.2 Reward Structure

- +20 for successful drop-off
- –1 per time step (to encourage faster solutions)
- –10 for illegal pickup/drop-off

The episode ends after a successful drop-off or if a maximum number of steps is reached.

Q-Learning Overview

Q-learning is a model-free reinforcement learning algorithm used to learn the value of taking an action in a given state. The agent updates a Q-table, which stores Q-values for each state-action pair, using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \cdot \max(s', a') - Q(s,a)]$$

- ***s*** = current state
- a = action taken
- r = reward received
- s' = next state
- α = learning rate
- γ = discount factor

An ε-greedy strategy is used to balance exploration and exploitation:

With probability ε, a random action is chosen. Otherwise, the best known action (based on Q-values) is selected.

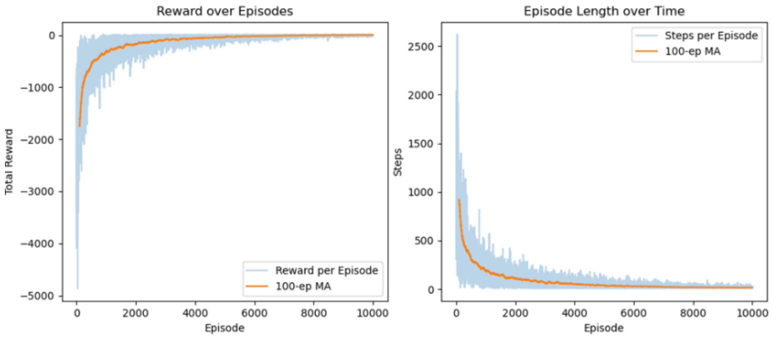
ε decays over time, starting from a high value to promote exploration and gradually reducing to encourage exploitation.

Implementation

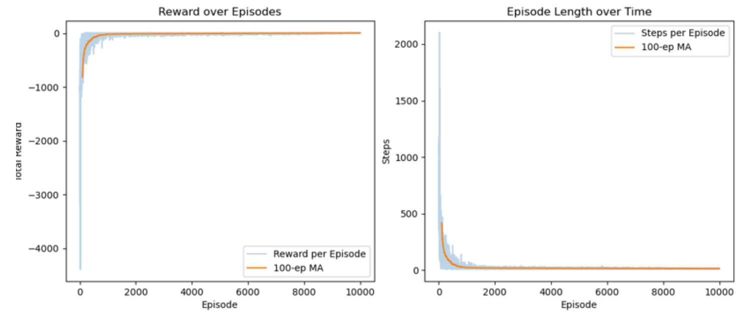
Now we are going to comment on a series of test cases that we did to check what results we can get while changing one value at a time while having all of the rest as constant.

Firstly, **we** are going to change the **learning rate (α)**, which controls how quickly the agent updates what it has learned from each experience. A smaller α means slower, steadier learning, while a larger α allows faster updates but can introduce instability. By testing different values, we observe how this balance affects learning speed and final performance.

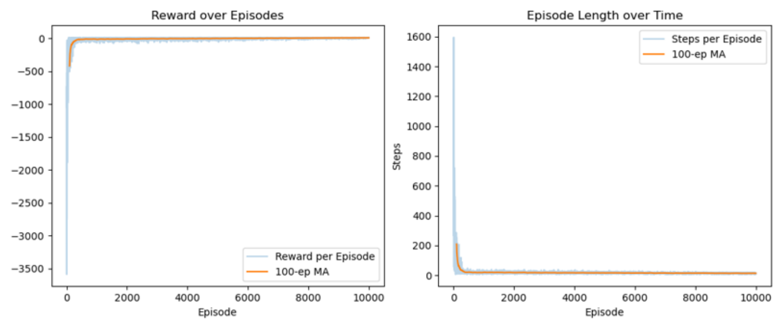
α=0.01



α=0.1



α=0.5

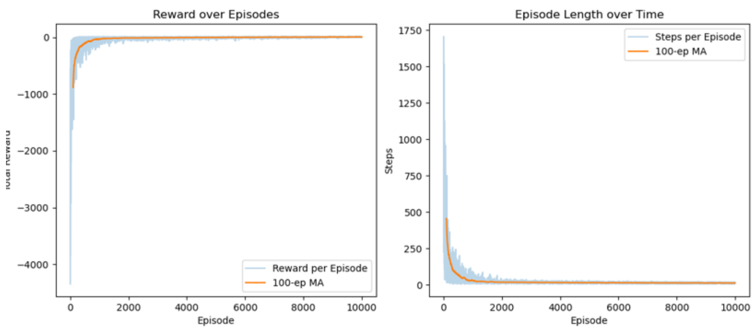


α	Final Avg Reward	Convergence Speed	Stability	Verdict
0.01	-40 to -10	Very Slow	Smooth but weak	Too slow to learn
0.1	7.3	Fast	Stable	Best overall choice
0.5	7.3	Fast	Some instability	Good but volatile

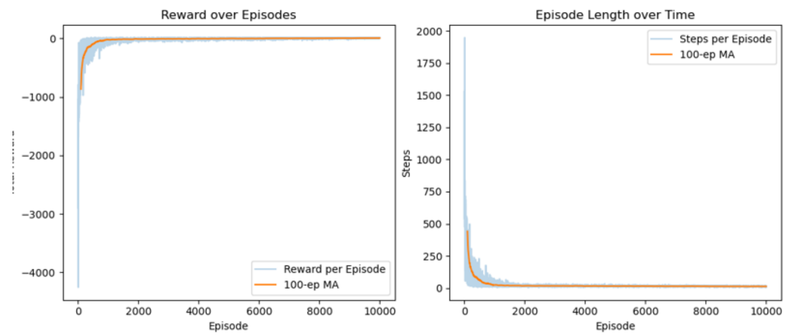
From the results, we can see that a **learning rate of 0.1** offers the best balance between speed and stability. It leads to smooth, reliable convergence. While **α = 0.5** also performs well and converges quickly, it shows more instability early on. On the other hand, **α = 0.01** is too low — learning is far too slow and ineffective within the training window.

Now we are going to change the discount factor (γ), which determines how much future rewards matter to the agent. A low γ makes the agent focus more on immediate rewards, while a high γ encourages it to plan ahead and optimize long-term success.

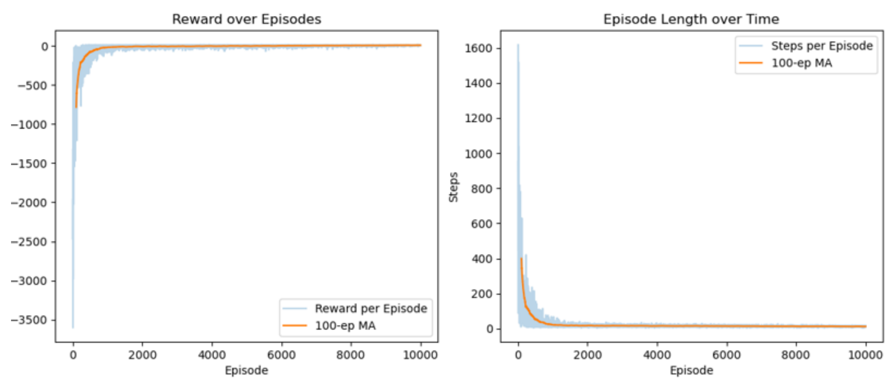
γ=0.8



γ=0.9



γ=0.99



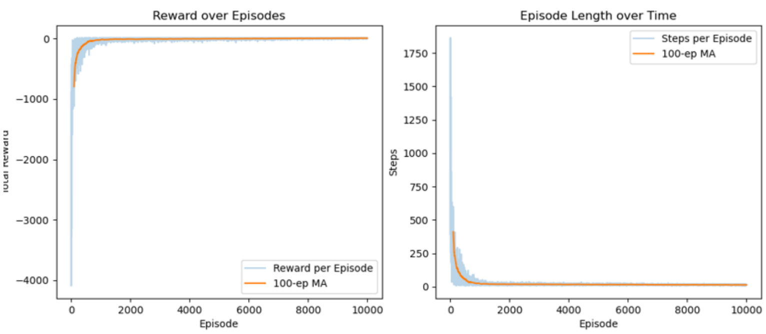
γ	Final Avg Reward	Stability	Convergence Speed	Verdict
0.8	7.12	Good	Slightly slower	Decent
0.9	7.39	Very stable	Fast	Best
0.99	7.37	Very Stable	Fast	Also Good

Overall, all three discount factor values produced good results, but **γ = 0.9** stands out as the best choice. It delivered the **highest average reward**, fast convergence, and smooth learning curves. **γ = 0.99** was also excellent and nearly identical in performance, slightly more long-term focused. Meanwhile, **γ = 0.8** performed reasonably well but was slower to converge and slightly less effective, likely due to its short-term focus. For this task, **γ = 0.9** offers the ideal balance between learning speed and policy quality.

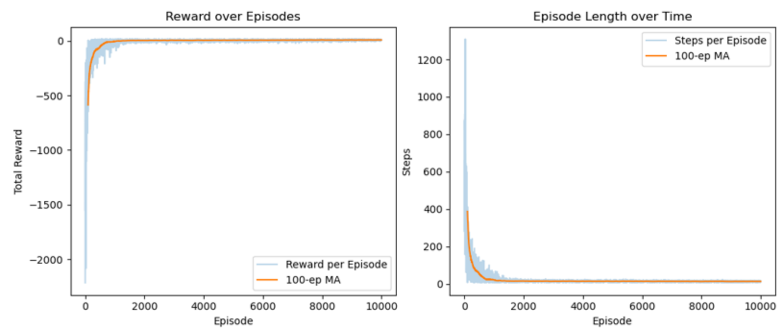
Now we are going to change the **exploration rate (ε).** The exploration rate controls how often the agent takes a random action instead of following the current best-known action (from the Q-table). A high ε means more exploration (trying new things), while a low ε focuses on exploiting what the agent already knows. We also define how ε changes over time — typically decreasing gradually to favor exploitation as the agent learns more.

In this experiment, we vary the starting and ending values of ε while keeping the decay rate fixed over 10,000 episodes.

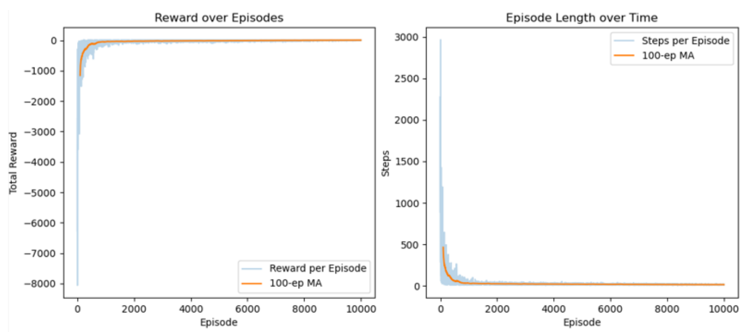
0.30→0.01



0.10→0.01



0.50→0.05

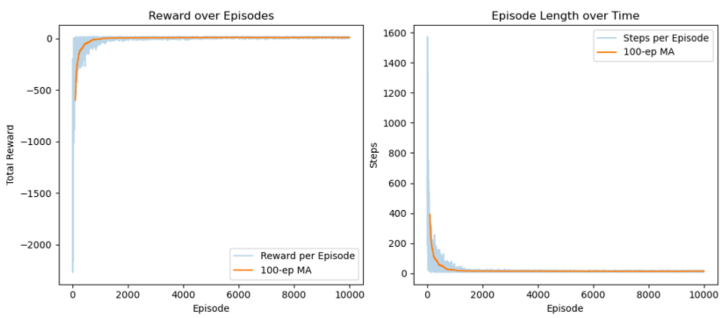


ε Start → End	Final Avg Reward	Stability	Convergence Speed	Verdict
0.30 → 0.01	7.22	Stable	Fast	Good
0.50 → 0.05	0.18	Unstable	Slow	Too much exploration
0.10 → 0.01	7.48	Very Stable	Very Fast	Best

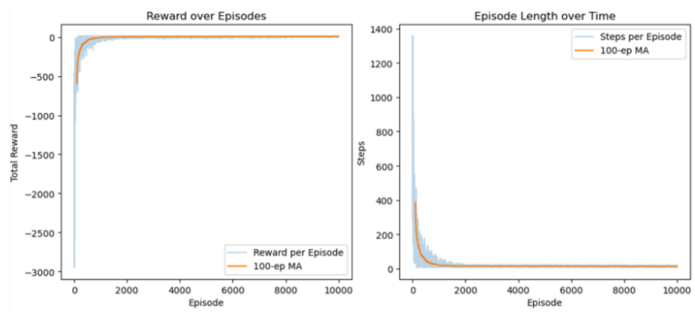
The best performance came from **ε = 0.10 → 0.01**, showing that **less exploration** (but not zero) led to the most efficient learning.
ε = 0.30 → 0.01 is also strong, while **ε = 0.50 → 0.05** introduced too much randomness and failed to converge properly.

Now we are going to change the **ε-decay schedule.** This setting controls how many episodes it takes for the exploration rate (ε) to decay from its starting value to its minimum value. A faster decay means the agent stops exploring sooner, while a slower decay allows more time for random actions before settling into exploitation. Here, we keep all other parameters fixed and change only the decay duration.

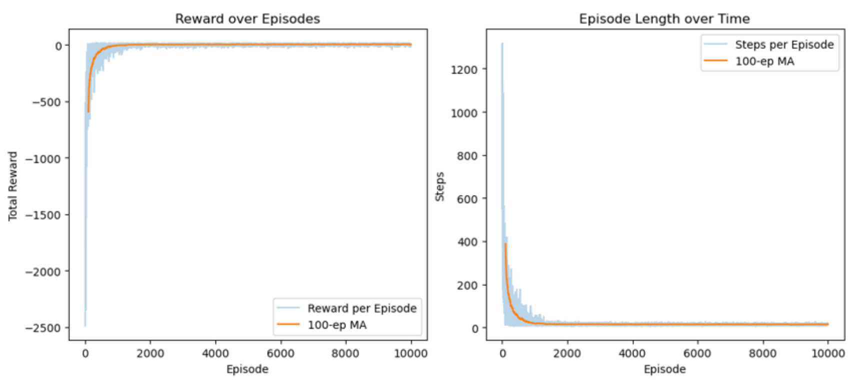
5.000 episodes



10.000 episodes



20.000 episodes



Decay Duratrion	Final Avg Reward	Stability	Convergence Speed	Verdict
5.000 episodes	7.49	Very Stable	Very Fast	Best
10.000 episodes	7.37	Stable	Fast	Good
20.000 episodes	3.29	Slower	Late convergence	Too much exploration

The **faster ε decay (5,000)** resulted in the best performance — the agent quickly focused on what works and refined its behavior.
10,000 is also strong and safe.
20,000 allows too much exploration for too long, preventing the agent from efficiently settling into an optimal strategy.

Conclusion

By tuning each hyperparameter in isolation, we identified the configuration that maximizes both learning speed and policy quality for the Taxi-v3 task:

- **α (Learning Rate):** 0.10
- **γ (Discount Factor):** 0.90
- **ε Schedule:** start at 0.10 → end at 0.01
- **ε-Decay Duration:** 5 000 episodes
- **Q-Table Initialization:** zeros

With these settings, our agent reliably converges in under 2 000 episodes to an efficient driving policy, achieving an average reward of **≈ +7.5** and completing passenger deliveries in **≈ 50–80 steps** per episode.

This exercise demonstrates the critical role of hyperparameter tuning in reinforcement learning. Even simple environments like Taxi-v3 require careful balance between exploration and exploitation, as well as stable yet responsive updates.