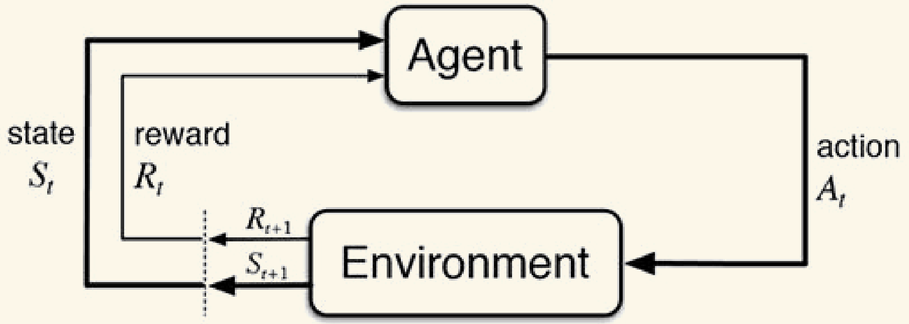
02 DECEMBER 2024

Reinforcement Learning with Q-Learning on Taxi-v3 Environment



Agenda

1. Introduction
2. Domain Selection and Setup
3. The Agent’s Goal
4. Q-Learning Overview
5. Hyperparameter Tuning
6. Training Performance
7. Challenges and Future Opportunities
8. Conclusion

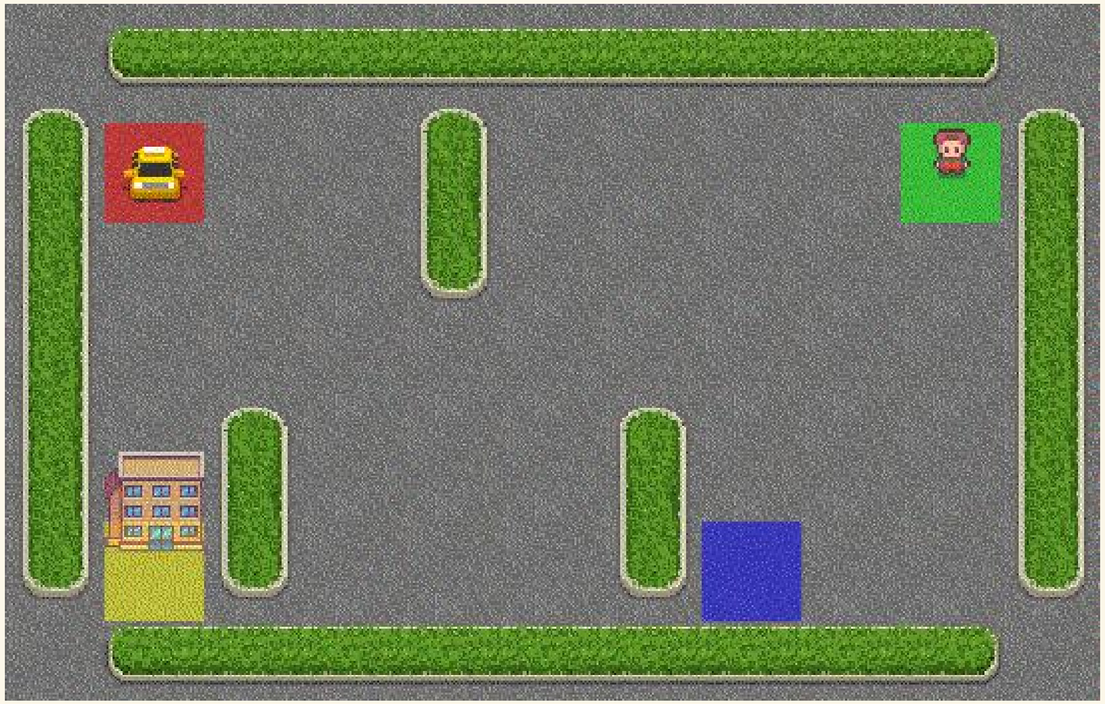
Introduction

Project Overview

Application of Q-Learning to the Taxi-v3 environment. Efficient navigation and passenger handling while minimizing penalties.

Significance

Insights into RL fundamentals and a baseline for more complex techniques.



Domain Selection and Setup

Why Taxi-v3?

Simple and structured environment ideal for understanding Q-Learning.

Environment Details:

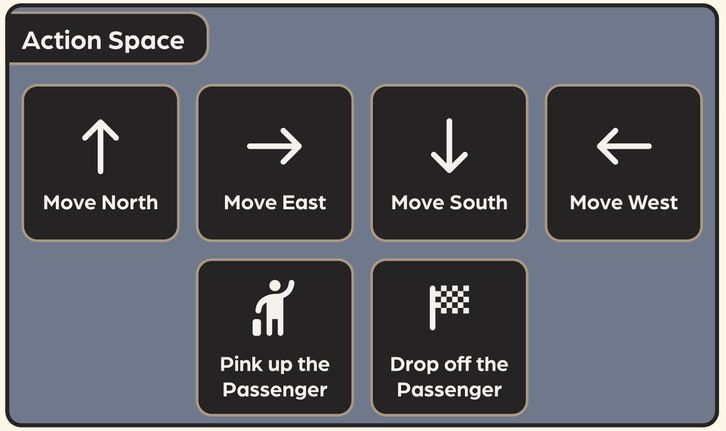
- Finite state space: 500 states.
- Deterministic actions.

Reward System:

- Positive rewards for correct actions.
- Penalties for mistakes and inefficiency.

project.py

```
Episode 500/5000, Total Reward: -109
Episode 1000/5000, Total Reward: -109
Episode 1500/5000, Total Reward: -38
Episode 2000/5000, Total Reward: 12
Episode 2500/5000, Total Reward: -18
Episode 3000/5000, Total Reward: -5
Episode 3500/5000, Total Reward: 6
Episode 4000/5000, Total Reward: -4
Episode 4500/5000, Total Reward: 10
Episode 5000/5000, Total Reward: 5
```



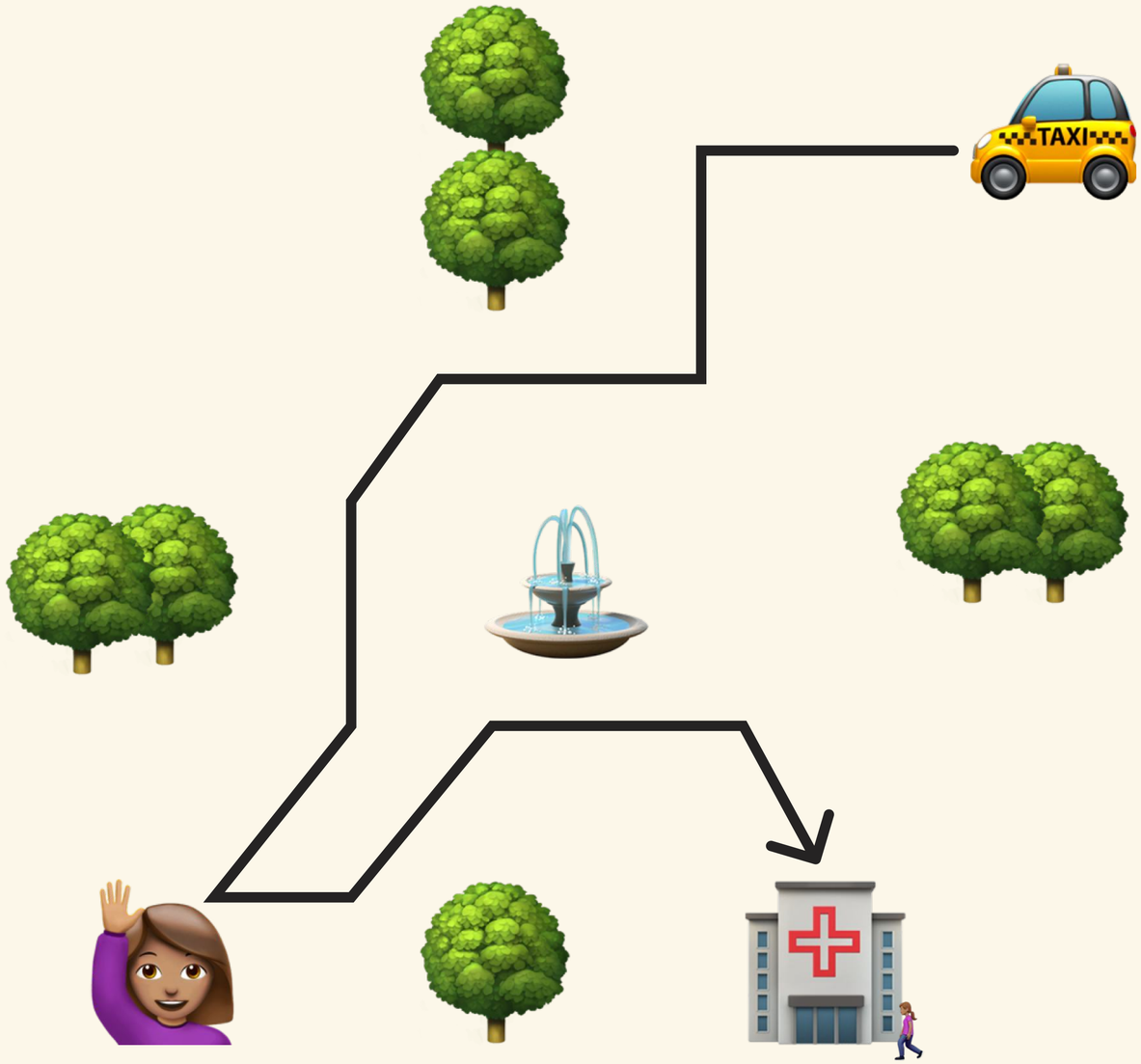
The Agent's Goal

Objective:

- Pick up passengers and drop them at correct destinations.
- Minimize penalties and steps taken.

Challenges:

- Sparse rewards requiring multiple steps for a single positive outcome.



Q-Learning Overview

Q-Table:

- Maps states to actions with expected rewards.
- Initially set to zero, updated iteratively.

Bellman Equation:

- Formula for Q-value updates.
- Key parameters: learning rate (α), reward (r), discount factor (γ).

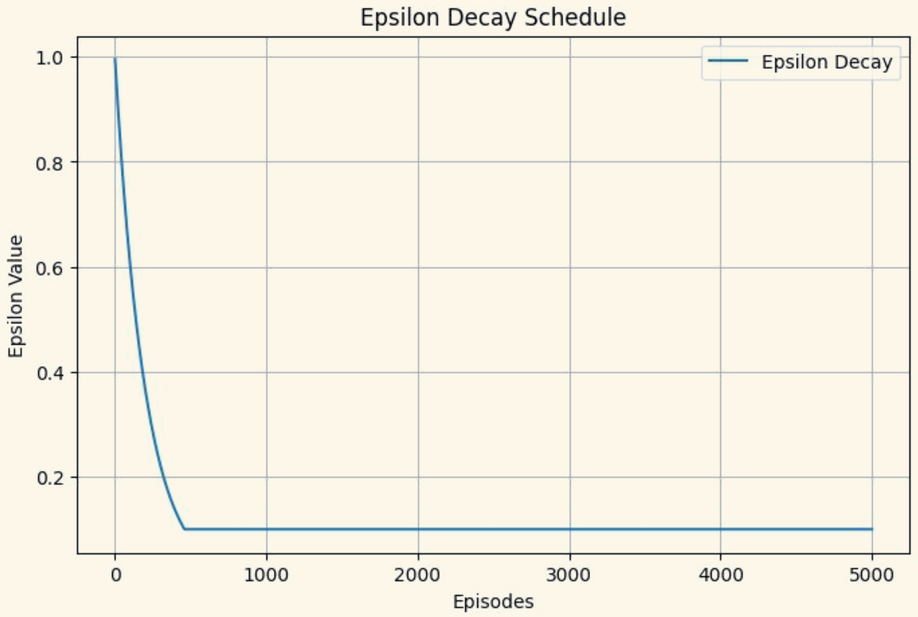
Exploration vs. Exploitation:

- ε-Greedy strategy for balancing new actions and known policies.

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

Here:

- $\boldsymbol{\alpha}$ is the learning rate, determining the weight of new information.
- $\boldsymbol{r}$ is the reward for the current action.
- $\boldsymbol{\gamma}$ $(discount\ factor)$ balances the importance of immediate versus long-term rewards.



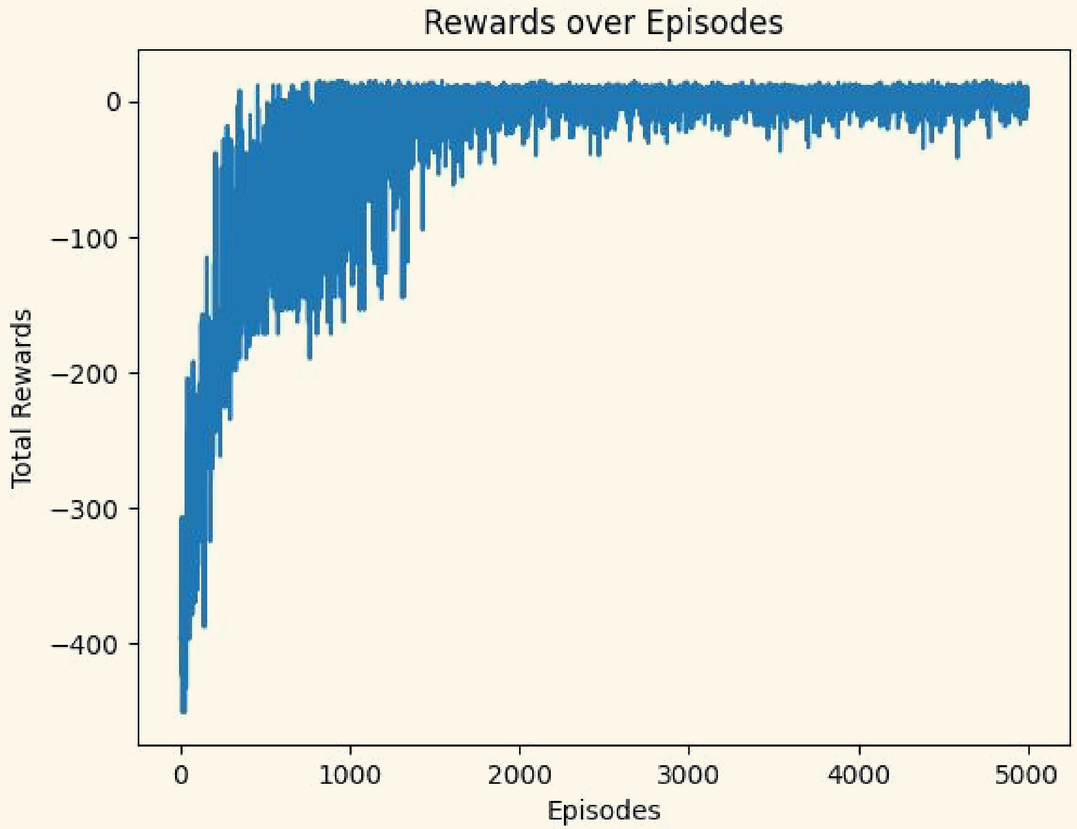
Hyperparameter Tuning

Key Parameters:

- Learning Rate (α): Balances updates and stability.
- Discount Factor (γ): Weighs immediate vs. long-term rewards.
- Exploration Strategy (ε): Linear vs. exponential decay.

Systematic Testing:

- Parameter combinations tested for optimization.



Training Performance

Cumulative Rewards:

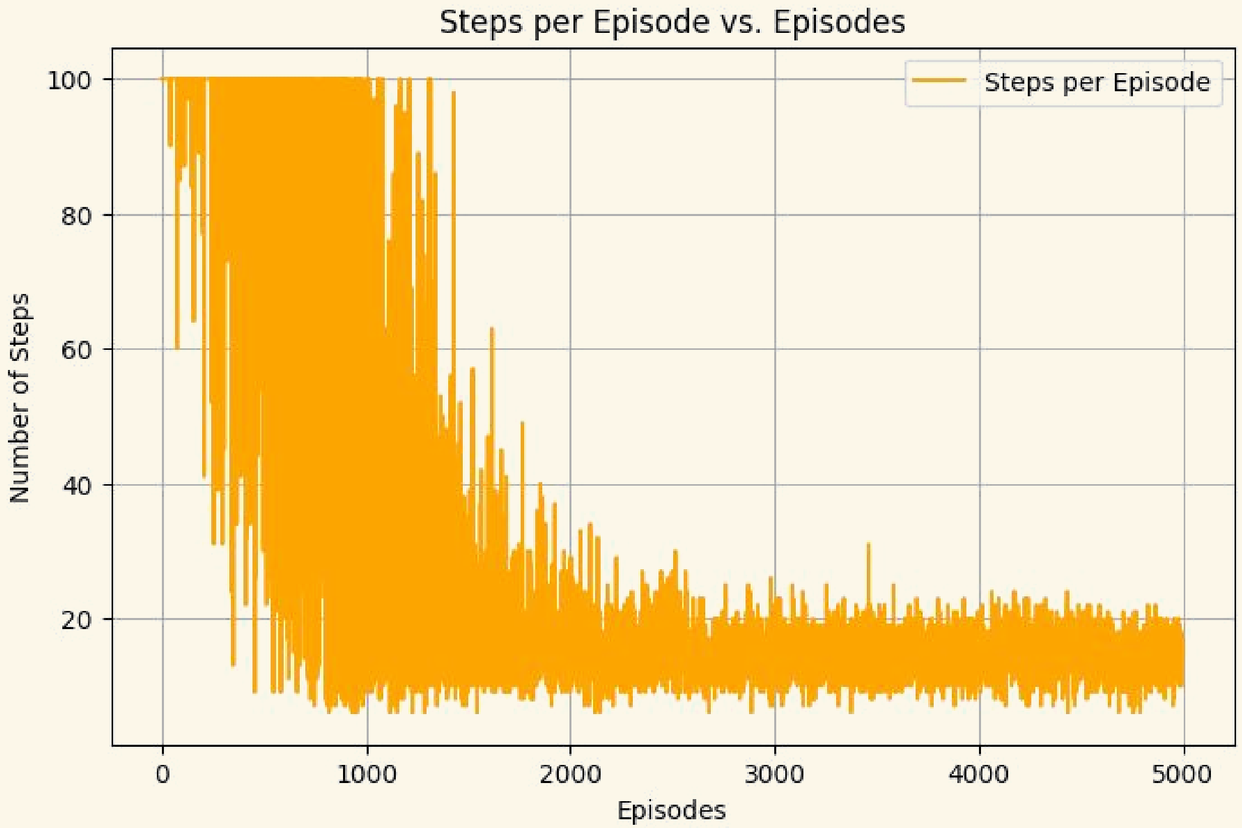
- Initial challenges due to sparse rewards.
- Steady improvement and eventual plateauing.

Steps Per Episode:

- Gradual reduction indicating increased efficiency.

Performance Consistency:

- Stability achieved after ~400 episodes.



Challenges and Future Opportunities

Challenges:

- Sparse rewards slow initial learning.
- Balancing exploration and exploitation.

Future Directions

- Transition to Approximate/Deep Q-Learning.
- Neural network integration for complex environments.
- Alternative exploration strategies (e.g., curiosity-driven learning).

project.py

```python
# Q-Learning algorithm
for episode in range(num_episodes):
    state, _ = env.reset()
    total_reward = 0
    steps = 0  # Initialize steps counter

    for step in range(max_steps):
        # Choose action using epsilon-greedy policy
        if np.random.uniform(0, 1) < epsilon:
            action = env.action_space.sample()  # Explore: choose random action
        else:
            action = np.argmax(q_table[state, :])  # Exploit: choose best action

        # Take the action and observe the outcome
        next_state, reward, done, truncated, info = env.step(action)

        # Update Q-value
        q_table[state, action] = q_table[state, action] + alpha * (
            reward + gamma * np.max(q_table[next_state, :]) - q_table[state, action]
        )

        state = next_state
        total_reward += reward
        steps += 1  # Increment steps counter

        if done:
            break

    # Store number of steps for this episode
    steps_per_episode.append(steps)

    # Decay epsilon
    epsilon = max(min_epsilon, epsilon * epsilon_decay)
    epsilon_values.append(epsilon)  # Store epsilon value for visualization

    rewards.append(total_reward)

    # Print progress
    if (episode + 1) % 500 == 0:
        print(f"Episode {episode + 1}/{num_episodes}, Total Reward: {total_reward}")
```

Conclusion

Project Outcomes:

- Effective demonstration of Q-Learning on discrete-state problems.
- Foundation for advanced RL techniques.

Impact:

- Practical guide for hyperparameter tuning and strategy optimization.

Next Steps:

- Application to real-world scenarios and more sophisticated environments.