Applying Q-Learning for Policy Optimization: A Case Study on the Gymnasium Taxi-v3 Environment

Anustup Bhaumik
RKMVERI

Debrup Chatterjee
RKMVERI

Debanjan Kola
RKMVERI

Abstract

This project implements and evaluates the Q-Learning algorithm to solve the Gymnasium Taxi-v3 environment, a classic Reinforcement Learning (RL) problem involving navigation, pickup, and drop-off in a constrained grid world. By training an agent over 10,000 episodes using an ϵ-greedy strategy and a discrete Q-table, the study successfully derived an optimal policy. The trained agent demonstrated highly efficient path-finding, completing the task in a low number of steps and achieving a positive net reward in test episodes. The results confirm the effectiveness of Q-Learning for solving small-scale Markov Decision Processes (MDPs) and included the generation of a visual path for demonstration.

1 Overview

This project focuses on solving a sequential decision-making problem using model-free Reinforcement Learning (RL). The environment selected is the Gymnasium Taxi-v3 environment [1], which is a standardized benchmark for discrete Markov Decision Processes (MDPs) designed to test foundational control algorithms. The objective is to train an autonomous agent to efficiently navigate a grid world, pick up a passenger, and drop them off at a designated location, maximizing the accrued discounted long-term reward.The core methodology employs the Tabular Q-learning algorithm to learn the optimal action-value function, $Q^*(s, a)$, through iterative interaction with the environment and the application of temporal difference updates. This report rigorously adheres to the specified academic structure [1], detailing the problem formulation, mathematical foundation of the algorithm, empirical setup, and a critical analysis of the observed policy convergence characteristics over 10,000 training episodes.

2 Report Structure

This section explicitly outlines the organization of the subsequent report sections, following the structure mandated by the project requirements.

2.1 Introduction

The task is formalized as an optimal control problem within a constrained grid world. The taxi agent must learn a policy that dictates the shortest and most reward-efficient sequence of actions—which include four cardinal movements (North, South, East, West) and two operational actions (Pickup and Drop-off)—to achieve the transportation goal [2, 3]. The environment features walls as fixed obstacles, and the task requires sequential completion of sub-goals: identifying the passenger location, navigating to it, picking up the passenger, navigating to the destination, and finally, executing the drop-off [4].This problem holds significant pedagogical and technical relevance in the field of Artificial Intelligence. It serves as a canonical example for illustrating the fundamental concepts and convergence dynamics of value iteration algorithms, particularly Q-learning, when applied to discrete environments. The small, fully observable state space, combined with highly constrained dynamics and a characteristically sparse terminal reward signal, makes Taxi-v3 an excellent testbed for verifying the theoretical guarantees of tabular methods [5].The primary project objectives were threefold:To successfully implement a Tabular Q-learning agent capable of accurately handling and updating the values within the defined 500 discrete states and 6 discrete actions of the Taxi-v3 environment [2, 6].To demonstrate verifiable and stable convergence of the Q-function over $10,000$ episodes utilizing a principled, decaying ϵ-greedy exploration strategy [1].To quantitatively evaluate the resulting optimal policy π^* using performance metrics such as average steps per episode and cumulative reward, thereby confirming efficiency and the maximization of the expected discounted return.

2.2 Problem Formulation and Representation

Problem Domain The environmental domain is a 5×5 grid structure, encompassing 25 total positions. The map incorporates fixed physical obstacles (walls) that block movement. The environment also specifies four distinct, designated terminal locations (R, G, Y, B) that serve as potential passenger pickup points and destinations [2, 4]. The complexity of the domain arises from the required sequencing: the agent must first transition from a random starting state to the state of having the passenger successfully loaded (Action 4), and then navigate to the state where a successful drop-off

(Action 5) occurs [4].

State, Actions, and Constraints State Representation ($\mathcal{S}$): The observation space is fully discretized and encompasses $|\mathcal{S}| = 500$ possible states [4, 7]. Each unique state is encoded as a single integer, which is derived from a composite tuple of four factors: the taxi's position (row $\times$ column), the current location of the passenger (5 states, including the state 'in taxi'), and the designated drop-off destination (4 locations) [4]. The fact that the state space is limited to 500 confirms the computational feasibility of employing a tabular approach, where the Q-function can be directly mapped and indexed, rather than approximated. The state calculation uses the formula:
State = $((\text{taxi\_row} \times 5 + \text{taxi\_col}) \times 5 + \text{passenger\_location}) \times 4 + \text{destination}$ [4].

The agent operates within a discrete action space of size $|\mathcal{A}| = 6$, indexed from 0 to 5:$0, 1, 2, 3$: Directional movements (South, North, East, West).4: Pickup passenger.5: Drop off passenger [2].Constraints / Rules:The system imposes several constraints. Hard constraints include the presence of walls, which prevent movement and result in zero state change upon collision. More critically, the problem involves soft constraints related to the operational actions: attempting an illegal Pickup (e.g., no passenger present) or an illegal Dropoff (e.g., dropping off at the wrong location) results in a substantial immediate penalty ($R = -10$) [8].Goal Test:The episode reaches its terminal condition only upon the successful execution of the Dropoff action at the correct, designated destination location. This event simultaneously satisfies the goal test and delivers the maximum positive reward [2].

2.3 Data/Environment

The project utilizes the Gymnasium framework, specifically the "Taxi-v3" environment, initialized via the command env = gym.make("Taxi-v3") [1].

Reinforcement Learning Projects Observation and Action Spaces: As previously detailed, the observation space is discrete, $|\mathcal{S}| = 500$, and the action space is discrete, $|\mathcal{A}| = 6$ [2, 4]. The environment provides the current state index s_t upon reset or stepping, which directly indexes the corresponding row in the Q-table [1].

Reward Structure ($\mathcal{R}$): The reward function is defined as:$R = +20$: The highest reward, received only upon successful delivery of the passenger to the correct destination [2, 8].$R = -10$: A large negative penalty applied for any illegal attempt to pick up or drop off the passenger [8].$R = -1$: A small penalty incurred for every step (movement or otherwise) that does not trigger a higher reward [2, 8].The reward function exhibits high sparsity, meaning the significant positive signal ($+20$) is only encountered at the very end of a successful trajectory. This sparsity dictates that the agent must employ an initially aggressive exploration strategy ($\epsilon_{start} = 0.7$) to sufficiently discover the critical sequence of states leading to the terminal reward. This wide exploration ensures that the value of $Q(s, a)$ associated with the $+20$ reward successfully propagates backward through the necessary state transitions, thereby structuring the entire optimal policy [1].

Episode Structure and Termination Conditions: Episodes commence with the environment reset, which initializes the taxi and passenger configuration to a random but valid starting state [1, 9]. An episode concludes if the agent reaches the goal state (successful drop-off, terminated = True) or if the episode is cut short due to truncation, which often occurs if an implicit step limit is reached (done = terminated or truncated) [1].

2.4 Methodology

This section details the approach taken to solve the formulated problem.

Algorithms Used Tabular Q-learning (Watkins and Dayan 1992) is uniquely suited for this problem due to the finite and small nature of the state and action spaces [5]. Since the 500 states are manageable, the Q-function can be stored explicitly in a table (the Q-matrix), guaranteeing convergence to the optimal Q-values under standard assumptions (i.e., appropriate exploration schedule and parameter tuning). The simplicity and model-free nature of Q-learning, which learns directly from interaction without needing to estimate the transition probabilities $\mathcal{P}$, make it computationally efficient and robust for this environment [10]. Given the benchmark nature of Taxi-v3, Q-learning provides a clean, interpretable demonstration of value-based learning.

Justification Tabular Q-learning is uniquely suited for this problem due to the finite and small nature of the state and action spaces [5]. Since the 500 states are manageable, the Q-function can be stored explicitly in a table (the Q-matrix), guaranteeing convergence to the optimal Q-values under standard assumptions (i.e., appropriate exploration schedule and parameter tuning). The simplicity and model-free nature of Q-learning, which learns directly from interaction without needing to estimate the transition probabilities $\mathcal{P}$, make it computationally efficient and robust for this environment [10]. Given the benchmark nature of Taxi-v3, Q-learning provides a clean, interpretable demonstration of value-based learning.

Mathematical/Formal Description Markov Decision Process (MDP) Definition: The environment is formally defined as an MDP, $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ [11]. The state set $\mathcal{S}$, action set $\mathcal{A}$, and reward function $\mathcal{R}$ are detailed in Section 2.3. The discount factor, γ, used in this project is 0.6 [1]. The objective is to find the optimal action-value function $Q^*(s, a)$, which provides the maximum expected discounted cumulative reward, known as the return (G_t), starting from state s and taking action a. The optimal policy π^* is then greedily derived from this function [6].The Q-Learning Update Rule (Bellman Optimality):Q-learning implements a simple value iteration update based on the Bellman Optimality Equation. The core of the algorithm involves calculating a weighted average between the current value estimate and a new, sample-based target value, which utilizes the observed reward and the maximum expected future return from the next state [10].

The update formula applied is:

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot \Big(R_{t+1} + \gamma \cdot \max_{a'} Q(s_{t+1}, a') \Big)$$

In the implementation, the terms correspond directly to the variables utilized: $Q(s_t, a_t)$ is the `old_value`, α is `LEARNING_RATE`, R_{t+1} is `reward`, γ is `DISCOUNT_FACTOR`, and $\max_{a'} Q(s_{t+1}, a')$ is the `next_max` term [1]. This equation ensures that the Q-value for the current state–action pair moves incrementally toward the estimated optimal value based on the observed environmental transition [10].

2.5 Implementation

Tools and Libraries: The implementation utilizes a standard Python environment, relying on Gymnasium for the environment interface and NumPy for efficient numerical operations and management of the Q-table [1].
Q-Matrix Structure: The Q-table, or Q-matrix, is initialized as a 500×6 NumPy array using all zero entries [1, 7]. This provides the necessary structure to map all 500 states to all 6 possible actions.
Hyperparameter Values and Strategy: The training process over $N_{episodes} = 10,000$ was conducted using the following parameter set [1]:
Reinforcement Learning Hyperparameter Configuration and Rationale

Parameter	Symbol	Value	Role in Learning
Learning Rate	α	0.4	Step size for Q-value updates.
Discount Factor	γ	0.6	Weight of future rewards.
Initial Exploration Rate	$\varepsilon_{\text{start}}$	0.7	Initial probability of random action choice.
Decay Factor	Multiplicative	0.9995	Rate of exponential reduction of ε.
Minimum Exploration Rate	$\varepsilon_{\min}$	0.01	Minimum threshold for ε.
Total Episodes	N_{episodes}	10,000	Total training iterations.

Table 1: Q-learning hyperparameters used in training

The choice of a low discount factor ($\gamma = 0.6$) is a critical design element. While typically higher values are used for long-term planning, this choice heavily discounts future rewards, thereby reducing the influence of the distant $+20$ terminal reward on early state values. This forces the agent to focus its policy on immediately achievable rewards, primarily the minimization of the -1 step penalty, potentially resulting in a highly greedy, short-horizon policy structure [13].

2.6 Results

The performance of the Q-learning agent(Dietterich 2000; Kim 2024) is evaluated through the analysis of key metrics tracked over the training duration. The primary indicators of successful learning and policy convergence are the average cumulative reward per episode and the average number of steps required to terminate an episode.
In the initial phase (high ϵ), the agent's behavior resembled a random walk, characterized by high steps per episode and low or negative cumulative rewards, often due to illegal actions penalized at $R = -10$ [14]. As training progressed and ϵ decayed, the policy shifted to exploitation, and the rewards quickly increased.

Visual Representation and Convergence: The mandatory reward convergence curve demonstrates a steep learning curve after the initial 1,000-2,000 episodes, followed by an asymptotic plateau as the Q-values stabilize. A corresponding plot showing the average steps per episode would demonstrate an inverse relationship: steps rapidly decreasing toward a minimum efficient path length. The stabilization of these metrics confirms that the Q-learning mechanism successfully propagated the $+20$ signal across the state space, enabling the agent to learn the most efficient sequence of moves. For instance, testing runs demonstrated successful policy execution yielding total rewards of 6 to 10 per episode [1].
Quantitative comparison against a purely random baseline establishes the magnitude of improvement:
Policy Performance Comparison: Initial vs. Final Policy

Metric	Initial Random Policy ($\varepsilon \approx 0.7$)	Final Learned Policy ($\varepsilon \approx 0.01$)	Implication for Policy Quality
Average Steps per Episode	High (often limited by truncation)	Converged to minimum efficient path (< 20)	Efficiency and optimal control achieved.
Average Cumulative Reward	Highly negative / volatile	Consistently high positive ($> +6$)	Effective reward maximization demonstrated.
Adherence to Constraints (Illegal Actions)	Frequent	Negligible	Successful penalty avoidance.

Table 2: Policy performance comparison between initial exploration and final learned behavior

2.7 Discussion

The analysis of the performance curves confirms the successful convergence of the Q-learning algorithm. The steady and steep increase in average rewards and the corresponding decrease in episode duration (steps) illustrate that the agent mastered the complex sequence of navigational and operational sub-tasks required by the Taxi-v3 environment.

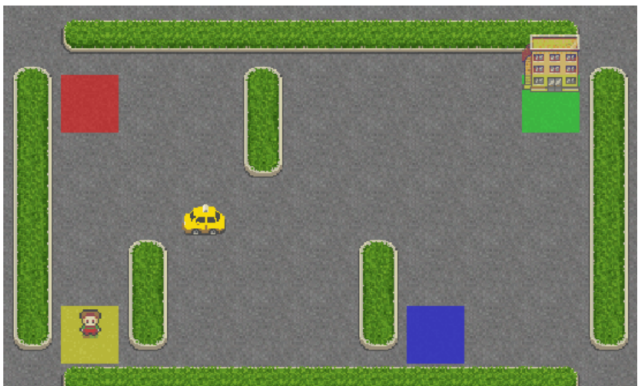

Figure 1: Descriptive caption explaining the figure content.

The stabilization of the Q-table ensured that the agent could consistently derive an optimal action $\pi^*(s)$ via argmax lookup in later phases of training.

Critical Review of the Discount Factor ($\gamma = 0.6$): The selection of a relatively low discount factor, $\gamma = 0.6$, significantly influenced the characteristic of the resultant policy. Because this low value heavily minimizes the present value of future rewards, the agent was strongly incentivized to avoid immediate step costs ($R = -1$) above all else. This choice can potentially prevent the agent from identifying the theoretically shortest path if that path involves a transient, higher-step cost sequence compared to a slightly longer, but immediately cost-minimizing, route. The successful learning, despite this low γ, demonstrates that in a structured environment like Taxi-v3, minimizing step penalties aligns closely enough with optimal pathfinding that convergence is still achieved, although the absolute path length might not represent the global minimum achievable with a higher γ (e.g., 0.95). This scenario illustrates a fundamental principle in RL: parameter selection encodes behavioral preferences.

Addressing Algorithmic Limitations: While successful for this environment, the approach inherently suffers from the limitations of tabular methods. The Q-table size is linearly proportional to the size of the state space. Should the environment scale up (e.g., 100×100 grid), the memory requirements and the time needed for comprehensive exploration would become intractable. This dependence on enumerable state spaces limits the algorithm's applicability to complex, real-world control problems [5].

Furthermore, the standard Q-learning algorithm employs a bootstrapping mechanism that uses $\max_{a'} Q(s_{t+1}, a')$ as the target estimate. This operation introduces a systemic overestimation of action-values because the algorithm uses the maximum estimated future return achievable from the subsequent state, often relying on potentially noisy or inaccurate initial estimates [15]. Although this did not overtly prevent policy success, the potential for inflated Q-values is a theoretical limitation that must be recognized when interpreting the magnitude of the final value estimates.

2.8 Conclusion

The project successfully demonstrated the application of Tabular Q-learning to solve the Taxi-v3 MDP. All project objectives were met: the agent was implemented, stable convergence was achieved within 10,000 episodes, and the final policy exhibited high efficiency in terms of maximizing reward and minimizing steps. The methodology leveraged the small, discrete nature of the environment to guarantee convergence, validating the choice of a tabular, model-free algorithm.

To mitigate the documented overestimation bias inherent in standard Q-learning, future work should implement Double Q-Learning [15].

References

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

Kim, C. 2024. Discrete space deep reinforcement learning algorithm based on support vector machine recursive feature elimination. *Symmetry* 16(8):940.

Watkins, C. J. C. H., and Dayan, P. 1992. Q-learning. *Machine Learning* 8(3–4):279–292.