Q-Learning: Training an Agent in the Discrete Gymnasium Taxi-v3 Environment

Pathri Vidya Praveen

August 25, 2025

Abstract

This document details the implementation of **Q-learning**, a model-free reinforcement learning algorithm, applied to the discrete **Taxi-v3** environment within the **Gymnasium** library. The project aims to train an autonomous agent (a taxi) to learn an optimal policy that maximizes cumulative rewards by successfully picking up and dropping off passengers. We cover the core theoretical concepts, including the Q-table, ϵ-greedy exploration strategy, and the Q-value update rule. The methodology is demonstrated specifically for the `Taxi-v3` environment. The document also includes a discussion of the training process, the visualization of the learning curve, and simulation results showcasing the trained agent's performance.

1 Introduction

Reinforcement Learning (RL) is a paradigm of machine learning where an agent learns to make decisions by performing actions in an environment to maximize some notion of cumulative reward. Unlike supervised or unsupervised learning, RL does not rely on predefined datasets; instead, the agent learns through trial and error.

Q-learning is a popular, off-policy, model-free reinforcement learning algorithm. "Model-free" means the agent doesn't need to understand the environment's dynamics (i.e., how actions lead to new states and rewards). "Off-policy" means it learns the value of the optimal policy independently of the agent's actual actions. The primary objective of Q-learning is to learn a **Q-function** (or **Q-table**) that estimates the expected future reward for taking a specific action in a given state.

2 Core Concepts of Q-Learning

2.1 States, Actions, and Rewards

In the context of reinforcement learning, the interaction between an agent and its environment is defined by three fundamental components:

- **State (s):** A complete description of the environment at a specific moment. For discrete environments like `Taxi-v3`, states are typically represented by integers.

- **Action (a):** A move or decision made by the agent within a given state. Actions are discrete for the `Taxi-v3` environment (e.g., moving North, Pick-up).
- **Reward (r):** A numerical feedback signal from the environment to the agent, indicating the desirability of an action taken. The agent's goal is to maximize the sum of these rewards over time.

2.2 The Q-Table

The central component of the Q-learning algorithm is the **Q-table** (or Q-matrix). This is a two-dimensional array where rows represent states (s) and columns represent actions (a). Each cell, $Q(s, a)$, stores an estimate of the maximum discounted future reward that can be obtained by taking action a in state s and then following an optimal policy thereafter. Initially, the Q-table is typically filled with zeros, signifying that the agent has no prior knowledge of the environment's rewards.

2.3 Hyperparameters

Several hyperparameters govern the learning process in Q-learning:

- **Learning Rate (α):** This parameter determines how much new information overrides old information. A value of $\alpha = 0$ means the agent learns nothing, while $\alpha = 1$ means the agent considers only the most recent information. A common range is $[0.1, 0.9]$.
- **Discount Factor (γ):** This factor determines the importance of future rewards. A $\gamma = 0$ makes the agent "myopic," considering only immediate rewards, while a γ closer to 1 makes the agent strive for long-term rewards. Typical values are in the range $[0.8, 0.99]$.
- **Exploration Rate (ϵ):** This parameter dictates the balance between exploration and exploitation. During training, ϵ often starts high and gradually decays over time, allowing the agent to explore initially and then exploit its learned knowledge.

2.4 ϵ-Greedy Exploration Strategy

To effectively learn, an agent must balance **exploration** (trying new actions to discover potentially better rewards) and **exploitation** (taking actions known to yield high rewards). The ϵ-greedy strategy achieves this balance:

- With probability ϵ, the agent chooses a random action (exploration).
- With probability $1 - \epsilon$, the agent chooses the action that has the maximum Q-value for the current state (exploitation).

During training, ϵ typically starts at 1.0 (pure exploration) and gradually decays towards a minimum value (e.g., 0.01), allowing the agent to shift from exploring to exploiting its learned policy.

2.5 Q-Value Update Rule

The core of the Q-learning algorithm lies in its iterative update rule for the Q-table. After taking an action a in state s, observing a reward r, and transitioning to a new state s', the Q-value for the (s, a) pair is updated using the following formula:

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

Let's break down the components of this formula:

- $Q(s, a)$: The current estimated Q-value for taking action a in state s.
- r: The immediate reward received after taking action a in state s.
- $\gamma \max_{a'} Q(s', a')$: The maximum predicted future reward from the next state s', discounted by γ. This term represents the optimal value of the next state.
- $r + \gamma \max_{a'} Q(s', a')$: This is the **target Q-value**, representing the sum of the immediate reward and the discounted maximum future reward.
- $r + \gamma \max_{a'} Q(s', a') - Q(s, a)$: This is the **temporal difference (TD) error**, which measures the difference between the new, more informed estimate of the Q-value (target Q-value) and the current estimate.
- $\alpha [\ldots]$: The learning rate α scales the TD error, determining how much of this error is applied to update the current $Q(s, a)$ value.

3 Gymnasium Environment: Taxi-v3

This project specifically focuses on the `Taxi-v3` environment from the Gymnasium library. This environment presents a challenge where an agent controls a taxi to perform pick-up and drop-off tasks.

3.1 Taxi-v3 Environment Description

The `Taxi-v3` environment features a 5×5 grid. The agent's primary task involves:

1. Navigating to one of four designated locations to **pick up** a passenger.
2. Driving the passenger to one of four designated **destination** locations.
3. **Dropping off** the passenger at the correct destination.

- **States:** 500 discrete states. These states encode the taxi's current location (row, column), the passenger's current location (which can be one of the four pick-up points or inside the taxi), and the passenger's destination location.
- **Actions:** 6 discrete actions:

  - Move South
  - Move North
  - Move East
  - Move West
  - Pick-up passenger
  - Drop-off passenger
- **Rewards:** The environment provides structured rewards to guide learning:
  - +20 for a successful passenger drop-off at the correct destination.
  - −1 for each step taken (encourages efficiency).
  - −10 for illegal pick-up or drop-off actions (penalizes incorrect behavior).

4 Implementation Details

The Python implementation follows a standard reinforcement learning workflow tailored for the `Taxi-v3` environment:

1. **Environment Initialization:** The `Taxi-v3` Gymnasium environment is instantiated. Its state space (500 states) and action space (6 actions) sizes are determined to initialize the Q-table.
2. **Q-Table Initialization:** The Q-table is created as a NumPy array of size 500×6, filled with zeros, representing no initial knowledge.
3. **Hyperparameter Setup:** The learning rate ($\alpha = 0.1$), discount factor ($\gamma = 0.99$), initial exploration rate ($\epsilon = 1.0$), and its decay rate ($\epsilon_{decay} = 0.999$) are defined. A larger total number of training episodes (e.g., 25,000) is set due to the `Taxi-v3` environment's complexity.
4. **Training Loop:**
   - For each of the 25,000 episodes, the environment is reset to an initial state where the taxi, passenger, and destination are randomly placed.
   - The agent interacts with the environment in a loop until the episode terminates (passenger successfully dropped off or maximum steps reached).
   - At each step, an action is selected using the ϵ-greedy strategy, balancing exploration of new routes/actions and exploitation of learned optimal paths.
   - The chosen action is executed using `env.step(action)`, and the new state, reward, and termination status are observed.
   - The Q-table is updated using the Q-learning update rule to refine the taxi's understanding of state-action values.

- The exploration rate ϵ is decayed exponentially per episode to shift the agent from random exploration to deterministic exploitation.
- The total reward for the episode is accumulated and recorded.

5. **Learning Curve Plotting:** After training, the recorded total rewards per episode are plotted against the episode number to visualize the agent's learning progression. This plot helps in understanding how quickly and effectively the taxi agent learns to maximize rewards.

6. **Agent Simulation:** The trained agent is run for a few episodes (e.g., 3 episodes) with pure exploitation ($\epsilon = 0$) to demonstrate its learned optimal policy. The `env.render()` function is used to visually display the taxi's movement, pick-up, and drop-off actions within the environment, providing clear feedback on the agent's performance.

5 Results and Discussion

5.1 Learning Curve

The learning curve, plotting the total reward accumulated per episode over the training duration, serves as a crucial indicator of the agent's learning progress. An upward trend in this curve signifies that the agent is successfully learning to achieve higher rewards and, consequently, improving its policy. For the `Taxi-v3` environment, the learning curve might initially show some fluctuations due to the larger state space and more complex reward structure, requiring more episodes for stable learning. However, it should eventually show a clear upward trend, converging towards higher average rewards as the agent masters the pick-up and drop-off tasks.

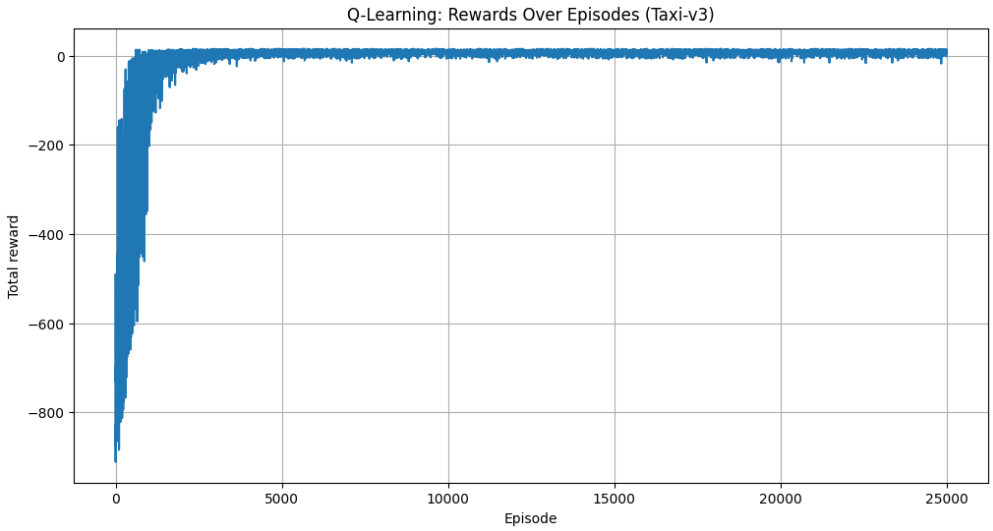

Figure 1: Q-Learning: Rewards Over Episodes for Taxi-v3

5.2 Simulation

The simulation phase demonstrates the agent's behavior after training. By running the agent with $\epsilon = 0$ (pure exploitation), we can observe its learned policy in action. For `Taxi-v3`, the simulation will show the taxi agent efficiently navigating the grid, picking up the passenger, driving to the correct destination, and dropping them off, all while incurring minimal penalties. The `env.render()` function provides a crucial visual representation of these actions, making the agent's learned behavior tangible.

6 Conclusion

This project successfully demonstrates the application of Q-learning to train an agent in the discrete Gymnasium `Taxi-v3` environment. By meticulously implementing and understanding the Q-table, ϵ-greedy strategy, and the Q-value update rule, the taxi agent effectively learns to navigate, pick up, and drop off passengers, achieving its goals while maximizing cumulative rewards. The ability to visualize learning through reward curves and simulate the trained agent's behavior provides clear insights into the algorithm's performance on this multi-step task. Future work could involve further hyperparameter tuning for optimal performance, exploring more advanced reinforcement learning algorithms, or integrating neural networks for handling larger or continuous state spaces, leading to the development of Deep Q-Networks (DQNs).