Q-learning Tutorial

September 21, 2025

1 Introduction

Q-learning is an off-policy, model-free reinforcement learning algorithm that learns the optimal action-value function by interacting with an environment. By iteratively updating state-action values with bootstrapped targets, Q-learning converges to the optimal policy under mild assumptions even when actions are selected using exploratory behaviour such as ε-greedy strategies.

2 Theory and Formulas

2.1 Action-Value Function

For a Markov Decision Process (MDP) with states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, transition probability P, and reward R, the optimal action-value function satisfies the Bellman optimality equation

$$Q^*(s,a) = \mathbb{E}\Big[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a\Big], \tag{1}$$

where $\gamma \in [0,1)$ is the discount factor.

2.2 Update Rule

Q-learning maintains an estimate Q_t that is updated after observing transition $(s_t, a_t, r_{t+1}, s_{t+1})$:

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha_t \Big[r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)\Big], \tag{2}$$

with learning rate α_t. Actions during data collection can follow an ε-greedy policy $\pi(a|s)$ that selects the greedy action with probability $1 - \varepsilon$ and explores otherwise.

2.3 Convergence Considerations

If learning rates satisfy $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$, all state-action pairs are visited infinitely often, and rewards are bounded, Q-learning converges to Q^* with probability 1. In practice, constant learning rates and decaying exploration are used. Function approximation and large state spaces require variants such as Deep Q-Networks (DQN) with replay buffers and target networks.

3 Applications and Tips

- **Game playing**: learn control policies in discrete environments (grid worlds, Atari) without explicit models.
- **Robotics and control**: discretized action spaces for navigation and low-level control.
- **Operations research**: optimize inventory management or queueing decisions via simulation.
- **Best practices**: normalize rewards, anneal exploration, monitor learning curves, and clip updates or rewards to stabilize training.

4 Python Practice

The script `gen_q_learning_figures.py` simulates a 2D grid-world with terminal rewards, applies tabular Q-learning, and records episode returns and greedy state values for visualization.

Listing 1: Excerpt from gen$_{ql}earning_{f}igures.py$

```python
for episode in range(num_episodes):
    state = env.reset()
    done = False
    G = 0.0
    while not done:
        action = epsilon_greedy(Q[state], epsilon)
        next_state, reward, done = env.step(action)
        best_next = np.max(Q[next_state])
        Q[state, action] += alpha * (reward + gamma * best_next - Q[state, action])
        state = next_state
        G += reward
    returns.append(G)
```

5 Result

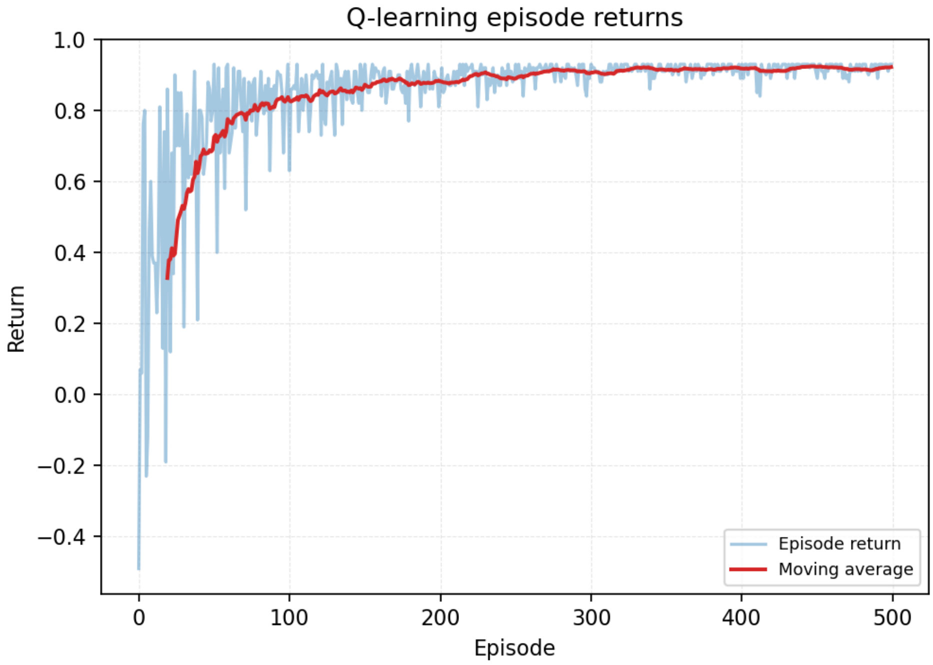

Figure 1: Q-learning episode returns moving toward the optimal value

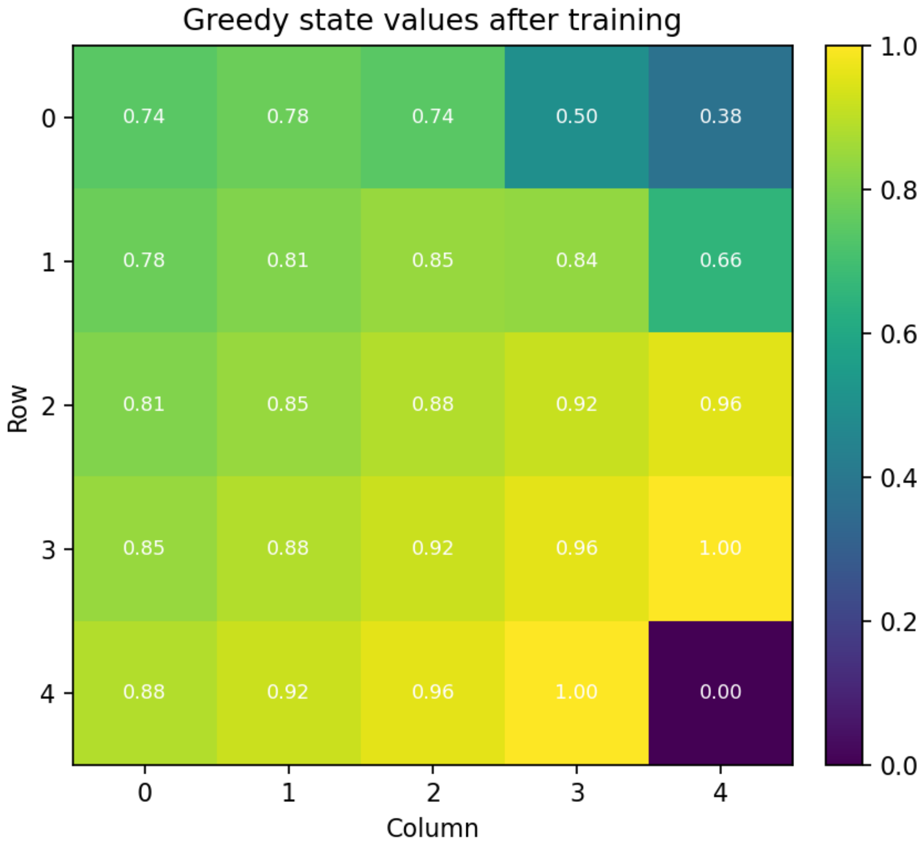

Figure 2: Heatmap of greedy state values after training, highlighting shortest-path structure

6 Summary

Q-learning estimates optimal action values via temporal-difference updates using max bootstrapping. Careful tuning of learning rate, exploration schedule, and reward scaling yields stable convergence. The grid-world example illustrates how episode returns improve over time and how learned state values encode optimal trajectories.