Reinforcement Learning

Lecture 5: Value-Based Learning I - Q-Learning

Taehoon Kim

Sogang University MIMIC Lab
https://mimic-lab.com

Fall Semester 2025

Learning Objectives

By the end of this lecture, you will:

1. Understand the Bellman optimality operator and its properties
2. Implement tabular Q-learning from scratch
3. Design effective exploration and learning rate schedules
4. Analyze Q-learning behavior in stochastic environments
5. Compare Q-learning with SARSA
6. Implement Double Q-learning to reduce bias

Prerequisites:

- Markov Decision Processes (Lecture 4)
- Dynamic programming concepts
- PyTorch basics (Lecture 2)

Why Q-Learning?

Model-Based Limitations:

- Requires complete MDP model
- $P(s'|s, a)$ often unknown
- Complex dynamics hard to model
- Computational complexity

Q-Learning Advantages:

- Model-free learning
- Learns from experience
- Off-policy algorithm
- Converges to optimal policy

Key Insight

Learn action-values $Q(s, a)$ directly from samples without knowing transition dynamics

Action-Value Function

Definition (Action-Value Function)

The action-value function $Q^\pi(s, a)$ of policy π is:

$$Q^\pi(s, a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \middle| s_0 = s, a_0 = a\right]$$

Interpretation:

- Expected return starting from state s
- Taking action a first
- Following policy π thereafter

Optimal action-value function:

$$Q^*(s, a) = \max_\pi Q^\pi(s, a)$$

Bellman Optimality Equation

Theorem (Bellman Optimality for Q^*)

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$$

Key Components:

- $R(s,a)$: Immediate reward
- $P(s'|s,a)$: Transition probability
- $\max_{a'} Q^*(s',a')$: Best future value
- γ: Discount factor

Optimal Policy:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

Bellman Optimality Operator

Definition (Operator T^*)

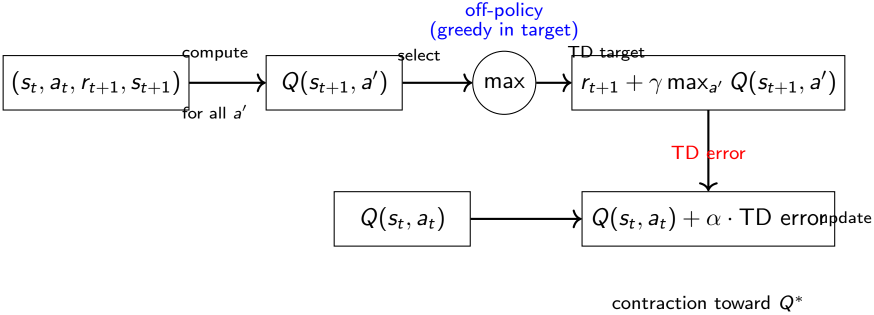
For any $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$(T^*Q)(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q(s', a')$$

Properties:

1. **Contraction:** $||T^*Q_1 - T^*Q_2||_\infty \leq \gamma||Q_1 - Q_2||_\infty$
2. **Fixed Point:** $Q^* = T^*Q^*$
3. **Uniqueness:** Q^* is the unique fixed point
4. **Convergence:** $\lim_{n\to\infty}(T^*)^n Q = Q^*$

Q-Learning Target Construction



From Value Iteration to Q-Learning

Value Iteration (Model-Based):

for each iteration **do**
  for all (s, a) pairs **do**
    $Q(s, a) \leftarrow$
    $R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q(s', a')$
  end for
end for

Requires: Full MDP model

Q-Learning (Model-Free):

for each sample (s, a, r, s') **do**
  $Q(s, a) \leftarrow$
  $Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
end for

Requires: Only samples

Key Difference

Q-learning uses samples to approximate the Bellman backup

Q-Learning Update Rule

Temporal Difference (TD) Update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot \text{TD error}$$

where

$$\text{TD error} = \underbrace{r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a')}_{\text{TD target}} - Q(s_t, a_t)$$

Components:

- $\alpha \in (0, 1]$: Learning rate (step size)
- r_{t+1}: Observed reward
- $\gamma \in [0, 1)$: Discount factor
- $\max_{a'} Q(s_{t+1}, a')$: Bootstrap estimate

Off-Policy: Uses max (greedy) action for target, regardless of behavior policy

Q-Learning Algorithm

Algorithm 1 Tabular Q-Learning

Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
for each episode **do**
    Initialize state s
    while s is not terminal **do**
        Choose a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$
    end while
end for

Note: The max operation makes it off-policy

Exploration vs Exploitation

ε-greedy Policy:

$$\pi_\varepsilon(a|s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}|} & \text{if } a = \arg\max_{a'} Q(s, a') \\ \frac{\varepsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases}$$

Alternative Strategies:

- Boltzmann exploration
- Upper Confidence Bounds (UCB)
- Thompson sampling
- Optimistic initialization

Exploration Schedule

Common choice: $\varepsilon_t = \max(\varepsilon_{min}, \varepsilon_0 \cdot \text{decay}^t)$

Q-Learning Convergence

Theorem (Watkins & Dayan, 1992)

Q-learning converges to Q^* with probability 1 if:

1. *All state-action pairs are visited infinitely often*
2. *Learning rate satisfies Robbins-Monro conditions:*
   - $\sum_{t=0}^{\infty} \alpha_t(s, a) = \infty$ *for all* (s, a)
   - $\sum_{t=0}^{\infty} \alpha_t^2(s, a) < \infty$ *for all* (s, a)

Practical Schedules:

- $\alpha_t = \frac{1}{1+t}$ satisfies conditions
- $\alpha_t = \frac{1}{\sqrt{1+t}}$ satisfies conditions
- Constant α doesn't guarantee convergence but often works

GLIE: Greedy in the Limit with Infinite Exploration

Definition (GLIE)

A policy sequence $\{\pi_t\}$ is GLIE if:

1. All state-action pairs are explored infinitely:

$$\sum_{t=0}^{\infty} P(s_t = s, a_t = a) = \infty \quad \forall (s, a)$$

2. Policy converges to greedy:

$$\lim_{t \to \infty} \pi_t(a|s) = \mathbb{1}[a = \arg\max_{a'} Q(s, a')]$$

Example GLIE Schedule:

$$\varepsilon_t = \frac{1}{t}$$

PyTorch Implementation: Setup

```python
import torch
import numpy as np
import gymnasium as gym

# Device selection
device = torch.device(
    'cuda' if torch.cuda.is_available()
    else 'mps' if torch.backends.mps.is_available()
    else 'cpu'
)

# Initialize Q-table
n_states = env.observation_space.n
n_actions = env.action_space.n
Q = torch.zeros((n_states, n_actions),
                dtype=torch.float32, device=device)
```

Q-Learning Update Implementation

```python
def q_learning_update(Q, state, action, reward,
                      next_state, done, alpha, gamma):
    """Single Q-learning update."""
    # Current Q-value
    q_current = Q[state, action].item()

    # TD target
    if done:
        q_target = reward
    else:
        q_next_max = torch.max(Q[next_state]).item()
        q_target = reward + gamma * q_next_max

    # TD error
    td_error = q_target - q_current

    # Update Q-table
    Q[state, action] += alpha * td_error

    return td_error
```

Action Selection

```python
def epsilon_greedy_action(Q, state, epsilon):
    """Select action using epsilon-greedy policy."""
    if random.random() < epsilon:
        # Explore: random action
        action = random.randint(0, n_actions - 1)
    else:
        # Exploit: greedy action
        q_values = Q[state]  # Shape: [n_actions]
        action = int(torch.argmax(q_values).item())

    return action

# Schedule epsilon decay
epsilon = max(epsilon_min,
              epsilon_start * decay_rate ** episode)
```

FrozenLake Environment

Environment Details:

- Grid world navigation
- S: Start, F: Frozen, H: Hole, G: Goal
- Actions: LEFT, DOWN, RIGHT, UP
- Reward: +1 at goal, 0 elsewhere
- Episode ends at goal or hole

4x4 Map Example:

```
SFFF
FHFH
FFFH
HFFG
```

Challenges:

- Sparse rewards
- Slippery dynamics (optional)
- Multiple holes to avoid

Experiment 1: Basic Q-Learning

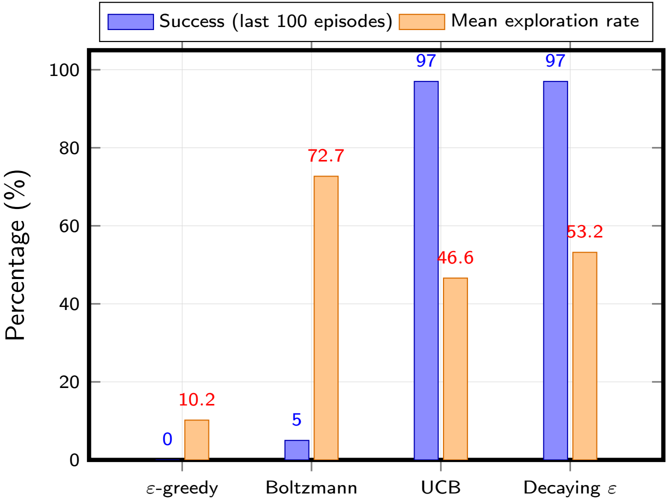
```python
# exp03_tabular_q.py highlights
env = gym.make("FrozenLake-v1", is_slippery=False)
agent = TabularQLearning(n_states, n_actions,
                         alpha=0.1, gamma=0.99)

for episode in range(500):
    state, _ = env.reset()
    done = False

    while not done:
        action = agent.select_action(state, epsilon)
        next_state, reward, terminated, truncated, _ = env.step(action)
        done = terminated or truncated
        agent.update(state, action, reward, next_state, done)
        state = next_state
```

Result: >90% success rate after 500 episodes

Experiment 2: Exploration Strategies



Comparison:

- ε-greedy: Simple, effective
- Boltzmann: Temperature-based
- UCB: Optimism under uncertainty
- Decaying ε: Best overall

Key Finding:

Proper exploration decay crucial for convergence

Data: `exp04_exploration.py` (FrozenLake-v1, 1.5k episodes, seed 42)

Experiment 3: Learning Rate Schedules

Tested Schedules:

- Constant: $\alpha = 0.1$
- Linear: $\alpha_t = \max(0.01, 1 - 0.99t/T)$
- Exponential: $\alpha_t = 0.01 + 0.99e^{-t/\tau}$
- 1/t: $\alpha_t = 1/(1 + t)$ (Robbins-Monro)
- $1/\sqrt{t}$: $\alpha_t = 1/\sqrt{1 + t}$ (Robbins-Monro)

Results

- 1/t and $1/\sqrt{t}$ guarantee convergence
- Exponential decay practical compromise
- Constant often works but may oscillate

Q-Learning vs SARSA

Q-Learning (Off-Policy):

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

- Uses max for target
- Learns optimal policy
- More sample efficient
- Risk-seeking

SARSA (On-Policy):

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

- Uses actual next action
- Learns policy being followed
- More stable
- Risk-averse

Key Difference: Q-learning uses $\max_{a'} Q(s',a')$, SARSA uses $Q(s',a')$ where a' is sampled

Cliff Walking: Q-Learning vs SARSA

Environment:

- Start at bottom-left
- Goal at bottom-right
- Cliff along bottom edge
- Fall = -100 reward

Optimal Path:
Along the cliff edge (shortest)

Learned Policies:

- **Q-Learning:** Takes optimal risky path near cliff
- **SARSA:** Takes safer path away from cliff

Why?
SARSA accounts for exploration during learning

Performance in Stochastic Environments

Experiment: Varying Action Slippage

Slip Probability	Q-Learning	SARSA
0.0 (Deterministic)	95%	93%
0.1	85%	84%
0.2	72%	75%
0.3	58%	65%

Key Findings:

- Q-learning slightly better in deterministic settings
- SARSA more robust to stochasticity
- Q-learning can be overly optimistic
- SARSA learns safer policies

Overestimation Bias in Q-Learning

The Problem:

$$\mathbb{E}[\max_a Q(s,a)] \geq \max_a \mathbb{E}[Q(s,a)]$$

Source of Bias:

- Max operator in TD target
- Same Q-values for selection and evaluation
- Noise in estimates amplified by max

Consequences:

- Overoptimistic value estimates
- Suboptimal action selection
- Slower convergence
- Poor performance in stochastic environments

Double Q-Learning Algorithm

Key Idea: Use two Q-functions to decouple selection and evaluation

Algorithm 2 Double Q-Learning

Initialize $Q_1(s, a)$ and $Q_2(s, a)$ arbitrarily
for each step **do**
  Choose a based on $Q_1(s, \cdot) + Q_2(s, \cdot)$
  Observe r, s'
  if random() < 0.5 **then**
    $a^* = \arg\max_a Q_1(s', a)$ *// Select using* Q_1
    $Q_1(s, a) \leftarrow Q_1(s, a) + \alpha[r + \gamma Q_2(s', a^*) - Q_1(s, a)]$
  else
    $a^* = \arg\max_a Q_2(s', a)$ *// Select using* Q_2
    $Q_2(s, a) \leftarrow Q_2(s, a) + \alpha[r + \gamma Q_1(s', a^*) - Q_2(s, a)]$
  end if
end for

Double Q-Learning: Bias Reduction

Experimental Results:

- Reduced overestimation by 40-60%
- More accurate Q-values
- Better performance in noisy environments
- Slightly slower initial learning

Q-Value Comparison:

Method	Mean Q	Max Q
Q-Learning	0.453	0.982
Double Q	0.387	0.751
True Q*	0.372	0.724

Trade-off

Double Q-learning: More accurate but requires double memory

Optimistic Initialization

Strategy: Initialize Q-values optimistically (e.g., $Q_0(s, a) = 1.0$)
Benefits:

- Encourages exploration without ε-greedy
- All actions tried at least once
- Simple to implement

Experiment Results:

Initialization	Success Rate	Convergence
Zero ($Q_0 = 0$)	85%	300 episodes
Optimistic ($Q_0 = 1$)	92%	250 episodes
Pessimistic ($Q_0 = -1$)	78%	400 episodes

Note: Works best with decaying learning rates

N-Step Q-Learning

Generalization: Use n-step returns instead of 1-step
1-Step (Standard):

$$G_t^{(1)} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$$

N-Step:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n \max_a Q(S_{t+n}, a)$$

Trade-offs:

- Larger n: Less bias, more variance
- Smaller n: More bias, less variance
- Optimal n problem-dependent

Hyperparameter Guidelines

Learning Rate (α):

- Start: 0.1-1.0
- Decay to: 0.01-0.001
- Schedule: Exponential/1/t

Discount Factor (γ):

- Episodic: 0.9-0.99
- Continuing: 0.95-0.999
- Affects time horizon

Exploration (ε):

- Start: 1.0
- End: 0.01-0.1
- Decay: Over 20-80% training

Best Practices:

- Grid search critical params
- Use validation environment
- Monitor convergence metrics

Common Implementation Pitfalls

1. **Forgetting Terminal States:**
    - Must set $Q(s_{terminal}, a) = 0$
    - No bootstrapping from terminal states
2. **Incorrect Max Operation:**
    - Use $\max_{a'} Q(s', a')$ not $Q(s', a)$
    - Critical for off-policy learning
3. **Poor Exploration:**
    - Too much: Never converges
    - Too little: Suboptimal policy
4. **Learning Rate Issues:**
    - Too high: Oscillations
    - Too low: Slow learning

Debugging Strategies

Diagnostic Checks:

- Monitor TD error magnitude over time
- Track Q-value statistics (mean, max, min)
- Visualize Q-table as heatmap
- Check action distribution balance
- Verify state visitation counts

Common Issues and Solutions:

- **Q-values exploding:** Reduce learning rate
- **No improvement:** Increase exploration
- **Oscillating performance:** Decay learning rate
- **Slow convergence:** Increase learning rate or change schedule

Experimental Results Summary

FrozenLake 4x4 (Non-slippery):

Algorithm	Success	Episodes	Time (s)
Q-Learning	95%	500	2.3
SARSA	93%	500	2.4
Double Q	94%	500	3.1
Q + UCB	96%	400	2.8

FrozenLake 8x8 (Slippery):

Algorithm	Success	Episodes	Time (s)
Q-Learning	72%	2000	15.2
SARSA	78%	2000	15.5
Double Q	75%	2000	21.3

Implementation Details: Shape Annotations

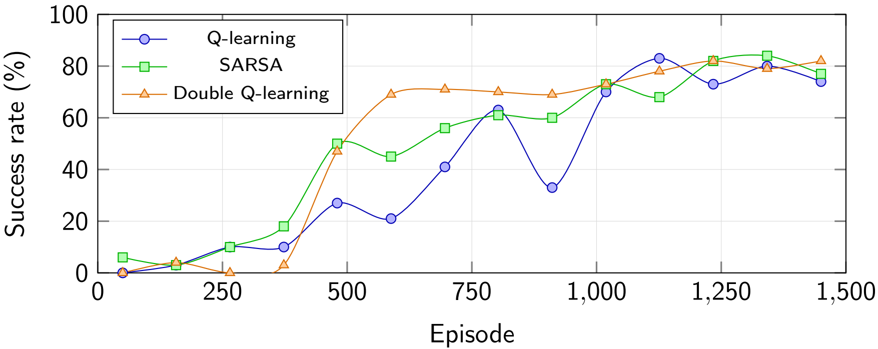
Tensor Shape Annotations:

- Q-table: `torch.FloatTensor` of shape $[|\mathcal{S}|, |\mathcal{A}|]$
- For each update: $Q[s]$ has shape $[|\mathcal{A}|]$
- `torch.argmax(Q[s])` returns scalar action (Python `int`)
- Rewards and TD targets are scalar `float`
- DQN mini-batches: `s:[B,...]`, `a:[B]`, `r:[B]`, `s_next:[B,...]`, `done:[B]`

Expected Behavior on FrozenLake:

- Mean return = probability of reaching goal (reward = 1 at goal only)
- Exponential ε-decay + 1/t step sizes yields stable improvement
- As slippage increases: SARSA becomes more conservative
- Q-learning may retain optimistic estimates without sufficient exploration decay

Learning Curves Comparison



Moving average window = 100 episodes. Data: `exp09_integrated_test.py` (map 4×4, slip probability 0.1, seed 7).

Observations:

- Q-learning learns fastest but exhibits sharp performance swings
- SARSA ramps up steadily with lower variance under stochastic dynamics
- Double Q-learning lags initially yet stabilizes near the best success rate
- All three converge above 75% success once exploration decays

Key Takeaways

1. **Q-Learning Fundamentals:**
   - Model-free, off-policy algorithm
   - Learns optimal policy from any behavior policy
   - Converges under appropriate conditions
2. **Implementation Insights:**
   - Exploration-exploitation balance crucial
   - Learning rate schedule affects convergence
   - Terminal state handling important
3. **Algorithm Variants:**
   - SARSA: On-policy, risk-averse
   - Double Q: Reduces overestimation
   - Each has specific use cases