Reinforcement Learning: Core Concepts and Algorithms

Reinforcement Learning: Introduction

Reinforcement Learning (RL) is a field of AI where agents learn to make decisions on their own by interacting with an environment. The goal is to maximize a cumulative reward signal without needing external knowledge about the problem. This document breaks down the foundational concepts and key algorithms that power RL agents.

1. The Foundations of RL

Before diving into algorithms, it's essential to understand the framework and challenges of RL.

Markov Decision Process (MDP)

The standard framework for agent-environment interaction is the Markov Decision Process. Its core principle is the Markov Property, which states that *"the future is independent of the past given the present."* This means an agent's next move only depends on its current state and chosen action, not the entire history of moves that led it there.

Value Functions: Judging the Situation

Value functions are the agent's way of estimating how good a particular state or action is in the long run. Let's use a simple 3x1 grid world as an example:

$$[S_1 \text{ (Start)}] \leftrightarrow [S_2 \text{ (Middle)}] \leftrightarrow [S_3 \text{ (Goal)}]$$

States: S_1, S_2, S_3.

Rewards: $+10$ for reaching the Goal (S_3), -1 for every other move (to encourage speed).

State-Value Function $V(s)$

$$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

Example: The value of being in S_2 is high because it's one step from the $+10$ reward. If the agent moves Right: $(-1) + 10 = 9$, so $V(S_2) \approx 9$. The value of S_1 is lower (around 8) since it is two steps away and incurs more penalties.

Action-Value Function $Q(s, a)$

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\big[G_t \bigm| S_t = s, A_t = a\big]$$

Example: At state S_2, $Q(S_2, \text{Right}) \approx 9$, while $Q(S_2, \text{Left})$ is much lower.

The Exploration vs. Exploitation Dilemma

A central challenge in RL is balancing two needs:

- **Exploitation:** Use known information to make the best possible move.
- **Exploration:** Try new moves to discover potentially better rewards.

A common solution is the Epsilon-Greedy (ε-greedy) strategy: exploit most of the time, but with a small probability ε, take a random action to explore.

2. Major RL Algorithm Families

A. Dynamic Programming (DP): Learning with a Map

DP requires a full model of the environment (a "map"). It uses the Bellman Equation to find the optimal policy by repeatedly evaluating its current strategy and then improving it.

$$V(s) = \max_{a} \sum_{s', r} p(s', r|s, a) \left[r + \gamma V(s')\right]$$

Example Walkthrough:

1. **Start:** Initialize all values $V(s) = 0$.
2. **Iteration 1:** For $V(S_2)$, moving Right leads to S_3 (value 0) with reward +10, so $V(S_2)$ becomes positive.
3. **Iteration 2:** For $V(S_1)$, moving Right leads to S_2 (positive value). Update $V(S_1)$ as step cost -1 plus $V(S_2)$.
4. Repeat until all values stabilize $\Rightarrow$ best path is revealed.

Algorithm: Policy Iteration (DP)

1. Initialize $V(s)$ arbitrarily for all states
2. Repeat until convergence:
   - **Policy Evaluation:** Update $V(s)$ using Bellman expectation equation
   - **Policy Improvement:** Update policy using:

     $$\pi(s) \leftarrow \arg\max_a \textstyle\sum_{s', r} p(s', r|s, a)[r + \gamma V(s')]$$

B. Monte Carlo (MC): Learning from Full Games

MC is model-free. The agent learns by playing many complete episodes and averaging the outcomes.

$$V(S_t) \leftarrow V(S_t) + \alpha\big[G_t - V(S_t)\big]$$

Example Walkthrough:

1. Episode: $S_1 \to S_2 \to S_1 \to S_2 \to S_3$.
2. Return from first S_1: $(-1) + (-1) + (-1) + (-1) + 10 = +6$.
3. Update $V(S_1)$ closer to +6, and similarly for other visited states.
4. After many episodes, values converge to true averages.

Monte Carlo On-Policy

Learns from the actual policy used.

Initialize $Q(s, a)$ and π arbitrarily;
for *each episode* **do**
  Generate an episode following π;
  For each state-action pair (s, a) in the episode:;
    Compute return G;
    Update $Q(s, a) \leftarrow Q(s, a) + \alpha\big[G - Q(s, a)\big]$;
    Improve policy $\pi(s) \leftarrow \arg\max_a Q(s, a)$;
end

Monte Carlo Off-Policy

Uses importance sampling to learn about a different target policy while following a behavior policy.

$$Q(s, a) \leftarrow Q(s, a) + \alpha\,\rho\,\big[G - Q(s, a)\big]$$

where ρ is the importance sampling ratio.

Initialize $Q(s, a)$, $C(s, a) = 0$;
for *each episode* **do**
  Generate episode using behavior policy b;
  Compute return G backward;
  For each (s, a) in episode:;
    $C(s, a) \leftarrow C(s, a) + W$;
    $Q(s, a) \leftarrow Q(s, a) + \frac{W}{C(s,a)}\big[G - Q(s, a)\big]$;
    $W \leftarrow W \cdot \frac{\pi(a|s)}{b(a|s)}$;
    If $a \neq \pi(s)$ then break;
end

C. Temporal-Difference (TD) Learning: Step-by-Step

TD Learning is model-free and combines ideas from both DP and MC.

$$V(S_t) \leftarrow V(S_t) + \alpha\big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\big]$$

Example Walkthrough:

1. Suppose $V(S_1) = 0$, $V(S_2) = 0$; agent moves $S_1 \to S_2$ with reward -1.
2. TD Target $= -1 + \gamma \cdot V(S_2) = -1$.
3. Update $V(S_1)$ closer to -1 immediately (no need to finish the game).

3. Key TD Algorithms in Practice

SARSA (On-Policy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$$

Example (Cliff Walking): SARSA learns based on its actual ε-greedy actions. It knows exploration might cause falling off the cliff, so it prefers a safer path.

Initialize $Q(s, a)$;
for *each episode* **do**
  Initialize s, choose a from ϵ-greedy policy;
  while *episode not ended* **do**
    Take a, observe r, s';
    Choose a' from s' using ϵ-greedy;
    $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma Q(s', a') - Q(s, a)\big]$;
    $s \leftarrow s'$, $a \leftarrow a'$;
  end
end

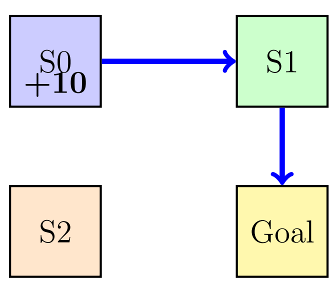

Figure 1: SARSA: Learns from actual actions taken (safer path).

Q-Learning (Off-Policy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_{a'} Q(S', a') - Q(S, A)\big]$$

Example (Cliff Walking): Q-Learning learns the value of the optimal path (shortest along the cliff), even if exploration sometimes leads to falling.

Initialize $Q(s, a)$;
for *each episode* **do**
  Initialize s;
  while *episode not ended* **do**
    Choose a using ϵ-greedy;
    Take a, observe r, s';
    $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\big]$;
    $s \leftarrow s'$;
  end
end

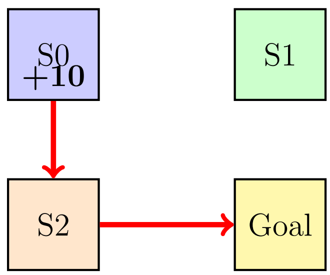

Figure 2: Q-Learning: Learns from greedy next action (optimal path).

1 Algorithm Comparison

Feature	**Dynamic Programming (DP)**	**Monte Carlo (MC)**	**SARSA**	**Q-Learning**
Model Required?	Yes (Model-Based)	No (Model-Free)	No (Model-Free)	No (Model-Free)
Learning Update	Iterates over all states	At end of episode	After every step (online)	After every step (online)
Bootstrapping?	Yes (uses estimates)	No (uses returns)	Yes (uses next state estimate)	Yes (uses next state estimate)
Policy Type	N/A (solves directly)	On-Policy	On-Policy	Off-Policy
Core Idea	Solves optimal policy using model	Learns from averaging outcomes	Learns realistic policy from own behavior	Learns optimal policy assuming greedy future

Key Equation	Bellman: $V(s) = \max_a \sum_{s',r} p(s', r \mid s, a) [r + \gamma V(s')]$	MC: $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$	SARSA: $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$	Q-Learning: $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_{a'} Q(S', a') - Q(S, A)]$