Introduction to Reinforcement Learning: First Principles and Foundations

AI Research Team

May 15, 2025

Abstract

This article provides a comprehensive introduction to reinforcement learning (RL), a paradigm of machine learning where agents learn to make decisions by interacting with an environment. We explore the fundamental principles, mathematical foundations, and key algorithms that form the backbone of reinforcement learning. The article begins with the basic concepts of RL, including the agent-environment interaction, Markov Decision Processes, and reward mechanisms. We then delve into classical algorithms such as Q-learning and policy gradients, providing mathematical formulations and intuitive explanations. This foundational understanding serves as a stepping stone for more advanced topics in reinforcement learning.

1 Introduction

Reinforcement learning (RL) stands as one of the three fundamental paradigms in machine learning, alongside supervised and unsupervised learning. Unlike these other approaches, RL is characterized by learning through interaction with an environment, rather than from a fixed dataset. This interaction-based learning mirrors how humans and animals naturally learn, making RL a compelling framework for developing intelligent systems.

As noted by [1], "Reinforcement learning is a type of machine learning where an agent learns to make decisions by taking actions in an environment to maximize a reward signal." This learning paradigm has led to remarkable achievements, from defeating world champions in complex games like Go and chess to enabling robots to learn dexterous manipulation skills.

In this article, we explore the foundational principles of reinforcement learning, providing both intuitive explanations and rigorous mathematical formulations. We begin with the core concepts that define the RL framework, then proceed to examine classical algorithms that have shaped the field.

2 Fundamental Concepts

2.1 The Agent-Environment Interface

At the heart of reinforcement learning lies the interaction between an agent and its environment. This interaction follows a cyclical pattern:

1. The agent observes the current state of the environment.
2. Based on this observation, the agent selects an action.
3. The environment transitions to a new state in response to the agent's action.
4. The environment provides a reward signal to the agent, indicating the immediate value of the state transition.

This cycle continues either indefinitely or until a terminal state is reached. Mathematically, we can formalize this interaction as follows:

At each time step t, the agent:

- Observes state $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the state space
- Takes action $a_t \in \mathcal{A}(s_t)$, where $\mathcal{A}(s_t)$ is the set of actions available in state s_t

Following the agent's action, the environment:

- Transitions to a new state $s_{t+1} \in \mathcal{S}$ according to the transition probability function $P(s_{t+1}|s_t, a_t)$
- Emits a reward $r_{t+1} \in \mathbb{R}$ according to the reward function $R(s_t, a_t, s_{t+1})$

2.2 Markov Decision Processes

The formal mathematical framework for reinforcement learning is the Markov Decision Process (MDP). An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where:

- $\mathcal{S}$ is the state space
- $\mathcal{A}$ is the action space
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function, where $P(s'|s, a)$ is the probability of transitioning to state s' given that action a was taken in state s
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, where $R(s, a, s')$ is the immediate reward received after transitioning from state s to state s' due to action a
- $\gamma \in [0, 1]$ is the discount factor, which determines the present value of future rewards

The "Markov" in MDP refers to the Markov property, which states that the future depends only on the present state and not on the sequence of events that preceded it. Formally, for an MDP:

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = P(s_{t+1}|s_t, a_t) \tag{1}$$

2.3 Policies and Value Functions

A policy π defines the agent's behavior by mapping states to actions. It can be deterministic, where $\pi(s) = a$, or stochastic, where $\pi(a|s)$ gives the probability of taking action a in state s.

Value functions estimate how good it is for an agent to be in a given state or to take a specific action in a state, under a particular policy. The state-value function $V^\pi(s)$ represents the expected return starting from state s and following policy π thereafter:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] \tag{2}$$

Similarly, the action-value function $Q^\pi(s, a)$ represents the expected return starting from state s, taking action a, and following policy π thereafter:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right] \tag{3}$$

The optimal value functions, denoted $V^*(s)$ and $Q^*(s, a)$, give the maximum expected return achievable by any policy:

$$V^*(s) = \max_\pi V^\pi(s) \quad \forall s \in \mathcal{S} \tag{4}$$

$$Q^*(s, a) = \max_\pi Q^\pi(s, a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{5}$$

2.4 The Bellman Equations

The Bellman equations are fundamental recursive relationships that characterize value functions. For a given policy π, the Bellman expectation equations are:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s, a)[R(s, a, s') + \gamma V^\pi(s')] \tag{6}$$

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} P(s'|s, a)[R(s, a, s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s', a')] \tag{7}$$

The Bellman optimality equations characterize the optimal value functions:

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s, a)[R(s, a, s') + \gamma V^*(s')] \tag{8}$$

$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a')] \tag{9}$$

3 Classical Reinforcement Learning Algorithms

3.1 Dynamic Programming

Dynamic Programming (DP) methods solve reinforcement learning problems by using the Bellman equations as update rules. These methods require complete knowledge of the MDP, including transition probabilities and reward functions.

Two fundamental DP algorithms are:

3.1.1 Policy Iteration

Policy iteration alternates between policy evaluation and policy improvement:

```
Initialize $\pi$ arbitrarily
repeat
  // Policy Evaluation
  repeat
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ do
      $v \leftarrow V(s)$
      $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
  until $\Delta < \theta$ (a small positive number)
  // Policy Improvement
  $policy\_stable \leftarrow true$
  for each $s \in \mathcal{S}$ do
    $old\_action \leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$
    if $old\_action \neq \pi(s)$ then
      $policy\_stable \leftarrow false$
    end if
  end for
until $policy\_stable$
return $\pi$
```

3.1.2 Value Iteration

Value iteration combines policy evaluation and improvement into a single update:

```
1: Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$
2: repeat
3:     $\Delta \leftarrow 0$
4:     for each $s \in \mathcal{S}$ do
5:         $v \leftarrow V(s)$
6:         $V(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$
7:         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
8:     end for
9: until $\Delta < \theta$ (a small positive number)
10: Extract policy $\pi(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$
11: return $\pi$
```

3.2 Monte Carlo Methods

Monte Carlo (MC) methods learn from complete episodes of experience without requiring knowledge of the environment's dynamics. They update value estimates based on the average returns observed after visits to states or state-action pairs.

A basic Monte Carlo algorithm for policy evaluation is:

```
1: Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$
2: Initialize Returns($s$) as an empty list for all $s \in \mathcal{S}$
3: repeat
4:     Generate an episode using policy $\pi$
5:     $G \leftarrow 0$
6:     for each step $t$ of the episode, in reverse order do
7:         $G \leftarrow \gamma G + R_{t+1}$
8:         Unless $S_t$ appears in the episode before step $t$:
9:         Append $G$ to Returns($S_t$)
10:        $V(S_t) \leftarrow$ average(Returns($S_t$))
11:    end for
12: until sufficient episodes have been generated
13: return $V$
```

3.3 Temporal Difference Learning

Temporal Difference (TD) learning combines ideas from both DP and MC methods. Like MC, TD learns from experience without requiring a model of the environment. Like DP, TD updates estimates based on other learned estimates without waiting for a final outcome.

3.3.1 TD(0)

The simplest TD algorithm, TD(0), updates the value function after each time step:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{10}$$

where α is the learning rate and $[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$ is the TD error.

3.3.2 Q-Learning

Q-learning is an off-policy TD control algorithm that directly approximates the optimal action-value function:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{11}$$

The Q-learning algorithm is:

```
1:  Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
2:  repeat
3:    Initialize $S$
4:    repeat
5:      Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
6:      Take action $A$, observe $R, S'$
7:      $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
8:      $S \leftarrow S'$
9:    until $S$ is terminal
10: until sufficient episodes have been generated
11: return $Q$
```

3.3.3 SARSA

SARSA is an on-policy TD control algorithm that updates the action-value function based on the current policy's next action:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{12}$$

The SARSA algorithm is:

4 Exploration vs. Exploitation

A fundamental challenge in reinforcement learning is balancing exploration (trying new actions to discover better strategies) and exploitation (using known good actions to maximize reward). Several approaches address this trade-off:

```
1: Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
2: repeat
3:   Initialize $S$
4:   Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
5:   repeat
6:     Take action $A$, observe $R, S'$
7:     Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
8:     $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
9:     $S \leftarrow S'$
10:    $A \leftarrow A'$
11:  until $S$ is terminal
12: until sufficient episodes have been generated
13: return $Q$
```

4.1 ϵ-greedy

The ϵ-greedy strategy selects the action with the highest estimated value with probability $1-\epsilon$, and selects a random action with probability ϵ:

$$A_t = \begin{cases} \arg\max_a Q(S_t,a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases} \tag{13}$$

4.2 Softmax Action Selection

Softmax action selection assigns a probability to each action based on its estimated value:

$$P(A_t = a|S_t) = \frac{e^{Q(S_t,a)/\tau}}{\sum_{a'} e^{Q(S_t,a')/\tau}} \tag{14}$$

where τ is a temperature parameter that controls the randomness of the selection.

4.3 Upper Confidence Bound (UCB)

UCB algorithms add an exploration bonus to actions that have been tried less frequently:

$$A_t = \arg\max_a \left[Q(S_t,a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right] \tag{15}$$

where $N_t(a)$ is the number of times action a has been selected prior to time t, and c is a parameter that controls the degree of exploration.

5 Function Approximation in Reinforcement Learning

In many real-world problems, the state and action spaces are too large for tabular methods. Function approximation techniques allow RL algorithms to generalize across similar states and actions.

5.1 Linear Function Approximation

Linear function approximation represents the value function as a linear combination of features:

$$V(s) \approx \hat{V}(s, \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s) \tag{16}$$

where $\mathbf{w}$ is a weight vector and $\mathbf{x}(s)$ is a feature vector for state s.

The weights are updated using gradient descent on the mean squared error:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[R_{t+1} + \gamma \hat{V}(S_{t+1}, \mathbf{w}) - \hat{V}(S_t, \mathbf{w})] \nabla_{\mathbf{w}} \hat{V}(S_t, \mathbf{w}) \tag{17}$$

5.2 Neural Network Function Approximation

Neural networks can approximate value functions for complex problems with high-dimensional state spaces. Deep Q-Networks (DQN) use neural networks to approximate the Q-function:

$$Q(s, a) \approx Q(s, a; \theta) \tag{18}$$

where θ represents the parameters of the neural network.

DQN introduces two key innovations:

- Experience replay: Storing and randomly sampling past experiences to break correlations in the training data
- Target networks: Using a separate network with frozen parameters for generating targets to stabilize learning

6 Policy Gradient Methods

Policy gradient methods directly optimize the policy without using a value function. They update the policy parameters in the direction of the gradient of the expected return.

The policy gradient theorem gives the gradient of the expected return with respect to the policy parameters θ:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a)] \tag{19}$$

A basic policy gradient algorithm, REINFORCE, updates the policy parameters as follows:

$$\theta \leftarrow \theta + \alpha G_t \nabla_\theta \log \pi_\theta(A_t|S_t) \tag{20}$$

where G_t is the return from time step t.

7 Conclusion

Reinforcement learning provides a powerful framework for developing agents that can learn to make decisions through interaction with an environment. This article has covered the fundamental concepts and classical algorithms that form the foundation of RL. From the basic agent-environment interface to sophisticated policy gradient methods, we have explored both the intuitive understanding and mathematical formulation of reinforcement learning.

As the field continues to advance, reinforcement learning is finding applications in diverse domains, from robotics and autonomous vehicles to healthcare and finance. The principles and algorithms discussed in this article serve as building blocks for more advanced techniques that push the boundaries of what intelligent systems can achieve.

References

[1] The IoT Academy (2023). *Reinforcement Learning.* Retrieved from https://www.theiotacademy.co/blog/reinforcement-learning/

[2] Google Research (2023). *Evolving Reinforcement Learning Algorithms.* Retrieved from https://research.google/blog/evolving-reinforcement-learning-algorithms/