Model-Free Reinforcement Learning

October 2025

Contents

Abstract

This tutorial covers the fundamental concepts and algorithms of **Model-Free Reinforcement Learning (RL)**. This approach allows an agent to learn optimal behavior directly from interacting with an environment, without needing an explicit model of its dynamics. We focus on Monte Carlo and Temporal Difference (TD) methods, culminating in the control algorithms Sarsa and Q-Learning.

1 Introduction to Model-Free Learning

Model-free RL methods enable an agent to learn **optimal behavior directly from experience** by interacting with the environment. This is crucial for real-world scenarios where the underlying dynamics (transition probabilities and reward functions) are unknown or too complex to model explicitly.

The overall learning objective is typically framed as two sub-problems:

1. **Prediction (Policy Evaluation):** Estimating the value function (v_π or q_π) for a given policy π.
2. **Control:** Finding an optimal policy (π_*) that maximizes the accumulated reward.

The generalized solution framework for control is **Generalized Policy Iteration (GPI)**, which alternates between evaluating the current policy and improving it based on the evaluations.

2 Monte Carlo (MC) Methods

Monte Carlo methods are model-free algorithms that learn **value functions by averaging sample returns from complete episodes of experience**.

2.1 Core Principles

- **Episodic Tasks Only:** MC is defined for tasks that are guaranteed to terminate. Updates only occur upon the **completion of an episode**.
- **Sample Returns (G_t):** The value of a state is estimated by averaging the **Return** (G_t), which is the total discounted reward accumulated from time step t onwards.

- **No Bootstrapping:** MC estimates are independent of the estimates of other states. They **do not** use estimated values to update other estimated values.

2.2 MC Control and Exploration

To solve the control problem, MC estimates the **Action-Value Function ($q_\pi(s,a)$)**. With these Q-values, the policy can be improved without a model by choosing the action with the highest value for a given state.

Exploration is managed using **Soft Policies**, such as the ϵ-greedy policy:

- With probability $1-\epsilon$, select the greedy (highest-value) action (exploitation).
- With probability ϵ, select a random action (exploration).

3 Temporal Difference (TD) Learning

Temporal Difference (TD) learning combines the model-free approach of Monte Carlo with the **bootstrapping** idea of Dynamic Programming.

3.1 Core Principles

- **Learning from Experience:** Like MC, TD learns directly from experience.
- **Bootstrapping:** TD **updates its value estimates based on other learned estimates**. It learns from incomplete episodes by using the current value estimate of the next state to estimate the return.

3.2 TD Prediction (TD(0))

The simplest TD method updates the value $V(S_t)$ after one step:

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

- **TD Target:** The one-step estimate of the return: $R_{t+1} + \gamma V(S_{t+1})$.
- **TD Error (δ_t):** The difference between the TD Target and the current estimate:
$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$
  This error is the engine that drives the learning.

4 Model-Free Control with TD

TD control also uses the GPI framework, focusing on action-value functions (q_π).

4.1 On-Policy TD Control: Sarsa

Sarsa is an **On-Policy** algorithm, meaning it learns the value of the policy it is currently following, including its exploratory actions (A_{t+1} is the action **actually taken**). Its name comes from the tuple of experience used for the update: **S**tate, **A**ction, **R**eward, **S**tate′, **A**ction′.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right] \tag{2}$$

4.2 Off-Policy TD Control: Q-Learning

Q-Learning is an **Off-Policy** algorithm. It learns about the **optimal (greedy) target policy** π while following a different, exploratory **behavior policy** μ. The update uses the maximum action value in the next state, regardless of the action actually chosen by the behavior policy.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right] \tag{3}$$

5 The Bias/Variance Tradeoff

MC and TD methods differ fundamentally in their updates, leading to a tradeoff:

Feature	**Monte Carlo (MC)**	**Temporal Difference (TD)**
Bias	Unbiased	Biased
Variance	High	Low
Dependency	Full episode G_t	One step of transition $(R_{t+1}, V(S_{t+1}))$

6 Unifying Methods: TD(λ)

The TD(λ) algorithm unifies TD(0) and MC methods using the parameter $\lambda \in [0, 1]$:

- $\lambda = 0$: Reverts to one-step TD(0)
- $\lambda = 1$: Equivalent to Monte Carlo

The TD(λ) update is practically implemented using **Eligibility Traces (E_t)**. This short-term memory tracks how recently a state or state-action pair was visited, allowing the single-step TD error to be efficiently **broadcast backwards** to update all previous states in proportion to their trace.