# Reinforcement Learning: An Introduction

**Zamboni Riccardo**

e-Novia, Italy, ricc.zamboni@gmail.com

**Keywords:** RL, Q-learning, Actor-Critic, Policy Gradient

## Abstract

Reinforcement Learning have become a word used by cool kids these days. Here it is a peek in its main ideas, implementations and algorithms, pros and cons and caos.

## 1 Introduction

Reinforcement Learning (RL) might sound like a mysterious word, as much as its applications have become renowned in even non-technical fields. Its framework is actually pretty simple and general, yet its simplicity stands for its power and curse. We will have a short look inside the main formalisms and concepts and deep dive into some implementations.

## 2 Reinforcement Learning in a nutshell

Let's imagine to be something able to perform actions, hereafter we will call ourselves agent. Let's be clear: here the capacity of taking an action does not assume any specific cognitive capacity in a broader sense, and here comes one of the more fascinating points of RL thinking to me. Let's assume to find ourselves in an unknown environment as well and to be able to obtain some rewards by interacting with the environment. An agent ought to take actions so as to maximize such rewards, which can be cumulated over the time. In reality, we could be a basketball-player robotic-arm aiming at getting a score out of a good shot. A positive reward would be given back from the environment once a basket is made, a negative one or nothing at all otherwise. Here, readers might notice that the generating process of the reward, and its design, could be a central part of the design of a RL algorithm, and other reward-based techniques such as LQR. They would be totally right. But Unfortunately reward design will not be part of the discussion.

The goal of RL is to learn a good strategy for the agent from trials and relative simple feedback received from the environment. With the optimal strategy, the agent is capable to actively adapt to the environment to maximize future rewards.

The agent is acting in an **environment**. How the environment reacts to certain actions is defined by a **model** which we may or may not know. The agent can stay in one of many **states** ($s \in \mathcal{S}$) of the environment, and choose to take one of many **actions** ($a \in \mathcal{A}$) to switch from one state to another, these variables coincides with the $s_t$ and $u_t$ of the LQR framework. Which state the agent will arrive in is decided by transition probabilities between states ($P$). Once an action is taken, the environment delivers a **reward** ($r \in \mathcal{R}$) as feedback.

The model defines the reward function and transition probabilities as well. We may or may not know how the model works and this differentiate two circumstances:

- **We know the model**: we will be planning with perfect information, which is often called model-based RL. When we fully know the environment, we can find the optimal solution by Dynamic Programming.
- **We do not know the model**: We will be learning with incomplete information; consequently we will be doing model-free RL or try to learn the model explicitly as part of the algorithm. LOL it's most of the times like this, my dear.

The agent's **policy** $\pi(s)$ provides the guideline on what is the action to take in a certain state with the goal to maximize the total rewards. Each state is associated with a **value** function $V(s)$ predicting the expected amount of future rewards we are able to receive in this state by acting with the corresponding policy. In other words, the value function quantifies how potentially good a state is. Both policy and some sort of value functions are what we try to learn in RL.

The interaction between the agent and the environment involves a sequence of actions and observed rewards in time, $t = 1, 2, \ldots, T$. During the process, the agent accumulates the knowledge about the environment, learns the optimal policy (hopefully), and makes decisions on which action to take next so as to efficiently learn the best policy. Let's label the state, action, and reward at time step t as $S_t$, $A_t$, and $R_t$, respectively. Thus the interaction sequence is fully described by one **episode** and the sequence might end at the terminal state $S_T$. This defines the episodic or continuing scenario, based on the existence of a terminal states. As for Model Predictive Control formulations, which can be seen as a RL approach, the existence of a closed solution might be based on whether the task is episodic or not, or on the capacity of the algorithm to deal with non-terminating tasks. An episode is then defined as:

$$S_1, A_1, R_2, S_2, A_2, \ldots, S_T$$

The main role of the model is to describe the environment. With a model, we can learn or infer how the environment would interact with the agent and provide feedback to the agent. The model has two major parts, transition probability function $P$ and reward function $R$.

Let's say when we are in state s, we decide to take action a to arrive in the next state s' and obtain reward r. This is known as one **transition** step, represented by a tuple $(s, a, s', r)$.

The transition function P records the probability of transitioning from state s to s' after taking action a while obtaining reward r. We use $\mathbb{P}$ as a symbol of "probability".

$$P(s', r|s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a] \tag{1}$$

Thus the state-transition function can be defined as a function of $\mathbb{P}(s', r|s, a)$:

$$P^a_{ss'} = \mathbb{P}(s'|s, a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a] = \sum_{r\in\mathcal{R}} P(s', r|s, a) \tag{2}$$

The reward function R predicts the next reward triggered by one action:

$$R(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} P(s', r|s, a) \tag{3}$$

The policy tells us which action to take in state s. It is a mapping from state s to action a and can be either deterministic or stochastic:

- Deterministic: $\pi(s) = a$, it is not sampled from a distribution probability.
- Stochastic: $\pi(a|s) = \mathbb{P}_\pi[A = a|S = s]$.

The value function measures the goodness of a state or how rewarding a state or an action is by a prediction of future reward. The future reward, also known as **return**, is a total sum of discounted rewards going forward. Let's compute the return $G_t$ starting from time t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{4}$$

The discounting factor $\gamma \in [0, 1]$ penalizes the rewards in the future, because:

- The future rewards may have higher uncertainty; i.e. stock market.
- The future rewards do not provide immediate benefits; i.e. As human beings, we might prefer to have fun today rather than 5 years later ;).
- Discounting provides mathematical convenience; i.e., we don't need to track future steps forever to compute return.
- We don't need to worry about the infinite loops in the state transition graph.

The **state-value** of a state s is the expected return if we are in this state at time t, $S_t = s$:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

Similarly, we define the **action-value** ("Q-value") of a state-action pair as:

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

Additionally, since we follow the target policy $\pi$, we can make use of the probability distribution over possible actions and the Q-values to recover the state-value:

$$V_\pi(s) = \sum_{a\in\mathcal{A}} Q_\pi(s, a)\pi(a|s)$$

The difference between action-value and state-value is the action **advantage** function ("A-value"):

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$$

### 2.1 Optimal Value and Policy

The optimal value function produces the maximum return:

$$V_*(s) = \max_\pi V_\pi(s), \quad Q_*(s, a) = \max_\pi Q_\pi(s, a) \tag{5}$$

The optimal policy achieves optimal value functions:

$$\pi_* = \arg\max_\pi V_\pi(s), \quad \pi_* = \arg\max_\pi Q_\pi(s, a) \tag{6}$$

And of course, we have $V_{\pi_*}(s) = V_*(s)$ and $Q_{\pi_*}(s, a) = Q_*(s, a)$.

### 2.2 Markov Decision Processes

Formally, almost all the RL problems can be framed as **Markov Decision Processes** (MDPs). All states in MDP has "Markov" property, referring to the fact that the future only depends on the current state and not the history:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t]$$

Or in other words, the future and the past are **conditionally independent** given the present, as the current state encapsulates all the statistics and information we need to decide the future.

A Markov decision process consists of a tuple of five elements $\mathcal{M} = \langle\mathcal{S}, \mathcal{A}, P, R, \gamma\rangle$ well aligned with RL problem settings (Coincidences? I do not believe this):

- $\mathcal{S}$ - a set of states;

- $\mathcal{A}$ - a set of actions;
- $P$ - transition probability function;
- $R$ - reward function;
- $\gamma$ - discounting factor for future rewards.

In an unknown environment, we do not have perfect knowledge about $P$ and $R$, in such cases these are said to be non-observable variables, or hopefully partially observable portions. Partially observable MDPs (POMDPs) are a broader case which will be further discussed in another chapter, hopefully.

### 2.3 Bellman Equations

Bellman equations refer to a set of equations that decompose the value function into the immediate reward plus the discounted future values:

$$\begin{aligned} V(s) &= \mathbb{E}[G_t|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots)|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s] \end{aligned} \tag{7}$$

And similarly for Q-value,

$$\begin{aligned} Q(s,a) &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{a\sim\pi} Q(S_{t+1}, a) \mid S_t = s, A_t = a] \end{aligned} \tag{8}$$

### 2.4 Bellman Equations in Expectation

The recursive update process can be further decomposed to be equations built on both state-value and action-value functions. As we go further in future action steps, we extend V and Q alternatively by following the policy $\pi$.

$$\begin{aligned} V_\pi(s) &= \sum_{a\in\mathcal{A}} \pi(a|s) Q_\pi(s,a) \\ Q_\pi(s,a) &= R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P^a_{ss'} V_\pi(s') \\ V_\pi(s) &= \sum_{a\in\mathcal{A}} \pi(a|s)\big(R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P^a_{ss'} V_\pi(s')\big) \\ Q_\pi(s,a) &= R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P^a_{ss'} \sum_{a'\in\mathcal{A}} \pi(a'|s') Q_\pi(s',a') \end{aligned} \tag{9}$$

### 2.5 Bellman Optimality Equations

If we are only interested in the optimal values, rather than computing the expectation following a policy, we could jump right into the maximum returns during the alternative updates without using a policy. RECAP: the optimal values $V_*$ and $Q_*$ are the best returns we can obtain, as defined [here](optimal-value-and-policy).

$$\begin{aligned} V_*(s) &= \max_{a\in\mathcal{A}} Q_*(s,a) \\ Q_*(s,a) &= R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P^a_{ss'} V_*(s') \\ V_*(s) &= \max_{a\in\mathcal{A}} \big(R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P^a_{ss'} V_*(s')\big) \\ Q_*(s,a) &= R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P^a_{ss'} \max_{a'\in\mathcal{A}} Q_*(s',a') \end{aligned} \tag{10}$$

Unsurprisingly they look very similar to Bellman equations in expectation.

If we have complete information of the environment, this turns into a planning problem, solvable by DP. Unfortunately, in most scenarios, we do not know $P^a_{ss'}$ or $R(s,a)$, so we cannot solve MDPs by directly applying Bellmen equations, but these lay the theoretical foundation for many RL algorithms.

## 3 Common Approaches

### 3.1 Dynamic Programming

When the model is fully known, following Bellman equations, we can use Dynamic Programming.

#### 3.1.1 Policy Evaluation

Policy Evaluation is to compute the state-value $V_\pi$ for a given policy $\pi$:

$$\begin{aligned} V_{t+1}(s) &= \mathbb{E}_\pi[r + \gamma V_t(s')|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} P(s',r|s,a)(r + \gamma V_t(s')) \end{aligned} \tag{11}$$

#### 3.1.2 Policy Improvement

Based on the value functions, Policy Improvement generates a better policy $\pi' \geq \pi$ by acting greedily.

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s, A_t = a] \\ &= \sum_{s',r} P(s',r|s,a)(r + \gamma V_\pi(s')) \end{aligned} \tag{12}$$

#### 3.1.3 Policy Iteration

The **Generalized Policy Iteration (GPI)** algorithm refers to an iterative procedure to improve the policy when combining policy evaluation and improvement.

$$\pi_0 \xrightarrow{\text{evaluation}} V_{\pi_0} \xrightarrow{\text{improve}} \pi_1 \xrightarrow{\text{evaluation}} V_{\pi_1} \xrightarrow{\text{improve}} \pi_2 \xrightarrow{\text{evaluation}} \dots \xrightarrow{\text{improve}}$$

In GPI, the value function is approximated repeatedly to be closer to the true value of the current policy and in the meantime, the policy is improved repeatedly to approach optimality.

This policy iteration process works and always converges to the optimality, but why this is the case?

Say, we have a policy $\pi$ and then generate an improved version $\pi'$ by greedily taking actions, $\pi'(s) = \arg\max_{a \in \mathcal{A}} Q_\pi(s, a)$. The value of this improved $\pi'$ is guaranteed to be better because:

$$\begin{aligned} Q_\pi(s, \pi'(s)) &= Q_\pi(s, \arg\max_{a \in \mathcal{A}} Q_\pi(s, a)) \\ &= \max_{a \in \mathcal{A}} Q_\pi(s, a) \geq Q_\pi(s, \pi(s)) = V_\pi(s) \end{aligned} \tag{13}$$

### 3.2 Monte-Carlo Methods

First, let's recall that $V(s) = \mathbb{E}[G_t | S_t = s]$. Monte-Carlo (MC) methods uses a simple idea: It learns from episodes of raw experience without modeling the environmental dynamics and computes the observed mean return as an approximation of the expected return. To compute the empirical return $G_t$, MC methods need to learn from **complete** episodes $S_1, A_1, R_2, \ldots, S_T$ to compute $G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$ and all the episodes must eventually terminate.

The empirical mean return for state s is:

$$V(s) = \frac{\sum_{t=1}^{T} \mathbb{1}[S_t = s] G_t}{\sum_{t=1}^{T} \mathbb{1}[S_t = s]}$$

where $\mathbb{1}[S_t = s]$ is a binary indicator function. We may count the visit of state s every time so that there could exist multiple visits of one state in one episode ("every-visit"), or only count it the first time we encounter a state in one episode ("first-visit"). This way of approximation can be easily extended to action-value functions by counting (s, a) pair.

$$Q(s, a) = \frac{\sum_{t=1}^{T} \mathbb{1}[S_t = s, A_t = a] G_t}{\sum_{t=1}^{T} \mathbb{1}[S_t = s, A_t = a]}$$

To learn the optimal policy by MC, we iterate it by following a similar idea to [GPI](policy-iteration).

1. Improve the policy greedily with respect to the current value function: $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$.
2. Generate a new episode with the new policy $\pi$ (i.e. using algorithms like epsilon-greedy helps us balance between exploitation and exploration.)
3. Estimate Q using the new episode: $q_\pi(s, a) = \frac{\sum_{t=1}^{T} \left( \mathbb{1}[S_t = s, A_t = a] \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \right)}{\sum_{t=1}^{T} \mathbb{1}[S_t = s, A_t = a]}$

### 3.3 Temporal-Difference Learning

Similar to Monte-Carlo methods, Temporal-Difference (TD) Learning is model-free and learns from episodes of experience. However, TD learning can learn from **incomplete** episodes and hence we don't need to track the episode up to termination. TD learning is so important that Sutton & Barto (2017) in their RL book describes it as "one idea … central and novel to reinforcement learning".