# Lecture 3b: Taxonomy of RL Algorithms, Q-Learning

Qi Zhang

Last Updated: October 2025

## 1 Taxonomy of RL Algorithms

In this note, we begin studying RL algorithms. We will focus on the infinite-horizon, discounted-reward setting where the MDP is specified by a tuple $M = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$ with initial state distribution $d_0 \in \Delta(\mathcal{S})$. The RL agent, or learner, does not know $(P, R)$ a priori and interacts with the MDP following Protocol 1, where the role of the learner's RL algorithm is highlighted in green.

**Protocol 1** RL interaction (infinite-horizon)

```
1: learner initializes and will maintain its internal state;
2: repeat over episodes:
3:     learner observes initial state $s_0 \sim d_0$ of the current episode;
4:     for $h = 1, \dots, H$ until termination or truncation do    ▷ truncation: $s_{H+1}$ is not terminal
5:         learner chooses action $a_h$ based on its internal state;
6:         learner takes $a_h$ and observes next state $s_{h+1} \sim P(s_h, a_h)$ and reward $r_h = R(s_h, a_h)$;
7:         learner updates its internal state;
8:     end for
9: until some stopping criterion is met
10: learner outputs a policy $\widehat{\pi}$ based on its internal state;
```

Crucially, the learner maintains its *internal state* that 1) gets updated based on the newest transition (line 7) and 2) determines the very next action to take (line 5). This way, the learner is fully adaptive: in general, it chooses the current action based on all previous transitions. The learner's internal state can include some or all of the following components:

- Value estimates: $V : \mathcal{S} \to \mathbb{R}$, $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- Policy/actor: $\pi : \mathcal{S} \to \Delta(\mathcal{A})$
- Replay buffer: $\mathcal{D} = (s, a, r, s')$ storing previous transitions
- Model estimate: $(\widehat{P}, \widehat{R})$ that estimates the ground-truth $(P, R)$ based on previous transitions

RL algorithms are often characterized by what components above are (not) included in the learner's internal state. Table 1 provides a (simplified) taxonomy, although it is usually of little importance to memorize such a table.

## 2 Tabular Q-Learning

We now introduce our first RL algorithm, *Q-Learning*. Its key idea is that, in order to find an optimal policy (by the end of Protocol 1 at line 10), it suffices to find out the optimal Q-value $Q^*$ because an optimal policy can be then extracted by acting greedily with respect to it , i.e., $\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a)$. In Q-learning, the learner's internal state includes only a Q-value

Table 1: A Taxonomy of RL Algorithms.

| | Value estimates $V, Q$ | Policy $\pi$ | Model estimate $(\widehat{P}, \widehat{R})$ |
|---|---|---|---|
| Model-free | - | - | Y |
| Model-based | - | - | Y |
| Value-based | Y | N | N |
| Policy-based | N | Y | N |
| Actor-critic | Y | Y | N |
| Tabular | Represented by tables, one entry for each $(s)$ or $(s,a)$ | | |
| Function Approximation | Represented by generic functions | | |

estimate $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and therefore it is a value-based (and also model-free) RL algorithm. The goal of Q-learning is to update $Q$ such that it eventually approximates $Q^*$ well and recommends the outputs the policy as $\widehat{\pi}(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$. *Tabular* Q-learning represents $Q$ as a table/vector with $|\mathcal{S} \times \mathcal{A}|$ entries, one entry per $(s,a)$ pair, and, upon a transition $(s_t, a_t, r_t, s_{t+1})$, updates the entry of $(s_t, a_t)$ as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \left( r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t) \right). \tag{1}$$

Here, the index $t$ is the total number of transitions the learner has experienced in Protocol 1, which keeps increasing across episodes; $\alpha_t \in (0,1)$ controls how aggressive the update is, which in general depends on $t$.

To understand why update (1) makes sense, recall Q-value iteration: starting with an arbitrary $Q \in \mathbb{R}^{|\mathcal{S} \times \mathcal{A}|}$, update it iteratively as $Q \leftarrow \mathcal{T} Q$ where *Bellman optimality operator* $\mathcal{T} : \mathbb{R}^{|\mathcal{S} \times \mathcal{A}|} \to \mathbb{R}^{|\mathcal{S} \times \mathcal{A}|}$ is defined as:

$$(\mathcal{T}Q)(s,a) := R(s,a) + \gamma \, \mathbb{E}_{s' \sim P(s,a)} \left[ \max_{a \in \mathcal{A}} Q(s', a) \right] \tag{2}$$

$$\approx r + \gamma \max_{a \in \mathcal{A}} Q(s', a) \tag{3}$$

where the $\approx$ approximates the expectation by a single sample of transition $(s, a, r, s')$ with $r = R(s,a)$ and $s' \sim P(s,a)$. In the planning setting, we can perform the Q-value iteration and we have the convergence of $\mathcal{T}^k Q \to Q^*$ as $k \in \infty$, where $\mathcal{T}^k$ iteratively applies $\mathcal{T}$ for $k$ times. In the RL setting, we cannot perform a full value iteration update $(\mathcal{T}Q)$ because $(P, R)$ is unknown. Instead, upon on a new transition $(s, a, r, s')$, we update $Q(s,a)$ to be closer to the one-sample target (3):

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha \left( r + \gamma \max_{a \in \mathcal{A}} Q(s', a) \right) = Q(s,a) + \alpha \left( r + \gamma \max_{a \in \mathcal{A}} Q(s', a) - Q(s,a) \right)$$

where $\alpha \in (0,1)$ controls how aggressively $Q(s,a)$ is updated to target (3). Setting $(s, a, r, s') = (s_t, a_t, r_t, s_{t+1})$ and $\alpha = \alpha_t$ in the update above recovers (1).

It turns out that, if the following conditions hold, we have the convergence guarantee of $Q \to Q^*$ as $t \to \infty$:

- Every $(s,a)$ pair is visited infinitely often.

- $\alpha_t$ decreases but not too quickly.

A formal description of the first condition is $\lim_{t\to\infty} N_t(s,a) = \infty$ for any $(s,a)$, where $N_t(s,a)$ is the number of times $(s,a)$ is visited by $t$. This means the learner must do *exploration* when choosing actions at line 5 in Protocol 1. For example, one can do $\epsilon$-greedy w.r.t. the current $Q$. A formal description of the second condition is $\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty$ (e.g., $\alpha_t = \frac{1}{t}$ satisfies this).

## 3 Deep Q-Network

The obvious issue of tabular Q-learning is the difficulty of scaling to large state/action spaces. *Function approximation* solves this issue by representing the Q-value estimates with a generic function. The intuition is that, by updating the Q-value on a specific state-action pair, the changes made to the function will also influence other state-action pairs.

**Protocol 2** DQN algorithm

```
1: learner initializes parameter $\theta$ of the Q network and sets replay buffer $\mathcal{D} = \{\}$ and target
   network parameter $\bar{\theta} = \theta$;
2: repeat over episodes:
3:     learner observes initial state $s \sim d_0$ of the current episode;
4:     for timesteps within the episode do
5:         learner chooses action $a$ that is $\epsilon$-greedy w.r.t. $Q_\theta(s, \cdot)$;  ▷ $\epsilon$ usually decreases over time
6:         learner takes $a$ and observes next state $s'$ and reward $r$;
7:         learner add $(s, a, r, s')$ to replay buffer $\mathcal{D}$;
8:         learner samples a batch of $N$ transitions from $\mathcal{D}$;
9:         learner takes a gradient step minimizing $L(\theta)$;
10:        learner sets $\bar{\theta} \leftarrow \theta$ every $c$ updates of $\theta$;       ▷ updates target network periodically
11:    end for
12: until some stopping criterion is met
13: learner outputs a policy $\widehat{\pi}$ as the greedy policy w.r.t. $Q_\theta$;
```

A representative work is *Deep Q-Network* (DQN), where the Q-value estimates is represented using a neural network, $Q_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ where $\theta$ is the neural network's parameters. Although people tried neural networks for Q-learning before DQN, those prior efforts updated the parameters on the most recent transition, in a similar fashion as in tabular Q-learning, and observed limited success due to DQN improved the training (i.e., parameters updating) of the networks with the following innovations:

- It stores the past transitions in a memory, often referred to as *replay buffer*, $\mathcal{D} = \{(s, a, r, s')\}$.
- On a given transition $(s, a, r, s')$, it forms target (3) using another network $Q_{\bar{\theta}}$, which is of the same structure as $Q_\theta$ but with its parameter $\bar{\theta}$ updated more slowly than $\theta$.
- To update $\theta$, it samples a batch of transitions from the replay buffer and update $\theta$ with a single gradient step to reduce the Q-value estimates with the formed target values.

This leads to the following loss function for network $Q_\theta$:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(Q_\theta(s_i, a_i) - \left(r_i + \gamma \max_{a\in\mathcal{A}} Q_{\bar{\theta}}(s_i', a)\right)\right)^2 \quad \text{where } (s_i, a_i, r_i, s_i') \sim \mathcal{D} \text{ for } i = 1, \ldots, N.$$