Deep Reinforcement Learning Notes

Yixuan Wang & Zhixiao Xiong

Weiyang College, Tsinghua University
{wangyixu21,xiong-zx21}@mails.tsinghua.edu.cn

July 17, 2024

Chapter 1 Reinforcement Learning Basics

Outline

In this chapter, we will introduce some important prerequisite knowledge and basics ideas in reinforcement learning. Based on these ideas, we will present several specific reinforcement learning algorithms, which will later serve as an important foundation to understand *deep* reinforcement learning .

In section1.1, we introduce some basic concepts in sequential decision-making model, including the *state-value function* and the *Bellman equation*. Based on this, we can do *iterative value evaluation*, which later give rise to *value iteration*.

In section 1.2, we add actions to MRP and therefore formulates a policy. Then we want to evaluate how good a policy is, so we have *policy evaluation*. We also want to polish up the policy, so we have *policy improvement* and *policy iteration*.

Up to now, the model (typically the reward functions and the transition probabilities) are assumed known for decision making. However, in real life we have to figure out the model on our own. Therefore we introduce several estimation methods in section 1.3, including *Monte-Carlo evaluation* which sample the entire episode, and *temporal difference evaluation* which updates on a per-action basis, with the idea of *dragging the value toward what Bellman equation requires*

But both TD and MC deal with value function and we still need the transition functions in order to do policy improvement. We solve this problem by estimating Q function which helps us get rid of the transition probability and goes directly into the action part.

1.1 Markov Reward Process (MRP)

A Markov Reward Process (MRP) is a sequential decision-making model that describes the interaction between an agent and an environment in a series of states, where the agent receives rewards based on the states it transitions through. In an MRP, the next state of the agent solely depends on the current state and there is no notion of actions or decision-making. It addresses questions like "What is the value of being in a certain state?"

A typical MRP model consists of the following elements:

- State $\mathcal{S}$
- Dynamics/transition model p
- Reward function r (return at time t is R_t)
- Discount factor $\gamma \in [0, 1]$
- Return

$$G_t = r_t + \gamma r_{t+1} + \cdots + \gamma^{H-1} r_{t+H-1}$$

Definition (State-value function). $V(s)$ is the expected return starting from state s:

$$V(s) = \mathbb{E}\left[G_t | s_t = s\right] = \mathbb{E}\left[\sum_{k=0}^{H-1} \gamma^k r_{t+k} | s_t = s\right] \tag{1.1.1}$$

If $H = \infty$, then $V(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s\right]$.

Theorem 1.1. The MRP value function V satisfies the **Bellman equation**:

$$V(s) = r(s) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s)V(s') \tag{1.1.2}$$

And the following equation must hold for a policy to be optimal:

$$V^{\star}(s) = \max_{a \in A(s)} \sum_{s' \in S} P_a\left(s' \mid s\right)\left[r\left(s, a, s'\right) + \gamma V\left(s'\right)\right] \tag{1.1.3}$$

We can break this equation down:

$$V^{\star}(s) = \overbrace{\max_{a \in A(s)}}^{\text{best action from } s} \overbrace{\underbrace{\sum}_{\text{for every state}} P_a\left(s' \mid s\right)\left[\underbrace{r\left(s, a, s'\right)}_{\text{immediate reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{V\left(s'\right)}_{\text{value of } s'}\right]}^{\text{expected reward of executing action } a \text{ in state } s} \tag{1.1.4}$$

First, we calculate the expected reward for each action. The reward of an action is: the sum of the immediate reward for all states possibly resulting from that action plus the discounted future reward of those states.Second, the optimal value $V^{\star}(s)$ is the value of the action with the maximum the expected reward.

If states are **finite**, then in matrix form:

$$\begin{aligned}
\begin{pmatrix} V(s_1) \\ \vdots \\ V(s_N) \end{pmatrix} &= \begin{pmatrix} r(s_1) \\ \vdots \\ r(s_N) \end{pmatrix} + \gamma \begin{pmatrix} p(s_1|s_1) & \cdots & p(s_N|s_1) \\ p(s_1|s_2) & \cdots & p(s_N|s_2) \\ \vdots & \ddots & \vdots \\ p(s_1|s_N) & \cdots & p(s_N|s_N) \end{pmatrix} \begin{pmatrix} V(s_1) \\ \vdots \\ V(s_N) \end{pmatrix}, \\
V &= R + \gamma PV, \\
V - \gamma PV &= R, \\
V &= (I - \gamma P)^{-1} R.
\end{aligned} \tag{1.1.5}$$

Iterative algorithm for computing value of a MRP is shown in Alg. 1. Computational complexity is $O(|\mathcal{S}|^2)$ for each iteration.

Algorithm 1 Iterative value evaluation for MRP

Require: Initialize $V(s) = 0, \forall s \in \mathcal{S}$

1: **repeat**
2: **for** $s \in \mathcal{S}$ **do**
3: $V_k(s) = r(s) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s) V_{k-1}(s')$
4: **until** Convergence

1.2 Markov Decision Process (MDP)

A Markov Decision Process (MDP) extends the concept of an MRP by introducing the notion of actions and decision-making. It is a mathematical framework that models sequential decision-making problems in a controlled environment. In an MDP, the agent takes actions at each state, which influences the transition to the next state, the rewards received, and ultimately affects the agent's long-term goals. It addresses questions like "What is the optimal policy for the agent to follow to maximize its long-term rewards?"

Similar to MRP, a MDP model have the following elements:

- State space $\mathcal{S}$
- Action space $\mathcal{A}$
- Dynamics/transition model p

$$p(s'|s,a) = \Pr(s_{t+1} = s'|s_t = s, a_t = a) \tag{1.2.1}$$

- Reward function r

$$r(s_t = s, a_t = a) = \mathbb{E}[r_t|s_t = s, a_t = a] \tag{1.2.2}$$

- Discount factor $\gamma \in [0,1]$
- Policy π

$$\pi(a|s) = \Pr(a_t = a|s_t = s) \tag{1.2.3}$$

MDP + π $\implies$ MRP. In other words, MRP is $(\mathcal{S}, R^\pi, P^\pi, \gamma)$ where

$$\begin{aligned} r^\pi(s) &= \sum_{a\in\mathcal{A}} \pi(a|s) r(s,a), \\ p^\pi(s'|s) &= \sum_{a\in\mathcal{A}} \pi(a|s) p(s'|s,a). \end{aligned} \tag{1.2.4}$$

1.2.1 Policy Evaluation

How to evaluate a policy π? The following iterative algorithm in Alg. 2 provides a solution. The computational complexity is $O(|\mathcal{A}||\mathcal{S}|^2)$for each iteration. The value function V_k^π converges to V^π as $k \to \infty$, which is the value function of policy π.

Algorithm 2 Iterative Policy Evaluation

Require: Initialize $V_0^\pi(s) = 0, \forall s \in \mathcal{S}$
1: **repeat**
2: **for** $s \in \mathcal{S}$ **do**
3: Compute $V_k^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \left[R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a) V_{k-1}^\pi(s')\right]$
4: **until** V_k^π converge
5: **return** Value function V^π

Note. When the policy π is deterministic, the above algorithm can be simplified as

$$V_k^\pi(s) = r(s,\pi(s)) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,\pi(s)) V_{k-1}^\pi(s'). \tag{1.2.5}$$

Question. Can we initialize the value function $V_0^\pi(s)$ to be anything other than 0? Will the V_k^π converge to V^π?

Optimal Policy

- The optimal policy
$$\pi^*(s) = \arg\max_\pi V^\pi(s)$$
- There exists a **unique** optimal value function
- The optimal policy for a **infinite** MDP is
  - **Deterministic**: given any state, the optimal policy prescribes exactly one action to perform, rather than a probability distribution over multiple actions.
  - **Stationary**:the optimal policy remains the same over time and is not dependent on the current time step.
  - **Not necessarily unique**: there may be multiple optimal policies that achieve the same maximum expected cumulative reward

Remark. Number of deterministic policies is $|\mathcal{A}|^{|\mathcal{S}|}$.

1.2.2 Partial Observable MDP (POMDP)

Things could get more complicated when we don't have access to the full state s_t at time t. Instead, we have access to an observation o_t at time t, which is a function of the state s_t at time t.

A POMDP has the following components:

- State space $\mathcal{S}$
- Action space $\mathcal{A}$
- Observation space $\mathcal{O}$
- Dynamics/transition model p
- Emission probability $\mathcal{E}$
$$\mathcal{E}(o|s) = \Pr(o_t = o|s_t = s) \tag{1.2.6}$$
- Reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- Discount factor $\gamma \in [0, 1]$

Note. For the sake of simplicity, we assume that the model is fully observable in the following sections.

1.2.3 Policy Iteration

Definition (State-Action Value). $Q^\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π.

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi\left[G_t | s_t = s, a_t = a\right] \\ &= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^\pi(s') \\ &= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s',a') \end{aligned} \tag{1.2.7}$$

Note that $Q^\pi(s, a)$ differs from $V^\pi(s)$ in that it specify a perticular action a to take at state s, so the following equations would hold:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q^\pi(s,a) \tag{1.2.8}$$

$$V^\star(s) = \max_{a \in A(s)} Q(s,a) \tag{1.2.9}$$

The following algorithm in Alg. 3 provides an iterative method to find the optimal policy.

The idea of policy evaluation is to estimate how good the policy is (i.e., the value function V^π of a policy π). And the idea of policy improvement is to find a better policy π' based on the value function V^π.

Algorithm 3 Policy Iteration

Require: Initialize π_0 arbitrarily
1: **repeat**
2: Policy evaluation: compute $V^{\pi_k}(s)$ for all $s \in \mathcal{S}$
3: Policy improvement: $\pi_{k+1}(s) = \arg\max_a Q^{\pi_k}(s,a)$ for all $s \in \mathcal{S}$
4: $k \longleftarrow k+1$
5: **until** $\|\pi_k - \pi_{k-1}\|_1 = 0$
6: **return** Optimal policy π^*

In the policy evaluation step, what we do is essentially iterate over k until the value for each state converges. Note that this is quite similar to value iteration, only without taking the arg max of a:

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T\left(s, \pi_i(s), s'\right)\left[R\left(s, \pi_i(s), s'\right) + \gamma V_k^{\pi_i}\left(s'\right)\right] \tag{1.2.10}$$

In the policy improvement step, we perform a one-step look-ahead to get a better policy using **policy extraction** (select the action with the highest expected reward):

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T\left(s, a, s'\right)\left[R\left(s, a, s'\right) + \gamma V^{\pi_i}\left(s'\right)\right] \tag{1.2.11}$$

Remark. Evaluating a policy π is time-consuming, which has the complexity of $O(|\mathcal{A}||\mathcal{S}|^2)$ for each iteration. This have to be done for several iterations until the value function of policy V^π converges.

Policy improvement is also time-consuming, which has the complexity of $O(|\mathcal{A}||\mathcal{S}|^2)$ for each iteration. Why is this? Because evaluating each $Q(s,a)$ requires computing for $O(|\mathcal{S}||\mathcal{A}|)$ times, and to to improve the policy π, we need to compute $Q^\pi(s,a)$ for each state s.

When the policy doesn't change for any state, the policy is optimal.

Definition. A policy π_1 is better than a policy π_2 if

$$V^{\pi_1} \geq V^{\pi_2} : V^{\pi_1}(s) \geq V^{\pi_2}(s), \forall s \in \mathcal{S}. \tag{1.2.12}$$

Theorem 1.2 (Monotonic Improvement in Policy).

$$V^{\pi_{i+1}} \geq V^{\pi_i}, \forall i \geq 0. \tag{1.2.13}$$

with strict inequality unless π_i is suboptimal, where π_{i+1} is the policy obtained by Alg. 3 from π_i.

Proof

$$\begin{aligned} V^{\pi_i}(s) &\leq \max_a Q^{\pi_i}(s,a) \\ &= \max_a \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^{\pi_i}(s') \right] \\ &= r(s, \pi_{i+1}(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, \pi_{i+1}(s)) V^{\pi_i}(s') \\ &\leq r(s, \pi_{i+1}(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, \pi_{i+1}(s)) V^{\pi_{i+1}}(s) \\ &= V^{\pi_{i+1}}(s). \end{aligned} \tag{1.2.14}$$

Note. The value function V^{π_i} is a sequence of monotonically increasing functions which converges to V^{π^*}, the optimal value.

1.2.4 Value Iteration

Policy iteration computes infinite horizon value of a policy and then improves the policy. Value iteration computes the optimal value function (i.e., the value function under the optimal policy π^*) directly by

- Maintain optimal value of starting in a state s if have a finite number of steps k left in the episode
- Iterate to consider longer and longer episodes

Algorithm 4 Value Iteration

Require: Initialize $V_0(s) = 0, \forall s \in \mathcal{S}$
1: **repeat**
2: **for** $s \in \mathcal{S}$ **do**
3: $V_k(s) = \max_a [r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_{k-1}(s')]$
4: **until** Convergence, e.g., $\|V_{k+1} - V_k\| < \epsilon$

Note that in value iteration, we perform a max over action space. That is to say we always maintain the state value to be optimal. However in the evaluation step in policy iteration, we employ the transition probability instead of the argmax, where the optimal assumption does not hold.

Remark. For each iteration in Alg. 4, we need to compute the value of each state s, which has the complexity of $O(|\mathcal{A}||\mathcal{S}|^2)$. This seems to be much more efficient than policy iteration. However, in reality, the policy iteration usually converges much faster, which explains why both methods are used in practice.

1.2.5 Bellman Backup Operator

In this section, we will introduce the **Bellman backup operator**, which contains some math and hence is quite elegant.

Theorem 1.3 (Bellman Equation). Value function of a policy π satisfies the Bellman equation

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, \pi(s)) V^\pi(s'). \tag{1.2.15}$$

Definition (Bellman Backup Operator). The Bellman backup operator for policy π is defined as

$$BV(s) = \max_a [r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s')]. \tag{1.2.16}$$

It is easy to seet that the Bellman backup operator

- applies to a value function
- returns a new value function
- improves the value if possible

We can denote the Bellman operator applied to value function at all states as BV. Furthermore, we can define the Bellman operator for policy π as B_π.

$$\begin{aligned} B^\pi V(s) &= r^\pi(s) + \gamma \sum_{s' \in \mathcal{S}} p^\pi(s'|s) V(s') \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s') \right] \end{aligned} \tag{1.2.17}$$

Then the policy evaluation is equivalent to

$$V^\pi = B^\pi B^\pi \dots B^\pi V.$$

We might want to ask, then, does value iteration converge in theory? Does it converge to a unique optimal value function? Fortunately, the answer is yes and is guaranteed by the fact that the Bellman equation is a **contraction mapping**.

Definition (Contraction Mapping). Let $T : M \to N$ be a mapping. T is a contraction if there exists a constant $\gamma \in [0, 1)$ such that for all $x, y \in M$, the following holds:

$$\|T(x) - T(y)\|_M \leq \gamma \|x - y\|_N$$

where $\|\cdot\|_M$ and $\|\cdot\|_N$ denote proper norms on M and N, respectively.

Theorem 1.4. The Bellman operator $\mathcal{B}$ is a contraction mapping for value function V w.r.t. the ∞-norm.

Proof. Let $|\cdot|$ denote the absolute value and $\|\cdot\|$ the infinity norm. $\forall s \in \mathcal{S}$ and value functions V_1, V_2:

$$\begin{aligned}
|\mathcal{B}V_1(s) - \mathcal{B}V_2(s)| &= \left| \left(\max_a \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_1(s') \right] \right) - \left(\max_a \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_2(s') \right] \right) \right| \\
&\leq \max_a \left| \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_1(s') \right] - \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_2(s') \right] \right| \\
&= \gamma \left| \max_a \sum_{s' \in \mathcal{S}} p(s'|s,a) |V_1(s') - V_2(s')| \right| \\
&\leq \gamma \max_{s'} |V_1(s') - V_2(s')| \cdot \max_a \sum_{s' \in \mathcal{S}} p(s'|s,a) \\
&= \gamma \max_{s'} |V_1(s') - V_2(s')| \cdot 1 \\
&= \gamma \max_{s'} |V_1(s') - V_2(s')| \\
&= \gamma \|V_1 - V_2\|
\end{aligned}$$

Therefore, with $0 < \gamma < 1$, we have:

$$\|\mathcal{B}V_1 - \mathcal{B}V_2\| \leq \gamma \|V_1 - V_2\|$$

which proves that $\mathcal{B}$ is a contraction mapping. □

1.3 Model-Free Value Methods

In previous sections we introduced MRP and MDP. The key idea of MRP is to understand the value of states in terms of the expected cumulative rewards the agent can achieve starting from those states. MDP builds on this idea by introducing actions and characterizing the state-action transitions and rewards to determine the optimal policy.

However in previous discussions, we assume that we have access to the dynamics P and reward function R. (Recall that we need to know dynamics in order to do eigher value or policy iteration.)

In this section, let's consider a more general yet more difficult case: we don't have access to the dynamics P and reward function R. We try to figure these out by estimation methods.

1.3.1 Monte Carlo (MC) Policy Evaluation

Monte Carlo policy evaluation is a rather straightforward estimation method that rely merely on experience gained through interaction with the environment.

In Monte-Carlo Policy Evaluation, an agent interacts with the environment, following a specific policy, and collects a sequence of state-action-reward trajectories. The value of a state is then estimated by averaging the returns obtained from all visits to that state. This process is repeated over multiple episodes, and the state values are updated after each episode.

The algorithm is also quite simple: