Chapter 4

Value Iteration and Policy Iteration

With the preparation in the previous chapters, we are ready now to present the first reinforcement learning (RL) algorithms: *value iteration* and *policy iteration*. They have a common name called *dynamic programming*. The algorithms introduced in this chapter are *model-based* since they require knowing the exact probability model of the environment. They are the simplest RL algorithms. Understanding them is the foundation for understanding *model-free* RL algorithms that will be introduced later in the book.

This chapter contains three parts. The first part introduces the value iteration algorithm. This algorithm is exactly the algorithm suggested by the contraction mapping theorem to solve the Bellman optimality equation. Here, we introduce the implementation details of this algorithm. The second part introduces the policy iteration algorithm. This algorithm is fundamental in RL. Its idea is widely used by many model-free RL algorithms. The third part introduces truncated policy iteration, which is an algorithm that can unify value iteration and policy iteration. We will show that value iteration and policy iteration are two special cases of truncated policy iteration.

4.1 Value iteration

This section introduces the value iteration algorithm, one classic model-based RL algorithm. This algorithm is exactly the algorithm suggested by the contraction mapping theorem to solve the Bellman optimality equation as introduced in the last chapter (Theorem 3.3). For a quick reference, it is

$$v_{k+1} = \max_{\pi}(r_\pi + \gamma P_\pi v_k), \quad k = 0, 1, 2, \ldots$$

It is guaranteed by Theorem 3.3 that v_k will converge to the optimal state value and π_k will converge to an optimal policy. This algorithm is iterative. In order to implement it, we can further decompose every iteration into two steps.

– The first step in every iteration is called *policy update*. Mathematically, it is to find a

policy solving the following optimization problem:

$$\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_k).$$

– The second step is called *value update*. Mathematically, it is to substitute π_{k+1} and do the following operation:

$$v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k.$$

The above description of the value iteration algorithm relies on the matrix-vector form of the Bellman optimality equation. However, in order to implement the algorithm, we need to introduce the elementwise form of the algorithm. The matrix-vector form is useful to understand the core idea of the algorithm, whereas the elementwise form is good for explaining the implementation details.

4.1.1 Elementwise form and implementation

We next show how to implement the value iteration algorithm in detail.

– First, the elementwise form of the *policy update step* $\pi_{k+1} = \arg\max_\pi(r_\pi + \gamma P_\pi v_k)$ is

$$\pi_{k+1}(s) = \arg\max_{\pi} \sum_a \pi(a|s) \underbrace{\left(\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_k(s') \right)}_{q_k(s,a)}, \quad s \in \mathcal{S}.$$

The optimal policy solving the above optimization problem is

$$\pi_{k+1}(a|s) = \begin{cases} 1 & a = a_k^*(s), \\ 0 & a \neq a_k^*(s). \end{cases}$$

where $a_k^*(s) = \arg\max_a q_k(a,s)$. If $a_k^*(s) = \arg\max_a q_k(a,s)$ has multiple solutions, we can simply select any of them. Since the new policy π_{k+1} selects an action with the greatest value of $q_k(s,a)$, such a policy is greedy.

– Second, the elementwise form of the *value update step* $v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k$ is

$$v_{k+1}(s) = \sum_a \pi_{k+1}(a|s) \underbrace{\left(\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_k(s') \right)}_{q_k(s,a)}, \quad s \in \mathcal{S}.$$

Since π_{k+1} is a greedy policy, the above equation is simply

$$v_{k+1}(s) = \max_a q_k(a,s).$$

The details of the implementation are summarized in the pseudocode.

Pseudocode: Value iteration algorithm

Initialization: The probability model $p(r|s,a)$ and $p(s'|s,a)$ for all (s,a) are known. Initial guess v_0.
Aim: Search for the optimal state value and an optimal policy solving the Bellman optimality equation.

While the state value has not converged, for the kth iteration, do
  For every state $s \in \mathcal{S}$, do
    For every action $a \in \mathcal{A}(s)$, do
      q-value: $q_k(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_k(s')$
    Maximum action value: $a_k^*(s) = \arg\max_a q_k(a,s)$
    Policy update: $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise
    Value update: $v_{k+1}(s) = \max_a q_k(a,s)$

One problem of value iteration that may confuse beginners is whether v_k is a state value. The answer is no although v_k converges to the optimal state value eventually. That is because v_k is not guaranteed to satisfy any Bellman equation. For example, we do not have $v_k = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k$ or $v_k = r_{\pi_k} + \gamma P_{\pi_k} v_k$ in general. When representing a state value, we will explicitly indicate the corresponding policy in the subscript. For example, v_{π_k} is the state value of policy π_k. Instead, v_k is not a state value and simply the kth intermediate value of an iterative process. Since v_k is not a state value, q_k is not an action value either. This might be one of the most confusing problems for beginners studying this algorithm. It is easier if we simply treat them as intermediate variables that emerge when solving the Bellman optimality equation.

4.1.2 Illustrative examples

We next present an example to illustrate the implementation details of the value iteration algorithm.

This example is a two-by-two grid with one forbidden area and the target area is s_4. The reward setting is $r_{\text{boundary}} = r_{\text{forbidden}} = -1$, $r_{\text{target}} = 1$. The discount rate is $\gamma = 0.9$.

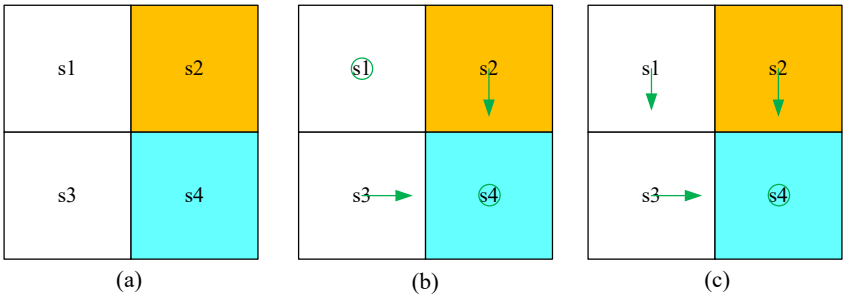


Figure 4.1: An example to demonstrate the implementation of the value iteration algorithm.

The expression of the q-value for each state-action pair is shown in Table 4.1, where

q-value	a_1	a_2	a_3	a_4	a_5
s_1	$-1+\gamma v(s_1)$	$-1+\gamma v(s_2)$	$0+\gamma v(s_3)$	$-1+\gamma v(s_1)$	$0+\gamma v(s_1)$
s_2	$-1+\gamma v(s_2)$	$-1+\gamma v(s_2)$	$1+\gamma v(s_4)$	$0+\gamma v(s_1)$	$-1+\gamma v(s_2)$
s_3	$0+\gamma v(s_1)$	$1+\gamma v(s_4)$	$-1+\gamma v(s_3)$	$-1+\gamma v(s_3)$	$0+\gamma v(s_3)$
s_4	$-1+\gamma v(s_2)$	$-1+\gamma v(s_4)$	$-1+\gamma v(s_4)$	$0+\gamma v(s_3)$	$1+\gamma v(s_4)$

Table 4.1: The expression of $q(s,a)$ for the example shown in Figure 4.1.

q-value	a_1	a_2	a_3	a_4	a_5
s_1	-1	-1	0	-1	0
s_2	-1	-1	1	0	-1
s_3	0	1	-1	-1	0
s_4	-1	-1	-1	0	1

Table 4.2: The value of $q(a,s)$ at $k=0$.

q-table	a_1	a_2	a_3	a_4	a_5
s_1	$-1+\gamma 0$	$-1+\gamma 1$	$0+\gamma 1$	$-1+\gamma 0$	$0+\gamma 0$
s_2	$-1+\gamma 1$	$-1+\gamma 1$	$1+\gamma 1$	$0+\gamma 0$	$-1+\gamma 1$
s_3	$0+\gamma 0$	$1+\gamma 1$	$-1+\gamma 1$	$-1+\gamma 1$	$0+\gamma 1$
s_4	$-1+\gamma 1$	$-1+\gamma 1$	$-1+\gamma 1$	$0+\gamma 1$	$1+\gamma 1$

Table 4.3: The value of $q(a,s)$ at $k=1$.

each row is indexed by a state and each column by an action.

– When $k=0$: set the initial values as $v_0(s_1)=v_0(s_2)=v_0(s_3)=v_0(s_4)=0$.

q-value calculation: Substituting $v_0(s_i)$ into Table 4.1 gives the q-values as shown in Table 4.2.

Policy update: Select the actions with the greatest q-value:

$$\pi_1(a_5|s_1)=1, \quad \pi_1(a_3|s_2)=1, \quad \pi_1(a_2|s_3)=1, \quad \pi_1(a_5|s_4)=1.$$

This policy is visualized in Figure 4.1(b). It is clear that this policy is not good, because it selects to stay unchanged at s_1. It is worth mentioning that, since the q-values for (s_1,a_5) and (s_1,a_3) are the same for s_1, we randomly selected an action.

Value update: Update the v-value as the greatest q-value for each state:

$$v_1(s_1)=0, \quad v_1(s_2)=1, \quad v_1(s_3)=1, \quad v_1(s_4)=1.$$

– When $k=1$:

q-value calculation: Substituting $v_1(s_i)$ into Table 4.1 gives the q-values as shown in Table 4.3.

Policy update: The policy is updated to select the greatest q-values:

$$\pi_2(a_3|s_1) = 1, \quad \pi_2(a_3|s_2) = 1, \quad \pi_2(a_2|s_3) = 1, \quad \pi_2(a_5|s_4) = 1.$$

This policy is visualized in Figure 4.1(c).

Value update: Update the v-value as the greatest q-value for each state:

$$v_2(s_1) = \gamma 1, v_2(s_2) = 1 + \gamma 1, v_2(s_3) = 1 + \gamma 1, v_2(s_4) = 1 + \gamma 1.$$

– $k = 2, 3, 4, \ldots$

In fact, policy π_2 as illustrated in Figure 4.1(c) is already optimal. In practice, we can run a few more steps until the value of v_k converges (that is v_k only varies a sufficiently small amount between consequent steps).

4.2 Policy iteration

This section presents another important algorithm, policy iteration. Different from value iteration, policy iteration is not an algorithm directly solving the Bellman optimality equation. However, it has an intimate relationship to value iteration. Moreover, its idea is widely used by many other RL algorithms such as Monte Carlo learning as we will see in the next chapter.

4.2.1 Algorithm analysis

Policy iteration is also an iterative algorithm. In each iteration, it has two steps.

– The first step is *policy evaluation*. As its name suggests, this step aims to evaluate a given policy by calculating the corresponding state value. Mathematically, it is to solve the Bellman equation of π_k:

$$v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}. \tag{4.1}$$

This is the matrix-vector form of the Bellman equation, where r_{π_k} and P_{π_k} are known. Here, v_{π_k} is the state value to be solved.

– The second step is *policy improvement*. As its name suggests, this step is to improve the policy. How to do that? Once v_{π_k} is calculated in the first step, a new and improved policy could be obtained as

$$\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_{\pi_k}).$$

Here, the maximization is componentwise.

Three questions naturally follow the above description of the algorithm.

– In the policy evaluation step, how to solve the state value v_{π_k}?

– In the policy improvement step, why is the new policy π_{k+1} better than π_k?

– Why can this algorithm finally reach an optimal policy?

We next answer the three questions one by one.

How to calculate v_{π_k}?

We have already studied the methods to solve the Bellman equation in (4.1) in Chapter 2. We revisit the two methods briefly as follows. The first is a closed-form solution: $v_{\pi_k} = (I - \gamma P_{\pi_k})^{-1} r_{\pi_k}$. Although the closed-form solution is useful for theoretical analysis, it is not practical since it involves calculating a matrix inverse which requires sophisticated numerical algorithms. The second method is an iterative algorithm that can be implemented easily:

$$v_{\pi_k}^{(j+1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j)}, \quad j = 0, 1, 2, ... \tag{4.2}$$

Starting from any initial guess $v_{\pi_k}^{(0)}$, it is ensured that $v_{\pi_k}^{(j)} \to v_{\pi_k}$ as $j \to \infty$.

While the policy evaluation step involves an iterative algorithm as in (4.2), it is interesting to note that policy iteration is an iterative algorithm with another iterative algorithm embedded in the policy evaluation step. This embedded iterative algorithm (4.2) requires an *infinite* number of steps to converge to the true state value v_{π_k} in theory. This is, however, impossible to achieve in practice. In practice, the iteration would stop when a criterion is satisfied. For example, $\|v_{\pi_{k+1}} - v_{\pi_k}\|$ is less than a prespecified small value or j exceeds certain threshold. If we can not calculate the precise value of v_{π_k} by running an infinite number of steps, the imprecise value will be substituted to the policy improvement step. Would it cause problems? The answer is no. The reason will be clear when we introduce the truncated policy iteration algorithm later in this chapter.

Why π_{k+1} is better than π_k?

Why can the policy improvement step improve the policy? The answer is as follows.

Lemma 4.1 (Policy improvement)**.** *If $\pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_k})$, then $v_{\pi_{k+1}} \geq v_{\pi_k}$.*

Here, $\geq$ is componentwise. That is, $v_{\pi_{k+1}} \geq v_{\pi_k}$ means $v_{\pi_{k+1}}(s) \geq v_{\pi_k}(s)$ for all s. The proof of the lemma is given in the shaded box.

Proof of Lemma 4.1

First of all, since $v_{\pi_{k+1}}$ and v_{π_k} are state values, they satisfy the Bellman equation:

$$\begin{aligned} v_{\pi_{k+1}} &= r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}, \\ v_{\pi_k} &= r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}. \end{aligned}$$

Since $\pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_k})$, we know

$$r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k} \geq r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}.$$

It then follows that

$$\begin{aligned} v_{\pi_k} - v_{\pi_{k+1}} &= r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k} - (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}) \\ &\leq r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k} - (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}) \\ &\leq \gamma P_{\pi_{k+1}} (v_{\pi_k} - v_{\pi_{k+1}}). \end{aligned}$$

Therefore,

$$\begin{aligned} v_{\pi_k} - v_{\pi_{k+1}} \leq \gamma^2 P_{\pi_{k+1}}^2 (v_{\pi_k} - v_{\pi_{k+1}}) \leq \ldots &\leq \gamma^n P_{\pi_{k+1}}^n (v_{\pi_k} - v_{\pi_{k+1}}) \\ &\leq \lim_{n\to\infty} \gamma^n P_{\pi_{k+1}}^n (v_{\pi_k} - v_{\pi_{k+1}}) = 0. \end{aligned}$$

The limit is due to $\gamma^n \to 0$ as $n \to \infty$ whereas $P_{\pi_{k+1}}^n$ is always a nonnegative stochastic matrix for any n. Here, a stochastic matrix refers to a nonnegative matrix whose row sum is equal to one for all rows.

Why can policy iteration eventually find an optimal policy?

The policy iteration algorithm generates a sequence: $v_{\pi_0}, v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_k}, \ldots$. Suppose v^* is the optimal state value. Then, $v_{\pi_k} \leq v^*$ for all k. Since the policies are continuously improved according to Lemma 4.1, we know that

$$v_{\pi_0} \leq v_{\pi_1} \leq v_{\pi_2} \leq \cdots \leq v_{\pi_k} \leq \cdots \leq v^*.$$

Since v_{π_k} is nondecreasing and always bounded from above by v^*, it follows from the monotone convergence theorem that v_{π_k} will converge to a constant value v_∞ as $k \to \infty$. However, we are not sure whether v_∞ is v^* at this moment. To see that, we give a rigorous analysis in the following result.

Theorem 4.1 (Convergence of policy iteration)**.** *The state value sequence $\{v_{\pi_k}\}_{k=0}^\infty$ generated by the policy iteration algorithm converges to the optimal state value v^*. As a result, the policy sequence $\{\pi_k\}_{k=0}^\infty$ converges to an optimal policy.*

Proof of Theorem 4.1

The idea of the proof is to show that policy iteration converge faster than value iteration. Since value iteration has been proven to be convergent, the convergence of policy iteration immediately follows.

In particular, to prove the convergence of $\{v_{\pi_k}\}_{k=0}^{\infty}$, we introduce another sequence $\{v_k\}_{k=0}^{\infty}$ generated by

$$v_{k+1} = f(v_k) = \max_{\pi}(r_\pi + \gamma P_\pi v_k).$$

This iterative algorithm is exactly the value iteration algorithm. We already know that v_k converges to v^* given any initial value v_0.

We next show that $v_k \leq v_{\pi_k} \leq v^*$ for all k by induction.

For $k = 0$, we can always able to select v_0 such that $v_{\pi_0} \geq v_0$ for any π_0.

For $k \geq 1$, suppose $v_{\pi_k} \geq v_k$.

For $k + 1$,

$$\begin{aligned}
v_{\pi_{k+1}} - v_{k+1} &= (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}) - \max_{\pi}(r_\pi + \gamma P_\pi v_k) \\
&\geq (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k}) - \max_{\pi}(r_\pi + \gamma P_\pi v_k) \\
&\qquad \big(\text{because } v_{\pi_{k+1}} \geq v_{\pi_k} \text{ by Lemma 4.1 and } P_{\pi_{k+1}} \geq 0\big) \\
&= (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k}) - (r_{\pi'_k} + \gamma P_{\pi'_k} v_k) \\
&\qquad \big(\text{suppose } \pi'_k = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_k)\big) \\
&\geq (r_{\pi'_k} + \gamma P_{\pi'_k} v_{\pi_k}) - (r_{\pi'_k} + \gamma P_{\pi'_k} v_k) \\
&\qquad \big(\text{because } \pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_{\pi_k})\big) \\
&= \gamma P_{\pi'_k}(v_{\pi_k} - v_k).
\end{aligned}$$

Since $v_{\pi_k} - v_k \geq 0$ and $P_{\pi'_k}$ is nonnegative, we have $P_{\pi'_k}(v_{\pi_k} - v_k) \geq 0$ and hence $v_{\pi_{k+1}} - v_{k+1} \geq 0$.

Therefore, we show by induction that $v_{\pi_k} \geq v_k$ for any $k \geq 0$. Since v_k converges to v^*, v_{π_k} also converges to v^*.

The above proof not only shows the convergence of policy iteration but also reveals the relationship between policy iteration and value iteration. Loosely speaking, if the two algorithms start from the same initial guess of the state value, policy iteration will converge faster than value iteration due to the additional infinite numbers of iterations embedded in the policy evaluation step. This point will be more clear when we introduce the truncated policy iteration algorithm later.

4.2.2 Elementwise form and implementation

In order to implement the policy iteration algorithm, it is necessary to study its elementwise form.

– First, the policy evaluation step is to solve v_{π_k} from the Bellman equation $v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$ by using the iterative algorithm in (4.2). The elementwise form of this algorithm is

$$v_{\pi_k}^{(j+1)}(s) = \sum_a \pi_k(a|s) \left(\sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}^{(j)}(s') \right), \quad s \in \mathcal{S},$$

where $j = 0, 1, 2, \ldots$.

– Second, the policy improvement step is to solve $\pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_k})$. The elementwise form of this equation is

$$\pi_{k+1}(s) = \arg\max_\pi \sum_a \pi(a|s) \underbrace{\left(\sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}(s') \right)}_{q_{\pi_k}(s,a)}, \quad s \in \mathcal{S}.$$

Here, $q_{\pi_k}(s,a)$ is the action value under policy π_k. Let $a_k^*(s) = \arg\max_a q_{\pi_k}(a,s)$. Then, the greedy optimal policy is

$$\pi_{k+1}(a|s) = \begin{cases} 1 & a = a_k^*(s), \\ 0 & a \neq a_k^*(s). \end{cases}$$

The implementation details are summarized in the pseudocode.

4.2.3 Illustrative examples

A simple example

Consider the example in Figure 4.2. In this example, each state is associated with three possible actions: a_ℓ, a_0, a_r, which represent to move leftwards, stay unchanged, and move rightwards, respectively. The reward setting is $r_{\text{boundary}} = -1$ and $r_{\text{target}} = 1$. The discount rate is $\gamma = 0.9$.

We next show the implementation details of the policy iteration algorithm step by step. When $k = 0$, we start with the initial policy shown in Figure 4.2(a). This policy satisfies $\pi_0(a_\ell|s_1) = 1$ and $\pi_0(a_\ell|s_2) = 1$. This policy is not good because it does not move toward the target area. We next follow the policy iteration algorithm to see if an optimal policy can be obtained.

Pseudocode: Policy iteration algorithm

Initialization: The probability model $p(r|s,a)$ and $p(s'|s,a)$ for all (s,a) are known. Initial guess π_0.

Aim: Search for the optimal state value and an optimal policy.

While the policy has not converged, for the kth iteration, do

Policy evaluation:

Initialization: an arbitrary initial guess $v_{\pi_k}^{(0)}$

While $v_{\pi_k}^{(j)}$ has not converged, for the jth iteration, do

For every state $s \in \mathcal{S}$, do

$$v_{\pi_k}^{(j+1)}(s) = \sum_a \pi_k(a|s) \left[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}^{(j)}(s') \right]$$

Policy improvement:

For every state $s \in \mathcal{S}$, do

For every action $a \in \mathcal{A}(s)$, do

$$q_{\pi_k}(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}(s')$$

$a_k^*(s) = \arg\max_a q_{\pi_k}(s,a)$

$\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise

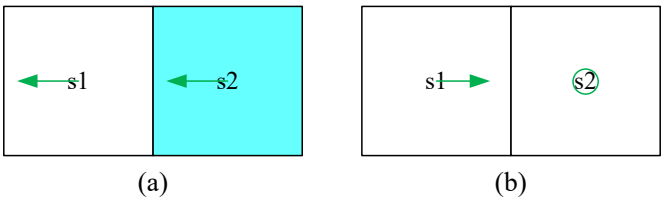


Figure 4.2: An example to illustrate the implementation of the policy iteration algorithm.

– First, in the policy evaluation step, we need to solve the Bellman equation

$$v_{\pi_0}(s_1) = -1 + \gamma v_{\pi_0}(s_1),$$
$$v_{\pi_0}(s_2) = 0 + \gamma v_{\pi_0}(s_1).$$

Since the equation is simple, it can be solved manually that

$$v_{\pi_0}(s_1) = -10, \quad v_{\pi_0}(s_2) = -9.$$

In practice, it is usually solved by the iterative algorithm in (4.2). For example, select

the initial of the state values as $v_{\pi_0}^{(0)}(s_1) = v_{\pi_0}^{(0)}(s_2) = 0$. It follows from (4.2) that

$$\begin{cases} v_{\pi_0}^{(1)}(s_1) = -1 + \gamma v_{\pi_0}^{(0)}(s_1) = -1, \\ v_{\pi_0}^{(1)}(s_2) = 0 + \gamma v_{\pi_0}^{(0)}(s_1) = 0, \end{cases}$$
$$\begin{cases} v_{\pi_0}^{(2)}(s_1) = -1 + \gamma v_{\pi_0}^{(1)}(s_1) = -1.9, \\ v_{\pi_0}^{(2)}(s_2) = 0 + \gamma v_{\pi_0}^{(1)}(s_1) = -0.9, \end{cases}$$
$$\begin{cases} v_{\pi_0}^{(3)}(s_1) = -1 + \gamma v_{\pi_0}^{(2)}(s_1) = -2.71, \\ v_{\pi_0}^{(3)}(s_2) = 0 + \gamma v_{\pi_0}^{(2)}(s_1) = -1.71, \end{cases}$$
$$\vdots$$

With more iterations, we can see the trend that $v_{\pi_0}^{(j)}(s_1) \to v_{\pi_0}(s_1) = -10$ and $v_{\pi_0}^{(j)}(s_2) \to v_{\pi_0}(s_2) = -9$ as j increases.

– Second, in the policy improvement step, the key is to calculate $q_{\pi_0}(s, a)$ for each state-action pair. The following q-table can be used to demonstrate such a process:

$q_{\pi_k}(s,a)$	a_ℓ	a_0	a_r
s_1	$-1 + \gamma v_{\pi_k}(s_1)$	$0 + \gamma v_{\pi_k}(s_1)$	$1 + \gamma v_{\pi_k}(s_2)$
s_2	$0 + \gamma v_{\pi_k}(s_1)$	$1 + \gamma v_{\pi_k}(s_2)$	$-1 + \gamma v_{\pi_k}(s_2)$

Table 4.4: The expression of $q_{\pi_k}(s, a)$ for the example in Figure 4.2.

Substituting $v_{\pi_0}(s_1) = -10, v_{\pi_0}(s_2) = -9$ as obtained in the policy evaluation step into Table 4.4 gives By seeking the greatest value of q_{π_0}, the improved policy is:

$q_{\pi_0}(s,a)$	a_ℓ	a_0	a_r
s_1	-10	-9	-7.1
s_2	-9	-7.1	-9.1

Table 4.5: The value of $q_{\pi_k}(s, a)$ when $k = 0$.

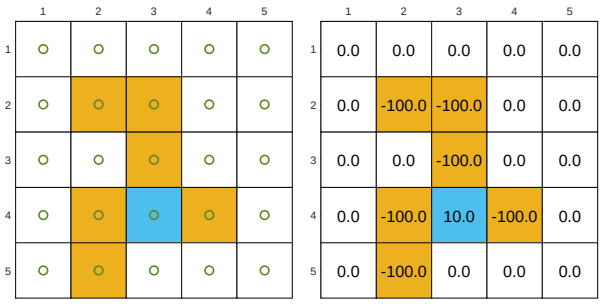
$$\pi_1(a_r|s_1) = 1, \quad \pi_1(a_0|s_2) = 1.$$

This policy is illustrated in Figure 4.2(b). It is intuitively clear that this policy is optimal.

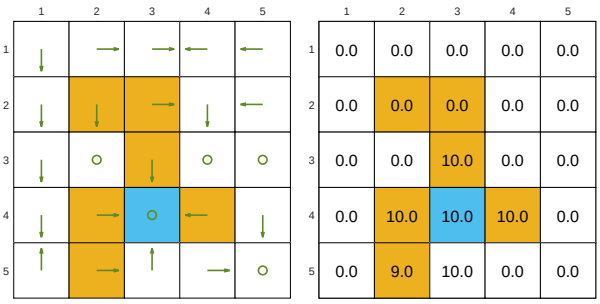
The above process shows that one single iteration can successfully find the optimal policy for this simple example. Of course, more iterations are required for more complex examples.

A more complicated example

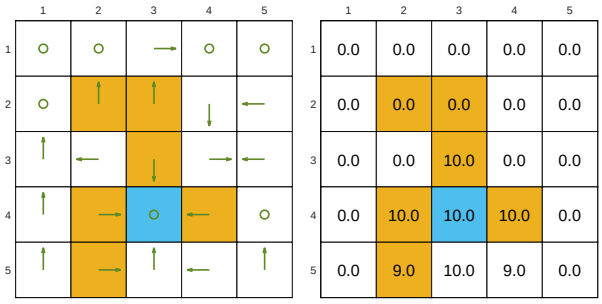
We next examine the properties of the policy iteration algorithm by considering more complicated examples as shown in Figure 4.3. The reward setting is $r_{\text{boundary}} = -1$,



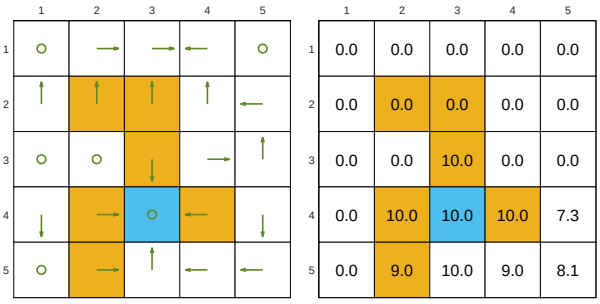
(a) π_0 and v_{π_0}



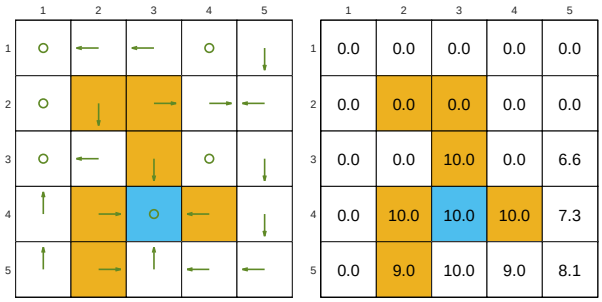
(b) π_1 and v_{π_1}



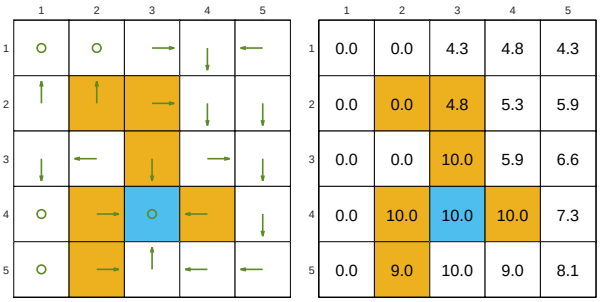
(c) π_2 and v_{π_2}



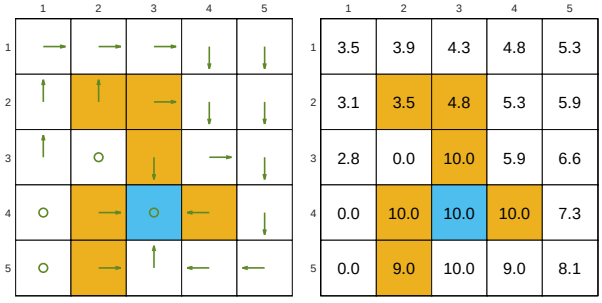
(d) π_3 and v_{π_3}



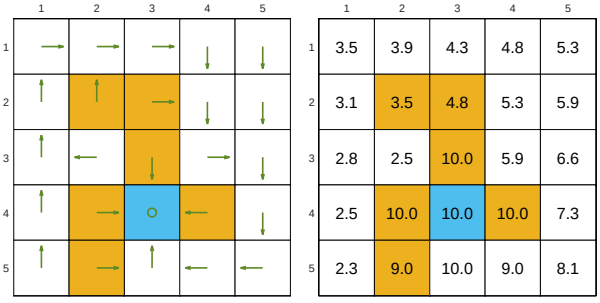
(e) π_4 and v_{π_4}



(f) π_5 and v_{π_5}



(g) π_9 and v_{π_9}



(h) π_{10} and $v_{\pi_{10}}$

Figure 4.3: The evolution process of the policies generated by the policy iteration algorithm.

$r_{\text{forbidden}} = -10$, $r_{\text{target}} = 1$. The discount rate is $\gamma = 0.9$. The policy iteration algorithm can converge to the optimal policy (Figure 4.3(h)) starting from a random initial policy (Figure 4.3(a)).

Two interesting phenomena are observed in the process of policy searching.

– First, if we observe how the policy evolves, an interesting pattern is that the states that are close to the target area find the optimal policies earlier than those states far away. Only if the close states can find trajectories to the target and then update their state values, those farther states can find trajectories passing through the close ones to reach the target.

– Second, if we observe the spatial pattern of the state values of the final policy, the state values exhibit an interesting pattern: the states that are located closer to the target have greater state values. The reason is that the agent starting from a farther state has to travel for many steps to get a positive reward when reaching the target. Such a reward would be severely discounted and hence small. The "closeness" is of course evaluated based on the final policy. That is, if a state has to travel many steps to reach the target, then it is not close to the target, even though its Euclidean distance to the target is short.

4.3 Truncated policy iteration

We next introduce a more general algorithm called *truncated policy iteration*. This algorithm is more general in the sense that value iteration and policy iteration can be viewed as two extremes of it.

4.3.1 Compare value iteration and policy iteration

First of all, we list the steps of policy iteration and value iteration, respectively.

– Policy iteration: starting from any policy π_0,

  ◦ Step 1: policy evaluation (PE). Given π_k, solve v_{π_k} from

$$v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}.$$

  ◦ Step 2: policy improvement (PI). Given v_{π_k}, solve π_{k+1} from

$$\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_{\pi_k}).$$

– Value iteration: starting from any v_0,

	Policy iteration algorithm	Value iteration algorithm	Comments
1) Policy:	π_0	N/A	
2) Value:	$v_{\pi_0} = r_{\pi_0} + \gamma P_{\pi_0} v_{\pi_0}$	$v_0 \doteq v_{\pi_0}$	
3) Policy:	$\pi_1 = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_0})$	$\pi_1 = \arg\max_\pi (r_\pi + \gamma P_\pi v_0)$	The two policies are the same
4) Value:	$v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$	$v_1 = r_{\pi_1} + \gamma P_{\pi_1} v_0$	$v_{\pi_1} \geq v_1$ since $v_{\pi_1} \geq v_{\pi_0}$
5) Policy:	$\pi_2 = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_1})$	$\pi_2' = \arg\max_\pi (r_\pi + \gamma P_\pi v_1)$	π_2 is better than π_2' ($v_{\pi_2} \geq v_{\pi_2'}$)
⋮	⋮	⋮	⋮

Table 4.6: Compare the implementation steps of policy iteration and value iteration.

- Step 1: policy update (PU). Given v_k, solve π_{k+1} from

$$\pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_k).$$

- Step 2: value update (VU). Given π_{k+1}, solve v_{k+1} from

$$v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k.$$

We can compare the two algorithms more closely by writing out their iteration processes:

$$\text{Policy iteration: } \pi_0 \xrightarrow{PE} v_{\pi_0} \xrightarrow{PI} \pi_1 \xrightarrow{PE} v_{\pi_1} \xrightarrow{PI} \pi_2 \xrightarrow{PE} v_{\pi_2} \xrightarrow{PI} \dots$$

$$\text{Value iteration: } v_0 \xrightarrow{PU} \pi_1' \xrightarrow{VU} v_1 \xrightarrow{PU} \pi_2' \xrightarrow{VU} v_2 \xrightarrow{PU} \dots$$

The iteration processes of the two algorithms are very similar.

To illustrate their difference, we let the two algorithms start from the *same initial condition*: $v_0 = v_{\pi_0}$. The evolvement processes of the two algorithms are shown in Table 4.6. In the first three steps, the two algorithms generate the same results since $v_0 = v_{\pi_0}$. They become different in the fourth step. In the fourth step, policy iteration solves $v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$ and requires an infinite number of calculations, whereas value iteration executes $v_1 = r_{\pi_1} + \gamma P_{\pi_1} v_0$, which is a one-step calculation. If we explicitly write out the iterative process solving $v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$ in the fourth step, everything will

be crystal:

$$
\begin{aligned}
& & v_{\pi_1}^{(0)} &= v_0 \\
\text{value iteration} \leftarrow v_1 \longleftarrow & & v_{\pi_1}^{(1)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(0)} \\
& & v_{\pi_1}^{(2)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(1)} \\
& & &\vdots \\
\text{truncated policy iteration} \leftarrow \bar{v}_1 \longleftarrow & & v_{\pi_1}^{(j)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(j-1)} \\
& & &\vdots \\
\text{policy iteration} \leftarrow v_{\pi_1} \longleftarrow & & v_{\pi_1}^{(\infty)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(\infty)}
\end{aligned}
$$

where the initial value $v_{\pi_1}^{(0)}$ is set to be $v_{\pi_1}^{(0)} = v_0 = v_{\pi_0}$. It can be seen clearly from the above iterative process that

– if the iteration is run *only once*, then $v_{\pi_1}^{(1)}$ is actually v_1 as calculated in the value iteration algorithm;

– if the iteration is run *an infinite number of times*, then $v_{\pi_1}^{(\infty)}$ is actually v_{π_1} as calculated in the policy iteration algorithm.

– if the iteration is run *a finite number of times* denoted as j_{truncate}, then such an algorithm is called *truncated policy iteration*. It is called this name simply because the rest iterations from j_{truncate} to ∞ are truncated.

As a result, value iteration and policy iteration can be viewed as two extreme cases of truncated policy iteration, which stop at $j_{\text{truncate}} = 1$ and $j_{\text{truncate}} = \infty$, respectively.

Finally, the above comparison is based on the condition that $v_{\pi_1}^{(0)} = v_0 = v_{\pi_0}$. Without this condition, the two algorithms cannot be compared because they start from different initial conditions.

4.3.2 Truncated policy iteration algorithm

In a nutshell, truncated policy iteration is the same as policy iteration except that it merely runs a finite number of iterations in the policy evaluation step. The implementation details are summarized in the pseudocode. It must be noted that v_k, $v_k^{(j)}$, and q_k in the algorithm are no longer state values or action values. That is simply because, if we only run a finite number of iterations in the policy evaluation step, we can only get an approximation of the true state values.

If v_k is not v_{π_k} due to truncation, will the truncated policy iteration algorithm still be able to find optimal policies? The answer is yes. Intuitively, truncated policy iteration is in between value iteration and policy iteration. On the one hand, it converges faster than the value iteration algorithm, because it computes more than one iteration in the

Pseudocode: Truncated policy iteration algorithm

Initialization: The probability model $p(r|s,a)$ and $p(s'|s,a)$ for all (s,a) are known. Initial guess π_0.

Aim: Search for the optimal state value and an optimal policy.

While the policy has not converged, for the kth iteration, do

Policy evaluation:

Initialization: select the initial guess as $v_k^{(0)} = v_{k-1}$. The maximum iteration is set to be j_{truncate}.

While $j < j_{\text{truncate}}$, do

For every state $s \in \mathcal{S}$, do

$$v_k^{(j+1)}(s) = \sum_a \pi_k(a|s) \left[\sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_k^{(j)}(s') \right]$$

Set $v_k = v_k^{(j_{\text{truncate}})}$

Policy improvement:

For every state $s \in \mathcal{S}$, do

For every action $a \in \mathcal{A}(s)$, do

$$q_k(s,a) = \sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_k(s')$$

$a_k^*(s) = \arg\max_a q_k(s,a)$

$\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise

policy evaluation step. On the other hand, it converges slower than the policy iteration algorithm, because it only computes a finite number of iterations. This intuition is illustrated by Figure 4.4. Such intuition is also supported by the following mathematical analysis.

Proposition 4.1 (Value improvement)**.** *Consider the iterative algorithm in the policy evaluation step:*

$$v_{\pi_k}^{(j+1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j)}, \quad j = 0, 1, 2, ...$$

If the initial guess is selected as $v_{\pi_k}^{(0)} = v_{\pi_{k-1}}$*, it holds that*

$$v_{\pi_k}^{(j+1)} \geq v_{\pi_k}^{(j)}$$

for every $j = 0, 1, 2, \ldots$.

Proof of Proposition 4.1

First, since $v_{\pi_k}^{(j)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j-1)}$ and $v_{\pi_k}^{(j+1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j)}$, we have

$$v_{\pi_k}^{(j+1)} - v_{\pi_k}^{(j)} = \gamma P_{\pi_k} (v_{\pi_k}^{(j)} - v_{\pi_k}^{(j-1)}) = \cdots = \gamma^j P_{\pi_k}^j (v_{\pi_k}^{(1)} - v_{\pi_k}^{(0)}). \tag{4.3}$$

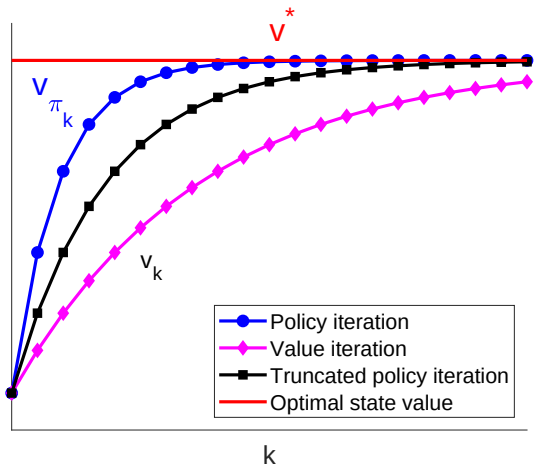


Figure 4.4: Illustration of the relationship among value iteration, policy iteration, and truncated policy iteration.

Second, since $v_{\pi_k}^{(0)} = v_{\pi_{k-1}}$, we have

$$v_{\pi_k}^{(1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(0)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_{k-1}} \geq r_{\pi_{k-1}} + \gamma P_{\pi_{k-1}} v_{\pi_{k-1}} = v_{\pi_{k-1}} = v_{\pi_k}^{(0)}.$$

where the inequality is due to $\pi_k = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_{k-1}})$. Therefore, $v_{\pi_k}^{(1)} \geq v_{\pi_k}^{(0)}$, substituting which into (4.3) gives $v_{\pi_k}^{(j+1)} \geq v_{\pi_k}^{(j)}$.

It is notable that Proposition 4.1 is valid based on the assumption that $v_{\pi_k}^{(0)} = v_{\pi_{k-1}}$. However, $v_{\pi_{k-1}}$ is unavailable in practice whereas only v_{k-1} is available. Nevertheless, Proposition 4.1 still sheds light on the convergence of truncated policy iteration. A more in-depth treatment on this topic can be found in [2, Section 6.5]

Up to now, the advantages of truncated policy iteration are clear. Compared to policy iteration which requires an infinite number of iterations in the policy evaluation step, the truncated one merely requires a finite number of iterations and is more computationally efficient. Compared to value iteration, the truncated policy iteration algorithm can speed up the convergence rate by running a few more iterations in the policy evaluation step, which will be verified by the example studied in the following.

4.3.3 Illustrative examples

We next use examples to demonstrate the impact of the selection of j_{truncate} on the convergence rate of the algorithm. The setup is the same as the example in Figure 4.3. Define $\|v_k - v^*\|$ as the state value error at time k. Starting from the same initial guess of the state value, the evolution of the state value errors are shown in Figure 4.5. The stop criterion is $\|v_k - v^*\| < 0.01$.

Here, j_{truncate} is selected as $1, 3, 6, 100$. When $j_{\text{truncate}} = 1$, the truncated policy iteration algorithm is exactly the value iteration algorithm; when $j_{\text{truncate}} = 100$, it is

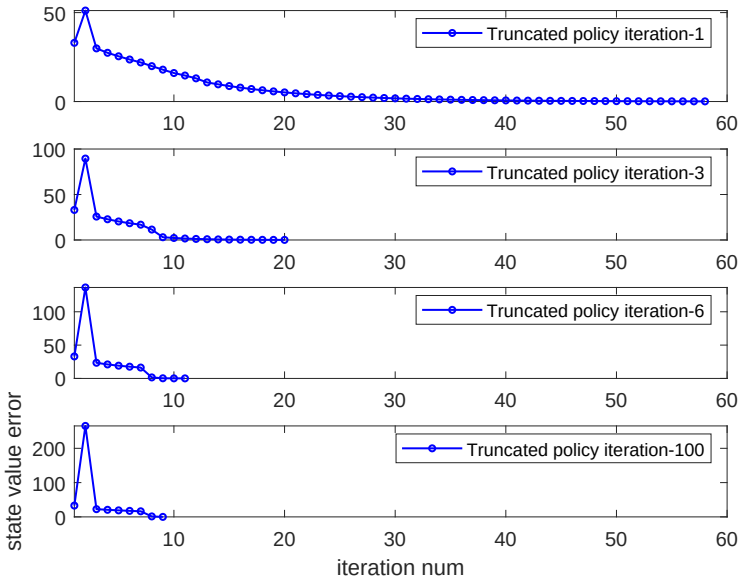


Figure 4.5: Convergence processes of truncated policy iteration. Here, j_{truncate} is selected as $1, 3, 6, 100$. That means the policy evaluation step merely runs j_{truncate} iterations.

approximately the policy iteration algorithm. It is clear that the greater the value of j_{truncate} is, the faster the value estimate converges, which is consistent with the previous theoretical analysis. However, the benefit of increasing j_{truncate} drops quickly when j_{truncate} is large. For example, the overall convergence rate is not enhanced significantly when j_{truncate} increases from 6 to 100. Therefore, it usually can generate satisfactory performance in practice if we simply run a few iterations in the policy evaluation step.

4.4 Summary

This chapter introduced three model-based RL algorithms.

– Value iteration: Value iteration is nothing but the iterative algorithm suggested by the contraction mapping theorem for solving the Bellman optimality equation. When put in the context of RL, it can be decomposed into two steps, value update and policy update. It is notable that the intermediate values generated by value iteration are not state values of any policies in general.

– Policy iteration: Policy iteration is a slightly more complicated algorithm than value iteration. It contains two steps: policy evaluation and policy improvement. The policy evaluation step requires solving a Bellman equation by using an iterative algorithm. This iterative algorithm further requires an infinite number of iterations to solve the true state value. As a result, the values calculated by the policy evaluation step are true state values.

– Truncated policy iteration: Truncated policy iteration is in between value iteration

and policy iteration. That is because it runs a finite number of iterations in the policy evaluation step rather than merely one iteration as the value iteration algorithm or infinite iterations as the policy iteration algorithm. As a result, value iteration and policy iteration can be viewed as two extremes of truncated policy iteration. The intermediate values generated by the truncated policy iteration algorithm are not state values.

All the three algorithms share the same common point. That is, every iteration has two steps. One step is to update the value estimate and the other is to update the policy. Such kind of value-policy interaction in each iteration, which is known as *generalized policy iteration* [3], is a basic idea for many RL algorithms.

Finally, the algorithms introduced in this chapter are model-based. They are also the simplest and first RL algorithms ever introduced in this book. Starting from the next chapter, we will study model-free RL algorithms. Readers will see that model-free RL algorithms such as Monte Carlo based methods are straightforward to understand if the model-based ones have been well understood.

4.5 Q&A

– Q: What steps are there in the value iteration algorithm?

  A: In each iteration of the value iteration algorithm, there are two steps: policy update and value update.

– Q: Why is it guaranteed that value iteration can find optimal policies?

  A: That is because value iteration is exactly the algorithm suggested by the contraction mapping theorem when we solve the Bellman optimality equation in the last chapter. The convergence of this algorithm is guaranteed by the contraction mapping theorem.

– Q: Are the intermediate value generated by the value iteration algorithm state values?

  A: No, because these values are not guaranteed to satisfy the Bellman equation of any policy.

– Q: What steps are there in the policy iteration algorithm?

  A: In each iteration of the policy iteration algorithm, there are two steps: policy evaluation and policy improvement. In the policy evaluation step, the algorithm aims to solve the Bellman equation to obtain the state value of the current policy. In the policy improvement step, the algorithm aims to update the policy so that the new policy has greater state values.

– Q: Is there another iteration algorithm embedded in the policy iteration algorithm?

  A: Yes. In the policy evaluation step of the policy iteration algorithm, an iterative algorithm is required to solve the Bellman equation.

– Q: Are the intermediate values generated by the policy iteration algorithm state values?
A: Yes. Since these values are obtained by solving the Bellman equation, they are state values of the intermediate policies generated during the iteration process.

– Q: Is it guaranteed that the policy iteration algorithm can find optimal policies?

A: Yes. We have given a rigorous proof of its convergence in this chapter.

– Q: What is the relationship between truncated policy iteration and policy iteration?

A: As its name suggested, the truncated policy iteration algorithm only runs a finite number of iterations in the policy evaluation step, whereas the policy iteration algorithm requires running an infinite number of iterations in theory.

– Q: What is the relationship between truncated policy iteration and value iteration?

A: Value iteration can be viewed as an extreme of truncated policy iteration, where a single iteration is run in the policy evaluation step.

– Q: Are the intermediate values generated by the truncated policy iteration algorithm state values?

A: No. That is because we need to run an infinite number of iterations in the policy evaluation step (in theory) in order to get true state values. Therefore, with a finite number of iterations, we can only get approximated values instead of true state values.

– Q: How many iterations should we run in the policy evaluation step in the truncated policy iteration algorithm?

A: The general guideline is to run a few but not too many. As demonstrated in Figure 4.5, a few iterations in the policy evaluation step can help speed up the overall convergence rate, but the benefit of running more iterations is not significant.

– Q: While truncated policy iteration cannot accurately evaluate a policy in every iteration, is it still convergent?

A: Yes. We have discussed the intuition and mathematics behind this conclusion in this chapter. Since value iteration and policy iteration are two extremes of the truncated one and both the two extremes are convergent, it is intuitively clear that the truncated one is also convergent.

– Q: What is generalized policy iteration?

A: Generalized policy iteration is not a specific algorithm. Instead, it refers to the general idea or framework of switching between policy-evaluation and policy-improvement processes for the purpose of optimal policy searching. It is rooted in the policy iteration algorithm. Different from the policy iteration algorithm, generalized policy iteration does not specify the details of the policy-evaluation and policy-improvement. Many reinforcement learning algorithms falls into the scope of generalized policy iteration.