Subject: Westlake University, Reinforce Learning, Lecture 4, Value Iteration and Policy Iteration Algorithms

Date: from January 15, 2025 to January 17, 2025

Contents

Lecture 4, Value Iteration and Policy Iteration Algorithms

Bilibili: Lecture 4, Value Iteration and Policy Iteration Algorithms

Outline

本节将在上一节 Bellman Optimality Equation 的基础上介绍三种寻找 optimal policy 的算法，分别为：

$$\begin{cases} 1. & \text{Value iteration algorithm（值迭代算法）} \\ 2. & \text{Policy iteration algorithm（策略迭代算法）} \\ 3. & \text{Truncated policy iteration algorithm（1 和 2 是其两种特殊情况）} \end{cases}$$

上述算法都被称为动态规划算法（dynamic programming），需要系统的模型（model-based），是下一节 model-free 算法的重要基础.

Value iteration algorithm

Value iteration algorithm

在 Lecture 3 的 Theorem 2 中，我们使用如下算法来求解出最优的 v:

$$v_{k+1} = f(v_k) = \max_\pi (r_\pi + \gamma P_\pi v_k), k = 1, 2, 3, \cdots$$

实际上此算法可分解为两步：

1. **Step 1: policy update(PU).**

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

其中 v_k 是上一步迭代得到的. 其 elementwise form 为

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

根据 Lecture 3，解得

$$\pi_{k+1}(a \mid s) = \begin{cases} 1, & a = a_k^*(s) \\ 0, & a \neq a_k^*(s) \end{cases}, \quad a_k^*(s) = \arg\max_a q_k(a, s)$$

此时由于 π_{k+1} 选择了最大的 q 值，因此被称为 greedy policy.

2. **Step 2: value update(VU).**

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

其 elementwise form 为

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

由于π_{k+1}为 greedy policy，因此得到

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

Note 1. 注意，此处的v_k并非 *state value*. 因为在 *value update* 中并不满足 $v_k = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}}v_k$ 或 $v_k = r_{\pi_k} + \gamma P_{\pi_k}v_k$，因此不是一个 *Bellman equation*，自然就不是 *state value*，而仅仅是一个值. 同时 q_k 也不是 *action value*.

Summary

总结以上两个步骤，计算流程为：

1. 得到 $v_k(s)$
2. 根据 $q_k(s,a)=\sum_r p(r|s,a)r+\gamma\sum_{s'}p(s'|s,a)v_k(s')$ 计算得到 $q_k(s,a)$
3. 得到 greedy ploicy $a_k^*(s)=\arg\max_a q_k(a,s)$
4. 做 value update $v_{k+1}=\max_a q_k(s,a)$

$$v_k(s)\to q_k(s,a)\to \text{greedy policy } \pi_{k+1}(a\mid s)\to \text{new value } v_{k+1}=\max_a q_k(s,a)$$

总结得到如下算法：

Algorithm 1 Value Iteration Algorithm

Initialization: The probability model $p(r\mid s,a)$ and $p(s'\mid s,a)$ for all (s,a) are known. Initial guess v_0, error threshold θ.
Aim: Search for the optimal state value and an optimal policy solving the BOE.
while $\|v_k - v_{k-1}\|\geqslant\theta$ **do**
  for each state $s\in\mathcal{S}$ **do**
    for each action $a\in\mathcal{A}(s)$ **do**
      $q_k(s,a)\leftarrow\sum_r p(r\mid s,a)r+\gamma\sum_{s'}p(s'\mid s,a)v_k(s')$ ▷ *q*-value
    end for
    $a_k^*(s)\leftarrow\arg\max_a q_k(s,a)$ ▷ Maximum action value
    Update policy: ▷ greedy policy

$$\pi_{k+1}(a\mid s)\leftarrow\begin{cases}1 & \text{if } a=a_k^*(s),\\ 0 & \text{otherwise}\end{cases}$$

    Update value: $v_{k+1}(s)\leftarrow\max_a q_k(s,a)$
  end for
end while

Policy iteration algorithm

Policy iteration algorithm

Policy iteration 不直接求解 BOE，但是与 value iteration 有潜在联系，其思想在强化学习算法中被广泛使用. Policy iteration 也是一种迭代算法，其分为如下两步:

1. **Step 1: Policy evaluation(PE).**

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

其中 policy π_k 由上一步迭代得到，r_{π_k}, P_{π_k} 均为已知系统模型，v_{π_k} 为预求未知量. 其 elementwise form 为

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

2. **Step 2: Policy improvement(PI).**

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

其 elementwise form 为

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

同样地选择 greedy policy

$$\pi_{k+1}(a \mid s) = \begin{cases} 1, & a = a_k^*(s) \\ 0, & a \neq a_k^*(s) \end{cases}, \quad a_k^*(s) = \arg\max_a q_k(a,s)$$

对于上面提到的 policy iteration 算法框架，有三个细节问题需要回答:

1. 在 **Policy evaluation** 中如何求解 $v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$?
2. 在 **Policy improvement(PI)** 中为什么 π_{k+1} 比 π_k 更好?
3. 算法最终一定收敛至 optimal policy吗? 为什么?

Q1: 在 Policy evaluation 中如何求解 $v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$?

A1: 在 Lecture 2 中提到有闭式解(closed-form solution)和迭代解(iterative solution)两种解法，但由于闭式解 $v_{\pi_k} = (I - \gamma P_{\pi_k})^{-1} r_{\pi_k}$ 中矩阵求逆很困难，因此转而使用迭代法:

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

Q2: 在 **Policy improvement(PI)** 中为什么 π_{k+1} 比 π_k 更好?

A2: 详见附录A **Theorem 1** 及其证明.

Q3: 算法最终一定收敛至 optimal policy吗? 为什么?

A3: 一定收敛至 optimal policy. 详见附录B **Theorem 2** 及其证明.

Algorithm

Policy iteration 算法如下：

Algorithm 2 Policy Iteration Algorithm

Initialization: The probability model $p(r \mid s, a)$ and $p(s' \mid s, a)$ for all (s, a) are known. Initial guess π_0, error threshold θ_1, θ_2.
Aim: Search for the optimal state value and an optimal policy.
while $||v_{\pi_k} - v_{\pi_k}|| \geqslant \theta_1$, for the k-th iteration **do** ▷ means not converged
  Step 1: Policy evaluation(PE): ▷ to calculate the state value of π_k
  Initialization: an arbitrary initial guess $v_{\pi_k}^{(0)}$
  while $||v_{\pi_k}^{(j+1)} - v_{\pi_k}^{(j)}|| \geqslant \theta_2$ **do** ▷ means not converged
    for every state $s \in S$ **do** ▷ iteration method
      $v_{\pi_k}^{(j+1)}(s) \leftarrow \sum_a \pi_k(a \mid s) \left[\sum_r p(r \mid s, a) r + \gamma \sum_{s'} p(s' \mid s, a) v_{\pi_k}^{(j)}(s') \right]$
    end for
  end while
  Step 2: Policy improvement(PI):
  for every state $s \in S$ **do**
    for every action $a \in A(s)$ **do**
      $q_{\pi_k}(s, a) \leftarrow \sum_r p(r \mid s, a) r + \gamma \sum_{s'} p(s' \mid s, a) v_{\pi_k}(s')$
    end for
    $a_k^*(s) \leftarrow \arg\max_a q_{\pi_k}(s, a)$
    Update policy:

$$\pi_{k+1}(a \mid s) \leftarrow \begin{cases} 1 & \text{if } a = a_k^*(s), \\ 0 & \text{otherwise} \end{cases}$$

  end for
end while

Note 2. 对于此算法需要有以下说明：

1. 此算法会先后产生如下序列：

$$\pi_0 \xrightarrow{PE} v_{\pi_0} \xrightarrow{PI} \pi_1 \xrightarrow{PE} v_{\pi_1} \xrightarrow{PI} \pi_2 \xrightarrow{PE} v_{\pi_2} \xrightarrow{PI} \cdots$$

2. *value iteration* 和 *policy iteration* 有何关系?

   (a) 证明 *policy iteration* 收敛使用了 *value iteration* 收敛的结果

   (b) 二者都是 *truncated iteration* 的极端情况

3. 通过例子可以发现，在迭代过程中接近目标的 *state*会相较于远离目标的 *state* 会先找到 *optimal policy*，因为前者往往是后者的前提.

4. 通过例子可以发现，距离目标较近的 *state* 相较于远离目标的 *state* 的 *state value* 更大，因为较远的 *state* 在接近目标时获得的正奖励(*positive reward*)已经多次 *discount*.

Truncated policy iteration

Comparison of Policy iteration and Value iteration

Policy iteration algorithm	Value iteration algorithm
PE: $v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$ **PI:** $\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_{\pi_k})$	**PU:** $\pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_k)$ **VU:** $v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k$
$\pi_0 \xrightarrow{PE} v_{\pi_0} \xrightarrow{PI} \pi_1 \xrightarrow{PE} v_{\pi_1} \xrightarrow{PI} \pi_2 \xrightarrow{PE} \cdots$	$v_0 \xrightarrow{PU} \pi_1' \xrightarrow{VU} v_1 \xrightarrow{PU} \pi_2' \xrightarrow{VU} v_2 \xrightarrow{PU} \cdots$

[1] 红色与蓝色部分为两种算法迭代的不同侧重点.
[1] PE: Policy evaluation; PI: Policy improvement; PU: Policy update; VU: Value update.

Table 1: Comparison of two algorithms, part 1

	Policy iteration algorithm	Value iteration algorithm
1) Policy:	π_0	N/A
2) Value:[1]	$v_{\pi_0} = r_{\pi_0} + \gamma P_{\pi_0} v_{\pi_0}$	$v_0 := v_{\pi_0}$
3) Policy:[2]	$\pi_1 = \arg\max_\pi \left(r_\pi + \gamma P_\pi v_{\pi_0}\right)$	$\pi_1 = \arg\max_\pi \left(r_\pi + \gamma P_\pi v_0\right)$
4) Value:[3]	$v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$	$v_1 = r_{\pi_1} + \gamma P_{\pi_1} v_0$
5) Policy:	$\pi_2 = \arg\max_\pi \left(r_\pi + \gamma P_\pi v_{\pi_1}\right)$	$\pi_2' = \arg\max_\pi \left(r_\pi + \gamma P_\pi v_1\right)$
$\vdots$	$\vdots$	$\vdots$

[1] 实际上可以为任何初始值，为进行比较，此处赋值v_{π_0}.
[2] two policies are same.
[3] $v_{\pi_1} \geqslant v_1$ since $v_{\pi_1} \geqslant v_{\pi_0}$.

Table 2: Comparison of two algorithms, part 2

Truncated policy iteration

两种算法第一次出现差异实际上是 Table 2 中的第 4) 步，实际上 value iteration 只需计算一次，而 policy iteration 为求解 Bellman equation $v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$ 需要迭代求解无数次。Truncated policy iteration 的意思是在二者间取折中的方案，计算j次，即：

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

Note 3. *1.* 上述比较都建立于假设 $v_{\pi_1}^{(0)} = v_0 = v_{\pi_0}$，否则两种算法不能比较

2. 实际上计算 *policy iteratio n*时也一定不会计算到无穷，也是一个截断的

3. 由于 *truncated Policy Iteration* 在计算 v_{π_k} 时次数多于 *value iteration* 算法但少于 *policy iteration* 算法，因此收敛速度快于 *value iteration* 算法但慢于 *policy iteration* 算法. 详细分析见附录C ***Theorem 3***及其证明.

Algorithm 3 Truncated Policy Iteration Algorithm

Initialization: The probability model $p(r \mid s, a)$ and $p(s' \mid s, a)$ 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 **do**
  Policy evaluation:
  Initialization: select the initial guess as $v_k^{(0)} = v_{k-1}$. The maximum iteration is set to j_{truncate}.
  while $j < j_{\text{truncate}}$ **do**
    for every state $s \in S$ **do**
      $$v_k^{(j+1)}(s) \leftarrow \sum_a \pi_k(a \mid s)\left[\sum_r p(r \mid s, a)r + \gamma \sum_{s'} p(s' \mid s, a)v_k^{(j)}(s')\right]$$
    end for
  end while
  Set $v_k \leftarrow v_k^{(j_{\text{truncate}})}$
  Policy improvement:
  for every state $s \in \mathcal{S}$ **do**
    for every action $a \in \mathcal{A}(s)$ **do**
      $q_k(s, a) \leftarrow \sum_r p(r \mid s, a)r + \gamma \sum_{s'} p(s' \mid s, a)v_k(s')$
    end for
    $a_k^*(s) \leftarrow \arg\max_a q_k(s, a)$
    Update policy:
$$\pi_{k+1}(a \mid s) \leftarrow \begin{cases} 1 & \text{if } a = a_k^*(s), \\ 0 & \text{otherwise} \end{cases}$$
  end for
end while

Summary

Comparison of Policy iteration and Value iteration

上述三种算法都包含两步：1. 更新 value，2. 更新 policy，这种思想被称为 **generalized policy iteration**(1)，是强化学习算法中的重要思想.

Policy iteration 与 Value iteration 都是 model-based reinforcement learning(MBRL)的方法，更准确地说是动态规划(dynamic programming)的方法，其中 Policy iteration 是下一节 Monte Carlo Learning(model-free) 的基础.

A Policy improvement

Policy improvement

Theorem 1 (Policy improvement). 若 $\pi_{k+1} = \arg\max_{\pi} (r_\pi + \gamma P_\pi v_{\pi_k})$，那么 $\forall k, s$，$v_{\pi_{k+1}}(s) \geqslant v_{\pi_k}(s)$.

Proof. 在 policy iteration 算法中，由于 $v_{\pi_{k+1}}$ 和 v_{π_k} 都是由 policy evaluation 中求解 Bellman equation 得到，因此分别满足 BE

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

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

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

将 $v_{\pi_{k+1}}$ 与 v_{π_k} 作差得到

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

因此

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

□

B Convergence of policy iteration algorithm

Convergence of policy iteration algorithm

Theorem 2 (Convergence of policy iteration). *Policy iteration* 算法产生的序列 $\{v_{\pi_k}\}_{k=0}^{\infty}$ 会收敛至 *optimal state value*，即序列 $\{\pi_k\}_{k=0}^{\infty}$ 会收敛至 *optimal policy.*

Proof. 根据 Lecture 3 的 **Theorem 2** 已知在 value iteration 算法中由如下方式产生的序列 $\{v_k\}_{k=0}^{\infty}$ 从任意初始猜测 v_0 开始迭代都会收敛至 v^*.

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

下面对 $v_{\pi_{k+1}}$ 和 v_{k+1} 作差：

$$\begin{aligned}
v_{\pi_{k+1}} - v_{k+1} &= (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}) - \max_{\pi}(r_\pi + \gamma P_\pi v_k) \\
&\geqslant (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k}) - \max_{\pi}(r_\pi + \gamma P_\pi v_k) \\
&\xlongequal{\pi'_k = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_k)} (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k}) - (r_{\pi'_k} + \gamma P_{\pi'_k} v_k) \\
&\geqslant (r_{\pi'_k} + \gamma P_{\pi'_k} v_{\pi_k}) - (r_{\pi'_k} + \gamma P_{\pi'_k} v_k) \\
&= \gamma P_{\pi'_k}(v_{\pi_k} - v_k).
\end{aligned}$$

其中第一个不等号是因为根据 **Theorem 1** 有 $v_{\pi_{k+1}} \geqslant v_{\pi_k}$ 且$P_{\pi_{k+1}} \geqslant 0$；第二个不等号是因为 $\pi_{k+1} = \arg\max_\pi(r_\pi + \gamma P_\pi v_{\pi_k})$.

由于$P_{\pi'_k}$非负定，因此只需要选取 $v_{\pi_0} \geqslant v_0$ (这总是可以做到的)就可以保证 $v_{\pi_{k+1}} - v_{k+1} \geqslant 0$. 因此可以说明 $\forall k, v_k \leqslant v_{\pi_k} \leqslant v^*$，由于 $v_k \to v^*$，故 $v_{\pi_k} \to v^*$. □

C Value improvement

Value improvement

Theorem 3 (Value improvement)**.** 在如下 *policy evaluation* 的迭代算法中，

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

若 $v_{\pi_k}^{(0)} = v_{\pi_{k-1}}$，则

$$v_{\pi_k}^{(j+1)} \geqslant v_{\pi_k}, \quad j = 0, 1, 2, \cdots$$

Proof.

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

由于 $v_{\pi_k}^{(0)} = v_{\pi_{k-1}}$，$\pi_k = \arg\max_\pi(r_\pi + \gamma P_\pi v_{\pi_{k-1}})$，因此

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

故

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

□

First updated: January 16, 2025
Last updated: April 23, 2025

References

[1] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.