Reinforcement Learning CS234

Xiyuan Yang
2025.10.26

Lecture notes for reinforcement learning

目录

§ 1. Introduction

§ 2. Tabular MDP Planning

§ 2.1. Markov Reward Process

Definition 2.1.1 Markov Process.

$$P\left(X_{t_{n+1}} = x_{n+1} \mid X_{t_n} = x_n, ..., X_{t_1} = x_1\right) = P\left(X_{t_{n+1}} = x_{n+1} \mid X_{t_n} = x_n\right)$$

We define G_t with discount factor.

Definition of State value Function $V(s)$ as a MRP.

In MRP, we only have states, transitions, reward functions and discount factors. We have no actions. The transitions are computed all based upon probability.

§ 2.1.1. Bellman Equation

$$V(s) = \underbrace{R(s)}_{\text{Immediate Rewards}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s)V(s')}_{\text{Discounted Sum of future rewards}}$$

- $V(s)$: expected reward from the beginning state s
- $R(s)$: Intermediate rewards
- γ: discount factor
- $\sum_{s' \in S} P(s'|s)V(s')$: expected reward of the next state

Recordings Why only next state?.
- 贝尔曼方程最神奇的问题在于只需要推演下一个时间状态的概率和期望值
- 同时，我们保证 V 代表的是该策略下的最优决策，这本质上是一种递归的思想。因此我们并不需要把未来无穷时间步的 reward 全部计算出来。
- 贝尔曼方程是马尔科夫决策状态下许多算法的基础，一般来说，只要认为状态是马尔科夫状态，则算法的关键在于正确的估计最优价值函数 $V(s)$

§ 2.1.2. Finite Space Solving

For finite state of MDP, we can express $V(s)$ as a matrix or a tabular.

$$\begin{pmatrix} V(s_1) \\ V(s_2) \\ V(s_3) \\ \dots \\ V(s_n) \end{pmatrix} = \begin{pmatrix} R(s_1) \\ R(s_2) \\ R(s_3) \\ \dots \\ R(s_n) \end{pmatrix} + \gamma \begin{pmatrix} P(s_1|s_1) & P(s_2|s_1) & \dots & P(s_n|s_1) \\ P(s_1|s_2) & P(s_2|s_2) & \dots & P(s_n|s_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(s_1|s_n) & P(s_2|s_n) & \dots & P(s_n|s_n) \end{pmatrix} \begin{pmatrix} V(s_1) \\ V(s_2) \\ V(s_3) \\ \dots \\ V(s_n) \end{pmatrix}$$

很明显，对于有限状态 n 的马尔科夫决策过程，本质上就是求解一个最基本的线性方程组：

$$\boldsymbol{V} = \boldsymbol{R} + \gamma \boldsymbol{P}^T \boldsymbol{V}$$

$$\boldsymbol{V} = \left(\boldsymbol{I} - \gamma \boldsymbol{P}^T\right)^{-1} \boldsymbol{R}$$

§ 2.1.3. Infinite Space Solving

在实际情况中，矩阵求解的方法时间复杂度过于昂贵，并且实际情况下的状态总数 n 会变得极大，这也就意味着模型需要极大的一个状态转移矩阵，这对计算资源的消耗是不可接受的。

因此，下面介绍使用动态规划的算法来实现：

k means the iteration steps. (For the initial state, we purpose $V_0(s) = 0$)

$$V_{k(s)} = \underbrace{R(s)}_{\text{Immediate Rewards}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s)V_{k-1}(s')}_{\text{Discounted Sum of future rewards}}$$

这实际上可以算作是一个优化的估计问题，因为我们事先无得知未来的状态的价值函数，因此我们需要多次迭代来毕竟最优的价值函数，当这个数收敛的时候，就代表估计成功了。

从数学上来看这一点：我们寻找的是最优的价值函数，满足贝尔曼方程：

$$V^*(s) = \max_a \left(\underbrace{R(s,a)}_{\text{Immediate Rewards}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s)V^*(s')}_{\text{Discounted Sum of future rewards}} \right)$$

定义贝尔曼优化算子 $\mathcal{T}$:

$$\mathcal{T}(V(s)) = \max_a \left(\underbrace{R(s,a)}_{\text{Immediate Rewards}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s)V(s')}_{\text{Discounted Sum of future rewards}} \right)$$

We hope:

$$V_k = \mathcal{T}(V_{k-1})$$

Definition 2.1.3.1 压缩映射.

$$\|\mathcal{T}V_a - \mathcal{T}V_b\|_\infty \le \gamma \|V_a - V_b\|_\infty$$

而压缩映射可证明：保证有巴拿赫不动点！ 而我们可以证明贝尔曼算子在 γ 小于 1 的时候是一个压缩算子。

For each iteration, the time complexity is $O(|S|^2)$

§ 2.2. Markov Decision Process

$$S, A, P, R, \gamma$$

Thus, the transition is: $P(s'|s,a)$

§ 2.2.1. MDP Policies

$$\pi(a|s) = P(a_t = a | s_t = s)$$

It is a Stochastic Policy. MDP with $\pi(a|s)$ is MRP.

Recordings MDP and MRP.
- 在 MDP 中，我们加入了策略，即智能体会有一个在给定状态下的概率分布，而这个概率分布会让智能体做出相对应的决策。
- 而一旦这个智能体的概率分布被固定，这个本质上就是一个 MRP 系统！因为最终都会化归到一个概率分布上。

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

For MDP Iteration:

$$V_k^\pi(s) = \sum_a \pi(a|s) \left[R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V_{k-1}^\pi(s') \right]$$

§ 2.2.2. Policy Search

对于确定状态空间下的确定性策略下，穷举搜索所有可行性策略的总数为 $O(|A|^{|S|})$，这是非常大的搜索空间！

确定性策略 π 的时间复杂度为一个从状态到动作的映射函数。

§ 2.2.2.1. MDP Policy Iteration

The intuition of MDP Policy Iteration is about **evaluation** and **improvement**. That is, a good policy may bring high values, and in turn leads to better policy optimizations.

Definition 2.2.2.1.1 MDP Policy Iteration.

- Initialize $\pi_0(s)$ for random policy selection.
- If `i == 0` and $\|\pi_i - \pi_{i-1}\|_1 > 0$ (L1-norm):
  - Do evaluation for policy π_k, and update the value of V^{π_k}
    - Using **Bellman Equations**

$$V_k^\pi(s) = \sum_a \pi(a|s) \left[R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V_{k-1}^\pi(s') \right]$$

- Do **Policy Improvement**: calculate Q function (state, action, reward function) and update π_{k+1}

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi_k}(s')$$

$$\pi_{k+1}(s) = \text{argmax}_a \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi_k}(s') \right) = \text{argmax}_a Q^{\pi_i}(s,a)$$

- `i++`

§ 2.2.2.2. Policy Improvement

Let's see the core function: the Q function!

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi_k}(s')$$

Actually, it is the greedy search (selecting $\max_a Q^{\pi_i}(s,a)$)!

$$\max_a Q^{\pi_i}(s,a) \geq R\left(s, \pi_{i(s)}\right) + \gamma \sum_{s' \in S} P\left(s'|s, \pi_{i(s)}\right) V^{\pi_i}(s') = V^{\pi_i}(s)$$

- $Q^{\pi_i}(s,a)$，在当前状态下执行动作 a，并且在下一时刻严格开始遵循当前策略 π_i 获得的期望总回报
- $V^{\pi_i}(s)$：当前状态下严格执行当前策略 π_i 获得的期望总回报。
  - 只有当且仅当最优策略对应的 action 就是 $\pi_{i(s)}$ 的时候，两者取到等号。

§ 2.2.2.3. Monotonic Improvement

Definition 2.2.2.3.1 Judgement between value functions.

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

下面，我们希望证明我们的更新方法是单调的，这也是我们动态规划正确性的基础！

Proof.

We need to prove: $V^{\pi_{i+1}} \geq V^{\pi_i}$.

$$V^{\pi_{i+1}} = Q^{\pi_{i+1}} = (s, \pi_{i+1}(s)) = R(s, \pi_{i+1}(s)) + \underbrace{\sum_{s' \in S} P(s'|s, \pi_{i+1}(s)) V^{\pi_{i+1}}(s')}_{\text{注意是}\pi_{i+1}}$$

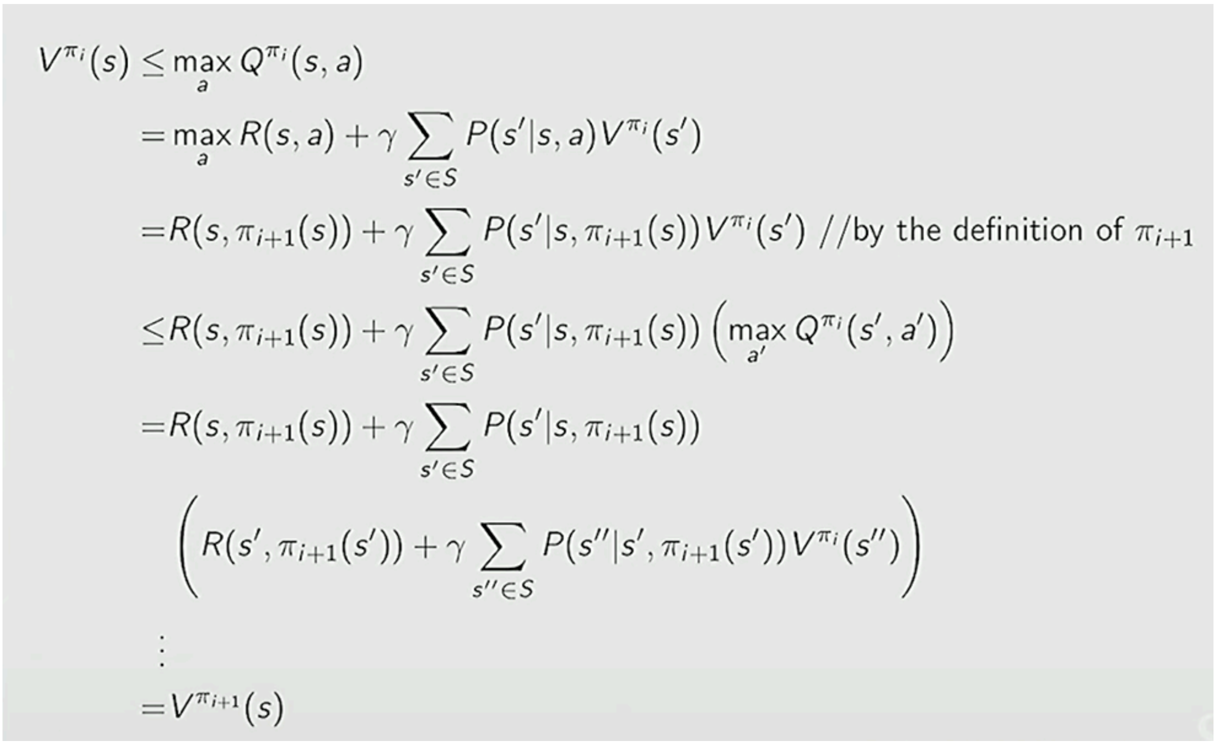
$$Q^{\pi_i}(s, \pi_{i+1}(s)) = \max_a Q^{\pi_i}(s, a) = R(s, \pi_{i+1}(s)) + \gamma \underbrace{\sum_{s' \in S} P(s'|s, \pi_{i+1}(s)) V^{\pi_i}(s')}_{\text{注意是}\pi_i}$$

下面我们证明的目标是：$V^{\pi_{i+1}} \geq V^{\pi_i}$

$$\begin{aligned} V^{\pi_i}(s) &\leq \max_a Q^{\pi_i}(s, a) = Q^{\pi_i}(s, \pi_{i+1}(s)) \\ &= R(s, \pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s, a) V^{\pi_i}(s') \end{aligned}$$

$$V^{\pi_i}(s') \leq Q^{\pi_i}(s', \pi_{i+1}(s'))$$

将上式不断展开，最终将会得到 $V^{\pi_{i+1}}(s)$，这也就是最终迭代更新后的价值函数！ □



$$\begin{aligned} V^{\pi_i}(s) &\leq \max_a Q^{\pi_i}(s, a) \\ &= \max_a R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^{\pi_i}(s') \\ &= R(s, \pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi_{i+1}(s)) V^{\pi_i}(s') \quad \text{//by the definition of } \pi_{i+1} \\ &\leq R(s, \pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi_{i+1}(s)) \left(\max_{a'} Q^{\pi_i}(s', a')\right) \\ &= R(s, \pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi_{i+1}(s)) \\ &\quad \left(R(s', \pi_{i+1}(s')) + \gamma \sum_{s'' \in S} P(s''|s', \pi_{i+1}(s')) V^{\pi_i}(s'')\right) \\ &\quad \vdots \\ &= V^{\pi_{i+1}}(s) \end{aligned}$$

图 1 Proof for policy improvement.

因此，我们最终证明了两件事情：

- 基于上述算法的优化是单调的
  - 可以证明对于任何一个状态，更新后的价值函数所获得的期望收益会高于更新前的期望收益
  - 证明了马尔科夫算子压缩映射的不动点的存在性
- 策略函数的选择是有限的

因此，最终我们会收敛到最优的策略函数 π^*

§ 2.2.3. Value Iteration

- Policy iteration computes infinite horizon value of a policy and then improves that policy
- Value iteration is another technique

- Idea: 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

For the algorithm:

- Waiting loop for convergence: $\|V_{k+1} - V(k)\|_\infty \leq \varepsilon$
- 迭代的关键使用贝尔曼优化算子，输入一个价值函数，输出一个更新后的价值函数。

$$V_{k+1}(s) = \mathcal{T}(V_k(s))$$

而这个价值策略函数的迭代过程也包含隐式策略改进：

$$\pi_{k+1}(s) = \text{argmax}_a \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_k(s') \right]$$

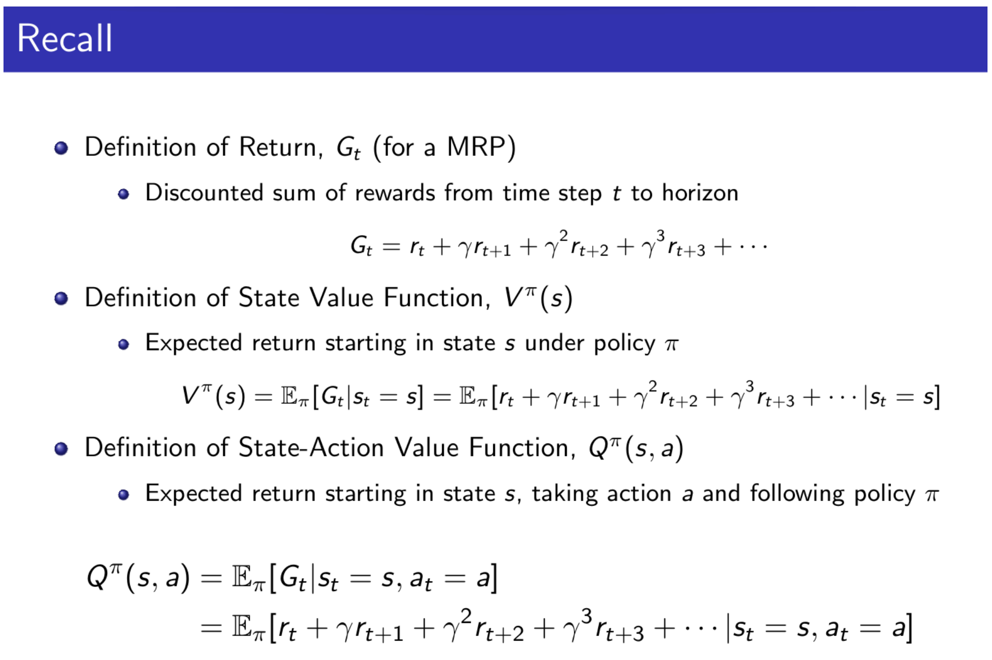
Recordings Value Iteration and Policy Iterations.

- Policy Iteration
  - 使用贝尔曼期望方程作为主要原理进行迭代
  - 交替更新 V_{π_k} and π_k
- Value Iteration
  - 直接更新贝尔曼方程

§ 3. Model-Free Policy Evaluation

Policy Evaluation Without Knowing How the World Works!

§ 3.1. Recall



Recall

- Definition of Return, G_t (for a MRP)
  - Discounted sum of rewards from time step t to horizon

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots$$

- Definition of State Value Function, $V^\pi(s)$
  - Expected return starting in state s under policy π

$$V^\pi(s) = \mathbb{E}_\pi[G_t | s_t = s] = \mathbb{E}_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots | s_t = s]$$

- Definition of State-Action Value Function, $Q^\pi(s,a)$
  - Expected return starting in state s, taking action a and following policy π

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi[G_t | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots | s_t = s, a_t = a] \end{aligned}$$

We can do policy evaluation through dynamic programming, and V_k^π is just an estimate of V^π. (Policy evaluation during the policy search.)

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

Recordings BootStrapping.

$$\sum_{s' \in S} p(s'|s, \pi(s)) \to \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s]$$

§ 3.2. Model-Free Policy Evaluation

Recordings Model-Free RL.

- 在之前的模型中，例如 Tabular MDP Process，我们默认状态之间的转移概率是已知的，即我们明确转移概率 $P(s'|s, a)$ 和 $R(s, a)$。
- 但是在真实的世界模型中，存在如下瓶颈：
  - 状态转移空间巨大，我们难以枚举所有的状态转移概率和奖励函数到一个表格中
  - 转移规律未知并且复杂

§ 3.3. Monte Carlo Policy Evaluation

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi}[G_t | s_t = s]$$

- Expectation over trajectories τ generated by following - Simple idea: **Value = mean return**
- If trajectories are all finite, sample set of trajectories & average returns
- Note: all trajectories may not be same length (e.g. consider MDP with terminal states)

§ 3.3.1. First Visit Monte Carlo Evaluation

The loop for single First-Visit Monte Carlo Evaluation is:

- Sample Episode $i = (s_{i,1}, a_{i,1}, r_{i,1}), (s_{i,2}, a_{i,2}, r_{i,2}), \ldots, (s_{i,T_i}, a_{i,T_i}, r_{i,T_i})$
- Calculate the real return from the environment, which is $G_{i,t}$
- For each time step t until T_i:
  - If is the first time t that state s is visited:
    - N(s) ++
    - G(s) += $G_{i,t}$
    - Update Estimate $V^\pi = \frac{G(s)}{N(s)}$

Recordings Update Incrementally.

$$V^\pi(s) = V^\pi(s) \frac{N(s) - 1}{N(s)} + \frac{G_{i,t}}{N(s)}$$

More generally:

$$V^\pi(s_{i,t}) = V^\pi(s_{i,t}) + \alpha(G_{i,t} - V^\pi(s_{i,t}))$$

- α is the learning rate.
- The learning part is the TD Error.

Recordings Monte Carlo.

- Intuition 就是根据这个模拟来代替采样
- 对于一个情节，只使用状态 s 在该情节中第一次出现时的回报 $G(i,t)$ 来更新其价值估计。如果在同一个情节中状态 s 再次出现，该次的回报会被忽略。
  - 在实际过程中 这一项是通过 sample 新的采样点实现的
- 根据大数定律，这个数最终会收敛到期望值。

§ 3.3.2. Every Visit Monte Carlo Evaluation

Not only the first time for each episode.

§ 3.3.3. Bias, Variance and MSE

Definition 3.3.3.1 Bias.

$$\mathrm{Bias}_\theta\left(\hat{\theta}\right) = \mathbb{E}_{x|\theta}\left[\hat{\theta}\right] = \theta$$

$$\mathrm{Var}\left(\hat{\theta}\right) = \mathbb{E}_{x|\theta}\left[\left(\hat{\theta} - \mathbb{E}\left[\hat{\theta}\right]\right)^2\right]$$

$$\mathrm{MSE}\left(\hat{\theta}\right) = \mathbb{E}\left[\left(\hat{\theta} - \theta\right)^2\right] = \mathrm{Var}(\theta) + \mathrm{Bias}_\theta\left(\hat{\theta}\right)^2$$

Recordings First Visit and Every Visit.

- In the first visit MC:
  - V^π is the **unbiased estimator** of true $\mathbb{E}_{\pi[G_t|s_t=s]}$
  - 因为可以保证每一个数据点都来自于不同的采样（至多被采样一次），而不同的采样之间保证独立性
- Every Visit MC:
  - a biased estimator
  - 因为一个 episode 可能同时出现这个状态，无法保证数据之间的独立性

§ 3.4. Temporal Difference (TD(0))

Goal: $V^\pi(s)$ given episodes generated under policy π. ($V^\pi(s) = \mathbb{E}[G_t|s_t = s]$)

In incremental every-visit Monte carlo evaluation, we can update the estimate by the learning rate α:

$$V^\pi(s) = V^\pi(s) + \alpha\left(G_{i,t} - V^\pi(s)\right)$$

Since we have an estimate of V^π, we can use it to estimate the expected return:

$$V^\pi(s) = V^\pi(s) + \alpha([r_t + \gamma V^\pi(s_{t+1})] - V^\pi(s))$$

- $[r_t + \gamma V^\pi(s_{t+1})]$ is the **bootstrapping**, where the model just take one single step forward!

§ 4. Conclusion