

蒙特卡洛方法 (Monte Carlo Methods)

吉建民

USTC

`jianmin@ustc.edu.cn`

2023 年 10 月 9 日

Used Materials

Disclaimer: 本课件大量采用了 Rich Sutton's RL class, David Silver's Deep RL tutorial 和其他网络课程课件, 也采用了 GitHub 中开源代码, 以及部分网络博客内容

Table of Contents

课程回顾

背景

蒙特卡洛预测 (Monte Carlo Prediction)

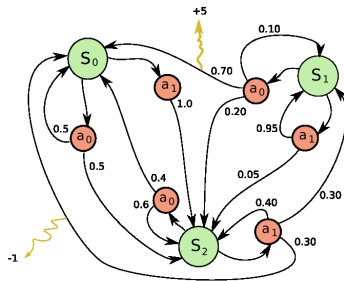
蒙特卡洛控制 (Monte Carlo Control)

On-policy

Off-policy

Markov Decision Processes

- ▶ MDP 模型是一个四元组 $\langle S, A, T, R \rangle$
 - ▶ Markov Property: $P(s_{t+1} \mid s_1, \dots, s_t) = P(s_{t+1} \mid s_t)$
 - ▶ $T(s, a, s') = P(s' \mid s, a)$
 - ▶ Policy: $\pi : S \times A \rightarrow [0, 1]$, $\pi(a \mid s)$
- ▶ 已知 MDP 模型
 - ▶ Prediction: 给定 MDP 和 policy π , 计算值函数 V_π 或 Q_π
 - ▶ Control: 给定 MDP, 计算最优策略 π^* 或最优值函数 V^* 或 Q^*



State-Value Function and Action-Value Function

- ▶ 回报 (return): 回报 G_t 是从时刻 t 开始的总折扣奖励:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=1}^{\infty} \gamma^k R_{t+k+1}$$

- ▶ 状态值函数 (state-value function): 状态值函数 $V_{\pi}(s)$ 是从状态 s 出发, 按照策略 π 采取行动得到的期望回报:

$$\begin{aligned} V_{\pi}(s) &= E_{\pi}(G_t \mid S_t = s) \\ &= E_{\pi}(R_{t+1} + \gamma G_{t+1} \mid S_t = s) \\ &= E_{\pi}(R_{t+1} + \gamma V_{\pi}(S_{t+1}) \mid S_t = s) \end{aligned}$$

- ▶ 行动值函数 (action-value function, action-state-value function): 行为值函数 $Q_{\pi}(s, a)$ 是从状态 s 出发, 采取行动 a 后, 然后按照策略 π 采取行动得到的期望回报:

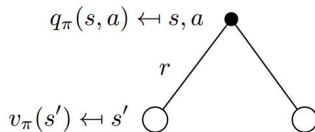
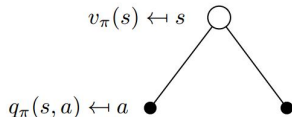
$$\begin{aligned} Q_{\pi}(s, a) &= E_{\pi}(G_t \mid S_t = s, A_t = a) \\ &= E_{\pi}(R_{t+1} + \gamma Q_{\pi}(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a) \end{aligned}$$

State-Value Function and Action-Value Function (con't)

- ▶ $V_\pi(s)$ 与 $Q_\pi(s, a)$ 之间的关系

$$V_\pi(s) = \sum_{a \in A} \pi(a | s) Q_\pi(s, a)$$

$$Q_\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_\pi(s')$$



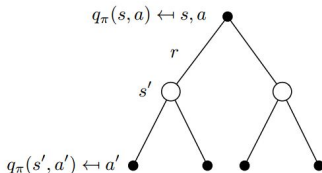
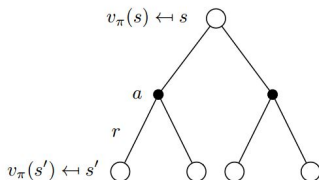
Bellman Equations

- ▶ V_π 自身的递推关系:

$$V_\pi(s) = \sum_{a \in A} \pi(a | s) \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_\pi(s') \right)$$

- ▶ Q_π 自身的递推关系:

$$Q_\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \sum_{a' \in A} \pi(a' | s') Q_\pi(s', a')$$



Bellman Equations (con't)

- ▶ 最优状态值函数：最优值函数 $V^*(s)$ 是在所有策略上的最大值函数：

$$V^*(s) = \max_{\pi} V_{\pi}(s)$$

- ▶ 最优行为值函数：最优行为值函数 $Q^*(s, a)$ 是在所有策略上的最大行为值函数

$$Q^*(s, a) = \max_{\pi} Q_{\pi}(s, a)$$

- ▶ $V^*(s)$ 与 $Q^*(s, a)$ 之间的关系

$$\begin{aligned} V^*(s) &= \max_{\pi} V_{\pi}(s) \\ &= \max_{\pi} \sum_{a \in A} \pi(a | s) Q_{\pi}(s, a) \\ &= \max_{a \in A} Q^*(s, a) \end{aligned}$$

因为：MDP 的 optimal policy 是 deterministic policy, *i.e.*,
 $\pi^*(s) = a$

Bellman Equations (con't)

- ▶ $V^*(s)$ 与 $Q^*(s, a)$ 之间的关系

$$\begin{aligned} Q^*(s, a) &= \max_{\pi} Q_{\pi}(s, a) \\ &= \max_{\pi} \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_{\pi}(s') \right) \\ &= R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \end{aligned}$$

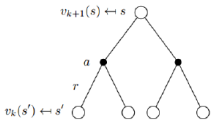
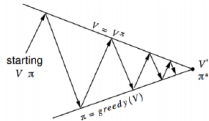
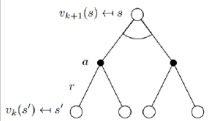
- ▶ $V^*(s)$ 自身的递推关系

$$V^*(s) = \max_{a \in A} \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \right)$$

- ▶ $Q^*(s, a)$ 自身的递推关系

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \max_{a' \in A} Q^*(s', a')$$

Dynamic Programming

Algorithm	Iterative Policy Evaluation	Policy Iteration	Value Iteration
			
Bellman Equation	Bellman Expectation Equation	Bellman Expectation Equation Policy Iteration + Greedy Policy Improvement	Bellman Optimality Equation
Problem	Prediction	Control	Control

Small Grid World

Policy π , an equiprobable random action



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R = -1$
on all transitions

$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, ..., 14
- Terminal state: one, shown in shaded square
- Actions that take the agent off the grid leave the state unchanged
- Reward is -1 until the terminal state is reached

V_k for the
Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0.0

Greedy Policy
w.r.t. V_k

	→	→	→
→	→	→	→
→	→	→	→
→	→	→	→

random
policy

Small Grid World (con't)

Policy π , an equiprobable random action



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R = -1$
on all transitions

$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, ..., 14
- Terminal state: one, shown in shaded square
- Actions that take the agent off the grid leave the state unchanged
- Reward is -1 until the terminal state is reached

V_k for the
Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Greedy Policy
w.r.t. V_k

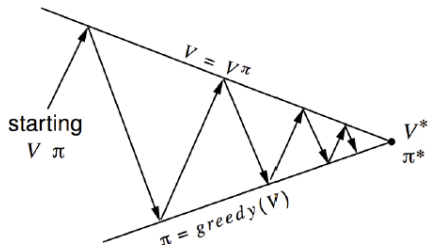
random
policy

optimal
policy

Deterministic Policy and Stochastic Policy

- ▶ Deterministic Policy: $\pi : S \rightarrow A, \pi(s) = a$
- ▶ Stochastic Policy: $\pi : S \times A \rightarrow [0, 1], \pi(a | s) \in [0, 1]$
- ▶ Dynamic programming algorithms 初始输入可以是 stochastic policy 但返回总是一个 deterministic policy
- ▶ 有没有可能存在 optimal policy π^* , 它是 stochastic policy, 并且对所有 deterministic policy π' , $V_{\pi^*} > V_{\pi'}$?
 - ▶ “Markov Decision Process – Discrete Stochastic Dynamic Programming” by Martin L. Puterman (John Wiley and Sons Ed.). It is proved that if the reward function is deterministic, the optimal policy exists and is also deterministic.
 - ▶ 反证法, 假定一个 stochastic policy π^* 是最优的, 并且不存在 deterministic policy 大于等于它的期望效用。当 π^* 在用概率选择两个行动时, 总可以选后续效用较大的那个行动; 当两者后续效用相等, 就固定的选一个

Generalized Policy Iteration



Policy evaluation Estimate v_π

Any policy evaluation algorithm

Policy improvement Generate $\pi' \geq \pi$

Any policy improvement algorithm

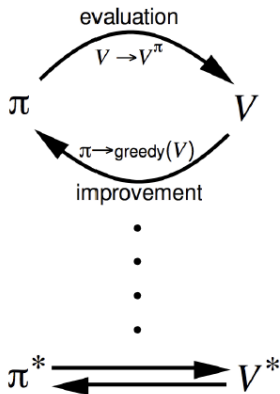


Table of Contents

课程回顾

背景

蒙特卡洛预测 (Monte Carlo Prediction)

蒙特卡洛控制 (Monte Carlo Control)

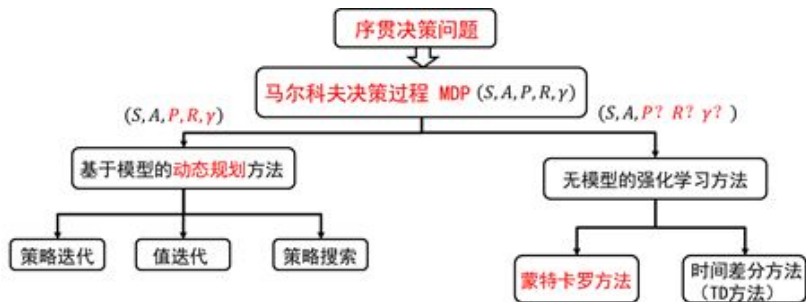
On-policy

Off-policy

Model-Free Reinforcement Learning

- ▶ From MDP to RL: MDP $\langle S, A, T, R \rangle$, where T and R are unknown
- ▶ Model-based Methods: learn T and R , then solve the MDP (like Dynamic Programming)
- ▶ Model-free Methods: learn policy without T or R
 - ▶ Explore the environment and learn policy at the same time
 - ▶ Monte-Carlo method
 - ▶ Temporal difference method
- ▶ Monte Carlo method
 - ▶ Model-free prediction: Estimate the value function of an unknown MDP using Monte Carlo
 - ▶ Model-free control: Optimise the value function of an unknown MDP using Monte Carlo

强化学习方法

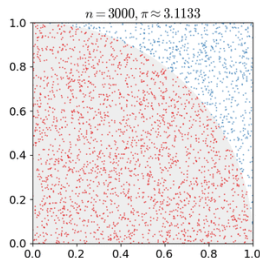


蒙特卡洛方法 (Monte Carlo Method)

- ▶ 蒙特卡洛（一个赌城的名字）方法又叫做统计模拟方法，它使用随机数（或伪随机数）来解决计算问题
- ▶ 蒙特卡洛方法的基本思路：模拟 -> 抽样 -> 估值
 - ▶ 用蒙特卡洛方法求 π 的近似值。在图中随机生成 N 个点，再统计有 M 个点在圆内

$$\pi \approx \frac{4M}{N}$$

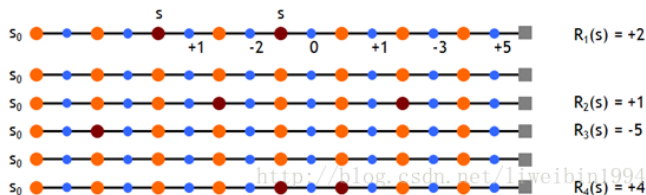
- ▶ $N = 30,000$, $\pi \approx 3.1524$



蒙特卡洛强化学习 (Monte Carlo Reinforcement Learning)

- ▶ 蒙特卡洛方法仅仅需要经验就可以求解最优策略，这些经验可以在线获得或者根据某种模拟机制获得
- ▶ 蒙特卡洛方法定义在 episode task 上，所谓的 episode task 就是指不管采取哪种策略 π ，都会在有限时间内到达终止状态并获得回报的任务。比如玩棋类游戏，在有限步数以后总能达到输赢或者平局的结果并获得相应回报
- ▶ 经验就是训练样本。比如在初始状态 s ，遵循策略 π ，最终获得了总回报 R ，这就是一个样本。如果我们有许多这样的样本，就可以估计在状态 s 下，遵循策略 π 的期望回报，也就是状态值函数 $V_{\pi}(s)$ 了。蒙特卡罗方法就是依靠样本的平均回报来解决强化学习问题的

Episode Task



- ▶ episode 就是经历，每条 episode 就是一条从起始状态到结束状态的经历。例如在走迷宫，一条 episode 就是从你开始进入迷宫，到最后走出迷宫的路径
- ▶ 我们要得到的是某一个状态 s 的平均收获。所需的 episode 要经过状态 s 。上图中第二条路径没有经过状态 s ，对于 s 来说就不能使用它了。所有 episode 都是要求达到终点的，有相应的回报值
- ▶ 无论采用哪个策略，都会在有限时间内到达终点并获得回报。如上图，每条样本都会最终到达终点。现实中，我们的棋类游戏，都会在有限步数以后达到输赢或者平局的结果并获得相应的回报

Table of Contents

课程回顾

背景

蒙特卡洛预测 (Monte Carlo Prediction)

蒙特卡洛控制 (Monte Carlo Control)

On-policy

Off-policy

Monte Carlo Policy Evaluation

- ▶ 目标：在一个固定的策略 π 下，从一系列的 episodes 中学习该策略下的状态价值函数 $V_\pi(s)$

$$s_1, a_1, r_2, \dots, s_k \sim \pi$$

- ▶ 回报 G_t 为：

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R^T$$

- ▶ 值函数 $V_\pi(s)$ 为期望回报

$$V_\pi(s) = E_\pi(G_t \mid S_t = s)$$

- ▶ 蒙特卡罗策略评估则是使用多个 episodes 的平均获得值来代替这个状态价值函数

$$V_\pi(s) = E_\pi(G_t \mid S_t = s) \approx \frac{\sum_{i: S_t^i = s} G_t^i}{N}$$

根据大数定律，当 N 逼近 ∞ 时，我们可以得到确切的函数期望值

First-visit Monte Carlo policy evaluation

s may be visited multiple times in the same episode

- ▶ To evaluate state s
- ▶ The **first** time-step t that state s is visited in an episode
- ▶ Increment counter $N(s) \leftarrow N(s) + 1$
- ▶ Increment total return $S(s) \leftarrow S(s) + G_t$
- ▶ Value is estimated by mean return $V(s) = S(s)/N(s)$
- ▶ By law of large numbers, $V(s) \rightarrow V_\pi(s)$ as $N(s) \rightarrow \infty$

First-visit Monte Carlo policy evaluation

First-visit MC policy evaluation (returns $V \approx v_\pi$)

Initialize:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

Generate an episode using π

For each state s appearing in the episode:

$G \leftarrow$ return following the first occurrence of s

Append G to $Returns(s)$

$V(s) \leftarrow \text{average}(Returns(s))$

Every-visit Monte Carlo policy evaluation

s may be visited multiple times in the same episode

- ▶ To evaluate state s
- ▶ **Every** time-step t that state s is visited in an episode
- ▶ Increment counter $N(s) \leftarrow N(s) + 1$
- ▶ Increment total return $S(s) \leftarrow S(s) + G_t$
- ▶ Value is estimated by mean return $V(s) = S(s)/N(s)$
- ▶ By law of large numbers, $V(s) \rightarrow V_\pi(s)$ as $N(s) \rightarrow \infty$

MC policy evaluation: Example

undiscounted Markov Reward Process

two states A and B

transition matrix and reward function are unknown

observed two sample episodes

$A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}$

$B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$

$A + 3 \rightarrow A$ indicates a transition from state A to state A, with a reward of +3

Using **first-visit**, state-value functions $V(A)$, $V(B)$ - ?

Using **every-visit**, state-value functions $V(A)$, $V(B)$ - ?

MC policy evaluation: Example

$A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}$

$B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$

first-visit

$$V(A) = 1/2(2 + 0) = 1$$

$$V(B) = 1/2(-3 + -2) = -5/2$$

every-visit

$$V(A) = 1/4(2 + -1 + 1 + 0) = 1/2$$

$$V(B) = 1/4(-3 + -3 + -2 + -3) = -11/4$$

Monte Carlo Estimation of Action Values (Q)

expected total reward $Q^\pi(s, a) = E[\sum_{t=1}^T r_t | s, a]$

expectation of trajectory-wise rewards



sample trajectory m times,

approximate the expectation by average

$$Q^\pi(s, a) = \frac{1}{m} \sum_{i=1}^m R(\tau_i) \quad \tau_i \text{ is sample by following } \pi \text{ after } s, a$$

Incremental Mean

The mean μ_1, μ_2, \dots of a sequence x_1, x_2, \dots can be computed incrementally

$$\begin{aligned}\mu_k &= \frac{1}{k} \sum_{j=1}^k x_j \\ &= \frac{1}{k} \left(x_k + \sum_{j=1}^{k-1} x_j \right) \\ &= \frac{1}{k} (x_k + (k-1)\mu_{k-1}) \\ &= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})\end{aligned}$$

Incremental Monte Carlo Updates

Update $V(s)$ incrementally after episode $S_1, A_1, R_2, \dots, S_T$

For each state S_t with return G_t ,

$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)} (G_t - V(S_t))$$

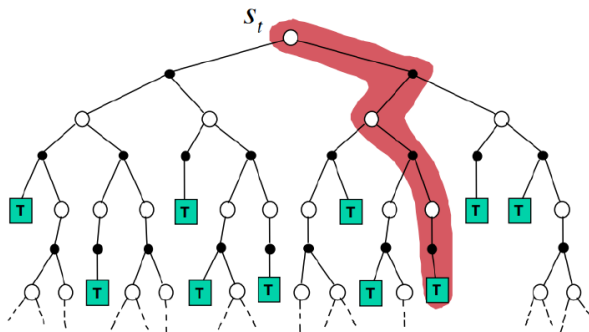
In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$

- ▶ $\alpha = \frac{1}{N(s)}$: identical to every visit MC
- ▶ $\alpha > \frac{1}{N(s)}$: forget older data, helpful for nonstationary domains

Monte Carlo Backup

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$



Dynamic Programming Backup

$$V(S_t) \leftarrow \mathbb{E}_{\pi} [R_{t+1} + \gamma V(S_{t+1})]$$

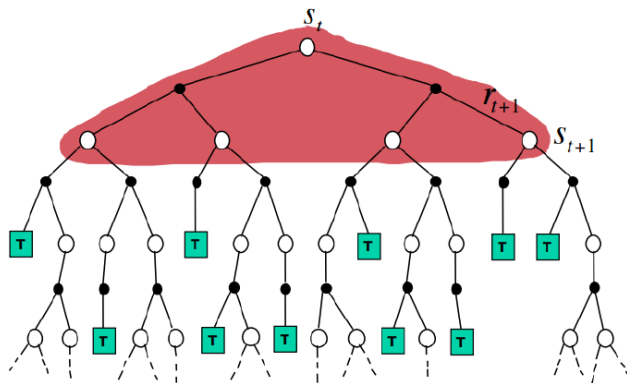


Table of Contents

课程回顾

背景

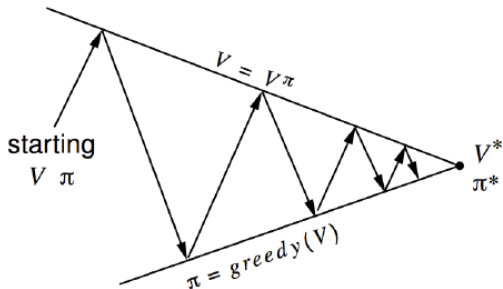
蒙特卡洛预测 (Monte Carlo Prediction)

蒙特卡洛控制 (Monte Carlo Control)

On-policy

Off-policy

Generalized Policy Iteration with Monte Carlo Evaluation



Policy evaluation Monte-Carlo policy evaluation, $V = v_\pi$?

Policy improvement Greedy policy improvement?

Model-Free Policy Iteration Using Action-Value Function

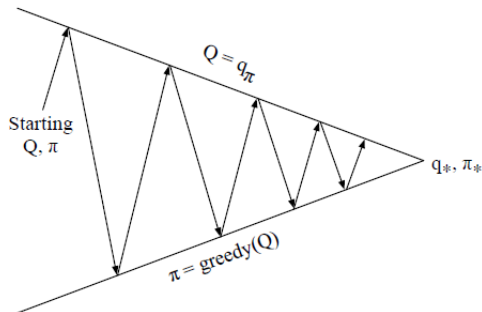
- ▶ Greedy policy improvement over $V(s)$ requires model of MDP

$$\pi'(s) = \operatorname{argmax}_{a \in A} \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V(s') \right)$$

- ▶ Greedy policy improvement over $Q(s, a)$ is model-free

$$\pi'(s) = \operatorname{argmax}_{a \in A} Q(s, a)$$

Generalised Policy Iteration with Action-Value Function



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement Greedy policy improvement?

Monte Carlo Exploring Starts

为了收敛，需要有如下假设：

1. 策略估计过程需要无限个 episode 才会收敛到回报的期望；
2. Exploring starts, 即能够保证状态集合 S 中的所有状态都是有可能被选中为每个 episode 的初始状态。

在实际的算法中这两个假设是不可能实现的，必须想法去掉这两个假设。

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize, for all $s \in S, a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow \text{arbitrary}$

$\pi(s) \leftarrow \text{arbitrary}$

$Returns(s, a) \leftarrow \text{empty list}$

Repeat forever:

Choose $S_0 \in S$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability > 0

Generate an episode starting from S_0, A_0 , following π

For each pair s, a appearing in the episode:

$G \leftarrow$ the return that follows the first occurrence of s, a

Append G to $Returns(s, a)$

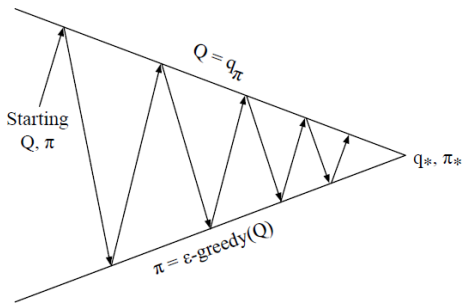
$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

For each s in the episode:

$\pi(s) \leftarrow \operatorname{argmax}_a Q(s, a)$

Monte Carlo Policy Iteration: 方法 1

- ▶ 坚持在每次策略评估的过程中接近 Q_π^k
- ▶ 虽然理论上必须有无限个 episode 来作为样本去评估 Q_π^k ，实际上做不到的，我们尽力多产生点 episode，尽可能的去接近这个收敛值

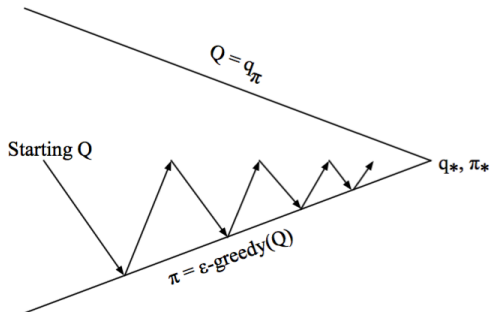


Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement ϵ -greedy policy improvement

Monte Carlo Policy Iteration: 方法 2

- ▶ 既然要很多 episode 才能收敛，那么索性就不管它收不收敛了，只做迭代改进，类似值迭代思路



Every episode:

Policy evaluation Monte-Carlo policy evaluation, $Q \approx q_\pi$

Policy improvement ϵ -greedy policy improvement

On-policy vs Off-policy

为了去掉 Exploring Starts assumption, 有两种方法:

- ▶ On-policy methods:
 - ▶ Learning while doing the job
 - ▶ Learning policy π from the episodes that generated using π
 - ▶ 生成 episode 的策略和迭代优化的策略是同一个, 并且这个策略是一种软 (soft) 策略, 即 $\pi(a | s) > 0$ for all $s \in S, a \in A$
- ▶ Off-policy methods:
 - ▶ Learning while watching other people doing the job
 - ▶ Learning policy π from the episodes generated using another policy μ
 - ▶ 分别在策略估计和策略提升的时候使用两种策略, 一个具有探索性的策略专门用于产生 episode 积累经验, 称为 behavior policy μ , 另一个则是更为贪婪, 用来学习成为最优策略的 target policy π 。要求策略 π 下的所有行为在策略 μ 下被执行过, 也就是要求对所有满足 $\pi(a | s) > 0$ 的 (s, a) 均有 $\mu(a | s) > 0$

Table of Contents

课程回顾

背景

蒙特卡洛预测 (Monte Carlo Prediction)

蒙特卡洛控制 (Monte Carlo Control)

On-policy

Off-policy

On-policy

- ▶ In On-policy control methods the policy is generally “soft”, meaning that:

$$\pi(a | s) > 0 \quad \forall s \in S, a \in A(s)$$

- ▶ ϵ -Greedy Policy Improvement:
 - ▶ All policies have a probability to be chosen, but gradually the selected policy is closer and closer to a deterministic optimal policy by controlling the ϵ value.
 - ▶ 需要指定 ϵ 值，通常这个值应当不断衰减，直到收敛到一个比较好的结果

Example of Greedy Action Selection



"Behind one door is tenure - behind the other is flipping burgers at McDonald's."

- There are two doors in front of you.
- You open the left door and get reward 0
 $V(\text{left}) = 0$
- You open the right door and get reward +1
 $V(\text{right}) = +1$
- You open the right door and get reward +3
 $V(\text{right}) = +2$
- You open the right door and get reward +2
 $V(\text{right}) = +2$
- \vdots
- Are you sure you've chosen the best door?

ϵ -贪婪搜索 (ϵ -Greedy Policy Exploration)

- ▶ ϵ -贪婪搜索的目标是使得某一个状态下所有可能的行为都有一定的几率被选择到以此来保证足够的探索，其中 $1 - \epsilon$ 的概率选择当前认为最好的行为，而 ϵ 的概率在所有可能的 m 个行为中选择 (需要注意此处包含最好的行为)。即：

$$\pi(a | s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \operatorname{argmax}_{a \in A} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

ϵ -Greedy Policy Improvement

Theorem

For any ϵ -greedy policy π , the ϵ -greedy policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s, a) \\ &= \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \max_{a \in \mathcal{A}} q_\pi(s, a) \\ &\geq \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \epsilon/m}{1 - \epsilon} q_\pi(s, a) \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a) = v_\pi(s) \end{aligned}$$

Therefore from policy improvement theorem, $v_{\pi'}(s) \geq v_\pi(s)$

Note that, π is required to be an ϵ -greedy policy.

ϵ -Greedy Policy Improvement (con't)

- ▶ $q_\pi(s, \pi'(s))$ 表示期望, 即 $\mathbb{E}_{\pi'} [q_\pi(s, \pi'(s))]$ 的缩写, 所以

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

- ▶ $\pi(a | s)$ 本身是一个 ϵ -greedy policy, 具有以下形式:

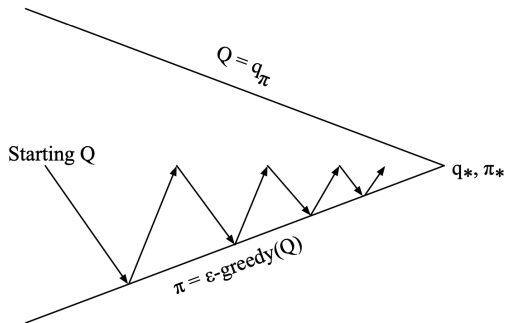
$$\pi(a | s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

只不过 $\pi(a | s) = \epsilon/m + 1 - \epsilon$ 时所对应的 a 不一定等于 $\operatorname{argmax}_{a' \in \mathcal{A}} q_\pi(s, a)$ 所对应的 a'

- ▶ “Policy improvement theorem” 指下面的推导过程, 在已知 $v_\pi(s) \leq q_\pi(s, \pi'(s))$ 的条件下得到 $v_\pi(s) \leq v_{\pi'}(s)$

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'} [R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) | S_t = s] \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma R_{t+2} + \dots | S_t = s] = v_{\pi'}(s) \end{aligned}$$

Monte-Carlo Control



Every episode:

Policy evaluation Monte-Carlo policy evaluation, $Q \approx q_\pi$

Policy improvement ϵ -greedy policy improvement

On-policy first-visit Monte Carlo control

On-policy first-visit MC control (for ε -soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε -soft policy

$Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg \max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Definition

Greedy in the Limit with Infinite Exploration (GLIE)

- All state-action pairs are explored infinitely many times,

$$\lim_{k \rightarrow \infty} N_k(s, a) = \infty$$

- The policy converges on a greedy policy,

$$\lim_{k \rightarrow \infty} \pi_k(a|s) = \mathbf{1}(a = \operatorname{argmax}_{a' \in \mathcal{A}} Q_k(s, a'))$$

- For example, ϵ -greedy is GLIE if ϵ reduces to zero at $\epsilon_k = \frac{1}{k}$

GLIE Monte-Carlo Control

- Sample k th episode using π : $\{S_1, A_1, R_2, \dots, S_T\} \sim \pi$
- For each state S_t and action A_t in the episode,

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

- Improve policy based on new action-value function

$$\epsilon \leftarrow 1/k$$

$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

Theorem

GLIE Monte-Carlo control converges to the optimal action-value function, $Q(s, a) \rightarrow q_(s, a)$*

Other ways of soft policies improvement: UCB

- ▶ UCB (Upper Confidence Bound): choose by action quality + confidence

$$a_t = \operatorname{argmax}_{a \in A} \left(Q_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}} \right)$$

- ▶ $N_t(a)$: the number of times that action a has been selected prior to time t
- ▶ c 为控制 degree of exploration 的参数
- ▶ 当 $N_t(a) = 0$, 则为最大
- ▶ UCB 中根号里面是 measure the uncertainty or variance in the estimate of a 's value, 对应 a sort of upper bound on the possible true value of action a , with c determining the confidence level.
 - ▶ a 每被选择一次, $N_t(a)$ 变大, 则 a 的 uncertainty 变小; 每次 t 增加, $N_t(a)$ 不变 (没有选择 a), 则 a 的 uncertainty 变大
 - ▶ 每个 action 都会被 explore, 不过 actions with lower value estimates 或者已经被选择过多次, 则较少概率被选
 - ▶ 对比的实验效果 UCB 比 ϵ -greedy 效果好; 处理 non stationary problems 更复杂没有 ϵ -greedy 简单有效

Table of Contents

课程回顾

背景

蒙特卡洛预测 (Monte Carlo Prediction)

蒙特卡洛控制 (Monte Carlo Control)

On-policy

Off-policy

Off-policy

- ▶ Learning policy π by following the data generated using policy μ
- ▶ Why is it important?
 - ▶ Learn from observing humans or other agents
 - ▶ Re-use experience generated from old policies
 - ▶ Learn about optimal policy while following exploratory policy
- ▶ We call
 - ▶ π the **target policy**: the policy being learned about
 - ▶ μ the **behavior policy**: the policy generates the moves
- ▶ **Assumption of coverage**: for all $s \in S$ and $a \in A(s)$

$$\pi(a \mid s) > 0 \text{ implies } \mu(a \mid s) > 0$$

Every action which is taken under policy π must have a non-zero probability to be taken as well under policy μ

- ▶ Typically the target policy π would be a greedy policy with respect to the current action-value function

重要性采样 (Importance Sampling)

- ▶ 对 Off-policy, 采样并不来自于当前的策略, 通常采用重要性采样 (Importance Sampling), *i.e.*, 给定服从一种分布的样本情况下, 估计另外一种分布下期望值的一般方法
- ▶ 重要性采样原理
 - ▶ 重要性采样源自求期望:

$$E[f] = \int p(x) f(x) dx$$

当随机变量 x 的分布非常复杂时, 无法利用解析的方法产生用于逼近期望的样本

- ▶ 我们可以选用一个概率分布很简单, 产生样本很容易的概率分布 $q(x)$, 比如正态分布。原来的期望可变为:

$$E[f] = \int q(x) \frac{p(x)}{q(x)} f(x) dx$$

- ▶ 定义重要性权重: $\omega^n = \frac{p(x^n)}{q(x^n)}$, 普通的重要性采样结果为:

$$E[f] = \frac{1}{N} \sum_n \omega^n f(x^n)$$

重要性采样 (Importance Sampling)

importance sampling:

$$\begin{array}{ccc} E[f] = \int_x p(x) f(x) dx & = & \int_x q(x) \frac{p(x)}{q(x)} f(x) dx \\ \downarrow \text{sample from } p & & \downarrow \text{sample from } q \\ \frac{1}{m} \sum_{i=1}^m f(x) & & \frac{1}{m} \sum_{i=1}^m \frac{p(x)}{q(x)} f(x) \end{array}$$

Off-policy 预测问题中的重要性采样

- 根据轨迹在 target policy 和 behavior policies 下发生的相关概率来对 returns 赋予权重，给定初始状态 S_t ，state-action 轨迹为 $A_t, S_{t+1}, A_{t+1}, \dots, S_T$ 在策略 π 下发生的概率为

$$\prod_{k=t}^{T-1} \pi(A_k | S_k) T(S_k, A_k, S_{k+1})$$

其中 $T(S_k, A_k, S_{k+1})$ 为状态转移函数

- 轨迹在 target policy 和 behavior policy 下发生的相关概率 (即 importance-sampling ratio) 为

$$\rho_{t:T-1} = \frac{\prod_{k=t}^{T-1} \pi(A_k | S_k) T(S_k, A_k, S_{k+1})}{\prod_{k=t}^{T-1} \mu(A_k | S_k) T(S_k, A_k, S_{k+1})} = \prod_{k=t}^{T-1} \frac{\pi(A_k | S_k)}{\mu(A_k | S_k)}$$

可以看到， $\rho_{t:T-1}$ 与 MDP 没有关系，仅与两个策略相关

Off-policy 预测问题中的重要性采样 (con't)

- 我们要求 target policy π 下回报的期望，但现在我们是根据 behavior policy μ 生成的 episode，得到的自然是一个错误的期望值 $E[G_t|S_t] = V_\mu(S_t)$ ，但我们要的是 $V_\pi(S_t)$ ，所以基于重要性采样方法利用 $\rho_{t:T-1}$

$$E[\rho_{t:T-1} G_t | S_t] = V_\pi(S_t)$$

- 定义 $\mathcal{T}(s)$: 对 every-visit 方法，它代表所有状态 s 被 visit 的时刻的集合，对 first-visit 方法，它仅代表所有状态 s 在某个 episode 中第一次被 visit 的时刻的集合


t = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

$$\mathcal{T}(s) = \{4, 15\} \quad T(4) = 7, T(15) = 19$$

Off-policy 预测问题中的重要性采样 (con't)

- ▶ 一般重要性采样 (Ordinary importance sampling):

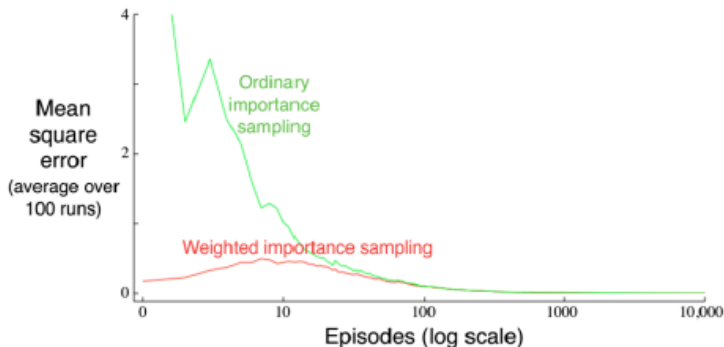
$$V(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}$$

- ▶ 加权重要性采样 (Weighted importance sampling):

$$V(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}$$

一般重要性采样 vs. 加权重要性采样

- In practice the weighted estimator has dramatically lower variance and is therefore strongly preferred.



增量式求均值

- 假设我们得到了一系列回报 G_1, G_2, \dots, G_{t-1} ，对于 off-policy 来说，因为我们利用了重要性采样，所以多了一个权重的因素，设每个回报的权重为 W_k

$$V_n = \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}$$

- 转换为增量式：

$$\begin{aligned} V_{n+1} &= \frac{\sum_k^{n+1} W_k G_k}{\sum_k^{n+1} W_k} = \frac{\sum_k^n W_k G_k + W_{n+1} G_{n+1}}{\sum_k^n W_k} \frac{\sum_k^n W_k}{\sum_k^{n+1} W_k} = V_n \frac{C_{n-1}}{C_n} + W_n \frac{G_n}{C_n} \\ &= V_n + \frac{W_n G_n - V_n W_n}{C_n} \\ &= V_n + \frac{W_n}{C_n} (G_n - V_n) \end{aligned}$$

- 结果：

$$V_{n+1} = V_n + \frac{W_n}{C_n} (G_n - V_n)$$

$$C_{n+1} = C_n + W_{n+1}$$

Off-policy Monte Carlo prediction

Off-policy MC prediction, for estimating $Q \approx q_\pi$

Input: an arbitrary target policy π

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

$C(s, a) \leftarrow 0$

Repeat forever:

$b \leftarrow$ any policy with coverage of π

Generate an episode using b :

$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$

$G \leftarrow 0$

$W \leftarrow 1$

For $t = T - 1, T - 2, \dots$ down to 0:

$G \leftarrow \gamma G + R_{t+1}$

$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$

$W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$

If $W = 0$ then exit For loop

Off-policy Monte Carlo control

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

$C(s, a) \leftarrow 0$

$\pi(s) \leftarrow \operatorname{argmax}_a Q(S_t, a)$ (with ties broken consistently)

Repeat forever:

$b \leftarrow$ any soft policy

Generate an episode using b :

$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$

$G \leftarrow 0$

$W \leftarrow 1$

For $t = T - 1, T - 2, \dots$ down to 0:

$G \leftarrow \gamma G + R_{t+1}$

$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$

$\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$ (with ties broken consistently)

If $A_t \neq \pi(S_t)$ then exit For loop

$W \leftarrow W \frac{1}{b(A_t|S_t)}$

Pros and cons of MC

MC has several advantages over DP:

- Can learn V and Q directly from interaction with environment (using episodes!)
- No need for full models (using episodes!)
- No need to learn about ALL states (using episodes!)

However, there are some limitations:

- MC only works for episodic (terminating) environments
- MC learns from complete episodes, so no bootstrapping
- MC must wait until the end of an episode before return is known

Next lecture

Solution: Temporal-Difference

- TD works in continuing (non-terminating) environments
- TD can learn online after every step
- TD can learn from incomplete sequences

小结

- ▶ 蒙特卡洛的算法不是一个效率很高的算法，但是能够展现免模型类算法的特性
- ▶ 先进行策略评估，之后找到一个方向改进算法；为了使策略能够有效更新，需要引入对环境的探索；对环境的探索，有 On/Off-policy 两种方法
- ▶ 蒙特卡洛的算法有一个很显然的缺陷：一定要拿到整个轨迹以后，才能更新模型