Subject: Reinforcement Learning, Westlake University

Course Speaker: Shiyu Zhao

Note Author: Changrui Fang

Date: from December 20, 2024 to February 19, 2025

Contents

1 Lecture 1, Basic Concepts

Bilibili:Lecture 1, Basic Concepts

Basic Concepts – under a grid world example

State: grid world 中指的是位置(location)，记为s_i.

State Space: 状态的集合，记为$\mathcal{S} = \{s_i\}_{i=1}^n$.

Action: 记为a_i.

Action Space of state: 记为$\mathcal{A}(s_j) = \{a_i\}_{i=1}^k$，其依赖于状态.

Station Transition: 某一个状态采取了行动转变成另一个状态，例如记为$s_1 \xrightarrow{a_2} s_2$.其定义了 agent 与 environment 交互的行为，可以用表格(tabular)的形式表现出来，但只能表现确定性的情况(deterministic case).

State transition probability: 使用概率表示 state transition，如$\mathbb{P}(s_2|s_1, a_2) = 0.9$表示在 state s_1时采取 action a_2 变成 state s_2 的概率.这样就可以将 deterministic case 变为 stochastic case.

Policy: 告诉 agent 在某状态时应该采取什么行动，记为π，其是一串和为一对条件概率.例如在 state s_1时可以采用3种 action，记为$\pi(a_1|s_1) = 0, \pi(a_2|s_1) = 1, \pi(a_3|s_1) = 0$，那么说明一定会采取 action a_2. Policy 也可以使用表格(tabular)形式表达.

Reward: 一个时数标量，一般当为正时表示对模型的奖励(encouragement)，为负时表示对模型的惩罚(punishment)[a].其实际上是人与机器交互的手段(human machine interface). Reward 也可以使用表格(tabular)形式表达.同时，reward 也可以用条件概率以随机化，例如$\mathbb{P}(r = -1|s_1, a_1) = 0.9$表示在 state s_1 时采取 action a_1 产生的 reward 为 $r = -1$ 的概率为0.9.注意 reward 取决于当前的状态和动作，并不取决于下一步的状态，如原地不动和"撞墙"下一步的状态相同，但是 reward 一般不同.

Trajectory: 是一个 state - action - reward chain，可以用来评估一个 policy 的好坏。例如

$$s_1 \xrightarrow[r=0]{a_2} s_2 \xrightarrow[r=0]{a_3} s_5 \xrightarrow[r=0]{a_3} s_8 \xrightarrow[r=1]{a_2} s_9$$

Return: 针对一个 Trajectory，将所有的 reward 累和.

Discounted return: 一个 trajectory 有时是无穷的，例如当达到目的地后可能"原地踏步"，如果此时仍然有施加 reward 就会导致Return $\to \infty$，此时可以给每一步的 reward 施加一个"折扣"(discount) $\gamma \in [0, 1)$，第 k 个 reward r_k 变成$\gamma^k r_k$，由于$\gamma < 1$，所以最后的 return 不会发散. 并且通过控制 $gamma$ 的大小，可以控制模型更关注前面的(γ 小) action 还是未来的(γ 大) action.

Episode/Trial: 通常当一个任务会终止时（存在 terminal states），即有限长度的 Trajectory 被称为 Episode/Trial.这样的任务被称为 episodic task. 当不存在 terminal states 时，这种任务被称为 continuing tasks[b].

[a] 不设置/设置 reward=0 就是不惩罚，也意味着鼓励；鼓励也可以是负，惩罚相应变成正，本质上数学是一样的

[b] 一般来说没有绝对的一直持续的任务，但是根据解决问题的时间尺度可以这样设置. 事实上有统一的方法描述 episodic 和 continuing task，例如1. 到达 target state 后一直"原地踏步"，并且同时设置之后的 reward=0；2. 就把 target state 当成一个普通的 state，这样有可能还能跳出局部较优的 state这种方式更具一般化，不把 target 区别对待.本课程使用2.

Markov decision process (MDP)

Key elements of MDP:

- Sets:
  - State: the set of states $\mathcal{S}$
  - Action: the set of actions $\mathcal{A}(s)$ is associated for state $s \in \mathcal{S}$.
  - Reward: the set of rewards $\mathcal{R}(s, a)$.
- Probability distribution:
  - State transition probability: at state s, taking action a, the probability to transit to state s' is $p(s'|s, a)$
  - Reward probability: at state s, taking action a, the probability to get reward r is $p(r|s, a)$
- Policy: at state s, the probability to choose action a is $\pi(a|s)$
- *Markov property*: memoryless property

$$p(s_{t+1}|a_{t+1}, s_t, \ldots, a_1, s_0) = p(s_{t+1}|a_{t+1}, s_t),$$
$$p(r_{t+1}|a_{t+1}, s_t, \ldots, a_1, s_0) = p(r_{t+1}|a_{t+1}, s_t).$$

All the concepts introduced in this lecture can be put in the framework in MDP.

Figure 1: MDP

当 Markov decision process 中的 decision/policy 确定了，那么其就和马尔可夫过程(Markov process)融为一体了

2 Lecture 2, Bellman Equation

Bilibili:Lecture 2, Bellman Equation

Outline

A core concept: state value

A fundamental tool: the Bellman equation

Contents:

1. Motivating examples
2. State value
3. Bellman equation: Derivation
4. Bellman equation: Matrix-vector form
5. Bellman equation: Solve the state values
6. Action value
7. Summary

Motivating examples – Grid Space

Q1: Why return is important?

A1: Retun 可以评估一个 policy. 这是将我们对一个策略好坏的 直觉(intuition) 进行数学化的重要定量工具. 只有量化了一个 policy 我们才能不断改进策略.

Q2: How to calculate return?

A2:

1. By definition: 不断累和
2. Bootstrapping: 当前 state 的 return 依赖于其他 state 的 return，最后循环回到自身. 这样将所有的 state 组合起来就通过矩阵形式求解.(也就可以得到一般的 Bellman equation)

State Value

Single-step process:

$$S_t \xrightarrow{A_t} R_{t+1}, S_{t+1} \tag{1}$$

其中大写字母是因为表示随机变量(random variables)，引入随机性.

- $S_t \to A_t$ is governed by $\pi(A_t = a \mid S_t = s)$，即采取某 action 的概率
- $S_t, A_t \to R_{t+1}$ is governed by $p(R_{t+1} = r \mid S_t = s, A_t = a)$，表示第$t$-state 采取 action 后获得某 reward 的概率
- $S_t, A_t \to S_{t+1}$ is governed by $p(S_{t+1} = s' \mid S_t = s, A_t = a)$，表示第$t$-state 采取 action 后获得变成某 state 的概率

Multi-step (random) trajectory:

$$S_t \xrightarrow{A_t} R_{t+1}, S_{t+1} \xrightarrow{A_{t+1}} R_{t+2}, S_{t+2} \xrightarrow{A_{t+2}} R_{t+3}, \dots \tag{2}$$

Discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \tag{3}$$

State value: G_t 的期望(expectation / expected alue / mean)，全称 state-value function:

$$v_\pi(s) = \mathbb{E}[G_t \mid S_t = s] \tag{4}$$

注意，state value 是 state s 的函数，并且依赖于 policy π，如果 $v_\pi(s)$ 更大，那么说明对于该 state 此 policy 较好.

Q: What is the relationship between return and state value?

A:

1. state value 是所有情况下的 return 求和后 G_t 的期望；return 仅针对一个单独的 trajectory，而 state value 是全部可能的 trajectory.
2. 当没有随机性时，即只有一条确定性的 trajectory 时，return 的累和 G_t 就与 state value 相同.

Bellman Equation – Elementwise form

对于Multi-step (random) trajectory:

$$S_t \xrightarrow{A_t} R_{t+1}, S_{t+1} \xrightarrow{A_{t+1}} R_{t+2}, S_{t+2} \xrightarrow{A_{t+2}} R_{t+3}, \dots$$

Discounted return 可以写为:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) \\ &= R_{t+1}(\text{immediate reward}) + \gamma G_{t+1}(\text{future reward}) \end{aligned} \tag{5}$$

进而 state value 可以写为:

$$v_\pi(s) = \mathbb{E}[G_t|S_t = s] = \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] = \mathbb{E}[R_{t+1}|S_t = s] + \gamma\mathbb{E}[G_{t+1}|S_t = s] \tag{6}$$

其中第一项为 **mean of immediate rewards**，为:

$$\mathbb{E}[R_{t+1} \mid S_t = s] = \sum_a \pi(a \mid s)\mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] = \sum_a \pi(a \mid s) \sum_r p(r \mid s, a) \cdot r \tag{7}$$

第二项为 **mean of future reward**，为:

$$\begin{aligned} \mathbb{E}[G_{t+1}|S_t = s] &= \sum_{s'} \mathbb{E}[G_{t+1}|S_t = s, S_{t+1} = s']p(s'|s) \\ &\xrightarrow[\text{process property}]{\text{Markov decision}} \sum_{s'} \mathbb{E}[G_{t+1}|S_{t+1} = s']p(s'|s) \\ &= \sum_{s'} v_\pi(s')p(s'|s) = \sum_{s'} v_\pi(s') \sum_a p(s'|s, a)\pi(a|s) \end{aligned} \tag{8}$$

最终就可以得到如下的 **Bellman Equation**:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[R_{t+1}|S_t = s] + \gamma \mathbb{E}[G_{t+1}|S_t = s] \\ &= \underbrace{\sum_a \pi(a|s) \sum_r p(r|s,a) r}_{\text{mean of immediate rewards}} + \underbrace{\gamma \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) v_\pi(s')}_{\text{mean of future rewards}} \\ &= \sum_a \pi(a|s) \left[\sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_\pi(s') \right], \quad \forall s \in \mathcal{S} \end{aligned} \tag{9}$$

- Bellman equation 描述了不同 state 的 state-value function 的关系
- Bellman equation 是一族方程，其包含了状态空间$\mathcal{S}$中全部 state，共 $|\mathcal{S}|$个方程，求解方法就是 Bootstrapping
- $\pi(a|s)$ 是一个给定的 policy，因此求解 Bellman equation 也被称为 policy evalution，即评价一个 policy 的好坏
- $p(r|s,a)$和$p(s'|s,a)$表示 dynamic model / environment model，这个 model 有时知道有时不知道

Bellman equation – Matrix-vector form

Bellman equation (9) 一共有 $|\mathcal{S}|$ 个公式，无法单独求解，但是将他们组合起来就可以得到一组线性方程，即求解一个线性方程组，也就是 matrix-vecotr form.

将式(9)改写为：

$$v_\pi(s) = r_\pi(s) + \gamma \sum_{s'} p_\pi(s'|s) v_\pi(s') \tag{10}$$

其中

$$r_\pi(s) \triangleq \sum_a \pi(a|s) \sum_r p(r|s,a) r \text{(average of immediate reward)}$$

$$p_\pi(s'|s) \triangleq \sum_a \pi(a|s) p(s'|s,a)$$

对于n个状态的状态空间$\mathcal{S} = \{s_1, s_2, \cdots, s_n\}$，Bellman equation可写为：

$$v_\pi(s_i) = r_\pi(s_i) + \gamma \sum_{s_j} p_\pi(s_j \mid s_i) v_\pi(s_j) \tag{11}$$

写成 matrix-vector form 就是：

$$\boldsymbol{v}_\pi = \boldsymbol{r}_\pi + \gamma \boldsymbol{P}_\pi \boldsymbol{v}_\pi \tag{12}$$

其中

- $\boldsymbol{v}_\pi = [v_\pi(s_1), \ldots, v_\pi(s_n)]^T \in \mathbb{R}^n$
- $\boldsymbol{r}_\pi = [r_\pi(s_1), \ldots, r_\pi(s_n)]^T \in \mathbb{R}^n$
- $\boldsymbol{P}_\pi \in \mathbb{R}^{n \times n}$，其中$[\boldsymbol{P}_\pi]_{ij} = p_\pi(s_j|s_i)$，为状态转移矩阵(state transition matrix)

写成矩阵形式就是

$$\underbrace{\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_n) \end{bmatrix}}_{\boldsymbol{v}_\pi} = \underbrace{\begin{bmatrix} r_\pi(s_1) \\ r_\pi(s_2) \\ \vdots \\ r_\pi(s_n) \end{bmatrix}}_{\boldsymbol{r}_\pi} + \gamma \underbrace{\begin{bmatrix} p_\pi(s_1|s_1) & \cdots & p_\pi(s_{n-1}|s_1) & p_\pi(s_4|s_1) \\ p_\pi(s_1|s_2) & \cdots & p_\pi(s_{n-1}|s_2) & p_\pi(s_4|s_2) \\ \vdots & \ddots & \vdots & \vdots \\ p_\pi(s_1|s_n) & \cdots & p_\pi(s_{n-1}|s_4) & p_\pi(s_n|s_n) \end{bmatrix}}_{\boldsymbol{P}_\pi} \underbrace{\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_n) \end{bmatrix}}_{\boldsymbol{v}_\pi}.$$

Bellman equation: Solve the state values

第一种求解方式就是直接求其闭式解(closed-form solution)，为

$$\boldsymbol{v}_\pi = (I - \gamma \boldsymbol{P}_\pi)^{-1} \boldsymbol{r}_\pi \tag{13}$$

但是实际中大规模矩阵求逆矩阵很困难，仍然需要使用特殊的数值方法.

第二种方法是迭代法找其迭代解(iterative solution)：

$$\boldsymbol{v}_{k+1} = \boldsymbol{r}_\pi + \gamma \boldsymbol{P}_\pi \boldsymbol{v}_k \tag{14}$$

最终可以证明(证明过程略[a])

$$\boldsymbol{v}_k \xrightarrow{k\to\infty} \boldsymbol{v}_\pi$$

Note 1. *1.* 计算 *state value* 可以评价一个 *policy* 好不好

2. 不同的 *policy* 也可以得到相同的 *state value*

[a] 这其实就是一个不动点迭代，Banach不动点定理

Action value

State value 与 Action value 辨析

- state value 是指一个 agent 从一个 state 出发能得到的 average return，定义为

$$v_\pi(s) = \mathbb{E}[G_t \mid S_t = s]$$

- action value 是指一个 agent 从一个 state 出发并且做出一个 action 后能得到的 average return

Action value 定义为

$$q_\pi(s,a) = \mathbb{E}[G_t | S_t = s, A_t = a] \tag{15}$$

- $q_\pi(s,a)$ 是 state-action pair (s,a) 的函数
- $q_\pi(s,a)$ 依赖于 policy π

由于

$$\underbrace{\mathbb{E}[G_t|S_t=s]}_{v_\pi(s)}=\sum_a \underbrace{\mathbb{E}[G_t|S_t=s,A_t=a]}_{q_\pi(s,a)}\pi(a|s)$$

$$=\sum_a \pi(a|s)\underbrace{\left[\sum_r p(r|s,a)r+\gamma\sum_{s'}p(s'|s,a)v_\pi(s')\right]}_{q_\pi(s,a)}$$

因此

$$v_\pi(s)=\sum_a \pi(a|s)q_\pi(s,a) \tag{16a}$$

$$q_\pi(s,a)=\sum_r p(r|s,a)r+\gamma\sum_{s'}p(s'|s,a)v_\pi(s') \tag{16b}$$

式(16a)和(16b)就像一个硬币的两面，式(16a)说明可以从 action value 获得 state value，式(16b)说明可以从 state value 获得 action value.

Note 2. 当一个确定性的 *policy* 中在某个 *state* 只有一个 *action* 时，其他的 *action* 产生的 *action value* 并不是*0*，而是也可以计算，此时的 *immediate reward* 一般为*0*，但是仍然有 *future reward*。这样就可以与这个确定的 *action* 比较看是否这个 *action* 是好的.

Last updated: December 19, 2024

3 Lecture 3, Optimal Policy and Bellman Optimality Equation

Bilibili:Lecture 3, Bellman Optimality Equation

Outline

2 core concepts: optimal state value, optimal policy

1 fundamental tool: Bellman optimality equation(BOE)

Optimal Policy

State value 可以衡量一个 policy 的好坏，回忆其定义为：

$$v_\pi(s) = \mathbb{E}[G_t \mid S_t = s]$$

Definition 1 (Better policy)**.** 对于两个 *policy* π_1, π_2，若他们的 *state value* 有如下关系：

$$v_{\pi_1}(s) \geqslant v_{\pi_2}(s), \quad \forall s \in \mathcal{S}$$

则称 *policy* π_1 好于 π_2，其中 $\mathcal{S}$ 为 *state space.*

Definition 2 (optimal policy)**.** 若 *policy* π^* 的 *state value* 有：

$$v_{\pi^*}(s) \geqslant v_\pi(s), \quad \forall s \in \mathcal{S}, \forall \pi$$

则称 *policy* π^* 最优*(optimal).*

这就产生了如下问题（都可以用贝尔曼最优公式回答）：

1. optimal policy 是否存在？若存在，是否唯一？
2. optimal policy 是确定性的(deterministic) 还是随机性的(stochastic)？
3. optimal policy 如何得到？

Bellman optimality equation(BOE)

先回忆 Bellman equation 式(9):

$$\begin{aligned} v_\pi(s) &= \underbrace{\sum_a \pi(a|s) \sum_r p(r|s,a) r}_{\text{mean of immediate rewards}} + \underbrace{\gamma \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) v_\pi(s')}_{\text{mean of future rewards}} \\ &= \sum_a \pi(a|s) \left[\sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_\pi(s') \right], \quad \forall s \in \mathcal{S} \end{aligned}$$

Bellman optimality equation 则是将 policy π固定为最优的（elementwise form）：

$$\begin{aligned} v(s) &= \max_\pi \sum_a \pi(a|s) \left(\sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v(s') \right), \quad \forall s \in \mathcal{S} \\ &\triangleq \max_\pi \sum_a \pi(a|s) q(s,a) \end{aligned} \tag{17}$$

写成 matrix-vector form 就是：

$$\boldsymbol{v} = \max_{\pi}(\boldsymbol{r}_\pi + \gamma \boldsymbol{P}_\pi \boldsymbol{v}) \tag{18}$$

$$\begin{aligned} [r_\pi]_s &\triangleq \sum_a \pi(a|s) \sum_r p(r|s,a) r, \\ [P_\pi]_{s,s'} &\triangleq p(s'|s) \triangleq \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) \end{aligned} \tag{19}$$

其中

$$\max_{\pi} \underbrace{\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_n) \end{bmatrix}}_{\max_\pi \boldsymbol{v}_\pi} = \begin{bmatrix} \max_\pi v_\pi(s_1) \\ \max_\pi v_\pi(s_2) \\ \vdots \\ \max_\pi v_\pi(s_n) \end{bmatrix}$$

Note 3. *1.* $p(r|s,a), p(s'|s,a)$ 均已知

2. $v(x)$, $v(s')$ 均未知且需求解

*3. Bellman equation (9)*中 *policy* 固定，但 *BOE(17)*中需要求解出最优 *policy*

How to maximize the right-hand side of BOE

由式18可以看出，式中有两个未知量需要求解，即$\boldsymbol{v}$和π，也就是说一个式子求两个位置量：

$$\boldsymbol{v} = \max_{\pi}(\boldsymbol{r}_\pi + \gamma \boldsymbol{P}_\pi \boldsymbol{v})$$

解决方式是先固定住$\boldsymbol{v}$，将其看作常量，然后对式子整体求π能让整个式子达到最大的取值，固定之后再通过等式两边求解$\boldsymbol{v}$即可.

Note 4. 例如：

$$x = \max_a(2x - 1 - a^2)$$

首先固定x，显然当$a = 0$时整个式子才可能有最大值，那么就得到

$$x = 2x - 1 \quad \Rightarrow \quad x = 1$$

在式17中，实际上会对$v(s')$赋予初始值，这样实际上$q(s,a)$就是已知的，那么只需要确定$\pi(a|s)$即可.

$$\begin{aligned} v(s) &= \max_\pi \sum_a \pi(a|s) \left(\sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v(s') \right), \quad \forall s \in \mathcal{S} \\ &\triangleq \max_\pi \sum_a \pi(a|s) q(s,a) \end{aligned}$$

由于$\sum_a \pi(a \mid s) = 1$，那么实际上只需取最大的$q(s,a)$即可：

$$\max_\pi \sum_a \pi(a|s) q(s,a) = \max_{a \in \mathcal{A}(s)} q(s,a)$$

$$\pi(a \mid s) = \begin{cases} 1, & a = a^* \\ 0, & a \neq a^* \end{cases}, \quad a^* = \arg\max_a q(s,a).$$

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

Solve the BOE

根据我们最大化 BOE 右端项的思想，我们先固定v不动，再找到能够使整个式子最大的π，最后整个式子只剩下一个变量v，此时我们将其记为

$$f(v) := \max_\pi (r_\pi + \gamma P_\pi v) \tag{20}$$

这样 BOE 就变成

$$v = f(v), \quad [f(v)]_s = \max_\pi \sum_a \pi(a|s) q(s,a), \quad s \in \mathcal{S}$$

根据压缩映射原理(Contraction Mapping Theorem)，对于形如$x = f(x)$ 的方程，如果 f 是压缩映射，那么有如下结论：

1. 不动点$x^* = f(x^*)$，即方程的解是存在唯一的
2. 算法$x_{k+1} = f(x_k)$可以不断逼近此不动点，且指数收敛

事实上，BOE 中的映射 f 刚好是一个压缩映射（证明后补），那么自然地就可以得到如下重要定理：

Theorem 1 (Existence, Uniqueness, and Algorithm)**.** 对于 *BOE* $v = f(v) = \max_\pi (r_\pi + \gamma P_\pi v)$，总存在唯一解，且该解可以被如下方式迭代逼近：

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

且产生的序列 $\{v_k\}$ 指数收敛至不动点 v^*，收敛速率由 γ 控制.

Policy optimality

当我们求出了最优 policy v^* 后，v^* 显然满足

$$v^* = \max_\pi (r_\pi + \gamma P_\pi v^*)$$

注意此时的 π 都是固定的最优的，不妨将其记为 π^*，那么为达到 max，其满足

$$\pi^* = \arg\max_\pi (r_\pi + \gamma P_\pi v^*)$$

这样就将 BOE 转化为一个特殊的 BE：

$$v^* = r_{\pi^*} + \gamma P_{\pi^*} v^* \tag{21}$$

因此说 BOE 是特殊情形的 BE.

此处还有一个结论（证明暂略）：

Theorem 2. v^* 是最优 *state value*，即

$$v^* \geqslant v_\pi, \forall \pi, \forall v$$

根据前面的分析，最优的 policy π^*满足：

Theorem 3 (Greedy Optimal Policy). $\forall s \in \mathcal{S}$，*BOE* 的最优 *policy* (也称为 *deterministic greedy policy*)为：

$$\pi^*(a \mid s)_\star = \begin{cases} 1 & \text{if } a = a^*(s) \\ 0 & \text{if } a \neq a^*(s) \end{cases} \tag{22}$$

其中

$$a^*(s) = \arg\max_a q^*(a, s)$$
$$q^*(s, a) := \sum_r p(r \mid s, a) r + \gamma \sum_{s'} p(s' \mid s, a) v^*(s')$$

Analyzing optimal policies

对于 BOE

$$v(s) = \max_\pi \sum_a \pi(a|s) \left(\sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v(s') \right)$$

黑色部分是未知并需求解的量，红色部分为已知量，可能对最终结果造成影响，其中

1. r: 预先设计的 reward
2. $p(r|s,a), p(s'|s,a)$: 概率模型 / 系统模型(system model)
3. γ: discount rate

由于系统的模型一般难以改变，所以下面仅分析 r 和 γ 的改变对 BOE 结果的影响.

根据一些简答的例子可以发现：

1. 当 γ 比较大时，agent 会比较"远视"，重视未来的 reward；较小时，会比较"近视"(short-sighted)，重视较近的 reward
2. 当对所有的 reward 作线性变换$r \to ar + b$时，optimal policy 并不会改变，因为重要的是不同 reward 相互间的相对差异(relative value)，而不是绝对差异

Theorem (Optimal Policy Invariance)

Consider a Markov decision process with $v^* \in \mathbb{R}^{|\mathcal{S}|}$ *as the optimal state value satisfying* $v^* = \max_\pi(r_\pi + \gamma P_\pi v^*)$. *If every reward* r *is changed by an affine transformation to* $ar + b$, *where* $a, b \in \mathbb{R}$ *and* $a \neq 0$, *then the corresponding optimal state value* v' *is also an affine transformation of* v^*:

$$v' = av^* + \frac{b}{1-\gamma}\mathbf{1},$$

where $\gamma \in (0, 1)$ *is the discount rate and* $\mathbf{1} = [1, \ldots, 1]^T$. *Consequently, the optimal policies are invariant to the affine transformation of the reward signals.*

Figure 2: Optimal Policy Invariance

Theorem 4 (Optimal Policy Invariance)**.**

discount rate 的存在使得一些无意义的绕远路(meaningless detour)的策略被 pass，因为这样的 reward 会被延后且“打折”

Last updated: January 16, 2025

4 Lecture 4, Value Iteration and Policy Iteration Algorithms

Bilibili:Lecture 4, Value Iteration and Policy Iteration Algorithms

Outline

Outline
1. Value iteration algorithm
2. Policy iteration algorithm
3. Truncated policy iteration algorithm
   （Value iteration 和Policy iteration 是其两种极端情况）

4.1 Value iteration algorithm

Value iteration algorithm

在 Lecture 3 的 Theorem1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 elementwise form 为

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

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 5. 注意，此处的v_k并非 *state value*，因为在 *value update* 中左右两端并非都是 v_k，因此不是一个 *Bellman equation*，自然就不是 *state value*，而仅仅是一个值.

Summary

总结以上两个步骤，计算流程为：

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

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.
Aim: Search for the optimal state value and an optimal policy solving the Bellman optimality equation.
while $\|v_k - v_{k-1}\| >$ threshold **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:

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

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

4.2 Policy iteration algorithm

Example

首先我们直接给出完整的 Policy iteration algorithm 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.
Aim: Search for the optimal state value and an optimal policy.
while the policy has not converged **do**
    Step 1: Policy evaluation(PE): ▷ to calculate the state value of π_k
    Initialization: an arbitrary initial guess $v_{\pi_k}^{(0)}$
    while $v_{\pi_k}^{(j)}$ has not converged **do**
        for every state $s \in S$ **do**
            $$v_{\pi_k}^{(j+1)}(s) \leftarrow \sum_a \pi_k(a \mid s)\left[\sum_r p(r \mid s,a)r + \gamma \sum_{s'} p(s' \mid s,a) v_{\pi_k}^{(j)}(s')\right]$$
        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

此算法会先后产生如下序列:

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

Note 6. 对于此算法需要有以下说明:

1. 在 *Step 1 PE* 中，计算 *state value* 时需要求解 *Bellman equation*

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

   此时由于闭式解*(closed-form solution)*$v_{\pi_k} = (I - \gamma P_{\pi_k})^{-1} r_{\pi_k}$需要求矩阵的逆，因此转而使用算法中的迭代方法。所以在 *Policy iteration algorithm* 这个大的迭代算法中又嵌套有小的迭代算法.

2. 可以证明在 *PI* 中迭代产生的下一步策略比当前策略确实要更好*(*证明暂略*)*:

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

3. 可以证明此迭代算法最终可以找到最优 *policy*(证明暂略):

   Theorem 5 (Convergence of policy iteration)**.** *policy iteration algorithm* 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.*

4. *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

5. 通过例子可以发现，在迭代过程中，接近目标的状态会先变好，远离目标的状态会后变好

4.3 Truncated policy iteration

Comparison of Policy iteration and Value iteration

Policy iteration: start from π_0

1. Policy evaluation(PE):
$$v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$$
2. Policy improvement(PI):
$$\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_k)$$

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

Value iteration: start from v_0

1. Policy update(PU):
$$\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_k)$$
2. Value update(VU):
$$v_{\pi_{k+1}} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k}$$

$$u_0 \xrightarrow{PU} \pi_1' \xrightarrow{VU} u_1 \xrightarrow{PU} \pi_2' \xrightarrow{VU} u_2 \xrightarrow{PU} \cdots$$

	Policy iteration algorithm	**Value iteration algorithm**	**Comments**
1) Policy:	π_0	N/A	
2) Value:	$v_{\pi_0} = r_{\pi_0} + \gamma P_{\pi_0} v_{\pi_0}$	$v_0 := {v_{\pi_0}}^{1}$	
3) Policy:	$\pi_1 = \arg\max_\pi \left(r_\pi + \gamma P_\pi v_{\pi_0}\right)$	$\pi_1 = \arg\max_\pi \left(r_\pi + \gamma P_\pi v_0\right)$	two policies are 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:	$\pi_2 = \arg\max_\pi \left(r_\pi + \gamma P_\pi v_{\pi_1}\right)$	$\pi_2' = \arg\max_\pi \left(r_\pi + \gamma P_\pi v_1\right)$	
⋮	⋮	⋮	

[1] 实际上可以为任何初始值，为进行比较，此处赋值v_{π_0}.

Table 1: Comparison of two algorithms

Truncated policy iteration

在求解 Bellman equation 时，两种算法实际上也是不同的。例如求解$v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$:

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

Truncated policy iteration 的意思是，在计算 value iteration 时第一步便输出了结果，而 policy iteration 时理论上要计算无穷多次，因此取一个折中的方案，计算j次，就是 truncated policy iteration.

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

[a] 实际上计算policy iteration时也一定不会计算到无穷，也是一个截断的

4.4 Summary

Comparison of Policy iteration and Value iteration

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

Last updated: January 16, 2025

5 Lecture 5, Monte Carlo Learning

Bilibili:Lecture 5, Monte Carlo Learning

Outline

Outline
1. Motivating example
2. The simplest MC-based RL algorithm —- MC basic
3. Use data more efficiently —- MC Exploring Starts
4. MC without exploring starts —- MC ϵ-Greedy

5.1 Motivating example

Motivating example

我们考虑最简单的掷硬币的实验，掷一次的结果记为 X，其是一个随机变量，若为正，则 $X=+1$，反之 $X=-1$，我们的目标是估计 $\mathbb{E}[X]$.

Method 1: **Model-based**

当我们已知概率模型时，例如

$$p(X=1)=0.5, \quad p(X=-1)=0.5$$

那么根据定义就有：

$$\mathbb{E}[X]=\sum_x xp(x)=1\times 0.5+(-1)\times 0.5=0$$

Method 2: **Model-free**

但是有时准确的概率模型获取非常困难，那么就可以进行多次实验以逼近真实情况。例如投掷了 N次硬币，结果记为 $\{x_1,x_2,\dots,x_N\}$，那么根据大数定律(Law of Large Numbers)就有

$$\mathbb{E}[X]\approx \bar{x}=\frac{1}{N}\sum_{j=1}^{N}x_j$$

这就是 Monte Carlo estimation 的基本思想.

5.2 MC basic —- the simplest MC-based RL algorithm

Convert policy iteration to be model-free

MC basic 最本质的 idea 是将 policy iteration 中与模型相关的部分使用 monte-carlo estimation 替换为 model-free 的。回顾 policy iteration 的两个步骤：

$$\begin{aligned}&\text{Policy evaluation: } v_{\pi_k}=r_{\pi_k}+\gamma P_{\pi_k}v_{\pi_k}\\&\text{Policy improvement: } \pi_{k+1}=\arg\max_{\pi}(r_\pi+\gamma P_\pi v_{\pi_k})\\&=\arg\max_{\pi}\sum_a \pi(a|s)q_{\pi_k}(s,a), \quad s\in\mathcal{S}\end{aligned}$$

其中$q_{\pi_k} = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_{\pi_k}(s')$，可以看出 policy improvement 的关键是计算 q_{π_k}。此处分为 model-based 和 model-free 两种方式：

Method 1: model-based，与 Lecture 4 中一样，根据事先给定的概率模型有：

$$q_{\pi_k}(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_{\pi_k}(s') \tag{23}$$

Method 2: model-free(using MC)，根据 action value 最原始的定义式(15)：

$$q_{\pi_k}(s,a) = \mathbb{E}[G_t \mid S_t = s, A_t = a] \tag{24}$$

其中 G_t 为discounted return，我们需要估计它.

如果从 (s,a) 开始，根据 policy π_k 可以产生一次模拟(episode)，此 episode 的 return 记为 $g(s,a)$，那么 $g(s,a)$就是 G_t 的一个样本(sample，在强化学习中称为经验，experience)。当我们产生了很多 samples (记为 $\{g^{(j)}(s,a)\}$)后，根据 Monte-Carlo estimation 就有：

$$q_{\pi_k}(s,a) = \mathbb{E}[G_t \mid S_t = s, A_t = a] \approx \frac{1}{N}\sum_{i=1}^{N} g^{(i)}(s,a) \tag{25}$$

MC basic algorithm

Algorithm 4 MC Basic Algorithm (a model-free variant of policy iteration)

Initialization: Initial guess π_0.
Aim: Search for an optimal policy.
while the value estimate has not converged, for the k-th iteration **do**
  for every state $s \in \mathcal{S}$ **do**
    for every action $a \in \mathcal{A}(s)$ **do**
      Collect sufficiently many episodes starting from (s,a) following π_k.
      MC-based policy evaluation step:

$$q_{\pi_k}(s,a) \leftarrow \text{average return of all the episodes starting from } (s,a)$$

    end for
    Policy improvement step:

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

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

  end for
end while

关于 MC basic algorithm 有几点说明：

1. model-free 算法 MC basic 是建立在 model-based 算法 policy iteration 之上的，唯一区别在于对 q_{π_k} 的计算从基于模型直接计算变成了基于 MC 的估计.
2. MC basic 在理论上具有启发性，是别的算法的基石，但是由于其效率较低，因此几乎没有应用.
3. MC basic 是直接估计 action value q_{π_k} 的，而 policy iteration 是先计算 state value

再根据模型计算 policy iteration，这样做是因为当没有模型时，我们应该直接估计更直接的 action value，否则做两次估计可能误差较大.

4. 虽然使用了 MC 估计，但是算法与 policy iteration 一样，仍然是收敛的.

5.3 MC Exploring Starts —- Use data more efficiently

Use data more efficiently

在一个 episode 中，我们会连续访问许多 state-action pair，例如对于这样的一个 episode:

$$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots$$

我们称每一个 state-action pair (s_i, a_j)为一个 **visit**.

在 MC basic 算法中，使用的策略是 **Initial-visit method**，即用一整个 episode 计算出发点处 state-action pair (s_1, a_2)的 action value $q_\pi(s_1, a_2)$, 但是其缺点是一个 episode 的数据没有被充分利用，因为一个 episode 事实上可以依次拆分成多个 episode:

$$\begin{aligned}
s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots &\text{[original episode, estimate } q_\pi(s_1, a_2)\text{]} \\
s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots &\text{[episode starting from } (s_2, a_4)\text{, estimate } q_\pi(s_2, a_4)\text{]} \\
s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots &\text{[episode starting from } (s_1, a_2)\text{, estimate } q_\pi(s_1, a_2)\text{]} \\
s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots &\text{[episode starting from } (s_2, a_3)\text{, estimate } q_\pi(s_2, a_3)\text{]} \\
s_5 \xrightarrow{a_1} \dots &\text{[episode starting from } (s_5, a_1)\text{, estimate } q_\pi(s_5, a_1)\text{]}
\end{aligned} \tag{26}$$

为能够充分利用一个 episode 的数据，这里有两种方法:

1. first-visit method：例如在例(26)中，出现了两次 (s_1, a_2)，那么只使用第一次出现时计算得到的 $q_\pi(s_1, a_2)$ 作为返回值，第二次的不计算了

2. every-visit method：对于重复出现的 state-action pair，每一次都计算 action value

Update value estimate more efficiently

在 MC-based reinforcement 中另一个重要的问题是什么时候更新 policy，这里仍然有两种方式:

1. average return: 在 MC basic 中，是使用多个 episode 以取平均计算 average return，得到 action value 的估计值，这样需要获得多个 episode 后才能得到，效率较慢但是一次得到的 action value 较准确

2. single episode: 只使用一次 episode 来估计 action value，这样虽然可能不准确，但是更新效率更高。这种思想与 truncated policy iteration 相同

上面介绍的算法都可以归入名为 generalied policy iteration(GPI) 的框架中，这种框架是指算法在 policy-evaluation 和 policy-improvement 之间不断循环迭代，而 policy-evaluation 可以精确也可以使用不精确的方式。许多 model-based 和 model-free 的强化学习算法都可以归入这一框架中.

Algorithm: MC exploring starts

MC exploring starts 算法中 exploring 指的是我们需要从一个 state-action pair 出发，访问到所有的 state-action pair 才能知道某一个 action 是否为最优， starts 是指虽然对于每一个 state-action pair，visit 和 start 都能够访问到，但是由于 visit 有访问不到该 state-action pair 的可能性，因此还是选用 start 来便利所有的 state-action pair。算法如下：

Algorithm 5 MC Exploring Starts (a sample-efficient variant of MC Basic)

Initialization: Initial guess π_0.
Aim: Search for an optimal policy.
for each episode **do**
    Episode generation:
    Randomly select a starting state-action pair (s_0, a_0) and ensure that all pairs can be possibly selected.
    Following the current policy, generate an episode of length T: $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$.
    Policy evaluation and policy improvement:
        Initialization: $g \leftarrow 0$
    for each step of the episode, $t = T-1, T-2, \ldots, 0$ **do** ▷ compute inversely to improve efficiency
        $g \leftarrow \gamma g + r_{t+1}$
        Use the **first-visit** strategy:
        if (s_t, a_t) does not appear in $(s_0, a_0, s_1, a_1, \ldots, s_{t-1}, a_{t-1})$ **then**
            $\text{Returns}(s_t, a_t) \leftarrow \text{Returns}(s_t, a_t) + g$
            $q(s_t, a_t) \leftarrow \text{average}(\text{Returns}(s_t, a_t))$
            $\pi(a \mid s_t) \leftarrow \begin{cases} 1 & \text{if } a = \arg\max_a q(s_t, a), \\ 0 & \text{otherwise.} \end{cases}$
        end if
    end for
end for

5.4 MC ϵ-Greedy —- MC without exploring starts

Soft policies / ϵ-greedy policy

在实际情况中，对于每一个 state-action pair，使用 start 进行计算有可能很难实现(例如真实的机器人在网格世界中每一次 start 都要物理搬运至相应格点)，这时候就需要将 starts 改变成有保障的 visits，就需要使用到 soft policies，其核心就是引入随机性，在更新 policy 时将原本确定性选择最大的 $q_k(s,a)$变为：

$$\pi(a \mid s) = \begin{cases} 1 - \frac{\varepsilon}{|\mathcal{A}(s)|}(|\mathcal{A}(s)| - 1), & \text{if } a = \text{greedy action} \\ \frac{\varepsilon}{|\mathcal{A}(s)|}, & \text{for the other } |\mathcal{A}(s)| - 1 \text{ actions} \end{cases} \tag{27}$$

其中 $\epsilon \in [0, 1]$，$|\mathcal{A}(s)|$ 为 state s 能够对应的所有 action.

Note 7. 可以看到 *soft policy / ϵ-greedy policy* 仍然将最大的概率保留给了 *action value* 最大

的 *greedy action*，这是由于：

$$1-\frac{\varepsilon}{|\mathcal{A}(s)|}(|\mathcal{A}(s)|-1)=1-\varepsilon+\frac{\varepsilon}{|\mathcal{A}(s)|}\geq\frac{\varepsilon}{|\mathcal{A}(s)|} \tag{28}$$

Why use ϵ-greedy? 使用 ϵ-greedy policy 的原因就是平衡 **exploitation**(剥削/选择最优) 和 **exploration**(探索)。Exploitation 是指只选择最优的 action，也就是 greedy，而 exploration 指也指给别的 action 机会，防止陷入局部最优.

1. 当 $\epsilon=0$ 时，就变成了 greedy，此时 less exploration but more exploitation!
2. 当 $\epsilon=1$ 时，所有的 action 选择概率相同，此时 less exploitation but more exploration!
3. 在实际使用中，可以设置较小的 ϵ，然后再逐渐减小.

Consistency: 在 ϵ-Greedy 中，每一个 state 采取的 action 都有一定的概率，但是最大概率对应的 action 与 optimal action 一致，那么就称此 ϵ 下的算法是 consistent 的。一般来说当 ϵ 较小时会是 consistent.

MC ϵ-Greedy algorithm

将 ϵ-Greedy 的策略替换掉原先的 greedy 的策略，就可以直接得到 MC ϵ-Greedy algorithm，如下：

Algorithm 6 MC ϵ-Greedy (a variant of MC Exploring Starts)

Initialization: Initial guess π_0 and the value of $\epsilon\in[0,1]$.
Aim: Search for an optimal policy.
for each episode **do**
  Episode generation:
  Randomly select a starting state-action pair (s_0,a_0). Following the current policy, generate an episode of length T: $s_0,a_0,r_1,\ldots,s_{T-1},a_{T-1},r_T$.
  Policy evaluation and policy improvement:
    Initialization: $g\leftarrow 0$
  for each step of the episode, $t=T-1,T-2,\ldots,0$ **do** ▷ for efficiency
    $g\leftarrow\gamma g+r_{t+1}$
    Use the **every-visit** method: ▷ maybe some visits are long enough
    if (s_t,a_t) does not appear in $(s_0,a_0,s_1,a_1,\ldots,s_{t-1},a_{t-1})$ **then**
      $\text{Returns}(s_t,a_t)\leftarrow\text{Returns}(s_t,a_t)+g$
      $q(s_t,a_t)\leftarrow\text{average}(\text{Returns}(s_t,a_t))$
      Let $a^*=\arg\max_a q(s_t,a)$ and
      $$\pi(a\mid s_t)\leftarrow\begin{cases}\frac{1-\epsilon}{|\mathcal{A}(s_t)|}+\epsilon & \text{if } a=a^*,\\ \frac{\epsilon}{|\mathcal{A}(s_t)|} & \text{if } a\neq a^*.\end{cases}$$
    end if
  end for
end for

Last updated: January 20, 2025

6 Lecture 6, Stochastic Approximation and Stochastic Gradient Decent

Bilibili:Lecture 6, Stochastic Approximation and Stochastic Gradient Decent

6.1 Motivating example: mean estimation

Example

在实际估计均值/期望时，我们常常通过采样求平均的方式进行计算，例如采样了 N 个样本 $\{x_i\}_{i=1}^N$:

$$\mathbb{E}[X] \approx \bar{x} := \frac{1}{N}\sum_{i=1}^{N} x_i \tag{29}$$

根据大数定律，我们可以知道 $\bar{x} \to \mathbb{E}[X] \quad \text{as} \quad N \to \infty$.

在强化学习中，很多值例如 state/action value 都是由均值定义的，因此均值估计(mean estimation)很重要，但式(29)是需要等到 N 个样本采样完成后才能得到的，这会造成算法效率的降低。一种替代方式是对 $\bar{x}$ 进行增量式地迭代更新，更新后立即可以使用而无需等待，这样虽然损失了部分精度，但是提升了算法效率.

事实上，规定

$$w_{k+1} = \frac{1}{k}\sum_{i=1}^{k} x_i, \quad k = 1, 2, \ldots \tag{30a}$$

$$w_k = \frac{1}{k-1}\sum_{i=1}^{k-1} x_i, \quad k = 2, 3, \ldots \tag{30b}$$

那么就有

$$w_{k+1} = \frac{1}{k}\sum_{i=1}^{k} x_i = \frac{1}{k}\left(\sum_{i=1}^{k-1} x_i + x_k\right) = \frac{1}{k}\left((k-1)w_k + x_k\right) = w_k - \frac{1}{k}(w_k - x_k) \tag{31}$$

更一般地，

$$w_{k+1} = w_k - \alpha_k(w_k - x_k) \tag{32}$$

此时 $\alpha_k > 0$，当其满足一些条件时，仍然会有 $x_k \to \mathbb{E}[X] \quad \text{as} \quad k \to \infty$.

6.2 Robbins-Monro algorithm(RM Alg.)

Stochastic approximation

随机近似(Stochastic approximation, SA) 是一类算法，用于求方程的求根问题或优化问题。相较于一些例如牛顿法求根、梯度下降进行优化等算法，SA 可以在不知道函数表达式（但可以获得有噪/无噪输出）的情况下使用迭代更新的方式进行求解。其中，RM 算法是 SA 的非常具有开创性的工作，著名的随机梯度下降算法和 mean estimation 都是 RM 算法的特殊情况.

Robbins-Monro algorithm

Problem statement: 考虑一个方程的求根问题：

$$g(w) = 0$$

实际上优化问题也是这样的形式：

$$g(w) = \nabla_w J(w) = 0$$

当 $g(w)$ 的表达式已知时，有很多算法可以求解这个问题；但是，当 $g(w)$ 的表达式未知时，例如神经网络就可以抽象为 $g(w) = y$ 的形式，但是 g 的形式是未知的.

Algorithm: RM 算法通过如下迭代方式逼近真实解：

$$w_{k+1} = w_k - a_k \tilde{g}(w_k, \eta_k), \quad k = 1, 2, 3, \ldots \tag{33}$$

其中 w_k是对解的第 k 次估计，$\tilde{g}(w_k, \eta_k) = g(w_k) + \eta_k$ 是第 k 个有噪观测，η_k 为噪声，$g(w)$ 是一个黑盒(black box)，其表达式未知，因此此算法只能依赖于数据：

1. Input sequence: $\{w_k\}$
2. Noisy output sequence: $\{\tilde{g}(w_k, \eta_k)\}$

Robbins-Monro algorithm – Convergence properties

Theorem 6 (Robbins-Monro theorem)**.** 对于 *Robbins-Monro* 算法，若

1. $\forall w, 0 < c_1 \leq \nabla_w g(w) \leq c_2$（梯度的要求）
2. $\sum_{k=1}^{\infty} a_k = \infty, \sum_{k=1}^{\infty} a_k^2 < \infty$（系数的要求）
3. $\mathbb{E}[\eta_k | \mathcal{H}_k] = 0, \mathbb{E}[\eta_k^2 | \mathcal{H}_k] < \infty$（测量误差的要求）

其中 $\mathcal{H}_k = \{w_k, w_{k-1}, \ldots\}$。设方程 $g(w) = 0$ 的根为 w^*，满足 $g(w^*) = 0$，那么 w_k 以概率*1* 收敛*(with probability 1, w.p.1)*至 w^*.

Note 8. 对于上述三个条件，有如下详细解读：

1. 条件 *1* – 梯度的要求
   - *(a)* 要求梯度 $\nabla_w g(w) > 0$，保证原函数单调上升，根据零点存在定理，方程 $g(w) = 0$ 一定有唯一解
   - *(b)* 要求梯度 $\nabla_w g(w) > 0$ 这一条件一般是可以接受的，因为当处理的是一个优化问题时，$g(w)$ 本身就是一个梯度，再求梯度并要求其大于 *0* 后就表示原函数是凸的，这是一个较常见的限制条件
2. 条件 *2* – 系数的要求
   - *(a)* $\sum_{k=1}^{\infty} a_k^2 < \infty$ 表明$a_k \xrightarrow{k \to \infty} 0$，这是由于 $w_{k+1} - w_k = -a_k \tilde{g}(w_k, \eta_k)$，因此为保证收敛需要 $a_k \xrightarrow{k \to \infty} 0$
   - *(b)* $\sum_{k=1}^{\infty} a_k = \infty$ 表明$a_k \xrightarrow{k \to \infty} 0$ 没有那么快，这是由于根据递推式 $w_2 = w_1 - a_1 \tilde{g}(w_1, \eta_1), w_3 = w_2 - a_2 \tilde{g}(w_2, \eta_2), \ldots, w_{k+1} = w_k - a_k \tilde{g}(w_k, \eta_k)$，将两端累和消去可以得到

$$w_1 - w_\infty = \sum_{k=1}^{\infty} a_k \tilde{g}(w_k, \eta_k)$$

因此根据 $\sum_{k=1}^{\infty} a_k < \infty$，$tildeg$ 有界，那么$\sum_{k=1}^{\infty} a_k \tilde{g}(w_k, \eta_k)$ 有界，这样如果 $a_k \xrightarrow{k\to\infty} 0$ 过快会要求初始猜测 w_1 不能够离 w^* 过远，反之若收敛至零较慢则可以增大初始猜测的范围，要求更宽，使得算法的可行性更好

(c) 取 $a_k = \frac{1}{k}$是可以满足要求的，因为有

$$\lim_{n\to\infty}\left(\sum_{k=1}^{n}\frac{1}{k} - \ln n\right) = \kappa, \quad \sum_{k=1}^{\infty}\frac{1}{k^2} = \frac{\pi^2}{6} < \infty(\textit{Basel problem})$$

其中 κ 为 *Euler-Mascheroni* 常数[a].

(d) 常见的做法并不是直接另系数为 $\frac{1}{k}$，因为这样当 k 较大时，会导致后面的数据在算法中起到的作用被无限缩小但是这并不是我们想要的，因此常见的做法是开始时赋予较小的值再慢慢减小但是也不会变得非常小以至于趋于 *0*

3. 条件 *3* – 测量误差的要求

(a) 常见的做法是取 $\{\eta_k\}$ 来自一个 *i.i.d.* 的随机序列，满足 $\mathbb{E}[\eta_k] = 0, \mathbb{E}[\eta_k^2] < \infty$

(b) 这里并不要求 η 满足高斯性

[a]欧拉-马斯刻若尼常数是一个数学常数，定义为调和级数与自然对数的差值

Robbins-Monro algorithm – Apply to mean estimation

定义 $g(w) \triangleq w - \mathbb{E}[X]$，我们的目的解出 $g(w) = 0$，我们可以得到的有噪观察是 $\tilde{g}(w, x) \triangleq w - x$，其中 x 是对 X 的采样，那么可以得出

$$\tilde{g}(w, \eta) = (w - \mathbb{E}[X]) + (\mathbb{E}[X] - x) \triangleq g(w) + \eta$$

这样代入 RM 算法(33)中就得到:

$$w_{k+1} = w_k - \alpha_k \tilde{g}(w_k, \eta_k) = w_k - \alpha_k(w_k - x_k)$$

再套用 Theorem6 即可得到相应的收敛性结论.

Dvoretzkys convergence theorem (optional)

Theorem 7 (Dvoretzkys convergence theorem)**.** 考虑如下随机过程:

$$w_{k+1} = (1 - \alpha_k)w_k + \beta_k\eta_k$$

其中 $\{\alpha_k\}_{k=1}^{\infty}, \{\beta_k\}_{k=1}^{\infty}, \{\eta_k\}_{k=1}^{\infty}$ 均为随机序列*(stochastic sequences)*，$\forall k, \alpha_k \geqslant 0, \beta_k \geqslant 0$，那么若如下条件满足则 $w_k \xrightarrow{w.p.1} 0$:

1. $\sum_{k=1}^{\infty}\alpha_k = \infty, \sum_{k=1}^{\infty}\alpha_k^2 < \infty; \sum_{k=1}^{\infty}\beta_k^2 < \infty$ *uniformly w.p.1*
2. $\mathbb{E}[\eta_k|\mathcal{H}_k] = 0, \mathbb{E}[\eta_k^2|\mathcal{H}_k] \leq C$ *w.p.1*

其中 $\mathcal{H}_k = \{w_k, w_{k-1}, \ldots, \eta_{k-1}, \ldots, \alpha_{k-1}, \ldots, \beta_{k-1}, \ldots\}$

Note 9. 关于 *Dvoretzkys convergence theorem* 有如下几点说明:

1. *Dvoretzkys convergence theorem* 是*RM theorem* 更一般化的结论，可以用于证明 *RM theorem*
2. *Dvoretzkys convergence theorem* 可以用于直接分析 *mean estimation* 问题
3. *Dvoretzkys convergence theorem* 的推广版本可用于分析后续的 *Q-learning* 和 *TD learning* 算法

6.3 Stochastic gradient decent

Stochastic gradient decent – algorithm

1. Stochastic gradient decent 是 RM 算法的特殊情况
2. Mean estimation 是 Stochastic gradient decent 的特殊情况

Problem statement: 我们需要解决如下优化问题：

$$\min_w \quad J(w) = \mathbb{E}[f(w, X)] \tag{34}$$

其中

1. w 为优化的目标参数/变量
2. X 为随机变量，取均值就是对 X 取的
3. w, X 可为标量也可以为向量，$f(\cdot)$ 为标量
4. 在实际算法中，取期望一般用采样累和代替

Gradient Decent(GD) 梯度下降法：

$$w_{k+1} = w_k - \alpha_k \nabla_w \mathbb{E}[f(w_k, X)] = w_k - \alpha_k \mathbb{E}[\nabla_w f(w_k, X)] \tag{35}$$

其中 α_k 为步长，控制下降的快慢。此算法的缺点是期望是很难求取的。

Batch Gradient Decent(BGD) 由于 GD 中梯度的期望很难准确求得，因此就需要用采样的方式代替（蒙特卡洛的思想）：

$$\mathbb{E}[\nabla_w f(w_k, X)] \approx \frac{1}{n} \sum_{i=1}^{n} \nabla_w f(w_k, x_i) \tag{36a}$$

$$w_{k+1} = w_k - \alpha_k \frac{1}{n} \sum_{i=1}^{n} \nabla_w f(w_k, x_i) \tag{36b}$$

此算法的缺陷是在每一次迭代中需要采样很多样本，因此计算也很困难。

Stochastic Gradient Decent(SGD) 随机梯度下降仅使用一个随机的样本替代期望的计算，即将 BGD 中的 $n = 1$：

$$w_{k+1} = w_k - \alpha_k \nabla_w f(w_k, x_k) \tag{37}$$

虽然 SGD 会较不精确，但是可以提升算法的效率.

Stochastic gradient decent – Examples

Example: 考虑在深度学习中常见的均方损失函数：

$$\min_{w} \quad J(w) = \mathbb{E}[f(w, X)] = \mathbb{E}\left[\frac{1}{2}\|w - X\|^2\right]$$

其中 $f(w, X) = \|w - X\|^2/2, \nabla_w f(w, X) = w - X.$

那么有如下三个练习（附答案）：

1. $w^* = \mathbb{E}[X]$
2. 解决此问题的 GD 算法为：
3. 解决此问题的 SGD 算法为：

Stochastic gradient decent – Convergence

对比 GD 与 SGD 可以发现，变化就是将对梯度取期望换成采样一次作为期望的预估值：

$$\begin{aligned} w_{k+1} &= w_k - \alpha_k \mathbb{E}[\nabla_w f(w_k, X)] \\ &\Downarrow \\ w_{k+1} &= w_k - \alpha_k \nabla_w f(w_k, x_k) \end{aligned}$$

那么 $\nabla_w f(w_k, x_k)$ 就可以看作 $\mathbb{E}[\nabla_w f(w, X)]$ 的有噪估计：

$$\nabla_w f(w_k, x_k) = \mathbb{E}[\nabla_w f(w, X)] + \underbrace{\nabla_w f(w_k, x_k) - \mathbb{E}[\nabla_w f(w, X)]}_{\eta}$$

由于

$$\nabla_w f(w_k, x_k) \neq \mathbb{E}[\nabla_w f(w, X)]$$

因此我们需要考虑是否 $w_k \to w^*$ as $k \to \infty$，证明的思路是证明 SGD 是一个特殊的 RM 算法（实际上直接证明也可以，但是证明会很复杂）.

首先优化问题(34)可以转化为如下求根问题：

$$J(w) = \mathbb{E}[f(w, X)] \Rightarrow g(w) \triangleq \nabla_w J(w) = \mathbb{E}[\nabla_w f(w, X)] = 0$$

令

$$\begin{aligned} \tilde{g}(w, \eta) &= \nabla_w f(w, x) \\ &= \underbrace{\mathbb{E}[\nabla_w f(w, X)]}_{g(w)} + \underbrace{\nabla_w f(w, x) - \mathbb{E}[\nabla_w f(w, X)]}_{\eta} \end{aligned}$$

那么解决 $g(w) = 0$ 的 RM 算法为：

$$w_{k+1} = w_k - a_k \tilde{g}(w_k, \eta_k) = w_k - a_k \nabla_w f(w_k, x_k)$$

这就是 SGD，这样根据 Theorem 6就可以得到如下结论.

Theorem 8 (Convergence of SGD). 对于 *SGD* 算法，定义满足 $\nabla_w \mathbb{E}[f(w, X)] = 0$ 的根为 w^*。若满足以下条件则 $w_k \xrightarrow{w.p.1} w^*$：

1. $0 < c_1 \leq \nabla_w^2 f(w, X) \leq c_2$（严格凸性）

2. $\sum_{k=1}^{\infty} a_k = \infty, \quad \sum_{k=1}^{\infty} a_k^2 < \infty$

3. $\{x_k\}_{k=1}^{\infty}$ *i.i.d.*

虽然根据定理8知道了最终满足一定条件的 SGD 会收敛，但是由于引入了随机性，因此有必要分析一下其收敛的性质，探究其是否具有随机性或者收敛很慢。为此需要引入如下相对误差(relative error)：

$$\delta_k \triangleq \frac{|\nabla_w f(w_k, x_k) - \mathbb{E}[\nabla_w f(w_k, X)]|}{|\mathbb{E}[\nabla_w f(w_k, X)]|}$$

由于 $\mathbb{E}[\nabla_w f(w^*, X)] = 0$，因此根据中值定理(mean value theorem)可以引入二阶导数和$(w_k - w^*)$项：

$$\delta_k = \frac{|\nabla_w f(w_k, x_k) - \mathbb{E}[\nabla_w f(w_k, X)]|}{|\mathbb{E}[\nabla_w f(w_k, X)] - \mathbb{E}[\nabla_w f(w^*, X)]|} = \frac{|\nabla_w f(w_k, x_k) - \mathbb{E}[\nabla_w f(w_k, X)]|}{|\mathbb{E}[\nabla_w^2 f(\tilde{w}_k, X)(w_k - w^*)]|}$$

其中 $\tilde{w}_k \in]w_k, w^*]$.

由于一般都会对 f 做严格凸的假设，因此不妨先做如下假设：

$$\nabla_w^2 f \geq c > 0, \quad \forall w, X$$

其中 c 有界。那么就有

$$\begin{aligned} &\left|\mathbb{E}[\nabla_w^2 f(\tilde{w}_k, X)(w_k - w^*)]\right| = \left|\mathbb{E}[\nabla_w^2 f(\tilde{w}_k, X)](w_k - w^*)\right| \\ &= \left|\mathbb{E}[\nabla_w^2 f(\tilde{w}_k, X)]\right| |(w_k - w^*)| \geq c|w_k - w^*| \end{aligned}$$

这样就得到：

$$\delta_k \leq \frac{\overbrace{\nabla_w f(w_k, x_k)}^{\text{stochastic gradient}} - \overbrace{\mathbb{E}[\nabla_w f(w_k, X)]}^{\text{true gradient}}}{\underbrace{c|w_k - w^*|}_{\text{distance to the optimal solution}}}$$

因此

1. 当 $w_k - w^*$ 较大，即离最优解较远时，SGD 会表现得更像 GD，梯度下降更新的更准确
2. 当 $w_k - w^*$ 较小，即离最优解较近时，确实会存在一定的随机性，但是此时已经接近了最优解

Stochastic gradient decent – Experiment

实验 here

Stochastic gradient decent – A deterministic formulation

可以看到梯度下降算法的问题设置中包含了求期望，但是在深度学习常常涉及到的是

最小话训练样本的误差，即：

$$\min_{w} \quad J(w) = \frac{1}{n}\sum_{i=1}^{n} f(w, x_i)$$

其中 $f(w, x_i)$ 是参数化的函数，w 为优化的目标变量，$\{x_i\}_{i=1}^n$ 为采样得到的随机变量.

此时求解此问题的 GD 算法和 SGD 算法分别为：

$$GD: w_{k+1} = w_k - \alpha_k \nabla_w J(w_k) = w_k - \alpha_k \frac{1}{n}\sum_{i=1}^{n} \nabla_w f(w_k, x_i) \tag{38a}$$

$$SGD: w_{k+1} = w_k - \alpha_k \nabla_w f(w_k, x_k) \tag{38b}$$

如果我们将 x_i 的选取看作是从 $p_k = \frac{1}{n}$ 的均匀分布中取样得到的，那么显然算法(38b)仍然是一种 SGD 算法；并且此时由于随机性，那么 x_k 的选取是随机的，并不应该排序且可能反复取到.

6.4 BGF, MBGD, and SGD

Example

回忆三种算法如下：

$$\begin{aligned} w_{k+1} &= w_k - \alpha_k \frac{1}{n}\sum_{i=1}^{n} \nabla_w f(w_k, x_i), \quad \text{(BGD)} \\ w_{k+1} &= w_k - \alpha_k \frac{1}{m}\sum_{j\in\mathcal{I}_k} \nabla_w f(w_k, x_j), \quad \text{(MBGD)} \\ w_{k+1} &= w_k - \alpha_k \nabla_w f(w_k, x_k). \quad \text{(SGD)} \end{aligned} \tag{39}$$

1. 可以认为 MBGD 囊括了 BGD 和 SGD：当 batch $|\mathcal{I}_k| \to n$ 时，MBGD $\to$ BGD，当 batch $|\mathcal{I}_k| \to 1$ 时，MBGD $\to$ SGD
2. 直观上来说随机性 GD $>$ MBGD $>$ BGD，效率 BGD $>$ MBGD $>$ GD
3. 注意 MBGD 中 $\mathcal{I}_k$ 是一个采样集合，有可能采样到重复的，因此即使是 $|\mathcal{I}_k| = n$ BGD 与 MBGD 也有细微差别，因为 BGD 是 $\{1, \cdots, n\}$ 每个数字采样一次，但是 MBGD 是可能重复的.

Last updated: January 24, 2025

7 Lecture 7, Temporal-Difference Learning

Bilibili:Lecture 7, Temporal-Difference Learning

Outline

Value & Policy iteration(model-based) → Monte Carlo Learning(model free)[a]

Monte Carlo Learning(non-incremental) → Temporal-Difference Learning(incremental)

TD Learning of state values → TD learning of action values → TD learning of optimal action values

[a]Monte Carlo Learning 是本课程所学习的第一个 model-free 的方法

7.1 Motivating examples

Motivating examples

Example 1: mean estimation(review)　对于 mean estimation 问题

$$w = \mathbb{E}[X]$$

令 $g(w) = w - \mathbb{E}[X]$，转化为求解 $g(w) = 0$，同时为使用 RM 算法，得到采样的有噪观测为：

$$\tilde{g}(w, \eta) = w - x = (w - \mathbb{E}[X]) + (\mathbb{E}[X] - x) \doteq g(w) + \eta$$

再使用 RM 算法即可求解 mean estimation 问题：

$$w_{k+1} = w_k - \alpha_k \tilde{g}(w_k, \eta_k) = w_k - \alpha_k (w_k - x_k)$$

Example 2: mean estimation of a function $v(X)$ (a little bit more complex)

此时我们求解 $w = \mathbb{E}[v(X)]$，其中 $v(\cdot)$ 为函数。类似地，可以得到：

$$\begin{aligned} &g(w) = w - \mathbb{E}[v(X)] \\ &\tilde{g}(w, \eta) = w - v(x) = (w - \mathbb{E}[v(X)]) + (\mathbb{E}[v(X)] - v(x)) \doteq g(w) + \eta \\ &w_{k+1} = w_k - \alpha_k \tilde{g}(w_k, \eta_k) = w_k - \alpha_k [w_k - v(x_k)] \end{aligned}$$

Example 3: mean estimation of a two random variables (complex situation)

此时我们求解 $w = \mathbb{E}[R + \gamma v(X)]$，其中 R, X 均为随机变量，γ 为常数，$v(\cdot)$ 为函数，记 $\{x\}, \{r\}$ 分别为 X, R 的采样，那么类似地可以得到：

$$\begin{aligned} &g(w) = w - \mathbb{E}[R + \gamma v(X)], \\ &\tilde{g}(w, \eta) = w - [r + \gamma v(x)] = (w - \mathbb{E}[R + \gamma v(X)]) + (\mathbb{E}[R + \gamma v(X)] - [r + \gamma v(x)]) \\ &\doteq g(w) + \eta \\ &w_{k+1} = w_k - \alpha_k \tilde{g}(w_k, \eta_k) = w_k - \alpha_k [w_k - (r_k + \gamma v(x_k))] \end{aligned}$$

7.2 TD Learning

7.2.1 TD Learning of state values

TD Learning of state values

在强化学习中，TD 算法一般指的是一大类算法，包括著名的 Q-learning，但是这一部分首先讲解最基础和简单的 TD learning 算法，并直接称之为 TD learning，其用于估计给定 policy 的 state value，用于 policy evaluation.

首先 TD learning 算法依赖于数据而非模型，将其需要的数据/experience 记为

$$(s_0, r_1, s_1, \ldots, s_t, r_{t+1}, s_{t+1}, \ldots) \text{ or } \{(s_t, r_{t+1}, s_{t+1})\}_t$$

其是由给定的 policy π 生成的 state-reward 序列.

首先直接给出估计 π 的 state value 的 TD learning 算法:

$$v_{t+1}(s_t) = v_t(s_t) - \alpha_t(s_t)\left[v_t(s_t) - [r_{t+1} + \gamma v_t(s_{t+1})]\right] \tag{40a}$$

$$v_{t+1}(s) = v_t(s), \quad \forall s \neq s_t \text{(未被访问的状态不更新，常省略此式)} \tag{40b}$$

为更细致分析，将涉及到更新过程的式(40a)更详细地写为:

$$\underbrace{v_{t+1}(s_t)}_{\text{new estimate}} = \underbrace{v_t(s_t)}_{\text{current estimate}} - \alpha_t(s_t)\left[\overbrace{v_t(s_t) - \underbrace{[r_{t+1} + \gamma v_t(s_{t+1})]}_{\text{TD target } \bar{v}_t}}^{\text{TD error } \delta_t}\right] \tag{41}$$

其中，定义 TD target 为

$$\bar{v}_t \doteq r_{t+1} + \gamma v(s_{t+1}) \tag{42}$$

定义 TD error 为

$$\delta_t \doteq v(s_t) - [r_{t+1} + \gamma v(s_{t+1})] = v(s_t) - \bar{v}_t \tag{43}$$

由此看出，新的估计 $v_{t+1}(s_t)$ 是当前估计 $v_t(s_t)$ 和 TD error (43) 的组合.

首先详细分析 TD target (42) 的性质：将 $\bar{v}_t$ 称为 target 是因为我们迭代的目标就是让 $v(s_t)$ 不断靠近 $\bar{v}_t$.

根据式(41)，可以得到

$$v_{t+1}(s_t) = v_t(s_t) - \alpha_t(s_t)\left[v_t(s_t) - \bar{v}_t\right] \Longrightarrow v_{t+1}(s_t) - \bar{v}_t = v_t(s_t) - \bar{v}_t - \alpha_t(s_t)\left[v_t(s_t) - \bar{v}_t\right]$$
$$\Longrightarrow v_{t+1}(s_t) - \bar{v}_t = [1 - \alpha_t(s_t)][v_t(s_t) - \bar{v}_t] \Longrightarrow |v_{t+1}(s_t) - \bar{v}_t| = |1 - \alpha_t(s_t)||v_t(s_t) - \bar{v}_t|$$

再由 $\alpha_t(s_t)$ 为一个较小的正数，得到:

$$\alpha_t(s_t) > 0 \Rightarrow 0 < 1 - \alpha_t(s_t) < 1 \Rightarrow |v_{t+1}(s_t) - \bar{v}_t| \leq |v_t(s_t) - \bar{v}_t|$$

因此 $v_t(s_t) \to \bar{v}_t$.

接着分析 TD error 的性质:

$$\delta_t \doteq v_t(s_t) - [r_{t+1} + \gamma v_t(s_{t+1})] = v_t(s_t) - \bar{v}_t$$

可以看到算法的迭代过程和此定义都涉及到 t 与 $t+1$ 两个时间步，因此将此算法称为 "temporal difference"，即时序差分.由于算法迭代的目标是 $v_t \to v_\pi$，因此不妨将其均进行替换得到:

$$\delta_{\pi,t} \doteq v_\pi(s_t) - [r_{t+1} + \gamma v_\pi(s_{t+1})]$$

根据$v_\pi(s_t)$的定义：$v_\pi(s_t) = \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s_t\right]$，注意到

$$\mathbb{E}[\delta_{\pi,t}|S_t = s_t] = v_\pi(s_t) - \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s_t\right] = 0$$

因此如果 $v_t = v_\pi$，那么 $\delta_t = 0$，否则 $v_t \neq v_\pi$.

TD Learning of state values: The idea of the algorithm

What does this TD learning alg. do mathematically? – 求解给定策略 π 的 Bellman equation，与前面 closed-form 和 iterative solution 不同的是 TD learning 算法是 model-free 的算法，在没有模型的情况下求解 Bellman (expectation) equation.

Bellman expectation equation – 首先介绍一个新的 Bellman equation，其与原先介绍的 Bellman equation (9) 等价，只不过使用了期望的数学形式，因此看起来更简洁。

根据 state value 的定义

$$v_\pi(s) = \mathbb{E}\left[R + \gamma G \mid S = s\right], \quad s \in S$$

其中 G 为 discounted return。由于

$$\mathbb{E}[G|S = s] = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) v_\pi(s') = \mathbb{E}[v_\pi(S')|S = s]$$

其中 S' 为下一 state。这样式(9) 就可以重写为

$$v_\pi(s) = \mathbb{E}\left[R + \gamma v_\pi(S') \mid S = s\right], \quad s \in \mathcal{S} \tag{44}$$

RM algorithm for solving the Bellman expectation equation (44)

首先定义求解的目标

$$g(v(s)) = v(s) - \mathbb{E}\left[R + \gamma v_\pi(S') \mid s\right] \to g(v(s)) = 0$$

进而根据 R 和 S' 的采样 r, s' 得到

$$\tilde{g}(v(s)) = \underbrace{\left(v(s) - \mathbb{E}\left[R + \gamma v_\pi(S') \mid s\right]\right)}_{g(v(s))} + \underbrace{\left(\mathbb{E}\left[R + \gamma v_\pi(S') \mid s\right] - \left[r + \gamma v_\pi(s')\right]\right)}_{\eta}.$$

因此求解此问题的 RM 算法为：

$$\begin{aligned} v_{k+1}(s) &= v_k(s) - \alpha_k \tilde{g}(v_k(s)) \\ &= v_k(s) - \alpha_k \left(v_k(s) - \left[r_k + \gamma v_\pi(s'_k)\right]\right), \quad k = 1, 2, 3, \ldots \end{aligned} \tag{45}$$

其中 $v_k(s)$ 为 $v_\pi(s)$ 在第 k 步的估计，r_k, s'_k 分别为 R, S' 在第 k 步的采样.

需要注意的两点是

1. 算法(45)中仅是在对一个 state s 进行估计，在 TD learning 中我们需要将固定的 $\{(s, r, s')\}$ 变为序列 $\{s_t, t_t, s_{t+1}\}$，而没有更新到的 state 就不变（如式40b）
2. 算法(45)中使用了我们估计的目标 v_π 进行迭代，在 TD learning 中我们需要使用 $v_k(s'_k)$ 替换 $v_\pi(s'_k)$.

TD Learning of state values: Algorithm convergence

Theorem 9 (Convergence of Sarsa learning)**.** 使用 *TD algorithm* 时，若满足如下条件则 $\forall s \in \mathcal{S}$， $v_t(s) \xrightarrow[t\to\infty]{w.p.1.} v_\pi(s)$*:*

1. $\sum_t \alpha_t(s) = \infty$（每一个状态 s 都会被访问很多/无穷次，因为未被访问时$\alpha_t(s) = 0$）
2. $\sum_t \alpha_t^2(s) < \infty, \forall s \in \mathcal{S}$（意味着$\alpha_t \to 0$，实际中设为随着 t 最后变得很小的数）

TD Learning of state values: comparison

Comparison between TD learning and MC learning[1]	
TD/Sarsa learning	**MC learning**
Online: TD learning is online. It can update the state/action values immediately after receiving a reward.	**Offline:** MC learning is offline. It has to wait until an episode has been completely collected.[2]
Continuing tasks: Since TD learning is online, it can handle both episodic and continuing tasks.	**Episodic tasks:** Since MC learning is offline, it can only handle episodic tasks that have terminate states.
Bootstrapping: TD bootstraps because the update of a value relies on the previous estimate of this value. Hence, it **requires initial guesses**.	**Non-bootstrapping:** MC is not bootstrapping, because it can directly estimate state/action values without any initial guess.
Low estimation variance: TD has lower variance than MC because there are fewer random variables. For instance, Sarsa requires $R_{t+1}, S_{t+1}, A_{t+1}$.	**High estimation variance:** To estimate $q_\pi(s_t, a_t)$, we need samples of $R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$. Suppose the length of each episode is L. There are $\lvert A\rvert^L$ possible episodes.
Biased to unbiased Since TD uses less data at first, it's expectation is biased at first and gradually become unbiased with more data been used.	**Unbiased** Though MC has high variance, it uses more data and is unbiased.

[1] 由于 TD learning 与Sarsa 基本一样，因此放在一起比较.
[2] MC learning 需要使用采样完成后的数据一起估计均值.

Table 2: Comparison between TD learning and MC learning

7.2.2 TD Learning of action values

TD Learning of action values: Sarsa

首先，我们的目的是估计给定 policy π 的 action value，由于没有模型，因此我们需要有数据/ experience $\{(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})\}_t$。直接给出 Sarsa 算法为

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\left[q_t(s_t, a_t) - [r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})]\right] \tag{46a}$$

$$q_{t+1}(s, a) = q_t(s, a), \quad \forall (s, a) \neq (s_t, a_t), t = 0, 1, 2, \cdots \tag{46b}$$

其中 $q_t(s_t, a_t)$ 是对 $q_\pi(s_t, a_t)$ 的估计，α_t 为 learning rate.

Why called Sarsa? – 源自 experience state-action-reward-state-action 的缩写.

The relationship between Sarsa and TD learning – Sarsa 是 TD learning 的 action-value version，只需要将 TD learning 中的 $v(s)$ 替换为 $q(s, a)$ 即可.

What does the Sarsa do mathematically? – 类似于 TD learning 在求解一个 Bellman equation，Sarsa 将关于 action value 的 Bellman equation (16b)简化为

$$q_\pi(s, a) = \mathbb{E}\left[R + \gamma q_\pi(S', A')|s, a\right], \quad \forall s, a$$

Convergence –

Theorem 10 (Convergence of Sarsa learning)**.** 使用 *Srasa algorithm* 时，若满足如下条件则 $\forall s \in \mathcal{S}, a \in \mathcal{A}$， $q_t(s, a) \xrightarrow[t\to\infty]{w.p.1.} q_\pi(s, a)$*:*

1. $\sum_t \alpha_t(s, a) = \infty$（每对 (s, a) 都会被访问很多/无穷次，因为未被访问时$\alpha_t(s, a) = 0$）
2. $\sum_t \alpha_t^2(s, a) < \infty, \forall s \in \mathcal{S}, a \in \mathcal{A}$（意味着$\alpha_t \to 0$，实际中设为随着 t 最后变得很小的数）

TD Learning of action values: Sarsa - Algorithm

Sarsa 算法只能够做到给定一个 policy π，通过迭代得到相应的 action value，仍然需要结合 ploicy improvement 才能得到 optimal policy。实际上二者结合后也被成为 Sarsa，且常说的 Sarsa 就是指结合后的算法.

Algorithm 7 Policy Searching by Sarsa

for each episode **do**
  if the current s_t is not the target state **then**
    Collect the experience $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$: In particular, take action a_t following $\pi_t(s_t)$, generate r_{t+1}, s_{t+1}, and then take action a_{t+1} following $\pi_t(s_{t+1})$.
    Update q-value(policy evaluation):

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\left[q_t(s_t, a_t) - (r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1}))\right]$$

    Update policy(policy improvement):

$$\pi_{t+1}(a \mid s_t) = \begin{cases} 1 - \frac{\epsilon}{|A|-1} & \text{if } a = \arg\max_a q_{t+1}(s_t, a) \\ \frac{\epsilon}{|A|} & \text{otherwise} \end{cases}$$

  end if
end for

Note 10. *1.* 注意在做 *policy evalution* 时只做了一次就进行了 *policy improvement* 并继续迭代，并没有准确进行估计，因此是基于 *gneralized policy iteration* 的想法.

TD Learning of action values: Expected Sarsa

Expected Sarsa 算法是在原先的 Sarsa 基础上变形得到的，将 $r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$ 变

为 $r_{t+1}+\gamma\mathbb{E}[q_t(s_{t+1},A)]$，因此 Expected Sarsa 算法如下：

$$q_{t+1}(s_t,a_t)=q_t(s_t,a_t)-\alpha_t(s_t,a_t)\left[q_t(s_t,a_t)-(r_{t+1}+\gamma\mathbb{E}[q_t(s_{t+1},A)])\right] \tag{47a}$$

$$q_{t+1}(s,a)=q_t(s,a),\quad \forall(s,a)\neq(s_t,a_t) \tag{47b}$$

其中

$$\mathbb{E}[q_t(s_{t+1},A)])=\sum_a \pi_t(a|s_{t+1})q_t(s_{t+1},a)\doteq v_t(s_{t+1})$$

Note 11. 可以看到 *expected Sarsa* 不需要对 a_{t+1} 进行采样，取而代之需要更大的计算量去估计 $\mathbb{E}[q_t(s_{t+1},A)])$，因此相较于 *Sarsa* 计算量增大，但是由于使用的随机变量的个数变少了，因此减小了随机性.

What does the expected Sarsa do mathematically? – 实际上 expected Sarsa 在求解如下方程：

$$q_\pi(s,a)=\mathbb{E}\left[R_{t+1}+\gamma\mathbb{E}_{A_{t+1\sim\pi(S_{t+1})}}[q_\pi(S_{t+1},A_{t+1})]|S_t=s,A_t=a\right],\quad \forall s,a.$$

此方程是 Bellman equation 的另一种展开式：

$$q_\pi(s,a)=\mathbb{E}\left[R_{t+1}+\gamma v_\pi(S_{t+1})|S_t=s,A_t=a\right]$$

TD Learning of action values: n-step Sarsa

n-step Sarsa 是Sarsa 的一个变形，也是一个推广，其包含了 Sarsa 和 Monte Carlo 两种方法。

首先，根据定义，action value 写为

$$q_\pi(s,a)=\mathbb{E}[G_t|S_t=s,A_t=a]$$

其中 G_t 为 discounted return，其可以被写为多种不同的形式：

$$\begin{aligned}
\text{Sarsa}\longleftarrow\quad & G_t^{(1)}=R_{t+1}+\gamma q_\pi(S_{t+1},A_{t+1}),\\
& G_t^{(2)}=R_{t+1}+\gamma R_{t+2}+\gamma^2 q_\pi(S_{t+2},A_{t+2}),\\
& \cdots\\
n\text{-step Sarsa}\longleftarrow\quad & G_t^{(n)}=R_{t+1}+\gamma R_{t+2}+\cdots+\gamma^n q_\pi(S_{t+n},A_{t+n}),\\
\text{MC}\longleftarrow\quad & G_t^{(\infty)}=R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\ldots
\end{aligned}$$

Note 12. 注意此处所有含有上标的 $G_t^{(1)},\cdots,G_t^{(\infty)}$ 都是等价的，不同的仅是如何分解.

得到 G_t 不同的分解后，将其代入 action value 定义式中，得到

$$\begin{aligned}
\text{Sarsa to solve:}\quad & q_\pi(s,a)=\mathbb{E}[G_t^{(1)}|s,a]=\mathbb{E}[R_{t+1}+\gamma q_\pi(S_{t+1},A_{t+1})|s,a]\\
n\text{-step Sarsa:}\quad & q_\pi(s,a)=\mathbb{E}[G_t^{(n)}|s,a]=\mathbb{E}[R_{t+1}+\gamma R_{t+2}+\cdots+\gamma^n q_\pi(S_{t+n},A_{t+n})|s,a]\\
\text{MC learning:}\quad & q_\pi(s,a)=\mathbb{E}[G_t^{(\infty)}|s,a]=\mathbb{E}[R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\ldots|s,a]
\end{aligned}$$

相应地得到不同的算法形式：

$$\begin{aligned}
\text{Sarsa:}\quad & q_{t+1}(s_t,a_t) = q_t(s_t,a_t) - \alpha_t(s_t,a_t)\left[q_t(s_t,a_t) - [r_{t+1} + \gamma q_t(s_{t+1},a_{t+1})]\right] \\
n\text{-step Sarsa:}\quad & q_{t+1}(s_t,a_t) = q_t(s_t,a_t) - \alpha_t(s_t,a_t)\left[q_t(s_t,a_t) - [r_{t+1} + \cdots + \gamma^n q_t(s_{t+n},a_{t+n})]\right] \\
\text{MC learning:}\quad & q_{t+1}(s_t,a_t) = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n t_{t+n} + \cdots
\end{aligned}$$

当 n-step Sarsa 中的 $n = 1$ 时，就变为了 Sarsa，当 $\alpha_t = 1, n = \infty$ 时，就变为了 MC learning，因此说 n-step Sarsa 包含了 Sarsa 和 TD learning。当 n 较大时，n-step Sarsa 的性质类似于 TD learning，有比较大的 variance，但较小的 bias，反之 当n 较小时，其性质类似于 Sarsa，有较小的 variance 但较大的 bias.

下面比较这三种算法的不同之处：

① **TD target:** Sarsa 与 n-step Sarsa 的 TD target是不同的：

$$\begin{aligned}
\text{Sarsa:}\quad & \bar{v}_t^{(1)} \doteq r_{t+1} + \gamma v(s_{t+1}) \\
n\text{-step Sarsa:}\quad & \bar{v}_t^{(2)} \doteq r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n q_t(s_{t+n},a_{t+n})
\end{aligned}$$

② **Data needed:** 由于三种算法都是 model free 的算法，因此都需要数据，但是他们所需要的数据不同：

$$\begin{aligned}
\text{Sarsa:}\quad & (s_t,a_t,r_{t+1},s_{t+1},a_{t+1}) \\
n\text{-step Sarsa:}\quad & (s_t,a_t,r_{t+1},s_{t+1},a_{t+1},\cdots,r_{t+n},s_{t+n},a_{t+n}) \\
\text{TD learning:}\quad & (s_t,a_t,r_{t+1},s_{t+1},a_{t+1},\cdots,r_{t+n},s_{t+n},a_{t+n},\cdots)
\end{aligned}$$

③ **Online vs. Offline:** 我们已经知道 Sarsa 是使用当前时间节点的数据立即进行算法迭代，为 online 的算法，而 MC learning 需要等待到足够多的数据才可以估算 action value，为 offline 的算法，而 n-step Sarsa 则既不是 online 也不是 offline 的算法，或将其看作是特殊的 online 算法，因为其需要等到 $t+n$ 时刻才可以更新 (s_t,a_t) 的 action value：

$$\begin{aligned}
& q_{t+n}(s_t,a_t) = q_{t+n-1}(s_t,a_t) \\
& - \alpha_{t+n-1}(s_t,a_t)\left[q_{t+n-1}(s_t,a_t) - [r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n q_{t+n-1}(s_{t+n},a_{t+n})]\right]
\end{aligned}$$

7.2.3 TD Learning of optimal action values

TD Learning of optimal action values: Q-learning

前面介绍的 Sarsa 及其变形都是在估计 action value，将其与 policy evaluation 结合后迭代就可以进行 policy searching(如Algorithm7所示)。下面介绍非常经典且常用[a]的算法 Q-learning，其直接估计 optimal action value，而无需在 policy evaluation 和 policy improvement 间交替运行.

首先直接给出 Q-learning 算法：

$$q_{t+1}(s_t,a_t) = q_t(s_t,a_t) - \alpha_t(s_t,a_t)\left[q_t(s_t,a_t) - [r_{t+1} + \gamma \max_{a\in\mathcal{A}} q_t(s_{t+1},a)]\right] \tag{48a}$$

$$q_{t+1}(s,a) = q_t(s,a), \quad \forall (s,a) \neq (s_t,a_t) \tag{48b}$$

可以看到 Q-learning 的形式与 Sarsa 基本一样，唯一区别在于 TD target:

$$\text{Q-learning:} \quad r_{t+1} + \gamma \max_{a \in \mathcal{A}} q_t(s_{t+1}, a)$$

$$\text{Sarsa:} \quad r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$$

What does the Q-learning do mathematically? – 与 Sarsa 在求解一个 action value 的 Bellman equation 不同，Q-learning 是在求解一个 Bellman optimality equation(BOE)，且与(17)不同的是，这里求解的是一个 action value 形式的、带有期望的 BOE（proof in textbook）.

$$q(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_a q(S_{t+1}, a) | S_t = s, A_t = a\right], \quad \forall s, a.$$

[a]现在常用的算法是 deep Q-learning，为 Q-learning 的变形

Off vs. on-policy

首先介绍 TD learning 中存在的两种 policy：

Definition 3 (Two policies in TD learning). *TD learning* 中存在两种策略，分别为 ***behavior policy*** 和 ***target policy***。其中 *behavior policy* 用于生成 *experience samples*，是智能体与环境的交互策略，包含探索成分； *target policy* 用于不断迭代至最优策略。

基于上述两种 policy，可以将不同的强化学习算法分为 on-policy 和 off-policy 两类：

Definition 4 (on-policy and off-policy). 当一种学习方法的 *behavior policy* 与 *target policy* 一致时，称之为 ***on-policy***，即智能体在学习过程中是通过执行自己的*target policy*来收集数据和更新策略；当一种学习方法的 *behavior policy* 与 *target policy* 不一致时，称之为 ***off-policy***，即*behavior policy*与*target policy*不一致，智能体可以通过*behavior policy* 收集数据，而通过*target policy* 来进行学习（此时二者也可以相同）.

Advantages of off-policy learning: off-policy 的好处是其可以使用别的（例如探索性更强的）policy 生成的 experience samples 去搜索 optimal policy，这样也可以加速学习的过程.

How to judge On or off-policy? 1. 搞清楚算法在数学上做了什么事情（例如求解BE/BOE?）；2. 检查算法运行需要什么东西.

Sarsa is on-policy：首先，Sarsa 的目标是求解给定 policy π 的 Bellman equation:

$$q_\pi(s,a) = \mathbb{E}\left[R + \gamma q_\pi(S', A') | s, a\right], \quad \forall s, a.$$

其中 $R \sim p(R|s,a), S' \sim p(S'|s,a), A' \sim \pi(A'|S')$.

其次，在算法上，Sarsa 的算法为

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\left[q_t(s_t, a_t) - [r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})]\right]$$

其需要的数据为 $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$，其中

1. (s_t, a_t) 是给定的，(r_{t+1}, s_{t+1}) 依赖于 $p(R|s,a), p(S'|s,a)$而不依赖任何 policy[a]
2. a_{t+1} 是和 policy $\pi_t(s_{t+1})$ 有关的，因此 π_t 就是 behavior policy

而我们已经知道 Sarsa 的目的就是将 π_t 的 action value 不断迭代至 π 的 action value，因此 π_t 既是 behavior policy 也是 target policy.

$$\pi_t \to \text{experience} \to q_{\pi_t} \to \pi_{t+1} \to \cdots$$

MC learning is on-policy: 首先，MC learning 的目标是求 action value 的期望：

$$q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \ldots | S_t = s, A_t = a\right], \quad \forall s, a.$$

其次，在算法是使用多个采样来近似估算：

$$q(s,a) \approx r_{t+1} + \gamma r_{t+2} + \ldots$$

其过程为：对于当前的策略 π，生成相应的 experience / trajectory，再使用该 experience / trajectory 的 return 得到 action value，再改进 π，即如下流程：

$$\pi \to \text{experience / trajectory} \to q_\pi \to \pi \to \cdots$$

因此这里的 π 既是 behavior policy 也是 target policy.

Q-learning is off-policy: 首先 Q-learning 数学上在求解一个 BOE：

$$q(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_a q(S_{t+1}, a) | S_t = s, A_t = a\right], \quad \forall s, a.$$

其次，在算法上

$$q_{t+1}(s_t,a_t) = q_t(s_t,a_t) - \alpha_t(s_t,a_t)\left[q_t(s_t,a_t) - [r_{t+1} + \gamma \max_{a\in\mathcal{A}} q_t(s_{t+1},a)]\right]$$

因此需要的数据为 $(s_t, a_t, r_{t+1}, s_{t+1})$，其中若 (s_t, a_t) 给定，那么 r_{t+1} 和 s_{t+1} 分别依赖于 $\mathbb{P}(r|s,a), \mathbb{P}(s'|s,a)$，而不依赖于 policy.

Q-learning 的 behavior policy 用于从 state s_t 产生 antion a_t，其可以是任何 policy，而 target policy 则为 optimal policy，其根据 $q \to q^*$ 来判断相应策略收敛至 optimal policy。因此 Q-learning 是 off-policy 的.

[a] 由于不知道这两个概率，因此需要通过采样

Q-learning – Implementation

由于 Q-learning 是 off-policy 的，其 behavior policy 与 target policy 可以不同也可以相同，分别能够得到如下两种版本的算法：

Algorithm 8 Policy Searching by Q-learning (on-policy version)

for each episode **do**
  if the current s_t is not the target state **then**
    Collect the experience $(s_t, a_t, r_{t+1}, s_{t+1})$: In particular, take action a_t following $\pi_t(s_t)$, generate r_{t+1}, s_{t+1}.
    Update q-value:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\left[q_t(s_t, a_t) - \left(r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)\right)\right]$$

    Update policy:

$$\pi_{t+1}(a \mid s_t) = \begin{cases} 1 - \frac{\epsilon}{|A|-1} & \text{if } a = \arg\max_a q_{t+1}(s_t, a), \\ \frac{\epsilon}{|A|} & \text{otherwise.} \end{cases}$$

  end if
end for

Algorithm 9 Optimal Policy Search by Q-learning (off-policy version)

for each episode $\{s_0, a_0, r_1, s_1, a_1, r_2, \dots\}$ generated by π_b **do** ▷ π_b is behavior policy
  for each step $t = 0, 1, 2, \dots$ of the episode **do**
    Update q-value:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\left[q_t(s_t, a_t) - \left(r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)\right)\right]$$

    Update target policy:

$$\pi_{T,t+1}(a \mid s_t) = \begin{cases} 1 & \text{if } a = \arg\max_a q_{t+1}(s_t, a), \\ 0 & \text{otherwise.} \end{cases}$$

▷ π_T is target policy

  end for
end for

Note 13. 在 *off-policy version* 中直接使用 *greedy* 策略进行 *target policy* 的更新是因为此时我们的 *target policy* 并不需要去生成数据（这是 *behavior policy* 做的），因此不需要具备探索性，只需要每次直接选择最好的即可；相反在 *on-policy version* 中，由于 *target policy* 也需要再生成数据，因此需要保持一定的探索性，故使用 $\epsilon-$*greedy* 的更新模式.

A unified point of view

本节中所介绍的所有算法实际上都可以表达为一个统一的形式：

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - \bar{q}_t]$$

其中 $\bar{q}_t$ 为 TD target。所有的 TD 算法的不同点就在于 TD target 和数学上求解的 BE / BOE：

Different TD target in different TD algorithms	
Algorithm	**Expression of $\bar{q}_t$**
Sarsa	$\bar{q}_t = r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$
n-step Sarsa	$\bar{q}_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n q_t(s_{t+n}, a_{t+n})$
Expected Sarsa	$\bar{q}_t = r_{t+1} + \gamma \sum_a \pi_t(a \mid s_{t+1}) q_t(s_{t+1}, a)$
Q-learning	$\bar{q}_t = r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)$
Monte Carlo[1]	$\bar{q}_t = r_{t+1} + \gamma r_{t+2} + \dots$

[1] 只需要将 $\alpha_t(s_t, a_t) = 1$，就可以得到 $q_{t+1}(s_t, a_t) = \bar{q}_t$

Table 3: Different TD target in different TD algorithms

Different equation aim to solve in different TD algorithms	
Algorithm	**Equation aim to solve**
Sarsa	BE: $q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \middle| S_t = s, A_t = a\right]$
n-step Sarsa	BE: $q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n q_\pi(s_{t+n}, a_{t+n}) \middle| S_t = s, A_t = a]$
Expected Sarsa	BE: $q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \mathbb{E}_{A_{t+1}}[q_\pi(S_{t+1}, A_{t+1})] \middle| S_t = s, A_t = a\right]$
Q-learning	BOE: $q(s,a) = \mathbb{E}\left[R_{t+1} + \max_a q(S_{t+1}, a) \middle| S_t = s, A_t = a\right]$
Monte Carlo	BE: $q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \dots \middle| S_t = s, A_t = a]$

Table 4: Different TD target in different TD algorithms

Last updated: February 12, 2025

8 Lecture 8, Value Function Approximation

Bilibili:Lecture 8, Value Function Approximation

8.1 Motivating examples: curve fitting

Table

在本节以前，state value 与 action value 都是以表格(table)的形式给出，例如 action value:

	a_1	a_2	a_3	a_4	a_5
s_1	$q_\pi(s_1, a_1)$	$q_\pi(s_1, a_2)$	$q_\pi(s_1, a_3)$	$q_\pi(s_1, a_4)$	$q_\pi(s_1, a_5)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_9	$q_\pi(s_9, a_1)$	$q_\pi(s_9, a_2)$	$q_\pi(s_9, a_3)$	$q_\pi(s_9, a_4)$	$q_\pi(s_9, a_5)$

Table 5: Table of action values

s_1	s_2	s_3	s_4	$\cdots$	s_9
$v_\pi(s_1)$	$v_\pi(s_2)$	$v_\pi(s_3)$	$v_\pi(s_4)$	$\cdots$	$v_\pi(s_9)$

Table 6: Table of state values

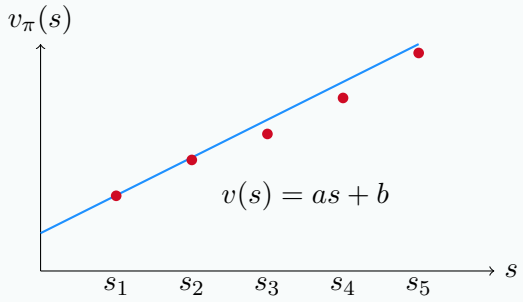
使用表格形式处理数据的优点在于其易于理解和分析，但是其缺点在于难以处理大型或者连续的 state / action space，原因有三点：

1. **Storage 1:** 当空间较大时，需要大量存储空间
2. **Storage 2:** 当空间是连续时需要进行离散化（使用网格），当网格稀疏时不能很好地近似函数空间，当网格稠密时又面临存储问题
3. **Generalization ability:** 当有很多数据时，往往不能将这些数据全部访问完，这样就造成泛化能力弱

Curve fitting

函数近似的思想实际上很简单，就是使用简单的函数近似 action / state value 与其自变量 (s, a) / s 间的函数关系，例如考虑如下一个简单的例子：

- 考虑一维的 state $s_1, \ldots, s_{|\mathcal{S}|}$，其中 $\mathcal{S}$ 为 state space，$|\mathcal{S}|$ 为所有的状态个数
- 上述 state 对应的 state value 为 $v_\pi(s_1), \ldots, v_\pi(s_{|\mathcal{S}|})$，其中 π 为给定的 policy
- 假设 $|\mathcal{S}|$ 非常大，我们希望使用一个简单的曲线拟合这些 state - state value 对，即 $(s, v_\pi(s))$（如下图所示），这样我们就只需要存储该曲线较少的参数.



最简单的方式就是使用直线去拟合这些点，其可以写为如下形式：

$$\hat{v}(s,w) = as + b = \underbrace{[s,1]}_{\phi^T(s)} \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{w} = \phi^T(s)w \tag{49}$$

其中 w 为 parameter vector（参数向量），$\phi(s)$ 为 s 的 feature vector（特征向量）.

使用直线（仅针对本例，实际可以更复杂一些）去拟合的好处在于：

1. 原本我们需要存储 $|\mathcal{S}|$ 个 state value，但是现在只需要存储两个参数 a 和 b
2. 每当我们需要使用到 s 的 state value 时，我们只需要计算 $\phi^T(s)w$ 即可
3. **However.** 这样做也是有代价的，由于这只是一个拟合/近似（approximation），因此会引入误差.

也可以使用更复杂的形式，例如使用二阶的曲线进行拟合：

$$\hat{v}(s,w) = as^2 + bs + c = \underbrace{[s^2,s,1]}_{\phi^T(s)} \underbrace{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}_{w} = \phi^T(s)w. \tag{50}$$

这样做的目的是通过增加参数量以减小拟合的误差.

Note 14. 值得注意的是，此时的函数 $\hat{v}(s,w)$ 对于数据 s 是非线性的，但是对于参数 w 来说仍然是线性的.

Summary

1. **Idea:** 使用参数化的函数去近似 state / action value: $\hat{v}(s,w) \approx v_\pi(s)/\hat{p}(s,a,w) \approx p_\pi(s,a,w)$，其中 $w \in \mathbb{R}^m$ 为 parameter vector
2. **Advantage**
   (a) **Storage:** 由于 w 的维数较 $|\mathcal{S}|$ 少很多，因此具有存储上的巨大优势
   (b) **Generalization:** 在使用 table 形式存储数据时，当某个 state / action value 发生改变时其他的值并不会进行更新，因此如果不能将全部的数据访问完，则会造成大量 state 的 state value 未更新；但是若使用函数拟合，改变了一点的 state / action value 就会改变整体的 parameter vector，这样全部的 state / action value 都会被更新.

Selection of function approximators

在进行拟合时，一个关键问题是选择什么样的拟合函数 $\hat{v}(s,w)$，这主要有两种方式：

1. **Linear function:** 以前广泛使用的是使用线性函数

$$\hat{v}(s,w) = \phi^T(s)w$$

进行拟合，这里的 $\phi(s)$ 为 feature vector，可以选取为多项式基底、Fourier 基底等

2. **Neural networks:** 当今被广泛使用的是用神经网络作为非线性函数逼近器，即神经网络的参数为 w，输入神经网络的为 state，输出为非线性的 $\hat{v}(s,w)$.

Linear function approximator

选用线性函数逼近有如下优缺点：

1. **Disadvantages:** 选取合适的特征向量(feature vectors)需要经验，且即使如此也很困难

2. **Advantages:**

   (a) 线性情形下的理论分析较非线性情形时更简单，且可以帮助理解非线性情形，目前已经有了很多的理论结果

   (b) 虽然说线性函数的拟合能力不如非线性，但是已经比较好了，并且之前的 tabular representation 实际上是线性函数逼近的一个特殊形式，因此其可以将 tarbular 与 value function approximation 统一.

当我们将 feature vector 选取为单位向量时，即

$$\phi(s) = e_s \in \mathbb{R}^{|S|}$$

其中 e_s 只有第 s 个元素为1，其余为0.
那么此时就有

$$\hat{v}(s,w) = e_s^T w = w(s)$$

其中 $w(s)$ 为 w 的第 s 个元素.
此时将 $\phi(s_t) = e_s$ 带入 TD-linear 的算法10中的

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma\phi^T(s_{t+1})w_t - \phi^T(s_t)w_t\right]\phi(s_t)$$

中得到

$$w_{t+1} = w_t + \alpha_t \left(r_{t+1} + \gamma w_t(s_{t+1}) - w_t(s_t)\right)e_{s_t}$$

由于 e_{s_t} 只有第 s_t 非零，因此得到

$$w_{t+1}(s_t) = w_t(s_t) + \alpha_t \left(r_{t+1} + \gamma w_t(s_{t+1}) - w_t(s_t)\right)$$

将 w 替换为 v 就与 tarbular TD algprithm 形式一模一样.

8.2 Algorithm for state value estimation

8.2.1 Objective function

Objective function

现在，在 state evaluation 任务中我们的目标是找到最优的参数向量 w，使其能够对于所有的 s 都能最好地近似 $v_\pi(s)$，为此我们需要两个步骤：

1. 定义目标函数(objective function)
2. 使用合适的算法去优化该目标函数

我们直接给出目标函数的定义如下：

$$J(w) = \mathbb{E}[(v_\pi(S) - \hat{v}(S, w))^2] \tag{51}$$

我们的目标是最小化 $J(w)$ 以找到 optimal w。注意这里的 S 是一个随机变量，需要取期望，而方式也有不同，主要有如下两种方式：

1. **uniform distribution:** 即取到每个 state 的概率是一样的，都是 $1/|\mathcal{S}|$，此时目标函数可写为：

$$J(w) = \mathbb{E}[(v_\pi(S) - \hat{v}(S, w))^2] = \frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}}(v_\pi(s) - \hat{v}(s, w))^2$$

   虽然平均分布是直观的，但是假设了所有的状态是平等的，但是实际上接近目标的状态应该是更重要的，需要分配更多的权重.

2. **stationary / steady-state / limiting distribution:** 描述了一个 **long-run behavior**，即不断采取同一个策略与环境交互，达到一个平稳状态时每个 state 被访问次数的概率分布，将其记为 $\{d_\pi(s)\}_{s\in\mathcal{S}}$，其中 $d_\pi(s) \geq 0, \sum_{s\in\mathcal{S}} d_\pi(s) = 1$。这样得到的目标函数为

$$J(w) = \mathbb{E}[(v_\pi(S) - \hat{v}(S, w))^2] = \sum_{s\in\mathcal{S}} d_\pi(s)(v_\pi(s) - \hat{v}(s, w))^2.$$

Objective function – Stationary distribution

Illustrative example: 给定一个 policy，我们不断采用该 policy 与环境交互以研究其 long-run behavior，记 $n_\pi(s)$ 为 state s 被访问到的次数，那么 $d_\pi(s)$ 就可以被近似为

$$d_\pi(s) \approx \frac{n_\pi(s)}{\sum_{s'\in\mathcal{S}} n_\pi(s')}$$

实际上不需要进行仿真实验也可以在理论上得到 $d_\pi(s)$ 的理论值。由于 $d_\pi(s)$ 是在平稳状态时的分布，所以实际上其满足

$$d_\pi^T = d_\pi^T P_\pi$$

其中 P_π 为状态转移矩阵，那么显然在理论上 $d_\pi(s)$ 为 P_π 的特征向量.

实际上 Stationary distribution 还有更加丰富的内涵，详见 textbook

8.2.2 Optimization algorithms

Gradient decent

为最小化目标函数 $J(w)$，我们可以使用梯度下降算法：

$$w_{k+1} = w_k - \alpha_k \nabla_{*w} J(w_k)$$

其中梯度为

$$\begin{aligned}\nabla_w J(w) &= \nabla_w \mathbb{E}[(v_\pi(S) - \hat{v}(S,w))^2] = \mathbb{E}[\nabla_w (v_\pi(S) - \hat{v}(S,w))^2] \\ &= -2\mathbb{E}[(v_\pi(S) - \hat{v}(S,w))\nabla_w \hat{v}(S,w)]\end{aligned}$$

由于期望 $\mathbb{E}$ 无法计算，就可以使用随机梯度代替真实的梯度：

$$w_{t+1} = w_t + \alpha_t (v_\pi(s_t) - \hat{v}(s_t,w_t))\nabla_w \hat{v}(s_t,w_t),$$

其中 s_t 为 S 的一个采样，这里可以将系数 2 融入 α_t 从而更简洁，但是不影响优化进程.

但是可以看到在进行梯度下降时，已经使用了我们需要求解的目标 v_π，因此我们需要将其替换，常见的有两种方法：

1. **Monte Carlo:** 使用从 s_t 出发的一个 episode 的 discounted return g_t 代替 $v_\pi(s_t)$：

$$w_{t+1} = w_t + \alpha_t (g_t - \hat{v}(s_t,w_t))\nabla_w \hat{v}(s_t,w_t)$$

2. **TD learning:** 使用 $r_{t+1} + \gamma \hat{v}(s_{t+1},w_t)$ 代替 $v_\pi(s_t)$：

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \hat{v}(s_{t+1},w_t) - \hat{v}(s_t,w_t)\right] \nabla_w \hat{v}(s_t,w_t)$$

Algorithm 10 TD Learning with Function Approximation

Initialization: A function $\hat{v}(s,w)$ that is differentiable in w. Initial parameter w_0.
Aim: Approximate the true state values of a given policy π.
for each episode generated following the policy π **do**
  for each step (s_t, r_{t+1}, s_{t+1}) **do**
    In the general case:

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \hat{v}(s_{t+1},w_t) - \hat{v}(s_t,w_t)\right] \nabla_w \hat{v}(s_t,w_t)$$

    In the linear case(TD linear):

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \phi^T(s_{t+1}) w_t - \phi^T(s_t) w_t\right] \phi(s_t)$$

  end for
end for

8.2.3 Theoretical analysis

Theoretical analysis

Summary of the story:

objective function → gradient decent → replace the true value function by approximation

事实上我们之前所说的 objective function 并不是真正理论上的目标函数，目标函数也有多种：

1. **Objective function 1: True value error**

$$J_E(w) = \mathbb{E}[(v_\pi(S) - \hat{v}(S,w))^2] = \|\hat{v}(w) - v_\pi\|_D^2 = (\hat{v}(w) - v_\pi)^T D(\hat{v}(w) - v_\pi)$$

其中 D 为 S 对应的分布.

2. **Objective function 2: Bellman error** 由于实际上我们在求解一个 Bellman equation，所以目的使得 $\hat{v}(w) = r_\pi + \gamma P_\pi \hat{v}(w)$

$$J_{BE}(w) = \|\hat{v}(w) - (r_\pi + \gamma P_\pi \hat{v}(w))\|_D^2 \doteq \|\hat{v}(w) - T_\pi(\hat{v}(w))\|_D^2$$

其中 $T_\pi(x) \doteq r_\pi + \gamma P_\pi x$.

3. **Objective function 3: Projected Bellman error** 由于使用的函数可能出于函数结构的原因永远无法达到 $\hat{v}(w) = r_\pi + \gamma P_\pi \hat{v}(w)$，因此作出一个投影 M 使得投影到 $\hat{v}$ 的空间上可以二者相等

$$J_{PBE}(w) = \|\hat{v}(w) - MT_\pi(\hat{v}(w))\|_D^2$$

其中 M 为投影矩阵(projection matrix)

8.3 Sarsa with function approximation

Sarsa with function approximation

Sarsa 与之前的算法唯一的不同就在于是在使用 $\hat{q}(s,a,w)$ 估计 action value $q(s,a)$ 而不是使用 $\hat{v}(s,w)$ 来估计 state value $v(s)$，因此非常自然地可以得到使用了 function approximation 的 Sarsa 算法为:

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t)\right] \nabla_w \hat{q}(s_t, a_t, w_t) \tag{52}$$

式(52)实际上仍然在做 policy evaluation，将其与 policy improvement 结合就得到如下完整算法:

Algorithm 11 Sarsa with Function Approximation

1: **Aim:** Search a policy that can lead the agent to the target from an initial state-action pair (s_0, a_0).
2: **for** each episode **do**
3: **if** the current s_t is not the target state **then**
4: Take action a_t following $\pi_t(s_t)$, generate r_{t+1}, s_{t+1}, and then take action a_{t+1} following $\pi_t(s_{t+1})$
5: **Value update (parameter update):**

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t)\right] \nabla_w \hat{q}(s_t, a_t, w_t)$$

6: **Policy update:**

$$\pi_{t+1}(a \mid s_t) = \begin{cases} 1 - \frac{\epsilon}{|A(s)|-1} & \text{if } a = \arg\max_a \hat{q}(s_t, a, w_{t+1}) \\ \frac{\epsilon}{|A(s)|} & \text{otherwise.} \end{cases}$$

7: **end if**
8: **end for**

8.4 Deep Q-learning / Deep Q-network

Q-learning with function approximation]]

Q-learning 只需要将 Sarsa 中的 $\hat{q}(s_{t+1}, a_{t+1}, w_t)$ 替换为 $\max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t)$，就得到

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \right] \nabla_w \hat{q}(s_t, a_t, w_t) \tag{53}$$

Algorithm 12 Q-learning with Function Approximation (on-policy version)

1: **Initialization:** Initial parameter vector w_0. Initial policy π_0. Small $\epsilon > 0$.
2: **Aim:** Search a good policy that can lead the agent to the target from an initial state-action pair (s_0, a_0).
3: **for** each episode **do**
4: **if** the current s_t is not the target state **then**
5: Take action a_t following $\pi_t(s_t)$, and generate r_{t+1}, s_{t+1}.
6: **Value update (parameter update):**

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \max_{a \in A(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \right] \nabla_w \hat{q}(s_t, a_t, w_t)$$

7: **Policy update:** ▷ Since this is on-policy, so exploration needed

$$\pi_{t+1}(a \mid s_t) = \begin{cases} 1 - \frac{\epsilon}{|A(s_t)|-1} & \text{if } a = \arg\max_{a' \in A(s_t)} \hat{q}(s_t, a', w_{t+1}) \\ \frac{\epsilon}{|A(s_t)|} & \text{otherwise.} \end{cases}$$

8: **end if**
9: **end for**

Deep Q-learning – basic idea

Deep Q-learning 也被称为Deep Q-network(DQN)，它是最早也是最成功的将深度学习引入强化学习的算法[a]。深度学习在 DQN 中的作用是非线性函数逼近器(nonlinear function approximator)，但是有别于式(53)的是，由于神经网络强大的表达能力，我们不需要再这么底层地来计算 $\hat{q}$.

Objective / Loss function: DQN 目的是最小化如下目标函数/损失函数：

$$J(w) = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w) \right)^2 \right] \tag{54}$$

其中 (S, A, R, S') 为随机变量.

实际上这里就是在不断逼近 TD target，此 loss function 为 Bellman optimality error，因为根据 BOE 定义：

$$q(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a \in \mathcal{A}(S_{t+1})} q(S_{t+1}, a) | S_t = s, A_t = a \right], \quad \forall s, a$$

因此在期望意义下 $R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w)$ 应当为0.

Optimize: 最小化损失函数最直接的想法就是使用梯度下降。在损失函数式(54)中参数 w 不仅存在于 $\hat{q}(S,A,w)$ 中，还存在于不好计算梯度的 $\gamma \max_{a\in\mathcal{A}(S')} \hat{q}(S',a,w)$ 中，此处定义

$$y \triangleq R + \gamma \max_{a\in\mathcal{A}(S')} \hat{q}(S',a,w)$$

将其整体看作 w 是（暂时）固定的，这样就不用计算其梯度.

DQN technique 1: Two networks 在 DQN 中引入了两个网络：

1. **main network:** 第一个网络目的与之前相同，都是为了逼近 $\hat{q}(s,a,w)$
2. **target network:** 第二个网络用于逼近 $\hat{q}(s,a,w_T)$，每隔一段时间就将 w 赋值给 w_T，并持续一段时间内将 $\hat{q}(s,a,w_T)$ 固定、无需进行梯度下降.

这样损失函数就变为了

$$J = \mathbb{E}\left[\left(R + \gamma \max_{a\in\mathcal{A}(S')} \hat{q}(S',a,w_T) - \hat{q}(S,A,w)\right)^2\right] \tag{55}$$

这样整个算法和训练的流程就是

$$\text{将 } w \text{ 赋值给 } w_T \to \text{固定 } w_T \text{ 只更新 } \hat{q}(S,A,w) \text{ 中的 } w \to \text{将 } w \text{ 赋值给 } w_T$$

DQN technique 2: Experience replay 第二个技巧是使用 experience replay 的方式训练网络：虽然我们收集到数据是有先后顺序的，但是实际使用时，我们需要将其打散，全部归入一个集合 $\mathcal{B} \doteq \{(s,a,r,s')\}$，称之为 replay buffer，然后使用均匀分布在此集合中采样，就称这些采样为 experience replay，因为有可能被重复取到。这样做的原因在于：根据我们对 loss function 的定义：

$$J = \mathbb{E}\left[\left(R + \gamma \max_{a\in\mathcal{A}(S')} \hat{q}(S',a,w) - \hat{q}(S,A,w)\right)^2\right]$$

这其中有四个随机变量 (R,S',S,A)，其中根据 system model 有 $R \sim p(R|S,A), S' \sim p(S'|S,A)$，而我们可以将 (S,A) 看作是一个随机变量[b]，由于赋予高斯分布等需要先验知识（以判断哪些数据是重要的），如若没有则需要一视同仁，即使用均匀分布。由于采集数据时数据是有先后顺序的，因此为实现均匀分布，需要将这些数据打散，破除他们之间的 correlation，这也是使用 replay buffer 的原因.

[a] 虽然在 DQN 之前也有一些将深度学习结合强化学习的尝试，但是 DQN 是效果最好的，能够在一些游戏人物上达到人类玩家水准

[b] 就像平面坐标下两个坐标表示一个点一样

Questions about DQN

Q: 为什么 tarbular Q-learning 不需要 experience replay?
A: 因为在表格形式中根本没有使用到分布的需要，是在不断求解所有 (s,a)的 BOE，而在 DQN 中，损失函数定义使用了期望，因此需要分布.

Q: 可以在表格形式的 Q-learning 中使用 experience replay 吗?
A: 可以！并且这样可以反复使用一次采样的经验，能够让数据(sample)使用地更高效.

DQN Algorithm

Algorithm 13 Deep Q-learning (off-policy version)

Aim: Learn an optimal target network to approximate the optimal action values from the experience samples generated by a behavior policy π_b.
Store the experience samples generated by π_b in a replay buffer $\mathcal{B} = \{(s, a, r, s')\}$.
for each iteration **do**
    Uniformly draw a mini-batch of samples from $\mathcal{B}$.
    for each sample (s, a, r, s') **do**
        Calculate the target value as ▷ w_T is the parameter of the target network

$$y_T = r + \gamma \max_{a' \in A(s')} \hat{q}(s', a', w_T),$$

    end for
    Update the **main network** to minimize

$$(y_T - \hat{q}(s, a, w))^2$$

    using the mini-batch $\{(s, a, y_T)\}$.
    Set $w_T = w$ every C iterations.
end for

Note 15. *1.* 由于算法*13*是 *off-policy* 的，因此不需要进行 *policy update*

2. 为利用神经网络高效的"黑盒"性，因此不再使用之前基于梯度的 *policy update* 方式

3. DQN 原文中的算法是 *on-policy* 的，与此处略有不同

Last updated: February 14, 2025

9 Lecture 9, Policy Gradient Methods

Bilibili:Lecture 9, Policy Gradient Methods

Outline

本节将 policy-based 算法的学习，直接建立一个 policy 的目标函数，通过优化此函数就可以直接得到最优的策略；此前所学习的算法都是 value-based，核心是进行 evaluation 后，根据此 action / state value 不断迭代选择更好的 policy.

value-based methods $\rightarrow$ policy-based methods
value function approximation $\rightarrow$ policy function approximation

Basic idea of policy gradient

在此前，policy 都是以表格形式存储的，即不同 state 下采取某 action 的概率都存储在如下表格中：

	a_1	a_2	a_3	a_4	a_5
s_1	$\pi(s_1,a_1)$	$\pi(s_1,a_2)$	$\pi(s_1,a_3)$	$\pi(s_1,a_4)$	$\pi(s_1,a_5)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_9	$\pi(s_9,a_1)$	$\pi(s_9,a_2)$	$\pi(s_9,a_3)$	$\pi(s_9,a_4)$	$\pi(s_9,a_5)$

Table 7: Table of probability of all states

现在我们参考 policy function approximation 的想法，也将 $\pi(s,a)$ 函数化，将其记为 $\pi(a|s;\theta)$，其中 $\theta \in \mathbb{R}^d$ 为参数向量，也可以记其为 $\pi_\theta(a,s)$ 和 $\pi_\theta(a|s)$。目前常见的做法是使用什么网络函数化 $\pi(s,a)$，网络的参数就是 θ，输入为 s，输出为 $\pi(a_1,s;\theta),\cdots$.

Advantage: 1. storage; 2. generalization

Basic idea

1. 定义一个 metric / objective function $J(\theta)$ 以定义最优策略（**How to define this metric?**）
2. （梯度上升）优化目标函数（**How to compute the gradient?**）

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t)$$

Differences between tabular and function representations

1. 最优策略 π^* 的定义不同
   (a) 表格形式下：策略 π^* 满足 $v_{\pi^*}(s) \geqslant v_\pi(s), \forall s$
   (b) 函数形式下：策略 π^* 最大化了一个预先定义的函数（作为标量度量，scalar metric）
2. 获得某 action 的概率 $\pi(s,a)$ 的方式不同
   (a) 表格形式下：直接查表
   (b) 函数形式下：重新计算一遍即可

3. 如何更新某个概率 $\pi(s,a)$

   (a) 表格形式下：直接更改相应的概率值

   (b) 函数形式下：直接改变参数，从而间接改变相应地概率值

9.1 Metrics to define optimal policies

Metric 1: Average value

最简单和直观的想法就是对所有的 state value 取加权平均，这样的 metric 被称为 average state value / average value：

$$\bar{v}_\pi = \sum_{s\in\mathcal{S}} d(s)v_\pi(s) \tag{56}$$

其中 $\bar{v}_\pi$ 为所有 state value 的加权平均，$d(s) \geqslant 0,\ \forall s$，由于 $\sum_{s\in\mathcal{S}} d(s) = 1$ 因此若将 $d \sim S$，那么就有

$$\bar{v}_\pi = \mathbb{E}[v_\pi(S)] = \sum_s p(s)v_\pi(s)$$

将上述公式写成向量形式（方便计算 $\bar{v}_\pi$ 梯度），就有

$$\bar{v}_\pi = \sum_{s\in\mathcal{S}} d(s)v_\pi(s) = d^T v_\pi$$

其中

$$v_\pi = [\ldots, v_\pi(s), \ldots]^T \in \mathbb{R}^{|\mathcal{S}|}, \quad d = [\ldots, d(s), \ldots]^T \in \mathbb{R}^{|\mathcal{S}|}.$$

Case 1: Independent on policy 最简单的情况是 d 的选择不依赖于 π，这样在计算梯度时就不需要对 d 的部分进行计算，此时将其记为 d_0。此时 d_0 也有不同的选择：

1. 将所有的 state 平等对待，即 $d_0(s) = 1/|\mathcal{S}|$
2. 可能只关心某一个状态 s_0（例如固定 s_0 处出发），此时可以赋值 $d_0(s_0) = 1, \quad d_0(s \neq s_0) = 0.$

Case 2: Dependent on policy 第二种情况是 d 的选择依赖于 π。一个常见的做法是 $d = d_\pi(s)$，即 policy π 的 stationary distribution（满足 $d_\pi^T P_\pi = d_\pi^T$）

Metric 2: Average reward

第二种metric 为 **average one-step reward / average reward**，即 immediate reward 的加权平均：

$$\bar{r}_\pi \doteq \sum_{s\in\mathcal{S}} d_\pi(s)r_\pi(s) = \mathbb{E}[r_\pi(S)] \tag{57}$$

其中 $S \sim d_\pi$，为 stationary distribution，$r_\pi(s)$ 为 state s 处的 immediate reward：

$$r_\pi(s) \doteq \sum_{a\in\mathcal{A}} \pi(a|s)r(s,a)$$

其中

$$r(s,a) = \mathbb{E}[R|s,a] = \sum_r rp(r|s,a)$$

求解过程为：

$$r(s,a) \to r_\pi(s) \to \bar{r}_\pi$$

Usual form: 在论文中，常见的形式是对于一个给定策略 π，已经生成了一个 trajectory，且 reward 为 $(R_{t+1}, R_{t+2}, \ldots)$，此时此 trajectory 的 single-step reward 为

$$\lim_{n\to\infty} \frac{1}{n}\mathbb{E}\left[R_{t+1} + R_{t+2} + \cdots + R_{t+n} | S_t = s_0\right] = \lim_{n\to\infty} \frac{1}{n}\mathbb{E}\left[\sum_{k=1}^{n} R_{t+k} | S_t = s_0\right]$$

其中 s_0 为初始 state.

值得注意的是无穷多步时初始位置已不再有意义（**see details in textbook**）：

$$\lim_{n\to\infty} \frac{1}{n}\mathbb{E}\left[\sum_{k=1}^{n} R_{t+k} | S_t = s_0\right] = \lim_{n\to\infty} \frac{1}{n}\mathbb{E}\left[\sum_{k=1}^{n} R_{t+k}\right] \overset{\text{大数定律}}{=\!=\!=} \sum_s d_\pi(s) r_\pi(s) = \bar{r}_\pi$$

Note 16. *1.* 由于上述的 π 都是被 θ 参数化的，因此定义的 *metric* 都是 θ 的函数

2. 实际上我们只介绍了 *discounted case* $\gamma \in [0,1)$，没有介绍 *undiscounted case*，虽然在 *immediate reward* 下二者的 *metric* 相同，但后者事实上更复杂(***see details in textbook***)

3. 上面介绍的两种 *metric* 实际上在 *discounted case* 下有如下关系：

$$\bar{r}_\pi = (1-\gamma)\bar{v}_\pi$$

因此优化他们的效果是一样的

4. ***Test***

$$\begin{aligned} J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1}\right] &= \sum_{s\in\mathcal{S}} d(s)\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | S_0 = s\right] \\ &= \sum_{s\in\mathcal{S}} d(s) v_\pi(s) = \bar{v}_\pi \end{aligned}$$

Small Summary: 关于两种 metric 都有两种等价的定义

1. **Average value:**

   (a)
   $$J(\theta) = \sum_{s\in\mathcal{S}} d(s) v_\pi(s)$$

   (b)
   $$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1}\right]$$

2. **Average Reward:**

   (a)
   $$J(\theta) = \sum_{s\in\mathcal{S}} d_\pi(s) r_\pi(s)$$

(b)

$$J(\theta) = \lim_{n\to\infty} \frac{1}{n}\mathbb{E}\left[\sum_{k=1}^{n} R_{t+k}|S_t = s_0\right]$$

9.2 Gradients of the metrics

Example

事实上计算 metric 的梯度是 policy gradient methods 中最复杂的部分，因为有很多的情况，例如

1. 区分使用的 metric 是 $\bar{v}_\pi, \bar{r}_\pi, \bar{v}_\pi^0$
2. 区分是 discounted case 还是discounted case

此处仅做简要介绍(**see details in textbook**).

我们直接给出梯度的公式：

$$\nabla_\theta J(\theta) = \sum_{s\in\mathcal{S}} \eta(s) \sum_{a\in\mathcal{A}} \nabla_\theta \pi(a|s,\theta) q_\pi(s,a) \tag{58}$$

其中 $J(\theta)$ 可以为 $\bar{v}_\pi, \bar{r}_\pi$ 和 $\bar{v}_\pi^0$；$=$ 可以为严格相等(=)、约等于（$\approx$）与成比例（$\propto$）；η 为分布/ state 的权重.

事实上有

$$\begin{aligned}
\nabla_\theta \bar{r}_\pi &\simeq \sum_s d_\pi(s) \sum_a \nabla_\theta \pi(a|s,\theta) q_\pi(s,a) \\
\nabla_\theta \bar{v}_\pi &= \frac{1}{1-\gamma} \nabla_\theta \bar{r}_\pi \quad \text{(in discounted case)} \\
\nabla_\theta \bar{v}_\pi^0 &= \sum_{s\in\mathcal{S}} \rho_\pi(s) \sum_{a\in\mathcal{A}} \nabla_\theta \pi(a|s,\theta) q_\pi(s,a)
\end{aligned}$$

其中第一个式子中 discounted case 是约等于，undiscounted case 是严格等于.

根据 $\nabla_\theta \ln \pi(a|s,\theta) = \frac{\nabla_\theta \pi(a|s,\theta)}{\pi(a|s,\theta)}$可以得到如下非常有用的形式

$$\begin{aligned}
\nabla_\theta J(\theta) &= \sum_{s\in\mathcal{S}} \eta(s) \sum_{a\in\mathcal{A}} \nabla_\theta \pi(a|s,\theta) q_\pi(s,a) \\
&= \sum_s d(s) \sum_a \pi(a \mid s,\theta) \nabla_\theta \ln \pi(a \mid s,\theta) q_\pi(s,a) \\
&= \mathbb{E}_{S\sim d}\left[\sum_a \pi(a|S,\theta) \nabla_\theta \ln \pi(a|S,\theta) q_\pi(S,a)\right] \\
&= \mathbb{E}\left[\nabla_\theta \ln \pi(A|S,\theta) q_\pi(S,A)\right] \\
&\approx \nabla_\theta \ln \pi(a|s,\theta) q_\pi(s,a)
\end{aligned}$$

这样做的目的是求期望可以用采样来替代（变成 stochastic gradient）.

Note 17. 为保证 $\ln \pi(a|s,\theta)$ 合理，需要保证 $\pi(a|s,\theta) > 0$，因此 *greedy* 的方式是不可以的，可以使用 *softmax function* $\pi(a|s,\theta) = \frac{e^{h(s,a,\theta)}}{\sum_{a'\in\mathcal{A}} e^{h(s,a',\theta)}}$，其中 $h(s,a',\theta)$ 为另外的函数，例如神经网络，直接将其输出层定为 *softmax*.

9.3 Gradient-ascent algorithm(REINFORCE)

Gradient-ascent algorithm

最大化 $J(\theta)$ 的梯度上升算法理论上为

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta) = \theta_t + \alpha \mathbb{E}\left[\nabla_\theta \ln \pi(A|S,\theta_t) q_\pi(S,A)\right]$$

但是求期望时需要预先知道分布，这可能并不知道，所以就要使用如下随机版本（stochastic gradient）

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \ln \pi(a_t|s_t,\theta_t) q_\pi(s_t,a_t)$$

但仍然需要对 q_π （未知）进行替换，得到如下梯度

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \ln \pi(a_t|s_t,\theta_t) q_t(s_t,a_t) \tag{59}$$

How to do sampling? 由于我们需要通过采样将期望进行替换，即

$$\mathbb{E}_{S\sim d, A\sim\pi}\left[\nabla_\theta \ln \pi(A|S,\theta_t) q_\pi(S,A)\right] \longrightarrow \nabla_\theta \ln \pi(a|s,\theta_t) q_\pi(s,a)$$

1. 采样 S 时，$S \sim d$，其中 d 是 π 的 long run behavior，但是一般不会这么做，有数据就不错了，不会等那么久
2. 采样 A时，$A \sim \pi(A|S,\theta)$，a_t 应当来自当前策略 $\pi(\theta_t)$ 下的 state s_t，因此 policy gradient method 是 on-policy 的[a].

How to do interpret this algorithm? 由于 $\nabla_\theta \ln \pi(a_t|s_t,\theta_t) = \frac{\nabla_\theta \pi(a_t|s_t,\theta_t)}{\pi(a_t|s_t,\theta_t)}$，因此式(59)可以写为：

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha \nabla_\theta \ln \pi(a_t|s_t,\theta_t) q_t(s_t,a_t) = \theta_t + \alpha \underbrace{\left(\frac{q_t(s_t,a_t)}{\pi(a_t|s_t,\theta_t)}\right)}_{\beta_t} \nabla_\theta \pi(a_t|s_t,\theta_t) \\ &\triangleq \theta_t + \alpha\beta_t \nabla_\theta \pi(a_t|s_t,\theta_t)\end{aligned} \tag{60}$$

这样 $\alpha\beta_t$ 就可以作为新的步长。并且可以看出，实际上式(60)是在优化 π，因为将 $\pi(a_t|s_t,\theta_t)$ 看作是 $f(\theta_t)$，那么显然这就是牛顿法。这就要保证新的步长 $\alpha\beta_t$ 需要很小.

根据微分，当 $\theta_{t+1} - \theta_t$ 非常小时，有

$$\pi(a_t|s_t,\theta_{t+1}) \approx \pi(a_t|s_t,\theta_t) + (\nabla_\theta \pi(a_t|s_t,\theta_t))^T(\theta_{t+1}-\theta_t) = \pi(a_t|s_t,\theta_t) + \alpha\beta_t \|\nabla_\theta \pi(a_t|s_t,\theta_t)\|^2$$

因此 $\beta_t > 0$ 则 $\pi(a_t|s_t,\theta_{t+1}) > \pi(a_t|s_t,\theta_t)$；若 $\beta_t < 0$ 则 $\pi(a_t|s_t,\theta_{t+1}) < \pi(a_t|s_t,\theta_t)$.

Details about β_t: 事实上 $\beta_t = \frac{q_t(s_t,a_t)}{\pi(a_t|s_t,\theta_t)}$ 是在平衡探索(exploration) 和剥削(exploitation)

1. **exploitation:** 可以看到 β_t 与 $q_t(s_t,a_t)$ 成正比，因此

   $$q_t\text{大} \to \beta_t\text{大} \to \pi(a_t|s_t)\text{大}$$

   也就是说对于 action value 更大的 action (q_t 更大)，当前的 policy 倾向于选择它（$\pi_t(a_t|s_t)$）大，因此是一种 exploitation

2. **exploration:** 可以看到 β_t 与 $\pi(a_t|s_t,\theta_t)$ 成反比，因此

$$\pi(a_t|s_t)\text{小} \to \beta_t\text{大} \to \pi(a_t|s_t)\text{大}$$

也就是说当前时间步 t 选择某个 action 的概率小，那么下一时间步 $t+1$ 选择该 action 的概率会变大，这就是一种 exploration.

[a]也有 off-policy 的情况，但是需要额外技巧

REINFORCE

在式(59)中，如果使用了 MC 的方法得到 $q_t(s_t,a_t)$，即采样得到一个 episode，计算相应的 return(g_t)，赋值给 q_t，那么就称其为 **REINFORCE** 算法，其是最早也是最简单的 policy gradient 的算法.

Algorithm 14 Policy Gradient by Monte Carlo (REINFORCE)

Initialization: A parameterized function $\pi(a \mid s,\theta)$, $\gamma \in (0,1)$, and $\alpha > 0$.
Aim: Search for an optimal policy maximizing $J(\theta)$.
for the kth iteration **do**
  Select s_0 and generate an episode following $\pi(\theta_k)$. Suppose the episode is $\{s_0,a_0,r_1,\ldots,s_{T-1},a_{T-1},r_T\}$. ▷ **offline**(because need a full episode)
  for $t = 0,1,\ldots,T-1$ **do**
    Value update:

$$q_t(s_t,a_t) = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$$

    Policy update:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \ln \pi(a_t \mid s_t,\theta_t) q_t(s_t,a_t)$$

  end for
  Set $\theta_k = \theta_T$
end for

First updated: January 16, 2025
Last updated: January 18, 2025

10 Lecture 10, Actor-Critic Methods

Bilibili:Lecture 10, Actor-Critic Methods

Introduction

Actor-Critic Methods 仍然是 policy gradient method，实际上其将 value function approximation 结合进了 policy gradient method。这里 actor 指的是 policy update，因为 policy 需要用于实施行动，就是一个 actor 的角色； critic 指的是 policy evaluation / value estimation，因为 action / state value 需要估计，就是一个 critic 的角色.

10.1 QAC: the simplest actor-critic

QAC

首先回顾一下 policy gradient 的 basic idea:

1. 定义 scalar metric $J(\theta)$，其可以为 $\bar{v}_\pi$ 或 $\bar{r}_\pi$
2. 最大化 $J(\theta)$ 的 gradient ascent algorithm 理论上为：

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t) = \theta_t + \alpha \mathbb{E}_{S\sim\eta, A\sim\pi} \left[\nabla_\theta \ln \pi(A|S,\theta_t) q_\pi(S,A)\right]$$

3. 将理论上的期望计算转化为采样的 stocahstic 版本：

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \ln \pi(a_t|s_t,\theta_t) q_t(s_t,a_t)$$

从第3点也可以看出，整个式子就是 "actor"，因为改变 θ 实际上就是在 update policy； $q_t(s_t,a_t)$ 的算法就是 "critic".

不同的估计 action value q_t 的方法也就产生了不同的算法：

1. 若使用 MC 的方法，那么对应的算法就称为 REINFORCE 或者 Monte Carlo policy gradient
2. 若使用 TD 的方法，那么对应的算法就称为 actor-critic

Algorithm 15 The Simplest Actor-Critic Algorithm (QAC)

Aim: Search for an optimal policy by maximizing $J(\theta)$.
for each time step t in each episode **do**
  Generate a_t following $\pi(a \mid s_t, \theta_t)$, observe r_{t+1}, s_{t+1}, and then generate a_{t+1} following $\pi(a \mid s_{t+1}, \theta_t)$. ▷ Generate the experience sample $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ for SARSA
  Critic (value update):

$$w_{t+1} = w_t + \alpha_w \left[r_{t+1} + \gamma q(s_{t+1}, a_{t+1}, w_t) - q(s_t, a_t, w_t)\right] \nabla_w q(s_t, a_t, w_t)$$

  Actor (policy update):

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \ln \pi(a_t \mid s_t, \theta_t) q(s_t, a_t, w_{t+1})$$

end for

1. **critic:** SARSA + value function approximation
2. **actor:** the policy update algorithm
3. **on-policy:** 由于理论上进行梯度上升时 $\mathbb{E}_{S\sim\eta,A\sim\pi}$ 中 $\theta\sim\pi$，这里的 π 既是 behavior policy(要按照这个采样)，也是 target policy
4. 虽然是 on-policy 的，但是由于算法本身具有随机性（探索能力），因此无需使用 $\epsilon-$greedy
5. 此算法是最简单的 actor-critic 算法，Q 是指 action value $q-$value

10.2 Advantage actor-critic(A2C)

A2C: Baseline invariance

Advantage actor-critic(A2C) 是QAC 的一个推广，因为名称有两个 "a" 所以简称 A2C。其核心 idea 是引入一个 baseline / 偏置量已减小估计的方差:

$$\nabla_\theta J(\theta)=\mathbb{E}_{S\sim\eta,A\sim\pi}\left[\nabla_\theta \ln\pi(A|S,\theta_t)(q_\pi(S,A)-b(S))\right] \tag{61}$$

其中 $b(S)$ 为 S 的标量函数，被称为 baseline.

下面证明 baseline 有如下两个特点:

1. **Valid:** 引入 baseline 不会对 $\nabla_\theta J(\theta)$ 的值造成影响
2. **Useful:** 引入 baseline 可以降低 variance，提高采样结果稳定性

Valid: 引入 baseline 不会对 $\nabla_\theta J(\theta)$ 的值造成影响

Proof. 为证明

$$\mathbb{E}_{S\sim\eta,A\sim\pi}\left[\nabla_\theta \ln\pi(A|S,\theta_t)q_\pi(S,A)\right]=\mathbb{E}_{S\sim\eta,A\sim\pi}\left[\nabla_\theta \ln\pi(A|S,\theta_t)(q_\pi(S,A)-b(S))\right]$$

即要证明

$$\mathbb{E}_{S\sim\eta,A\sim\pi}\left[\nabla_\theta \ln\pi(A|S,\theta_t)b(S)\right]=0$$

将上式展开即可得到结论:

$$\begin{aligned}
&\mathbb{E}_{S\sim\eta,A\sim\pi}\left[\nabla_\theta \ln\pi(A|S,\theta_t)b(S)\right]=\sum_{s\in\mathcal{S}}\eta(s)\sum_{a\in\mathcal{A}}\pi(a|s,\theta_t)\nabla_\theta \ln\pi(a|s,\theta_t)b(s)\\
&=\sum_{s\in\mathcal{S}}\eta(s)\sum_{a\in\mathcal{A}}\nabla_\theta\pi(a|s,\theta_t)b(s)=\sum_{s\in\mathcal{S}}\eta(s)b(s)\sum_{a\in\mathcal{A}}\nabla_\theta\pi(a|s,\theta_t)\\
&=\sum_{s\in\mathcal{S}}\eta(s)b(s)\nabla_\theta\sum_{a\in\mathcal{A}}\pi(a|s,\theta_t)=\sum_{s\in\mathcal{S}}\eta(s)b(s)\nabla_\theta 1=0
\end{aligned}$$

□

Useful: 引入 baseline 可以降低 variance，提高采样结果稳定性

Proof. 下面证明 $b(S)$ 会对 $\text{var}(X)$ 造成影响，为此我们考虑 $\text{tr}[\text{var}(X)] = \mathbb{E}[X^TX] - \bar{x}^T\bar{x}$，就有

$$\begin{aligned}\mathbb{E}[X^TX] &= \mathbb{E}\left[(\nabla_\theta \ln \pi)^T(\nabla_\theta \ln \pi)(q(S,A)-b(S))^2\right] \\ &= \mathbb{E}\left[\|\nabla_\theta \ln \pi\|^2(q(S,A)-b(S))^2\right]\end{aligned}$$

(**see details in textbook**) □

因此可以看出，方差较小的算法会使得在采样时采样得到的 x 接近 $\mathbb{E}[X]$，所以我们希望通过改变 baseline 来尽量降低方差，这就需要选择最优的 b.

Optimal baseline: 实际上最优的 baseline 为(**see details in textbook**)

$$b^*(s) = \frac{\mathbb{E}_{A\sim\pi}[\|\nabla_\theta \ln \pi(A|s,\theta_t)\|^2 q(s,A)]}{\mathbb{E}_{A\sim\pi}[\|\nabla_\theta \ln \pi(A|s,\theta_t)\|^2]} \tag{62}$$

但是由于其有些复杂，在平常的算法中常使用如下 baseline:

$$b(s) = \mathbb{E}_{A\sim\pi}[q(s,A)] = v_\pi(s) \tag{63}$$

A2C: algorithm

根据前面的分析，我们令 $b(s) = v_\pi(s)$，这样得到的 gradient-ascent algorithm 为

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha\mathbb{E}\left[\nabla_\theta \ln \pi(A|S,\theta_t)[q_\pi(S,A) - v_\pi(S)]\right] \\ &\doteq \theta_t + \alpha\mathbb{E}\left[\nabla_\theta \ln \pi(A|S,\theta_t)\delta_\pi(S,A)\right]\end{aligned} \tag{64}$$

其中 $\delta_\pi(S,A) \doteq q_\pi(S,A) - v_\pi(S)$ 为优势函数(advantage function)，称此名称是因为根据定义 $v_\pi(S) = \sum_a \pi(a|S)q_\pi$，因此若 $\delta_\pi > 0$，就说明此 action 比平均的 action 要好，是有优势的.

这样得到梯度上升算法为：

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha\nabla_\theta \ln \pi(a_t|s_t,\theta_t)[q_t(s_t,a_t) - v_t(s_t)] = \theta_t + \alpha\nabla_\theta \ln \pi(a_t|s_t,\theta_t)\delta_t(s_t,a_t) \\ &= \theta_t + \alpha\frac{\nabla_\theta \pi(a_t|s_t,\theta_t)}{{}_*\pi(a_t|s_t,\theta_t)}\delta_t(s_t,a_t) = \theta_t + \alpha\underbrace{\left(\frac{\delta_t(s_t,a_t)}{\pi(a_t|s_t,\theta_t)}\right)}_{\text{step size}}\nabla_\theta \pi(a_t|s_t,\theta_t)\end{aligned}$$

同上一节一样，step size 此时是探索(exploration) 和剥削(exploitation)的一个平衡。并且由于我们更关心相对值，因此上一节的 q_t 没有这一节的 δ_t 好.

Trick: 这里有一个小的技巧是将 δ_t 进行替换：

$$\delta_t = q_t(s_t,a_t) - v_t(s_t) \to r_{t+1} + \gamma v_t(s_{t+1}) - v_t(s_t)$$

由于在期望上这二者相同

$$\mathbb{E}[q_\pi(S,A) - v_\pi(S)|S=s_t, A=a_t] = \mathbb{E}\left[R + \gamma v_\pi(S') - v_\pi(S)|S=s_t, A=a_t\right]$$

因此这样替换是合理可行的。并且此时 δ_t 只需要一个神经网络近似 v_t 即可，原先需要两个网络分别近似 q_t 和 v_t.

Algorithm 16 Advantage Actor-Critic (A2C) or TD Actor-Critic(on-policy)

1: **Aim:** Search for an optimal policy by maximizing $J(\theta)$.
2: **for** each time step t in each episode **do**
3: Generate a_t following $\pi(a \mid s_t, \theta_t)$ and then observe r_{t+1}, s_{t+1}.
4: **TD error (advantage function):**

$$\delta_t = r_{t+1} + \gamma v(s_{t+1}, w_t) - v(s_t, w_t)$$

5: **Critic (value update):**

$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w v(s_t, w_t)$$

6: **Actor (policy update):**

$$\theta_{t+1} = \theta_t + \alpha_\theta \delta_t \nabla_\theta \ln \pi(a_t \mid s_t, \theta_t)$$

7: **end for**

10.3 Off-policy actor-critic

Review

由于计算梯度时理论期望要使用 $A \sim \pi$(见下式)，因此 behavior policy 和 target policy 相同.

$$\nabla_\theta J(\theta) = \mathbb{E}_{S\sim\eta, A\sim\pi}[*]$$

因此前面介绍的两种 AC 算法都是 off-policy 的，如果想使用一些既有经验，就需要将算法改进为 off-policy 的，方法就是使用 **importance sampling**[a].

[a] importance sampling 也可以在别的地方例如 MC、TD 使用

Importance sampling

Motivating example 首先给出一个例子: 我们已经知道根据大数定律，当未知一个概率分布时可以根据采样对其进行估算，例如想要估计概率分布 p_0，就可以依 p_0 采样得到 samples $\{x_i\}$，然后得到

$$\bar{x} = \sum_{i=1}^{n} \frac{1}{n} x_i \to \mathbb{E}_{X\sim p_0}[X]$$

但是我们希望实现依据一个已知但是不同于 p_0 的分布 p_1 采样，最后估算出 p_0 的期望，这也就对应了 off-policy 中使用与 target policy 不同的 behavior policy 产生策略，但是最后估计的是 target policy

Importance sampling 注意到我们实际上可以使用 p_1 代替 p_0 进行采样，即:

$$\mathbb{E}_{X\sim p_0}[X] = \sum_x p_0(x) x = \sum_x p_1(x) \underbrace{\frac{p_0(x)}{p_1(x)}}_{f(x)} x = \mathbb{E}_{X\sim p_1}[f(X)]$$

这样就可以使用 p_1 采样，计算 $\bar{f}$

$$\bar{f} \doteq \frac{1}{n}\sum_{i=1}^{n} f(x_i), \quad \text{where } x_i \sim p_1$$

最后得到

$$\mathbb{E}_{X\sim p_0}[X] \approx \bar{f} = \frac{1}{n}\sum_{i=1}^{n} f(x_i) = \frac{1}{n}\sum_{i=1}^{n} \frac{p_0(x_i)}{p_1(x_i)} x_i$$

其中 $\frac{p_0(x_i)}{p_1(x_i)}$ 被称为重要性权重(importance weight)，一个直观的解释是若某样本点 x_i 在 p_0 下很容易被采到但是在 p_1 下很难被采到，那么就要很珍惜这个样本点，直观上来说这个样本点的权重应该比较大，这与 $\frac{p_0(x_i)}{p_1(x_i)}$ 此时很大刚好对应.

Note 18. 这里的逻辑应该是这样的：如果可以使用 p_0 进行采样，那么直接根据大数定律使用 $\bar{x} = \sum_{i=1}^{n} \frac{1}{n} x_i \to \mathbb{E}_{X\sim p_0}[X]$ 即可，但是现在我们只能根据 p_1 进行采样，并且同时对于采出来的样本点 $\{x_i\}$，只知道这些离散点的 $p_0(x_i)$（同时知道所有点的 $p_1(x)$），例如使用一个神经网络逼近 p_0，但是此时 p_0 都没有表达式。由于定义式 $\mathbb{E}_{X\sim p_0}[X] = \sum_x p_0(x)x$ 需要知道所有点的 p_0 即连续情形，因此无法计算，最终只能通过转化成 p_1 下有限个点来计算 p_0 的期望。总的来说，就是将本该在 p_0 下使用无限个点（连续情形）才能计算的期望转化为在 p_1 下使用有限个点（离散情形）逼近的期望（大数定律保证结果）.

Off-policy policy gradient

考虑 off-policy 情形，即 behavior policy 不同于 target policy π 时的情况，设此时 begavior policy 为 β。此时 metric 定义为

$$J(\theta) = \sum_{s\in\mathcal{S}} d_\beta(s) v_\pi(s) = \mathbb{E}_{S\sim d_\beta}[v_\pi(S)]$$

其中 d_β 为 policy β 的 stationary distribution.

直接给出 $J(\theta)$ 的梯度如下：

$$\nabla_\theta J(\theta) = \mathbb{E}_{S\sim\rho, A\sim\beta}\left[\frac{\pi(A|S,\theta)}{\beta(A|S)} \nabla_\theta \ln \pi(A|S,\theta) q_\pi(S,A)\right] \tag{65}$$

其中 $\gamma \in (0,1)$ 为 discounted rate, ρ 为 state distribution.(**see details in textbook**)

此时仍然加上 baseline $b(s)$，得到

$$\nabla_\theta J(\theta) = \mathbb{E}_{S\sim\rho, A\sim\beta}\left[\frac{\pi(A|S,\theta)}{\beta(A|S)} \nabla_\theta \ln \pi(A|S,\theta) \left(q_\pi(S,A) - b(S)\right)\right]$$

根据前面的结构，令 $b(s) = v_\pi(s)$，得到

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\frac{\pi(A|S,\theta)}{\beta(A|S)} \nabla_\theta \ln \pi(A|S,\theta) \left(q_\pi(S,A) - v_\pi(S)\right)\right]$$

对应的 stocahtisc 版本即

$$\theta_{t+1} = \theta_t + \alpha_\theta \frac{\pi(a_t|s_t,\theta_t)}{\beta(a_t|s_t)} \nabla_\theta \ln \pi(a_t|s_t,\theta_t) \left(q_t(s_t,a_t) - v_t(s_t)\right)$$

与 on-policy 类似地有

$$q_t(s_t,a_t)-v_t(s_t)\approx r_{t+1}+\gamma v_t(s_{t+1})-v_t(s_t)\doteq\delta_t(s_t,a_t)$$

如此得到算法

$$\begin{aligned}\theta_{t+1}&=\theta_t+\alpha_\theta\frac{\pi(a_t|s_t,\theta_t)}{\beta(a_t|s_t)}\nabla_\theta\ln\pi(a_t|s_t,\theta_t)\delta_t(s_t,a_t)\\&=\theta_t+\alpha_\theta\left(\frac{\delta_t(s_t,a_t)}{\beta(a_t|s_t)}\right)\nabla_\theta\pi(a_t|s_t,\theta_t)\end{aligned}$$

Algorithm 17 Off-policy Actor-Critic Based on Importance Sampling

1: **Initialization:** A given behavior policy $\beta(a\mid s)$. A target policy $\pi(a\mid s,\theta_0)$ where θ_0 is the initial parameter vector. A value function $v(s,w_0)$ where w_0 is the initial parameter vector.
2: **Aim:** Search for an optimal policy by maximizing $J(\theta)$.
3: **for** each time step t in each episode **do**
4: Generate a_t following $\beta(s_t)$ and then observe r_{t+1},s_{t+1}.
5: **TD error (advantage function):**

$$\delta_t=r_{t+1}+\gamma v(s_{t+1},w_t)-v(s_t,w_t)$$

6: **Critic (value update):**

$$w_{t+1}=w_t+\alpha_w\frac{\pi(a_t\mid s_t,\theta_t)}{\beta(a_t\mid s_t)}\delta_t\nabla_w v(s_t,w_t)$$

7: **Actor (policy update):**

$$\theta_{t+1}=\theta_t+\alpha_\theta\frac{\pi(a_t\mid s_t,\theta_t)}{\beta(a_t\mid s_t)}\delta_t\nabla_\theta\ln\pi(a_t\mid s_t,\theta_t)$$

8: **end for**

10.4 Deterministic Actor-Critic(DPG)

Deterministic Actor-Critic

值得注意的是前面介绍的所有的算法的 $\pi(a|s,\theta)>0$，也就是说都带有 exploration 性，都是 stochastic 策略。但是 $\pi(a|s,\theta)>0$ 有 action 个数是有限个的隐含要求，因为 $\sum_a\pi(a|s;\theta)=1$，因此无法处理 action 无限即 continuous action 的情形.

下面考虑 deterministic 的情形，此时重新定义一个 deterministic policy

$$a=\mu(s,\theta)\doteq\mu(s)$$

这与 $\pi(a|s;\theta)$ 不同的是直接输出 action，而不是采取某些 action 对应的概率值.

1. μ 是状态空间到动作空间的映射：$\mu:\mathcal{S}\to\mathcal{A}$
2. μ 实际中可以用参数为 θ 的神经网络替代，输入 s 输出 a

与之前的推导过程类似，首先定义 discounted case 下的 metric

$$J(\theta) = \mathbb{E}[v_\mu(s)] = \sum_{s\in\mathcal{S}} d_0(s) v_\mu(s)$$

其中 $d_0(s)$ 为概率分布，满足 $\sum_{s\in\mathcal{S}} d_0(s) = 1$，最简单的情形是与 μ 不相关（例如只有某 s_0 的 $d_0(s_0) = 1$，或 behavior policy 的 stationary distribution）.

直接给出 discounted case 下的梯度表达式为

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_{s\in\mathcal{S}} \rho_\mu(s) \nabla_\theta \mu(s) \left(\nabla_a q_\mu(s,a)\right)|_{a=\mu(s)} \\ &= \mathbb{E}_{S\sim\rho_\mu}\left[\nabla_\theta \mu(S) \left(\nabla_a q_\mu(S,a)\right)|_{a=\mu(S)}\right]\end{aligned}$$

其中 $\gamma \in (0,1)$ 为 discounted rate, ρ_μ 为 state distribution.(**see details in textbook**)

注意在此梯度的表达式中最后已经没有 a，即对 target policy 的更新已经不再和 behavior policy 相关，因此天然是 off-policy 的.

再代入梯度上升算法中：

$$\theta_{t+1} = \theta_t + \alpha_\theta \mathbb{E}_{S\sim\rho_\mu}\left[\nabla_{\theta\mu(S)} \left(\nabla_a q_\mu(S,a)\right)|_{a=\mu(S)}\right]$$

使用 stochastic 版本:

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \mu(s_t) \left(\nabla_a q_\mu(s_t,a)\right)|_{a=\mu(s_t)}$$

Algorithm 18 Deterministic Actor-Critic Algorithm

1: **Initialization:** A given behavior policy $\beta(a \mid s)$. A deterministic target policy $\mu(s,\theta_0)$ where θ_0 is the initial parameter vector. A value function $v(s,w_0)$ where w_0 is the initial parameter vector.
2: **Aim:** Search for an optimal policy by maximizing $J(\theta)$.
3: **for** each time step t in each episode **do**
4: Generate a_t following β and then observe r_{t+1}, s_{t+1}.
5: **TD error:**

$$\delta_t = r_{t+1} + \gamma q(s_{t+1}, \mu(s_{t+1},\theta_t), w_t) - q(s_t,a_t,w_t)$$

6: **Critic (value update):**

$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w q(s_t,a_t,w_t)$$

7: **Actor (policy update):**

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \mu(s_t,\theta_t) \left(\nabla_a q(s_t,a,w_{t+1})\right)\Big|_{a=\mu(s_t)}$$

8: **end for**

Note 19. 1. *DPG* 算法是 *off-policy* 的，*behavior policy* β 可以与 *target policy* μ 不同

2. β 也可以是 μ+*noise*，这样就有了探索性（此时就是 *on-policy* 的实现方式）

3. 当使用线性函数逼近 $q(s,a,w)$ 时，即 $q(s,a,w)=\phi^T(s,a)$，此时算法为 *DPG*（原文中也是如此）

4. 当使用神经网络逼近 $q(s,a,w)$ 时，被称为 *deep deterministic policy gradient(DDPG)*

Last updated: February 19, 2025

References