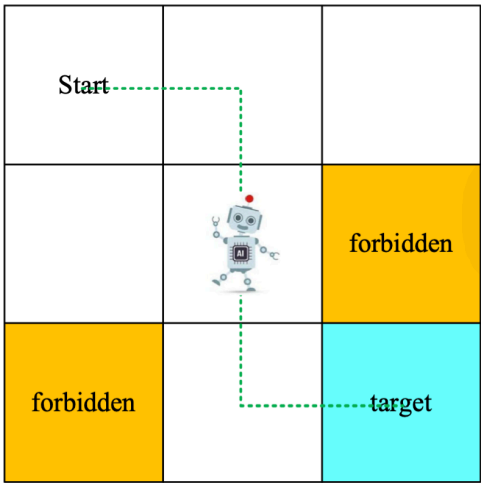
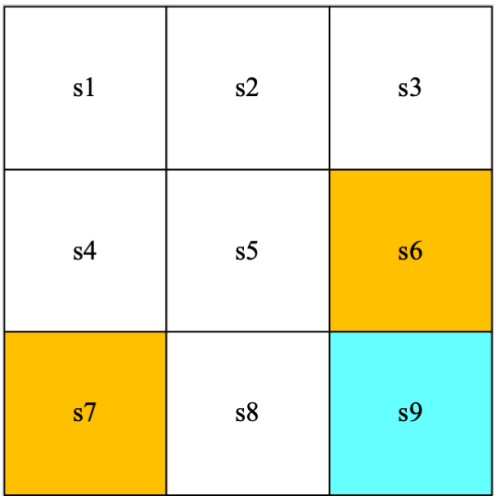
> 参考西湖大学赵世钰老师 github

# Ch1.基本概念

grid-world example，机器人从 start 走到 target，并且尽量避免 forbidden

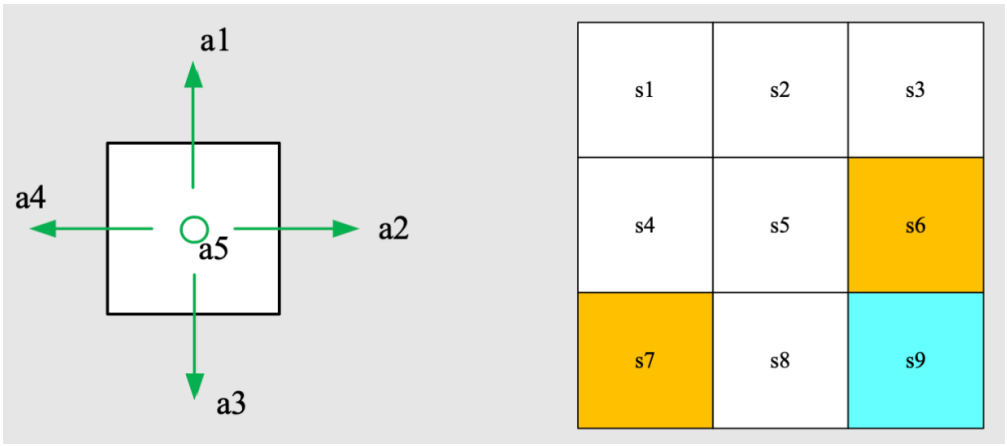



## State：agent相对于环境的状态




- state 指 agent 所在的位置
- state place 指所有 state 的集合 $S = \{s_i\}_{i=1}^{9}$

## Action：对于每一个 State，可以采取的行动

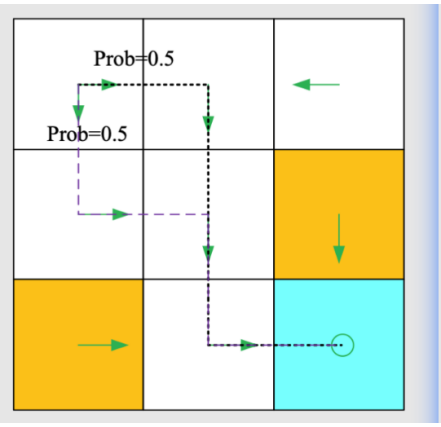


- 每个状态存在 5 个可能的 actions，即 $a_1, a_2, a_3, a_4, a_5$
- action place 指所有 action 的集合 $A(s_i) = \{a_i\}_{i=1}^5$ ，action 和 state 相互依赖

## State transition：采取 action 时，state 转移的过程

- State transition 表示方式
  - $s_1 \xrightarrow{a_2} s_2$
  - $s_1 \xrightarrow{a_1} s_1$（边界碰壁撞回）
  - $s_5 \xrightarrow{a_2} s_6$（forbidden 允许进入但有惩罚）$s_5 \xrightarrow{a_2} s_5$（forbidden 碰壁撞回）
- State transition probability
  - 确定的状态转移：$p(s_2|s_1, a_2) = 1 \qquad p(s_i|s_1, a_2) = 1, \forall i \neq 2$
  - 随机的状态转移：$p(s_2|s_1, a_2) = 0.5 \qquad p(s_5|s_1, a_2) = 0.5$

## Policy：agent 在某个 state 应该采取的 action

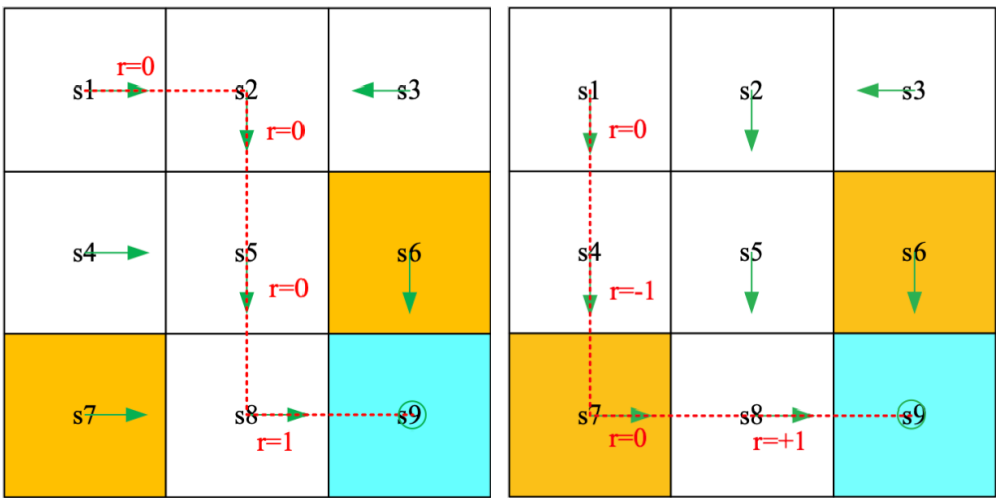



$$\pi(a_1|s_1) = 0$$
$$\pi(a_2|s_1) = 0.5$$
$$\pi(a_3|s_1) = 0.5$$
$$\pi(a_4|s_1) = 0$$
$$\pi(a_5|s_1) = 0$$

## Reward：采取 action 后的奖励

- $r_{bound} = -1 \quad r_{forbid} = -1 \quad r_{target} = +1 \quad r_{otherwise} = 0$
- reward 只与当前 state 和 action 相关，与 next state 无关

## Trajectory：state-action-reward 链




- 一个 trajectory 的所有 reward 叫做 return，比较 return 可以评估哪个 policy 更好
- $s_1 \xrightarrow[r=0]{a_2} s_2 \xrightarrow[r=0]{a_3} s_5 \xrightarrow[r=0]{a_3} s_8 \xrightarrow[r=1]{a_2} s_9$ ，$return_1 = 0 + 0 + 0 + 1 = 1$

- $s_1 \xrightarrow[r=0]{a_3} s_4 \xrightarrow[r=-1]{a_3} s_7 \xrightarrow[r=0]{a_2} s_8 \xrightarrow[r=+1]{a_2} s_9$ ，$return_2 = 0 - 1 + 0 + 1 = 0$
- discount return：引入 discount rate $\gamma \in (0, 1)$
  - 对于第一个 trajectory，如果一直停留在 $s_9$ ，return 会无限制增加
  - $discounted\ return = 0 + \gamma 0 + \gamma^2 0 + \gamma^3 1 + \gamma^4 1 + \gamma^5 1 + \ldots = \gamma^3(1 + \gamma + \gamma^2 + \ldots) = \gamma^3 \frac{1}{1-\gamma}$
  - $\gamma \to 0$ 更近视，$\gamma \to 1$ 更远视

## Episode：一个 resulting（到头）的 trajectory

- episode（会结束）可变为 continuing tasks（一直持续）
  - Option 1：将 target 视作特殊的 absorbing state，一旦达到，修改 action space 仅可继续留在 target 中，同时 reward 改为 0
  - Option 2：将 target 视作正常的 normal state，agent 仍可以退出 target 和进入 target，重新进入时 reward 仍为 1

## Markov decision process (MDP)

- 要素1：集合
  - State：$S$
  - Action：$A(s)$，$s \in S$
  - Reward：$R(s, a)$
- 要素2：概率分布
  - State transition probability：$p(s'|s, a)$
  - Reward probability：$p(r|s, a)$
- 要素3：决策
  - $\pi(a|s)$，$\sum_{a \in A(s)} \pi(a|s) = 1$
- **要素4：Markov性质，即无记忆性**
  - $p(s_{t+1}|a_{t+1}, s_t, \ldots, a_1, s_0) = p(s_{t+1}|a_{t+1}, s_t)$
  - $p(r_{t+1}|a_{t+1}, s_t, \ldots, a_1, s_0) = p(r_{t+1}|a_{t+1}, s_t)$

---

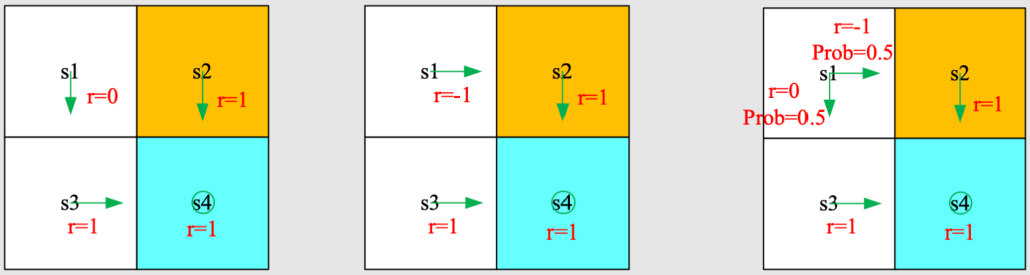
# Ch2. 贝尔曼公式

## State Value

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

$$v_\pi(s) = \mathbb{E}[G_t|S_t = s]$$
$$\text{其中 } G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

- return 与 state value 的区别
  - return 指单个 trajectory
  - state value 对多个 trajectory 求平均




Recall the returns obtained from $s_1$ for the three examples:

$$v_{\pi_1}(s_1) = 0 + \gamma 1 + \gamma^2 1 + \cdots = \gamma(1 + \gamma + \gamma^2 + \ldots) = \frac{\gamma}{1-\gamma}$$

$$v_{\pi_2}(s_1) = -1 + \gamma 1 + \gamma^2 1 + \cdots = -1 + \gamma(1 + \gamma + \gamma^2 + \ldots) = -1 + \frac{\gamma}{1-\gamma}$$

$$v_{\pi_3}(s_1) = 0.5\left(-1 + \frac{\gamma}{1-\gamma}\right) + 0.5\left(\frac{\gamma}{1-\gamma}\right) = -0.5 + \frac{\gamma}{1-\gamma}$$

# Bellman equation

**推导**

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[G_t \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} \mid S_t = s] + \gamma \mathbb{E}[G_{t+1} \mid S_t = s] \end{aligned}$$

$$\begin{aligned} \mathbb{E}[R_{t+1} \mid S_t = s] &= \sum_a \pi(a|s) \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] \\ &= \sum_a \pi(a|s) \sum_r p(r|s,a) r \end{aligned}$$

$$\begin{aligned} \mathbb{E}[G_{t+1} \mid S_t = s] &= \sum_{s'} \mathbb{E}[G_{t+1} \mid S_t = s, S_{t+1} = s'] p(s'|s) \\ &= \sum_{s'} \mathbb{E}[G_{t+1} \mid S_{t+1} = s'] p(s'|s) \quad \text{马尔可夫链无记忆性} \\ &= \sum_{s'} v_\pi(s') p(s'|s) \\ &= \sum_{s'} v_\pi(s') \sum_a p(s'|s,a) \pi(a|s) \end{aligned}$$

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[R_{t+1} \mid S_t = s] + \gamma \mathbb{E}[G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_r p(r|s,a) r + \gamma \sum_{s'} v_\pi(s') \sum_a p(s'|s,a) \pi(a|s) \\ &= \sum_a \pi(a|s) [\sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_\pi(s')], \quad \forall s \in S \end{aligned}$$

- 求解 $v_\pi(s)$ $v_\pi(s')$ 使用 bootstrapping
- 求解整个公式也叫做 policy evaluation，因为 policy 即 $\pi(a|s)$ 是给定的
- $p(r|s,a)$ $p(s'|s,a)$ 代表 dynamic model（后续又分 model-based / model-free ）

**矩阵表示**

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

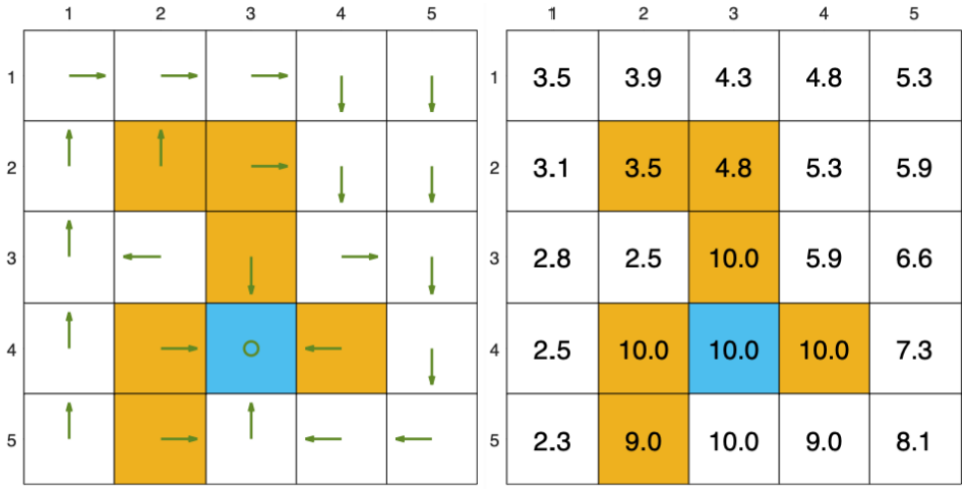
$$\text{其中} \quad r_\pi(s) \triangleq \sum_a \pi(a|s) \sum_r p(r|s,a) r, \quad p_\pi(s'|s) \triangleq \sum_a \pi(a|s) p(s'|s,a)$$

$$\begin{aligned} v_\pi &= r_\pi + \gamma P_\pi v_\pi \\ \text{其中} v_\pi &= [v_\pi(s_1), \ldots, v_\pi(s_n)]^T \in \mathbb{R}^n \\ r_\pi &= [r_\pi(s_1), \ldots, r_\pi(s_n)]^T \in \mathbb{R}^n \\ P_\pi &\in \mathbb{R}^{n \times n}, \quad \text{where } [P_\pi]_{ij} = p_\pi(s_j \mid s_i) \end{aligned}$$

$$\underbrace{\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ v_\pi(s_3) \\ v_\pi(s_4) \end{bmatrix}}_{v_\pi} = \underbrace{\begin{bmatrix} r_\pi(s_1) \\ r_\pi(s_2) \\ r_\pi(s_3) \\ r_\pi(s_4) \end{bmatrix}}_{r_\pi} + \gamma \underbrace{\begin{bmatrix} p_\pi(s_1|s_1) & p_\pi(s_2|s_1) & p_\pi(s_3|s_1) & p_\pi(s_4|s_1) \\ p_\pi(s_1|s_2) & p_\pi(s_2|s_2) & p_\pi(s_3|s_2) & p_\pi(s_4|s_2) \\ p_\pi(s_1|s_3) & p_\pi(s_2|s_3) & p_\pi(s_3|s_3) & p_\pi(s_4|s_3) \\ p_\pi(s_1|s_4) & p_\pi(s_2|s_4) & p_\pi(s_3|s_4) & p_\pi(s_4|s_4) \end{bmatrix}}_{P_\pi} \underbrace{\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ v_\pi(s_3) \\ v_\pi(s_4) \end{bmatrix}}_{v_\pi}$$

**求解 state value（评价 policy ）**

$$\begin{aligned} v_\pi &= r_\pi + \gamma P_\pi v_\pi \\ closed-form\ \ solution \quad v_\pi &= (I - \gamma P_\pi)^{-1} r_\pi \\ iterative\ \ solution\ (preferred) \quad v_{k+1} &= r_\pi + \gamma P_\pi v_k \quad v_k \to v_\pi = (I - \gamma P_\pi)^{-1} r_\pi,\ k \to \infty \end{aligned}$$




```python
import numpy as np

def build_gridworld(gamma=0.9, n=5, forbid=None, target=None, arrow_map=None):
    """
    r_doundary = r_forbidden = -1
    r_target = +1
    r_otherwise = 0
    gamma = 0.9
    """
    n_states = n * n
    reward = np.zeros(n_states)
    P = np.zeros((n_states, n_states))

    if forbid is None:
        forbid = []
    if target is None:
        target = 0
```

```
    for i in range(n):
        for j in range(n):
            s = i * n + j
            # target state
            if s == target:
                P[s, s] = 1
                reward[s] = 1
                continue
            action = arrow_map[i][j]
            ni, nj = i + action[0], j + action[1]
            if 0 <= ni < n and 0 <= nj < n:
                s2 = ni * n + nj
                P[s, s2] = 1
                # forbidden state
                if s2 in forbid:
                    reward[s] = -1
                # target state
                elif s2 == target:
                    reward[s] = 1
            # boundary state
            else:
                P[s, s] = 1
                reward[s] = -1
    return P, reward

def value_iteration(P, reward, gamma=0.9, n_iter=100):
    v = np.zeros_like(reward)
    # 迭代法, 初始全设置为0即可, slides ch2 P40有数学证明
    for _ in range(n_iter):
        v = reward + gamma * P @ v
    return v

def value_closed_form(P, reward, gamma=0.9):
    n_states = len(reward)
    I = np.eye(n_states)
    v = np.linalg.solve(I - gamma * P, reward)
    return v

def main():
    gamma = 0.9
    n = 5
    forbid = [6, 7, 12, 16, 18, 21]
    target = 17
    """
    actions = {
        (0, 1): 1,     # right
        (0, -1): -1,   # left
        (1, 0): 5,     # down
        (-1, 0): -5    # up
    }
    """
    arrow_map = [
        [(0, 1), (0, 1), (0, 1), (1, 0), (1, 0)],
```

```
        [(-1, 0), (-1, 0), (0, 1), (1, 0), (1, 0)],
        [(-1, 0), (0, -1), (1, 0), (0, 1), (1, 0)],
        [(-1, 0), (0, 1), (0, 0), (0, -1), (1, 0)],
        [(-1, 0), (0, 1), (-1, 0), (0, -1), (0, -1)]
    ]
    P, reward = build_gridworld(gamma, n, forbid, target, arrow_map)

    # 迭代法
    v_iter = value_iteration(P, reward, gamma)
    print("值迭代法结果:")
    print(np.round(v_iter.reshape(n, n), 2))

    # 闭式解法
    v_closed = value_closed_form(P, reward, gamma)
    print("\n矩阵公式法结果:")
    print(np.round(v_closed.reshape(n, n), 2))

if __name__ == "__main__":
    main()

"""
值迭代法结果:
[[ 3.49  3.87  4.3   4.78  5.31]
 [ 3.14  3.49  4.78  5.31  5.9 ]
 [ 2.82  2.54 10.    5.9   6.56]
 [ 2.54 10.   10.   10.    7.29]
 [ 2.29  9.   10.    9.    8.1 ]]

矩阵公式法结果:
[[ 3.49  3.87  4.3   4.78  5.31]
 [ 3.14  3.49  4.78  5.31  5.9 ]
 [ 2.82  2.54 10.    5.9   6.56]
 [ 2.54 10.   10.   10.    7.29]
 [ 2.29  9.   10.    9.    8.1 ]]
"""
```

## Action Value

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

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

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a|s) q_\pi(s,a) \\ &= \sum_a \pi(a|s)[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_\pi(s')] \end{aligned}$$

$$s.t. \quad q_\pi(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_\pi(s')$$

---

# Ch3. 贝尔曼最优公式

## Optimal policy

如果对于所有的 $s$ 和其他策略 $\pi$，都有 $v_{\pi^*} \geq v_\pi(s)$，则策略 $\pi^*$ 是最优的

## Bellman optimality equation (BOE)

**表示方式**

- elementwise form

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

- matrix-vector form

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

**求解（值迭代）**

- 先求解最优策略（固定 $v$ 求 $\pi$ ）

因为 $\sum_a \pi(a|s) = 1$，所以 $v$ 的最大值就是取最大的 $q(s,a)$，然后令 $\pi(s,a) = 1$ 即可

$$
\begin{aligned}
v &= \max_\pi \sum_a \pi(a|s) q(s,a) \\
&= \max_{a \in A(s)} q(s,a)
\end{aligned}
$$

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

- 再求解最优价值（已知最优策略 $\pi$ 和 $P_\pi$ ）

$v = f(v) = \max_\pi (r_\pi + \gamma P_\pi v)$ 是一个 **contraction mapping**，存在最优解且唯一，因此迭代求解最终便可收敛

$$
\begin{aligned}
v_{k+1} &= f(v_k) = \max_\pi (r_\pi + \gamma P_\pi v_k) \\
&\text{给定初始值} v_0, \{v_k\} \text{最终收敛至} v^*
\end{aligned}
$$

**性质**

- 实质：贝尔曼最优是策略取 $\pi^*$ 时的一种特殊贝尔曼
- 存在性：最优 $v$ 一定存在
- 唯一性：最优 $v$ 一定唯一，$\pi^*$ 不一定
- 相对性：$r \to ar + b$，得到的最优策略是一样的
- 无意义 detour：如果两个普通区域的移动 reward 为 0，理论上讲从其中一个区域直接到 target 和从这个普通区域绕一大圈再进 target 这两种策略是一样，但因为 $\gamma$ 的存在，最优策略不会是无意义的 detour

# Ch4.值迭代和策略迭代

## Value iteration

给定初始价值 $v_k$

- Step 1: policy update（给定 $v_k$ ）

$$\text{矩阵表示} \quad \pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_k)$$
$$\text{求解使用 } element - wise\ form$$

因为已知 $v_k$，可以穷举每个 $s$ 的策略 $A(s)$ 求得 $\max\ q_k(a,s)$，$\pi_{k+1}$ 是一种贪婪策略

$$\pi_{k+1}(s) = \arg\max_{\pi} \sum_a \pi(a|s)\underbrace{[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v(s')]}_{q_k(s,a)}, \quad \forall s \in S$$

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

- Step 2: value update（给定 $P_{\pi\,k+1}$ ）

$$\text{矩阵表示} \quad v_{k+1} = r_{\pi\,k+1} + \gamma P_{\pi\,k+1} v_k$$
$$\text{求解使用 } element - wise\ form$$

因为已知 $\pi_{k+1}$，所以 $v_{k+1}(s)$ 即是最大的 $q_k(s,a)$

$$v_{k+1}(s) = \sum_a \pi_{k+1}(a|s)\underbrace{[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v(s')]}_{q_k(s,a)}, \quad \forall s \in S$$
$$= \max_a q_k(s,a)$$

**Pseudocode: Value iteration algorithm**

**Initialization:** The probability model $p(r|s,a)$ and $p(s'|s,a)$ for all $(s,a)$ are known. Initial guess $v_0$.

**Aim:** Search the optimal state value and an optimal policy solving the Bellman optimality equation.

While $v_k$ has not converged in the sense that $\|v_k - v_{k-1}\|$ is greater than a predefined small threshold, for the $k$th iteration, do

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

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

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

Maximum action value: $a_k^*(s) = \arg\max_a q_k(a,s)$

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

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

```
import numpy as np


def calculate_q(n, s, a, v, r, action_choice, gamma, forbidden, target):
    """
    计算 q 值
    """
```

```
    now_i, now_j = s // n, s % n
    offset = action_choice[a]
    next_i, next_j = now_i + offset[0], now_j + offset[1]

    # 检查是否越界
    if next_i < 0 or next_i >= n or next_j < 0 or next_j >= n:
        return r['boundary'] + gamma * v[s]

    next_s = next_i * n + next_j
    if next_s in forbidden:
        return r['forbidden'] + gamma * v[next_s]
    if next_s == target:
        return r['target'] + gamma * v[next_s]

    return r['otherwise'] + gamma * v[next_s]


def value_iteration(n, v, pi, reward, action_choice, forbidden, target, gamma=0.9,
n_iter=10000, epsilon=1e-6):
    iteration = 0
    v_old = np.ones_like(v)

    while iteration < n_iter and np.max(np.abs(v - v_old)) > epsilon:
        v_old = v.copy()
        # for each state
        for s in range(n**2):
            # for each action
            q_final, a_final = -np.inf, -1
            for a in range(5):
                q = calculate_q(n, s, a, v, reward, action_choice, gamma, forbidden, target)
                if q > q_final:
                    q_final = q
                    a_final = a
            # policy update
            pi[s] = a_final
            # value update
            v[s] = q_final
        iteration += 1

    return v, pi, iteration


def print_policy(pi, n, action_str):
    res = np.zeros((n, n), dtype=str)
    for i in range(n):
        for j in range(n):
            s = i * n + j
            action = pi[s]
            res[i, j] = action_str[action]
    return res


def print_state_value(v, n):
    res = np.zeros((n, n), dtype=float)
```

```
    for i in range(n):
        for j in range(n):
            s = i * n + j
            res[i, j] = v[s]
    return res


def main():
    reward = {
        'boundary': -1,
        'target': 1,
        'forbidden': -1,
        'otherwise': 0
    }
    action_choice = {
        0: (0, 1),   # right
        1: (-1, 0),   # up
        2: (0, -1),  # left
        3: (1, 0),  # down
        4: (0, 0)    # stay
    }
    action_str = {
        0: '→',   # right
        1: '↑',   # up
        2: '←',   # left
        3: '↓',   # down
        4: '◦'  # stay
    }
    gamma = 0.9

    # n = 2
    # forbidden = [1]
    # target = 3
    # n_states = n * n

    n = 5
    forbidden = [6, 7, 12, 16, 18, 21]
    target = 17
    n_states = n * n

    v = np.zeros(n_states)
    pi = np.zeros(n_states, dtype=int)
    v, pi, iteration = value_iteration(n, v, pi, reward, action_choice, forbidden, target,
gamma=gamma)
    print(f"Value Iteration completed in {iteration} iterations.")
    print("Optimal Policy:")
    print(print_policy(pi, n, action_str))
    print("State Values:")
    print(print_state_value(v, n))


if __name__ == "__main__":
    main()
```

# Policy iteration

给定随机初始策略 $\pi_k$

- Step 1: policy evaluation (PE)

已知 $\pi_k$ 求解 $v_{\pi k}$，具体求解参考 iterative solution

$$v_{\pi k} = r_{\pi k} + \gamma P_\pi v_{\pi k}$$

越靠近 target 的区域 value 变好的越快，近水楼台先得月，因为 $v$ 的更新严重依赖 $\pi$ 的好坏

- Step 2: policy improvement (PI)

最优化求解更新策略 $\pi_{k+1}$，具体求解参考 value evaluation step 1

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

**Pseudocode: Policy iteration algorithm**

**Initialization:** The probability model $p(r|s,a)$ and $p(s'|s,a)$ for all $(s,a)$ are known. Initial guess $\pi_0$.

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

While $v_{\pi_k}$ has not converged, for the $k$th iteration, do

*Policy evaluation:*

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

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

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

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

*Policy improvement:*

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

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

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

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

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

```python
import numpy as np


def calculate_q(n, s, a, v, r, action_choice, gamma, forbidden, target):
    """
    计算 q 值
    """
    now_i, now_j = s // n, s % n
    offset = action_choice[a]
    next_i, next_j = now_i + offset[0], now_j + offset[1]

    # 检查是否越界
    if next_i < 0 or next_i >= n or next_j < 0 or next_j >= n:
```

```
        return r['boundary'] + gamma * v[s]

    next_s = next_i * n + next_j
    if next_s in forbidden:
        return r['forbidden'] + gamma * v[next_s]
    if next_s == target:
        return r['target'] + gamma * v[next_s]

    return r['otherwise'] + gamma * v[next_s]


def pi_to_P_r(n, pi, action_choice, reward, forbidden, target):
    """
    将策略转换为转移概率矩阵
    """
    n_states = n * n
    P = np.zeros((n_states, n_states))
    r = np.zeros(n_states)

    for s in range(n_states):
        action = pi[s]
        offset = action_choice[action]
        next_i, next_j = s // n + offset[0], s % n + offset[1]

        if 0 <= next_i < n and 0 <= next_j < n:
            next_s = next_i * n + next_j
            P[s, next_s] = 1.0
            if next_s in forbidden:
                r[s] = reward['forbidden']
            elif next_s == target:
                r[s] = reward['target']
            else:
                r[s] = reward['otherwise']
        else:
            P[s, s] = 1.0
            r[s] = reward['boundary']

    return P, r


def iteration_solution(P, r, gamma=0.9, n_iter=100, epsilon=1e-3):
    v = np.zeros_like(r)
    v_old = np.ones_like(v)
    iteration = 0
    while iteration < n_iter and np.max(np.abs(v - v_old)) > epsilon:
        v_old = v.copy()
        v = r + gamma * P @ v
        iteration += 1
    return v, iteration


def policy_iteration(n, v, pi, reward, action_choice, forbidden, target, gamma=0.9,
n_iter=10000, epsilon=1e-6):
    iteration, inner_iteration = 0, 0
```

```
    n_states = n * n
    v_old = np.ones_like(v)

    while iteration < n_iter and np.max(np.abs(v - v_old)) > epsilon:
        v_old = v.copy()

        # policy evaluation
        P, r = pi_to_P_r(n, pi, action_choice, reward, forbidden, target)
        v, ii = iteration_solution(P, r, gamma=gamma, n_iter=100)
        inner_iteration += ii

        # policy improvement
        for s in range(n_states):
            q_final, a_final = -np.inf, -1
            for a in range(len(action_choice)):
                q = calculate_q(n, s, a, v, reward, action_choice, gamma, forbidden, target)
                if q > q_final:
                    q_final = q
                    a_final = a
            pi[s] = a_final

        iteration += 1

    return v, pi, iteration, inner_iteration


def print_policy(pi, n, action_str):
    res = np.zeros((n, n), dtype=str)
    for i in range(n):
        for j in range(n):
            s = i * n + j
            action = pi[s]
            res[i, j] = action_str[action]
    return res


def print_state_value(v, n):
    res = np.zeros((n, n), dtype=float)
    for i in range(n):
        for j in range(n):
            s = i * n + j
            res[i, j] = v[s]
    return res


def main():
    reward = {
        'boundary': -1,
        'target': 1,
        'forbidden': -1,
        'otherwise': 0
    }
    action_choice = {
        0: (0, 1),   # right
```

```
        1: (-1, 0),   # up
        2: (0, -1),  # left
        3: (1, 0),  # down
        4: (0, 0)    # stay
    }
    action_str = {
        0: '→',   # right
        1: '↑',   # up
        2: '←',   # left
        3: '↓',   # down
        4: '◦'  # stay
    }
    gamma = 0.9

    # n = 2
    # forbidden = [1]
    # target = 3
    # n_states = n * n

    n = 5
    forbidden = [6, 7, 12, 16, 18, 21]
    target = 17
    n_states = n * n

    v = np.zeros(n_states)
    pi = np.zeros(n_states, dtype=int)
    v, pi, iteration, inner_iteration = policy_iteration(n, v, pi, reward, action_choice,
forbidden, target, gamma=gamma)
    print(f"Policy Iteration completed in {iteration} outer_iterations and {inner_iteration}
inner_iteration.")
    print("Optimal Policy:")
    print(print_policy(pi, n, action_str))
    print("State Values:")
    print(print_state_value(v, n))


if __name__ == "__main__":
    main()
```

## Truncated Policy iteration

- *value iteration* 和 *policy iteration* 的步骤基本相反
- 可以把 *value iteration* 的 *value update* 看作一次效果不是很好的 *policy evaluation*
- $v_{\pi 1} \geq u_1$

▷ The two algorithms are very similar:

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

Value iteration: $u_0 \xrightarrow{PU} \pi_1' \xrightarrow{VU} u_1 \xrightarrow{PU} \pi_2' \xrightarrow{VU} u_2 \xrightarrow{PU} \dots$

PE=policy evaluation. PI=policy improvement.

PU=policy update. VU=value update.

Consider the step of solving $v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$:

$$v_{\pi_1}^{(0)} = v_0$$

$$\text{value iteration} \leftarrow v_1 \longleftarrow v_{\pi_1}^{(1)} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(0)}$$

$$v_{\pi_1}^{(2)} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(1)}$$

$$\vdots$$

$$\text{truncated policy iteration} \leftarrow \bar{v}_1 \longleftarrow v_{\pi_1}^{(j)} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(j-1)}$$

$$\vdots$$

$$\text{policy iteration} \leftarrow v_{\pi_1} \longleftarrow v_{\pi_1}^{(\infty)} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(\infty)}$$

**Pseudocode: Truncated policy iteration algorithm**

**Initialization:** The probability model $p(r|s,a)$ and $p(s'|s,a)$ for all $(s,a)$ are known. Initial guess $\pi_0$.

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

While $v_k$ has not converged, for the $k$th iteration, do

*Policy evaluation:*

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

While $j < j_{\text{truncate}}$, do 设定一个迭代最大次数比如100，实际前边的

For every state $s \in \mathcal{S}$, do 代码实现就已经是 truncated policy iteration

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

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

*Policy improvement:*

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

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

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

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

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

# Ch5.Monte Carlo Learning

## MC Basic

**核心：把 policy iteration 的 model-based 转为 model-free。而 model-based 的核心在于求** $q_{\pi k}(s,a)$（数据和模型必有其一）

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

$$\downarrow$$

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

**Pseudocode: MC Basic algorithm (a model-free variant of policy iteration)**

**Initialization:** Initial guess $\pi_0$.

**Aim:** Search for an optimal policy.

区别于 policy iteration（PE求v，PI将v转为q再求pi），MC Basic直接在PE阶段穷举出q，然后接PI

For the $k$th iteration ($k = 0, 1, 2, \ldots$), 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 $\pi_k$

      *Policy evaluation:*

      $q_{\pi_k}(s,a) \approx q_k(s,a)$ = average return of all the episodes starting from $(s,a)$

    *Policy improvement:*

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

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

## MC Exploring Starts

- first-visit and every-visit
  - first-visit：对于初始 $(s_0,a_0)$，仅使用此 episode 的 $q(s_0,a_0)$ 估计它
  - every-visit：对于初始 $(s_0,a_0)$，使用零时刻的 $q_{t_0}(s_0,a_0)$ 估计它，且如果 $(s_0,a_0)$ 重新出现在此 episode 之后的某个位置，也可以使用 $q_{t_n}(s_0,a_0)$ 估计它
- MC basic and MC Exploring Starts
  - MC basic：针对每一个 $(s,a)$ 求 $q(s,a)$，相当于要把 $(s,a)$ 对应的所有 episodes 全部穷举
  - MC Exploring Starts：从每个 $(s,a)$ 出发生成一些 episodes，并尽量确保所有的 $(s,a)$ 都包含在其中。对每一条 episode，从后向前依次更新包含 $(s_t,a_t)$ 的 $r(s_t,a_t)$ 和 $q(s_t,a_t)$，并以 $a = \arg\max_a q(s_t,a)$ 更新当前 $s_t$ 的策略 $\pi_(a|s_t) = 1$

（下图代码属于 every-visit ）

**Algorithm: MC Exploring Starts (an efficient variant of MC Basic)**

**Initialization:** Initial policy $\pi_0(a|s)$ and initial value $q(s,a)$ for all $(s,a)$. Returns$(s,a) = 0$ and Num$(s,a) = 0$ for all $(s,a)$.
**Goal:** Search for an optimal policy.

For each episode, do

*Episode generation:* Select a starting state-action pair $(s_0, a_0)$ and ensure that all pairs can be possibly selected (this is the exploring-starts condition). Following the current policy, generate an episode of length $T$: $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$.
Initialization for each episode: $g \leftarrow 0$
For each step of the episode, $t = T-1, T-2, \ldots, 0$, do

$g \leftarrow \gamma g + r_{t+1}$
Returns$(s_t, a_t) \leftarrow$ Returns$(s_t, a_t) + g$
Num$(s_t, a_t) \leftarrow$ Num$(s_t, a_t) + 1$
*Policy evaluation:*
$q(s_t, a_t) \leftarrow$ Returns$(s_t, a_t)$/Num$(s_t, a_t)$
*Policy improvement:*
$\pi(a|s_t) = 1$ if $a = \arg\max_a q(s_t, a)$ and $\pi(a|s_t) = 0$ otherwise

# MC $\epsilon$-Greedy

- $\epsilon \to 1$: exploration，更探索，其他策略概率更高，更大概率需更少的 episodes 即包含所有的 $(s,a)$
- $\epsilon \to 0$: expliotation，更剥削利用，$q(s,a)$ 更大的策略概率高，$= 0$ 退化为 greedy

$$\pi_{k+1}(s) = \arg\max_{\pi \in \Pi_\varepsilon} \sum_a \pi(a|s) q_{\pi_k}(s,a)$$

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

（下图代码属于 every-visit ）

**Algorithm: MC $\epsilon$-Greedy (a variant of MC Exploring Starts)**

**Initialization:** Initial policy $\pi_0(a|s)$ and initial value $q(s,a)$ for all $(s,a)$. Returns$(s,a) = 0$ and Num$(s,a) = 0$ for all $(s,a)$. $\epsilon \in (0,1]$
**Goal:** Search for an optimal policy.

For each episode, do
*Episode generation:* Select a starting state-action pair $(s_0, a_0)$ (the exploring starts condition is not required). Following the current policy, generate an episode of length $T$: $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$.
Initialization for each episode: $g \leftarrow 0$
For each step of the episode, $t = T-1, T-2, \ldots, 0$, do
$g \leftarrow \gamma g + r_{t+1}$
Returns$(s_t, a_t) \leftarrow$ Returns$(s_t, a_t) + g$
Num$(s_t, a_t) \leftarrow$ Num$(s_t, a_t) + 1$
*Policy evaluation:*
$q(s_t, a_t) \leftarrow$ Returns$(s_t, a_t)$/Num$(s_t, a_t)$
*Policy improvement:*
Let $a^* = \arg\max_a q(s_t, a)$ and

$$\pi(a|s_t) = \begin{cases} 1 - \frac{|\mathcal{A}(s_t)|-1}{|\mathcal{A}(s_t)|}\epsilon, & a = a^* \\ \frac{1}{|\mathcal{A}(s_t)|}\epsilon, & a \neq a^* \end{cases}$$

# CH6. 随机近似和随机梯度下降

## Mean estimation

随机近似算法每次求 $E(x)$，都要用到之前的所有 $x$，为简化效率，引入 $w_k$

$$w_{k+1} = \frac{1}{k}\sum_{i=1}^{k} x_i \qquad k = 1, 2, \ldots$$

$$w_{k+1} = w_k - \frac{1}{k}(w_k - x_k)$$

## Robbin-Monro algorithm (RM)

目标：求解 $g(w) = 0$，但不知道 $g(w)$ 的形式，只能根据 $w$ 拿到输出 $g(w)$，假设最优解为 $w^*$

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

- $w_k$ 是第 $k$ 次估计
- $\tilde{g}(w_k, \eta_k) = g(w_k) + \eta_k$ 是第 $k$ 次观测，$\eta_k$ 是白噪声
- $a_k$ 是正系数，一般取 $a_k = 1/k$，但实际可能会选一个很小的正数，保证迭代次数很大时的新数据也有作用
- 没有模型，仅依赖数据

**Theorem (Robbins-Monro Theorem)**

*In the Robbins-Monro algorithm, if*

1) $0 < c_1 \leq \nabla_w g(w) \leq c_2$ *for all* $w$;
2) $\sum_{k=1}^{\infty} a_k = \infty$ *and* $\sum_{k=1}^{\infty} a_k^2 < \infty$;
3) $\mathbb{E}[\eta_k | \mathcal{H}_k] = 0$ *and* $\mathbb{E}[\eta_k^2 | \mathcal{H}_k] < \infty$;

*where* $\mathcal{H}_k = \{w_k, w_{k-1}, \dots\}$, *then* $w_k$ *converges with probability 1 (w.p.1) to the root* $w^*$ *satisfying* $g(w^*) = 0$.

g(w)梯度为正且要小于一个常数（一般对应梯度函数的凸函数性质）

ak最终收敛到0但又不要收敛太快

## Stochastic gradient descent algorithm (SGD)

优化问题如下，优化参数 $w$ 以求 $J$ 的最小值

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

- 方法1：GD，缺点是需要计算 $X$ 的期望

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

- 方法2：BGD，缺点是每次迭代需要很多样本计算 $\frac{1}{n}\sum_{i=1}^{n} \nabla_w f(w_k, x_i)$

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

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

- 方法3：SGD，每次迭代仅需随机一个值更新（不排序，随机抽取即可）
  - $w_k$ 远离 $w^*$ 时，SGD 与 GD 别无二异
  - $w_k$ 接近 $w^*$ 时，SGD 在更新 $w_k$ 时会在 $w^*$ 附近横跳，是一个比较好的性质

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

**Theorem (Convergence of SGD)**

*In the SGD algorithm, if*

1) $0 < c_1 \leq \nabla_w^2 f(w, X) \leq c_2$;
2) $\sum_{k=1}^{\infty} a_k = \infty$ *and* $\sum_{k=1}^{\infty} a_k^2 < \infty$;
3) $\{x_k\}_{k=1}^{\infty}$ *is iid;*

*then* $w_k$ *converges to the root of* $\nabla_w \mathbb{E}[f(w, X)] = 0$ *with probability 1.*

- MBGD
  - $m = 1$，MBGD = SGD
  - $m = n$，MBGD 和 BGD 严格意义不相同，因为 MBGD 是在 $n$ 个样本有放回抽样 $m$ 次，可能重复

# Ch7.Temporal-difference learning

## TD learning of state value

$(s_0, r_1, s_1, \ldots, s_t, r_{t+1}, s_{t+1}, \ldots)$ 是由策略 $\pi$ 生成的，目标是估计 $\{v_\pi(s)_{s\in S}\}$，TD learning 算法如下

$$
\begin{aligned}
v_{t+1}(s_t) &= v_t(s_t) - \alpha_t(s_t)\left[v_t(s_t) - [r_{t+1} + \gamma v_t(s_{t+1})]\right] \\
v_{t+1}(s) &= v_t(s), \quad \forall s \neq s_t
\end{aligned}
$$

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

算法实际上是让 $v(s_t)$ 朝着 $\overline{v_t} = [r_{t+1} + \gamma v_t(s_{t+1})]$ 渐进

$$
\begin{aligned}
& v_{t+1}(s_t) = v_t(s_t) - \alpha_t(s_t)[v_t(s_t) - \bar{v}_t] \\
\Longrightarrow\ & v_{t+1}(s_t) - \bar{v}_t = v_t(s_t) - \bar{v}_t - \alpha_t(s_t)[v_t(s_t) - \bar{v}_t] \\
\Longrightarrow\ & v_{t+1}(s_t) - \bar{v}_t = [1 - \alpha_t(s_t)][v_t(s_t) - \bar{v}_t] \\
\Longrightarrow\ & |v_{t+1}(s_t) - \bar{v}_t| = |1 - \alpha_t(s_t)||v_t(s_t) - \bar{v}_t| \\
& 0 < 1 - \alpha_t(s_t) < 1 \\
& |v_{t+1}(s_t) - \bar{v}_t| \leq |v_t(s_t) - \bar{v}_t|
\end{aligned}
$$

TD 算法只能用来估计给定策略的 state value，不能估计 action value，也不能找到最优策略

| TD/Sarsa learning | MC learning |
| --- | --- |
| **Online:** TD learning is online. It can update the state/action values immediately after receiving a reward. | episode都生成完才开始计算<br>**Offline:** MC learning is offline. It has to wait until an episode has been completely collected. |
| **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 has 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 than MC because there are fewer random variables. For instance, Sarsa requires $R_{t+1}, S_{t+1}, A_{t+1}$. 随机变量少、估计方差低 | **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 $\lvert\mathcal{A}\rvert^L$ possible episodes. |

## Sarsa（求解贝尔曼方程）

$\{(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})\}_t$，目标是估计 $\{q_\pi(s,a)_{s\in S}\}$，Sarsa 算法如下

$$
\begin{aligned}
q_{t+1}(s_t, a_t) &= q_t(s_t, a_t) - \alpha_t(s_t, a_t)\left[q_t(s_t, a_t) - [r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})]\right] \\
q_{t+1}(s, a) &= q_t(s, a), \quad \forall (s, a) \neq (s_t, a_t)
\end{aligned}
$$

以上仅是估计 action value，即 policy evaluation，结合 policy improvement 代码如下

**Pseudocode: Policy searching by Sarsa**

For each episode, do

Generate $a_0$ at $s_0$ following $\pi_0(s_0)$

If $s_t$ $(t = 0, 1, 2, \dots)$ is not the target state, do

Collect an experience sample $(r_{t+1}, s_{t+1}, a_{t+1})$ given $(s_t, a_t)$: generate $r_{t+1}, s_{t+1}$ by interacting with the environment; generate $a_{t+1}$ following $\pi_t(s_{t+1})$.

*Update q-value for* $(s_t, a_t)$*:*

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

*Update policy for* $s_t$*:*

$\pi_{t+1}(a|s_t) = 1 - \frac{\epsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)| - 1)$ if $a = \arg\max_a q_{t+1}(s_t, a)$

$\pi_{t+1}(a|s_t) = \frac{\epsilon}{|\mathcal{A}(s_t)|}$ otherwise

$s_t \leftarrow s_{t+1}$, $a_t \leftarrow a_{t+1}$

## n-step Sarsa

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

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

$\{(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \dots, r_{t+n}, s_{t+n}, a_{t+n})\}_t$，目标是估计 $\{q_\pi(s,a)_{s\in S}\}$，n-step Sarsa 算法如下（实际到了 $t+n$ 得到数据才能更新 $t$ 时刻的 $q_{t+1}(s_t, a_t)$ ）

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\left[q_t(s_t, a_t) - [r_{t+1} + r_{t+2} + \cdots + \gamma^n q_t(s_{t+n}, a_{t+n})]\right]$$

## Q-learning（求解贝尔曼最优方程）

$\{(s_t, a_t, r_{t+1}, s_{t+1})\}_t$

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

- behavior policy: 用来生成 experience samples 的策略
- target policy: 更新最优策略
  - on-policy: behavior = target
    - Sarsa: 用 $\pi_t$ 生成 experiences，计算 $q_{\pi_t}(s_t, a_t)$，反过来更新 $\pi_t \to \pi_{t+1}$
    - MC: 用 $\pi$ 生成 experiences，计算 $q_\pi(s_t, a_t)$，反过来更新 $\pi \to \pi$
  - off-policy: behavior $\neq$ target
    - Q-learning: 因为仅需要 $s_t, a_t, r_{t+1}, s_{t+1}$，当给定 $(s_t, a_t)$，$r_{t+1}$ 和 $s_{t+1}$ 是依靠采样获得的，不需要依赖于任何策略，而且也不需要 $a_{t+1}$。**直观解释是，寻找的最优策略 $\pi^*$ 在更新后不会影响后续的 experience 采样**

## Pseudocode: Policy searching by Q-learning (on-policy version)

For each episode, do

If $s_t$ $(t = 0, 1, 2, \dots)$ is not the target state, do

Collect the experience sample $(a_t, r_{t+1}, s_{t+1})$ given $s_t$: generate $a_t$ following $\pi_t(s_t)$; generate $r_{t+1}, s_{t+1}$ by interacting with the environment.

*Update q-value for $(s_t, a_t)$:*

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

*Update policy for $s_t$:*

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

**Pseudocode: Optimal policy search by Q-learning (off-policy version)**

**Goal:** Learn an optimal target policy $\pi_T$ for all states from the experience samples generated by $\pi_b$.

For each episode $\{s_0, a_0, r_1, s_1, a_1, r_2, \dots\}$ generated by $\pi_b$, do

For each step $t = 0, 1, 2, \dots$ of the episode, do

*Update q-value for $(s_t, a_t)$:*

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

*Update target policy for $s_t$:*

$$\pi_{T,t+1}(a|s_t) = 1 \text{ if } a = \arg\max_a q_{t+1}(s_t, a)$$

$$\pi_{T,t+1}(a|s_t) = 0 \text{ otherwise}$$

# Ch8.Value Function Approximation

将表格存储的 $q(s,a)$ 和 $v(s)$ 转化为函数表示，这样可以节省存储空间，只需要存放函数的参数即可。比如使用二次函数拟合

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

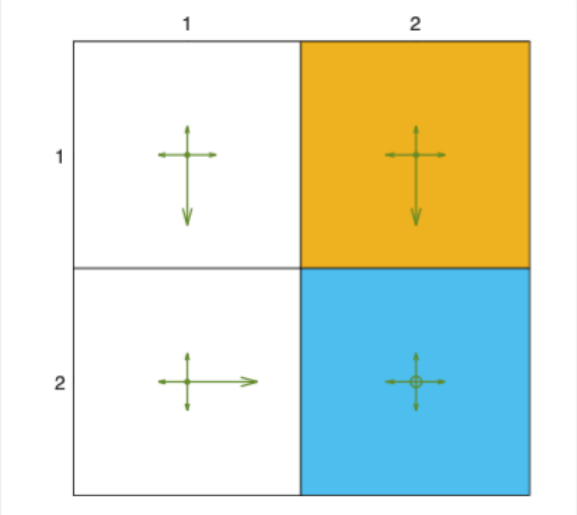
泛化能力强，因为之前更新 $v(s)$ 只更新一个具体 $s_n$ 的值，现在更新函数的话，会导致整个周围的 $s_{n-1}$，$s_{n+1}$ 等的 value 也会被更新

设定目标函数

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

$d_\pi(s) \geq 0$ and $\sum_{s \in S} d_\pi(s) = 1$：使用一种策略生成很多的 experience，达到一种平稳状态（ stationary distribution ），生成一个 $s$ 的概率分布。有 $P_\pi$ 时，$d_\pi$ 迭代求解公式为 $d_\pi^T = d_\pi^T P_\pi$

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



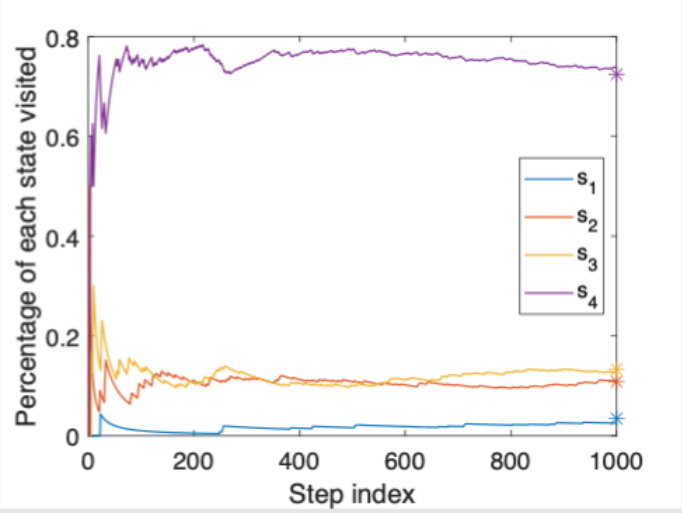




Figure: Long-run behavior of an $\epsilon$-greedy policy with $\epsilon = 0.5$.

## 优化方法

$$\begin{aligned}
w_{k+1} &= w_k - \alpha_k \nabla_w J(w_k) \\
\nabla_w J(w) &= \nabla_w \mathbb{E}[(v_\pi(S) - \hat{v}(S,w))^2] \\
&= \mathbb{E}[\nabla_w (v_\pi(S) - \hat{v}(S,w))^2] \\
&= 2\mathbb{E}[(v_\pi(S) - \hat{v}(S,w))(-\nabla_w \hat{v}(S,w))] \\
&= -2\mathbb{E}[(v_\pi(S) - \hat{v}(S,w))\nabla_w \hat{v}(S,w)]
\end{aligned}$$

$$\begin{aligned}
w_{k+1} &= w_k + \alpha_k \mathbb{E}[(v_\pi(S) - \hat{v}(S,w))\nabla_w \hat{v}(S,w)] \\
&\Downarrow \\
w_{t+1} &= w_t + \alpha_t (v_\pi(s_t) - \hat{v}(s_t,w_t))\nabla_w \hat{v}(s_t,w_t) \quad 2\alpha_t \to \alpha_t
\end{aligned}$$

此算法无法实施，因为需要获取真实的 $v_\pi$，因此使用近似算法估计 $v_\pi$

- Monte Carlo: 使用 $g_t$ (以 $s_t$ 为起点的 episode 的 discounted return) 估计 $v_\pi(s_t)$，
  $w_{t+1} = w_t + \alpha_t(g_t - \hat{v}(s_t,w_t))\nabla_w \hat{v}(s_t,w_t)$
- TD learning: 使用 $r_{t+1} + \gamma \hat{v}(s_{t+1},w_t)$ 估计 $v_\pi(s_t)$，
  $w_{t+1} = w_t + \alpha_t(r_{t+1} + \gamma \hat{v}(s_{t+1},w_t) - \hat{v}(s_t,w_t))\nabla_w \hat{v}(s_t,w_t)$

**Pseudocode: TD learning of state values with function approximation**

**Initialization:** A function $\hat{v}(s,w)$ that is differentiable in $w$. Initial parameter $w_0$.
**Goal:** Learn the true state values of a given policy $\pi$.

For each episode $\{(s_t, r_{t+1}, s_{t+1})\}_t$ generated by $\pi$, do
  For each sample $(s_t, r_{t+1}, s_{t+1})$, 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,
    $w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \phi^T(s_{t+1}) w_t - \phi^T(s_t) w_t\right] \phi(s_t)$

# Sarsa with function approximation

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

**Pseudocode: Sarsa with function approximation**

**Initialization:** Initial parameter $w_0$. Initial policy $\pi_0$. $\alpha_t = \alpha > 0$ for all $t$. $\epsilon \in (0,1)$.
**Goal:** Learn an optimal policy to lead the agent to the target state from an initial state $s_0$.
For each episode, do
  Generate $a_0$ at $s_0$ following $\pi_0(s_0)$
  If $s_t$ $(t = 0, 1, 2, \dots)$ is not the target state, do
    Collect the experience sample $(r_{t+1}, s_{t+1}, a_{t+1})$ given $(s_t, a_t)$: generate $r_{t+1}, s_{t+1}$ by interacting with the environment; generate $a_{t+1}$ following $\pi_t(s_{t+1})$.
    Update q-value (update parameter):
    $w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t)\right] \nabla_w \hat{q}(s_t, a_t, w_t)$
    Update policy:
    $\pi_{t+1}(a|s_t) = 1 - \frac{\varepsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)| - 1)$ if $a = \arg\max_{a \in \mathcal{A}(s_t)} \hat{q}(s_t, a, w_{t+1})$
    $\pi_{t+1}(a|s_t) = \frac{\epsilon}{|\mathcal{A}(s_t)|}$ otherwise
    $s_t \leftarrow s_{t+1}$, $a_t \leftarrow a_{t+1}$

# Q-learning with function approximation

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

**Pseudocode: Q-learning with function approximation (on-policy version)**

**Initialization:** Initial parameter $w_0$. Initial policy $\pi_0$. $\alpha_t = \alpha > 0$ for all $t$. $\epsilon \in (0, 1)$.
**Goal:** Learn an optimal path to lead the agent to the target state from an initial state $s_0$.

For each episode, do

If $s_t$ $(t = 0, 1, 2, \dots)$ is not the target state, do

Collect the experience sample $(a_t, r_{t+1}, s_{t+1})$ given $s_t$: generate $a_t$ following $\pi_t(s_t)$; generate $r_{t+1}, s_{t+1}$ by interacting with the environment.

Update value (update parameter):

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

Update policy:

$$\pi_{t+1}(a|s_t) = 1 - \frac{\varepsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)| - 1) \text{ if } a = \arg\max_{a \in \mathcal{A}(s_t)} \hat{q}(s_t, a, w_{t+1})$$

$$\pi_{t+1}(a|s_t) = \frac{\varepsilon}{|\mathcal{A}(s_t)|} \text{ otherwise}$$

# Deep Q-learning (DQN)

**两个 network**

- main network: $\hat{q}(s, a, w)$
- target network: $\hat{q}(s, a, w_T)$

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

$w_T$ 一段时间内不更新，作为常量，之后用这段时间训练得到的新 $w$ 更新 $w_t$ 进行下阶段的训练

**Experience replay**

收集的 experiences 放到一起，并非按顺序送进网络，而是**均匀分布的抽取**，这样的好处是数据可以重复使用

$$replay\ buffer\ \mathcal{B} = \{(s, a, r, s')\}$$

**Tabular Q-learning vs DQN**

- Tabular Q-learning 有模型，在计算贝尔曼最优方程，不需要 $(s, a)$ 的这种均匀分布
- DQN 没有模型，需要计算 $\mathbb{E}$，需要有 $(s, a)$ 这种均匀分布

**Pseudocode（代码中没有显示使用上述公式，是因为现在神经网络可以自己计算梯度 backward ）**

**Pseudocode: Deep Q-learning (off-policy version)**

**Initialization:** A main network and a target network with the same initial parameter.
**Goal:** Learn an optimal target network to approximate the *optimal* action values from the experience samples generated by a given behavior policy $\pi_b$.

Store the experience samples generated by $\pi_b$ 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 $(s, a, r, s')$, calculate the target value as $y_T = r + \gamma \max_{a \in \mathcal{A}(s')} \hat{q}(s', a, w_T)$, where $w_T$ is the parameter of the target network
    Update the main network to minimize $(y_T - \hat{q}(s, a, w))^2$ using the mini-batch of samples
  Set $w_T = w$ every $C$ iterations

# Ch9.Policy Gradient Methods

## Metric

### average value

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

选择分布 $d$

- $d$ 和 $\pi$ 独立:
  - $d_0(s) = 1/|\mathbb{S}|$
  - $d_0(s_0) = 1, \quad d_0(s \neq s_0) = 0$, 特别关注 $s_0$ 时
- $d$ 和 $\pi$ 相关:
  - $d_\pi^T P_\pi = d_\pi^T$

### average reward

$$
\begin{aligned}
\bar{r}_\pi &= \sum_{s \in S} d_\pi(s) r_\pi(s) = \mathbb{E}[r_\pi(S)], \\
r_\pi(s) &= \sum_{a \in A} \pi(a|s) r(s, a) \\
r(s, a) &= \mathbb{E}[R|s, a] = \sum_r r p(r|s, a)
\end{aligned}
$$

论文常见形式

$$
\begin{aligned}
&\lim_{n\to\infty}\frac{1}{n}\mathbb{E}\left[R_1+R_2+\cdots+R_n\middle|S_0=s_0\right]\\
&=\lim_{n\to\infty}\frac{1}{n}\mathbb{E}\left[\sum_{t=0}^{n-1}R_{t+1}\middle|S_0=s_0\right]\\
&=\lim_{n\to\infty}\frac{1}{n}\mathbb{E}\left[\sum_{t=0}^{n-1}R_{t+1}\right]\quad\text{无穷步从哪开始已经不重要}\\
&=\sum_s d_\pi(s)r_\pi(s)\\
&=\bar{r}_\pi
\end{aligned}
$$

$\bar{r}_\pi$ **vs** $\bar{v}_\pi$

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

| Metric | Expression 1 | Expression 2 | Expression 3 |
|---|---|---|---|
| $\bar{v}_\pi$ | $\sum_{s\in\mathcal{S}} d(s)v_\pi(s)$ | $\mathbb{E}_{S\sim d}[v_\pi(S)]$ | $\lim_{n\to\infty}\mathbb{E}\left[\sum_{t=0}^{n}\gamma^t R_{t+1}\right]$ |
| $\bar{r}_\pi$ | $\sum_{s\in\mathcal{S}} d_\pi(s)r_\pi(s)$ | $\mathbb{E}_{S\sim d_\pi}[r_\pi(S)]$ | $\lim_{n\to\infty}\frac{1}{n}\mathbb{E}\left[\sum_{t=0}^{n-1} R_{t+1}\right]$ |

# Gradients of the Metrics

$J(\theta)$ 可以指 $\bar{r}_\pi$ $\bar{v}_\pi$ $\bar{v}_\pi^0$

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

$$\pi(a|s,\theta)=\frac{e^{h(s,a,\theta)}}{\sum_{a'\in A}e^{h(s,a',\theta)}}$$

# Gradient-ascent algorithm

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

- REINFORCE：使用 Monte Carlo 的长 episode 采样得到的 $q_t(s_t,a_t)$ 近似无法获得的 $q_\pi(s_t,a_t)$

- on-policy

**Pseudocode: Policy Gradient by Monte Carlo (REINFORCE)**

**Initialization:** Initial parameter $\theta$; $\gamma \in (0,1)$; $\alpha > 0$.
**Goal:** Learn an optimal policy to maximize $J(\theta)$.

For each episode, do
  Generate an episode $\{s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T\}$ following $\pi(\theta)$.
  For $t = 0, 1, \ldots, T-1$:
    *Value update:* $q_t(s_t, a_t) = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
    *Policy update:* $\theta \leftarrow \theta + \alpha \nabla_\theta \ln \pi(a_t|s_t, \theta) q_t(s_t, a_t)$

# Ch10.Actor-Critic Methods

## QAC

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

- Actor: policy update，整个式子就是在更新策略
- Critic: policy evaluation，进行策略的评估
  - REINFORCE: 使用 Monte Carlo，即 $q_t(s_t, a_t)$ 近似 $q_\pi(s_t, a_t)$
  - Actor-Critic: 使用 TD ( Sarsa + value function approximation )

**The simplest actor-critic algorithm (QAC)**

**Initialization:** A policy function $\pi(a|s, \theta_0)$ where $\theta_0$ is the initial parameter. A value function $q(s, a, w_0)$ where $w_0$ is the initial parameter. $\alpha_w, \alpha_\theta > 0$.
**Goal:** Learn an optimal policy to maximize $J(\theta)$.

At time step $t$ in each episode, do
  Generate $a_t$ following $\pi(a|s_t, \theta_t)$, observe $r_{t+1}, s_{t+1}$, and then generate $a_{t+1}$ following $\pi(a|s_{t+1}, \theta_t)$.
  Actor (policy update):
    $$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \ln \pi(a_t|s_t, \theta_t) q(s_t, a_t, w_t)$$
  Critic (value update):
    $$w_{t+1} = w_t + \alpha_w \left[ r_{t+1} + \gamma q(s_{t+1}, a_{t+1}, w_t) - q(s_t, a_t, w_t) \right] \nabla_w q(s_t, a_t, w_t)$$

# A2C

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

使用 $b(S)$ 来降低 variance

$$\begin{aligned}b^*(s) &= \frac{\mathbb{E}_{A\sim\pi}[\|\nabla_\theta \ln \pi(A|s,\theta_t)\|^2 q_\pi(s,A)]}{\mathbb{E}_{A\sim\pi}[\|\nabla_\theta \ln \pi(A|s,\theta_t)\|^2]} \\ b(s) &= \mathbb{E}_{A\sim\pi}[q_\pi(s,A)] = v_\pi(s) \quad \text{一般用此简单次形态}\end{aligned}$$

使用 advantage function $\delta_\pi(S,A)$ 简化方程

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

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

$$\text{greater } \delta_t(s_t,a_t) \Longrightarrow \text{greater } \beta_t \Longrightarrow \text{greater } \pi(a_t|s_t,\theta_{t+1})$$

$$\text{smaller } \pi(a_t|s_t,\theta_t) \Longrightarrow \text{greater } \beta_t \Longrightarrow \text{greater } \pi(a_t|s_t,\theta_{t+1})$$

**Advantage actor-critic (A2C) or TD actor-critic**

**Initialization:** A policy function $\pi(a|s,\theta_0)$ where $\theta_0$ is the initial parameter. A value function $v(s,w_0)$ where $w_0$ is the initial parameter. $\alpha_w, \alpha_\theta > 0$.
**Goal:** Learn an optimal policy to maximize $J(\theta)$.

At time step $t$ in each episode, do

Generate $a_t$ following $\pi(a|s_t,\theta_t)$ and then observe $r_{t+1}, s_{t+1}$.

Advantage (TD error):

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

Actor (policy update):

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

Critic (value update):

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

A2C: on-policy

# off-policy actor-critic

**Importance sampling**

给定 $x$ 的情况下可以得到 $p_0(x)$，但是得不到 $E_{X\sim p_0}(x)$，可以使用重要性采样，使用 $x_i \sim p_1$ 估计 $E_{X\sim p_0}(x)$

$$\begin{aligned}\mathbb{E}_{X\sim p_0}[X] = \sum_x p_0(x)x = \sum_x p_1(x)\underbrace{\frac{p_0(x)}{p_1(x)}}_{f(x)}x = \mathbb{E}_{X\sim p_1}[f(X)] \\ = \frac{1}{n}\sum_{i=1}^n f(x_i)\end{aligned}$$

- $\frac{p_0(x)}{p_1(x)}$: importance weight

假设 $\beta$ 是生成 experiences 的 behavior policy，目标是用这些样本更新 target policy $\pi(\theta)$ 下的 metric $J(\theta)$

$$\begin{aligned}J(\theta) &= \sum_{s\in S} d_\beta(s)v_\pi(s) = \mathbb{E}_{S\sim d_\beta}[v_\pi(S)] \\ \nabla_\theta J(\theta) &= \mathbb{E}_{S\sim\rho, A\sim\pi}\left[\nabla_\theta \ln \pi(A|S,\theta)q_\pi(S,A)\right] \\ &= \mathbb{E}_{S\sim\rho, A\sim\beta}\left[\frac{\pi(A|S,\theta)}{\beta(A|S)}\nabla_\theta \ln \pi(A|S,\theta)q_\pi(S,A)\right] \\ &= \mathbb{E}_{S\sim\rho, A\sim\beta}\left[\frac{\pi(A|S,\theta)}{\beta(A|S)}\nabla_\theta \ln \pi(A|S,\theta)(q_\pi(S,A)-b(S))\right] \quad b(S) = v_\pi(s)\end{aligned}$$

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

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

**Off-policy actor-critic based on importance sampling**

**Initialization:** A given behavior policy $\beta(a|s)$. A target policy $\pi(a|s,\theta_0)$ where $\theta_0$ is the initial parameter. A value function $v(s,w_0)$ where $w_0$ is the initial parameter. $\alpha_w, \alpha_\theta > 0$.
**Goal:** Learn an optimal policy to maximize $J(\theta)$.

At time step $t$ in each episode, do

Generate $a_t$ following $\beta(s_t)$ and then observe $r_{t+1}, s_{t+1}$.

Advantage (TD error):

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

Actor (policy update):

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

Critic (value update):

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

## Deterministic actor-critic (DPG)

之前的策略都是严格大于 0 且相加为 1 的，即 $\pi(a|,\theta) \in (0,1]$，仅适用于有限的 actions。DPG 适配无限策略

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

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

因为最后会用 $\mu(S)$ 替换 $a$，没用使用具体的策略 $a$，因此 behavior policy 可以任选，算法是 off-policy

## Deterministic policy gradient or deterministic actor-critic

**Initialization:** A given behavior policy $\beta(a|s)$. A deterministic target policy $\mu(s, \theta_0)$ where $\theta_0$ is the initial parameter. A value function $q(s, a, w_0)$ where $w_0$ is the initial parameter. $\alpha_w, \alpha_\theta > 0$.
**Goal:** Learn an optimal policy to maximize $J(\theta)$.

At time step $t$ in each episode, do

Generate $a_t$ following $\beta$ and then observe $r_{t+1}, s_{t+1}$.

TD error:

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

Actor (policy update):

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \mu(s_t, \theta_t) \big(\nabla_a q(s_t, a, w_t)\big)|_{a=\mu(s_t)}$$

Critic (value update):

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

- DPG: $q(s, a, w) = \phi^T(s, a)w$
- Deep DPG: 神经网络拟合 $q(s, a, w)$