马尔可夫奖励过程贝尔曼方程

马尔可夫奖励过程的预测（贝尔曼方程）

预测（prediction ）和控制（control ）

任务	输入	输出
预测	马尔可夫决策过程<S,A,P,R,γ> 、策略 π	每个状态的价值函数V
控制	马尔可夫决策过程<S,A,P,R,γ>	最佳策略 π、最佳价值函数V

马尔可夫奖励过程不涉及策略，因此只有预测问题没有控制问题

预测问题是给定一个马尔可夫奖励过程，我们要确定每个状态的价值是多少。

蒙特卡洛方法：从某一状态开始，采样生成很多轨迹，把这些轨迹的回报都计算出来，然后将其取平均值作为我们进入该状态的价值。（计算机数值模拟）

有没有一种方法能够像解一个公式一样解出每个状态的价值呢?

贝尔曼方程：（解析法）

贝尔曼方程（Bellman Equation）

$$V(s)=\underbrace{R(s)}_{\text{即时奖励}}+\underbrace{\gamma\sum_{s'\in S}p\left(s'\mid s\right)V\left(s'\right)}_{\text{未来奖励的折扣总和}}$$

贝尔曼方程的推导过程如下：

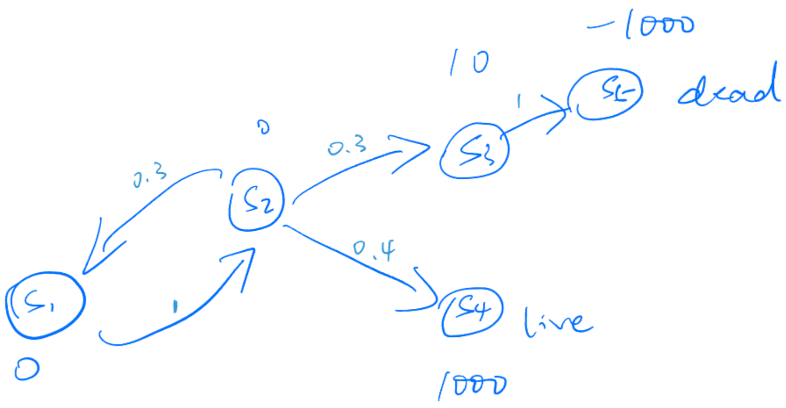
$$\begin{aligned}
V(s)&=\mathbb{E}\left[G_t\mid s_t=s\right]\\
&=\mathbb{E}\left[r_{t+1}+\gamma r_{t+2}+\gamma^2 r_{t+3}+\ldots\mid s_t=s\right]\\
&=\mathbb{E}\left[r_{t+1}|s_t=s\right]+\gamma\mathbb{E}\left[r_{t+2}+\gamma r_{t+3}+\gamma^2 r_{t+4}+\ldots\mid s_t=s\right]\\
&=R(s)+\gamma\mathbb{E}[G_{t+1}|s_t=s]\\
&=R(s)+\gamma\mathbb{E}[V(s_{t+1})|s_t=s]\\
&=R(s)+\gamma\sum_{s'\in S}p\left(s'\mid s\right)V\left(s'\right)
\end{aligned}$$

$$\mathbb{E}[V(s_{t+1})|s_t]=\mathbb{E}[\mathbb{E}[G_{t+1}|s_{t+1}]|s_t]=\mathbb{E}[G_{t+1}|s_t]$$

$$\begin{aligned}
\mathbb{E}\left[\mathbb{E}\left[G_{t+1}\mid s_{t+1}\right]\mid s_t\right]&=\mathbb{E}\left[\mathbb{E}\left[g'\mid s'\right]\mid s\right]\\
&=\mathbb{E}\left[\sum_{g'}g'\,p\left(g'\mid s'\right)\mid s\right]\\
&=\sum_{s'}\sum_{g'}g'p\left(g'\mid s',s\right)p\left(s'\mid s\right)\\
&=\sum_{s'}\sum_{g'}\frac{g'p\left(g'\mid s',s\right)p\left(s'\mid s\right)p(s)}{p(s)}\\
&=\sum_{s'}\sum_{g'}\frac{g'p\left(g'\mid s',s\right)p\left(s',s\right)}{p(s)}\\
&=\sum_{s'}\sum_{g'}\frac{g'p\left(g',s',s\right)}{p(s)}\\
&=\sum_{s'}\sum_{g'}g'p\left(g',s'\mid s\right)\\
&=\sum_{g'}\sum_{s'}g'p\left(g',s'\mid s\right)\\
&=\sum_{g'}g'p\left(g'\mid s\right)\\
&=\mathbb{E}\left[g'\mid s\right]=\mathbb{E}\left[G_{t+1}\mid s_t\right]
\end{aligned}$$

贝尔曼方程（Bellman Equation）手算

$$V(s) = \underbrace{R(s)}_{\text{即时奖励}} + \underbrace{\gamma \sum_{s' \in S} p(s' \mid s) V(s')}_{\text{未来奖励的折扣总和}}$$



$R = [0\ 0\ 10\ 1000\ -1000]$ $\gamma = 0.95$

init $V = [0, 0, 0, 0, 0]$

init $V = [0, 0, 0, 0, 0]$

round 1 $V(s_1) = R(s_1) + \gamma \sum_{s' \in S} p(s'|s) V(s')$

$= 0 + 0.95\,(p(s_2|s_1)V(s_2) + p(s_3|s_1)V(s_3) + \cdots + p(s_5|s_1)V(s_5))$ ($=0$, $=0$)

$= 0.95 \cdot 1 \cdot 0 = 0$ 迭代后$V(s_1)$

$V(s_2) = R(s_2) + 0.95\,(p(s_1|s_2)V(s_1) + p(s_3|s_2)V(s_3) + p(s_4|s_2)V(s_4))$

$= 0.95\,(0.3 \cdot 0 + 0.3 \cdot 0 + 0.4 \cdot 0) = 0$

$V(s_3) = R(s_3) + 0.95\,(p(s_5|s_3) V(s_5))$

$= 10 + 0.95 \times 1 \times 0$

$= 10.$

$V(s_4) = R(s_4) = 1000$

$V(s_5) = R(s_5) = -1000$

更新一轮后: $V = [0\ 0\ 10,\ 1000, -1000]$

round 2 $V(s_1) = R(s_1) + p(s_2|s_1)V(s_2) = 0$

$V(s_2) = R(s_2) + \gamma(P(s_3|s_2) V(s_3) + P(s_4|s_2)V(s_4) + P(s_1|s_2)V(s_1))$

$= 0 + (0.3 \times 10 + 0.3 \times 0 + 0.4 \times 1000) \times 0.95$

$= 382.85$

$V(s_3) = R(s_3) + p(s_4|s_3)V(s_4)\gamma$

$= 10 - 1000 \times \gamma$

$= -940$

$V(s_4) = -1000$ $V(s_5) = 1000$

更新第二轮后 $V = [0, 382.85, -940, -1000, 1000]$

Round 3 ...

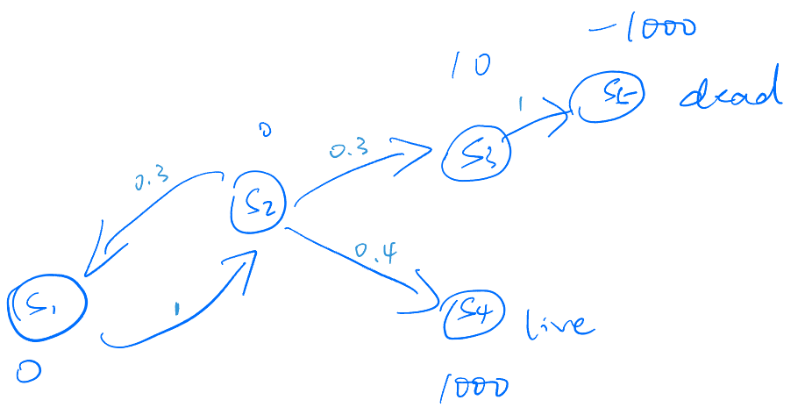
Round 4 ...

Round n (until $V(s) - V(s)_{update} < \epsilon$)

$\epsilon = 0.1/1 \cdots$

贝尔曼方程（Bellman Equation）程序模拟

$$V(s) = \underbrace{R(s)}_{\text{即时奖励}} + \underbrace{\gamma \sum_{s' \in S} p(s' \mid s)\, V(s')}_{\text{未来奖励的折扣总和}}$$



$R = [0\ 0\ 10\ 1000\ -1000]$ $\gamma = 0.95$

init $V = [0, 0, 0, 0, 0]$

```python
import numpy as np

# 定义状态转移概率矩阵
transition_probabilities = np.array([
    [0, 1, 0.0, 0, 0.0],
    [0.3, 0.0, 0.3, 0.4, 0.0],
    [0.0, 0.0, 0.0, 0.0, 1.0],
    [0.0, 0.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0]
])

# 定义奖励向量
rewards = np.array([0, 0, 10, 1000, -1000])

# 定义折扣因子
discount_factor = 0.95

# 初始化状态值函数
values = np.zeros(len(rewards))

# 迭代计算状态值函数
epsilon = 1e-6  # 定义收敛阈值
cnt=0
while True:
    prev_values = np.copy(values)
    for state in range(len(rewards)):
        values[state] = rewards[state] + discount_factor * np.sum(transition_probabilities[state] * values)
    if np.max(np.abs(prev_values - values)) < epsilon:
        break
    cnt+=1
    print(f'第{cnt}轮迭代结果:',values)


# 解析法求解状态值
values = np.linalg.inv(np.eye(5) - discount_factor * transition_probabilities).dot(rewards)

# 打印最终的状态值
print("Final state values:")
print(values)
```

```
/Users/tommyzhou/PythonProjects/timer&dairy/venv/bin/python /Users/tommyzhou/
第1轮迭代结果: [    0.     0.    10.  1000. -1000.]
第2轮迭代结果: [    0.    382.85  -940.    1000.   -1000.  ]
第3轮迭代结果: [  363.7075     215.7566375  -940.          1000.         -1000.
第4轮迭代结果: [  204.96880562   170.5161096   -940.          1000.
 -1000.        ]
第5轮迭代结果: [  161.99030412   158.26723668  -940.          1000.
 -1000.        ]
第6轮迭代结果: [  150.35387484   154.95085433  -940.          1000.
 -1000.        ]
第7轮迭代结果: [  147.20331161   154.05294381  -940.          1000.
 -1000.        ]
第8轮迭代结果: [  146.35029662   153.80983454  -940.          1000.
 -1000.        ]
第9轮迭代结果: [  146.11934281   153.7440127   -940.          1000.
```

```
第17轮迭代结果: [  146.03359864   153.71957561  -940.          1000.
 -1000.        ]
第18轮迭代结果: [  146.03359683   153.7195751   -940.          1000.
 -1000.        ]
Final state values:
[  146.03359616   153.71957491  -940.          1000.
 -1000.        ]
```

1: 对于所有状态 $s \in S, V'(s) \leftarrow 0, V(s) \leftarrow \infty$
2: 当 $\|V - V'\| > \epsilon$ 执行
3: $V \leftarrow V'$
4: 对于所有状态 $s \in S, V'(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s) V(s')$
5: 结束循环
6: 返回 $V'(s)$ 对于所有状态 $s \in S$

贝尔曼方程（Bellman Equation）
解析法

$$V(s) = \underbrace{R(s)}_{\text{即时奖励}} + \underbrace{\gamma \sum_{s' \in S} p(s' \mid s) V(s')}_{\text{未来奖励的折扣总和}}$$

当我们把贝尔曼方程写成矩阵形式后，可以直接求解：

$$\begin{aligned} \boldsymbol{V} &= \boldsymbol{R} + \gamma \boldsymbol{P} \boldsymbol{V} \\ \boldsymbol{I}\boldsymbol{V} &= \boldsymbol{R} + \gamma \boldsymbol{P} \boldsymbol{V} \\ (\boldsymbol{I} - \gamma \boldsymbol{P}) \boldsymbol{V} &= \boldsymbol{R} \\ \boldsymbol{V} &= (\boldsymbol{I} - \gamma \boldsymbol{P})^{-1} \boldsymbol{R} \end{aligned}$$

我们可以直接得到**解析解**（analytic solution）：

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

```
# 解析法求解状态值
values = np.linalg.inv(np.eye(5) - discount_factor * transition_probabilities).dot(rewards)
```

```
迭代法结果:
[  146.03359634   153.71957496  -940.          1000.
 -1000.        ]
解析法结果:
[  146.03359616   153.71957491  -940.          1000.
 -1000.        ]
```

Copilot

计算一个$n \times n$矩阵的逆矩阵的时间复杂度通常是$O(n^3)$。

所以不适用于状态空间很大的情况

本章小结

- 什么是马尔可夫奖励过程的预测问题？
- 如何使用蒙特卡洛方法解决上面的问题？
- 如何使用贝尔曼方程解决上面的问题？
- 什么是贝尔曼方程的解析法和迭代法？

下一章：马尔可夫决策过程

Credit goes to: EasyRL