强化学习原理：第二讲
——有限马尔科夫决策过程

杨博渊

yby@nankai.edu.cn

2023.02.24

提纲

- 多臂赌博机
- 马尔科夫过程
- 马尔科夫奖励过程
- 马尔科夫决策过程
- Python基础

强化学习的核心思想

强化学习区别于其他类型学习的最大特征：

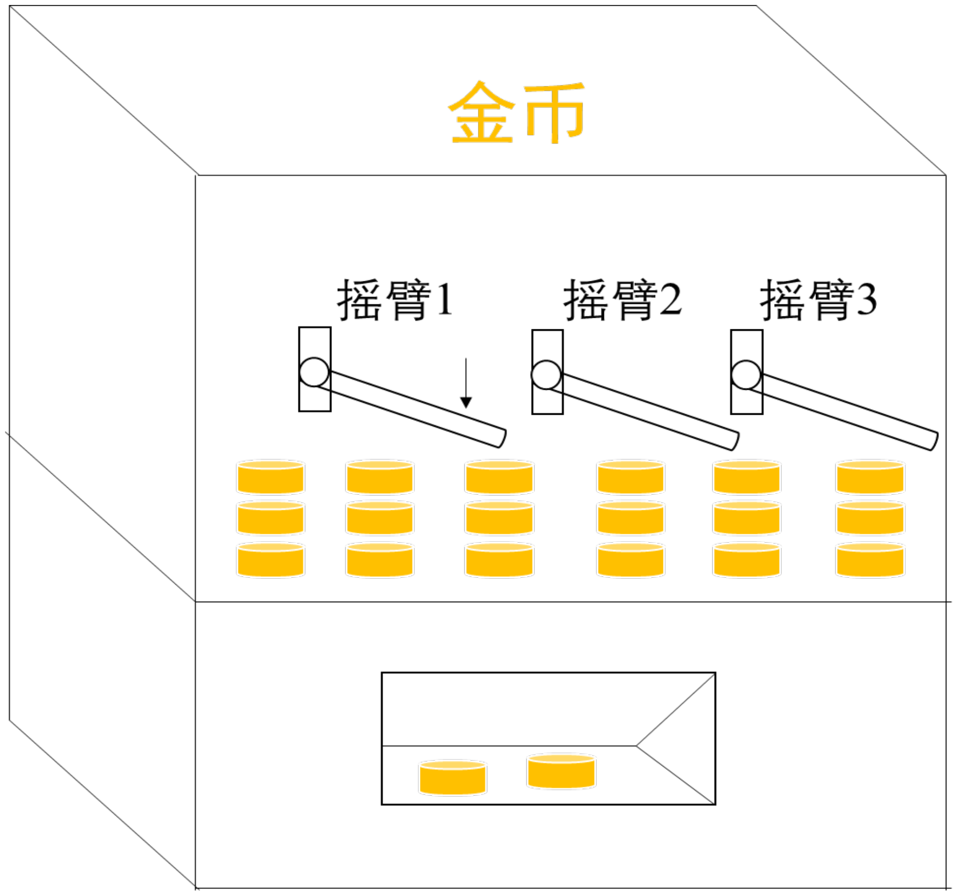
利用训练信息来评估动作，而不是通过给定的正确动作

监督学习：利用“动作对还是不对”这个信息

强化学习：利用“每个动作多好或多坏”

采用该动作，观察该动作带来的后继回报，利用后继回报来进行评估

多臂赌博机



多臂赌博机：

K个臂

摇动每个臂时，得到不同概略分布的回报

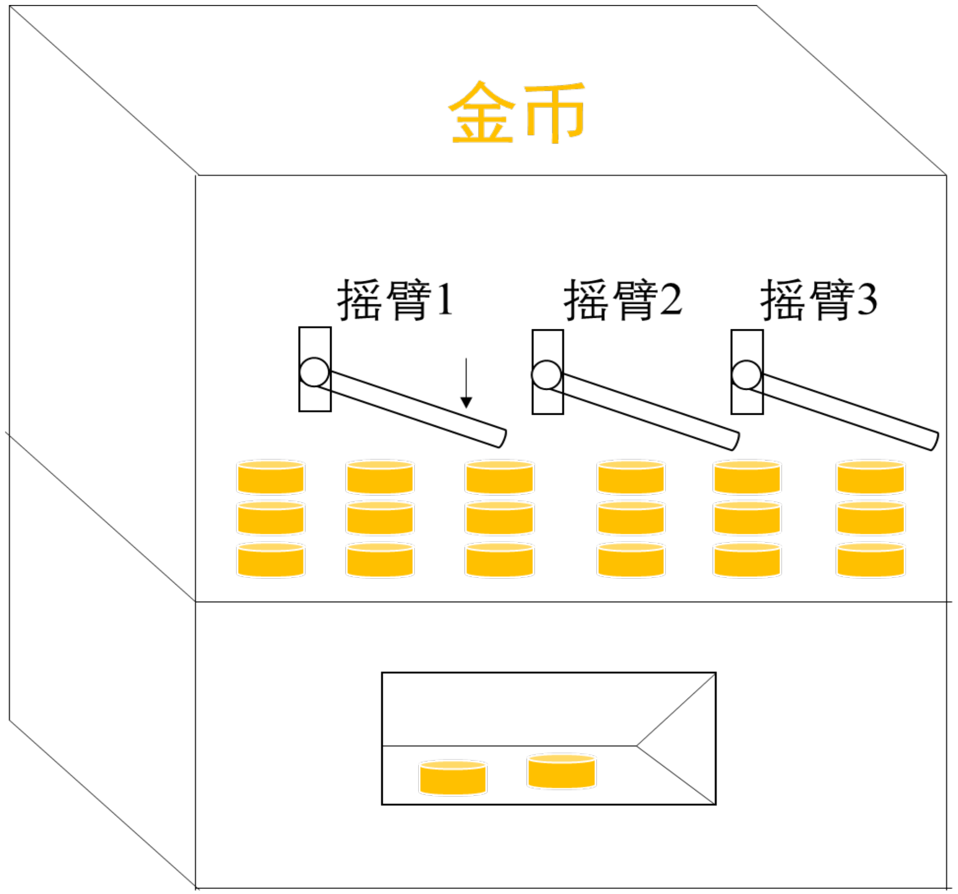
目的：

如何摇动N次得到最高的回报

问题：

如何通过学习知道，摇哪个臂能得到最好的回报

多臂赌博机



动作：a = [0, 1, 2]

立即回报：r，服从三个不同的概率分布的回报

状态：s

目标：在状态s，最优的动作

定义动作的值：$q_*(a) = \mathbb{E}[R_t \mid A_t = a]$

强化学习：利用回报r学习最优的动作，学习动作的评估

$$s \xrightarrow[r=0.2]{a=0} s_T$$

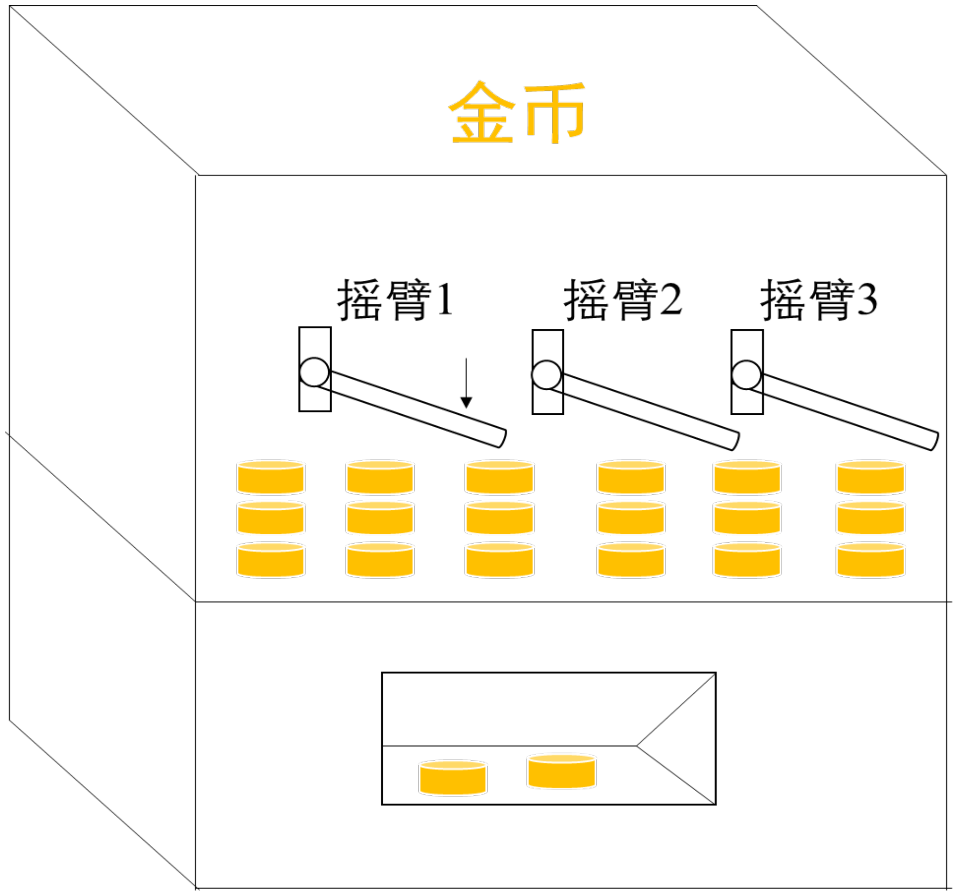
$$s \xrightarrow[r=0.3]{a=1} s_T$$

$$s \xrightarrow[r=0.6]{a=2} s_T$$

定义每个动作的估计值：

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathrm{II}_{A_i=a}}{\sum_{i=1}^{t-1} \mathrm{II}_{A_i=a}}$$

多臂赌博机



$$s \xrightarrow[r=0.2]{a=0} s_T,\ q[0]=0.2$$

$$s \xrightarrow[r=0.3]{a=1} s_T,\ q[1]=0.3$$

$$s \xrightarrow[r=0.6]{a\ =\ 2} s_T, q[2]=0.6$$

在每个杆都试过一次后，你如何选下一个杆？

利用当前的知识

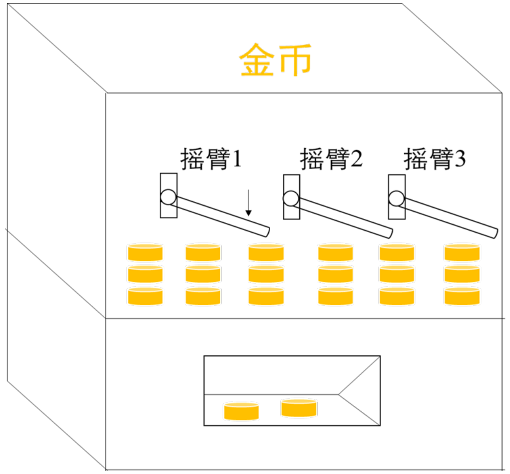
贪婪策略：$\mathrm{a}=\arg\max_a q(a)$ exploitation

选择非贪婪动作 exploration

探索-利用平衡策略与环境进行交互：

$$\varepsilon-greedy{:}a=\begin{cases}\arg\max_a q(a) & with\ probability\ 1-\varepsilon\\ a\ random\ action & with\ probability\ \varepsilon\end{cases}$$

多臂赌博机



A simple bandit algorithm

Initialize, for $a = 1$ to k:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$

增量式计算：

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)Q_n\right) \\
&= \frac{1}{n}\left(R_n + nQ_n - Q_n\right) \\
&= Q_n + \frac{1}{n}\left[R_n - Q_n\right]
\end{aligned}
$$

多臂赌博机

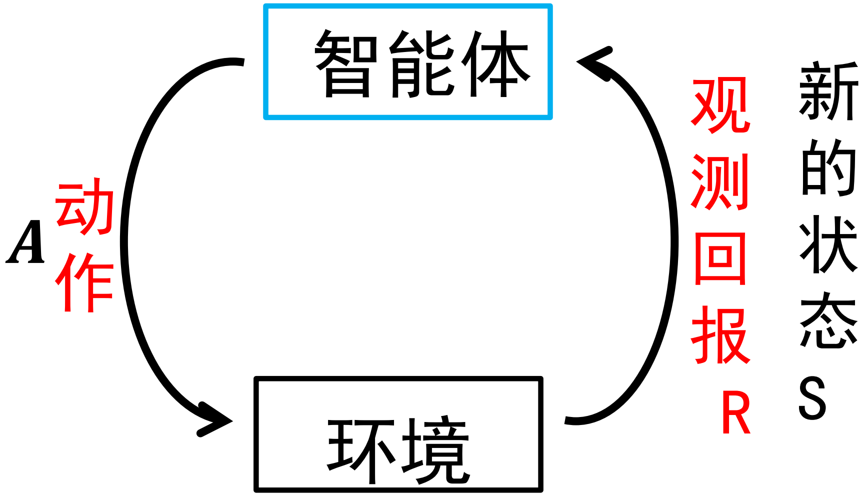
A simple bandit algorithm

Initialize, for $a = 1$ to k:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \operatorname{argmax}_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$



状态转移概率 $P(S_{t+1}|S_t, a)$

所有强化学习算法包括且只包括两个过程
：

采集数据：利用探索-利用平衡策略进行采集数据

动作评估

学习：利用采集到的数据优化当前策略

多臂赌博机

Upper-Confidence-Bound 动作选择

$$A_t = \arg\max_a [Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}]$$

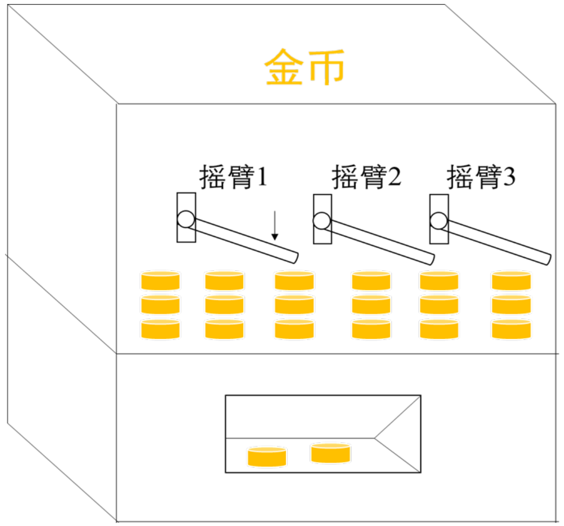
$$\downarrow$$

不确定性的度量

玻尔兹曼分布：动作选择

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

多臂赌博机



A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

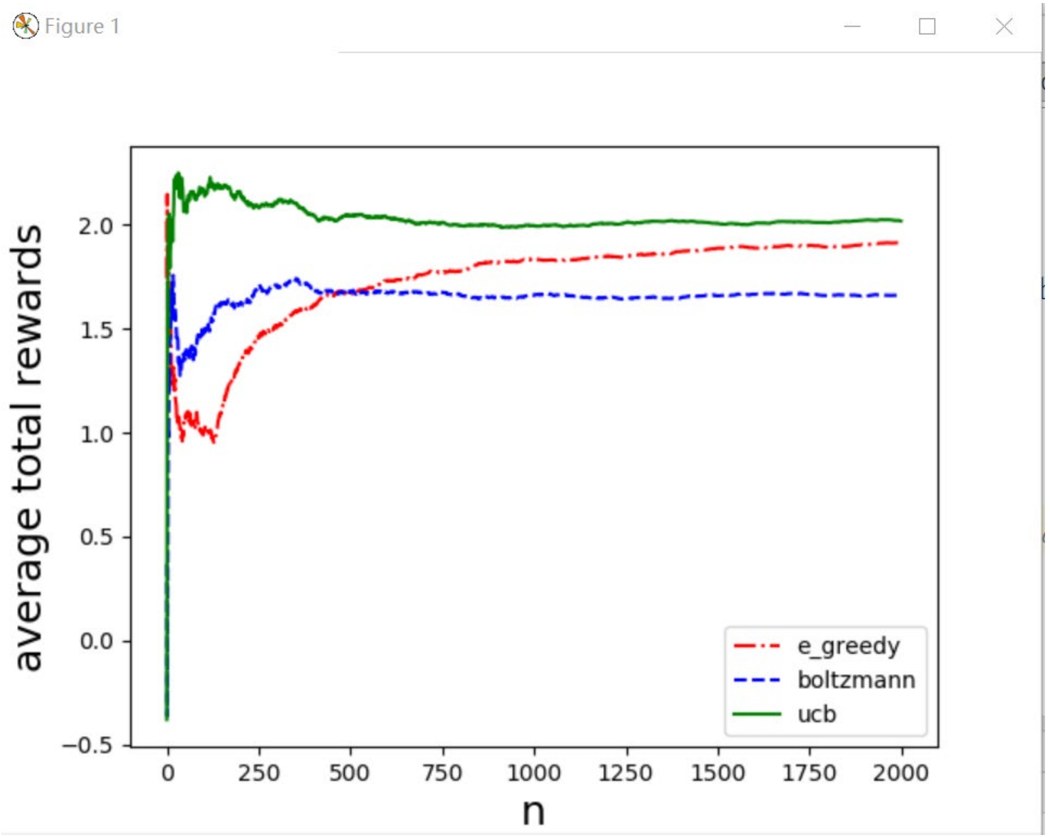
Loop forever:

$$A \leftarrow \begin{cases} \operatorname{argmax}_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

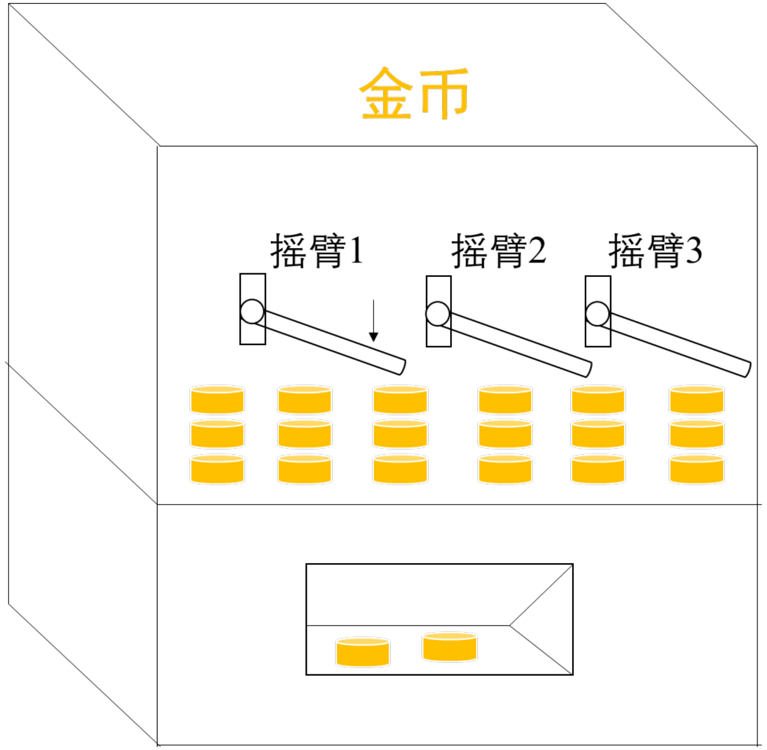
$N(A) \leftarrow N(A) + 1$

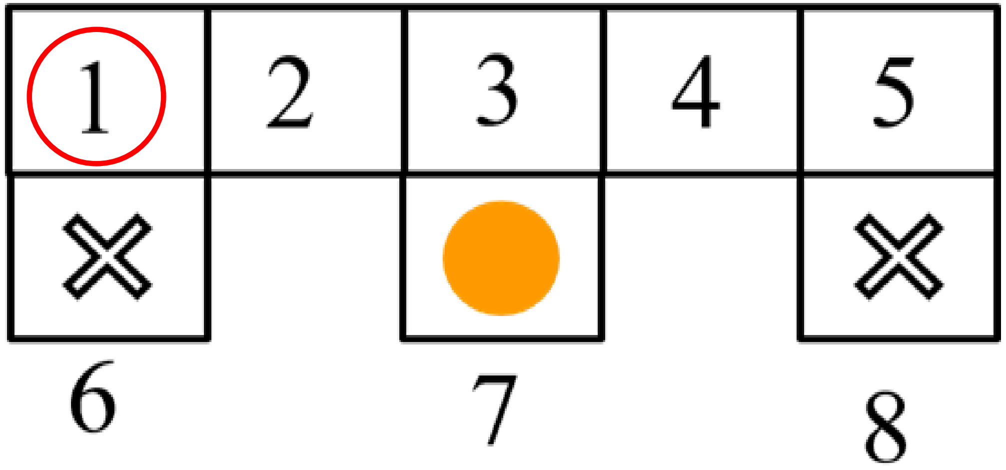
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$



多臂赌博机的学习曲线

从多臂赌博机到马尔科夫决策过程





单状态 ⟹ 多状态

从多臂赌博机到马尔科夫决策过程

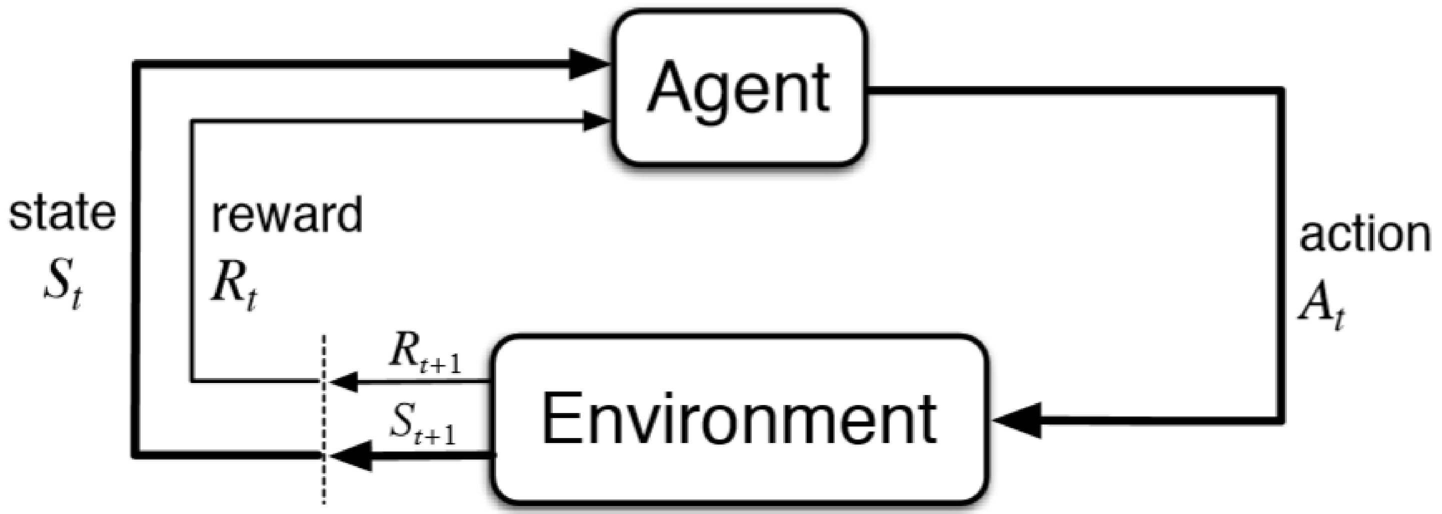
Agent: 智能体

Environment: 环境

智能体与环境进行序贯交互：

trajectory(轨迹)

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \cdots$$



智能体与环境交互

如果将状态 S_t 和回报 R_t 视为随机变量，那么该轨迹可以用一个随机过程来描述

随机变量，条件概率，分布，期望，方差，随机过程，马尔科夫过程的概念

马尔科夫过程：马尔科夫性

马尔科夫性： 系统的下一个状态只与当前状态有关，与以前状态无关。

定义

一个状态St是马尔科夫的，当且仅当：

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, \cdots, S_t]$$

1. 当前状态蕴含所有相关的历史信息
2. 一旦当前状态已知，历史信息将会被抛弃

马尔科夫过程：状态转移

对于一个马尔可夫状态s和它的下一个状态s’，状态转移函数可以定义为

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

所有状态s和所有可能的下一个状态s’，定义了状态转移矩阵

$$\mathcal{P} = \begin{array}{cc} & to \\ from & \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \end{array}$$

马尔科夫过程：定义

马尔科夫过程 又叫马尔科夫链(Markov Chain)，它是一个无记忆的随机过程。

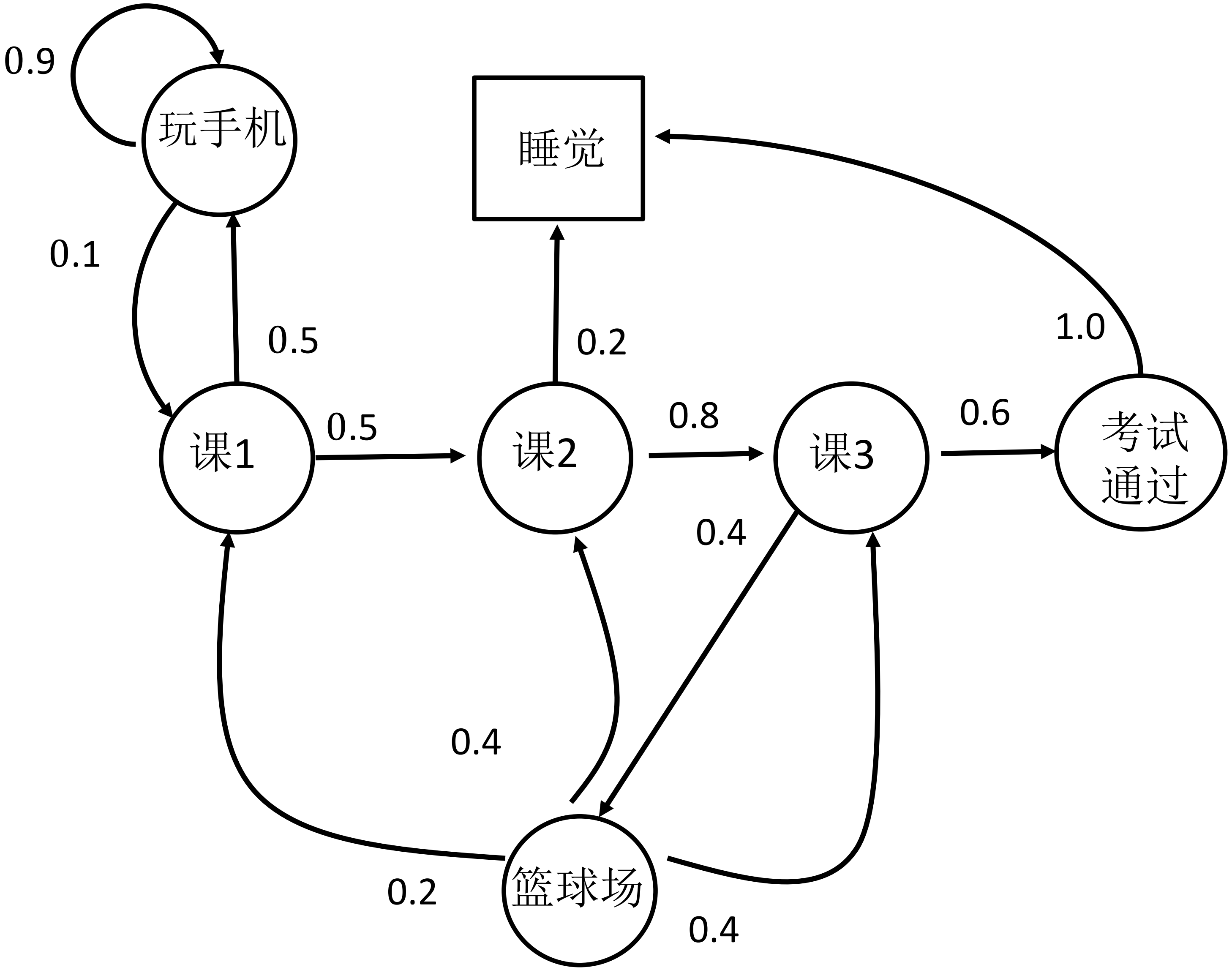
定义

马尔科夫过程是元组：<S, P>

- S是有限状态的集合
- P是状态转移概率矩阵

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

马尔科夫过程：示例



马尔科夫奖励过程：定义

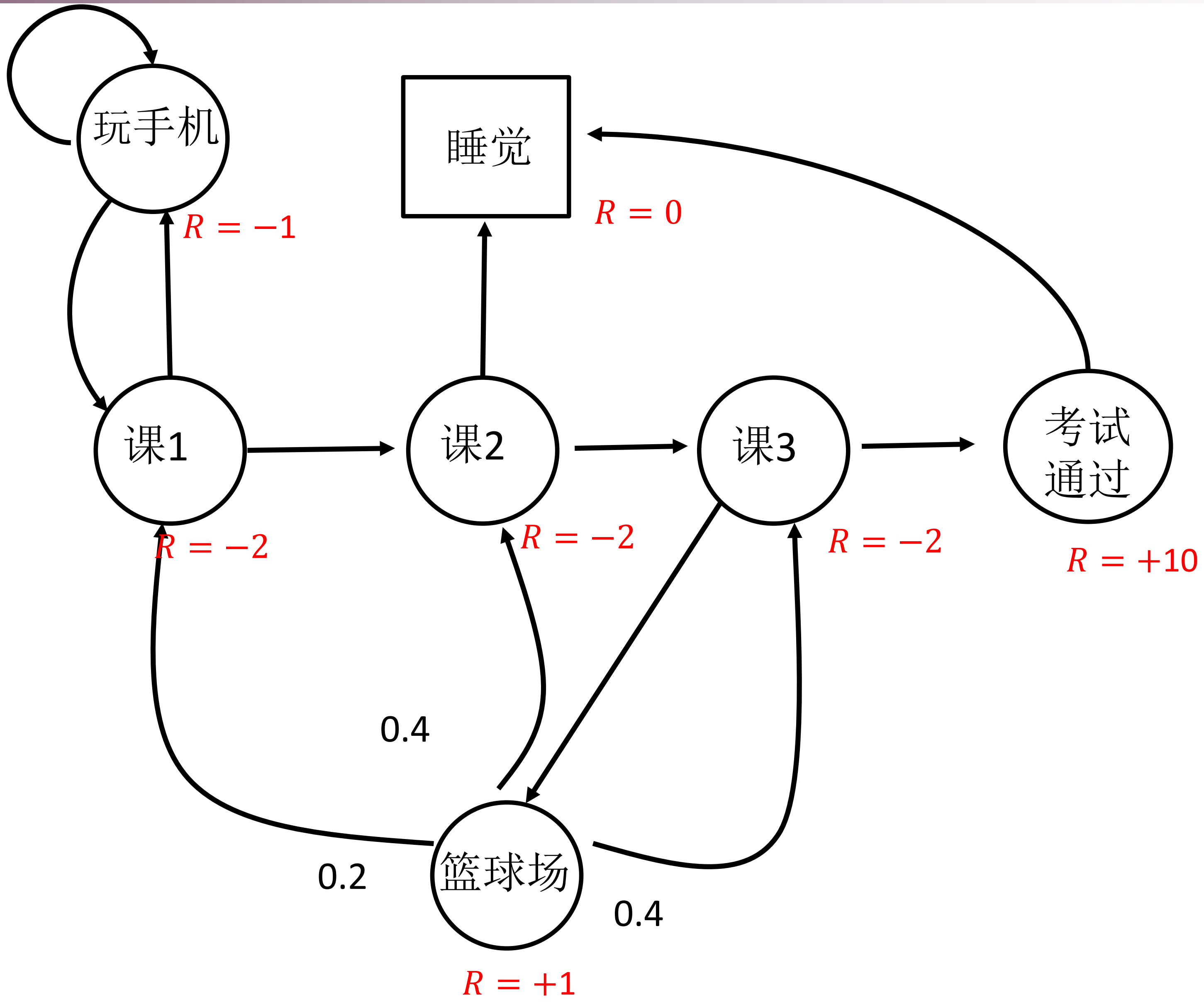
马尔科夫奖励过程在马尔科夫过程的基础上增加了奖励R和衰减系数γ。

定义

马尔科夫奖励过程是元组：<S, P, R, γ>

- S是有限状态的集合
- P是状态转移概率矩阵 $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$
- R是奖励函数 $R_s = E[R_{t+1} | S_t = s]$
- γ是衰减系数 γ∈ [0, 1]

马尔科夫奖励过程：示例



马尔科夫奖励过程：回报

定义

回报G_t为在一个马尔科夫奖励链上从t时刻开始往后所有的奖励的有衰减的总和。

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

γ 称为折扣率，并且 $0 \le \gamma \le 1$

$\gamma=0$： 智能体是短视的（myopic），最大化立即回报

γ接近1，说明智能体是有远见的（farsighted），考虑将来的回报

马尔科夫奖励过程：值函数

值函数给出了某一状态或某一行为的长期价值。

定义

一个马尔科夫奖励过程中某一状态的值函数为从该状态开始的马尔科夫链回报的期望

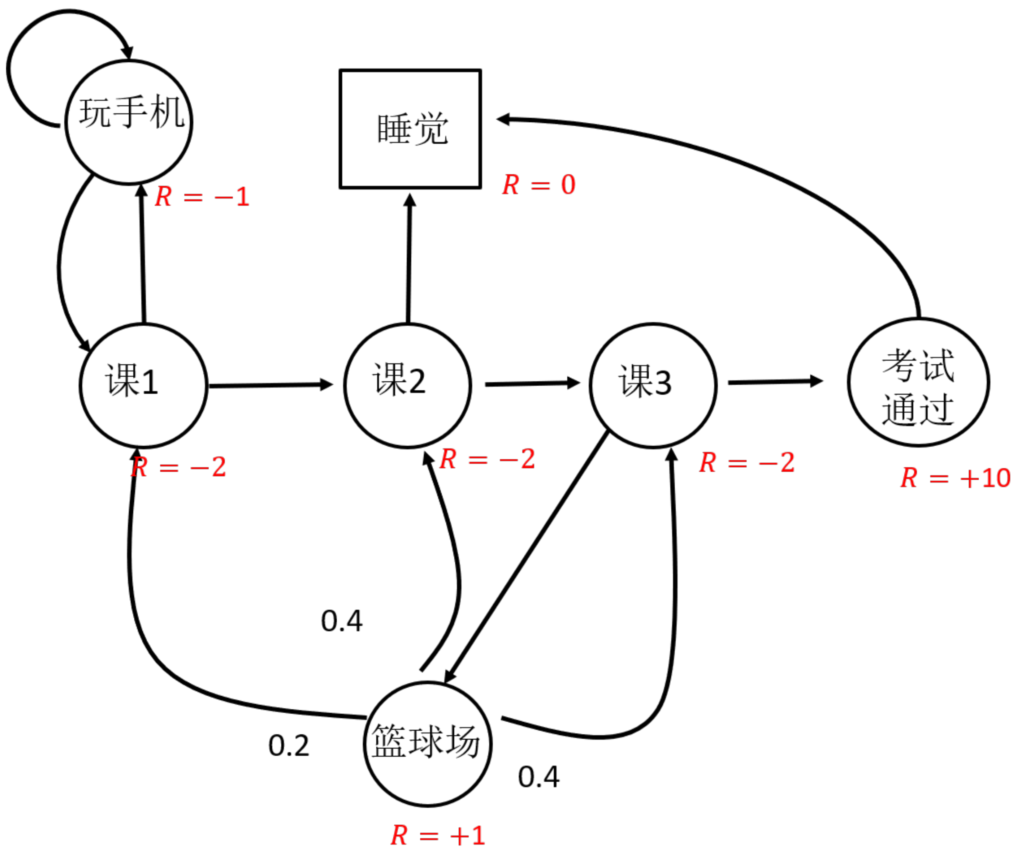
$$v(s) = E[G_t | S_t = s]$$

马尔科夫奖励过程：回报示例

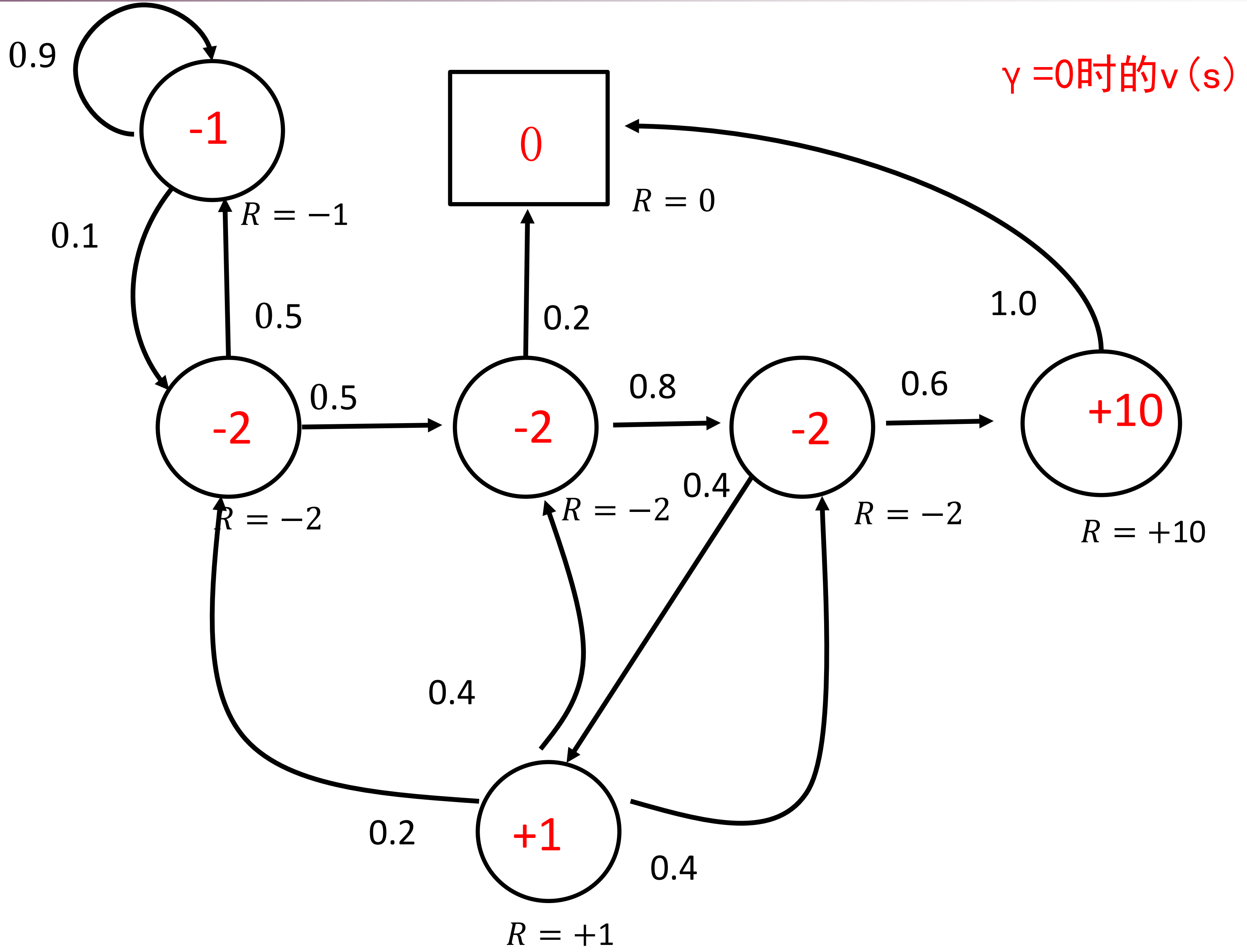
学生马尔科夫奖励过程的回报的例子

假设 γ =0. 5，考虑如下的从课1开始的马尔科夫链

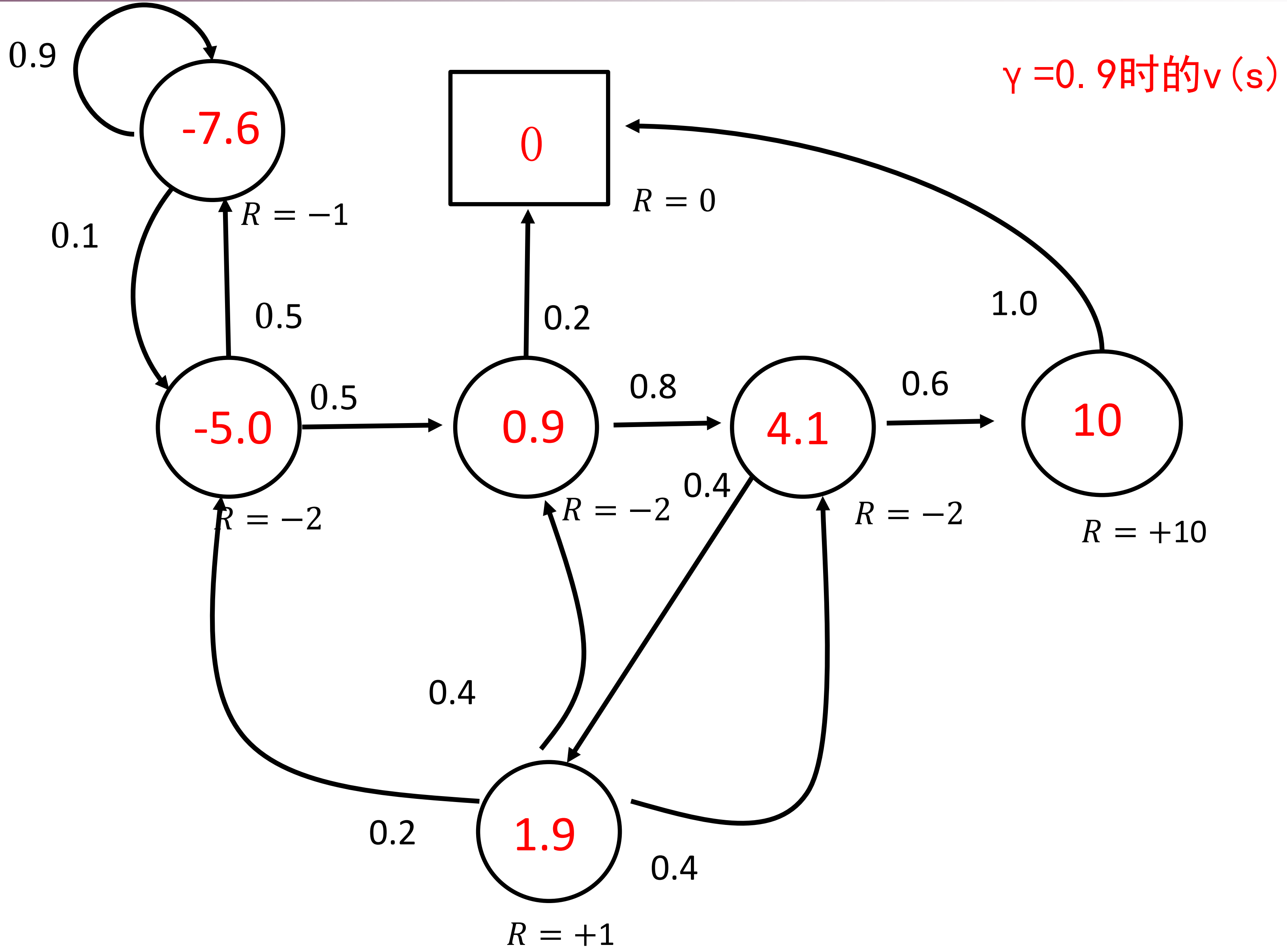
$$G_1 = R_2 + \gamma R_3 + \ldots + \gamma^{T-2} R_T$$



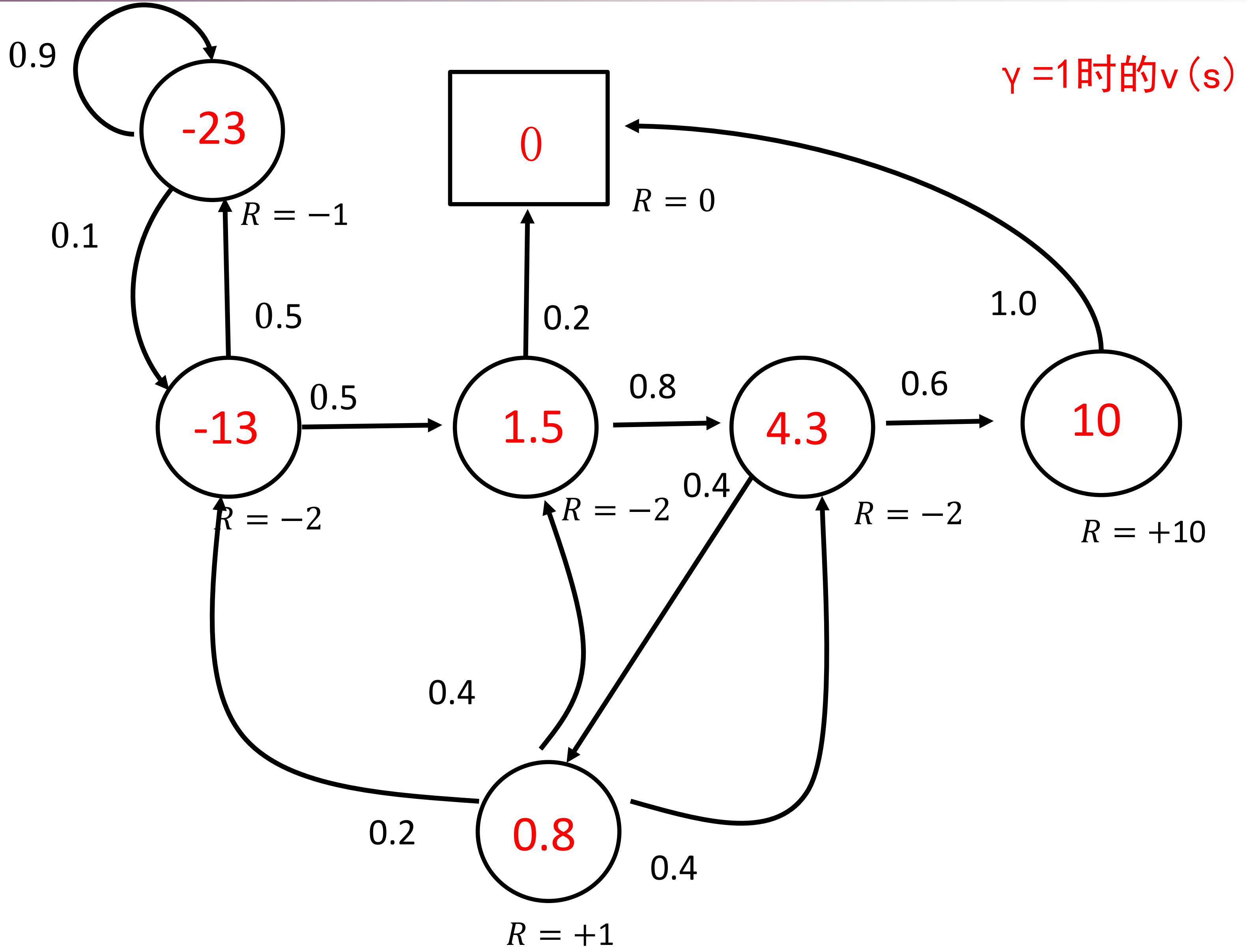
马尔科夫奖励过程：状态值函数示例



马尔科夫奖励过程：状态值函数示例



马尔科夫奖励过程：状态值函数示例



马尔科夫奖励过程：贝尔曼方程

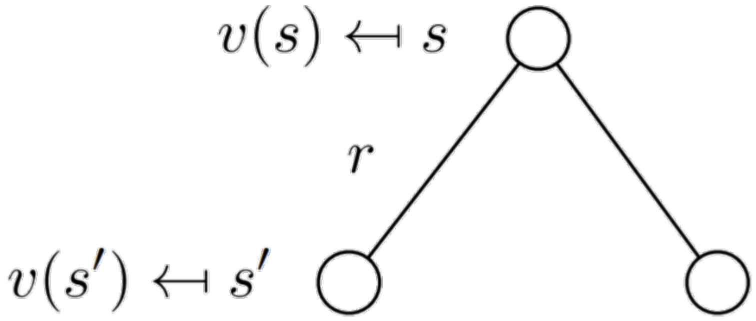
$$
\begin{aligned}
v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]
\end{aligned}
$$

状态值函数可以分解成两部分：

- 立即回报R_{t+1}
- 下一个状态的折扣值

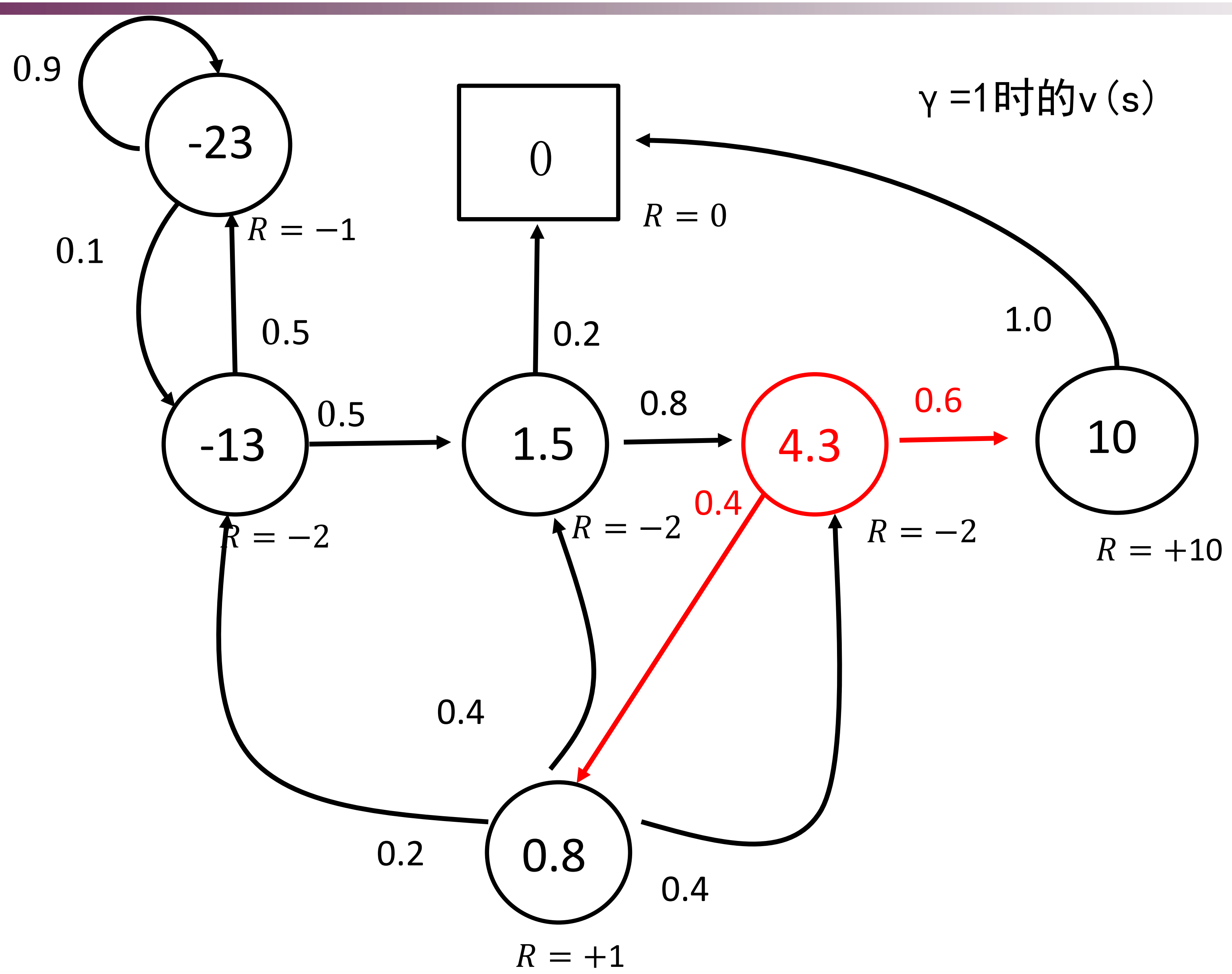
马尔科夫奖励过程：贝尔曼方程

$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]$$



$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

马尔科夫奖励过程：贝尔曼方程



马尔科夫奖励过程：贝尔曼方程

贝尔曼方程可以写成矩阵形式

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

v是列向量，元素是每一个进入状态的值函数

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{11} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

马尔科夫奖励过程：贝尔曼方程

$$v = (I - \gamma \mathcal{P})^{-1} \mathcal{R}$$

计算复杂度是O(n^3)， n是状态数量

直接求解仅适用于小规模的MRPs

大规模MRP的求解通常使用迭代法

- 动态规划 Dynamic Programming
- 蒙特卡洛评估 Monte-Carlo evaluation
- 时间差分学习 Temporal-Difference

马尔科夫决策过程：定义

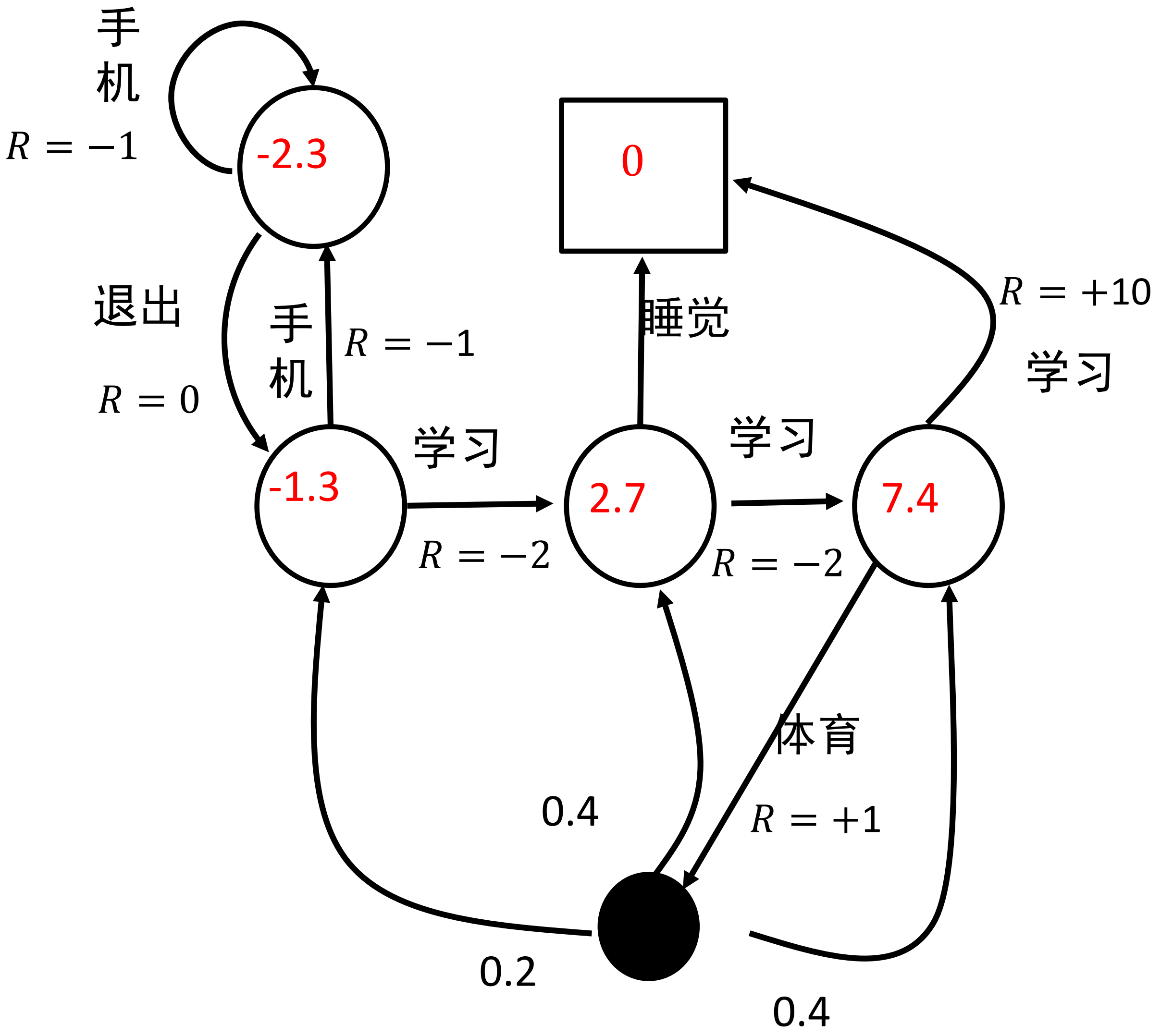
马尔科夫决策过程是带有决策的马尔科夫奖励过程。
是一个所有状态都具有马尔科夫性的环境。

定义

马尔科夫决策过程是元组：<S, A, P, R, γ >

- S是有限状态的集合
- A是有限动作的集合
- P是状态转移概率矩阵 $\mathcal{P}_{ss'}^{a} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- R是奖励函数 $\mathcal{R}_{s}^{a} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- γ 是衰减系数 γ ∈ [0, 1]

马尔科夫决策过程：示例



马尔科夫决策过程：策略

定义

策略是在给定状态s下，采取行动的概率的集合或者分布

$$\pi(a|s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$$

- 一个策略完整定义了智能体的行为方式
- MDPs的策略仅和当前的状态有关，与历史信息无关
- 策略是静态的（与时间无关）；但是智能体可以随着时间更新策略

马尔科夫决策过程：策略

对于一个给定的MDP: <S,A,P,R,γ>和一个策略π

那么状态序列$S_1, S_2 \ldots$是一个马尔科夫过程$<S, P^{\pi}>$

状态和奖励序列$S_1, R_2, S_2, R_3, S_3 \ldots$是一个马尔科夫奖励过程$<S, P^{\pi}, R^{\pi}, \gamma>$

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

$$\mathcal{R}^{\pi}_{s} = \sum_{a \in \mathcal{A}} \pi(a|s)\mathcal{R}^{a}_{s}$$

马尔科夫决策过程：值函数

定义

$v_\pi(s)$ 是在MDP下的基于策略π的状态值函数，表示从状态s开始，遵循当前策略时所获得的收获的期望；或者说在执行当前策略π时，衡量个体处在状态s时的价值大小

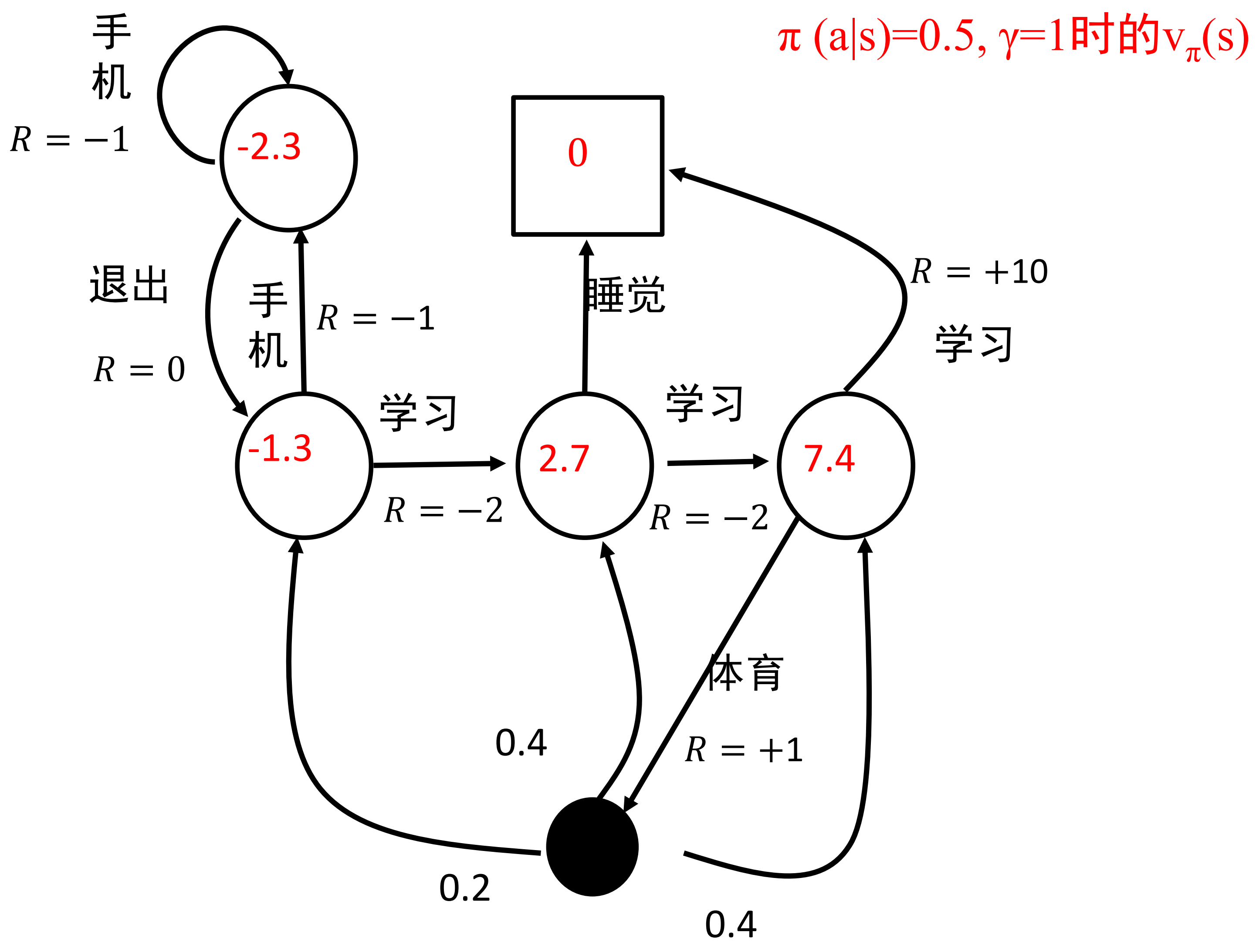
$$v_\pi(s) = \mathbb{E}_\pi\left[G_t \mid S_t = s\right]$$

定义

$q_\pi(s,a)$是行为值函数，表示在执行策略π时，对当前状态s执行某一具体行为a所能的到的收获的期望；或者说在遵循当前策略π时，衡量对当前状态执行行为a的价值大小

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

马尔科夫决策过程：值函数例子



马尔科夫决策过程：贝尔曼期望方程

MDP下的状态值函数与MRP下的值函数类似，可以改用下一时刻状态值函数来表达，具体方程如下：

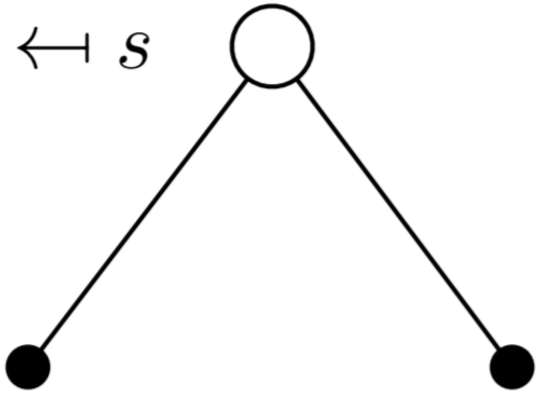
$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]$$

同样地，对于行为值函数：

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

马尔科夫决策过程：贝尔曼期望方程

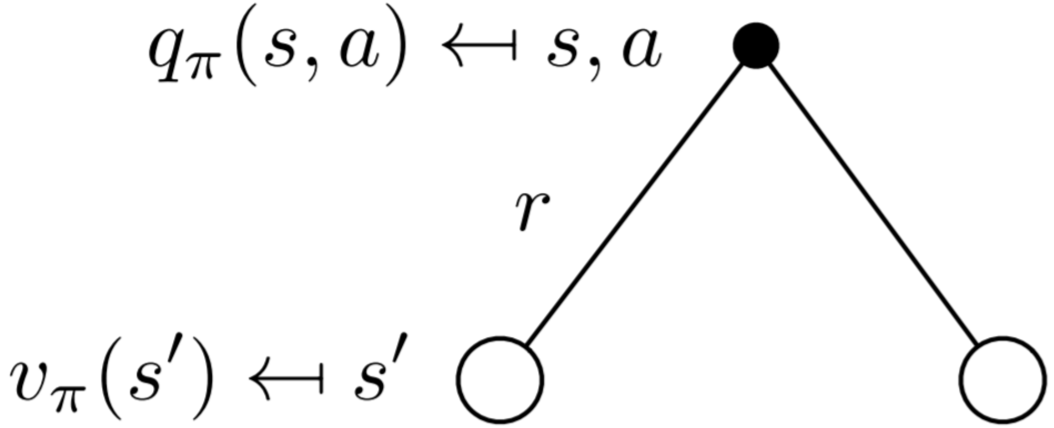
$v_\pi(s) \hookleftarrow s$



$q_\pi(s,a) \hookleftarrow a$

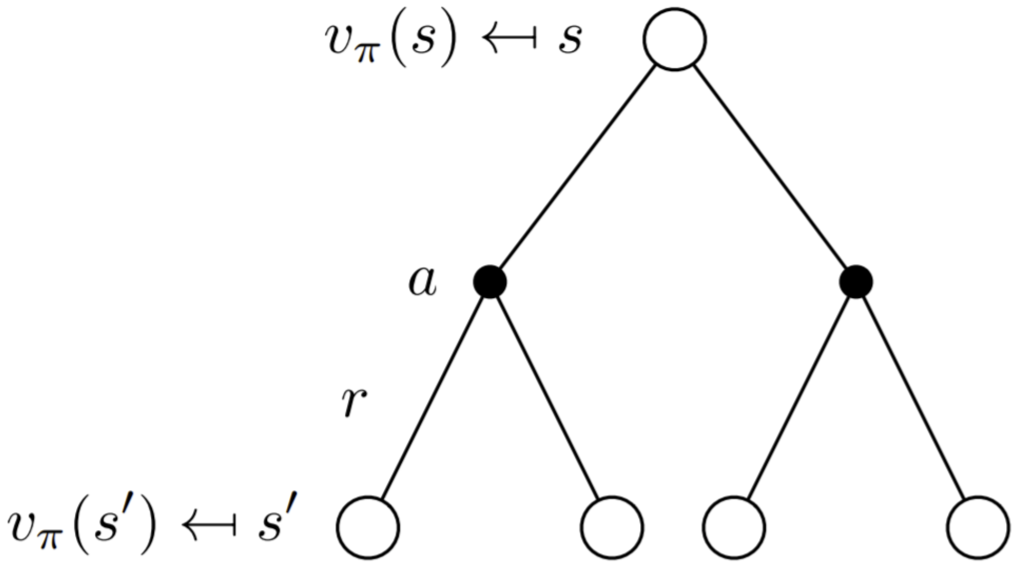
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

马尔科夫决策过程：贝尔曼期望方程



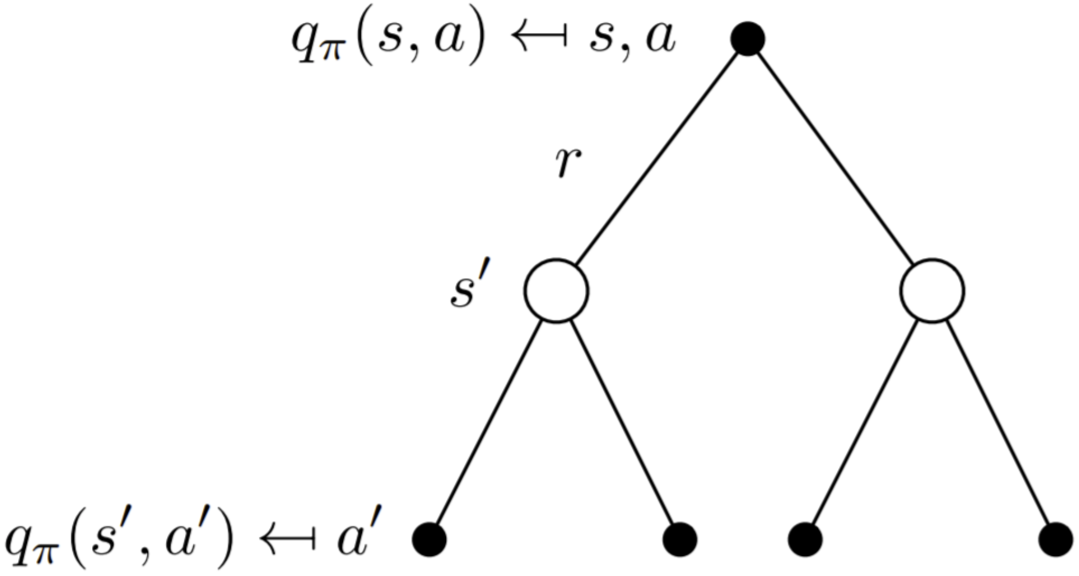
$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

马尔科夫决策过程：贝尔曼期望方程



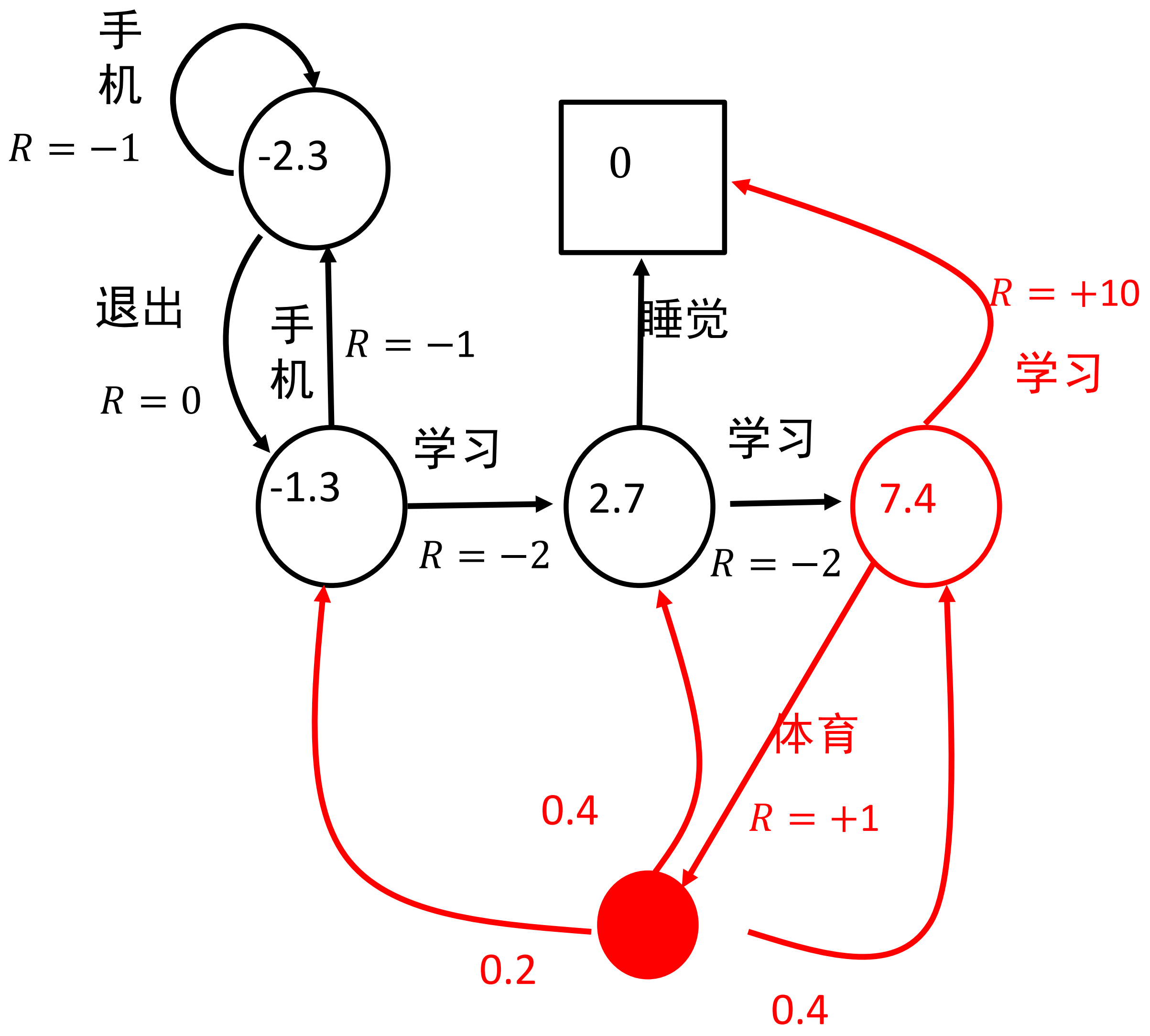
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

马尔科夫决策过程：贝尔曼期望方程



$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

马尔科夫决策过程：贝尔曼期望方程



马尔科夫决策过程：贝尔曼期望方程

类似MRP，我们将贝尔曼期望方程写成矩阵形式

$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi$$

直接求解

$$v_\pi = (I - \gamma \mathcal{P}^\pi)^{-1} \mathcal{R}^\pi$$

马尔科夫决策过程：最优值函数

定义

最优状态值函数 $v_*(s)$ 指的是在从所有策略产生的状态值函数中，选取使状态s值最大的函数：

$$v_* = \max_{\pi} v_\pi(s)$$

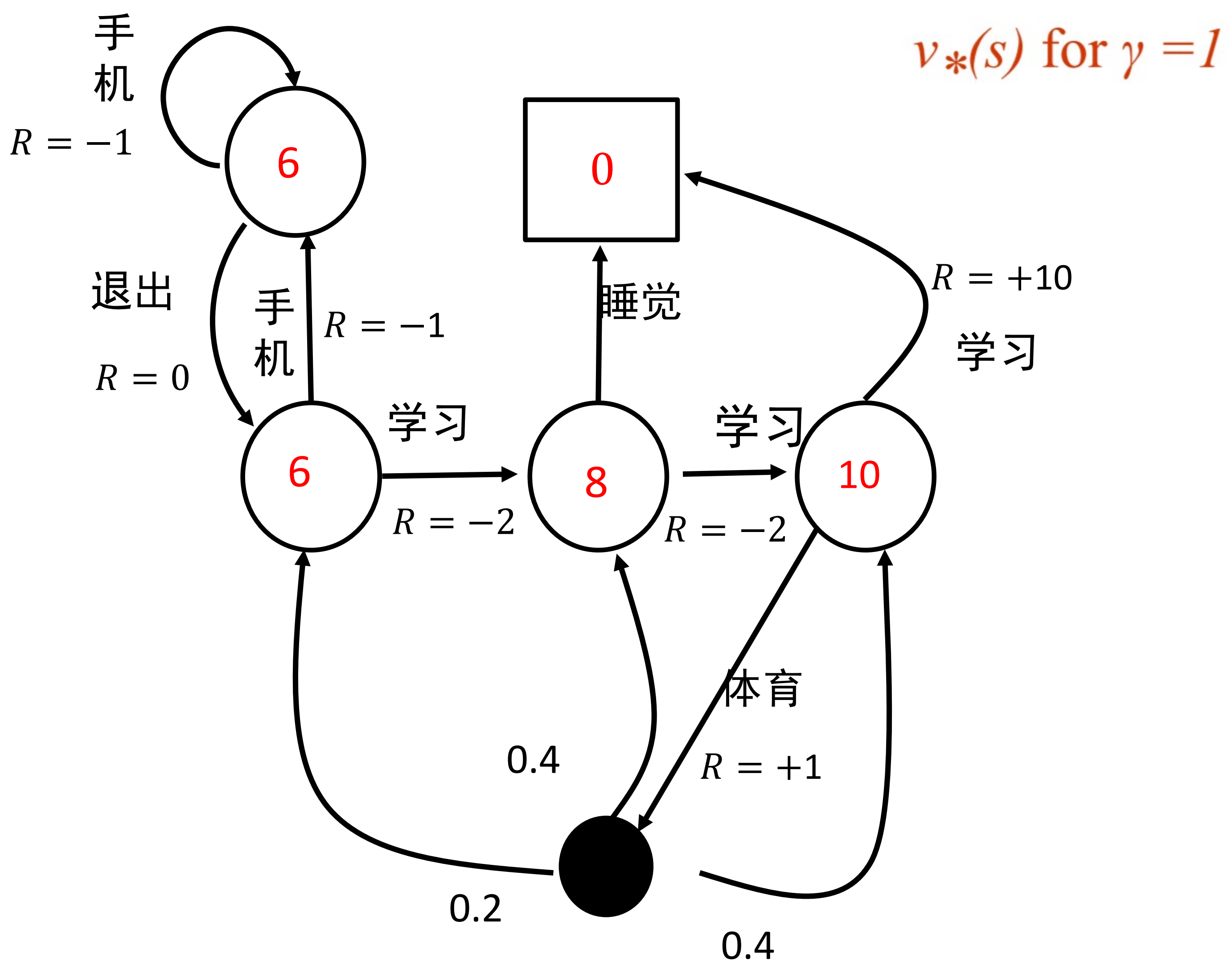
类似的，最优行为值函数 $q_*(s,a)$ 指的是从所有策略产生的行为值函数中，选取是状态行为对<s,a>最大的函数

$$q_*(s,a) = \max_{\pi} q_\pi(s,a)$$

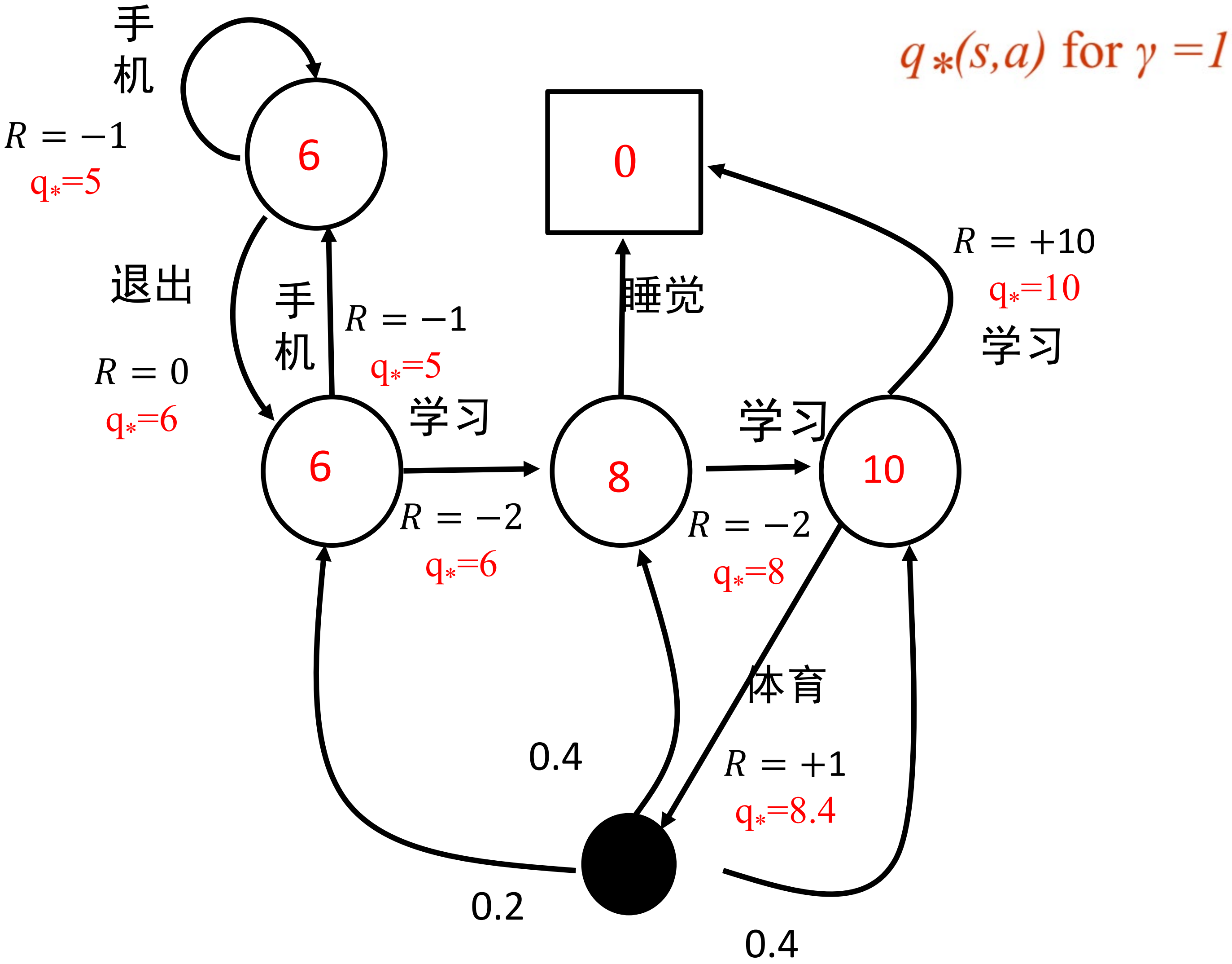
最优价值函数明确了MDP的最优可能表现

当我们知道了最优值函数，这时便认为这个MDP获得了解决

马尔科夫决策过程：最优值函数



马尔科夫决策过程：最优值函数



马尔科夫决策过程：最优策略

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

定理 对于任何MDP，下面几点成立：

1. 存在一个最优策略，比任何其他策略更好或至少相等；

2. 所有的最优策略有相同的最优价值函数；

3. 所有的最优策略具有相同的行为价值函数。

马尔科夫决策过程：寻找最优策略

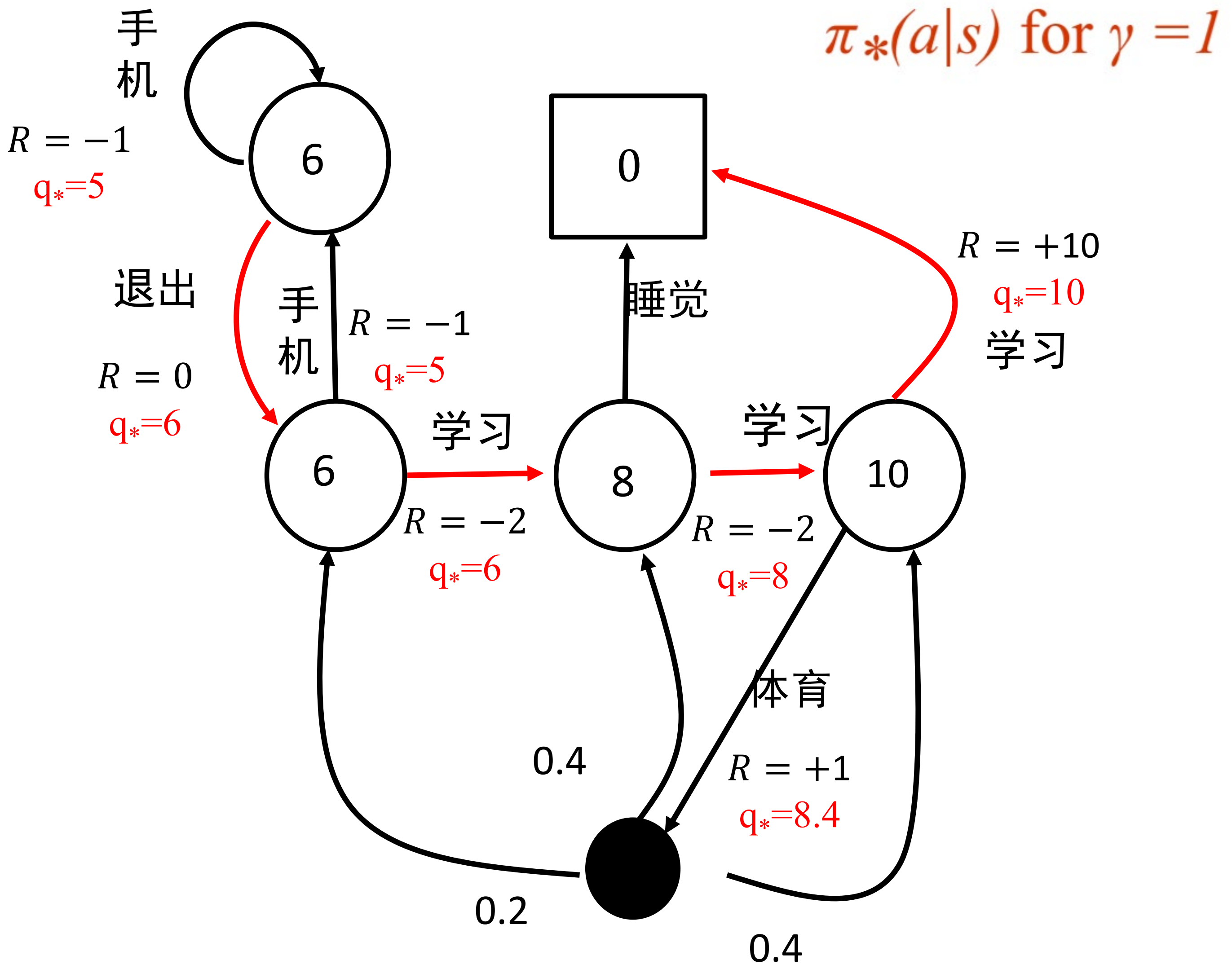
可以通过最大化最优行为价值函数来找到最优策略：

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ q_*(s,a) \\ 0 & otherwise \end{cases}$$

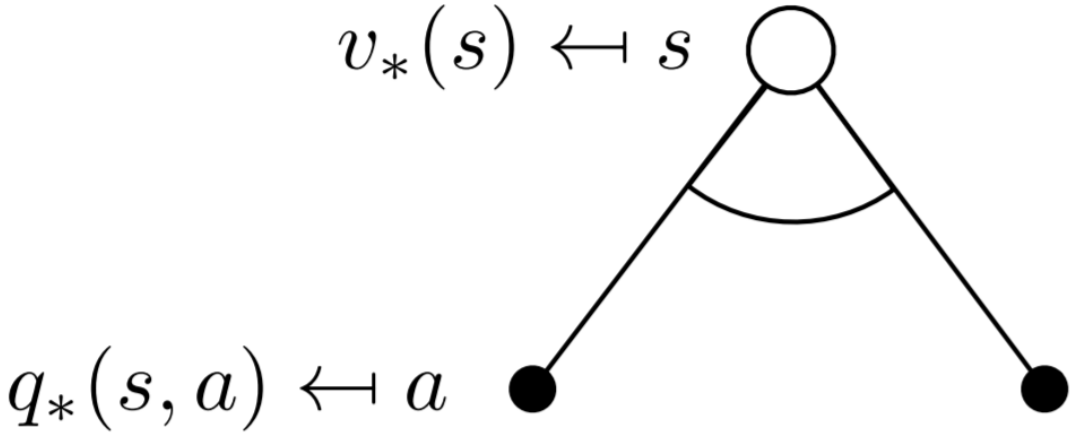
对于任何MDP问题，总存在一个确定性的最优策略；

同时如果我们知道最优行为价值函数，则表明我们找到了最优策略。

马尔科夫决策过程：最优策略示例

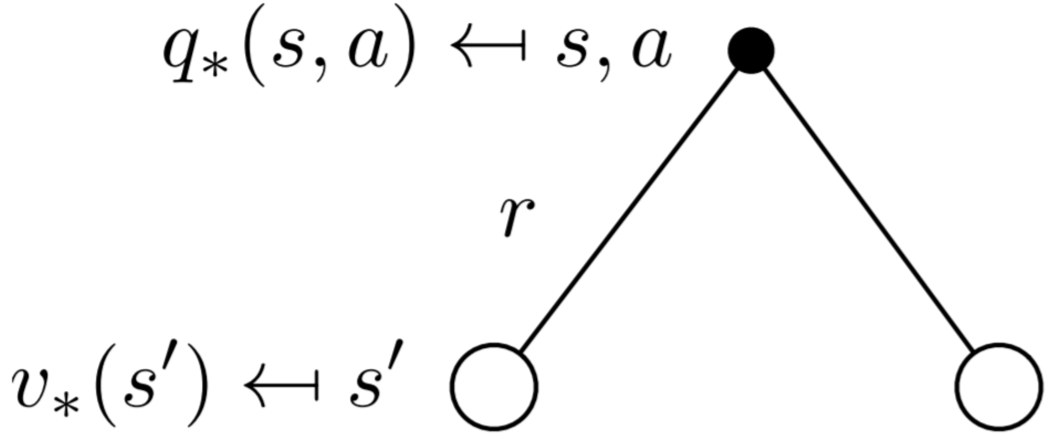


马尔科夫决策过程：贝尔曼最优方程



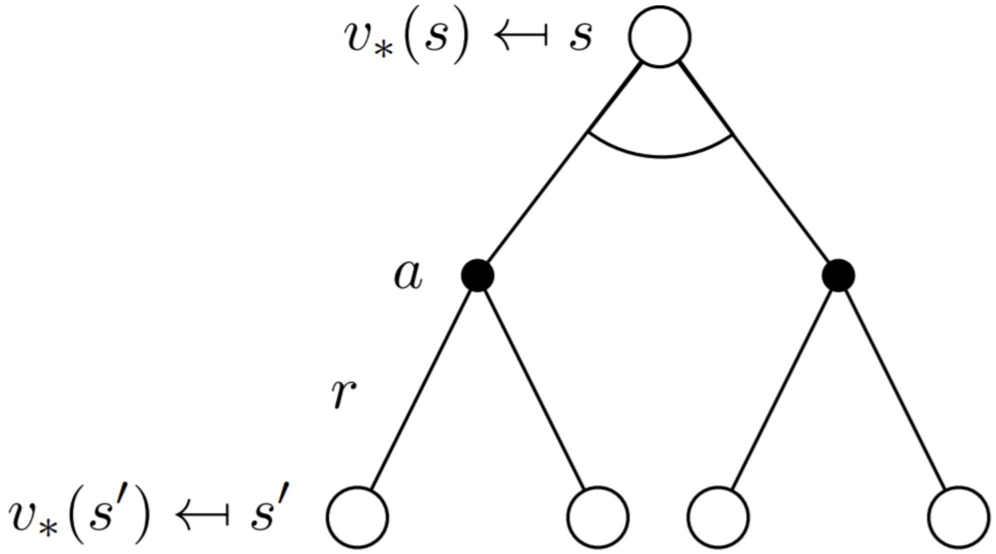
$$v_*(s) = \max_a q_*(s, a)$$

马尔科夫决策过程：贝尔曼最优方程



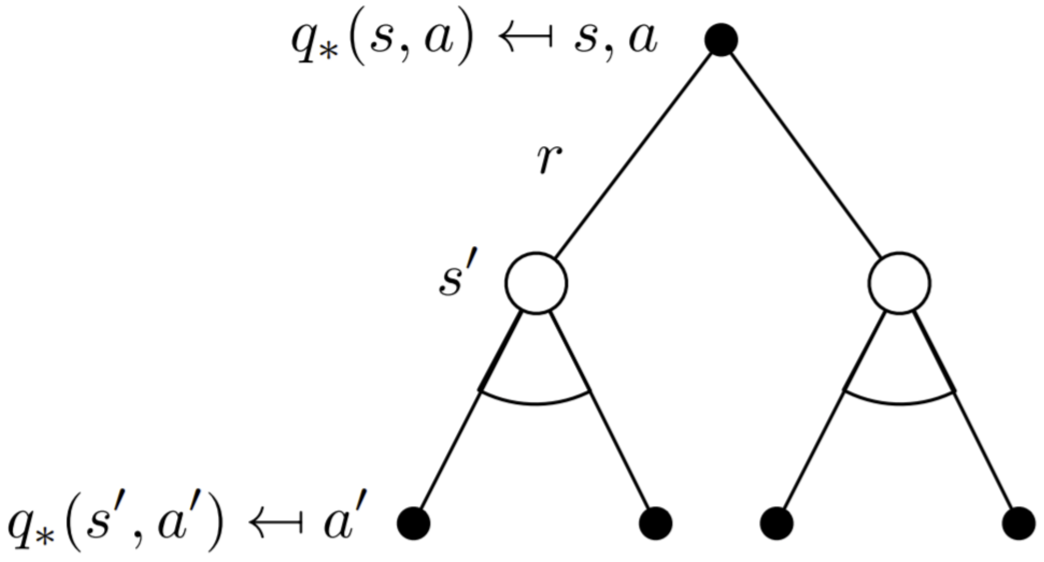
$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

马尔科夫决策过程：贝尔曼最优方程



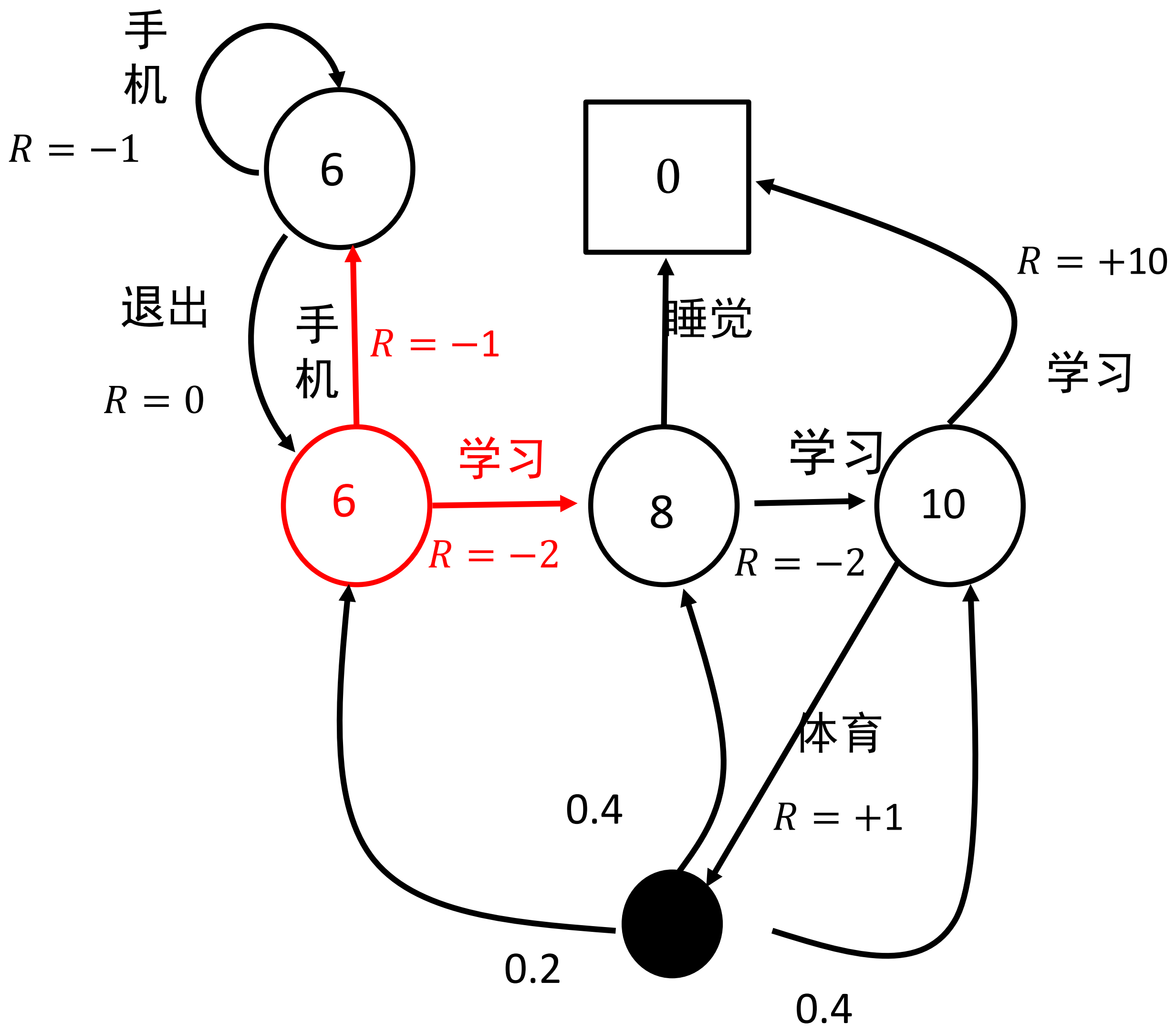
$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

马尔科夫决策过程：贝尔曼最优方程



$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a')$$

马尔科夫决策过程：贝尔曼最优方程



马尔科夫决策过程：求解贝尔曼最优方程

贝尔曼最优方程是非线性的

通常情况下，没有闭式解

通过一些迭代方法求解

- 值迭代
- 策略迭代
- Q learning
- Sarsa

第二次课程作业

1. 阅读 Sutton 书第一章、第二章、第三章，并作读书报告

2. 安装 gym，阅读gym中的代码，写出gym中至少三款游戏的状态、动作、回报、状态转移概率如何设置。并查找一篇近几年的文献，写出文献中马尔科夫决策过程模型。

Python基础

1. print 语句: print()

```
print('hello reinforcement learning')
```

1.1 打印字符串：%s

```
print('My name is %s'%('bao zi xian er'))
```

1.2 打印整数：%d

```
print("I'm %d years old"%(31))
```

1.3 打印浮点数：%f

```
print("I'm %f meters in height"%(1.75))
```

1.4 打印浮点数,并保留两位有效数字：%f

```
print("I'm %.2f meters in height"%1.75)
```

1.5 当然也可以打印中文,中英混合

```
print("老师how萌傻！")
```

2. 条件语句: if…else…

```
score =700
if score >700:
    print("上清华或北大！")
else:
    print("复读")

score = 600
if score >700:
    print("上清华")
elif score>=650:
    print("上其他双一流大学")
elif score > 600 or score==600:
    print("上一本")
else:
    print("复读")
```

3. 循环语句：

3.1. for… in，依次将list或tuple中的每个元素迭代出来

```
a=[1,3,5,7,9]
for i in a:
    if i==1:
        print("10以内的奇数为\n%d"%i)
    else:
        print(i)
```

更多例子：

```
b=["天","地","玄","黄"]
for i in b:
    print(i)
for i in range(100):
    print(i)
```

3.2.While循环，只要条件满足，就一直循环下去

```
i=0
while i<100:
    print(i)
    i+=1
```

continue和break应用

```
while i<100:
    if i<50:
        i+=1
        continue
    print(i)
    i+=1
    if i>80:
        break
```

跳出本次循环，不往下继续执行

结束大循环

Python基础

4. 函数定义

利用 def fun(x) 定义函数，其中fun为定义的函数名，x为参数名。

实例：

```
def step(s, a):
    s_next = s+a*0.01
    return s_next
if __name__=="__main__":
    print(step(2,3))
```

5. 类，面向对象，对象为程序的基本单元。类包括成员变量和成员函数

```
class maze:
    def __init__(self, dt):
        #成员变量用self
        self.dt = dt
    #成员函数
    def step(self,s,a):
        s_next = s+a*self.dt
        return s_next
if __name__=="__main__":
    maze1=maze(dt=0.01)
    s_next=maze1.step(2,3)
    print(s_next)
```

Numpy基础

1. 创建矩阵

```
import numpy as np
from numpy import *
A = np.array([[1,2,3],[2,4,6]])
```

2. numpy创建的矩阵的属性

维数：ndim

```
print("The dimision of A is %d"%A.ndim)
```

形状：shape

```
print("The shape of A is", A.shape)
```

大小：size

```
print("The size of A is %d"%A.size)
```

3. 创建数组

```
B = np.array([1,2,3])
```

将数组变为矩阵（加一个维度）：

```
B=B[np.newaxis,:]
```

4. 创建全零矩阵

```
C = np.zeros((3,3))
print(C)
```

5. 利用np.arange函数创建整数数组

```
D = np.arange(10)
print(D)
```

6. 利用reshape函数改变数据的形状

```
E= D.reshape(2,5)
print(E)
```

Numpy基础

6. 矩阵乘法：

对应元素相乘： `print(A*B)`

矩阵相乘： `print(np.dot(A,b))`

7. 矩阵转置：

```
print(np.transpose(A))
```

8. 矩阵元素的访问

```
print(A[1,1])
print(a1[0,:])
print(a1[0,1:2])
print(a1[0,-1])
```

9. 矩阵的合并

行合并： `print(np.hstack((a1,a2)))`

列合并： `print(np.vstack((a1,a2)))`

10. 常用的函数：sin, cos, exp()

```
print(np.sin(1))
print(np.cos(1))
print(np.exp(1))
```