【零基础友好】从强化学习基本概念到Q-learning算法与代码

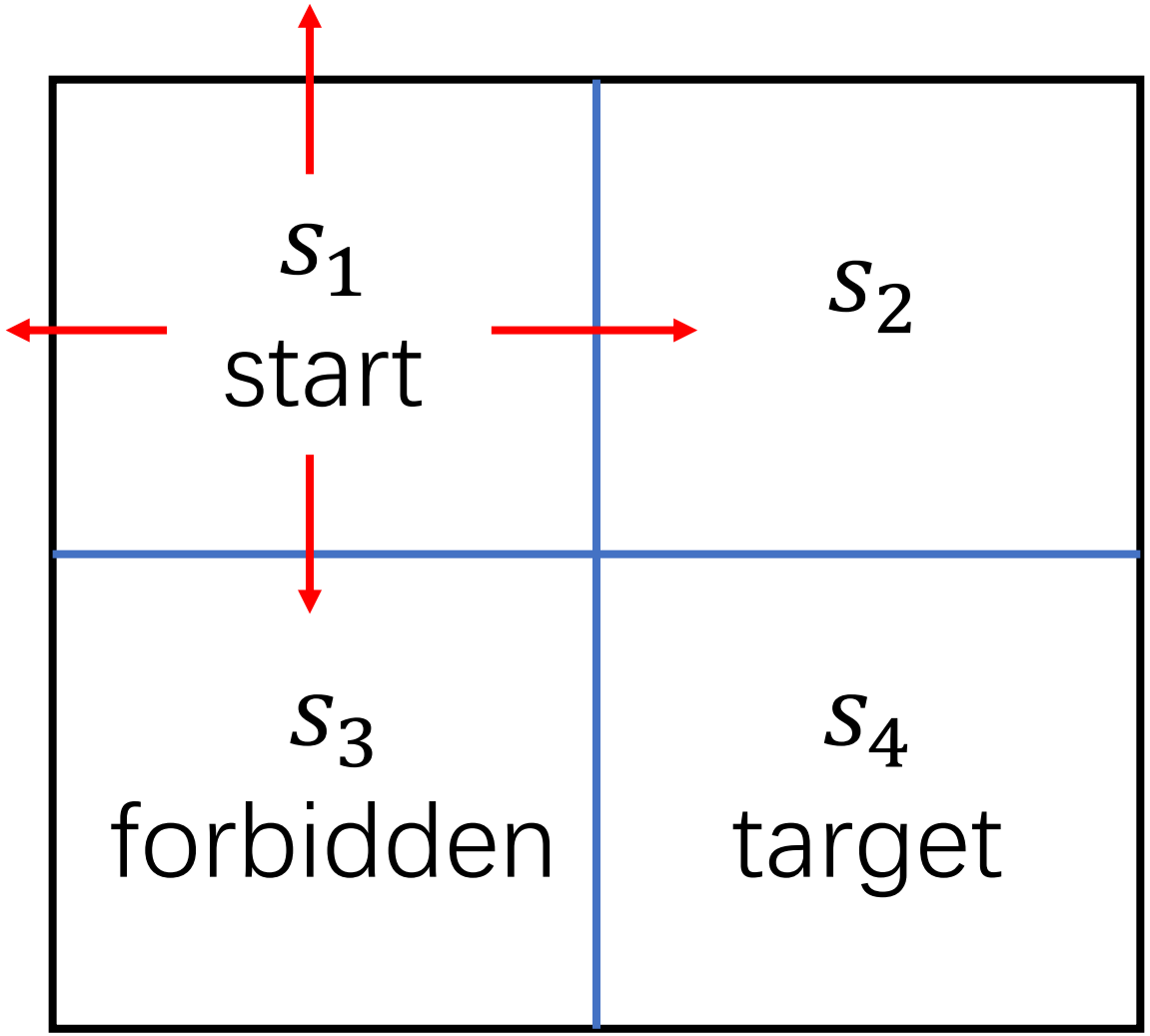
东川路第一可爱猫猫虫

主要内容

感谢粉丝大佬的充电
蕾西亚万岁啊我透
Asph0delus

- 强化学习基础概念
- 贝尔曼公式
- 贝尔曼最优公式
- Q-learning算法
- Q-learning算法的代码

Grid-World Example



- s_1为起点，s_4为终点
- state
  状态
- state space
  状态空间
- action
  行动
- action space of a state
  状态的行动空间
- transition dynamics
  状态转移
- reward function
  奖励函数

基本概念

- 笛卡尔积（卡氏积）

  集合A和B的笛卡尔积为:

  取A的元素为第一个元素、B的元素为第二个元素的有序对组成的集合

- 状态转移函数P

  $S \times A \mapsto \Delta(S)$ $\Delta(S)$表示S上的概率分布

- 奖励函数R

  $S \times A \mapsto \Delta(\mathbb{R})$ $\Delta(\mathbb{R})$表示$\mathbb{R}$上的概率分布

基本概念

- policy策略

  策略是一种条件概率 $\mathrm{S} \rightarrow \Delta(\mathrm{A})$

  $\pi(a_1|s_1) = 0, \pi(a_2|s_1) = 1$

- trajectory轨迹

  state-action-reward chain

  $$S_1 \xrightarrow[r=0]{a_3} S_4 \xrightarrow[r=-1]{a_3} S_7 \xrightarrow[r=0]{a_2} S_8$$

- return回报

  沿着trajectory的reward相加

基本概念

- discount rate折扣率

  $\gamma \in [0,1)$

  衡量当前奖励与未来奖励

- discounted return

  对未来第 t 步的奖励，乘以一个折扣因子 $\boldsymbol{\gamma^t}$

  距离当前越远的reward，权重越低

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

state value与action value

- state value

  $S \to \mathbb{R}$

  从一个state出发，对各个trajectory的return求期望

  $v_\pi(s) = E[G_t|S_t = s]$

  从状态 s 出发，遵循策略 π 时，未来所有回报的期望总和

- action value

  $S \times A \to \mathbb{R}$

  $Q_\pi(s,a) = E[G_t|S_t = s, A_t = a]$

  从状态 s 出发，采取动作 a 后，再遵循策略 π 时，未来所有回报的期望总和

最优策略

- state value可以用来衡量策略的好坏

  如果$v_{\pi_2}(s) > v_{\pi_1}(s)$，那么策略$\pi_2$优于$\pi_1$

  如果$v_{\pi*}(s) > v_{\pi}(s)$对于任意的状态s和策略π都成立

  则称π^*为最优策略

- 最优state value V

  遵循最优策略π^*时，状态s的最大可能价值

- 最优action value Q

  遵循最优策略π^*时，(s,a)的最大可能价值

强化学习

- 强化学习描述的是智能体以最大化累计奖励为目标，在与环境交互的过程中学习到最优策略的过程。
- 这个交互指的是智能体在不同状态依据当前的策略采取不同的行动，完成状态转移，同时环境向智能体反馈即时奖励。
- 智能体根据已知信息更新策略，最终学到最优策略
- 这也是即将介绍的Q-learning算法所遵循的核心思想

贝尔曼公式

- $v_\pi(s) = E[G_t|S_t = s]$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$
$$= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots)$$
$$= R_{t+1} + \gamma G_{t+1}$$

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

贝尔曼公式

- $E[R_{t+1}|S_t = s]$

  $= \Sigma_a \pi(a|s) E[R_{t+1}|S_t = s, A_t = a]$

  $= \Sigma_a \pi(a|s) \Sigma_r p(r|s,a) r$

- $E[G_{t+1}|S_t = s]$

  $= \Sigma_{s'} E[G_{t+1}|S_{t+1} = s'] p(s'|s)$

  $= \Sigma_{s'} v_\pi(s') p(s'|s)$

  $= \Sigma_{s'} v_\pi(s') \Sigma_a p(s'|s,a) \pi(a|s)$

- $v_\pi(s) = \Sigma_a \pi(a|s)[\Sigma_r p(r|s,a) r + \gamma \, \Sigma_{s'} p(s'|s,a) v_\pi(s')]$

  $= \Sigma_a \pi(a|s)[\mathrm{R(s,a)} + \gamma \, \Sigma_{s'} p(s'|s,a) v_\pi(s')]$

贝尔曼最优公式

- $v(s) = \Sigma_a \pi(a|s) Q(s,a)$
- $v*(s) = \Sigma_a \pi*(a|s) Q*(s,a)$
- 我们想要让$v*$最大，只需令$\pi*(a|s) =$

  1，若$a = argmax_a Q*(s,a)$　　0，其他

  $v*(s') = max_{a'} Q*(s',a')$ 最优状态价值是最优动作价值的 “最大值
- $\Sigma_a \pi(a|s)[\mathrm{R(s,a)} + \gamma \Sigma_{s'} p(s'|s,a) v_\pi(s')] = \Sigma_a \pi(a|s) Q(s,a)$

  对比得到：

  $Q(s,a) = R(s,a) + \gamma \Sigma_{s'} p(s'|s,a) v_\pi(s')$

  $Q*(s,a) = R(s,a) + \gamma \Sigma_{s'} p(s'|s,a) v*(s')$

贝尔曼最优公式

- 由$Q*(s,a)=R(s,a)+\gamma\,\Sigma_{s'}p(s'|s,a)v*(s')$

  以及$v*(s')=max_{a'}Q*(s',a')$

  可以得到：$Q*(s,a)=R(s,a)+\gamma\,\Sigma_{s'}p(s'|s,a)\,max_{a'}Q*(s',a')$

  $Q*(s,a)=R(s,a)+\gamma\,max_{a'}Q*(s',a')$
- 这就是在确定性环境下的贝尔曼最优公式

Q-learning算法

- 目标是学习到最优的action value函数$Q*(s,a)$

  本质上是要学习到一个行为s、列为a的表格

  我们先初始化一个Q表，用它来存储对最优Q值的近似，不断更新

- 更新公式：

  新估计值=旧估计值+步长×[目标-旧估计值]

  $$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s,a) + \gamma maxQ(S',a) - Q(s,a))$$

  这里的目标Q值就是我们刚才的贝尔曼最优公式

Q-learning的步骤 

- 初始化Q表 初始化为全0
- 以ε的概率**探索** 从动作空间里随机选择一个动作
  以1-ε的概率**利用** 选择当前状态下Q值最大的动作
- 不断执行动作，获得环境反馈：
  下一个状态s 即时奖励r
- 利用更新公式来更新表格
- 直到下一个状态达到终止条件，开始下一个episode
  比如到达终点、撞墙次数上限、迭代步数上限…
- 直到Q表收敛，算法终止

初始化Q表 初始化为全0

```
def reset_qtable(self):
    """Reset the Q-table."""
    self.qtable = np.zeros((self.state_size, self.action_size))
```

以ε的概率**探索** 以1-ε的概率**利用**

```
class EpsilonGreedy:
    def __init__(self, epsilon):
        self.epsilon = epsilon
    def choose_action(self, action_space, state, qtable):
        """Choose an action `a` in the current world state (s)."""
        explor_exploit_tradeoff = rng.uniform(0, 1)
        if explor_exploit_tradeoff < self.epsilon:
            action = action_space.sample()
        else:
            max_ids = np.where(qtable[state, :] == max(qtable[state, :]))[0]
            action = rng.choice(max_ids)
        return action
```

不断执行动作，获得环境反馈

```
action = explorer.choose_action(
                    action_space=env.action_space, state=state,
  qtable=learner.qtable
)
 all_states.append(state)
 all_actions.append(action)
new_state, reward, terminated, truncated, info = env.step(action)
done = terminated or truncated
learner.qtable[state, action] = learner.update(
                    state, action, reward, new_state
)
```

利用更新公式来更新表格

```
def update(self, state, action, reward, new_state):
    """Update Q(s,a):= Q(s,a) + lr [R(s,a) + gamma * max Q(s',a') - Q(s,a)]"""
    delta = (
        reward
        + self.gamma * np.max(self.qtable[new_state, :])
        - self.qtable[state, action]
    )
    q_update = self.qtable[state, action] + self.learning_rate * delta
    return q_update
```