深度之眼
deepshare.net

强化学习专题:基础知识

导师：Alex

深度之眼
deepshare.net

目录

专题大纲

Syllabus

课程大纲

Syllabus

本专题的大致规划如下：

- 基础知识
- 近几年出现的强化学习方法
  - 集中于Model free，single agent
  - Value-based(DQN及变种，C51，Rainbow)
  - Policy-based(REINFORCE，PPO)
  - 其他一些改进(TD3，SAC)
- 有趣的应用
  - 待定

深度之眼
deepshare.net

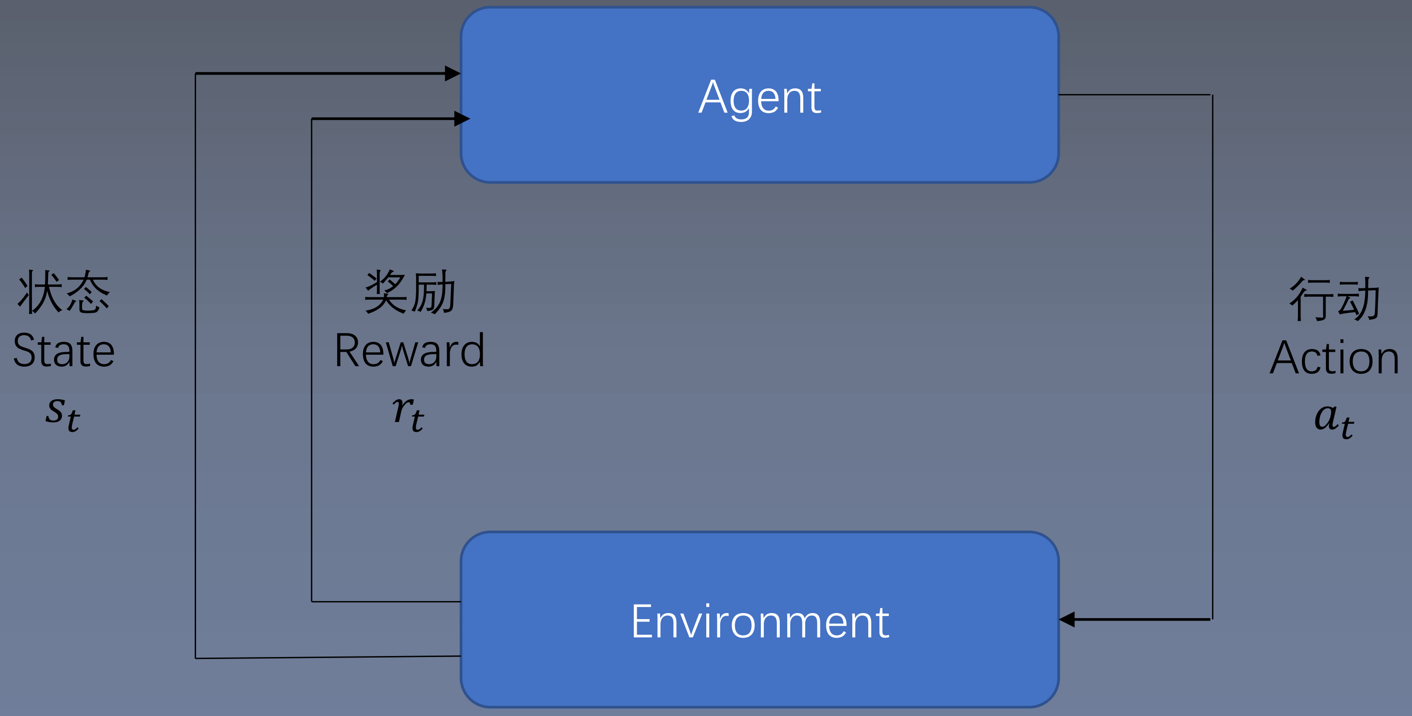
基础元素

Basic Elements

三大元素

Three Elements

- 环境(environment)
- 代理人(agent)
- 奖励(reward)



- 为什么奖励被单独当成一个元素？

因为有时我们需要自己定义奖励函数(Reward Function)，而不是环境直接反馈

一些例子

Examples

深度之眼
deepshare.net

环境(Env)	代理人(Agent)	行动(Action)	状态(State)	奖励(Reward)
Dota/LoL/王者				
围棋				
篮球				
自动驾驶				

一些例子

Examples

深度之眼 deepshare.net

环境(Env)	代理人(Agent)	行动(Action)	状态(State)	奖励(Reward)
Dota/LoL/王者	玩家			
围棋	两名选手			
篮球	所有球员，教练			
自动驾驶	车辆AI			

一些例子

Examples

环境(Env)	代理人(Agent)	行动(Action)	状态(State)	奖励(Reward)
Dota/LoL/王者	玩家	所有操作		
围棋	两名选手	落子		
篮球	所有球员，教练	跑位，战术，动作		
自动驾驶	车辆AI	转向，加速，减速		

一些例子

Examples

环境(Env)	代理人(Agent)	行动(Action)	状态(State)	奖励(Reward)
Dota/LoL/王者	玩家	所有操作	屏幕信息	
围棋	两名选手	落子	棋盘	
篮球	所有球员，教练	跑位，战术，动作	场上信息	
自动驾驶	车辆AI	转向，加速，减速	周围环境	

一些例子

Examples

环境(Env)	代理人(Agent)	行动(Action)	状态(State)	奖励(Reward)
Dota/LoL/王者	玩家	所有操作	屏幕信息	金钱，经验，胜负
围棋	两名选手	落子	棋盘	胜负
篮球	所有球员，教练	跑位，战术，动作	场上信息	数据，得分，胜负
自动驾驶	车辆AI	转向，加速，减速	周围环境	？？？

深度之眼
deepshare.net

有限马尔科夫决策过程

Finite Markov Decision Process

数学符号

Mathematical Symbols

深度之眼 deepshare.net

- 大写字母：随机变量
- 小写字母：具体的值
- 花写字母：随机变量的空间

- 具体表示：
  - State : $S_t \in \mathcal{S}$
  - Action : $A_t \in \mathcal{A}$
  - Reward : R_t

定义

Definition

一般的马尔科夫决策过程(Markov Decision Process, MDP)可以描述为一串序列 $s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2 \ldots \ldots$

$$p(S_{t+1}=s_{t+1}, R_{t+1}=r_{t+1} | all\ history) = p(S_{t+1}=s_{t+1}, R_{t+1}=r_{t+1} | S_t=s_t, A_t=a_t)$$

有限马尔科夫决策过程则限制当$t=T$时，交互终止

基本元素

Basic Elements

- 目标(goal)： 最大化获得总奖励(reward)的期望值
- 回报(return)： 某一时刻起未来的总奖励
  - $G_t = \sum_{k=0}^{T-t-1} R_{t+1+k}$
  - $G_t = \sum_{k=0}^{T-t-1} \gamma^{\mathrm{k}} R_{t+1+k}$
  - $G_t = R_{t+1} + \gamma G_{t+1}$
- 轮(episode)：每一段独立的agent与environment的交互

深度之眼
deepshare.net

策略估计和提升

Policy Evaluation and Improvement

定义

Definition

深度之眼 deepshare.net

- 策略(policy)：即agent选择action的方式
- 策略函数(policy function)：
  - state空间到action空间上的分布的映射
  - $\pi : \mathcal{S} \mapsto \mathcal{A}$
  - $A_t \sim \pi(S_t)$
- 价值函数(value function)
  - $v_\pi(s_t) = \mathbb{E}[G_t | S_t = s_t]$
- 行动-价值函数(action-value function)
  - $q_\pi(s_t, a_t) = \mathbb{E}[G_t | S_t = s_t, A_t = a_t]$
- 思考：如何用v_π，q_π 互相表示对方？
  - 可以使用$\pi(a|s)$和$p(s', r|s, a)$，这也被成为env的dynamics

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a)$$

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

最优性

Optimality

- 最优策略函数(optimal policy function)
  - 使得期望回报最大的策略，π_*
- 最优价值函数
  - 最优(optimal)价值函数 $v_*(s_t) = \max_{\pi} v_\pi(s_t)$
- 最优行动-价值函数
  - 最优行动-价值函数$q_*(s_t, a) = \max_{\pi} q_\pi(s_t, a)$
- 思考：如何用v_*，q_* 互相表示对方？
- 思考：能否用递归形式写出v_*，q_*满足的式子？
  - 可以使用$p(s', r|s, a)$
  - 会得到bellmen equation
  - 可以根据这个式子计算具体的v_*，q_*

策略估计

Policy Evaluation

简化问题：假设env只有有限个状态$s_1\ s_2 \cdots s_n$，并且对于任意$k \in \{1, 2, \dots, n\}$我们有

$$v_\pi(s_k) = \sum_{i=1}^{n} p_{k,i}\, v_\pi(s_i) + c_k, \sum_{i=1}^{n} p_{k,i} = 1$$

如何求出所有的$v_\pi(s_k)$?

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a)$$

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

- 方法一：解方程
- 方法二：动态规划

策略估计

Policy Evaluation

- 如果我们把所有的$v_\pi(s_k)$按顺序写入一个列向量v，c_k写入c，我们就有$v = \gamma P v + c$，$P_{ij} = p_{i,j}$
- 给v一个非0初始向量值v_0
- 不断计算$v_{k+1} = \gamma P v_k + c$，直到收敛

```
import numpy as np

## P = np.array(...)
## c = np.array(...)
## gamma = 0.99
## Assume we know P and c

v = np.random.random((P.shape[1], 1))

convergence = False
while not convergence:
    v_new = gamma * P.dot(v) + c
    if #convergent condition here :
        convergent = True
    v = v_new.copy()
```

策略提升

Policy Improvement

定理：对于确定性的策略π和π'，如果对于所有的$s \in \mathcal{S}$，都有$q_\pi(s, \pi'(s)) \geq v_\pi(s)$，那么对于所有的$s \in \mathcal{S}$我们有$v_{\pi'}(s) \geq v_\pi(s)$

思考：根据这个结论，我们如何根据已经获得的策略，得到更好的策略？

答：贪心(greedy)策略即可，

$$\pi'(s) = \operatorname*{argmax}_a q_\pi(s, a)$$

$$
\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \mid S_t = s\right] \\
&\ \ \vdots \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s\right] \\
&= v_{\pi'}(s).
\end{aligned}
$$

策略迭代

Policy Iteration

深度之眼 deepshare.net

根据目前已经掌握的知识，我们可以根据以下策略寻找到最优策略：

- 初始化：随机生成一个策略π_0
- 反复进行接下来的三个步骤：
    - 策略估计（参照之前的PPT），得到v_{π_k}
    - 策略提升（参照之前的PPT），得到π_{k+1}
    - 判断收敛：如果$\pi_{k+1} = \pi_k$则跳出循环
- 最后得到的策略即为最优策略

局限性

Limits

我们之前提到的方法有很明显的局限性：

- 只讨论了决定性的策略
- State和Action的数量都不能太大
- 需要知道env的dynamics
- 即便满足以上两个条件，仍然有过多的计算量

接下来的课程中，我们将进一步讨论这些问题的解决方案，并进入现代的深度强化学习领域

结语

结 语

我们的课程才刚刚开始，还希望同学们打好基础，这样才能更好地理解后面的更复杂的思想和算法