rl_入门提要

概念

增强学习

智能体（Agent）和环境（Enviroment）不断交互，通过观察状态（State）的改变，以及获得的奖励（Reward）来迭代优化动作控制的策略（Policy）。

在某个状态s下根据策略π决定采取什么动作a。

$a = \pi(s)$ 或者 $\pi(a|s)$

时刻t的期望回报：

$G_t = R_{t+1} + \lambda R_{t+2} + \ldots = \sum_{k=0}^{\infty} \lambda^k R_{t+k+1}$

λ是折扣因子，表示越近的反馈越重要。

value function： 状态未来的潜在价值

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

那么，通过估计每一个状态的$V(S)$值即可确s_t状态下应该采取什么样的动作了（例如π可简单的选择对应$V(s_{t+1})$最大的动作a_t）。

Bellman方程

把v的定义展开：

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

$= E[R_{t+1} + \lambda R_{t+2} + \lambda^2 R_{t+3} + \ldots |S_t = s]$

$= E[R_{t+1} + \lambda(R_{t+2} + \lambda R_{t+3} + \ldots)|S_t = s]$

$= E[R_{t+1} + v(S_{t+1})|S_t = s]$

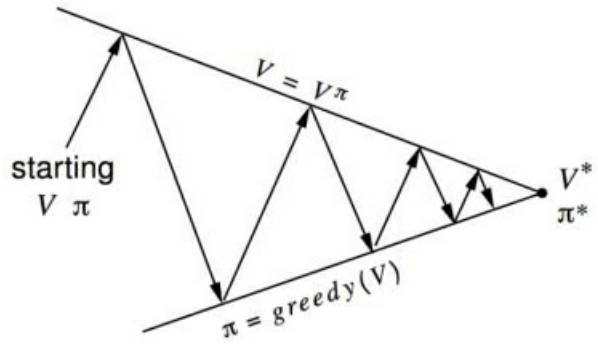
> 表明： Value Function可以通过迭代来进行计算。

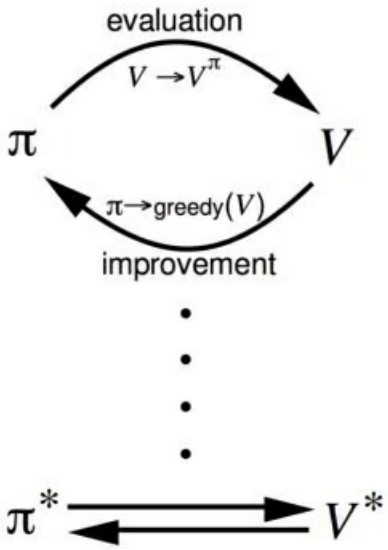
概念方法

策略迭代

两个阶段：

1. Policy Evaluation， 根据当前策略产生新样本，并迭代更新Value Function。
2. Policy Improvement，根据当前value function改进策略





Policy evaluation Estimate v_π
Any policy evaluation algorithm
Policy improvement Generate $\pi' \geq \pi$
Any policy improvement algorithm

具体算法

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\left[r + \gamma V(s')\right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $a \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
   If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return V and π; else go to 2

:

价值迭代（Q-learning，DQN等）

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
$\Delta \leftarrow 0$
For each $s \in \mathcal{S}$:
$v \leftarrow V(s)$
$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$

直接通过采样，迭代更新V(s)直至收敛，方法更加直观。

上述两个方法最大的问题在于对于概率分布 $p(s', r|s, a)$ 有依赖，而一般情况下是无法得知的。

Q-learning

Q-learning也是基于value iteration的方法，但是不要求每次迭代都能够遍历所有状态以获得 $p(s', r|s, a)$ ，通过有限的系列样本就能进行迭代。

Action-Value function 动作价值函数

$Q^\pi(s, a)$： 状态s下执行动作a的reward；作为对比，V(s)是状态s对应的多种动作a的reward的期望。

$Q^\pi(s, a) = E[r_{t+1} + \lambda r_{t+2} + \lambda^2 r_{t+3} + \ldots | s, a]$

$= E_{s'}[r + \lambda Q^\pi(s', a') | s, a]$

最优动作价值函数：

$Q^*(s, a) = \max_\pi Q^\pi(s, a)$

Q^*递推式：

$Q^\star(s, a) = E_{s'}[r + \lambda \max_{a'} Q^\star(s', a') | s, a]$

Q-learning Q值更新方法：

$Q_{k+1}(S_t, A_t) \leftarrow Q_k(S_t, A_t) + \alpha(R_{t+1} + \lambda max_a Q_k(S_{t+1}, a) - Q_k(S_t, A_t))$

α可以理解为学习率，$R_{t+1} + \lambda max_a Q_k(S_{t+1}, a) - Q_k(S_t, A_t)$ 可以认为是根据新采样的Reward对Q值估计误差的评估。

算法描述：

初始化$Q(s,a), \forall s \in S, a \in A(s)$,任意的数值，并且$Q(terminal-state, \cdot) = 0$

重复（对每一节episode）：

初始化 状态S

重复（对episode中的每一步）：

使用某一个policy比如（$\epsilon - greedy$)根据状态S选取一个动作执行

执行完动作后，观察reward和新的状态S'

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \lambda \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$

$S \leftarrow S'$

循环直到S终止

其中，$\epsilon - greedy$ **采样策略**：ϵ 表示一个很小的概率进行随机动作采样（探索），否则按照最优策略采取动作（利用）。它是一种简单粗暴的探索方法，对于复杂任务，这种探索未知空间的方式是不可取的，需要更加复杂的方式来改进。