

强化学习数学原理

第一章基本概念

- grid-world example : 一个机器人走网格的经典例子, 机器人尽量避免进入forbidden grid、尽量减少拐弯、不要走出边界、.....
- state: 状态, 表示为一个节点, 在grid-world中可以表示为一个格子 (也可以添加其他信息到状态, 如速度等)
- state space: 状态空间, 所有状态的集合。
- action: 行动, 能够使得状态变化的动作。(如向上/下/左/右移动, 等)
- action space: 行动的集合, 通常依赖于当前的状态。
- state transition: 状态转移, 从一个状态移动到另一个状态。
 $s_5 \xrightarrow{a_1} s_6$ 表示从状态 s_5 经过动作 a_1 到达状态 s_6
- state transition probability: 状态转移的条件概率。(例如:
 $p(s_2|s_1, a_2) = 0.8$ 代表在状态 s_1 , 行动 a_2 下, s_2 的概率是0.8)
- Policy: 策略, 用箭头来表示。表示在某个状态更倾向于走哪个action
 $\pi(a_1|s_1) = 0, \pi(a_2|s_1) = 1, \pi(a_3|s_1) = 0, \pi(a_4|s_1) = 0$ 表示在状态 s_1 有1的概率进行行动 a_2 。显然 $\sum_{i=1}^k \pi(a_i|s_1) = 1$
- reward: 他是一个实数, 代表我们的奖励, 如果 $reward > 0$, 则代表希望它发生, $reward < 0$ 则表示不希望它发生。
例如我们可以将“尝试逃出边界的时候, 我们设 $r_{bound} = -1$, 将到达目的地设为 $r_{target} = 1$
因此我们可以通过设计reward来实现到达目的地。
 $p(r = -1|s_1, a_1) = 1, p(r \neq -1|s_1, a_1) = 0$ 表示在状态 s_1 进行 a_1 得到-1的reward的概率是1, 得到不是-1的reward的概率是0
- trajectory: 一个由state、action、reward连接成的链。
- return: 一个trajectory中所有的reward的总和。通过比较return来评估策略是好是坏
- Discounted rate : $\gamma \in [0, 1)$ 。 $discountedreturn = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$,

γ 通常表示是否更看重未来, γ 越小, 则越看重现在。

- Episode: 能够到达terminal states(停止状态) 的trajectory。一个Episode也叫一个Episode task与之对应的是continuing task (指永无止境的任务)。

Markov decision process (MDP) ∴

- 集合:
 - State: 状态集合
 - Action: 对于每个状态s的行动集合 $A(s)$
 - Reward: 奖励集合 $R(s, a)$
- 概率要素(probability distribution):
 - State transition probability: $p(s'|s, a)$ 在状态s下, 进行行动a, 到达另一个状态 s' 的概率。
 - Reward probability: $p(r|s, a)$ 在状态s下, 进行行动a, 得到r的奖励的概率。
 - Policy: $\pi(a|s)$ 在状态s下, 进行行动a的概率。

MDP的独特性质 (markov property): 与历史无关

$$p(s_{t+1}|a_{t+1}s_t \dots a_1s_0) = p(s_{t+1}|a_{t+1}, s_t)$$

$$p(r_{t+1}|a_{t+1}s_t \dots a_1s_0) = p(r_{t+1}|a_{t+1}, s_t)$$

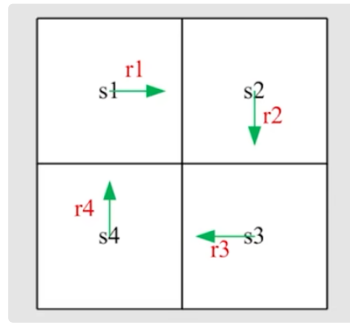
第二章 贝尔曼公式

return ∴

为什么return很重要? 因为return评估的策略的好坏。

如何计算return? 用 v_i 表示从 s_i 出发得到的return。

以下面的状态图为例:



- 那么根据定义有

$$v_1 = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$$

$$v_2 = r_2 + \gamma r_3 + \gamma^2 r_4 + \dots$$

$$v_3 = r_3 + \gamma r_4 + \gamma^2 r_1 + \dots$$

$$v_4 = r_4 + \gamma r_1 + \gamma^2 r_2 + \dots$$

- 也可以递推得到(Booststrapping):一个状态依赖于其他状态

$$v_1 = r_1 + \gamma(r_2 + \gamma r_3 + \dots) = r_1 + \gamma v_2$$

$$v_2 = r_2 + \gamma(r_3 + \gamma r_4 + \dots) = r_2 + \gamma v_3$$

$$v_3 = r_3 + \gamma(r_4 + \gamma r_1 + \dots) = r_3 + \gamma v_4$$

$$v_4 = r_4 + \gamma(r_1 + \gamma r_2 + \dots) = r_4 + \gamma v_1$$

然后就有 $v = r + \gamma P * v$, 这里的 v, r 是向量, P 是变换矩阵。于是就能求解得出 v 的值。

state value

$S_t \xrightarrow{A_t} R_{t+1}, S_{t+1}$, 指在 S_t 下经过 A_t 得到 (R_{t+1}, S_{t+1})

state value: $v_\pi(s) = \mathbb{E}[G_t | S_t = s]$ 当前状态为 s 的时候所有的 return 的期望值。

考虑下面的 trajectory: $S_t \xrightarrow{A_t} R_{t+1}, S_{t+1} \xrightarrow{A_{t+1}} R_{t+2}, S_{t+2} \xrightarrow{A_{t+2}} R_{t+3}, S_{t+3} \dots$

那么 $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = R_{t+1} + \gamma G_{t+1}$

则有

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[G_t | S_t = s] \\ &= \mathbb{E}[R_t | S_t = s] + \mathbb{E}[G_{t+1} | S_t = s] \end{aligned}$$

公式中的第一项

$$\mathbb{E}[R_{t+1} | S_t = s] = \sum_a [\pi(a|s) \sum_r p(r|s, a) r]$$

公式中的第二项

$$\begin{aligned}
\mathbb{E}[G_{t+1}|S_t = s] &= \sum_{s'} \{\mathbb{E}[G_{t+1}|S_t = s, S_{t+1} = s']p(s'|s)\} \\
&= \sum_{s'} \mathbb{E}[G_{t+1}|S_{t+1} = s']p(s'|s) \\
&= \sum_{s'} \{v_\pi(s')p(s'|s)\} \\
&= \sum_{s'} \{v_\pi(s') \sum_a [p(s'|s, a)\pi(a|s)]\}
\end{aligned}$$

- 于是，贝尔曼公式：

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}[R_t|S_t = s] + \mathbb{E}[G_{t+1}|S_t = s] \\
&= \sum_a \pi(a|s) [\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_\pi(s')]
\end{aligned}$$

- 之后我们列出每个 s_i 对应的 $v_\pi(s_i)$ 的公式，然后求解方程组即可得到每个状态的 state value。

贝尔曼公式的矩阵/向量形式

- 由贝尔曼公式： $v_\pi(s) = \sum_a \pi(a|s) [\sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_\pi(s')]$

可以简略写为 $v_\pi(s) = r_\pi(s) + \gamma \sum_{s'} p_\pi(s'|s)v_\pi(s')$ ，(这里有

$$\begin{aligned}
r_\pi(s) &\iff \sum_a \pi(a|s) \sum_r p(r|s, a)r \\
p_\pi(s'|s) &\iff \sum_a \pi(a|s) p(s'|s, a)
\end{aligned}$$

对 s 进行标号得出 $v_\pi(s_i) = r_\pi(s_i) + \gamma \sum_{s_j} p_\pi(s_j|s_i)v_\pi(s_j)$

于是化为矩阵向量形式： $v_\pi = r_\pi + \gamma P_\pi v_\pi$ ，这里的 v_π, r_π 均为向量， P_π 为状态转移矩阵 ($[P_\pi]_{i,j} = p_\pi(s_j|s_i)$)

- 求解贝尔曼公式，进而得到 state value 是评判策略好坏 (policy evaluation) 的关键。

- 求解贝尔曼公式：

- $v_\pi = (I - \gamma P_\pi)^{-1} r_\pi$ ，很简洁的公式，但是求逆太难算了。

- 迭代求法： $v_{k+1} = r_\pi + \gamma P_\pi v_k$

- 当 $k \rightarrow \infty$ 时，有 $v_k = v_\pi$

action value

从一个状态出发， 选择了一个action之后， 得到的return的期望。

在做判断时， 根据Action value的大小来判断选择哪个Action。

求解状态s下进行行动a的action value($q_{\pi}(s, a)$)

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

计算action value:

1. 根据state value求出
2. 直接计算action value

第三章 贝尔曼最优公式

- 直观上地说， 选择action value比较大的action， 将他设置为1， 其他设置为0。不断如此迭代， 就可以找到最优的策略。
- 严格证明则需要贝尔曼最优公式。

如果对于所有的s， 都有 $v_{\pi_1}(s) \geq v_{\pi_2}(s)$ for all $s \in S$,那么说 π_1 是比 π_2 要好的。

1. 问题一： 这样最优的策略是否存在？
2. 问题二： 这个最优的策略是唯一的吗？
3. 问题三： 策略是stochastic还是deterministic？
4. 问题四： 如何找到这么一个策略？

贝尔曼最优公式

- 贝尔曼最优公式堂堂登场！

$$v(s) = \max_a \left(\sum_a (\pi(a|s)q(s, a)) \right), s \in S$$

可以发现， 假设当 $a = a'$ 时， $q(s, a)$ 最大， 那么令 $\pi(a|s) = \begin{cases} 1 & a = a' \\ 0 & a \neq a' \end{cases}$,就会得到最大的 $v(s)$ 。

即 $v(s) = \max_a \left(\sum_a (\pi(a|s)q(s, a)) \right) = \max_{a \in A(s)} (q(s, a))$

- 将公式变为矩阵-向量形式

$$v = \max_{\pi} (r_{\pi} + \gamma P_{\pi} v)$$

我们设一个映射 $f(v) := \max_{\pi} (r_{\pi} + \gamma P_{\pi} v)$,那么原式就可以化为 $v = f(v)$

一些概念

FixedPoint 不动点: 对于映射 $f: X \rightarrow X$, 存在 $x \in X, f(x) = x$, 那么 x 是不动点。

Contraction mapping : 在映射后两点的距离更小。

$\|f(x_1) - f(x_2)\| = \gamma \|x_1 - x_2\|, \gamma < 1$ 。 (例如 $f(x) = 0.5x$ 就是一个 contraction mapping)

contraction Theorem

如果 f 是一个 contraction mapping。那么一定有

1. 存在一个 x^* , 满足 $f(x^*) = x^*$, 即 x^* 是一个 FixedPoint
2. 这样的 x^* 一定有且只有一个
3. 可以通过迭代算法求出这个 x^* : $x_{k+1} = f(x_k)$, 当 $k \rightarrow \infty$ 时, 有 $x_k \rightarrow x^*$

例如 $f(x) = 0.5x$,那么 $f(0) = 0, x^* = 0$,给出任意 x , 在不断进行 $x = 0.5x$ 迭代后, 会收敛于0

求解贝尔曼最优公式

可以证明在贝尔曼最优公式中 $f(v) = \max_{\pi} (r_{\pi} + \gamma P_{\pi} v)$ 是一个 contraction mapping, 那么 $v = f(v)$ 。于是就可以通过迭代算法来求解出来。

假设 v^* 是贝尔曼最优公式的解, 即是他的不动点。即 $v^* = \max_{\pi} (r_{\pi} + \gamma P_{\pi} v^*)$

所以就可以利用 contraction Theorem 中的迭代算法来求得 v^*

所以贝尔曼最优公式就是特殊的贝尔曼公式。

第四章 Value iteration & Policy iteration

Value iteration algorithm(值迭代算法) ::

值迭代算法就是根据贝尔曼最优公式来迭代求解优化问题。

$$v_{k+1} = f(v_k) = \max_{\pi} (r_{\pi} + \gamma P_{\pi} v_k)$$

求解步骤：最开始生成一个任意的状态 v_0 ,不断循环以下两步

1. policy update更新策略: $\pi_{k+1} = \underset{\pi}{argmax} (r_{\pi} + \gamma P_{\pi} v_k)$
2. value update 更新值: $v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k$

需要注意的是 v_k 只是一个值, 并不是一个state value。

不断迭代直到 $v_k - v_{k-1}$ 足够小就认为已经收敛了。

Policy iteration algorithm(策略迭代算法) ::

最开始生成一个任意策略 π_0

1. policy evaluation(PE): $v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$
2. policy improvement(PI): $\pi_{k+1} = \underset{\pi}{argmax} (r_{\pi} + \gamma P_{\pi} v_{\pi_k})$

整体过程即 $\pi_0 \xrightarrow{PE} v_{\pi_0} \xrightarrow{PI} \pi_1 \xrightarrow{PE} v_{\pi_1} \xrightarrow{PI} \pi_2 \xrightarrow{PE} v_{\pi_2} \xrightarrow{PI} \pi_3 \dots$

- 几个核心问题：
 1. 在policy evaluation中如何求解 state value?
 2. 为什么进行PI后, π_{k+1} 比 π_k 更优?
 3. 为什么最终能找到最优解?
 4. Policy iteration和Value iteration 有什么关系?
- Q1: 有两种方法(即求解贝尔曼公式的两种方法):
 1. closed-form solution : $v_{\pi_k} = (I - \gamma P_{\pi_k})^{-1} r_{\pi_k}$
 2. iterative solution: $v_{\pi_k}^{j+1} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j)}, j = 0, 1, 2, \dots$
- Q2: $\pi_{k+1} = \underset{\pi}{argmax}(r_{\pi} + \gamma P_{\pi} v_{\pi_k})$, 因为 π_{k+1} 一定比 π_k 要更大
- Q3: $v_{\pi_0} \leq v_{\pi_1} \leq v_{\pi_2} \leq \dots \leq v_{\pi_k} \leq v^*$
- Q4: 二者是两个极端

truncated policy iteration algorithm ∴

他是值迭代算法和策略迭代算法的推广，值迭代算法和策略迭代算法是 truncated policy iteration algorithm的极端情况。

Policy iteration: $\pi_0 \xrightarrow{PE} v_{\pi_0} \xrightarrow{PI} \pi_1 \xrightarrow{PE} v_{\pi_1} \xrightarrow{PI} \pi_2 \xrightarrow{PE} v_{\pi_2} \xrightarrow{PI} \pi_3 \dots$

Value iteration: $u_0 \xrightarrow{PU} \pi'_1 \xrightarrow{VU} u_1 \xrightarrow{PU} \pi'_2 \xrightarrow{VU} u_2 \dots$

Policy Iteration algorithm		Value iteration algorithm	Comments
1) Policy:	π_0	N/A	
2) Value:	$v_{\pi_0} = r_{\pi_0} + \gamma P_{\pi_0} v_{\pi_0}$	$v_0 := v_{\pi_0}$	
3) Policy:	$\pi_1 = \underset{\pi}{argmax}(r_{\pi} + \gamma P_{\pi} v_{\pi_0})$	$\pi_1 = \underset{\pi}{argmax}(r_{\pi} + \gamma P_{\pi} v_0)$	两个算法的第一步Policy是相同的。
4) Value:	$v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$	$v_1 = r_{\pi_1} + \gamma P_{\pi_1} v_0$	两个算法求 v_{π} 的方法是不一样的
5) Policy:	$\pi_2 = \underset{\pi}{argmax}(r_{\pi} + \gamma P_{\pi} v_{\pi_1})$	$\pi'_2 = \underset{\pi}{argmax}(r_{\pi} + \gamma P_{\pi} v_1)$	
...

区别在于计算 v_{π_1} 的时候是使用贝尔曼公式求，还是直接继承上一步的求法。

考虑公式 $v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$

$$\begin{aligned}
v_{\pi_1}^{(0)} &= v_0 \\
v_{\pi_1}^{(1)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(0)} && \rightarrow \text{value iteration } v_1 \\
v_{\pi_1}^{(2)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(1)} \\
&\dots \\
v_{\pi_1}^{(j)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(j-1)} && \rightarrow \text{truncated policy iteration } \bar{v}_1 \\
&\dots \\
v_{\pi_1}^{(\infty)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(\infty)} && \rightarrow \text{policy iteration } v_{\pi_1}
\end{aligned}$$

可以发现，value iteration就是在得到第一个 v 后就进行下一步操作；policy iteration则是不断你迭代直到收敛。那么 truncated policy iteration则是二者的结合，选择在中间的某一步停下。

第五章 MonteCarlo learning

蒙特卡洛方法是一个model-free RL的方法。（前面讲的算法都是model-based RL方法）

抛硬币例子

假设抛硬币问题：抛一个硬币，正面价值为1，反面为-1，期望是多少？

那么 model-based方法：直接计算数学期望

$$\mathbb{E}[X] = \sum_x xp(x) = 1 * 0.5 + (-1) * 0.5 = 0$$

结果很精确，但是通常很难找到这样的数学模型。

model-free方法：做实验，随机扔硬币，然后统计值，最终可以得到近似值。

一个简单的MC-based RL算法

（我们称这个算法为MC-basic算法）

可以通过改变Policy iteration算法来变成model-free 算法。

1. policy evaluation(PE): $v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$
2. policy improvement(PI): $\pi_{k+1} = \underset{\pi}{argmax} (r_{\pi} + \gamma P_{\pi} v_{\pi_k})$

$$\pi_{k+1}(s) = \underset{\pi}{argmax} \sum_a \pi(a|s) q_{\pi_k}(s, a)$$

关键在于计算 $q_{\pi_k}(s, a)$ ，两种方法：

1. 需要模型: $q_{\pi_k}(s, a) = \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_{\pi_k}(s')$

2. 不需要模型: $q_{\pi_k}(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$

基于蒙特卡洛的model即通过大量采样来估计 G_t

MC exploring Starts

遵循策略 π ，我们会得到一个episode如下：

$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots$

定义Visit：一个episode中访问的 $(state, action)$ 对的数量。

在MC-basic方法中，使用的是Initial-visit method，即只考虑 $s_1 \xrightarrow{a_2}$ 这一个 $(state, action)$ 对。这导致了没有充分利用了整个episode。

那么对于一个episode:

$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4}$	$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3}$	$s_5 \xrightarrow{a_1} \dots$	$[original\ episode]$
$s_2 \xrightarrow{a_4}$	$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3}$	$s_5 \xrightarrow{a_1}$	$\dots [episode\ starting\ from(s_2, a_4)]$
	$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3}$	$s_5 \xrightarrow{a_1}$	$\dots [episode\ starting\ from(s_1, a_2)]$
	$s_2 \xrightarrow{a_3}$	$s_5 \xrightarrow{a_1}$	$\dots [episode\ starting\ from(s_2, a_3)]$
		$s_5 \xrightarrow{a_1}$	$\dots [episode\ starting\ from(s_5, a_1)]$

因此我们就可以通过这一个episode来估计

$(s_1, a_2), (s_2, a_4), (s_1, a_2), (s_2, a_3), (s_5, a_1), \dots$ 的action value。而不是仅仅用于 (s_1, a_2) 。

- first-visit：指在遇到相同的 $(state, action)$ 时，只使用第一次遇到的。
- every-visit：指在遇到相同的 $(state, action)$ 时，每个都做考虑，最后综合起来。

generalized policy iteration(广义策略迭代)：指并不是精确求解的代码，使用迭代来得到策略，像truncated policy iteration algorithm和 MC都属于 generalized policy iteration。

soft policies

因为我们从一个 $(state, action)$ 出发能够到达多个状态，所以我们也就没必要把所有的 $(state, action)$ 都设置为出发点了。

那么如何选择出发点？

ϵ - greedy policies :

$$\pi(a|s) = \begin{cases} 1 - \frac{\epsilon}{|A(s)|} (|A(s)| - 1) & \text{for the greedy action} \\ \frac{\epsilon}{|A(s)|} & \text{for the other } |A(s)| - 1 \text{ actions} \end{cases}$$

这里的greedy action指的就是 $q_{\pi}(s, a^*)$ 最大的那个action。(通常很小)，这样在保证greedy action被选择的概率较大的情况下，其他的action同样有一些概率被选择。

- ϵ - greedy policies能够平衡exploitation和exploration

exploitation: 指的是充分利用value，贪心于当前。

exploration: 指的是探索当前非最佳的情况，可能会找到未来更优的情况。

这样选择一个(state,action)作为出发点，就可以通过exploration来得到所有的(state,action)的策略。

MC ϵ -Greedy algorithm

对于之前的方法，只会选择最优的action，即 a^* 。

$$\pi_{k+1}(s) = \arg \max_{\pi \in \Pi_{\epsilon}} \sum_a \pi(a|s) q_{\pi_k}(s, a)$$

那么对于MC ϵ -Greedy algorithm

$$\pi(a|s) = \begin{cases} 1 - \frac{\epsilon}{|A(s)|} (|A(s)| - 1) & a = a_k^* \\ \frac{\epsilon}{|A(s)|} & a \neq a_k^* \end{cases}$$

便是给了其他action一个较小的 $\frac{\epsilon}{|A(s)|}$

ϵ -Greedy algorithm 中的 ϵ 较大的时候，探索性很强，但是最优性比较差。

我们可以通过起初设置较大的 ϵ ，然后逐渐减小他来平衡探索性和最优性。

第六章 Stochastic Approximation & Stochastic Gradient Descent

Stochastic Approximation (随机近似理论) 和 Stochastic Gradient Descent (随机梯度下降)

Motivating example

mean estimation problem:

1. 考虑有一个随机变量 X
2. 目标是计算期望 $\mathbb{E}[X]$
3. 假设我们有 N 个采样 $\{x_i\}_{i=1}^N$
4. 那么期望可以被估计为 $\mathbb{E}[X] \approx \bar{x} := \frac{1}{N} \sum_{i=1}^N x_i$
5. 当 $N \rightarrow \infty$, $\bar{x} \rightarrow \mathbb{E}[X]$

怎么计算 *mean* \bar{x} ?

方法一： 计算所有的总和，然后除以 N

方法二 (iterative mean estimation): 实时估计 \bar{x} ，当出现新的 x_i 时，更新 \bar{x}

我们规定 w_{k+1} 表示前 k 个 x 的均值。即 $w_{k+1} = \frac{1}{k} \sum_{i=1}^k x_i$ 。 (一般设置 $w_1 = x_1$)

那么根据如下公式

$$\begin{aligned} w_{k+1} &= \frac{1}{k} \sum_{i=1}^k x_i = \frac{1}{k} \left(\frac{1}{k-1} \sum_{i=1}^{k-1} x_i + x_k \right) \\ &= \frac{1}{k} ((k-1)w_k + x_k) = w_k - \frac{1}{k} (w_k - x_k) \end{aligned}$$

我们就可以迭代地计算 w_{k+1} 了。

于是我们稍作改进，把 $\frac{1}{k}$ 换成 α 。于是我们就可以通过调整 α 来改变公式的计算了。

$$w_{k+1} = w_k - \alpha(w_k - x_k)$$

Robbins-Monro algorithm

stochastic approximation能够做到在不知道函数具体公式的情况下求出解。

RM算法是stochastic approximation中的开创性工作。

而stochastic gradient descent algorithm 则是RM的一种特殊情况。

问题：求解 $g(w) = 0$ 方程，其中 w 是未知量， g 是函数。

于是RM算法可以求解如下问题：

$$w_{k+1} = w_k - \alpha_k \tilde{g}(w_k, \eta_k)$$

其中 η_k 是噪声， $\tilde{g}(w_k, \eta_k) = g(w_k) + \eta_k$ ， α 是一个正数。

函数 $g(w_k)$ 是一个黑盒函数，我们无法得出它的具体公式。

不断迭代这个公式，就能够收敛到 $g(w) = 0$

RM算法-Convergence properties

RM算法的三个条件：

1. $0 < c_1 \leq \nabla_w g(w) \leq c_2$ ，即导数大于0，并且不会趋于无穷。
2. $\sum_{k=1}^{\infty} a_k = \infty$ 并且 $\sum_{k=1}^{\infty} a_k^2 < \infty$ 。
 $\sum_{k=1}^{\infty} a_k^2 < \infty$ 保证了 a_k 一定会收敛到0。
 $\sum_{k=1}^{\infty} a_k = \infty$ 保证了 a_k 收敛的不会太快，否则加起来就不会是无穷了。
3. $\mathbb{E}[\eta_k] = 0$ 并且 $\mathbb{E}[\eta_k^2 | \mathcal{H}] < \infty$ (这里的 $\mathbb{E}[\eta_k^2 | \mathcal{H}]$ 的意思是 η_k 的方差)。通常这里的噪声通过同分布(Independent and Identically Distributed)采样得来，并且在此处 η_k 并没有强制要求满足高斯分布。

对于条件二的解释：

根据上面的公式 $w_{k+1} - w_k = a_k \tilde{g}(w_k, \eta_k)$ ，那么 a_k 收敛到0，才能保证 $w_{k+1} - w_k$ 不断收敛到0，从而趋于稳定。

而将 $k = 1, 2, \dots, \infty$ 的公式相加可以得到

$$w_{\infty} - w_1 = \sum_{k=1}^{\infty} a_k \tilde{g}(w_k, \eta_k) .$$

$w^* \approx w_{\infty}$ 是我们猜测的值， w_1 是初始值，那么 $\sum_{k=1}^{\infty} a_k = \infty$ 保证了不管我们选的初始值 w_1 离目标值有多远，最终都可以通过不断迭代得到 w^*

a_k 取什么值是符合条件的？ $a_k = \frac{1}{k}$ 。（但一般在 k 很大的时候，不会让 a_k 一直变小，达到某个较小值后则会不再改变）

SGD(stochastic gradient descent)

目标是解决如下优化问题：

$$\min_w J(w) = \mathbb{E}[f(w, X)]$$

算法一、 gradient descent(GD)梯度下降法：

$$w_{k+1} = w_k - \alpha_k \nabla_w \mathbb{E}[f(w_k, X)] = \alpha_k \mathbb{E}[\nabla_w f(w_k, X)]$$

这里的 α_k 是步长，就是学习率。但是一般无法得到准确的梯度的期望。

算法二、batch gradient descent(BGD)批量梯度下降法：

$$[\nabla_w f(w_k, X)] \approx \frac{1}{n} \sum_{i=1}^n \nabla_w f(w_k, x_i)$$

$$\text{于是得到: } w_{k+1} = w_k - \alpha_k \frac{1}{n} \sum_{i=1}^n \nabla_w f(w_k, x_i)$$

不要求期望，用多次采样的平均值来代替期望值，但是每次都要求n个数的平均太耗时了。（n为采样次数）

算法三、stochastic gradient descent(SGD) 随机梯度下降：

$$w_{k+1} = w_k - \alpha_k \nabla_w f(w_k, x_k)$$

和GD相比，替使用随机梯度来替换准确的期望的梯度。

和BGD相比，其实就是把n设置为了1。

SGD 的例子和练习

假设如下例子： $\min_w J(w) = \mathbb{E}[f(w, X)] = \mathbb{E}[\frac{1}{2} \|w - X\|^2]$

此处的 $f(w, X) = \|w - X\|^2/2$, $\nabla_w f(w, X) = w - X$

三个练习：

1. 证明最优解 w^* 满足 $w^* = \mathbb{E}[X]$

$$\begin{aligned} \nabla_w J(w) &= 0 \\ \Rightarrow \mathbb{E}[\nabla_w f(w, X)] &= 0 \\ \Rightarrow \mathbb{E}[w - X] &= 0 \\ \Rightarrow w^* &= \mathbb{E}[X] \end{aligned}$$

2. 这个例子的GD算法是什么？

$$\begin{aligned} w_{k+1} &= w_k - \alpha_k \nabla_w J(w_k) \\ &= w_k - \alpha_k \mathbb{E}[\nabla_w J(w_k)] \\ &= w_k - \alpha_k \mathbb{E}[w_k - X] \end{aligned}$$

3. 这个例子的SGD算法是什么？

不求期望了，直接用某一个的 $w_k - x_k$ 来代替 $\mathbb{E}[w_k - X]$

$$w_{k+1} = w_k - \alpha_k \nabla_w f(w_k, x_k) = w_k - \alpha_k (w_k - x_k)$$

我们发现最后的公式和mean algorithm算法是一样的，所以mean algorithm算法就是一种特殊的SGD算法。

SGD算法的收敛性(convergence) ..

1. 首先证明SGD是一种特殊的RM算法:

SGD的目标是最小化 $J(w) = \mathbb{E}[f(w, X)]$, 这个问题可以转换为寻根问题:

$$\nabla_w J(w) = \mathbb{E}[\nabla_w f(w, X)] = 0$$

$$\text{设 } g(w) = \nabla_w J(w) = \mathbb{E}[\nabla_w f(w, X)]$$

那么SGD的目标就是找到 $g(w) = 0$ 的根。

我们可以测量的是:

$$\begin{aligned}\tilde{g}(w, \eta) &= \nabla_w f(w, x) \\ &= \mathbb{E}[\nabla_w f(w, X)] + (\nabla_w f(w, x) - \mathbb{E}[\nabla_w f(w, X)])\end{aligned}$$

而与之对应的RM算法是

$$w_{k+1} = w_k - \alpha_k \tilde{g}(w, \eta) = w_k - \alpha_k \nabla_w f(w_k, x_k)$$

2. 接下来我们就可以应用RM算法的收敛性条件, 来证明SGD是收敛的。

SGD算法的收敛模式 ..

SGD收敛的过程中, 是否会收敛很慢或者收敛随机?

我们定义相对误差 δ_k

$$\delta_k = \frac{|\nabla_w f(w, x) - \mathbb{E}[\nabla_w f(w, X)]|}{|\mathbb{E}[\nabla_w f(w, X)]|}$$

此处的 $\mathbb{E}[\nabla_w f(w, X)]$ 是true gradient, 而 $\nabla_w f(w, x)$ 是 stochastic gradient。

性质: 当 w_k 离 w^* 较远时, 相对误差较小, 当 w_k 离 w^* 很近的时候, 才会有比较大的相对误差 (即随机性)

如何得到如上性质?

使用拉格朗日中值定理 $f(x_1) - f(x_2) = f'(x_3)(x_1 - x_2)$

那么由 $\mathbb{E}[\nabla_w f(w^*, X)] = 0$ 和中值定理, 我们有

$$\delta_k = \frac{|\nabla_w f(w, x) - \mathbb{E}[\nabla_w f(w, X)]|}{|\mathbb{E}[\nabla_w f(w, X)] - \mathbb{E}[\nabla_w f(w^*, X)]|} = \frac{|\nabla_w f(w, x) - \mathbb{E}[\nabla_w f(w, X)]|}{|\mathbb{E}[\nabla_w^2 f(\tilde{w}, X)(w_k - w^*)]|}$$

我们假设 $\nabla_w^2 f \geq c > 0$

那么我们考虑分母项，就有

$$\begin{aligned} |\mathbb{E}[\nabla_w^2 f(\tilde{w}, X)(w_k - w^*)]| &= |\mathbb{E}[\nabla_w^2 f(\tilde{w}, X)](w_k - w^*)| \\ &= |\mathbb{E}[\nabla_w^2 f(\tilde{w}, X)]| |w_k - w^*| \geq c |w_k - w^*| \end{aligned}$$

于是误差 δ_k 满足

$$\delta_k \leq \frac{|\nabla_w f(w, x) - \mathbb{E}[\nabla_w f(w, X)]|}{c |w_k - w^*|}$$

于是当 w_k 距离 w^* 比较远，那么分母比较大，相对误差 δ_k 的上界比较小。

当 w_k 距离 w^* 比较近，那么分母比较小，此时相对误差 δ_k 的上界才会变大一些。

BGD, MBGD 和 SGD

- BGD(batch gradient descent)，用到所有的采样来平均求期望
- MBGD(min-batch gradient descent)，选择一部分采样(m 个采样)
- SGD(stochastic gradient descent)，选择一个采样

在 MBGD 中，当 MBGD 中的采样数量 $m = 1$ 时，等价于 SGD。

当采样数量 $m = n$ 时，趋近于 BGD(注意！此时不完全等于 BGD，因为 BGD 是取出所有的 n 个样本，而 MBGD 是对样本集进行 n 次的采样)

考虑如下优化问题：

$$\min_w J(w) = \frac{1}{2n} \sum_{i=1}^n \|w - x_i\|^2$$

那么三种算法的迭代公式如下：

$$w_{k+1} = w_k - \alpha_k \frac{1}{n} \sum_{i=1}^n (w_k - x_i) = w_k - \alpha_k (w_k - \bar{x}) \quad (BGD)$$

$$w_{k+1} = w_k - \alpha_k \frac{1}{m} \sum_{j \in I_k} (w_k - x_j) = w_k - \alpha_k (w_k - \bar{x}^{(m)}) \quad (MBGD)$$

$$w_{k+1} = w_k - \alpha_k (w_k - x_k) \quad (SGD)$$

第七章 Temporal-Difference learning

Temporal-Difference learning (TD) 时序差分算法

这是一个 incremental 迭代式的算法。

motivating example ..

1. 先考虑一个简单的问题 mean estimation : 计算

$w = [X]$, (X 是一些iid(独立同分布)采样 $\{x\}$)

令 $g(w) = w - \mathbb{E}[X]$, 则有

$$\tilde{g}(w, \eta) = w - x = (w - \mathbb{E}[X]) + (\mathbb{E}[X] - x) \approx g(w) + \eta$$

然后根据RM算法, 可以得到 $w_{k+1} = w_k - \alpha_k \tilde{g}(w_k, \eta_k) = w_k - \alpha_k (w_k - x_k)$

2. 考虑一个复杂一些的例子: 计算

$w = \mathbb{E}[v(X)]$, (X 是一些iid(独立同分布)采样 $\{x\}$)

令 $g(w) = w - \mathbb{E}[v(X)]$

$$\tilde{g}(w, \eta) = w - v(x) = (w - \mathbb{E}[X]) + (\mathbb{E}[X] - v(x)) \approx g(w) + \eta$$

然后根据RM算法, 可以得到 $w_{k+1} = w_k - \alpha_k \tilde{g}(w_k, \eta_k) = w_k - \alpha_k (w_k - v(x_k))$

3. 第三个例子: 计算

$w = [R + \gamma v(X)]$, (R, X 是随机变量, γ 是常量, $v(\cdot)$ 是函数)

令 $g(w) = w - \mathbb{E}[R + \gamma v(X)]$,

$$\begin{aligned}\tilde{g}(w, \eta) &= w - \mathbb{E}[R + \gamma v(X)] \\ &= (w - \mathbb{E}[R + \gamma v(X)]) + (\mathbb{E}[R + \gamma v(X)] - [r + \gamma v(X)]) \\ &\approx g(w) + \eta\end{aligned}$$

然后根据RM算法, 可以得到

$$w_{k+1} = w_k - \alpha_k \tilde{g}(w_k, \eta_k) = w_k - \alpha_k [w_k - (r_k + \gamma v(x_k))]$$

TD算法中的state values ..

注意:

- TD算法通常指的是一大类的RL算法。
- TD算法也可以特指一种用于估计state values的算法。

TD算法基于数据: $(s_0, r_1, s_1, \dots, s_t, r_{t+1}, s_{t+1}, \dots)$ 或者 $\{(s_t, r_{t+1}, s_{t+1})\}_t$, 这种数据通过给定的策略 π 来生成。

TD算法则是:

$$\begin{aligned}v_{t+1}(s_t) &= v_t(s_t) - \alpha_t(s_t)[v_t(s_t) - [r_{t+1} + \gamma v_t(s_{t+1})]] & (1) \\ v_{t+1}(s) &= v_t(s), \forall s \neq s_t & (2)\end{aligned}$$

对于公式(2) 表示，如果现在的状态是 s_t ，那么其他状态的value是不更新的。

我们关注于第一个式子：

$$v_{t+1}(s_t) = v_t(s_t) - \alpha_t(s_t)[v_t(s_t) - [r_{t+1} + \gamma v_t(s_{t+1})]]$$

其中的 $v_{t+1}(s_t)$ 是新的估计值， $v_t(s_t)$ 是现在的估计值。

$v_t(s_t) - [r_{t+1} + \gamma v_t(s_{t+1})]$ 是误差 δ_t

$[r_{t+1} + \gamma v_t(s_{t+1})]$ 是目标 \bar{v}_t

为什么 \bar{v}_t 是“TD目标”？因为每次 $v(s_t)$ 都会向着 \bar{v}_t 移动。

$$\begin{aligned} v_{t+1}(s_t) &= v_t(s_t) - \alpha_t(s_t)[v_t(s_t) - \bar{v}_t] \\ \Rightarrow v_{t+1}(s_t) - \bar{v}_t &= v_t(s_t) - \bar{v}_t - \alpha_t(s_t)[v_t(s_t) - \bar{v}_t] \\ \Rightarrow v_{t+1}(s_t) - \bar{v}_t &= [1 - \alpha_t(s_t)][v_t(s_t) - \bar{v}_t] \end{aligned}$$

因为 $0 < 1 - \alpha_t(s_t) < 1$

于是 $|v_{t+1}(s_t) - \bar{v}_t| \leq |v_t(s_t) - \bar{v}_t|$

为什么 δ_t 是“TD error”？

$$\delta_t = v(s_t) - [r_{t+1} + \gamma v(s_{t+1})]$$

因为发生在t和t+1两个时刻，所以才叫时序差分，

TD error 描述了 v_t 和 v_π 之间的误差。

当 $v_t = v_\pi$ 时，那么应该有 $\delta_t = 0$ 。

TD error是一种 innovation，这是经验 (s_t, r_{t+1}, s_{t+1}) 的一种新的信息。

TD算法的数学意义 ..

他解决了给定 π ，求解贝尔曼公式。

新的贝尔曼公式：

$$v_\pi(s) = \mathbb{E}[R + \gamma G | S = s], s \in S$$

在这之中G是下个状态的Reward，所以 $\mathbb{E}[G | S = s]$ 可以表示为：

$$\mathbb{E}[G | S = s] = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) v_\pi(s') = \mathbb{E}[v_\pi(S') | S = s]$$

其中 S' 是下一个状态

于是 s 的state value可以写为:

$$v_{\pi}(s) = \mathbb{E}[R + \gamma v_{\pi}(S') | S = s], s \in S$$

这个公式也被称为贝尔曼期望公式。

接下来使用RM算法来求解这个贝尔曼期望公式:

定义 $g(v(s)) = v(s) - \mathbb{E}[R + \gamma v_{\pi}(S') | S = s] = 0$

于是我们有 $g(v(s)) = 0$

$$\begin{aligned}\tilde{g}(v(s)) &= v(s) - [r + \gamma v_{\pi}(s')] \\ &= (v(s) - \mathbb{E}[R + \gamma v_{\pi}(S') | s]) + (\mathbb{E}[R + \gamma v_{\pi}(S') | s] - [r + \gamma v_{\pi}(s')])\end{aligned}$$

在这之中, $g(v(s)) = (v(s) - \mathbb{E}[R + \gamma v_{\pi}(S') | s])$, 误差

$$\eta = \mathbb{E}[R + \gamma v_{\pi}(S') | s] - [r + \gamma v_{\pi}(s')]$$

那么与之对应的RM算法是:

$$\begin{aligned}v_{k+1}(s) &= v_k(s) - \alpha_k \tilde{g}(v_k(s)) \\ &= v_k(s) - \alpha_k (v_k(s) - [r_k + \gamma v_{\pi}(s'_k)]), k = 1, 2, 3, \dots\end{aligned}$$

这里的 $v_k(s)$ 代表 $v_{\pi}(s)$ 在第 k 步的估计, 而 r_k, s'_k 是第 k 步中从 R, S' 中取出的样本。

对公式做以下替换:

- 将一组采样 $\{(s, r, s')\}$ 替换为一组序列 $\{s_t, r_{t+1}, s_{t+1}\}$, 从而做到对所有的 s 都进行更新。
- 将 $v_{\pi}(s')$ 换为 $v_k(s'_k)$, 即我们直接用 s' 在第 k 步的估计值来替代真实值。虽然会有一些偏差, 但是最终会收敛到 v_{π}

TD算法的收敛:

对于所有状态 $s \in S$ 。当 $t \rightarrow \infty$ 时, $v_t(s)$ 以概率1收敛到策略 π 下的状态值函数 $v_{\pi}(s)$ 。

如果对于所有的状态 $s \in S$, 步长参数序列 $\alpha_t(s)$ 都满足 $\sum_t \alpha_t = \infty$ 并且 $\sum_t \alpha_t^2(s) < \infty$ 那么上述收敛成立。

TD/Sarsa learning	MC learning
online: TD学习是在线的，在接收到一个奖励后可以更新state/action value	Not online: MCLearning是非在线的，必须等到整个episode已经完成之后，计算return值然后进行估计。
continuing tasks: 即能处理一直持续下去的任务，同时也能解决episodic tasks。	Episodic tasks: 必须是有限步的episode，才能等到他的返回值。
Bootstrapping: 会基于之前对状态的猜测，加上一些新的信息来形成一个新的猜测	Non-bootstrapping: 直接根据当前的episode计算return，不涉及到之前的估计值
Low estimation variance : 在算法过程中涉及到的随机变量比较少，所以方差会比较小	High estimation variance: 它涉及到了很多的variable，因为一次episode会涉及到很多的Reward，而只用其中一次的采样，所以就会有比较大的方差。
bias: 因为基于之前的经验，所以可能会因为之前的经验而产生bias，导致有偏估计，但是在不断增加经验后还是会趋于正确结果	no bias: 不基于之前的估计，所以不会产生bias

TD算法中的action values: Sarsa

Sarsa是经验集 $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ 的拼接。

TD算法是用来估计给定策略 π 的state value，但我们需要估计的是action value。下面引入Sarsa。

假设我们有如下经验 $\{(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})\}_t$ ，那么我们定义Sarsa公式如下：

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - [r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})]]$$

$$q_{t+1}(s, a) = q_t(s, a), \forall (s, a) \neq (s_t, a_t)$$

这个式子和TD算法几乎一样，只是类似地把 $v_t(s_t)$ 改成了 $q_t(s_t, a_t)$ 这样子。

Sarsa的数学意义和TD也是几乎一样的。（如贝尔曼公式，收敛性等）

Sarsa所求解的贝尔曼公式：

$$q_{\pi}(s, a) = \mathbb{E}[R + \gamma q_{\pi}(S', A') | s, a], \forall s, a$$

1. 收集经验: $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, 遵循 $\pi_t(s_t)$ 执行 a_t , 得到 r_{t+1} 的奖励, 然后走到状态 s_{t+1} 并遵循 $\pi_t(s_{t+1})$ 来采取行动 a_{t+1} 。
2. 更新q值(q value update/policy evaluation):

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - [r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})]]$$
3. 更新策略policy(policy update/policy improvement):

$$\pi_{t+1}(a|s_t) = 1 - \frac{\epsilon}{|A|}(|A| - 1), \quad \text{if } a = \arg \max_a q_{t+1}(s_t, a)$$

$$\pi_{t+1}(a|s_t) = \frac{\epsilon}{|A|}, \quad \text{otherwise}$$

注意这里的PE和PI是立刻执行的, 而不是等return之后再精确计算。

注意这个策略是一个 $\epsilon - greedy$ 策略, 也就是说倾向于采取qvalue最大的action, 但是其他的action同样有概率取到。

Expected Sarsa

公式如下:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - (r_{t+1} + \gamma \mathbb{E}[q_t(s_{t+1}, A)])]$$

$$q_{t+1}(s, a) = q_t(s, a), \forall (s, a) \neq (s_t, a_t)$$

此处的 $\mathbb{E}[q_t(s_{t+1}, A)] = \sum_{\pi} \pi_t(a|s_{t+1})q_t(s_{t+1}, a) \approx v_t(s_{t+1})$

和普通的sarsa的区别是用 $(r_{t+1} + \gamma \mathbb{E}[q_t(s_{t+1}, A)])$ 替换了 $r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$ 。

不再需要 a_{t+1} 了, 随机性会减小一些, 但是需要更大的计算量。

Expected Sarsa的数学意义也是在求解贝尔曼公式:

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

n-step Sarsa

是Sarsa的一个推广, 包含了Sarsa和蒙特卡罗方法。

我们的action value如下定义: $q_{\pi}(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$

那么 G_t 可以被写成如下形式:

$$\begin{aligned}
\text{Sarsa} \leftarrow G_t^{(1)} &= R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \\
G_t^{(2)} &= R_{t+1} + \gamma R_{t+1} + \gamma^2 q_\pi(S_{t+2}, A_{t+2}) \\
&\dots \\
\text{n-step Sarsa} \leftarrow G_t^{(n)} &= R_{t+1} + \gamma R_{t+2} + \dots + \gamma^n q_\pi(S_{t+n}, A_{t+n}) \\
&\dots \\
MC \leftarrow G_t^{(\infty)} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots
\end{aligned}$$

所以n-step Sarsa对应的贝尔曼公式是：

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - [r_{t+1} + \gamma r_{t+2} + \dots + \gamma^n q_t(s_{t+n}, a_{t+n})]]$$

n-step Sarsa需要的数据是 $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \dots, r_{t+n}, s_{t+n}, a_{t+n})$

所以他的数据需要等到 $t+n$ 时刻，才能进行更新。是online和offline的结合。

- 当n比较大的时候，更接近于MC，会有比较大的variance，比较小的bias。
- 当n比较小的时候，更接近于Sarsa，会有比较小的variance，比较大的bias。

TD中最优action value学习:Q-learning

算法如下：

$$\begin{aligned}
q_{t+1}(s_t, a_t) &= q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - [r_{t+1} + \gamma \max_{a \in \mathcal{A}} q_t(s_{t+1}, a)]] \\
q_{t+1}(s, a) &= q_t(s, a), \forall (s, a) \neq (s_t, a_t)
\end{aligned}$$

和Sarsa相比，用 $r_{t+1} + \gamma \max_{a \in \mathcal{A}} q_t(s_{t+1}, a)$ 替换了 $r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$

Q-learning求解的数学问题是（不是在求解贝尔曼方程）：

求解一个贝尔曼最优方程：

$$q(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_a q(S_{t+1}, a) | S_t = s, A_t = a], \forall s, a$$

off-policy 和 on-policy

两种策略：

1. behavior policy用来生成经验样本
2. target policy不断地更新来将target policy更新到optimal policy。

基于这两种策略，可以分为两类算法：

- **on-policy**: 其中的behavior policy和target policy是相同的，即用自己的策略来和环境交互，然后得到经验并改进自己的策略，之后再用相同的策略和环境交互。
- **off-policy**: 用一个策略和环境交互得到大量经验，然后用这些经验来不断改进策略（一步到位，不再通过新的策略引入新的经验）

on-policy的好处就是可以不断接收新的经验，实时更新策略。

off-policy的好处就是可以直接使用别人已经获取过的经验。如用之前通过探索性较强的算法得到的经验。

如何判断一个TD算法是on-policy还是off-policy?

1. 看这个TD算法是在解决什么样的数学问题
2. 看在算法的执行过程中需要什么东西才能使算法跑起来

- Sarsa是on-policy的：

Sarsa在数学上就是在求解一个贝尔曼公式：

$$q_{\pi}(s, a) = \mathbb{E}[R + \gamma q_{\pi}(S', A') | s, a], \forall s, a$$

此处的 $R \sim p(R|s, a)$, $S' \sim p(S'|s, a)$, $A' \sim \pi(A'|S')$

Sarsa在算法中：

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - [r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})]]$$

1. 如果给定了 (s_t, a_t) 那么 r_{t+1} 和 s_{t+1} 和任何策略无关，和 $p(r|s, a), p(s'|s, a)$ 有关。
2. a_{t+1} 是由策略 $\pi_t(s_{t+1})$ 产生。所以 Π_t 既是behavior policy也是target policy

- MC learning 是on-policy的：

MC目的是求解如下贝尔曼方程：

$$q_{\pi}(s, a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s, A_t = a]$$

MC的实现是

$$q(s, a) \approx r_{t+1} + \gamma r_{t+2} + \dots$$

我们用策略 Π 来得到trajectory经验，然后得到return来近似估计 q_π 进而改进 Π

- Q learning 是off-policy的:

Q learning求解的数学问题是:

求解贝尔曼最优公式:

$$q(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_a q(S_{t+1}, a) | S_t = s, A_t = a], \forall s, a$$

Q learning的实现过程是:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - [r_{t+1} + \gamma \max_{a \in A} q_t(s_{t+1}, a)]]$$

需要的经验是 $(s_t, a_t, r_{t+1}, s_{t+1})$

注意这里的经验不包含 a_{t+1}

如果 (s_t, a_t) 给定，那么 r_{t+1} 和 s_{t+1} 不依赖于策略。

behavior policy是从 s_t 出发得到 a_t

target policy 是根据 q_π 来选择action

Q-learning 的实施

如果将Q-learning中的behavior policy 和target policy强行设置为一致的，那么它可以是on-policy的:

0. 对每个episode执行以下三步

1. 收集经验 $(s_t, a_t, r_{t+1}, s_{t+1})$ ，在这一步根据 $\pi_t(s_t)$ 采取行动 a_t 来生成 (r_{t+1}, s_{t+1})

2. 更新q-value:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - [r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)]]$$

3. 更新policy:

$$\pi_{t+1}(a|s_t) = 1 - \frac{\epsilon}{|A|} (|A| - 1) \text{ if } a = \underset{a}{\operatorname{argmax}} q_{t+1}(s_t, a)$$
$$\pi_{t+1}(a|s_t) = \frac{\epsilon}{|A|} \text{ otherwise}$$

也可以是off-policy的:

0. 对每个episode生成策略 π_b (这里的b代表behavior),这个策略用来生成 experience
1. 对episode的每一步 $t = 0, 1, 2, \dots$ 执行以下两步:
2. 更新q-value:
$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - [r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)]]$$
3. 更新target policy:
$$\pi_{T,t+1}(a|s_t) = 1 \text{ if } a = \underset{a}{\operatorname{argmax}} q_{t+1}(s_t, a)$$
$$\pi_{T,t+1}(a|s_t) = 0 \text{ otherwise}$$

注意这里的第三步是greedy不是 $\epsilon - greedy$ ，因为我们不需要新的策略来生成经验，所以也就不需要使用 $\epsilon - greedy$ 来增加探索性，只需要保证最优性。

使用off-policy的话，使用的behavior policy最好是探索度比较强的策略，否则可能得不到好的target policy。

TD的统一表示

所有的TD算法都能用如下公式表达:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)[q_t(s_t, a_t) - \bar{q}_t]$$

这里的 \bar{q}_t 就是TD target。

TD算法的目标就是接近TD target，减小TD error

算法	\bar{q}_t 的表示
Sarsa	$\bar{q}_t = r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$
n-step Sarsa	$\bar{q}_t = r_{t+1} + \gamma r_{t+2} + \dots + \gamma^n q_t(s_{t+n}, a_{t+n})$
Expected Sarsa	$\bar{q}_t = r_{t+1} + \gamma \sum_a \pi_i(a s_{t+1}) q_t(s_{t+1}, a)$
Q-learning	$\bar{q}_t = r_{t+1} + \gamma \max_a q_t(s_{t+1}, a)$
Monte Carlo	$\bar{q}_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots$

第八章 value function Approximation

在此之前，所有的state value和action value都是用表格表示出来的，例如：

	a_1	a_2	a_3	a_4	a_5
s_1	$q_\pi(s_1, a_1)$	$q_\pi(s_1, a_2)$	$q_\pi(s_1, a_3)$	$q_\pi(s_1, a_4)$	$q_\pi(s_1, a_5)$
...
s_9	$q_\pi(s_9, a_1)$	$q_\pi(s_9, a_2)$	$q_\pi(s_9, a_3)$	$q_\pi(s_9, a_4)$	$q_\pi(s_9, a_5)$

使用表格的好处就是可以直观地分析

坏处就是无法处理很大的state space 或者action space、无法处理连续的state和action。泛化能力不强。

Value Function Approximation的含义 :

使用直线来拟合点：

$$\hat{v}(s, w) = as + b = [s, 1] \begin{bmatrix} a \\ b \end{bmatrix} = \phi^T(s)w$$

这里的w是parameter vector(参数向量)

$\phi(s)$ 是s的feature vector(特征向量)

$\hat{v}(s, w)$ 是w的linear(线性关系)

使用函数来拟合可以节省存储空间(只需要存w的值(a,b)即可)

缺点是拟合后不太精确。

同样可以用二次函数来拟合：

$$\hat{v}(s, w) = as^2 + bs + c = \phi^T(s)w$$

(需要注意的是，这样的曲线对于w来说同样是一种线性的拟合)

- 使用Value Function Approximation的优点：
 1. 便于存储，只需要存储w即可，不用存储大量数据。
 2. 泛化能力更强，假设s2进行了更改，在表格存储中只会更改s2对应的内容，而使用value function approximation则会改变w的值，从而影响其他的值（如s1和s3），这样会增强泛化能力

objective function :

令 $v_\pi(s)$ 作为真正的state value。 $\hat{v}(s, w)$ 是函数的近似值。

我们目标就是找到最优的w来使得 $\hat{v}(s, w)$ 可以很好的拟合出 $v_\pi(s)$

目标函数objective function如下：

$$J(w) = \mathbb{E}[(v_{\pi}(S) - \hat{v}(S, w))^2]$$

我们目标就是找到最好的 w 使得 $J(w)$ 最小化。

求解期望常见的两种方法：

uniform distribution 平均分布：

认为每一个状态都是同等重要的。那么每一个状态的权重就是 $\frac{1}{|S|}$

$$J(w) = \mathbb{E}[(v_{\pi}(S) - \hat{v}(S, w))^2] = \frac{1}{|S|} \sum_{s \in S} (v_{\pi}(s) - \hat{v}(s, w))^2$$

使用均匀策略的缺点是，我们将很远处的状态和距离目标近处的状态设置权重一样。导致没有侧重点

第二个概率分布：

stationary distribution：

这是一种Markov下的long-run behavior

让 $\{d_{\pi}(s)\}_{s \in S}$ 作为在策略 π 下的Markov stationary distribution，通过定义 $d_{\pi}(s) \geq 0$ and $\sum_{s \in S} d_{\pi}(s) = 1$

于是目标函数objective function可以被写为：

$$J(w) = \mathbb{E}[(v_{\pi}(S) - \hat{v}(S, w))^2] = \sum_{s \in S} d_{\pi}(s) (v_{\pi}(s) - \hat{v}(s, w))^2$$

Stationary distribution 也被称为steady-state distribution 或者limit distribution。

怎么求 $d_{\pi}(s)$ ？ 给出一个很长很长的episode：

我们定义 $n_{\pi}(s)$ 表示 s 出现的次数。于是我们用频率来近似估计 $d_{\pi}(s)$

$$d_{\pi}(s) \approx \frac{n_{\pi}(s)}{\sum_{s' \in S} n_{\pi}(s')}$$

我们没必要真正的模拟这个episode并统计次数，可以通过数学公式得到：

最终趋于稳定的 d_{π}^T 要满足：

$$d_{\pi}^T = d_{\pi}^T P_{\pi}$$

这里的 P_{π} 是一个矩阵，代表从 s 到 s' 转移的概率 $p(s'|s)$

optimization algorithms ::

目标函数的优化算法(optimization algorithms of objective function)

为了最小化目标函数 $J(w)$ ，我们可以使用梯度优化gradient-descent

$$w_{k+1} = w_k - \alpha_k \nabla_w J(w_k)$$

这里的true gradient是:

$$\begin{aligned}\nabla_w J(w) &= \nabla_w \mathbb{E}[(v_\pi(S) - \hat{v}(S, w))^2] \\ &= \mathbb{E}[\nabla_w (v_\pi(S) - \hat{v}(S, w))^2] \\ &= 2\mathbb{E}[(v_\pi(S) - \hat{v}(S, w))(-\nabla_w \hat{v}(S, w))] \\ &= -2\mathbb{E}[(v_\pi(S) - \hat{v}(S, w))\nabla_w \hat{v}(S, w)]\end{aligned}$$

但是求期望很麻烦，所以我们用stochastic gradient descent来替代gradient descent

$$w_{t+1} = w_t + \alpha_k (v_\pi(s_t) - \hat{v}(s_t, w_t)) \nabla_w \hat{v}(s_t, w_t)$$

但是在现实中，我们是无法得知 $v_\pi(s_t)$ 的。

有如下方法：

1. Monte Carlo learning 和 value function approximation结合

让 g_t 表示从 s_t 开始的episode的return。那么

$$w_{t+1} = w_t + \alpha_t (g_t - \hat{v}(s_t, w_t)) \nabla_w \hat{v}(s_t, w_t)$$

2. TD learning 和value function approximation结合

在TD算法中可以用 $r_{t+1} + \gamma \hat{v}(s_{t+1}, w_t)$ 来作为 $v_\pi(s_t)$ 的一个估计值。于是有

$$w_{t+1} = w_t + \alpha_t [r_{t+1} + \gamma \hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)] \nabla_w \hat{v}(s_t, w_t)$$

目前只能用来估计给定策略的state values。

selection of function approximators ::

函数的选取方法

1. 选择线性函数（之前广为使用）：

$$\hat{v}(s, w) = \phi^T(s)w$$

此处的 $\phi(s)$ 是特征向量，他是基于多项式的(polynomial basis),基于傅里叶的(Fourier basis),...

2. 选择神经网络（现在广为使用）：

网络的参数是 w ，神经网络的输入是state，输出是估计值 $\hat{v}(x, w)$

考虑线性函数： $\hat{v}(s, w) = \phi^T(s)w$,我们有

$$\nabla_w \hat{v}(s, w) = \phi(s)$$

代入TD算法可以得到TD-Linear：

$$w_{t+1} = w_t + \alpha_t [r_{t+1} + \gamma \phi^T(s_{t+1})w_t - \phi^T(s_t)w_t] \phi(s_t)$$

Linear function approximation的劣势：

- 很难去选择一个合适的feature vectors

Linear function approximation的优势：

- 数学原理清晰，能够可以帮助我们更透彻地研究。
- 表征能力还算可以。表格形式tabular representation是Linear function approximation 的特殊形式。

Tabular representation 是Linear function的一种特殊情况：

首先考虑如下的特征向量：

$$\phi(s) = e_s \in \mathbb{R}^{|S|}$$

这里的 e_s 是一个只有一个1，其他都是0的向量。

那么这样的话 $\hat{v}(s, w) = e_s^T w = w(s)$ ，这样 $w(s)$ 就是 w 的第 s 个元素了。也就是tabular representation。

然后我们把 $\phi(s_t) = e_{s_t}$ 带入到TD-Linear中。

$$w_{t+1} = w_t + \alpha(r_{t+1} + \gamma w_t(s_{t+1}) - w_t(s_t)) e_{s_t}$$

因为 e_{s_t} 只有在 w_t 的位置是1，其他位置是0，所以在 w_t 中只有 s_t 的位置被更新了。

$$w_{t+1}(s_t) = w_t(s_t) + \alpha_t(r_{t+1} + \gamma w_t(s_{t+1}) - w_t(s_t))$$

于是我们发现这个式子和之前tabular的TD算法是一样的。

Sarsa

Sarsa和value function estimation结合：

$$w_{t+1} = w_t + \alpha_t[r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t)] \nabla_w \hat{q}(s_t, a_t, w_t)$$

除了标蓝的地方略有差别，其他和tabular是一样的。

1. 因为我们描述的参数从state变为了parameter, 所以更新的内容从 $q(s, a)$ 变为了 w
2. 使用估计值来估计一个点的state, 所以原先的 $q(s, a)$ 变为了由函数估计产生的 $\hat{q}(s, a, w)$
3. 最后乘上 \hat{q} 的梯度来使得向零点移动。

步骤如下：

1. 对每个episode 执行如下操作：
2. 遵循 $\pi_t(s_t)$ 执行动作 a_t , 然后生成 r_{t+1}, s_{t+1} , 然后遵循 $\pi_t(s_{t+1})$ 执行 a_{t+1} .
3. 更新值 (parameter update) :

$$w_{t+1} = w_t + \alpha_t[r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t)] \nabla_w \hat{q}(s_t, a_t, w_t)$$

4. 更新策略(policy update):

$$\pi_{t+1}(a|s_t) = 1 - \frac{\epsilon}{|A(s)|} (|A(s)| - 1) \quad \text{if } a = \arg \max_{a \in A(s_t)} \hat{q}(s_t, a, w_{t+1})$$

$$\pi_{t+1}(a|s_t) = \frac{\epsilon}{|A(s)|} \quad \text{otherwise}$$

Q-learning

Q-learning 和value function estimation 结合：

$$w_{t+1} = w_t + \alpha_t[r_{t+1} + \gamma \max_{a \in A(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a, w_t)] \nabla_w \hat{q}(s_t, a_t, w_t)$$

蓝色部分为value function estimation 版本的Q-learning和tabular 版本之间的区别。

步骤如下 (on-policy) :

1. 对每个episode 执行如下操作:
2. 遵循 $\pi_t(s_t)$ 执行动作 a_t , 然后生成 r_{t+1}, s_{t+1}

3. 更新值 (parameter update) :

$$w_{t+1} = w_t + \alpha_t [r_{t+1} + \gamma \max_{a \in A(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t)] \nabla_w \hat{q}(s_t, a_t, w_t)$$

4. 更新策略(policy update):

$$\begin{aligned} \pi_{t+1}(a|s_t) &= 1 - \frac{\epsilon}{|A(s)|} (|A(s)| - 1) & \text{if } a = \arg \max_{a \in A(s_t)} \hat{q}(s_t, a, w_{t+1}) \\ \pi_{t+1}(a|s_t) &= \frac{\epsilon}{|A(s)|} & \text{otherwise} \end{aligned}$$

Deep Q-learning(deep Q-network) ..

首先定义损失函数: objective function/loss function:

$$J(w) = \mathbb{E}[(R + \gamma \max_{a \in A(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w))^2]$$

这是一个贝尔曼最优误差, 下式为Q-learning的Bellman optimality error:

$$q(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a \in A(S_{t+1})} q(S_{t+1}, a) | S_t = s, A_t = a], \forall s, a$$

因此 $R + \gamma \max_{a \in A(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w)$ 的期望应该是0.

如何最小化这个损失函数? 使用梯度下降法 Gradient-descent:

求 $J(w)$ 关于 w 的梯度, 难点在于表达式中由两个地方出现了 w , 于是基本思想为我们把前半部分当作一个常数 y :

$$y = R + \gamma \max_{a \in A(S')} \hat{q}(S', a, w)$$

这样就只有 $\hat{q}(S, A, w)$ 项包含 w 了, 就可以比较简单地求解梯度。

求解方法:

引入两个网络

- main network 表示 $\hat{q}(s, a, w)$
- target network $\hat{q}(s, a, w_T)$

将损失函数改写为：

$$J(w) = \mathbb{E}[(R + \gamma \max_{\alpha \in A(S')} \hat{q}(S', a, w_T) - \hat{q}(S, A, w))^2]$$

即我们保持target network一段时间不动，因此就能将前半部分作为常数，然后来更新main network。之后再更新target network。这样也能保证两个网络最后都收敛。

DQN-Experience replay

Experience replay经验回放，具体为：

- 我们在收集完数据后，并不根据他们被收集的顺序来使用他们。
- 而是我们把他们存储到一个集合replay buffer 中， $\mathcal{B} = \{(s, a, r, s')\}$
- 每次训练神经网络的时候，把他们混到一起，然后取出一些样本(mini-batch)来进行训练。
- 在取出样本的时候一定要服从均匀分布(uniform distribution)

为什么需要经验回放？为什么必须服从均匀分布？

我们观察损失函数：

$$J(w) = \mathbb{E}[(R + \gamma \max_{\alpha \in A(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w))^2]$$

- $(S, A) \sim d$ 我们根据索引(S,A) 就能够找到一个唯一的随机变量d
- $R \sim p(R|S, A), S' \sim p(S'|S, A)$ ， R和S' 由给定的模型决定。
- 在数据采集的时候，我们可能并不是按照均匀分布采样的。因为他们被确定的策略产生。
- 为了打破连续样本之间的相关性(通常他们有很强的时间相关性)，我们可以从replay buffer中进行随机均匀采样。
- 这就是为什么经验回放是必须的，并且是必须均匀分布采样的。

Deep Q-learning

off-policy version:

目标是从通过 behavior policy π_b 生成的一些经验中学习一个优化的target network来逼近最优action values。

步骤如下:

1. 存储由behavior policy 生成的经验, 存放到replay buffer中
| $\mathcal{B} = \{(s, a, r, s')\}$
2. 对每一次迭代重复如下动作:
3. 从 \mathcal{B} 中均匀提取一些样本(mini-batch)
4. 对于每个样本 (s, a, r, s') , 计算target value $y_T = r + \gamma \max_{\alpha \in \mathcal{A}(s')}(\hat{q}, a, w_T)$
| , 在这之中 w_T 是target network(两个网络之一)
5. 使用mini-batch $\{(s, a, y_T)\}$ 更新main network 来最小化 $(y_T - \hat{q}(s, a, w))^2$
6. 每进行C次迭代, 更新target network: $w_T = w$

第九章 Policy Function Approximation

policy function approximation 也叫policy gradient

之前的方法都是value based, 本次的算法是policy based

在这之前的策略都是用表格来表达的: 即给定 (s_i, a_j) , 会得到一个策略 $\pi(a_i | s_j)$ 。

我们将他写成函数

$$\pi(a | s, \theta)$$

这里的 θ 是一个向量, 表示参数。

用函数代替表格的好处:

- 如果state有很多很多个, 那么在存储上会很费力,
- 难以进行泛化, 在表格中, 如果要更改 $\pi(a | s)$, 那么一定要访问 $\pi(a | s)$
| , 而如果用表达式来表达的话, 则可以通过更改参数来更改一系列的
| π

tabular 和function representations的区别:

1. 定义最优的策略：

在表格情况下，策略 π 当在每一个state value上都最大的时候是最优的。

在函数表示中，策略 π 当能够最大化scalar metrics的时候是最优的

2. 怎么获取一个action的probability?

在表格中，使用索引来得到一个action的probability。

在函数中，需要放入神经网络中进行计算得出。

3. 如何更新policies?

在表格中：直接改变表格中的 $\pi(a|s)$ 的值

在函数中，通过更改 θ 间接修改策略。

policy gradient的目标函数(metric)是：最大化 $J(\theta)$

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} J(\theta_t)$$

policy gradient的两个metrics

metric（度量）

1. 第一个metric是state value的加权平均。

$$\bar{v}_{\pi} = \sum_{s \in S} d(s) v_{\pi}(s) = d^T v_{\pi}$$

在这里 $\sum_{s \in S} d(s) = 1$ ，既可以代表权重，可以代表选择 $v_{\pi}(s)$ 的概率。

如何选择 $d(s)$?

第一种情况是 d 和 π 无关。

- 那么求 \bar{v}_{π} 的梯度不需要得到 d 对 π 的梯度。
- 例如令 $d_0(s) = \frac{1}{|S|}$,得到均匀分布。
- 或者我们对其中的某些状态很关心 $d_0(s_0) = 1, d_0(s \neq s_0) = 0$ ，这时候 $\bar{v}_{\pi} = v_{\pi}(s_0)$

第二种情况是 d 和 π 有关,即 d 依赖于 π

- 一种基本的情况是 d_{π} 满足 $d_{\pi}^T P_{\pi} = d_{\pi}^T$
- 那么会有些状态访问的次数较多，有些状态访问的次数较少。

2. 第二个metric是average one-step reward

$$r_{\pi} \approx \sum_{s \in S} d_{\pi}(s) r_{\pi}(s) = [r_{\pi}(S)]$$

在这里 $S \sim d_{\pi}$,有如下公式:

$$r_{\pi}(s) \approx \sum_{a \in \mathcal{A}} \pi(a|s) r(s, a)$$

代表在一个状态得到的reward的加权平均。

$$r(s, a) = \mathbb{E}(R|s, a) = \sum_r r p(r|s, a)$$

下面给出average one-step reward的第二种形式:

假设根据一个给定的策略生成了一个trajectory, 并有着 $(R_{t+1}, R_{t+2}, R_{t+3}, \dots)$ 的reward。

那么在这个trajectory中, 平均的single-step reward是

$$\begin{aligned} & \lim_{n \rightarrow \infty} \frac{1}{n} \mathbb{E}[R_{t+1} + R_{t+2} + \dots + R_{t+n} | S_t = s_0] \\ &= \lim_{n \rightarrow \infty} \frac{1}{n} \mathbb{E}\left[\sum_{k=1}^n R_{t+k} | S_t = s_0\right] = \lim_{n \rightarrow \infty} \frac{1}{n} \mathbb{E}\left[\sum_{k=1}^n R_{t+k}\right] \end{aligned}$$

注意到最后当 $n \rightarrow \infty$ 时, 可以将 $S_t = s_0$ 省去, 因为当走了无穷步的时候, 从哪一步开始就已经无所谓了。

注意点1:

所有这些metrics都是 π 的函数。因为 π 是由参数 θ 决定, 这些metrics也是关于 θ 的函数, 也就是说不同的 θ 会产生不同的metric values. 我们就可以找到最优的 θ 来使得这些metrics最优。

注意点2:

这些metrics可以被定义为有折旧的情况(discounted case), 即 $\gamma \in [0, 1)$, 也可以是undiscounted case的, 即 $\gamma = 1$

注意点3:

在直观上, \bar{r}_{π} 和 \bar{v}_{π} 相比, 似乎更加“近视”, 因为他更多地考虑立即的reward, 而 \bar{v}_{π} 考虑整个过程的reward。

实则不然, 两个metrics实际上是相互等价的。当 $\gamma < 1$, 有如下公式:

$$\bar{r}_{\pi} = (1 - \gamma)(\bar{v}_{\pi})$$

metric还有一种常见的表示形式如下:

$$J(\theta) = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i R_{t+1}\right]$$

它实际上和 \bar{v}_π 是相同的，下面给出推导过程：

$$\begin{aligned} J(\theta) &= \mathbb{E}\left[\sum_{i=0}^{\infty} (\gamma^i R_{t+1})\right] \\ &= \mathbb{E}\left[\sum_{i=0}^{\infty} (\gamma^i \sum_{s \in S} d(s) R_{t+1,s})\right] \\ &= \sum_{s \in S} (d(s) \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | S_0 = s\right]) \\ &= \sum_{s \in S} d(s) v_\pi(s) = \bar{v}_\pi \end{aligned}$$

目标函数的梯度计算

梯度如下：

$$\nabla_\theta J(\theta) = \sum_{s \in S} \eta(s) \sum_{a \in \mathcal{A}} \nabla_\theta \pi(a|s, \theta) q_\pi(s, a)$$

又可以写作

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \ln \pi(A|S, \theta) q_\pi(S, A)]$$

写成期望的形式便于我们用采样来模拟期望。

下面是推导期望公式的过程：

由链式法则得：

$$\begin{aligned} \nabla_\theta \ln \pi(a|s, \theta) &= \frac{\nabla_\theta \pi(a|s, \theta)}{\pi(a|s, \theta)} \\ \nabla_\theta \pi(a|s, \theta) &= \pi(a|s, \theta) \nabla_\theta \ln \pi(a|s, \theta) \end{aligned}$$

把式子带入梯度表达式中：

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_s d(s) \sum_a \nabla_\theta \pi(a|s, \theta) q_\pi(s, a) \\ &= \sum_s d(s) \sum_a \pi(a|s, \theta) \nabla_\theta \ln \pi(a|s, \theta) q_\pi(s, a) \\ &= \mathbb{E}_{S \sim d} \left[\sum_a \pi(a|S, \theta) q_\pi(S, A) \right] \\ &= \mathbb{E}_{S \sim d, A \sim \pi} [\nabla_\theta \ln \pi(A|S, \theta) q_\pi(S, A)] \\ &= \mathbb{E}[\nabla_\theta \ln \pi(A|S, \theta) q_\pi(S, A)] \end{aligned}$$

因为我们求了 $\ln(\pi(a|s, \theta))$ 所以我们的 π 必须满足 $\pi(a|s, \theta) > 0$,可以用softmax方式来将 $(-\infty, \infty)$ 的值域归一化到 $(0, 1)$

$$z_i = \frac{e^{x_i}}{\sum_{j=1}^b e^{x_j}}$$

同时还满足 $\sum_{i=1}^b z_i = 1$

REINFORCE

reinforce是一种 policy gradient algorithm

根据上一节的目标函数，我们可以得到迭代方程：

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha_{\theta} J(\theta) \\ &= \theta_t + \alpha_{\theta} \mathbb{E}[\nabla_{\theta} \ln \pi(A|S, \theta) q_{\pi}(S, A)]\end{aligned}$$

但我们知道，期望是很难算的，我们要用stochastic方式来代替真实的期望。

$$\theta_{t+1} = \theta_t + \alpha_{\theta} \nabla_{\theta} \ln \pi(a_t | s_t, \theta_t) q_{\pi}(s_t, a_t)$$

但我们的 q_{π} 是不知道的，于是我们可以用 q_t 来代替 q_{π}

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} \ln \pi(a_t | s_t, \theta_t) q_t(s_t, a_t)$$

如何对 q_t 进行采样？

1. 基于蒙德卡罗的方法，即本节的REINFORCE
2. 其他方法

注意一：

如何做采样？

1. 如何对S做采样？ $S \sim d$ ，经过 π 下不断迭代得到 d ,然后采样得到S。但实际上我们没时间等 d 趋于平稳再采样。
2. 如何对A做采样？ $A \sim \pi(A|S, \theta)$ ，根据在 s_t 处的策略 $\pi(\theta_t)$ 来对A进行采样。

因此，这个policy gradient 方法是on-policy的。

注意二：

如何理解算法？

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha \nabla_{\theta} \ln \pi(a_t | s_t, \theta_t) q_t(s_t, a_t) \\ &= \theta_t + \alpha \left(\frac{q_t(s_t, a_t)}{\pi(a_t | s_t, \theta_t)} \right) \nabla_{\theta} \pi(a_t | s_t, \theta_t)\end{aligned}$$

我们设定 $\beta_t = \frac{q_t(s_t, a_t)}{\pi(a_t | s_t, \theta_t)}$

于是原式变为如下：

$$\theta_{t+1} = \theta_t + \alpha \beta_t \nabla_{\theta} \pi(a_t | s_t, \theta_t)$$

于是我们就可以发现，我们是通过改变 θ ，来优化 $\pi(a_t | s_t, \theta)$ 的值。

所以我们步长 $\alpha \beta_t$ 要足够小才能收敛。

- 当 $\beta_t > 0$ ，那么选择 (s_t, a_t) 的可能更大，于是有 $\pi(a_t | s_t, \theta_{t+1}) > \pi(a_t | s_t, \theta_t)$
- 当 $\beta_t < 0$ ，那么 $\pi(a_t | s_t, \theta_{t+1}) < \pi(a_t | s_t, \theta_t)$

数学推导如下：

当 $\theta_{t+1} - \theta_t$ 足够小时，有

$$\begin{aligned}\pi(a_t | s_t, \theta_{t+1}) &\approx \pi(a_t | s_t, \theta_t) + (\nabla_{\theta} \pi(a_t | s_t, \theta_t))^T (\theta_{t+1} - \theta_t) \\ &= \pi(a_t | s_t, \theta_t) + \alpha \beta_t (\nabla_{\theta} \pi(a_t | s_t, \theta_t))^T (\nabla_{\theta} \pi(a_t | s_t, \theta_t)) \\ &= \pi(a_t | s_t, \theta_t) + \alpha \beta_t \|\nabla_{\theta} \pi(a_t | s_t, \theta_t)\|^2\end{aligned}$$

系数 β_t 能够很好地平衡 *exploration* 和 *exploitation*

首先 β_t 与 $q_t(s_t, a_t)$ 成正比。

- 如果 $q_t(s_t, a_t)$ 比较大，那么 β_t 也比较大。
- 因此此时算法倾向于使用更大的值来增强 q_t ，也就是 *exploitation*

其次： β_t 与 $\pi(a_t | s_t, \theta_t)$ 成反比。

- 如果 $\pi(a_t | s_t, \theta_t)$ 很小，那么 β_t 很大。下次的 $\pi(a_t | s_t, \theta_{t+1})$ 更大，就会给出更大的选择概率。
- 因此算法倾向于概率较低的探索操作。即 *exploration*

REINFORCE算法实现如下：

初始化参数 $\pi(a | s, \theta), \gamma \in (0, 1), \alpha > 0$

目标是最大化 $J(\theta)$

对于第 k 次迭代：

1. 选择一个 s_0 , 遵循 $\pi(\theta_k)$ 生成 episode , 假设 episode 是 $\{s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T\}$
2. 对于每个 $t = 0, 1, 2, \dots, T-1$ 执行3和4:
3. value update: $q_t(s_t, a_t) = \sum_{k=t+1}^T \gamma^{k-t-1} r_k$
4. policy update: $\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} \ln \pi(a_t | s_t, \theta_t) q_t(s_t, a_t)$

第十章 actor-critic 方法

actor-critic本身就是policy gradient

The simplest actor-critic

也称QAC（这里的Q是公式中的q，也就是action value）

policy gradient算法:

$$\begin{aligned}\theta_{t+1} &= \theta + \alpha \nabla_{\theta} J(\theta_t) \\ &= \theta_t + \alpha \mathbb{E}_{S \sim \eta, A \sim \pi} [\nabla_{\theta} \ln \pi(A | S, \theta_t) q_{\pi}(S, A)] \\ \theta_{t+1} &= \theta_t + \alpha \nabla_{\theta} \ln \pi(a_t | s_t, \theta_t) q_t(s_t, a_t)\end{aligned}$$

这个更新策略的算法就是actor， critic则用来估计 $q_t(s_t, a_t)$

如何得到 $q_t(s_t, a_t)$?

两种方法:

1. MC learning: 这样结合就得到了REINFORCE算法。
2. Temporal-difference learning: actor-critic算法。

优化目标函数 $J(\theta)$, 使其最大化。

对于每个episode的第t步，执行如下:

1. 遵循 $\pi(a | s_t, \theta_t)$ 生成 a_t , 得到 (r_{t+1}, s_{t+1}) , 然后遵循 $\pi(a | s_{t+1}, \theta_t)$ 生成 a_{t+1}
2. Critic (value update) :

$$w_{t+1} = w_t + \alpha_w [r_{t+1} + \gamma q(s_{t+1}, a_{t+1}, w_t) - q(s_t, a_t, w_t)] \nabla_w q(s_t, a_t, w_t)$$

Actor (policy update):

$$\theta_{t+1} = \theta_t + \alpha_{\theta} \nabla_{\theta} \ln \pi(a_t | s_t, \theta_t) q(s_t, a_t, w_{t+1})$$

这个算法是on-policy 的。

The simplest actor-critic实际上就是 SARSA + value function approximation

advantage actor-critic ::

也叫AAC , A2C

首先我们为policy gradient 引入一个新的baseline (b函数)

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \mathbb{E}_{S \sim \eta, A \sim \pi} [\nabla_{\theta} \ln \pi(A|S, \theta_t) q_{\pi}(S, A)] \\ &= \mathbb{E}_{S \sim \eta, A \sim \pi} [\nabla_{\theta} \ln \pi(A|S, \theta_t) (q_{\pi}(S, A) - b(S))]\end{aligned}$$

为什么引入新的b 函数, 等式依然成立?

因为如下公式成立:

$$\mathbb{E}_{S \sim \eta, A \sim \pi} [\nabla_{\theta} \ln \pi(A|S, \theta_t) b(S)] = 0$$

详细地说:

$$\begin{aligned}\mathbb{E}_{S \sim \eta, A \sim \pi} [\nabla_{\theta} \ln \pi(A|S, \theta_t) b(S)] &= \sum_{s \in S} \eta(s) \sum_{a \in \mathcal{A}} \pi(a|s, \theta_t) \nabla_{\theta} \ln \pi(a|s, \theta_t) b(s) \\ &= \sum_{s \in S} \eta(s) \sum_{a \in \mathcal{A}} \nabla_{\theta} \pi(a|s, \theta_t) b(s) \\ &= \sum_{s \in S} \eta(s) b(s) \sum_{a \in \mathcal{A}} \nabla_{\theta} \pi(a|s, \theta_t) \\ &= \sum_{s \in S} \eta(s) b(s) \nabla_{\theta} \sum_{a \in \mathcal{A}} \pi(a|s, \theta_t) \\ &= \sum_{s \in S} \eta(s) b(s) \nabla_{\theta} 1 = 0\end{aligned}$$

引入这个b函数有什么用?

我们说 $\nabla_{\theta} J(\theta) = \mathbb{E}[X]$

那么我们知道

- $\mathbb{E}[X]$ 和 $b(S)$ 无关。
- X 的方差和 b 有关。

所以我们可以通过设置b函数来减小方差。

设置b函数为如下值时, 能使得方差最小:

$$b^*(s) = \frac{\mathbb{E}_{A \sim \pi} [||\nabla_{\theta} \ln \pi(A|s, \theta_t)||^2 q(s, A)]}{\mathbb{E}_{A \sim \pi} [||\nabla_{\theta} \ln \pi(A|s, \theta_t)||^2]}$$

其中 $||\nabla_{\theta} \ln \pi(A|s, \theta_t)||^2$ 可以被认为是一个权重。

但是这个公式太复杂了。我们一般直接用

$$b(s) = \mathbb{E}_{A \sim \pi}[q(s, A)] = v_{\pi}(s)$$

把上式带入公式中，我们可以得到gradient-ascent算法：

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha \mathbb{E}[\nabla_{\theta} \ln \pi(A|S, \theta_t)(q_{\pi}(S, A) - v_{\pi}(S))] \\ &= \theta_t + \alpha \mathbb{E}[\nabla_{\theta} \ln \pi(A|S, \theta_t)(\delta_{\pi}(S, A))]\end{aligned}$$

我们叫 $\delta_{\pi}(S, A) = q_{\pi}(S, A) - v_{\pi}(S)$ 为advantage function（优势函数）

$v_{\pi}(S)$ 是某个状态下的action的平均值， 所以 $\delta_{\pi}(S, A)$ 描述了当前的action和同状态的其他action相比的优劣。

公式还可以写成下面：

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha \nabla_{\theta} \ln \pi(a_t|s_t, \theta_t) \delta_t(s_t, a_t) \\ &= \theta_t + \alpha \frac{\nabla_{\theta} \pi(a_t|s_t, \theta_t)}{\pi(a_t|s_t, \theta_t)} \delta_t(s_t, a_t) \\ &= \theta_t + \alpha \frac{\delta_t(s_t, a_t)}{\pi(a_t|s_t, \theta_t)} \nabla_{\theta} \pi(a_t|s_t, \theta_t)\end{aligned}$$

于是我们公式中的 $\frac{\delta_t(s_t, a_t)}{\pi(a_t|s_t, \theta_t)}$ 决定了step-size（和第9讲REINFORCE中的 β_t 一样能够很好地平衡*exploration* 和*exploitation*）

A2C，或者TD actor-critic 的过程：

目标是寻找最大的 $J(\theta)$

在每个episode的第t时刻，我们执行如下：

1. 遵循 $\pi(a|s_t, \theta_t)$ 生成 a_t 然后得到 r_{t+1}, s_{t+1}
2. TD error(advantage function):
$$\delta_t = r_{t+1} + \gamma v(s_{t+1}, w_t) - v(s_t, w_t)$$
3. Critic (value update):
$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w v(s_t, w_t)$$
4. Actor(policy update):
$$\theta_{t+1} = \theta_t + \alpha_{\theta} \delta_t \nabla_{\theta} \ln \pi(a_t|s_t, \theta_t)$$

这是一个on-policy 的。

off-policy actor-critic

Policy gradient是on-policy的原因是梯度必须服从 π 策略，这里的 π 既是behavior policy，同时这个 π 也是我们要更新的target policy。

可以使用importance sampling 来把on-policy转为off-policy。

$$\mathbb{E}_{X \sim p_0}[X] = \sum_x p_0(x)x = \sum_x p_1(x) \frac{p_0(x)}{p_1(x)} x = \mathbb{E}_{X \sim p_1}[f(X)]$$

于是我们就可以通过 p_1 进行采样，然后估计 p_0 采样下的均值。那么怎么计算 $\mathbb{E}_{X \sim p_1}[f(X)]$ ？

令 f 为如下函数：

$$f = \frac{1}{n} \sum_{i=1}^n f(x_i), \text{ where } x_i \sim p_i$$

那么就有

$$\begin{aligned}\mathbb{E}_{X \sim p_1}[\bar{f}] &= \mathbb{E}_{X \sim p_1}[f(X)] \\ \text{var}_{X \sim p_1}[\bar{f}] &= \frac{1}{n} \text{var}_{X \sim p_1}[f(X)]\end{aligned}$$

所以 \bar{f} （ f 的平均数）就可以用来估计 $\mathbb{E}_{X \sim p_1}[\bar{f}] = \mathbb{E}_{X \sim p_0}[X]$

$$\mathbb{E}_{X \sim p_0}[X] \approx \bar{f} = \frac{1}{n} \sum_{i=1}^n f(x_i) = \frac{1}{n} \sum_{i=1}^n \frac{p_0(x_i)}{p_1(x_i)} x_i$$

这里的 $\frac{p_0(x_i)}{p_1(x_i)}$ 可以被认为是权重，那么直观地看就是对于 p_0 相对难取的样本，赋予更高的权重。

这个权重叫做 importance权重。

就是因为我们只能知道 $p_0(x)$ ，但求不出 $\mathbb{E}_{X \sim p_0}[X]$ ，所以才需要importance sampling。

假设 β 是behavior policy生成的经验采样。

我们的目标是更新target policy π 来最大化 $J(\theta)$

$$J(\theta) = \sum_{s \in S} d_\beta(s) v_\pi(s) = \mathbb{E}_{S \sim d_\beta}[v_\pi(S)]$$

他的梯度如下：

$$\nabla_\theta J(\theta) = \mathbb{E}_{S \sim \rho, A \sim \beta} \left[\frac{\pi(A|S, \theta)}{\beta(A|S)} \nabla_\theta \ln \pi(A|S, \theta) q_\pi(S, A) \right]$$

这里的 β 是behavior policy， ρ 是state distribution。

优化：

我们仍然可以通过加上baseline来进行优化： $\delta_\pi(S, A) = q_\pi(S, A) - v_\pi(S)$ 。

$$\theta_{t+1} = \theta_t + \alpha_\theta \frac{\pi(a_t|s_t, \theta_t)}{\beta(a_t|s_t)} \nabla_\theta \ln \pi(a_t|s_t, \theta_t) (q_t(s_t, a_t) - v_t(s_t))$$

在这之中

$$q_t(s_t, a_t) - v_t(s_t) \approx r_{t+1} + \gamma v_t(s_{t+1}) - v_t(s_t) = \delta_t(s_t, a_t)$$

于是最终的算法就是

$$\theta_{t+1} = \theta_t + \alpha_\theta \frac{\delta_t(s_t, a_t)}{\beta(a_t|s_t)} \nabla_\theta \ln \pi(a_t|s_t, \theta_t) \pi(a_t|s_t, \theta_t)$$

Deterministic actor-critic

DPG和之前的（QAC，A2C、off-policy的actor-critic）相比的一大特点就是他的策略 $\pi(a|s, \theta)$ 可以是负数。

于是我们用deterministic policies来解决continuous action（无限个的、连续的action）

之前我们是通过策略 $\pi(a|s, \theta) \in [0, 1]$ 来决定要采取哪个动作a。

而现在我们改成下面这样：

$$a = \mu(s, \theta)$$

意味着我们直接通过s得到a的值，而不是借助每一个action的概率来决定选择哪个a。

$$J(\theta) = \mathbb{E}[v_\mu(s)] = \sum_{s \in S} d_0(s) v_\mu(s)$$

d_0 的选择和 μ 无关。

选择 d_0 的两种特殊的情况：

1. $d_0(s_0) = 1, d_0(s \neq s_0) = 0$. 在这里 s_0 是一个特殊的开始状态。
2. d_0 取决于behavior policy 在 μ 上的内容。

$$\begin{aligned} \nabla_\theta J(\theta) &= \rho_\mu(s) \nabla_\theta \mu(s) (\nabla_a q_\mu(s, a))|_{a=\mu(s)} \\ &= \mathbb{E}_{S \sim \rho_\mu} [\nabla_\theta \mu(s) (\nabla_a q_\mu(s, a))|_{a=\mu(s)}] \end{aligned}$$

这里的梯度没有action A。

所以这个deterministic policy gradient 是一个off-policy的方法。（因为我们不需要关心这个a是通过哪个策略得到的）

梯度上升：

$$\theta_{t+1} = \theta_t + \alpha_{\theta} \mathbb{E}_{S \sim \rho_{\mu}} [\nabla_{\theta} \mu(s) (\nabla_a q_{\mu}(s, a)) |_{a=\mu(s)}]$$

$$\theta_{t+1} = \theta_t + \alpha_{\theta} \nabla_{\theta} \mu(s_t) (\nabla_a q_{\mu}(s_t, a)) |_{a=\mu(s)}$$

注意：

- β 和 μ 是不同的。
- β 也可以设置为 $\mu + noise$.

如何选取 $q(s, a, w)$?

1. 线性函数： $q(s, a, w) = \phi^T(s, a)w$
2. 神经网络： DDPG