深度强化学习

强化学习（Reinforcement Learning， RL），也叫增强学习，是指一类从交互中不断学习的问题以及解决这类问题的方法。强化学习问题可以描述为一个智能体从与环境的交互中不断学习以完成特定目标。和深度学习类似，强化学习中的关键问题也是贡献度分配问题，每一个动作并不能直接得到监督信息，需要通过整个模型的最终监督信息（奖励）得到，并且有一定的延时性。

强化学习也是机器学习中的一个重要分支。强化学习和监督学习的不同在于，强化学习问题不需要给出“正确”策略作为监督信息，只需要给出策略的（延迟）回报，并通过调整策略来取得最大化的期望回报。

强化学习问题

典型例子

多臂赌博机问题
悬崖行走问题

强化学习定义

- **智能体**（agent）可以感知外界环境的状态（state）和反馈的奖励（reward），并进行学习和决策。

  智能体的决策功能是指根据外界环境的状态来做出不同的动作（action），而学习功能是指根据外界环境的奖励来调整策略。
- **环境**（environment）是智能体外部的所有事物，并受智能体动作的影响而改变其状态，并反馈给智能体相应的奖励。

  基本要素：
- 状态 s 是对环境的描述，可以是离散的或连续的，其状态空间为 S；
- 动作 a 是对智能体行为的描述，可以是离散的或连续的，其动作空间为 A；
- 策略$\pi(a|s)$是智能体根据环境状态 s 来决定下一步的动作 a 的函数；
- 状态转移概率$p(s'|s,a)$是在智能体根据当前状态 s 做出一个动作 a 之后，环境在下一个时刻转变为状态s'的概率；
- 即时奖励$r(s,a,s')$是一个标量函数，即智能体根据当前状态 s 做出动作 a 之后，环境会反馈给智能体一个奖励，这个奖励也经常和下一个时刻的状态s'有关

策略 智能体的策略（policy）就是智能体如何根据环境状态 s 来决定下一步的动作 a，通常可以分为确定性策略（Deterministic Policy）和随机性策略（Stochastic Policy）。

通常情况下，强化学习一般使用随机性的策略。随机性的策略可以有很多优点。比如在学习时可以通过引入一定随机性更好地探索环境。二是使得策略更加地多样性。

马尔可夫决策过程

马尔可夫过程（Markov Process）是具有马尔可夫性的随机变量序列$s_0, s_1, \cdots, s_t \in S$，其下一个时刻的状态$s_{t+1}$只取决于当前状态$s_t$，

$$p(s_{t+1}|s_t, \cdots, s_0) = p(s_{t+1}|s_t)$$

马尔可夫决策过程（Markov Decision Process， MDP）在马尔可夫过程中加入一个额外的变量：动作 a，即下一个时刻的状态s_{t+1}和当前时刻的状态s_t以及动作a_t相关，

$$p(s_{t+1}|s_t,a_t,\cdots,s_0,a_0)=p(s_{t+1}|s_t,a_t)$$

给定策略π(a|s)，马尔可夫决策过程的一个轨迹（trajectory）$\tau=s_0,a_0,s_1,r_1,a_1,\cdots,s_{T-1},a_{T-1},s_T,r_T$的概率为

$$p(\tau)=p(s_0,a_0,s_1,a_1,\cdots)=p(s_0)\prod_{t=0}^{T-1}\pi(a_t|s_t)p(s_{t+1}|s_t,a_t)$$

强化学习的目标函数

总回报

给定策略π(a|s)，智能体和环境一次交互过程的轨迹τ所收到的累积奖励为总回报（return）。

$$G(\tau)=\sum_{t=0}^{T-1}r_{t+1}=\sum_{t=0}^{T-1}r(s_{t+1},s_t,a_t)$$

假设环境中有一个或多个特殊的终止状态（terminal state），当到达终止状态时，一个智能体和环境的交互过程就结束了。这一轮交互的过程称为一个回合（episode）或试验（trial）。

如果环境中没有终止状态（比如终身学习的机器人），即 T=∞，称为持续性强化学习任务，其总回报也可能是无穷大。为了解决这个问题，我们可以引入一个折扣率来降低远期回报的权重。折扣回报（discounted return）定义为

$$G(\tau)=\sum_{t=0}^{T-1}\gamma^t r_{t+1}$$

目标函数

因为策略和状态转移都有一定的随机性，每次试验得到的轨迹是一个随机序列，其收获的总回报也不一样。强化学习的目标是学习到一个策略π(a|s)来最大化期望回报（expected return），即希望智能体执行一系列的动作来获得尽可能多的平均回报。

$$J(\theta)=E_{\tau\sim p_\theta(\tau)}[G(\tau)]=E_{\tau\sim p_\theta(\tau)}[\sum_{t=0}^{T-1}\gamma^t r_{t+1}]$$

值函数

状态值函数

一个策略π期望回报可以分解为

$$E_{\tau\sim p(\tau)}[G(\tau)]=E_{\tau\sim p(s_0)}\left[E_{\tau\sim p(\tau)}\left[\sum_{t=0}^{T-1}\gamma^t r_{t+1}|\tau_{s0}=s\right]\right]=E_{\tau\sim p(s_0)}[V^\pi(s)]$$

$$V^\pi(s)=E_{\tau\sim p(\tau)}[\sum_{t=0}^{T-1}\gamma^t r_{t+1}|\tau_{s0}=s]$$

如果给定策略π(a|s)，状态转移概率p(s′|s,a)和奖励r(s,a,s′)，我们就可以通过迭代的方式来计算$V^\pi(s)$。由于存在折扣率，迭代一定步数后，每个状态的值函数就会固定不变。

状态-动作值函数

第二个期望是指初始状态为 s 并进行动作 a，然后执行策略π得到的期望总回报，称为状态-动作值函数（state-action value function），

为了方便起见，我们用 $\tau_{0:T}$ 来表示从轨迹 $s_0, a_0, s_1, \cdots, s_T$，用 $\tau_{1:T}$ 表示轨迹 $s_1, a_1, \cdots, s_T$，因此有 $\tau_{0:T} = s_0, a_0, \tau_{1:T}$。

根据马尔可夫性，$V^\pi(s)$ 可展开得到

$$V^\pi(s) = \mathbb{E}_{\tau_{0:T}\sim p(\tau)}\left[r_1 + \gamma \sum_{t=1}^{T-1} \gamma^{t-1} r_{t+1} | \tau_{s_0} = s\right] \tag{14.15}$$

$$= \mathbb{E}_{a\sim\pi(a|s)}\mathbb{E}_{s'\sim p(s'|s,a)}\mathbb{E}_{\tau_{1:T}\sim p(\tau)}\left[r(s,a,s') + \gamma \sum_{t=1}^{T-1} \gamma^{t-1} r_{t+1} | \tau_{s_1} = s'\right] \tag{14.16}$$

$$= \mathbb{E}_{a\sim\pi(a|s)}\mathbb{E}_{s'\sim p(s'|s,a)}\left[r(s,a,s') + \gamma \mathbb{E}_{\tau_{1:T}\sim p(\tau)}\left[\sum_{t=1}^{T-1} \gamma^{t-1} r_{t+1} | \tau_{s_1} = s'\right]\right] \tag{14.17}$$

$$= \mathbb{E}_{a\sim\pi(a|s)}\mathbb{E}_{s'\sim p(s'|s,a)}\left[r(s,a,s') + \gamma V^\pi(s')\right]. \tag{14.18}$$

公式 (14.18) 也称为贝尔曼方程（Bellman equation），表示当前状态的值函数可以通过下个状态的值函数来计算。

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

$$V^\pi(s) = E_{a\sim\pi(a|s)}[Q^\pi(s,a)]$$

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

值函数的作用

值函数可以看作是对策略π的评估。如果在状态 s，有一个动作 a 使得$Q^\pi(s,a) > V^\pi(s)$，说明执行动作 a 比当前的策略π(a|s)要好，我们就可以调整参数使得策略π(a|s)的概率增加。

深度强化学习

将强化学习和深度学习结合在一起，用强化学习来定义问题和优化目标，用深度学习来解决策略和值函数的建模问题，然后使用误差反向传播算法来优化目标函数。深度强化学习在一定程度上具备解决复杂问题的通用智能，并在很多任务上都取得了很大的成功。

基于值函数的学习方法

值函数是对策略π的评估，如果策略π有限（即状态数和动作数都有限）时，可以对所有的策略进行评估并选出最优策略π^*。

$$\forall s, \pi^* = arg\max_\pi V^\pi(s)$$

一种可行的方式是通过迭代的方法不断优化策略，直到选出最优策略。对于一个策略π(a|s)，其 Q 函数为$Q^\pi(s,a)$，我们可以设置一个新的策略π′(a|s)，

$$\pi'(a|s) = \begin{cases} 1, a = arg\max_{\hat{a}} Q^\pi(s,a) \\ 0, otherwise \end{cases}$$

先随机初始化一个策略，计算该策略的值函数，并根据值函数来设置新的策略，然后一直

反复迭代直到收敛。基于值函数的策略学习方法中最关键的是如何计算策略π的值函数，一般有动态规划或蒙特卡罗两种计算方式。

动态规划算法

如果知道马尔可夫决策过程的状态转移概率$p(s'|s,a)$和奖励$r(s,a,s')$，我们直接可以通过贝尔曼方程来迭代计算其值函数。这种模型已知的强化学习算法也称为基于模型的强化学习（Model-Based Reinforcement Learning）算法。

策略迭代

- **策略评估（policy evaluation）**：计算当前策略下，每个状态的值函数。策略评估可以通过贝尔曼方程进行迭代计算$V^{\pi}(s)$。
  如果状态数有限时，也可以通过直接求解 Bellman 方程来得到得到$V^{\pi}(s)$。
- **策略改进（policy improvement）**：根据值函数来更新策略。

算法 14.1: 策略迭代算法

输入: MDP 五元组：$\mathcal{S},\mathcal{A},P,r,\gamma$;

初始化：$\forall s,\forall a,\pi(a|s)=\frac{1}{|\mathcal{A}|}$;

repeat

  // 策略评估

  repeat

    根据贝尔曼方程（公式(14.18)），计算$V^{\pi}(s)$, $\forall s$;

  until $\forall s$, $V^{\pi}(s)$ 收敛;

  // 策略改进

  根据公式(14.19)，计算$Q(s,a)$;

  $\forall s,\pi(s)=\arg\max_a Q(s,a)$;

until $\forall s$, $\pi(s)$ 收敛;

输出: 策略π

值迭代

值迭代（Value Iteration）方法将策略评估和策略改进两个过程合并，来直接计算出最优策略。假设最优策略π^*对应的值函数称为最优值函数，那么最优状态值函数$V^*(s)$和最优状态-动作值函数$Q^*(s,a)$的关系为

$$V^*(s)=\max_a Q^*(s,a)$$

根据贝尔曼方程可知，最优状态值函数$V^*(s)$和最优状态-动作值函数$Q^*(s,a)$也可以进行迭代计算。

$$V^*(s)=\max_a E_{s'\sim p(s'|s,a)}[r(s,a,s')+\gamma V^{\pi}(s')]$$

$$Q^*(s,a)=E_{s'\sim p(s'|s,a)}[r(s,a,s')+\gamma\max_{a'}Q^*(s',a')]$$

策略迭代 VS 值迭代

- 在策略迭代中，每次迭代的时间复杂度最大为$O(|S|^3|A|^3)$，最大迭代次数为$|A|^{|S|}$。而在值迭代中，每次迭代的时间复杂度最大为$O(|S|^2|A|)$，但迭代次数要比策略迭代算法更多。
- 策略迭代是根据贝尔曼方程来更新值函数，并根据当前的值函数来改进策略。而值迭代算法是直接使用贝尔曼最优方程来更新值函数，收敛时的值函数就是最优的值函数，其对应

算法 14.2: 值迭代算法

输入: MDP 五元组：$\mathcal{S},\mathcal{A},P,r,\gamma$;

初始化：$\forall s\in\mathcal{S},V(s)=0$;

repeat

  $\forall s, V(s)\leftarrow\max_a \mathbb{E}_{s'\sim p(s'|s,a)}\left[r(s,a,s')+\gamma V(s')\right]$;

until $\forall s$，$V(s)$ 收敛;

根据公式 (14.19) 计算 $Q(s,a)$;

$\forall s,\pi(s)=\arg\max_a Q(s,a)$;

输出: 策略 π

的策略也就是最优的策略。

- 值迭代和策略迭代都需要经过非常多的迭代次数才能完全收敛。在实际应用中，可以不必等到完全收敛。这样，当状态和动作数量有限时，经过有限次迭代就可以能收敛到近似最优策略。

基于模型的强化学习算法实际上是一种动态规划方法。

- 一是要求模型已知，即要给出马尔可夫决策过程的状态转移概率$p(s'|s,a)$和奖励函数$r(s,a,s')$，这个要求很难满足。如果是事先不知道模型，但仍然希望通过基于模型的学习算法，也可以通过与环境交互来学习出状态转移概率和奖励函数。一个简单的计算模型的方法为 R-max，通过随机游走的方法来探索环境。每次随机一个策略并执行，然后收集状态转移和奖励的样本。在收集一定的样本后，就可以通过统计或监督学习来重构出马尔可夫决策过程。但是，这种基于采样的重构过程的复杂度也非常高，只能应用于状态数非常少的场合。
- 二是效率问题，当状态数量较大的时候，算法的效率比较低。不管是值迭代还是策略迭代，以当前计算机的计算能力，根本无法计算。一个有效的方法是通过一个函数（比如神经网络）来近似计算值函数，以减少复杂度，并提高泛化能力。

蒙特卡罗方法

Q 函数$Q^{\pi}(s,a)$为初始状态为 s，并执行动作 a 后的所能得到的期望总回报，

$$Q^{\pi}(s,a)=E_{\tau\sim p(\tau)}[G(\tau_{s_0=s,a_0=a})]$$

假设我们进行 N 次试验，得到 N 个轨迹$\tau^{(1)},\tau^{(2)},\cdots,\tau^{(N)}$，其总回报分别为$G(\tau^{(1)}),G(\tau^{(2)}),\cdots,G(\tau^{(N)})$。

$$Q^{\pi}(s,a)\approx\hat{Q}^{\pi}(s,a)=\frac{1}{N}\sum_{n=1}^{N}G(\tau^{(n)}_{s_0=s,a_0=a})$$

尽可能覆盖所有的状态和动作，为了平衡利用和探索，我们可以采用 ϵ-贪心法（ϵ-greedy method）。对于一个目标策略π，其对应的 ϵ-贪心法策略为

$$\pi^{\epsilon}(s)=\begin{cases}\pi(s),1-\epsilon\\ \text{随机选择待选项动作},\epsilon\end{cases}$$

- **同策略**　在蒙特卡罗方法中，如果采样策略是$\pi^{\epsilon}(s)$，不断改进策略也是$\pi^{\epsilon}(s)$而不是目标策略$\pi(s)$。这种采样与改进策略相同（即都是$\pi^{\epsilon}(s)$）的强化学习方法叫做同策略（on policy）

方法。

- **异策略** 如果采样策略是$\pi^{\epsilon}(s)$，而优化目标是策略π，可以通过重要性采样，引入重要性权重来实现对目标策略π的优化。这种采样与改进分别使用不同策略的强化学习方法叫做异策略（off policy）方法。

时序差分学习方法

结合了动态规划和蒙特卡罗方法，比仅仅使用蒙特卡罗采样方法的效率要高很多。时序差分学习是模拟一段轨迹，每行动一步(或者几步)，就利用贝尔曼方程来评估行动前状态的价值。当时序差分学习中每次更新的动作数为最大步数时，就等价于蒙特卡罗方法。

首先，将蒙特卡罗方法中 Q 函数$Q^{\pi}(s,a)$的估计改为增量计算的方式，假设第 N 次试验后值函数$\hat{Q}^{\pi}{}_{N}(s,a)$的平均为

$$\hat{Q}_N^{\pi}(s,a) = \frac{1}{N}\sum_{n=1}^{N} G(\tau^{(n)}_{s_0=s,a_0=a}) \tag{14.32}$$

$$= \frac{1}{N}(G(\tau^{(N)}_{s_0=s,a_0=a}) + \sum_{n=1}^{N-1} G(\tau^{(n)}_{s_0=s,a_0=a})) \tag{14.33}$$

$$= \frac{1}{N}\Big(G(\tau^{(N)}_{s_0=s,a_0=a}) + (N-1)\hat{Q}^{\pi}_{N-1}(s,a)\Big) \tag{14.34}$$

$$= \hat{Q}^{\pi}_{N-1}(s,a) + \frac{1}{N}\Big(G(\tau^{(N)}_{s_0=s,a_0=a}) - \hat{Q}^{\pi}_{N-1}(s,a)\Big), \tag{14.35}$$

算法 14.3: SARSA：一种同策略的时序差分学习算法

输入: 状态空间$\mathcal{S}$，动作空间$\mathcal{A}$，折扣率γ，学习率α

随机初始化$Q(s,a)$;
$\forall s, \forall a, \pi(a|s) = \frac{1}{|\mathcal{A}|}$;
repeat
  初始化起始状态s;
  选择动作$a = \pi^{\epsilon}(s)$;
  repeat
    执行动作a，得到即时奖励r和新状态s';
    在状态s'，选择动作$a' = \pi^{\epsilon}(s')$;
    $Q(s,a) \leftarrow Q(s,a) + \alpha\Big(r + \gamma Q(s',a') - Q(s,a)\Big)$;
    更新策略：$\pi(s) = \arg\max_{a\in|\mathcal{A}|} Q(s,a)$;
    $s \leftarrow s', a \leftarrow a'$;
  until s为终止状态;
until $\forall s, a$，$Q(s,a)$收敛;

输出: 策略$\pi(s)$

时序差分学习是强化学习的主要学习方法，其关键步骤就是在每次迭代中优化 Q 函数来减少现实$r+\gamma Q(s',a')$和预期$Q(s,a)$的差距。

时序差分学习和蒙特卡罗方法的主要不同为：蒙特卡罗需要完整一个路径完成才能知道其总回报，也不依赖马尔可夫性质；而时序差分学习只需要一步，其总回报需要依赖马尔可夫性质来进行近似估计。

Q 学习

一种异策略的时序差分学习算法。在 Q 学习中，Q 函数的估计方法为

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

算法 14.4: Q学习：一种异策略的时序差分学习算法

输入: 状态空间$\mathcal{S}$，动作空间$\mathcal{A}$，折扣率γ，学习率α

随机初始化$Q(s,a)$;
$\forall s, \forall a, \pi(a|s) = \frac{1}{|\mathcal{A}|}$;
repeat
  初始化起始状态s;
  repeat
    在状态s，选择动作$a = \pi^{\epsilon}(s)$;
    执行动作a，得到即时奖励r和新状态s';
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$;
    $s \leftarrow s'$;
  until s为终止状态;
until $\forall s, a$，$Q(s,a)$收敛;
输出: 策略$\pi(s) = \arg\max_{a \in |\mathcal{A}|} Q(s,a)$

深度Q网络

我们需要学习一个参数ϕ来使得函数$Q_\phi(\mathbf{s},\mathbf{a})$可以逼近值函数$Q^\pi(s,a)$。如果采用蒙特卡罗方法，就直接让$Q_\phi(\mathbf{s},\mathbf{a})$去逼近平均的总回报$\hat{Q}^\pi(s,a)$；如果采样时序差分方法，就让$Q_\phi(\mathbf{s},\mathbf{a})$去逼近$\mathbb{E}_{\mathbf{s}',\mathbf{a}'}[r + \gamma Q_\phi(\mathbf{s}',\mathbf{a}')]$。

以Q学习为例，采用随机梯度下降，目标函数为

$$L(s,a,s'|\emptyset) = (r + \gamma \max_{a'} Q_\emptyset(s',a') - Q_\emptyset(s,a))^2$$

一是目标不稳定，参数学习的目标依赖于参数本身；二是样本之间有很强的相关性。深度Q网络（deep Q-networks， DQN)。深度Q网络采取两个措施：一是目标网络冻结（freezing target networks)，即在一个时间段内固定目标中的参数，来稳定学习目标；二是经验回放（experience replay)，构建一个经验池来去除数据相关性。经验池是由智能体最近的经历组成的数据集。训练时，随机从经验池中抽取样本来来代替当前的样本用来进行训练。这样，也可以就打破了和相邻训练样本的相似性，避免模型陷入局部最优。经验回放在一定程度上类似于监督学习。先收集样本，然后在这些样本上进行训练。

整体上，在基于值函数的学习方法中，策略一般为确定性的策略。策略优化通常都依赖于值函数。最优策略一般需要遍历当前状态s下的所有动作，并找出最优的$Q(s,a)$。如果动作空间离散但是很大时，那么遍历求最大需要很高的时间复杂度；如果动作空间是连续的并且$Q(s,a)$非凸时，也很难求解出最佳的策略。

基于策略函数的学习方法

策略梯度（policy gradient）是一种基于梯度的强化学习方法。假设$\pi_\theta(a|s)$是一个关于θ的连续可微函数，我们可以用梯度上升的方法来优化参数θ使得目标函数$J(\theta)$最大。

$$\frac{\partial \mathcal{J}(\theta)}{\partial \theta} = \frac{\partial}{\partial \theta} \int p_\theta(\tau) G(\tau) d\tau \tag{14.44}$$

$$= \int \left(\frac{\partial}{\partial \theta} p_\theta(\tau) \right) G(\tau) d\tau \tag{14.45}$$

$$= \int p_\theta(\tau) \left(\frac{1}{p_\theta(\tau)} \frac{\partial}{\partial \theta} p_\theta(\tau) \right) G(\tau) d\tau \tag{14.46}$$

$$= \int p_\theta(\tau) \left(\frac{\partial}{\partial \theta} \log p_\theta(\tau) \right) G(\tau) d\tau \tag{14.47}$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\frac{\partial}{\partial \theta} \log p_\theta(\tau) G(\tau) \right], \tag{14.48}$$

$$\frac{\partial}{\partial \theta} \log p_\theta(\tau) = \frac{\partial}{\partial \theta} \log \left(p(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \right) \tag{14.49}$$

$$= \frac{\partial}{\partial \theta} \left(\log p(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t) \right) \tag{14.50}$$

$$= \sum_{t=0}^{T-1} \frac{\partial}{\partial \theta} \log \pi_\theta(a_t|s_t). \tag{14.51}$$

$$\frac{\partial \mathcal{J}(\theta)}{\partial \theta} = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=0}^{T-1} \frac{\partial}{\partial \theta} \log \pi_\theta(a_t|s_t) \right) G(\tau) \right] \tag{14.52}$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=0}^{T-1} \frac{\partial}{\partial \theta} \log \pi_\theta(a_t|s_t) \right) \left(G(\tau_{0:t-1}) + \gamma^t G(\tau_{t:T}) \right) \right] \tag{14.53}$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-1} \left(\frac{\partial}{\partial \theta} \log \pi_\theta(a_t|s_t) \gamma^t G(\tau_{t:T}) \right) \right], \tag{14.54}$$

REINFORCE 算法

算法 14.6: REINFORCE 算法

输入: 状态空间 $\mathcal{S}$，动作空间 $\mathcal{A}$，可微分的策略函数 $\pi_\theta(a|s)$，折扣率 γ，学习率 α;

随机初始化参数 θ;

repeat

  根据策略 $\pi_\theta(a|s)$ 生成一条轨迹 $\tau = s_0, a_0, s_1, a_1, \cdots, s_{T-1}, a_{T-1}, s_T$;

  for t=0 **to** T **do**

    计算 $G(\tau_{t:T})$;

    // 更新策略函数参数

    $\theta \leftarrow \theta + \alpha \gamma^t G(\tau_{t:T}) \frac{\partial}{\partial \theta} \log \pi_\theta(a_t|s_t)$;

  end

until θ 收敛;

输出: 策略 π_θ

带基准线的 REINFORCE 算法

REINFORCE 算法的一个主要缺点是不同路径之间的方差很大，导致训练不稳定，这是在高维空间中使用蒙特卡罗方法的的通病。一种减少方差的通用方法是引入一个控制变量。假设要估计函数 f 的期望，为了减少 f 的方差，我们引入一个已知期望的函数 g。

带基准线的 REINFORCE 算法 在每个时刻 t，其策略梯度为

$$\frac{\partial \mathcal{J}_t(\theta)}{\partial \theta} = \mathbb{E}_{s_t}\left[\mathbb{E}_{a_t}\left[\gamma^t G(\tau_{t:T})\frac{\partial}{\partial \theta}\log \pi_\theta(a_t|s_t)\right]\right]. \tag{14.62}$$

为了减小策略梯度的方差，我们引入一个和 a_t 无关的基准函数 $b(s_t)$，

$$\frac{\partial \hat{\mathcal{J}}_t(\theta)}{\partial \theta} = \mathbb{E}_{s_t}\left[\mathbb{E}_{a_t}\left[\gamma^t \Big(G(\tau_{t:T}) - b(s_t)\Big)\frac{\partial}{\partial \theta}\log \pi_\theta(a_t|s_t)\right]\right]. \tag{14.63}$$

因为 $b(s_t)$ 和 a_t 无关，有

$$\begin{aligned}
\mathbb{E}_{a_t}\left[b(s_t)\frac{\partial}{\partial \theta}\log \pi_\theta(a_t|s_t)\right] &= \int_{a_t}\left(b(s_t)\frac{\partial}{\partial \theta}\log \pi_\theta(a_t|s_t)\right)\pi_\theta(a_t|s_t)da_t && (14.64)\\
&= \int_{a_t} b(s_t)\frac{\partial}{\partial \theta}\pi_\theta(a_t|s_t))da_t && (14.65)\\
&= \frac{\partial}{\partial \theta}b(s_t)\int_{a_t}\pi_\theta(a_t|s_t))da_t && (14.66)\\
&= \frac{\partial}{\partial \theta}\big(b(s_t)\cdot 1\big) = 0. && (14.67)
\end{aligned}$$

为了可以有效地减小方差，$b(s_t)$ 和 $G(\tau_{t:T})$ 越相关越好，一个很自然的选择是令 $b(s_t)$ 为值函数 $V^{\pi_\theta}(s_t)$。但是由于值函数未知，我们可以用一个可学习的函数 $V_\phi(s_t)$ 来近似值函数，目标函数为

$$\mathcal{L}(\phi|s_t,\pi_\theta) = \Big(V^{\pi_\theta}(s_t) - V_\phi(s_t)\Big)^2, \tag{14.68}$$

其中 $V^{\pi_\theta}(s_t) = \mathbb{E}[G(\tau_{t:T})]$ 也用蒙特卡罗方法进行估计。采用随机梯度下降法，参数 ϕ 的梯度为

$$\frac{\partial \mathcal{L}(\phi|s_t,\pi_\theta)}{\partial \phi} = -\Big(G(\tau_{t:T}) - V_\phi(s_t)\Big)\frac{\partial V_\phi(s_t)}{\partial \phi}. \tag{14.69}$$

策略函数参数 θ 的梯度为

$$\frac{\partial \hat{\mathcal{J}}_t(\theta)}{\partial \theta} = \mathbb{E}_{s_t}\left[\mathbb{E}_{a_t}\left[\gamma^t \Big(G(\tau_{t:T}) - V_\phi(s_t)\Big)\frac{\partial}{\partial \theta}\log \pi_\theta(a_t|s_t)\right]\right]. \tag{14.70}$$

Actor-Critic 算法

在 Actor-Critic 算法中的策略函数$\pi_\theta(s,a)$和值函数$V_\emptyset(s)$都是待学习的函数，需要在训练过程中同时学习。

假设从时刻 t 开始的回报$G(\tau_{t:T})$，我们用下面公式近似计算。

$$\widehat{G}(\tau_{t:T}) = r_{t+1} + \gamma V_\emptyset(s_{t+1})$$

在每步更新中，分别进行策略函数$\pi_\theta(s,a)$和值函数$V_\emptyset(s)$的学习。一方面，更新参数 ϕ 使得值函数$V_\emptyset(s_t)$接近于估计的真实回报$\widehat{G}(\tau_{t:T})$，

$$\min_{\emptyset}(\widehat{G}(\tau_{t:T}) - V_{\emptyset}(s_t))^2$$

另一方面，将值函数$V_{\emptyset}(s_t)$作为基函数来更新参数 θ，减少策略梯度的方差。

$$\theta = \theta + \alpha\gamma^t(\widehat{G}(\tau_{t:T}) - V_{\emptyset}(s_t))\frac{\partial}{\partial\theta}log\pi_\theta(a_t|s_t)$$

算法 14.8: actor-critic 算法

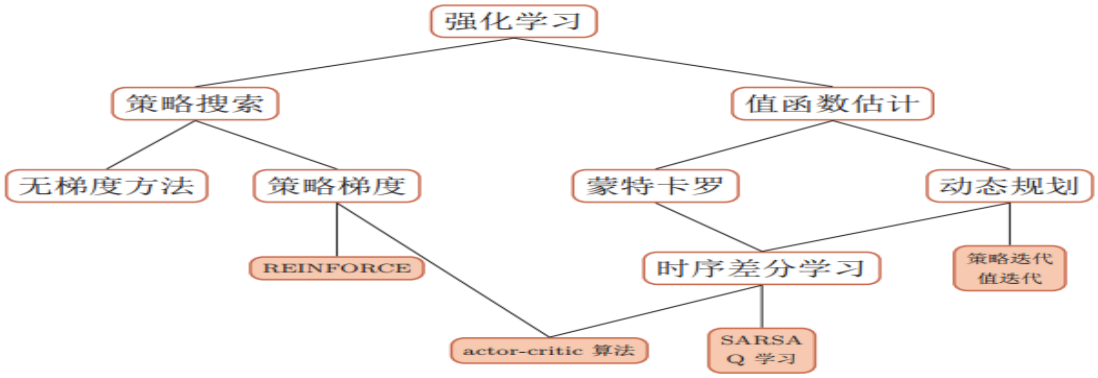
输入: 状态空间 $\mathcal{S}$，动作空间 $\mathcal{A}$;
可微分的策略函数 $\pi_\theta(a|s)$;
可微分的状态值函数 $V_\phi(s)$;
折扣率 γ，学习率 $\alpha > 0, \beta > 0$;

随机初始化参数 θ, ϕ;
repeat
  初始化起始状态 s;
  $\lambda = 1$;
  repeat
    在状态 s，选择动作 $a = \pi_\theta(a|s)$;
    执行动作 a，得到即时奖励 r 和新状态 s';
    $\delta \leftarrow r + \gamma V_\phi(s') - V_\phi(s)$;
    $\phi \leftarrow \phi + \beta\delta\frac{\partial}{\partial\phi}V_\phi(s)$;
    $\theta \leftarrow \theta + \alpha\lambda\delta\frac{\partial}{\partial\theta}\log\pi_\theta(a|s)$;
    $\lambda \leftarrow \gamma\lambda$;
    $s \leftarrow s'$;
  until s 为终止状态;
until θ 收敛;

输出: 策略 π_θ

强化学习和监督学习的区别在于：（1）强化学习的样本通过不同与环境进行交互产生，即试错学习，而监督学习的样本由人工收集并标注；（2）强化学习的反馈信息只有奖励，并且是延迟的；而监督学习需要明确的指导信息（每一个状态对应的动作）。

强化学习的算法非常多，大体上可以分为基于值函数的方法（包括动态规划、时序差分学习等）、基于策略函数的方法（包括策略梯度等）以及融合两者的方法。



基于值函数的方法策略更新时可能会导致值函数的改变比较大，对收敛性有一定影响，而基于策略函数的方法在策略更新时更加更平稳些。但后者因为策略函数的解空间比较大，难以进行充分的采样，导致方差较大，并容易收敛到局部最优解。Actor-Critic 算法通过融合两种方法，取长补短，有着更好的收敛性。

算法	步骤
SARSA	（1）执行策略，生成样本：s, a, r, s', a' （2）估计回报：$Q(s,a) \leftarrow Q(s,a) + \alpha\Big(r + \gamma Q(s',a') - Q(s,a)\Big)$ （3）更新策略：$\pi(s) = \arg\max_{a \in \lvert\mathcal{A}\rvert} Q(s,a)$
Q 学习	（1）执行策略，生成样本：s, a, r, s' （2）估计回报：$Q(s,a) \leftarrow Q(s,a) + \alpha\Big(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big)$ （3）更新策略：$\pi(s) = \arg\max_{a \in \lvert\mathcal{A}\rvert} Q(s,a)$
REINFORCE	（1）执行策略，生成样本：$\tau = s_0, a_0, s_1, a_1, \cdots$ （2）估计回报：$G(\tau) = \sum_{t=0}^{T-1} r_{t+1}$ （3）更新策略：$\theta \leftarrow \theta + \sum_{t=0}^{T-1} \Big(\frac{\partial}{\partial\theta} \log \pi_\theta(a_t \lvert s_t) \gamma^t G(\tau_{t:T})\Big)$
Actor-Critic	（1）执行策略，生成样本：s, a, s', r （2）估计回报：$G(s) = r + \gamma V_\phi(s')$ $\phi \leftarrow \phi + \beta\Big(G(s) - V_\phi(s)\Big) \frac{\partial}{\partial\phi} V_\phi(s)$ （3）更新策略：$\lambda \leftarrow \gamma\lambda$ $\theta \leftarrow \theta + \alpha\lambda\Big(G(s) - V_\phi(s)\Big) \frac{\partial}{\partial\theta} \log \pi_\theta(a \lvert s)$

部分可观测马尔可夫决策过程 部分可观测马尔可夫决策过程（Partially Observable Markov Decision Processes，POMDP）是一个马尔可夫决策过程的泛化。POMDP 依然具有马尔可夫性，但是假设智能体无法感知环境的状态 s，只能知道部分观测值 o。比如在自动驾驶中，智能体只能感知传感器采集的有限的环境信息。

POMDP 可以用一个 7 元组描述：$(\mathcal{S}, \mathcal{A}, T, \mathcal{R}, \Omega, \mathcal{O}, \gamma)$，其中 $\mathcal{S}$ 表示状态空间，为隐变量，$\mathcal{A}$ 为动作空间，$T(s'|s,a)$ 为状态转移概率，$\mathcal{R}$ 为奖励函数，$\Omega(o|s,a)$ 为观测概率，$\mathcal{O}$ 为观测空间，γ 为折扣系数。

逆向强化学习 强化学习的基础是智能体可以和环境进行交互，得到奖励。但在某些情况下，智能体无法从环境得到奖励，只有一组轨迹示例（demonstration）。比如在自动驾驶中，我们可以得到司机的一组轨迹数据，但并不知道司机在每个时刻得到的即时奖励。虽然我们可以用监督学习来解决，称为行为克隆。但行为克隆只是学习司机的行为，并没有深究司机行为的动机。

逆向强化学习（Inverse Reinforcement Learning，IRL）就是指一个不带奖励的马尔可夫决策过程，通过给定的一组专家（或教师）的行为轨迹示例来逆向估计出奖励函数 $r(s,a,s')$ 来解释专家的行为，然后再进行强化学习。

分层强化学习 分层强化学习（Hierarchical Reinforcement Learning，HRL）是指将一个复杂的强化学习问题分解成多个小的、简单的子问题 [Barto and Mahadevan, 2003]，每个子问题都可以单独用马尔可夫决策过程来建模。这样，我们可以将智能体的策略分为高层次策略和低层次策略，高层次策略根据当前状态决定如何执行低层次策略。这样，智能体就可以解决一些非常复杂的任务。