

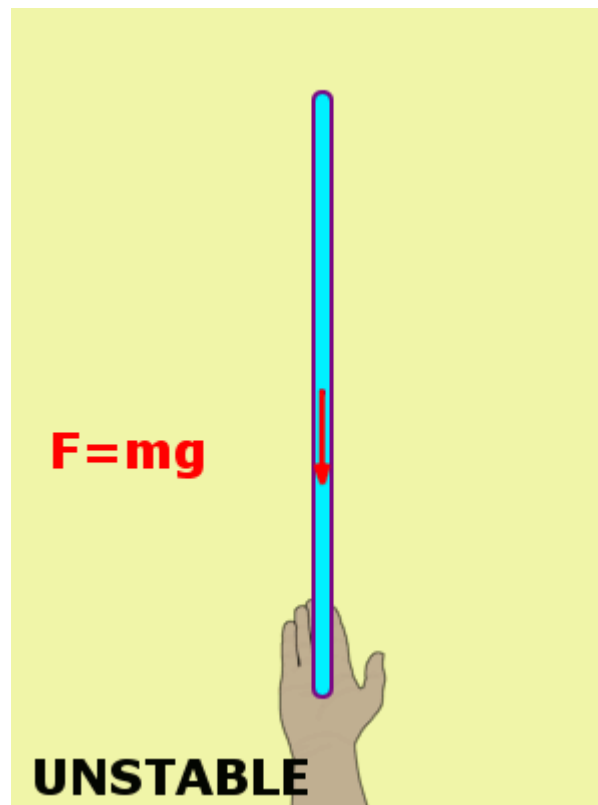
《神经网络与深度学习》



深度强化学习

<https://nndl.github.io/>

一个例子



强化学习

▶智能体（Agent）

- ▶感知外界环境的状态（State）和奖励反馈（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）

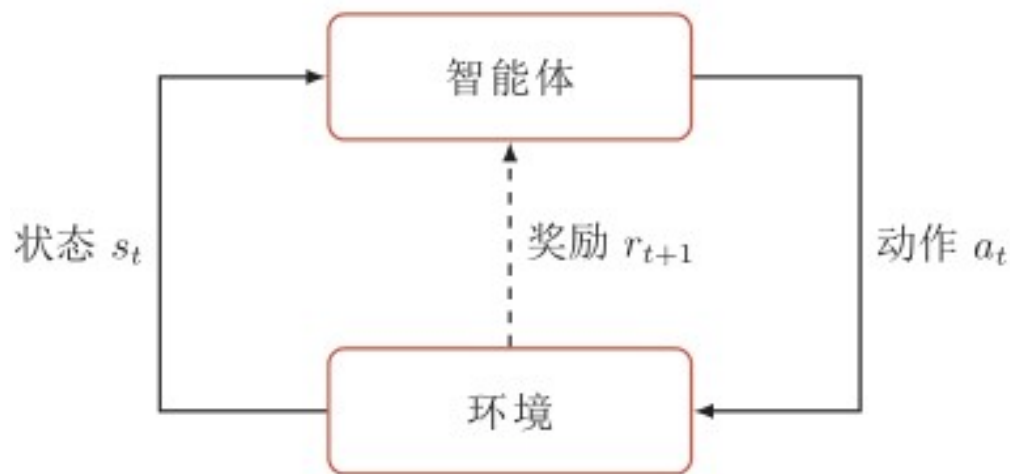
- ▶智能体外部的所有事物，并受智能体动作的影响而改变其状态，并反馈给智能体相应的奖励。

强化学习中的基本要素

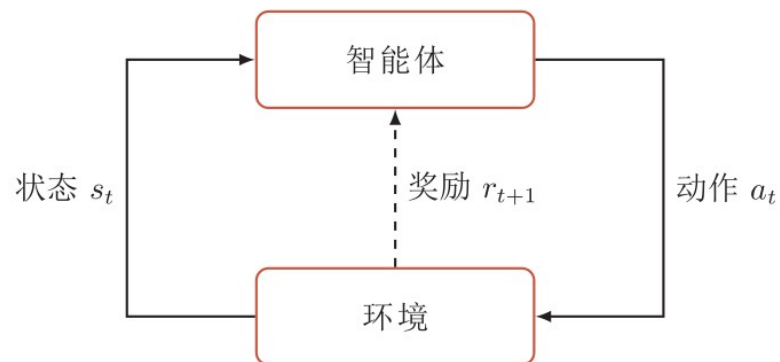
- 环境的**状态集合**： S ；
- 智能体的**动作集合**： A ；
- **状态转移概率**： $p(s' | s, a)$ ，即智能体根据当前状态 s 做出一个动作 a 之后，下一个时刻环境处于不同状态 s' 的概率；
- **即时奖励**： $R: S \times A \times S' \rightarrow R$ ，即智能体根据当前状态做出一个动作之后，环境会反馈给智能体一个奖励，这个奖励和动作之后下一个时刻的状态有关。

强化学习

- ▶ 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（比如取得最大奖励值）。
- ▶ 强化学习就是智能体不断与环境进行交互，并根据经验调整其策略来最大化其长远的所有奖励的累积值。



马尔可夫决策过程



$$s_0, a_0, s_1, r_1, a_1, \dots, s_{t-1}, r_{t-1}, a_{t-1}, s_t, r_t, \dots,$$

马尔可夫过程

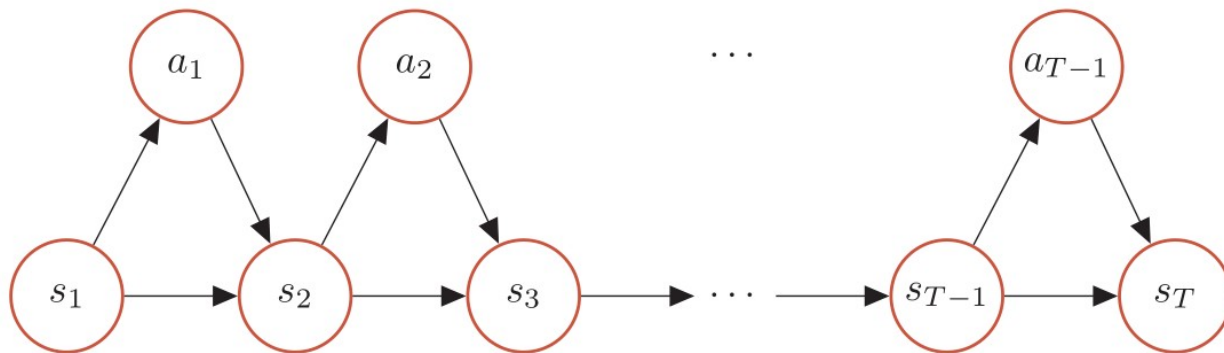
$$p(s_{t+1} | s_t, a_t, \dots, s_0, a_0) = p(s_{t+1} | s_t, a_t),$$

策略 $\pi(a|s)$

► 马尔可夫决策过程的一个轨迹 (trajectory)

$$\tau = s_0, a_0, s_1, r_1, a_1, \dots, s_{T-1}, a_{T-1}, s_T, r_T$$

► τ 的概率



$$\begin{aligned} p(\tau) &= p(s_0, a_0, s_1, a_1, \dots), \\ &= p(s_0) \prod_{t=0}^{T-1} \pi(a_t | s_t) p(s_{t+1} | s_t, a_t). \end{aligned}$$

总回报

- ▶ 给定策略 $\pi(a|s)$ ，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）

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

- ▶ $\gamma \in [0,1]$ 是折扣率。当 γ 接近于 0 时，智能体更在意短期回报；而当 γ 接近于 1 时，长期回报变得更重要。
- ▶ 环境中有一个或多个特殊的终止状态（terminal state）

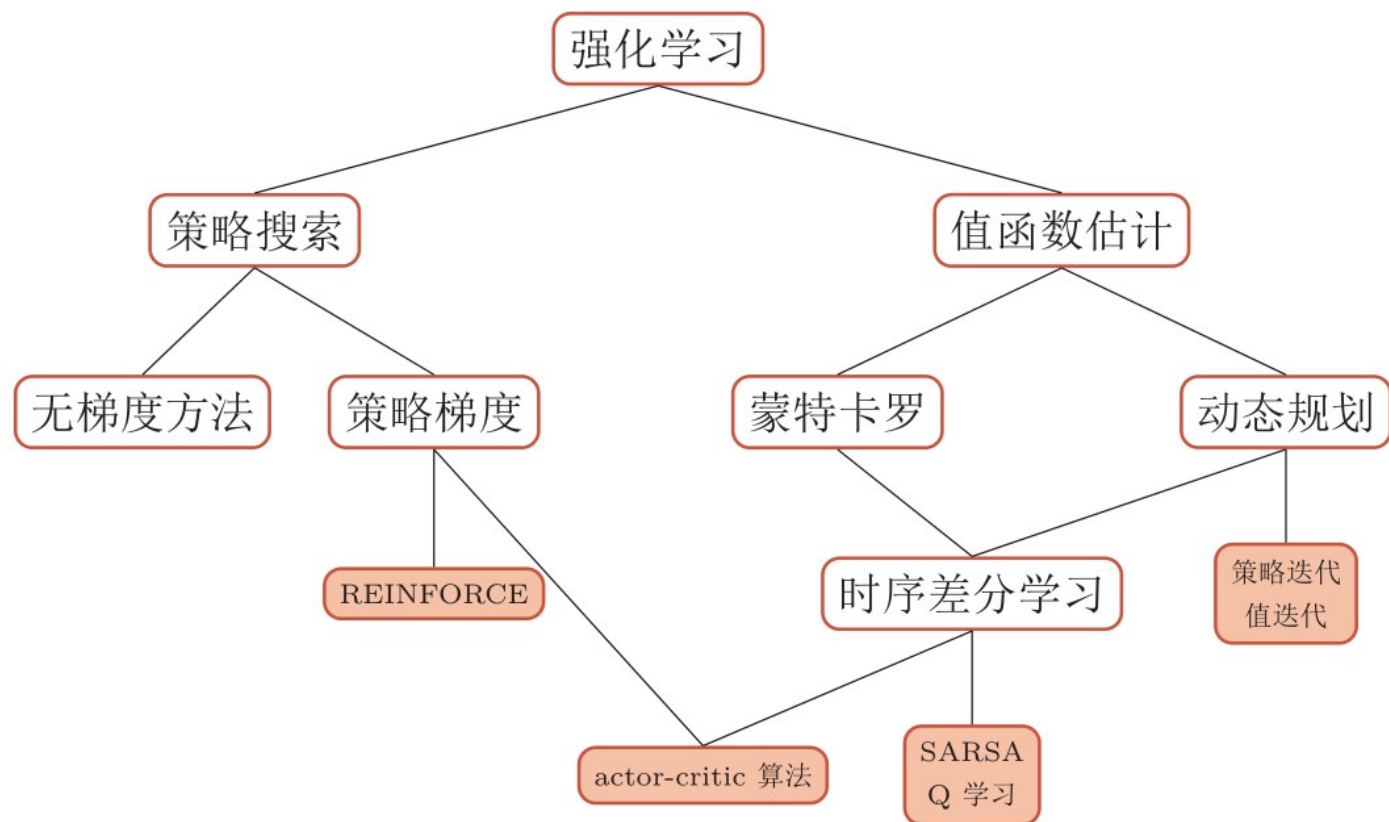
强化学习目标函数

- 强化学习的目标是学习到一个策略 $\pi_{\theta}(a|s)$ 来最大化期望回报（expected return）

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

- θ 为策略函数的参数

强化学习的算法



深度强化学习

- ▶ 深度强化学习是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，用深度学习来解决状态表示、策略表示等问题。
- ▶ 两种不同的结合强化学习和深度学习的方式，分别用深度神经网络来建模强化学习中的值函数、策略，然后用误差反向传播算法来优化目标函数。



基于值函数的策略学习

如何评估策略 $\pi_{\theta}(a|s)$?

- ▶ 两个值函数
 - ▶ 状态值函数
 - ▶ 状态 - 动作值函数

状态值函数

- ▶ 一个策略 π 期望回报可以分解为

$$\begin{aligned}\mathbb{E}_{\tau \sim p(\tau)}[G(\tau)] &= \mathbb{E}_{s \sim p(s_0)} \left[\mathbb{E}_{\tau \sim p(\tau)} \left[\sum_{t=0}^{T-1} \gamma^t r_{t+1} \mid \tau_{s_0} = s \right] \right] \\ &= \mathbb{E}_{s \sim p(s_0)} [V^\pi(s)],\end{aligned}$$

- ▶ 值函数：从状态 s 开始，执行策略 π 得到的期望总回报

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

Bellman 方程

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{\tau_{0:T} \sim p(\tau)} \left[r_1 + \gamma \sum_{t=1}^{T-1} \gamma^{t-1} r_{t+1} \mid \tau_{s_0} = s \right] \\ &= \mathbb{E}_{a \sim \pi(a|s)} \mathbb{E}_{s' \sim p(s'|s,a)} \mathbb{E}_{\tau_{1:T} \sim p(\tau)} \left[r(s, a, s') + \gamma \sum_{t=1}^{T-1} \gamma^{t-1} r_{t+1} \mid \tau_{s_1} = s' \right] \\ &= \mathbb{E}_{a \sim \pi(a|s)} \mathbb{E}_{s' \sim p(s'|s,a)} \left[r(s, a, s') + \gamma \mathbb{E}_{\tau_{1:T} \sim p(\tau)} \left[\sum_{t=1}^{T-1} \gamma^{t-1} r_{t+1} \mid \tau_{s_1} = s' \right] \right] \\ &= \mathbb{E}_{a \sim \pi(a|s)} \mathbb{E}_{s' \sim p(s'|s,a)} \left[r(s, a, s') + \gamma V^\pi(s') \right]. \end{aligned}$$

状态 - 动作值函数 (Q 函数)

- ▶ 状态 - 动作值函数是指初始状态为 s 并进行动作 a ，然后执行策略 π 得到的期望总回报。

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

- ▶ Q 函数的贝尔曼方程

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

最优策略

- ▶ **最优策略：** 存在一个最优的策略 π^* ， 其在所有状态上的期望回报最大。

$$\forall s, \quad \pi^* = \arg \max_{\pi} V^{\pi}(s).$$

- ▶ 难以实现
- ▶ **策略改进：**
 - ▶ 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 $\pi(a|s)$ 要好， 我们就可以调整参数使得策略 $\pi(a|s)$ 的概率增加。

如何计算值函数？

► 基于模型的强化学习算法

- 基于 MDP 过程：状态转移概率 $p(s' | s, a)$ 和奖励函数 $R(s, a, s')$
- 策略迭代
- 值迭代

► 模型无关的强化学习

- 无 MDP 过程
- 蒙特卡罗采样方法
- 时序差分学习



基于模型的强化学习

策略迭代

算法 15.1: 策略迭代算法

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

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

2 **repeat**

 // 策略评估

3 **repeat**

4 | 根据贝尔曼方程 (公式 (15.18)), 计算 $V^\pi(s)$, $\forall s$;

5 **until** $\forall s$, $V^\pi(s)$ 收敛;

 // 策略改进

6 根据公式 (15.19), 计算 $Q(s, a)$;

7 $\forall s, \pi(s) = \arg \max_a Q(s, a)$;

8 **until** $\forall s$, $\pi(s)$ 收敛;

输出: 策略 π

值迭代

- 值迭代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。

算法 15.2: 值迭代算法

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

1 初始化: $\forall s \in \mathcal{S}, V(s) = 0$;

2 **repeat**

3 $\forall s, V(s) \leftarrow \max_a \mathbb{E}_{s' \sim p(s'|s,a)} \left[r(s, a, s') + \gamma V(s') \right]$;

4 **until** $\forall s, V(s)$ 收敛;

5 根据公式 (15.19) 计算 $Q(s, a)$;

6 $\forall s, \pi(s) = \arg \max_a Q(s, a)$;

输出: 策略 π



模型无关的强化学习

蒙特卡罗采样方法

► 策略学习过程

► 通过采样的：

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

► 当 $N \rightarrow \infty$ 时， $\hat{Q}_{\pi}(s, a) \rightarrow Q_{\pi}(s, a)$ 。

- 在似估计出 $Q_{\pi}(s, a)$ 之后，就可以进行策略改进。
- 然后在新的策略下重新通过采样来估计 Q 函数，并不断重复，直至收敛。

ϵ - 贪心法

► 利用和探索

- 对当前策略的利用（Exploitation），
- 对环境的探索（Exploration）以找到更好的策略
- 对于一个确定性策略 π ，其对应的 ϵ - 贪心法策略为

$$\pi^\epsilon(s) = \begin{cases} \pi(s), & \text{按概率 } 1 - \epsilon, \\ \text{随机选择 } \mathcal{A} \text{ 中的动作}, & \text{按概率 } \epsilon. \end{cases}$$

时序差分学习方法

结合了动态规划和蒙特卡罗方法

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



蒙特卡罗误差

$$\hat{Q}^\pi(s, a) \leftarrow \hat{Q}^\pi(s, a) + \alpha \left(G(\tau_{s_0=s, a_0=a}) - \hat{Q}^\pi(s, a) \right)$$

$$r(s, a, s') + \gamma \hat{Q}^\pi(s', a')$$

从 s, a 开始，采样下一步的状态和动作 (s', a') ，并得到奖励 $r(s, a, s')$ ，然后利用贝尔曼方程来近似估计 $G(\tau)$

SARSA 算法 (State Action Reward State Action , SARSA)

算法 15.3: 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

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

Q 学习算法

► Q 学习算法不通过 π^ϵ 来选下一步的动作 a' ，而是直接选最优的 Q 函数，

算法 15.4: Q 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

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

基于值函数的深度强化学习

- 为了在连续的状态和动作空间中计算值函数 $Q_{\pi}(s,a)$ ，我们可以用一个函数 $Q_{\phi}(s,a)$ 来表示近似计算，称为值函数近似（Value Function Approximation）

$$Q_{\phi}(\mathbf{s}, \mathbf{a}) \approx Q^{\pi}(s, a)$$

目标函数

$$\mathcal{L}(s, a, s'; \phi) = \left(r + \gamma \max_{a'} Q_{\phi}(s', a') - Q_{\phi}(s, a) \right)^2$$

► 存在两个问题：

- 目标不稳定，参数学习的目标依赖于参数本身；
- 样本之间有很强的相关性。

► 深度 Q 网络

- 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。

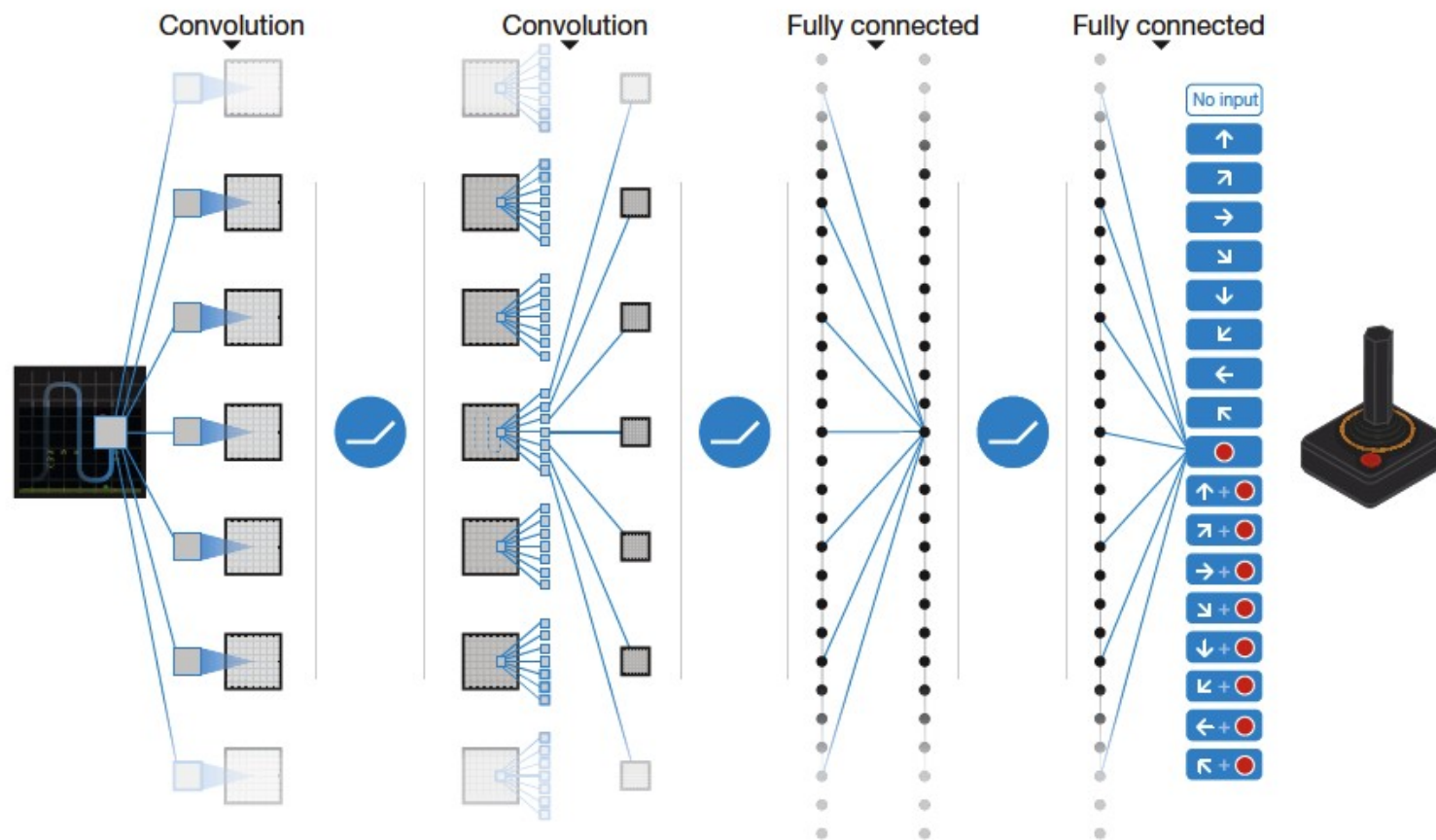
算法 15.5: 带经验回放的深度 Q 网络

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

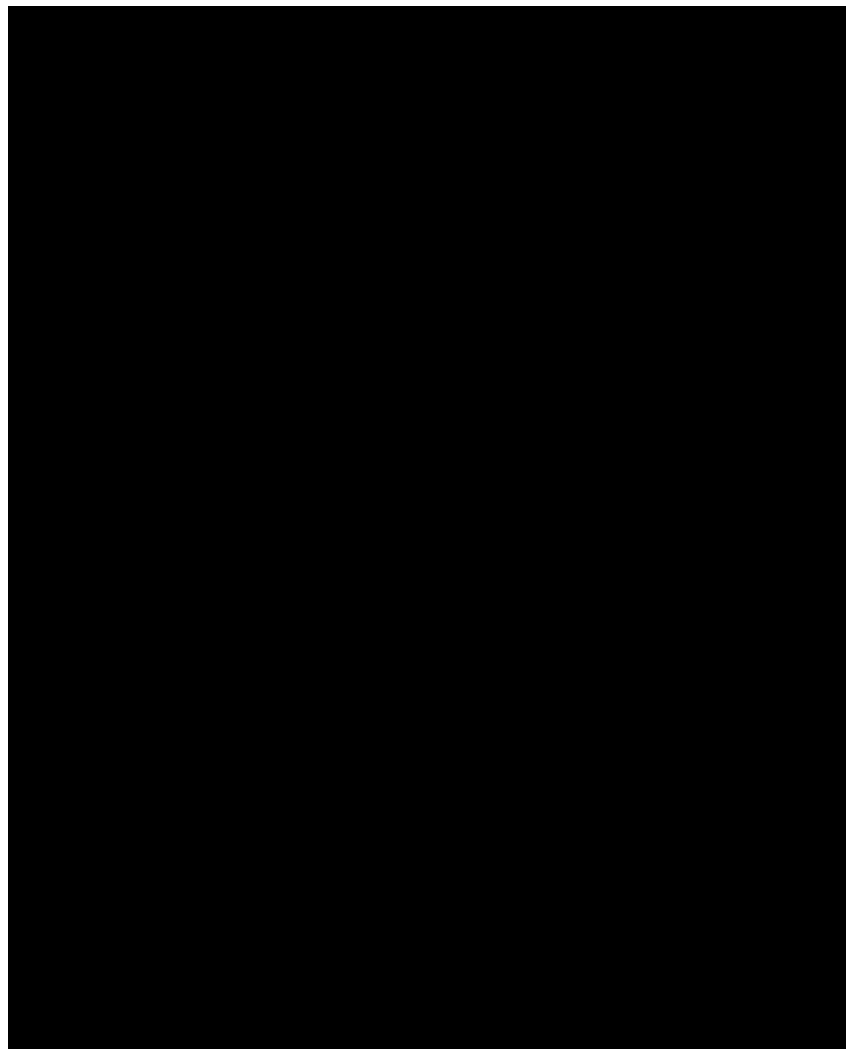
- 1 初始化经验池 $\mathcal{D}$, 容量为 N ;
 - 2 随机初始化 Q 网络的参数 ϕ ;
 - 3 随机初始化目标 Q 网络的参数 $\hat{\phi} = \phi$;
 - 4 **repeat**
 - 5 初始化起始状态 s ;
 - 6 **repeat**
 - 7 在状态 s , 选择动作 $a = \pi^\epsilon$;
 - 8 执行动作 a , 观测环境, 得到即时奖励 r 和新的状态 s' ;
 - 9 将 s, a, r, s' 放入 $\mathcal{D}$ 中;
 - 10 从 $\mathcal{D}$ 中采样 ss, aa, rr, ss' ;
 - 11
$$y = \begin{cases} rr, & ss' \text{ 为终止状态,} \\ rr + \gamma \max_{a'} Q_{\hat{\phi}}(ss', a'), & \text{否则} \end{cases};$$
 - 12 以 $(y - Q_{\phi}(s, a))^2$ 为损失函数来训练 Q 网络;
 - 13 $s \leftarrow s'$;
 - 14 每隔 C 步, $\hat{\phi} \leftarrow \phi$;
 - 15 **until** s 为终止状态;
 - 16 **until** $\forall s, a, Q_{\phi}(s, a)$ 收敛;
- 输出:** Q 网络 $Q_{\phi}(s, a)$
-

DQN in Atari

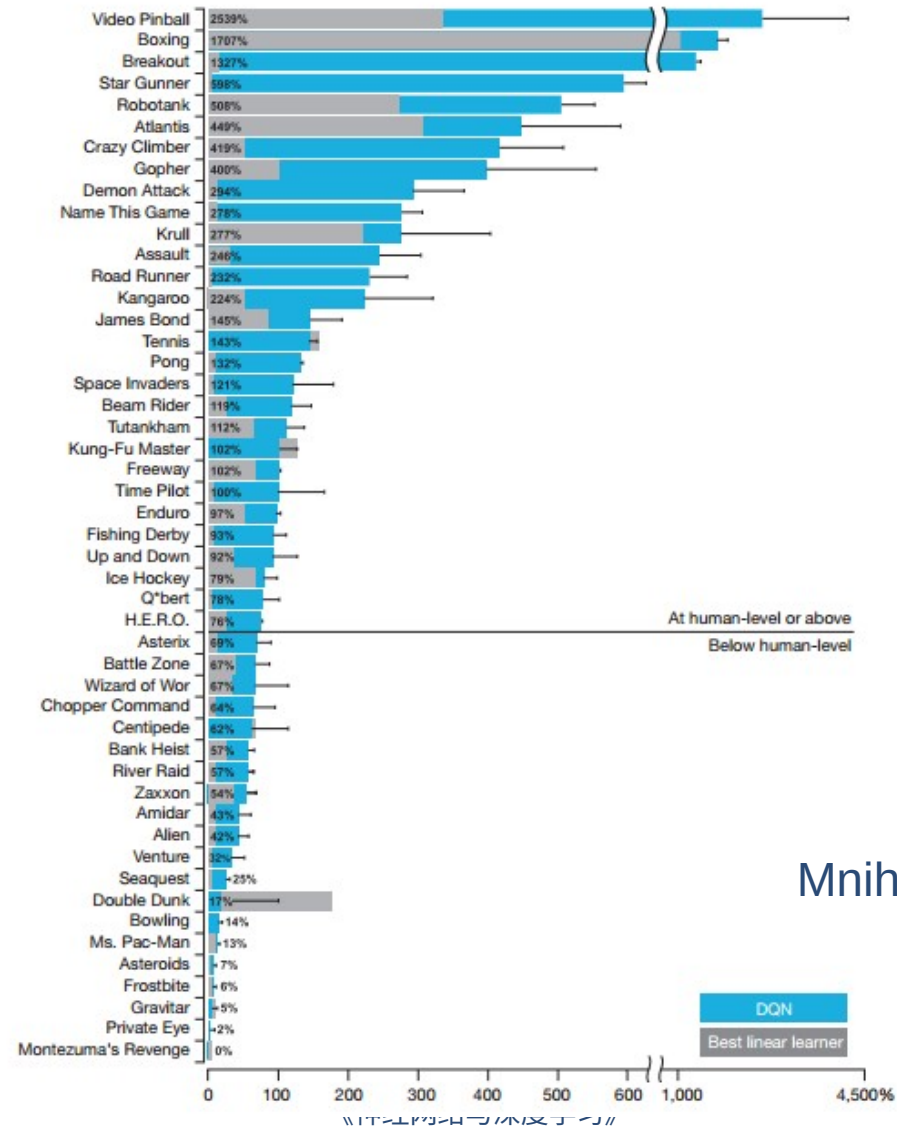
Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." *Nature* 518.7540 (2015): 529-533.



DQN in Atari



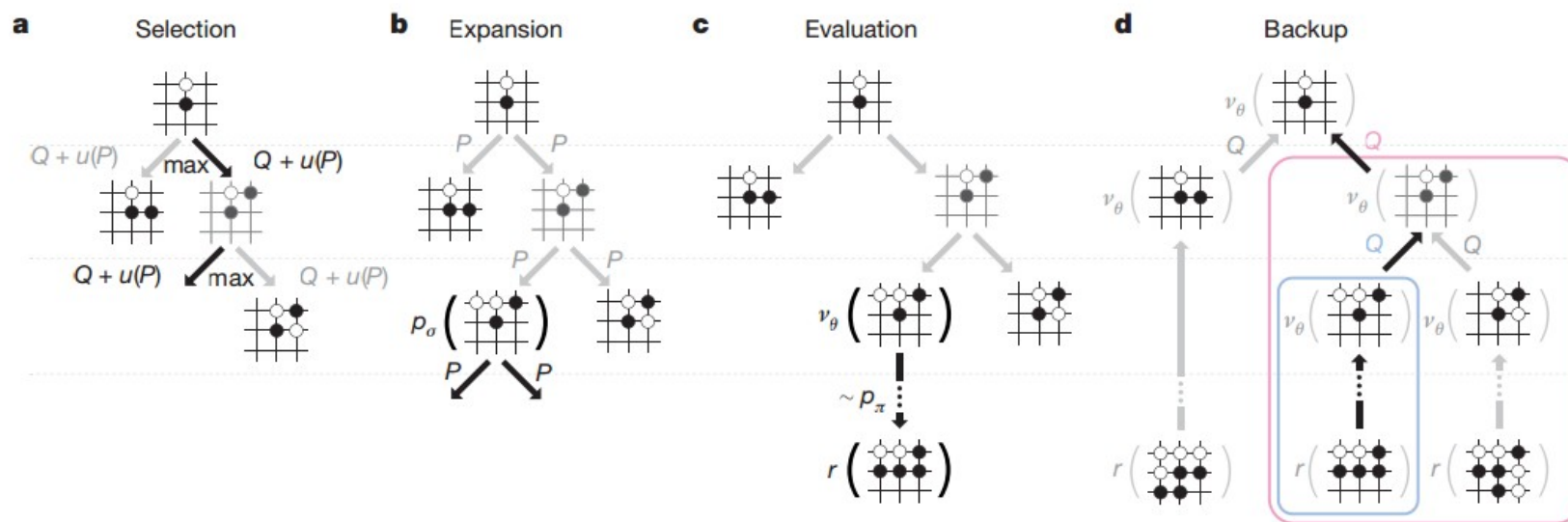
DQN in Atari : Human Level Control



Mnih, Volodymyr, et al. 2015.

AlphaGO: Monte Carlo Tree Search

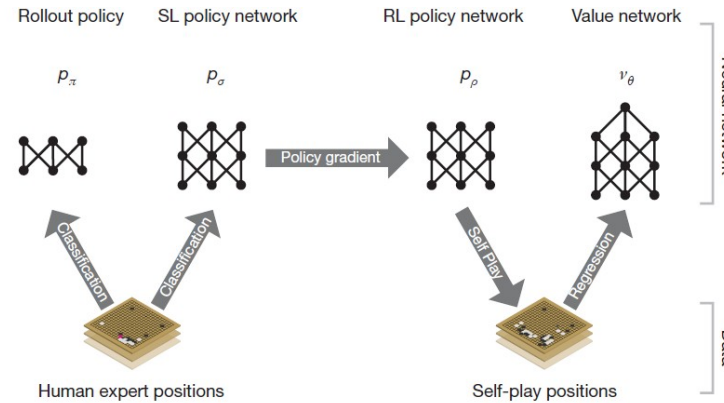
- **MCTS: Model look ahead to reduce searching space by predicting opponent's moves**



Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." Nature 529.7587 (2016): 484-489.

AlphaGO: Learning Pipeline

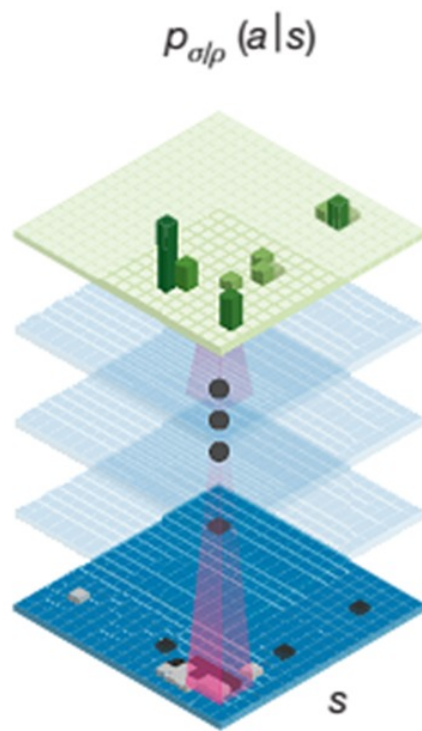
- Combine SL and RL to learn the search direction in MCTS



- **SL policy Network**
 - Prior search probability or potential
- **Rollout:**
 - combine with MCTS for quick simulation on leaf node
- **Value Network:**
 - Build the Global feeling on the leaf node situation

Learning to Prune : SL Policy Network

Policy network



- ▶ 13-layer CNN
- Input board position s
- Output: $p_{\sigma}(a|s)$, where a is the next move

Extended Data Table 2 | Input features for neural networks

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

Feature planes used by the policy network (all but last feature) and value network (all features).



策略梯度

基于策略函数的深度强化学习

- ▶ 可以直接用深度神经网络来表示一个参数化的从状态空间到动作空间的映射函数： $a = \pi_{\theta}(s)$ 。
- ▶ 最优的策略是使得在每个状态的总回报最大的策略，因此策略搜索的目标函数为

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)}[G(\tau)] = \mathbb{E}_{\tau \sim p_{\theta}(\tau)}\left[\sum_{t=0}^{T-1} \gamma^t r_{t+1}\right]$$

策略梯度 (Policy Gradient)

- ▶ 策略搜索是通过寻找参数 θ 使得目标函数 $J(\theta)$ 最大。
- ▶ 梯度上升：

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta} &= \frac{\partial}{\partial \theta} \int p_{\theta}(\tau) G(\tau) d\tau \\ &= \int \left(\frac{\partial}{\partial \theta} p_{\theta}(\tau) \right) G(\tau) d\tau \\ &= \int p_{\theta}(\tau) \left(\frac{1}{p_{\theta}(\tau)} \frac{\partial}{\partial \theta} p_{\theta}(\tau) \right) G(\tau) d\tau \\ &= \int p_{\theta}(\tau) \left(\frac{\partial}{\partial \theta} \log p_{\theta}(\tau) \right) G(\tau) d\tau \\ &= \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[\frac{\partial}{\partial \theta} \log p_{\theta}(\tau) G(\tau) \right],\end{aligned}$$

进一步分解

- ▶ 参数 θ 优化的方向是使得总回报 $G(\tau)$ 越大的轨迹 τ 的概率 $p_\theta(\tau)$ 也越大。

$$\begin{aligned}\frac{\partial}{\partial \theta} \log p_\theta(\tau) &= \frac{\partial}{\partial \theta} \log \left(p(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \right) \\ &= \frac{\partial}{\partial \theta} \left(\log p(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t) \right) \\ &= \sum_{t=0}^{T-1} \frac{\partial}{\partial \theta} \log \pi_\theta(a_t|s_t).\end{aligned}$$

$\partial \log p_\theta(\tau) / \partial \theta$ 是和状态转移概率无关，只和策略函数相关。

策略梯度

因此，策略梯度 $\frac{\partial \mathcal{J}(\theta)}{\partial \theta}$ 可写为

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

其中 $G(\tau_{t:T})$ 为从时刻 t 作为起始时刻收到总回报

$$G(\tau_{t:T}) = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'+1}.$$

REINFORCE 算法

算法 15.6: 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 ;

1 随机初始化参数 θ ;

2 **repeat**

3 根据策略 $\pi_{\theta}(a|s)$ 生成一条轨迹

$\tau = s_0, a_0, s_1, a_1, \dots, s_{T-1}, a_{T-1}, s_T$;

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

5 计算 $G(\tau_{t:T})$;

 // 更新策略函数参数

6 $\theta \leftarrow \theta + \alpha \gamma^t G(\tau_{t:T}) \frac{\partial}{\partial \theta} \log \pi_{\theta}(a_t|s_t)$;

7 **end**

8 **until** θ 收敛;

输出: 策略 π_{θ}

如何减少方差？

► 训练方差

- 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通病。

► 控制变量法

- 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 ，令

$$\hat{f} = f - \alpha(g - \mathbb{E}[g])$$



$$\text{var}(\hat{f}) = \text{var}(f) - 2\alpha \text{cov}(f, g) + \alpha^2 \text{var}(g)$$



$$\begin{aligned}\text{var}(\hat{f}) &= \left(1 - \frac{\text{cov}(f, g)^2}{\text{var}(g) \text{var}(f)}\right) \text{var}(f) \\ &= \left(1 - \text{corr}(f, g)^2\right) \text{var}(f),\end{aligned}$$

引入基线

► 在每个时刻 t ，其策略梯度为

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

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

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

如何选择 $b(s_t)$ ？

如何选择 $b(s_t)$?

- ▶ 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 $V^{\pi_\theta}(s_t)$.
- ▶ 一个可学习的函数 $V_\phi(s_t)$ 来近似值函数

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

- ▶ 策略函数参数 θ 的梯度为

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

带基准线的 REINFORCE 算法

算法 15.7: 带基准线的 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 $\pi_\theta(a|s)$, 可微分的状态值函数 $V_\phi(s)$, 折扣率 γ , 学习率 α, β ;

1 随机初始化参数 θ, ϕ ;

2 **repeat**

3 根据策略 $\pi_\theta(a|s)$ 生成一条轨迹

$\tau = s_0, a_0, s_1, a_1, \dots, s_{T-1}, a_{T-1}, s_T$;

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

5 计算 $G(\tau_{t:T})$;

6 $\delta \leftarrow G(\tau_{t:T}) - V_\phi(s_t)$;

 // 更新值函数参数

7 $\phi \leftarrow \phi + \beta \delta \frac{\partial}{\partial \phi} V_\phi(s_t)$;

 // 更新策略函数参数

8 $\theta \leftarrow \theta + \alpha \gamma^t \delta \frac{\partial}{\partial \theta} \log \pi_\theta(a_t|s_t)$;

9 **end**

10 **until** θ 收敛;

输出: 策略 π_θ

Actor-Critic 算法

- ▶ 演员-评论员算法 (Actor-Critic Algorithm) 是一种结合策略梯度和时序差分学习的强化学习方法。
 - ▶ 演员 (actor) 是指策略函数 $\pi_{\theta}(s, a)$ ，即学习一个策略来得到尽量高的回报。
 - ▶ 评论员 (critic) 是指值函数 $V_{\phi}(s)$ ，对当前策略的值函数进行估计，即评估actor的好坏。
- ▶ 借助于值函数，Actor-Critic算法可以进行单步更新参数，不需要等到回合结束才进行更新。

Actor-Critic 算法

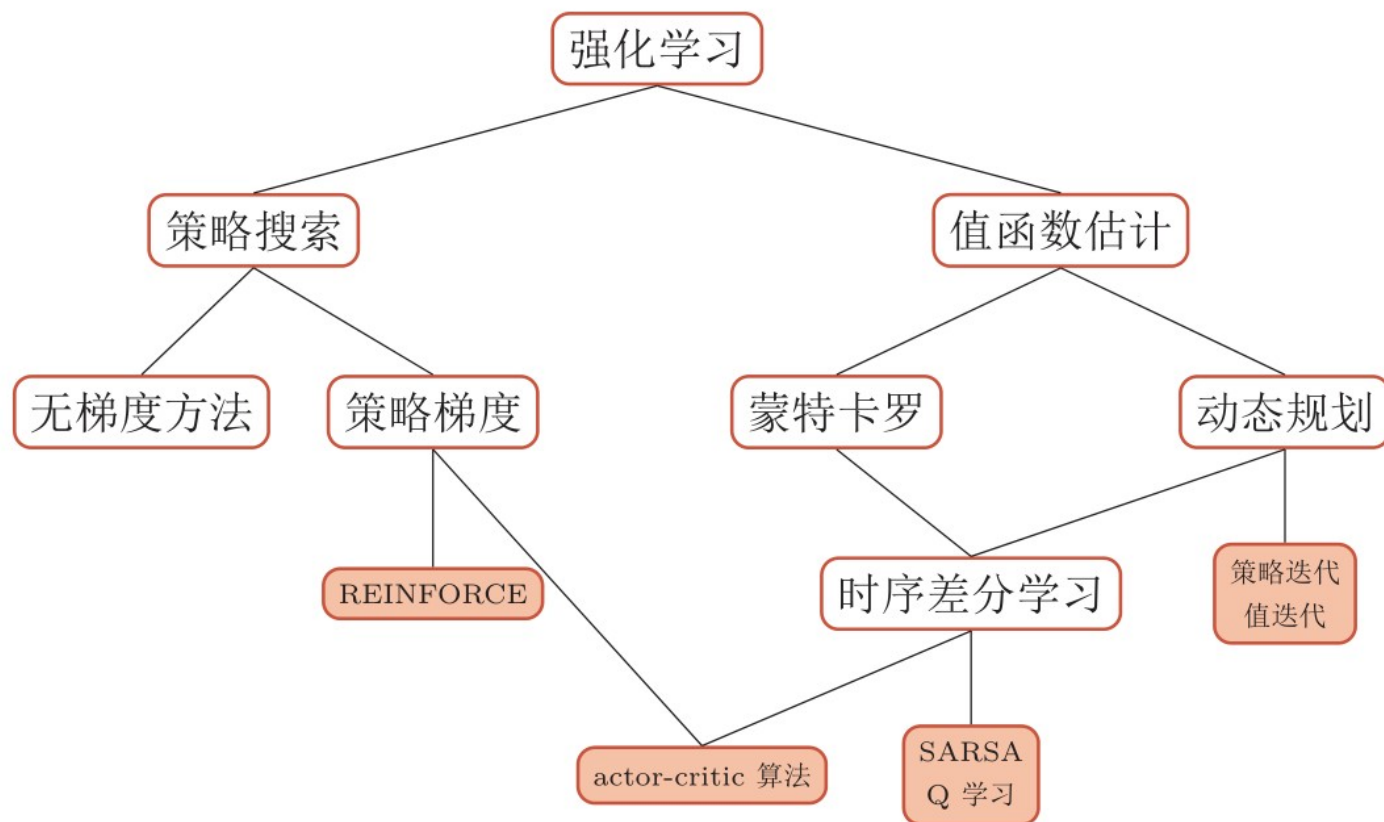
算法 15.8: actor-critic 算法

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 $\alpha > 0, \beta > 0$;

- 1 随机初始化参数 θ, ϕ ;
- 2 **repeat**
 - 3 初始化起始状态 s ;
 - 4 $\lambda = 1$;
 - 5 **repeat**
 - 6 在状态 s , 选择动作 $a = \pi_{\theta}(a|s)$;
 - 7 执行动作 a , 得到即时奖励 r 和新状态 s' ;
 - 8 $\delta \leftarrow r + \gamma V_{\phi}(s') - V_{\phi}(s)$;
 - 9 $\phi \leftarrow \phi + \beta \delta \frac{\partial}{\partial \phi} V_{\phi}(s)$;
 - 10 $\theta \leftarrow \theta + \alpha \lambda \delta \frac{\partial}{\partial \theta} \log \pi_{\theta}(a|s)$;
 - 11 $\lambda \leftarrow \gamma \lambda$;
 - 12 $s \leftarrow s'$;
 - 13 **until** s 为终止状态;
- 14 **until** θ 收敛;

输出: 策略 π_{θ}

不同强化学习算法之间的关系



汇总

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

谢谢！

<https://nndl.github.io/>