# 经典与现代强化学习的数学原理与实验

很多人在学习完强化学习的内容后，如果将书合上，会发现自己很难重述强化学习的故事。这种奇怪的感觉来自于强化学习的描述的是一个过程。它的目标在于给智能体寻找一个策略，智能体在这个策略的指导下在与动态环境交互的过程中产生最优的数据分布（价值在状态和动作上的分布），这个最优的数据分布可以使奖励的期望最大。由此可见，数据的分布是变化的，一旦策略更新，智能体能看到的数据分布也会变化。而在基于深度学习的预测任务中，数据的分布往往是不变的，我们希望用数据和模型复线真实的映射关系，使预测与真实值更加接近。

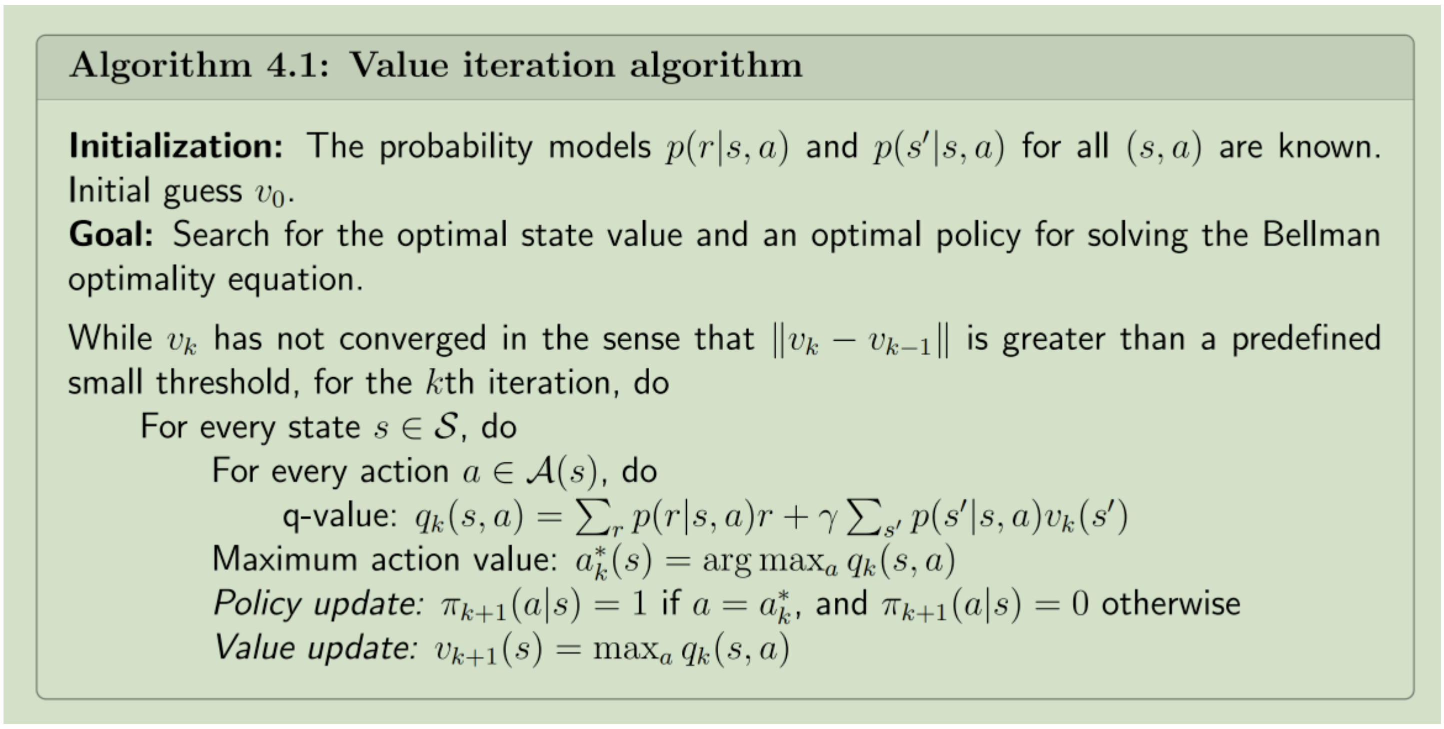
本文档重点在于理清强化学习被后的数学原理，追求做到在解决实际问题的过程中，如果算法效果不好，在分析原因的时候有指导思想和数学依据；必要的时候需要自己创造新的强化学习算法，此时可能需要从贝尔曼方程开始重新审查，构建随机过程并证明收敛性。

## 价值迭代

用来分析最优策略和最优价值的贝尔曼最优方程

$$\begin{aligned} v(s) &= \max_{\pi(s)\in\Pi(s)} \sum_{a\in\mathcal{A}} \pi(a \mid s)\left(\sum_{r\in\mathcal{R}} p(r \mid s,a)r + \gamma \sum_{s'\in\mathcal{S}} p\left(s' \mid s,a\right) v\left(s'\right)\right) \\ &= \max_{\pi(s)\in\Pi(s)} \sum_{a\in\mathcal{A}} \pi(a \mid s) q(s,a), \end{aligned}$$

的右边是一个收缩映射（Constraction Map），根据收缩映射定理，智能体与环境交互的过程中，$v(s)$ 会收缩到最优状态价值。这个过程不够直观，可以理解为函数的变化总是比自变量的变化要小，那么随着自变量不断地在上一步的基础上变化，对应智能体不断的在当前状态下和环境交互，函数的变化量会逐渐变小，直至收敛到固定点（fixed point）。

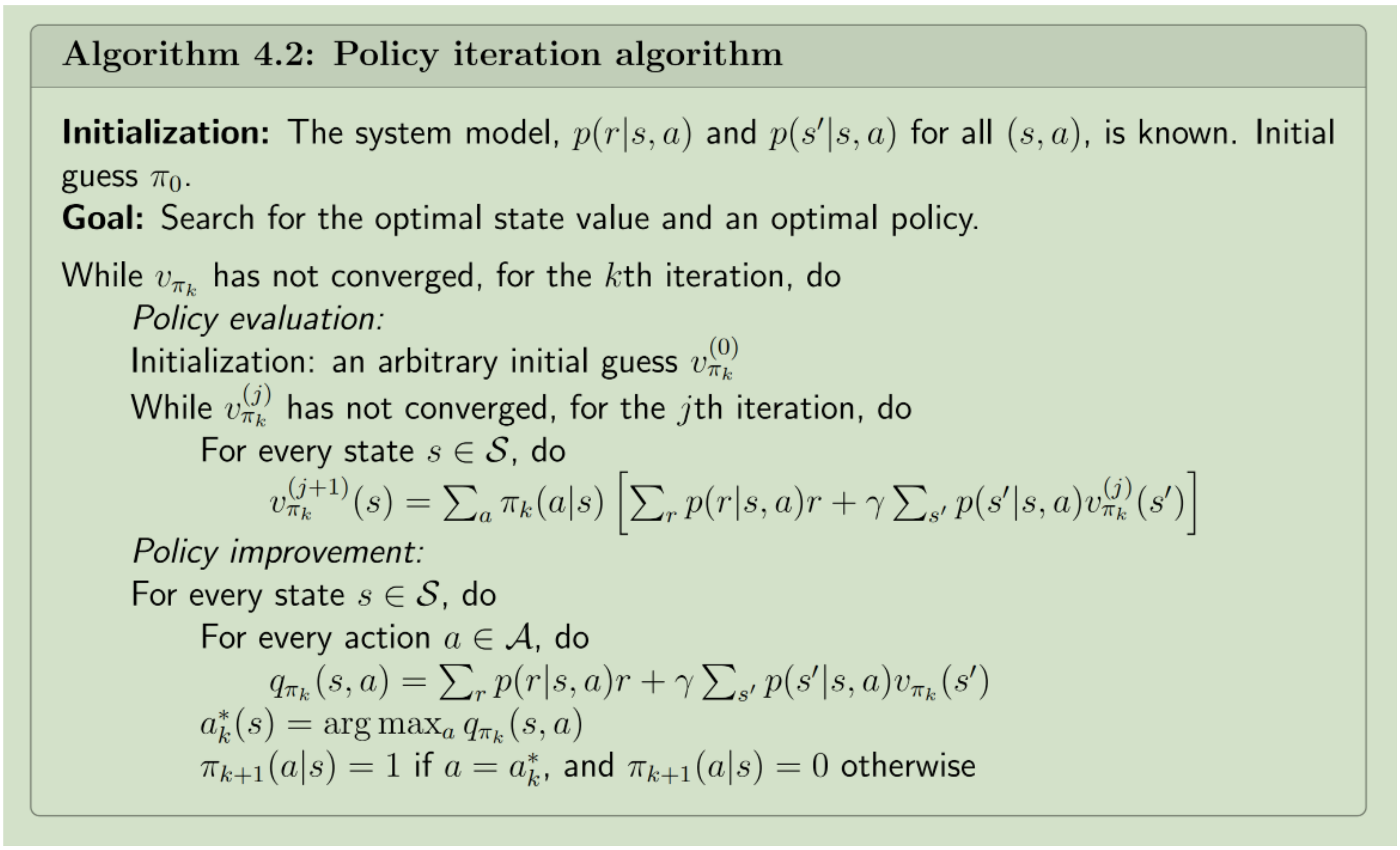


**Algorithm 4.1: Value iteration algorithm**

**Initialization:** The probability models $p(r|s,a)$ and $p(s'|s,a)$ for all $(s,a)$ are known. Initial guess $v_0$.
**Goal:** Search for the optimal state value and an optimal policy for solving the Bellman optimality equation.

```
While $v_k$ has not converged in the sense that $\|v_k - v_{k-1}\|$ is greater than a predefined small threshold, for the $k$th iteration, do
    For every state $s \in \mathcal{S}$, do
        For every action $a \in \mathcal{A}(s)$, do
            q-value: $q_k(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_k(s')$
        Maximum action value: $a_k^*(s) = \arg\max_a q_k(s,a)$
        *Policy update:* $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise
        *Value update:* $v_{k+1}(s) = \max_a q_k(s,a)$
```

## 策略迭代



**Algorithm 4.2: Policy iteration algorithm**

**Initialization:** The system model, $p(r|s,a)$ and $p(s'|s,a)$ for all $(s,a)$, is known. Initial guess $\pi_0$.
**Goal:** Search for the optimal state value and an optimal policy.

```
While $v_{\pi_k}$ has not converged, for the $k$th iteration, do
    *Policy evaluation:*
    Initialization: an arbitrary initial guess $v_{\pi_k}^{(0)}$
    While $v_{\pi_k}^{(j)}$ has not converged, for the $j$th iteration, do
        For every state $s \in \mathcal{S}$, do
            $v_{\pi_k}^{(j+1)}(s) = \sum_a \pi_k(a|s)\left[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_{\pi_k}^{(j)}(s')\right]$
    *Policy improvement:*
    For every state $s \in \mathcal{S}$, do
        For every action $a \in \mathcal{A}$, do
            $q_{\pi_k}(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_{\pi_k}(s')$
        $a_k^*(s) = \arg\max_a q_{\pi_k}(s,a)$
        $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise
```

## 截断策略迭代

从理论分析来看，价值迭代是截断策略迭代在 $j_{\text{truncate}} = 1$ 的特殊情况，而策略迭代是截断策略迭代在 $j_{\text{truncate}} = \infty$ 的特殊情况。但是在实际的编程实现中，策略迭代往往退化成阶段策略迭代，因为必须设定一个终止迭代条件。

价值迭代、策略迭代、截断策略迭代都包含价值更新和策略更新这两个交替步骤，可以将这些迭代方法都看作**广义策略迭代**（generalization policy iteration）。

**Algorithm 4.3: Truncated policy iteration algorithm**

```
Initialization: The probability models $p(r|s,a)$ and $p(s'|s,a)$ for all $(s,a)$ are known. Initial guess $\pi_0$.
Goal: Search for the optimal state value and an optimal policy.

While $v_k$ has not converged, for the $k$th iteration, do
    Policy evaluation:
    Initialization: select the initial guess as $v_k^{(0)} = v_{k-1}$. The maximum number of iterations is set as $j_{\text{truncate}}$.
    While $j < j_{\text{truncate}}$, do
        For every state $s \in \mathcal{S}$, do
            $v_k^{(j+1)}(s) = \sum_a \pi_k(a|s) \left[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_k^{(j)}(s')\right]$
    Set $v_k = v_k^{(j_{\text{truncate}})}$
    Policy improvement:
    For every state $s \in \mathcal{S}$, do
        For every action $a \in \mathcal{A}(s)$, do
            $q_k(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_k(s')$
        $a_k^*(s) = \arg\max_a q_k(s,a)$
        $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise
```

## 蒙特卡洛

**Algorithm 5.1: MC Basic (a model-free variant of policy iteration)**

```
Initialization: Initial guess $\pi_0$.
Goal: Search for an optimal policy.

For the $k$th iteration $(k = 0, 1, 2, \ldots)$, do
    For every state $s \in \mathcal{S}$, do
        For every action $a \in \mathcal{A}(s)$, do
            Collect sufficiently many episodes starting from $(s,a)$ by following $\pi_k$
            Policy evaluation:
            $q_{\pi_k}(s,a) \approx q_k(s,a) =$ the average return of all the episodes starting from $(s,a)$
        Policy improvement:
        $a_k^*(s) = \arg\max_a q_k(s,a)$
        $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise
```

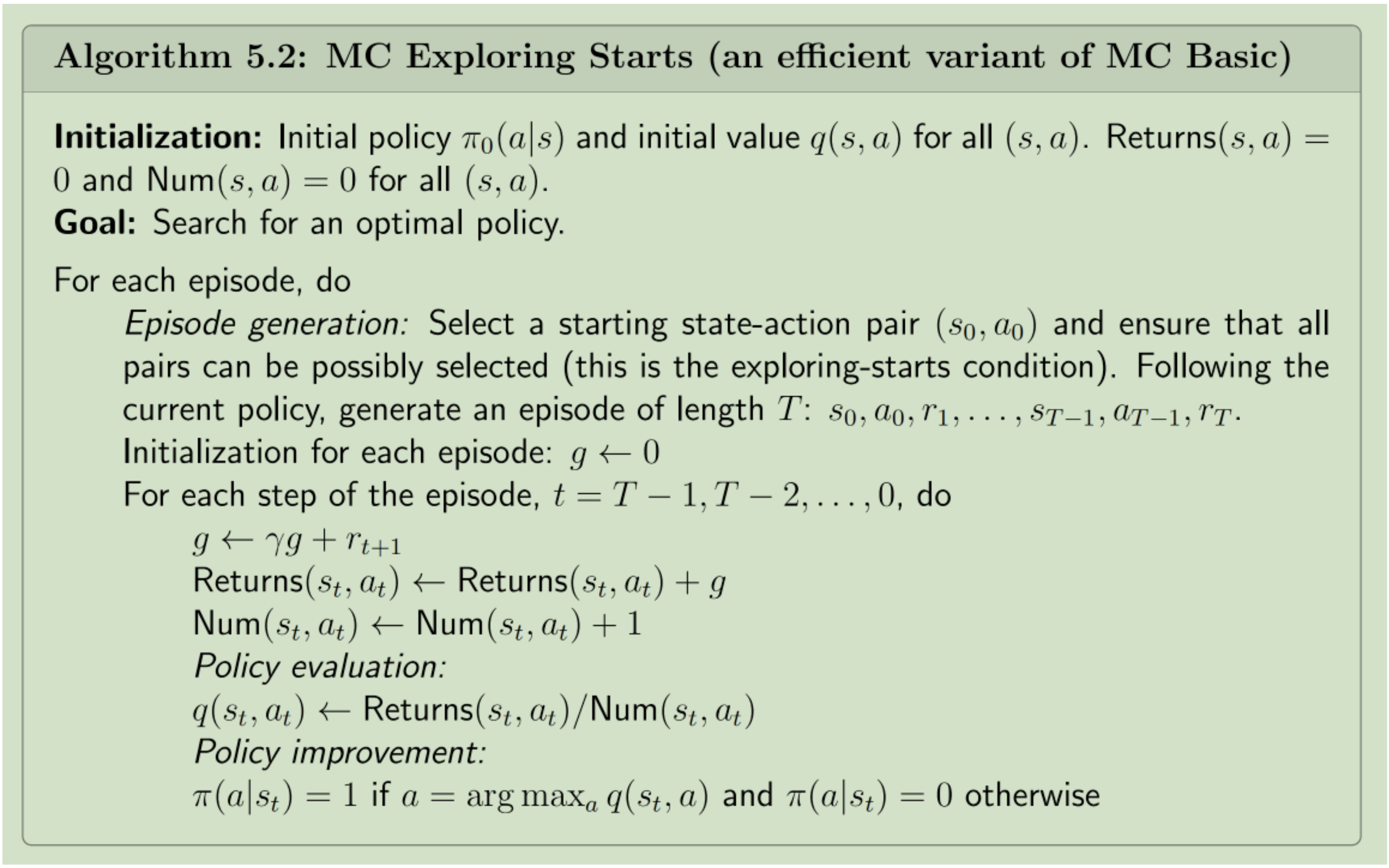
Monte Carlo Exploring Starts发生作用的数学原理是大数定律，当采样了足够多条轨迹之后，相当于每个状态动作对都会有若干条轨迹从它开始，再根据状态动作价值的定义：

$$\begin{aligned} q_{\pi_k}(s,a) &= \mathbb{E}\left[G_t \mid S_t = s, A_t = a\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s, A_t = a\right] \\ &\approx \frac{1}{n}\sum_{i=1}^{n} g_{\pi_k}^{(i)}(s,a) \end{aligned}$$

其中

$$g_{\pi_k}^{(i)}(s,a) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$$

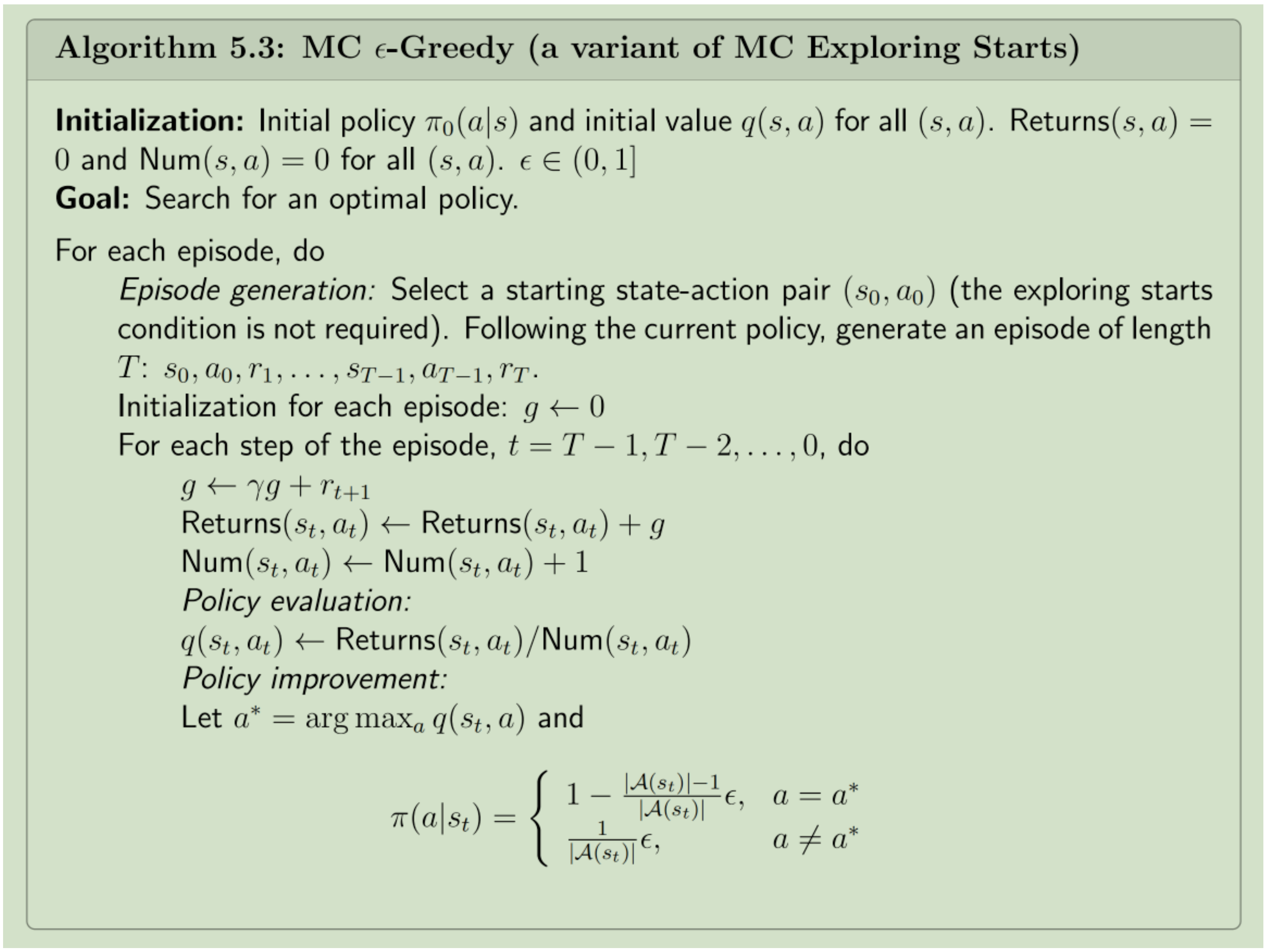
基于大数定律，当episode的数量 $n$ 足够大，上面对于 $q_{\pi_k}(s,a)$ 的估计就会越准确。**也正是因为有大数定律的存在，使得蒙特卡洛树搜索可以被进行多线程加速——不同叶子节点的不同Rollout可以同时进行，都可以看作是从一个状态开始的采样轨迹；同一个叶子节点的Rollout和back propagate可以异步进行，从而可以以事件驱动的线程池响应机制构建蒙特卡洛树搜索。**



**Algorithm 5.2: MC Exploring Starts (an efficient variant of MC Basic)**

**Initialization:** Initial policy $\pi_0(a|s)$ and initial value $q(s,a)$ for all $(s,a)$. Returns$(s,a) = 0$ and Num$(s,a) = 0$ for all $(s,a)$.
**Goal:** Search for an optimal policy.

```
For each episode, do
    Episode generation: Select a starting state-action pair $(s_0, a_0)$ and ensure that all pairs can be possibly selected (this is the exploring-starts condition). Following the current policy, generate an episode of length $T$: $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$.
    Initialization for each episode: $g \leftarrow 0$
    For each step of the episode, $t = T-1, T-2, \ldots, 0$, do
        $g \leftarrow \gamma g + r_{t+1}$
        Returns$(s_t, a_t) \leftarrow$ Returns$(s_t, a_t) + g$
        Num$(s_t, a_t) \leftarrow$ Num$(s_t, a_t) + 1$
        Policy evaluation:
        $q(s_t, a_t) \leftarrow$ Returns$(s_t, a_t)$/Num$(s_t, a_t)$
        Policy improvement:
        $\pi(a|s_t) = 1$ if $a = \arg\max_a q(s_t, a)$ and $\pi(a|s_t) = 0$ otherwise
```



**Algorithm 5.3: MC $\epsilon$-Greedy (a variant of MC Exploring Starts)**

**Initialization:** Initial policy $\pi_0(a|s)$ and initial value $q(s,a)$ for all $(s,a)$. Returns$(s,a) = 0$ and Num$(s,a) = 0$ for all $(s,a)$. $\epsilon \in (0,1]$
**Goal:** Search for an optimal policy.

```
For each episode, do
    Episode generation: Select a starting state-action pair $(s_0, a_0)$ (the exploring starts
    condition is not required). Following the current policy, generate an episode of length
    $T$: $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$.
    Initialization for each episode: $g \leftarrow 0$
    For each step of the episode, $t = T-1, T-2, \ldots, 0$, do
        $g \leftarrow \gamma g + r_{t+1}$
        Returns$(s_t, a_t) \leftarrow$ Returns$(s_t, a_t) + g$
        Num$(s_t, a_t) \leftarrow$ Num$(s_t, a_t) + 1$
        Policy evaluation:
        $q(s_t, a_t) \leftarrow$ Returns$(s_t, a_t)$/Num$(s_t, a_t)$
        Policy improvement:
        Let $a^* = \arg\max_a q(s_t, a)$ and
```

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

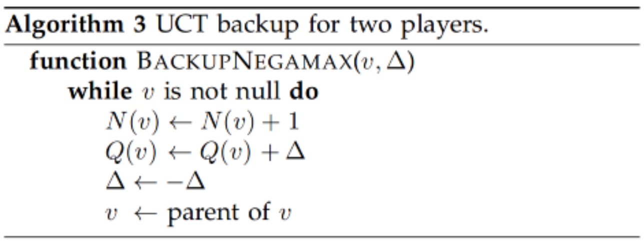
**蒙特卡洛树搜索**

UTC算法中选择最优子节点用的是

$$\underset{v' \in \text{ children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$$

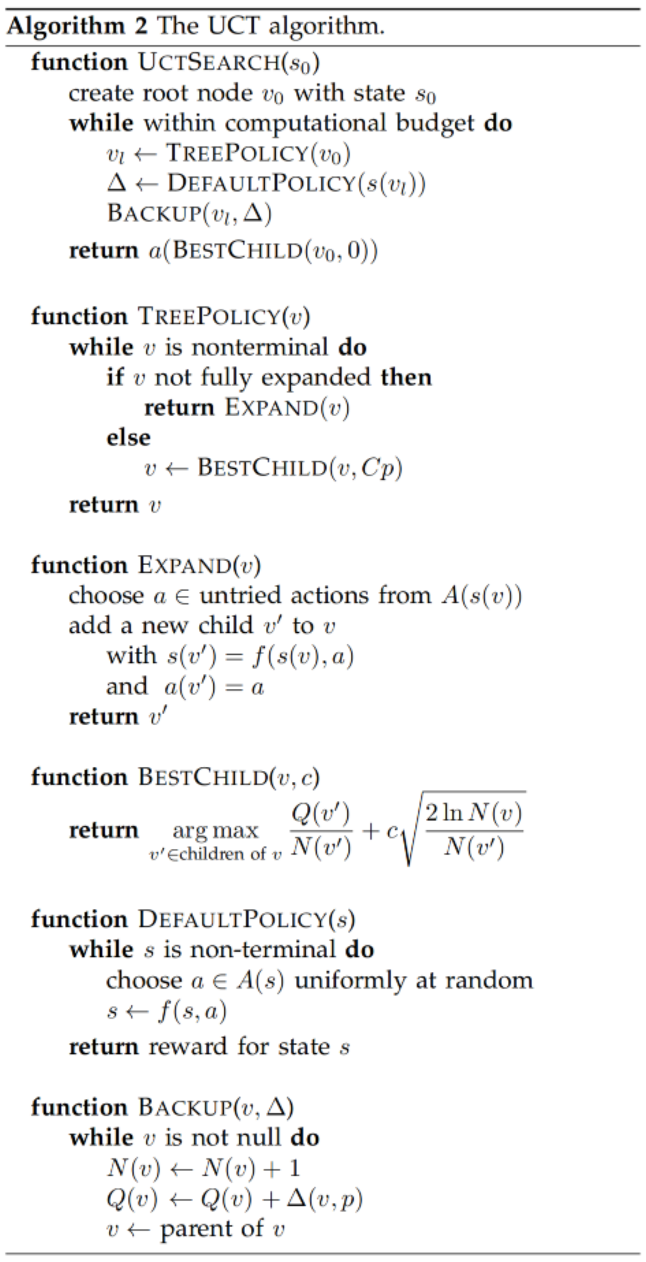
如果用并行，可以用四个线程全部探索四个子节点，返回的仿真回报可以直接加在一起求均值。其背后的数学原理也是上述大数定律，假设从每个节点出发有n条轨迹，第i条轨迹的回报是

$$q_{\pi_k}(s,a) = \mathbb{E}[G_t \mid S_t = s, A_t = a] \approx \frac{1}{n}\sum_{i=1}^{n} g_{\pi_k}^{(i)}(s,a)$$



**Algorithm 3** UCT backup for two players.

```
function BACKUPNEGAMAX($v, \Delta$)
    while $v$ is not null do
        $N(v) \leftarrow N(v) + 1$
        $Q(v) \leftarrow Q(v) + \Delta$
        $\Delta \leftarrow -\Delta$
        $v \leftarrow$ parent of $v$
```

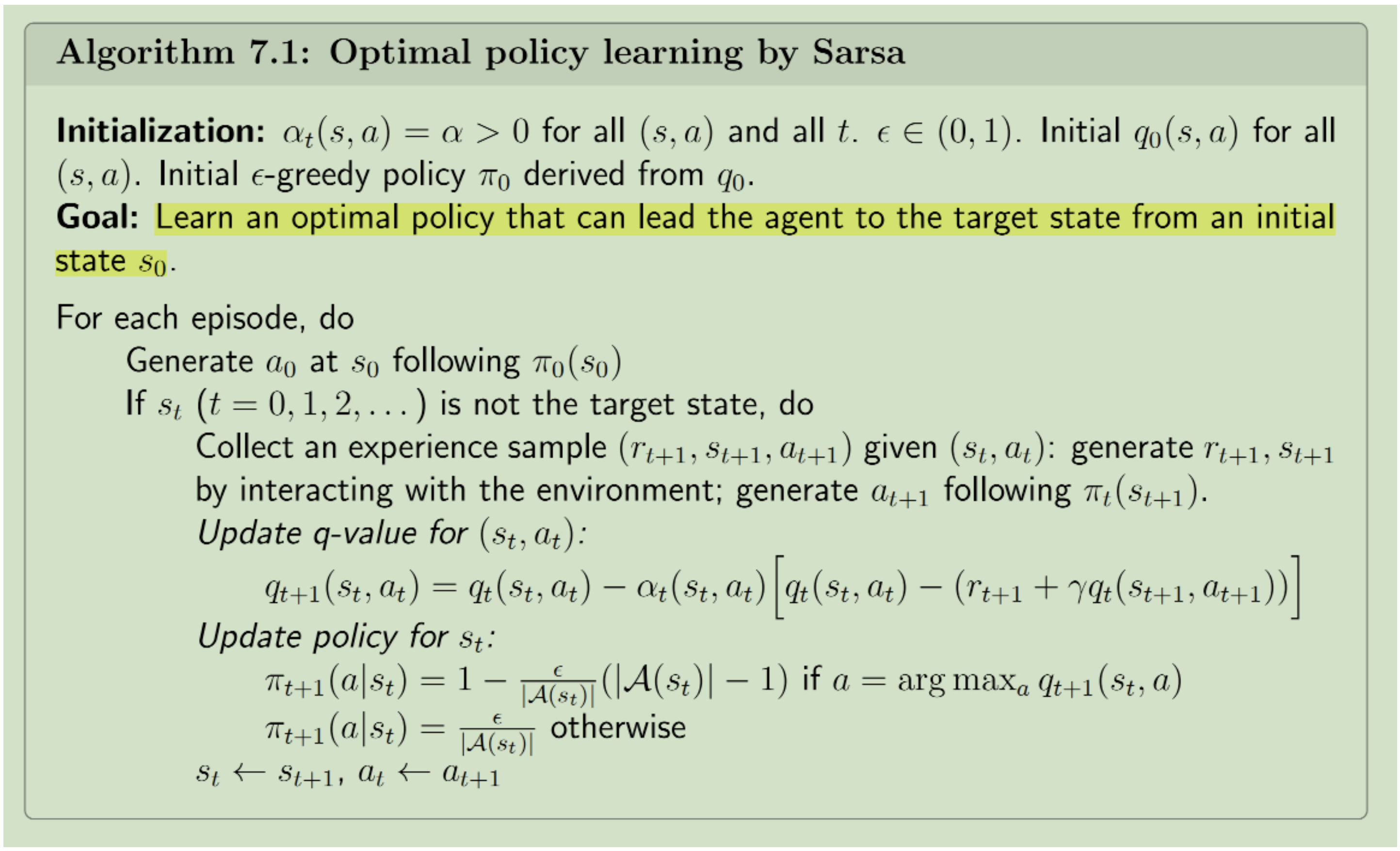

**Algorithm 2** The UCT algorithm.

```
function UCTSEARCH($s_0$)
    create root node $v_0$ with state $s_0$
    while within computational budget do
        $v_l \leftarrow$ TREEPOLICY($v_0$)
        $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
        BACKUP($v_l, \Delta$)
    return $a$(BESTCHILD($v_0, 0$))

function TREEPOLICY($v$)
    while $v$ is nonterminal do
        if $v$ not fully expanded then
            return EXPAND($v$)
        else
            $v \leftarrow$ BESTCHILD($v, Cp$)
    return $v$

function EXPAND($v$)
    choose $a \in$ untried actions from $A(s(v))$
    add a new child $v'$ to $v$
        with $s(v') = f(s(v), a)$
        and $a(v') = a$
    return $v'$

function BESTCHILD($v, c$)
    return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$

function DEFAULTPOLICY($s$)
    while $s$ is non-terminal do
        choose $a \in A(s)$ uniformly at random
        $s \leftarrow f(s, a)$
    return reward for state $s$

function BACKUP($v, \Delta$)
    while $v$ is not null do
        $N(v) \leftarrow N(v) + 1$
        $Q(v) \leftarrow Q(v) + \Delta(v, p)$
        $v \leftarrow$ parent of $v$
```

参考：A Survey of Monte Carlo Tree Search Methods

## 时间差分

sarsa算法的有效性来自于Dvoretzky’s收敛定理。可以证明其算法的运行过程满足Dvoretzky’s收敛定理中的随机过程。



**Algorithm 7.1: Optimal policy learning by Sarsa**

**Initialization:** $\alpha_t(s,a)=\alpha>0$ for all $(s,a)$ and all $t$. $\epsilon\in(0,1)$. Initial $q_0(s,a)$ for all $(s,a)$. Initial $\epsilon$-greedy policy $\pi_0$ derived from $q_0$.
**Goal:** Learn an optimal policy that can lead the agent to the target state from an initial state $s_0$.

```
For each episode, do
    Generate $a_0$ at $s_0$ following $\pi_0(s_0)$
    If $s_t$ $(t=0,1,2,\dots)$ is not the target state, do
        Collect an experience sample $(r_{t+1},s_{t+1},a_{t+1})$ given $(s_t,a_t)$: generate $r_{t+1},s_{t+1}$
        by interacting with the environment; generate $a_{t+1}$ following $\pi_t(s_{t+1})$.
        Update q-value for $(s_t,a_t)$:
            $q_{t+1}(s_t,a_t)=q_t(s_t,a_t)-\alpha_t(s_t,a_t)\Big[q_t(s_t,a_t)-(r_{t+1}+\gamma q_t(s_{t+1},a_{t+1}))\Big]$
        Update policy for $s_t$:
            $\pi_{t+1}(a|s_t)=1-\frac{\epsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)|-1)$ if $a=\arg\max_a q_{t+1}(s_t,a)$
            $\pi_{t+1}(a|s_t)=\frac{\epsilon}{|\mathcal{A}(s_t)|}$ otherwise
        $s_t\leftarrow s_{t+1}$, $a_t\leftarrow a_{t+1}$
```

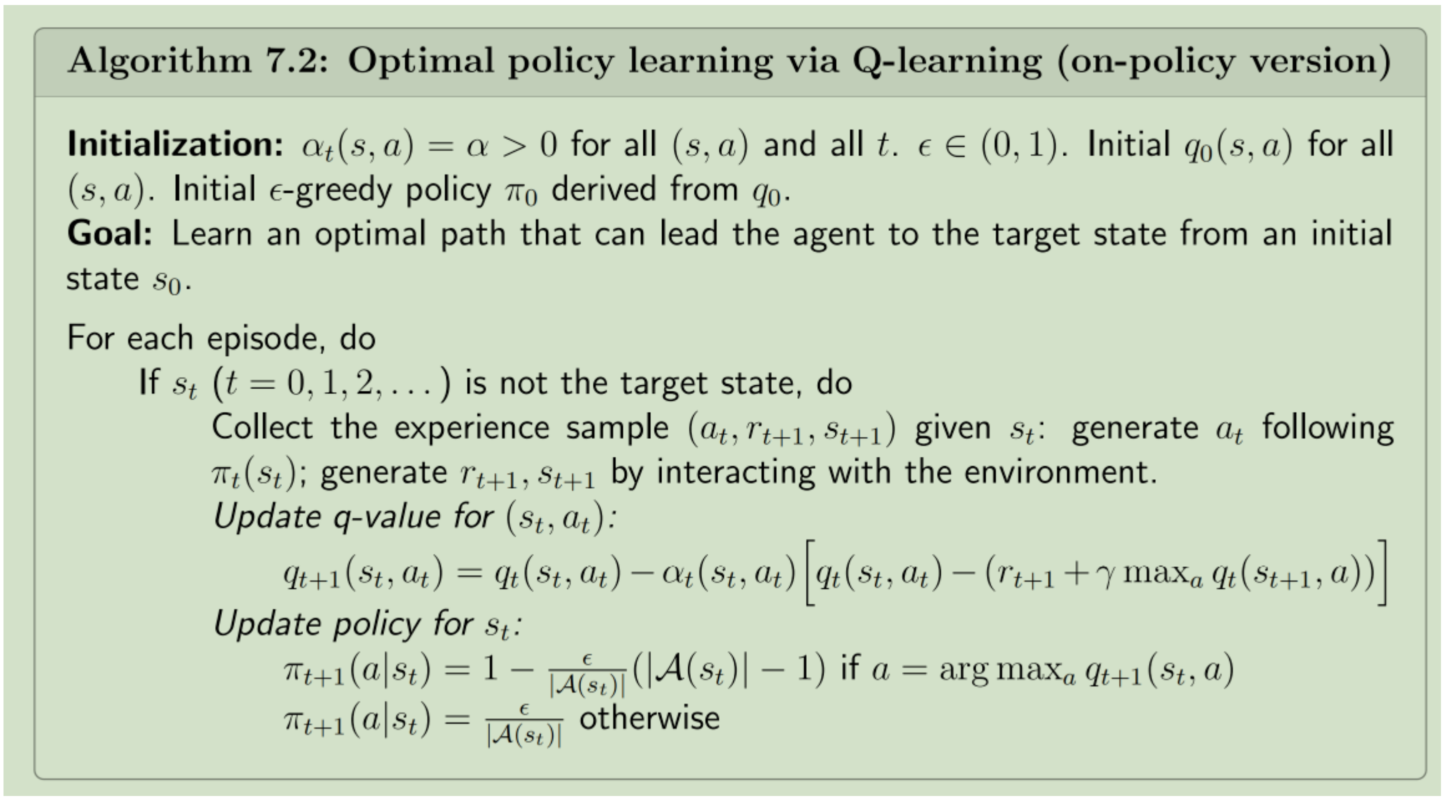
Q-learning算法本质上是对下面等式的随机估计算法：

$$q(s,a)=\mathbb{E}\left[R_{t+1}+\gamma\max_{a_{t+1}}q\left(S_{t+1},a_{t+1}\right)\mid S_t=s,A_t=a\right]$$

这是贝尔曼最优方程的动作价值表达形式。Q-learning算法的随机过程为：

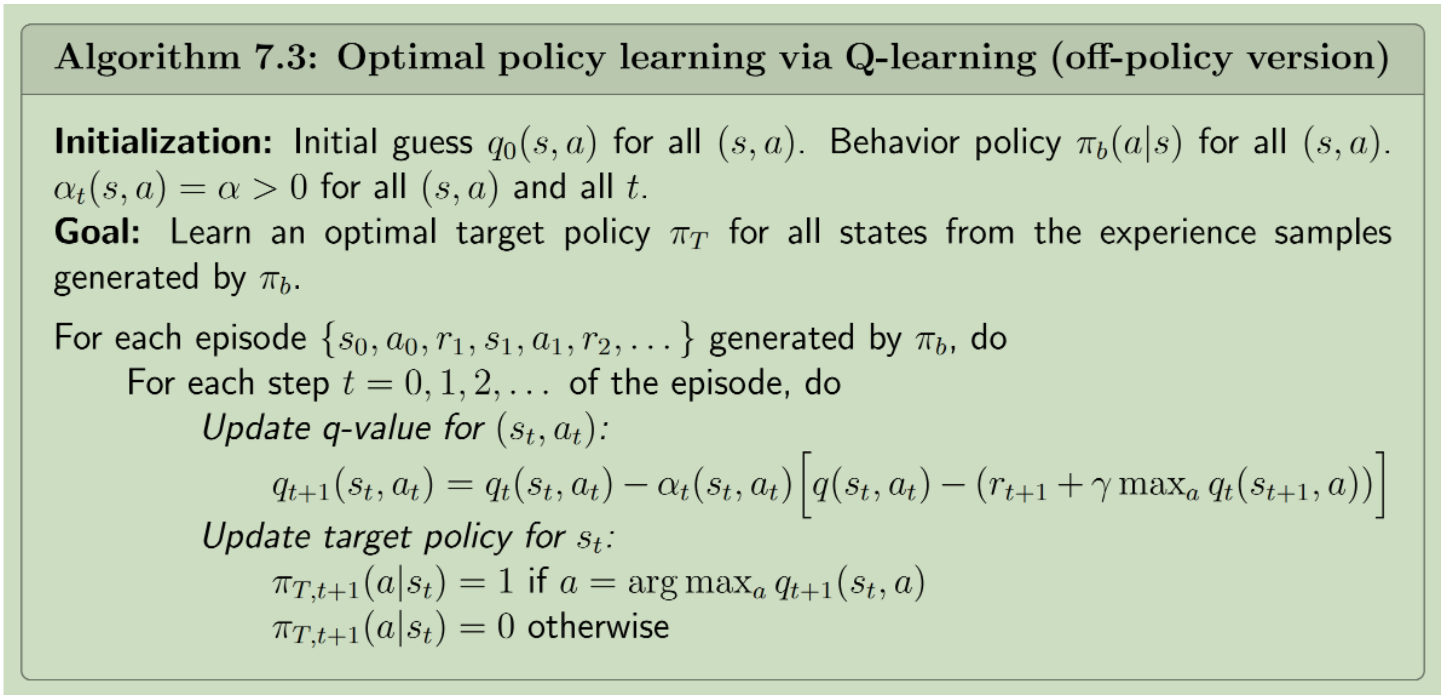
$$\begin{aligned}q_{t+1}\left(s_t,a_t\right)&=q_t\left(s_t,a_t\right)-\alpha_t\left(s_t,a_t\right)\left[q_t\left(s_t,a_t\right)-\left(r_{t+1}+\gamma\max_{a\in\mathcal{A}(s_{t+1})}q_t\left(s_{t+1},a\right)\right)\right]\\ q_{t+1}(s,a)&=q_t(s,a),\qquad\text{for all }(s,a)\neq(s_t,a_t)\end{aligned}$$

其收敛性证明可借助于Robbins Monro定理。



**Algorithm 7.2: Optimal policy learning via Q-learning (on-policy version)**

```
Initialization: $\alpha_t(s,a)=\alpha>0$ for all $(s,a)$ and all $t$. $\epsilon\in(0,1)$. Initial $q_0(s,a)$ for all $(s,a)$. Initial $\epsilon$-greedy policy $\pi_0$ derived from $q_0$.
Goal: Learn an optimal path that can lead the agent to the target state from an initial state $s_0$.

For each episode, do
    If $s_t$ $(t=0,1,2,\dots)$ is not the target state, do
        Collect the experience sample $(a_t,r_{t+1},s_{t+1})$ given $s_t$: generate $a_t$ following $\pi_t(s_t)$; generate $r_{t+1},s_{t+1}$ by interacting with the environment.
        Update q-value for $(s_t,a_t)$:
            $q_{t+1}(s_t,a_t)=q_t(s_t,a_t)-\alpha_t(s_t,a_t)\Big[q_t(s_t,a_t)-(r_{t+1}+\gamma\max_a q_t(s_{t+1},a))\Big]$
        Update policy for $s_t$:
            $\pi_{t+1}(a|s_t)=1-\frac{\epsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)|-1)$ if $a=\arg\max_a q_{t+1}(s_t,a)$
            $\pi_{t+1}(a|s_t)=\frac{\epsilon}{|\mathcal{A}(s_t)|}$ otherwise
```



**Algorithm 7.3: Optimal policy learning via Q-learning (off-policy version)**

```
Initialization: Initial guess $q_0(s,a)$ for all $(s,a)$. Behavior policy $\pi_b(a|s)$ for all $(s,a)$. $\alpha_t(s,a)=\alpha>0$ for all $(s,a)$ and all $t$.
Goal: Learn an optimal target policy $\pi_T$ for all states from the experience samples generated by $\pi_b$.

For each episode $\{s_0,a_0,r_1,s_1,a_1,r_2,\dots\}$ generated by $\pi_b$, do
    For each step $t=0,1,2,\dots$ of the episode, do
        Update q-value for $(s_t,a_t)$:
            $q_{t+1}(s_t,a_t)=q_t(s_t,a_t)-\alpha_t(s_t,a_t)\Big[q(s_t,a_t)-(r_{t+1}+\gamma\max_a q_t(s_{t+1},a))\Big]$
        Update target policy for $s_t$:
            $\pi_{T,t+1}(a|s_t)=1$ if $a=\arg\max_a q_{t+1}(s_t,a)$
            $\pi_{T,t+1}(a|s_t)=0$ otherwise
```

## 值函数估计

强化学习框架下智能体对环境的状态价值估计问题的函数估计方式可以建模为找到最优的参数 $w$ 使得 $\hat{v}(s,w)$ 是对每个状态 $s$ 的价值 $v_\pi(s)$ 最佳估计，目标函数：

$$J(w)=\mathbb{E}\left[\left(v_\pi(S)-\hat{v}(S,w)\right)^2\right] \tag{1}$$

如果每个状态都会被同样的重视，那么公式 (1) 中的期望就可以用均匀分布的概率乘积和计算。但是智能体一般都有一定的策略去和环境进行交互，并且假设该过程是一个马尔可夫过程，在这个前提下，可以引入稳态分布来描述马尔可夫决策过程的长期行为。具体而言，在智能体在一个给定策略指

导下执行了足够长的时间以后，智能体在任何一个状态下的概率可以被这个稳态分布 $d_\pi$ 描述，于是 (1) 可以近似为

$$J(w)=\sum_{s\in\mathcal{S}} d_\pi(s)\left(v_\pi(s)-\hat{v}(s,w)\right)^2 \tag{2}$$

稳态分布分布 $d_\pi$ 不需要知道其确切的值，但是正是因为它的存在，可以将 (1) 中的随机变量 $S$ 写成一个采样值 $s_t$ ，从而引入了随机梯度下降法。

为了最小化 (1) 中的目标函数，可以构建随机梯度下降法的随机随机过程（随机梯度下降法是一个特殊的Robbins Monro算法）

$$w_{k+1}=w_k-\alpha\nabla_w J\left(w_k\right)$$

其中

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

于是随机梯度下降法可以写成

$$w_{k+1}=w_k+2\alpha_k\mathbb{E}\left[\left(v_\pi(S)-\hat{v}\left(S,w_k\right)\right)\nabla_w \hat{v}\left(S,w_k\right)\right]$$

基于上面的稳态分布的论述，进一步可以写成

$$w_{k+1}=w_k+\alpha_k\left(\left(v_\pi(s_t)-\hat{v}\left(s_t,w_k\right)\right)\nabla_w \hat{v}\left(s_t,w_k\right)\right)$$

不过此时我们还不知道 $v_\pi(s_t)$ ，但是可以借助蒙特卡洛和时间差分的方法进行估计：

1. 蒙特卡洛

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

2. 时间差分

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

**Algorithm 8.1: TD learning of state values with function approximation**

**Initialization:** A function $\hat{v}(s,w)$ that is a differentiable in $w$. Initial parameter $w_0$.
**Goal:** Learn the true state values of a given policy $\pi$.

```
For each episode $\{(s_t, r_{t+1}, s_{t+1})\}_t$ generated by $\pi$, do
    For each sample $(s_t, r_{t+1}, s_{t+1})$, do
        In the general case, $w_{t+1} = w_t + \alpha_t [r_{t+1} + \gamma\hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)] \nabla_w \hat{v}(s_t, w_t)$
        In the linear case, $w_{t+1} = w_t + \alpha_t \big[r_{t+1} + \gamma\phi^T(s_{t+1})w_t - \phi^T(s_t)w_t\big]\phi(s_t)$
```

**Algorithm 8.2: Sarsa with function approximation**

```
Initialization: Initial parameter $w_0$. Initial policy $\pi_0$. $\alpha_t = \alpha > 0$ for all $t$. $\epsilon \in (0,1)$.
Goal: Learn an optimal policy that can lead the agent to the target state from an initial state $s_0$.
For each episode, do
    Generate $a_0$ at $s_0$ following $\pi_0(s_0)$
    If $s_t$ $(t = 0,1,2,\dots)$ is not the target state, do
        Collect the experience sample $(r_{t+1}, s_{t+1}, a_{t+1})$ given $(s_t, a_t)$: generate $r_{t+1}, s_{t+1}$
        by interacting with the environment; generate $a_{t+1}$ following $\pi_t(s_{t+1})$.
        Update q-value:
        $w_{t+1} = w_t + \alpha_t\Big[r_{t+1} + \gamma\hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t)\Big]\nabla_w\hat{q}(s_t, a_t, w_t)$
        Update policy:
        $\pi_{t+1}(a|s_t) = 1 - \frac{\varepsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)| - 1)$ if $a = \arg\max_{a\in\mathcal{A}(s_t)} \hat{q}(s_t, a, w_{t+1})$
        $\pi_{t+1}(a|s_t) = \frac{\epsilon}{|\mathcal{A}(s_t)|}$ otherwise
        $s_t \leftarrow s_{t+1}$, $a_t \leftarrow a_{t+1}$
```

参考Q-learning的数学建模过程，深度Q-learning的目标是最小化

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

参考前述贝尔曼最优方程的动作价值形式，这个目标函数可以看作**平方贝尔曼最优误差**。又由于

$$q(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a\in\mathcal{A}(S_{t+1})} q\left(S_{t+1}, a\right) \mid S_t = s, A_t = a\right], \quad \text{for all } s, a$$

所以 $J$ 的期望值应该等于0，于是可以通过SGD最小化 $J$。考虑到实际应用的时候，动作空间大小 $|\mathcal{A}\left(S'\right)|$ 可能会很大，神经网络可能会比较复杂，所以 $\max_{a\in\mathcal{A}(S_{t+1})} q\left(S_{t+1}, a\right)$ 计算可能代价比较大，为了方便计算，可以认为在比较小的事件片段 $C$ 内，这个值是不变的。另外，$\max_{a\in\mathcal{A}(S_{t+1})} q\left(S_{t+1}, a\right)$ 通常不是一个凸运算，所以神经网络训练会不稳定。

需要仔细理解这里的回放缓冲区 $\mathcal{B}$ 的设计、网络参数异步更新的设计，这些新的设计不能破坏SGD的收敛性。回顾定理6.4，随机梯度下降法的收敛性定理：

> 对于随机梯度下降算法
>
> $$w_{k+1} = w_k - \alpha\nabla_w f\left(w_k, x_k\right)$$
>
> 如果下列条件满足
>
> $$\begin{aligned} &(a) \qquad 0 < c_1 \le \nabla_w^2 f(w, X) \le c_2; \\ &(b) \qquad \sum_{k=1}^{\infty} a_k = \infty \quad and \quad \sum_{k=1}^{\inf} a_k^2 < \infty; \\ &(c) \qquad \{x_k\}_{k=1}^{\infty} \quad are \quad i.i.d. \end{aligned}$$
>
> 那么 $w_k$ 将会几乎必然（almost surely）收敛到方程 $\nabla_w\mathbb{E}\left[f\left(w, X\right)\right] = 0$ 的根。

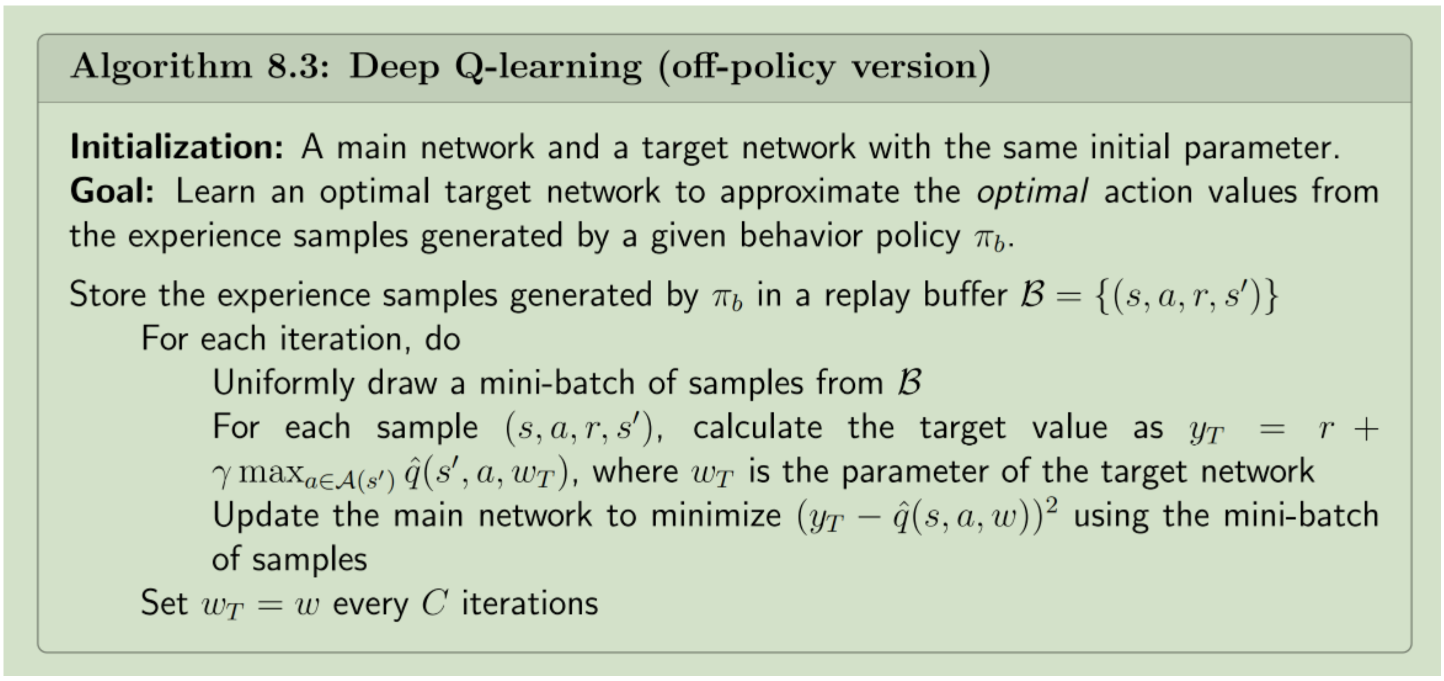
缓冲区中选取数据的时候使用了随机采样，在 MDP 中交互采样得到的数据本身不满足独立假设，因为这一时刻的状态和上一时刻的状态有关。非独立同分布的数据对训练神经网络有很大的影响，会使神经网络拟合到最近训练的数据上。缓冲区回放可以打破样本之间的相关性，让其满足独立假设；两套神经网络参数的异步更新本身不会影响条件条件 $(b)$、$(c)$，对于条件 $(a)$，我们假设

$$\begin{aligned} y\left(w_t\right) &\doteq r_{t+1}+\gamma \max_{a_{t+1} \in \mathcal{A}\left(s^{\prime}\right)} \hat{q}\left(s^{\prime}, a, w_t\right) \\ y\left(w_T\right) &\doteq r_{t+1}+\gamma \max_{a_{t+1} \in \mathcal{A}\left(s^{\prime}\right)} \hat{q}\left(s^{\prime}, a, w_T\right) \\ \delta &= y\left(w_t\right)-y\left(w_T\right) \end{aligned}$$

于是引入两套神经网络的目标函数可以写成

$$J=\left(r_{t+1}+\gamma \max_{a_{t+1} \in \mathcal{A}\left(s^{\prime}\right)} \hat{q}\left(s^{\prime}, a, w_t\right)-\delta-\hat{q}\left(s_t, a_t, w_t\right)\right)^2$$

不难看出条件 $(a)$ 仍然满足。



**Algorithm 8.3: Deep Q-learning (off-policy version)**

**Initialization:** A main network and a target network with the same initial parameter.
**Goal:** Learn an optimal target network to approximate the *optimal* action values from the experience samples generated by a given behavior policy $\pi_b$.

```
Store the experience samples generated by $\pi_b$ in a replay buffer $\mathcal{B}=\{(s,a,r,s')\}$
    For each iteration, do
        Uniformly draw a mini-batch of samples from $\mathcal{B}$
        For each sample $(s,a,r,s')$, calculate the target value as $y_T = r + \gamma \max_{a\in\mathcal{A}(s')} \hat{q}(s',a,w_T)$, where $w_T$ is the parameter of the target network
        Update the main network to minimize $(y_T - \hat{q}(s,a,w))^2$ using the mini-batch of samples
    Set $w_T = w$ every $C$ iterations
```

## 策略梯度方法

“一个好的策略应该能够使得状态的价值最大化”，这句话是策略梯度方法的核心思想。优化的目标有如下两类及其不同但是等价的表达形式：

| $Metric$ | $Expression1$ | $Expression2$ | $Expression3$ |
|---|---|---|---|
| $\bar{v}_\pi$ | $\sum_{s \in \mathcal{S}} d(s) v_\pi(s)$ | $\mathbb{E}_{S \sim d}\left[v_\pi(S)\right]$ | $\lim_{n \rightarrow \infty} \mathbb{E}\left[\sum_{t=0}^n \gamma^t R_{t+1}\right]$ |
| $\bar{r}_\pi$ | $\sum_{s \in S} d_\pi(s) r_\pi(s)$ | $\mathbb{E}_{S \sim d_\pi}\left[r_\pi(S)\right]$ | $\lim_{n \rightarrow \infty} \frac{1}{n} \mathbb{E}\left[\sum_{t=0}^{n-1} R_{t+1}\right]$ |

策略梯度方法的优化目标的梯度有如下统一形式：

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_{s\in\mathcal{S}}\eta(s)\sum_{a\in\mathcal{A}}\nabla_\theta\pi(a\mid s,\theta)q_\pi(s,a)\\ &= \mathbb{E}_{s\sim\eta,A\sim\pi(S,\theta)}\left[\nabla_\theta\ln\pi\left(A\mid S,\theta\right)q_\pi\left(S,A\right)\right]\end{aligned}$$

其中 $\eta$ 是状态的分布。可以证明，当使用 $\bar{r}_\pi$ 或者 $\bar{v}_\pi$ 作为优化目标时，$\eta$ 就是 $d_\pi(s)$：

$$\begin{aligned}\nabla_\theta\bar{r}_\pi = (1-\gamma)\nabla_\theta\bar{v}_\pi &\approx \sum_{s\in S}d_\pi(s)\sum_{a\in A}\nabla_\theta\pi(a\mid s,\theta)q_\pi(s,a)\\ &= \mathbb{E}\left[\nabla_\theta\ln\pi(A\mid S,\theta)q_\pi(S,A)\right]\end{aligned}$$

其中 $S\sim d_\pi$ 且 $A\sim\pi\left(S,\theta\right)$。优化 $J(\theta)=\bar{r}_\pi$ 的随机梯度方法为

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

$q_t\left(s_t,a_t\right)$ 的计算借助于蒙特卡洛方法，根据状态动作价值的定义：

$$\begin{aligned}q_{\pi_k}(s,a) &= \mathbb{E}\left[G_t\mid S_t=s,A_t=a\right]\\ &= \mathbb{E}\left[R_{t+1}+\gamma R_{t+2}+\gamma^2R_{t+3}+\ldots\mid S_t=s,A_t=a\right]\\ &\approx \underbrace{r_{t+1}+\gamma r_{t+2}+\gamma^2r_{t+3}+\ldots}_{q_k(s,a)}\end{aligned}$$

在数学上其收敛性由大数定律保证。

**Algorithm 9.1: Policy Gradient by Monte Carlo (REINFORCE)**

**Initialization:** Initial parameter $\theta$; $\gamma\in(0,1)$; $\alpha>0$.
**Goal:** Learn an optimal policy for maximizing $J(\theta)$.

```
For each episode, do
    Generate an episode $\{s_0,a_0,r_1,\ldots,s_{T-1},a_{T-1},r_T\}$ following $\pi(\theta)$.
    For $t=0,1,\ldots,T-1$:
        Value update: $q_t(s_t,a_t)=\sum_{k=t+1}^{T}\gamma^{k-t-1}r_k$
        Policy update: $\theta\leftarrow\theta+\alpha\nabla_\theta\ln\pi(a_t|s_t,\theta)q_t(s_t,a_t)$
```

## 策略更新与价值更新结合

Actor-critic算法结合了策略更新和价值更新的过程。这一类方法都有统一的优化目标 $J(\theta)$ 和随机梯度方法的优化过程：

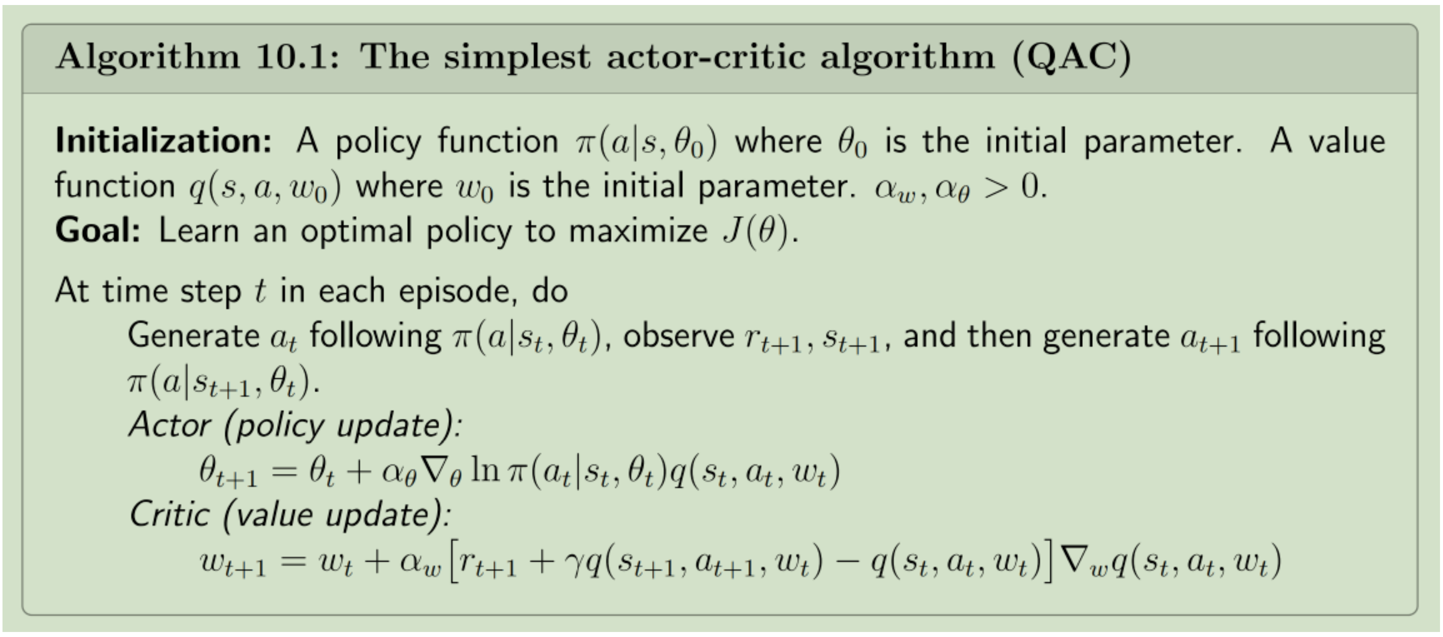
$$\begin{aligned}\theta_{t+1} &= \theta_t+\alpha\nabla_\theta J\left(\theta_t\right)\\ &= \theta_t+\alpha\mathbb{E}_{S\sim\eta,A\sim\pi}\left[\nabla_\theta\ln\pi\left(A\mid S,\theta_t\right)q_\pi\left(S,A\right)\right]\end{aligned}$$

$$\Downarrow \text{随机梯度估计}$$

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

不同算法的核心区别在于不同的场景下优化目标 $J(\theta)$ 的具体形式的不同以及随机梯度估计的方法不同。

QAC算法中的动作价值函数的更新采用了前述的值函数估计方法，策略更新使用了前述的策略梯度方法。



**Algorithm 10.1: The simplest actor-critic algorithm (QAC)**

```
Initialization: A policy function $\pi(a|s,\theta_0)$ where $\theta_0$ is the initial parameter. A value function $q(s,a,w_0)$ where $w_0$ is the initial parameter. $\alpha_w, \alpha_\theta > 0$.
Goal: Learn an optimal policy to maximize $J(\theta)$.

At time step $t$ in each episode, do
    Generate $a_t$ following $\pi(a|s_t,\theta_t)$, observe $r_{t+1}, s_{t+1}$, and then generate $a_{t+1}$ following $\pi(a|s_{t+1},\theta_t)$.
    Actor (policy update):
        $\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \ln \pi(a_t|s_t,\theta_t) q(s_t,a_t,w_t)$
    Critic (value update):
        $w_{t+1} = w_t + \alpha_w \big[r_{t+1} + \gamma q(s_{t+1},a_{t+1},w_t) - q(s_t,a_t,w_t)\big] \nabla_w q(s_t,a_t,w_t)$
```

为了减少随机梯度估计的方差，增强算法的稳定性，A2C算法引入了一个baseline $b(s)$ ，可以证明，最优的 $b(s)$ 为

$$b^*(s) = \frac{\mathbb{E}_{A\sim\pi}\left[\|\nabla_\theta \ln \pi(A \mid s,\theta_t)\|^2 q_\pi(s,A)\right]}{\mathbb{E}_{A\sim\pi}\left[\|\nabla_\theta \ln \pi(A \mid s,\theta_t)\|^2\right]}, \quad s \in S.$$

为了计算方便将权重 $\|\nabla_\theta \ln \pi\left(A \mid s,\theta_t\right)\|^2$ 移除，得到次优的 $b(s)$

$$b^\dagger(s) = \mathbb{E}_{A\sim\pi}\left[q_\pi(s,A)\right] = v_\pi(s)$$

此时随机梯度算法为

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

其中

$$\delta_\pi(S,A) \doteq q_\pi(S,A) - v_\pi(S)$$

```
Algorithm 10.2: Advantage actor-critic (A2C) or TD actor-critic

Initialization: A policy function $\pi(a|s,\theta_0)$ where $\theta_0$ is the initial parameter. A value
function $v(s,w_0)$ where $w_0$ is the initial parameter. $\alpha_w,\alpha_\theta > 0$.
Goal: Learn an optimal policy to maximize $J(\theta)$.

At time step $t$ in each episode, do
    Generate $a_t$ following $\pi(a|s_t,\theta_t)$ and then observe $r_{t+1}, s_{t+1}$.
    Advantage (TD error):
        $\delta_t = r_{t+1} + \gamma v(s_{t+1},w_t) - v(s_t,w_t)$
    Actor (policy update):
        $\theta_{t+1} = \theta_t + \alpha_\theta \delta_t \nabla_\theta \ln \pi(a_t|s_t,\theta_t)$
    Critic (value update):
        $w_{t+1} = w_t + \alpha_w \delta_t \nabla_w v(s_t,w_t)$
```

前面的算法都是on-policy的方式，当我们希望**用一个策略指导下的智能体与环境交互的样本去优化另一个策略**，就需要借助于off-policy的方式，其优化目标变成

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

其中 $d_\beta(s)$ 是策略 $\beta$ 下的稳态分布而 $v_\pi$ 是策略 $\pi$ 下的状态价值。**借助于importance sampling可以用行为策略 $\beta$ 下的状态的稳态分布估计目标策略 $\pi$ 的状态的稳态分布**，其梯度为

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

其中状态分布 $\rho$ 为

$$\rho(s) \doteq \sum_{s'\in\mathcal{S}} d_\beta(s')\operatorname{Pr}(s\mid s'), \qquad s\in\mathcal{S}$$

其中 $\operatorname{Pr}(s\mid s')=\sum_{k=0}^{\infty}\gamma^k\left[P_\pi^k\right]_{s's}=\left[(I-\gamma P_\pi)^{-1}\right]_{s's}$ 是在策略 $\pi$ 下从状态 $s'$ 转移向 $s$ 的衰减全概率。于是，再结合前述的减小随机估计方差的次优baseline的方法我们有

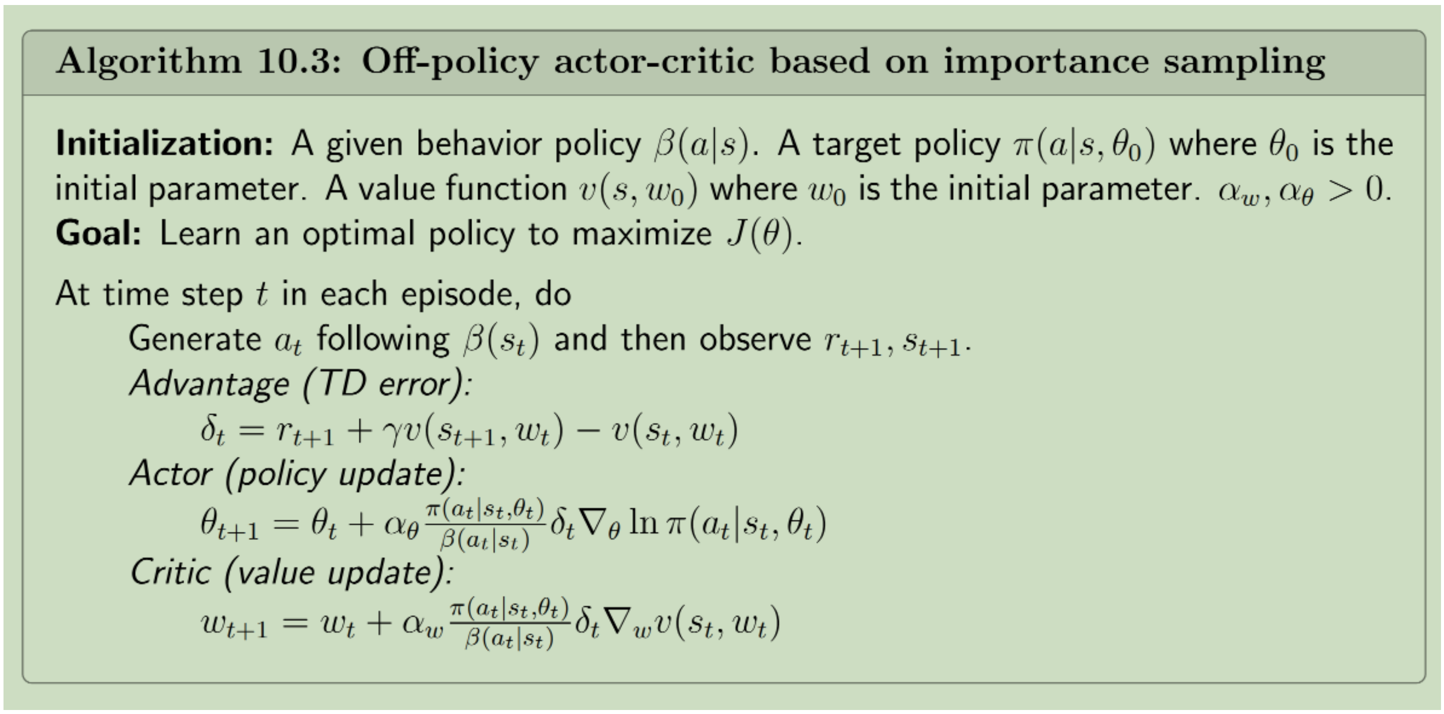
$$\nabla_\theta J(\theta)=\mathbb{E}\left[\frac{\pi(A|S,\theta)}{\beta(A|S)}\nabla_\theta \ln \pi(A|S,\theta) q_\pi(S,A)-v_\pi(S)\right]$$

相应的随机梯度算法为

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

类似于前述方法，这其中的 $q_t(s_t,a_t)-v_t(s_t)$ 可以用时间差分误差代替，即

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



**Algorithm 10.3: Off-policy actor-critic based on importance sampling**

**Initialization:** A given behavior policy $\beta(a|s)$. A target policy $\pi(a|s, \theta_0)$ where $\theta_0$ is the initial parameter. A value function $v(s, w_0)$ where $w_0$ is the initial parameter. $\alpha_w, \alpha_\theta > 0$.
**Goal:** Learn an optimal policy to maximize $J(\theta)$.

```
At time step $t$ in each episode, do
    Generate $a_t$ following $\beta(s_t)$ and then observe $r_{t+1}, s_{t+1}$.
    *Advantage (TD error):*
        $\delta_t = r_{t+1} + \gamma v(s_{t+1}, w_t) - v(s_t, w_t)$
    *Actor (policy update):*
        $\theta_{t+1} = \theta_t + \alpha_\theta \frac{\pi(a_t|s_t,\theta_t)}{\beta(a_t|s_t)} \delta_t \nabla_\theta \ln \pi(a_t|s_t, \theta_t)$
    *Critic (value update):*
        $w_{t+1} = w_t + \alpha_w \frac{\pi(a_t|s_t,\theta_t)}{\beta(a_t|s_t)} \delta_t \nabla_w v(s_t, w_t)$
```

前述所有算法中on-policy的方式样本效率会比较低；另外，如果是概率策略，那么能处理的动作空间是有限的，因为概率策略需要从所有的动作概率分布上采样，虽然可以将无限动作空间离散化，但是这样做难免会粗糙。于是，off-policy的、确定性策略（Deterministic Policy）的算法被提出来。**所谓的确定性策略是指某个状态下，只有一个动作有一定的概率被选择，其他的动作被选择的概率为0**。

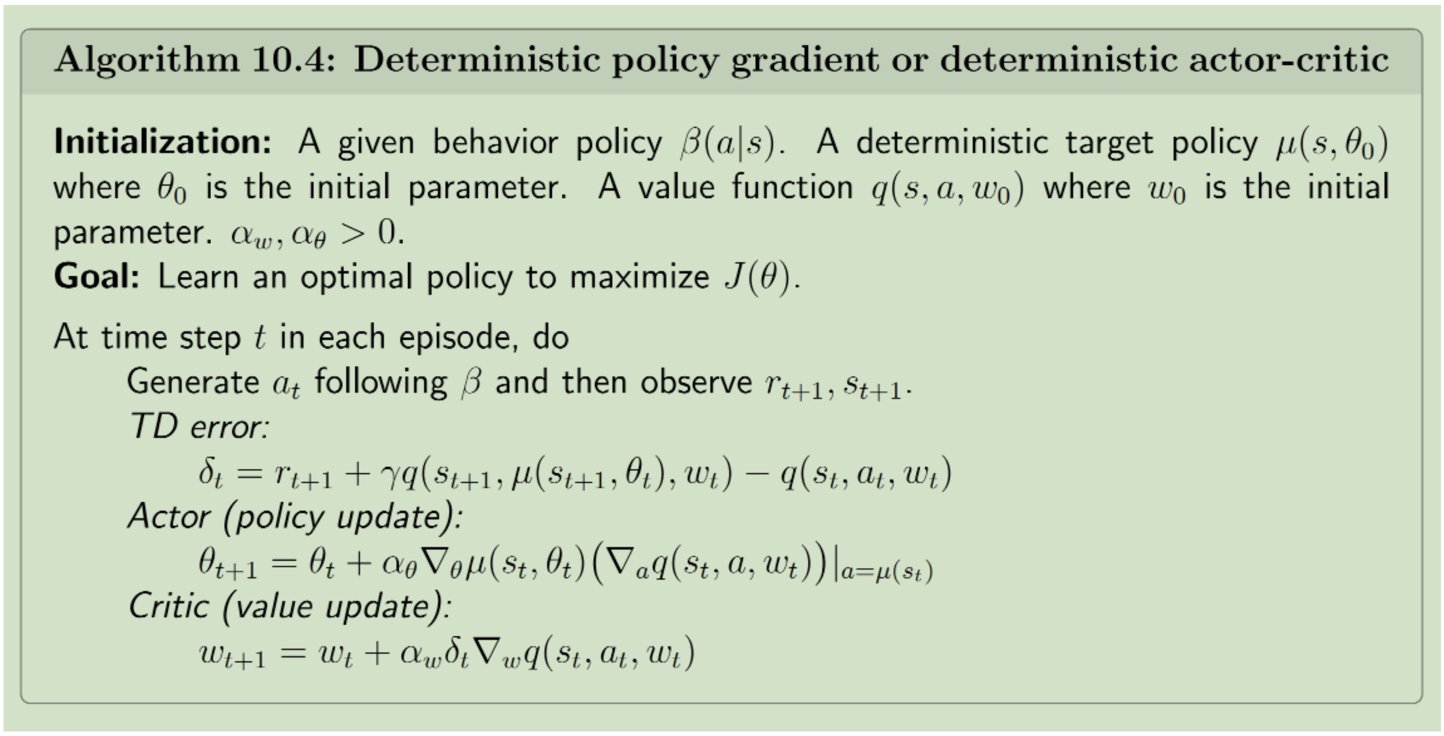
优化目标为确定性策略 $\mu$ 的Average Reward $\bar{r}_\mu$

$$
\begin{aligned}
J(\theta) = \bar{r}_\mu &= \sum_{s \in \mathcal{S}} d_\mu(s) r_\mu(s) \\
&= \mathbb{E}_{S \sim d_\mu}\left[r_\mu(S)\right]
\end{aligned}
$$

从优化目标中可以看出稳态分布只和 $\mu$ 有关，不借助于行为策略的采样去估计目标分布，所以下面的算法中没有使用importance sampling。其随机梯度算法为

$$
\begin{aligned}
\theta_{t+1} &= \theta_t + \alpha_\theta \mathbb{E}_{S \sim d_\mu}\left[\nabla_\theta \mu(S) \left(\nabla_a q_\mu(S, a)\right)\big|_{a=\mu(S)}\right] \\
&\approx \theta_t + \alpha_\theta \nabla_\theta \mu(s_t) \left(\nabla_a q_\mu(s_t, a)\right)\big|_{a=\mu(s_t)}
\end{aligned}
$$

其中 $\mu$ 是确定性策略。



**Algorithm 10.4: Deterministic policy gradient or deterministic actor-critic**

**Initialization:** A given behavior policy $\beta(a|s)$. A deterministic target policy $\mu(s,\theta_0)$ where $\theta_0$ is the initial parameter. A value function $q(s,a,w_0)$ where $w_0$ is the initial parameter. $\alpha_w, \alpha_\theta > 0$.
**Goal:** Learn an optimal policy to maximize $J(\theta)$.

```
At time step $t$ in each episode, do
    Generate $a_t$ following $\beta$ and then observe $r_{t+1}, s_{t+1}$.
    TD error:
        $\delta_t = r_{t+1} + \gamma q(s_{t+1}, \mu(s_{t+1}, \theta_t), w_t) - q(s_t, a_t, w_t)$
    Actor (policy update):
        $\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \mu(s_t, \theta_t)\big(\nabla_a q(s_t, a, w_t)\big)|_{a=\mu(s_t)}$
    Critic (value update):
        $w_{t+1} = w_t + \alpha_w \delta_t \nabla_w q(s_t, a_t, w_t)$
```

## Soft Actor Critic算法（SAC）

在具体的应用中，动作空间和状态空间往往会有多模态的形式，例如自动驾驶中动作有加速度、方向盘转角两个模态，在不同的驾驶场景中状态空间也可能有多种模态。传统的强化学习只能收敛到选择最大的动作价值的策略，虽然可以是用类似 $\varepsilon$ 贪婪的方法探索一些非最大价值的动作，但是这个方法相比于在价值中引入了熵的强化学习方法来说有诸多弊端。其中最重要的，便是对于**多模态的Q值**情况下，soft Q-learning可以学习到直接对应于 $Q(S,A)$ 分布的策略函数。最大熵强化学习方法在奖励中加入了策略函数的熵，熵在信息论的领域中是对不确定性的度量，熵的引入增强了策略的探索性。在奖励中引入熵的强化学习优化过程如下

$$\pi^* = \arg\max_\pi \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[\sum_t R(s_t,a_t) + \alpha H(\pi(\cdot \mid s_t))\right]$$

其中 $H(\cdot) = -\ln(\cdot)$ 是信息熵。可以单独考察 $\sum_t \alpha H(\pi(\cdot \mid s_t))$ 获得intuition：在一个episode中，要想使这个值比较大，同时又要满足 $\sum_{a\in\mathcal{A}} \pi(a \mid s_t) = 1$ ，那么每个时刻对每个动作的选择概率应当尽量都接近0但是总和为1，故而熵的引入增强了策略的探索性。注意！对于连续的动作空间，这里的 $\pi$ 实际上是动作概率密度，这一点在SAC原论文、OpenAI Spining Up都有明确支持，网上很多公开资料关于这一块的表述或者理解是有问题的。对于连续型随机变量动作 $A$ ，其在某个点 $a_t$ 处的概率是 $0$ ，所以这里的 $\pi(a_t \mid s_t)$ 也自然应该是概率密度，不然它的值就一直是零了。搞清楚这个概念，在后面对 $a_t$ 进行非线性压缩变换到 $(-1,1)$ 的时候，其概率密度 $\pi$ 也需要借助概率密度变换定理进行修正，如果这个概念没有理清，会很奇怪“为什么会将概率密度当作概率”。

引入了信息熵的动作价值为

$$Q_{soft}(s_t,a_t) = r(s_t,a_t) + \gamma \mathbb{E}_{s_{t+1},a_{t+1}\sim\mu}[Q_{soft}(s_{t+1},a_{t+1}) - \alpha \ln \pi(a_{t+1}|s_{t+1})]$$

又参考

$$Q(s_t,a_t)=r(s_t,a_t)+\gamma\mathbb{E}_{s_{t+1}\sim\rho}[V(s_{t+1})]$$

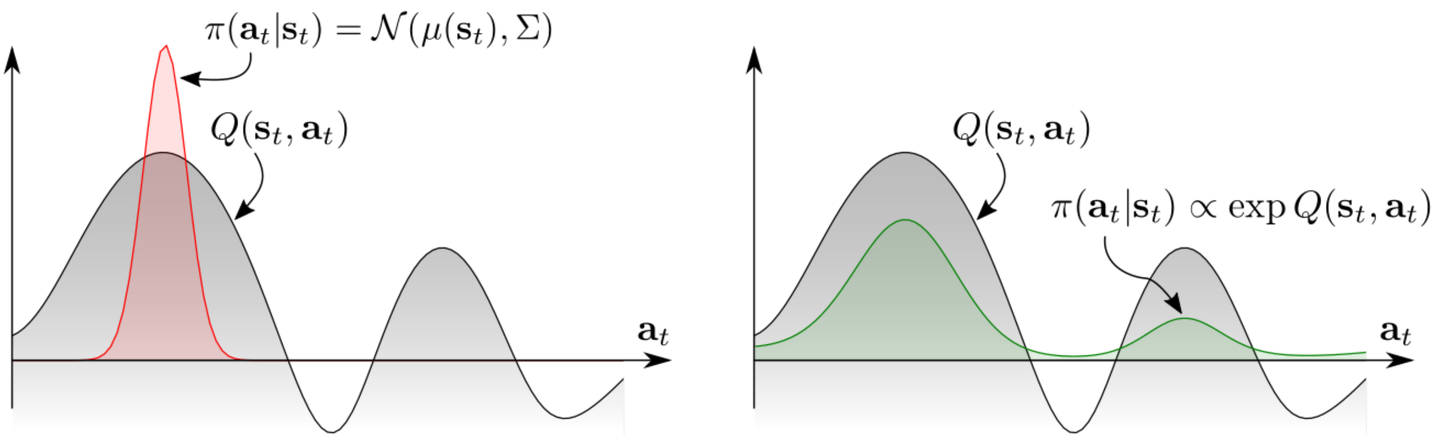
所以可以得到

$$V_{soft}(s_t)=\mathbb{E}_{a_t\sim\pi}[Q_{soft}(s_t,a_t)-\alpha\ln\pi(a_t|s_t)] \tag{A}$$

在传统的Q-learning过程中，根据贝尔曼最优方程，我们有

$$V(s_t)=\max_a Q(s_t,a)$$

在多模态的Q值中如果使用传统的Q-learning方法，那么获得的策略函数可能只对一种模态有较好的效果。如下图左边所示：


$\pi(\mathbf{a}_t|\mathbf{s}_t)=\mathcal{N}(\mu(\mathbf{s}_t),\Sigma)$
$Q(\mathbf{s}_t,\mathbf{a}_t)$
$\mathbf{a}_t$
$Q(\mathbf{s}_t,\mathbf{a}_t)$
$\pi(\mathbf{a}_t|\mathbf{s}_t)\propto\exp Q(\mathbf{s}_t,\mathbf{a}_t)$
$\mathbf{a}_t$

于是在SAC使用的Soft Q-learning中，我们使用 $\max(\cdot)$ 的平滑版本 $\operatorname{soft}\max(\cdot)$ ——这个操作是凸的但不是严格凸的

$$V_{soft}(s_t)=\operatorname{soft}\max(Q(s_t,a))=-\frac{1}{\alpha}\ln\sum_a\exp\left(-\frac{1}{\alpha}Q_{soft}(s_t,a)\right)$$

直接获得对应于 $Q(S,A)$ 分布的策略函数，如上图右边。为了能够达到策略分布直接对应于 $Q(S,A)$ 的分布特征，可以借助玻尔兹曼分布，并令其能量函数为 $-1/\alpha\cdot Q_{soft}(s_t,a_t)$，进而

$$\pi(a_t|s_t)\propto\exp\left(-\frac{1}{\alpha}Q_{soft}(s_t,a_t)\right)$$

然后，综合考虑动作价值与状态价值的关系、上述 $V_{soft}(s_t)$ 的计算过程、所有动作概率和为 $1$ 不难写出

$$\pi(s_t,a_t)=\frac{\exp\left(-\frac{1}{\alpha}Q_{soft}(s_t,a_t)\right)}{\sum_a\exp\left(-\frac{1}{\alpha}Q_{soft}(s_t,a)\right)}$$

有了上面的分析，不难写出Soft贝尔曼方程：

$$q_\pi(s,a)=r(s,a)+\gamma\sum_{s'\in\mathcal{S}}\mathcal{P}_{ss'}^a\sum_{a'\in\mathcal{A}}\pi\left(a'\mid s'\right)\left(q_\pi\left(s',a'\right)-\alpha\ln\left(\pi\left(a'\mid s'\right)\right)\right)$$

基于贝尔曼方程，便可以使用策略评估和策略改善的框架迭代求解最优的策略和最优的动作价值。

根据定义我们有

$$q_\pi(s,a)=\sum_{r\in\mathcal{R}}p(r\mid s,a)r+\gamma\sum_{s'\in\mathcal{S}}p\left(s'\mid s,a\right)v_\pi\left(s'\right)$$

我们可以写出 $Q_{soft}$ 场景下的优化目标：

$$J_Q(\psi)=\mathbb{E}_{(s_t,a_t,s_{t+1})\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\psi(s_t,a_t)-\left(r(s_t,a_t)+\gamma V_\theta(s_{t+1})\right)\right)^2\right]$$

又根据上述等式 $(A)$，优化目标又可以写成

$$J_Q(\psi)=$$
$$\mathbb{E}_{(s_t,a_t,s_{t+1})\sim\mathcal{D},a_{t+1}\sim\pi_\phi}\left[\frac{1}{2}\left(Q_\psi(s_t,a_t)-\left(r(s_t,a_t)+\gamma\left(Q_\theta\left(s_{t+1},a_{t+1}\right)-\alpha\ln\pi_\phi\left(a_{t+1}|s_{t+1}\right)\right)\right)\right)^2\right]$$

其中 $\mathcal{D}$ 是一个replay buffer。这里同样使用了目标网络（$\theta$ ）和主网络（$\psi$ ）两套神经网络，但是网络的参数更新的方式不同，SAC中使用了exponentially moving average方法更新目标网络的参数，其效果是**可以对近期的数据赋予了更大的权重，而对较早的数据赋予较小的权重，平滑目标网络的更新，使得训练更加稳定**。在每个梯度反向传播参数更新的步骤中增加了额外的一步参数更新

$$\theta\leftarrow\tau\psi+(1-\tau)\theta$$

不过论文中指出Q-learning中使用的周期性的参数更新方式也是一种选择，并给出了实验对比结果。

对于策略更新，目标是最小化当前策略分布 $\pi'$ 与目标分布之间的KL散度：

$$\pi_{\text{new}}=\arg\min_{\pi'}D_{KL}\left(\pi'(\cdot\mid s)\middle\|\underbrace{\frac{\exp\left(\frac{1}{\alpha}Q^{\pi_{\text{old}}}(s,\cdot)\right)}{Z^{\pi_{\text{old}}}(s,\cdot)}}_{\text{target distribution}}\right)$$

即最小化KL散度的期望

$$\begin{aligned}J_\pi(\phi)&=\mathbb{E}_{s_t\sim\mathcal{D}}\left[D_{KL}\left(\pi_\phi\left(\cdot\mid s_t\right)\middle\|\frac{\exp Q_\psi\left(s_t,\cdot\right)}{Z_\psi\left(s_t\right)}\right)\right]\\&=\mathbb{E}_{s_t\sim D,a_t\sim\pi_\phi}\left[\ln\left(\frac{\pi_\phi(a_t\mid s_t)}{\exp\left(\frac{1}{\alpha}Q_\psi(s_t,a_t)-\ln Z(s_t)\right)}\right)\right]\\&=\mathbb{E}_{s_t\sim D,a_t\sim\pi_\phi}\left[\ln\pi_\phi(a_t\mid s_t)-\frac{1}{\alpha}Q_\psi(s_t,a_t)+\ln Z(s_t)\right]\end{aligned}$$

以往我们通过概率 $\pi\left(a\mid\pi\right)$ 采样动作，这样的做法会使得梯度的计算变得不可微，SAC使用了重参数化技巧（reparameterization trick）来生成动作

$$a_t=f_\phi(\epsilon_t;s_t)=f_\phi^\mu(s_t)+\epsilon_t\odot f_\phi^\sigma(s_t)$$

其中，$f_\phi^\mu(s_t)$ 是策略网络的输出的均值，表示在状态 $s_t$ 下的动作分布的均值；$f_\phi^\sigma(s_t)$ 是策略网络的输出的方差，表示在状态 $s_t$ 下的动作分布的方差；$\epsilon_t$ 是从标准正态分布 $\mathcal{N}(0,1)$ 中采样的噪声；$\odot$ 表示逐元素乘法。再去掉不影响梯度值的常量 $\ln Z(s_t)$，优化目标 $J_\pi$ 转化成

$$J_\pi(\phi)=\mathbb{E}_{s_t\sim\mathcal{D},\epsilon_t\sim\mathcal{N}}\left[\ln\pi_\phi\left(f_\phi(\epsilon_t;s_t)\mid s_t\right)-\frac{1}{\alpha}Q_\theta(s_t,f_\phi(\epsilon_t;s_t))\right]$$

其随机梯度为

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \ln \pi_\phi(a_t|s_t) + (\nabla_{a_t} \ln \pi_\phi(a_t|s_t) - \frac{1}{\alpha}\nabla_{a_t} Q(s_t, a_t))\nabla_\phi f(\epsilon_t; s_t)$$

其中 $a_t$ 由上述重参数化技巧估算。

论文2的SAC算法伪代码如下，不过当前工作的核心是算法的数学原理，所以并没有使用温度参数的自动调整；同时为了训练的稳定，原论文还使用了双值Q网络的方法，这里不展开讨论。

**Algorithm 1** Soft Actor-Critic

```
Input: $\theta_1, \theta_2, \phi$                                              ▷ Initial parameters
  $\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$                          ▷ Initialize target network weights
  $\mathcal{D} \leftarrow \emptyset$                                                ▷ Initialize an empty replay pool
  for each iteration do
      for each environment step do
          $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$                              ▷ Sample action from the policy
          $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$                  ▷ Sample transition from the environment
          $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$   ▷ Store the transition in the replay pool
      end for
      for each gradient step do
          $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$      ▷ Update the Q-function parameters
          $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$                        ▷ Update policy weights
          $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$                          ▷ Adjust temperature
          $\bar{\theta}_i \leftarrow \tau\theta_i + (1 - \tau)\bar{\theta}_i$ for $i \in \{1, 2\}$        ▷ Update target network weights
      end for
  end for
Output: $\theta_1, \theta_2, \phi$                                             ▷ Optimized parameters
```

这里补充一个重参数化技巧获取动作 $a_t$ 时需要用到的**概率密度变换定理：**

> 设随机变量 $X$ 有概率密度函数 $f(x)$, $x \in (a, b)$ （$a$ 、$b$ 可以为 $\infty$），而 $y = g(x)$ 在 $x \in (a, b)$ 上是严格单调的连续函数，存在唯一的反函数 $x = h(y)$, $y \in (\alpha, \beta)$ 并且 $h'(y)$ 存在且连续，那么 $Y = g(X)$ 也是连续型随机变量且有概率密度函数
>
> $$p(y) = f\left(h(y)\right)\left|h'(y)\right|$$

由于仿真环境可能需要将 $a_t$ 映射到 $[-1, 1]$ 的范围内，所以用神经网络生成某个正态分布的均值 $\mu$ 和标准差 $\sigma$ 后

$$Z \sim \rho(z) \doteq \mathcal{N}\left(\mu, \sigma^2\right)$$

当获得一个采样 $z$ 以后，需要通过 $a_t = \tanh(z)$ 将其变换到 $(-1, 1)$ 上，但是 $\tanh(\cdot)$ 是一个非线性变换，所以变换后得到的 $a_t$ 的概率密度已经变了，需要借助概率密度变换定理对其进行修正：

$$\ln \pi\left(a \mid s\right) = \ln \rho(z) - \ln\left(1 - \tanh^2(z)\right)$$

另外，如果动作的维度不是1，$a = \left[a^{(1)}, a^{(2)}, \ldots, a^{(n)}\right]$ ，此时需要求动作的联合分布的概率密度，同时考虑到各个维度的独立性，所以可以对返回的概率密度对数进行求和操作：

$$\begin{aligned}\ln \pi\left(a^{(1)}, a^{(2)}, \ldots, a^{(d)} \mid s\right) &= \ln \pi\left(a^{(1)} \mid s\right) \pi\left(a^{(2)} \mid s\right) \ldots \pi\left(a^{(d)} \mid s\right) \\ &= \ln \pi\left(a^{(1)} \mid s\right) + \ln \pi\left(a^{(2)} \mid s\right) + \ldots + \ln \pi\left(a^{(d)} \mid s\right)\end{aligned}$$

## 模型预测控制（MPC）