策略网络

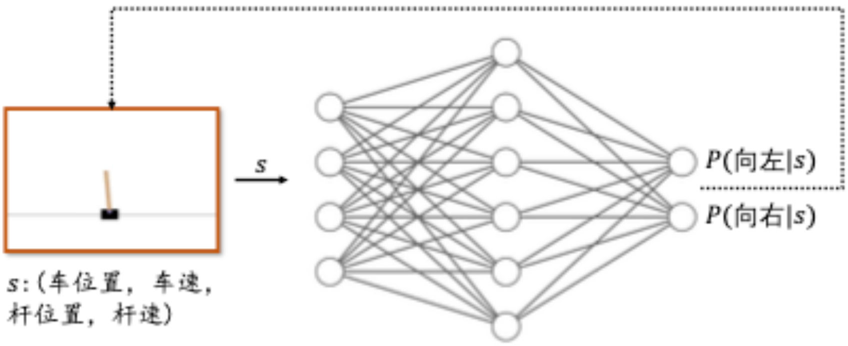
策略的**输入**是状态s，**输出**为动作a或动作的分布$\pi_\theta(a|s)$，如果是动作的分布，那么满足概率之和

$$\sum_{a\in A}\pi_\theta(a|s)=1$$

其中A为所有动作的集合。该网络表示策略，称为策略网络。将策略 函数具体化为输入节点为 4，中间多个全连接隐藏层，输出层的输出节点数为 2 的神经网 络。在交互时，选择概率最大的动作。所以｜号就是**输入｜输出**的意思

$$a_t=\underset{a}{argmax}\,\pi_\theta(a|s_t)$$

最简单的策略网络：



PPO网络

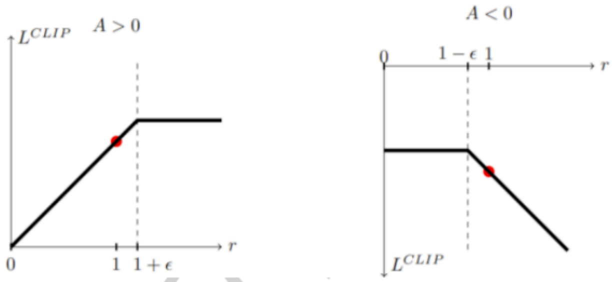
重要性采样

利用重要性采样，可以使用同一个运行轨迹的记录去训练多次网络。每训练一次网络，策略网络就已经发生了变化，按理说就不能使用原先的运行轨迹了。但是又了重要性采样，只要用新概率除旧概率就行

$$\begin{aligned}\mathbb{E}_{\tau\sim p}[f(\tau)]&=\int p(\tau)f(\tau)d\tau\\&=\int\frac{p(\tau)}{q(\tau)}q(\tau)f(\tau)d\tau\\&=\mathbb{E}_{\tau\sim q}\left[\frac{p(\tau)}{q(\tau)}f(\tau)\right]\end{aligned}$$

但是重要性采样的前提是，旧的策略网络和新的策略网络的分布不能相差太大，于是就加了一些约束

$$\mathcal{L}_{\theta}^{CLIP}(\theta)=\widehat{\mathbb{E}}_t\left[min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\bar{\theta}}(a_t|s_t)}\hat{A}_t, clip\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\bar{\theta}}(a_t|s_t)},1-\epsilon,1+\epsilon\right)\hat{A}_t\right)\right]$$



真实环境中的奖励r 并不是分布在 0 周围，很多游戏的奖励全是正数，使得$R(\tau)$总是 大于 0，**网络会倾向于增加所有采样到的动作的概率，而未采样到的动作出现的概率也就 相对下降**。这并不是我们希望看到的，我们希望$R(\tau)$能够分布在 0 周围，因此我们引入一 个偏置变量b，称之为基准线，它代表了回报$R(\tau)$的平均水平

值函数方法

状态值函数(State Value Function，V 函数)

它定义为**从状态s_t开始**，在策略π控制下能获得的期望回报值

$$V^\pi(s_t)=\mathbb{E}_{\tau\sim p(\tau)}\left[R(\tau_{t:T})|\tau_{s_t}=s_t\right]$$

状态值函数的数值**反映了当前策略下状态的好坏**，$V\pi(s_t)$越大，说明当前状态的总回报期望越大。

状态值函数的贝尔曼方程：

$$\begin{aligned}V^\pi(s_t)&=\mathbb{E}_{\tau\sim p(\tau)}[r_t+\gamma R(\tau_{t+1:T})]\\&=\mathbb{E}_{\tau\sim p(\tau)}[r_t+\gamma V^\pi(s_{t+1})]\end{aligned}$$

在所有策略中，最优策略$\pi*$是指能取得$V\pi(s)$最大值的策略，对于最优策略，同样满足贝尔曼方程

$$V^*(s_t)=\mathbb{E}_{\tau\sim p(\tau)}[r_t+\gamma V^*(s_{t+1})]$$

状态-动作值函数(State-Action Value Function，Q 函数)

它定义为从状态s_t并执行动作 a_t的双重设定下，在策略π控制下能获得的期望回报值

$$Q^\pi(s_t,a_t)=\mathbb{E}_{\tau\sim p(\tau)}\left[R(\tau_{t:T})|\tau_{a_t}=a_t,\tau_{s_t}=s_t\right]$$

Q函数和V函数的关系：

$$V^\pi(s_t)=\mathbb{E}_{a_t\sim\pi(a_t|s_t)}[Q^\pi(s_t,a_t)]$$

当V的下一个动作采样子V的策略时，两个期望值就想等了，此时

$$V^*(s_t) = \max_{a_t} Q^*(s_t, a_t)$$

同时

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

$$= \mathbb{E}_{\tau\sim p(\tau)}\left[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})\right]$$

把Q函数和V函数之间的差值定义为优势值函数，反映了在状态s下采取动作a比平均水平的差异

$$A^\pi(s, a) := Q^\pi(s, a) - V^\pi(s)$$

TD时差分析法

回顾 V 函数的贝尔曼方程

$$V^\pi(s_t) = \mathbb{E}_{\tau\sim p(\tau)}[r_t + \gamma V^\pi(s_{t+1})]$$

因此构造 TD 误差项$\delta = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$，通过如下方式

$$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha\big(r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)\big)$$

其中$\alpha \in [0,1]$为更新步长。同样的方式，Q 函数的贝尔曼最优方程为

$$Q^*(s_t, a_t) = \mathbb{E}_{\tau\sim p(\tau)}\left[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})\right]$$

同样的方式，构造 TD 误差项$\delta = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) - Q^*(s_t, a_t)$，并利用

$$Q^*(s_t, a_t) \leftarrow Q^*(s_t, a_t) + \alpha\left(r(s_t, a_t) + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) - Q^*(s_t, a_t)\right)$$

这样直接就可以利用神经网络去估计t+1的Q和当前的Q

DQN算法

利用TD时差分析法直接更新策略网络，把策略网络的损失值定义为优势值函数刚刚好

$$\mathcal{L} = \left(r_t + \gamma \max_a Q_\theta(s_{t+1}, a) - Q_\theta(s_t, a_t)\right)^2$$

由于两个Q都来自于同一个网络，具有强相关性，两项措施解决。添加经验回放池，创建影子网络。影子网络的更新速度慢于训练的target网络，在代码中暂定20个epoch后，影子网络拉取target网络的新参数

Double DQN (Hasselt, Guez, & Silver, 2015) 中目标$r_t + \gamma\bar{Q}\left(s_{t+1}, \max_a Q(s_{t+1}, a)\right)$的 Q 网络和估值的$\bar{Q}$网络被分离，并按着误差函数

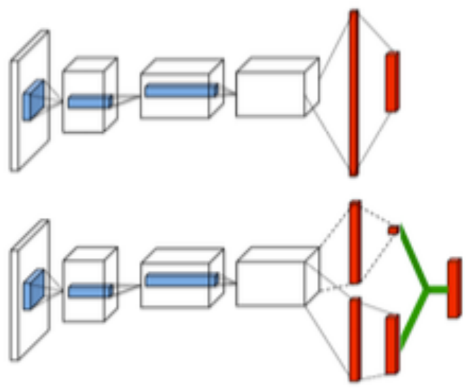
$$\mathcal{L} = \left(r_t + \gamma\bar{Q}\left(s_{t+1}, \max_a Q(s_{t+1}, a)\right) - Q(s_t, a_t)\right)^2$$

优化更新。

Dueling DQN (Wang, Freitas, & Lanctot, 2015) 将网络的输出首先分开为$V(s)$和$A(s,a)$两个中间端，如图 14.20(下)所示，并通过

$$Q(s,a) = V(s) + A(s,a)$$

合成 Q 函数估计$Q(s,a)$，其他部分和 DQN 保存不变。



Actor-Critic 方法

在Actor-Critic中存在两个网络，一个网络用来训练优势值，一个网络用来更新策略。

Actor作为策略网络，loss使用刚开始的方式进行更新

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

后面的一项就作为优势值，式子可以写成：

$$\mathcal{L}^{PG}(\theta) = \hat{\mathbb{E}}_t\left[\log \pi_\theta (a_t|s_t)\hat{A}_t\right]$$

使用优势值的时候要断开critic网络的梯度连接，同时loss添加Entropy Bonus，保证动作的概率不会太集中到某一个动作