强化学习面试

写出Q-learning的公式

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)]$$

Sarsa的公式以及和Q-leaning的区别

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$$

更新Q值的方式不一样：

Sarsa会在下一个状态动作发生后才更新Q值，从而记住每一次错误的探索，表现会更加谨慎，所以收敛的可能会比较慢；

Q-learning会选取下个状态动作的最大价值去更新Q值，只在乎最大化价值，表现会更加冒进，一般收敛较快。

各种DQN讲一下

最初关于值函数的表格型方法，譬如Q-learning[9]，Sarsa[10]，只是解决离散状态下的离散动作问题。随着状态空间的增大，需要引入深度神经网络来“代替”这个表格（表格没有大的存储量）。所以DQN全称为Deep Q Network[4]，可以用来解决连续状态下的离散动作问题。为了逼近真实的值函数，可以将平方误差作为DQN的目标函数：

$$L(\theta) = \frac{1}{2N}\sum_{i=1}^{N}[Q_\theta(s_i,a_i) - (r_i + \gamma \max_{a' \in A} Q_\theta(s_i^{'},a_i^{'}))]^2$$

- 一些技巧：

1. 增加目标网络，使网络训练更稳定

（1）训练用的网络f_θ用于计算预测Q_θ，并在每一步使用梯度下降更新参数；

（2）目标网络f_{θ^-}用于计算$r + \gamma \max Q_{\theta^-}$，损失函数变为：

$$L(\theta) = \frac{1}{2N}\sum_{i=1}^{N}[Q_\theta(s_i,a_i) - (r_i + \gamma \max_{a' \in A} Q_{\theta^-}(s_i^{'},a_i^{'}))]^2$$

（3）每隔c步，目标网络f_{θ^-}的参数与训练网络f_θ的参数同步一次；

2. 使用经验回放池：

（1）使训练样本满足独立假设

（2）缓冲区的样本可以重复使用，从而提高样本的使用效率；

（3）优先级经验回放：TD误差大（$TD_{error} = Q_\theta(s_i, a_i) - (r_i + \gamma \max_{a' \in A} Q_\theta(s_i^{'}, a_i^{'}))$）的样本应该给予更高的权重

- DDQN

用于解决被高估的Q值问题

对比：Q为训练网络，$Q^{'}$为目标网络

1. 普通DQN: $Q(s_t, a_t) \leftarrow r_t + \gamma \max_a Q^{'}(s_{t+1}, a)$
2. DDQN: $Q(s_t, a_t) \leftarrow r_t + \gamma Q^{'}(s_{t+1}, \arg\max_a Q(s_{t+1}, a))$

- Dueling DQN

根据优势函数表示采取不同动作差异性的意义：$A(s, a) = Q(s, a) - V(s)$，神经网络不直接输出Q值，Dueling DQN改为输出状态价值（V值）和优势值（A值），再求和得出Q值，网络架构改动如图3所示。

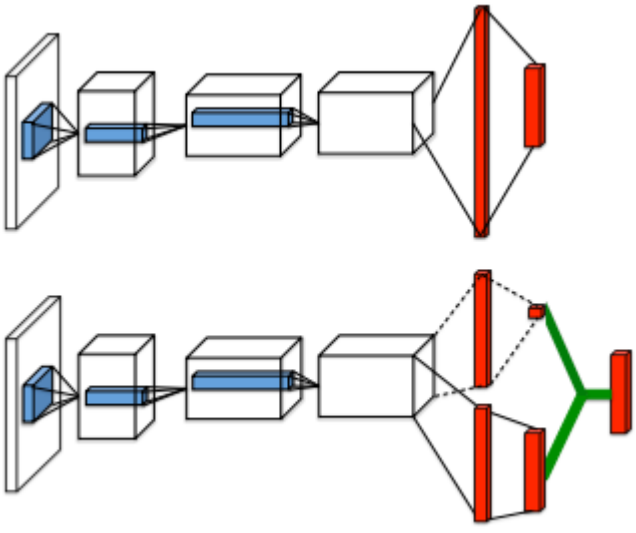


图3 普通DQN（上）和Dueling DQN（下）架构对比[14]

$V(s)$相当于网络对当前状态的一个基本判断，然后再根据当前状态决定采取哪个动作具有优势增益，即$A(s,a)$，所以这种拆分也是十分符合人类的习惯。

然而，这里还有一个网络输出的唯一性问题要解决。试想想，令$V^{'}=V+10, A^{'}=A-10$，则$Q=V+A=V^{'}+A^{'}$，表明无法通过学习Q来唯一确定V和A，这样会导致训练的不稳定性。论文中增加一个$\max_{a^{'}} A(s,a^{'})$约束，于是模型变为$Q=V+A-\max A$，因为所有动作的优势之和必为0，这不改变模型的输出。类似得对V值和A值进行加和求Q值：

$$\begin{aligned} Q &= V+A-\max A \\ &= (V^{'}-10)+(A^{'}+10)-\max(A^{'}-10) \\ &= V^{'}+A^{'}-10 \end{aligned}$$

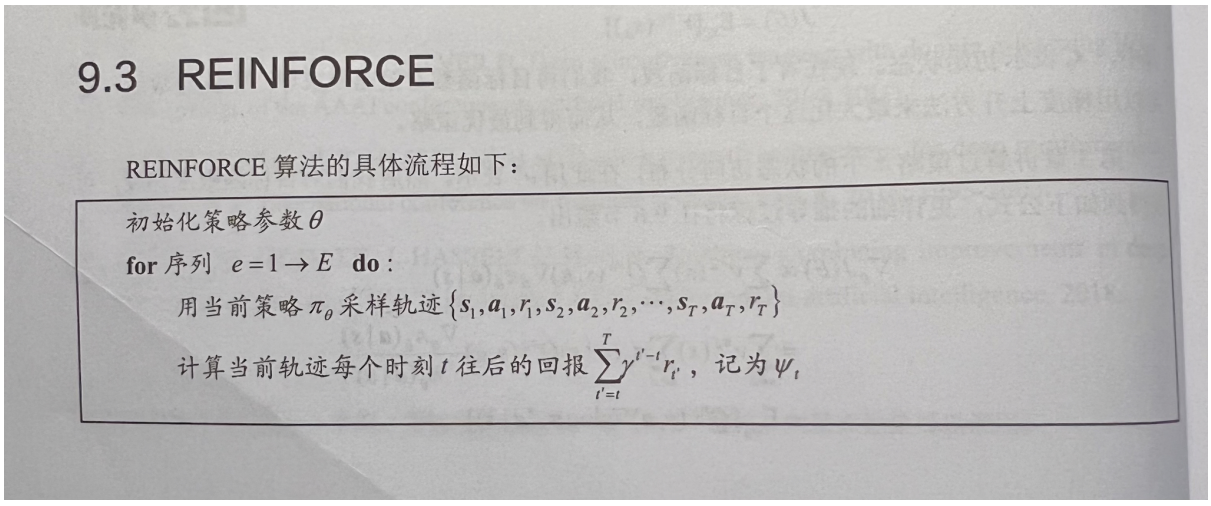
可见网络学习到的V和A是唯一的。

策略梯度(PG)

期望的奖励 $R_\theta=\sum_\tau R(\tau)p_\theta(\tau)$，于是最大化期望奖励:

$\partial R_\theta=\sum_\tau R(\tau)\partial p_\theta(\tau)=\mathbb{E}_{\tau\sim p_\theta(\tau)}[\mathbb{R}(\tau)\partial log p_\theta(\tau)]$ ，对策略的偏导乘以一个权重

REINFORCE算法：



9.3 REINFORCE

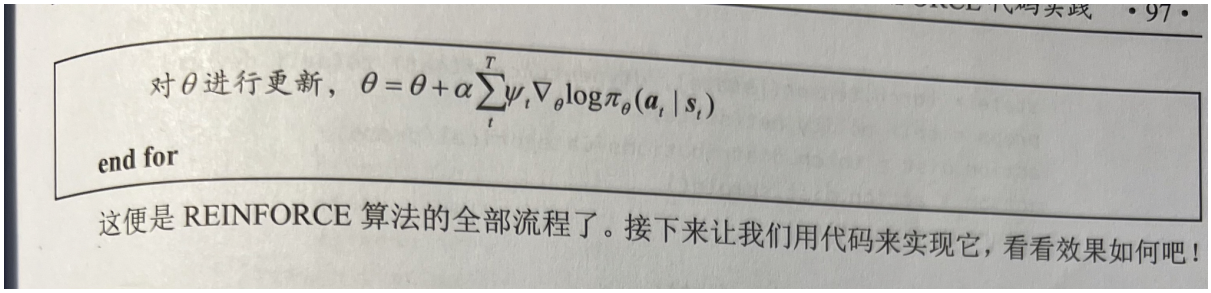
REINFORCE 算法的具体流程如下：

初始化策略参数 θ

for 序列 $e=1\to E$ **do**:

用当前策略 π_θ 采样轨迹 $\{s_1,a_1,r_1,s_2,a_2,r_2,\cdots,s_T,a_T,r_T\}$

计算当前轨迹每个时刻 t 往后的回报 $\sum_{t'=t}^{T}\gamma^{t'-t}r_{t'}$，记为 ψ_t



对θ进行更新，$\theta=\theta+\alpha\sum_{t}^{T}\psi_t\nabla_\theta\log\pi_\theta(a_t|s_t)$

end for

这便是 REINFORCE 算法的全部流程了。接下来让我们用代码来实现它，看看效果如何吧！

不足：

1. 需要完整的片段才能执行单个训练步骤
2. 高梯度方差（改进：可以将价值权重减去一个基线，通常为V函数）
3. 探索（改进：在损失函数中减去熵$H(\pi)=-\sum\pi(a|s)log\pi(a|s)$，以惩罚智能体过于确定要采取的动作）
4. 样本的相关性（改进：考虑同多个环境交互）

DDPG讲一下

深度确定性策略梯度（DDPG），也是一种actor-critic方法，用于处理连续动作问题，其直接输出动作值而非动作概率分布，

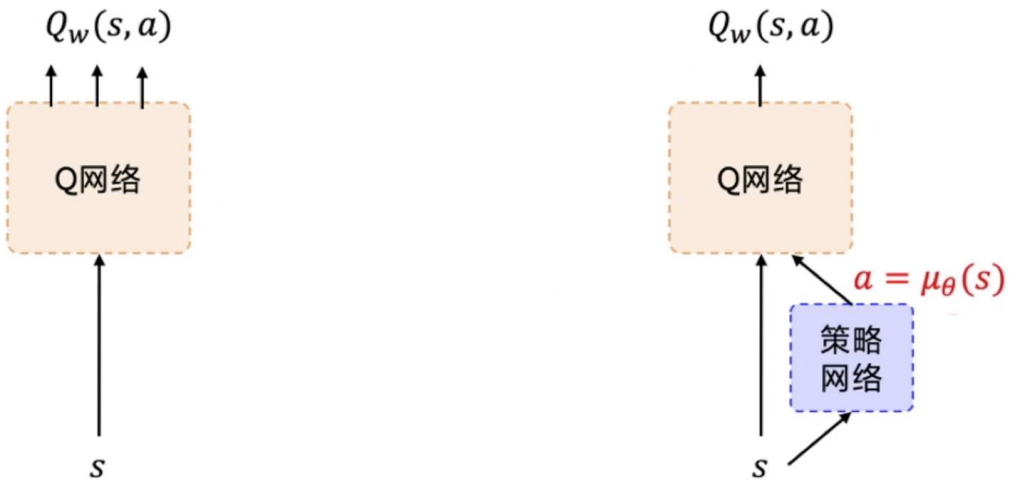


图 12.5　从深度 Q 网络到 DDPG

与A2C的对比：A2C中策略是随机的，无法对随机采样的步骤进行微分，这阻碍了反向传播。DDPG中策略是确定性的，因此可以根据Q来计算梯度（Q是从critic神经网络中获取），可以通过SGD进行端到端的优化。

架构：DDPG有4个网络：actor当前网络和actor目标网络，critic当前和critic目标

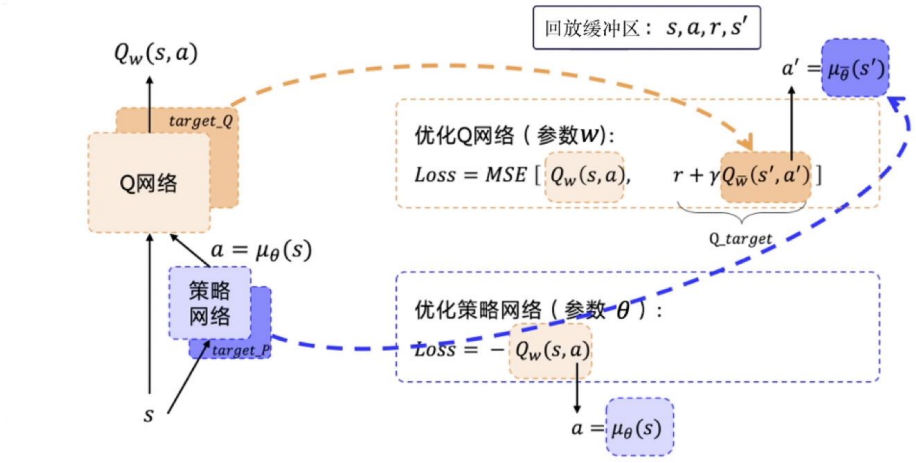


图 12.8　目标网络和经验回放

现在策略是确定性的，要对探索环境进行改进：当actor输出动作，在该动作作用于环境前，添加噪声：$a + \epsilon N$，其中ϵ为高斯噪声的标准差，N为长度为动作空间大小的随机数

关于TD3

双延迟深度确定性策略梯度（twin delayed DDPG）

解决DDPG的问题：对于超参和其他类型的调整很敏感，如高估的Q值

流程：

1. 学习两个Q函数（twin），其中最小的值会被作为Q-target

$$y(r, s^{'}, d) = r + \gamma(1-d)\min_{i=1,2} Q_{\theta_i, targ}(s^{'}, a_{TD3}(s^{'}))$$

2. 以较低的频率更新actor网络，以较高的频率更新critic网络
2. 在目标动作中加入噪声，通过平滑Q沿动作的变化，使策略更难利用Q函数的误差

$$a_{TD3}(s^{'}) = clip(\mu_{\theta,targ}(s^{'}) + clip(\epsilon, -c, c), a_{low}, a_{high})$$

Actor-Critic

Actor输出策略，采用策略梯度进行更新

Critic输出Q用于策略梯度的计算，采用时序差分残差的损失函数

△ 回顾策略梯度，求梯度形式如下：

$$\nabla R = E\left[\sum_{t=0}^{T} \psi_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

其中 ψ_t 可以有如下形式：

累积奖励，无偏，但方差大

1. $\sum_{t'=0}^{T} \gamma^{t'} r_{t'}$ ：轨迹的总回报
2. $\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ ：动作 a_t 之后的回报
3. $\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} - b(s_t)$ 基准线版本的改进

方差小，但有偏差

4. $Q_{\pi_\theta}(s_t, a_t)$ 动作价值函数
5. $A_{\pi_\theta}(s_t, a_t)$ 优势函数
6. ☆ $r_t + \gamma V_{\pi_\theta}(s_{t+1}) - V_{\pi_\theta}(s_t)$ 时序差分残差

△ Actor 采用策略梯度更新

Critic 采用如下时序差分残差的损失函数：

$$L(w) = \frac{1}{2}\left(r + \gamma V_w(s_{t+1}) - V_w(s_t)\right)^2$$

$V_w(s_{t+1})$ 不参与梯度运算

$$\therefore \nabla_w L(w) = -\left(r + \gamma V_w(s_{t+1}) - V_w(s_t)\right) \nabla_w V_w(s_t)$$

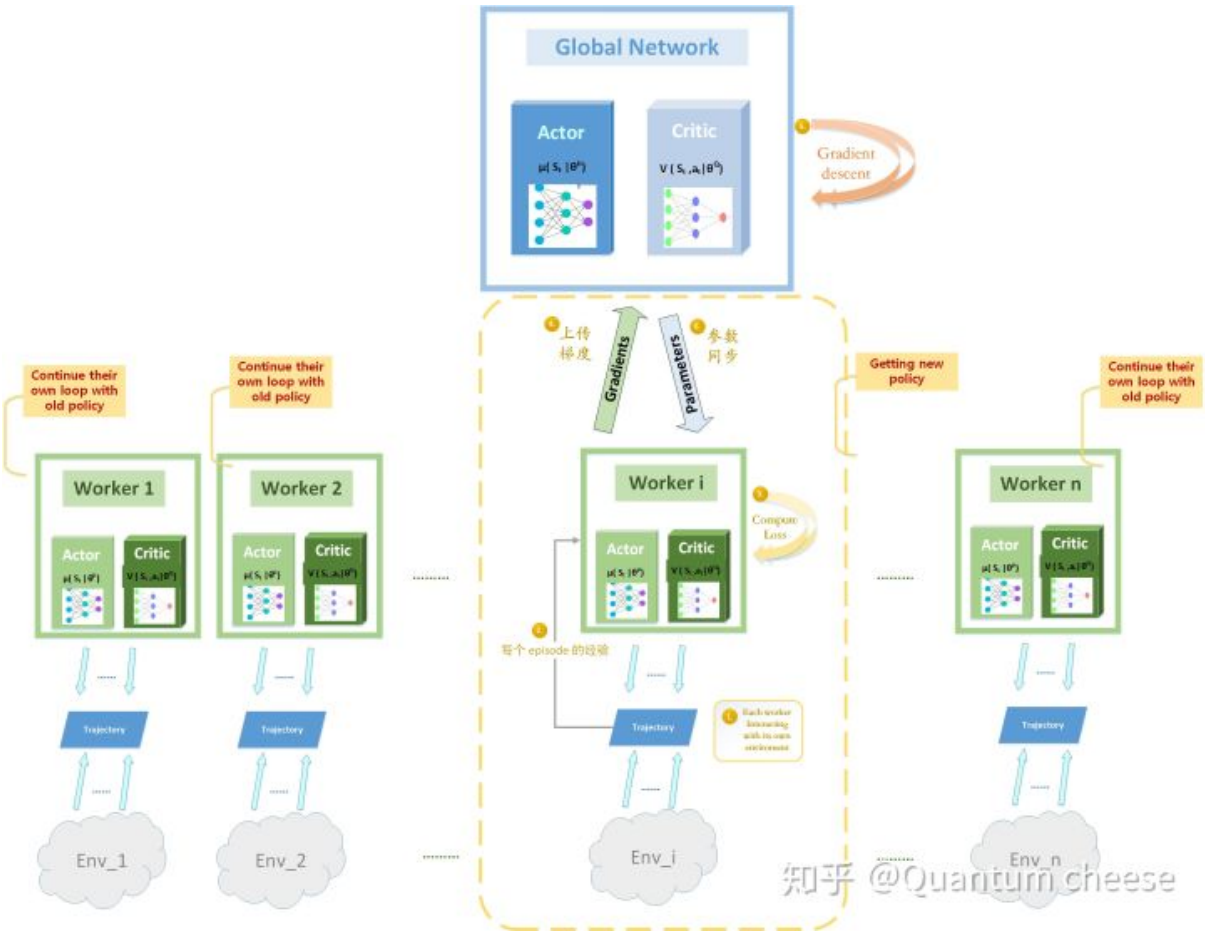
使用梯度下降法，最小化 $L(w)$

A2C和A3C讲一下

基础Actor-Critic中策略偏导的权重采用优势函数A(s,a)即为A2C。在具体实现时，Critic不用同时输出Q和V，只需输出V即可，优势函数可用近似计算即可：

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

A3C就是A2C的并行版本，其架构如下：



- 每个 worker 就是一个并行的独立进程，当某个 worker_i 当前 episode 并利用独立的经验计算出相应的损失函数后，会把梯度传递给 Global network，然后 Global network 进行梯度下降更新参数，并把最新的网络参数传递回对应的那个 worker_i ，这样 worker_i 就跟主网络同步了；
- 当 worker_i 跟主网络同步后，它会使用更新后的策略与环境交互。但此时其它进程中的 workers 并没有跟主网络同步，它们还是使用上次更新后的策略在运行，直到跑完当前 episode 才能跟主网络同步；
- 这意味在特定时刻，每个 worker 都不是同步的，很有可能各个 worker 都分别使用着一套不同的策略 ，独立的跟自己的环境交互。而主网络保持着最新的策略，各 worker 跟主网络同步的时间也是不一样的，只要有一个 worker 完成当前

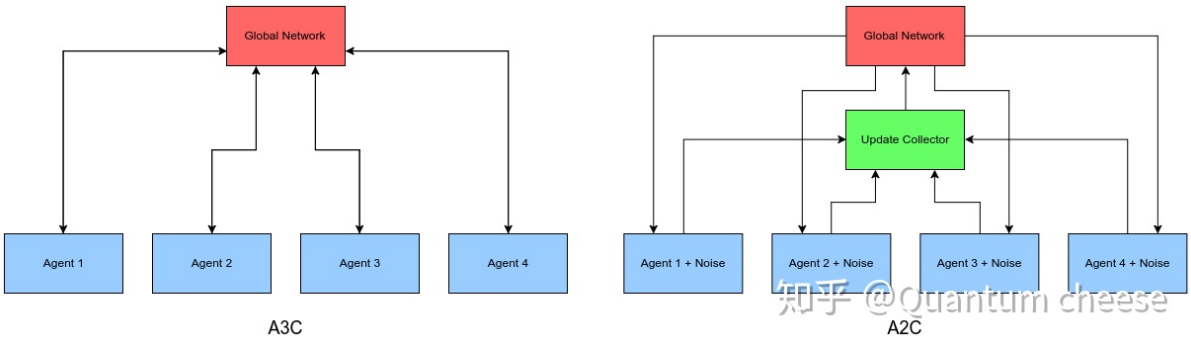
episode，主网络就会根据它的梯度进行更新，并不影响其它仍旧在使用旧策略的worker。这就是异步并行的核心思想。

实际上A2C也可以是多个work，只不过这些work是同步的。

即每轮训练中，Global network 都会等待每个 worker 各自完成当前的 episode，然后把这些 worker 上传的梯度进行汇总并求平均，得到一个统一的梯度并用其更新主网络的参数，最后用这个参数同时更新所有的 worker。 相当于在 A3C 基础上加入了一个同步的环节。

A2C 跟 A3C 的一个显著区别就是，在任何时刻，不同 worker 使用的其实是同一套策略，它们是完全同步的，更新的时机也是同步的。由于各 worker彼此相同，其实 A2C 就相当于只有两个网络，其中一个 Global network 负责参数更新，另一个负责跟环境交互收集经验，只不过它利用了并行的多个环境，可以收集到去耦合的多组独立经验。

对比：



PPO和TRPO讲一下

SAC讲一下

强化学习on-policy 、off-policy以及policy-based 和value-based的区别以及分别有哪些算法

MC和TD 方法的区别

对做的项目进行深挖 就强化学习的动作、状态以及奖励如何定义的，指标有哪些，包括状态和动作的维度是多少，那些算法效果比较好

如何选择DRL算法