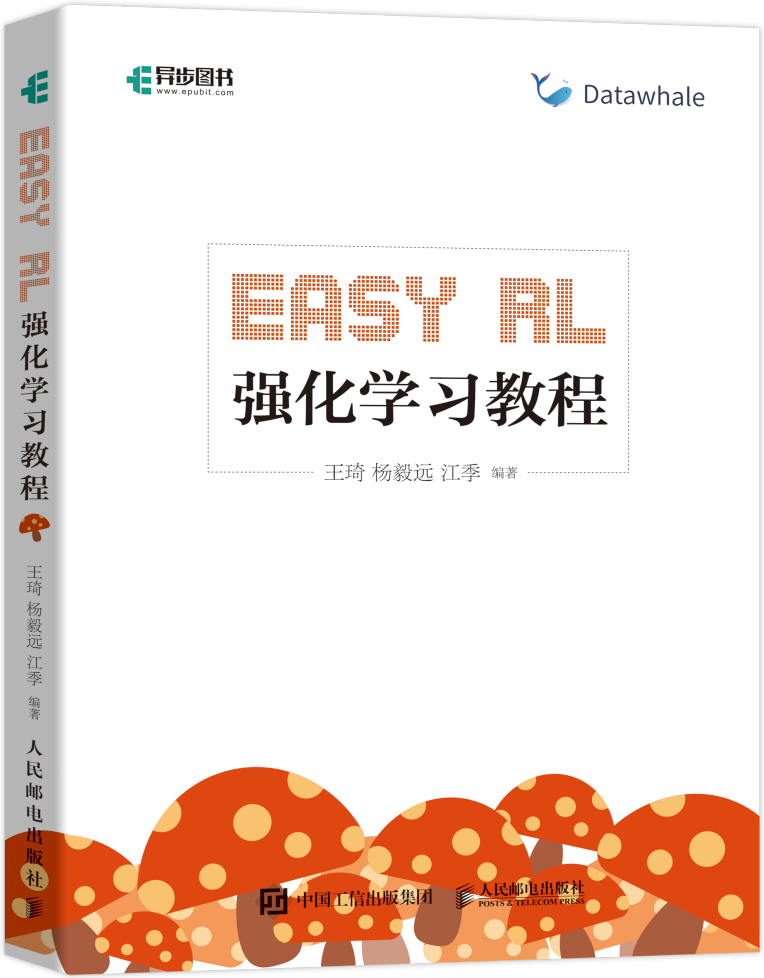
《Easy RL》面试题汇总

作者：凯鲁嘎吉 - 博客园 http://www.cnblogs.com/kailugaji/

本博客汇总了蘑菇书《Easy RL》强化学习中文教程涉及的面试题及答案(v.1.0.3)。强化学习介绍可参看：强化学习(Reinforcement Learning) - 凯鲁嘎吉 - 博客园。更多强化学习内容，请看：随笔分类 - Reinforcement Learning。



- **高冷的面试官**: 看来你对于强化学习还是有一定了解的,那么可以用一句话谈一下你对于强化学习的认识吗?

答: 强化学习包含环境、动作和奖励三部分，其本质是智能体通过与环境的交互，使其做出的动作对应的决策得到的总奖励最大，或者说是期望最大。

- **高冷的面试官**: 你认为强化学习与监督学习和无监督学习有什么区别?

答:

首先，强化学习和无监督学习是不需要有标签样本的，而监督学习需要许多有标签样本来进行模型的构建和训练。

其次，对于强化学习与无监督学习，无监督学习直接基于给定的数据进行建模，寻找数据或特征中隐藏的结构，一般对应聚类问题；强化学习需要通过延迟奖励学习策略来得到模型与目标的距离，这个距离可以通过奖励函数进行定量判断，这里我们可以将奖励函数视为正确目标的一个稀疏、延迟形式。

另外，强化学习处理的多是序列数据，样本之间通常具有强相关性，但其很难像监督学习的样本一样满足独立同分布条件。

- **高冷的面试官**: 根据你上面介绍的内容,你认为强化学习的使用场景有哪些呢?

答: 7个字总结就是"多序列决策问题"，或者说是对应的模型未知，需要通过学习逐渐逼近真实模型的问题。并且当前的动作会影响环境的状态，即具有马尔可夫性的问题。同时应满足所有状态是可重复到达的条件，即满足可学习条件。

- **高冷的面试官**: 强化学习中所谓的损失函数与深度学习中的损失函数有什么区别呀?

答:

深度学习中的损失函数的目的是使预测值和真实值之间的差距尽可能小;

强化学习中的损失函数的目的是使总奖励的期望尽可能大。

- **高冷的面试官**: 你了解model-free和model-based吗?两者有什么区别呢?

答: 两者的区别主要在于是否需要对于真实的环境进行建模。

model-free不需要对环境进行建模，直接与真实环境进行交互即可，所以其通常需要较多的数据或者采样工作来优化策略，这也使其对于真实环境具有更好的泛化性能；

model-based需要对环境进行建模，同时在真实环境与虚拟环境中进行学习，如果建模的环境与真实环境的差异较大，那么会限制其泛化性能。

- **高冷的面试官**: 请问马尔可夫过程是什么?马尔可夫决策过程又是什么?其中马尔可夫最重要的性质是什么呢?

答: 马尔可夫过程是一个二元组 $ <S,P> $，$S$为状态的集合，$P$为状态转移概率矩阵；

马尔可夫决策过程是一个五元组 $ <S,P,A,R,\gamma> $，其中 R 表示为从 S 到 S' 能够获得的奖励期望，γ为折扣因子，A为动作集合；

马尔可夫最重要的性质是无后效性，即下一个状态只与当前状态有关，与之前的状态无关，也就是 $P[S_{t+1} | S_t] = P[S_{t+1}|S_1,S_2,...,S_t]$。

- **高冷的面试官**: 请问我们一般怎么求解马尔可夫决策过程?

答: 我们求解马尔可夫决策过程时，可以直接求解贝尔曼方程或动态规划方程，即\[V(s)=R(S)+ \gamma \sum_{s' \in S}P(s'|s)V(s')\]

特别地，其矩阵形式为：$V=R+\gamma PV$。但是贝尔曼方程很难求解且计算复杂度较高，所以可以使用动态规划、蒙特卡洛、时间差分等方法求解。

- **高冷的面试官**: 请问如果数据流不满足马尔科夫性怎么办？应该如何处理?

答: 如果不具备马尔可夫性，即下一个状态与之前的状态也有关，若仅用当前的状态来求解决策过程，势必导致决策的泛化能力变差。为了解决这个问题，可以利用循环神经网络(RNN)对历史信息建模，获得包含历史信息的状态表征，表征过程也可以使用注意力机制等手段，最后在表征状态空间求解马尔可夫决策过程问题。

- **高冷的面试官**: 请分别写出基于状态值函数的贝尔曼方程以及基于状态-动作值函数的贝尔曼方程。

答:

1) 基于状态值函数的贝尔曼方程：${{V}^{\pi }}(s)={{\mathbb{E}}_{a \sim \pi (a|s)}}{{\mathbb{E}}_{s'\sim p(s'|s,a)}}[r(s,a,s')+\gamma {{V}^{\pi }}(s')]$

2) 基于状态-动作值函数(Q函数)的贝尔曼方程：${{Q}^{\pi }}(s,a)={{\mathbb{E}}_{s'\sim p(s'|s,a)}}\left[r(s,a,s')+\gamma {{\mathbb{E}}_{a'\sim \pi (a'|s')}}[{{Q}^{\pi }}(s',a')] \right]$

公式参看：https://www.cnblogs.com/kailugaji/p/15354491.html#_label3_0_0_6

- **高冷的面试官**: 请问最佳价值函数(optimal value function) V^* 和最佳策略(optimal policy) π^* 为什么等价呢?

答: 最佳价值函数的定义为： $V^* (s)=\max_{\pi} V^{\pi}(s)$，即我们去搜索一种策略$\pi$来让每个状态的价值最大。$V^*$就是到达每一个状态，它的值的极大化情况。在这种极大化情况上面，我们得到的策略就可以说它是最佳策略(optimal policy)，即$ \pi^{*}(s)=\underset{\pi}{\arg \max }~ V^{\pi}(s) $。最佳策略使得每个状态的价值函数都取得最大值。所以如果我们可以得到一个最佳价值函数，就可以说某一个马尔科夫决策过程的环境被解。在这种情况下，它的最佳的价值函数是一致的，即其达到的上限的值是一致的，但这里可能有多个最佳策略对应于相同的最佳价值。

- **高冷的面试官**：能不能手写一下第n步的值函数更新公式呀？另外，当n越来越大时，值函数的期望和方差分别变大还是变小呢?

答：n越大，方差越大，期望偏差越小。值函数的更新公式如下：\[Q\left(S, A\right) \leftarrow Q\left(S, A\right)+\alpha\left[\sum_{i=1}^{n} \gamma^{i-1} R_{t+i}+\gamma^{n} \max _{a} Q\left(S',a\right)-Q\left(S, A\right)\right]\]

- **高冷的面试官**：同学，你能否简述on-policy(同策略)和off-policy(异策略)的区别?

答： off-policy和on-policy的根本区别在于生成样本的策略和网络参数更新时的策略是否相同。

对于on-policy，行为策略和要优化的策略是同一个策略，更新了策略后，就用该策略的最新版本对数据进行采样。

例如，SARAS是基于当前的策略直接执行一次动作选择，然后用动作和对应的状态更新当前的策略，因此生成样本的策略和学习时的策略相同，所以SARAS算法为on-policy算法。该算法会遭遇探索-利用窘境，仅利用目前已知的最优选择，可能学不到最优解，不能收敛到局部最优，而加入探索又降低了学习效率。$\epsilon $ - 贪心算法是这种矛盾下的折衷，其优点是直接了当、速度快，缺点是不一定能够找到最优策略。

对于off-policy，生成样本的策略与网络更新参数时使用的策略不同。其使用任意的一个行为策略来对数据进行采样，并利用其更新目标策略。

例如，Q-learning在计算下一状态的预期奖励时使用了最大化操作，直接选择最优动作，而当前策略并不一定能选择到最优的动作，因此这里生成样本的策略和学习时的策略不同，所以Q-learning为off-policy算法。

- **高冷的面试官**：小同学，能否讲一下Q-Learning，最好可以写出其 $Q(s,a)$ 的更新公式。另外，它是on-policy还是off-policy，为什么?

答： Q-learning是通过计算最优动作值函数来求策略的一种时序差分的学习方法，其更新公式为：\[Q(s,a)\leftarrow Q(s,a)+\alpha [r(s,a)+\underset{{{a}'}}{\mathop{\gamma \max }}\,Q({s}',{a}')-Q(s,a)]\]

其是off-policy的，由于Q更新使用了下一个时刻的最大值，所以我们只关心哪个动作使得 $Q(s_{t+1}, a)$ 取得最大值，而实际到底采取了哪个动作(行为策略)，Q-learning并不关心。这表明优化策略并没有用到行为策略的数据，所以说它是off-policy的。

- **高冷的面试官**：小朋友，能否讲一下SARSA，最好可以写出其Q(s,a)的更新公式。另外，它是on-policy还是off-policy，为什么？

答：SARSA可以算是Q-learning的改进（这句话出自「神经网络与深度学习」的第 342 页）（可参考SARSA「on-line q-learning using connectionist systems」的 abstract 部分），其更新公式为：\[Q(s, a) \larr Q(s, a) + \alpha [r(s,a) + \gamma Q(s', a') - Q(s, a)]\]

其为on-policy的，SARSA必须执行两次动作得到$(s,a,r,s',a') $才可以更新一次；而且$a'$是在特定策略$\pi$的指导下执行的动作，因此估计出来的$Q(s,a)$是在该策略 π之下的Q值，样本生成用的π和估计的π是同一个，因此是on-policy。

- **高冷的面试官**：请问value-based和policy-based方法的区别是什么？

答：

1) 生成策略上的差异，前者确定，后者随机。基于价值的方法中动作-价值对的估计值最终会收敛（通常是不同的数，可以转化为0～1 的概率），因此通常会获得一个确定的策略；基于策略的方法不会收敛到一个确定的值，另外他们会趋向于生成最佳随机策略。如果最佳策略是确定的，那么最优动作对应的值函数的值将远大于次优动作对应的值函数的值，值函数的大小代表概率的大小。

2) 动作空间是否连续，前者离散，后者连续。基于价值的方法，对于连续动作空间问题，虽然可以将动作空间离散化处理，但离散间距的选取不易确定。过大的离散间距会导致算法取不到最优动作，会在最优动作附近徘徊；过小的离散间距会使得动作的维度增大，会和高维度动作空间一样导致维度灾难，影响算法的速度。而基于策略的方法适用于连续的动作空间，在连续的动作空间中，可以不用计算每个动作的概率，而是通过正态分布选择动作。

3) 基于价值的方法，例如Q学习算法，是通过求解最优价值函数而间接地求解最优策略；基于策略的方法，例如REINFORCE等算法直接将策略参数化，通过策略搜索、策略梯度或者进化方法来更新参数以最大化回报。基于价值的方法不易扩展到连续动作空间，并且当同时采用非线性近似、自举等策略时会有收敛问题。策略梯度具有良好的收敛性。

4) 另外，对于价值迭代和策略迭代，策略迭代有两个循环，一个是在策略估计的时候，为了求当前策略的价值函数需要迭代很多次；另一个是外面的大循环，即策略评估、策略提升。价值迭代算法则是一步到位，直接估计最优价值函数，因此没有策略提升环节。

更多关于两者的区别，请参看：https://www.cnblogs.com/kailugaji/p/15354491.html#_label3_0_0_7

- **高冷的面试官**：请简述以下时序差分(Temporal Difference，TD)算法。

答：时序差分算法是使用广义策略迭代来更新Q函数的方法，核心使用了自举（bootstrapping），即值函数的更新使用了下一个状态的值函数来估计当前状态的值。也就是使用下一步的 Q 值 $Q(S_{t+1},A_{t+1})$ 来更新我这一步的 Q 值 $Q(S_t,A_t) $。完整的计算公式如下：

\[Q(S_t,A_t) \larr Q(S_t,A_t) + \alpha [R_{t+1}+\gamma Q(S_{t+1},A_{t+1})-Q(S_t,A_t)]\]

- **高冷的面试官**：请问蒙特卡洛方法（Monte Carlo Algorithm，MC）和时序差分(Temporal Difference，TD)算法是无偏估计吗？另外谁的方差更大呢？为什么呢？

答：蒙特卡洛方法是无偏估计，时序差分方法是有偏估计；蒙特卡洛方法的方差较大，时序差分方法的方差较小，原因在于时序差分方法中使用了自举，实现了基于平滑的效果，导致估计的价值函数的方差更小。

- **高冷的面试官**：能否简单说下动态规划、蒙特卡洛和时序差分的异同点？

答：

相同点：都用于进行价值函数的描述与更新，并且所有方法都基于对未来事件的展望计算一个回溯值。

不同点：蒙特卡洛方法和时序差分方法属于model-free方法，而动态规划属于model-based方法；时序差分方法和蒙特卡洛方法，因为都是model-free的方法，所以对于后续状态的获知也都是基于试验的方法；时序差分方法和动态规划方法的策略评估，都能基于当前状态的下一步预测情况来得到对于当前状态的价值函数的更新。

另外，时序差分方法不需要等到试验结束后才能进行当前状态的价值函数的计算与更新，而蒙特卡洛方法需要与环境交互，产生一整条马尔可夫链并直到最终状态才能进行更新。时序差分方法和动态规划方法的策略评估不同之处为model-free和model-based，动态规划方法可以凭借已知转移概率推断出后续的状态情况，而时序差分方法借助试验才能知道。

蒙特卡洛方法和时序差分方法的不同在于，蒙特卡洛方法进行了完整的采样来获取长期的回报值，因而在价值估计上会有更小的偏差，但是也正因为收集了完整的信息，所以价值的方差会更大，原因在于其基于试验的采样得到，和真实的分布有差距，不充足的交互导致较大方差。而时序差分方法则相反，因为它只考虑了前一步的回报值，其他都是基于之前的估计值，因而其价值估计相对来说具有偏差大方差小的特点。

- 三者的联系：对于$TD(\lambda)$方法，如果 $ \lambda = 0$ ，那么此时等价于时序差分方法，即只考虑下一个状态；如果$ \lambda = 1$，等价于蒙特卡洛方法，即考虑 $T-1$ 个后续状态直到整个试验结束。

- **高冷的面试官**：同学来吧，给我手工推导一下策略梯度公式的计算过程。

答：首先我们目的是最大化奖励函数，即调整 θ ，使得期望回报最大，可以用公式表示如下\[J(\theta)=E_{\tau \sim p_{\theta(\mathcal{T})}}[\sum_tr(s_t,a_t)]\]

对于上面的式子， τ 表示从从开始到结束的一条完整路径。通常，对于最大化问题，我们可以使用梯度上升算法来找到最大值，即\[\theta^* = \theta + \alpha\nabla J({\theta})\]

所以我们仅仅需要计算并更新$\nabla J({\theta})$ ，也就是计算奖励函数$J({\theta})$关于 θ的梯度，也就是策略梯度，计算方法如下：

\[\nabla_{\theta}J(\theta) = \int {\nabla}_{\theta}p_{\theta}(\tau)r(\tau)d \tau \]

\[= \int p_{\theta}{\nabla}_{\theta}logp_{\theta}(\tau)r(\tau)d \tau \]

\[= E_{\tau \sim p_{\theta}(\tau)}[{\nabla}_{\theta}logp_{\theta}(\tau)r(\tau)]\]

接着我们继续讲上式展开，对于 $p_{\theta}(\tau)$ ，即 $p_{\theta}(\tau|{\theta})$ ：\[p_{\theta}(\tau|{\theta}) = p(s_1)\prod_{t=1}^T \pi_{\theta}(a_t|s_t)p(s_{t+1}|s_t,a_t)\]

取对数后为：\[logp_{\theta}(\tau|{\theta}) = logp(s_1)+\sum_{t=1}^T log\pi_{\theta}(a_t|s_t)p(s_{t+1}|s_t,a_t)\]

继续求导：\[\nabla logp_{\theta}(\tau|{\theta}) = \sum_{t=1}^T \nabla_{\theta}log \pi_{\theta}(a_t|s_t)\]

带入第三个式子，可以将其化简为：

\[\nabla_{\theta}J(\theta) = E_{\tau \sim p_{\theta}(\tau)}[{\nabla}_{\theta}logp_{\theta}(\tau)r(\tau)] \]

\[= E_{\tau \sim p_{\theta}}[(\nabla_{\theta}log\pi_{\theta}(a_t|s_t))(\sum_{t=1}^Tr(s_t,a_t))] \]

\[= \frac{1}{N}\sum\limits_{i=1}^{N}{[(\sum\limits_{t=1}^{T}{{{\nabla }_{\theta }}\log {{\pi }_{\theta }}({{a}_{i,t}}|{{s}_{i,t}})})(\sum\limits_{t=1}^{T}{r({{s}_{i,t}},{{a}_{i,t}})})]}\]

推导可参看：https://www.cnblogs.com/kailugaji/p/15354491.html#_label3_0_2_0

- **高冷的面试官**：可以说一下你了解到的基于梯度策略的优化时的技巧吗？

答：

1) 增加基线(Add a baseline)：为了防止所有奖励都为正，从而导致每一个状态和动作的变换，都会使得每一个变换的概率上升，我们把奖励减去一项b，称b为基线。当减去b以后，就可以让奖励$R(\tau^n)-b$这一项， 有正有负。 所以如果得到的总奖励(total reward)$R(\tau^n)$大于b的话，就让它的概率上升。如果这个总奖励小于b，就算它是正的，正的很小也是不好的，就要让这一项的概率下降。 如果$R(\tau^n)<b$ ，就要让采取这个动作的奖励下降，这样也符合常理。但是使用基线会让本来奖励很大的"动作"的奖励变小，降低更新速率。

2) 指派合适的分数(Assign suitable credit)：首先，原始权重是整个回合的总奖励。现在改成从某个时间点t开始，假设这个动作是在时间点t被执行的，那么从时间点t，一直到游戏结束所有奖励的总和，才真的代表这个动作是好的还是不好的；接下来我们再进一步，把未来的奖励打一个折扣，这里我们称由此得到的奖励的和为Discounted Return(折扣回报) 。

3) 综合以上两种技巧，我们将其统称为优势函数，用A来代表优势函数。优势函数取决于状态和动作，即我们需计算的是在某一个状态s采取某一个动作a的时候，优势函数有多大。

- **高冷的面试官**：请问什么是重要性采样呀？

答：使用另外一种数据分布，来逼近所求分布的一种方法，算是一种期望修正的方法，公式是：

$$\int f(x) p(x) d x = \int f(x) \frac{p(x)}{q(x)} q(x) d x$$

$$= E_{x \sim q}[f(x){\frac{p(x)}{q(x)}}]$$

$$= E_{x \sim p}[f(x)]$$

我们在已知 q 的分布后，可以使用上述公式计算出从 p 分布的期望值。也就可以使用 q 来对于 p 进行采样了，即为重要性采样。

重要性采样可参看：https://www.cnblogs.com/kailugaji/p/15401383.html#_lab2_0_1

- **高冷的面试官**：请简述下PPO算法。其与TRPO算法有何关系呢？

答：近端策略优化算法(PPO)借鉴了信任区域策略优化算法(TRPO)，通过采用一阶优化，在采样效率、算法表现以及实现和调试的复杂度之间取得了新的平衡。这是因为近端策略优化算法会在每一次迭代中尝试计算新的策略，让损失函数最小化，并且保证每一次新计算出的策略能够和原策略相差不大。换句话说，其为在避免使用重要性采样时由于在 θ 下的 $p_{\theta}\left(a_{t} | s_{t}\right)$ 跟 在 θ' 下的 $ p_{\theta'}\left(a_{t} | s_{t}\right) $ 差太多，导致重要性采样结果偏差较大而采取的算法。

TRPO可参见：信赖域策略优化(Trust Region Policy Optimization, TRPO) - 凯鲁嘎吉 - 博客园

PPO可参见：近端策略优化算法(Proximal Policy Optimization Algorithms, PPO) - 凯鲁嘎吉 - 博客园

- **高冷的面试官**：请问DQN（Deep Q-Network）是什么？其两个关键性的技巧分别是什么？

答：深度Q网络(DQN)是基于深度学习的Q学习算法，其结合了价值函数近似(Value Function Approximation)与神经网络技术，并采用了目标网络(Target Network)和经验回放(Experience Replay)技巧进行网络的训练。

DQN详看：https://www.cnblogs.com/kailugaji/p/15354491.html#_label3_0_1_3

- **高冷的面试官**：接上题，DQN中的两个技巧：目标网络(target network)和经验回放(experience replay)的具体作用是什么呢？

答：

在DQN中某个动作值函数的更新依赖于其他动作值函数。如果我们一直更新值网络的参数，会导致更新目标不断变化，也就是我们在追逐一个不断变化的目标，这样势必会不太稳定。 为了解决在基于时序差分的网络的问题时，优化目标 $\mathrm{Q}^{\pi}\left(s_{t}, a_{t}\right) =r_{t}+\mathrm{Q}^{\pi}\left(s_{t+1}, \pi\left(s_{t+1}\right)\right)$ 左右两侧会同时变化使得训练过程不稳定，从而增大回归(regression)的难度。目标网络选择将上式的右部分即 $r_{t}+\mathrm{Q}^{\pi}\left(s_{t+1}, \pi\left(s_{t+1}\right)\right)$ 固定，通过改变上式左部分的网络的参数，进行回归。

对于经验回放，其会构建一个回放缓冲区(Replay Buffer, Replay Memory)，用来保存许多数据，每一个数据的形式如下：在某一个状态s_t，采取某一个动作a_t，得到了奖励r_t，然后跳到状态s_{t+1}。我们使用 π 去跟环境互动很多次，把收集到的数据都放到这个回放缓冲区中。当我们的回放缓冲区"装满"后，就会自动删去最早进入缓冲区的数据。在训练时，对于每一轮迭代都有相对应的批量（与我们训练普通网络一样，通过采样得到），然后用这个批量中的数据去更新Q函数。即Q函数在采样和训练的时候会用到过去的经验数据，也可以消除样本之间的相关性。

关于这两个问题的回复，可详看：https://www.cnblogs.com/kailugaji/p/15354491.html#_label3_0_1_3

- **高冷的面试官**：DQN（Deep Q-learning）和Q-learning有什么异同点？

答：整体来说，从名称就可以看出，两者的目标价值以及价值的更新方式基本相同，另外一方面，不同点在于：

1) 首先，DQN 将 Q-learning 与深度学习结合，用深度网络来近似动作价值函数，而 Q-learning 则是采用表格存储。

2) DQN 采用了我们前面所描述的经验回放（Experience Replay）训练方法，从历史数据中随机采样，而 Q-learning 直接采用下一个状态的数据进行学习。

- **高冷的面试官**：请问，随机性策略和确定性策略有什么区别吗？

答：随机策略表示为某个状态下动作取值的分布，确定性策略在每个状态只有一个确定的动作可以选。从熵的角度来说，确定性策略的熵为0，没有任何随机性。随机策略有利于我们进行适度的探索，确定性策略的探索问题更为严峻。

- **高冷的面试官**：请问不打破数据相关性，神经网络的训练效果为什么就不好？

答：在神经网络中通常使用随机梯度下降法。随机的意思是我们随机选择一些样本来增量式地估计梯度，比如常用的批量训练方法。如果样本是相关的，就意味着前后两个批量很可能也是相关的，那么估计的梯度也会呈现出某种相关性。但是在极端条件下，后面的梯度估计可能会抵消掉前面的梯度估计量，从而使得训练难以收敛。

- **高冷的面试官**：DQN都有哪些变种？

答：深度Q网络(DQN)有3 个经典的变种：双深度Q网络(Double DQN)、竞争深度Q网络(Dueling DQN)、优先级双深度Q网络(Prioritized Replay Buffer)。

1）双深度Q网络：将动作选择和价值估计分开，避免Q值被过高估计。

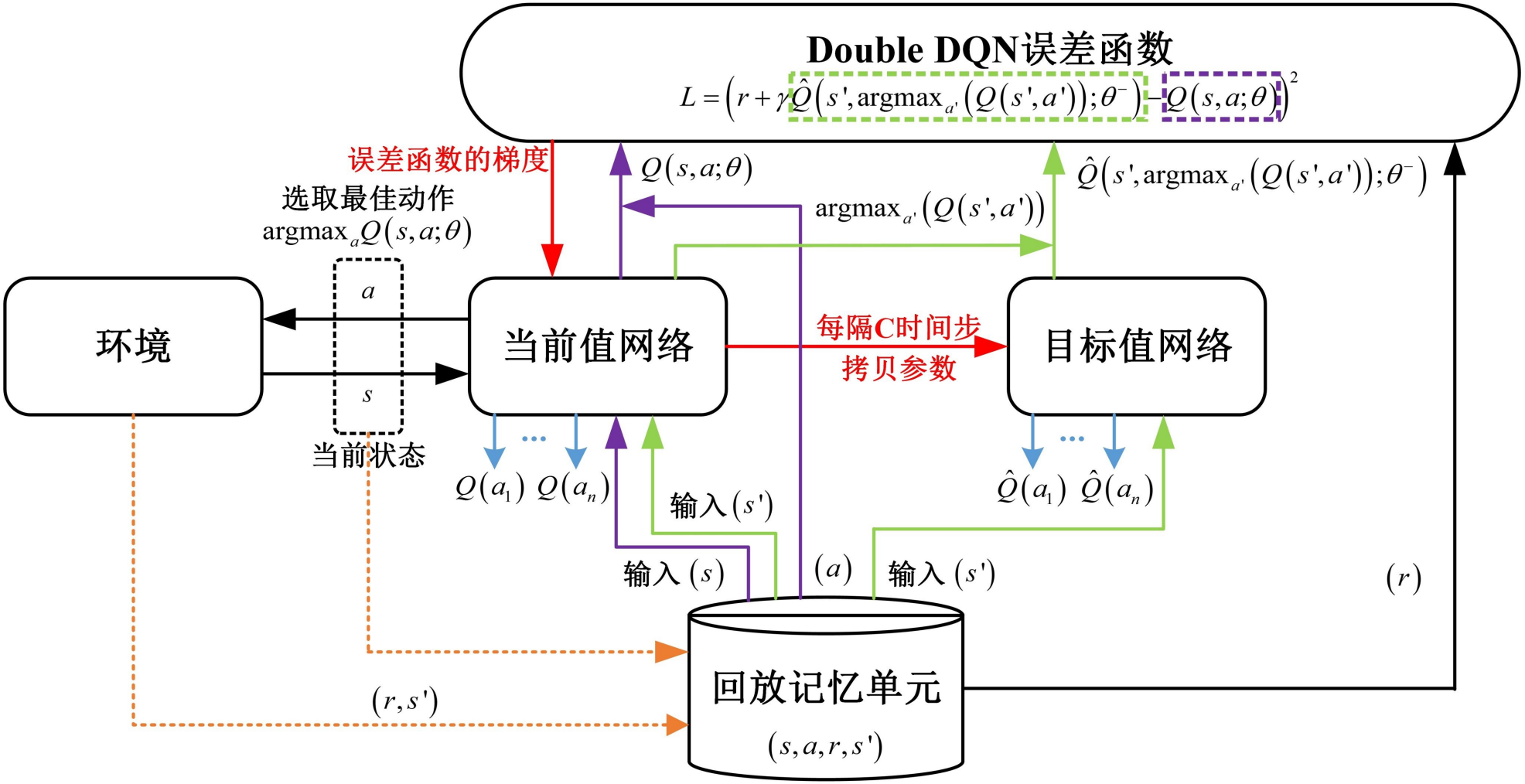
2）竞争深度Q网络：将Q值分解为状态价值和优势函数，得到更多有用信息。

3）优先级双深度Q网络：将经验池中的经验按照优先级进行采样。

- **高冷的面试官**：简述Double DQN(双深度Q网络)原理？

答：深度Q网络(DQN)由于总是选择当前最优的动作价值函数来更新当前的动作价值函数，因此存在过估计问题（估计的价值函数值大于真实的价值函数值）。为了解耦这两个过程，双深度Q网络使用两个价值网络，一个网络用来执行动作选择，然后用另一个网络的价值函数对应的动作值更

新当前网络。

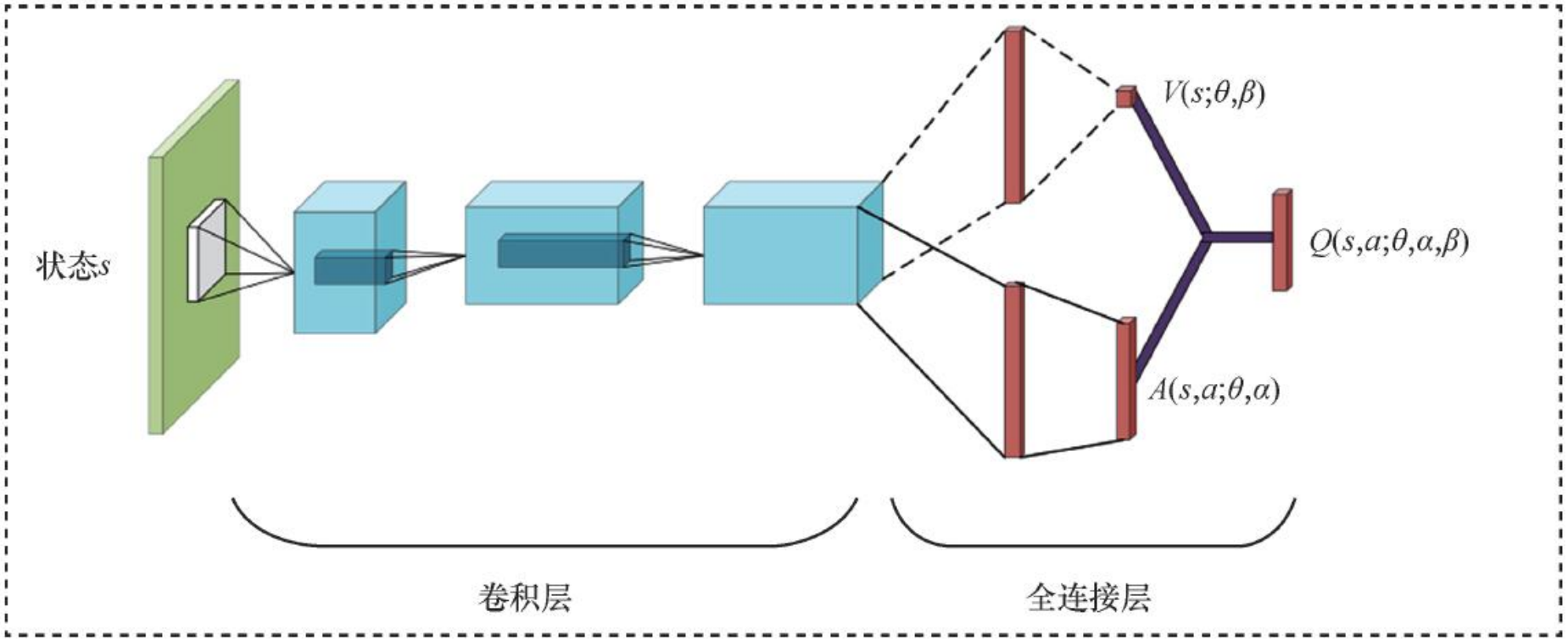


图源：https://zhuanlan.zhihu.com/p/79712897

- **高冷的面试官**：请问Dueling DQN(竞争深度Q网络)模型有什么优势呢？

答：对于我们的 $Q(s,a)$ 其对应的状态由于为表格的形式，所以是离散的，而实际中的状态大都不是离散的。对于 $Q(s,a)$ 的计算公式，$Q(s,a)=V(s)+A(s,a)$ 。其中的 $V(s)$ 是对于不同的状态都有值，$A(s,a)$ 对于不同的状态都有不同的动作对应的值。所以本质上来说，我们最终的矩阵 $Q(s,a)$ 的结果是将每一个 $V(s)$ 加到矩阵 $A(s,a)$ 中得到的。从模型的角度考虑，我们的网络直接改变的 $Q(s,a)$ 而是更改的

$V、A$ 。但是有时我们更新时不一定会将 $V(s)$ 和 $Q(s,a)$ 都更新。我们将其分成两个部分后，就不需要将所有的状态-动作对都采样一遍，我们可以使用更高效的估计Q值的方法将最终的 $Q(s,a)$计算出来。



- **高冷的面试官**：请问Actor - Critic有何优点呢?

答:

1) 相比以值函数为中心的算法，Actor - Critic应用了策略梯度的做法，这能让它在连续动作或者高维动作空间中选取合适的动作，而 Q-learning 做这件事会很困难甚至瘫痪。

2) 相比单纯策略梯度，Actor - Critic应用了Q-learning或其他策略评估的做法，使得Actor - Critic能进行单步更新而不是回合更新，比单纯的策略梯度的效率要高。

更多关于演员-评论员算法，请看：https://www.cnblogs.com/kailugaji/p/15354491.html#_lab2_0_3

- **高冷的面试官**：Actor-Critic两者的区别是什么?

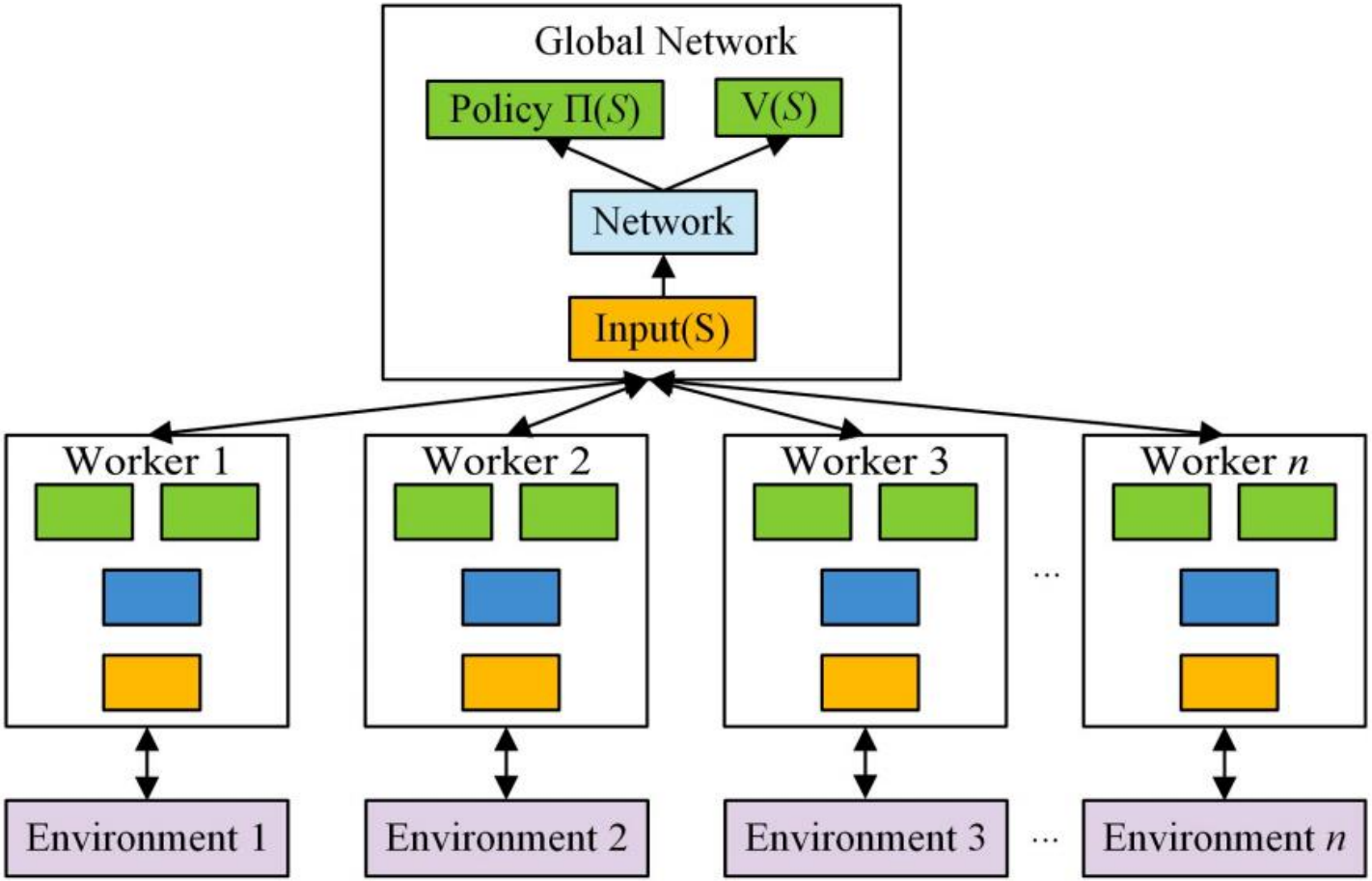
答：Actor是策略模块，输出动作；critic是判别器，用来计算值函数。

- **高冷的面试官**：actor-critic框架中的critic起了什么作用?

答：critic表示了对于当前决策好坏的衡量。结合策略模块，当critic判别某个动作的选择时有益的，策略就更新参数以增大该动作出现的概率，反之降低动作出现的概率。

- **高冷的面试官**：请简述一下A3C算法吧，另外A3C是on-policy还是off-policy呀?

答：A3C就是异步优势演员-评论家方法（Asynchronous Advantage Actor-Critic）：评论家学习值函数，同时有多个演员并行训练并且不时与全局参数同步。A3C旨在用于并行训练，是 on-policy 的方法。



- **高冷的面试官**：请问A3C算法具体是如何异步更新的?

答：下面是算法大纲：

- 定义全局参数 θ 和 w 以及特定线程参数 $θ'$ 和 w'。

- 初始化时间步 $t=1$。

- 当 $T<=T_{max}$：

- 重置梯度：$dθ=0$ 并且 $dw=0$。

- 将特定于线程的参数与全局参数同步：$θ'=θ$ 以及 $w'=w$。

- 令 $t_{start} =t$ 并且随机采样一个初始状态 s_t。

- 当 （$s_t!=$ 终止状态）并$t-t_{start}<=t_{max}$：

- 根据当前线程的策略选择当前执行的动作 $a_t~π_{θ'}(a_t|s_t)$，执行动作后接收回报r_t然后转移到下一个状态s_{t+1}。

- 更新 t 以及 T：$t=t+1$ 并且 $T=T+1$。

- 初始化保存累积回报估计值的变量

- 对于 $i=t_1,...,t_{start}$：

- $r←\gamma r+{r}_{i}$；这里 r 是 G_{i} 的蒙特卡洛估计。

- 累积关于参数 θ'的梯度：$dθ←dθ+∇θ'logπθ'({a}_{i}|{s}_{i})(r-Vw'({s}_{i}))$;

- 累积关于参数 w' 的梯度：$dw←dw+2(r-Vw'({s}_{i}))∇w'(r-Vw'({s}_{i}))$.

- 分别使用 dθ以及 dw异步更新 θ以及 w。

- **高冷的面试官**：简述A3C的优势函数?

答：$A(s,a)=Q(s,a)-V(s)$是为了解决基于价值方法具有高变异性。它代表着与该状态下采取的平均行动相比所取得的进步。

- 如果 A(s,a)>0: 梯度被推向了该方向

- 如果 A(s,a)<0: (我们的action比该状态下的平均值还差) 梯度被推向了反方

但是这样就需要两套价值函数，所以可以使用时序差分方法做估计：$A(s,a)=r+\gamma V(s')-V(s)$。

- **高冷的面试官**：请简述一下DDPG算法?

答：深度确定性策略梯度算法(Deep Deterministic Policy Gradient，简称 DDPG)使用演员-评论员(Actor - Critic)结构，但是输出的不是动作的概率，而是具体动作，其可以用于连续动作的预测。优化的目的是将深度Q 网络(DQN)扩展到连续的动作空间。另外，其含义如其名：

1）深度(Deep)是因为用了深度神经网络；

2）确定性(Deterministic)表示其输出的是一个确定的动作，可以用于连续动作的环境；

3）策略梯度(Policy Gradient)代表的是它用到的是策略网络。REINFORCE算法每个回合(episode)就会更新一次网络，但是深度确定性策略梯度算法每个步骤(step)都会更新一次策略网络，它是一个单步更新的策略网络。

- **高冷的面试官**：你好，请问DDPG是on-policy还是off-policy，原因是什么呀？

答：异策略(off-policy)算法。

1）深度确定性策略梯度算法(DDPG)是优化的深度Q网络(DQN)，其使用了经验回放，所以为异策略算法。

2）因为深度确定性策略梯度算法(DDPG)为了保证一定的探索，对输出动作加了一定的噪声，行为策略不再是优化的策略。

- **高冷的面试官**：你是否了解过D4PG算法呢？描述一下吧。

答：分布的分布式深度确定性策略梯度算法（distributed distributional deep deterministic policy gradient，D4PG），相对于深度确定性策略梯度算法，其优化部分如下。

1）分布式评论员(critic)：不再只估计Q值的期望值，而是估计期望Q值的分布，即将期望Q值作为一个随机变量来估计。

2）N 步累计回报：计算时序差分误差时，D4PG 计算的是N步的时序差分目标值而不仅仅只有一步，这样就可以考虑未来更多步骤的回报。

3）多个分布式并行演员(actor)：D4PG 使用K个独立的演员并行收集训练数据并存储到同一个回放缓冲区中。

4）优先经验回放（prioritized experience replay，PER）：使用一个非均匀概率π从回放缓冲区中进行数据采样。

参考：Qi Wang, Yiyuan Yang, Ji Jiang, Easy RL 强化学习中文教程, 2021. https://github.com/datawhalechina/easy-rl/releases