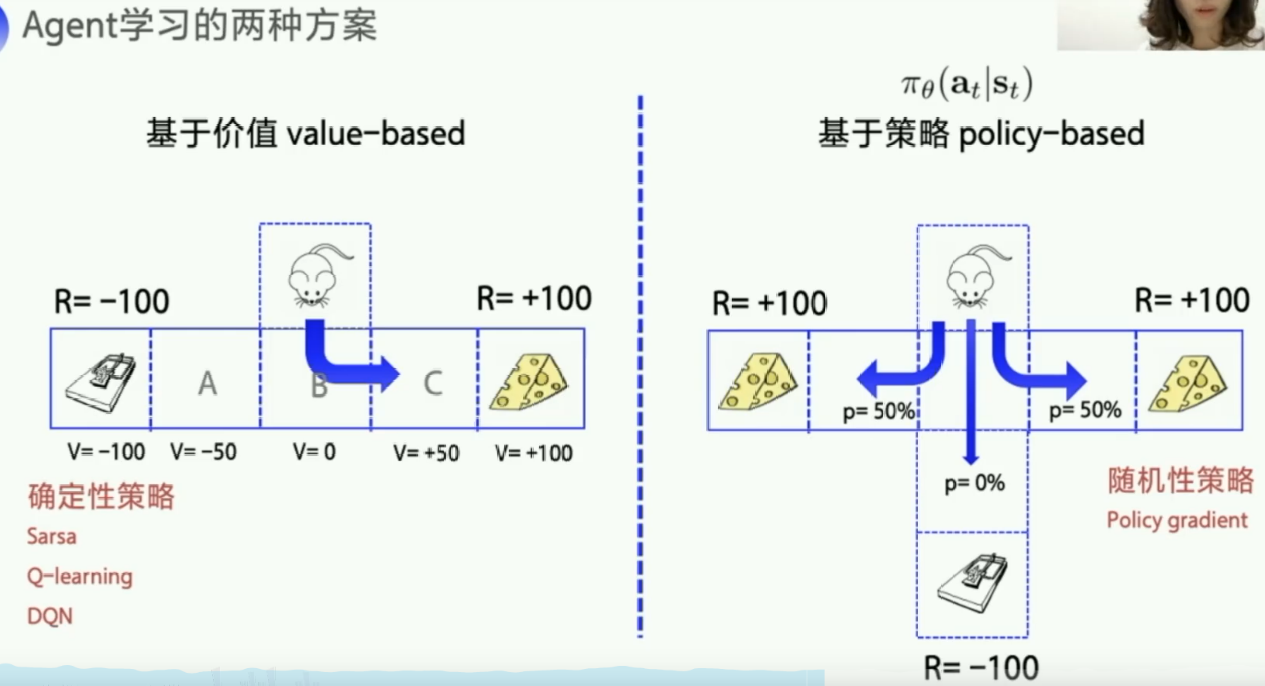
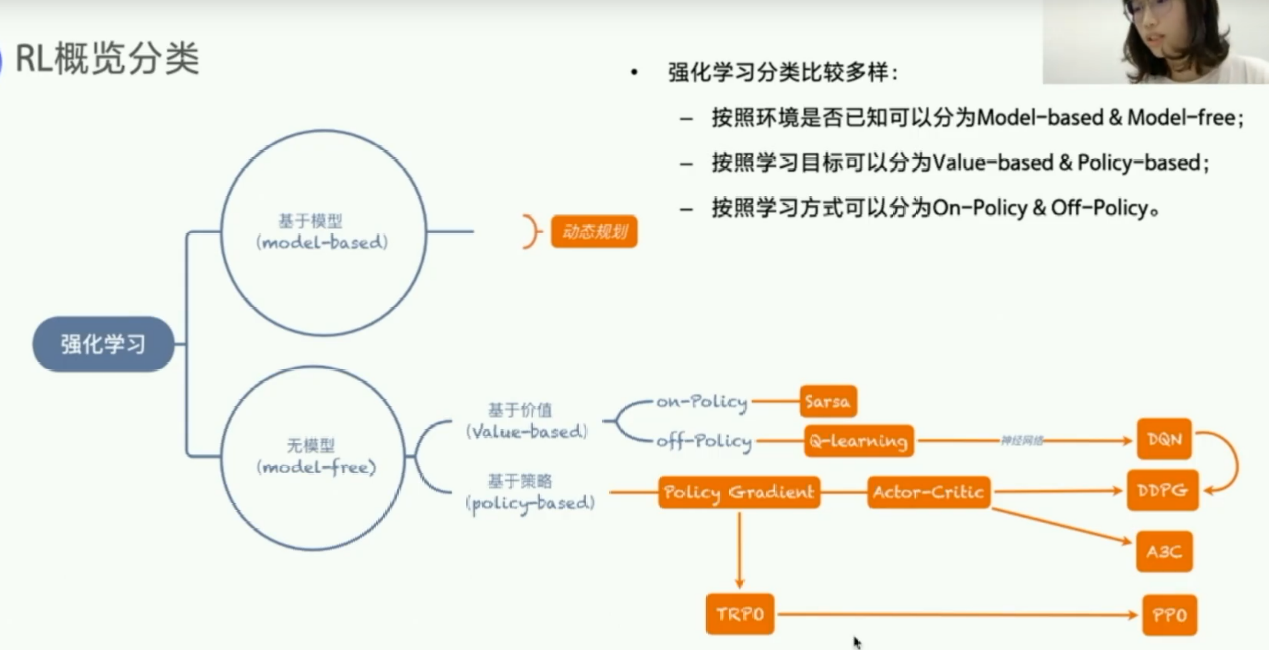


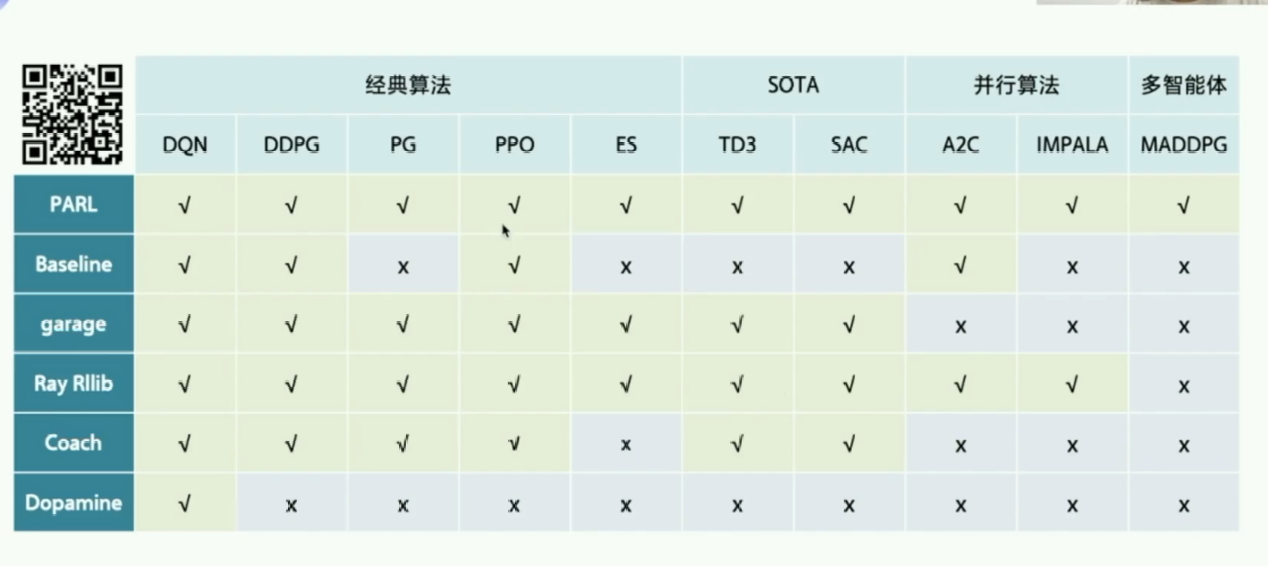
监督学习有样本，也有结果，用结果来校验训练精度，相当于向老师学习，老师会的我才会，而强化学习是从自己的经验中学习，自己去尝试，然后更具奖励去学习；

非监督学习的话，

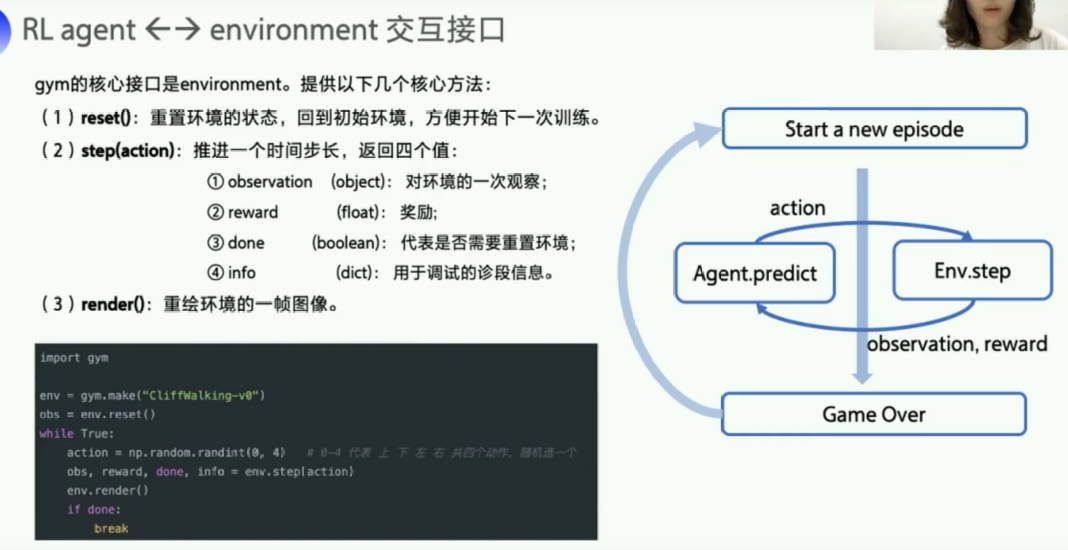


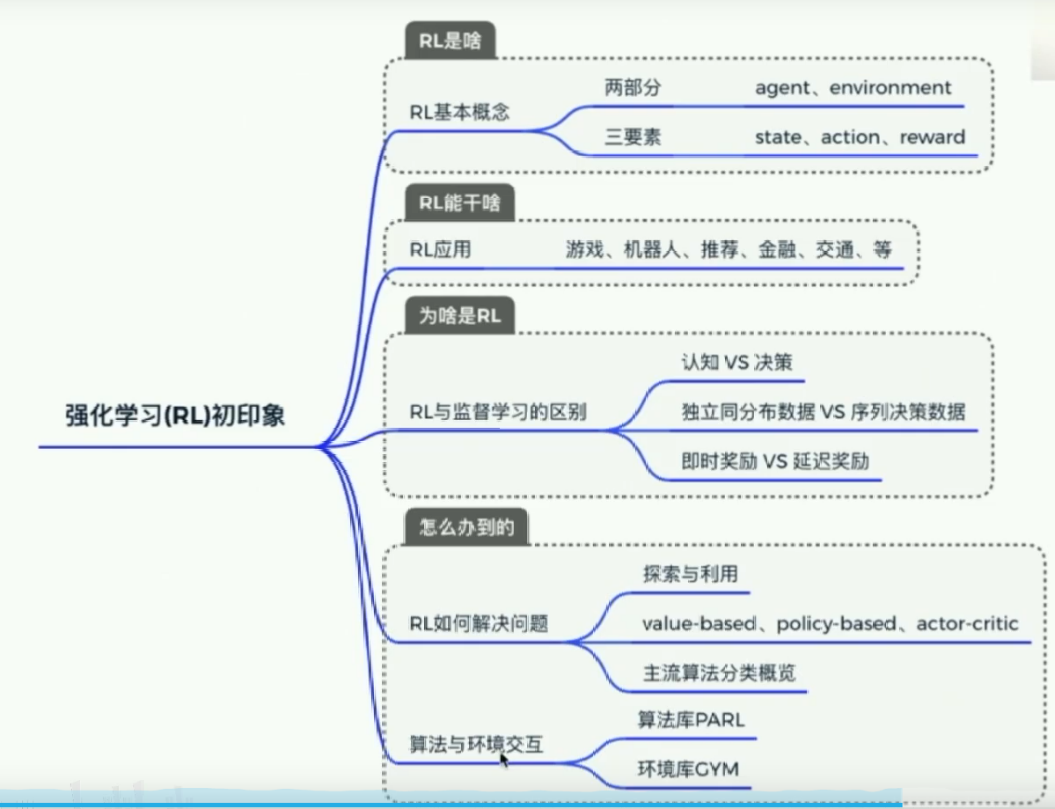
随机性策略就是输出的最终策略是有概率的，例如执行某个动作的概率是多少

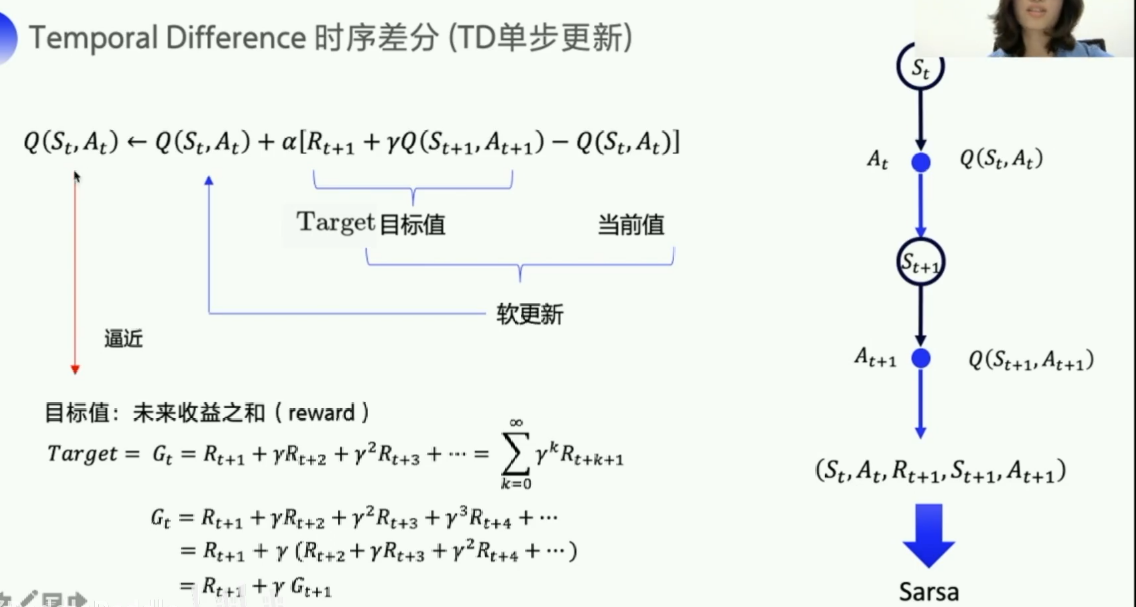






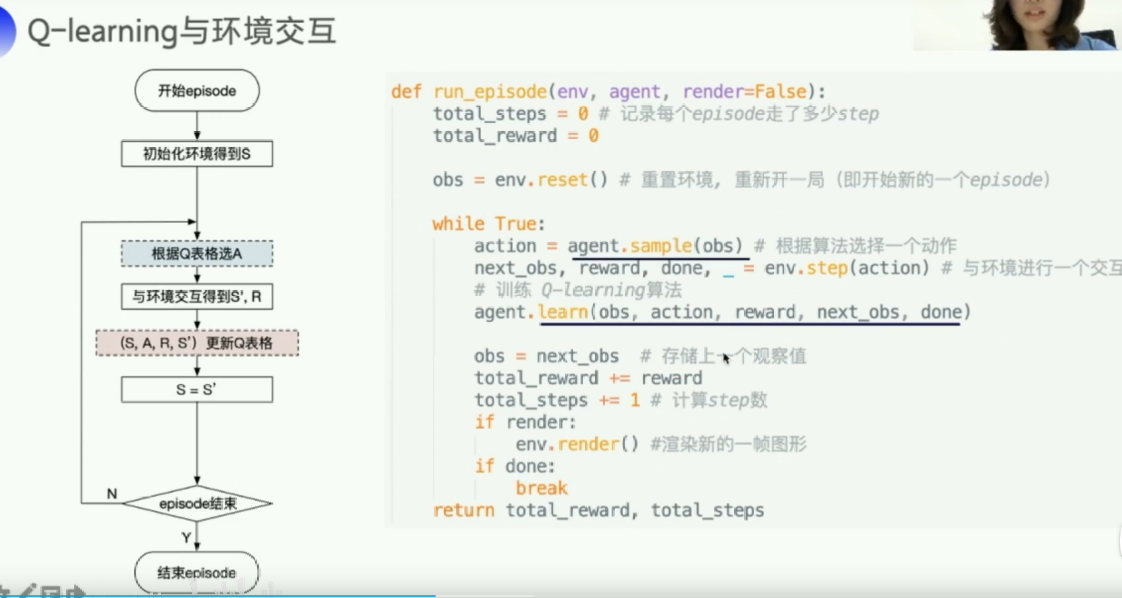


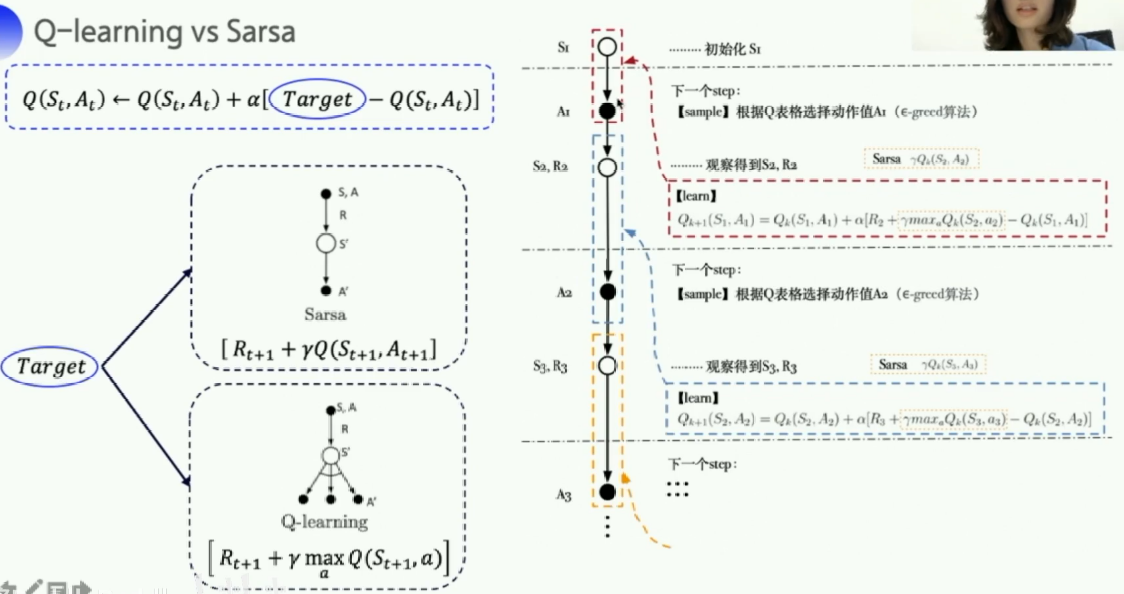




Sarsa就相当于（s,a,r,s,a）它需要获取下一时刻的状态和下一状态执行的动作；

而下面的Q-learning则不需要下一状态执行的动作；





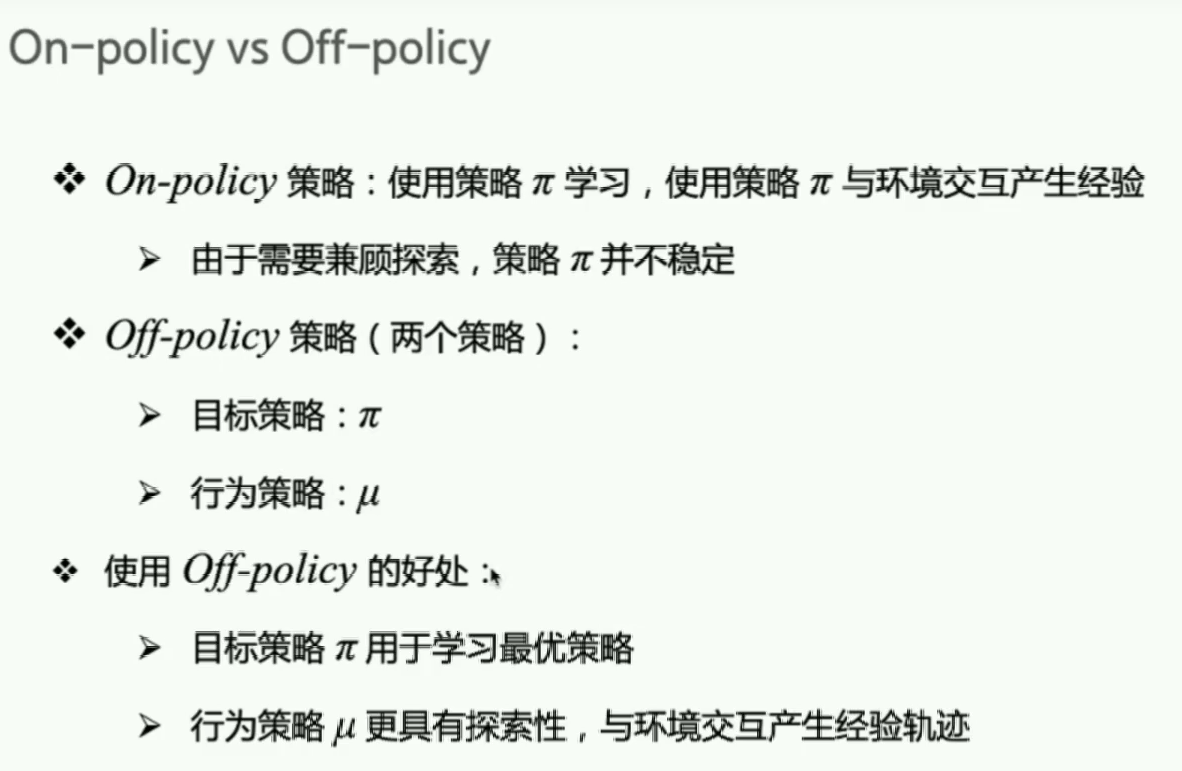
Q-learning优化的时候默认下一个动作就是Q最大的动作，它知道下一步动作有百分之十的探索，但它不担心后面的探索策略，而是按照最优的策略来优化目标策略，就是默认下一个动作就是Q最大的动作；我感觉就是这里把探索和优化分开了，训练过程中的实际执行里面有探索这个过程，但在优化的时候，只利用最优策略（下一状态最大的Q值）来优化；

而sarsa这里就是利用探索策略来优化，它用于优化的下一步的Q值就是实际执行的下一步的Q值；

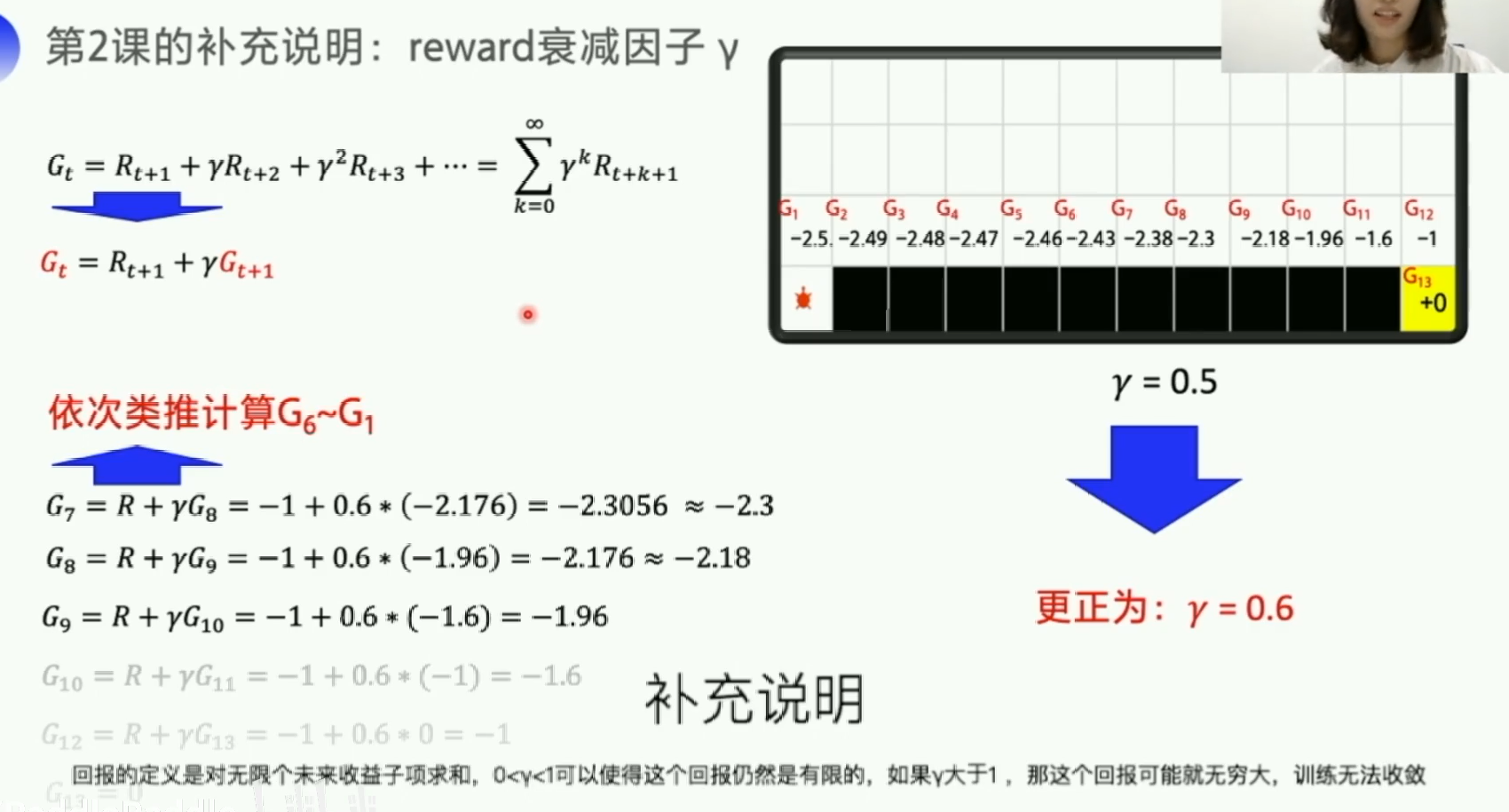
为什么悬崖问题sarsa和Q-learning训练的结果不一样？

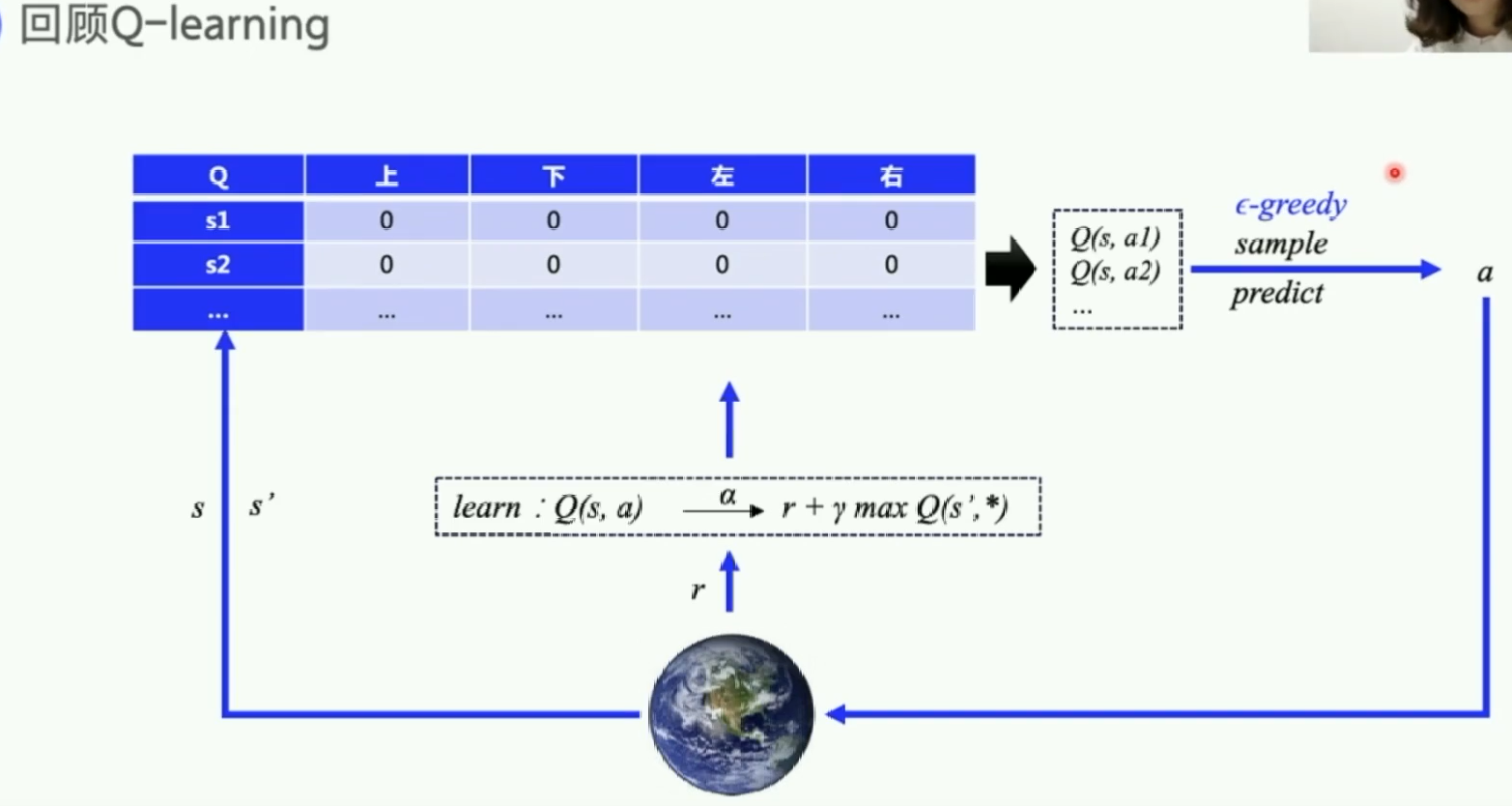
最后的策略是啥？结束训练之后得到的是啥？

我感觉是一个Q表，然后按照这个Q表来执行；这个Q表就是最优策略







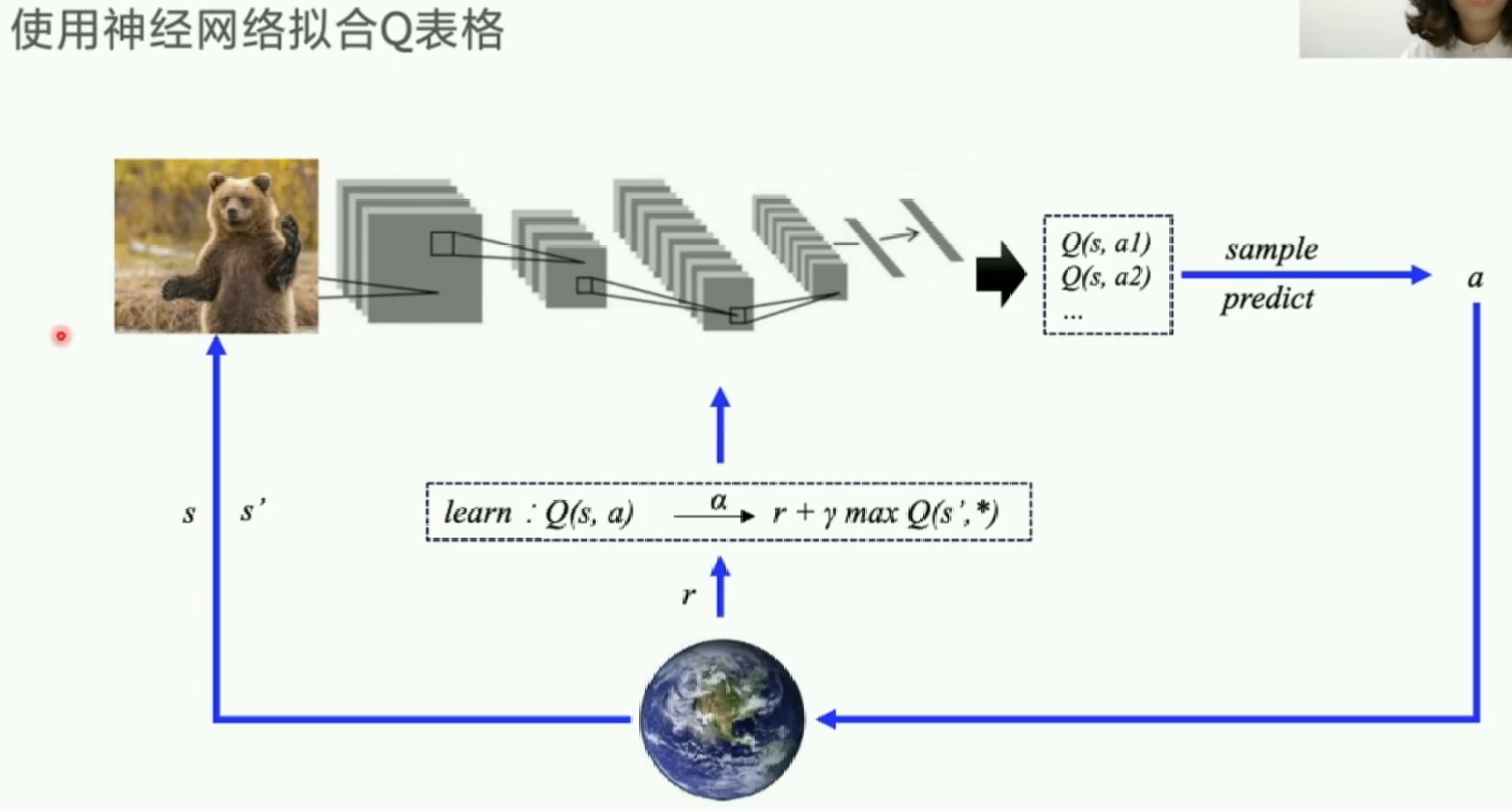


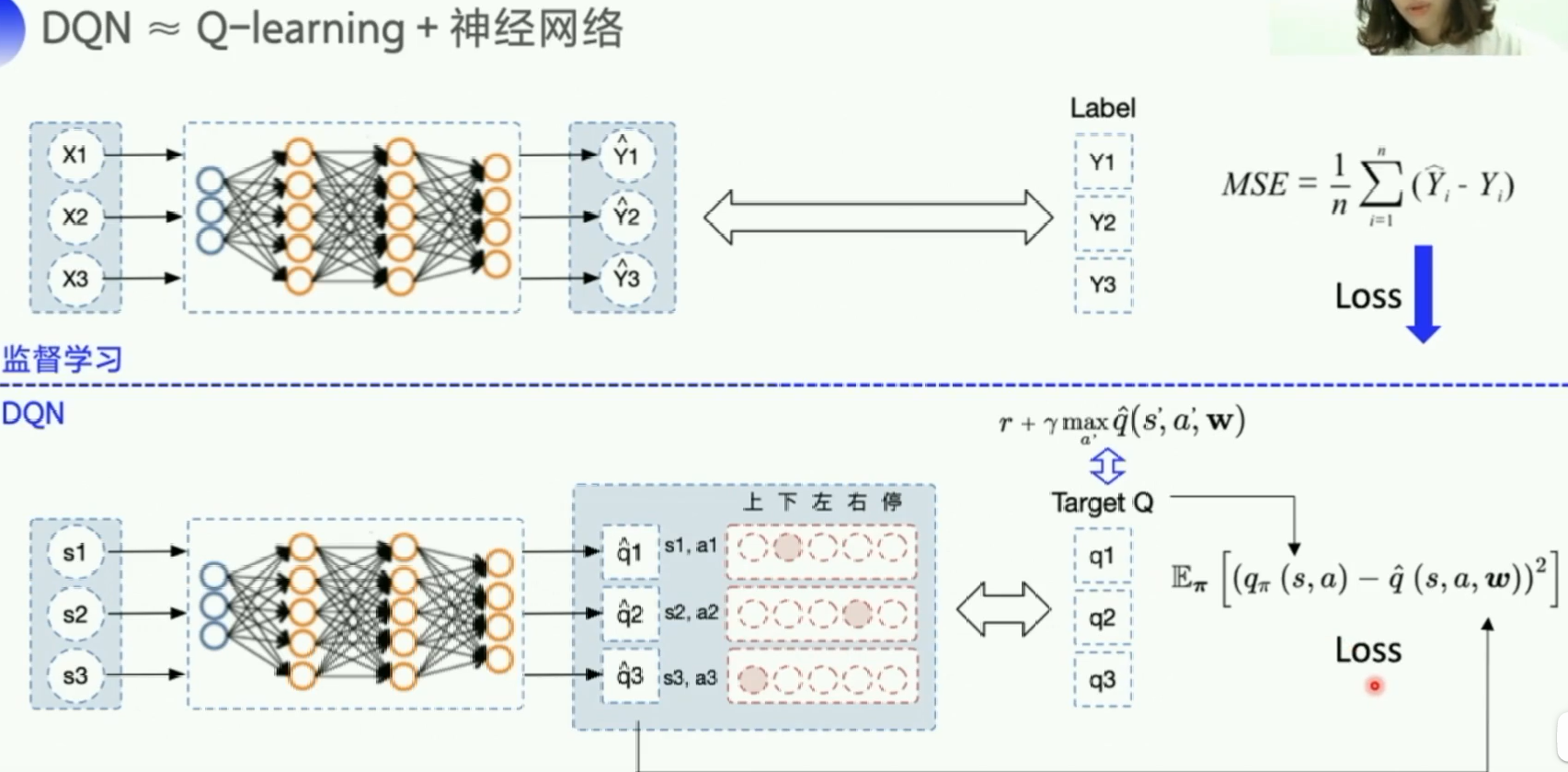
基于表格的方法有局限性：

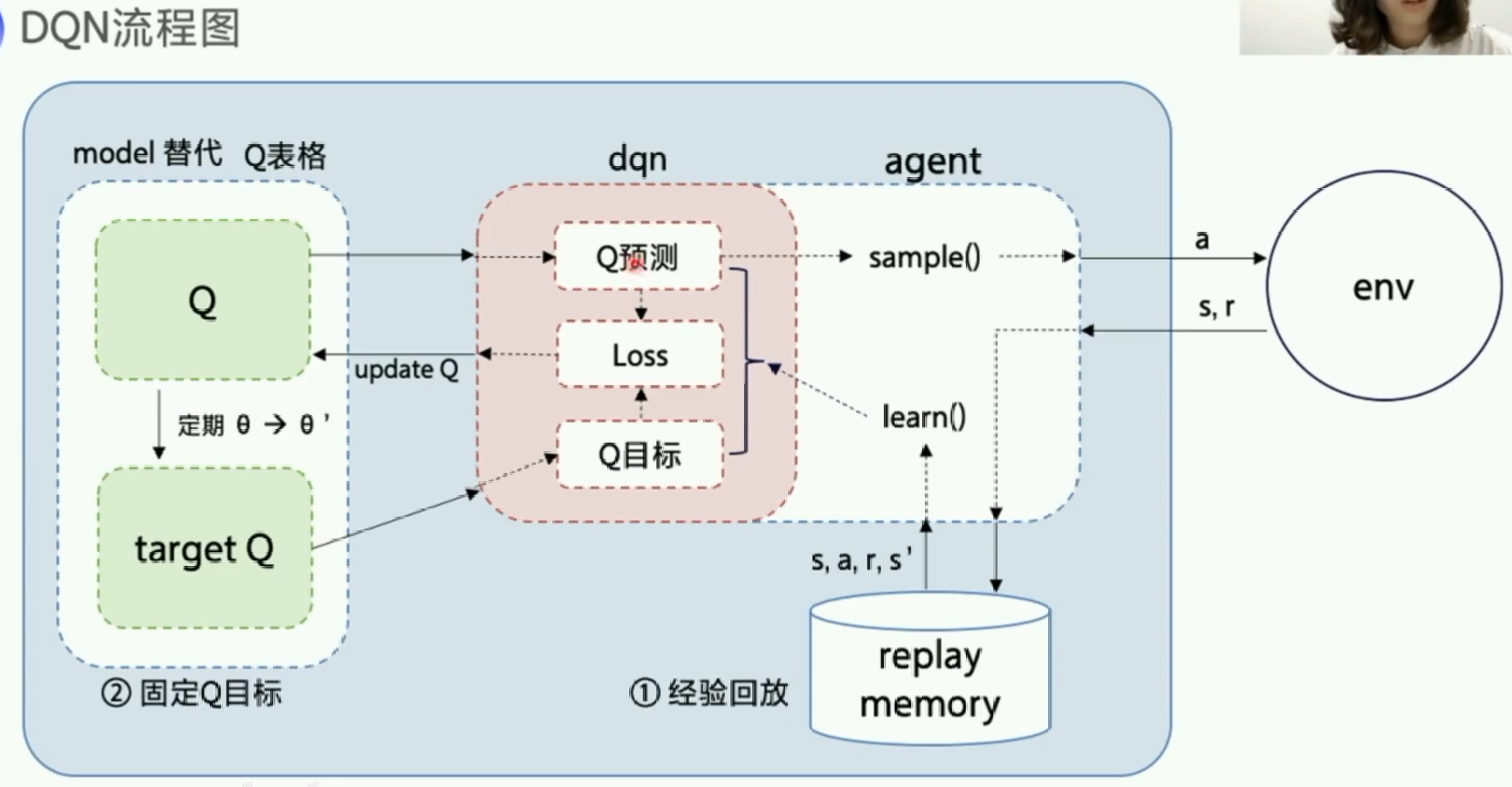
当 状态量太多的时候，需要很大的内存空间;状态量多了，查表格也很费时间；

RL---deep RL:少数状态到数不清的状态

DQN

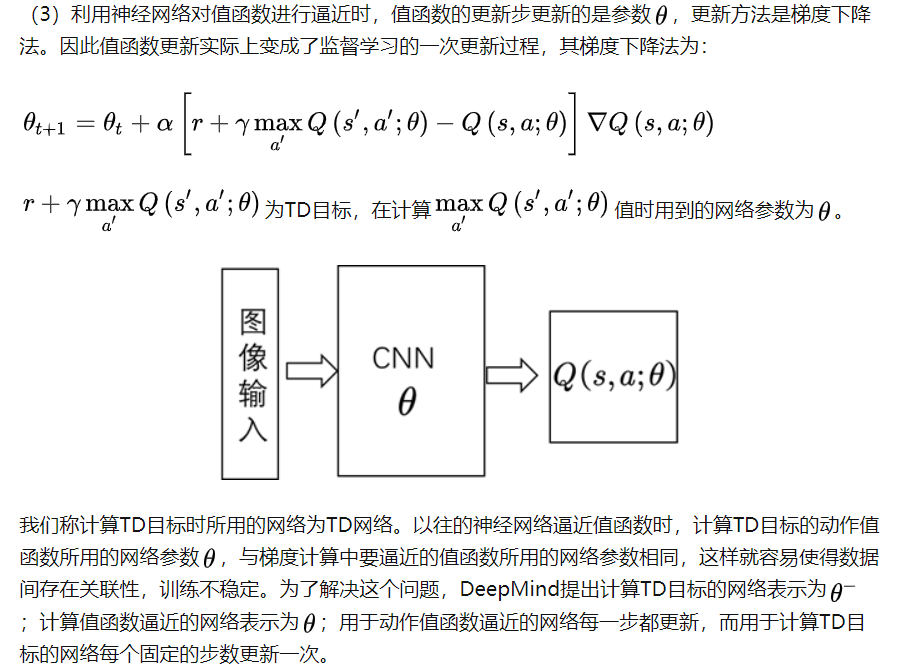


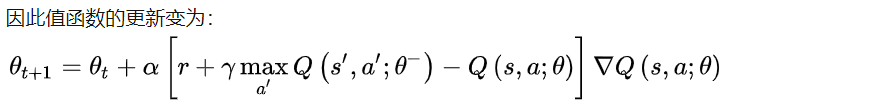




1. DQN利用深度卷积神经网络[下右图的神经网络]逼近值函数。此处的值函数对应着一组参数，在神经网络里参数是每层网络的权重，用θ表示，用公式表示的值函数Q(s,a;θ)。此时更新值函数时其实是更新参数θ，当神经网络确定时，θ就表示值函数。
2. 经验回放的动机是：①深度神经网络作为有监督学习模型，要求数据满足**独立同分布；**②通过强化学习采集的数据之间存在着关联性，利用这些数据进行顺序训练，神经网络表现不稳定，而经验回放可以打破数据间的关联。

在强化学习过程中，智能体将数据存储到一个数据库中，再利用均匀随机采样的方法从数据库中抽取数据，然后利用抽取的数据训练神经网络（13年的NIPS中已提出）。





注意：用目标神经网络逼近 值函数的时候，利用了t时刻（s状态下）采取动作之后得到的**奖励**和利用目标网络（target-Q）计算的下一状态的最大Q值【计算状态下，所有动作的状态动作价值函数Q（），从里面选一个最大的】之和来代替真值（**真正的Q(s,a)，或者说是更准确的值**）。

TD是时序差分：



实际上，DeepMind 还使用了很多其他的技巧来使训练有效 - 比如 target network, error clipping, reward clipping 等等，但这些都超出了本文的介绍范围。

这个算法最令人惊奇的地方在于它能学到任何东西。考虑一下，因为我们的**q函数**是**随机初始化**的，**它最初输出的是完全无用的**。我们使用**这个无用的输出**（**下一个状态的最大 Q 值**）作为目标，只是偶尔**堆叠**小小的**奖励值（reward）**[这就上一段说的奖励**，这个奖励是真正有用的**]。这听起来好像完全没什么道理，这样怎么可能学到具有真实意义的东西?但事实是，它确实学到了。

链接：https://www.jianshu.com/p/41659ce20f15

来源：简书

虽然一开始的参数θ是随机的，但我们假设它是正确的，然后用于预测下一状态的Q值，将这个Q值与R相加，然后减去用神经网络算出的s状态下的Q值，得到差值，这个过程就是和Q-learning的算法类似，为了让差值更小，使用梯度下降的方法来修改参数θ。。？？

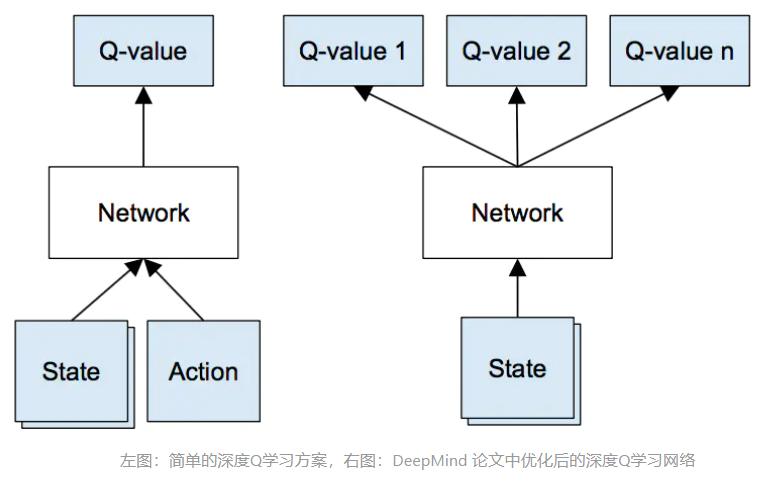
感觉这个算法里面有两个很重要的部分（关系到它如何能够学习成功）：

1. 减小目标Q值与实际Q值的差距：梯度下降法；
2. 累计的R的影响？？？（还需要思考？如何累积的？）
3. 上面的梯度公式具体怎么实现的？

回到我们的打砖块游戏，环境的状态可以用横板的位置，小球的位置和运动方向，和每一个砖块的已经被消除还是依然存在来表示。这种直观的表达方式限于特定的游戏。我们能不能想出一个更普遍的表达方式，可以适用于所有游戏。显而易见，我们可以使用屏幕像素。除了小球的速度和方向外，他们还隐式地包含了所有游戏当前的相关信息。两个连续的屏幕影像也可以覆盖在一起。

如果我们像 DeepMind 论文中提到的那样对游戏图像进行预处理，取最近的 4 帧图像，调整为 84 X 84，转换成 256 级灰度的图像。这样整个游戏就会有 256 的 84x84x4 次方，大约 10 的 67970 次方个不同的 state 。这意味着我们的 Q值表里要有 10 的 67970 次方个列。这比已知宇宙中原子的数量还多。有人会说实际屏幕中很多像素也许一直不会发生变化，我们可以将他们设计成一个包含只读状态值的稀疏表。即使如此，**大部分状态值可能很少被访问过，而且要让 Q 值表收敛也许要花上一辈子时间。理想情况下，我们还希望能对我们从未见过的状态进行 Q 值的准确猜测。**

这就是深度学习的目标所在。神经网络极其擅长处理高度结构化的数据。我们可以使用神经网络来表示我们的 Q 函数。取 4 四帧游戏影像作为 state，加上可执行的所有动作，作为输入，然后输出相应的 Q 值（下左图）。**另外我们也可以只将游戏影像作为输入，输出每个 action 对应的 Q 值（下右图）。**这样的方式反而更有优势，如果我们想要执行 Q 值的更新，或者选择具有最高 Q 值的对应的 action，我们只需经过整个网络一次就能立刻获得任意动作对应的 Q 值。



链接：https://www.jianshu.com/p/41659ce20f15

来源：简书

