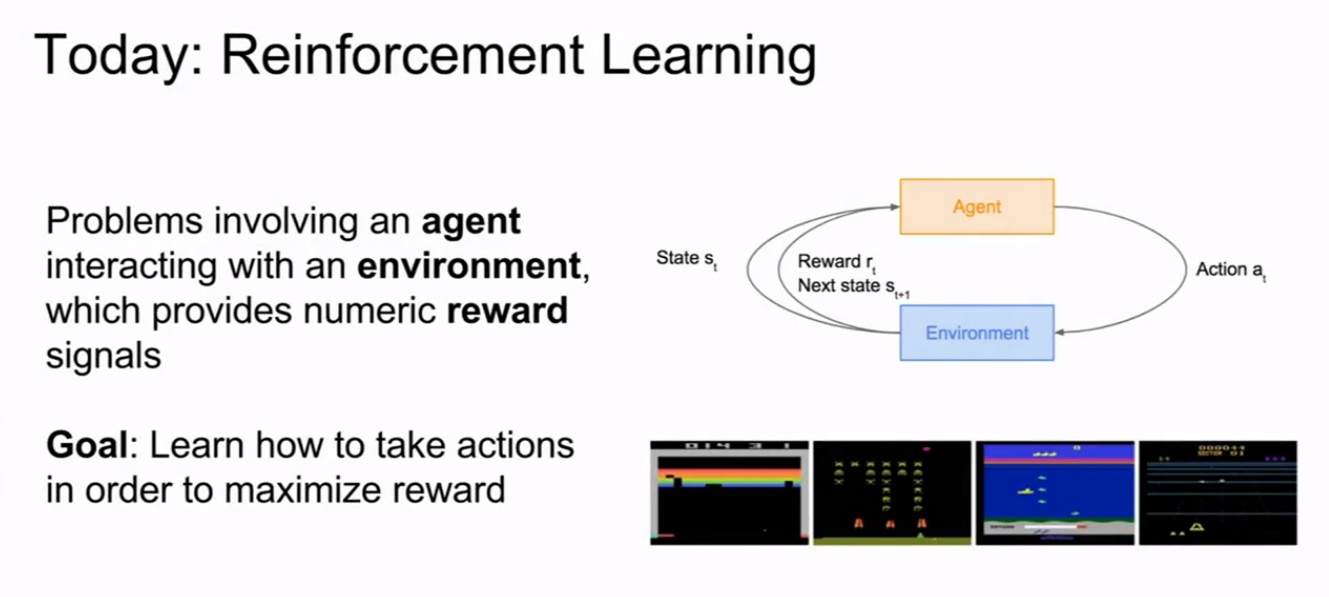
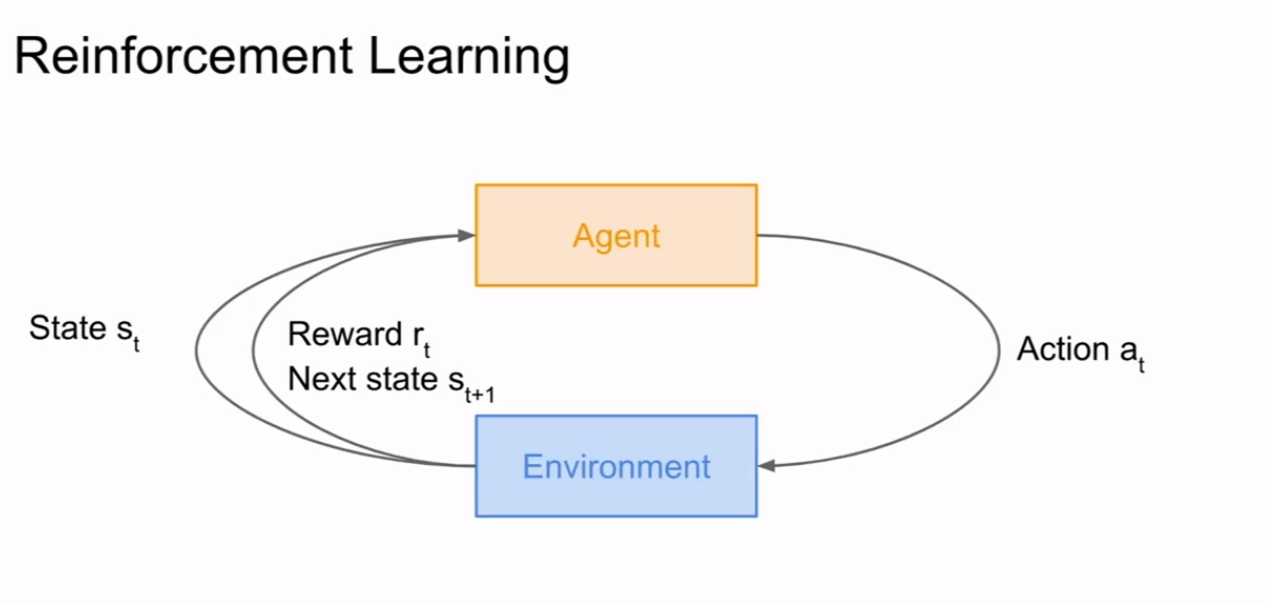
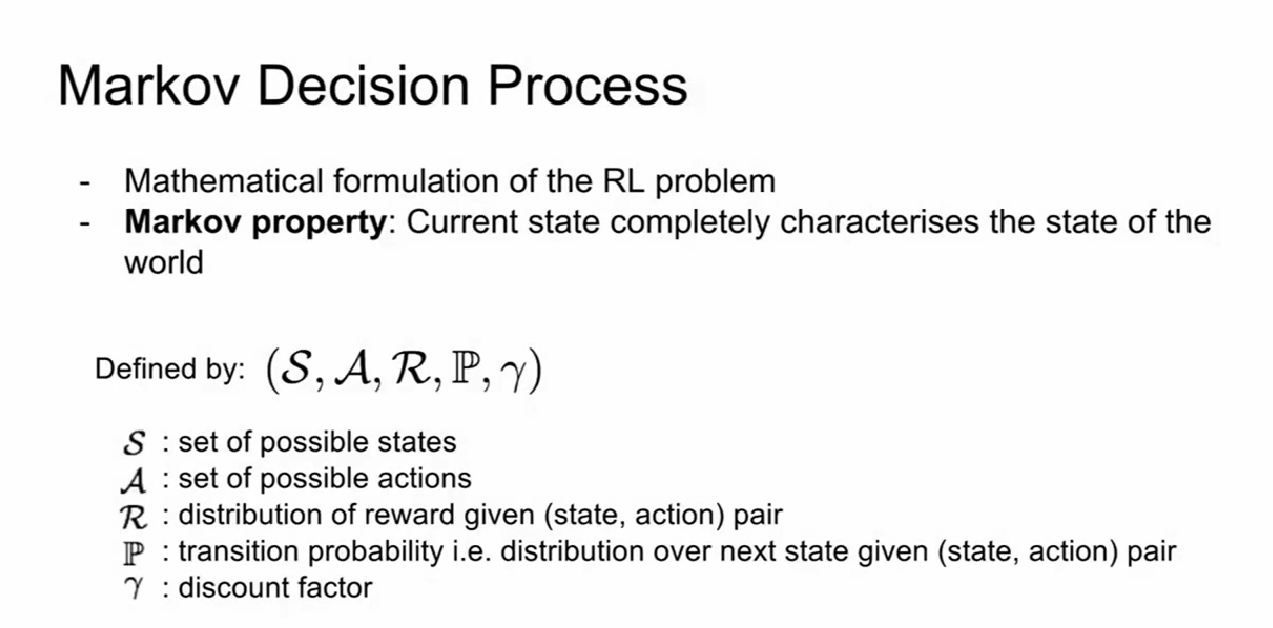
通过一个代理和环境进行交互执行一系列行动，通过这些行动可以获取最大的奖励(学习如何获取到最大的奖励)  


环境给予代理一个状态，代理根据这个状态执行相对应的动作，然后环境给这个动作一个奖励(该奖励为正时为奖励，为负时代表惩罚）然后环境会给予代理下一个状态循环往复，最后从中学会一个策略对环境给出的状态能够得到一个好的结果。



马尔可夫决策过程(Markov Decision Process)

当前状态由四个参数决定如下图所示，来自课程截图



决策过程:

1.在第一步初始化当前状态s0和当前状态的概率P0.

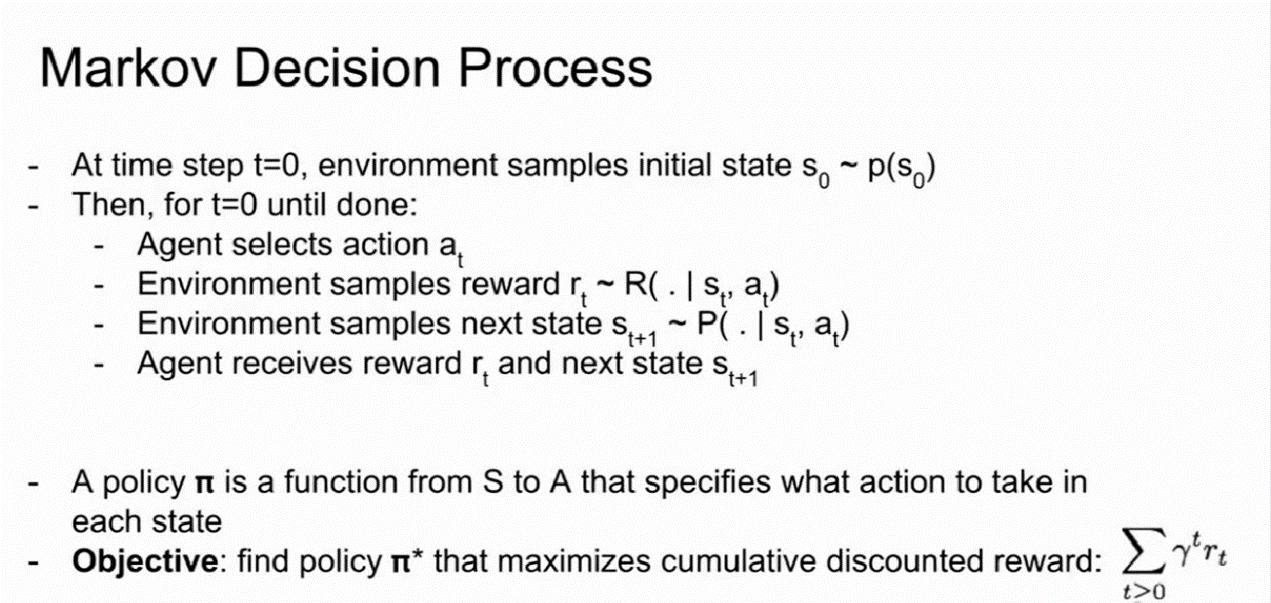
2.循环以下行动直到完成训练:

2.1 Agent(代理)从动作集合中选择当前一个动作应用于当前状态

2.2代理执行动作后环境给出一个评价对应当前的状态和动作(告诉代理它做的对不对，给出惩罚或者奖励)。

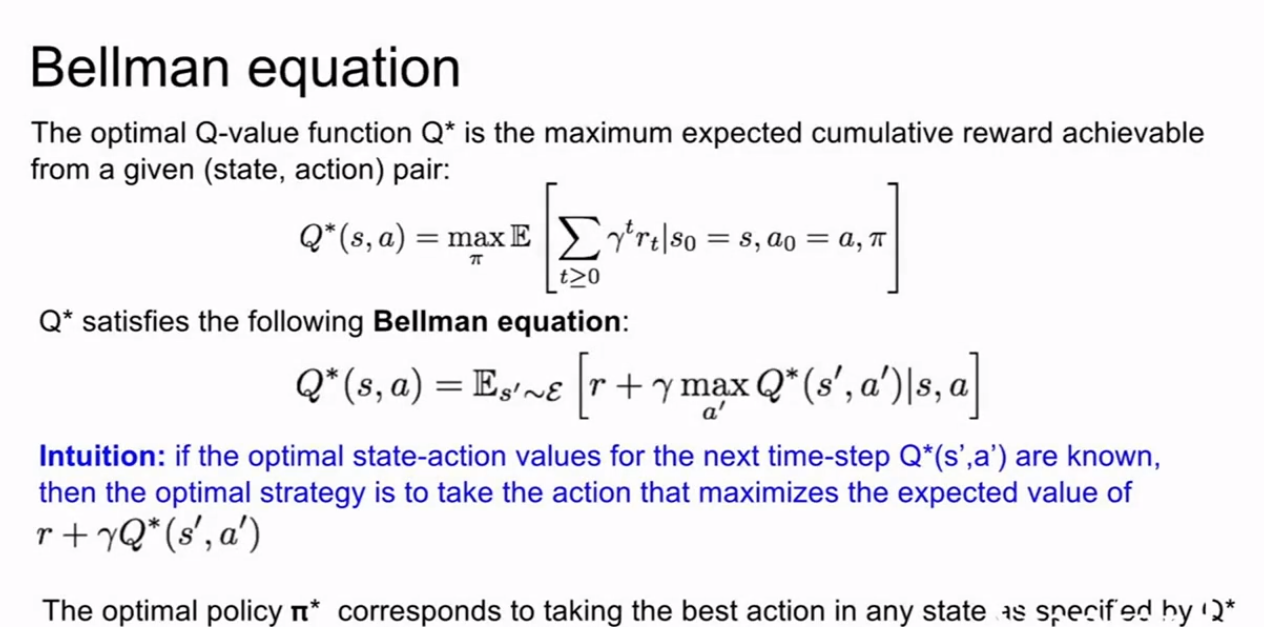
2.3环境生成下一步的状态(由当前状态和动作得到一个概率，该概率用来生成下一步状态)

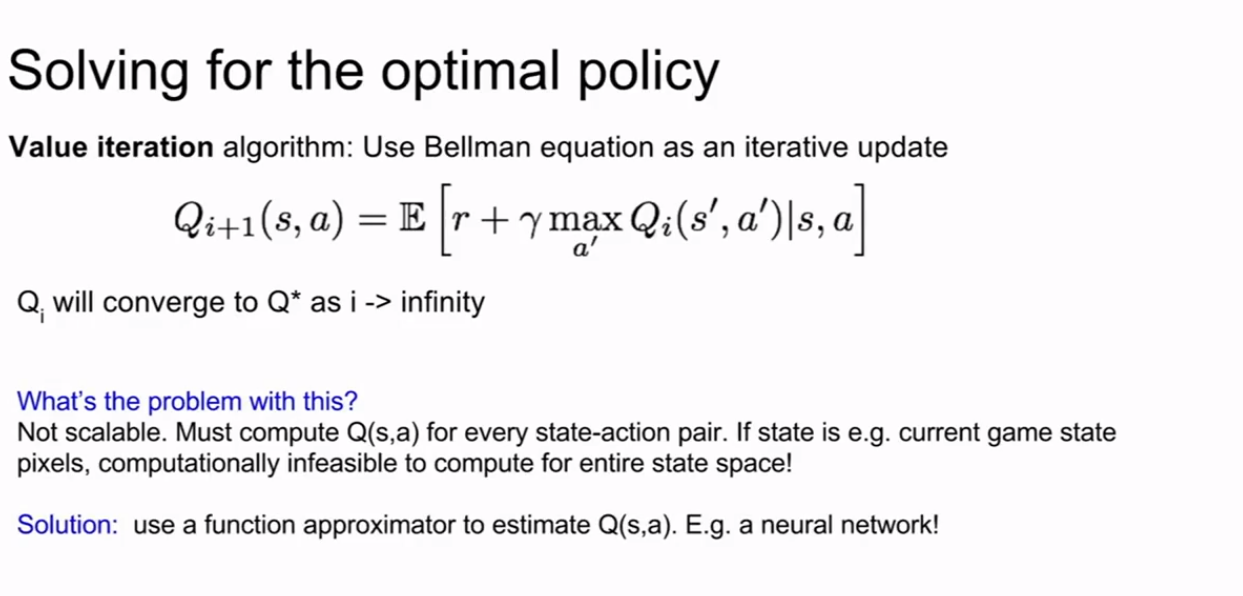
2.4代理接收到奖励和下一步的状态，准备进入下一次循环



Bellman equation

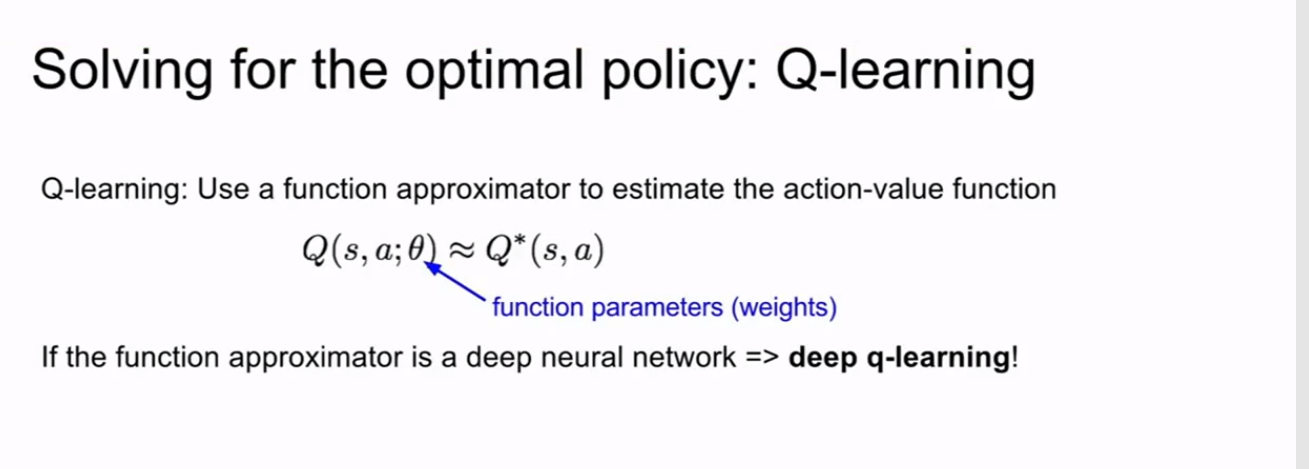
通过执行一些行动我们可以得到一个最大期望的Q值，通过迭代来达到一个收敛值。缺点:不可扩展，对于每一个状态都需要进行一次计算，计算量大。





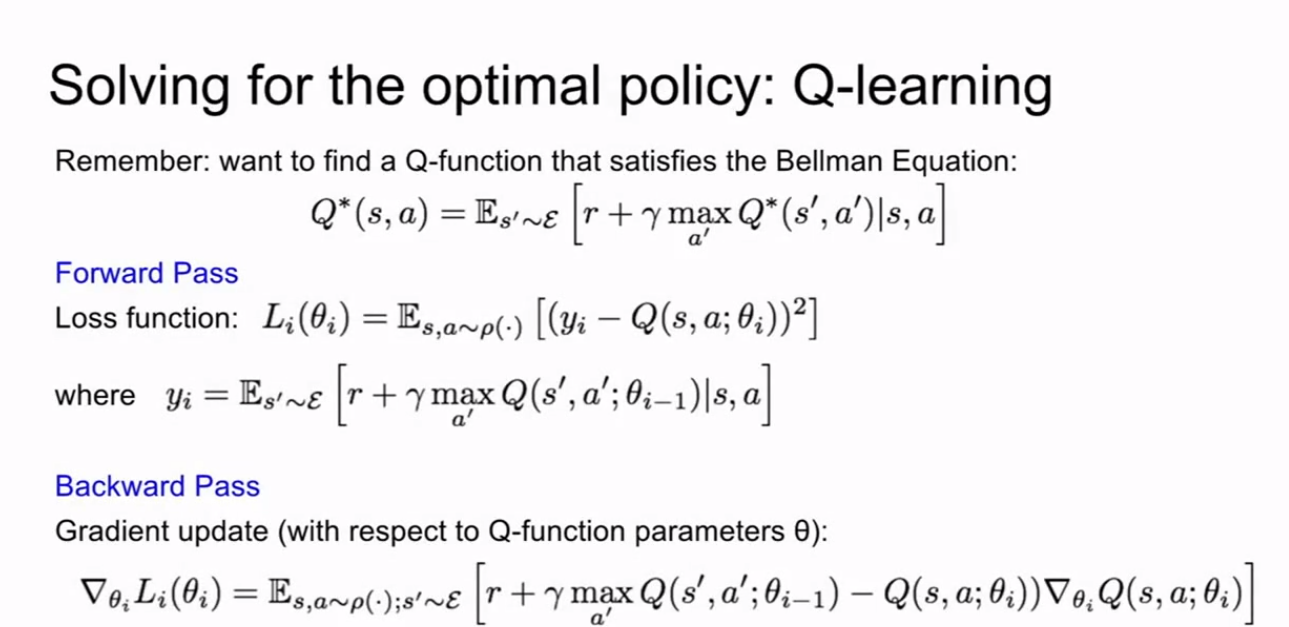
Q-learning

使用一个函数去估计action-value函数，如果这个逼近器(进行近似估计的工具是神经网络)就是进行Deep Q-learning

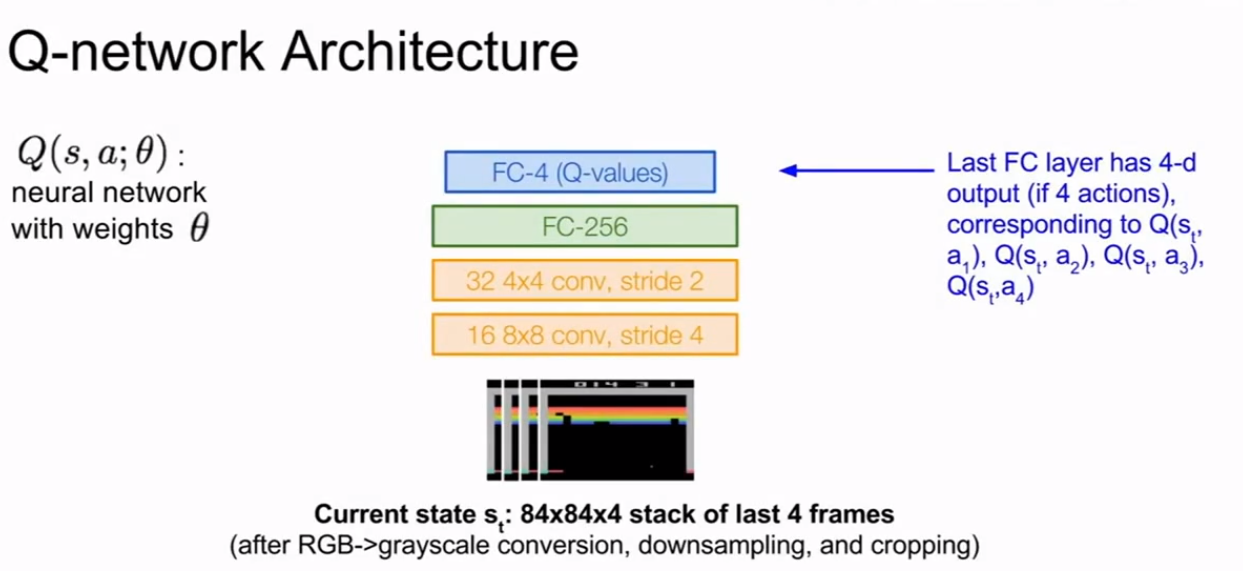


我们想要寻找的Q-function是瞒住Bellman Equation(Bellman 方程)

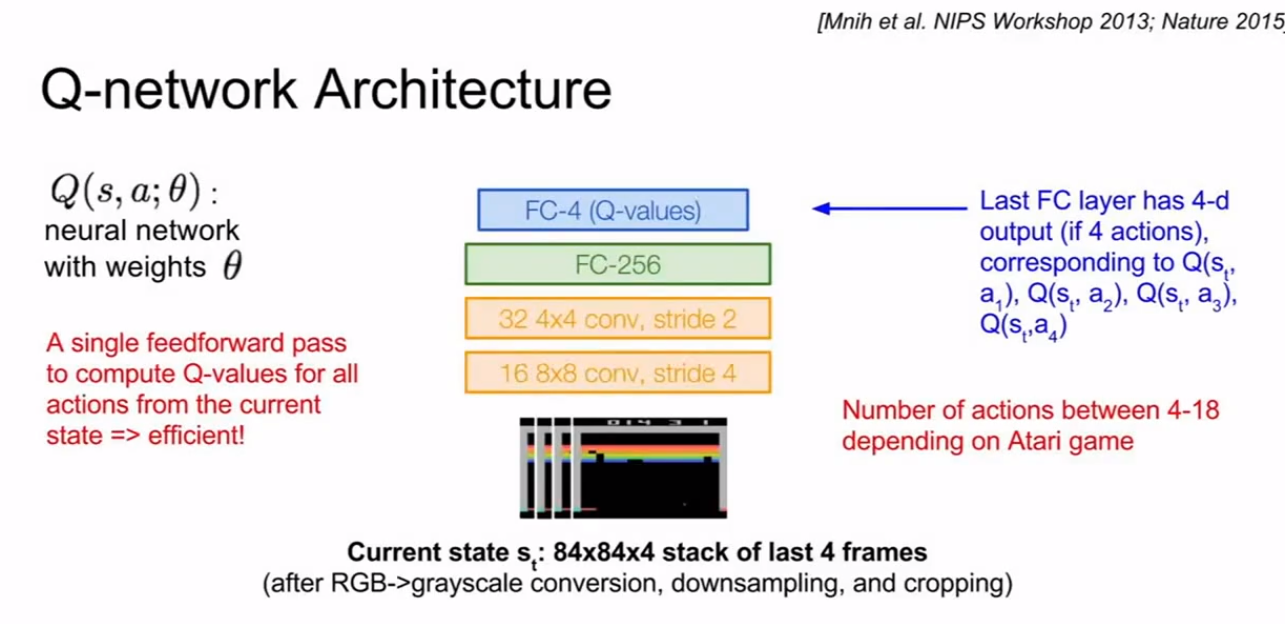
损失函数是寻找最小的Bellman方程的误差



一个示例(Q-network Architecture)

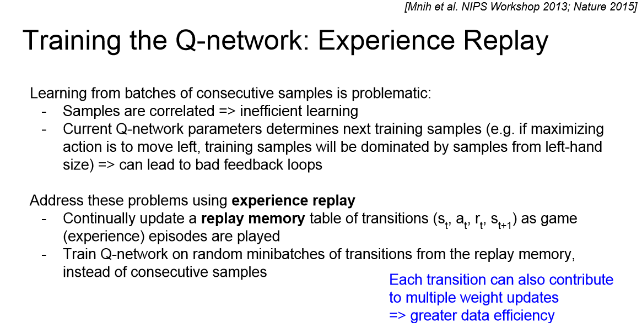


无softmax，我们主要用于对Q值进行回归

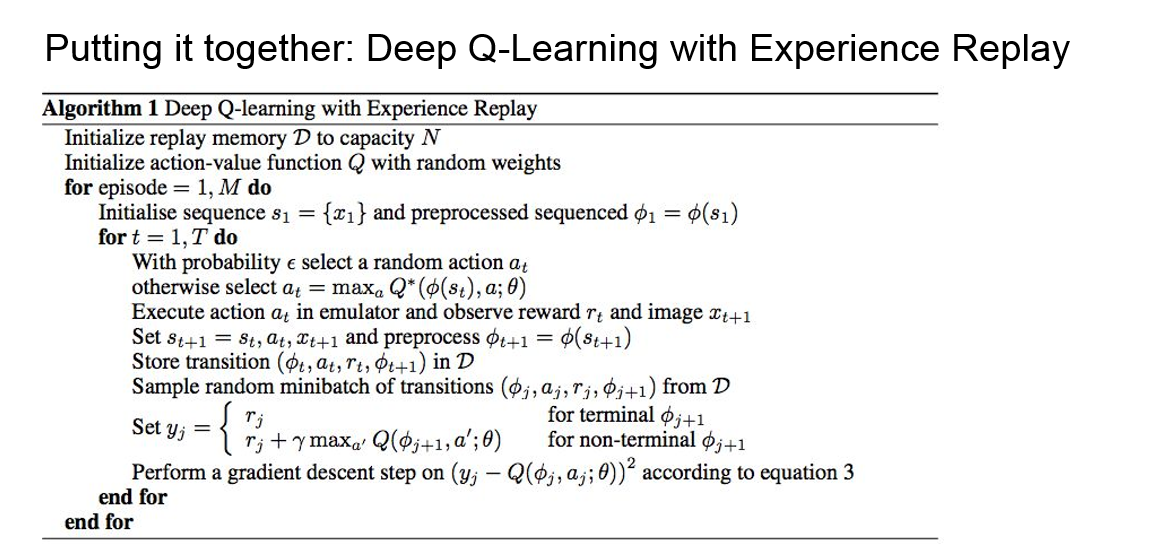


使用之前提到的损失函数进行训练这个Q-network。

对于之前架构训练时的一些问题和解决方法(进行状态转换（经验重放），例不断转换游戏中的状态，学习到更多的经验）

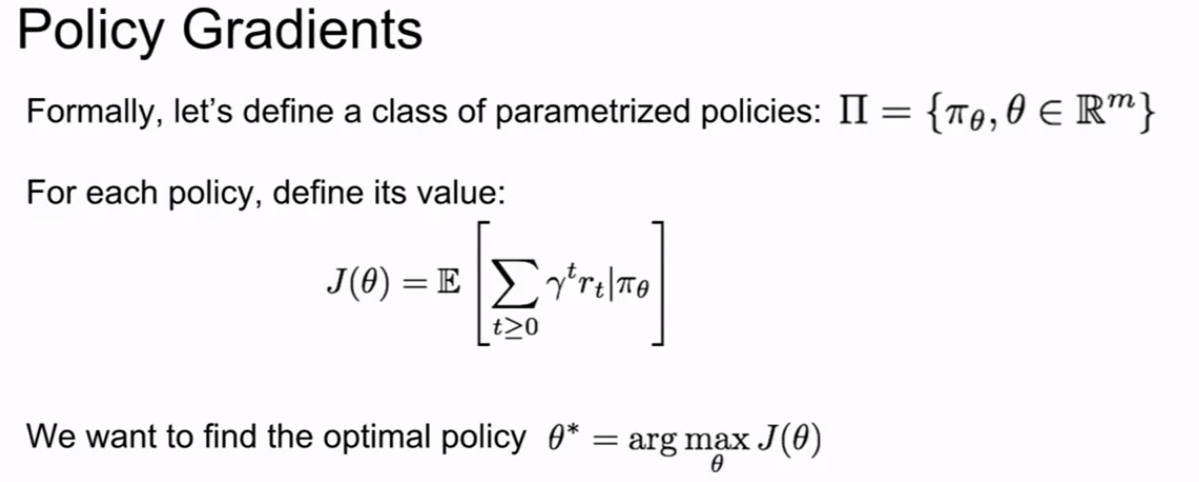


修正后的学习算法(带有经验重放的Deep Q-learning)



Q-learning中存在的问题:Q-function会非常复杂

因而引入决策梯度(该算法用于评测每一个决策的价值)



使用梯度上升来获取到函数J的最大值。

Actor-Critic算法

