1. **蒙特卡洛、TD、动态规划的关系**

三者之间的相同点：如果说三者的相同点 那么也只有都是用于进行值函数的描述与更新的了；

两者之间的共同点：蒙特卡洛和TD算法隶属于model-free，而动态规划属于model-based；TD算法和蒙特卡洛的方法 因为都是基于model-free的方法，因而对于后续状态的获知 也都是基于试验的方法；TD算法和动态规划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值函数的更新

两者之间的不同点：TD算法和蒙特卡洛的方法当然也有不同之处在于：TD算法不需要等到实验结束后 才能进行当前状态的值函数的计算与更新，而蒙特卡洛的方法需要试验交互 产生一整条的马尔科夫链「到最终状态」才能进行更新。TD算法和动态规划的策略评估不同之处还是就回到model-free和model-based 这一点上来了，动态规划可以凭借已知转移概率就能推断出来后续的状态情况，而TD只能借助试验才能知道了「因而 在强化学习的实验中 设置多次episode 以遍历尽量多的状态 和马尔科夫链 来降低方差」

蒙特卡洛方法和TD方法的一个区别，因为蒙特卡洛方法进行完整的采样来获取了长期的回报值，因而在价值估计上会有着更小的偏差，但是也正因为收集了完整的信息 所以价值的方差会更大「毕竟基于试验的采样得到 和真实的分布还是有差距 不充足的交互导致的较大方差」；而TD算法与其相反 因为只考虑了前一步的回报值 其他都是基于之前的估计值 因而估计具有偏差 但方差也较小；

1. **off-policy和on-policy的区别**

对于两者区别的划分众所周知：就是看选取动作的策略和进行评估的策略是不是同一个策略就是了；而一般不同之处都是在评估的时候，进行对于下一个状态值函数预测的时候，qlearning选用的直接是next\_state的最大的那个Q动作值函数，而sarsa还是依策略的和之前选取state时候动作action一样的选取next\_state的动作action\_进而确定Q动作值函数；举个SARSA和qlearning的例子就是：前者基于某一个策略如ε-greedy进行选取action，后面需要值函数进行更新的时候 也还是ε-greedy策略。后者则不然 选取action的策略是基于ε-greedy，更新值函数的时候依旧是选取max的那个；

on-policy优点因为是单纯的依照交互数据来进行更新，所以直接了当。但速度不一定快 毕竟试验交互时候还有样本的采样效率的问题，可以看到off-policy可以用到之前的经验，而on-policy只能从新采样，因而速度不一定快。同时劣势是不一定找到最优策略。

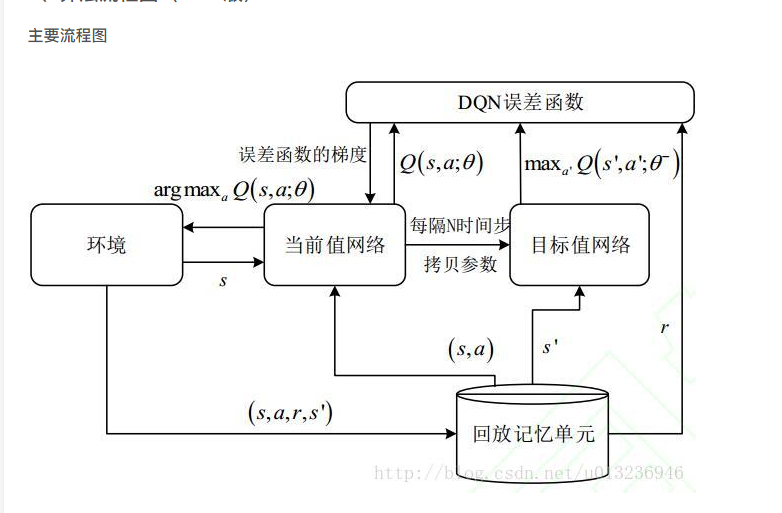
而off-policy优势在于更强的通用性 保证了探索性，其强大是因为它确保了数据全面性，所有行为都能覆盖。「这里所说的就是在选取snext对应action的时候需要选取max的一个 那么需要全部action的q值 于是覆盖了全部的行为，相比之下on-policy只是单纯的依照策略选取了某个action；」但劣势也很明显 就是每次对于状态的值函数的估计 都是过高的进行估计。同时毕竟是基于采样的方式的 所以会有些状态没有被采样到，这就产生的偏差

1. **值函数近似**

表格式到函数近似

本质上还是为了建立从状态到值函数 或者状态动作到值函数之间的映射，既然是映射，那么就肯定存在一个函数可以来表示这个映射关系，参考机器学习中的各类方法，我们可以建立一个模型来表示这个从状态到值函数 或者状态动作到值函数之间的映射关系，进而就转化为标准的机器学习中有监督学习的回归问题求解：选择好算法，损失函数，优化方法，进而就能完成求解

1. **DQN**



很显然DQN相比于qlearning主要有三处改变：

首先DQN采用了深度卷积神经网络来进行的值函数逼近，这里选取卷积神经网络的原因也在于 原文是针对Atari游戏来作为environment的，输入状态采用的是84x84的图片，那么这里其实就涉及一个深度学习中一个选取问题就是CNN的优势 毕竟不是深度学习的部分 这里直接说结论，CNN凭借着本身稀疏连接和参数共享 相比于FCN来说 计算量大大减少，存储的参数量也大大减少；进而本身有着的局部特征和平移等变性 也让CNN很适合用于处理图片。当然尽管说这么多，这也不是什么新鲜操作了；但加入了第二点 就完全不同了，解决了RL里面的一个痛点；

加入了经验回放的操作来训练强化学习；首先我们要知道 如果直接借助强化学习交互产生的数据 本身是带有关联性的，而在神经网络或者直接说机器学习中 对于数据的基本要求就是独立同分布。因而这里引入了经验回放这个操作 来打破了数据间的关联性，具体操作就是：agent在与环境交互的时候 将交互数据存放在一个库里面，然后训练的时候 从中随机采样数据进行训练；

加入了目标网络的概念来单独的处理TD算法中的TD偏差；首先这一点并没有在13年版的DQN里面使用，而是在15年版的DQN才出现的，这也是13年和15年版的区别。我们都知道 强化学习在表格式的时候 直接使用Q∗(s,a)=Q(s,a)+α(r+γmaxa′Q(s′,a′)−Q(s,a))Q∗(s,a)=Q(s,a)+α(r+γmaxa′Q(s′,a′)−Q(s,a)) 来对于值函数就直接更新了；但函数近似后不行，现在的值函数本质上就是个函数的形式如Q=f(θ;feature)Q=f(θ;feature) ，那么对于它的优化与更新也更多的是采用诸如梯度下降等方法来对于权值θ进行更新，有：TargetQ=r+γmaxa′Q(s′,a′;θ)TargetQ=r+γmaxa′Q(s′,a′;θ) 和 L(θ)=E[(TargetQ−Q(s,a;θ))2]L(θ)=E[(TargetQ−Q(s,a;θ))2] 。因而目标网络的改进就在于这里的TargetQTargetQ使用了一个单独的网络 ；「emmm 这么一说 本来qlearning里面就不是按照交互顺序 而是使用的最大值来得到的对于当前值函数的估计，现在又进一步打破关联性 使用另一个网络的输出来进行估计 进一步减少关联性」也就是说现在有着TargetQ=r+γmaxa′Q(s′,a′;θ−)TargetQ=r+γmaxa′Q(s′,a′;θ−) ，对于这个target 网络的权值θ−θ− 和前面的计算Q值的主网络不同，主网络是每一步一更新，而target网络每隔一段时间一更新；「联系后面的DDQN的话 两者的区别其实也就在于：DDQN是在选取动作的时候基于一个网络 而评价是基于另一个网络；也就是TargetQ=R+γQ(St+1,argmaxaQ(St+1,a;θt);θ′t)TargetQ=R+γQ(St+1,argmaxaQ(St+1,a;θt);θt′) 可以看到区别就是对于targetQ构建的时候 next state的action和q值选取还是不一样的」

1. **深度强化学习中的DQN里面的experience replay方法和A3C的asynchronous**

显而易见的 前者的经验重放就是一种标准的off-policy的方法，毕竟在DQN里面使用的时候是在对估计Q(s,a)，estimate\_Q(s, a) = r + lamda \* maxQ(s2, a) ，如果按照on-policy理解的话 这个时候的应该是estimate\_Q(s, a) = r + lamda \* Q(s2, a2) 也就是说这个时候应该保存当时候的网络模型来进行输出对应的Q值情况，进而进行基于相应的值函数 在基于相同的如ε-greedy来选择情况；现在经验重播中只保存了<s,a,r,snext> 谈何on-policy呢。

同时上面也说了是直接保存的交互数据进行训练，这样观察数据往往波动很大且前后sample相互关联「机器学习也要求样本彼此独立同分布」 经验重播的方法很不适合on-policy；

但多线程的synchronous不一样；因为是多个agent在多个环境实例中并行且异步的执行和学习。 数据就不存在上面所说的数据关联性的问题。多个并行的actor可以有助于exploration。在不同线程上使用不同的探索策略，使得经验数据在时间上的相关性很小。这样不需要DQN中的experience replay也可以起到稳定学习过程的作用，意味着学习过程可以是on-policy的。

1. **policy-based**

最优控制中求解最优值函数进而求解得到策略的思路

接在策略空间直接进行搜索，不再借助值函数那样的间接完成对于最优策略的确定；于是也带来了收敛性快这样的好处；方法包括策略梯度 DDPG等

同时和value-based相比，value-based对于连续动作 不能对每个动作都给予一个Q值 因而在连续动作集合中就不能有很好的表现；而policy-based可以有效地处理连续动作集的问题「值函数方法无法确定一个对应max Q的action」，但同样的容易收敛到局部最小值、方差较大。「这里的方差较大的原因其实和蒙特卡洛方法类似 每次交互都会产生一整条轨迹 然后基于这个策略更新的时候 会导致回报估计的波动 因而会导致高方差」

1. **actor-critic**

对于策略梯度这种因为采样问题导致的高方差，就像之前提到过MC和TD存在的问题一样，MC基于交互产生整条马尔科夫链而进行估计值函数，因为采样的问题 导致的高方差，而TD就只是利用连续两个时刻预测值的差值来更新模型，尽管基于后续状态对于当前状态进行估计，因此计算出来的策略梯度都存在偏差，但同样的换来较小的方差；

同样的思路：构建出来一个独立的模型来估计模型的长期误差，而不是单纯的使用轨迹的真实分布；进而产生了作为基于值函数方法和基于策略梯度方法的actor-critic方法；

actor是基于策略梯度的方法进行选取动作，而critic是基于值函数的方法来评价它，两者协作完成；

Actor - Critic的优点

（总的来说就是结合两种方式优点）：

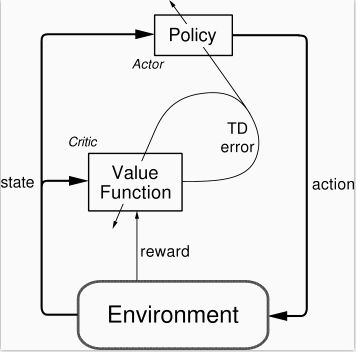
相比以值函数为中心的算法，Actor - Critic应用了策略梯度的做法，这能让它在连续动作或者高维动作空间中选取合适的动作, 而 Q-learning 做这件事会很困难甚至瘫痪。

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Policy Gradient的效率要高。最重要的还是不再使用采样得到的真实回报 降低了因为采样率导致的方差

Actor - Critic和两者的区别

和策略梯度的区别，很明显 关于策略梯度的算法表述里面 并没有涉及关于critic部分或者具体点说就是值函数更新的部分；参考RL:an introduction 里面所描述的那样P292中所描述的那样： 策略梯度方法里 状态值函数更多的是作为一种基准而非是critic；也就是说只是作为一种基准来判断哪个状态需要被更新「for state-function」而事实上 为了实现Policy Gradient，不管我们是计算Q，还是V，都需要一个对应的网络，这就是Critic。换句话讲，我们只有在使用Policy Gradient时完全不使用Q，仅使用reward真实值来评价，才叫做Policy Gradient，要不然Policy Gradient就需要有Q网络或者V网络，就是Actor Critic。 「对于其中采用什么值来当做R(τ) 参考这来理解https://zhuanlan.zhihu.com/p/26882898」

和值函数的方法 差别就更大了；首先对于策略的描述 就不是值函数方法里面借助 argmaxaQ(s,a) 而是使用的策略梯度的方法；当然使用对于该策略的好坏的评价 但也是一样的值函数；



如图 基于策略梯度的actor基于概率来对于某状态来选取动作action；而critic基于actor的行为判别行为的得分；actor进而基于该评价值来计算出来一个td error修改选择行为的概率「换句话说就是：actor的策略梯度的方法生成得到梯度的方向「也就说之前的logP(τ;θ)logP(τ;θ)」，然后进行沿着方向进行梯度的增减；我们需要一个值来判断这一个增减的方向是否正确 于是需要critic来计算出来td error」，同时基于选取的action计算TD error来更新critic；

具体到网络里面：Actor和Critic各为一个网络，Actor输入是状态输出的是动作，loss就是logprob×tderrorlogprob×tderror,(和策略梯度相对应，注意到这里的loss和Policy Gradient中的差不多，只是vt换成td\_error，引导奖励值vt换了来源（Critic给的）而已)，Critic输入的是状态输出的是Q值，loss是square((r+gamma\*Q\_next) - Q\_eval)也就是square(td\_error)，也就是说这里更新critic对应Q-learning是一样的均方误差。

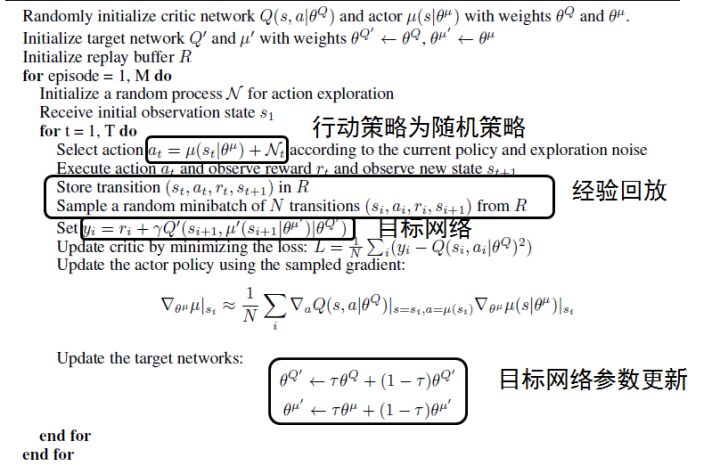
1. **A3C**

A3C的全名是Asynchronous Advantage Actor-Critic

而是将one-step Sarsa, one-step Q-learning, n-step Q-learning和advantage AC扩展至多线程异步架构。 可以看到几种方法的不同： 「AC是on-policy的policy搜索方法，而Q-learning是off-policy value-based方法。这也体现了该框架的通用性。 」

1. **DPG和DDPG**

DDPG 和DPG的区别也只是在于：使用DNN来逼近值函数和确定性策略



1. **强化学习解决的是什么样的问题？**

我答：“序列决策问题。”

1. **强化学习的损失函数（loss function）是什么**

log likelihood——对数似然函数值，更常使用log likelihood log： log (pi(a|s,theta)), 我们就可以构造一个损失函数如下：

L(theta) = sum( log(pi(a|s,theta)\*f(s,a) ))

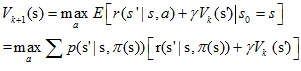
1. **贝尔曼方程**

“贝尔曼方程（Bellman Equation）”也被称作“动态规划方程（Dynamic Programming Equation）”，由理查·贝尔曼（Richard Bellman）发现。贝尔曼方程是动态规划（Dynamic Programming）这种数学最佳化方法能够达到最佳化的必要条件。此方程将“决策问题在特定时间点的值”以“来自初始选择的报酬 及 由初始选择衍生的决策问题的值”的形式表示。藉这个方式将动态最佳化问题变成较简单的子问题，而这些子问题遵守由贝尔曼所提出的“最佳化原理”。

1. **最优值函数和最优策略为什么等价？**

策略迭代算法包含了一个策略估计的过程，而策略估计则需要扫描(sweep)所有的状态若干次，其中巨大的计算量直接影响了策略迭代算法的效率。我们必须要获得精确的Vπ值吗？事实上不必，有几种方法可以在保证算法收敛的情况下，缩短策略估计的过程。

值迭代（Value Iteration）就是其中非常重要的一种。它的每次迭代只扫描(sweep)了每个状态一次。值迭代的每次迭代对所有的s∈S按照下列公式更新：



1. **使得策略梯度的方差最小时的基线b**