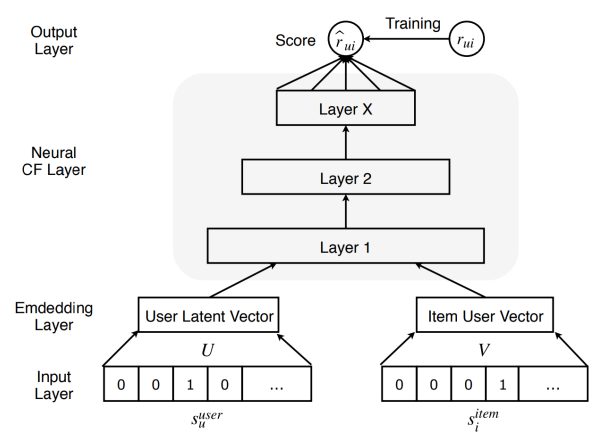
**Deep Learning based Recommender System:A Survey and New Perspectives (2018.09)**

**+ AAAI2019 + 其他论文**

基于深度学习的推荐系统可以根据深度学习种类进行划分：

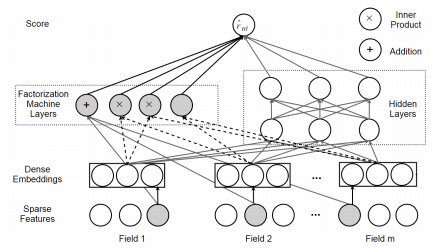
**·多层感知机MLP**

1.NCF



Embedding -> Layer1有两种连接方式：concat以及element-wise相乘。

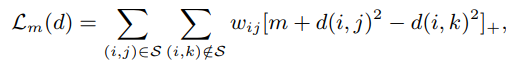
2.deepFM

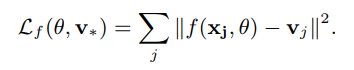


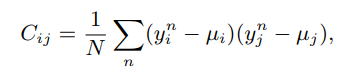
Deep&Wide网络的例子：Deep为普通的MLP模型，Wide为利用FM算出的特征交叉项。

3.CML(Collaborative Metric Learning)

相当于distance\_loss+embed\_loss+regular

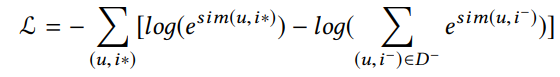


f为MLP

，y是user或者item向量

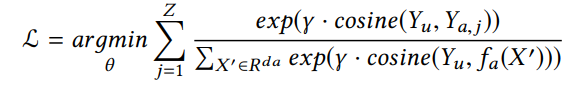
4.DSSM based(Deep Structured Semantic Model)

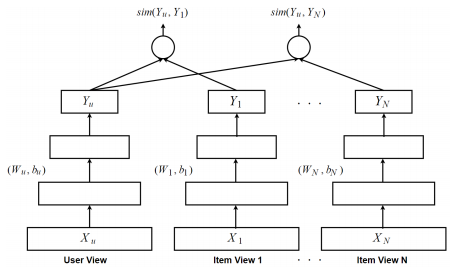
4.1 DSPR(Deep Semantic Similarity based Personalized Recommendation)



使用MLP获取user和item的Embedding向量并计算余弦相似度。

4.2 MV-DNN(Multi-View Deep Neural Network)

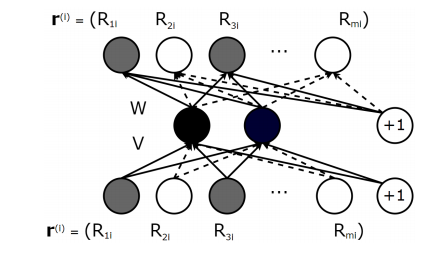


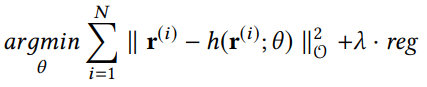


**·自编码器AE**

1.Autoencoder based Collaborative Filtering

1.1 AutoRec



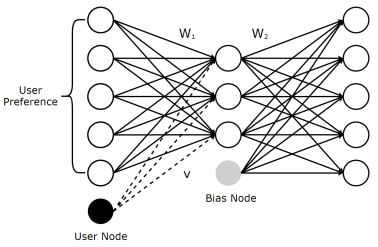
，

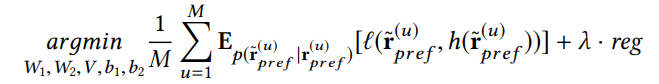
重构user或者item向量。

1.2 CFN

，输入融合了噪声，{}代表cancat，s为副信息。

1.3 CDAE(Collaborative Denoising Auto-Encoder)







输入融合噪声，p为条件高斯分布，l为平方损失或log损失，Vu是user节点。

1.4 Muli-VAE and Multi-DAE

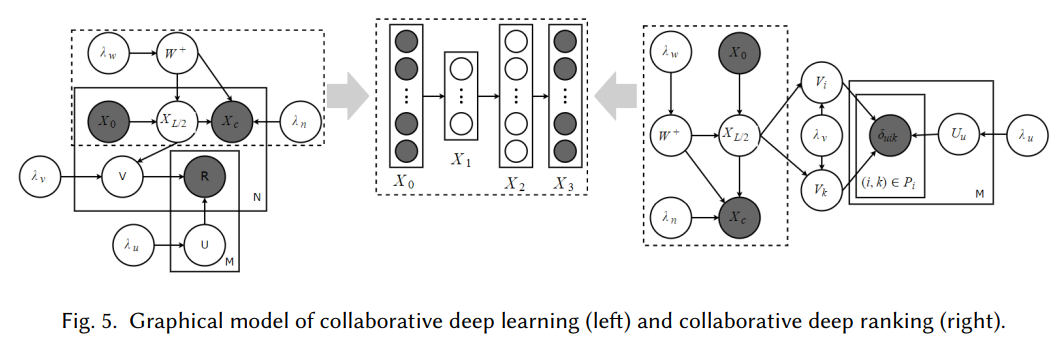
引入了贝叶斯推断进行参数估计

2.Feature Representation Learning with Autoencoder.

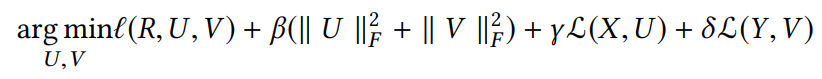
2.1 CDL(Collaborative Deep Learning)

结合stacked denoising autoencoder (SDAE)和probabilistic matrix factorization的组合贝叶斯模型。

2.2 CDR(Collaborative Deep Ranking)



2.3 Deep Collaborative Filtering Framework.



L表示连接深度学习以及协同过滤模型的中枢以及结合副信息以及隐因子的连接。

2.4 AutoSVD++、HRCD

**·卷积神经网络CNNs**

1.Feature Representation Learning with CNNs

1.1 CNNs for Image Feature Extraction

1.2 CNNs for Text Feature Extraction

1.3 CNNs for Audio and Video Feature Extraction

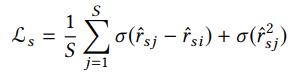
2.CNNs based Collaborative Filtering

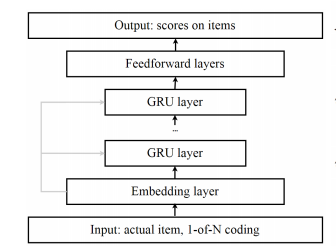
3.Graph CNNs for Recommendation

**·循环神经网络RNNs**

1.Session-based Recommendation without User Identifier

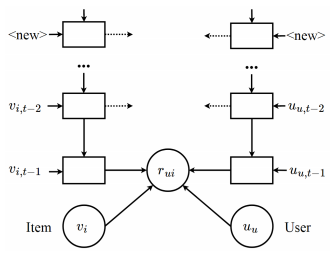
GRU4Rec

ranking-loss ，j for pos and I for neg



2.Sequential Recommendation with User Identier

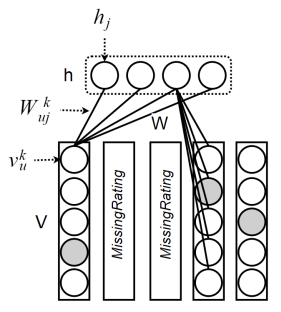
RRN(Recurrent Recommender Network)

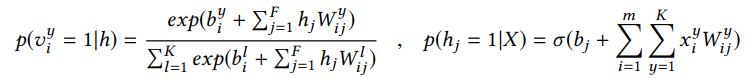
 

使用两条LSTM对user以及item在时间t下的行为进行建模。

3.Feature Representation Learning with RNNs

**·限制玻尔兹曼机RBM**





X为K×m矩阵，K为分数域， m为电影数。

**·神经自回归分布估计器Neural Autoregressive Distribution Estimator, NADE**

**·注意力神经元AN**

**·生成对抗网络GAN**

**·深度强化学习DRL**

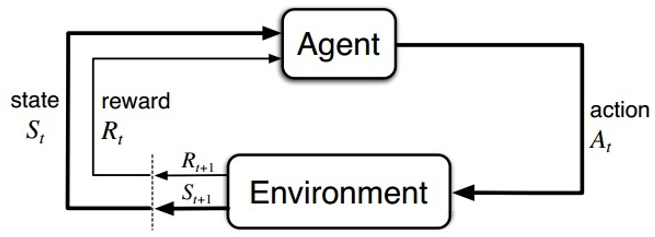
**·混合模型HM，以上两种或以上组合**

**强化学习**

1.基本介绍

机器处在某个环境中，对当前环境的感知为当前状态；机器（或agent智能体）可以通过动作来影响环境，即当机器对环境执行某个动作后，环境会按照某个概率转移到另一个状态，同时环境会根据潜在的奖赏函数反馈给机器对应动作的奖罚。

以上，比较重要的是状态、动作、转移概率以及奖赏函数。通过一系列动作策略与环境交互，产生新的数据，再利用新的数据去修改自身的动作策略，经过数次迭代后，agent就会学习到完成任务所需要的动作策略。



2.强化学习特点

无特定数据，只有奖励信号

奖励信号不一定实时

主要研究时间序列信号，而非独立同分布数据

当前行为会对后续数据产生影响

3.相关概念

（1）Agent智能体

主要涉及到：策略（Policy），价值函数（Value Function）和模型（Model）。

Policy：可以理解为行动指南，让agent执行什么动作，在数学上可以理解为从状态state到动作action的映射，可分为确定性策略（Deterministic policy）和随机性策略（Stochastic policy），前者是指在某特定状态下执行某个特定动作，后者是根据概率来执行某个动作。

Value Function：对未来总Reward的一个预测。

Model：一个对环境的认知框架，可以预测采取动作后的下一个状态是什么，很多情况下是没有模型的，Agent只能通过与环境互动来提升策略。

（2）State状态

可以细分为三种，Environment State，Agent State和Information State。

Environment State：Agent所处环境包含的信息，简单理解就是很多特征数据，也包含了无用的数据。

Agent State：输入给Agent的信息，也就是特征数据。

Information State：当前状态包含了对未来预测所需要的有用信息，过去信息对未来预测不重要，该状态就满足马尔科夫性（Markov Property）。

Environment State，Agent State都可以是满足Markov Property的。

（3）Environment环境

可以分为完全可观测环境（Fully Observable Environment）和部分可观测环境（Partially Observable Environment）。

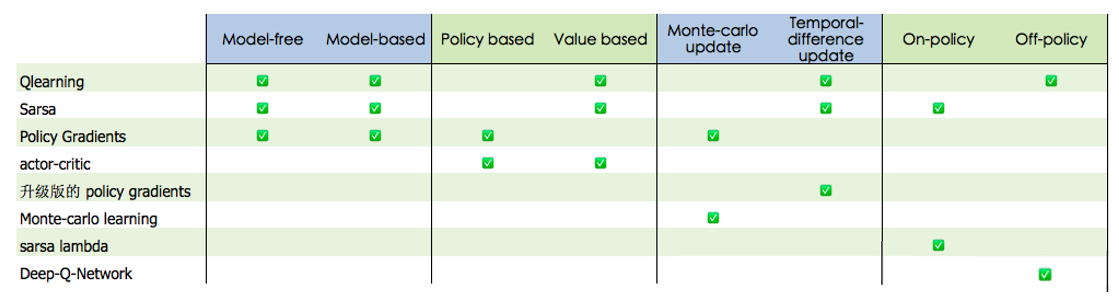
Fully Observable Environment：Agent了解了整个环境，显然是一个理想情况。

Partially Observable Environment：Agent了解部分环境的情况，剩下的需要靠agent去探索。

（4）关算法

按照Agent分类，可以分为：关注最优策略（Policy based）、关注最优奖励总和（Value based）、关注每一步的最优行动（Action based）

其他分类情况如下



（4）强化学习要素

第一个是环境的状态S

t时刻环境的状态St是它的环境状态集中某一个状态。

第二个是个体的动作A

t时刻个体采取的动作At是它的动作集中某一个动作。

第三个是环境的奖励R

t时刻个体在状态St采取的动作At对应的奖励Rt+1会在t+1时刻得到。

第四个是个体的策略(policy)π

它代表个体采取动作的依据，即个体会依据策略π来选择动作。最常见的策略表达方式是一个条件概率分布π(a|s), 即在状态s时采取动作a的概率。即π(a|s)=P(At=a|St=s).此时概率大的动作被个体选择的概率较高。

第五个是个体在策略π和状态s时，采取行动后的价值（value），一般用vπ(s)表示。

这个价值一般是一个期望函数。虽然当前动作会给一个延时奖励Rt+1,但是光看这个延时奖励是不行的，因为当前的延时奖励高，不代表到了t+1,t+2,...时刻的后续奖励也高。我们的价值要综合考虑当前的延时奖励和后续的延时奖励。价值函数vπ(s)一般可以表示为下式，不同的算法会有对应的一些价值函数变种，但思路相同：

vπ(s)=Eπ(Rt+1+γRt+2+γ2Rt+3+...|St=s)

其中γ是第六个模型要素，即奖励衰减因子，在[0，1]之间

如果为0，则是贪婪法，即价值只由当前延时奖励决定，如果是1，则所有的后续状态奖励和当前奖励一视同仁。大多数时候，我们会取一个0到1之间的数字，即当前延时奖励的权重比后续奖励的权重大。

第七个是环境的状态转化模型

可以理解为一个概率状态机，它可以表示为一个概率模型，即在状态s下采取动作a,转到下一个状态s′的概率，表示为Pass′。

第八个是探索率ϵ

这个比率主要用在强化学习训练迭代过程中，由于我们一般会选择使当前轮迭代价值最大的动作，但是这会导致一些较好的但我们没有执行过的动作被错过。因此我们在训练选择最优动作时，会有一定的概率ϵ不选择使当前轮迭代价值最大的动作，而选择其他的动作。

4. 马尔科夫决策过程（Markov Decision Process，MDP）

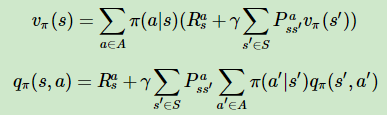
 => 



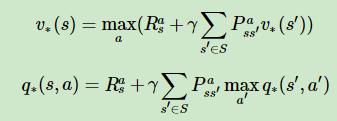
贝尔曼方程将当前状态下的价值分解为当前状态的奖励以及后一个状态的价值，动作价值也可以进行类似的分解。

进一步，价值可由动作价值进行求期望得到，反过来，利用可以得到动作价值的价值表达。

因此，可以得到



为了选取最优的策略，为了使得状态价值最大，只需，此时递推式变成



利用上面的公式可以对一个链式过程找出最优策略。

5.动态规划DP求解强化学习的预测以及控制问题

强化学习的两个基本问题：第一个问题是预测，即给定强化学习的6个要素：状态集S，动作集A，模型状态转化概率矩阵P，即时奖励R，衰减因子γ, 给定策略π，求解该策略的状态价值函数v(π)；第二个问题是控制，也就是求解最优的价值函数和策略。给定强化学习的5个要素：状态集S，动作集A，模型状态转化概率矩阵P，即时奖励R，衰减因子γ，求解最优的状态价值函数v∗和最优策略π∗。

（1）预测问题

策略评估法(Policy Evaluation)

策略评估的基本思路是从任意一个状态价值函数开始，依据给定的策略，结合贝尔曼期望方程、状态转移概率和奖励同步迭代更新状态价值函数，直至其收敛，得到该策略下最终的状态价值函数。

，该式子里，价值函数v含有下标k，代表轮数。收敛意味着两边的值应该很接近，因此说明价值函数到达真值。

（2）控制问题

策略迭代法(Policy Iteration)

根据我们之前基于任意一个给定策略评估得到的状态价值来及时调整我们的动作策略。

在策略迭代过程中，我们循环进行两部分工作。第一步是使用当前策略π∗评估计算当前策略的最终状态价值v∗，第二步是根据状态价值v∗根据一定的方法（比如贪婪法）来更新策略π∗，接着回到第一步，一直迭代下去，最终得到收敛的策略π∗和状态价值v∗。



异步动态规划算法：每一次迭代依据一定的原则有选择性的更新部分状态的价值

第一种是原位动态规划 (in-place dynamic programming)， 此时我们不会另外保存一份上一轮计算出的状态价值。而是即时计算即时更新。这样可以减少保存的状态价值的数量，节约内存。代价是计算以及收敛速度可能稍慢。

第二种是优先级动态规划 (prioritised sweeping)：该算法对每一个状态进行优先级分级，优先级越高的状态其状态价值优先得到更新。通常使用贝尔曼误差来评估状态的优先级，贝尔曼误差即新状态价值与前次计算得到的状态价值差的绝对值。这样可以加快计算以及收敛速度，代价是需要维护一个优先级队列。

第三种是实时动态规划 (real-time dynamic programming)：实时动态规划直接使用个体与环境交互产生的实际经历来更新状态价值，对于那些个体实际经历过的状态进行价值更新。这样个体经常访问过的状态将得到较高频次的价值更新，而与个体关系不密切、个体较少访问到的状态其价值得到更新的机会就较少。收敛速度可能稍慢。

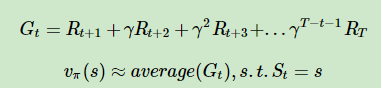
6.MC求解强化学习

模型状态转化概率矩阵P始终是已知的，即MDP已知，一般称这种强化学习问题为基于模型的强化学习问题。若P未知，则为不基于模型的强化学习问题。

蒙特卡罗法通过采样若干经历完整的状态序列(episode)来估计状态的真实价值。有了很多组这样经历完整的状态序列，我们就可以来近似的估计状态价值，进而求解预测和控制问题了。

（1）预测问题

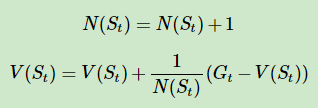
对于蒙特卡罗法来说，如果要求某一个状态的状态价值，只需要求出所有的完整序列中该状态出现时候的收获再取平均值即可近似求解。



优化：

a.若同一状态在一条完整的序列中多次出现则可以只取第一次出现时的情况或每次都纳入计算，分别称为首次访问（first visit）和每次访问（every visit）蒙特卡罗法。

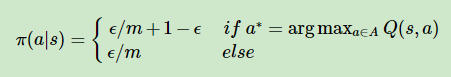
b. 累进更新平均值（incremental mean），即为了新出现一个状态时可以基于前一步计算得到的均值计算得到，加快计算。



当n特别大的时候，可以使用alpha代替下面的1/N，类似于滑动平均法，keras的moving\_average\_update函数会用到。

（2）控制问题

蒙特卡罗法求解控制问题的思路和动态规划价值迭代的的思路类似。蒙特卡罗法一般是优化最优动作价值函数q∗，而不是状态价值函数v∗，因此在MC法中不需要算v,而是计算q，一般采用ϵ−贪婪法更新，通过设置一个较小的ϵ值，使用1−ϵ的概率贪婪地选择目前认为是最大行为价值的行为，而用ϵ 的概率随机的从所有m 个可选行为中选择行为，代表了EE问题。

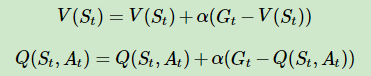


7.时序差分（TD）求解

若无法获得完整的状态序列，需要TD法进行求解。

利用贝尔曼方程可以得到收获Gt的近似值，称为TD目标值，而为TD误差，该近似过程称为引导过程。

（1）预测问题



迭代V的式子跟MC类似，只是使用alpha代替了1/N，同时G的计算也不一样。在知道结果之前就可以学习，也可以在没有结果时学习，还可以在持续进行的环境中学习。

（2）控制问题

包括在线控制算法（on-policy）：SARSA算法；离线控制算法（off-policy）：Q-Learning算法。

n步时序差分收获



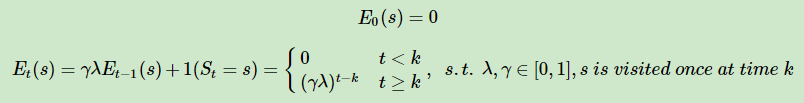
TD(lambda)

综合考虑所有步长的时序差分收获：

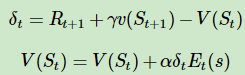
频率启发(frequency heuristic) 式；归因于发生次数最多状态的影响

就近启发(recency heuristic) 式：归因于最近少数几次状态的影响

效用(eligibility, E)：表示该状态对后续状态的影响，能够同时利用到上述两个启发。所有状态的效用值总称为效用迹(eligibility traces,ES)。



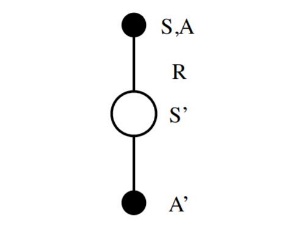
TD(lambda)正向迭代公式：

TD(lambda)反向迭代公式：

上下两条迭代公式是一致的，这里的等价需要在每一步更新时计算所有在序列中出现过的状态和动作下的价值函数，而不是只算一个，因此s代表遍历。

8.SARSA算法

S,A,R分别代表状态（State），动作(Action),奖励(Reward)，SARSA代表下图过程：先基于ϵ−贪婪法在当前状态S选择一个动作A，这样系统会转到一个新的状态S′, 同时给我们一个即时奖励R, 在新的状态S′，我们会基于ϵ−贪婪法在状态S‘′选择一个动作A′，但是注意这时候我们并不执行这个动作A′，只是用来更新的我们的价值函数，价值函数的更新公式是



算法流程

算法输入：迭代轮数T，状态集S, 动作集A, 步长α，衰减因子γ, 探索率ϵ

输出：所有的状态和动作对应的价值Q

1.随机初始化所有的状态和动作对应的价值Q. 对于终止状态其Q值初始化为0.

2. for i from 1 to T，进行迭代。

a) 初始化S为当前状态序列的第一个状态。设置A为ϵ−贪婪法在当前状态S选择的动作。

b) 在状态S执行当前动作A,得到新状态S′和奖励R

　c) 用ϵ−贪婪法在状态S′选择新的动作A′

　d) 更新价值函数Q(S,A):

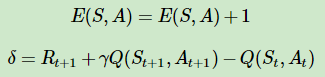
　e) S=S′,A=A′

f) 如果S′是终止状态，当前轮迭代完毕，否则转到步骤b)

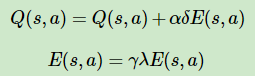
SARSA(lambda)

即利用前面的TD(lambda)来更新动作价值函数，引入多步参数λ

(d) => 更新效用迹函数E(S,A)和TD误差δ



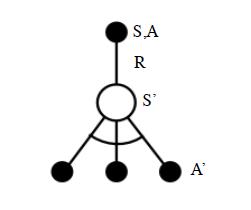
(e) =>对当前序列所有出现的状态s和对应动作a, 更新价值函数Q(s,a)和效用迹函数E(s,a)



缺陷：在状态或者动作百万级别时，Q表的存储很困难。

9.Q-Learning算法

对于Q-Learning，我们会使用ϵ−贪婪法来选择新的动作，这部分和SARSA完全相同。但是对于价值函数的更新，Q-Learning使用的是贪婪法，而不是SARSA的ϵ−贪婪法。这一点就是SARSA和Q-Learning本质的区别。



对于Q-Learning，它基于状态S′，没有使用ϵ−贪婪法选择A′，而是使用贪婪法选择A′，也就是说，选择使Q(S′,a)最大的a作为A′来更新价值函数



对应到上图中就是在图下方的三个黑圆圈动作中选择一个使Q(S′,a)最大的动作作为A′。

此时选择的动作只会参与价值函数的更新，不会真正的执行。价值函数更新后，新的执行动作需要基于状态S′，用ϵ−贪婪法重新选择得到。这一点也和SARSA稍有不同。

QL算法流程

算法输入：迭代轮数T，状态集S, 动作集A, 步长α，衰减因子γ, 探索率ϵ,

输出：所有的状态和动作对应的价值Q

1. 随机初始化所有的状态和动作对应的价值Q. 对于终止状态其Q值初始化为0.

2. for i from 1 to T，进行迭代。

a) 初始化S为当前状态序列的第一个状态。

b) 用ϵ−贪婪法在当前状态S选择出动作A

　c) 在状态S执行当前动作A,得到新状态S′和奖励R

　d) 更新价值函数Q(S,A):

　e) S=S′

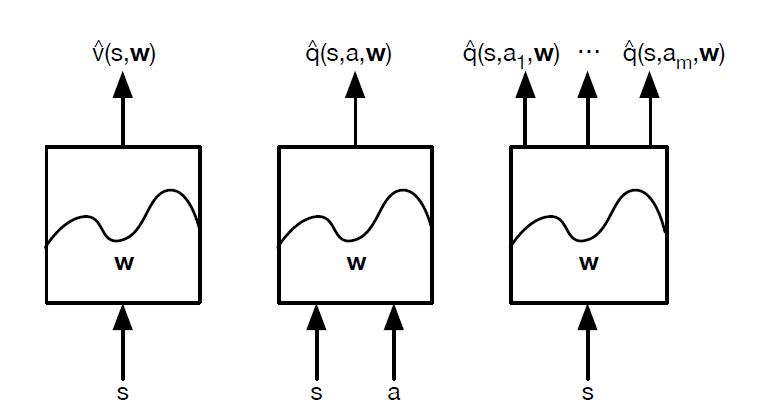
　f) 如果S′是终止状态，当前轮迭代完毕，否则转到步骤b)

PS. QL没有多步长算法是因为每一轮迭代中，它采用贪婪法选取下一状态动作使得n步时序差分收获都是同一个值，因此多步没有效果。

10.Deep QL

针对状态空间巨大（如连续状态空间）导致存储的Q表无法在内存表示的问题，提出Deep算法，本质上是对价值函数进行近似表示。

含参w的近似状态价值以及状态动作价值函数的引入：，。对应的例子可以是线性表示法，φ是状态s的特征向量。进一步，可以用决策树，最近邻，傅里叶变换，神经网络来表达我们的状态价值函数，对于NN，我们可以采用DNN、CNN、RNN来建模。对于状态价值函数，神经网络的输入是状态s的特征向量，输出是状态价值v^(s,w)；对于动作价值函数，有两种方法，一种是输入状态s的特征向量和动作a，输出对应的动作价值q^(s,a,w)，另一种是只输入状态s的特征向量，动作集合有多少个动作就有多少个输出q^(s,ai,w)。这里隐含了我们的动作是有限个的离散动作。



DQN（深度Q-Learning）输入状态对应的状态向量，输出对应的动作价值函数。由于Q表已经不存在了，因此，DQN主要使用的技巧是经验回放（experience replay）,即将每次和环境交互得到的奖励与状态更新情况都保存起来，用于后面目标Q值的更新。

基于NIPS 2013 DQN的算法流程

算法输入：迭代轮数T，状态特征维度n，动作集A，步长α，衰减因子γ，探索率ϵ，Q网络结构，批量梯度下降的样本数m。

输出：Q网络参数

1.随机初始化Q网络的所有参数w，基于w初始化所有的状态和动作对应的价值Q，清空经验回放的集合D

2.for i from 1 to T，进行迭代

a) 初始化S为当前状态序列的第一个状态, 拿到其特征向量ϕ(S)

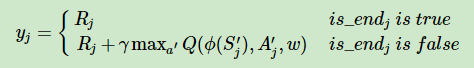
　b) 在Q网络中使用ϕ(S)作为输入，得到Q网络的所有动作对应的Q值输出，用ϵ−贪婪法在当前Q值输出中选择对应的动作A

　c) 在状态S执行当前动作A,得到新状态S′对应的特征向量ϕ(S′)和奖励R$，是否终止状态is\_end

d) 将{ϕ(S),A,R,ϕ(S′),is\_end}这个五元组存入经验回放集合D

　e) S=S′

　f) 从经验回放集合D中采样m个样本{ϕ(Sj),Aj,Rj,ϕ(S′j),is\_endj},j=1,2.,,,m，计算当前目标Q值yj：



g) 使用均方差损失函数，通过神经网络的梯度反向传播来更新Q网络的所有参数w

h) 如果S′是终止状态，当前轮迭代完毕，否则转到步骤b)

注意，上述第二步的f步和g步的Q值计算也都需要通过Q网络计算得到。另外，实际应用中，为了算法较好的收敛，探索率ϵ需要随着迭代的进行而变小。

11.DQN改进：Nature DQN(NIPS 2015)

基本DQN在计算目标yj的时候需要用到当前的网络，但是yj又会用于网络的更新导致目标计算和参数更新相互依赖，独立性过强，不利于算法收敛，NDQN则是引入两个Q-Net来减轻这种依赖关系。

Nature DQN使用了两个Q网络，一个当前Q网络Q用来选择动作，更新模型参数，另一个目标Q网络Q′用于计算目标Q值。目标Q网络的网络参数不需要迭代更新，而是每隔一段时间从当前Q网络Q复制过来，即延时更新，目标Q网络参数更新频率C来控制，这样可以减少目标Q值和当前的Q值相关性。

该模型算法流程与DQN类似，只是在计算y的时候使用目标Q-Net。

12.其他改进版本

a改进目标Q值的计算方式：Double DQN

b改进经验回放集合的采样方式：Prioritised Replay DQN

c 动作价值的引入：Dueling DQN

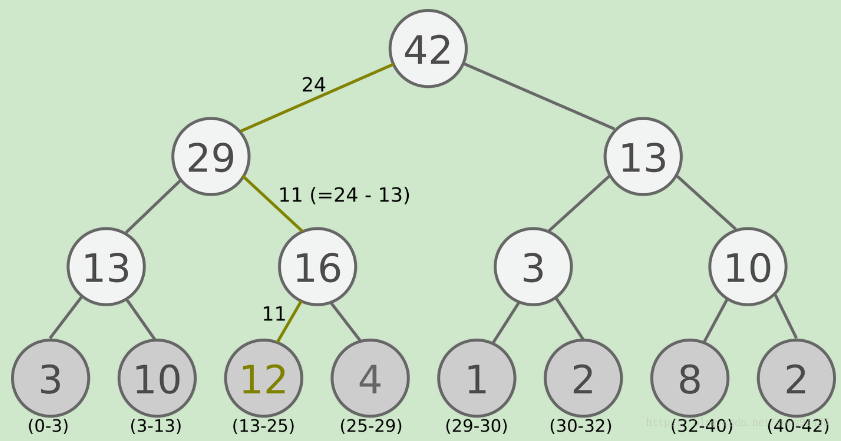
Double DQN

每次计算y的时候选择max的方向容易过拟合，导致过度估计，为此，DDQN通过解耦目标Q值动作的选择和目标Q值的计算这两步，来达到消除过度估计的问题。

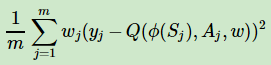
DDQN沿用NDQN的双网络结构，但在计算y的时候，先在当前Q网络中先找出最大Q值对应的动作：，再计算。

Prioritised Replay DQN

根据每个样本的TD误差绝对值|δ(t)|，给定该样本的优先级正比于|δ(t)|，将这个优先级的值存入经验回放池。因为这个优先级大小会影响它被采样的概率。在实际使用中，我们通常使用SumTree这样的二叉树结构来做我们的带优先级的经验回放池样本的存储。



ST是一棵完全二叉树，所有的经验回放样本只保存在最下面的叶子节点上面，内部节点不保存样本数据。而叶子节点除了保存数据以外，还要保存该样本的优先级，就是图中的显示的数字。对于内部节点每个节点只保存自己的儿子节点的优先级值之和，如图中内部节点上显示的数字。该结构的优点是方便采样。若是多个样本的采样，如每轮minibatch的规模是k，那么把区间分成k段，然后每段均匀采样即可。

损失函数也根据采样进行了调整：，其中w由优先级归一化得到。

PR DDQN算法流程

算法输入：迭代轮数T，状态特征维度n, 动作集A，步长α，采样权重系数β，衰减因子γ，探索率ϵ，当前Q网络Q，目标Q网络Q′，批量梯度下降的样本数m，目标Q网络参数更新频率C，SumTree的叶子节点数S。

输出：Q网络参数。

1. 随机初始化所有的状态和动作对应的价值Q. 随机初始化当前Q网络的所有参数w,初始化目标Q网络Q′的参数w′=w。初始化经验回放SumTree的默认数据结构，所有SumTree的S个叶子节点的优先级pj为1。

2. for i from 1 to T，进行迭代。

a) 初始化S为当前状态序列的第一个状态, 拿到其特征向量ϕ(S)

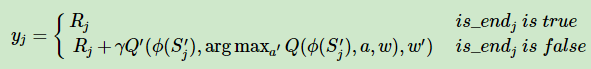
　b) 在Q网络中使用ϕ(S)作为输入，得到Q网络的所有动作对应的Q值输出。用ϵ−贪婪法在当前Q值输出中选择对应的动作A

c) 在状态S执行当前动作A,得到新状态S′对应的特征向量ϕ(S′)和奖励R,是否终止状态is\_end

d) 将{ϕ(S),A,R,ϕ(S′),is\_end}这个五元组存入SumTree

e) S=S′

　f) 从SumTree中采样m个样本{ϕ(Sj),Aj,Rj,ϕ(S′j),is\_endj},j=1,2.,,,m，每个样本被采样的概率基于，损失函数权重，计算当前目标Q值yj：



　g) 使用均方差损失函数，通过神经网络的梯度反向传播来更新Q网络的所有参数w

　h) 重新计算所有样本的TD误差δj，更新SumTree中所有节点的优先级pj=|δj|

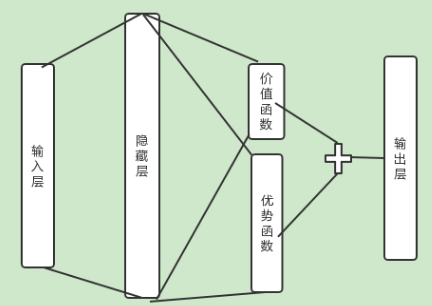
　i) 如果T%C=1,则更新目标Q网络参数w′=w

　j) 如果S′是终止状态，当前轮迭代完毕，否则转到步骤b)

Dueling DQN

Dueling DQN考虑将Q网络分成两部分，第一部分是仅仅与状态S有关，与具体要采用的动作A无关，这部分我们叫做价值函数部分，记做V(S,w,α)，第二部分同时与状态状态S和动作A有关，这部分叫做优势函数(Advantage Function)部分，记为A(S,A,w,β)，那么最终我们的价值函数可以重新表示为：，其中w是共有网络参数，可以理解为特征提取层，而alpha和beta是各自对应的私有参数，可以看做是对特征进行不同的处理，实际上会对A进行中心化处理：



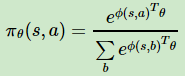


13.策略梯度：Policy-based RL

前面所有的算法都是基于价值函数的方法，可以称之为value-based，应用十分广泛，但也有对应的缺点：a.对连续动作处理能力不足；b.对受限状态问题处理能力不足；c.无法解决随机策略问题。为此，引入policy-based RL，对策略进行近似表示：。

使用梯度上升法来优化策略，最简单的优化目标是初始状态收获的期望：；若初始状态不明确，可以采用平均价值来定义：，d为状态的静态分布；也可以定义为第一时间步的平均奖励：，则梯度可以表示为

，构成：E代表各个状态s下的价值期望，里面的式子是对某一动作具体的价值（里面没有再取期望是因为实际例子中动作只会有一个）。若采用其他优化目标，只会改变梯度的Q部分，前面的log部分称为分值函数（score function）。

针对策略函数，最常用的是softmax策略：，φ为特征函数，对应的分值函数的导数部分为；除此以外，在连续行为空间里，常用高斯分布产生策略函数：，求导得到。

最简单的策略梯度rl算法是蒙特卡洛策略，使用价值函数v近似代替策略梯度中的Q。算法流程为：

输入：N个蒙特卡罗完整序列，训练步长α

输出：策略函数的参数θ

1. for 每个蒙特卡罗序列:

a. 用蒙特卡罗法计算序列每个时间位置t的状态价值vt

　b. 对序列每个时间位置t，使用梯度上升法，更新策略函数的参数θ：



2. 返回策略函数的参数θ

14.Actor-Critic：Policy-based + Value-based

Actor-Critic从名字上看包括两部分，演员(Actor)和评价者(Critic)。其中Actor使用我们上一节讲到的策略函数，负责生成动作(Action)并和环境交互。而Critic使用我们之前讲到了的价值函数，负责评估Actor的表现，并指导Actor下一阶段的动作。前面的蒙特卡洛法没有一般的C部分而且需要完整序列，因此应用受限。

而在一般的A-C算法中，需要两组近似：策略函数和价值函数。对应上一部分的梯度更新部分，一般情况下，vt部分需要从C得到。因此，总的来说，Critic通过Q网络计算状态的最优价值vt，而Actor利用vt这个最优价值迭代更新策略函数的参数θ，进而选择动作，并得到反馈和新的状态，Critic使用反馈和新的状态更新Q网络参数w，在后面Critic会使用新的网络参数w来帮Actor计算状态的最优价值vt。

根据C部分，可以对A-C算法按照C评估点进行分类：a.基于状态价值：；b.基于动作价值：；c.基于TD误差：误差可以由状态价值或状态动作价值求解：；d.基于优势函数：，DDQN的A函数，即动作状态价值与状态价值的差值；e.基于TD(LAMBDA)误差：。

A-C算法流程

算法输入：迭代轮数T，状态特征维度n，动作集A，步长α,β，衰减因子γ，探索率ϵ，Critic网络结构和Actor网络结构。

输出：Actor 网络参数θ，Critic网络参数w

1. 随机初始化所有的状态和动作对应的价值Q

2. for i from 1 to T，进行迭代。

a) 初始化S为当前状态序列的第一个状态, 拿到其特征向量ϕ(S)

　b) 在Actor网络中使用ϕ(S)作为输入，输出动作A,基于动作A得到新的状态S′,反馈R。

c) 在Critic网络中分别使用ϕ(S)，ϕ(S‘′)作为输入，得到Q值输出V(S)，V(S′)

　d) 计算TD误差

　e) 使用均方差损失函数作Critic网络参数w的梯度更新

f) 更新Actor网络参数θ：

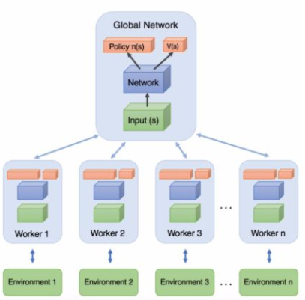
该简单版本的A-C算法难以收敛，需要进一步的改进。

15.A3C：Asynchronous Advantage Actor-Critic

A3C利用多线程的方法，同时在多个线程里面分别和环境进行交互学习，每个线程都把学习的成果汇总起来，整理保存在一个公共的地方。并且，定期从公共的地方把大家的齐心学习的成果拿回来，指导自己和环境后面的学习交互。通过这种方法，A3C避免了经验回放相关性过强的问题，同时做到了异步并发的学习模型。

优化点：a.异步训练框架；b.网络结构优化；c.C评估点优化。

a.



Global Network就是上一节说的共享的公共部分，主要是一个公共的神经网络模型，这个神经网络包括Actor网络和Critic网络两部分的功能。下面有n个worker线程，每个线程里有和公共的神经网络一样的网络结构，每个线程会独立的和环境进行交互得到经验数据，这些线程之间互不干扰，独立运行。

每个线程和环境交互到一定量的数据后，就计算在自己线程里的神经网络损失函数的梯度，但是这些梯度却并不更新自己线程里的神经网络，而是去更新公共的神经网络。也就是n个线程会独立的使用累积的梯度分别更新公共部分的神经网络模型参数。每隔一段时间，线程会将自己的神经网络的参数更新为公共神经网络的参数，进而指导后面的环境交互。

b.

在A3C这里，我们把两个网络放到了一起，即输入状态S，可以输出状态价值V,和对应的策略π，除此以外，还可以把Actor和Critic看做独立的两块，分别处理，与上一节一致。

c.

在A3C中，使用了N步采样，以加速收敛，因此A3C中使用的优势函数表达为：

；除此以外，A3C的策略损失函数加入了策略的熵项，系数为c，使得梯度更新为。

A3C算法流程

输入：公共部分的A3C神经网络结构，对应参数位θ，w，本线程的A3C神经网络结构，对应参数θ′，w′，全局共享的迭代轮数T，全局最大迭代次数Tmax，线程内单次迭代时间序列最大长度Tlocal，状态特征维度n，动作集A，步长α,β，熵系数c，衰减因子γ，探索率ϵ

输入：公共部分的A3C神经网络参数θ,w

1. 更新时间序列t=1

2. 重置Actor和Critic的梯度更新量:dθ←0,dw←0

3. 从公共部分的A3C神经网络同步参数到本线程的神经网络：θ′=θ,w′=w

4. tstart=t，初始化状态st

5. 基于策略π(at|st;θ)选择出动作at

6. 执行动作at得到奖励rt和新状态st+1

7. t←t+1,T←T+1

8. 如果st是终止状态，或t−tstart==tlocal,则进入步骤9，否则回到步骤5

9. 计算最后一个时间序列位置st的Q(s,t):



10. for i∈(t−1,t−2,...tstart):

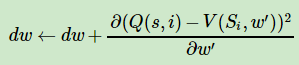
1) 计算每个时刻的Q(s,i)：



　2) 累计Actor的本地梯度更新：



　3) 累计Critic的本地梯度更新：



11. 更新全局神经网络的模型参数：



12. 如果T>Tmax,则算法结束，输出公共部分的A3C神经网络参数θ,w，否则进入步骤3

16.深度确定性策略梯度（DDPG）

确定性策略是和随机策略相对而言的，作为随机策略，在相同的策略，在同一个状态处，采用的动作是基于一个概率分布的，即是不确定的。而确定性策略则决定简单点，在同一个状态处，我们只取最大概率的动作，去掉这个概率分布，即作为确定性策略，相同的策略，在同一个状态处，动作是唯一确定的，即策略变成。

跟随机性策略梯度相比，确定性策略梯度计算公式从变成

，少了对动作的积分，多了Q函数对a的求导。

DDPG使用了经验回放，双网络的trick。

DDPG 4个网络的功能定位：

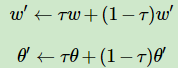
1. Actor当前网络：负责策略网络参数θ的迭代更新，负责根据当前状态S选择当前动作A，用于和环境交互生成S′,R。

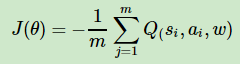
2. Actor目标网络：负责根据经验回放池中采样的下一状态S′选择最优下一动作A′。网络参数θ′定期从θ复制。

3. Critic当前网络：负责价值网络参数w的迭代更新，负责计算负责计算当前Q值Q(S,A,w)。目标Q值

4. Critic目标网络：负责计算目标Q值中的Q′(S′,A′,w′)部分。网络参数w′定期从w复制。

在复制过程，DDPG没有使用简单的完全复制法，而是使用软更新，即部分更新：

，r是比较小的数，如0.1，0.001等。

同时在学习过程中，对选择出来的动作A执行加噪声处理。对于C网络，其损失函数与前面一致：，但对于A网络，其策略梯度改变为，可以看做是对关于theta求导。

DDPG算法流程

输入：Actor当前网络，Actor目标网络，Critic当前网络，Critic目标网络，参数分别为θ，θ′，w，w′，衰减因子γ，软更新系数τ，批量梯度下降的样本数m，目标Q网络参数更新频率C，最大迭代次数T，随机噪音函数N

输出：最优Actor当前网络参数θ，Critic当前网络参数w

1. 随机初始化θ,w，w′=w,θ′=θ。清空经验回放的集合D

2. for i from 1 to T，进行迭代。

a) 初始化S为当前状态序列的第一个状态, 拿到其特征向量ϕ(S)

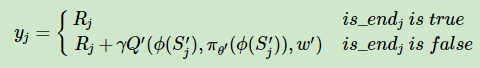
　b) 在Actor当前网络基于状态S得到动作

　c) 执行动作A,得到新状态S′,奖励R,是否终止状态is\_end

　d) 将{ϕ(S),A,R,ϕ(S′),is\_end}这个五元组存入经验回放集合D

　e) S=S'

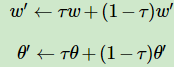
f) 从经验回放集合D中采样m个样本{ϕ(Sj),Aj,Rj,ϕ(S′j),is\_endj},j=1,2.,,,m，计算当前目标Q值yj：



　g) 使用均方差损失函数，通过神经网络的梯度反向传播来更新Critic当前网络的所有参数

　h) 使用A损失函数，通过神经网络的梯度反向传播来更新Actor当前网络的所有参数

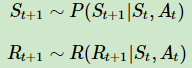
　i) 如果T%C=1,则更新Critic目标网络和Actor目标网络参数：



　j) 如果S′是终止状态，当前轮迭代完毕，否则转到步骤b)

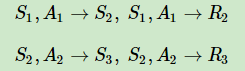
17.model-based强化学习：Dyna算法框架

基于模型的强化学习尝试从环境的模型去学习，一般是下面两个相互独立的模型：一个是状态转化预测模型，输入当前状态s和动作a，预测下一个状态s′。另一个是奖励预测模型，输入当前状态s和动作a，预测环境的奖励r。即模型可以描述为下面两个式子：

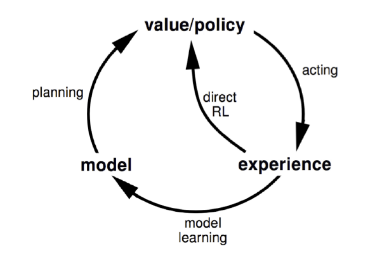


可以看出，基于模型的强化学习是从模型中学习，而不基于模型的强化学习是从和环境交互的经历去学习，即在当前状态s以及选取的动作下，是否需要跟环境交互产生下一状态以及当前延时奖励。

为了对模型进行学习，可以每组完整的状态链进行分解：

 => ，至此可以使用传统的有监督学习进行模型学习，也是使用简单的查表法进行估计。

由于上面的做法过于简单，不能有效解决复杂的情况，因此需要更好的解决方案：将基于模型的算法和非基于模型的算法进行结合，使用Dyna算法框架。下图中间线即非基于模型强化学习，另一条路径代表基于模型强化学习。



由于该框架可以有很多方法，以结合查表法和非深度Q-Learning的Dyna-Q算法的算法流程作为例子。

1. 初始化任意一个状态s，和任意一个动作a对应的状态价值Q(s,a)，初始化奖励模型R和状态模型P

2. for i=1 to 最大迭代次数T：

a) S ← current state

b) A ← ϵ−greedy(S,Q)

　c) 执行动作A,得到新状态S′和奖励R

d) 使用Q-Learning更新价值函数

　e) 使用S,A,S′更新状态模型P，使用S,A,R更新状态模型R

　f) for j=1 to 最大次数n：

i) 随机选择一个之前出现过的状态S, 在状态S上出现过的动作中随机选择一个动作A

　 ii) 基于模型P得到S′, 基于模型R得到R

　　iii) 使用Q-Learning更新价值函数

Dyna框架在每个迭代轮中，会先和环境交互，并更新价值函数和（或）策略函数，接着进行n次模型的预测，同样更新价值函数和（或）策略函数。这样同时利用上了和环境交互的经历以及模型的预测。

Dyna-2算法框架

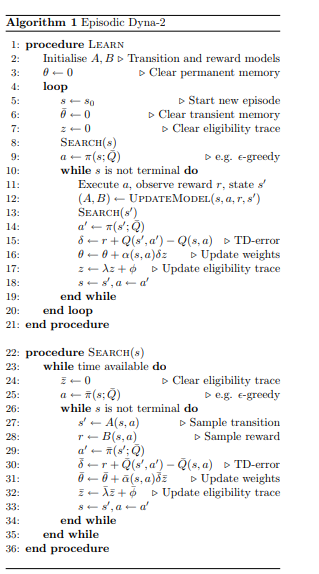
在Dyna算法框架的基础上后来又发展出了Dyna-2算法框架。和Dyna相比，Dyna-2将和环境交互的经历以及模型的预测这两部分使用进行了分离。还是以Q函数为例，Dyna-2将记忆分为永久性记忆（permanent memory）和瞬时记忆（transient memory）, 其中永久性记忆利用实际的经验来更新，瞬时记忆利用模型模拟经验来更新。

永久性记忆的Q函数：；瞬时记忆的Q函数：；组合记忆函数：



Dyna-2的基本思想是在选择实际的执行动作前，智能体先执行一遍从当前状态开始的基于模型的模拟，该模拟将仿真完整的轨迹，以便评估当前的动作值函数。智能体会根据模拟得到的动作值函数加上实际经验得到的值函数共同选择实际要执行的动作。

价值函数的更新方式类似于SARSA(λ)。



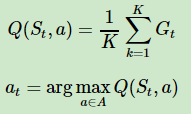
18.model-based + not model-based：基于模拟的搜索（Simulation-based Search）

基于模拟的搜索：主要是两个点：一个是模拟，一个是搜索。模拟就是基于强化学习模型进行采样，得到样本数据。搜索，则是为了利用模拟的样本结果来帮我们计算到底应该采用什么样的动作，以实现我们的长期受益最大化。

最简单的前向搜索法需要将所有的状态以及其转化通过树形结构进行MDP过程求解，得到对应的动作后再往下进行搜索，进行sub-MDP求解。在状态动作较多的情况下，这种方法效率很低，因此基于模拟的搜索可以较好解决这样的问题。

简单蒙特卡洛搜索

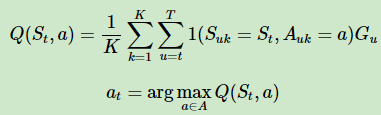
简单蒙特卡罗搜索基于一个强化学习模型M和一个模拟策略π。在此基础上，对于当前我们要选择动作的状态St，对每一个可能采样的动作a∈A，都进行K轮采样，这样每个动作a都会得到K组经历完整的状态序列(episode)，再基于蒙特卡罗法来计算其动作价值函数并选择最优的动作。



但是对于更大的状态动作空间，该方法效率还是太低了。

蒙特卡洛树搜索（Monte-Carlo Tree Search）

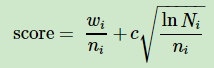
MCTS摒弃了简单蒙特卡罗搜索里面对当前状态St每个动作都要进行K次模拟采样的做法，而是总共对当前状态St进行K次采样。采样完毕后，我们可以基于采样的结果构建一颗MCTS的搜索树，然后近似计算Q(st,a)和最大Q(st,a)对应的动作。



MCTS搜索的策略分为两个阶段：第一个是树内策略(tree policy)：为当模拟采样得到的状态存在于当前的MCTS时使用的策略。树内策略可以使ϵ−贪婪策略，随着模拟的进行策略可以得到持续改善，还可以使用上限置信区间算法UCT，这在棋类游戏中很普遍；第二个是默认策略(default policy)：如果当前状态不在MCTS内，使用默认策略来完成整个状态序列的采样，并把当前状态纳入到搜索树中。默认策略可以使随机策略或基于目标价值函数的策略。

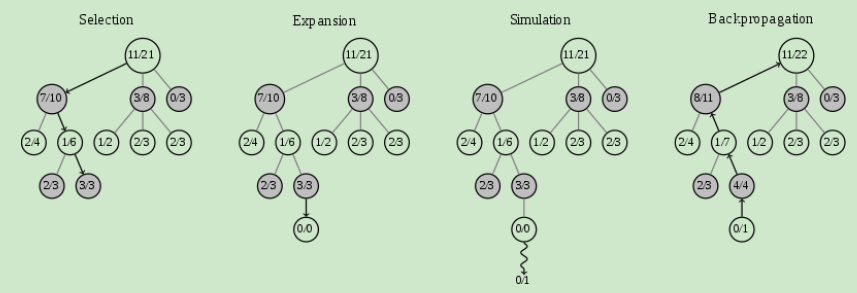
在某些不能及时知道中间状态的延时奖励的过程，如棋类博弈过程，需要对MCTS进行结构上的细化。

上限置信区间算法UCT

该算法常用于棋类游戏，为了在最优策略和探索度进行一个平衡，可以采用UCT进行策略的选择。UCT首先计算每一个可选动作节点对应的分数，这个分数考虑了历史最优策略和探索度：，c 越大就越偏向于广度搜索，c 越小就越偏向于深度搜索。

棋类游戏的MCTS搜索

对于MCTS的树结构，如果是最简单的方法，只需要在节点上保存状态对应的历史胜负记录。在每条边上保存采样的动作。这样MCTS的搜索需要走4步，注意根节点是白棋对应的数据，下面的路径是白棋选择的动作，往下是黑棋，以此循环。



第一步是选择(Selection):这一步会从根节点开始，每次都选一个“最值得搜索的子节点”，一般使用UCT选择分数最高的节点，直到来到一个“存在未扩展的子节点”的节点，如图中的 3/3 节点。之所以叫做“存在未扩展的子节点”，是因为这个局面存在未走过的后续着法，也就是MCTS中没有后续的动作可以参考了。这时我们进入第二步。

第二步是扩展(Expansion)，在这个搜索到的存在未扩展的子节点，加上一个0/0的子节点，表示没有历史记录参考。这时我们进入第三步。

第三步是仿真(simulation)，从上面这个没有试过的着法开始，用一个简单策略比如快速走子策略（Rollout policy）走到底，得到一个胜负结果。快速走子策略一般适合选择走子很快可能不是很精确的策略。因为如果这个策略走得慢，结果虽然会更准确，但由于耗时多了，在单位时间内的模拟次数就少了，所以不一定会棋力更强，有可能会更弱。这也是为什么我们一般只模拟一次，因为如果模拟多次，虽然更准确，但更慢。

第四步是回溯(back propagation), 将我们最后得到的胜负结果回溯加到MCTS树结构上。注意除了之前的MCTS树要回溯外，新加入的节点也要加上一次胜负历史记录，如上图最右边所示。

**基于强化学习的推荐算法**

1.强化学习在美团“猜你喜欢”的实践

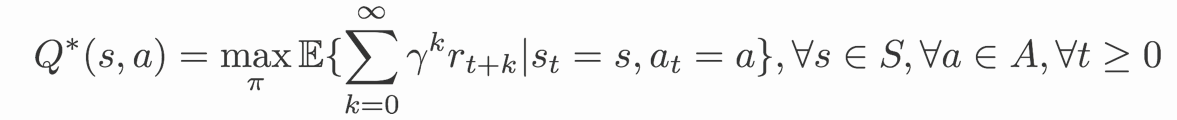
Agent：推荐系统

Environment：用户

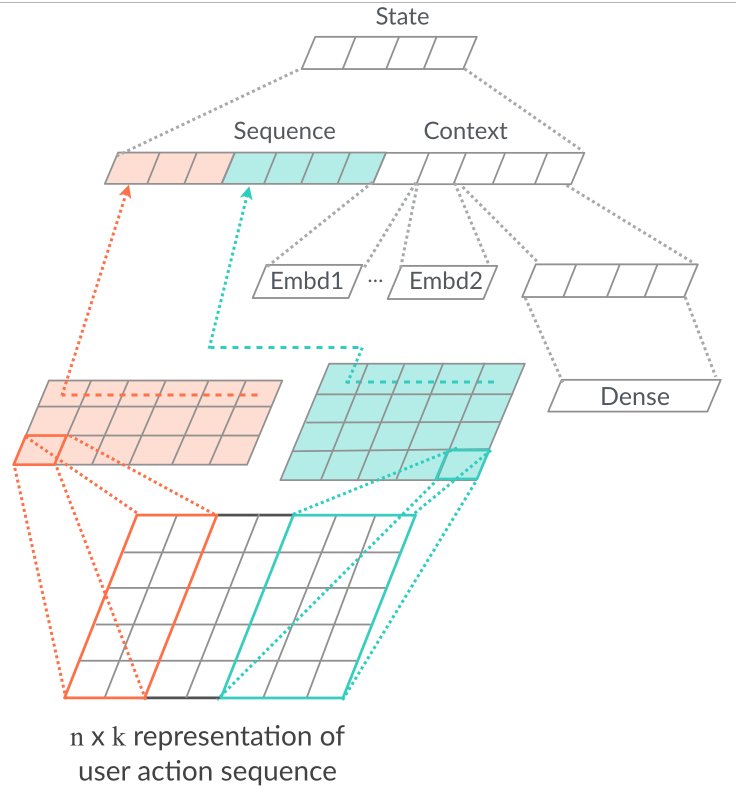
State：用户意图及推荐场景

Action：List-wise粒度对推荐列表做调整，考虑长期收益的影响

Reward：用户反馈对推荐系统的奖赏，根据业务指标指定

优化目标：

State建模，即φ



Sequence即对用户行为序列进行类文本建模，使用的是text-CNN结构，用于捕捉不同长度时间窗口内动作的关联性；Context是对非序列离散特征以及连续特征进行建模，使用dense网络。

动作设计，即对推荐结果进行排序

利用点击排序模型以及支付排序模型进行组合，两个排序模型均为wide&deep模型，组合形式为score=click\*(pay+faiz)，超参数faiz的物理意义是调整全量数据集中点击和下单模型的Trade Off，通过综合考虑点击和下单两个任务的AUC确定，没有个性化的因素。可以让动作a与faiz结合，变为faiz\*a。

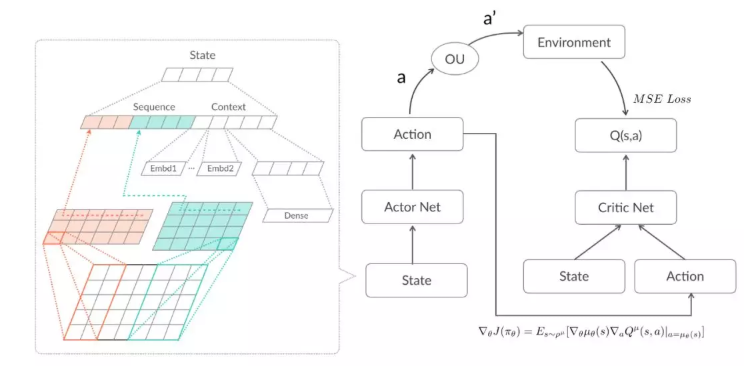
奖励计算

，引入惩罚项：对没有任何转化（点击或下单）行为的中间交互页面的惩罚1以及对没有发生任何转化且用户离开页面的惩罚2。



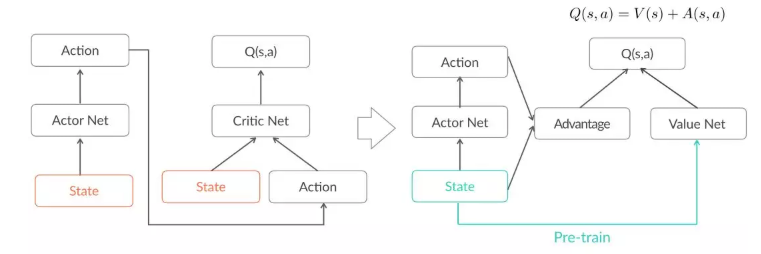
改进DDPG模型

基本结构如下



改进a：将C网络中的Q进行分解，类似Dual DQN，Q=V+A，实验发现主要由V做贡献，训练方式为先根据状态和收益训练V，再使用Q-V的残差训练A，发现很大程度上提升了训练稳定性。

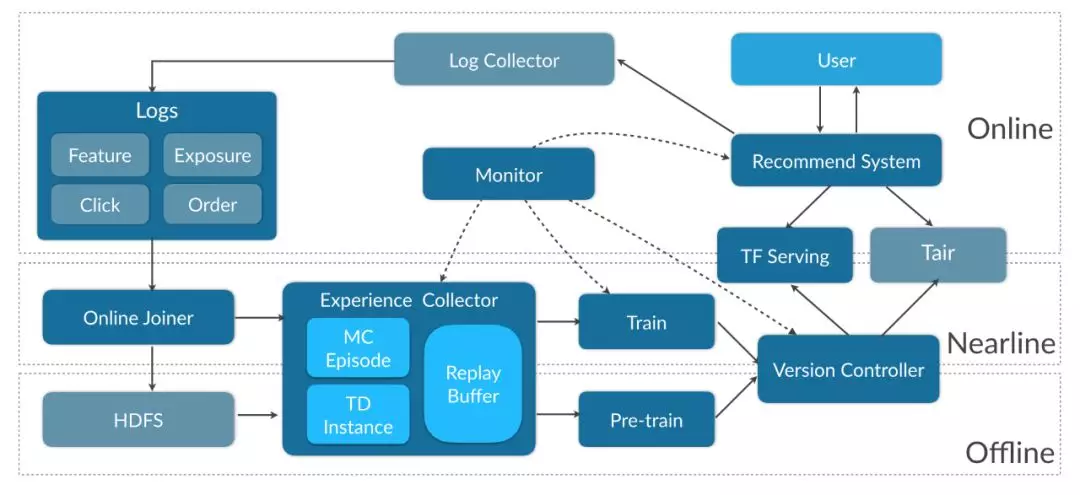
改进b：state权值共享



改进c：引入同步实现，摒弃异步实现，即A网络和C网络各只有一个网络。

改进d：多组并行策略，线上多组实验共享State表达和V(s)的估计，每个策略训练自己的A(s,a)网络且能快速收敛，这样的结构一方面使训练过程更加稳定，另一方面为强化学习策略全量提供了可能性。

基于TF的轻量级实时DRL系统



2.JD：Deep Reinforcement Learning for List-wise Recommendations

该论文描述了京东的基于深度强化学习的列表型（list-wise）推荐算法，里面主要涉及到DDPG网络以及他们自己提出的在线环境模拟策略。

在其推荐场景中，MDP的五要素定义为：

（a）状态空间S：st={s1,s2,…,sN}，用户的浏览记录；

（b）动作空间A：at={a1,a2,…,aK}，推荐系统推荐的K个物品

（c）奖励R：r(st, at)基于用户反馈产生的值，如点击、排序；

（d）转移概率：P(st+1|st,at)，相当于用户历史行为的改变

（e）衰减因子：用于衡量奖励的延时性

在线环境模拟策略

主要为了解决两个问题：

（a）离线训练时拓展用户MDP，加快离线训练效率

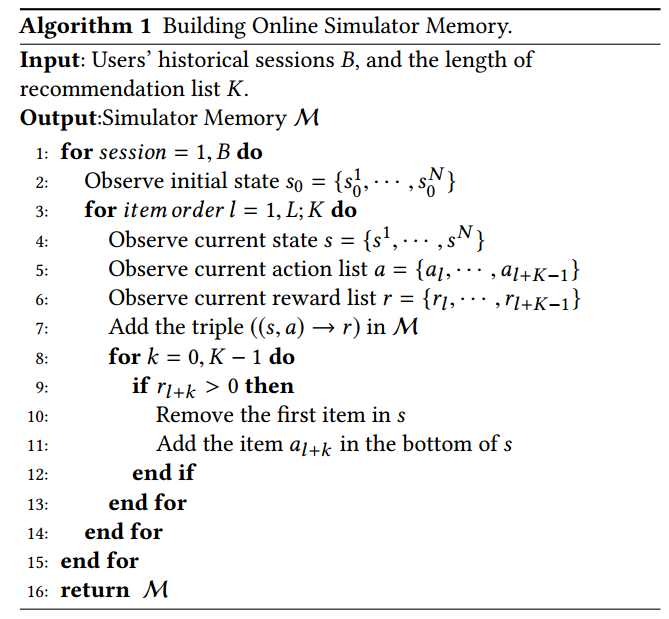
（b）item空间巨大，普通的离线数据不能包含所有的item，利用模拟环境可以缓解。

目标：f:(st, at) -> rt：即根据状态和动作预测奖励。

第一步是构建在线模拟记忆集合

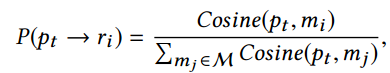
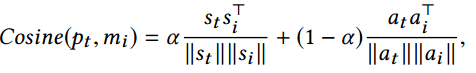
其形式为M={m1,m2,…}，mi=((si,ai)->ri)相当于一个映射关系，通过元组形式存储。构造方法即将用户历史行为状态以及推荐系统动作进行分解，通过滑动窗口进行M集合的构造。

关键点：i.奖励是针对推荐动作的，一个动作一个奖励；ii.存储的元组形式都是向量形式；iii.滑动的时候需要根据奖励以及动作不断更新状态。



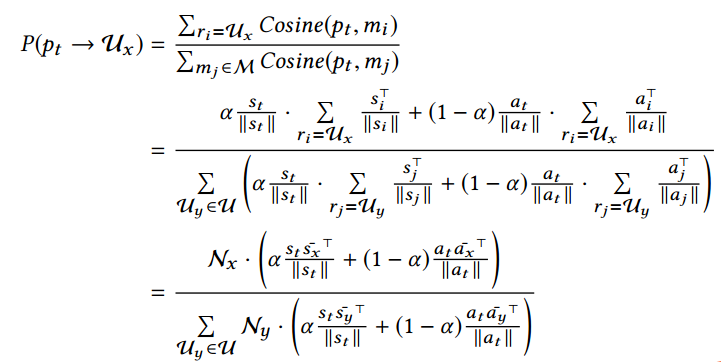
第二步时根据记忆集合推断未出现元组p(s,a)的奖励

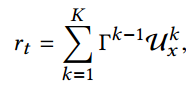
先通过cos相似度计算p与记忆中各个元素的相似度，再根据相似度计算概率



后续发现m集合太大，计算复杂度高，所以采用按照奖励分组计算概率。先对奖励向量r进行分组，若K=2，点击/不点击对应奖励值为1/0，那么分组

U={u1,u2,u3,u4}={(0,0),(0,1),(1,0),(1,1)}。因此可以通过以下式子计算新的概率值。



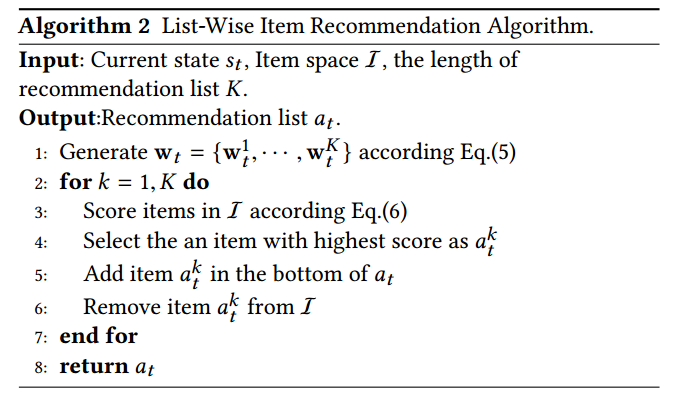
最后是计算整个推荐列表的分数，通过列表中各个item的概率的加权求和得到。

DDPG网络

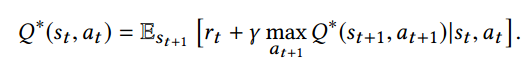
Actor网络

解决状态到动作的映射，分成两步：基于状态的分数生成以及动作生成。

第一步，使用深度网络进行计算，得到K维的weight向量；再使用內积计算各个动作得分，再排序取最大值。



Critic网络

解决对(s,a)的评价工作，通过延时奖励可以得到对应Q函数，但由于A网络的存在以及采用DDPG，因此at+1是确定的，且max可以去掉代替为Q即可。损失函数为且，采用双网络策略进行训练。

