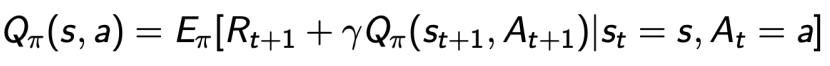
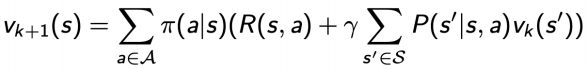
supervised learning vs RL: a.RL does not have dataset b. reward is used in RL c. Need exploration. Input: Sequential data (not i.i.d) **Policy function**: how action is selected • Stochastic policy: probability over actions (20% take LEFT, 80% take RIGHT)找global maximum更好 • Deterministic policy: take action LEFT. **Value function**: how good is each state of action

Model: state transition of the environmentBellman Expectation Equation:state-value function

 action-value function

1.Policy evaluation: 给个MDP, evaluate the value of a policy 

2.Control: 给个MDP, 找optimal policy(leads to optimal value)分Value iteration和Policy iteration.

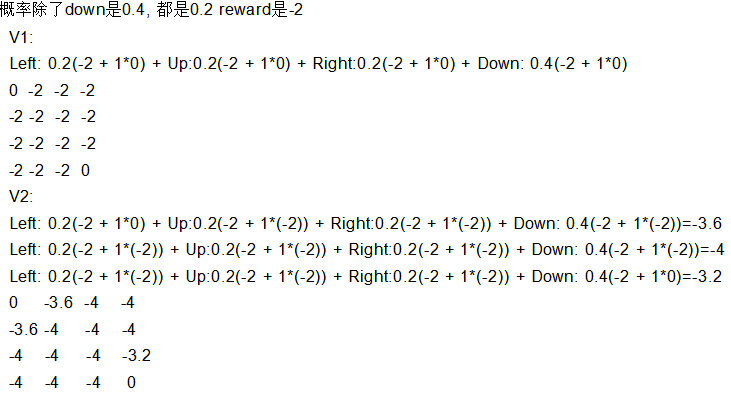
Markov Property:The future is independent of the past given the present p(st+1|st) =p(st+1|ht)和p(st+1|st , at) =p(st+1|ht , at)

为什么要discount Factor gamma: 1.Avoid infinite returns in cyclic Markov processes

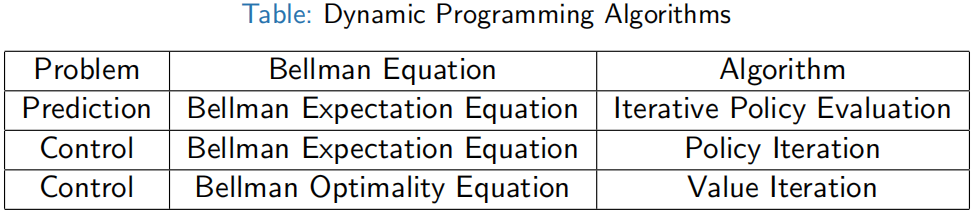
2 Uncertainty about the future: If the reward is financial, immediate rewards may earn more interest than delayed rewards

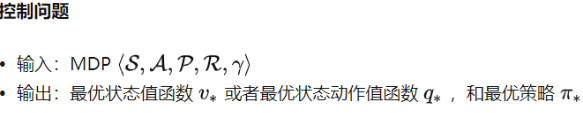
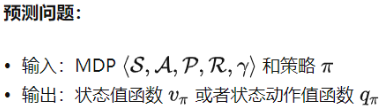
3 Animal/human behavior shows a preference for immediate reward. Matrix Form of Bellman Equation for MRP很慢,要O(N^3), 所以要用Iterative Algorithm(dp, MC, TD). MDP: (S, A, P(model), R, γ). Policy evaluation=policy下evaluate value=prediction=船问题. **Prediction**:

Input: MDP < S, A,P, R,gamma> and policy π or MRP < S,P^π, R^π,gamma> Output: value function v^π. **Control** (找optimal policy): Input: MDP < S, A,P, R,gamma> Output: optimal value function v^∗and optimal policy π∗可以用DP解决. DP可以解决的问题的property:can be decomposed into subproblems + Subproblems recur many times +Solutions can be cached and reused. Markov decision processes满足propertie因为Bellman equation gives recursive decomposition +Value function stores and reuses solutions

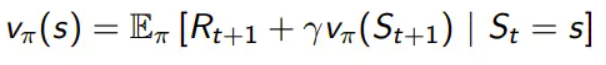


都是gamma乘下个位置的reward Q(11,down) = r(11, down) + gamma \* (V(s’)) = -2 + 1\*0 = -2  
 Q(7,left) = -2 + 1\* -7.76 = -9.76

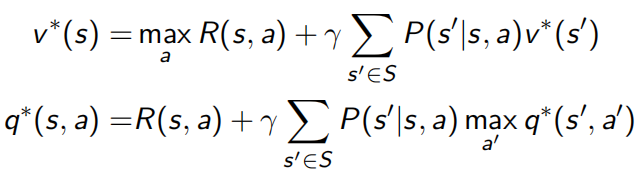




Bellman Expectation Equation



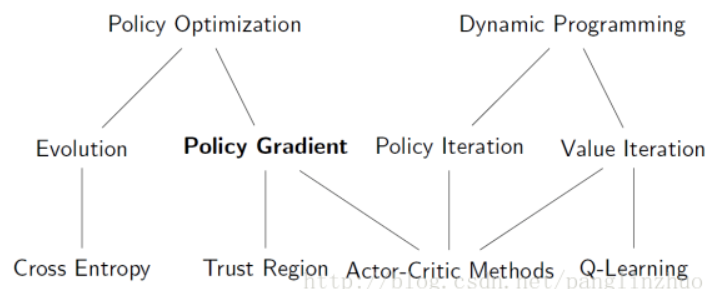
Bellman Optimality Equation



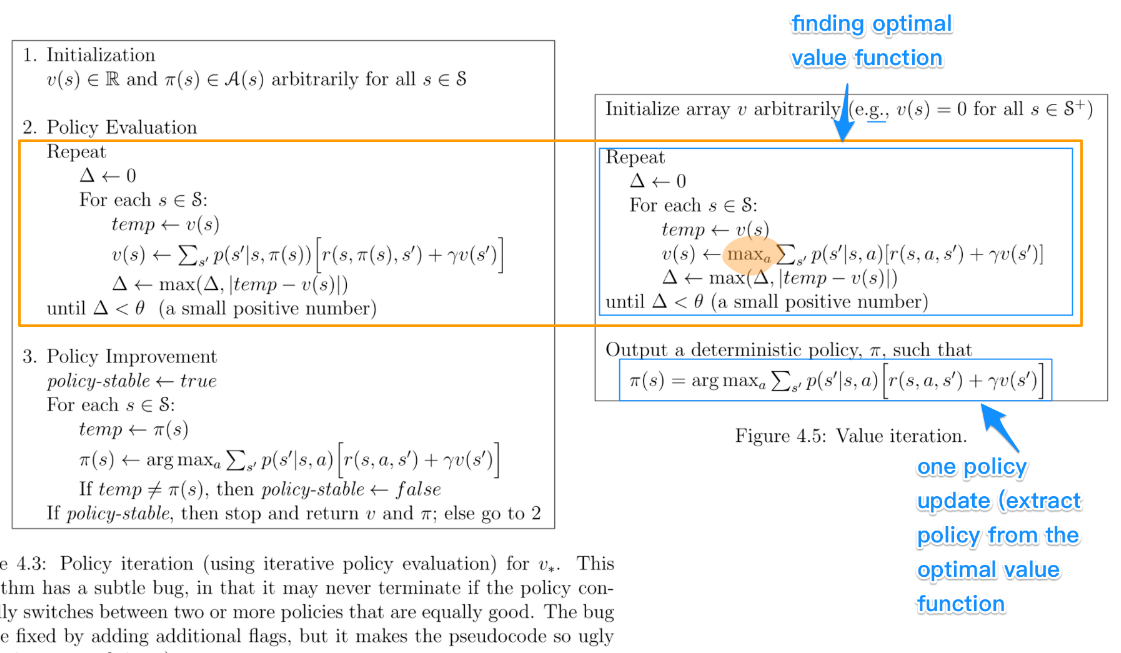
两种DP：**Policy iteration**=policy evaluation(用当前的v(s)对当前策略进行评估，计算出每一个状态的v(s)，直到v(s)收敛，才算训练好了这个状态价值函数V(s)) + policy improvement(在每个状态s对每个可能动作a都算采取这个动作后到下一个状态的期望价值。看哪个动作可到达的状态的期望价值函数最大，就选这个动作。以此更新了π(s))重复这两步, and the two are repeated iteratively until policy converges. **Value iteration**=finding optimal value function(对每一个当前状态s,对每个可能的动作a,都计算一下采取这个动作后到达的下一个状态的期望价值。看看哪个动作可以到达的状态的期望价值函数最大，就将这个最大的期望价值函数作为当前状态的价值函数v(s) 循环执行这个步骤，直到价值函数收敛) + one policy extraction(用之前步骤得到的optimal价值函数和状态转移概率，就算出每个s应该用的optimal a,这个是deterministic). There is no repeat of the two because once the value function is optimal, then the policy out of it should also be optimal (i.e. converged). 值迭代算法是策略评估过程只进行一次迭代的策略迭代算法

1.策略迭代的第二步policy evaluation与值迭代的第二步finding optimal value function十分相似，除了后者用了max操作，前者没有max.因此后者可以得出optimal value function, 而前者不能得到optimal function.

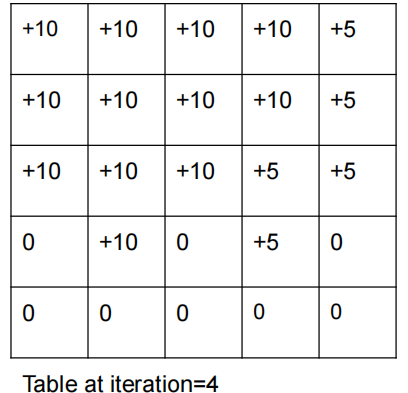
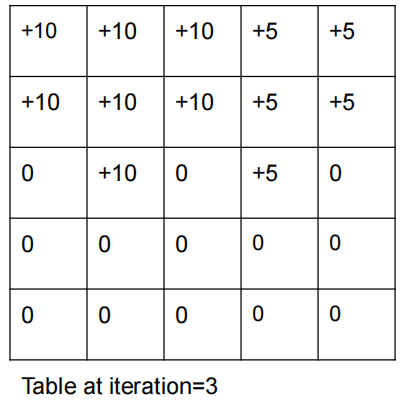
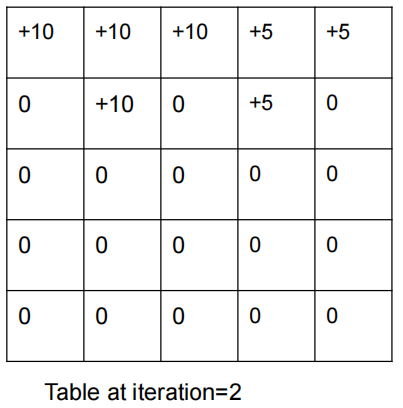
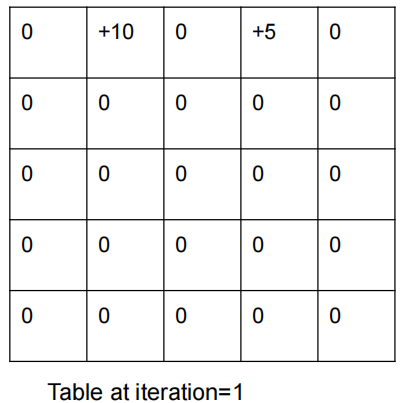
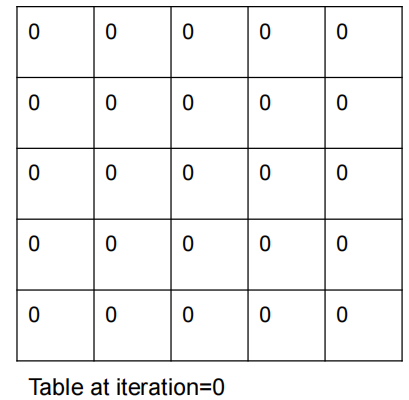
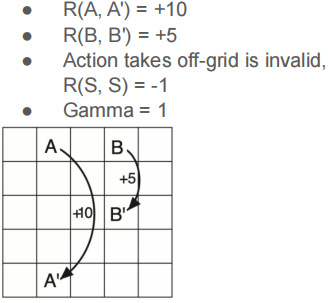
2.策略迭代的收敛速度更快，在状态空间较小时最好用策略迭代方法。当状态空间较大时，值迭代的计算量小

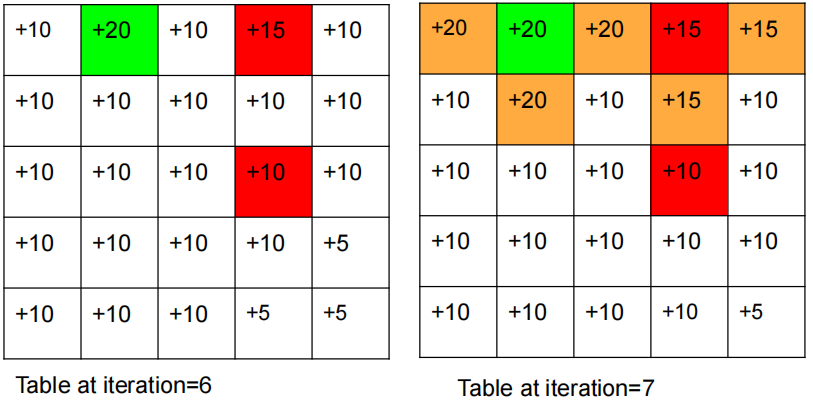
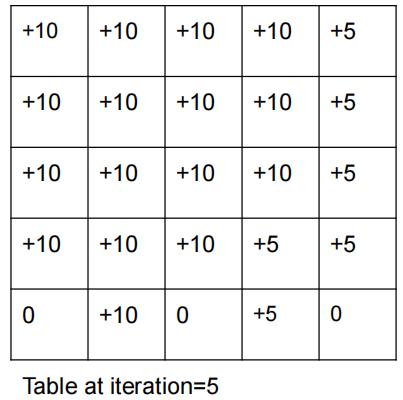


Policy Evaluation Grid里面如果要出边界了,那么V(s’)--下个状态的value就是自己, 如果动作是determinstic的,那么下个动作就是概率1,不是什么0.25,0.4,只要考虑reward下个状态的value



**Value Iteration:**





**DP缺点**：require sweeps of the state set, 如果state set太大,sweep一次要很久; Asychronous DP algorithms are in-place iterative DP that are not organized in terms of systematic sweeps of the state set;values of some states may be updated several times before the values of others are updated once

Synchronous DP太慢,变asynchronous DP的3个方法In-place dp(只存1个copy of value function,而不是2个

);Prioritized sweeping;Real-time dp(After each time-step St , At , backup the state St)  
**Sample backup**:Q-learning和SARSA的design, 用< S, A, R, S’ >pair而不是R和P函数.好处:Model-free: no advance knowledge of MDP required.Break the curse of dimensionality through sampling.Cost of backup is constant, independent of n = |S|

**Model-free prediction**(MC和TD):不需要知道Reward和Policy

**Monte Carlo (MC)**: Simulation-based prediction/policy evaluation 

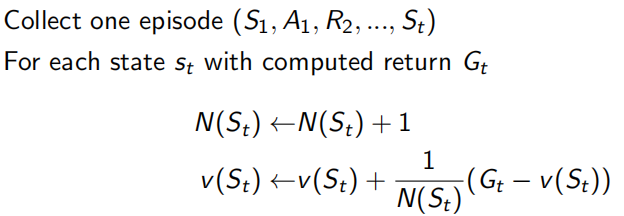
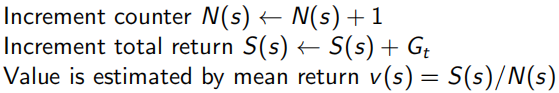
MC simulation: sample a lot of trajectories, compute the actual returns for all the trajectories, then average them

policy evaluation/Return:

MC policy evaluation用empirical mean return不用expected return

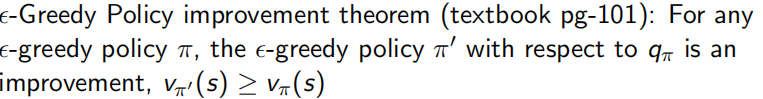
MC不需要MDP dynamics/rewards, no bootstrapping, and 不用 assume state is Markov. Only applied to episodic MDPs (each episode terminates)

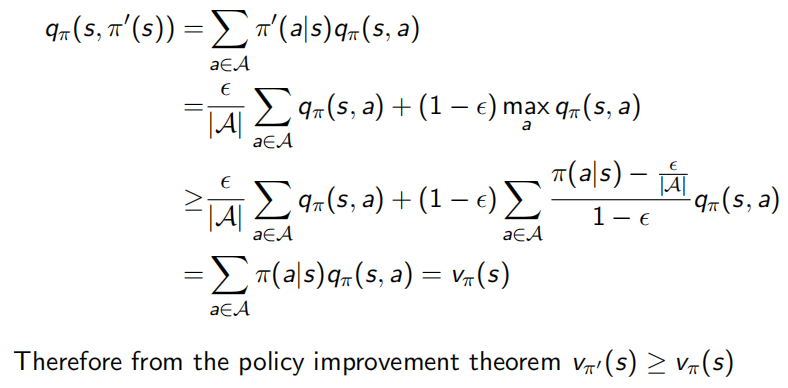
policy evaluation:



Or use a running mean (old episodes are forgotten). Good for non-stationary problems.



**MC with E-greedy exploration**: 1-E概率选greedy action, E的概率选random,

、、、、、、

**MC vs DP**

1.MC works when the environment is unknown 2.Working with sample episodes has a huge advantage, even when one

has complete knowledge of the environment’s dynamics, for example, transition probability is complex to compute

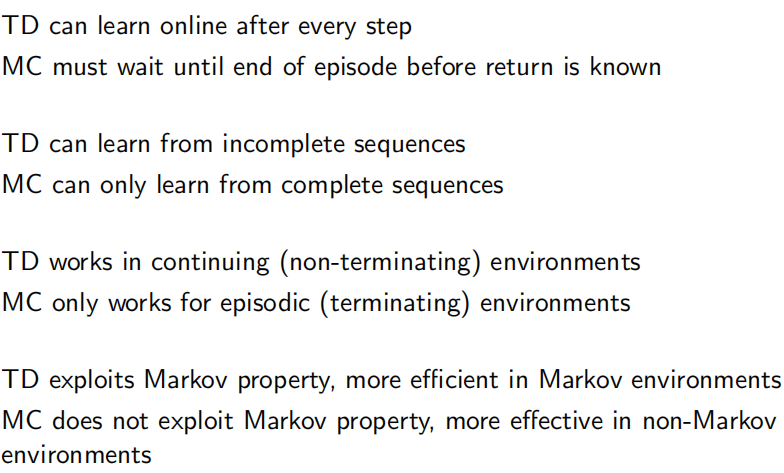
3 Cost of estimating a single state’s value is independent of the total number of states. So you can sample episodes starting from the states of interest then average returns

**Temporal Difference (TD)**: learns from incomplete episodes, by bootstrapping

最简单的**TD prediction**：

TD target(return,用来更新v(st)):TD error: 

**TD vs MC:**

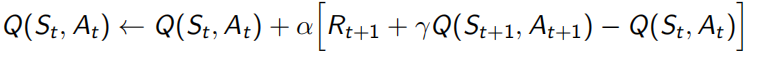


1. step TD prediction:return: 

Bootstrapping: update involves an estimate: MC not, DP TD do

Sampling: update samples an expectation: MC和TD do, DP not

DP相当于BFS, MC相当于DFS, TD介于之间

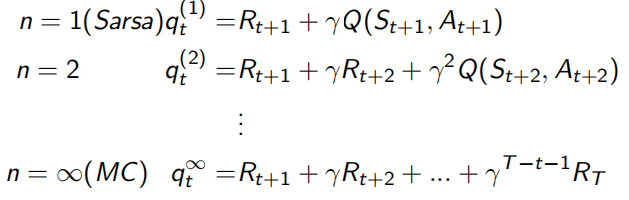
**TD比MC好的地方:**Lower variance, Online, Incomplete sequences, 所以在control loop用TD: Apply TD to Q(S, A);Use greedy policy improvement;Update every time-step rather than at the end of one episode

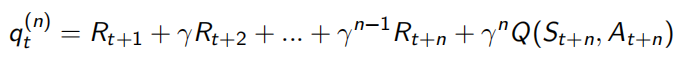
**SARSA(on-policy TD Control, policy(Stochastic or epsilon greedy)是啥就用啥)**:E-greedy policy for one step, then bootstrap the action value function

The update is done after every transition from a nonterminal state.

policy 在下个s产生a during TD是deterministic policy with epsilon-greedy exploration

TD target:



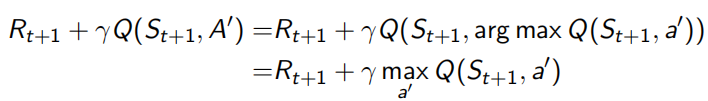
n-step Q-return:

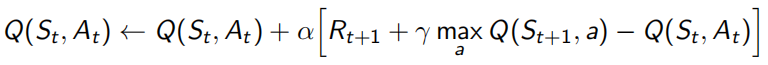
n-step Sarsa updates Q(s,a) towards the n-step Q-return

**off-policy learning**用两种polices: 1.target policy: learned and becomes the optimal policy 2. behavior policy: more exploratory + used to generate trajectories/collect data S1, A1, R2, ..., ST ∼ µ. Update π using S1, A1, R2, ..., ST好处:Learn about optimal policy while following exploratory policy

2 Learn from observing humans or other agents

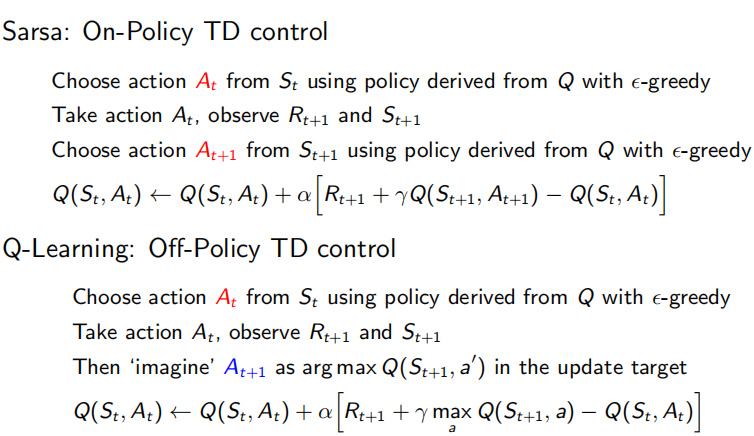
3 Re-use experience generated from old policies π1, π2, ..., πt−1

**Q-Learning(off-policy TD control)**的target policy is greedy:  
Behavior policy能完全random,但用E-greedy, target:

Update:

policy在下个s产生a during TD是deterministic policy without epsilon-greedy exploration.

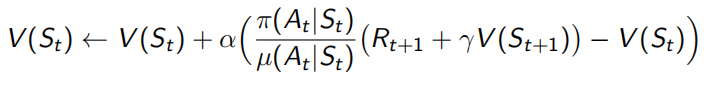
Sarsa和Q-learning的behavior policy都是deterministic policy with epsilon-greedy exploration.



**Importance sampling**:无法p(x)抽样/抽样成本高->从简单的分布q(x)抽样 + 乘上p(x)/q(x)来修正从不同分布采用的概率 ->off-policy learning里面就是Estimate the expectation of return using trajectories τi sampled from

another policy (behavior policy µ)

**Off-policy MC用IS**: 先从behavior policy µ里得到episode并return G. -> 根据policy的相似度来Weight G(Multiply importance sampling corrections along whole episode) -> Update value towards correct return 

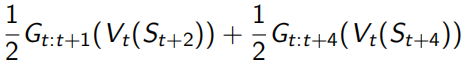
**Off-Policy TD用IS**：Use TD targets generated from µ to evaluate π -> Weight TD target R + γV(S 0 ) by importance sampling -> Only need a single importance sampling correction Policies only need to be similar over a single step

**为什么Q-learning不用Importance sampling**:1,Q-learning用deterministic policy so no action probability. 2,Q-learning does not make expected value estimates over the policy distribution

1. learning can be considered as sample update of value iteration, except instead of using the expected value over the transition dynamics, we use the sample collected from the environment. Q-learning is over the transition distribution, not over policy distribution thus no need to correct different policy distributions

**Eligibility traces** provide an efficient, incremental way of shifting and choosing between Monte Carlo and TD

a backup can be done toward a target that is half of a two-step return and half of a four-step return





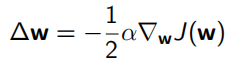


For λ = 1, updating is a Monte Carlo algorithm.λ = 0 it becomes a one-step TD method.

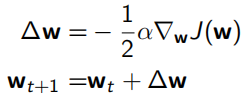
**Week4**

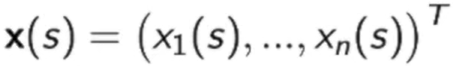
**Function approximators**: Linear combinations of features, Neural networks, Decision trees,前两个differentiable

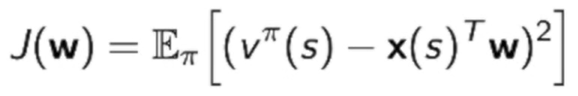
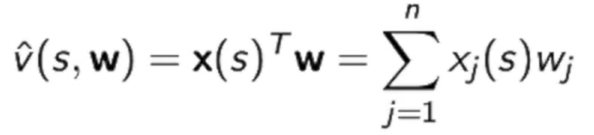
**Gradient Descend**: minimize J(w)使他到达极小点. J(w)关于参数 w的梯度将是损失函数（loss function）上升最快的方向。而我们要最小化loss，只需要将参数沿着梯度相反的方向前进一个步长，就可以实现目标函数（loss function）的下降。这个步长 a又称为学习速率 △J(w)是参数的梯度

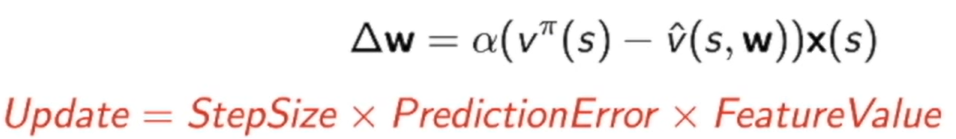
Adjust w in the direction of negative gradient, a is step size

**Value Function Approximation(VFA): with oracle**(有个真值函数->对于每个s知道应当的价值函数是多少)

所以loss func=迭代找local minimum

用feature vector来表示每个s，比如车的位置, 速度，角度等

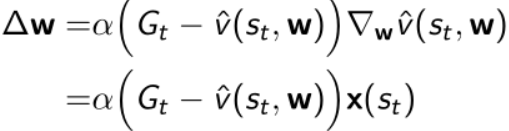
所以

**Linear value func approximation**里的更新可以表示成

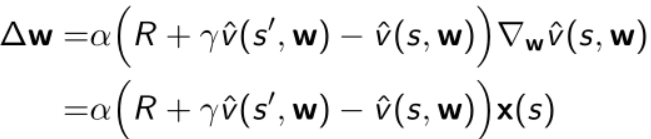
Linear的好处是Stochastic gradient descent(SGD)可以converge到global optimum,因为linear case里只有一个optimum  
Table lookup feature是one-hot vector(向量基本上都是0，只有一个元素是1，当前状态等于某一个状态，对应的那个元素会变成1，除了那个状态其他都是0), 所以可以得到拟合价值函数就等于当前对应于某个位置的wk。因此现在优化的就是去估计wk

**Without oracle**, 没有，则可以在model-free时候把func appro放到loop，一边优化value func，一边利用优化好的value func优化value func的appro

用target来替代真值, **MC用Gt(**Gt是unbiased,但对于真值是noisy sample), so要采样很多次->得到很多pair<S,G>->产生gradient,线性的话可以提出特征x(st) -> gradient可以对VFA的参数优化 -> 得到近似的VF



**在TD(0)**, 用TD target(biased, 不等于,因为TD target包含了正在优化的w)来替代, 由两部分组成：实际走完这一步的reward；bootstrapping估计得到下一个状态的近似价值函数. training pair<S,>, 可以放到gradient里面。也叫做semi-gradient，不是真实的gradient，因为它包含优化的参数w，不同的时刻w不同所以gradient不一定很准



TD(0)如果采取的是线性特征x ( s ) x(s)x(s)，得到的是全局最优解

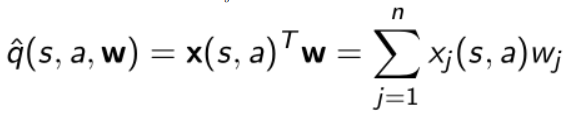
**同理VFA for control**

Policy evaluation：approximate近似 policy evaluation，将q table用一个带参数w的函数来近似

Policy improvement：采用ϵ-greedy 改进算法



**Linear Action-Value Function Approximation**



MC:

Sarsa:开始的时候初始化需要优化的w；如果是结束状态的话就用return；如果不是结束状态的话就往前走一步，采样出A’，构造出它的TD target作为oracle，然后算出它的gradient；每往前走一步更新一次w；S和A都更新

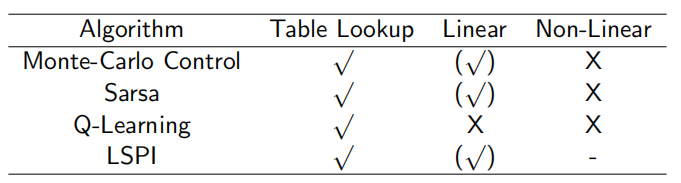
Q-learning: 

**Convergence问题:**if TD with VFA -> gradient inaccurate，因为gradient包含了w; update过程有2个近似Bellman backup和underlying value function -> noisy; TD diverge->下面Off-policy training

**Deadly trial for the danger of divergence**: Function approximation: use VFA/Q func -> error

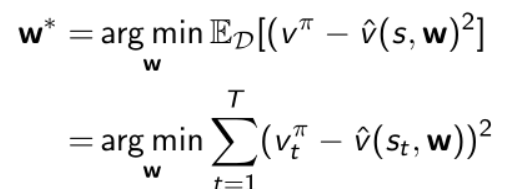
2 Bootstrapping: TD appro based on previous appro -> noise. MC > TD因为MC use real return(unbiased)

3 Off-policy training: behavior policy collect data, 但use target policy to train -> uncertainty

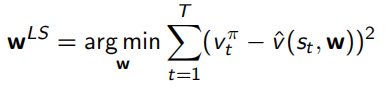
打钩代表可以找到最优解

上面都是incremental gradient descend update,效率低->用**batch-based method**去优化batch里面所有experience

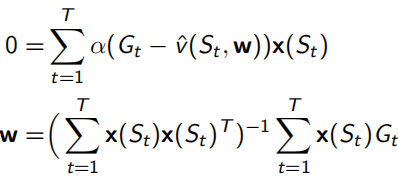
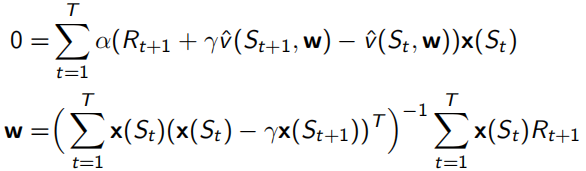
**Least Square Prediction**: experience D, pair < s1 , v1π>value可以用return或者TD target来替换.优化w来拟合整个采集到的数据D，使得在这个数据库里面每个pair都极小化



**SGD with Experience Replay**(随机采样):D很大,数据没法全放 -> 迭代去: 随机取几个算gradient去优化函数

这样迭代和一步优化得到的w一样

At minimum of LS(w), the expected update must be zero

LSMC:LSTD:

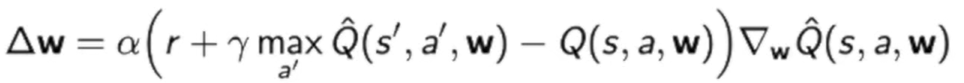
**Least Square Control with VFA(**Off-policy**)**:Policy evaluation: Policy evaluation by least squares Q-learning

LSDQ:sum of gradient = 0

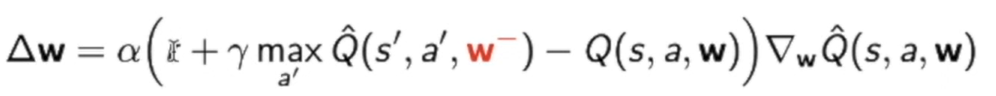
**Non-Linear value func approximation**不用feature design, Non-Linear VFA:Deep Neural Network

**Deep RL**: 用Deep Neural Network代表Value/policy func, Model。**问题:** 太多parameter要optimize + deadly trial

**DQN**:用Experience Replay和Fixed Q target来解决sample correlation太高和Non-stationary target的问题

**Experience Replay**：存储transition(st,at,,rt,st+1) in replay memory D -> 在D里随机采样得到target value, 用SGD update weight

**Fixed Q target:**improve stability,在target计算里多一个w- -> fix target

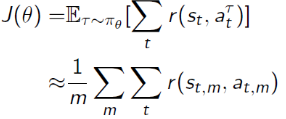


**Week5:**

**Policy-based RL**:**优点**: guaranteed to converge on a local optimum (worst case) or global optimum (best case); Policy gradient is more effective in high-dimensional action space;Policy gradient can learn stochastic policies, while value function cannot **缺点**: converges to a local optimum; evaluating a policy is inefficient and has high variance

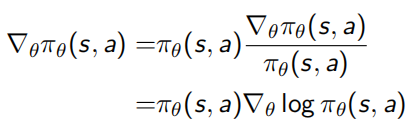
**Evaluate policy**:1.从环境角度 1.1 episodic 的环境, 用最开始的value

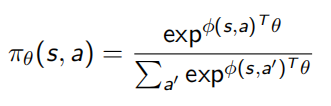
1.2continuing环境，用平均的value平均的reward per time-step

2.Episode角度:value function可以表示为 , goal是 

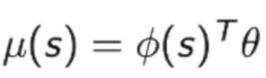
**Optimize J(θ)**:1.J(θ) differentiable, 用gradient-based methods: gradient ascend, conjugate gradient 2.non-differentiable用black box optimization: Cross-entropy method (CEM)对分配采样，得到100个policy func去和环境交互得到100个J(θ),再选top10%, Evolution strategy, Finite difference

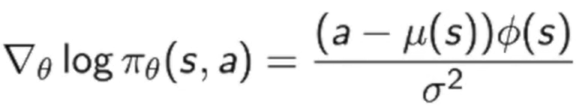
是probability func, differentiable的话可求gradient

这样得到score func

**EX：Softmax policy**：输入s输出概率。weight actions using linear combination of featuresProbability of action is proportional to the exponetiated weight score func:

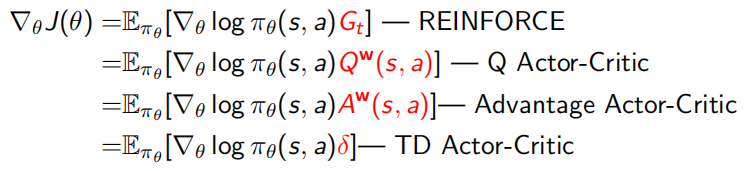


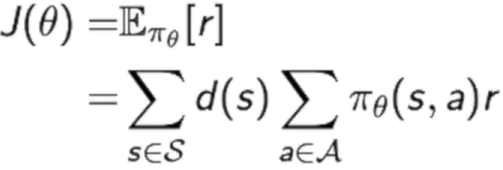
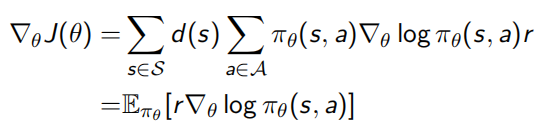
**Gaussian Policy**：在continuous action space, mean:Variance:  Policy is Gaussian, the continuous

score func:

**Policy Gradient**(policy optimization的算法;**On-Policy RL**)**one-step MDP** d代表状态集

Forms:

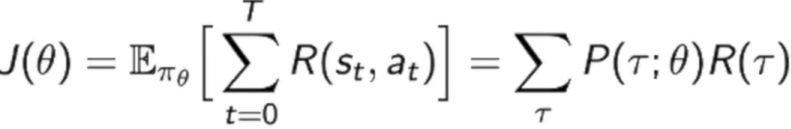


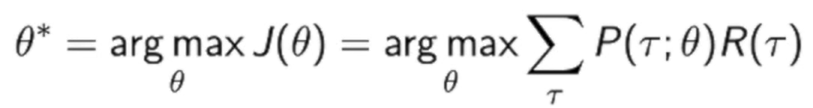
Gradient:

**Log likelihood trick****Cancel trick**

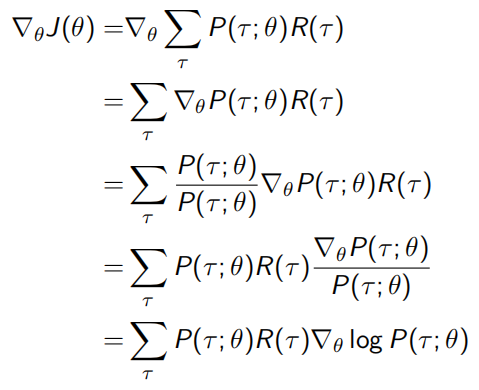
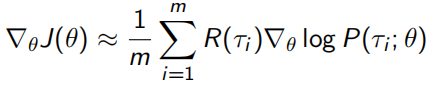
**Multi-step MDP:**

State-action trajectory R(t): sum of reward

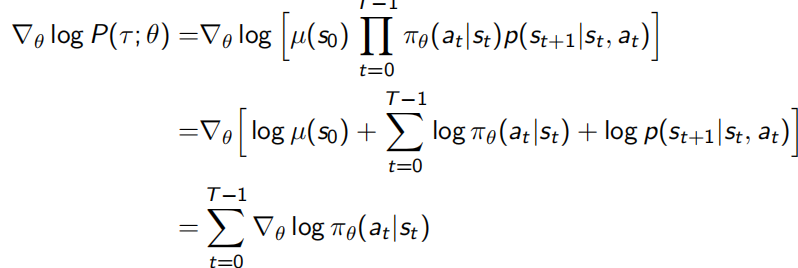
P(t)policy和环境交互产生trajectory的概率

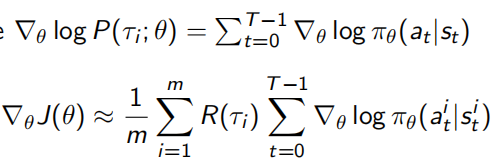
Goal:

**MC policy gradien**t推导过程：用likelihood ratio trick还原成log形式->可以把连乘变成加和的形式

用MC去采样m个trajectory

Decompose 

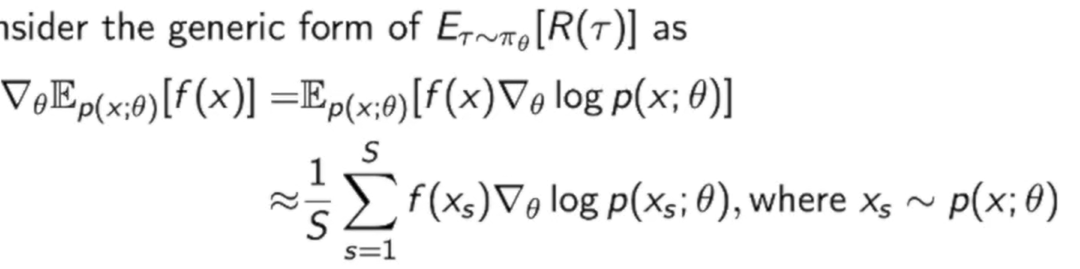




整个推导过程就让求梯度变成加和的形式，例如10条轨迹，就把10条轨迹上的奖励加和加起来，对于每一步也有 score function（就是 likelihood function 取个梯度） 也是加和起来(变成连加形式 ->无用的量被消去)，然后就直接得到了客观函数的梯度，整个过程不用知道dynamics model，这也是 policy gradient 的一个特殊之处

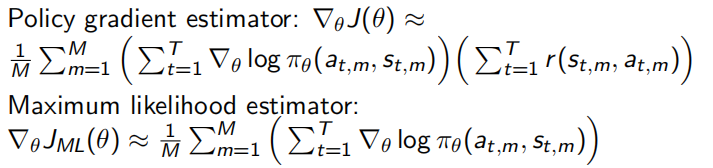
**降低Variance**

**Score Function Gradient Estimator：**近似我们的客观函数的梯度，通过采样 S 个来求平均



**Policy Gradient Estimator vs Maximum Likelihood Estimator**

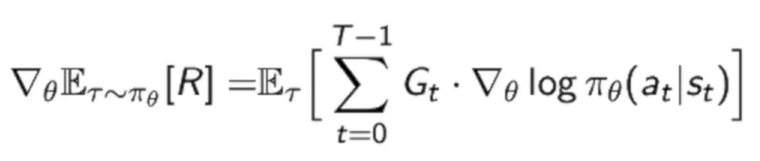
Policy Gradient Estimator多了一个reward func, 相当于加权后的Maximum Likelihood Estimator



**问题**：Unbiased但variance因为用MC.

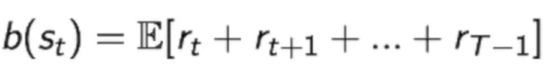
**解决**：**1.use Temporal Causality**(对时序上面的因果关系运用进去 -> 去掉不必要的reward)

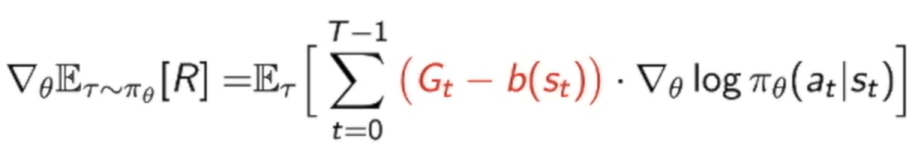
Causality：policy at time t’ cannot affect reward at time t when t < t’



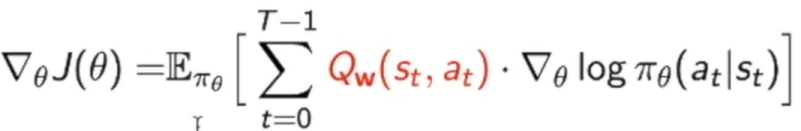
应用：REINFORCE: a Monte-Carlo policy gradient algorithm

**2.Include baseline**(对reward做的normalization（归一化）-> R这一项本身的variance减小)

Baseline = expected return=0不会改变policy gradient值



用critic



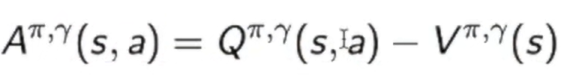
**Actor-Critic Policy Gradient**: 结合了policy和value func

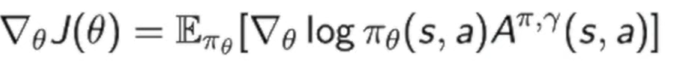
**Actor**: policy func(generate action), Updates policy parameters θ, in direction suggested by critic

**Critic**: value func(evaluate actions reward). Updates action-value func parameters w,用policy eva(MC, TD,LS)去estimate 

**Actor-Critic Function Approximators**：结合Value/policy func共享feature输出

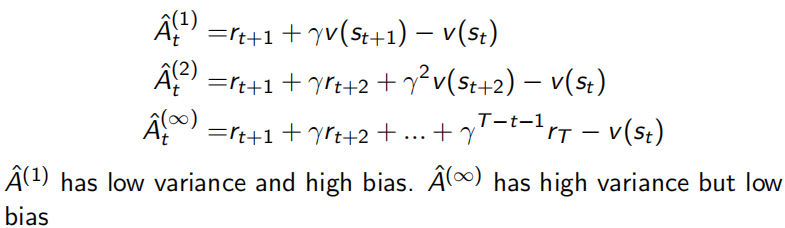
**Advantage func**:combine Q和Baseline V(Q func的平均) 为了减小variance



policy gradient变成：问题Q和V都有自身参数

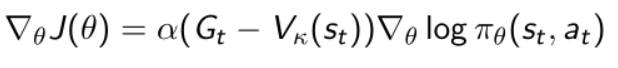
还可以用TD error重写

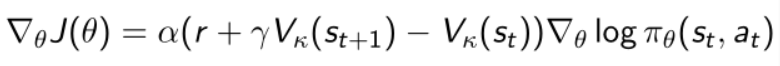
这样只要去拟合V就行

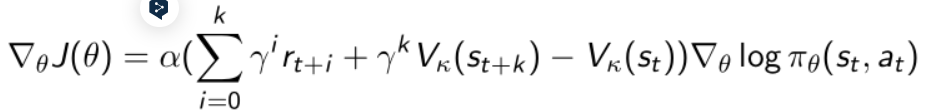


对于actor，用不同方法得到不同policy gradient



MC:

TD:

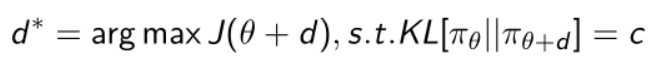
k-step return

**Week 6:**

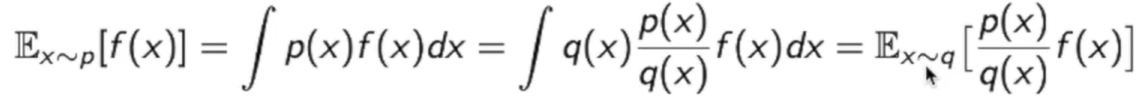
**Stochastic policy-based approaches**: Policy Gradient→TRPO→ACKTR→PPO

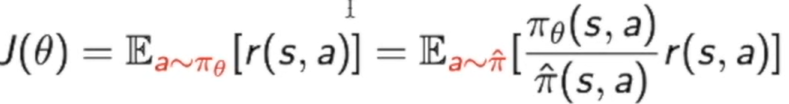
**PG问题**：1.poor sample efficiency, 因为PG要collect data+optimize policy 2.Unstable training process,因为bad step size->bad->bad policy->bad data collect->destroy training->collapse performance

**解决**：1.用importance sampling改成off-policy(TRPO) 2.用trust region + natural policy gradient

**Natural policy gradient**:PG的value func和parameter 的更新 no relation ->用KL-divergence(固定的小的数)使得更新前后的policy的差异度尽可能小**.缺点:**必须计算Fisher Information Matrix

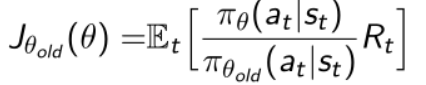
**PG用IS**:用别的policy func采样a -> 乘ratio来近似(用behavior policy产生实际的trajectory)



乘ratio后:

Deep Q learning里用behavior policy采的data放到replay buffer重用 ->现在policy/old policy的ratio可能非常大->unstable process -> 用KL限制优化的policy和old policy距离(TRPO)

**TRPO(Natural policy gradient + IS)**: **Trust Region**(Robustness增加):

**KL constraint**

δ限制更新之后的policy和old policy的近似度，δ和learning rate直接联系起来了。

**好处**：1.不设step-size，只要trust region的大小(更新前后输出距差查)，然后用距离推step-size。trust region设的小 -> 稳定 2.用conjugate gradient(CG)算FIM的H

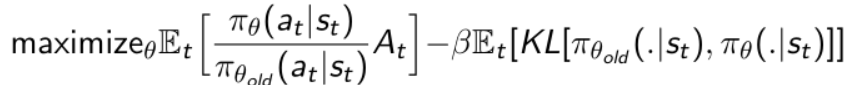
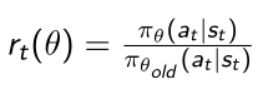
**Limit**:计算量大,虽然用CG,但每一步都要算H; CG复杂

**ACKTR**:用K-FAC加速计算FIM H^-1(用kronecker product做近似)

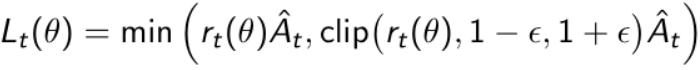
**PPO**:1.用了unconstrained form把TRPO两个条件结合->同时优化函数+考虑条件 **KL penalty**

2.做了Adaptive KL Penalty:更新的policy比old policy大(KL-divergence)(ex大于1.5δ->penalty变大，θk+1的第二项被考虑更多->更小更新 β\*2;反之/2

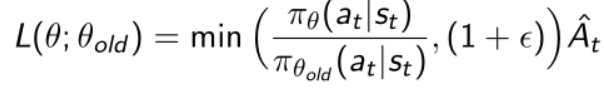
PPO(用first-order optimization, SGD没算KL-divergence和FIM)比TRPO快, same performance，安全性高,实现简单

当前policy和old的ratio

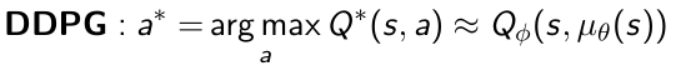
Clipping函数处理loss的优化，根据上面ratio, 大于1+ϵ或小于1 − ϵ就会把它clip掉 -> 1−ϵ <ratio < 1 + ϵ ϵ通常=0.2



If advantage正数 -> 鼓励当前action(ratio分子会大)用min(ratio和1+E), 反之,不鼓励，用max(ratio和1-E)



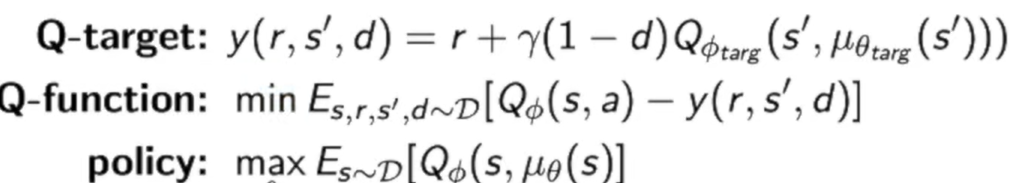
**Deterministic policy-based (Value-based) approaches**: Q-learning→DDPG→TD3->SAC

**DDPG(Deep Deterministic PG)**:DQN扩展到连续的动作空间

=policy. 放s->μθ输出一个连续的值->放到Q network得到Q值

action a 是连续的->Q function对于a Qϕ(s , a)可求导->可以结合PG和value func的优化

也用replay buffer+target network，value network和policy network都有target network



**问题**：Q-func overestimate Q-value -> unstable train

**TD3(Twin Delayed DDPG)改进点**：用两个Q-func(取更小的那个) + 更新policy比更新Q-func慢(解决overestimate) + smoothing(add noise -> regularize)

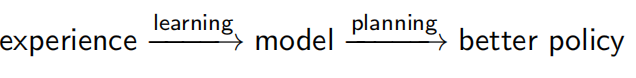
**SAC(Soft Actor-Critic)=TD3+entropy+reparameterization**：1.用了entropy regularization->trade-off expected return和entropy希望对未知空间有一定探索-> 把entropy写到Value func/Q func

2.用2个Q-func,取小的 3.用reparameterization(让expectation本来是a对policy function的采样，变成跟参数θ无关的采样带参的θz在μ和σ里->和ϵ随机性无关

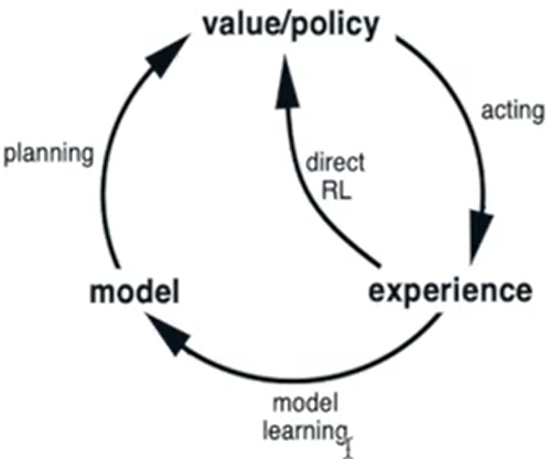
**Week 7:**

Learn environment model -> improve value/policy optimization

可以先和环境model接触,再和环境接触



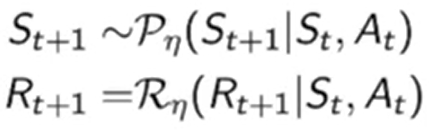
State-space planning: search through the state space for an optimal policy or an optimal path to a goal



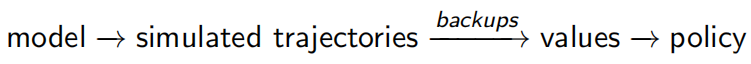
experience能优化value/policy和model

Model-based RL好处：Higher sample efficiency,对robot很重要+能用supervised learning

Model-based RL坏处:learn model+v/p function会导致2个approximation error;不能guarantee of convergence

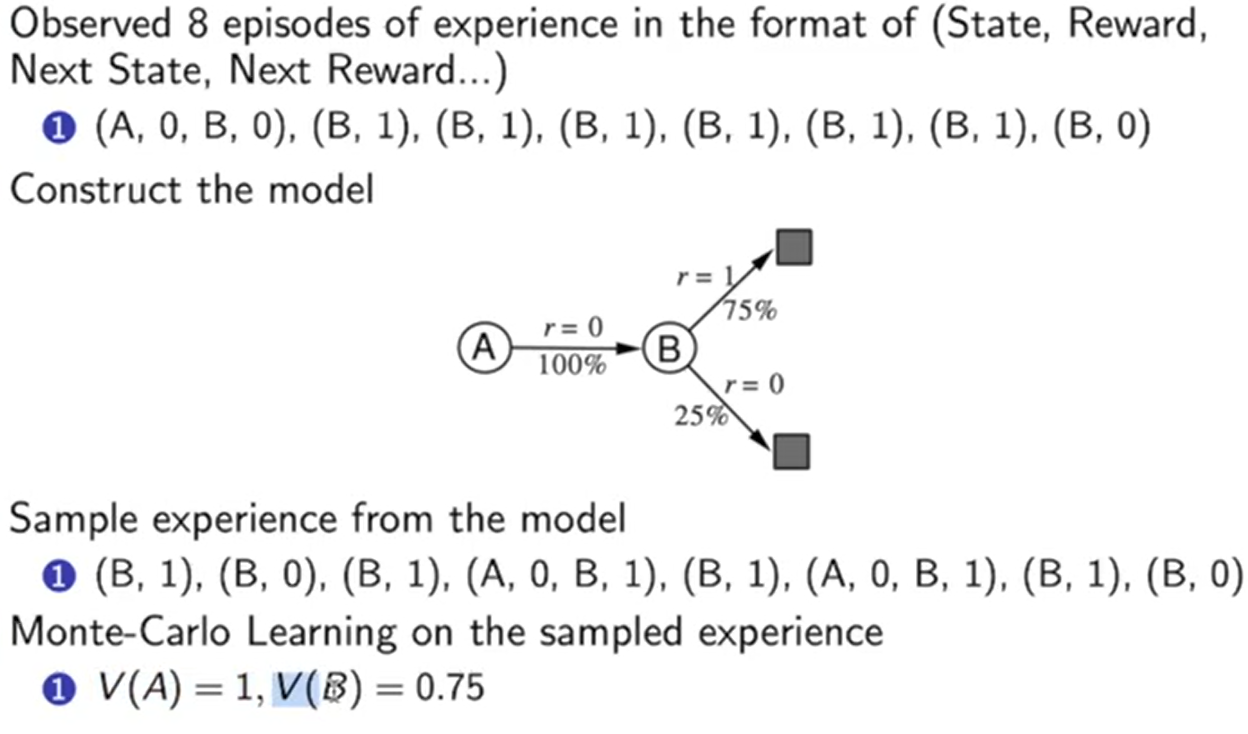
Model组成: P和R相互独立

**Model-based value optimization** methods structure



用supervised方法,用S,A得到R(regression problem),S’(density estimation problem)

Loss func:mean-squared error, KL divergence -> 优化model  
Ex of model: Table lookup model(just count:S,A->s’ or S,A->R看看有多少种), Linear Expectation m,



75%概率是因为6/8

得到model后planning步骤:对model进行sampling ->得到trajectories->用model-free对value func进行统计

But model inaccurate -> planning a suboptimal policy,解决:model accuracy低就用model-free RL

**现在有2种Experience**: real ex(用model-free和环境交互得到)，simulated ex(从环境model sampling得到)

**Model-free RL**：No model +Learn value function (and/or policy) from real ex

**Model-based RL** (using Sample-based Planning): Learn a model from real ex + Plan value function (and/or policy) from simulated experience

**Dyna**: Learn a model from real ex + Learn and plan value function (and/or policy) from both real and

simulated ex

**Model-based policy optimization methods structure**

Dynamics know -> optimal control问题，解决:LQR,iLQR. 最小化cost, cost func = negative reward.

Dynamics model unknown

算法1：random policy去交互环境->得到轨迹->learn dynamics model去最小化loss -> plan through f(s,a)去选action

->用supervised learning去train model去最小化error -> 用LQR去计算optimal trajectory

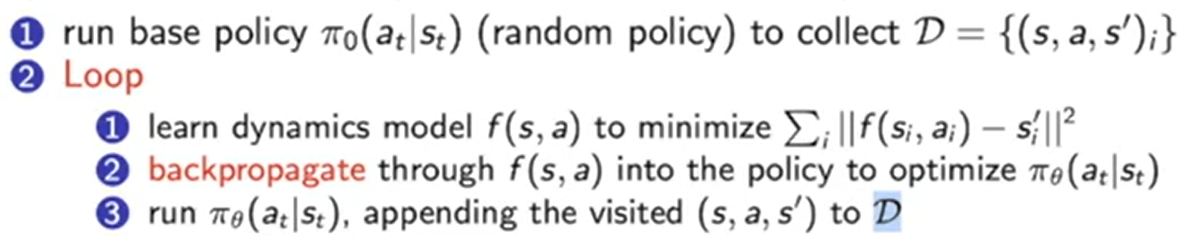
问题:drifting,最早对trajectory优化的时候有error -> 之后有更大error

算法2: 用loop把没包含trajectory里的data也包含数据集，再重新learn model

问题:产生的trajectory一开始就是far from我们需要的轨迹

MPC：对整个trajectory进行planning优化，但只执行最前面的trajectory -> observe -> replan

算法3：用MPC, loop each step,执行first planned a得到s’,把s,a,s’放到model

算法4：

Parameterizing the model:

Global model:st+1 = f(st, at) 每个状态都用同一个model, 用big neural network表示

好处:expressive+can use lots of data to fit 坏处:not great in low data regime, cannot express model uncertainty

Local Model:model the transition as time-varying linear-Gaussian dynamics

好处：data-efficient+can express model uncertainty坏处：slow when data big

**Advanced Topic:**

**DAgger=**Annotating More On-Policy Data to train Iteratively to solve off-course situations

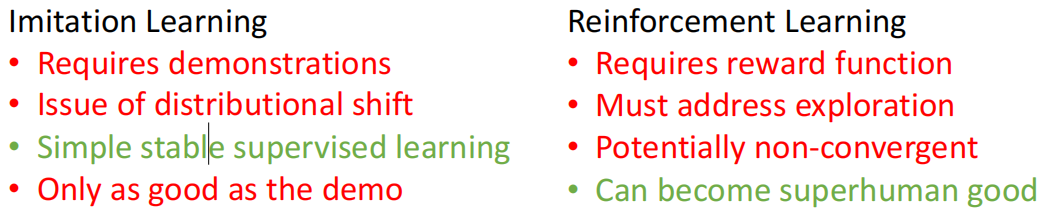
**DAgger Limit**:Need to query expert many times during the training; Very expansive if it is human expert

**解决**:把让人label改成其他算法

**Inverse RL(IRL)**:环境+behavior =>reward func,最大化human demonstration, 最小化robot attempt

**GAIL**:训练G产生trajectory让D没法区分是不是真实数据

**改进IL**:handle multimodal behavior; model the who history



**集合IL和RL：Pretrain & Finetune:** 先用expert demonstration initialize policy再用RL改善policy

**问题**：first batch of bad data can destroy initialization解决:用off-policy RL保留expert demonstration(off-policy gradient with important sampling, off-policy Q-learning放到replay buffer)

**3个IL的问题和解决:**1.How collect expert demonstrations: Crowdsourcing +Guided policy search or optimal control for trajectory optimization; 2.How optimize the policy for off-course situations: Simulate those situations to collect new labels + Use off-policy learning with the already collected samples + Combine IL and RL

3.How outperform the expert who provides the data: Offline RL

**Offline RL(Pre-collected large-scale static datasets to train) vs IL**:IL assumes offline data from expert, offline RL can learn from less accurate data; IL assumes no reward, offline considers the reward associated with the offline transitions

**Properties of Distributed System**:Consistency, Fault tolerance, Communication

**Model parallelism**: different machines are responsible for computation in different parts of a single network

**Data parallelism**: different machine has a complete copy of the model, each machine gets a different portion of the data