

## 南京大学俞扬博士：强化学习前沿（下）

本文作者：奕欣

2017-05-15 09:36



导语：本文根据俞扬博士在中国人工智能学会AIDL第二期人工智能前沿讲习班"机器学习前沿"所作报告《强化学习前沿》编辑整理而来，雷锋网在未改变原意的基础上略作了删减。

雷锋网 [ AI科技评论 ] 按：本文根据俞扬博士在中国人工智能学会AIDL第二期人工智能前沿讲习班"机器学习前沿"所作报告《强化学习前沿》编辑整理而来，雷锋网(公众号：雷锋网)在未改变原意的基础上略作了删减，并经俞扬博士指正确认，特此感谢。全文分为上下两篇，本文为下篇。

上篇传送门：《南京大学俞扬博士：强化学习前沿（上）》



俞扬博士、副教授，主要研究领域为人工智能、机器学习、演化计算。分别于2004年和2011年获得南京大学计算机科学与技术系学士学位和博士学位。

2011年8月加入南京大学计算机科学与技术系、机器学习与数据挖掘研究所（LAMDA）从事教学与科研工作。曾获2013年全国优秀博士学位论文奖、2011年中国计算机学会优秀博士学位论文奖。发表论文40余篇，包括多篇Artificial Intelligence、IJCAI、AAAI、NIPS、KDD等国际一流期刊和会议上，研究成果获得IDEAL'16、GECCO'11、PAKDD'08最佳论文奖，以及PAKDD' 06数据挖掘竞赛冠军等。

任《Frontiers of Computer Science》青年副编辑，任人工智能领域国际顶级会议IJCAI' 15/17高级程



奕欣



初心者



职业开脑洞 脱线段子手。邮箱：  
guoyixin@leiphone.com



发私信

### 当月热门文章

清华大学朱军博士：可扩展的贝叶斯方法与深度生成模型

发现未来连接创新，CCF-GAIR 2017将于7月7日再度起航！| GAIR 2017

南京大学俞扬博士万字演讲全文：强化学习前沿（上）

阿里云 iDST 总监初敏博士：AI技术发展商业化之路 | GMIC 2017

谷歌大脑撰文解析 AutoML：神经网络如何自行设计神经架构？ | Google I/O 2017

### 最新文章

清华大学舒继武教授：基于非易失存储器的存储系统软件层优化 | CCF-ADL 火热报名中

AI时代的产品经理，应该注意什么？

序委员、IJCAI'16/17 Publicity Chair、ICDM'16 Publicity Chair、ACML'16 Workshop Chair。指导的学生获天猫“双十一”推荐大赛百万大奖、Google奖学金等。

在此列出俞扬老师讲课目录，以供读者参考：

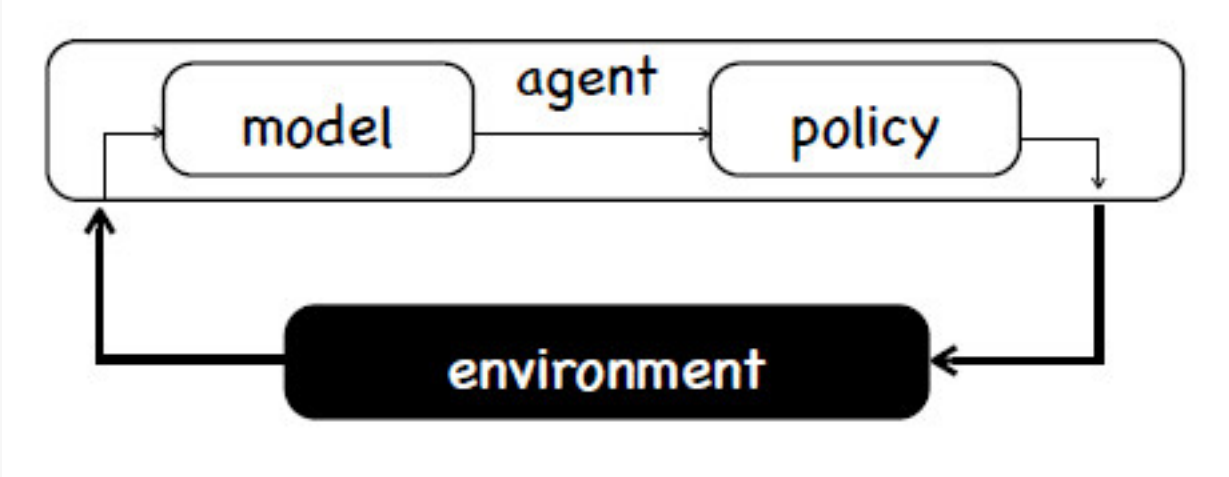
- 一、介绍 ( Introduction )
- 二、马尔可夫决策过程 ( Markov Decision Process )
- 三、从马尔可夫决策过程到强化学习 ( from Markov Decision Process to Reinforce Learning )
- 四、值函数估计 ( Value function approximation )
- 五、策略搜索 ( Policy Search )
- 六、游戏中的强化学习 ( Reinforcement Learning in Games )
- 七、强化学习总结
- 八、强化学习资源推荐

上篇介绍了前两个小节的内容，以下为下篇内容：

### 三、从马尔可夫决策过程到强化学习

在强化学习任务中，奖赏和转移都是未知的，需要通过学习得出。具体解决办法有两个：

一种是还原出奖赏函数和转移函数。首先把MDP还原出来，然后再在MDP上解这个策略，这类方法称为有模型 ( Model-Based ) 方法，这里的模型指的是MDP。



还有一类和它相对应的方法，免模型 ( Model-Free ) 法，即不还原奖赏和转移。

#### 基于模型的方法

在这类方法中，智能体会维护Model ( 即MDP )，然后从Model中求解策略。

从随机策略开始，把策略放到环境中运行，从运行的序列数据中把MDP恢复出来。因为序列数据可以提供环境转移和奖赏的监督信息，简单的做一个回归，就能知道一个状态做了一个动作下面会转移到哪儿，以及能得到的奖赏是多少。

这里有一个非常简单的环境探索方法——RMax，它用了计数这个非常简单的回归模型。

端午福利！ AI慕课学院请你来听课（适用专属优惠码）

谷歌输入法背后的机器智能：思你所思，想你所想！

清华大学博士生涂锋斌：设计神经网络硬件架构时，我们在思考些什么？（下）| 硬创公开课总结

一键调用Siri、Alexa、Cortana和Google Assistant，一款蓝牙耳机居然做到了

#### 热门搜索

- 智能手表
- 联想
- Instagram
- 看点
- 智能驾驶
- 创业公司
- Model S
- 新能源汽车
- Fitbit
- 高德
- 思科



# learn an MDP model

random walk, and record the transition and the reward.  
more efficiently, visit unexplored states

## RMax algorithm:

[Bertsekas, Tsitsiklis. R-Max---A general polynomial time algorithm for near-optimal reinforcement learning. JMLR'02]

```
initialize  $R(s)=R_{\max}$ ,  $P = \text{self-transition}$ 
loop
  choose action  $a$ , observe state  $s'$  and reward  $r$ 
  update transition count and reward count for  $s,a,s'$ 
  if count of  $s,a \geq m$ 
    update reward and transition from estimations
   $s = s'$ 
```

sample complexity  $\tilde{O}(|S|^2|A|V_{\max}^3/(\epsilon(1-\gamma))^3)$

[Strehl, et al. Reinforcement learning in finite MDPs: PAC analysis. JMLR'09]

虽然看起来很简单，但是还原MDP的样本复杂度是状态数的平方，远高于前面说到的求解策略的复杂度。从这里可以看出学习MDP的复杂度极高，所以大量的研究工作都集中在免模型学习上。

## 免模型学习

免模型学习简单介绍两种方法。一种叫做蒙特卡罗采样方法（Monte-Carlo method），一种是时序差分法（Temporal difference method）

- 蒙特卡罗采样方法介绍（Monte-Carlo method）

免模型学习和之前讲到的策略迭代的思路很像，首先，评估当前策略怎么样；第二，提高当前策略。

### 第一步 评估策略

在MDP里评估策略的时候，由于奖赏和转移都是知道的，所以可以直接用这两个函数算评估值。现在这两个函数都不知道，那怎么办呢？

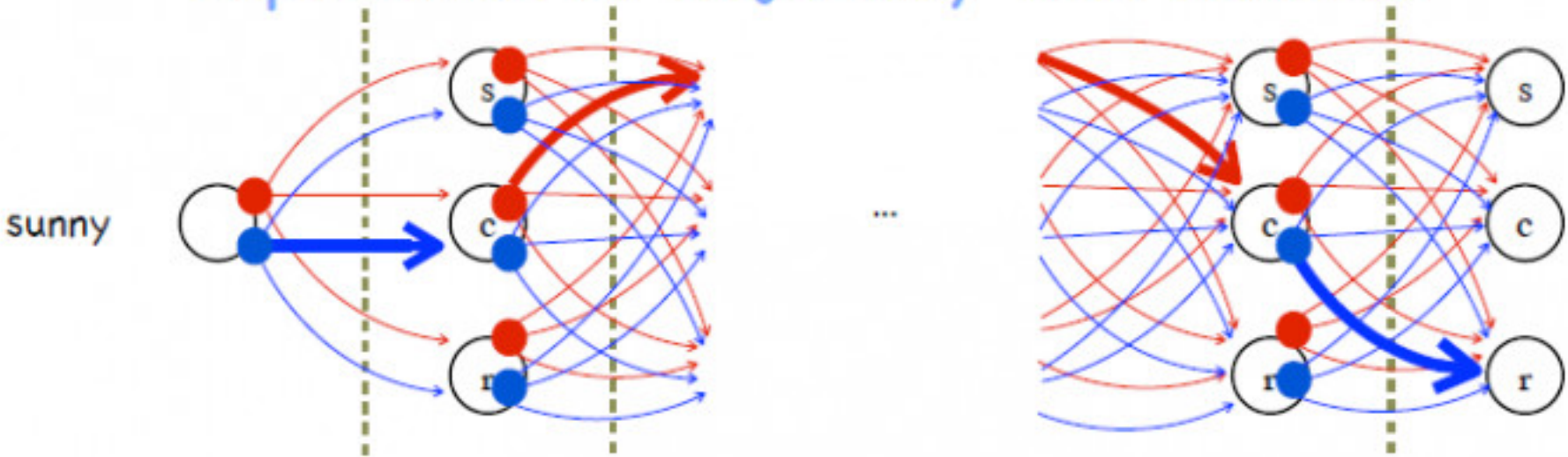
这个Q值函数实际上是个期望，所以直接用采样来替代期望就可以了。换句话说，就是拿该策略在环境里面去跑，看跑出来什么结果。

## Monte Carlo RL - evaluation

Q, not V

expected total reward  $Q^\pi(s,a) = E[\sum_{t=1}^T r_t | s,a]$

expectation of trajectory-wise rewards



sample trajectory m times,  
approximate the expectation by average

$$Q^\pi(s,a) = \frac{1}{m} \sum_{i=1}^m R(\tau_i) \quad \tau_i \text{ is sample by following } \pi \text{ after } s,a$$



比如跑了之后我得到一条轨迹：先是出太阳，接着是多云，最后是出太阳；再跑第二次得到一条轨迹，再跑第三次又得到一个轨迹。最后得到很多轨迹。我知道每条轨迹的奖赏是多少，然后把这些轨迹的奖赏平均起来，作为这个策略的值函数的估计，用频率来逼近期望。

## 第二步 更新/提高策略

如此一来，我们就可以去评价一个策略的好坏。评价完一个策略以后，就可以和刚才一样，取Q值函数指示最好的动作作为新的策略，更新过程是一样的。

Monte Carlo RL - evaluation+improvement

```
Q0 = 0
for i=0, 1, ..., m
  generate trajectory <s0, a0, r1, s1, ..., sT>
  for t=0, 1, ..., T-1
    R = sum of rewards from t to T
    Q(st, at) = (c(st, at)Q(st, at) + R) / (c(st, at) + 1)
    c(st, at) ++
  end for
  update policy π(s) = arg maxa Q(s, a)
end for
```

improvement ?

整个算法写出来比较简单。我们要做m次采样，每一次都把当前的策略拿到环境里面运行，然后会得到一个序列，根据序列让奖赏求和，然后更新Q值，这个Q值就是历史上采样的均值，c是计数。

在一条轨迹下来以后，更新Q值后，做第二条轨迹，这样就做到了不依赖MDP模型的强化学习方法。

## 然而该方法缺乏环境探索，难以更新策略

但是，这个有一个问题——如果得到了确定性策略，那么有可能采100个样本出来的轨迹都是一样的，导致无法评估策略在所有状态上的表现，所以无法提高策略。这里的关键在于它缺乏对环境的探索。

## 如何探索环境，以获得最大回报？

怎么探索？我们可以考虑一个最简单的强化学习问题：一个状态，两个动作，一个动作的回报高一点，一个动作回报低一点，但是这两个回报来自于两个分布。这个时候你选哪个动作，或者你怎么做能收到最大的回报？这其实就是bandit模型。

- 一个极端是，尝试100次，每个动作做50次，这个时候我可能知道哪个动作比较好，但是拿到的回报可能不是最高的，因为可能做10次以后，就已经知道第一个动作的回报要高一点了，如果剩下的投资还是均匀分布的话，就得不到最大回报。
- 另一个极端是，两个动作各试一次，看哪个回报高，剩下的98次全部投到最高的回报去。这个方法也不好，因为只试了1次，估计的回报很不稳定。

第一种情况是要有足够多的探索（即exploration），第二种情况是不需要过多的探索而有更好的投资（即exploitation），我们要在这两点之间找到平衡。

解决这个问题有多种方法。第简单的方法是，以1-ε的概率，现在看好哪个，就去投资它，剩下的ε概率就完全随机，每个动作都去尝试。这个方法称为ε-greedy。

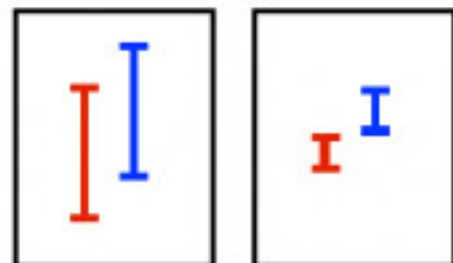
该方法可以保证所有状态都有一定的概率，哪怕是很小的概率，被访问到。所以当运行一段时间以后，它能够找到最优的策略。

但这个方法也有缺点，就是必须要指定一个 $\epsilon$ 值。通常这个值应当不断衰减，直到收敛到一个比较好的结果。还有一个效率问题，比如A动作尝试了10次以后，平均回报是1万，B动作尝试了10次以后是0.1，这个时候就已经没有必要尝试下去了，因为距离非常远。但是 $\epsilon$ -greedy的探索不会停下来，所以有了其他的方法，比如softmax——它会考虑到Q值本身，如果两个动作的值差别很大，探索的概率就很小。另一个在理论上比较漂亮的方法是UCB（Upper Confidence Bound）：

- 第一，考虑了Q值。如果Q值本身差距比较大，探索的可能性就很小；
- 第二，考虑了探索次数。如果探索次数很少，可能它的置信度就比较低，如果探索的次数较多，置信度就会比较高。

upper confidence bound (UCB):  
choose by action quality + confidence

$$Q(k) + \sqrt{2 \ln n / n_k}$$



所以，按照Q值加上置信度的上界来选择动作，它就会自动平衡。

不过，最常用的还是第一种 $\epsilon$ -greedy方法。给出一个策略 $\pi$ 以后，把它变成探索的策略，即随机挑选一个动作，把这个带探索的策略放到蒙特卡罗的算法里面。并且，这个轨迹并不是从 $\pi$ 中产生的，而是从带探索的 $\pi_\epsilon$ 中产生的，这就能保证策略可以不断更新了。

## Action-level exploration

$\epsilon$ -greedy policy:

given a policy  $\pi$

$$\pi_\epsilon(s) = \begin{cases} \pi(s), \text{ with prob. } 1 - \epsilon \\ \text{randomly chosen action, with prob. } \epsilon \end{cases}$$

ensure probability of visiting every state  $> 0$

下面介绍On/Off Policy：学习带探索/不带探索的策略。

大家可能常听On/Off Policy策略这个词。

this algorithm evaluates  $\pi_\epsilon$  ! on-policy

what if we want to evaluate  $\pi$  ? off-policy

在蒙特卡洛采样中使用了 $\pi_\epsilon$ 策略来采样，学的并不是 $\pi$ ，是带探索的 $\pi_\epsilon$ 。因为用来评估的数据，是从带探索的策略产出来的，而不是从我们想要学的策略上产生出来的。这个区别会导致把探索也作为策略的一部。这种采样与更新的策略是一样的算法叫做On Policy。

但很多时候，我们想学的实际是不带探索的策略，也就是说要从带探索的策略中采样，但更新的只是策略



本身，即Off Policy。这里面临一个问题就是，采样并不来自于当前的策略，常用的重要性采样（Importance Sampling）技术通过修改采样的分布，改成想要的样子。可以通过加权重这个简单的方法，修改策略的分布，然把这个分布加到具体算法里面去。也就是把奖赏加了一个权重，这样的算法就变成一个Off Policy的算法，这样它学习的就是π自己了。

importance sampling:

$$\begin{array}{ccc} E[f] = \int_x p(x) f(x) dx & = & \int_x q(x) \frac{p(x)}{q(x)} f(x) dx \\ \downarrow \text{sample from } p & & \downarrow \text{sample from } q \\ \frac{1}{m} \sum_{i=1}^m f(x) & & \frac{1}{m} \sum_{i=1}^m \frac{p(x)}{q(x)} f(x) \end{array}$$

## 蒙特卡洛算法总结

总体来说，蒙特卡洛的算法不是一个效率很高的算法，但是能够展现免模型类算法的特性。

我们要做这个策略的评估，然后做完评估以后找到一个改进的方向，就可以改进这个算法了；这里，为了使策略能够有效更新，需要引入对环境的探索；而对环境的探索里面，要注意On/Off Policy这么两个概念。

另外，蒙特卡洛的算法有一个很显然的缺陷：一定要拿到整个轨迹以后，才能更新模型。

- 时序差分（Temporal difference method）

那能不能每走一步都更新模型呢？蒙特卡洛算法里面有一个性质——即更新Q值的时候，实际上是在更新均值。

更新均值还可以写成： $\mu_t = \mu_{t-1} + \alpha(x_t - \mu_{t-1})$ ，意思是刚才我们更新的是Q值（算式如下图显示），其中  $R - Q(s_t, a_t)$  叫做蒙特卡罗误差。我们知道，Q是对奖赏的一个估计，R是是采完这个轨迹以后得到的真实的奖赏。换句话说，Q值d饿更新就是加上就是真实值和估计值的差别，即蒙特卡罗误差。

Incremental mean

$$Q(s_t, a_t) = (c(s_t, a_t) Q(s_t, a_t) + R) / (c(s_t, a_t) + 1)$$
$$\begin{aligned} \mu_t &= \frac{1}{t} \sum_{i=1}^t x_i = \frac{1}{t} \left( x_t + \sum_{i=1}^{t-1} x_i \right) = \frac{1}{t} (x_t + (t-1) \mu_{t-1}) \\ &= \mu_{t-1} + \frac{1}{t} (x_t - \mu_{t-1}) \end{aligned}$$

In general,  $\mu_t = \mu_{t-1} + \alpha(x_t - \mu_{t-1})$

Monte-Carlo update:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \underbrace{\alpha(R - Q(s_t, a_t))}_{\text{MC error}}$$

在TD算法里，我们走了一步得到了一步真实的奖赏，再往后走还没走，所以不知道后面真实的奖赏是多

少，但可以通过之前的Q值来估计之后的奖赏，这两个加起来就是当前知道的信息，用它来替代这个R，来减去老的预估值，我们称这个过程为时序差分。

如果用蒙特卡罗的话，需要先走到底，知道总体的结果之后，每一步的差别就能算出来；而对于TDL来说，只需要记录一步的信息，所以可以在线更新自己。

# TD Evaluation

## Monte-Carlo update:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \underbrace{(R - Q(s_t, a_t))}_{\text{MC error}}$$

## TD update:

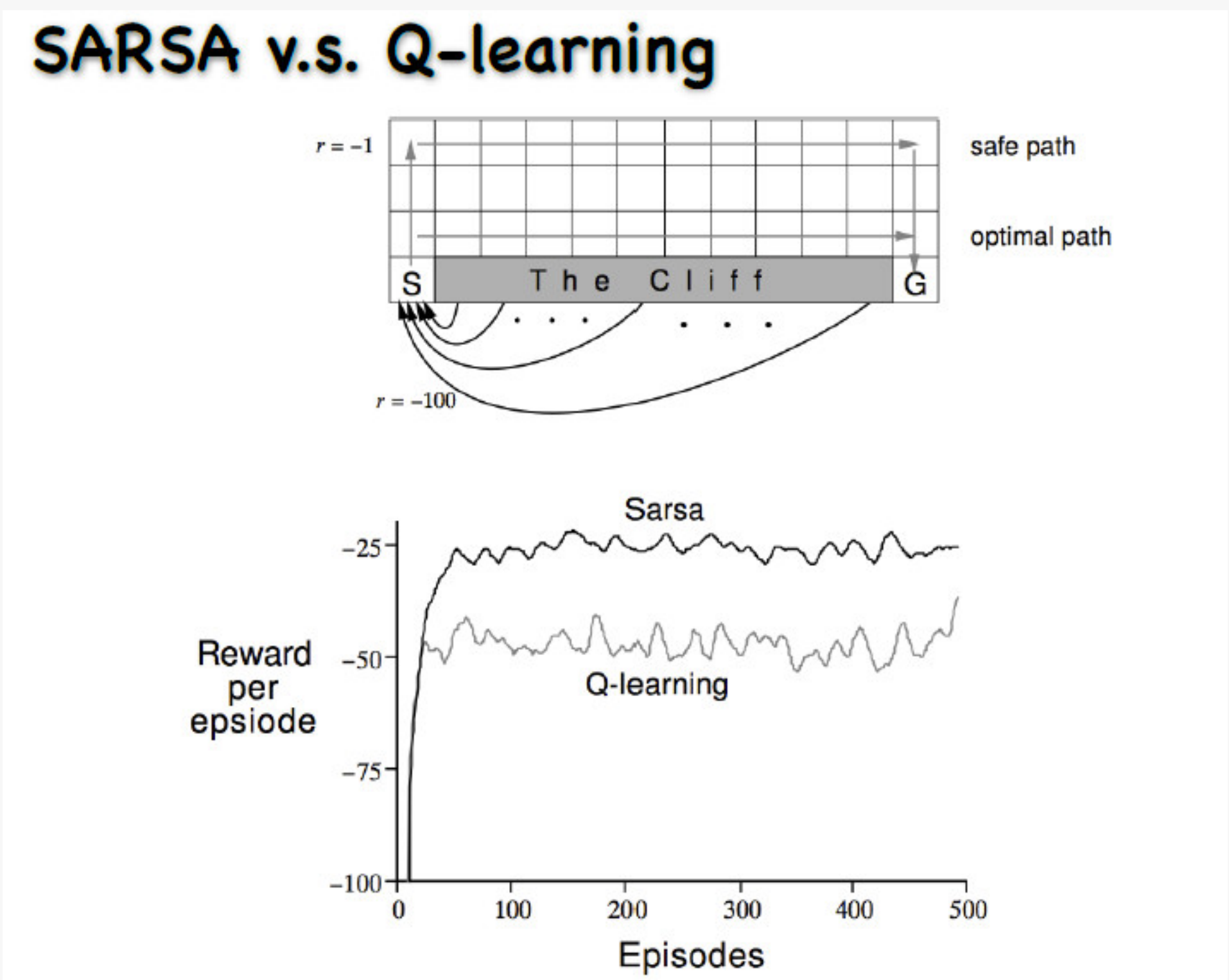
$$\begin{aligned} &Q(s_t, a_t) \\ &\leftarrow Q(s_t, a_t) + \alpha \underbrace{(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))}_{\text{TD error}} \end{aligned}$$

- SARSA

动态规划记录的是所有状态上面的信息。而把刚才的蒙特卡罗的error换成了TD errpr，就可以得到新的TD方法的强化学习方法。这个方法就不是采集整个轨迹了，而是根据探索的策略，用TDL来更新Q值，每走一步就更新一下对当前策略的评判，然后再更新策略。这个算法叫做SARSA，属于On Policy，而变成Off Policy的策略，只修改一处，用非探索策略来计算TD error，就得到Q-Learning算法。

- SARSA v.s. Q-learning

这是一个爬格子的问题，是典型的经典强化学习问题。



动作是上下左右的走，每走一步就会有一个-1的奖赏。从初始状态走到最终的状态，要走最短的路才能使奖赏最大。图中有一个悬崖，一旦走到悬崖奖赏会极小，而且还要再退回这个初始状态。

在这里用On Policy SARSA会有一定的概率去探索，也就有可能会掉到这个悬崖下面去，所以奖赏就会比



较小；而用Q Learning，因为最后的策略是不带任何探索的，没有任何的随机性，所以路径最短。

这就是两类强化学习算法的区别。你在学习过程中可以看到，Q Learning的值较低，这是因为学习的时候一定要带探索的学习，所以你训练的过程中一定是不断的去训练。

另外，前面讲的TD误差更新是走一步后的更新，实际上还可以做两步的更新、走N步的更新，都是可以的。所以有一种方法就是做很多步的，按照一个概率加权把它综合起来，综合起来以后到一个叫做λ—return，就是走一步、走两步和走多步的TD。

## 四、值函数估计 ( Value function approximation )

刚才讲的所有问题，前提是都能用表格表示。但是很多真实环境是无法用表格表示的。所以在强化学习发展的早期，一直没办法用在大规模的真实问题上去。后来大家就想，怎么把这个强化学习放在一个连续状态空间去，甚至说放在动作也是连续的情景中，比如控制一架直升机的。

大家可能觉得强化学习的学习过程和监督学习之间的差别比较大，算法、模型好像都完全不一样。但进入连续状态空间以后，两者就会出现很多相似的地方。

### Value function approximation

modern RL

tabular representation

$\pi =$ 

s	0	0.3
	1	0.7
c	0	0.6
	1	0.4
r	0	0.1
	1	0.9

very powerful representation  
can be all possible policies !

linear function approx.

$$\hat{V}(s) = w^\top \phi(s)$$
$$\hat{Q}(s, a) = w^\top \phi(s, a)$$
$$\hat{Q}(s, a_i) = w_i^\top \phi(s)$$

$\phi$  is a feature mapping  
w is the parameter vector  
may not represent all policies !

离散状态下可以用表格来表示值函数或策略；但进入连续状态空间就要用一个函数的近似来表示，这个方法叫做值函数近似。

比如，我们可以用一个线性函数来表示，V值是表示状态s下面的一个值，状态s先有一个特征的向量  $\phi(s)$ ，这个V值表达出来就是一个线性的参数乘以特征的内积。Q值里面有一个动作，假设这个动作是离散的，一种方式是把这个动作和状态放在一起变成一个特征，另一种方法是给每一个动作单独做一个模型。

当遇到连续空间的问题时，用近似来表示值函数V和Q，这个做法看起来很自然，但是近似以后会发现，以往很多的理论结果就不成立了。

但我们现在先不管那些问题，先看做了近似以后怎么来学？我们想知道的是，这里的Q值，是希望Q值近似以后，够尽量逼近真实的Q值。如果已经知道真实的Q值，怎么逼近呢？最简单的方法就是做一个最小二乘回归。其中一种解法是求导。求导以后，导数表示为，真实的Q和估计的Q的差值，然后再乘对Q值模型的导。可以看到，导数表达的含义与之前的模特卡罗误差、TD误差是一致的，只不过更新的是参数w。把这种更新方式套进Q learning里，其他地方都没有变，只得到了用值函数逼近的Q-Learning方法。



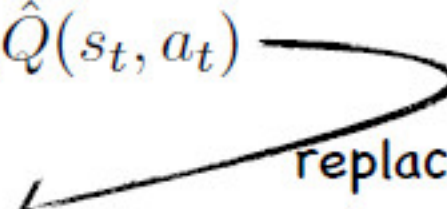
# Value function approximation

to approximate Q and V value function

least square approximation

$$J(w) = E_{s \sim \pi} [(Q^\pi(s, a) - \hat{Q}(s, a))^2]$$

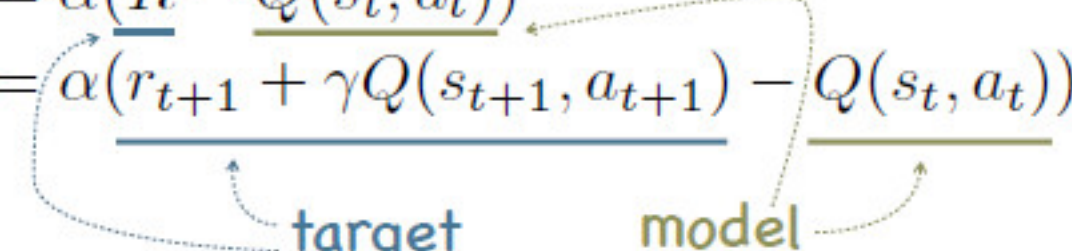
online environment: stochastic gradient on single sample

$$\Delta w_t = \theta (Q^\pi(s_t, a_t) - \hat{Q}(s_t, a_t)) \nabla_w \hat{Q}(s_t, a_t)$$


Recall the errors:

MC update:  $Q(s_t, a_t) + = \alpha (\underline{R} - \underline{Q}(s_t, a_t))$

TD update:  $Q(s_t, a_t) + = \underbrace{\alpha (r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))}_{\text{target}} - \underbrace{Q(s_t, a_t)}_{\text{model}}$



这个模型用什么函数呢？最简单就是用线性函数。但是线性函数有很多局限的，需要在特征的设计上下功夫，这需要很好的人工设计。

把它变成非线性函数，一个常用方法是用神经网络，直接用神经网络表示Q值。在更新的时候也很简单，只需要把梯度传到神经网络中去就可以了，因为神经网络的BP算法本身也是求梯度。

用批量学习改进

还有一些改进的方式。比如说我们在训练近似模型的时候，在一个样本上训练可能会不稳定，所以可以用 Batch Models 的方式，积累一批数据来训练这个模型。

## Batch RL methods

gradient on single sample introduces large variance

Batch mode evaluation:

collect trajectory and history data

$$D = \{(s_1, V_1^\pi), (s_2, V_2^\pi), \dots, (s_m, V_m^\pi)\}$$

solve batch least square objective

$$J(w) = E_D [(V^\pi - \hat{V}(s))^2]$$

linear function: closed form

neural networks: batch update/repeated stochastic update

LSMC, LSTD, LSTD( $\lambda$ )

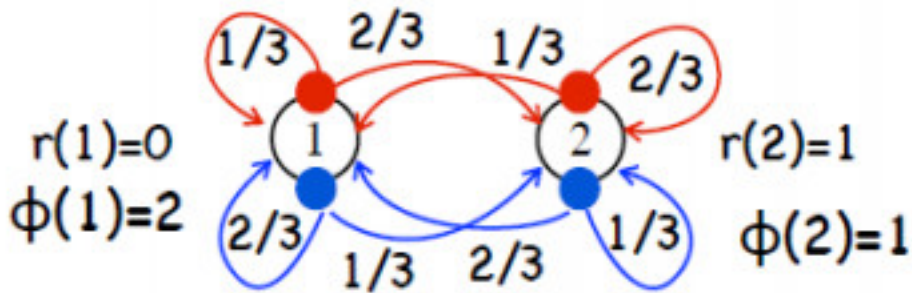
刚才讲的所有训练方法，都是先把V值或者Q值估计出来，然后再从中把这个策略导出来。我们称这种方法为基于值函数的强化学习方法。

## 五、策略搜索 ( Policy Search )



# policy degradation in value function based methods

[Bartlett. An Introduction to Reinforcement Learning Theory: Value Function Methods. Advanced Lectures on Machine Learning, LNAI 2600]



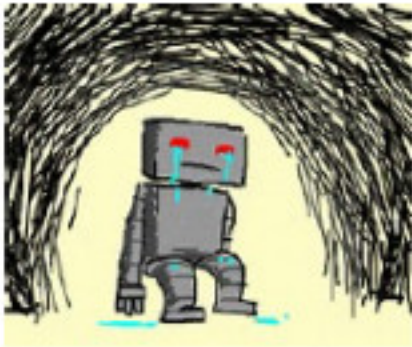
optimal policy:  $V^*(2) > V^*(1) > 0$

let  $\hat{V}(s) = w\phi(s)$ , to ensure  $\hat{V}(2) > \hat{V}(1)$ ,  $w < 0$

as value function based method minimizes  $\|\hat{V} - V^*\|$   
results in  $w > 0$

sub-optimal policy, better value  $\neq$  better policy

## Policy Search



### 值函数估计法存在的问题：策略退化

但是用值函数估计会有一个问题——这种方法可以收敛到最优策略，但前提必须是用表格的表达方式；如果用的是函数近似，则会出现策略退化，即对Q值估计越大，策略越差。

举一个简单的例子，现在有两个状态，一个是状态1，一个是状态2，状态1的特征为2，状态2的特征为1。我们设定奖赏，使得状态2的最优V值比状态1的要大。这时如果用一个线性函数来表示这个V，也就是用W乘以特征，这个特征只有一维，最优的这个V值2是比1大的，1的特征值要高一点，2的特征值要小一点，所以最优的W就应该是个负数，这样会使得V(2)比V(1)大，因而能导出最优策略。

但是基于值函数的做法是要使得V值尽量靠近最优的V值，最优的V值又是正值，这样会导致这个W一定是正的，无法得到最优的策略。这样值函数估计得越准，策略越差的现象被称为策略退化。

### 用策略搜索解决策略退化问题

为了避免策略退化，我们的方法是直接去找策略，这就是策略搜索。

先把策略参数化，对于离散动作来说，参数可以做成像Gibbs Policy一样，即每个动作有一个参数，然后把它归一，变成每一个动作有一个概率。如果是一个连续动作的话，可以用高斯分布来描述。里面这个参数，我在这里写的是一个线性的过程，但也可以用神经网络来替代。

直接优化策略的参数，使得收到的总回报达到最大的方法，就是策略搜索（Policy Search）。

### 策略搜索的优势

策略搜索和基于值函数的方法相比，优缺点各是什么？

- 第一，能处理连续状态和动作；
- 第二，对于高维的数据总的表现比较好。
- 第三，可以直接学出随机性策略
- 第四，Policy Search和监督学习的兼容性比较好。


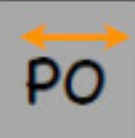

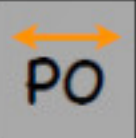




第三点用处很大，比如说玩“剪刀石头布”，如果选择确定性策略，那一定会输；一定要做一个带概率的输出才会赢。



还有另外一个例子，跟大家讲解一下为什么需要随机性策略。

Example: Aliased gridworld

state PO cannot be distinguished  
=> same action distribution

	 PO		 PO	
				

deterministic policy: stuck at one side

value function based policy is mostly deterministic

stochastic policy: either direction with prob. 0.5

policy search derives stochastic policies

骷髅代表走到这就死掉了；最优策略肯定是往中间走，但是这里有两个灰色格子，它们代表的是不完全观测的状态，即走到灰格子之后不知道该往左边还是右边；

- 如果这时又用了确定性策略，那就只能向左或向右走，只能是确定的，则有可能会遇到走不通的路径。
- 如果用随机性策略，向左和向右的概率都为50%，因此不管往哪边走总能到达目标。

这也体现了策略搜索的优势。

第四，策略搜索和监督学习的兼容性比较好。

这个策略是用参数表达的，它的目标是最大化的奖赏。最大化奖赏的意思就是说，把空间里所有的轨迹枚举出来。因为策略产生这些轨迹是有一定概率的，在某个状态上，策略做出相应动作的概率是由策略决定的，把所有一条轨迹上所有动作的概率相乘，就得出产生这条轨迹的概率。所以它总体的期望回报，就是所有轨迹的期望，也就是每条轨迹的概率乘以每条概率能获得的奖赏，这也是总回报的另外一种写法。这种写法有的好处就在于，它和策略参数目标有关，所以我对奖赏直接求导，来求解策略。另外一种写法用的是稳态分布（Stationary Distribution），用和上面写法完全等价，意思是完全一样的，在这里就跳过不讲了。

策略搜索也有一个缺点，其中一个缺点就是有很多局部最优解，失去了全局最优的收敛性保障，其次是训练过程方差非常高。

- 早期策略求导的方法：Finite Difference

相信大家都会求导，不过有一种方式大家可能没有见过——有限差分（Finite Difference），这是早期用来做策略求导的方法。

那什么时候会用到有限差分呢？可能是这个系统可能太复杂了，不容易求导，那就可以用一个简单的方式来逼近这个导数。拿到一个参数 $\theta$ ， $\theta$ 的导数就是看一下周围哪个方向走的比较快，这样给 $\theta$ 加一个很小的扰动的值，对 $\theta$ 周围的局部进行采样，对那个采样增长得最快，这个方向就当成是一个导数方向。这是最简单的方法，当然这个方法有很多缺陷，特别是在高维度的情况下，会需要很多采样，所以更直接的方法还是直接求导。

最后得到的一个导数，导数形式如下所示：



# Analytical optimization: REINFORCE

$$J(\theta) = \int_{Tra} p_{\theta}(\tau) R(\tau) \, d\tau$$

logarithm trick  $\nabla_{\theta} p_{\theta} = p_{\theta} \nabla_{\theta} \log p_{\theta}$

as  $p_{\theta}(\tau) = p(s_0) \prod_{i=1}^T p(s_i | a_i, s_{i-1}) \pi_{\theta}(a_i | s_{i-1})$

$$\nabla_{\theta} \log p_{\theta}(\tau) = \sum_{i=1}^T \nabla_{\theta} \log \pi_{\theta}(a_i | s_{i-1}) + \text{const}$$

gradient:  $\nabla_{\theta} J(\theta) = \int_{Tra} p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) R(\tau) \, d\tau$

$$= E \left[ \sum_{i=1}^T \nabla_{\theta} \log \pi_{\theta}(a_i | s_i) R(s_i, a_i) \right]$$

E是期望，1到T代表考虑的是T步的轨迹，每一步轨迹对策略输出值的对数取导数，然后乘以真实的奖赏（奖赏不取对数）。奖赏是个常数，即轨迹得到的奖赏值。

可以通过采样可以来逼近期望，对一个策略以后，去跑一些轨迹，然后计算平均梯度，作为梯度期望的逼近。

我们刚刚说到，这种方式有一个很大的缺陷，就是它的方差很大，直接用计算的梯度来更新策略（vanilla policy gradient），基本上得不到好的策略，因为它的方差太大，不稳定。

## 控制方差的方法 1、Actor-Critic

控制方差有多种方式，其中一种叫做Actor-Critic。用比如直接求导的方式把策略求出来，叫做Actor；对值函数进行估计，并用于评估策略，是Critic，意为它是一个评价者。

我们要维护一个值函数Q的模型。另外，用导数的方法来求策略的梯度的时候，不做直接使用奖赏，而是使用Critic提供的Q值。所以Actor-Critic会维护两个模型，第一个是策略的模型，第二个是Q函数的模型。

# Reduce variance by critic: Actor-Critic

Maintain another parameter vector  $w$

$$Q_w(s, a) = w^{\top} \phi(s, a) \approx Q^{\pi}(s, a)$$

value-based function approximated methods to update  $Q_w$   
MC, TD, TD( $\lambda$ ), LSPI

Multi-step MDPs:  $J(\theta) = \int_S d^{\pi_{\theta}}(s) \int_A \pi_{\theta}(a | s) Q^{\pi_{\theta}}(s, a) \, ds \, da$

$$\nabla_{\theta} J(\theta) = E[\nabla_{\theta} \log \pi_{\theta}(a | s) Q^{\pi_{\theta}}(s, a)]$$

Policy Gradient Theorem  
equivalent gradient for all objectives

[Sutton et al. Policy gradient methods for reinforcement learning with function approximation. NIPS'00]

$$\nabla_{\theta} J(\theta) \approx E[\nabla_{\theta} \log \pi_{\theta}(a | s) Q_w(s, a)]$$

if  $w$  is a minimizer of  $E[(Q^{\pi_{\theta}}(s, a) - Q_w(s, a))^2]$

Learn policy (actor) and Q-value (critic) simultaneously

对Q函数求近似的时候，式子和上面的那个导数形式一样，里面的经验奖赏换成了Q值。在求策略梯度



时，Q值是一个常数，是不更新的，它有自己的更新方式，且通常是真实的Q值。

## 控制方差的方法2、引入偏差项 ( bias term )

另一种控制方差的形式，是引入偏差项，只要这个函数是一个只跟状态有关、跟动作无关的函数，它的积分就是0，不影响梯度方向，而会影响梯度的方差。

对于简单的形式，我们可以直接求出来最优的偏差是什么。更一般的形式，我们可以用V值来替代bias。因为V值就是关于状态的估计值，和动作没有关系，所以它带到积分里面去的时候会是0。

把V值带进去，后面的Q就变成了Q-V，叫做Advantage Function，意思指：在这个状态上，V值相当于是一个平均值，Q值指某个动作比平均值高出来多少。用Advantage Function会使得做策略梯度以后，方差控制得比较好，只有当方差控制好了，这类算法才能真正起作用。

### Control variance by introducing a bias term

for any bias term  $b(s)$

$$\int_S d^{\pi_\theta}(s) \nabla_\theta \int_A \pi_\theta(a|s) \pi_\theta(a|s) b(s) \, ds da = 0$$

gradient with a bias term

$$\nabla_\theta J(\theta) = E[\nabla_\theta \log \pi_\theta(a|s) (Q^\pi(s, a) - b(s))]$$

obtain the bias by minimizing variance

obtain the bias by  $V(s)$

advantage function:  $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$

$$\nabla_\theta J(\theta) = E[\nabla_\theta \log \pi_\theta(a|s) A^\pi(s, a)]$$

learn policy, Q and V simultaneously

## 其他改进方法

梯度的改进方法还有Nature Policy Gradient。在监督学习里面，随机梯度是很容易并行的。最近有一些理论的工作，也探讨了它的并行不会影响到它的理论性质。在策略梯度里面，我们同样可以把这个梯度并行来做，这样可以使得它的速度下的很快。

还有对策略直接求导的方法，比如无梯度的优化 ( Derivative-Free Optimization )。这类方法不管强化学习是在做什么，而是直接优化策略里面的参数。优化完参数以后，试一下策略，得出这个值具体是多少。

这样，优化过的算法可以通过总体奖赏值来调整模型里面的参数。通常来说它比用Gradient Policy效率差，由于中间过程是忽略不计的，所以它对特别复杂的问题，反而有比较好的效果，比如俄罗斯方块游戏。

## 六、游戏中的强化学习 ( Reinforcement Learning in Games )

最后一部分，讲一下强化学习和游戏。

为什么讲游戏？一方面，是因为在游戏里面需要克服的一些问题，在真实应用中也常遇到；另外一方面，用游戏来做强化学习任务的成本比较低。

- 游戏推动深度强化学习 ( Deep Reinforcement Learning ) 的发展



2015年，DeepMind在Atari游戏上使用深度网络直接从屏幕图像训练强化学习，直接推动了“深度强化学习”的发展。

用深度神经网络，放在Policy Gradient里面，作为一个策略的模型；或者放在基于值函数的方法里面，作为值函数Q值的一个估计。这样的方法就称为深度强化学习。

## 深度强化学习

其实，深度强化学习里很多工作是在研究怎么让网络更稳定。特别是当输入数据比较少的时候，网络方差的浮动会比较大。这就可以用“延后更新”来解决——如果用深度神经网络，那么每走一步都更新模型会导致模型抖动非常大。而用“延后更新”，例如可以在100步里不更新策略，只是把神经网络更新一下，这个神经网络没有放到新的策略里面来，等神经网络有一个比较稳定的上升以后，再更新策略。还有，积累的数据不要丢掉，也拿出来，让这个神经网络更稳定一点。这两个技巧合起来放在Q-Learning里面，就是DQN。

- Deep Q-Network ( DQN )

DQN可以说是第一个声称深度强化学习算法，可能也是最广为人知的一个。基本上，它的整体结构就是一个函数近似的Q Learning，只不过用CNN做了近似函数。

# Deep Q-Network

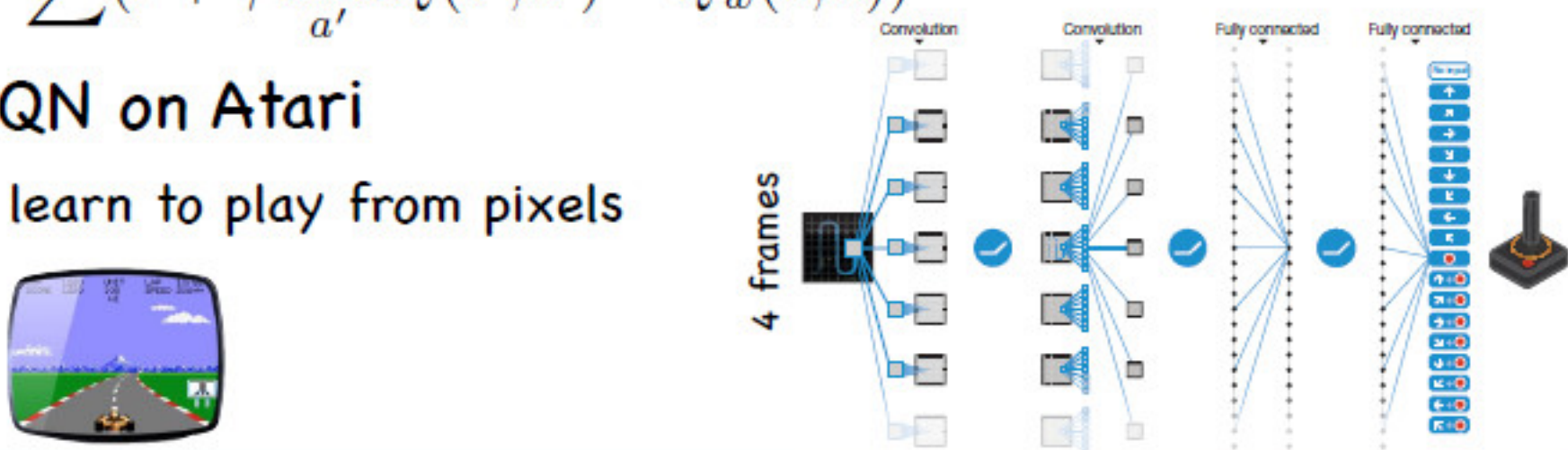
**DQN** [Mnih *et al.* Human-level control through deep reinforcement learning. Nature, 518(7540): 529-533, 2015]

- using  $\epsilon$ -greedy policy
- store 1million recent history (s,a,r,s') in **replay memory D**
- sample a mini-batch (32) from D
- calculate Q-learning target  $\tilde{Q}$
- update CNN by minimizing the **Bellman error** (delayed update)

$$\sum (r + \gamma \max_{a'} \tilde{Q}(s', a') - Q_w(s, a))^2$$

## DQN on Atari

learn to play from pixels



在玩这个游戏的时候，它已经有了100万个记录历史。每次训练神经网络的时候，要抓32个出来训练一次，并且训练完以后不去更新策略，而是在走一定的步数以后，再更新这个策略。除此之外，并不是直接从屏幕上把一帧图像拿进来，而是把历史上好几帧的屏幕拼起来，得到一个当前帧和前面好几帧合起来的一个总体的图作为CNN的输入。不过在最新的一些工作中，这个过程已经被被递归神经网络替代了，不再是把好几层拼起来，而是让好几帧分别输入例如LSTM的网络。

很多运用强化学习寻找策略的游戏已经比人玩得都好了，它玩的好的优势主要体现在反应速度上。但是在需要深入思考逻辑关系的游戏中，强化学习没有人做得好。

我们来看看它的游戏报告结果。

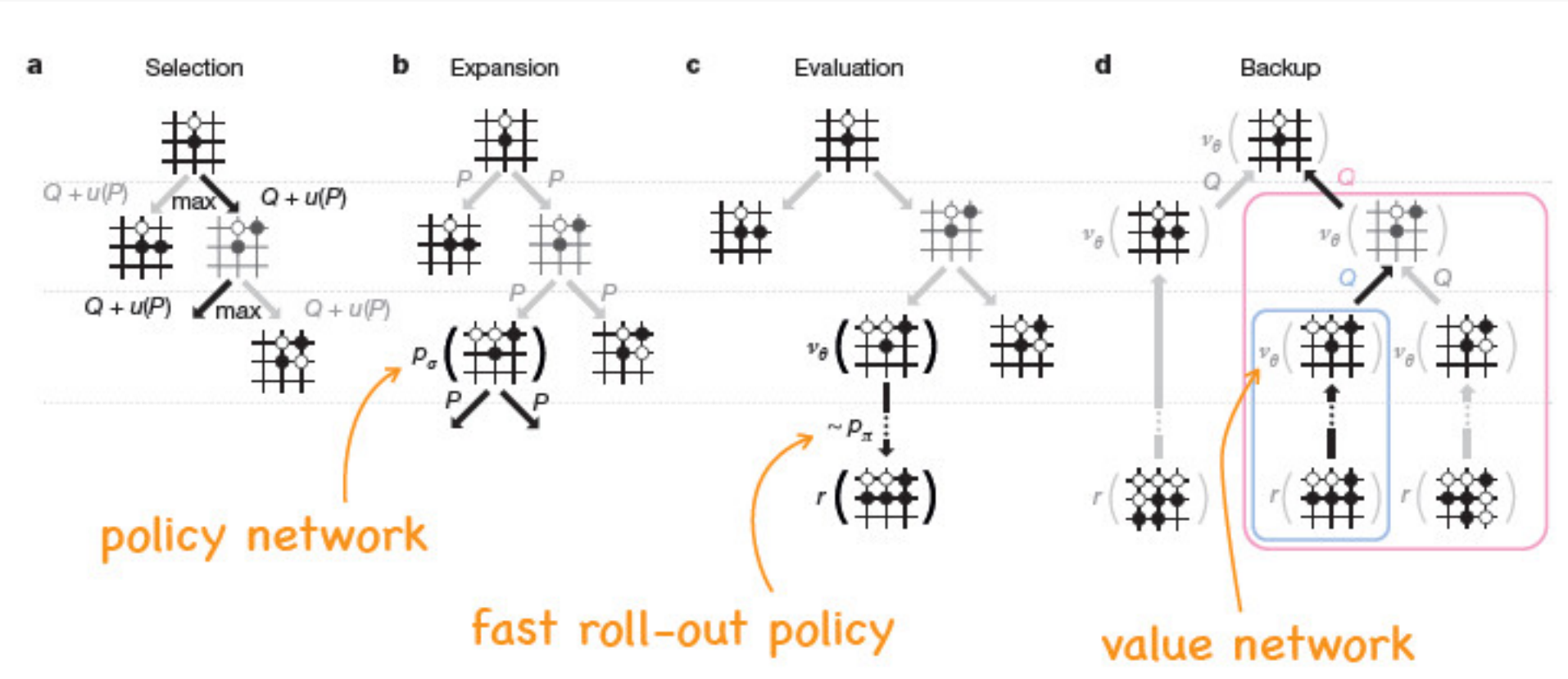


Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

这里面，“with replay”和“without replay”的意思是有没有用到历史数据，“with target Q”和“without target Q”就用了CNN还是线性网络。我们可以看到，神经网络在这里贡献并不是最大的。如果我们只用神经网络而不用replay的话，效果还不如用了replay，但只用线性模型而不用CNN。当然，同时使用深度模型和强化学习是最好的，这可以完成一些过去完成不了的事情。

## 在AlphaGo中的应用

AlphaGo系统的基础框架是蒙特卡洛树搜索，这是经典的树搜索的方法。但是单凭蒙特卡洛树搜索本身并不能取得很好的效果，只用树搜索大概只能达到业余的五六段。AlphaGo里面的一个创新的点就是引入强化学习来改进搜索树的深度和宽度。



这里面用了三个神经网络。

- 第一个policy network，在展开蒙特卡罗树搜索节点的时候起作用。这个网络是用策略梯度方法训练出来的。
- 第二个是一个很小的神经网络，蒙特卡罗树搜索里再往下做很深的搜索时会用到，这样它可以算得很快。这个小的网络是通过监督学习学出来的。
- 第三个网络是用来修正值的。它是通过强化学习中间产生的数据来学习。

由于大家对DQN比较熟悉，所以在尝试深度学习的时候，首先想到的算法大多是DQN。但因为它是一个基于值函数估计的强化学习方法，所以这种方法在稍微复杂一点的应用环境中可能运行不了，大家会感觉用DQN做强化学习效果没那么好。但同样是DeepMin做的围棋游戏，它的强化学习方法已经改成了Policy Gradient，而且以后的很多算法也都是以Policy Gradient为主的，用这种方法处理复杂问题效果更好。

## 在其他游戏上的应用

正是由于在计算机中模拟游戏的代价很低，所以不断有研究者借助游戏来发展强化学习。比如，有用在3D第一人称射击游戏中，可以在这个世界里面行走，并且寻找东西。去年有一个“DOOM”游戏比赛，参赛者要用计算机控制游戏角色，以第一视角进行3D射击。有了强化学习，参赛者就能控制游戏角色，让它做一些动作。由于这个游戏额环境比较复杂，所以在玩游戏的过程中，也发展出了一些创新方法。

例如，在游戏里面，如果让一个强化学习直接到游戏环境里面学习，那它又要捡医疗箱，又要去捡武器等

等，太复杂了。而其中一个团队，就采取了这样的做法：他们让强化学习从简单到复杂，一步一步的去学习——首先学一个策略，比如捡起医疗箱，然后在这个策略的基础上再来学怎么样来开枪、怎么样来射击敌人等等。

实际上游戏里面有很多很高难度的挑战，其中一个非常复杂游戏叫做StarCraft。这个游戏已经有很多年的历史了，现在有不少人，包括DeepMind，都希望在这么复杂的游戏上面能表现出一个比较好的性能，因为这个游戏的复杂度已经大到和很多真实应用的复杂度相当，即使人去学这个游戏，也要花很长时间才能学会。以前用强化学习，只是先取其中一个小问题来解决。比如说我和对方各派三个兵，想办法看这六个兵怎么打。这是一个很局部的战役，但能学到这样的东西也已经比较不错了。如果要学到整盘的打法，它里面涉及到很多问题，第一，它的规模远大于围棋的规模；第二，有很多对手的信息是观测不到的，比如敌方的行动。虽然在今年年初，德州扑克游戏上机器已经打赢了人类玩家，但德州扑克其实是一类很简单的牌类游戏，想让强化学习在大规模游戏任务中，在无法观测到对手信息的情况下，指挥200多个单位做连续的运动，还要持续半个多小时走几十万步，目前还做不好。

## 七、强化学习总结

之前介绍的只是强化学习的其中一小部分，强化学习还包括很多内容：

比如在MDP中如果出现了不可观测的情况，它就不属于Markov了，有一个专门的方向如POMDP来解决这个问题。

还有Learning from Demonstrations，意为人先做出示范，然后从示范数据中教智能体。例如AlphaGo，一开始训练的时候并不是直接上强化学习，而是首先搜集了很多人类对打的数据。

而怎么去设计奖赏函数也会有很多不同的方法。

下面总结一下两个大家比较关心的问题。

- **第一个问题：强化学习是否已经发展成熟？如何选择强化学习问题中的算法？**

如果碰到比较简单的强化学习问题，可以用基于值函数的方法，比如DQN，更复杂的问题可以用Policy Gradient的方法做策略梯度。

但是从目前的发展现状两看，强化学习的成熟度远远不够，也就是说在强化学习领域，还有很大的提升的空间，有可能能做出一个性能更好的全新的算法。但大规模的问题现在还是很难解决。这个大规模指是它的状态空间大，并且步数特别多。

- **第二个问题：在实际领域应用强化学习，会遇到什么瓶颈？**

1、 强化学习需要探索，在很多场景带来风险。

以推荐股票为例。我本来已经有一个还可以的推荐策略，每天能给我带来100万的收入。但是现在为了训练强化学习，要做探索，尝试一些随机的股票。假如告诉你这个探索会导致今天一下子要损失好几百万，而一个月以后可以赚回1个亿，那你就要衡量一下这里看面的风险有多高，敢不敢用了。

2、 为什么强化学习在很多游戏上面用的比较多？

游戏在计算机中运行，速度高、代价低。如果放到现实世界中来运行，比如放在推荐系统线上运行，那它就必须和真实的环境打交道。它的学习过程需要不断探索，而部署在真实环境里可能会遇到很多麻烦，如果能有一个比较好的模拟器，就可以减少这些麻烦；另外，如果有比较好的监督学习数据的话，也可以做一个初始的策略，不过这个策略可能一开始起点要稍微高一点。做机器人一般也有一个机器人模拟器，所以一般先在模拟器里面做，做好策略再放到机器人身上来学。但是其他现实世界问题，在模拟器里可能就没有那么好做了。

## 八、强化学习资源推荐书籍



强化学习的书不多，最经典的书是Richard S. Sutton的教科书；Masashi Sugiyama的书属于专著；Reinforcement Learning: State-of-the-Art属于文集，覆盖面比较广，但需要读者有一定基础；还有一些讲述MDP的书；另外，在机器学习的书里面也会提到强化学习。

## Books



Richard S. Sutton and Andrew G. Barto  
Reinforcement Learning: An Introduction



Masashi Sugiyama  
Statistical Reinforcement Learning:  
Modern Machine Learning Approaches



Marco Wiering and Martijn van Otterlo (eds)  
Reinforcement Learning: State-of-the-Art

## Also in MDP books



Mykel J. Kochenderfer  
Decision Making Under Uncertainty:  
Theory and Application

## and machine learning books



周志华  
机器学习

## 线上资源

OpenAI Gym：一个基础的强化学习平台，里面很多环境，研究人员可以在上面做实验，它对这个领域有很大的促进。还有AlphaGo技术负责人David Silver的线上教学视频，讲的非常好。

## Online resources

OpenAI Gym Reinforcement Learning toolkits

<https://gym.openai.com>

Awesome-RL <https://github.com/aikorea/awesome-rl>

Resources at MST [http://web.mst.edu/~gosavia/rl\\_website.html](http://web.mst.edu/~gosavia/rl_website.html)

Lectures by David Silver

<http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html>

## 论文发表地

强化学习论文主要发表在AI期刊和会议上，期刊有Artificial Intelligence, JAIR, JMLR, Machine Learning, JAAMAS等，会议有IJCAI, AAAI, NIPS, ICML, ICLR, AAMAS, IROS等等。

以上就是俞扬博士的演讲，更多内容请继续关注雷锋网。

雷锋网原创文章，未经授权禁止转载。详情见[转载须知](#)。






## 相关文章

强化学习	马尔可夫决策	马尔可夫决策过程	神经网络
			
One-Page AlphaGo --十分钟看懂 AlphaGo 的核心	美国罗德岛大学杨庆教授：如何把机器学习技术应用于	南京大学俞扬博士万字演讲全文：强化学习前沿（上）	如何让强化学习采样变得更简单？剑桥大学联合谷歌伯
			
解析   Facebook田渊栋：德州扑克上战胜人类的AI究	Facebook的数据预测工具 Prophet有何优势？用贝叶	小米Max 2发布，雷军说这是大号 iPhone 7 Plus	售价 3 万 9，微软 HoloLens 国行开卖；

## 文章点评：

我有话要说.....

☐ 同步到新浪微博

提交



## 热门关键字

热门标签 微信小程序平台 微信小程序在哪 CES 2017 CES 2016年最值得购买的智能硬件 2016 互联网 小程序 微信朋友圈 抢票软件 智能手机 智能家居 智能手环 智能机器人 智能电视 360智能硬件 智能摄像机 智能硬件产品 智能硬件发展 智能硬件创业 黑客 白帽子 大数据 云计算 新能源汽车 无人驾驶 无人机 大疆 小米无人机 特斯拉 VR游戏 VR电影 VR视频 VR眼镜 VR购物 AR 直播 扫地机器人 医疗机器人 工业机器人 类人机器人 聊天机器人 微信机器人 微信小程序 移动支付 支付宝 P2P 区块链 比特币 风控 高盛 人脸识别 指纹识别 黑科技 谷歌地图 谷歌 IBM 微软 乐视 百度 三星s8 腾讯 三星Note8 小米MIX 小米Note 华为 小米 阿里巴巴 苹果 MacBook Pro iPhone Facebook GAIR IROS 双创周 云栖大会 智能硬件公司 智能硬件 QQ红包 支付宝红包 敬业福 支付宝敬业福 支付宝集五福 Waymo 虚拟现实 深度学习 人工智能 中国银联 蚂蚁金服 WRC CNCC prisma airbar 5g ipod classic es 9038 爱因斯坦的大脑 工业无人机 三星gear s 吴碧瑄 skillshare airpods发售时间 何凯明博士 智能家居 今年零元快的挣钱行业 pixel 评测 更多



