深度强化学习初探

杨敬文

2016年年初备受瞩目的围棋 “人机大战”，以人类围棋冠军被血虐落下帷幕。这只谷歌DeepMind团队开发的围棋机器人阿法狗不仅赚足了眼球，更是掀起了一波关于人工智能的讨论狂潮。现在好像作报告还是写文章都要把阿法狗提一下才能紧跟时代潮流啊（好像也自黑了一下）。其实人家DeepMind不光是下围棋的，在他们的主页上写着大大的 “Solve Intelligence”。要“Solve Intelligence”光会下围棋可不行，还得能像人一样能干各种各样的事情。究竟有些什么样的技术能够作为支撑，使得他们敢说出这样的话呢？见识了阿法狗之后，可能很多人记住了深度学习，还有人记住了蒙特卡洛树。其实还有个叫深度强化学习的东西也是DeepMind一直在重点研究的领域。他们在这个方向上做了很多工作。这篇文章，我们抛开阿法狗，来简单看看DeepMind提出的被他们称之为**深度强化学习**的东西大概是怎么一回事。

大家一看到深度强化学习，大概会认为现如今深度学习这么火，什么东西都喜欢拿深度学习去套，也是个跟风的产物吧。当然我们可以这么认为。但是在这个套的过程中DeepMind套得恰到好处，大大扩大了强化学习的应用范围。

深度学习大家可能都比较了解，简单说来就是在多层神经网络的结构下，辅以结构设计和各种梯度技术，能够对比如图像分类之类的问题有很好的效果。它的优点在于不仅能够提供端到端的解决方案，而且能够提取出远比人工特征有效得特征向量。

而强化学习，大家可能就比较陌生了。但是要是说起波士顿动力，大家可能就又知道了，前段时间被刷屏的机器人，凭借出色的平衡性给大家留下了深刻的印象。像机器人控制这类领域就大量的使用了强化学习技术。除此之外，游戏领域，比如棋类游戏，甚至用户个性化比如推荐等领域都有应用。

什么是强化学习？

强化学习其实也是机器学习的一个分支，但是它与我们常见的机器学习（比如监督学习supervised learning）不太一样。它讲究在一系列的情景之下，通过多步恰当的决策来达到一个目标，是一种**序列多步决策**的问题。举一个周志华老师在《机器学习》【8】中种西瓜的例子来帮助大家理解。种瓜有很多步骤，要经过选种，定期浇水，施肥，除草，杀虫这么多操作之后最终才能收获西瓜。但是，我们往往要到最后收获西瓜之后，才知道种的瓜好不好，也就是说，我们在种瓜过程中执行的某个操作时，并不能立即获得这个操作能不能获得好瓜，仅能得到一个当前的反馈，比如瓜苗看起来更健壮了。因此我们就需要多次种瓜，不断摸索，才能总结一个好的种瓜策略。以后就用这个策略去种瓜。摸索这个策略的过程，实际上就是强化学习。可以看到强化学习有别于传统的机器学习，我们是不能立即得到标记的，而只能得到一个反馈，也可以说强化学习是一种**标记延迟的监督学习**。

通过这个种瓜的过程能够看出来，强化学习实际上和我们人类与环境的交互方式类似。是一套非常通用的框架，可以用来解决各种各样的人工智能的问题。

总结起来，强化学习的目标就是要寻找一个能使得我们获得最大累积奖赏的策略。为啥是累积奖赏？你之前不是说种个好瓜就可以了嘛。但是种瓜得浇水施肥啊，比如你在一个资源稀缺的地方，你当然就会希望我少用资源，又能种出好瓜。因此这样的定义适用性更广。

通常呢，累积奖赏表示为

$$J_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots$$

其中$\gamma(0 <= \gamma <= 1)$为折扣系数，t表示第几步。引入折扣系数这个东西，一方面是处于数学证明方便（预知详情可以去读强化学习的教科书），另一方面也是为了拓宽适用性。比如大家做买卖都喜欢现钱，因为夜长梦多，比如钱之后贬值了，那不就相当于少赚了嘛。

策略一般是用$\pi(s) = a$来表示，也就是在状态s下，根据π策略，我们采取a动作。有了策略的定义，我们还得知道一个策略到底好不好？最直接的就是用一个值函数来评估，值越高越好呗。比如可以定义一个值函数$Q^{\pi}(s,a)$，它表示在策略π下，从状态s出发，执行动作a所带来的累积奖赏。

怎么求解强化学习？

这两个基本的概念实际上给我们指了两条可以求解强化学习的明路。一种就是直接去寻找一个好的策略（这不是废话嘛）。另外一种就是能求解一个最优的值函数，这个最优的值函数能够告诉我们在这个状态下，选取哪个动作能得到的期望值最高，不管在什么状态，我们总是能从值函数表那里查表获得应该选取哪个动作。这个表实际上也就可以作为策略了。

我们首先来看看怎么直接寻找策略。前面也提到了策略实际上就是一个从状态到动作的一个映射，那么就可以用个参数θ的模型去近似它（表示为$a = \pi_\theta(s)$）。既然目标就是要让累积的奖赏最大，我们只要以这个目标求梯度$\nabla_\theta J$，按照梯度不断更新θ值，就可以最终得到期望的策略。

然后再来看看基于值函数的方法。几乎所有的强化学习的模型都是定义在**Markov决策过程**上的。这个是什么呢？其实就是包含了上面我们说的那些状态，动作，奖赏，以及影响状态转移的概率，在这个模型下，未来的状态只与当前状态有关，与以前的没有关系（即Markov性质）。一个Markov决策过程可以表示为

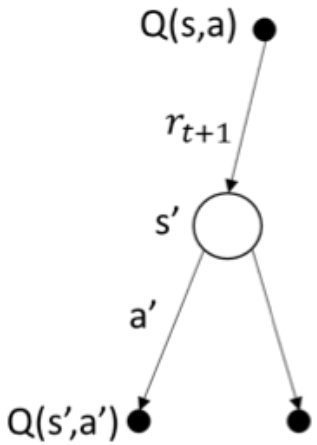
$$s_0, a_0, r_1, s_1, a_1, r_1, \ldots$$

基于这个假设，很容易得到值函数是满足一个递归式子的。

$$\begin{aligned} Q(s,a) &= E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t = s, a_t = a] \\ &= E[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \cdots) | s_t = s, a_t = a] \\ &= E[r_{t+1} + \gamma Q(s', a') | s_t = s, a_t = a] \end{aligned}$$

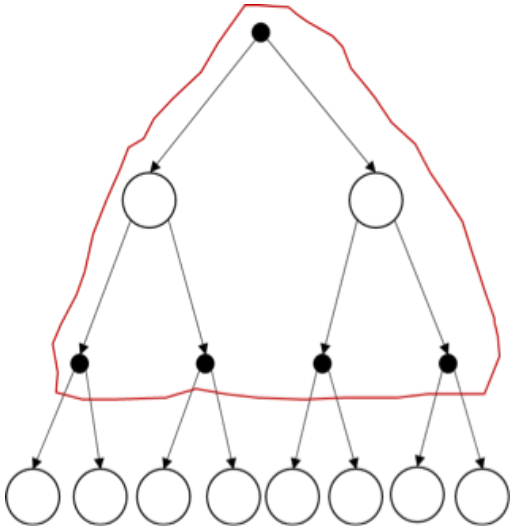
最优的值函数也是一样

$$Q^*(s,a) = E[r_{t+1} + \gamma Q^*(s',a')|s_t = s, a_t = a]$$



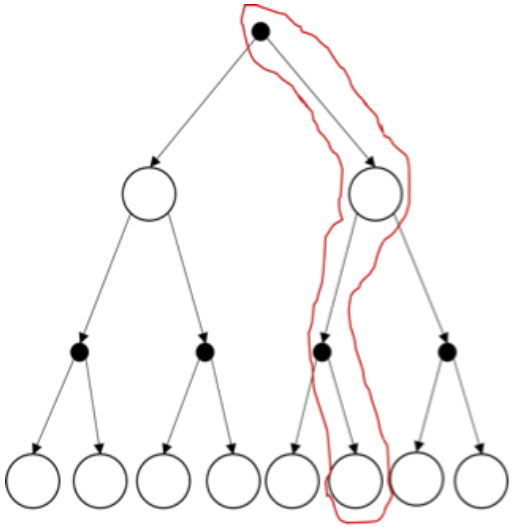
这就是著名的**Bellman Equation**，它是求解值函数的关键。

最优的Bellman Equation指导我们通过不断迭代就能够得到最优的值函数。如图所示，一个实心点就是值函数Q，白圈表示状态。上述的Bellman Equation实际上可以按照Markov决策过程展开成一个树状的结构，我们不断向前看，探索后面的情况，然后将这些得到的奖赏综合起来。然后就能得到一步一步往状态动作值函数里填。熟悉动态规划的同学，对于这种迭代求解的方法应该不会陌生。



如果我们对于这个环境了如指掌，那些状态之间的转移概率我们都清清楚楚，那么所有的转移都可以被展开，如图所示。利用动态规划就可以求得值函数的精确解。

但是大自然是伟大的，我们不是上帝，基本上不可能吃透它。转移概率这些东西是无从得知了，因此我们做不到像上面那样对所有的状态和动作进行展开。那怎么来算值函数呢？还是像种瓜一样，一次可能种不好，但是可以多种几次，来总结经验。求取平均的累积奖赏来作为期望累积奖赏的近似就可以估算值函数。

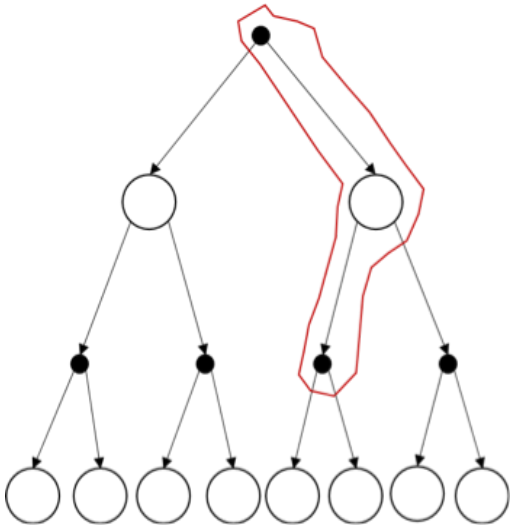


具体说来，也就是我们从起始状态出发，使用某种已掌握的策略进行探索，不断执行该策略，来得到一个完整的轨迹，如图所示红色标记的部分，记录其奖赏之和，作为一次的累积奖赏的采样值。多次采样之后，我们就可以根据这些轨迹得到一个估计。

有时候我们的策略可能是固定的，那么不是所有的轨迹都一样了嘛，或者策略总是一个很高的概率告诉我们执行某个动作，这样我们其实就是一直在认知范围内做最好的，那采样再多也没用。世界太大，我们需要出去看看，可能在未知的地方，有很重要的东西，就像哥伦布发现新大陆一样。

因此为了不做井底之蛙，能更好的获得估计值，通常会加一点“扰动”，比如ϵ的概率从所有动作中均匀随机选取一个，以$1-\epsilon$的概率选取当前策略给出的最优动作。通常将这种加了这种扰动的策略记为π^ϵ。(这只是一个简单的处理方法，其实也有很多研究，感兴趣的同学可以去看看**exploitation & exploration dilemma**)

但是注意到这里为了获得轨迹，瓜得全部种好啊。现在21世纪，时间就是金钱啊，哪有这个闲工夫种那么多瓜。话说不是有Markov假设嘛，能不能我种一点总结一点？答案是肯定的。



我们每次不走完，就只向前看一步，然后估计出一个值函数的目标。对比当前的值函数做改进。如下图所示，将只向前看一步的作为样本。假设已经有m个样本估计出一个$Q_m(s_t,a_t)$，如果现在有了第$m+1$个样本，那么就可以在原来$Q_m(s_t,a_t)$的基础上加上基于这m个样本估计的(s_t,a_t)最优Q值与当前Q值的差就能起到更新Q值的作用了。一轮中，我们实际上就只使用了一个样本来更新。

表达起来就是下面这个式子来进行增量式的学习

$$Q_{m+1}(s_t,a_t)=Q_{m+1}(s_t,a_t)+\alpha[r_{t+1}+\gamma Q_m(s_{t+1},a_{t+1})-Q_m(s_t,a_t)]$$

其中α起到一个控制学习步长的作用。

估计出值函数之后，要得到最优策略，还需要改进策略去达到这个估计出来的最优值函数。说来也简单，我们就在每个状态，枚举一下动作集合，然后选一个能达到最大值函数的动作就可以了

$$\pi(s)=argmax_aQ(s,a)$$

这里简单的放一个算法来描述这个过程，这就是强化学习中非常常用的**Q-Learning**。通过前面的铺垫，这个算法大家应该不难看懂。

输入：环境E;
动作空间A;
起始状态s_0;
奖赏折扣γ;
更新步长α.
过程：
1.$Q(s,a)=0, \pi(s,a)=\frac{1}{|A(s)|}$
2. $s=s_0$
3. **for** m = 1,2,... **do**
4. $r, s_{t+1}=$在E中执行动作$\pi^{\epsilon}(s)$产生的奖赏与转移的状态
5. $a_{t+1}=\pi(s_{t+1})$
6. $Q_{m+1}(s_t,a_t)=Q_m(s_t,a_t)+\alpha(r+\gamma Q_m(s_{t+1},a_{t+1})-Q_m(s_t,a_t))$
7. $\pi(s_t)=\arg max_{a''} Q(s_t,a'')$
8. $s=s_{t+1}, a=a_{t+1}$
9. **end for**
输出：策略π

可以看到，到目前为止，我们这个值函数似乎就是一张表啊，状态和动作好像都是离散的。如果状态是连续的怎么办？如果动作也是连续的怎么办？这种情况很常见啊，因为本来世界就是连续的。最简单的可以做离散化。但是连续空间离散出来怕是空间有点大，你这个表更新起来也麻烦，而且泛化性能可能也有问题。

一探深度强化学习

对于连续状态问题，科学家们就说了，可以用函数来近似这个连续空间，比如对于Q函数，可以将其表示为

$$Q(s,a)=[F(\theta)](s;a)$$

其中$F(\theta)$表示参数为θ的函数，只要能够求解这个θ就能近似值函数，进而得到策略。我们将上面Q-Learning里的Q函数用这个式子代换即可。这里函数可以是线性的也可以是非线性的，通常线性的函数由于更容易分析理论结果，经常被用在强化学习里面。即$Q(s,a)=\theta^T(s;a)$。

当然也可以用一个神经网络去代替值函数（假设网络权值向量ω，那么值函数可以表示为$Q(s,a,\omega)$），特别现在深度学习那么强，能够刻画很多高维的（图像）状态，如果换成深度网络能够带来很多好处（这里总算有点深度强化学习的意思了）。如何来确定网络的权值来逼近值函数

呢？

可以将更新Q值的目标函数设置为Q-learning中更新的差值的均方差

$$\ell(\omega) = E[(r + max_{a'}Q(s', a', \omega) - Q(s, a, \omega))^2]$$

然后就可以进行使用梯度下降来求解了

$$\frac{\partial \ell(\omega)}{\partial \omega} = E[(r + max_{a'}Q(s', a', \omega) - Q(s, a, \omega))\frac{\partial Q(s, a, \omega)}{\partial \omega}]$$

参照Q-Learning，它实际上每一步就用了一个样例来更新Q值，在这里计算梯度的时候，可以只用一个样例也可以使用mini-batch的方式来进行更新。

在Q-learning中使用神经网络的做法其实之前就有人想到了，但是效果并不好，因为一方面那时候深度学习的技术还没有现在这么成熟，变成个神经网络也会比其他模型更好。而另一方面也是最关键的问题来源于强化学习的特殊性。这个特殊性体现在两个方面，一个是它的数据是序列产生的，这样子数据之间并不满足机器学习中通常假设的独立同分布的性质，这样我们进行随机梯度下降，其实并不随机。套一个机器学习算法当然效果不好。另一个是，机器人采取什么动作实际上会影响环境的，好比走路，左转是阳关道，右转是独木桥，场景大不相同。这意味着后面产生数据的分布也在变化，训练出来的策略可能非常不稳定，产生震荡的情况。

但是DeepMind提出了简单而有效的技巧解决这两个问题。对于第一个问题，他们用了一个叫**Replay Memory**的东西记为D，就是字面含义，把产生的数据都往里面扔，然后要计算梯度的时候，就像看回放一样，从D里面随机采样。这样就使得数据之间关联性得到了打破。

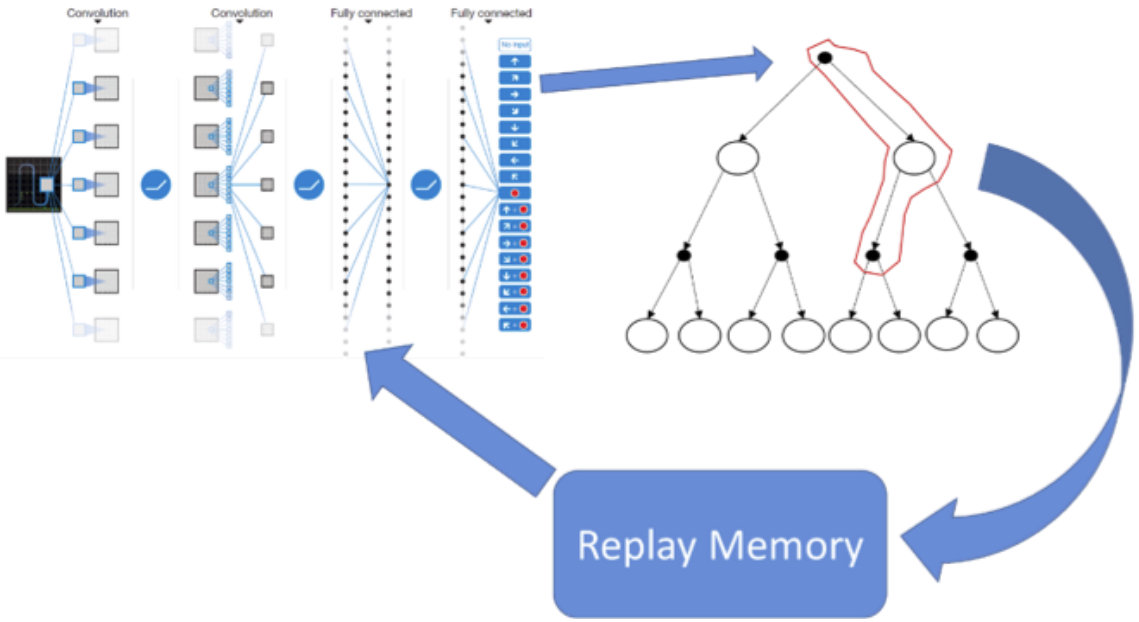
$$\ell(\omega) = E_{s,a,r,s' \sim D}[(r + max_{a'}Q(s', a', \omega) - Q(s, a, \omega))^2]$$

对于第二个问题，他们说，先随机设定一个网络权值，并且固定住，即一直为ω^-，慢慢优化，稳定了，再更新，再优化。这样监督信息实际上在很长的时间内是不变的，达到了一种监督学习的感觉。

$$\ell(\omega) = E[(r + max_{a'}Q(s', a', \omega^-) - Q(s, a, \omega))^2]$$

他们将这一套东西拿来玩Atari游戏机【1】，这是个风靡80年代的游戏机。画面就和马里奥差不多，像素一格一格的，很多经典游戏比如打砖块都是出自这个游戏机。游戏机本身很简单，就是一个摇杆，有18种操作。他们直接让游戏画面作为状态描述，摇杆作为动作集，游戏中的得分作为奖赏，搞了个深度神经网络来学习值函数。

我来给大家画一张图，大家一看就明白了，实际上就是游戏画面通过一个深度网络变成状态，然后按照Q-learning的流程会产生(s_t, a_t, r_t, s_{t+1})这样的数据，我们将这些数据扔进Replay Memory中。训练网络时，以Q-



learning的中的差作为目标函数，从Replay Memory中进行采样，进行mini-batch的梯度求解。

最后效果还不错，在40个游戏中，比传统的强化学习方法都好，有29个达到了人类75%的得分。 我们来看看他们提出的这两个技术到底起了多大的作用。从他们提供的统计表中我们可以看到replay贡献挺高，两个结合起来确实有奇效

	Q-learning	Q-learning + Target Q	Q-learning + Replay	Q-learning + Replay + Target Q
Breakout	3	10	241	**317**
Enduro	29	142	831	**1006**
River Raid	1453	2868	4103	**7447**
Seaquest	276	1003	823	**2894**
Space Invaders	302	373	826	**1089**

到这里好像一副就可以结尾了的样子。然而并没有。

二探深度强化学习

可以看到这个地方动作集是离散的，并且也就只有18种。前面我们也提到了，动作难道不能是连续的么？当然是可以的，比如你开车的时候，方向盘就不可能只有左右，是个关于角度的连续值啊。 那这些东西可以做连续动作的嘛？不可以。再来看改进策略的方式，这需要从动作集里面枚举的。这显然对连续情况是没法处理的。如果离散化呢？那可能是一个非常大的动作空间，那每一步改进的计算代价都会非常大。

$$\pi(s) = \arg\max_a Q(s,a)$$

那么怎么处理呢？大家别忘了求解策略的是两条路的。我们前面讲的都是基于值函数的方法。我们还能直接求策略的梯度啊。这里策略是一个单独的函数了，至于做什么动作，把状态带进入就可得到。这里自然的就可以处理连续的动作了，不涉及上面那个改进策略的式子。

由于采用梯度方法，因此这种求解方式有很好的收敛性保证。当然另一方面也是容易陷入到局部最优解。由于没有枚举动作改进策略的这个步骤，它不像值函数方法有对动作天然的泛化性，用人话说就是对于没有见过的动作，它就不知道是啥了。

看起来两种方法各有优势啊，能不能相互取长补短呢？于是聪明的科学家提出了一种叫做**actor-critic的框架**，也就是分为actor和critic两个部分。简单说来，critic就是类似之前说的那种Q-Learning那种值函数的方法，但是我们这里主要用它来评估策略，而actor就是梯度方法，我们用它来改进策略。这样子critic使得策略的评估得到了改善，能给actor一个更好的梯度估计值，能改善局部最优的问题，actor避免了值函数中低效的值估计过程，同时也能应对连续动作空间。

当然这两个部分也都是可以用深度神经网络来取代的，也就能自然的结合深度学习的优势了。

当然DeepMind不仅仅就是换个网络上去这么简单。他们更进一步，证明了在一些情况下，存在着决定性策略梯度而且具有很利于估计的形式。这个在之前普遍认为是不存在的。那这个决定性策略梯度到底是什么意思呢？之前算梯度的式子是（其中ρ^π是状态分布）

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

如果采用随机策略梯度的求法，需要对状态，动作进行采样的，如果动作空间大了，那么采样需要的样例也就多了。 他们证明可以这样【3】

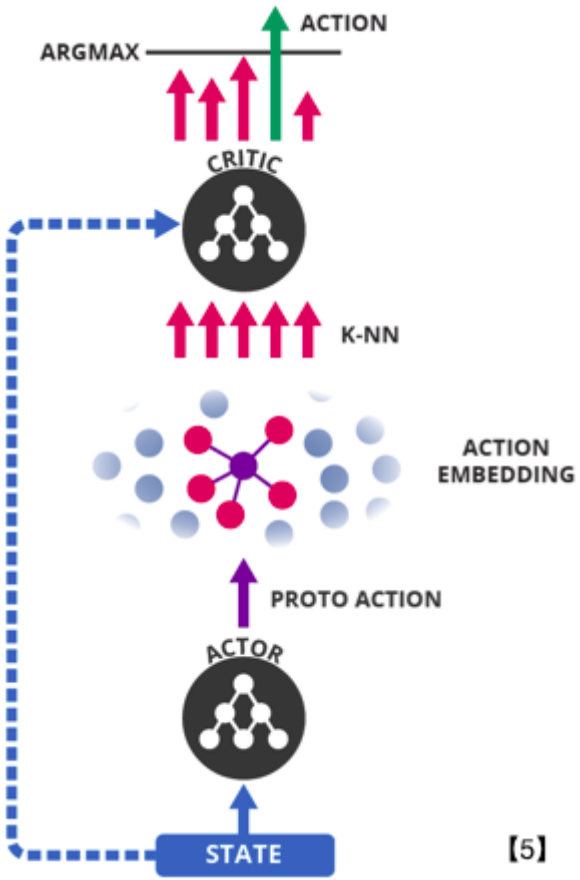
$$\nabla_\theta J(\pi_\theta) = E_{s\sim\rho^\pi}[\nabla_\theta \pi_\theta(s) \nabla_a Q^\pi(s,a)|_{a=\pi_\theta(s)}]$$

不要看式子好像很复杂，我们看求期望的部分，发现没有？我们不用管动作空间了，这极大的提高了在连续空间中梯度的估计效率。

同样加上前面提到的回放和固定目标的技术，深度强化学习不仅可以处理连续状态空间，也可以处理连续动作空间【4】。

三探深度强化学习

人类总是容易不满足，有些问题就是有非常大的离散动作空间，你光能处理连续的不行啊。如果动作有千千万。前面这些东西就又跪了。但是DeepMind的科学家想啊，上面这套处理连续的不是非常棒嘛，那我就把离散动作空间先变成连续的，用上面这一套来搞，然后再把连续都又



【5】

变回离散的动作，这不就皆大欢喜了。具体说来来呢，他们提出了一个叫**Wolpertinger的框架**（wolpertinger是一种欧美传说中的动物，有松鼠的身子，兔子的头，鹿的角，野鸡的翅膀……不知道他们是不是觉得他们这个框架杂糅了各种东西，所以起这么一个名字），框架大概就是图上这样。

再具体一点，就是将原来的策略函数变成这样，

$$f_{\theta^\pi}: \mathcal{S} \rightarrow R^n$$

$$f_\theta^\pi(s) = \hat{a}$$

先将状态与连续的动作空间对应，得到一个属于连续空间的“原型”动作(proto-action)。但是这个动作不一定合法，因此我们需要再把它映射到原动作空间去。其实就是寻找k个最接近的原始动作

$$g: R^n \rightarrow \mathcal{A}$$

$$g_k(\hat{a}) = \arg\min_{a \in \mathcal{A}} |a - \hat{a}|_2$$

为了鲁邦一点呢，就用值函数评估一下，选那个值最大的动作就ok了，

$$\pi_\theta(s) = \arg\max_{a \in g_k \circ f_{\theta^\pi}(s)} Q_\theta Q(s,a)$$

框架还是actor-critic的框架，多了这个离散到连续，连续到离散的转换，也就能够处理大规模的离散问题了。

大家这时候可能会有点纳闷，到底什么样的问题需要这么大的离散动作空间呢。实际上这样的问题还是挺多的。比如在推荐系统中，可以假设有一个机器给用户作推荐，推荐物品的集合就是动作的集合，每个用户可能会接受这个推荐或者不接受，接受了就相当于，这个动作带来的状态转移获得了正反馈。所以人的行为就是环境的反馈。由于现实生活中人的行为会随着时间一直变化着的，目前很多推荐系统都难以解决这一点。而强化学习的这种与环境不断交互，以谋求长期利益的模式很容易将这种情况建模起来。因此强化学习在这类问题上也是有其优点的。

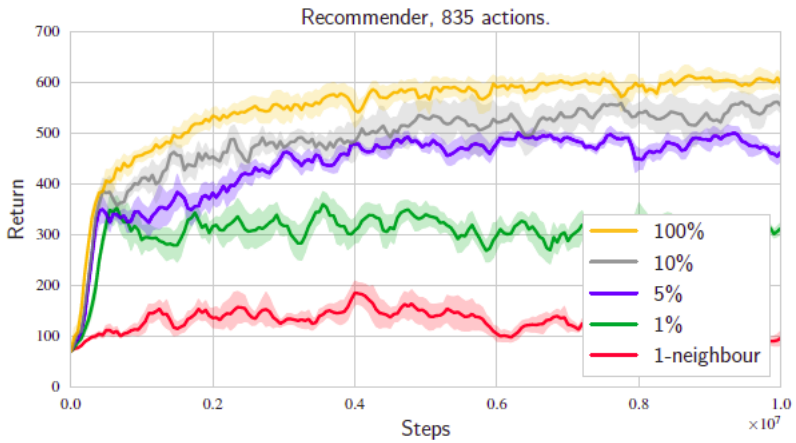


Figure 6. Performance on the 835-element recommender task for varying values of k, with exact nearest-neighbor lookup.

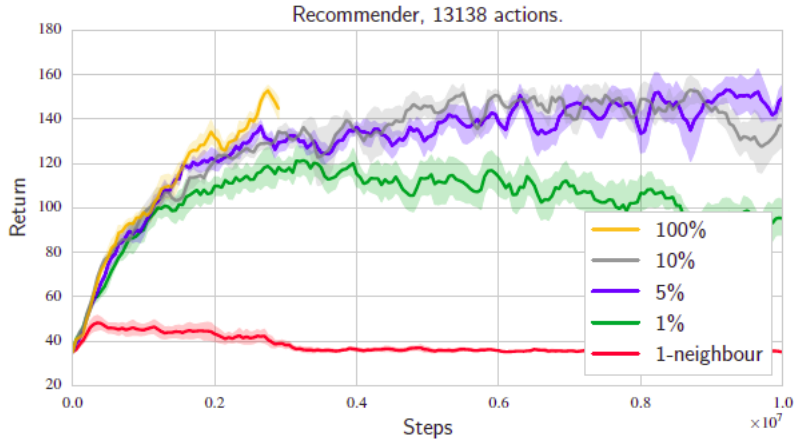


Figure 7. Agent performance for various numbers of nearest neighbors on 13k recommender task. Training with $k = 1$ failed to learn.

# Neighbors	Exact	Slow	Medium	Fast
1	31	50	69	68
1% – 131	23	37	37	37
5% – 656	10	13	12	14
10% – 1,313	7	7.5	7.5	7
100% – 13,138	1.5	1.6	1.5	1.4

Table 3. Median steps/second as a function of k & FLANN settings on the 13k recommender task. 【5】

可以看到当考虑近邻数为10%的时候能有比较好的近似，在第二张图中，如果考虑全部动作空间(100%)，已经跑不动了。

在速度上来看，下面这个表表示每秒钟能够进行的步数，越多表示越快。可以看到10%还是比全部考虑快很多的。因此通过他们这个离散到连续，连续再到离散过程，大规模离散动作空间的问题也能够得到解决了。

说到这里，基本把深度强化学习的几个方面都介绍了一下。我们可以看到，强化学习实际上是一套很通用的解决人工智能问题的框架，很值得大家去研究。另一方面，深度学习不仅能够为强化学习带来端到端优化的便利，而且使得强化学习不再受限于低维的空间中，极大地拓展了强

化学习的使用范围。

我觉得一句话能够很好的概括深度强化学习**“Reinforcement Learning + Deep Learning = AI”**，这是DeepMind中深度强化学习领头人**David Silver**写在他的slides上的。就用它作为结尾吧。

参考文献：

【1】 Playing Atari with Deep Reinforcement Learning(13’ NIPS Deep Learning Workshop)

【2】 Human Level Control Though Deep Reinforcement Learning(15’ Nature)

【3】 Deterministic Policy Gradient Algorithms(14’ ICML)

【4】 Continuous Control With Deep Reinforcement Learning(16’ ICLR)

【5】 Deep Reinforcement Learning in Large Discrete Action Spaces (ArXiv :1512.07679V2)

【6】 L. Busoniu, R. Babuska, B.D. Schutter and D. Ernst, Reinforcement Learning and Dynamic Programming using function approximators

【7】 R.S. Sutton and A.G. Barto, Reinforcement Learning: An Introduction

【8】 周志华，机器学习

Page generated 2016-05-17 18:50:31 CST, by jemdoc+MathJax.