第十四章、强化学习

赵涵

2023 年 5 月 11 日

强化学习并不是一个很新的领域。1954年Minsky首次提出“强化”和“强化学习”的概念和术语。1965年在控制理论中Waltz 和傅京孙也提出这一概念，描述通过奖惩的手段进行学习的基本思想。他们都明确了“试错”是强化学习的核心机制。1957 年，Bellman提出了求解最优控制问题以及最优控制问题的随机离散版本马尔可夫决策过程（Markov Decision Process，MDP）的动态规划（Dynamic Programming）方法，而该方法的求解采用了类似强化学习试错迭代求解的机制。尽管他只是采用了强化学习的思想求解马尔可夫决策过程，但事实上却导致了马尔可夫决策过程成为定义强化学习问题的最普遍形式，加上其方法的现实操作性，以致后来的很多研究者都认为强化学习起源于Bellman的动态规划，随后Howard 提出了求解马尔可夫决策过程的策略迭代方法。经过沉寂了了一段时间后。1989年，Watkins提出的Q 学习进一步拓展了强化学习的应用和完备了强化学习。Q学习使得在缺乏立即回报函数（仍然需要知道最终回报或者目标状态）和状态转换函数的知识下依然可以求出最优动作策略，换句话说，Q学习使得强化学习不再依赖于问题模型。Watkins和Dayan 发表在1992年的论文分析了Q学习的收敛，还证明了当系统是确定性的马尔可夫决策过程，并且回报是有限的情况下，也一定可以求出最优解。1994年的论文改进了Q 学习算法的收敛分析。至今，Q学习已经成为最广泛使用的强化学习方法。

1 基本概念

强化学习的刚开始的困难，就在于概念较多，作为初学者，应该把注意力多放在概念的理解上，在理解透彻概念的基础上，再去看算法，就不会有很大的困难。接下来，我们一一介绍强化学习的相关名词与对应的解释。

随机过程（Stochastic Process）：我们简单来说，随机过程是与时间相关的，一系列随机变量构成的一个序列，比如一个随机变量为S，它在t 时刻的取值为s_t，那么这个随机变量从$t = 1$时刻，我们都对它进行一个观测，随着时间流逝，我们就会得到一个随机变量的一个观测序列：$s_1, s_2, ...s_{t-1}, s_t, s_{t+1}, ...$，我们假设这个过程持续到无穷远，那么我们用$\{s_t\}_{t=1}^{\infty}$来表示。

智能体（agent）：强化学习的主体被称为智能体。通俗地说，由谁做动作或决策，谁就是智能体。比如在超级玛丽游戏中，玛丽奥就是智能体。在自动驾驶的应用中，无人车就是智能体。

环境（environment）：与智能体交互的对象，可以抽象地理解为交互过程中的规则或机理。在超级玛丽的例子中，游戏程序就是环境。在围棋、象棋的例子中，游戏规则就是环境。在无人驾驶的应用中，真实的物理世界则是环境。

马尔科夫链（Markov Chain）：一个随机过程，后面的观测值，是取决于前面的观测值的。比如说，一年前在学习日志上立下了一个目标，写下：我要发表一篇《nature》。那么每周对我的学习日志进行一个观测，到年末，可能第52次观测值，依旧没有发表《nature》，但是第52 次的观测值，或多或少都受到第一次观测学习日志的影响。再进一步来说，在第一次之后的观测，都会受到第一次观测值的影响。而马尔科夫链，是说，这一次的观测值仅依赖于上一次，与上一次之前的观测值**无关**。

用数学语言来表示，就是：$P(S_{t+1}|S_t, S_{t-1}, ..., S_1) = P(S_{t+1}|S_t)$。我们把这种性质称为马尔科夫性质（Markov Property）。

状态空间（State Space Model）：对随机变量的观测值构成的序列，就构成了状态空间。简单说就是马尔科夫链加观测，最典型的例子就是隐马尔科夫过程。在之前讲解概率图模型时，我们已经详细介绍过了，再此不再赘述。

马尔科夫奖励过程（Markov Reward Process）：所谓的奖励过程，就是每观测到一个状态，我们就会得到一个奖励。比如我在观测每周学习日志后，如果这周的进展很大，就去吃一顿大餐，如果没有进展，就惩罚自己去跑10公里。总的来说就是马尔科夫链加上奖励，注意到，这个奖励也是一个随机变量，在t时刻得到的奖励，用R_t来表示。

马尔科夫决策过程（Markov Decision Process）：在马尔科夫奖励过程的基础上，再加上策略，就构成了马尔科夫决策过程。举一个例子来说，对每周周末的学习日志进行观测之前，我是有自主权对这一周的时间进行规划的，我可以选择休息一周，也可以选择工作5天，休息2天，也可以选择工作7天。我做出的动作，很明显是会影响到周末对学习日志的观测值。我们把在每周开始前，我如何规划，叫做**策略**（Policy），用符号π来表示。规划完成后，具体地做选择，称为**动作**（Action），用随机变量A_t来表示。我们用如下的符号进行表示：

$$S_t \xrightarrow[R_{t+1}]{A_t} S_{t+1} \tag{1}$$

我们对上式进行一个简单的说明，S_t是t时刻的状态，之后我在t时刻，我做了一个动作A_t，然后得到了一个新的状态S_{t+1}，对应的得到了一个奖励，有的文献上把这个奖励记为R_t，有的记为R_{t+1}，我们在这里采用第二种记号，认为采取动作A_t后，得到的奖励，是随机变量R_{t+1}。那么很明显，这个奖励依赖于(S_t, A_t)。我们在讨论一下策略，策略分为确定策略和随机策略，确定性策略，意味着只要遇到状态s，就百分之百采用一个动作。随机策略意味着遇到状态s，智能体以一定的概率选择动作$\pi(a|s) \equiv P\{A = a|S = s\}$。举一个例子，比如超级玛丽，当面对前方的怪物，确定策略就是指百分之百地后退，随机性策略指的是以0.3 的概率后退，以0.5的概率跳起，以0.2的概率向前。用数学语言来表示，为：$\pi(back|s) = 0.3, \pi(up|s) = 0.5, \pi(forward|s) = 0.2$。

对于一个MDP，所有的状态构成一个集合，称为状态集合$\mathcal{S}$，所有的动作构成一个集合，称为动作集合$\mathcal{A}$，所有的奖励构成了一个集合，称为奖励集合$\mathcal{R}$。

状态转移函数：我们可以看到，一个MDP充满着不确定性，那么我们要用数学的语言来表示这种不确定性，就必须引入概率。我们定义如下的函数：

$$P(s', r|s, a) \equiv Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\} \tag{2}$$

对这个函数进行一下解释，很明显这是一个条件概率。在已知状态S 和执行的动作a 后，这个系统下一时刻会发生状态s'和得到的奖励r对应的概率。我们可以看到，在s, a都确定的情况下，奖励依旧是不确定的，举一个例子来说明，比如我上周观测到学习日志，上面记录了一些待解决的问题，然后我采取了一个动作，工作5 天，休息两天。虽然工作了5 天，但是不一定能解决上周遗留的问题，所以奖励依旧是一个未知数，如果解决了就是大餐，没解决就是跑步。那么我们可以对这个函数进行一个求和，那就是把未来得到的所有可能奖励进行一个求和，得到一个和奖励无关的函数，我们把它叫做状态转移函数：

$$P(s'|s, a) \equiv \sum_{r \in \mathcal{R}} P(s', r|s, a) \tag{3}$$

回报：从当前时刻开始到本回合结束的所有奖励的总和，称为回报。所以回报也叫做**累计奖励**（cumulative future reward）。把t时刻的回报记作随机变量G_t。如果一回合游戏结束，已经观测到所有奖励，那么就把回报记作g_t。设一个回合在时刻T 结束。定义回报为：

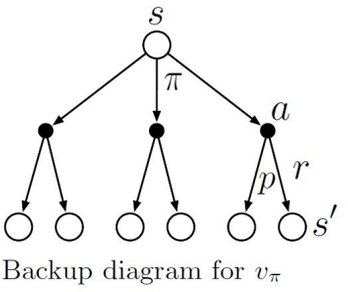
$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{4}$$

回报是未来获得的奖励总和，所以智能体的目标就是让回报越大越好。强化学习的目标就是寻找一个策略，使得回报的期望最大化。这个策略称为最优策略(optimum policy)。强化学习的目标是最大化回报，而不是最大化当前的奖励。比如，下棋的时候，你的目标是赢得一局比赛（回报），而非吃掉对方当前的一个棋子（奖励）。但是我们应该说一点，我们对未来的预期是有折扣的，换句话说，当下让你吃一顿大餐，和承诺你一年后请你吃一顿大餐，你对这两种情况，心理上是会有落差的，我们更想得到及时的奖励，所以未来的奖励是要打一个折扣，这里我们需要强调，这个奖励是包含惩罚的，及时的惩罚要比对未来许诺的惩罚对回报来说，贡献更大。所以我们引入**折扣因子**（discount factor）的概念，记为γ，它的取值范围为$[0,1]$。把回报与折扣因子综合在一起，得到如下的表达式：

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1},\ when(t \to \infty) \tag{5}$$

状态价值函数：对于智能体当前的状态，可以选择某一个动作，动作与环境交互，会得到新的状态，不同的动作，会让智能体的未来有不同的走向，每个走向都对应一条轨迹。那么我们假设穷尽了所有的轨迹，把每个轨迹上的回报，进行加权平均，把这个期望定义为状态价值函数。权重是走每条轨迹的概率大小，因为不同的策略，对应着走每条轨迹的可能性是不同的，再求期望时，应该对可能性大的轨迹分配更多的权重。这个函数只有当前的状态有关，记为$V(s) = E_\pi[G_t|S_t = s]$。期望符号右下角的$\pi$ 就代表着权重分配，不同的策略，就代表着不同的权重。

回溯图：回溯图是对马尔科夫决策过程的可视化。我们通过图1来进行说明：



Backup diagram for v_π

Figure 1: 回溯图

这个图片，展示了从s状态，到s'状态变化的过程，我们从图中可以看到，从当前的s状态，有三种可能的动作去执行，而智能体选择了最右边的策略a，那么未来的走向就跑到了最右边这条路上，当选择了动作a 后，环境会和智能体有一个交互，反馈给智能体一个新的奖励和新的状态，这个新的奖励和状态也是随机的，因为有可能相同的状态下，采用相同的动作，会得到不一样的结果。我们可以从图上看到，环境反馈给智能体的状态为s'，在这个图的最右边，但是这个图没有把奖励的随机性展示出来，只是进行了一个标注，在分支上用r 来代表奖励，但是我们要知道，这个r 是具有随机性的。

动作价值函数：我们通过回溯图，可以发现，在状态s下面，有多个分支，每个分支对应着不同的动作，这个分支其实是和智能体的策略是无关的，因为策略代表的是每个分支的权重。不会因为不同的策略，取消掉了一个分支，哪怕是确定策略，也只是百分之百往一条分支上进行，但是其它的分支还是存在的，只是智能体没有选择它们而已。在此基础上，我们可以定义动作价值函数，$q_\pi(s,a) \equiv E_\pi[G_t|S_t = s, A_t = a]$。这个函数代表的是，在当前状态$s$的分支处，选择动作$a$，对应$a$之后路径下的所有分支的回报期望。但是我们应该注意，在得到s' 状态后的所有分支，还是与策略π相关的，只不过当前的分支与π 无关。所以我们在q的右下角用π 来标注。

2 贝尔曼期望方程与优化方程

我们接下来讨论状态价值函数与动作价值函数的关系。通过定义我们就可以发现，$V_\pi(s)$是对未来所有的分支进行的加权平均，$q_\pi(s,a)$是当下s状态下其中一个分支的回报期望，自然地我们就应该有如下的关系：

$$V_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) q_\pi(s,a) \tag{6}$$

这个等式说明了，在状态s下，对所有可能的分支进行加权平均，权重是策略对应的每个动作的可能性。也得到了V_π与q_π的一个关系。在从动作a这个节点往下看，这个节点下面依旧有很多分支，这些分支我们之前强调过，不是由智能体来决定的，是由环境的反馈来决定的，所以就需要之前提到的状态转移函数来刻画，从动作a 走到状态s'，会得到一个及时的奖励r，然后还有状态s'对应的价值函数$V_\pi(s')$，由于存在对未来的折扣，所以我们需要在价值函数前乘以γ。然后我们对所有的分支加权平均，权重是之前提到的$P(s',r|s,a)$，我们得到如下的表达式：

$$q_\pi(s,a) = \sum_{r,s'} P(s',r|s,a)(r+\gamma V_\pi(s')) \tag{7}$$

这个表达式也可以从回报的定义出发得到，我们再此就不在推导了。我们可以把公式（7）代入到公式（6）中，或者把公式（6）代入到公式（7）中，我们就能得到如下的两个表达式：

$$V_\pi(s) = \sum_{s\mathcal{A}} \pi(a|s) \sum_{s',r} P(s',r|s,a)[r+\gamma V_\pi(s')] \tag{8}$$

$$q_\pi(s,a) = \sum_{s',r} P(s',r|s,a)[r+\gamma \sum_{a'\in\mathcal{A}} \pi(a'|s') q_\pi(s',a')] \tag{9}$$

把上面两个方程，称为贝尔曼期望方程（Bellman Expectation Equation）。如果状态是有限的，那么V_{pi}是一个列向量。接下来我们可以定义最优价值函数$V_\star$和最优动作价值函数$q_\star$，它们具有如下的表达式：

$$V_\star(s) \equiv \max_\pi V_\pi(s) \tag{10}$$

$$q_\star(s,a) \equiv \max_\pi q_\pi(s,a) \tag{11}$$

我们记$\pi_\star = \arg\max_\pi V_\pi(s) = \arg\max_\pi q_\pi(s,a)$，这个$\pi_\star$对应着最优策略。通过这个定义，我们很快就能得处一个直觉上的理解，就是在每个动作分支时，总选择那个对应最大的$q_\pi(s,a)$，应该就是最优策略。换句话说，对贝尔曼期望方程的策略期望，改成求最大值，就对应着贝尔曼优化方程，即$V_\star = \max_{a\mathcal{A}} q_{\pi_\star}(s,a)$，进而我们把$q(s,a)$与$V_\pi$ 的关系代入，得到：

$$V_\pi = \max_{a\in\mathcal{A}} \sum_{s',r} P(s',r|s,a)[r+V_\pi(s')] \tag{12}$$

同样我们也可以把$q_\star(s,a)$的期望表达式改写成最优化方程：

$$q_\pi(s,a) = \sum_{s',r} P(s',r|s,a)[r+\gamma \max_{a'\in\mathcal{A}} q_\pi(s',a')] \tag{13}$$

我们把公式（12）和（13）统称为贝尔曼优化方程（Bellman Optimality Equation）。在有的文献里，把我们写的求和号，会写成期望的形式，例如：$V_\pi(s) = E_\pi[R_{t+1}+\gamma V_\pi(S_{t+1})|S_t=s]$。

3 策略评估

在这一部分，我们讨论一个问题，已知MDP，在给定一个策略π时，我们对这个策略进行评估，即求出它的V_π。我们先可以讨论一种简单的情况，即状态是有限的，一共有$|\mathcal{S}|$ 个状态。那么V_π就对应着

一个$\mathbb{R}^{|\mathcal{S}|\times 1}$的列向量。我们从贝尔曼期望方程出发：

$$\begin{aligned} V_\pi(s) &= E_\pi[G_t|S_t=s] && (14)\\ &= E_\pi[R_{t+1}+\gamma V_\pi(S_{t+1})|S_t=s] && (15)\\ &= \sum_a \pi(a|s)\sum_{s',r} P(s',r|s,a)[r+\gamma V_\pi(s')] && (16)\\ &= \sum_a \pi(a|s)\sum_{s',r} P(s',r|s,a)r + \gamma\sum_a \pi(a|s)\sum_{s',r} P(s',r|s,a)V_\pi(s') && (17)\end{aligned}$$

我们把上面的两项分别计算，暂且称第一项为**1**，第二项为**2**。先看第一项：

$$\begin{aligned} \mathbf{1} &= \sum_a \pi(a|s)\sum_{s',r} rP(s',r|s,a) && (18)\\ &= \sum_a \pi(a|s)\sum_r r\mathbb{P}(r|s,a) && (19)\\ &= \sum_a \pi(a|s)r(s,a) \equiv r_\pi(s) && (20)\end{aligned}$$

在第二个等号时，我们把对状态s'求和给消掉了，得到了一个新的概率函数$\mathbb{P}$。第三个等号时，我们定义了一个新的函数$r(s,a)=\sum_r r\mathbb{P}(r|s,a)=E_\pi[R_{t+1}|S_t=s,A_t=a]$。最后一个等号，我们把对策略的加权平均，记为$r_\pi(s)$，我们可以看到，因为求和把策略$a$给消掉了，所以这个$r_\pi$ 只与状态有关，同样是一个$\mathbb{R}^{|\mathcal{S}|\times 1}$的列向量。

再来看第二项：

$$\begin{aligned} \mathbf{2} &= \gamma\sum_a \pi(a|s)\sum_{s',r} P(s',r|s,a)V_\pi(s') && (21)\\ &= \gamma\sum_a \pi(a|s)\sum_{s'} P(s'|s,a)V_\pi(s') && (22)\\ &= \gamma\sum_{s'}\sum_a \pi(a|s)P(s'|s,a)V_\pi(s') && (23)\\ &= \gamma\sum_{s'} P_\pi(s,s')V_\pi(s') && (24)\end{aligned}$$

这里我们定义了一个函数$\sum_a \pi(a|s)P(s'|s,a)=P_\pi(s,s')$，这个函数$P_\pi(s,s')$的所有取值共有$|\mathcal{S}|\times\mathcal{S}$ 的元素，我们可以用一个矩阵P_π 来表示，矩阵元为$P(s_i,s_j)$。所以第二项可以写成$\gamma P_\pi V_\pi$。最后把这两项合在一起，有：

$$V_\pi = r_\pi + \gamma P_\pi V_\pi \quad (25)$$

通过移项，可以得到V_π的表达式，即$V_\pi=(1-\gamma P_\pi)^{-1}r_\pi$。这个是对策略$\pi$的价值函数的解析解。可以看到，这个方程，随着矩阵的维度越大，运算会越困难，计算复杂度大约为$o(|\mathcal{S}|^3)$。退而求其次，我们不求解析解，可以通过数值迭代得到数值解。具体可以首先随机初始化一个列向量V_1，然后把这个向量代入到方程：

$$V_{k+1} = \max_{a\in\mathcal{A}}\sum_{s',r} P(s',r|s,a)[r+V_k(s')] \quad (26)$$

首先把V_1代入，k代表着迭代次数，会得到一个序列$\{V_k\}_{k=1}^{+\infty}$，直到迭代到数值不在发生变化，可以认为已经收敛。

4 策略改进

策略改进定理：如果$\forall s\in\mathcal{S}$，有$q_\pi(s,\pi'(s))\geq V_\pi(s)$，那么$\forall s\in\mathcal{S}$，都有：$V_{\pi'}(s)\geq V_\pi(s)$。

证明：当$V_\pi(s) \geq q_\pi(s, \pi'(s))$，我们进行如下的推导：

$$\begin{aligned} V_\pi(s) &\leq q_\pi(s, \pi'(s)) = E[R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s, A_t = \pi'(s)] && (27)\\ &\leq E_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1}))|S_t = s] && (28)\\ &= E_{\pi'}[R_{t+1} + \gamma E_{\pi'}[R_{t+2} + \gamma V_\pi(S_{t+2})|S_{t+1}]|S_t = s] && (29)\\ &= E_{\pi'}[R_{t+1} + \gamma E_{\pi'}[R_{t+2}|S_{t+1}] + \gamma^2 E_{\pi'}[V_\pi(S_{t+2})|S_{t+1}]|S_t = s] && (30)\\ &= E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 V_\pi(S_{t+2})|S_t = t] && (31)\\ &\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \cdots|S_t = s] && (32)\\ &= V_{\pi'}(s) && (33) \end{aligned}$$

我们对上面的证明过程给一个解释，对于R_{t+1}是由π'控制得到的，所以在第二行，我们把期望的右下角添加了π'，但是此时，后面的R还是由π来控制，所以在后面的步骤，不停地进行代换，最后把所有的R都是从π'来得到的，最后可以看到，是我们最后的结论，即π'对应的$V_{\pi'}$ 始终比π 对应的V_π要好。

显而易见，要想得到最优策略，那就在任意状态下，都采用最大的q值所对应的动作，就能得到最优策略。我们把这种方法成为贪婪策略，即$\pi'(s) = \arg\max_a q_\pi(s, a)$。我们之前从定义应该知道，$V_\pi(s)$是策略$\pi$ 对应的加权平均，所以肯定有$V_\pi(s) \leq \max_a q_\pi(s, a) = q_\pi(s, \pi'(s))$。所以策略改进定理给我们提供了一个方法，去寻找到更好的策略。接下来，我们要证明，当在迭代到$V_{\pi'} = V_\pi$时，那么我们说，此时的V_π就是最优策略。证明如下：
对于$\forall s \in \mathcal{S}$，有：

$$\begin{aligned} V_{\pi'}(s) &= \sum_a \pi'(a|s) q_{\pi'}(s, a) && (34)\\ &= \sum_a \pi'(a|s) q_\pi(s, a) && (35)\\ &= q_\pi(s, \pi'(s)) && (36)\\ &= \max_a q_\pi(s, a) && (37)\\ &= \max_a \sum_{s', r} P(s', r|s, a)[r + \gamma V_\pi(s')] && (38) \end{aligned}$$

第二个等号，我们利用了$V_{\pi'} = V_\pi$时，$q_{\pi'} = q_\pi$。第三个等号，利用了贪婪策略的定义，π' 总是要选择最大的q值。第四个等号，依旧使用了贪婪策略的定义，最后一行，我们发现，就是贝尔曼最优方程的右面，所以π'就对应着贝尔曼方程的左边$\pi_\star$。

5 迭代技巧

通过以上的介绍，我们对寻找最优策略已经有了一些认识，我们总结一下，策略迭代首先进行策略评估，评估好了之后，然后利用贪心策略手段，得到更好的策略，然后在评估当前策略，再次迭代，反复循环，迭代里套迭代，即具有如下的形式：

$$\pi_1 \to V_{\pi_1} \to \pi_2 \to V_{\pi_2} \cdots \tag{39}$$

我们可以看到，这种计算方法迭代是非常大的，一种好的办法是我们改进一下这种迭代方法，截断策略评估。即对策略评估，不在等到收敛，只迭代一次，然后直接改进策略。即：

$$V_{k+1}(s) = \max_a \sum_{s', r} P(s', r|s, a)[r + \gamma V_k(s')] \tag{40}$$

这里k代表着迭代次数。我们把这种迭代方法叫做**价值迭代**。即：

$$V_1 \to V_2 \to V_3 \cdots V_\star \tag{41}$$

换句话说，价值迭代是策略迭代下的极端情况。我们通过观察这个对一次更新，还是需要对所有的状态更新，即对每一个状态的V_π都进行更新。所以可以再次简化，我们每次就只更新一个状态对应的价值函数，把这种更新方法称为**异步更新**。这样我们从策略评估改进到价值迭代，又从遍历整个状态空间，到异步更新（异步更新又叫就地更新）。

6 深度强化学习

DQN首先由Mnih等人在2013年提出，这篇论文用Atari 游戏评价DQN 的表现，虽然DQN的表现优于已有方法，但是它还是比人类的表现差一截。相同的作者在2015 年发表了DQN 的改进版本，其主要改进在于使用“目标网络”(target network)；这个版本的DQN在Atari游戏上的表现超越了人类玩家。DQN的本质是对最优动作价值函数$Q_\star$的函数近似。早在1995年和1997 年发表的论文就把函数近似用于价值学习中。本章使用的TD算法叫做Q学习算法，它是由Watkins在1989 年在博士论文提出的。在2013 年，DeepMind 发表了利用强化学习玩Atari 游戏的论文，至此强化学习开始了新的十年。围棋作为人类的娱乐游戏中复杂度最高的一个，它横竖各有19 条线，共有361 个落子点，双方交替落子，状态空间高达10^{171}(注：宇宙中的原子总数是10^{80}，即使穷尽整个宇宙的物质也不能存下围棋的所有可能性)，但是2015 年10月，由Google DeepMind 公司开发的AlphaGo 程序击败了人类高级选手樊麾，成为第一个无需让子即可在19路棋盘上击败围棋职业棋手的计算机围棋程序，并写进了历史，论文发表在国际顶级期刊《Science》上。2016 年3月，透过自我对弈数以万计盘进行练习强化，AlphaGo 在一场五番棋比赛中4:1 击败顶尖职业棋手李世石。Master(AlphaGo版本)于2016 年12 月开始出现于弈城围棋网和腾讯野狐围棋网，取得60连胜的成绩，以其空前的实力轰动了围棋界。DeepMind如约公布了他们最新版AlphaGo论文(Nature)，其使用了蒙特卡洛树搜索与两个深度神经网络相结合的方法，其中一个是以估值网络来评估大量的选点，而以走棋网络来选择落子。在这种设计下，计算机可以结合树状图的长远推断，又可像人类的大脑一样自发学习进行直觉训练，以提高下棋实力，后使用了强化学习进行自我博弈，最终达到了战胜世界一流的人类围棋选手的成绩。其背后的原理成为每个人工智能爱好者都想了解的秘密。随后最强最新的版本AlphaGo Zero，使用纯强化学习，将价值网络和策略网络整合为一个架构，3天训练后就以100比0击败了上一版本的AlphaGo。AlphaGo 已经退休，但技术永存。DeepMind 已经完成围棋上的概念证明，接下来就是用强化学习创造改变世界的价值。

接下来我们在前面介绍概念的基础上，我们介绍一下对应的时序差分算法，策略学习和连续控制相关的知识。更详细的讨论，我会在最后给出一些参考文献供各位去阅读。

6.1 深度Q学习

我们首先需要介绍一下，什么是深度Q学习（Deep Q Learning）。所谓的深度Q学习，就是通过一个神经网络去逼近最优的动作价值函数$q_\star$，这个神经网络输入的是状态，输出的是对动作空间所有动作的打分，然后根据评分的高低，就可以去选择动作了，如图2。 通过一个神经网络，就可以把动作价值函数，写成$q(s,a;w)$，这里w代表着神经网络的参数。训练网络的方式，我们在神经网络的那一章已经详细地介绍过了，就是梯度反向传播算法，这里我们可以去求一下q 函数对参数w的梯度：

$$\nabla_w q(s,a;w) \equiv \frac{\partial q(s,a;w)}{\partial w} \tag{42}$$

表示函数值$q(s,a;w)$对参数w的梯度。在编程实现中，$Tensorflow$和$Pytorch$可以对DQN输出向量的一个元素关于变量w自动求梯度。

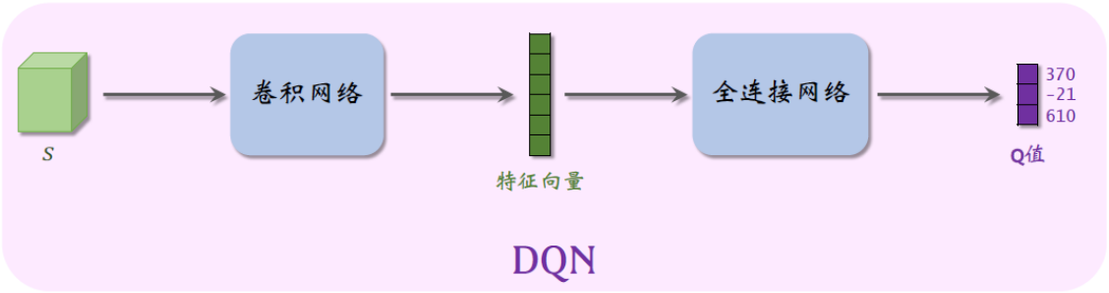


Figure 2: 通过一个卷积神经网络去提取图片的特征，当然在这里如果是序列信息，可以改成循环神经网络。得到特征后，输入到一个全连接层，这个全连接层最后输出一个列向量，动作空间有多少个取值，列向量的维度就有多少。列向量的每个元素代表着对每个动作的打分。

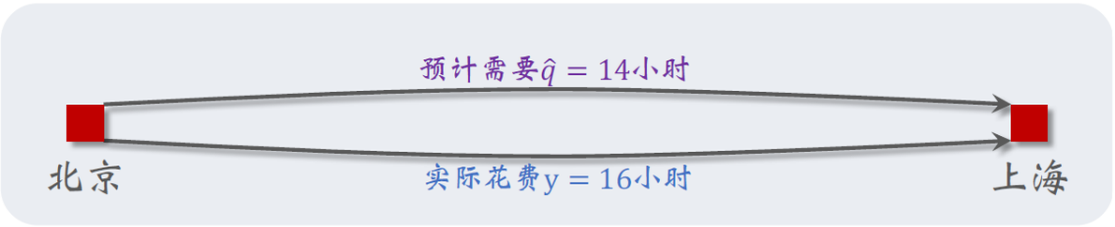


Figure 3: 估计的与实际的差距

6.2 时间差分

假设我们有一个模型$q(s,d;w)$，其中s是起点，d 是终点，w是参数。模型q可以预测开车出行的时间开销。这个模型一开始不准确，甚至是纯随机的。但是随着很多人用这个模型，得到更多数据、更多训练，这个模型就会越来越准，会像谷歌地图一样准。

我们该如何训练这个模型呢？在用户出发前，用户告诉模型起点s和终点d，模型做一个预测$\hat{q}=q(s,d;w)$。当用户结束行程的时候，把实际驾车时间y反馈给模型。两者之差$\hat{q}=y$反映出模型是高估还是低估了驾驶时间，以此来修正模型，使得模型的估计更准确。

假设有个用户，要从北京驾车去上海。从北京出发之前，让模型做预测，模型告诉我总车程是14小时：

$$\hat{q}\equiv q(\text{"}Beijing\text{"},\text{"}Shanghai\text{"};w)=14 \tag{43}$$

当我到达上海，我知道自己花的实际时间是16小时，并将结果反馈给模型；见图3。 可以用梯度下降对模型做一次更新，具体做法如下。把我的这次旅程作为一组训练数据：

$$s=\text{"}Beijing\text{"},d=\text{"}Shanghai\text{"},\hat{q}=14,y=16 \tag{44}$$

我们希望估计值$\hat{q}=Q(s,d;w)$尽量接近真实观测到的y，所以用两者差的平方作为损失函数：

$$L(w)=\frac{1}{2}[q(s,d;w)-y]^2 \tag{45}$$

用链式法则计算损失函数的梯度，得到：

$$\nabla_w L(w)=(\hat{q}-y)\nabla q(s,d;w) \tag{46}$$

然后通过一次梯度下降更新模型参数w：

$$w\leftarrow w-\alpha\nabla_w L(w) \tag{47}$$

Figure 4: $\hat{q}=14$和$\hat{q}'=11$是模型的估计值；$r=4.5$ 是实际观测值。

α是学习率。更新，如果让模型在做一次预测，会比原先的结果更接近$y=16$。

出发前模型估计全程时间为$\hat{q}=14$小时,模型建议的路线会途径济南。我从北京出发，过了$r=4.5$小时，我到达济南。此时我再让模型做一次预测，模型告诉我

$$\hat{q} \equiv q(\text{"}Jinan\text{"},\text{"}Shanghai\text{"};w)=11 \tag{48}$$

参看图4。假如此时我的车坏了，必须要在济南修理，我不得不取消此次行程。我没有完成旅途，那么我的这组数据是否能帮助训练模型呢？其实是可以的，用到的算法叫做**时间差分**（temporal difference），又叫TD算法。 模型估计从北京到上海一共需要$\hat{q}=14$ 小时，我实际用了$r=4.5$小时到达济南，模型估计还需要$\hat{q}'=11$小时从济南到上海。到达济南时，根据模型最新估计，整个旅程的总时间为：

$$\hat{y} \equiv r+\hat{q}'=4.5+11=15.5 \tag{49}$$

TD算法将$\hat{y}=15.5$称为**TD目标**（TD target）。它比最初的预测$\hat{q}=14$更可靠。最初的预测$\hat{q}=14$纯粹是估计的，没有任何事实的成分。TD目标$\hat{y}=15.5$也是个估计，但其中有事实的成分：其中的$r=4.5$就是实际的观测。基于以上讨论，我们认为TD目标$\hat{y}=15.5$比模型最初的估计值

$$\hat{q}=q(\text{"}Beijing\text{"},\text{"}Shanghai\text{"};w)=14 \tag{50}$$

更可靠，所以可以用$\hat{y}$对模型进行修正。我们期待估计值尽量接近TD目标$\hat{y}$，所以用两种差的平方作为损失函数：

$$L(w)=\frac{1}{2}[q(\text{"}Beijing\text{"},\text{"}Shanghai\text{"};w)-\hat{y}]^2 \tag{51}$$

此处把$\hat{y}$看做常数，尽管它依赖于w。计算损失函数的梯度：

$$\nabla_w L(w)=(\hat{q}-\hat{y})\nabla_w q(\text{"}Beijing\text{"},\text{"}Shanghai\text{"};w) \tag{52}$$

我们把$\hat{q}-\hat{y}$记作δ，称为**TD误差**（TD error）。做一次梯度下降更新模型参数w：

$$w \leftarrow w-\alpha\delta\nabla_w q(\text{"}Beijing\text{"},\text{"}Shanghai\text{"};w) \tag{53}$$

我们可以用TD算法训练DQN。注意，这里推导出的是最原始的TD算法，在实践中效果不佳。实际训练DQN的时候，应当使用一些高级技巧，在参考文献[1] 可以查看，我们这里只介绍基本算法。第一步，收集训练数据：我们可以用任何策略函数π去控制智能体与环境交互，这个π就叫做行为策略（behavior policy）。比较常用的是ϵ-greedy 策略：

$$a_t=\begin{cases}\arg\max_a q(s_t,a;w) & P=1-\epsilon \\ uniform(\mathcal{A}) & P=\epsilon\end{cases} \tag{54}$$

把智能体在一局游戏中的轨迹记作：

$$s_1,a_1,r_1,s_2,a_2,r_2,\cdots,s_n,a_n,r_n \tag{55}$$

把一条轨迹划分成n个(s_t, a_t, r_t, s_{t+1}) 这种四元组，存入数组，这个数组叫做**经验回放数组**(replay buffer)。第二步，随机从经验回放数组中取出一个四元组，记作(s_j, a_j, r_j, s_{j+1})。设DQN当前的参数为w_{now}，执行下面的步骤对参数做一次更新，得到新的参数w_{new}。

1. 对DQN做正向传播，得到q值：

$$\hat{q}_j = q(s_j, a_j; w_{now}) \tag{56}$$

$$\hat{q}_{j+1} = \max_{a \in \mathcal{A}} q(s_{j+1}, a; w_{now}) \tag{57}$$

2. 计算TD目标和TD误差：

$$\hat{y}_j = r_j + \gamma \hat{q}_{j+1} \tag{58}$$

$$\delta_j = \hat{q}_j - \hat{y}_j \tag{59}$$

3. 对DQN做反向传播，得到梯度：

$$h_i = \nabla_w q(s_j, a_j; w_{now}) \tag{60}$$

4. 做梯度下降更新DQN的参数：

$$w_{new} \leftarrow w_{now} - \alpha \delta h_j \tag{61}$$

智能体收集数据、更新DQN参数这两者可以同时进行。可以在智能体每执行一个动作之后，对w做几次更新。也可以在每完成一局游戏之后，对w做几次更新。

6.3 策略学习

本节的内容是**策略学习**(policy-based reinforcement learning) 以及**策略梯度**(policy gradient)。策略学习的意思是通过求解一个优化问题，学出最优策略函数或它的近似（比如策略网络）。本节介绍的REINFORCE 和actor-critic 只是帮助各位理解算法而已，实际效果并不好。在实践中应该用使用一些训练技巧，我们在此不做介绍了，感兴趣的同学，可以参看文献[2]。

本节考虑离散动作空间，比如$\mathcal{A} = \{left, right, up\}$。策略函数$\pi$是个条件概率质量函数：

$$\pi(a|s) \equiv P(A = a|S = s) \tag{62}$$

策略函数π的输入是状态s和动作a，输出是一个0到1之间的概率值。举个例子，把超级玛丽游戏当前屏幕上的画面作为s，策略函数会输出每个动作的概率值：

$$\pi(left|s) = 0.5 \tag{63}$$

$$\pi(right|s) = 0.2 \tag{64}$$

$$\pi(up|s) = 0.3 \tag{65}$$

如果我们有这样一个策略函数，我们就可以拿它控制智能体。每当观测到一个状态s，就用策略函数计算出每个动作的概率值，然后做随机抽样，得到一个动作a，让智能体执行a。

神经网络作为一个函数逼近器，当前最有效的方法是用神经网络$\pi(a_j|s;\theta)$ 近似策略函数$\pi(a_j|s)$。神经网络$\pi(a_j|s;\theta)$被称为策略网络。θ表示神经网络的参数；一开始随机初始化θ，随后利用收集的状态、动作、奖励去更新θ。

策略网络的结构如图5所示。策略网络的输入是状态s。在Atari 游戏、围棋等应用中，状态是张量（比如图片），那么应该如图所示用卷积网络处理输入。在机器人控制等应用中，状态s是向量，

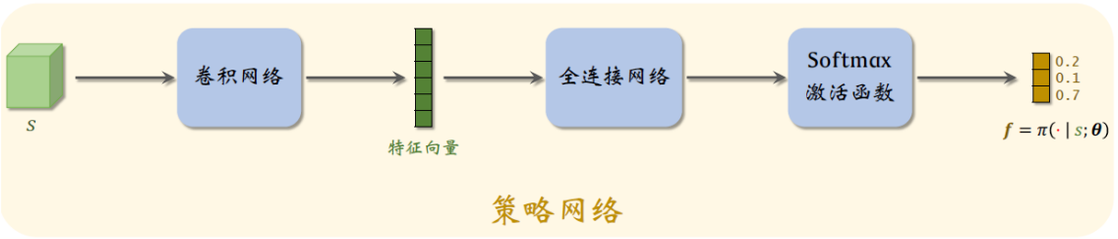


Figure 5: 策略网络$\pi(a|s;\theta)$的神经网络结构。输入的是状态s，输出是动作空间$\mathcal{A}$ 中每个动作的概率值。

它的元素是多个传感器的数值，那么应该把卷积网络换成全连接网络。策略网络输出层的激活函数是softmax，因此输出的向量（记作f）所有元素都是正数，而且相加等于1。动作空间$\mathcal{A}$的大小是多少，向量f的维度就是多少。在超级玛丽的例子中，$\mathcal{A} = \{left, right, up\}$，那么$f$就是3维的向量，比如$f = [0.2, 0.1, 0.7]$。$f$描述了动作空间$\mathcal{A}$ 上的离散概率分布，f每个元素对应一个动作：

$$f_1 = \pi(left|s) = 0.2, \tag{66}$$

$$f_2 = \pi(right|s) = 0.1, \tag{67}$$

$$f_3 = \pi(up|s) = 0.7 \tag{68}$$

当我们了解了策略网络的结构后，进而就是策略网络的目标函数了。其实早在刚开始介绍强化学习相关概念的时候，我们就有一个函数，恰好符合策略网络的更新要求，那就是价值状态函数$V_\pi(s)$，这个函数是依赖于当前的状态和策略。即当前状态s越好，V_π越大；策略π越好（即参数θ越好），那么$V_\pi(s)$也会越大。那么我们想有一个函数，只与π有关，那么很自然地就可以定义如下的函数：

$$J(\theta) = E_S[V_\pi(s)] \tag{69}$$

这个目标函数排除掉了所以状态$\mathcal{S}$的因素，只依赖于策略网络π的参数θ；策略越好，则$J(\theta)$越大。所以策略学习可以描述为这样一个优化问题：

$$\max_\theta J(\theta) \tag{70}$$

我们希望通过对策略网络参数θ的更新，使得目标函数$J(\theta)$越来越大，也就意味着策略网络越来越强。想要求解最大化问题，显然可以用梯度上升更新θ，使得$J(\theta)$增大。设当前策略网络的参数为θ_{now}。做梯度上升更新参数，得到新的参数θ_{new}：

$$\theta_{new} \leftarrow \theta_{now} + \beta \nabla_\theta J(\theta_{now}) \tag{71}$$

此处的β是学习率。上面的公式就是训练策略网络的基本想法，其中的梯度:

$$\nabla_\theta J(\theta_{now}) \equiv \left.\frac{\partial J(\theta)}{\partial \theta}\right|_{\theta=\theta_{now}} \tag{72}$$

被称作策略梯度。策略梯度可以写成下面定理中的期望形式。之后的算法推导都要基于这个定理，并对其中的期望做近似。

策略梯度定理：对函数$J(\theta)$的梯度为：

$$\frac{\partial J(\theta)}{\partial \theta} = E_S[E_{A\sim\pi(\cdot|S;\theta)}[\frac{\partial \ln \pi(A|S;\theta)}{\partial \theta} q_\pi(S, A)]] \tag{73}$$

需要注意，这里前面是差了一个常数$\frac{1-\gamma^n}{1-\gamma}$，当在做梯度上升时，这个常数会被学习率$\beta$ 吸收。这个公式复杂的地方在于，需要计算两个期望，实际中，我们第一次近似，只计算期望最里面的，当作近似的梯度，定义随机梯度：

$$h(s, a; \theta) \equiv q_\pi(s, a) \nabla_\theta \ln \pi(a|s;\theta) \tag{74}$$

根据以上的近似，我们可以做随机梯度上升来更新θ，使得目标函数$J(\theta)$逐渐增长：

$$\theta \leftarrow \theta + \beta h(s, a; \theta) \tag{75}$$

但是这种方法仍然不可行，我们计算不出$h(s, a; \theta)$，原因在于我们不知道动作价值函数$q_\pi(s, a)$。因此我们用两种方法对$q_\pi(s, a)$做近似：一种方法是REINFORCE，用实际观测的回报g 近似$q_\pi(s, a)$；另一种方法是actor-critic，用神经网络$q(s, a; w)$近似$q_\pi(s, a)$。

我们先介绍REINFORCE方法。设一局游戏有n步，一局中的奖励记作$R_1, \ldots, R_n$。t时刻的折扣回报定义为：

$$G_t = \sum_{k=t}^{n} \gamma^{k-t} R_{k+1} \tag{76}$$

而动作价值定义为G_t的条件期望：

$$q_\pi(s_t, a_t) = E[G_t | S_t = s_t, A_t = a_t] \tag{77}$$

我们可以用蒙特卡洛近似上面的条件期望。从时刻t开始，智能体完成一局游戏，观测到全部奖励$r_t, \ldots, r_n$，然后可以计算出$g_t = sum_{k=t}^{n} \gamma^{k-t} r_k$。因为$g_t$是随机变量$G_t$的观测值，所以$g_t$是上面公式中期望的蒙特卡洛近似。在实践中，可以用$g_t$ 代替$q_\pi(s_t, a_t)$，那么随机梯度$h(s_t, a_t; \theta)$ 可以近似成：

$$\tilde{h}(s_t, a_t; \theta) = g_t \nabla_\theta \ln \pi(a_t | s_t; \theta) \tag{78}$$

$\tilde{h}$是h的无偏估计，所以也是策略梯度$\nabla_\theta J(\theta)$的无偏估计；$\tilde{h}$ 也是一种随机梯度。利用反向传播计算出$\ln \pi$关于θ的梯度，而且可以实际观测到g_t，于是我们可以实际计算出随机梯度$\tilde{h}$的值。有了随机梯度的值，我们可以做随机梯度上升更新策略网络参数θ：

$$\theta \leftarrow \theta + \beta \tilde{h}(s_t, a_t | \theta) \tag{79}$$

根据上述推导，我们得到了训练策略网络的算法，它叫做REINFORCE。总结来说，具有如下的流程：

1. 用策略网络θ_{now}控制智能体从头开始玩一局游戏，得到一条轨迹（trajectory）：

$$s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_n, a_n, r_n \tag{80}$$

2. 计算所有的回报：

$$g_t = \sum_{k=t}^{n} \gamma^{k-t} r_k, \ \forall t = 1, \ldots, n \tag{81}$$

3. 用$\{(s_t, a_t)\}_{t=1}^{n}$作为数据，做反向传播计算梯度：

$$\nabla_\theta \ln \pi(a_t | s_t; \theta_{now}), \ \forall t = 1, \ldots, n \tag{82}$$

4. 做随机梯度上升更新策略网络参数：

$$\theta_{new} \leftarrow \theta_{now} + \beta \sum_{t=1}^{n} \gamma^{t-1} g_t \nabla_\theta \ln \pi(a_t | s_t; \theta_{now}) \tag{83}$$

在算法最后一步中，随机梯度前面乘以系数γ^{t-1}。是因为前面REINFORCE的推导是简化的，而非严谨的数学推导。在此我们不在介绍具体的推导步骤了。

另一种方法为Actor-Critic。Actor-critic方法用一个神经网络近似动作价值函数$q_\pi(s, a)$，这个神经网络叫做“价值网络”，记为$q(s, a; w)$，其中的w表示神经网络中可训练的参数。价值网络的输入是状

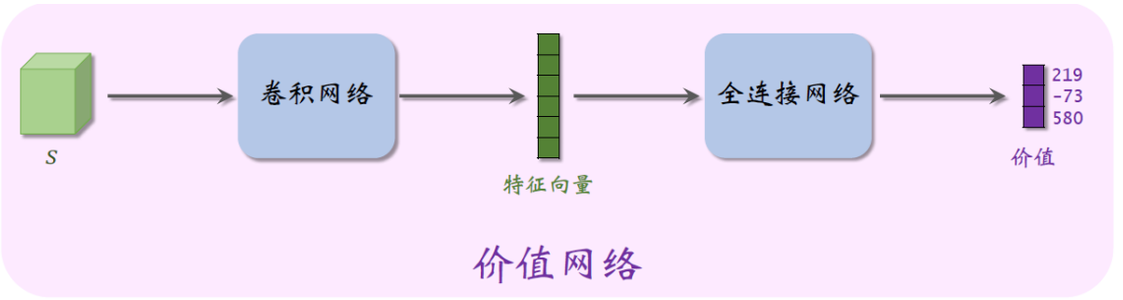


Figure 6: 价值网络$q(s,a;w)$的结构，输入是状态s，输出是每个动作的价值。

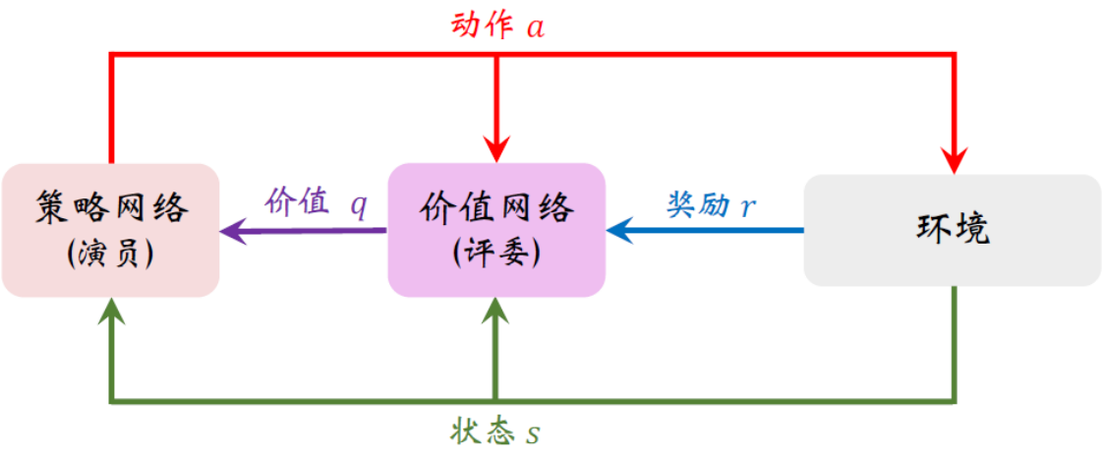


Figure 7: Actor-Critic两个神经网络之间的关系图。

态s，输出是每个动作的价值。动作空间$\mathcal{A}$中有多少种动作，那么价值网络的输出就是多少维的向量，向量每个元素对应一个动作。举个例子，动作空间是$\mathcal{A}=\{left,right,up\}$，价值网络的输出是：

$$q(s,left;w) = 219 \tag{84}$$

$$q(s,right;w) = -73 \tag{85}$$

$$q(s,up;w) = 580 \tag{86}$$

虽然价值网络$q(s,a;w)$与之前学的DQN有相同的结构，但是两者的意义不同，训练算法也不同。具体来说，有几以下两点：

- 价值网络是对动作价值函数$q_\pi(s,a)$的近似。而DQN 则是对最优动作价值函数$q_\star(s,a)$的近似。
- 对价值网络的训练使用的是SARSA算法（我们没有介绍SARAR算法，是TD算法的一个具体表现），它属于同策略（On-policy），不能用经验回放。对DQN 的训练使用的是q学习算法，它属于异策略（off-policy），可以用经验回放。

策略网络（Actor）和价值网络（评Critic）的关系如图7所示。 策略网络（Actor）想要改进自己的动作，但是Actor自己不知道什么样的动作才算更好，所以需要价值网络（Critic）的帮助。在Actor做出动作a之后，Critic会打一个分数$\hat{q}(s,a;w)$，并把分数反馈给Actor，帮助Actor做出改进。Actor利用当前状态s，自己的动作a，以及Critic的打分$\hat{q}$，计算近似策略梯度，然后更新自己的参数θ。最后我们总结以下训练流程：

1. 观测到当前状态s_t，根据策略网络做决策：$a_t \sim \pi(\cdot|s_t; _{now})$，并让智能体执行动作$a_t$。

2. 从环境中观测到奖励r_t和新的状态s_{t+1}。

3. 根据策略网络做决策：$\tilde{a}_{t+1} \sim \pi(\cdot|s_{t+1}; \theta_{now})$，但不让智能体执行动作$\tilde{a}_{t+1}$。

4. 让价值网络打分：

$$\hat{q}_t = q(s_t, a_t; w_{now}) \tag{87}$$

$$\hat{q}_{t+1} = q(s_{t+1}, \tilde{a}_{t+1}; w_{now}) \tag{88}$$

5. 计算TD目标和TD误差：

$$\hat{y} = r_t + \gamma \hat{q}_{t+1} \tag{89}$$

$$\delta_t = \hat{q}_t - \hat{y}_t \tag{90}$$

6. 更新价值网络：

$$w_{new} \leftarrow w_{now} - \alpha \delta_t \nabla_w q(s_t, a_t; w_{now}). \tag{91}$$

7. 更新策略网络：

$$\theta_{new} \leftarrow \theta_{now} + \beta \hat{q}_t \nabla_\theta \ln \pi(a_t|s_t; \theta_{now}). \tag{92}$$

最后我们要说：REINFORCE由Williams在1987年提出。Actor-critic由Barto等人在1983年提出。策略梯度定理由Marbach 和Tsitsiklis 1999 年的论文和Sutton等人2000年的论文独立提出。

6.4 连续控制

最后讨论连续控制，即动作空间是个连续集合，比如汽车的转向$\mathcal{A} = [-40°, 40°]$ 就是连续集合。如果把连续动作空间做离散化，那么离散控制的方法就能直接解决连续控制问题；我们讨论连续集合的离散化。然而更好的办法是直接用连续控制方法，而非离散化之后借用离散控制方法。本节介绍一种连续控制方法，确定策略网络。

考虑这样一个问题：我们需要控制一只机械手臂，完成某些任务，获取奖励。机械手臂有两个关节，分别可以在$[0°, 360°]$与$[0°, 180°]$的范围内转动。这个问题的自由度是$d = 2$，动作是二维向量，动作空间是连续集合$\mathcal{A} = [0°, 360°] \times [0°, 180°]$。

此前我们学过的强化学习方法全部都是针对离散动作空间，不能直接解决上述连续控制问题。想把此前学过的离散控制方法应用到连续控制上，必须要对连续动作空间做离散化（网格化）。比如把连续集合$\mathcal{A} = [0°, 360°] \times [0°, 180°]$变成离散集合$\mathcal{A} = \{0, 20, 40, ..., 360\} \times \{0, 20, 40, ..., 180\}$；见图8。 对动作空间做离散化之后，就可以应用之前学过的方法训练DQN或者策略网络，用于控制机械手臂。可是用离散化解决连续控制问题有个缺点。把自由度记作d。自由度d越大，网格上的点就越多，而且数量随着d指数增长，会造成维度灾难。动作空间的大小即网格上点的数量。如果动作空间太大，DQN和策略网络的训练都变得很困难，强化学习的结果会不好。上述离散化方法只适用于自由度d很小的情况；如果d不是很小，就应该使用连续控制方法。

确定策略梯度(Deterministic Policy Gradient, DPG))是最常用的连续控制方法。DPG是一种Actor-Critic 方法，它有一个策略网络（Actor），一个价值网络（Critic）。策略网络控制智能体做运动，它基于状态s做出动作a。价值网络不控制智能体，只是基于状态s给动作a打分，从而指导策略网络做出改进。图9是两个神经网络的关系。 本节的策略网络不同于前面章节的策略网络。在之前章节里，策略网络$\pi(a_j|s;)$是一个概率质量函数，它输出的是概率值。本节的确定策略网络$\mu(s; \theta)$ 的输出是d维的向量a，作为动作。两种策略网络一个是随机的，一个是确定性的：

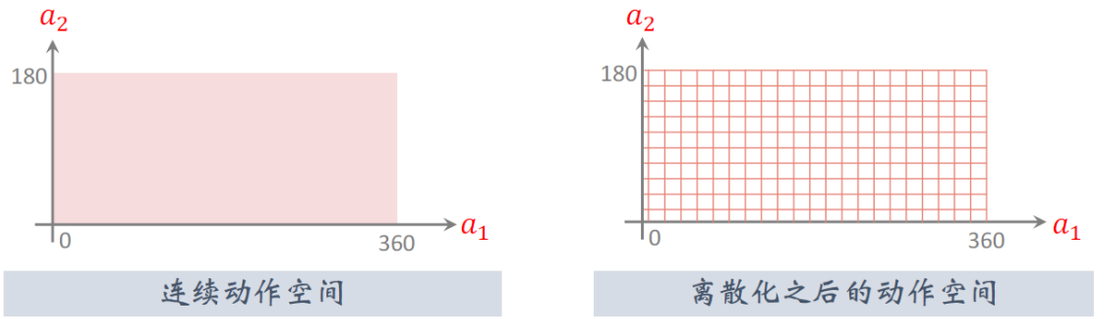


Figure 8: 对连续动作空间$\mathcal{A} = [0, 360] \times [0, 180]$做离散化。

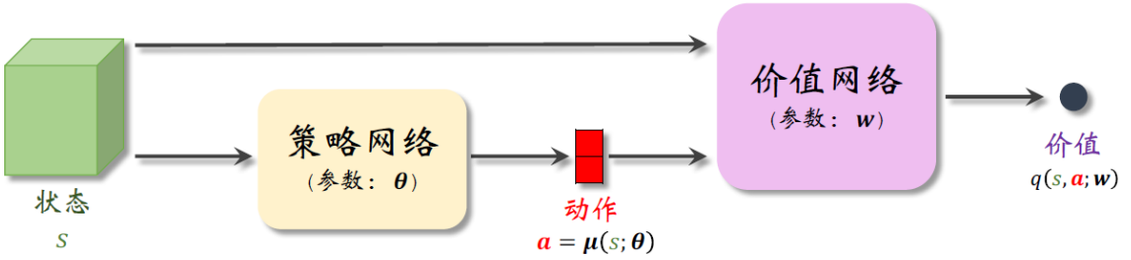


Figure 9: 确定策略梯度(DPG)方法的示意图。策略网络$\mu(s;\theta)$的输入是状态s，输出是动作a（d 维向量）。价值网络$q(s, a; w)$的输入是状态s和动作a，输出是价值（实数）。

- 之前章节中的策略网络$\pi(a_j|s;\theta)$ 带有随机性：给定状态s，策略网络输出的是离散动作空间$\mathcal{A}$上的概率分布；$\mathcal{A}$中的每个元素（动作）都有一个概率值。智能体依据概率分布，随机从$\mathcal{A}$中抽取一个动作，并执行动作。
- 本节的确定策略网络没有随机性：对于确定的状态s，策略网络μ输出的动作a是确定的。动作a直接是μ的输出，而非随机抽样得到的。

确定策略网络μ的结构如图10所示。如果输入的状态s 是个矩阵或者张量（例如图片、视频），那么μ 就由若干卷积层、全连接层等组成。 确定策略可以看做是随机策略的一个特例。确定策略$\mu(s;\theta)$的输出是d维向量，它的第i个元素记作$\hat{\mu} = [\mu(s;\theta)]_i$。定义下面这个随机策略：

$$\pi(a|s;\theta,\sigma) = \prod_{i=1}^{d} \frac{1}{\sqrt{(2\pi)}\sigma_i} \exp(-\frac{(a_i - \hat{\mu}_i)^2}{2\sigma_i^2}) \tag{93}$$

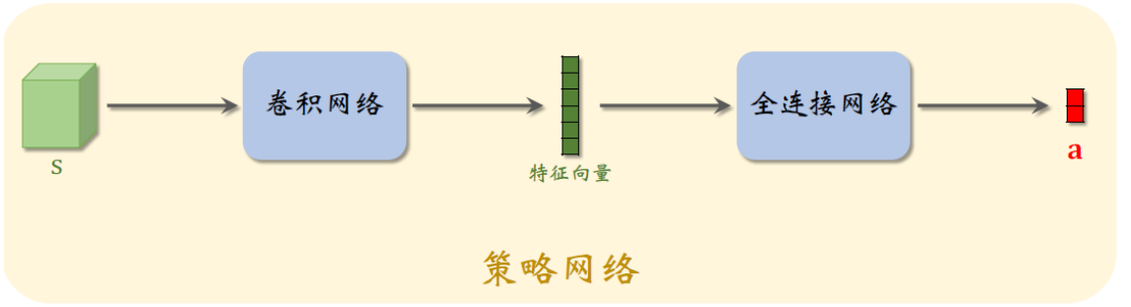


Figure 10: 确定策略网络$\mu(s;\theta)$的结构。输入是状态s，输出是动作a。

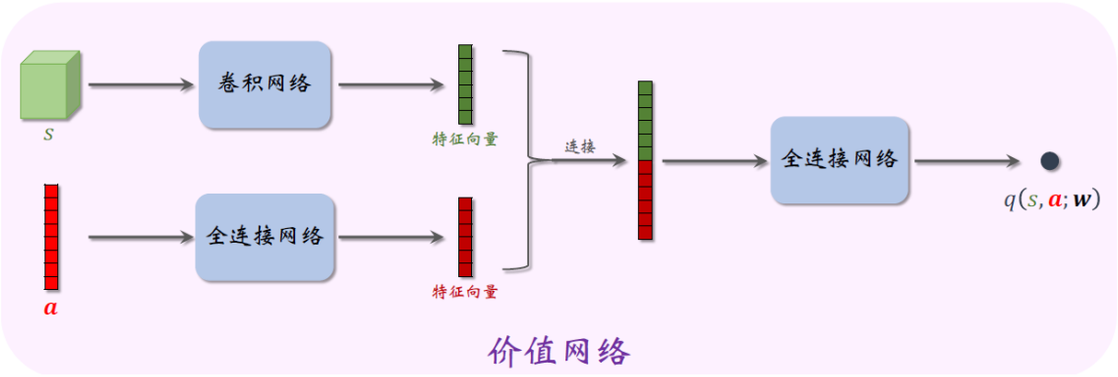


Figure 11: 价值网络$q(s,a;w)$的结构。输入是状态s和动作a，输出是实数。

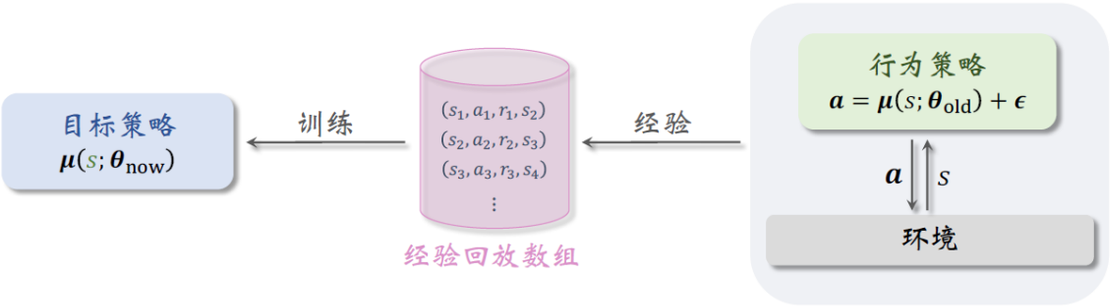


Figure 12: DPG属于异策略，收集经验与更新策略分开做。

这个随即策略是均值为$\mu(s;\theta)$、协方差矩阵为$diag(\sigma_1,\sigma_2,...,\sigma_d)$的多元高斯分布。我们介绍的确定性策略可以看作是上述随机策略在$\sigma=[\sigma_1,...,\sigma_d]$为全零向量时的特例。

本节的价值网络$q(s,a;w)$是对动作价值函数$q_\pi(s,a)$ 的近似。价值网络的结构如图11所示。价值网络的输入是状态s和动作a，输出的价值$\hat{q}=q(s,a;w)$ 是个实数，可以反映动作的好坏；动作a 越好，则价值$\hat{q}$ 就越大。所以价值网络可以评价策略网络的表现。在训练的过程中，价值网络帮助训练策略网络；在训练结束之后，价值网络就被丢弃，由策略网络控制智能体。 我们讨论完了结构，接着看一下，如何去训练网络。第一步依旧是是收集经验数据。用行为策略收集经验：本节的确定策略网络属于异策略(Off-policy) 方法，即行为策略(Behavior Policy) 可以不同于目标策略(Target Policy)。目标策略即确定策略网络$\mu(s;\theta_{now})$，其中θ_{now}是策略网络最新的参数。行为策略可以是任意的，比如

$$a=\mu(s;\theta_{old})+\epsilon. \tag{94}$$

公式的意思是行为策略可以用过时的策略网络参数，而且可以往动作中加入噪声$\epsilon\in\mathbb{R}^d$。异策略的好处在于可以把收集经验与训练神经网络分割开；把收集到的经验存入经验回放数组(Replay Buffer)，在做训练的时候重复利用收集到的经验。见图12。 用行为策略控制智能体与环境交互，把智能体的轨迹(Trajectory) 整理成(s_t,a_t,r_t,s_{t+1})这样的四元组，存入经验回放数组。在训练的时候，随机从数组中抽取一个四元组，记作(s_j,a_j,r_j,s_{j+1})。在训练策略网络$\mu(s;\theta)$的时候，只用到状态s_j。在训练价值网络$q(s,a;w)$的时候，要用到四元组中全部四个元素：s_j,a_j,r_j,s_{j+1}。第二步训练策略网络。首先通俗解释训练策略网络的原理。如图13所示，给定状态s，策略网络输出一个动作$a=\mu(s;\theta)$，然后价值网络会给a打一个分数：$\hat{q}=q(s,a;w)$。参数θ影响a，从而影响$\hat{q}$。分数$\hat{q}$ 可以反映出θ的好坏程度。训练策略网络的目标就是改进参数θ，使$\hat{q}$变得更大。把策略网络看做Actor，价值网络看做Critic。训练Actor（策略网络）的目的就是让他迎合迎合Critic（价值网络）的喜好，改变自己的动作技巧（即参数θ），使得评委打分$\hat{q}$的均值更高。 根据以上解释，我们来推导目标函数。如果当前状态是s，那么价值网络的

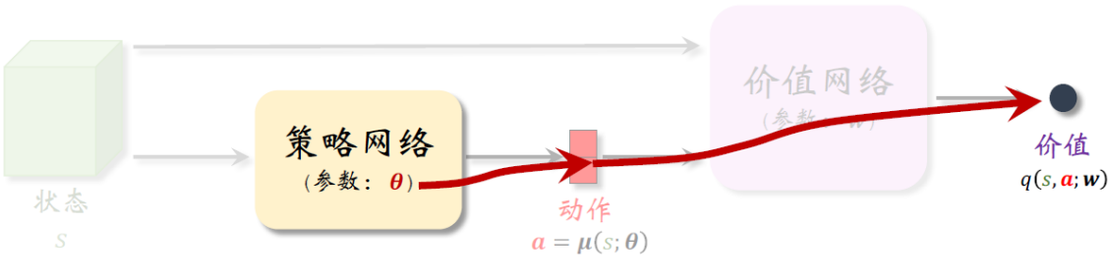


Figure 13: 给定状态s，策略网络的参数θ会影响a，从而影响$\hat{q} = q(s, a; w)$。

打分就是：

$$q(s, \mu(s;\theta); w) \tag{95}$$

我们希望打分的期望尽量高，所以把目标函数定义为打分的期望：

$$J(\theta) = E_S[q(S, \mu(S;\theta); w)]. \tag{96}$$

关于状态S求期望消除掉了S的影响；不管面对什么样的状态S，策略网络都应该做出很好的动作，使得平均分$J(\theta)$尽量高。策略网络的学习可以建模成这样一个最大化问题：

$$\max_{\theta} J(\theta) \tag{97}$$

注意，这里我们只训练策略网络，所以最大化问题中的优化变量是策略网络的参数θ，而价值网络的参数w被固定住。可以用梯度上升来增大$J(\theta)$。每次用随机变量S的一个观测值（记作s_j）来计算梯度：

$$h_i \equiv \nabla_\theta q(s_j, \mu(s_j;\theta); w). \tag{98}$$

它是$\nabla_\theta J(\theta)$的无偏估计。h_i叫做**确定策略梯度**(Deterministic Policy Gradient)，缩写DPG。用链式法则求出梯度：

$$\nabla_\theta q(s_j, \mu(s_j;\theta); w) = \nabla_\theta \mu(s_j;\theta) \nabla_a q(s_j, \hat{a}_j; w),\ \hat{a}_j = \mu(s_j;\theta) \tag{99}$$

由此我们得到更新θ的算法。每次从经验回放数组里随机抽取一个状态，记作s_j。计算$\hat{a}_j = \mu(s_j;\theta)$。用梯度上升更新一次$\theta$：

$$\theta \leftarrow \theta + \beta \nabla_\theta \mu(s_j;\theta) \nabla_a q(s_j, \hat{a}_j; w) \tag{100}$$

此处的β是学习率。这样做梯度上升，可以逐渐让目标函数$J(\theta)$增大，也就是让Critic给演员的平均打分更高。

最后训练价值网络：首先通俗解释训练价值网络的原理。训练价值网络的目标是让价值网络$q(s, a; w)$的预测越来越接近真实价值函数$q_\pi(s, a)$。如果把价值网络看做Critic，那么训练Critic的目标就是让他的打分越来越准确。每一轮训练都要用到一个实际观测的奖励r，r是真实值，用它来校准评Critic的打分。训练价值网络要用TD 算法。这里的TD算法与之前讲过的标准Actor-Critic类似，都是让价值网络去拟合TD目标。每次从经验回放数组中取出一个四元组(s_j, a_j, r_j, s_{j+1})，用它更新一次参数w。首先让价值网络做预测：

$$\hat{q}_j = q(s_j, a_j; w) \tag{101}$$

$$\hat{q}_{j+1} = q(s_{j+1}, \mu(s_{j+1};\theta); w) \tag{102}$$

计算TD目标$\hat{y}_j = r_j + \gamma \hat{q}_{j+1}$。定义损失函数：

$$L(w) = \frac{1}{2}[q(s_j, a_j; w) - \hat{y}_j]^2 \tag{103}$$

然后计算梯度：

$$\nabla_w L(w) = (\hat{q}_j - \hat{y}_j)\nabla_w q(s_j, a_j; w) \tag{104}$$

做一轮梯度下降更新参数w：

$$w \leftarrow w - \alpha \nabla_w L(w) \tag{105}$$

这样可以让损失函数$L(w)$减小，也就是让价值网络的预测$\hat{q} = q(s, a; w)$更接近TD目标$\hat{y}_j$。公式中的α是学习率。最后，我们把上述的步骤总结一下，得到如下的训练流程：

1. 让策略网络做预测：

$$\hat{a}_j = \mu(s_j; \theta_{now}) \tag{106}$$

$$\hat{a}_{j+1} = \mu(s_{j+1}; \theta) \tag{107}$$

   这里需要注意，计算动作$\hat{a}_j$用的是当前的策略网络$\mu(s_j; \theta_{now})$，用$\hat{a}_j$ 来更新θ_{now}；而从经验回放数组中抽取的a_j则是用过时的策略网络$\theta(s_j; \theta_{old})$算出的，用$a_j$来更新$w_{now}$。

2. 让价值网络做预测：

$$\hat{q}_j = q(s_h, a_j; w_{now}) \tag{108}$$

$$\hat{q}_{j+1} = q(s_{j+1}, \hat{a}_{j+1}; w_{now}) \tag{109}$$

3. 计算TD目标与TD误差：

$$\hat{y}_j = r_j + \gamma \hat{q}_{j+1} \tag{110}$$

$$\delta = \hat{q}_j - \hat{y}_j \tag{111}$$

4. 更新价值网络：

$$w_{new} \leftarrow w - \alpha \delta_j \nabla_w q(s_j, a_j; w_{now}) \tag{112}$$

5. 更新策略网络：

$$\theta_{new} \leftarrow \theta_{now} + \beta \nabla_\theta \mu(s_j; \theta_{now}) \nabla_a q(s_j, \hat{a}_j; w_{now}) \tag{113}$$

在实践中，上述算法的表现并不好，需要使用双延时确定策略梯度方法（TD3）。确定策略梯度(Deterministic Policy Gradient, DPG)方法由David Silver 等人在2014年提出。随后同一批作者把相似的想法与深度学习结合起来，提出深度确定策略梯度(Deep Deterministic Policy Gradient，缩写DDPG)，文章在2016 年发表。这两篇论文使得DPG方法流行起来。但值得注意的是，相似的想法在一些论文在之前也有提出，在此就不做介绍了。2018年的一篇论文提出三种对DPG的改进方法，并将改进的算法命名为TD3。2017年的论文提出了Soft Actor-Critic(SAC)，也可以解决连续控制问题。Degris等人在2012年发表的论文使用正态分布的概率密度函数作为策略函数，并且用线性函数近似均值和方差对数。类似的连续控制方法最早由Williams 在1987和1992年提出。

6.5 强化学习的应用与展望

到目前为止，深度强化学习最成功、最有名的应用仍然是Atari 游戏、围棋游戏、星际争霸游戏。深度强化学习有很多现实中的应用，但其中成功的应用并不多。本节探讨究竟是什么在制约深度强化学习的落地应用。

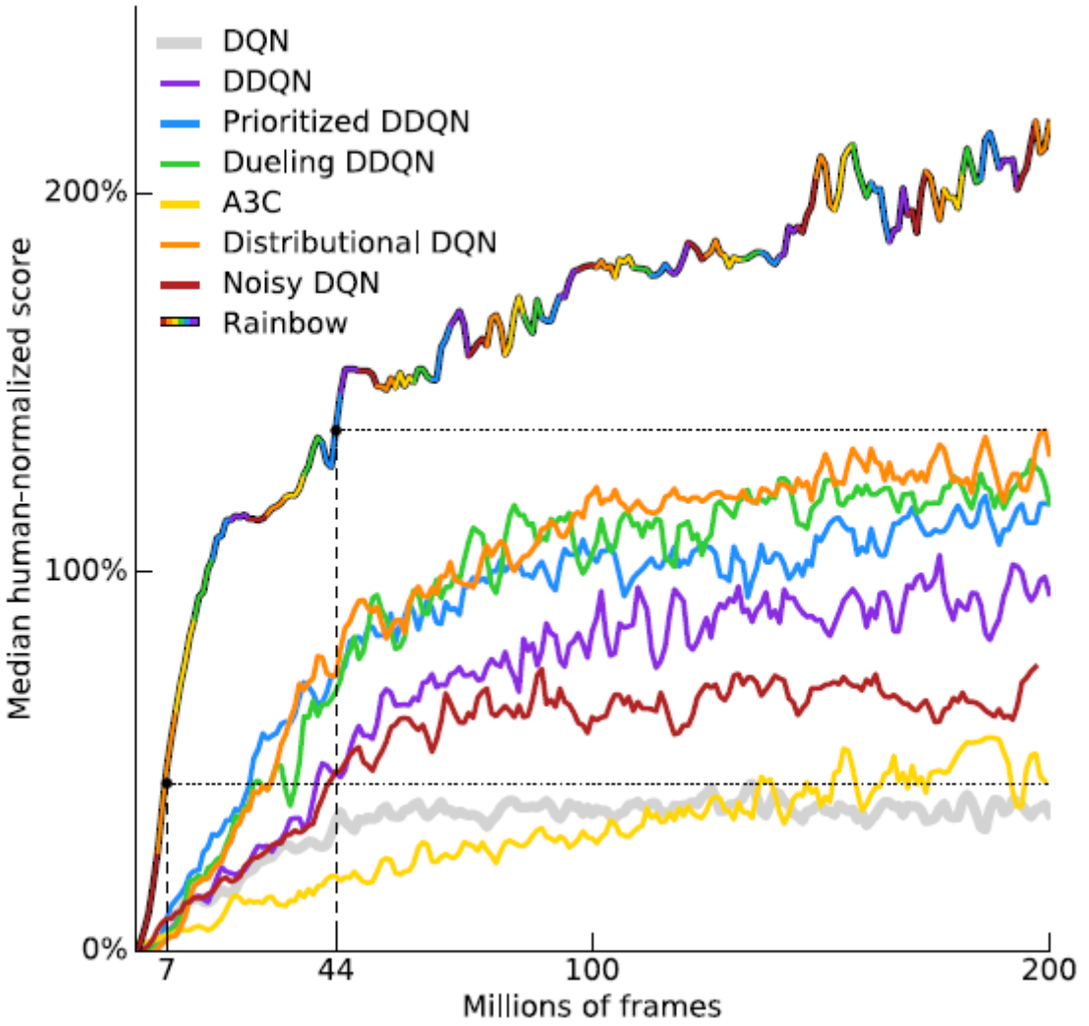


Figure 14: 使用多种技巧去训练DQN玩Atari游戏。

6.5.1 所需的样本数量过大

深度强化学习一个严重的问题在于需要巨大的样本量。举个例子，Atari游戏属于最简单的电子游戏，在现实世界中找不到这么简单的问题。2015年的论文用DQN玩Atari 游戏，取得了超越人类玩家的分数，在学术界内引起了轰动。2015 年提出的原始的DQN存在诸多问题，实验效果不够好。2018年的论文提出RainbowDQN，将多种技巧结合，让DQN的训练变得更快更好。论文在57种Atari游戏上比较了原始DQN、多种高级技巧、以及RainbowDQN。图14中纵轴是算法的分数与人类分数的比值，并关于57种游戏求中位数；100%表示达到人类玩家的水准。图中横轴是收集到的游戏帧数，即样本数量。RainbowDQN需要1千8百万帧才能达到人类玩家水平，超过1亿帧还未收敛；前提是已经调优了超过10种超参数。

在电子游戏中获取上亿样本并不困难，但是在现实问题中每获取一个样本都是比较困难的。在神经网络结构搜索的例子中，每获取一个奖励，需要训练一个CNN。从初始化到梯度算法收敛，需要一个GPU约一小时的计算量。物理世界的应用中获取奖励更为困难。举个例子，用机械手臂抓取一个物体至少需要几秒钟时间，那么一天只能收集一万个样本；同时用十个机械手臂，连续运转一百天，才能收集到一千万个样本，未必够训练一个深度强化学习模型。强化学习所需的样本量太大，这会限制强化学习在现实中的应用。

6.5.2 探索阶段代价太大

强化学习要求智能体与环境交互，用收集到的经验去更新策略。在交互的过程中，智能体会改变环境。在仿真、游戏的环境中，智能体对环境造成任何影响都无所谓。但是在现实世界中，智能体对环境的影响可能会造成巨大的代价。

在强化学习初始的探索阶段，策略几乎是随机的。如果是物理世界中的应用，智能体的动作难免造成很大的代价。如果应用到推荐系统中，如果上线一个随机的推荐策略，那么用户的体验会极差，很低的点击率也会给网站造成收入的损失。如果应用到自动驾驶中，随机的控制策略会导致车辆撞毁。如果应用到医疗中，随机的治疗方案会致死致残。在物理世界的应用中，不能直接让初始的随机策略与环境交互，而应该先对策略做预训练，再在真实环境中部署。一种方法是事先准备一个数据集，用行为克隆等监督学习方法做预训练。另一种方法是搭建模拟器，在模拟器中预训练策略。比如阿里巴巴提出的“虚拟淘宝”系统是对真实用户的模仿，用这样的模拟器预训练推荐策略。离线强化学习(Offline RL) 是一个热门而又有价值的研究方向。

6.5.3 超参数的影响非常大

深度强化学习对超参数的设置极其敏感，需要很小心调参才能找到好的超参数。超参数分两种：神经网络结构超参数、算法超参数。这两类超参数的设置都严重影响实验效果。换句话说，完全相同的方法，由不同的人实现，效果会有天壤之别。

6.5.4 稳定性极差

强化学习训练的过程中充满了随机性。除了环境的随机性之外，随机性还来自于神经网络随机初始化、决策的随机性、经验回放的随机性。想必大家都有这样的经历：用完全相同的程序、完全相同的超参数，仅仅更改随机种子（random seed），就会导致训练的效果有天壤之别。如示意图15所示，如果重复训练十次，往往会有几次完全不收敛。哪怕是非常简单的问题，也会出现这种不收敛的情形。在监督学习中，由于随机初始化和随机梯度中的随机性，即使用同样的超参数，训练出的模型表现也会不一致，测试准确率可能会差几个百分点。但是监督学习中几乎不会出现图15中这种情形；如果出现了，几乎可以肯定代码中有错。但是强化学习确实会出现完全不收敛的情形，哪怕代码和超参数都是对的。

References

[1] M. Hessel, J. Modayil, H Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. 2017.

[2] Zhuoran Yang, Yongxin Chen, Mingyi Hong, and Zhaoran Wang. *Provably Global Convergence of Actor-Critic: A Case for Linear Quadratic Regulator with Ergodic Cost.* Curran Associates Inc., Red Hook, NY, USA, 2019.

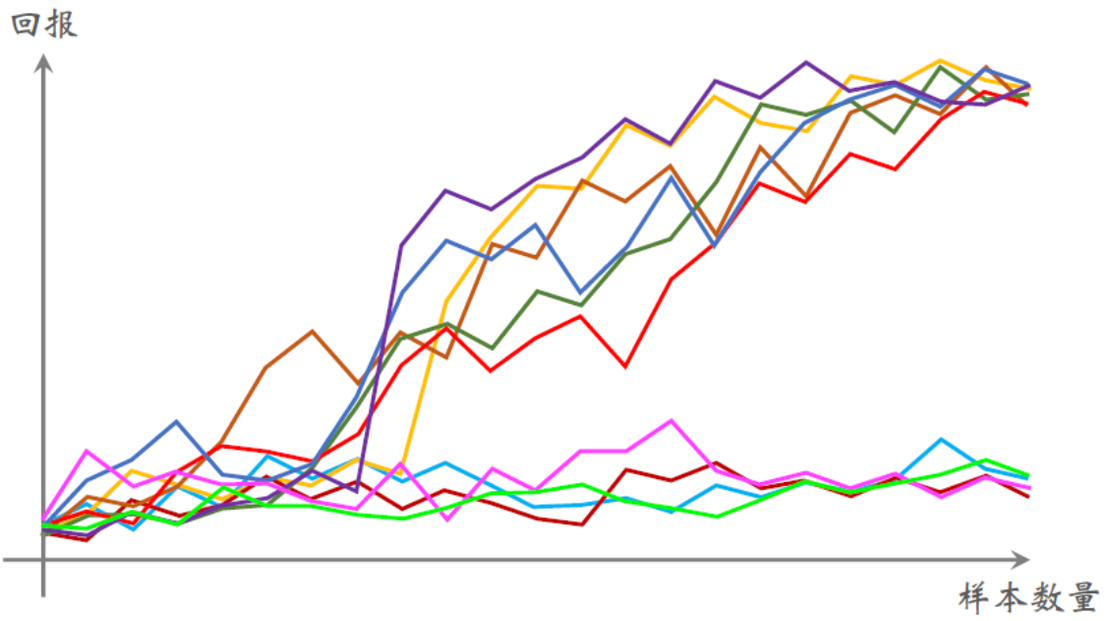


Figure 15: 用完全相同的超参数，用不同的随机种子，往往会得到截然不同的收敛曲线。