Introduction

深度增强学习Deep Reinforcement Learning是将深度学习与增强学习结合起来从而实现从Perception感知
Action动作的端对端学习的一种全新的算法。简单的说，就是和人类一样，输入感知信息比如视觉，然
通过深度神经网络，直接输出动作，中间没有hand-crafted工作。深度增强学习具备使机器人实现完全自
的学习一种甚至多种技能的潜力。

虽然将深度学习和增强学习结合的想法在几年前就有人尝试，但真正成功的开端是DeepMind在NIPS 20
上发表的**Playing Atari with Deep Reinforcement Learning**一文，之后DeepMind在Nature上发表了改进版
DQN文章，引起了广泛的关注。而Hinton，Bengio及Lecun三位大神在Nature上发表的Deep Learning 综述
文最后也将Deep Reinforcement Learning作为未来Deep Learning的发展方向。

Deep Reinforcement Learning因为具备真正实现AI的潜力，受到了Google等企业的关注。DeepMind 50多
的团队被Google以4亿美元的价格收购。而15年12月份刚刚由Elon Musk牵头成立的OpenAI，则一开始就
得了10亿美元的投资，而OpenAI中的好几位成员都来自UC Berkerley的Pieter Abbeel团队。

Pieter Abbeel团队紧随DeepMind之后，采用另一种方法直接实现了机器人的End-to-End学习，其成果也
起了大量的媒体报道和广泛关注。今年的NIPS 2015 更是由Pieter Abbeel及DeepMind的David Silver联合
织了Deep Reinforcement Learning workshop。可以说，目前在Deep Reinforcement Learning取得开拓性进
的主要集中在DeepMind和UC Berkerley团队。

为了深入地理解Deep Reinforcement Learning, 需要具备两大背景知识:

- 深度学习特别是CNN卷积神经网络（由于感知Perception大多来自于视觉信息）
- 增强学习基础知识

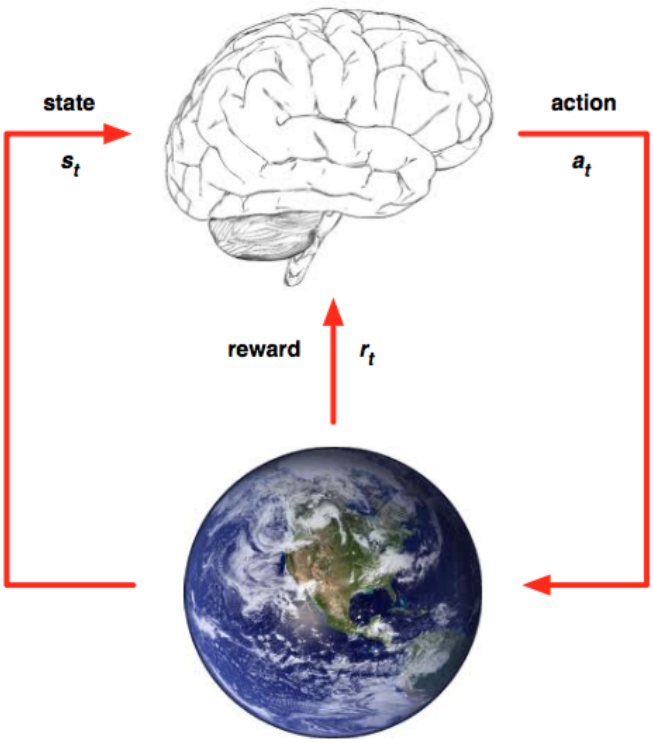
本文将介绍理解Deep Q-network所需要的增强学习的背景知识。

What is Reinforcement Learning 增强学习是什么

在人工智能领域，一般用 `Agent` 来表示一个具备行为能力的物体，比如机器人，无人车，人等等。那么
强学习考虑的问题就是Agent和 环境`environment$E$` 之间交互的任务。比如一个机械臂要拿起一个手机
那么机械臂周围的物体包括手机就是环境，机械臂通过外部的比如摄像头来感知环境，然后机械臂需要

出动作来实现拿起手机这个任务。再举玩游戏的例子，比如我们玩极品飞车游戏，我们只看到屏幕，这是环境，然后我们输出动作（键盘操作）来控制车的运动。

那么，不管是什么样的任务，都包含了一系列的 动作`action`，观察`Observation` 还有 反馈值`Reward`。所谓的Reward就是Agent执行了动作与环境进行交互后，环境会发生变化，变化的好与坏就用reward来表示，如上面的例子。如果机械臂离手机变近了，那么reward就应该是正的，如果玩赛车游戏赛车越来越偏离道，那么reward就是负的。接下来这里用了Observation观察一词而不是环境那是因为Agent不一定能得到环境的所有信息，比如机械臂上的摄像头就只能得到某个特定角度的画面。因此，只能用Observation来表Agent获取的感知信息。



- At each step t the agent:
  - Receives state s_t
  - Receives scalar reward r
  - Executes action a_t
- The environment:
  - Receives action a_t
  - Emits state s_t
  - Emits scalar reward r_t

上面这张图（来自David Silver）可以很清楚的看到整个交互过程。事实上，这就是人与环境交互的一种型化表示。在每个时间点time-step Agent都会从可以选择的动作集合A中选择一个　执行.这个动作集合以是连续的比如机器人的控制也可以是离散的比如游戏中的几个按键。动作集合的数量将直接影响整个务的求解难度，因此DeepMind才从玩最简单的游戏做起。

那么知道了整个过程，任务的目标就出来了，那就是要能获取尽可能多的reward。没有目标，控制也就从谈起，因此，获取reward就是一个量化的标准，reward越多，就表示执行得越好。每个时间片，Agen是根据当前的观察来确定下一步的动作。每次的观察就作为Agent的所处的 状态`state`，因此，状态Sta和动作Action存在映射关系，也就是一个state可以对应一个action，或者对应不同动作的概率（常常用概

来表示，概率最高的就是最值得执行的动作）。那么state到action的过程就称之为一个 策略Policy 。当然，也可以是之前的一系列的状态动作集合到action的映射，在下一节我们会看到只要当前状态就行）一般用 表示，也就是需要找到以下关系：

$$a = \pi(s)$$

$$\pi(a|s)$$

其中a是action，s是state。

> 增强学习的任务就是找到一个最优的策略policy从而使reward最多。

我们一开始并不知道最优的策略是什么，因此往往从随机的策略开始，使用随机的策略进行试验，就可得到一系列的状态,动作和反馈：

$$\{s_1, a_1, r_1, s_2, a_2, r_2, \ldots, a_{t-1}, s_t\}$$

这就是一系列的 样本Sample 。增强学习的算法就是需要根据这些样本来改进policy，从而使得得到的样中的reward更好。由于这种让reward越来越好的特性，所以这种算法就叫做增强学习Reinforcement Learning。

MDP（Markov Decision Process）马尔科夫决策过程

由于增强学习的样本是一个时间序列，因此将增强学习的问题模型化，就引入了MDP。

所谓的MDP，是基于这样一种假设：

> The future is independent of the past given the present

也就是，一个状态S_t是Markov当且仅当

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_1, \ldots, s_t)$$

P为概率。

简单的说就是下一个状态仅取决于当前的状态和当前的动作。注意这里的状态是完全可观察的全部的环
状态
由MDP的假设，如果这个世界就是MDP的，如果再加上一个假设，每个动作都是由完全的环境（比如
每个细胞导致的精神状态，意识）决定，那么有一个初始状态，后继状态就是全部确定的。当然，现实
况环境一般不完全可观察，然后有一些随机性stochastic是人类无法确定的。

绝大多数的增强学习都可以模型化为MDP的问题。

因为前面我们看到，我们要根据当前的状态来决定动作。而且由MDP我们知道我们只要使用当前的状态
行估计。但是，这里注意本节考虑的状态是完全可观察。对于部分可观察的状态，那么只能进行估计。

既然一个状态对应一个动作，或者动作的概率，而有了动作，下一个状态也就确定了。这就意味着每个
态可以用一个确定的值来进行描述。可以由此判断一个状态是好的状态还是不好的状态。比如，向左边
就是悬崖，悬崖肯定不是好的状态，再走一步可能就挂了，而向右走就是黄金，那么右边的状态就是好
状态。

那么状态的好坏其实等价于对未来回报的期望。因此，引入 回报`Return` 来表示某个时刻t的状态将具备
回报：

$$G_t = R_{t+1} + \lambda R_{t+2} + \ldots = \sum_{k=0}^{\infty} \lambda^k R_{t+k+1}$$

上面有一个λ是discount factor折扣因子，就是说一般当下的反馈是比较重要的，时间越久，影响越小。

那么实际上除非整个过程结束，否则显然我们无法获取所有的reward来计算出每个状态的Return，因此
再引入一个概念 估值函数`Value Function` ,用value function $v(s)$来表示一个状态未来的潜在价值。这就
变成是向左看感觉那么是悬崖然后左边的状态的估值就低。

从定义上看，value function就是回报的期望：

$$v(s) = E[G_t | S_t = s]$$

Bellman Equation

定义出value function代表问题解决了一半，接下来就是如何求解value function的问题。实际上发现，只从定义出发，就可以找到解决办法：

$$\begin{aligned} v(s) &= E[G_t|S_t = s] \\ &= E[R_{t+1} + \lambda R_{t+2} + \lambda^2 R_{t+3} + \ldots |S_t = s] \\ &= E[R_{t+1} + \lambda(R_{t+2} + \lambda R_{t+3} + \ldots)|S_t = s] \\ &= E[R_{t+1} + \lambda G_{t+1}|S_t = s] \\ &= E[R_{t+1} + \lambda v(S_{t+1})|S_t = s] \end{aligned}$$

因此

$$v(s) = E[R_{t+1} + \lambda v(S_{t+1})|S_t = s]$$

这就是Bellman 方程的基本形态，它表明value function是可以通过迭代来进行计算的。

Action-Value function 动作估值函数

前面我们引出了估值函数，考虑到每个状态之后都有多种动作可以选择，每个动作之下的状态又多不一样，我们更关心在某个状态下的不同动作的估值。显然。如果知道了每个动作的估值，那么就可以选择值最好的一个动作去执行了。这就是Action-Value function (s, a) 。那么同样的道理，也是使用reward来示，只是这里的reward和之前的reward不一样，这里是执行完动作action之后得到的reward，之前state对的reward则是多种动作对应的reward的期望值。显然，动作之后的reward更容易理解。

那么，有了上面的定义，动作估值函数就为如下表示：

$$\begin{aligned} Q^{\pi}(s, a) &= E[r_{t+1} + \lambda r_{t+2} + \lambda^2 r_{t+3} + \ldots |s, a] \\ &= E_{s'}[r + \lambda Q^{\pi}(s', a')|s, a] \end{aligned}$$

Optimal value function 最优动作估值函数

能计算动作估值函数是不够的，因为我们需要的是最优策略，现在求解最优策略等价于求解最优的valu function，找到了最优的value function，自然而然策略也就是找到。（当然，这只是求解最优策略的一种法，也就是value-based approach，由于DQN就是value-based，因此这里只讲这部分，之后我们会看到还

policy-based和model-based方法。一个就是直接计算策略函数，一个是估计未来的环境变化，从而可以i
出最优策略）
那么套用上一节得到的value function，可以得到

$$Q^*(s,a) = E_{s'}[r + \lambda \max_{a'} Q^*(s',a')|s,a]$$

因为最优的Q值必然为最大值，所以，等式右侧的Q值必然为使a'取最大的Q值。
很多增强学习的算法都是基于上面这个公式来估计出action value function，使用Bellman 公式来迭代更新
看下一小节。

Value iteration 估值迭代

上面的公式揭示了Q值可以通过迭代来进行计算，那么接下来我们就考虑如何迭代，因为等式右侧是下
个状态和动作的Q值。那么，直观的理解，我们还在计算当前的Q值，怎么能有下一个Q值呢？没有错。
以，我们只能用之前的Q值。也就是没次根据新得到的reward和原来的Q值来更新现在的Q值。根据这个
想，就可以得到下面的迭代公式：

$$Q_{i+1}(s,a) = E_{s'}[r + \lambda \max_{a'} Q_i(s',a')|s,a]$$

理论上可以证明这样的value iteration能够使Q值收敛到最优的action-value function。即当$i \to \infty$时
$Q\_i \to Q^*$。

Q Learning

Q Learning的思想完全根据value iteration得到。但要明确一点是value iteration每次都对所有的Q值更新一
遍，也就是所有的状态和动作。但事实上在实际情况下我们没办法遍历所有的状态，还有所有的动作，
们只能得到有限的系列样本。因此，只能使用有限的样本进行操作。那么，怎么处理？Q Learning提出
一种更新Q值的办法：

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha(R_{t+1} + \lambda \max_a Q(S_{t+1},a) - Q(S_t,A_t))$$

虽然根据value iteration计算出target Q值，但是这里并没有直接将这个Q值（是估计值）直接赋予新的Q
而是采用渐进的方式类似梯度下降，朝target迈近一小步，取决于α,这就能够减少估计误差造成的影响。
似随机梯度下降，最后可以收敛到最优的Q值。

具体的算法如下：

初始化 ，任意的数值，并且
重复（对每一节episode）：
初始化 状态S
重复（对episode中的每一步）：
使用某一个policy比如（)根据状态S选取一个动作执行
执行完动作后，观察reward和新的状态S'

$S \leftarrow S'$
循环直到S终止

Exploration and Exploitation 探索与利用

在上面的算法中，我们可以看到需要使用某一个policy来生成动作，也就是说这个policy不是优化的那个policy，所以Q-Learning算法叫做Off-policy的算法。另一方面，因为Q-Learning完全不考虑model模型也是环境的具体情况，只考虑看到的环境及reward，因此是model-free的方法。

回到policy的问题，那么要选择怎样的policy来生成action呢？有两种做法：

- 随机的生成一个动作
- 根据当前的Q值计算出一个最优的动作，这个policy π称之为greedy policy，也就是

$$\pi(S_{t+1}) = arg \max_a Q(S_{t+1}, a)$$

使用随机的动作就是exploration，也就是探索未知的动作会产生的效果，有利于更新Q值，获得更好的policy。而使用greedy policy也就是target policy则是exploitation，利用policy，这个相对来说就不好更新出好的Q值，但可以得到更好的测试效果用于判断算法是否有效。

将两者结合起来就是所谓的 policy,ϵ一般是一个很小的值，作为选取随机动作的概率值。可以改ϵ的值从而得到不同的exploration和exploitation的比例。

Value function approximation 估值函数估计

有了Q-Learning，已经可以解决一些增强学习的任务了。但是依然有个根本的问题就是状态空间和动作空间的量级问题。如果状态空间特别大，那么$Q(s,a)$这个矩阵就会非常大，仅通过反复的测试无法获得足够的样本来遍历每个状态，这必然导致算法失败。联系到人类的行为特点，我们也不可能遍历所有的情况，更多的是将新的情况与记忆进行比对，如果相似，那么就可以采用相似的做法。因此，自然而然的，如能用一个函数来表示value function，输入任意的状态都能输出结果，那么就可以把Q矩阵的更新问题变为一个函数拟合问题，相近的状态也就可以得到相近的输出动作。这就是value function approximator：

,我们需要更新参数w来使得Q函数逼近与最优的Q值。

既然是函数优化问题，那么就可以使用监督学习supervised learning的做法：

1. 确定一个loss function
2. 计算w关于loss function的梯度
3. 使用梯度下降比如随机梯度下降SGD等方法来更新θ

关于Supervised Learning可以参考Andrew Ng的cousera 机器学习Machine Learning的课程

通常情况下增强学习使用线性的函数估计，但有时候可以使用非线性的函数。那么如果使用深度神经网络来作为这个value function approximator呢？这样，我们就可以得到Deep Q-Learning，也就是Deep Reinforcement Learning的基本思想了。

Deep Q-Learning

Step 1: 用一个深度神经网络来作为Q值的网络，参数为w：

$$Q(s,a,w) \approx Q^{\pi}(s,a)$$

Step 2: 在Q值中使用均方差mean-square error 来定义目标函数objective function也就是loss function

$$L(w) = E[(\underbrace{r + \gamma \max_{a'} Q(s',a',w)}_{Target} - Q(s,a,w))^2]$$

可以看到，这里就是使用了Q-Learning要更新的Q值作为目标值。有了目标值，又有当前值，那么偏差能通过均方差来进行计算。

Step 3：计算参数w关于loss function的梯度

这个可以直接计算得到

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

Step 4: 使用SGD实现End-to-end的优化目标

有了上面的梯度，而 $\frac{\partial Q(s,a,w)}{\partial w}$ 可以从深度神经网络中进行计算，因此，就可以使用SGD 随机梯度下降来新参数，从而得到最优的Q值。