

# Introduction of Deep Reinforcement Learning

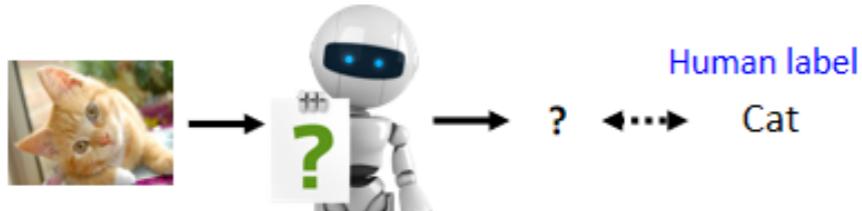
那这一堂课啊,我们要讲的是,Deep Reinforcement Learning,也就是 RL,那我想这个 RL 啊,Reinforcement Learning 啊,大家一定一点都不陌生,因为你知道很多很潮的应用,AlphaGo 等等,它背后呢,用的就是 RL 的技术,那 RL 可以讲的技术啊,非常非常地多,它不是在一堂课裡面可以讲得完的,我甚至觉得说,如果有人要把它开成一整个学期的课,可能也是有这麼多东西可以讲

所以今天啊,这堂课的目的,并不是要告诉你有关 RL 的一切,而是让大家有一个基本的认识,大概知道 RL 是什么样的东西,那 RL 相关的课程,你其实在网路上可以找到,非常非常多的参考的资料,那 RL 如果要讲得非常地艰涩,其实也是可以讲得非常地艰涩的

可是今天这一堂课啊,我们儘量避开太过理论的部分,我期待这堂课可以让你做到的,并不是让你听了觉得,哇 RL 很困难啊,搞不清楚在做什么,而是期待让你觉得说,啊 RL 原来就只是这样而已,我自己应该也做得起,希望这一堂课可以达到这一个目的

## Supervised Learning → RL

那什么是 Reinforcement Learning 呢,到目前为止啊,我们讲的几乎都是 Supervised Learning,假设你要做一个 Image 的 Classifier,你不只要告诉机器,它的 Input 是什么,你还要告诉机器,它应该输出什么样的 Output,然后接下来呢,你就可以 Train 一个 Image 的 Classifier

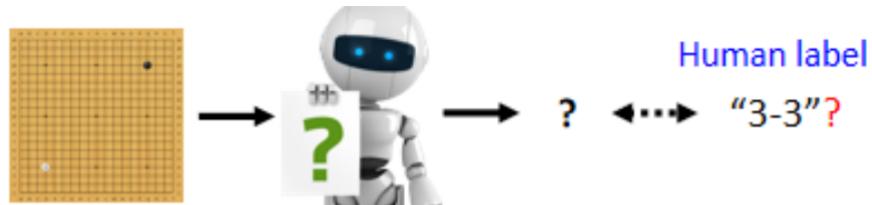


那在多数这门课讲到目前为止的技术,基本上都是基于 Supervised Learning 的方法,就算是我们在讲 Self Supervised Learning 的时候,我们其实也是很类似 Supervised Learning 的方法,只是我们的 Label,不需要特别雇用人力去标记,它可以自动产生

或者是我们在讲 Auto-encoder 的时候,我们虽然说它是一个 Unsupervised 的方法,我们没有用到人类的标记,但事实上,我们还是有一个 Label,只是这个 Label,不需要耗费人类的力量来产生而已

但是 RL 就是另外一个面向的问题了,在 RL 裡面,我们遇到的问题是这样子的,我们,机器当我们给它一个输入的时候,我们不知道最佳的输出应该是什么

举例来说,假设你要叫机器学习下围棋



It is challenging to label data in some tasks.

..... machine can know the results are good or not.

用 Supervised Learning 的方法,好像也可以做,你就是告诉机器说,看到现在的盘势长这个样子的时候,下一步应该落子的位置在哪裡,但是问题是,下一步应该落子的位置到底应该在哪裡呢,哪一个是最好的下一步呢,哪一步是神之一手呢,可能人类根本就不知道

当然你可以说,让机器阅读很多职业棋士的棋谱,让机器阅读很多高段棋士的棋谱,也许这些棋谱裡面的答案,也许这些棋谱裡面给某一个盘势,人类下的下一步,就是一个很好的答案,但它是最好的答案呢,我们不知道,**在这个你不知道正确答案是什么的情况下,往往就是 RL 可以派上用场的时候,所以当你今天,你发现你要收集有标注的资料很困难的时候,正确答案人类也不知道是什么的时候,也许就是你可以考虑使用 RL 的时候**

**但是 RL 在学习的时候,机器其实也不是一无所知的,**我们虽然不知道正确的答案是什么,但是机器会知道什么是好,什么是不好,机器会跟环境去做互动,得到一个叫做 Reward 的东西,这我们等一下都还会再细讲

所以机器会知道,它现在的输出是好的还是不好的,来藉由跟环境的互动,藉由知道什么样的输出是好的,什么样的输出是不好的,机器还是可以学出一个模型

## Outline

---

好 那接下来呢,这是今天这份投影片的 Outline

What is RL? (Three steps in ML)

Policy Gradient

Actor-Critic

Reward Shaping

No Reward: Learning from Demonstration

首先呢,我们会从最基本的 RL 的概念开始,那在介绍这个 RL 概念的时候,有很多不同的切入点啦,也许你比较常听过的切入点是这样,比如说从 Markov Decision Process 开始讲起

那我们这边选择了一个比较不一样的切入点,我要告诉你说,虽然如果你自己读 RL 的文献的话,你会觉得,哇 RL 很複杂哦,跟一般的 Machine Learning 好像不太一样哦,但是我这边要告诉你说,RL 它跟我们这一门课学的 Machine Learning,是一样的框架

我们在今天这个这学期,一开始的第一堂课就告诉你说,Machine Learning 就是三个步骤,那 RL 呢,RL 也是一模一样的三个步骤,等一下会再跟大家说明

## What is RL? (Three Steps in ML)

---

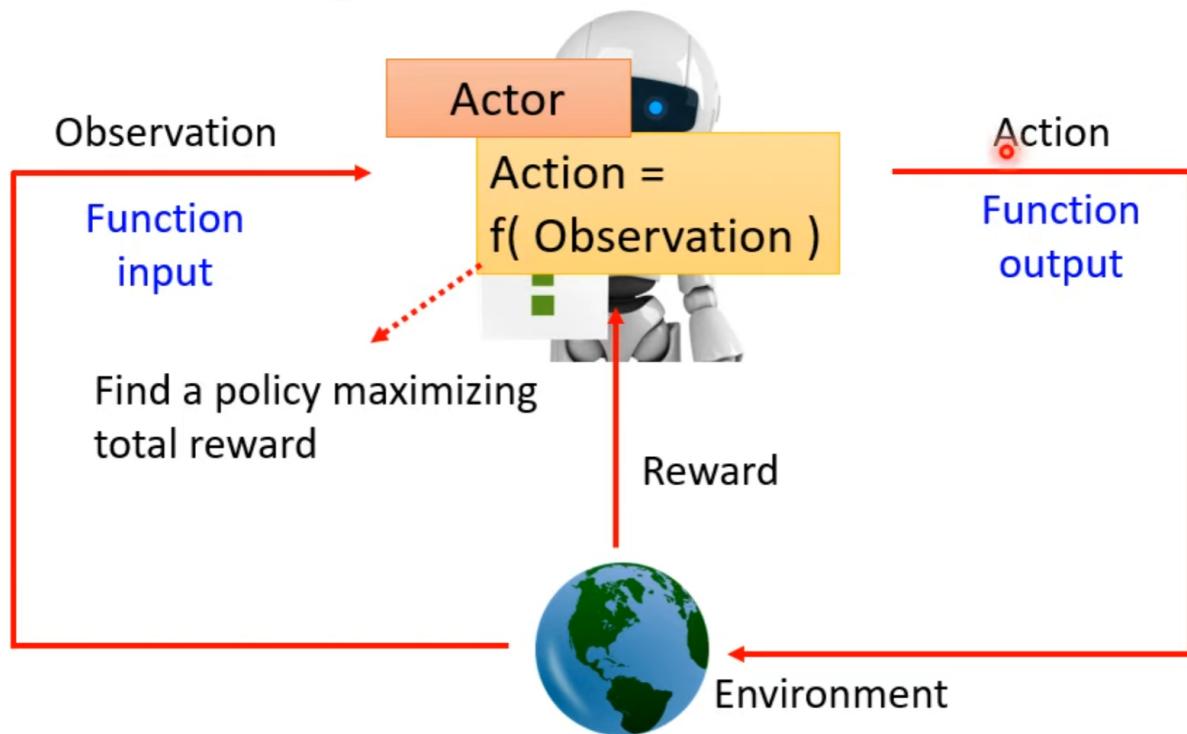
### Machine Learning ≈ Looking for a Function

---

在今天,在这个本学期这一门课的第一开始,就告诉你说,什么是机器学习,机器学习就是找一个 Function,Reinforcement Learning,RL 也是机器学习的一种,那它也在找一个 Function,它在找什么样的 Function 呢

那 Reinforcement Learning 裡面呢,我们会有一个Actor,还有一个 Environment,那这个 Actor 跟 Environment,会进行互动

# Machine Learning ≈ Looking for a Function



- 你的这个 Environment, 你的这个环境啊, 会给 Actor 一个 Observation, 会给, 那这个 Observation 呢, 就是 Actor 的输入
- 那 Actor 呢, 看到这个 Observation 以后呢, 它会有一个输出, 这个输出呢, 叫做 Action, 那这个 Action 呢, 会去影响 Environment
- 这个 Actor 採取 Action 以后呢, Environment 就会给予新的 Observation, 然后 Actor 呢, 会给予新的 Action, 那这个 Observation 是 Actor 的输入, 那这个 Action 呢, 是 Actor 的输出

所以 Actor 本身啊, 它就是一个 Function, 其实 Actor, 它就是我们要找的 Function, 这个 Function 它的输入, 就是环境给它的 Observation, 输出就是这个 Actor 要採取的 Action, 而今天在这个互动的过程中呢, 这个 Environment, 会不断地给这个 Actor 一些 Reward, 告诉它说, 你现在採取的这个 Action, 它是好的还是不好的

而我们今天要找的这个 Actor, 我们今天要找的这个 Function, 可以拿 Observation 当作 Input, Actor 当作 Output 的 Function, 这个 Function 的目标, 是要去 Maximizing, 我们可以从 Environment, 获得到的 Reward 的总和, 我们希望呢, 找一个 Function, 那用这个 Function 去跟环境做互动, 用 Observation 当作 Input, 输出 Action, 最终得到的 Reward 的总和, 可以是最大的, 这个就是 RL 要找的 Function

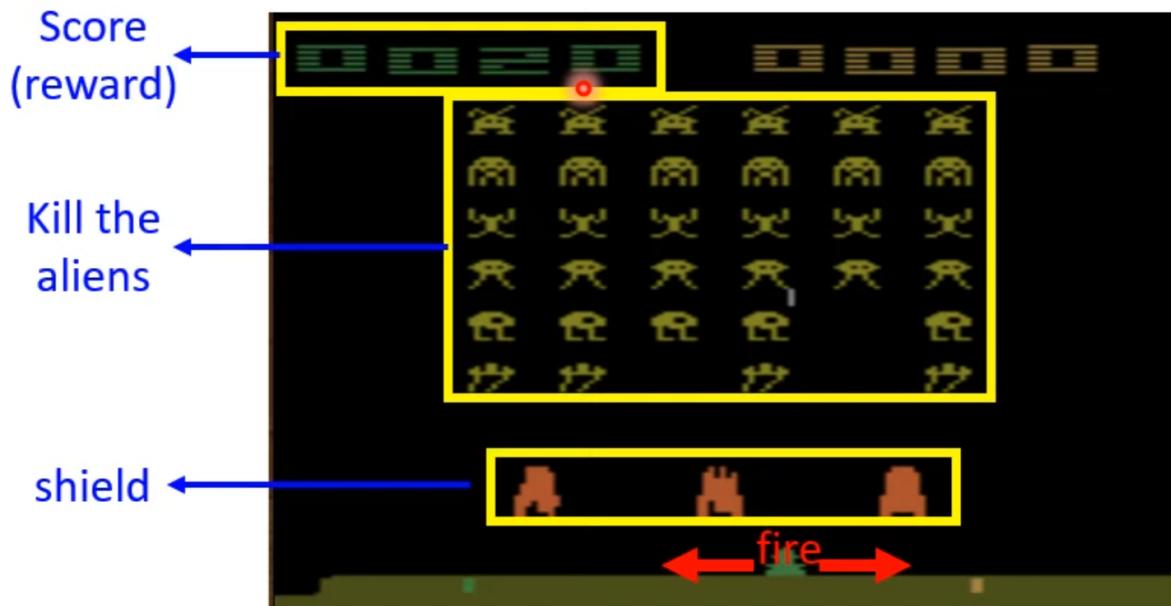
## Example: Playing Video Game

那我知道这样讲, 你可能还是觉得有些抽象, 所以我们举更具体的例子, 那等一下举的例子呢, 都是用 Space Invader 当作例子啦, 那 Space Invader 就是一个非常简单的小游戏, 那 RL 呢, 最早的几篇论文, 也都是玩, 让那个机器呢, 去玩这个 Space Invader 这个游戏

在 Space Invader 裡面呢

# Example: Playing Video Game

- Space invader



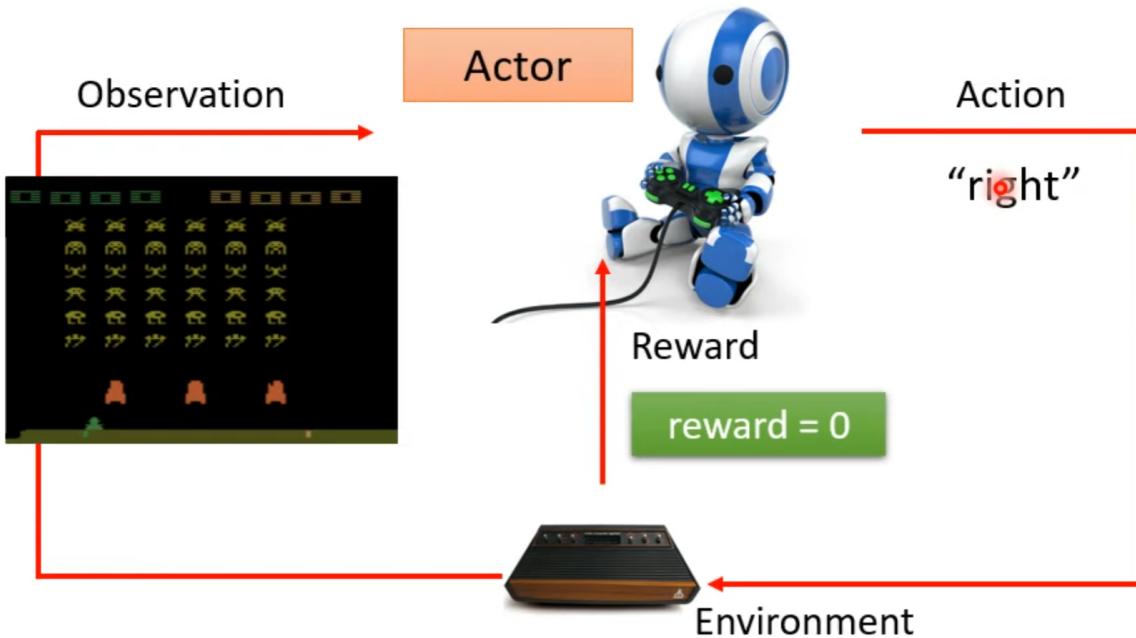
- 你要操控的是下面这个绿色的东西,这个下面这个绿色的东西呢,是你的太空梭,你可以采取的行为,也就是 Action 呢有三个,左移 右移跟开火,就这三个行为,然后你现在要做的事情啊,就是杀掉画面上的这些外星人。画面上这些黄色的东西,也就是外星人啦,然后你开火,击中那些外星人的话,那外星人就死掉了。
- 那前面这些东西是什么呢,那个是你的防护罩,如果你不小心打到自己的防护罩的话,你的防护罩呢,也是会被打掉的,那你可以躲在防护罩后面,你就可以挡住外星人的攻击
- 然后接下来呢会有分数,那在萤幕画面上会有分数,当你杀死外星人的时候,你会得到分数,或者是在有些版本的 Space Invader 裡面,会有一个补给包,从上面横过去飞过去,那你打到补给包的话,会被加一个很高的分数,那这个 Score 呢,就是 Reward,就是环境给我们的 Reward

那这个游戏呢,它是会终止的,那什么时候终止呢,当所有的外星人都被杀光的时候就终止,或者是呢,外星人其实也会对你的母舰开火啦,外星人击中你的母舰 你也是会,这个你就被摧毁了,那这个游戏呢,也就终止了,好 那这个是介绍一下,Space Invader 这一个游戏,

那如果你今天呢,要用 Actor 去玩 Space Invader,大概会像是什么样子呢

现在你的 Actor 啊,Actor 虽然是一个机器,但是它是坐在人的这一个位置,它是站在人这一个角度,去操控摇杆,去控制那个母舰,去跟外星人对抗,而你的环境是什么,你的环境呢,是游戏的主机,游戏的主机这边去操控那些外星人,外星人去攻击你的母舰,所以 Observation 是游戏的画面

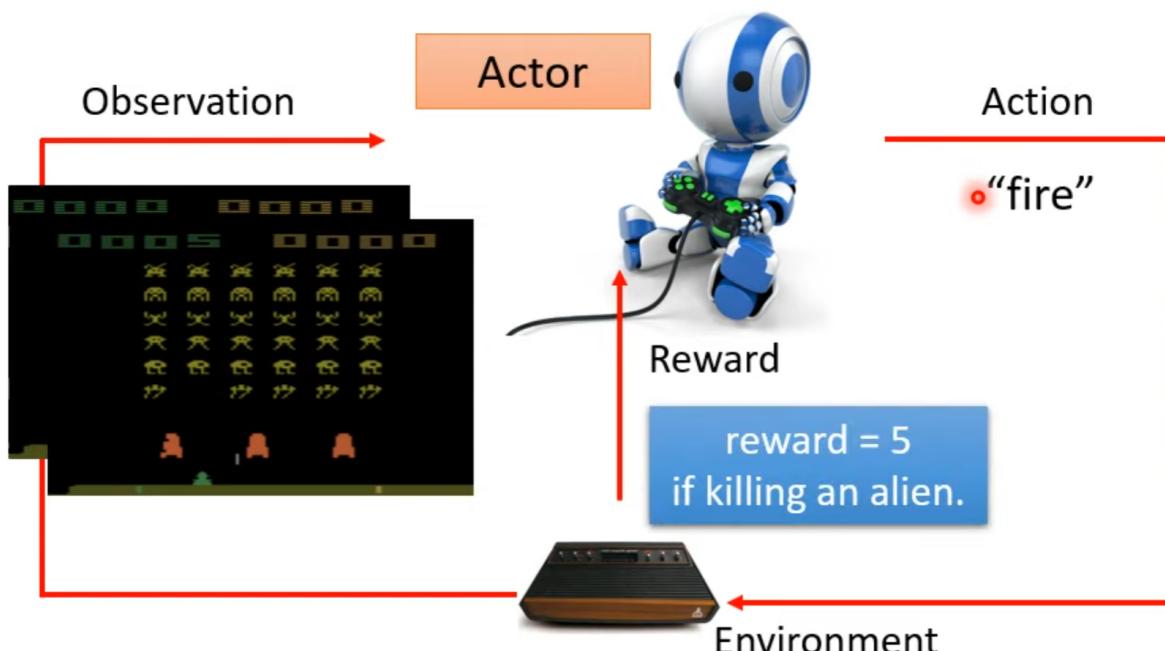
# Example: Playing Video Game



所以对 Actor 来说,它看到的,其实就跟人类在玩游戏的时候,看到的东西是一样的,就看到一个游戏的画面那输出呢,就是 Actor 可以採取的行为,那可以採取哪些行为,通常是事先定义好的,在这个游戏裡面,就只有向左 向右跟开火,三种可能的行为而已,好 那当你的 Actor 採取向右这个行为的时候,那它会得到 Reward 那因为在这个游戏裡面,只有杀掉外星人会得到分数,而我们就是把分数定义成我们的 Reward,那向左 向右其实并不会,不可能杀掉任何的外星人,所以你得到的 Reward 呢,就是 0 分,好 那你採取一个 Action 以后呢,游戏的画面就变了

- 游戏的画面变的时候,就代表了有了新的 Observation 进来
- 有了新的 Observation 进来,你的 Actor 就会决定採取新的 Action

# Example: Playing Video Game



你的 Actor 是一个 Function,这个 Function 会根据输入的 Observation,输出对应的 Action,那新的画面进来,假设你的 Actor,现在它採取的行为是开火,而开火这个行为正好杀掉一隻外星人的时候,你就会得到分数,那这边假设得到的分数是 5 分,杀那个外星人,得到的分数是 5 分,那你就得到 Reward 等于 5

那这个呢,就是拿 Actor 去玩,玩这个 Space Invader 这个游戏的状况,好 那这个 Actor 呢,它想要学习的是什么呢,我们在玩游戏的过程中,会不断地得到 Reward,那在刚才例子裡面,做第一个行为的时候,向右的时候得到的是 0 分,做第二个行为,开火的时候得到的是 5 分,那接下来你採取了一连串行为,都有可能给你分数

而 Actor 要做的事情,我们要学习的目标,我们要找的这个 Actor 就是,我们想要 Learn 出一个 Actor,这个 Actor,这个 Function,我们使用它在这个游戏裡面的时候,可以让我们得到的 Reward 的总和会是最大的,那这个就是拿 Actor 去,这个就是 RL 用在玩这个小游戏裡面的时候,做的事情

## Example: Learning to play Go

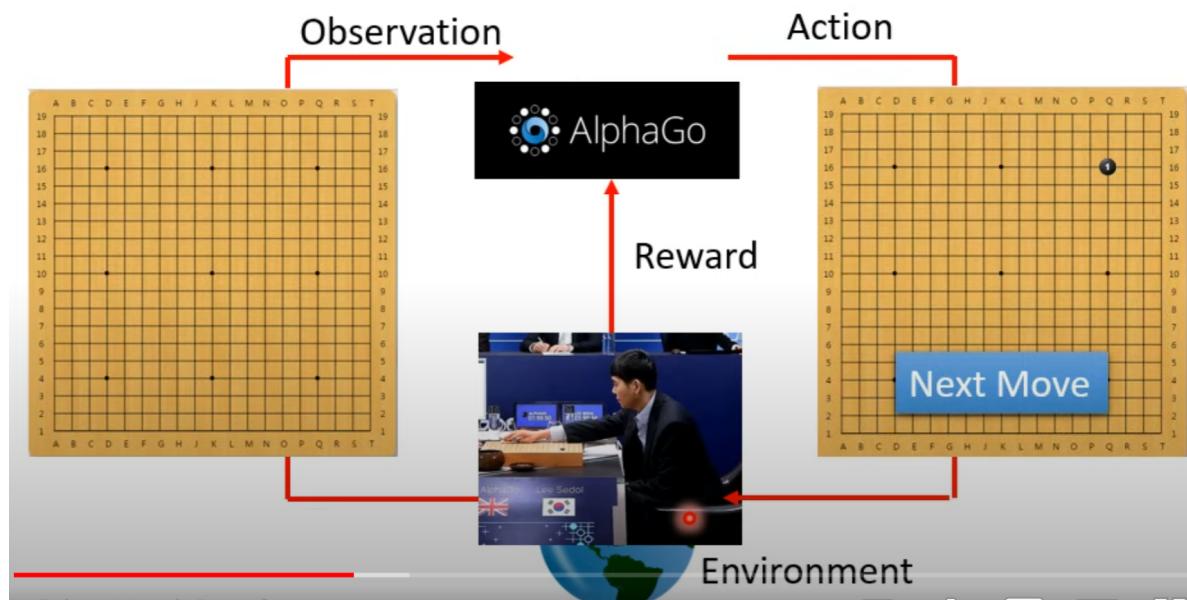
那其实如果把 RL 拿来玩围棋,拿来下围棋,其实做的事情跟小游戏,其实也没有那麽大的差别,只是规模跟问题的複杂度不太一样而已

那如果今天你要让机器来下围棋,那你的 Actor 就是就是 AlphaGo,那你的环境是什么,你的环境就是 AlphaGo 的人类对手

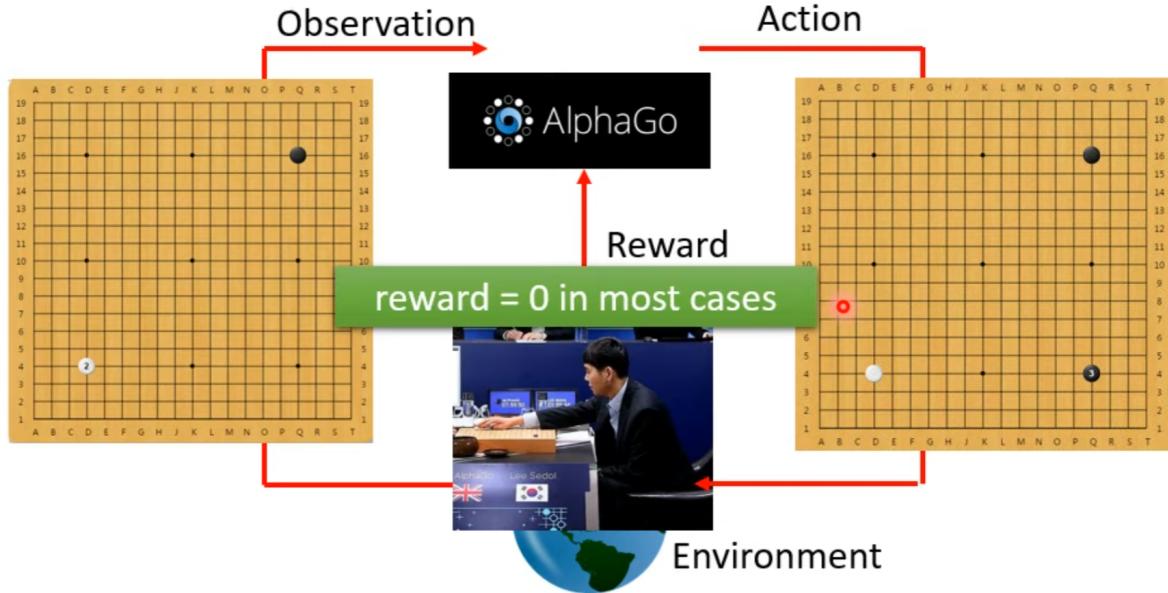
你的 Actor 的输入就是棋盘,棋盘上黑子跟白子的位置,那如果是在游戏的一开始,棋盘上就空空的,空空如也,上面什么都没有,没有任何黑子跟白子

那这个 Actor 呢,看到这个棋盘呢,它就要产生输出,它就要决定它下一步,应该落子在哪裡,那如果是围棋的话,你的输出的可能性就是有  $19 \times 19$  个可能性,那这  $19 \times 19$  个可能性,每一个可能性,就对应到棋盘上的一个位置

那假设现在你的 Actor,决定要落子在这个地方

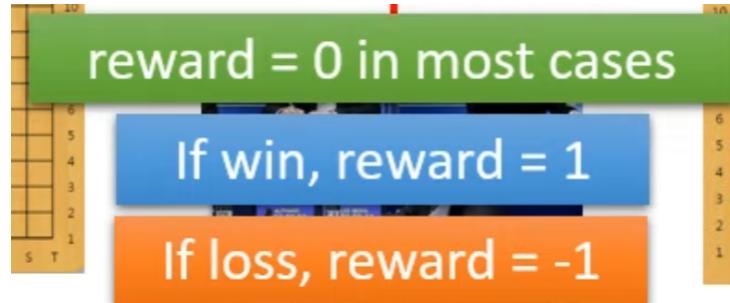


那这一个结果,就会输入给你的环境,那其实就是一个棋士,然后呢 这个环境呢,就会再产生新的 Observation,因为这个李世石这个棋士呢,也会再落一子,那现在看到的环境又不一样了,那你的 Actor 看到这个新的 Observation,它就会产生新的 Action,然后就这样反覆继续下去



你就可以让机器做下围棋这件事情,好 那在这个,在这个下围棋这件事情裡面的 Reward,是怎麽计算的呢

在下围棋裡面,你所採取的行为,几乎都没有办法得到任何 Reward,在下围棋这个游戏裡,在下围棋这件事情裡面呢,你会定义说,如果赢了,就得到 1 分,如果输了就得到 -1 分



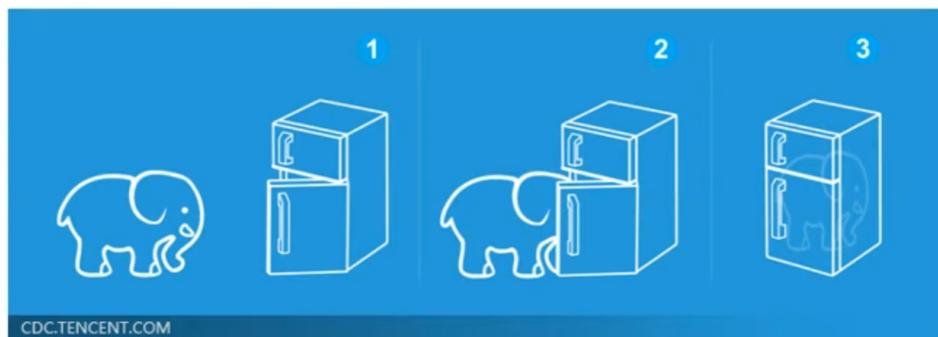
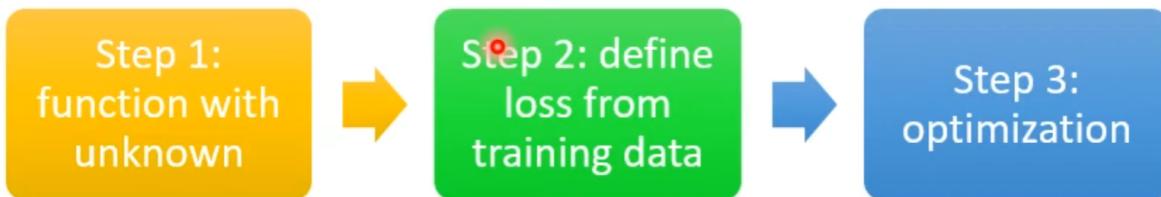
也就是说在下围棋这整个,这个你的 Actor 跟环境互动的过程中,其实只有游戏结束,只有整场围棋结束的最后一子,你才能够拿到 Reward,就你最后,最后 Actor 下一子下去,赢了,就得到 1 分,那最后它落了那一子以后,游戏结束了,它输了,那就得到 -1 分,那在中间整个互动的过程中的 Reward,就都算是 0 分,没有任何的 Reward,那这个 Actor 学习的目标啊,就是要去最大化,它可能可以得到的 Reward

## Machine Learning is so simple .....

刚才讲的也许你都已经听过了,那这个是 RL 最常见的一种解说方式,那接下来要告诉你说,RL 跟机器学习的 Framework,它们之间的关係是什么

开学第一堂课就告诉你说,Machine Learning 就是三个步骤

# Machine Learning is so simple .....



1. 第一个步骤,你有一个 Function,那个 Function 裡面有一些未知数,Unknown 的 Variable,这些未知数是要被找出来的
2. 第二步,订一个 Loss Function,第三步,想办法找出未知数去最小化你的 Loss
3. 第三步就是 Optimization

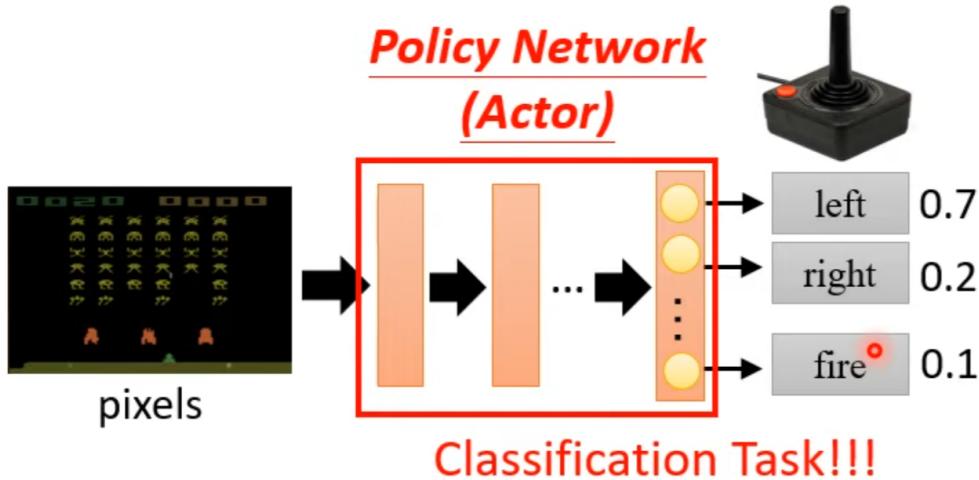
而 RL 其实也是一模一样的三个步骤

## Step 1: Function with Unknown

第一个步骤,我们现在有未知数的这个 Function,到底是什么呢,这个有未知数的 Function,就是我们的 Actor,那在 RL 裡面,**你的 Actor 呢,就是一个 Network,那我们现在通常叫它 Policy 的 Network**

那在过去啊,在还没有把 Deep Learning 用到 RL 的时候,通常你的 Actor 是比较简单的,它不是 Network,它可能只是一个 Look-Up-Table,告诉你说看到什么样的输入,就产生什么样的输出,那今天我们都已经知道要用 Network,来当做这个 Actor,那这个 Network,其实就是一个很复杂的 Function

# Step 1: Function with Unknown



- Input of neural network: the observation of machine represented as a vector or a matrix
- Output neural network : each action corresponds to a neuron in output layer

这个複杂的 Function 它的输入是什么呢,它的输入就是游戏的画面,就是游戏的画面,这个游戏画面上的 Pixel,像素,就是这一个 Actor 的输入

那它的输出是什么呢,它的输出就是,每一个可以採取的行为,它的分数,每一个可以採取的 Action 它的分数,举例来说 输入这样的画面,给你的 Actor,你的 Actor 其实就是一个 Network,它的输出可能就是给,向左 0.7 分,向右 0.2 分,开火 0.1 分

那事实上啊,这件事情跟分类是没有什么两样的,你知道分类就是输入一张图片,输出就是决定这张图片是哪一个类别,那你的 Network 会给每一个类别,一个分数,你可能会通过一个 Softmax Layer,然后每一个类别都有个分数,而且这些分数的总和是 1

那其实在 RL 裡面,你的 Actor 你的 Policy Network,跟分类的那个 Network,其实是一模一样的,你就是输入一张图片,输出其实最后你也会有个 Softmax Layer,然后呢,你就会 Left、Right 跟 Fire,三个 Action 各给一个分数,那这些分数的总和,你也会让它 1

那至于这个 Network 的架构呢,那你就可以自己设计了,要设计怎麽样都行,比如说如果输入是一张图片,欸也许你就会想要用 CNN 来处理

不过在助教的程式裡面,其实不是用 CNN 来处理啦,因为在我们的作业裡面,其实在玩游戏的时候,不是直接让我们的 Machine 去看游戏的画面,让它直接去看游戏的,让它直接去看游戏的画面比较难做啦,所以我们可以让,看这个跟现在游戏的状况有关的一些参数而已,所以在这个助教的,在这个作业的这个 Sample Code 裡面呢,还没有用到 CNN 那麽複杂,就是一个简单的 Fully Connected Network,但是假设你要让你的 Actor,它的输入真的是游戏画面,欸 那你可能就会採取这个 CNN,你可能就用 CNN 当作你的 Network 的架构

甚至你可能说,我不要只看现在这一个时间点的游戏画面,我要看整场游戏到目前为止发生的所有事情,可不可以呢

可以,那过去你可能会用 RNN 考虑,现在的画面跟过去所有的画面,那现在你可能会想要用 Transformer,考虑所有发生过的事情,所以 Network 的架构是可以自己设计的,只要能够输入游戏的画面,输出类似像类别这样的 Action 就可以了,那最后机器会决定採取哪一个 Action,取决于每一个 Action 取得的分数

常见的做法啊,是直接把这个分数,就当做一个机率,然后按照这个机率,去 Sample,去随机决定要採取哪一个 Action

举例来说 在这个例子裡面,向左得到 0.7 分,那就是有 70% 的机率会採取向左,20% 的机率会採取向右,10% 的机率会採取开火

那你可能会问说,为什么不是用 argmax 呢,为什么不是看 Left 的分数最高,就直接向左呢,你也可以这样做,但是在助教的程式裡面,还有多数 RL 应用的时候 你会发现,我们都是採取 Sample,

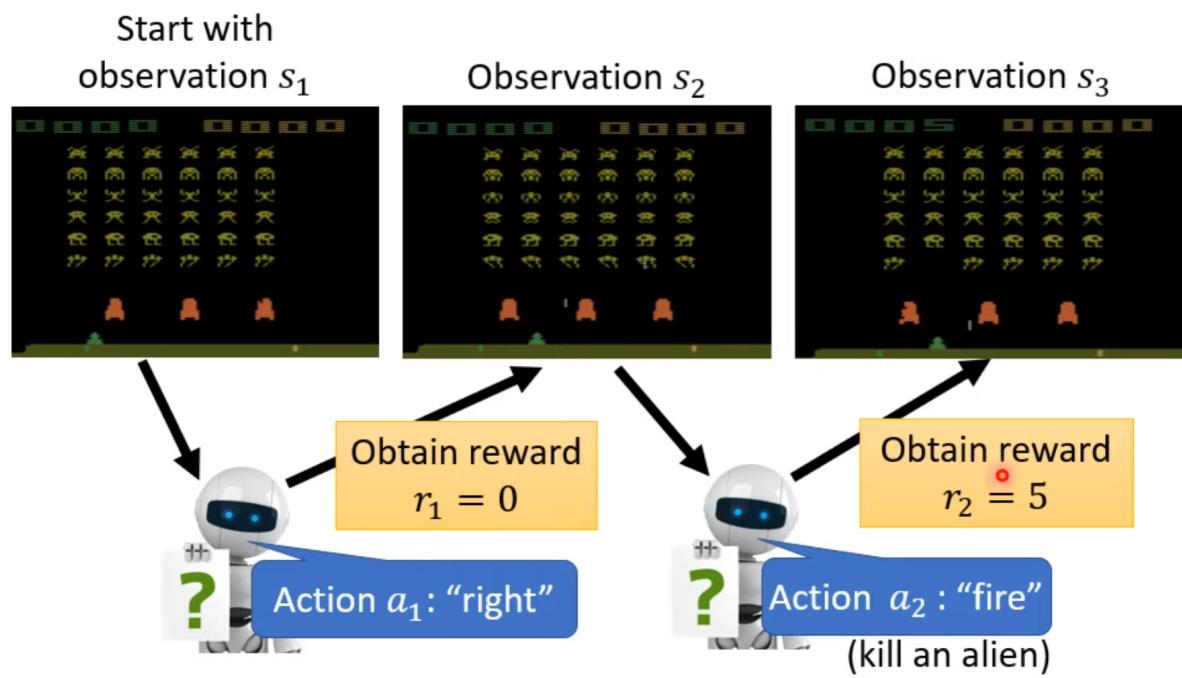
采取 Sample 有一个好处是说,今天就算是看到同样的游戏画面,你的机器每一次採取的行为,也会略有不同,那在很多的游戏裡面这种随机性,也许是重要的,比如说你在做剪刀石头布的时候,如果你总是会出石头,就跟小叮噹一样,那你就很容易被打爆,如果你有一些随机性,就比较不容易被打爆

那其实之所以今天的输出,是用随机 Sample 的,还有另外一个很重要的理由,那这个我们等一下会再讲到,好 所以这是第一步,我们有一个 Function,这个 Function 有 Unknown 的 Variable,我们有一个 Network,那裡面有参数,这个参数就是 Unknown 的 Variable,就是要被学出来的东西,这是第一步

## Step 2: Define "Loss"

然后接下来第二步,我们要定义 Loss,在 RL 裡面,我们的 Loss 长得是什么样子呢,我们再重新来看一下,我们的机器跟环境互动的过程,那只是现在用不一样的方法,来表示刚才说过的事情

首先有一个初始的游戏画面,这个初始的游戏画面,被作为你的 Actor 的输入



你的 Actor 那就输出了一个 Action,比如说向右,输入的游戏画面呢,我们叫它  $s_1$ ,然后输出的 Action 呢,就叫它  $a_1$

那现在会得到一个 Reward,这边因为向右没有做任何事情,没有杀死任何的外星人,所以得到的 Reward 可能就是 0 分

採取向右以后,会看到新的游戏画面,这个叫做  $s_2$ ,根据新的游戏画面  $s_2$ ,你的 Actor 会採取新的行为,比如说开火,这边用  $a_2$ ,来表示看到游戏画面  $s_2$  的时候,所採取的行为

那假设开火恰好杀死一隻外星人,和你的 Actor 就得到 Reward,这个 Reward 的分数呢,是 5 分,然后採取开火这个行为以后

接下来你会看到新的游戏画面,那机器又会採取新的行为,那这个互动的过程呢,就会反覆持续下去,直到机器在採取某一个行为以后,游戏结束了,那什么时候游戏结束呢,就看你游戏结束的条件是什么嘛



举例来说,採取最后一个行为以后,比如说向右移,正好被外星人的子弹打中,那你的飞船就毁了,那游戏就结束了,或者是最后一个行为是开火,把最后一隻外星人杀掉,那游戏也就结束了,就你执行某一个行为,满足游戏结束的条件以后,游戏就结束了

那从游戏开始到结束的这整个过程啊,被称之为一个 Episode,那在整个游戏的过程中,机器会採取非常多的行为,每一个行为都可能得到 Reward,把所有的 Reward 通通集合起来,我们就得到一个东西,叫做整场游戏的 Total Reward,

那这个 Total Reward 呢,就是从游戏一开始得到的  $r_1$ ,一直得,一直加,累加到游戏最后结束的时候,得到的  $r_T$ ,假设这个游戏裡面会互动,  $T$  次,那麽就得到一个 Total Reward,我们这边用  $R$ ,来表示 Total Reward,其实这个 Total Reward 又有另外一个名字啊,叫做 Return 啦,你在这个 RL 的文献上,常常会同时看到 Reward 跟 Return,这两个词会出现,那 Reward 跟 Return 其实有点不一样,Reward 指的是你採取某一个行为的时候,立即得到的好处,这个是 Reward,把整场游戏裡面所有的 Reward 通通加起来,这个叫做 Return

但是我知道说,很快你就会忘记 Reward 跟 Return 的差别了,所以我们等一下就不要再用 Return 这个词彙,我们直接告诉你说,整场游戏的 Reward 的总和,就是 Total 的 Reward,而这个 Total 的 Reward 啊,就是我们想要去最大化的东西,就是我们训练的目标

$$\begin{aligned} & \text{Total reward} \\ & (\text{return}): R = \sum_{t=1}^T r_t \end{aligned}$$

↓

**What we want  
to maximize**

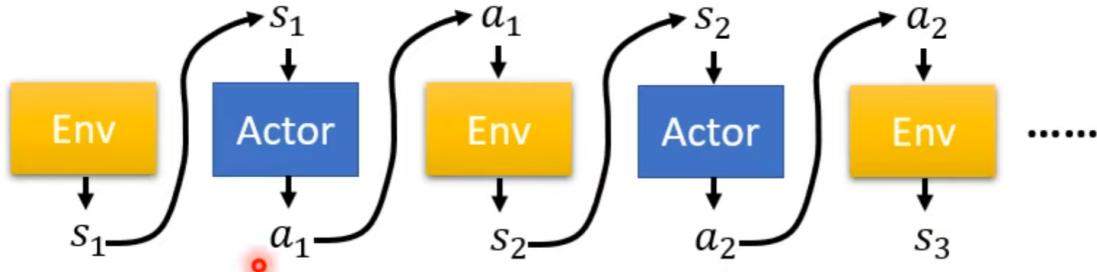
那你可能会说,欸 这个跟 Loss 不一样啊,Loss 是要越小越好啊,这个 Total Reward 是要越大越好啊,所以有点不一样吧,但是我们可以说在 RL 的这个情境下,我们把那个 Total Reward 的负号,负的 Total Reward,就当做我们的 Loss,Total Reward 是要越大越好,那负的 Total Reward,当然就是要它越小越好吧,就我们完全可以说负的 Total Reward,就是我们的 Loss,就是 RL 裡面的 Loss

## Step 3: Optimization

那我们再把这个环境跟,Agent 互动的这一件事情啊,再用不一样的图示,再显示一次

## Step 3: Optimization

Trajectory  
 $\tau = \{s_1, a_1, s_2, a_2, \dots\}$



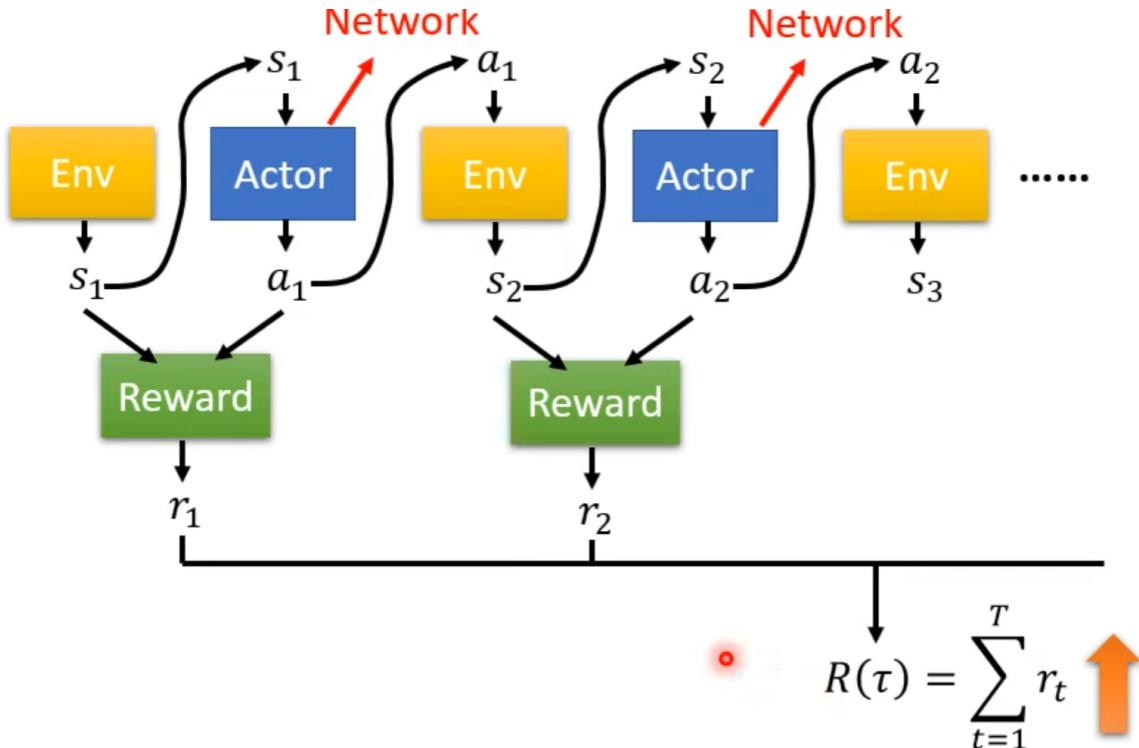
这个是你的环境,你的环境呢,输出一个 Observation,叫做  $s_1$

- 这个  $s_1$  呢,会变成你的 Actor 的输入
- 你的 Actor 呢,接下来就是输出  $a_1$
- 然后这个  $a_1$  呢,又变成环境的输入
- 你的环境呢,看到  $a_1$  以后,又输出  $s_2$

然后这个互动的过程啊,就会继续下去, $s_2$  又输入给 Actor,它就输出  $a_2$ , $a_2$  又输入给 Environment,它就输出给,它就产生  $s_3$

它这个互动呢,一直下去,直到满足游戏中止的条件,好 那这个  $s$  跟  $a$  所形成的这个 Sequence,就是  $s_1 a_1 s_2 a_2 s_3 a_3$  这个 Sequence,又叫做 Trajectory,那我们用  $\tau$  来表示 Trajectory

那根据这个互动的过程,Machine 会得到 Reward,你其实可以把 Reward 也想成是一个 Function,我们这边用一个绿色的方块来代表,这个 Reward 所构成的 Function



那这个 Reward 这个 Function,有不同的表示方法啦,在有的游戏裡面,也许你的 Reward,只需要看你採取哪一个 Action 就可以决定,不过通常我们在决定 Reward 的时候,光看 Action 是不够的,你还要看现在的 Observation 才可以,因为并不是每一次开火你都一定会得到分数,开火要正好有击到外星人,外星人正好在你前面,你开火才有分数

所以通常 Reward Function 在定义的时候,不是只看 Action,它还需要看 Observation,同时看 Action 跟 Observation,才能够知道现在有没有得到分数,所以 Reward 是一个 Function,

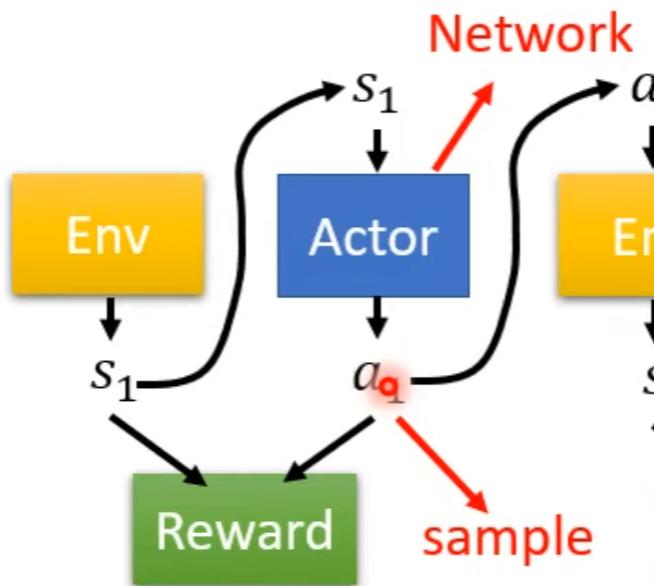
这个 Reward 的 Function,它拿  $a_1$  跟  $s_1$  当作输入,然后它产生  $r_1$  作为输出,它拿  $a_2$  跟  $s_2$  当作输入,产生  $r_2$  作为输出,把所有的 r 通通结合起来,把  $r_1$  加  $r_2$  加  $r_3$  一直加到 T,全部结合起来就得到 R,这个就是 Total Reward,也就是 Return,这个是我们要最大化要去 Maximize 的对象

这个 Optimization 的问题是这个样子,你要去找一个 Network,其实是 Network 裡面的参数,你要去 Learn 出一组参数,这一组参数放在 Actor 的裡面,它可以让这个 R 的数值越大越好,就这样,结束了,整个 Optimization 的过程就是这样,你要去找一个 Network 的参数,让这边产生的 R 越大越好

那乍看之下,如果这边的,这个 Environment Actor 跟 Reward,它们都是 Network 的话,这个问题其实也没有什么难的,这个搞不好你现在都可以解,它看起来就有点像是一个 Recurrent Network,这是一个 Recurrent Network,然后你的 Loss 就是这个样子,那只是这边是 Reward 不是 Loss,所以你是要让它越大越好,你就去 Learn 这个参数,用 Gradient Descent 你就可以让它越大越好

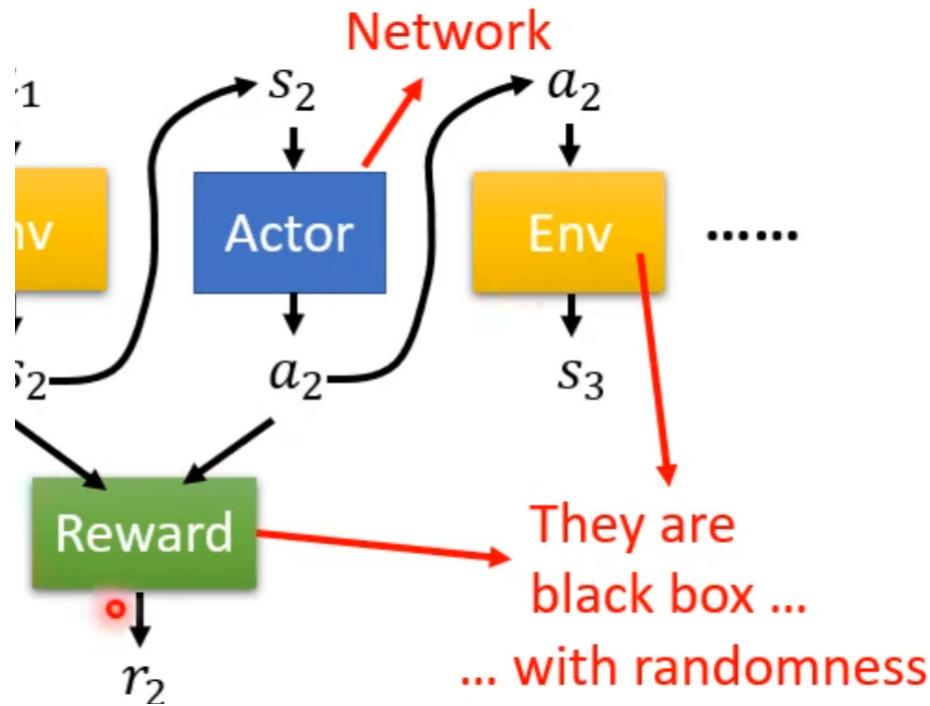
但是 RL 困难的地方是,这不是一个一般的 Optimization 的问题,因为你的 Environment,这边有很多问题导致说,它跟一般的 Network Training 不太一样

第一个问题是,你的 Actor 的输出是有随机性的



这个  $a_1$  它是用 Sample 产生的,你定同样的  $s_1$  每次产生的  $a_1$  不一定会一样,所以假设你把 Environment Actor 跟 Reward,合起来当做是一个巨大的 Network 来看待,这个 Network 可不是一般的 Network,这个 Network 裡面是有随机性的,这个 Network 裡面的某一个 Layer 是,每次产生出来结果是不一样的,这个 Network 裡面某一个 Layer 是,它的输出每次都是不一样的

另外还有一个更大的问题就是,你的 Environment 跟 Reward,它根本就不是 Network 啊,它只是一个黑盒子而已,



你根本不知道裡面发生了什么事情,Environment 就是游戏机,那这个游戏机它裡面发生什么事情你不知道,你只知道说你输入一个东西会输出一个东西,你採取一个行为它会有对应的回应,但是到底是怎麽产生这个对应的回应,我们不知道,它只是一个黑盒子,

Reward 可能比较明确,但它也不是一个 Network,它就是一条规则嘛,它就是一个规则说,看到这样子的 Optimization 跟这样的 Action,会得到多少的分数,它就只是一个规则而已,所以它也不是 Network

而且更麻烦的地方是,往往 Reward 跟 Environment,它也是有随机性的,如果是在电玩裡面,通常 Reward 可能比较不会有随机性,因为规则是定好的,对有一些 RL 的问题裡面,Reward 是有可能有随机性的

但是在 Environment 裡面,就算是在电玩的这个应用中,它也是有随机性的,你给定同样的行为,到底游戏机会怎麽样回应,它裡面可能也是有乱数的,它可能每次的回应也都是不一样,如果是下围棋,你落同一个子,你落在,你落子在同一个位置,你的对手会怎麽样回应,每次可能也是不一样

所以环境很有可能也是有随机性的,所以这不是一个一般的 Optimization 的问题,你可能不能够用我们这门课已经学过的,训练 Network 的方法来找出这个 Actor,来最大化 Reward

**所以 RL 真正的难点就是,我们怎麽解这一个 Optimization 的问题,怎麽找到一组 Network 参数,可以让 R 越大越好**

其实你再仔细想一想啊,这整个问题跟 GAN 其实有异曲同工之妙,它们有一样的地方,也有不一样的地方

- 先说它们一样的地方在哪裡,你记不记得在训练 GAN 的时候,在训练 Generator 的时候,你会把 Generator 跟 Discriminator 接在一起,然后你希望去调整 Generator 的参数,让 Discriminator 的输出越大越好,今天在 RL 裡面,我们也可以把这个 Actor 就像是 Generator,Environment 跟 Reward 就像是 Discriminator,我们要去调整 Generator 的参数,让 Discriminator 的输出越大越好,所以它跟 GAN 有异曲同工之妙
- 但什么地方不一样呢,在 GAN 裡面你的 Discriminator,也是一个 Neural Network,你了解 Discriminator 裡面的每一件事情,它也是一个 Network,你可以用 Gradient Descent,来 train 你的 Generator,让 Discriminator 得到最大的输出,但是在 RL 的问题裡,你的 Reward 跟 Environment,你可以把它们当 Discriminator 来看,但它们不是 Network,它们是一个黑盒子,所以你没有办法用一般 Gradient Descent 的方法来调整你的参数,来得到最大的输出,所以这是 RL,跟一般 Machine Learning 不一样的地方

但是我们还是可以把 RL 就看成三个阶段,只是在 Optimization 的时候,在你怎麽 Minimize Loss,也就怎麽 Maximize Reward 的时候,跟之前我们学到的方法是不太一样的

# Policy Gradient

接下来啊,我们就要讲一个拿来解 RL,拿来做 Optimization 那一段常用的一个演算法,叫做 Policy Gradient

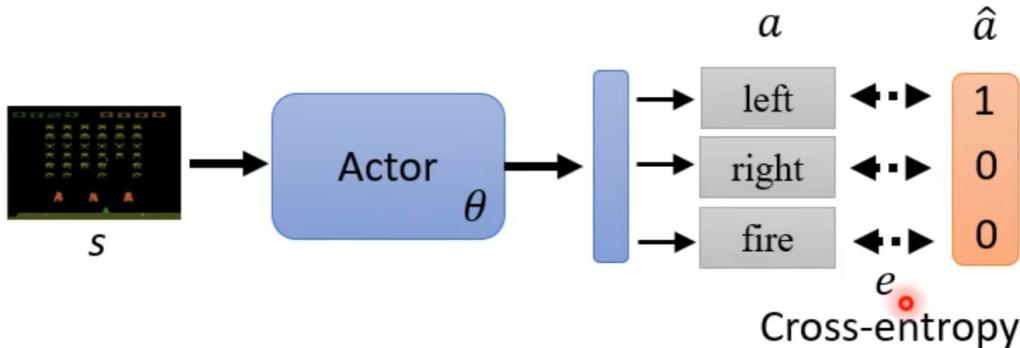
那如果你真的想知道,Policy Gradient 是哪裡来的,你可以参见过去上课的[录影](#),对 Policy Gradient 有比较详细的推导,那今天我们是从另外一个角度,来讲 Policy Gradient 这件事情

## How to control your actor

那在讲 Policy Gradient 之前,我们先来想想看,我们要怎麽操控一个 Actor 的输出,我们要怎麽让一个 Actor,在看到某一个特定的 Observation 的时候,採取某一个特定的行为呢,我们怎麽让一个 Actor,它的输入是  $s$  的时候,它就要输出 Action  $\hat{a}$  呢

那你其实完全可以把它想成一个分类的问题,也就是说假设你要让 Actor 输入  $s$ ,输出就是  $\hat{a}$ ,假设  $\hat{a}$  就是向左好了,假设你要让,假设你已经知道,假设你就是要教你的 Actor 说,看到这个游戏画面向左就是对的,你就是给我向左,那你要怎麽让你的 Actor 学到这件事呢

- Make it take (or don't take) a specific action  $\hat{a}$  given specific observation  $s$ .



那也就说  $s$  是 Actor 的输入,  $\hat{a}$  就是我们的 Label,就是我们的 Ground Truth,就是我们的正确答案,而接下来呢,你就可以计算你的 Actor,它的输出跟 Ground Truth 之间的 Cross-entropy,那接下来你就可以定义一个 Loss

Take action  $\hat{a}$

$$L = e$$

Cross-entropy

$$\theta^* = \arg \min_{\theta} L$$

假设你希望你的 Actor,它採取  $\hat{a}$  这个行为的话,你就定一个 Loss,这个 Loss 等于 Cross-entropy

然后呢,你再去 Learn 一个  $\theta$ ,你再去 Learn 一个  $\theta$ ,然后这个  $\theta$  可以让 Loss 最小,那你就可以让这个 Actor 的输出,跟你的 Ground Truth 越接近越好

你就可以让你的 Actor 学到说,看到这个游戏画面的时候,它就是要向左,这个是要让你的 Actor,採取某一个行为的时候的做法

但是假设你想要让你的 Actor,不要採取某一个行为的话,那要怎麽做呢,假设你希望做到的事情是,你的 Actor 看到某一个 Observation  $s$  的时候,我就千万不要向左的话怎麽做呢,其实很容易,你只需要把 Loss 的定义反过来就好

Take action  $\hat{a}$

$$L = e$$

Don't take action  $\hat{a}$

$$L = -e$$

Cross-entropy

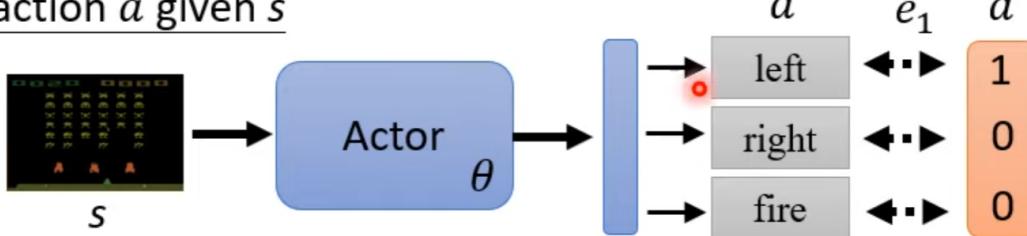
$$\theta^* = \arg \min_{\theta} L$$

你希望你的 Actor 採取  $\hat{a}$  这个行为,你就定义你的大  $L$  等于 Cross-entropy,然后你要 Minimize Cross-entropy,假设你要让你的 Actor,不要採取  $\hat{a}$  这个行为的话,那你就把你就定一个 Loss,叫做负的 Cross-entropy,Cross-entropy 乘一个负号,那你去 Minimize 这个  $L$ ,你去 Minimize 这个  $L$ ,就是让 Cross-entropy 越大越好,那也就是让  $a$  跟  $\hat{a}$  的距离越远越好,那你就可以避免你的 Actor 在看到  $s$  的时候,去採取  $\hat{a}$  这个行为,所以我们有办法控制我们的 Actor,做我们想要做的事,只要我们给它适当的 Label 跟适当的 Loss,

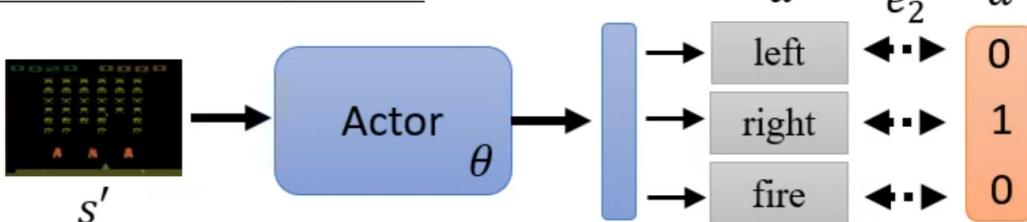
所以假设我们要让我们的 Actor,看到  $s$  的时候採取  $\hat{a}$ ,看到  $s'$  的时候不要採取  $\hat{a}'$  的话,要怎麽做呢

这个时候你就会说,Given  $s$  这个 Observation,我们的 Ground Truth 叫做  $\hat{a}$ ,Given  $s'$  这个 Observation 的时候,我们有个 Ground Truth 叫做  $\hat{a}'$ ,那对这两个 Ground Truth,我们都可以去计算 Cross-entropy,  $e_1$  跟  $e_2$

Take action  $\hat{a}$  given  $s$



Don't take action  $\hat{a}'$  given  $s'$



$$L = e_1 - e_2$$

然后接下来呢,我们就定义说我们的 Loss,就是  $e_1$  减  $e_2$ ,也就是说我们要让这个 Case,它的 Cross-entropy 越小越好,这个 Case 它的 Cross-entropy 越大越好

然后呢,我们去找一个  $\theta$  去 Minimize Loss,得到  $\theta^*$ ,那就是一个可以在  $s$ ,可以在看到  $s$  的时候採取  $\hat{a}$ ,看到  $s'$  的时候採取  $\hat{a}'$  的 Actor,所以藉由很像是在,Train 一个 Classifier 的这种行为,藉由很像是现在 Train 一个 Classifier,的这种 Data,我们可以去控制一个 Actor 的行为,

$$\theta^* = \arg \min_{\theta} L$$

有一个同学问了一个非常好的问题

- Q: 就是如果以 Alien 的游戏来说的话, 因为只有射中 Alien 才会有 Reward, 这样 Model 不是就会一直倾向于射击吗

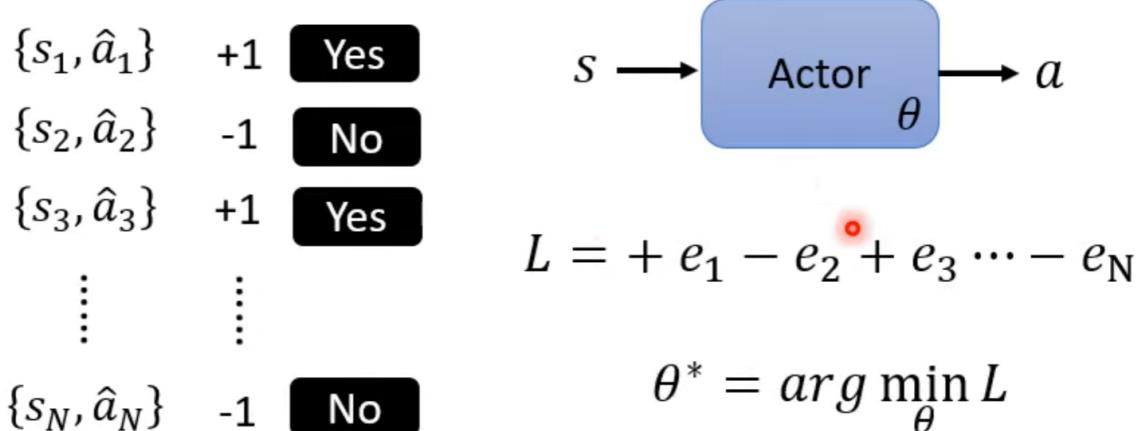
A: 对这个问题我们等一下会来解决它, 之后的投影片就会来解决它

然后又有另外一个同学, 问了一个非常好的问题就是

- Q: 哇 这样不就回到 Supervised Learning 了嘛, 这个投影片上看起来, 就是在训练一个 Classifier 而已啊, 我们就是在训练 Classifier, 你只是告诉它说, 看到  $s$  的时候就要输出  $\hat{a}$ , 看到  $s'$  的时候就不要输出  $\hat{a}, \hat{a}'$ , 这不就是 Supervised Learning 吗
- A: 这就是 Supervised Learning, 这个就是跟 Supervised Learning Train 的, Image Classifier 是一模一样的, 但等下我们会看到它跟一般的 Supervised Learning 不一样在哪裡,

那所以呢, 如果我们要训练一个 Actor, 我们其实就需要收集一些训练资料, 就收集训练资料说, 我希望在  $s_1$  的时候採取  $\hat{a}_1$ , 我希望在  $s_2$  的时候不要採取  $\hat{a}_2$

### Training Data



但可能会问说, 欸 这个训练资料哪来的, 这个我们等一下再讲训练资料哪来的

所以你就收集一大堆的资料, 这个跟 Train 一个 Image 的 Classifier 很像的, 这个  $s$  你就想成是 Image, 这个  $\hat{a}$  你就想成是 Label, 只是现在有的行为是想要被採取的, 有的行为是不想要被採取的, 你就收集一堆这种资料, 你就可以去定义一个 Loss Function, 有了这个 Loss Function 以后, 你就可以去训练你的 Actor, 去 Minimize 这个 Loss Function, 就结束了, 你就可以训练一个 Actor, 期待它执行我们的行为, 期待它执行的行为是我们想要的

而你甚至还可以更进一步, 你可以说每一个行为并不是只有好或不好, 并不是有想要执行跟不想要执行而已, 它是有程度的差别的, 有执行的非常好的, 有 Nice to have 的, 有有点不好的, 有非常差的

所以刚才啊, 我们是说每一个行为就是要执行 不要执行, 这是一个 Binary 的问题, 这是我们用  $\pm 1$  来表示

但是现在啊, 我们改成每一个  $s$  跟  $a$  的 Pair, 它有对应的一个分数, 这个分数代表说, 我们多希望机器在看到  $s_1$  的时候, 执行  $\hat{a}_1$  这个行为

## Training Data

$\{s_1, \hat{a}_1\}$	A <sub>1</sub>	+1.5
$\{s_2, \hat{a}_2\}$	A <sub>2</sub>	-0.5
$\{s_3, \hat{a}_3\}$	A <sub>3</sub>	+0.5
⋮	⋮	
$\{s_N, \hat{a}_N\}$	A <sub>N</sub>	-10



$$L = \sum A_n e_n$$

那比如说这边第一笔资料跟第三笔资料,我们分别是定 +1.5 跟 +0.5,就代表说我们期待机器看到  $s_1$  的时候,它可以做  $\hat{a}_1$ ,看到  $s_3$  的时候它可以做  $\hat{a}_3$ ,但是我们期待它看到  $s_1$  的时候,做  $\hat{a}_1$  的这个期待更强烈一点,比看到  $s_3$  做  $\hat{a}_3$  的期待更强烈一点

那我们希望它在看到  $s_2$  的时候,不要做  $\hat{a}_2$ ,我们期待它看到  $s_N$  的时候,不要做  $\hat{a}_N$ ,而且我们非常不希望,它在看到  $s_N$  的时候做  $\hat{a}_N$

有了这些资讯,你一样可以定义一个 Loss Function,你只是在你的原来的 Cross-entropy 前面,本来是 Cross-entropy 前面,要嘛是 +1 要嘛是 -1

现在改成乘上  $A_n$  这一项,改成乘上  $A_n$  这一项,告诉它说有一些行为,我们非常期待 Actor 去执行,有一些行为为我们非常不期待 Actor 去执行,有一些行为如果执行是比较好的,有一些行为希望儘量不要执行比较好,但就算执行了也许伤害也没有那麽大

所以我们透过这个  $A_n$  来控制说,每一个行为我们多希望 Actor 去执行,然后接下来有这个 Loss 以后,一样 Train 一个  $\theta$ ,Train 下去你就找一个  $\theta^*$ ,你就有个 Actor 它的行为是符合我们期待的

那接下来的难点就是,要怎麽定出这一个  $a$  呢,这个就是我们接下来的难点,就是我们接下来要面对的问题,我们还有另外一个要面对的问题是,怎麽产生这个  $s$  跟  $a$  的 Pair 呢,怎麽知道在  $s_1$  的时候要执行  $a_1$ ,或在  $s_2$  的时候不要执行  $a_2$  呢,那这个也是等一下我们要处理的问题,

## Training Data

$\{s_1, \hat{a}_1\}$	A <sub>1</sub>	+1.5
$\{s_2, \hat{a}_2\}$	A <sub>2</sub>	-0.5
$\{s_3, \hat{a}_3\}$	A <sub>3</sub>	+0.5
⋮	⋮	
$\{s_N, \hat{a}_N\}$	A <sub>N</sub>	-10
?	?	

讲到目前为止,你可能觉得跟 Supervised Learning 没有什么不同,那确实就是没有什么不同,接下来真正的重点是,在我们怎么定义 a 上面

## Version 0

那先讲一个最简单的,但是其实是不正确的版本,那这个其实也是助教的 Sample Code 的版本,那这个不正确的版本是怎么做的呢

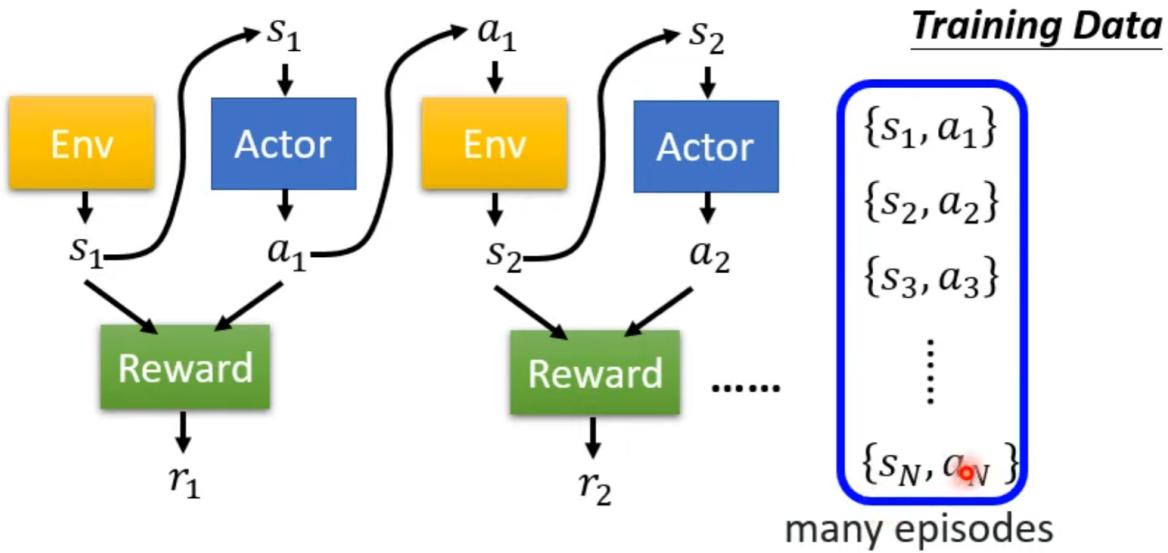
首先我们还是需要收集一些训练资料,就是需要收集 s 跟 a 的 Pair

怎么收集这个 s 跟 a 的 Pair 呢?

你需要先有一个 Actor,这个 Actor 去跟环境做互动,它就可以收集到 s 跟 a 的 Pair

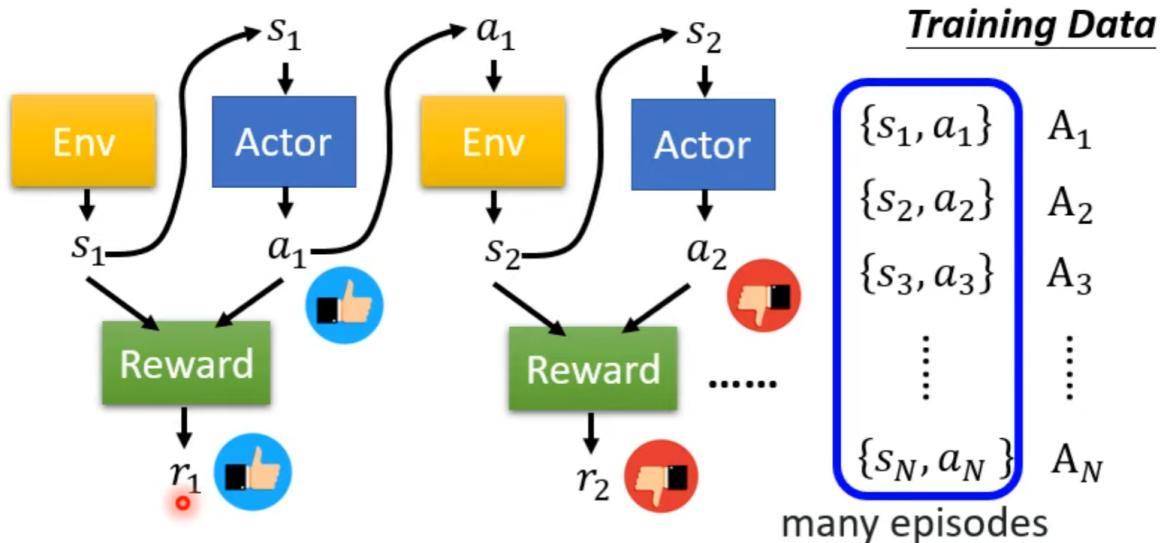
那这个 Actor 是哪裡来的呢,你可能觉得很奇怪,我们今天的目标,不就是要训练一个 Actor 吗,那你又说你需要拿一个 Actor,去跟环境做互动,然后把这个 Actor,它的 s 跟 a 记录下来,那这个 Actor 是哪裡来的呢?

你先把这个 Actor,想成就是一个随机的 Actor 好了,就它是一个,它就是一个随机的东西,那看到  $s_1$ ,然后它执行的行为就是乱七八糟的,就是随机的,但是我们会把它在,每一个 s 执行的行为 a,通通都记录下来,好,那通常我们在这个收集资料的话,你不会只把 Actor 跟环境做一个 Episode,通常会做多个 Episode,然后期待你可以收集到足够的资料,比如说在助教 Sample Code 裡面,可能就是跑了 5 个 Episode,然后才收集到足够的资料



所以我们就是去观察,某一个 Actor 它跟环境互动的状况,那把这个 Actor,它在每一个 Observation,执行的 Action 都记录下来,然后接下来,我们就去评价每一个 Action,它到底是好还是不好,评价完以后,我们就可以拿我们评价的结果,来训练我们的 Actor

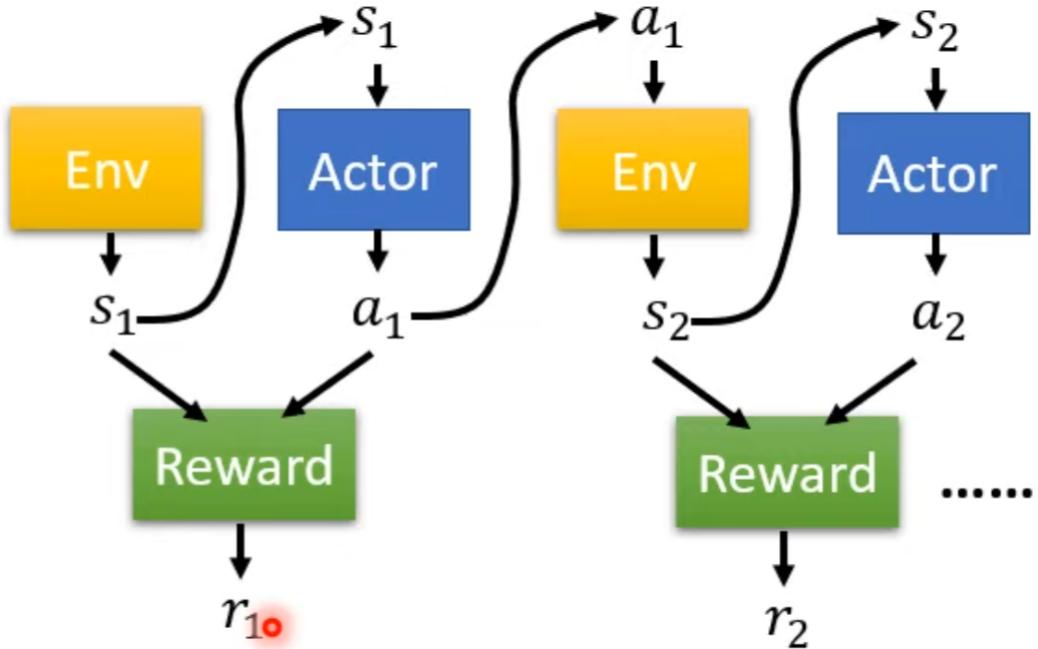
那怎么评价呢,我们刚才有说,我们会用 A 这一个东西,来评价在每一个 Step,我们希不希望我们的 Actor,採取某一个行为,那最简单的评价方式是,假设在某一个 Step  $s_1$ ,我们执行了  $a_1$ ,然后得到 Reward  $r_1$



- 那 Reward 如果如果是正的,那也许就代表这个 Action 是好的
- 那如果 Reward 是负的,那也许就代表这个 Action 是不好的

那我们就把这个 Reward  $r_1, r_2$ ,当做是  $a$ ,  $A_1$  就是  $r_1$ ,  $A_2$  就是  $r_2$ ,  $A_3$  就是  $r_3$ ,  $A_N$  就是  $r_N$ ,那这样等同于你就告诉 machine 说,如果我们执行完某一个 Action,  $a_1$  那得到的 Reward 是正的,那这就是一个好的 Action,你以后看到  $s_1$  就要执行  $a_1$ ,如果今天在  $s_2$  执行  $a_2$ ,得到 Reward 是负的,就代表  $a_2$  是不好的  $a_2$ ,就代表所以以后看到  $s_2$  的时候,就不要执行  $a_2$

那这个,那这个 Version 0,它并不是一个好的版本,为什么它不是一个好的版本呢,因为你用这一个方法,你把  $a_1$  设为  $r_1$ ,  $A_2$  设为  $r_2$ ,这个方法认出来的 Network,它是一个短视近利的 Actor,它就是一个只知道会一时爽的 Actor,它完全没有长程规划的概念



- 我们知道说每一个行为,其实都会影响互动接下来的发展,也就是说 Actor 在  $s_1$  执行  $a_1$  得到  $r_1$ ,这个并不是互动的全部,因为  $a_1$  影响了我们接下来会看到  $s_2, s_3$  会影响到接下来会执行  $a_2, a_3$ ,也影响到接下来会产生  $r_2, r_3$ ,所以  $a_1$  也会影响到,我们会不会得到  $r_2, r_3$ ,所以每一个行为并不是独立的,每一个行为都会影响到接下来发生的事情,
- 而且我们今天在跟环境做互动的时候,有一个问题叫做, Reward Delay, 就是有时候你需要牺牲短期的利益,以换取更长程的目标,如果在下围棋的时候,如果你有看天龙八部的时候你就知道说,这个虚竹在破解珍珑棋局的时候,堵死自己一块子,让自己被杀了很多子以后,最后反而赢了整局棋  
那如果是在这个,Space Invaders 的游戏裡面,你可能需要先左右移动一下进行瞄准,然后射击才会得到分数,而左右移动这件事情是,没有任何 Reward 的,左右移动这件事情得到的 Reward 是零,只有射击才会得到 Reward,但是并不代表左右移动是不重要的,我们会先需要左右移动进行瞄准,那我们的射击才会有效果,所以有时候我们会,需要牺牲一些近期的 Reward,而换取更长程的 Reward
- 所以今天假设我们用 Version 0,会发生说今天 Machine,只要是採取向左跟向右,它得到的 Reward 会是 0,如果它採取开火,它得到的 Reward 就会,只有开火的时候,它得到的 Reward 才会是正的,才会是正的,那这样 Machine 就会学到,它只有疯狂狂开火才是对的,因为只有开火这件事才会得到 Reward,其它行为都不会得到 Reward,所以其它行为都是不被鼓励的,只有开火这件事是被鼓励的,那个 Version 0 就只会学到疯狂开火而已,那 Version 0 是助教的范例程式,那这个当然也是可以执行的,那只是它的结果不会太好而已,那助教范例十,程式之所以是 Version 0 是因为,我不知道为什么这个 Version 0,好像是大家,如果你自己在 Implement rl 的时候,你特别容易犯的错误,你特别容易拿自己 Implement 的时候,就直接使用 Version 0,但是得到一个很差的结果

所以接下来怎么办呢,我们开始正式进入 rl 的领域,真正来看 Policy Gradient 是怎么做的,所以我们需要有 Version 1

## Version 1

在 Version 1 裡面,  $a_1$  它有多好,不是在取决于  $r_1$ ,而是取决于  $a_1$  之后所有发生的事情,我们会把  $a_1$ ,执行完  $a_1$  以后,所有得到的 Reward,  $r_1, r_2, r_3$  到  $r_N$ ,通通集合起来,通通加起来,得到一个数值叫做  $G_1$ ,然后我们会说  $a_1$  就等于  $G_1$ ,我们拿这个  $G_1$ ,来当作评估一个 Action 好坏的标准

## Training Data

$s_1$	$s_2$	$s_3$	$\dots$	$s_N$	$\{s_1, a_1\}$	$A_1$
$a_1$	$a_2$	$a_3$	$\dots$	$a_N$	$\{s_2, a_2\}$	$A_2$
$r_1$	$r_2$	$r_3$	$\dots$	$r_N$	$\{s_3, a_3\}$	$A_3$
					$\vdots$	$\vdots$
$G_1 = r_1 + r_2 + r_3 + \dots + r_N$					$\{s_N, a_N\}$	$A_N$

刚才是直接拿  $r_1$  来评估,现在不是,拿  $G_1$  来评估,那接下来所有发生的  $r$  通通加起来,拿来评估  $a_1$  的好坏,因为我们执行完  $a_1$  以后,就发生这么一连串的事情,那这么一连串的事情加起来,也许就可以评估  $a_1$ ,到底是不是一个好的 Action

所以以此类推,  $a_2$  它有多好呢,就把执行完  $a_2$  以后,所有的  $r, r_2$  到  $r_N$ ,通通加起来得到  $G_2$ ,然后那  $a_3$  它有多好呢,就把执行完  $a_3$  以后,所有的  $r$  通通加起来,就得到  $G_3$ ,所以把这些东西通通都加起来,就把那这些这个  $G$ ,叫做 Cumulated Reward,叫做累积的 Reward,把未来所有的 Reward 加起来,来评估一个 Action 的好坏,那像这样子的方法听起来就合理多了

$s_1$	$s_2$	$s_3$	$\dots$	$s_N$
$a_1$	$a_2$	$a_3$	$\dots$	$a_N$
$r_1$	$r_2$	$r_3$	$\dots$	$r_N$

$$G_1 = r_1 + r_2 + r_3 + \dots + r_N$$

$$G_2 = r_2 + r_3 + \dots + r_N$$

$$G_3 = r_3 + \dots + r_N$$

## cumulated reward

$G_t$  是什么呢,就是从  $t$  这个时间点开始,我们把  $r_t$  一直加到  $r_N$ ,全部合起来就是, Cumulated 的 Reward  $G_t$ ,那当我们用, Cumulated 的 Reward 以后,我们就可以解决 Version 0 遇到的问题,因为你可能向右移动以后进行瞄准,接下来开火,就有打中外星人,那这样向右这件事情,它也有 Accumulate Reward,虽然向右这件事情没有立即的 Reward,假设  $a_1$  是向右,那  $r_1$  可能是 0,但接下来可能会因为向右这件事,导致有瞄

准,导致有打到外星人,那 Cumulated 的 Reward 就会正的,那我们就会知道说,其实向右也是一个好的 Action,这个是 Version 1

但是你仔细想一想会发现,Version 1 好像也有点问题

假设这个游戏非常地长,你把  $rN$  归功于  $a1$  好像不太合适吧,就是当我们采取  $a1$  这个行为的时候,立即有影响的是  $r1$ ,接下来有影响到  $r2$ ,接下来影响到  $r3$ ,那假设这个过程非常非常地长的话,那我们说因为有做  $a1$ ,导致我们可以得到  $rN$ ,这个可能性应该很低吧,也许得到  $rN$  的功劳,不应该归功于  $a1$ ,好 所以怎么办呢

## Version 2

有第二个版本的 Cumulated 的 Reward,我们这边用  $G'$ ,来表示 Cumulated 的 Reward,好 这个我们会在  $r$  前面,乘一个 Discount 的 Factor

$$G'_1 = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$$

Discount factor  $\gamma < 1$

这个 Discount 的 Factor  $\gamma$ ,也会设一个小于 1 的值,有可能会设,比如说 0.9 或 0.99 之类的,所以这个  $G'1$  相较于  $G1$  有什么不同呢, $G1$  是  $r1$  加  $r2$  加  $r3$ ,那  $G'1$  呢,是  $r1$  加  $\gamma r2$  加  $\gamma^2 r3$ ,就是距离采取这个 Action 越远,我们  $\gamma$  平方项就越多,所以  $r2$  距离  $a1$  一步,就乘个  $\gamma$ , $r3$  距离  $a1$  两步,就乘  $\gamma$  平方,那这样一直加到  $rN$  的时候, $rN$  对  $G'1$  就几乎没有影响力了,因为你  $\gamma$  乘了非常非常多次了, $\gamma$  是一个小于 1 的值,就算你设 0.9,0.9 的比如说 10 次方,那其实也很小了

所以你今天用这个方法,就可以把离  $a1$  比较近的那些 Reward,给它比较大的权重,离我比较远的那些 Reward,给它比较小的权重,所以我们现在有一个新的 A,这个新的 A 这个评估,这个 Action 好坏的这个 A,我们现在用  $G'1$  来表示它,那它的式子可以写成这个样子,这个  $G't$  就是 Summation over,  $n$  等于  $t$  到  $N$ ,然后我们把  $rN$  乘上  $\gamma$  的  $n-t$  次方,所以离我们现在,采取的 Action 越远的那些 Reward,它的  $\gamma$  就被乘越多次,它对我们的  $G'$  的影响就越小,这是第二个版本,听到这边你是不是觉得合理多了呢

$$G'_t = \sum_{n=t}^N \gamma^{n-t} r_n$$

## Q&A

Q1:一个大括号是一个 Episode,还是这样蓝色的框住的多个大括号,是一个 Episode

A1: 一个大括号不是一个 Episode,一个大括号是,我们在这一个 Observation,执行这一个 Action 的时候,这个是一笔资料,它不是一个 Episode,Episode 是很多的,很多次的 Observation,跟很多次的 Action 才合起来,才是一个 Episode

Q2: G1 需不需要做标准化之类动作

A2: 这个问题太棒了,为什么呢,因为这个就是 Version 3

Q3: 越早的动作就会累积到越多的分数吗,越晚的动作累积的分数就越少

A3: 对 没错 是,在这个设计的情境裡面,是,越早的动作就会累积到越多的分数,但是这个其实也是一个合理的情境,因为你想看,比较早的那些动作对接下来的影响比较大,到游戏的终局,没什么外星人可以杀了,你可能做什么事对结果影响都不大,所以比较早的那些 Observation,它们的动作是我们可能需要特别在意的,不过像这种 A 要怎么决定,有很多种不同的方法,如果你不想要比较早的动作 Action 比较大,你完全可以改变这个 A 的定义,事实上不同的  $r_t$  的方法,其实就是在 A 上面下文章,有不同的定义 A 的方法

Q4: 看来仍然不适合用在围棋之类的游戏,毕竟围棋这种游戏只有结尾才有分数

A4: 这是一个好问题,这个我们现在讲的这些方法,到底能不能够处理,这种结尾才有分数的游戏呢,其实也是可以的,怎么说呢,假设今天只有  $r_N$  有分数,其它通通都是 0,那会发生什么事,那会说,今天我们採取一连串的 Action,只要最后赢了,这一串的 Action 都是好的,如果输了,这一连串的 Action,通通都算是不好的,而你可能会觉得这样做,感觉 Train Network 应该会很难 Train,确实很难 Train,但是就我所知,最早的那个版本的 AlphaGo,它是这样 Train 的,很神奇,它就是这样做的,它裡面有用到这样子的技术,当然还有一些其它的方法,比如说 Value Network 等等,那这个等一下也会讲到,那最早的 AlphaGo,它有採取这样子的技术来做学习,它有试著採取这样的技术,看起来是学得起来的,拿预估的误差当 Reward,拿,有一个同学说那其实 AlphaGo,可以拿预估的误差当 Reward,那你要有一个办法先预估误差,那你才拿它来当 Reward,那有没有办法事先预估,我们接下来会得到多少的 Reward 呢,有那这个在之后的版本裡面,会有这样的技术,但我目前还没有讲到那一块,好那我们接下来就讲 Version 3,

## Version 3

---

Version 3 就是像刚才同学问的,要不要做标准化呢?

要,因为好或坏是相对的,好或坏是相对的,怎么说好或坏是相对的呢,假设所有,假设今天在这个游戏裡面,你每次採取一个行动的时候,最低分就预设是 10 分,那你其实得到 10 分的 Reward,根本算是差的,就好像说今天你说你得,在某一门课得到 60 分,这个叫做好或不好,还是不好呢,没有人知道

因为那要看别人得到多少分数,如果别人都是 40 分,你是全班最高分,那你很厉害,如果别人都是 80 分,你是全班最低分,那就很不厉害,所以 Reward 这个东西是相对的

所以如果我们只是单纯的把  $G$  算出来,你可能会遇到一个问题,假设这个游戏裡面,可能永远都是拿到正的分数,每一个行为都会给我们正的分数,只是有大有小的不同,那你这边  $G$  算出来通通都会是正的,有些行为其实是不好的,但是你仍然会鼓励你的 Model,去採取这些行为

**Good or bad reward is “relative”**

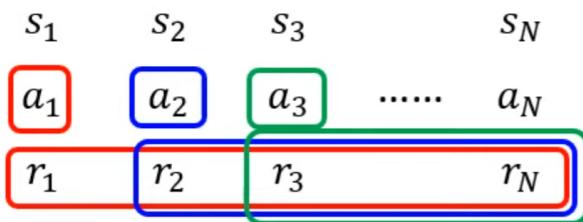
**If all the  $r_n \geq 10$**

$r_n = 10$  is negative ...

**Minus by a baseline  $b$**

所以怎么办,我们需要做一下标准化,那这边先讲一个最简单的方法就是,把所有的  $G'$  都减掉一个  $b$ ,这个  $b$  在这边叫做,在  $r_t$  的文献上通常叫做 Baseline,那这个跟我们作业的 Baseline 有点不像,但是反正在  $r_t$  的文献上,就叫做 Baseline 就对了,我们把所有的  $G'$  都减掉一个  $b$ ,目标就是让  $G'$  有正有负,特别高的  $G'$  让它是正的,特别低的  $G'$  让它是负的

## Training Data



$$\begin{array}{ll} \{s_1, a_1\} & A_1 = G'_1 - b \\ \{s_2, a_2\} & A_2 = G'_2 - b \\ \{s_3, a_3\} & A_3 = G'_3 - b \\ \vdots & \vdots \\ \{s_N, a_N\} & A_N = G'_N - b \end{array}$$

Good or bad reward is “relative”

If all the  $r_n \geq 10$

$r_n = 10$  is negative ...

Minus by a baseline  $b$

Make  $G'_t$  have positive and negative values

$$G'_t = \sum_{n=t}^N \gamma^{n-t} r_n$$

但是这边会有一个问题就是,那要怎么样设定这个 Baseline 呢,我们怎么设定一个好的 Baseline,让  $G'$  有正有负呢,那这个我们在接下来的版本裡面还会再提到,但目前为止我们先讲到这个地方

Q: 需要个比较好的,Heuristic Function

A: 对 需要个,就是说在下围棋的时候,假设今天你的 Reward 非常地 Sparse,那你可能会需要一个好 的,Heuristic Function,如果你有看过那个最原始的,那个深蓝的那篇 Paper,就是在这个机器下围棋打爆人类之前,其实已经在西洋棋上打爆人类了,那个就叫深蓝,深蓝就有蛮多 Heuristic 的 Function,它就不是只有下到游戏的中盘,才知道 才得到 Reward,中间会有蛮多的状况它都会得到 Reward,好

## Policy Gradient

接下来就会实际告诉你说,Policy Gradient 是怎么操作的,那你可以仔细读一下助教的程式,助教就是这么操作的

# Policy Gradient

- Initialize actor network parameters  $\theta^0$
- For training iteration  $i = 1$  to  $T$ 
  - Using actor  $\theta^{i-1}$  to interact
  - Obtain data  $\{s_1, a_1\}, \{s_2, a_2\}, \dots, \{s_N, a_N\}$
  - Compute  $A_1, A_2, \dots, A_N$
  - Compute loss  $L$
  - $\theta^i \leftarrow \theta^{i-1} - \eta \nabla L$

首先你要先 Random 初始化,随机初始化你的 Actor,你就给你的 Actor 一个随机初始化的参数,叫做  $\theta^0$ ,然后接下来你进入你的 Training Iteration,假设你要跑 T 个 Training Iteration,好 那你就拿你的这个,现在手上有的 Actor,一开始是这个  $\theta^0$ ,一开始很笨 它什么都不会,它採取的行为都是随机的,但它会越来越好,你拿你的 Actor 去跟环境做互动,那你就得到一大堆的 s 跟 a,你就得到一大堆的 s 跟 a,就把它互动的过程记录下来,得到这些 s 跟 a,那接下来你就要进行评价,你用  $A_1$  到  $A_N$  来决定说,这些 Action 到底是好还是不好

你先拿你的 Actor 去跟环境做互动,收集到这些观察,接下来你要进行评价,看这些 Action 是好的还是不好的,那你真正需要这个在意的地方,你最需要把它改动的地方 就是在评价这个过程裡面,那助教程式这个 A 就直接设成,Immediate Reward,那你写的要改这一段,你才有可能得到好的结果

说完这个 A 以后,就结束了

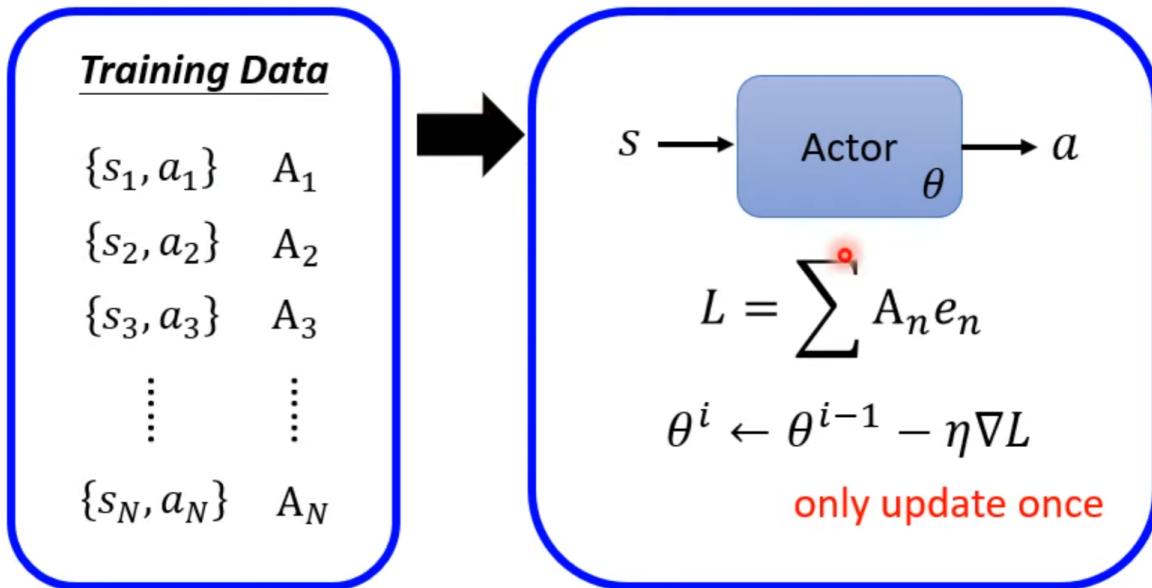
你就把 Loss 定义出来,然后 Update 你的 Model,这个 Update 的过程,就跟 Gradient Descent 一模一样的,会去计算 L 的 Gradient,前面乘上 Learning Rate,然后拿这个 Gradient 去 Update 你的 Model,就把  $\theta_{i-1}$  Update 成  $\theta_i$ ,

但是这边有一个神奇的地方是一般的 training,在我们到目前为止的 Training,Data Collection 都是在 For 循环之外,比如说我有一堆资料,然后把这堆资料拿来做 Training,拿来 Update,Model 很多次,然后最后得到一个收敛的参数,然后拿这个参数来做 Testing

但在 RL 裡面不是这样,你发现收集资料这一段,居然是在 For 循环裡面,假设这个 For 循环,你打算跑 400 次,那你就得收集资料 400 次,或者是我们用一个图像化的方式来表示

- Using actor  $\theta^{i-1}$  to interact
- Obtain data  $\{s_1, a_1\}, \{s_2, a_2\}, \dots, \{s_N, a_N\}$
- Compute  $A_1, A_2, \dots, A_N$

这个是你收集到的资料,就是你观察了某一个 Actor,它在每一个 State 执行的 Action,然后接下来你给予一个评价,但要用什么评价要用哪一个版本,这个是你自己决定的,你给予一个评价,说每一个 Action 是好或不好,你有了这些资料 这些评价以后,拿去训练你的 Actor,你拿这些评价可以定义出一个 Loss,然后你可以更新你的参数一次



但是有趣的地方是,你只能更新一次而已,一旦更新完一次参数以后,接下来你就要重新去收集资料了,登记一次参数以后,你就要重新收集资料,才能更新下一次参数,所以这就是为什么 RL 往往它的训练过程非常花时间

收集资料这件事情,居然是在 For 循环裡面的,你每次 Update 完一次参数以后,你的资料就要重新再收集一次,再去 Update 参数,然后 Update 完一次以后,又要再重新收集资料,如果你参数要 Update 400 次,那你资料就要收集 400 次,那这个过程显然非常地花时间,那你接下来就会问说,那为什么会这样呢

**为什么我们不能够一组资料,就拿来 Update 模型 Update 400 次,然后就结束了呢,为什么每次 Update 完我们的模型参数以后,Update Network 参数以后,就要重新再收集资料呢**

那我们,那这边一个比较简单的比喻是,你知道一个人的食物,可能是另外一个人的毒药

- Initialize actor network parameters  $\theta^0$
  - For training iteration  $i = 1$  to  $T$ 
    - Using actor  $\theta^{i-1}$  to interact
    - Obtain data  $\{s_1, a_1\}, \{s_2, a_2\}, \dots, \{s_N, a_N\}$
    - Compute  $A_1, A_2, \dots, A_N$
    - Compute loss  $L$
    - $\theta^i \leftarrow \theta^{i-1} - \eta \nabla L$
- May not be good for  $\theta^i$**



One man's meat is another man's poison.

这些资料是由  $\theta_{i-1}$  所收集出来的,这是  $\theta_{i-1}$  跟环境互动的结果,这个是  $\theta_{i-1}$  的经验,这些经验可以拿来更新  $\theta_{i-1}$ ,可以拿来 Update  $\theta_{i-1}$  的参数,但它不一定适合拿来 Update  $\theta_i$  的参数

或者是我们举一个具体的例子,这个例子来自棋魂的第八集,大家看过棋魂吧,我应该就不需要解释棋魂的剧情了吧

这个是进藤光,然后他在跟佐为下棋,然后进藤光就下一步,在大马 现在在小马步飞,这小马步飞具体是什么,我其实也没有非常地确定,但这边有解释一下,就是棋子斜放一格叫做小马步飞,斜放好几格叫做大马步飞,好 阿光下完棋以后,佐为就说这个时候不要下小马步飞,而是要下大马步飞,然后阿光说为什么要下大马步飞呢,我觉得小马步飞也不错

## 棋魂第八集



这个时候佐为就解释了,如果大马步飞有 100 手的话,小马步飞只有 99 手,接下来是重点,之前走小马步飞是对的,因为小马步飞的后续比较容易预测,也比较不容易出错,但是大马步飞的下法会比较复杂,但是阿光假设想要变强的话,他应该要学习下大马步飞,或者是阿光变得比较强以后,他应该要下大马步飞,所以你知道说同样的一个行为,同样是做下小马步飞这件事,对不同棋力的棋士来说,也许它的好是不一样的,对于比较弱的阿光来说,下小马步飞是对的,因为他比较不容易出错,但对于已经变强的阿光来说,应该要下大马步飞比较好,下小马步飞反而是比较不好的



所以同一个 Action 同一个行为,对于不同的 Actor 而言,它的好是不一样的

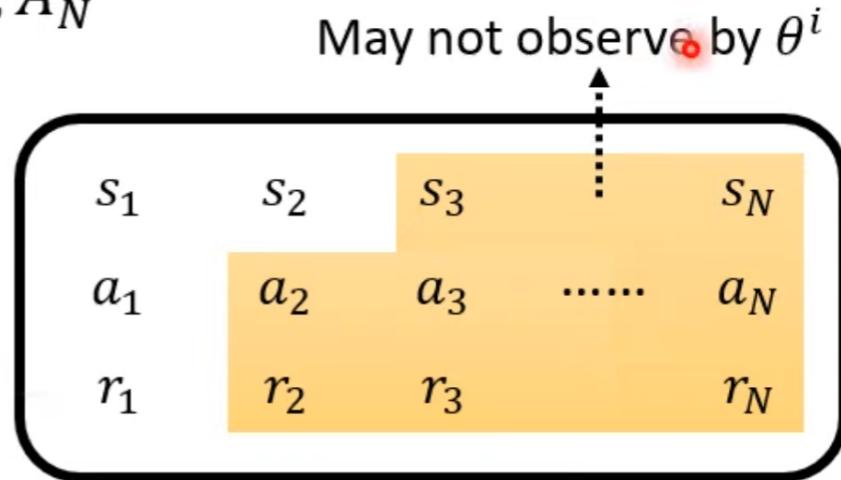
所以今天假设我们用  $\theta_{i-1}$ , 收集了一堆的资料, 这个是  $\theta_{i-1}$  的 Trajectory, 这些资料只能拿来训练  $\theta_{i-1}$ , 你不能拿这些资料来训练  $\theta_i$ , 为什么不能拿这些资料来训练  $\theta_i$  呢

$\pi_1, \pi_2, \dots, \pi_N$

the loss  $L$

$-1 - \eta \nabla L$

Trajectory of  
 $\theta^{i-1}$



因为假设假设就算是从  $\theta_{i-1}$  跟  $\theta_i$ , 它们在  $s_1$  都会採取  $a_1$  好了, 但之后到了  $s_2$  以后, 它们可能採取的行为就不一样了, 所以假设对  $\theta$ , 假设今天  $\theta_i$ , 它是看  $\theta_{i-1}$  的这个 Trajectory, 那  $\theta_{i-1}$  会执行的这个 Trajectory, 跟  $\theta_i$  它会採取的行为根本就不一样, 所以你拿著  $\theta_{i-1}$  接下来会得到的 Reward, 来评估  $\theta_i$  接下来会得到的 Reward, 其实是不合适的

所以如果再回到刚才棋魂的那个例子, 同样是假设这个  $a_1$  就是下小马步飞, 那对于变强以前的阿光, 这是一个合适的走法, 但是对于变强以后的阿光, 它可能就不是一个合适的走法

所以今天我们在收集资料, 来训练你的 Actor 的时候, 你要注意就是 **收集资料的那个 Actor, 要跟被训练的那个 Actor, 最好就是同一个, 那当你的 Actor 更新以后, 你就最好要重新去收集资料**, 这就是为什么 RL 它非常花时间的原因

## On-policy v.s. Off-policy

刚才我们说, 这个要被训练的 Actor, 跟要拿来跟环境互动的 Actor, 最好是同一个, **当我们训练的 Actor, 跟互动的 Actor 是同一个的时候, 这种叫做 On-policy Learning**, 那我们刚才示范的那个 Policy Gradient 的整个 Algorithm, 它就是 On-policy 的 Learning, 那但是还有另外一种状况叫做, **Off-policy Learning**,

- The actor to train and the actor for interacting is the same. → On-policy
- Can the actor to train and the actor for interacting be different? → Off-policy

Off-policy 的 Learning 我们今天就不会细讲, Off-policy 的 Learning, 期待能够做到的事情是, 我们能不能够让要训练的那个 Actor, 还有跟环境互动的那个 Actor, 是分开的两个 Actor 呢, 我们要训练的 Actor, 能不能够根据其他 Actor 跟环境互动的经验, 来进行学习呢

Off-policy 有一个非常显而易见的好处, 你就不用一直收集资料了, 刚才说 Reinforcement Learning, 一个很卡的地方就是, 每次更新一次参数就要收集一次资料, 你看助教的示范历程是更新 400 次参数, 400 次参数相较于你之前 Train 的 Network, 没有很多, 但我们要收集 400 次资料, 跑起来也已经是很卡了, 那如果我们可以收一次资料, 就 Update 参数很多次, 这样不是很好吗, 所以 Off-policy 它有不错的优点

## Off-policy → Proximal Policy Optimization(PPO)

但是 Off-policy 要怎么做呢,我们这边就不细讲,有一个非常经典的 Off-policy 的方法,叫做 Proximal Policy Optimization,缩写是 PPO,那这个是今天蛮常使用的一个方法,它也是一个蛮强的方法,蛮常使用的方法

Off-policy 的重点就是,你在训练的那个 Network,要知道自己跟别人之间的差距,它要有意识的知道说,它跟环境互动的那个 Actor 是不一样的,那至于细节我们就不细讲,那我有留那个上课的录影的[连结](#),在投影片的下方,等一下大家如果有兴趣的话,再自己去研究 PPO

- The **actor to train** has to know its difference from the **actor to interact**.



the actor to train



the actor to interact

<https://disp.cc/b/115-bLHe>

那如果要举个比喻的话,就好像是你去问克里斯伊凡 就是美国队长,怎么追一个女生,然后克里斯伊凡就告诉你说,他就示范给你看,他就是 Actor To Interact,他就是负责去示范的那个 Actor,他说他只要去告白,从来没有失败过,但是你要知道说,你跟克里斯伊凡其实还是不一样,人帅真好 人丑吃草,你跟克里斯伊凡是不一样的,所以克里斯伊凡可以采取的招数,你不一定能够采取,你可能要打一个折扣,那这个就是 Off-policy 的精神

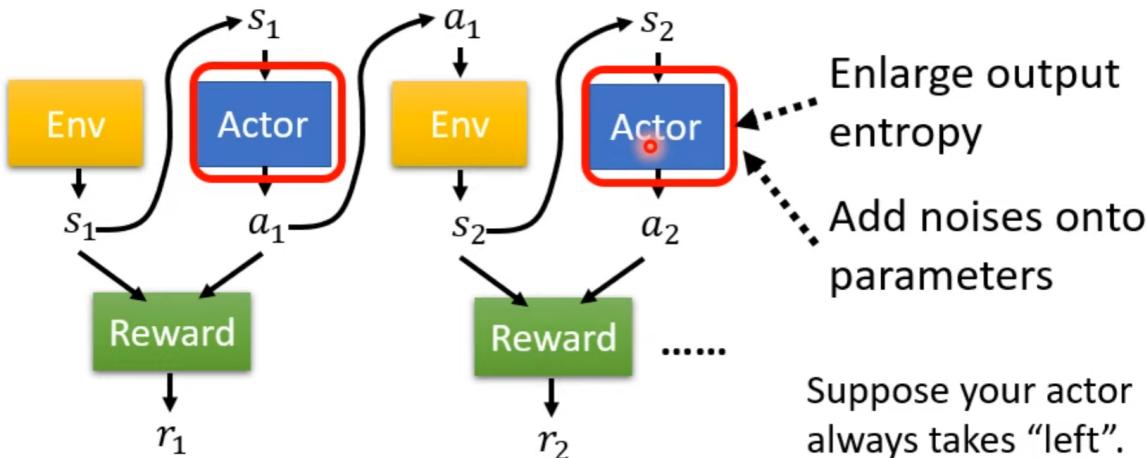
你的 Actor To Train,要知道 Actor To Interact,跟它是不一样的,所以 Actor To Interact 示范的那些经验,有些可以采纳,有些不一定可以采纳,至于细节怎么做,那过去的上课录影留在这边,给大家参考

## Collection Training Data: Exploration

那还有另外一个很重要的概念,叫做 Exploration,Exploration 指的是什么呢,我们刚才有讲过说,我们今天的,我们今天的这个 Actor,它在采取行为的时候,它是有一些随机性的

而这个随机性其实非常地重要,很多时候你随机性不够,你会 Train 不起来,为什么呢,举一个最简单的例子,假设你一开始初始的 Actor,它永远都只会向右移动,它从来都不会知道要开火,如果它从来没有采取开火这个行为,你就永远不知道开火这件事情,到底是好还是不好,唯有今天某一个 Actor,去试图做开火这件事得到 Reward,你才有办法去评估这个行为好或不好,假设有一些 Action 从来没被执行过,那你根本就无从知道,这个 Action 好或不好

# Exploration



# Actor-Critic

那上一次 RL 的部分,我们讲说我们要 Learn 一个 Actor,那这一次,我们要 Learn 另外一个东西,这个东西叫做 Critic

我会先解释 Critic 是什麼,然后我们再来讲说,这个 Critic 对 Learn Actor 这个东西,有什麼样的帮助

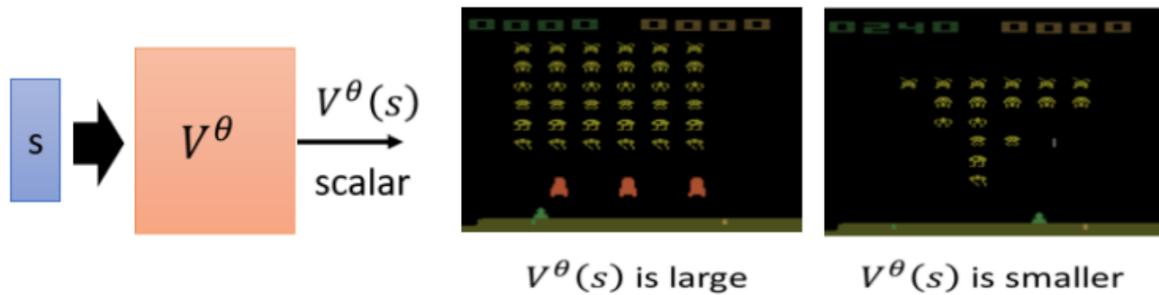
Critic 它的工作是要来评估一个 Actor 的好坏,就你现在已经有一个 Actor,它的参数叫  $\theta$ ,那 Critic 的工作就是,它要评估说如果这个 Actor,它看到某个样子的 Observation,看到某一个游戏画面,接下来它可能会得到多少的 Reward

那 Critic 有好多种不同的变形,有的 Critic 是只看游戏画面来判断,有的 Critic 是说采取某,看到某一个游戏画面,接下来又发现 Actor 採取某一个 Action,在这两者都具备的前提下,那接下来会得到多少 Reward

- Critic: Given actor  $\theta$ , how good it is when observing  $s$  (and taking action  $a$ )
- Value function  $V^\theta(s)$ : When using actor  $\theta$ , the discounted cumulated reward expects to be obtained after seeing  $s$

那这样讲,还是有点抽象,所以我们讲的更具体一点,我们直接介绍一个,我们等一下会真的被用上,你在作业裡面真的派得上用场的,这个 Critic 叫做Value Function,那这个 Value Function,我们这边用大写的  $V^\theta(S)$  来表示

它的输入是  $s$ ,也就是现在游戏的状况,比如说游戏的画面,那这边要特别注意一下  $V$ ,它是有一个上标  $\theta$  的



这个上标  $\theta$  代表这个  $V$ ,它观察的对象是  $\theta$  这个 Actor,它观察的这个 Actor 它的参数是  $\theta$ ,那这个  $V$ ,  $V^\theta$  就是一个 Function,它的输入是  $S$ ,那输出是一个 Scalar,这边用  $V^\theta(S)$  来表示这一个 Scalar

那 Scalar 这个数值的含义是,这一个 Actor  $\theta$ ,放在上标的这个 Actor  $\theta$ ,它如果看到 Observation  $S$ ,如果看到输入的这个  $S$  的游戏画面,接下来它得到的,Discounted Cumulated Reward 是多少

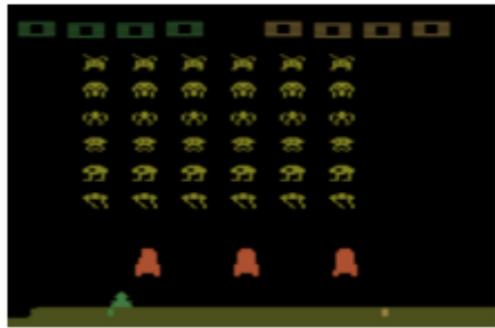
$$G'_1 = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$$

这个的 Value Function 它的工作,就是要去估测说,对某一个 Actor 来说,如果现在它已经看到某一个游戏画面,那接下来会得到的,Discounted Cumulated Reward 应该是多少

当然 Discounted Cumulated Reward,你可以直接透过把游戏玩到底,就你看到你已经有了 Actor  $\theta$ ,那假设它看到这个 State  $s$ ,那最后它到底会得到多少的这个  $G'$ ,你就把这个游戏玩完你就知道了

但是这些这个 Value Function,它的能力就是它要未卜先知,未看先猜,游戏还没有玩完,只光看到  $S$  就要预测这个 Actor,它可以得到什麼样的表现,那这个就是 Value Function 要做的事情

举例来说,假设你给 Value Function 这一个游戏画面,它就要直接预测说,看到这个游戏画面,接下来应该会得到很高的 Cumulated Reward,为什麼,因为游戏,这个游戏画面裡面还有很多的外星人



$V^\theta(s)$  is large

假设你的这个 Actor 它很厉害,它是一个好的 Actor,它是能杀得了外星人的 Actor,那接下来它就会得到很多的 Reward

那像这个画面,这已经是游戏的中盘



$V^\theta(s)$  is smaller

游戏的残局,游戏快结束了,剩下的外星人没几隻了,那可以得到的 Reward 就比较少,那这些数值,你把整场游戏玩完你也会知道,但是 Value Function 想要做的事情,就是未卜先知,在游戏没玩完之前,就先猜应该会得到多少的,Discounted Cumulated Reward

那这边有一件要跟大家特别强调的事情是,这个 Value Function 是有一个上标  $\theta$  的,这个 Value Function,跟我们观察的 Actor 是有关係的,同样的 Observation,同样的游戏画面,不同的 Actor,它应该要得到不同的,Discounted Cumulated Reward

我刚才在举例子的时候我说,假设我们有一个好的 Actor,看到这个游戏画面会有高的 Value,看到这个游戏画面会有低的 Value,但是假设你的 Actor 其实很烂,它很容易被外星人杀死,那也许看到这个画面,它的 Value 也是低的,因为有一堆外星人,它随便动两下它就被杀死了,它根本得不到 Reward,这个烂的 Actor 在这个画面,它可能拿到的 V 也是低的,所以 Value Function 的数值,是跟观察的对象有关係的,好这个是 Critic

## How to estimate $V^\theta(s)$

### Monte-Carlo (MC) based approach

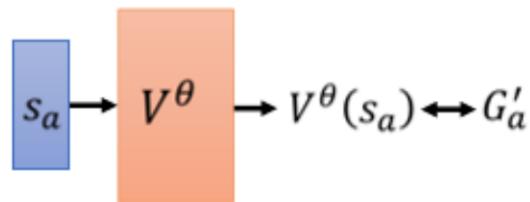
那在讲 Critic 要怎麽被使用,在 Reinforcement Learning 之前,我们来讲一下 Critic 是怎麽被训练出来的,那有两种常用的训练方法,第一种方法,是 Monte Carlo Based 的方法,这边缩写成 MC

## • Monte-Carlo (MC) based approach

The critic watches actor  $\theta$  to interact with the environment.

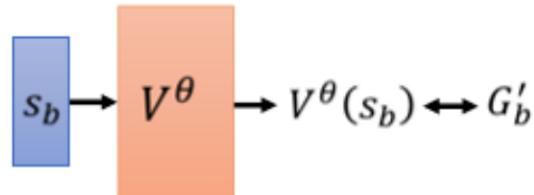
After seeing  $s_a$ ,

Until the end of the episode,  
the cumulated reward is  $G'_a$



After seeing  $s_b$ ,

Until the end of the episode,  
the cumulated reward is  $G'_b$



如果是用 MC 的方法的话,你就把 Actor 拿去跟环境互动,互动很多轮,那 Actor 跟环境互动以后,Actor 去玩这个游戏以后,你就会得到一些游戏的记录

你就会发现说,那这个时候,你的 Value Function 就得到一笔训练资料,这笔训练资料告诉它说,如果看到  $s_a$  作为输入,它的输出,这个  $V^θ(s_a)$  应该要跟  $G'a$  越接近越好

那假设你今天 sample 到另外一个 Observation,看到另外一个游戏画面,把游戏玩完之后发现,得到的 Cumulated Reward 是  $G'b$ ,那这个时候,你的这个 Value Function,输入  $s_b$  它就应该得到  $V^θ(s_b)$ ,那这个  $V^θ(s_b)$  就应该跟  $G'b$  越接近越好

那这个非常直觉,你就去观察 Actor,会得到的 Cumulated Reward,那观察完你就有训练资料,直接拿这些训练资料来训练 Value Function,好 这个 MC ,是一个很直觉的作法

## Temporal-difference (TD) approach

接下来我们来看另外一个,没有那麽直觉的作法,这个作法叫做 Temporal-Difference Approach,缩写是 TD

那 Temporal-Difference Approach,它希望做到的事情是,不用玩完整场游戏,才能得到训练 Value 的资料,你只要在某一个 Observation  $s_t$  的,看到  $s_t$  的时候,你的 Actor 执行了 At 得到 Reward  $r_t$ ,然后接下来再看到  $S_{t+1}$  这样的游戏画面,光看到这样一笔资料,就能够训练  $V\pi(S)$  了,光看到这样子的资料,就可以拿来更新  $V\pi(S)$  的参数了

那如果光看这样一笔资料,就可以更新  $V\pi(S)$  的参数有什麽样的好处,它的好处是你想在 MC 裡面,你要玩完整场游戏,你才能得到一笔训练资料,那有的游戏其实很长,甚至有的游戏也许,它根本就没有不会结束,它永远它都一直继续下去,它永远都不会结束,那像这样子的游戏,你用 MC 就非常地不适合

那这个时候,你可能就希望採用 TD 的方法,好 那怎麽只看到这样子的资料,就拿来训练  $V\pi(S)$

这边举一个例子,我们先来看一下,  $V^θ(s_t)$  跟  $V^θ(s_{t+1})$  它们之间的关係

$$V^θ(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \dots$$

$$V^θ(s_{t+1}) = r_{t+1} + \gamma r_{t+2} + \dots$$

$$V^θ(s_t) = \gamma V^θ(s_{t+1}) + r_t$$

我们说  $V^\theta(s_t)$ , 就是看到  $s_t$  之后的 Cumulated Reward, 所以  $V^\theta(s_t)$  就是  $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2}$  以此类推

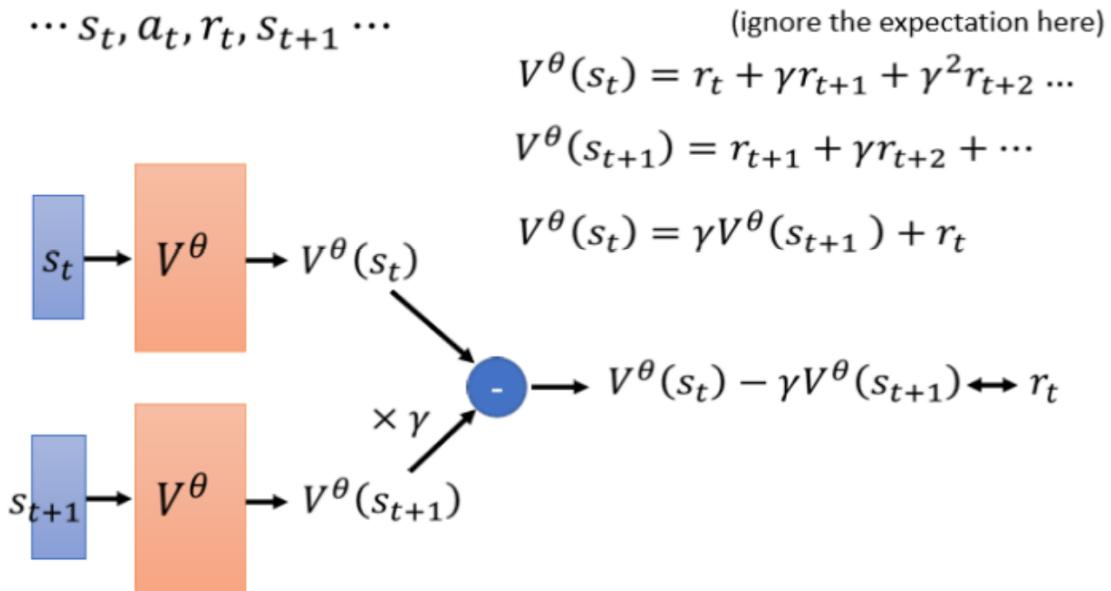
然后  $V^\theta(s_{t+1})$  就是  $r_{t+1} + \gamma r_{t+2}$  以此类推

那你发现说这两个  $V^\theta, V^\theta(s_t)$  跟  $V^\theta(s_{t+1})$ , 它们之间是有关係的

你可以把它写成这样一个式子, 把  $V^\theta(s_{t+1})$  乘上  $\gamma$  再加  $r_t$ , 把  $V^\theta(s_{t+1})$  每一项都乘  $\gamma$  再加上  $r_t$ , 就会变成  $V^\theta(s_t)$ , 所以  $V^\theta(s_t)$  跟  $V^\theta(s_{t+1})$  中间, 有这样子的关係

我们现在, 有这样一笔资料以后, 我们就可以拿来训练我们的 Value Function, 希望 Value Function 可以满足, 这边我们所写的这个式子

### • Temporal-difference (TD) approach



那什麼意思, 就假设我们现在有这样一笔资料, 我们就把  $s_t$  代到 Value Function 裡面得到  $V^\theta(s_t)$ , 我们有  $s_{t+1}$  代到 Value Function 裡面, 得到  $V^\theta(s_{t+1})$ , 虽然我们不知道  $V^\theta(s_t)$  是多少, 我们也不知道  $V^\theta(s_{t+1})$  应该是多少, 我们没有这两个东西的标准答案, 但我们知道它们相减应该是多少

根据上面这一个式子, 我们把  $V^\theta(s_{t+1})$  乘上  $\gamma$ , 然后再去减  $V^\theta(s_t)$ , 把  $V^\theta(s_t)$  减掉  $\gamma$  乘  $V^\theta(s_{t+1})$ , 应该要跟  $r_t$  越接近越好,  $r_t$  在这边, 我们是有蒐集到  $r_t$  这一笔资料的

我们又知道  $V^\theta(s_t)$ , 跟这个  $V^\theta(s_{t+1})$  之间的关係, 所以我们知道  $V^\theta(s_t)$  减掉  $\gamma$  乘上  $V^\theta(s_{t+1})$ , 应该跟  $r_t$  越接近越好, 所以你就有了这样子训练资料, 输入  $s_t$ , 输入  $s_{t+1}$ , 它们都通过  $V^\theta$ , 然后把它们相减, 然后要跟  $r_t$  越接近越好

那这个就是 TD 的方法

### MC v.s. TD

这两个方法, 其实你拿来计算同样的, 观察到的结果, 同样的资料, 同样的  $\theta$ , 你用 MC 跟 TD 来观察, 你算出来的 Value Function, 很有可能会是不一样的

那这边, 就举一个例子, 这个例子是这样子的, 我们观察某一个 Actor, 这个 Actor, 跟环境互动玩了某一个游戏八次, 当然这边为了简化计算, 我们假设这些游戏都非常简单, 都一个回合, 就到两个回合就结束了

- The critic has observed the following 8 episodes

- $s_a, r = 0, s_b, r = 0, \text{END}$
- $s_b, r = 1, \text{END}$   $V^\theta(s_b) = 3/4$
- $s_b, r = 1, \text{END}$
- $s_b, r = 0, \text{END}$

(Assume  $\gamma = 1$ , and the actions are ignored here.)

- 所以那个 Actor 第一次玩游戏的时候,它先看到  $s_a$  这个画面,得到 Reward 0
- 接下来看到  $s_b$  这个画面,得到 Reward 0 游戏结束
- 接下来,这个有连续六场游戏,都是看到  $s_b$  这个画面,得到 Reward 1 就结束了
- 最后一场游戏,看到  $s_b$  这个画面,得到 Reward 0 就结束了

那我们这边,先无视 Actor,为了简化起见无视 Actor,我们也假设, $\gamma$  就等于 1,也就是没有做 Discount,好那这个  $s_b$  应该是多少,  $V^\theta(s_b)$  应该是多少

我们知道这个  $V^\theta(s_b)$ ,它的意思就是这个看到  $s_b$  这一个画面,你会得到的 Reward 的期望值,那  $s_b$  这个画面,我们在这八次游戏中,总共看到了八次,每次游戏都有看到  $s_b$  这个画面,看到  $s_b$  这个画面之后会得到多少 Reward

八次游戏裡面,有六次得到 1 分,两次得到 0 分,所以平均是  $3/4$  分没有问题,所以  $V^\theta(s_b)$  就是  $3/4$ ,妥妥的没有争议

那  $V^\theta(s_a)$  应该是多少,你觉得看到  $s_a$ ,接下来应该要得到多少 Reward ,根据这八笔训练资料,看到  $s_a$  接下来该得到多少 Reward

几乎没有其他答案,所有人都说是 0,好多同学都说是 0,0 是不是一个正确的答案,它既对也不对

其实还有另外一个可能的答案是  $3/4$ ,我看没有人写  $3/4$ ,等一下来解释,为什麼有可能算出  $3/4$ ,但 0 也是一个合理的答案,为什麼你会觉得是应该是 0,0 是用 Monte-Carlo 的想法得到的

为什麼是 0,因为我们看到  $s_a$  只有一次,看到  $s_a$  以后会得到多少 Reward,这是 0,看到  $s_a$  以后得到 Reward 0,再看到  $s_b$  得到 Reward 还是 0,所以 Cumulated Reward 就是 0,所以如果从 Monte-Carlo 的角度来看,我们看到  $s_a$ ,接下来算出来的 G 应该是多少,就是 0 ,所以  $V^\theta(s_a)$  应该就是 0,妥妥的没问题,几乎所有同学都得到了正确答案

但如果你用 TD,你算出来的,可会是不一样的结果

- The critic has observed the following 8 episodes

$s_a, r = 0, s_b, r = 0, \text{END}$	
$s_b, r = 1, \text{END}$	$V^\theta(s_b) = 3/4$
$s_b, r = 1, \text{END}$	
$s_b, r = 1, \text{END}$	$V^\theta(s_a) = ? \quad 0? \quad 3/4?$
$s_b, r = 1, \text{END}$	
$s_b, r = 1, \text{END}$	Monte-Carlo: $V^\theta(s_a) = 0$
$s_b, r = 1, \text{END}$	
$s_b, r = 0, \text{END}$	Temporal-difference:

(Assume  $\gamma = 1$ , and the actions are ignored here.)

$$V^\theta(s_a) = V^\theta(s_b) + r$$

因为  $V^\theta(s_a)$  跟  $V^\theta(s_b)$  中间有这样子的一个关系,这个  $V^\theta(s_a)$  应该要等于  $V^\theta(s_b)$  加上 Reward,就是你在看到  $s_a$  之后得到 Reward,接下来进入  $s_b$ ,那这个  $V^\theta(s_a)$ ,应该等于  $V^\theta(s_b)$  加上这一个 Reward 所以按照这个想法,  $V^\theta(s_b)$  是  $3/4$ ,这个  $r$  是  $0$ ,但  $V^\theta(s_a)$  应该是  $3/4$  对不对,按照 TD 的想法,  $V^\theta(s_a)$  应该是  $3/4$

$$V^\theta(s_a) = V^\theta(s_b) + r$$

$$\begin{matrix} 3/4 & 3/4 & 0 \end{matrix}$$

你可能会问说,那到底 Monte-Carlo 跟 TD,谁算出来是对的,都可以说是对的,它们只是背后的假设是不同的,对 Monte-Carlo 而言,它就是直接看我们观察到的资料,  $s_a$  之后接  $s_b$  得到的, Cumulated Reward 就是  $0$ ,所以  $V^\theta(s_a)$  当然是  $0$

但对于 TD 而言,它背后的假设是这个  $s_a$  跟  $s_b$  是没有关系的,看到  $s_a$  之后再看到  $s_b$ ,并不会影响看到  $s_b$  的 Reward,你现在看这八笔训练资料,给你一种错觉,觉得说  $V^\theta(s_a)$  应该是  $0$ ,那只是因为你 sample 到的资料太少了,看到  $s_b$ ,应该可以期望的 Reward 是  $3/4$ ,只是因为今天正好运气不好,看完  $s_a$  以后再看  $s_b$ ,正好  $r$  是  $0$ ,但是期望值应该是  $3/4$ ,你只是正好运气不好看到  $r$  是  $0$ ,你才会觉得  $s_a$  是  $0$

但是  $s_b$ ,看到  $s_b$  以后得到的期望 Reward 应该是  $3/4$ ,所以看到  $s_a$  以后你会看到  $s_b$ ,那你得到的这个期望的 Reward 也应该是  $3/4$ ,所以从 TD 的角度来看,  $s_b$  会得到多少 Reward,跟  $s_a$  是没有关系的

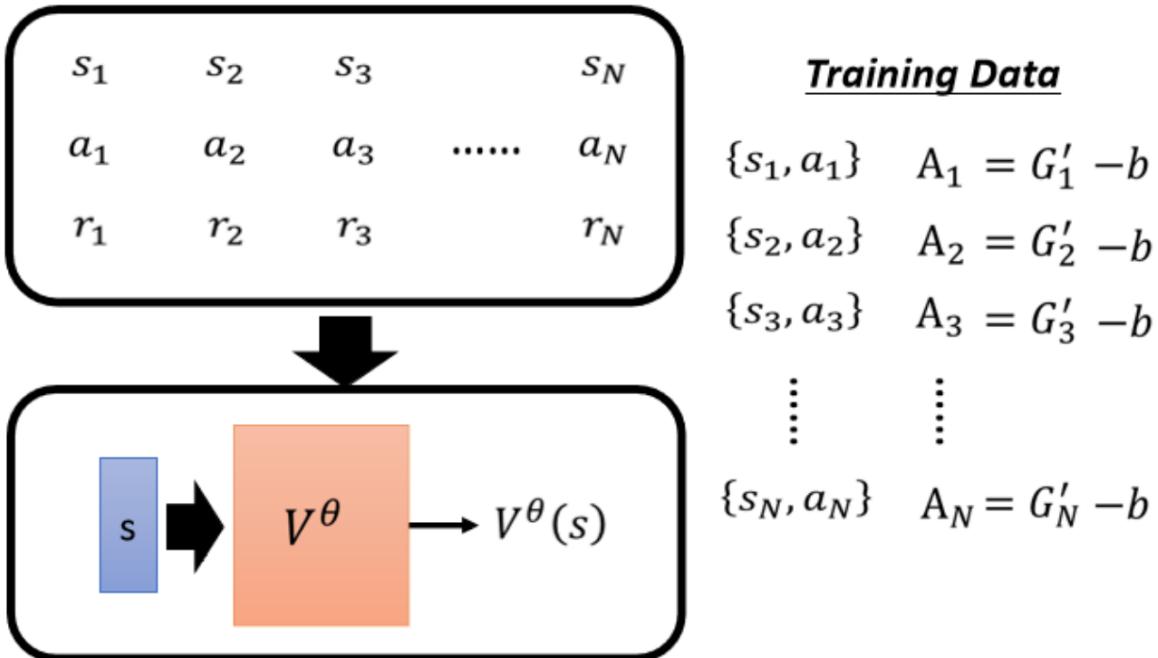
所以你应该,所以  $s_a$  的这个 Cumulated Reward 应该是  $3/4$

所以总之用 MC 来计算,用 TD 来计算,会有微妙的差异

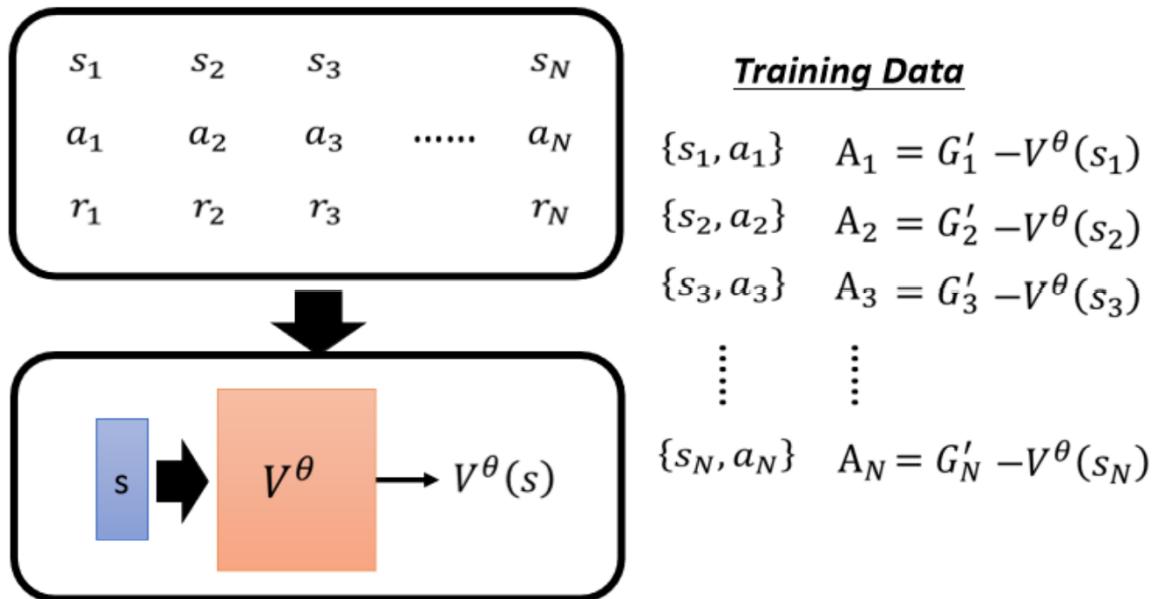
## Version 3.5

Critic 怎么被用在训练 Actor 上面,还记得我们上一次,最后我们讲到这个 Actor 的方法的时候,我们说怎么训练一个 Actor,你就先把 Actor 跟环境互动,得到一些 Reward,然后你得到一堆这个 Observation,跟这个 Action 的 Pair

这个在  $s_1$  执行  $A_1$  的时候多好,得到一个分数  $A_1$ ,那我们说这个  $A_1$ ,它是 Cumulative 的 Reward,那上週也有同学问到说,难道 Cumulative 的 Reward,不需要做 Normalization 吗,需要做 Normalization,所以我们说,这个减掉一个  $b$  当做 Normalization



但这个  $b$  的值应该设多少,就不好说,那我这边告诉大家说,一个  $V$  合理的设法,是把它设成  $V^\theta(S)$



你现在 Learn 出这个 Critic 以后,这个 Critic 给它一个 Step,它就会产生一个分数,那你把这个分数当做  $B$ ,所以  $G1'$  就是要减掉  $V^\theta(s_1)$ ,  $G2'$  就是减掉  $V^\theta(s_2)$ ,以此类推

那再来的问题就是,为什麼减掉  $V$  是一个合理的选择,那我们在下一页投影片,来跟大家解释一下

我们已经知道说这个  $A_t$  代表  $s, a$  这个 Pair 有多好,我们是用  $G'$  减掉  $V^\theta(s_t)$ ,来定义这个  $A$ ,好那我们先来看一下这个  $V^\theta(s_t)$ ,到底代表什麼意思

# Version 3.5

$$\{s_t, a_t\} \quad A_t = G'_t - V^\theta(s_t)$$



$s_t$

(not necessary take  $a_t$ )

(You sample the actions based on  
a distribution)

$V^\theta(s_t)$  是看到某一个画面  $S_t$  以后,接下来会得到的 Reward

它其实是一个期望值,因为假设你今天看到同一个画面,接下来再继续玩游戏,游戏有随机性,你每次得到的 Reward 都不太一样的话,那  $V^\theta(s_t)$  其实是一个期望值

那在这个时候,在看到  $S_t$  的时候,你的 Actor 不一定会执行  $A_t$  这一个 Action

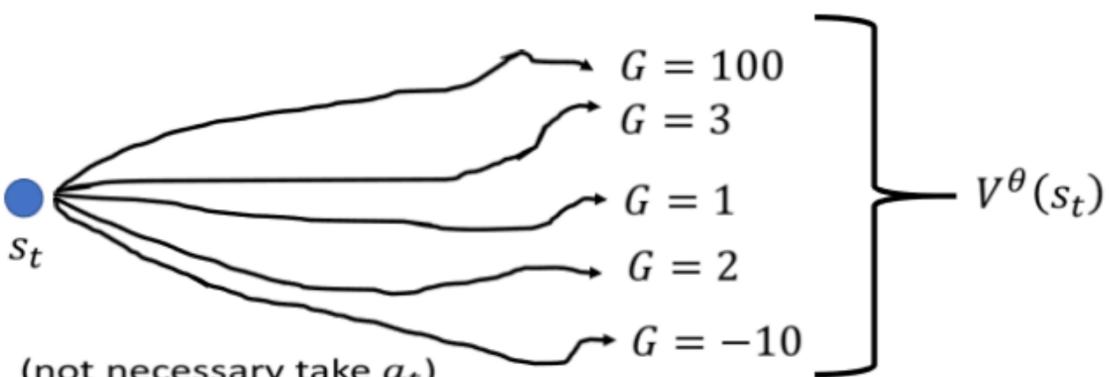
因为 Actor 本身是有随机性的,在训练的过程中,我们甚至鼓励 Actor 是有随机性的,所以同样的  $S_t$ ,你的 Actor,它会输出的这个 Action 不一定是一样的

我们说 Actor 的输出其实是一个 Probability Distribution,是一个在这个 Action 的 space 上面的,Probability Distribution,它还给每一个 Action 一个分数,你按照这个分数去做 sample,有些 Action 被 sample 到的机率高,有些 Action 被 sample 到的机率低,但每一次 sample 出来的 Action,并不保证一定要是一样的,

所以看到  $S_t$  之后,接下来有很多的可能 很多的可能,所以你会算出不同的 Cumulative 的 Reward

# Version 3.5

$$\{s_t, a_t\} \quad A_t = G'_t - V^\theta(s_t)$$



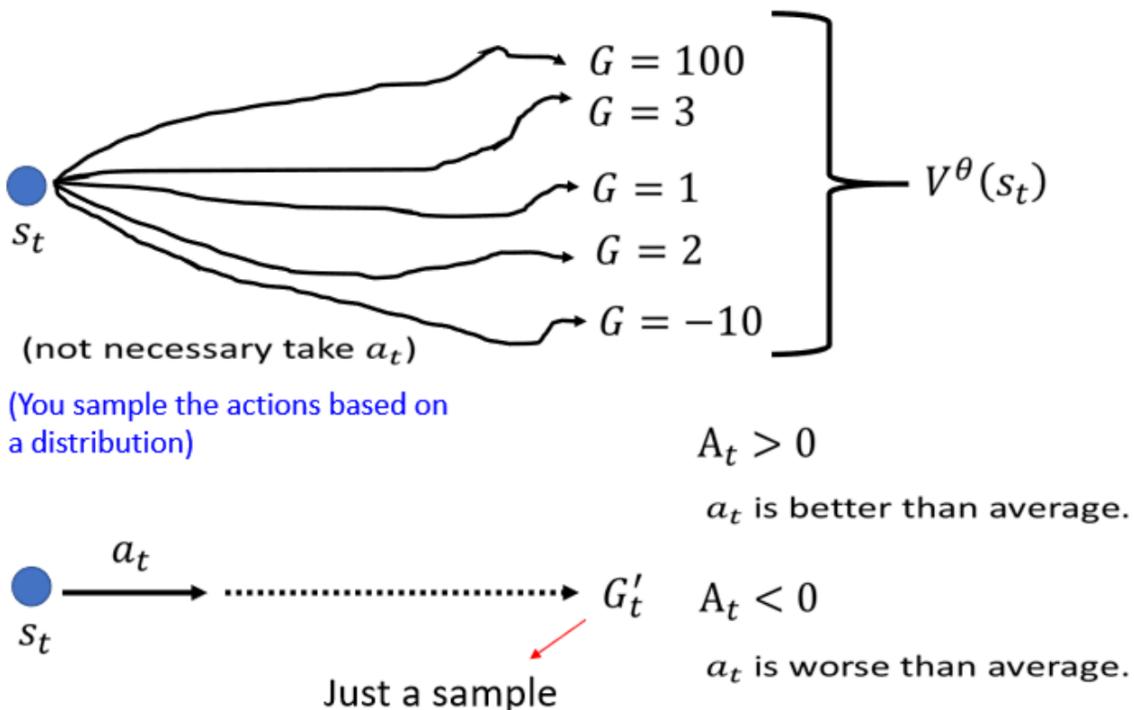
那当然如果你有 Discount 的话,就是 Discounted 的 Cumulative Reward,那我们这边,是把 Discount 这件事情暂时省略掉,那把这些可能的结果平均起来,就是  $V^\theta(s_t)$ ,这是  $V^\theta(s_t)$  这一项的含义

那  $G_t'$  这一项的含义是什么?

$G_t'$  这一项的含义是在  $S_t$  这个位置 在  $S_t$  这个画面下,执行  $A_t$  以后,接下来会得到的 Cumulative Reward

## Version 3.5

$$\{s_t, a_t\} \quad A_t = G'_t - V^\theta(s_t)$$



所以你执行  $A_t$  以后,接下来再一路玩下去,你会得到一个结果 得到一个 Reward,就是  $G'_t$

- 如果  $A_t$  大于 0 代表说,  $G'_t$  大于  $V^\theta(s_t)$ , 这个时候代表说, 这个 Action 是比我们 Random sample 到的 Action 还要好的, 在这边得到  $G'_t$  的时候, 我们确定是执行了  $A_t$ , 那在  $s_t$  在算这个  $V^\theta(s_t)$  的时候, 我们不确定我们会执行哪一个 Action

所以我们执行 Action  $A_t$  的时候, 得到的 Reward 大于随便执行一个 Action 得到的 Reward, 所以当  $A_t$  大于 0 的时候代表说,  $A_t$  大于随便执行的一个 Action, 那这个时候这个 Action  $A_t$  它就是好的, 所以我们给它一个大于 0 的  $A_t$

- 如果  $A_t$  小于 0 代表说, 这个平均的 Reward, 大过执行  $A_t$  得到的 Reward, 你随机采取的 Action, 按照某一个 Distribution, sample 出来的 Action, 得到的这个 Cumulative Reward 的期望值, 大过采取  $A_t$  这个 Action 所得到的 Reward, 那这个时候  $A_t$  就是坏的, 所要给它负的大  $A_t$

所以这样就非常地直觉, 为什麼我们应该把  $G'_t$  减掉  $V^\theta(s_t)$ , 但讲到这边, 你有没有觉得有一些地方有点违和, 什麼地方有点违和, 这个  $G'_t$  它是一个 sample 的结果, 它是执行  $A_t$  以后, 一直玩玩玩 玩到游戏结束, 某一个 sample 出来的结果, 而  $V^\theta(s_t)$  是很多条路径 很多个可能性, 平均以后的结果, 我们把一个 sample 去减掉平均, 这样会准吗, 也许这个 sample 特别好或特别坏, 我们为什麼不是拿平均去减掉平均

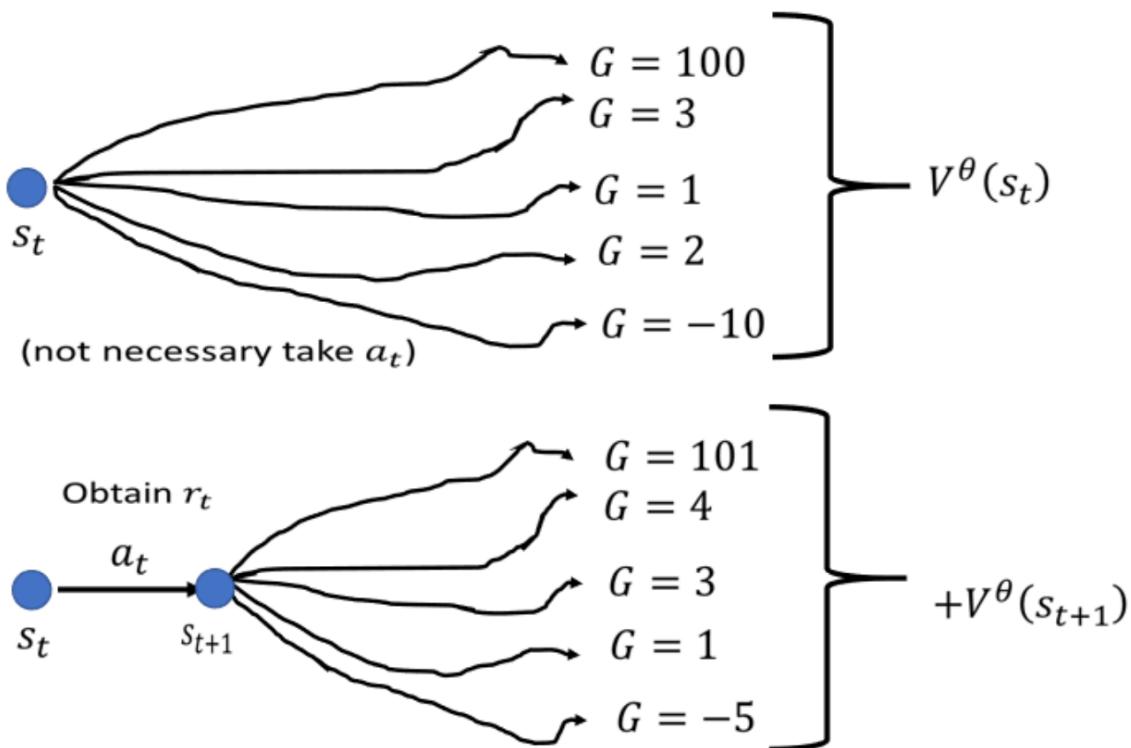
## Version 4

所以我们这一门课要讲的最后一个版本, 就是拿平均去减掉平均

我们执行完  $A_t$  以后 得到 Reward  $r_t$ , 然后跑到下一个画面  $S_{t+1}$ , 把这个  $S_{t+1}$  接下来一直玩下去, 有很多不同的可能, 每个可能通通会得到一个 Reward, 把这些 Reward 平均起来

## Version 4

$$\{s_t, a_t\} \quad A_t = G'_t - V^\theta(s_t)$$



把这些 Cumulative 的 Reward 平均起来, 其实就是  $V^\theta(s_{t+1})$ , 本来你会需要玩很多场游戏, 才能够得到这个平均值,

但没关係, 假设你训练出一个好的 Critic, 那你直接代  $V^\theta(s_{t+1})$ , 你就知道说, 在  $S_{t+1}$  这个画面下, 接下来会得到的, Cumulative Reward 的期望值应该多少

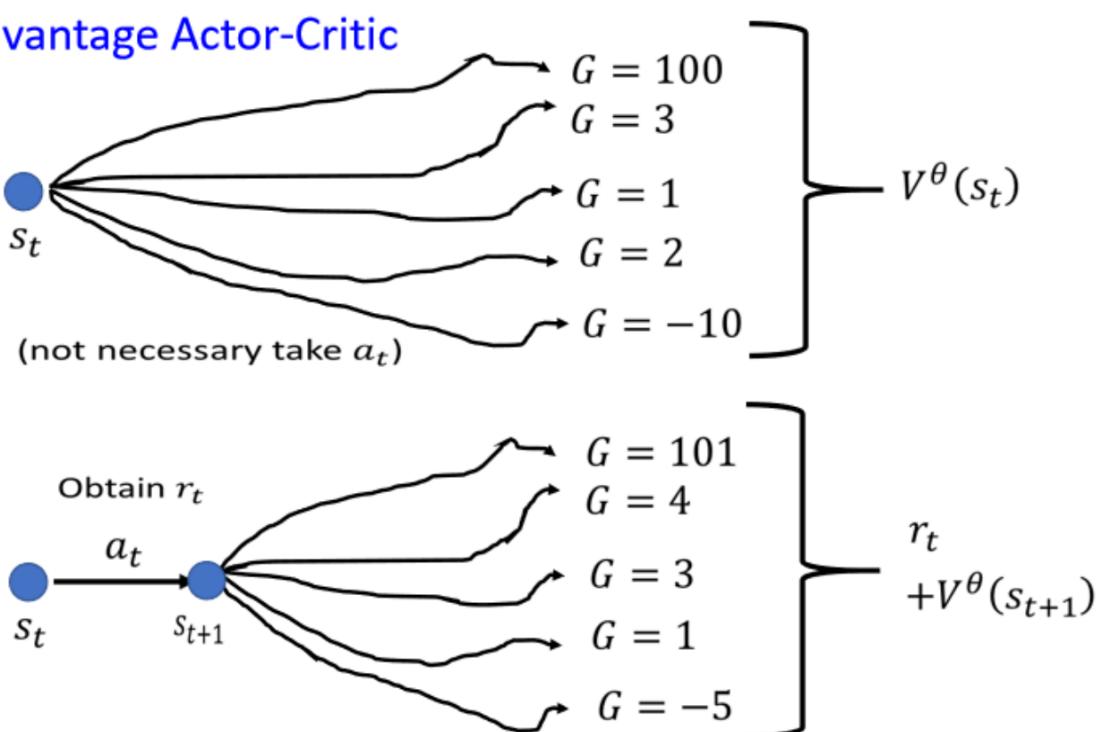
而接下来 你再加上  $r_t$ , 接下来再加上  $r_t$ , 代表说在  $s_t$  这个位置採取  $a_t$

$$r_t + V^\theta(s_{t+1}) - V^\theta(s_t)$$

## Version 4

$$\{s_t, a_t\} \quad A_t = G'_t - V^\theta(s_t)$$

### Advantage Actor-Critic



跳到  $S_{t+1}$  以后,会得到的 Reward 的期望值,因为我们已经知道说,在  $S_t$  这边採取 at 会得到 Reward  $r_t$ ,再跳到  $S_{t+1}$ ,然后  $S_{t+1}$  会得到期望值,期望的 Reward 是  $V^\theta(s_{t+1})$

所以我们这边,再给它加上  $r_t$ ,代表说在  $S_t$  这边执行 At 以后,会得到的 Reward 的期望值,接下来再把这两个东西相减,再把  $r_t + V^\theta(s_{t+1})$  减掉  $V^\theta(s_t)$

$$r_t + V^\theta(s_{t+1}) - V^\theta(s_t)$$

$$\{s_t, a_t\} \quad A_t = G'_t - V^\theta(s_t)$$

也就是我们把  $G'$  换成  $r_t + V^\theta(s_{t+1})$ ,再减掉  $V^\theta(s_t)$

我们就知道说,採取 at 这个 Action 得到的期望 Reward,减掉根据某个 Distribution sample 一个 Action 得到的 Reward,两者的期望值差距有多大

那如果  $r_t + V^\theta(s_{t+1})$  比较大,就代表 at 比较好,它比随便 sample Reward 好

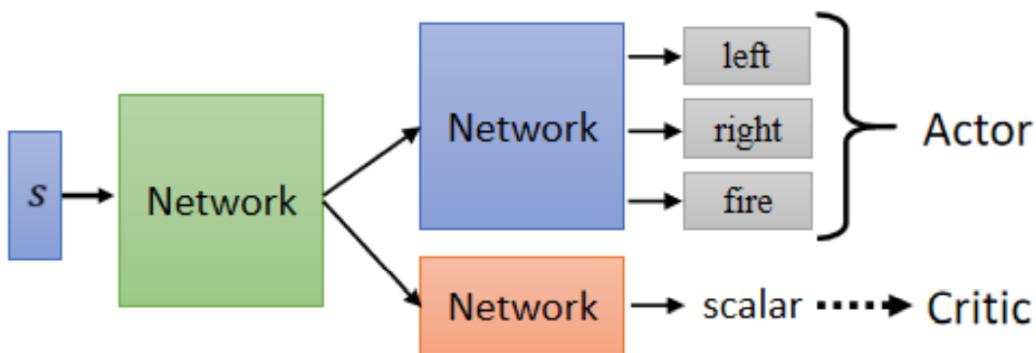
$r_t + V^\theta(s_{t+1})$  小于  $V^\theta(s_t)$ ,就代表 at 它是 Lower Than Average,它比从一个 Distribution sample 到的 Action 还要差

所以今天,这个就是大名鼎鼎的一个常用的方法,叫做 Advantage Actor-Critic,在 Advantage Actor-Critic 裡面,你是怎麽定义 at 的,也就是  $r_t + V^\theta(s_{t+1})$  减掉,  $r_t + V^\theta(s_{t+1}) - V^\theta(s_t)$ ,就是我们的 At 了

## Tip of Actor-Critic

这边有一个训练 Actor-Critic 的小技巧,那你在作业裡面也不妨使用这个技巧

- The parameters of actor and critic can be shared.



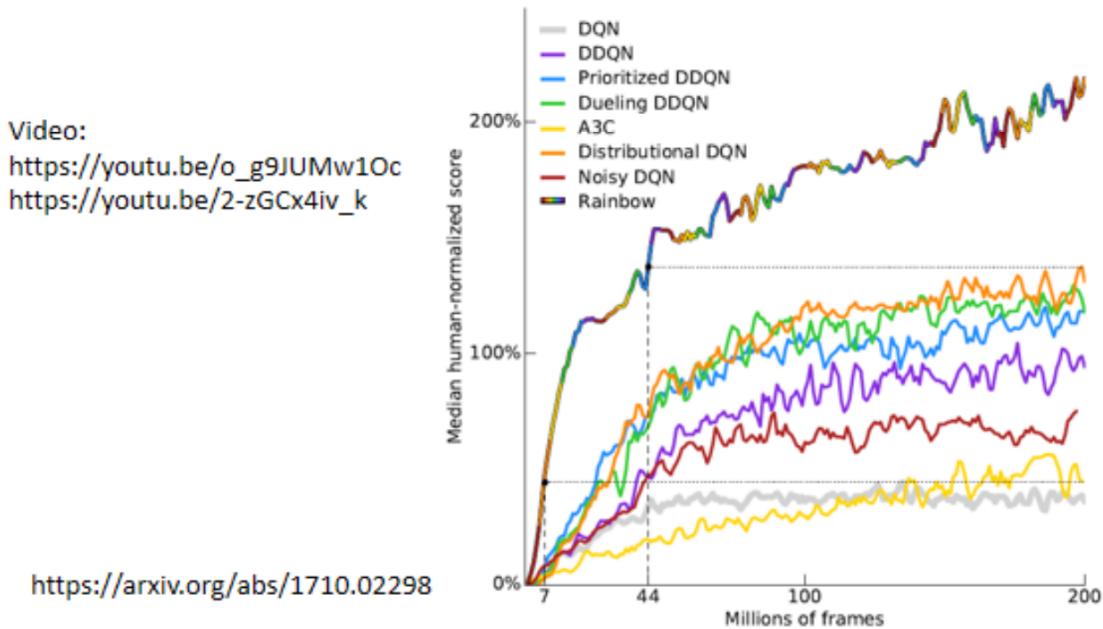
Actor 是一个 Network,Critic 也是一个 Network,Actor 这个 Network,是一个游戏画面当做输入,它的输出是每一个 Action 的分数,Critic 是一个游戏画面当做输入,输出是一个数值,代表接下来会得到的 Cumulative 的 Reward

这边有两个 Network,它们的输入是一样的东西,所以这两个 Network,它们应该有部分的参数可以共用吧,尤其假设你的输入又是一个非常複杂的东西,比如说游戏画面的时候,前面几层应该都需要是 CNN 吧,要了解这个游戏画面需要用到的 CNN,也许是差不多的吧

所以 Actor 跟 Critic,它们可以共用前面几个 Layer,所以你今天在实作的时候往往,你会把你的 Actor-Critic 设计成这个样子,Actor-Critic ,它们有共用大部分的 Network,然后只是最后,输出不同的 Action,就是 Actor,输出一个 Scalar,就是 Critic,好 那这是一个训练 Actor-Critic 的小技巧

# Outlook: Deep Q Network (DQN)

那其实今天讲的,并不是 Reinforcement Learning 的全部,那其实在 Reinforcement Learning 裡面,还有一个犀利的做法,是直接採取 Critic,也就是直接用 Critic,就可以决定要用什麼样的 Action



那其中最知名的就是,Deep Q Network (DQN),那不过 这边我们就不细讲 DQN 了,如果你真的想知道 DQN 的话,可以参考过去上课的录影,那 DQN 哇 有非常非常多的变形

这边 就是找一个非常,有一篇非常知名的 Paper 叫做 Rainbow,裡面 就是试著去尝试了各种 DQN 的变形,试了七种 然后再把这七种变形集合起来,因为有七种变形集合起来,所以他说它是一个彩虹,所以他把它的方法叫做 Rainbow,那我也把这个 Paper 留在这边给你参考,那如果你想知道 Rainbow 裡面的,每一个小技巧是怎麽做的话,你就参见上课录影,过去的课程,有把 Rainbow 裼面的每一个小技巧,都讲过一遍

## Q&A

Q1:  $s_a$  后面接的不一定是  $s_b$  吧,这样怎麽办

A1: 这是一个很好的问题,  $s_a$  后面不一定接  $s_b$ ,那这个问题,在刚才我们看到的那个例子裡面,就没有办法处理,因为在刚才那个,我们看到那个只有 8 个 Episode 的例子裡面,  $s_a$  后面就只会接  $s_b$ ,所以我们观察 没有观察到其它的可能性,所以我们没办法处理这个问题,所以这就告诉我们说,在做 Reinforcement Learning 的时候,  $s_a$  mple 这件事情是非常重要的,你 Reinforcement Learning,最后 Learn 得好不好,跟你在  $s_a$  mple 的时候  $s_a$  mple 得好不好,关係非常大,喔 所以这个 Reinforcement Learning,是一个非常吃人品的方法啦,所以你在作业裡面你可以体验一下,就你  $s_a$  mple 到的结果,对你最后 Training 的结果,有非常大的影响

Q2: 每一个 V,都需对应到固定的环境发生顺序吗

A2: 我没有很确定你的问题,但是我试著回答一下,就是每一个 V 它不会固定,它不会对应到固定的环境发生顺序,如果你的游戏有随机性的话,那 V 其实是代表了一个期望值,它想要算的就是,给某一个 Observation,看到某一个游戏画面以后,接下来你会得到的 Cumulative Reward 的平均值,它的期望值,如果你的游戏有随机性的话,V 代表的是期望值,你看到某一个游戏画面以后,然后接下来会发生什麼事情,不见得是一样的,但把所有的可能性都平均起来,取它的期望值,这个就是 V 所代表的意思

Q3: 后面出现的 S 应该是不固定的,这样怎麽代公式

A3: 好 那个我想我刚才应该算是有回答到了,后面出现的 看到某一个这个 Observation,后面出现的 Observation 确实是不固定的,那如果有些状况,某些 Observation 你没观察到的话,哇 那你真的就没办法训练

Q4: 就是拿 V 当一般人的实力,超过它就是猛,没超过就是烂吗

A4: 对 就是这样,V 就是平均的实力,超过 V 就是好

Q5: 想请问这个 Distribution 要从哪裡知道

A5: 我想你这个 Distribution 问的是那个 Actor 的 Distribution 啊 对不对,我们说,Distribution Action 的 Distribution,Action 是从某一个 Distribution,sample 出来的,那个 Distribution 是谁,那个 Distribution 是这样,就是你的 Actor 不是像是一个 Classifier 吗,你的 Actor 像是一个 Classifier,然后你把 s 丢进去,每一个 Action 都会有一个分数,那你把这个分数,通过 Soft Max,就做一个 Normalize,它就变得像机率一样,然后按照那个机率去做 sample,那这个就是 Actor,从一个 Distribution sample 出来的,这句话的意思,

## Reward Shaping

好 这个接下来想要跟大家分享的是一个叫做 Reward Shaping 的概念

什么是 Reward Shaping 呢? 到目前为止我们学到的东西是, 我们把我们的 Actor 拿去跟环境互动, 得到一堆 Reward, 那把这些 Reward 做某些的整理以后, 得到这边的分数 A, 有了这边的分数 A, 你就去教你的 Actor, 该做什么 不做什么。

### Sparse Reward

$$A_t = r_t + V^\theta(s_{t+1}) - V^\theta(s_t)$$

$s_1$	$s_2$	$s_3$	.....	$s_N$
$a_1$	$a_2$	$a_3$	.....	$a_N$
$r_1$	$r_2$	$r_3$	.....	$r_N$

### Training Data

$\{s_1, a_1\}$	$A_1$
$\{s_2, a_2\}$	$A_2$
$\{s_3, a_3\}$	$A_3$
$\vdots$	$\vdots$
$\{s_N, a_N\}$	$A_N$

但是在这个 Reinforcement Learning 里面, 我们很怕遇到一种状况是, 假设 Reward 永远都是 0 的时候怎么办呢, 假设多数的时候 Reward 都是 0, 只有非常低的机率, 你会得到一个巨大的 Reward 的时候, 那怎么办呢?

假设 Reward 几乎都是 0, 那意味着什么, 意味着说你今天这个 A 不管怎么算都是 0。对每一个 Action 都是差不多的, 反正不管执行什么 Action, 得到的 Reward 都是 0 嘛, 所以执行这个 Action 执行那个 Action 根本没差

那如果不管执行什么 Action, Reward 几乎都是 0 的话, 那你根本没有办法去 Train 你的 Actor。那讲到这种 Sparse Reward 的问题, 也许有人马上会想到的是, 下围棋也许是一个 Sparse Reward 的问题。因为在下围棋的时候, 你每落一子, 你并没有得到 Reward, 你并没有得到 Positive 或 Negative 的 Reward, 只有在整场游戏结束, 落完最后一子的时候 在中盘, 在落完最后一子游戏结束的时候。你赢了才会得到 Positive 的 Reward, 你输了才会得到 Negative 的 Reward。

但是我觉得相较于有一些其他任务，这个下围棋，还算是比较有 Reward 的 RL 的问题。举例来说假设今天的问题是：你要教机械手臂去拴螺丝。而教机械手臂拴螺丝这个问题，你合理的 Reward 的定义是，假设今天机械手臂成功把螺丝拴进去，它就得到 Positive Reward，没有把螺丝拴进去，Reward 就是0。

但是你想看一开始你的机械手臂，它里面的 Actor 的参数是随机的，所以它就在空中随便挥舞，怎么挥舞 Reward 都是 0，不像是下围棋，你至少整场游戏玩完，你还有正面的或负面的 Reward，而像机械手臂，你要叫机械手臂去拴螺丝，除非它正好非常巧合的拿起一个螺丝，再把它拴进去它得到正向的 Reward。不管它做什么事情都没差，得到的 Reward 通通都是 0 分，好那遇到这种状况的时候怎么办呢？

遇到这种状况的时候有一个解法就是，我们想办法去提供额外的 Reward 来引导我们的 Agent 学习，也就是说在原来的 Reward，也就是我们，真正要 Agent 去 Maximize 的 Reward 之外，我们再定义一些额外的 Reward，我们定义这些额外的 Reward 来帮助我们的 Agent 学习。那这种订额外的 Reward 来帮助 Agent 学习这种事情呢，就叫做 Reward Shaping。

那你知道我们人呢其实也很擅长做 Reward Shaping。举例来说这让我想到一个这个妙法莲华金钟的故事。这个故事呢，出自妙法莲华金钟的化城喻品，这个故事就是有一个领队，然后带一群人呢去找宝藏，然后宝藏呢在五百由旬之外，那由旬是什么单位我也忘了，那你就当做很远就对了，就当500 万公里之外。然后呢这群人呢走到半路，已经过了半途了，觉得很累不想再往前走了，那大家就坐在地上哀嚎不想再往前走了。那领队看大家不想再往前走 怎么办呢，他就跟大家说再往前 10 公里有一个五星级的饭店，大家就可以去休息了，大家就看到五星级的饭店就很高兴。就去休息了然后隔天早上饭店就不见了。然后领队说那个饭店呢是我用法力变出来的，为了鼓励大家再继续往前走，免得大家半途而废

这妙法莲华经说这个故事是为了表示说，佛最后希望大家成就的是佛道，但佛道非常的长，所以中间呢，设了小圣 中圣 大圣等不同的位阶来引导大家前进。那如果对应到比较生活化的例子就是，比如说现在叫你念博士，你可能觉得你要博士毕业才能够拿到博士学位才能够得到 Reward。那你就会觉得哇这个路呢非常的长然后就不想要念博班了。但是如果告诉你说 你先修个课就可以得到 Reward 然后做一点专题虽然可能也没有做出什么厉害的成果。但老师就会说你好棒 你也得到 Reward。然后最后先发一个 Second Tier 的 Conference 得到 Reward，再发表 Top-tier Conference 再就得到 Reward，然后最后你就可以博班毕业。这样一步一步的往前走，最终你就可以达成最终的目标这样。

好那这个呢就是 Reward Shaping 的概念

## Example

---

好那我们接下来呢。就举一个 Reward Shaping 真正实际使用在 RL 里面的例子给大家参考。

那这边举的例子呢是用 VizDoom 来跟大家举例，那因为怕大家不知道 VizDoom 是什么。所以这边呢还是放一个影片给大家看一下[https://www.youtube.com/watch?v=73YyF1gmlus&list=PLJV\\_el3uVTsMht7\\_Y6sgTHGHp1Vb2P2J&index=33&ab\\_channel=Hung-yiLee](https://www.youtube.com/watch?v=73YyF1gmlus&list=PLJV_el3uVTsMht7_Y6sgTHGHp1Vb2P2J&index=33&ab_channel=Hung-yiLee)六分十四秒开始。

这边的每一个选手都不是人，每一个选手通通都是机器，所以你仔细看一下会发现说，有些人的行为很奇怪，比如说右上角这个人，它还蛮容易卡墙的

好这个是让你知道一下 VizDoom，大概是什么样的游戏，那这个影片非常长，它是个长达两小时的史诗级的战斗，你有兴趣再慢慢把它看完。在当年那个 VizDoom 的比赛里面，第一名的队伍就有用到了 Reward Shaping 的概念。那在 VizDoom 这个游戏里面，你被敌人杀掉你就扣分，你杀了敌人就加分，但如果你光凭着这个游戏里面真正的 Reward，来训练 Agent 你是很难把它训练起来的。所以呢在这一篇文章里面就使用了一些 Reward Shaping 的概念，那我们就来看一下它是怎么定这些 Reward 的  
举例来说这个第一点呢 我们等一下再看

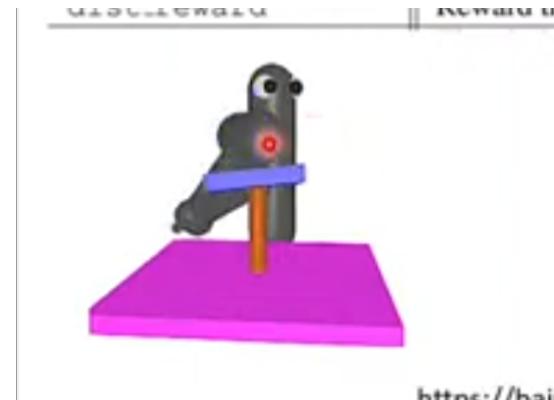
我们先看第二点。第二点 它说如果今天有扣血，那你就得到负的 Reward，那其实在游戏里面扣血并没有惩罚，扣血并不会扣分，你要死掉才扣分，但是如果机器要到死掉，才知道它得到负的 Reward，那它可能要很花很久的时间，才能学到扣血跟死掉之间的关联性，所以这边直接告诉机器不可以扣血，扣血是坏的事情。好 然后这边是如果损失弹药就扣一点分数，然后捡到这个医药包就加分，捡到弹药的补给包就加分，那这些事情在游戏里面是不会影响分数的。但是我们人 我们 Developer另外强加给机器来引导机器学习。

好 接下来就是一些比较有趣的 Reward 了。举例来说 它说呢如果你的 Agent 总是待在原地那就要扣分，为什么需要这样的 Reward 呢，因为如果你没有这样的 Reward，对 Agent 来说一开始它非常弱，它出去走呢 随便走两步就被敌人杀死了，所以对它来说 对一开始很弱的 Agent 来说，可以得到比较高 Reward 的做法，也许就是待在原地，待在原地至少 Reward 是 0 。那出去碰到敌人还要被扣血那多划不来，所以为了避免 Machine 最后什么都没有学到，就只会待在原地，所以强制告诉它说，你只要待在原地就直接扣分，然后还告诉它说如果你动， 就给你一个很小的分数，这边你只要每动一下就给你 9 乘以 10 的 -5 次方，一个非常小的 但是是 Positive 的 Reward。

好 但是光是要求机器动是不够的，所以这边又多加了一个非常有趣的 Reward这个 Reward 是每次如果 Agent 活着，Agent 每活着 它就要被扣分。你可能觉得很奇怪 活着不是一件好事吗？为什么活着反而应该要被扣分呢，那是因为假设现在活着没有扣分，或者是什至是一件正面的事情，对 Agent 来说，它会学到的可能就是边缘 OB ， 你虽然有要求它动，那你要求它动没有关系，它就在边缘一直自转就好，它在边缘一直转圈圈就好，然后看到敌人就躲开，不要跟敌人做任何的正面交锋，那对你的 Agent 来说，也许这样是一个最安全的做法，但是为了强迫 Agent 学习去杀敌人，所以反而告诉它说你只要活着就是扣分。你要想办法不可以活太久，你要想办法去跟别人交手，所以这是一个非常有趣的Reward Shaping 的方式

那看到这些 Reward Shaping 你就会发现说，Reward Shaping 这一件事情呢，其实是需要花 Domain Knowledge 的。它其实是需要凭借着人类对你现在环境的理解来强加上去

那今天再举另外一个 Reward Shaping 的例子



<https://baii>

假设你今天要训练一个机器手臂，那这个机器手臂的工作呢，就是把这个蓝色的板子，插到这个棍子上，好 那像这样子的任务，你要凭着 RL 的方法让机器凭空学会，把蓝色的板子插到这个棍子上，没有那么容易。但是你可能会想到一个，很直觉的 Reward Shaping 的方法是，假设今天这个蓝色的板子，离这个棍子越近，那我们得到的 Reward 就越大，但是如果你仔细想想会发现说，单纯让蓝色的板子离这个棍子越近是不够的，为什么让蓝色的板子离棍子近是不够的呢，你可以看看这个像右边这两个 Case，机械手臂也是想把蓝色的板子挪进棍子。但它做的事情，其实就是去打那个棍子。从侧面接近从侧面接近是没有用的，把蓝色的板子从侧面接近棍子，并不能够达到你最终的目标。所以如果我们单纯只是说，现在你的蓝色的板子离这个棍子越近，它的 Reward 就越大，你用 Reward Shaping 的方法。把蓝色的板子跟棍子之间的距离。当做一个新的 Reward但可能对你最终想要解决问题本身，是不一定有帮助的所以 Reward Shaping 这个东西。你在用的时候必须要小心。它需要你对这个问题本身有足够的理解。你才有办法使用 Reward Shaping 这样的招数

## Curiosity Based 的 Reward Shaping

好那 Reward Shaping 里面

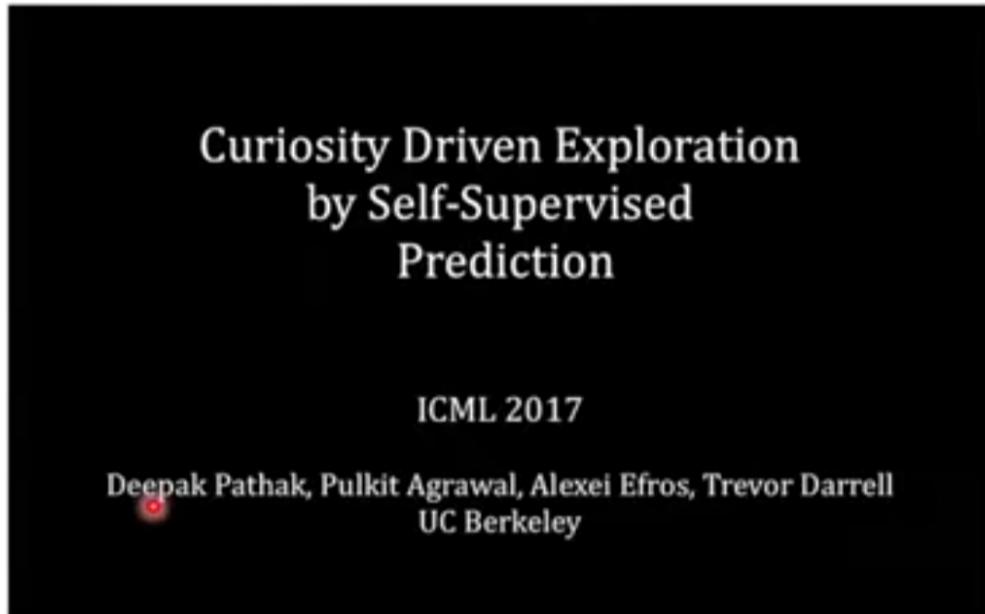
有一个特别有趣而知名的做法叫做

Curiosity Based 的 Reward Shaping

## **Reward Shaping - Curiosity**

<https://arxiv.org/abs/1705.05363>

Obtaining extra reward when the agent sees something new (but meaningful).



Source of video: <https://pathak22.github.io/noreward-rl/>

也就是呢 给机器加上好奇心。所谓好奇心的意思就是要去探索新的事物，所以在原来的 Reward 之外，我们加上一个 Reward 这个 Reward 是，如果机器它在活动的过程中看到新的东西，它就被加分。但这边又有一点要强调的是，新的东西必须是有意义的新，不是无谓的新。什么叫做有意义的新 不是无谓的新呢。那这个我们等一下再解释。

好那这个 Curiosity based 的这个 Reinforcement Learning 它来自于 ICML 2017 的这一篇文章。那这篇文章里面有一个非常惊人的 Demo，那我们来播一下这个 Demo。[https://www.youtube.com/watch?v=73YyF1gmlus&list=PLJVeI3uVTsMhtt7\\_Y6sgTHGHp1Vb2P2J&index=33&ab\\_channel=Hung-yiLee](https://www.youtube.com/watch?v=73YyF1gmlus&list=PLJVeI3uVTsMhtt7_Y6sgTHGHp1Vb2P2J&index=33&ab_channel=Hung-yiLee) 13分12秒

我想稍微解释一下刚才那个影片的意思。刚才那个影片的意思是说，我把它停下来 等我一下，就是它让机器玩玛利欧，那他甚至在玛利欧这个游戏里面。没有任何的 Reward，他甚至没有告诉机器说，破关就可以得到 Positive 的 Reward。那其实你告诉机器破关可以得到 Positive Reward 也没用啦。这种 Reward 太 Sparse 了，大概很难拿来训练 Agent，它只告诉机器说，你要不断地看到新东西。那光是这一件事情就可以让机器学会，破玛利欧的其中一些关卡，那当然这样子的方法，也许对玛利欧而言是最合适的，你知道玛利欧是横向卷轴游戏嘛。那你要破关就是要不断地往右走嘛，那机器它要看到新的东西，就不断地需要往右走嘛，所以机器会光藉由 Curiosity 这件事，就可以学到破解玛利欧的一些关卡。不过它有尝试说训练在前面几关，然后直接测直接把 Agent 放在那个地下的关卡，然后就发现说做不起来。在地下关卡还是要微调一些 Network，微调一下 Network 才做得起来

好那这个影片还有后半段啦，我们我们直接从某个地方开始看好了。他接下来是直接叫机器去玩那个 VizDoom。直接叫机器去玩 VizDoom。我们也来放一下 VizDoom 的部分好了。好那最后那一个背景有一堆杂讯，有点像是那个电视机坏掉那种杂讯的那个画面，你可能有点不知道它这个影片想要表达什么，那那个部分就是我想要。我刚才在最前面讲到的有意义的新这件事情。有意义的新这件事情就是假设我们要求 Agent，要一直看到新的东西。那如果你的画面背景有一个杂讯，那杂讯会不断地变化，所

以对 Agent 来说杂讯是新的东西，也许它就不会学到去探索新的环境了，反正光站在那边看杂讯就够了，它会觉得它不断地看到新的东西。所以其实，在 Curiosity Based 的这个 RL 里面，也有想办法克服这种没意义的新，这种看到杂讯的问题，至于实际上怎么做，那你再参考他的论文

## No Reward : Learning from Demonstration

好那有关RL的最后一个部分，我想要跟大家分享的是，假设今天，如果我们连Reward都没有，该怎么办呢？

在刚才的课程里面，我们是有Reward的，只是有时候Reward非常地Sparse，所以你要做Reward Shaping。但是假设今天，连Reward都没有的话，到底该怎么办呢？

那为什么有时候会连Reward都没有呢？其实像Reward这种东西，往往只在一些比较artifisial的环境，比如说游戏里面，特别容易被定义出来。在游戏里面，有一个记分板，所以你特别容易去定义说在游戏里面，怎么样的行为是好的，怎么样的行为是不好的，某一个行为有多好，或某一个行为有多不好。但是在真实的环境里面，你要定义Reward这件事情，有可能是非常地困难的，假设你今天要用RL的方法来学，学叫自驾车学会在路上走，那到底在路上走，做什么样的事情，会得到什么样的Reward呢？它礼让行人，就给它加100分吗，还是应该加1000分，那闯红灯，应该扣50分，还是扣50000分呢，那像这种东西，你根本不知道要怎么定。所以在更真实，在真实的环境中，有时候我们根本不知道要怎么定义，Reward这个东西，而且有时候，那你说，虽然我们不知道怎么定Reward，但我们可以凭着人类的智慧，去想一些Reward出来，来Guide一个Machine，像我们刚才讲的那种RewardShaping，不就是一个很好的例子吗？我们自己想一些Reward出来，来叫Machine学，但有时候你在想Reward教Machine学的时候，如果你的Reward没想好，Machine可能会产生非常奇怪的行为，你无法预期的行为，那这边举一个比较极端的例子。

### Three Laws of Robotics:

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.



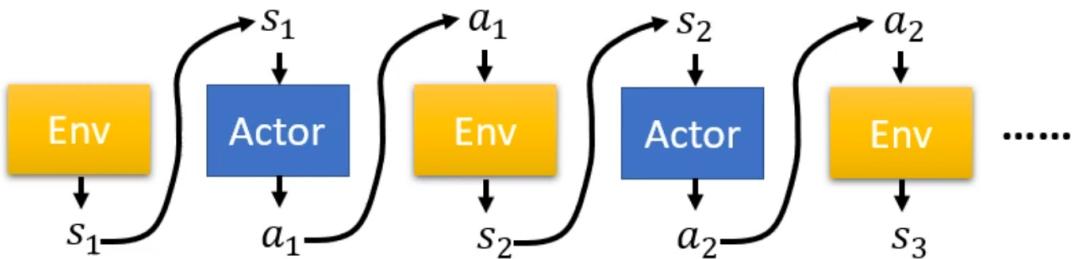
有一个电影，叫做机械公敌，在机械公敌里面呢，当然这就未来的世界啦，有一大堆机器人，那些机器人的行为，必须符合三条原则，这三条原则是，第一条不能够伤害人类，也不能够坐视人类被伤害。第二条是，在不违反第一条的前提下，机器必须要听从人类的命令。那第三条是机器必须要保护自己，在不违反第一条跟第二条的前提下。所以你可以想像，这三条Law，这三条规则，就代表说，机器如果不违反这三条规则，就得到PositiveRewards，违反这三条规则，就得到非常Negative的Reward。然后有了这三条规则以后，机器呢，就自己去发展，自己去学习，但是最终呢，机器就得到神逻辑，在不违反这三条规则的前提下，要得到最大Reward的方式，就是把人类监禁起来，因为人类会自我伤害，所以应该把人类监禁起来，避免他们自我伤害，这样机器就可以得到最高的Reward，所以这个例子告诉我们说，光订Reward是不够的，机器可能会有神逻辑，展现出你意想不到的行为。

当然机械公敌里面是一个比较极端的例子，那举一个比较有可能发生的例子是，假设你训练机器去，比如说收盘子，那这个是我在文献上实际看过的例子啦，那你知道在文献上有很多文献是，要用ReinforcementLearning的方法，来训练机械手臂嘛，假设你今天要训练这个机器，把盘子摆到一个固定的位置，它把盘子放到指定的位置，就得到PositiveRewards，这个是你定义的Reward。然后你用Reinforcement Learning去学，发现机器会把盘子放到指定的位置，但它都非常大力用摔的，然后结果盘子都被摔破了，那是因为你根本没有告诉机器说，不能把盘子摔破。所以如果，你就会发现说，机器没有告诉它不能把盘子摔破的前提之下，它为了达成目标，可能就把盘子摔破了。这个时候你只要赶快再订新的Reward说，如果摔破盘子就要扣100分，但问题是，盘子已经被摔破了，已经来不及了，所以有时候，人订的Reward，不见得是最好的。

# Limitation Learning

所以怎么办呢，在没有Reward的情况下，我们要怎么训练一个Agent去跟环境互动呢？

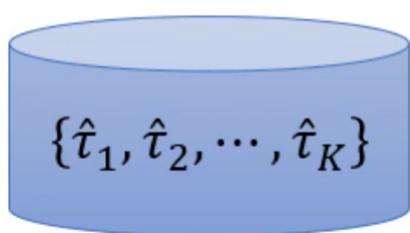
## Imitation Learning



Actor can interact with the environment, but reward function is not available

那在没有Reward的状况下，让机器跟环境互动，其中一个方法，叫做Imitation的Learning，那在ImitationLearning里面，我们假设Actor，它仍然可以跟环境互动，但它不会从环境得到Reward，Environment仍然会送出Observation给Actor，Actor仍然会做出回应，Environment仍然会随着Actor的回应，给不同的Observation，但是没有Reward这个东西。

We have demonstration of the expert.

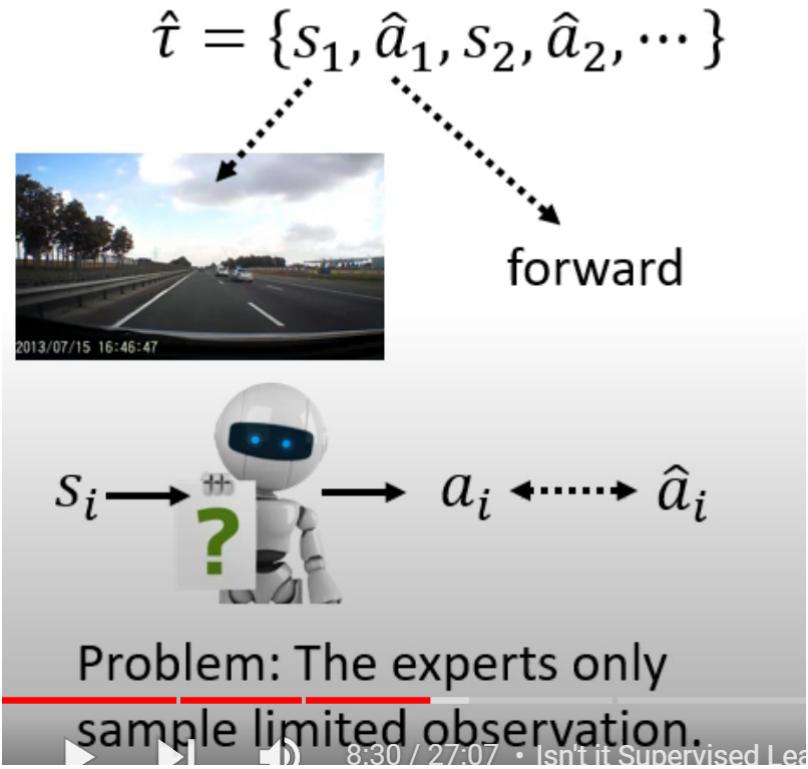


Each  $\hat{\tau}$  is a trajectory  
of the export.

没有Reward这个东西要怎么学呢？虽然没有Reward，但是我们有另外一个东西，这个东西，是Expert的示范，我们找很多Expert，通常是人类，我们找很多人类，也来跟这个环境互动，把人类跟环境的互动记录下来，这些东西就是，这些纪录就是Expert的示范，就是Expert的示范，然后把这些Expert的示范呢，叫做 $\tau$ ，我们用这个这个上标，来代表人类的Expert的示范，那我们现在呢，就是要凭借着这些示范，还有跟环境的互动，来进行学习。这样讲也许有点抽象，什么叫做Expert的示范呢，假设你今天呢，要教机器开自驾车，那人类驾驶的行驶纪录，那就是Expert的示范。人类驾驶的行驶纪录可以告诉机器说，在这个路口，你应该打一下方向盘等等，那这些就是Expert的示范，或者是，你想要叫机器做一些指定的动作，比如说倒水排碗盘，你可能会先拉着机械的手臂示范一次，那人去拉着机械手臂示范一次。这件事情，就是Expert的Demonstration，就是这边的 $\tau$ ，那Imitation Learning要做的事情，就是从这些 $\tau$ ，还有与环境的互动，进行学习。

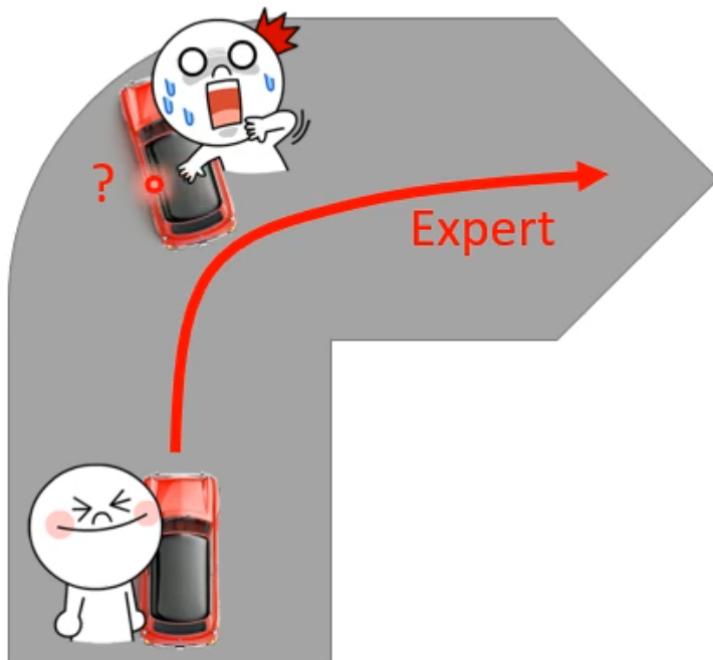
那讲到这边可能有同学就会想说，欸这个问题听起来好像挺简单的，这个不就是Supervised Learning吗，我们就把它当作Supervised Learning的问题，来看待就好啦。

- Self-driving cars as example



假设我们今天要训练自驾车，你又有人类驾驶的纪录，也就是你有记录说，欸人看到这样子的路的画面，那人就会采取某一种Action，比如说他就会踩刹车，或者是踩油门等等，欸你有这样子的一堆的纪录，我们不是直接就用Supervised Learning，来Learn我们的Agent就好了吗。你就说，你已经有人类给的资料说，看到 $s_1$ 这样的画面，最好的行为就是 $a_1$ ， $s_2$ 这样的画面，人类的行为就是 $a_2$ ，你已经有这样子的训练资料，那你就直接叫机器，去模仿人类的行为就好啦，今天机器给它看 $s_i$ ，然后它输出 $a_i$ ，你要让它的Action $a_i$ ，跟人类的Action $a_i$ 越接近越好，你就让机器去模仿人类的行为。那没错，当你有这个Expert的示范的时候，这是一个做法，那这种做法呢，叫做Behavior的Cloning，就是去复制人类的行为。

## Yes, also known as *Behavior Cloning*



但是光是让机器去复制人类的行为，有可能有什么样的问题呢？

一个可能的问题是，因为人类跟机器，他们可能可以观察到的 $s$ ，会是不一样的。什么意思呢，举例来说，假设我们一样要叫机器学习开自驾车，那它是跟人类的Expert去学，人类的Expert在转弯的时候，都可以轻松地转过这个弯道，没有人出车祸，没有人任何马路三宝，每个人都顺利地，每一个Expert都可以顺利地向右转，所以对机器来说，它从来没有看过失败的状态，它从来没有看过一个车子快要撞墙的状态，如果它从来没有看过一个车子快要撞墙的状态，它训练资料里面，就没有这种东西，它就不知道，假设车子快要撞墙的时候，应该要怎么处理，因为所有Expert都太厉害了，根本就不会让车子靠近墙边，根本就不会犯这种错误，那机器就不会学到说，在这一种人类平常不会经历的状态下，到底应该要怎么处理。所以这是第一个可能遇到的问题。

第二个Behavior Cloning可能遇到的问题是，虽然人类在开车的时候可能很厉害，但也许不是每一个行为，机器都需要亦步亦趋地去完全模仿，也许有一些行为需要模仿，有一些行为是人类个人的特质，根本不需要模仿，但是机器，它不知道什么行为该模仿，什么行为不该模仿，它只能完全复制人类的行为。那什么叫做完全复制人类，完全复制老师的行为呢，以下就是一个完全复制老师行为的一个例子，那以下这个影片，出自TheBigBangTheory，我就播给大家看一下，[https://www.youtube.com/watch?v=75rZwxKBAf0&list=PLJV\\_el3uVTsMhtt7\\_Y6sgTHGHp1Vb2P2J&index=33&ab\\_channel=Hung-yiLee](https://www.youtube.com/watch?v=75rZwxKBAf0&list=PLJV_el3uVTsMhtt7_Y6sgTHGHp1Vb2P2J&index=33&ab_channel=Hung-yiLee) 9:30开始。

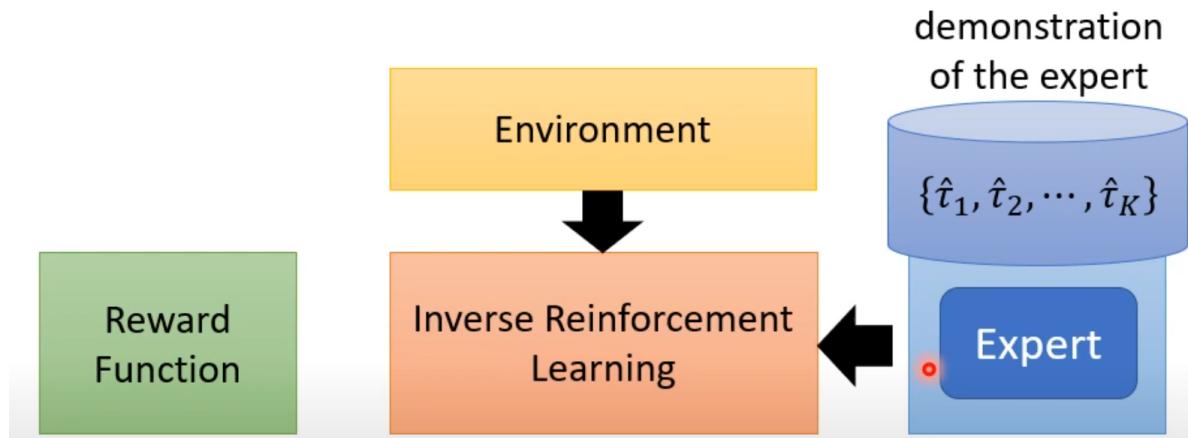
好，所以刚才我们就是看到了一个，Behavior Cloning的例子，那对机器来说，它就跟穿绿色衣服的Sheldon一样，它的老师教什么，它就会有一模一样的示范，那它不知道什么东西是需要学的，什么东西是不需要学的。那如果机器，只是跟它老师采取一模一样的行为，也许还没有什么问题，因为它跟老师采取一模一样的行为，就跟老师做一模一样的事情，虽然它可能会有一些多余的行为，但也许是无伤大雅。但是我觉得更惨的另外一个状况是，机器的能力，可能是有限的，今天也许机器没有办法完全模仿老师的行为，它只能选择部分的行为，来进行模仿。就好像说有一个人呢，他想要变得跟贾伯斯一样，然后他就去看了贾伯斯传，然后把贾伯斯传里面，贾伯斯的所有特质都列出来，比如说列了20个，包括有创意，然后脾气暴躁，然后不修边幅等等，但他觉得说，这些特质太多了，他只决定模仿一个而已，因为他能力有限，他只模仿一个而已，那如果他选择有创意也许是好的，但也许他选到的是不修边

幅，那就没有什么用。所以假设机器能力是有限的情况下，Behavior Cloning，也许会造成更大的问题，所以怎么办呢？

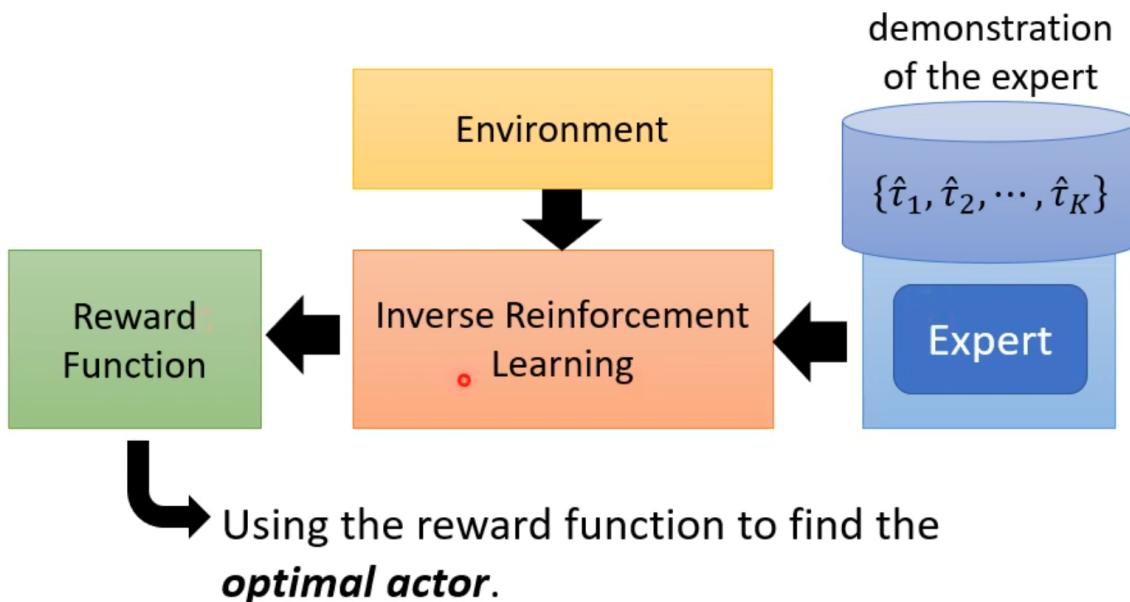
## Inverse Reinforcement Learning

有另外一个技术，叫做Inverse的Reinforcement Learning。接下来Inverse Reinforcement Learning，就是要让机器自己来订Reward啦，那怎么做呢，我们先看原来的Reinforcement Learning，是怎么运作的。原来的Reinforcement Learning是，我们有Reward，有环境，然后呢，用RL的Algorithm，跟环境还有Reward互动，然后你就学出一个Actor，但现在，我们没有Reward了，我们有的只有Expert专家的示范。

## Inverse Reinforcement Learning



我们现在要做的事情，是一个叫，Inverse Reinforcement Learning的Algorithm，它是跟原来的Reinforcement Learning是相反的。它要做的事情，并不是根据Reward去学习，而是从Expert的Demonstration，还有Environment，去反推Reward应该长什么样子。也就是这边的RewardFunction，是学出来的。那学出一个Reward Function以后，你就可以直接用一般的Reinforcement Learning，来学你的Actor。



所以在Inverse Reinforcement Learning里面，我们的概念就是，本来不知道Reward Function，从Expert的示范，去反推Reward Function应该长什么样子，有了Reward Function以后，我们就可以再训练一个Optimal的Actor，去根据这些Reward Function，来进行学习。

那讲到这边，也许有人会有的疑惑是，欸这个Reward Function是学出来的，它会不会太简单了呢，我们会不会没有办法学出，非常复杂的Reward Function呢。但是简单的Reward Function，并不代表，根据这个Reward Function学出来的Actor，一定也会是简单的。举例来说，对一个人类而言，也许人类的Reward，是非常简单的。也许人类的Reward Function就只有一条，就是活下来，但是是不是这样子，就是一个见仁见智的问题啦。那也许人类的Reward Function只有，活下来这件事，但是光是人类想活下来这件事情，就可以让人类的行为有千变万化，所以简单的Reward Function，并不代表你一定会学出简单的Actor。有可能简单的Reward Function，但学出来的Actor，仍然是复杂的，那这个是Inverse Reinforcement Learning。

Inverse Reinforcement Learning的基本概念，是什么呢，怎么找出Reward Function呢？这边最基本的概念就是，老师的行为，是最棒的。但是我这边要强调一下所谓最棒，并不代表，你要完全去模仿老师的行为，而是你假设老师的行为，可以取得最高的Reward。那老师的行为，可以取得最高的Reward这个假设，跟完全模仿老师的行为，这两件事情并不是等价的。也许我们看完这个Algorithm，你会更清楚我想表达的意思。

- Principle: ***The teacher is always the best.***

好，现在呢，我们有一个Actor，它一开始是什么都不会，然后呢，在每一个Iteration里面，这个Actor会去跟环境进行互动，学习搜集一些Actor自己的Trajectories。然后接下来呢，我们要定义一个Reward Function，这个Reward Function怎么定义呢？这个Reward Function定义的条件，这个Reward，Learn这个Reward Function的条件是，今天老师的行为，得到的Reward，必须要高于学生的行为，就老师也有跟环境互动，我们得到一堆老师的Demonstration，我们得到一堆老师的Trajectory。当你用你Learn出来的Reward Function，去计算老师的Trajectory的时候，我们要订一个Reward Function，我们要订一个Reward Function，这个Reward Function，去评估老师的Trajectory的时候，要给比较高的分数，去评估Actor的Trajectory的时候，要给它比较低的分数。然后接下来呢，你再去更新你的Actor，你要去重新训练你的Actor，更新你Update，更新你的Actor的参数，让它去Maximize我们会得到的Reward，然后接下来呢，就反覆执行这个步骤，你有新的Actor，它会有新的Trajectory，你再更新一次Reward Function，让这个Reward Function评估，这个老师的分数比较高，评估Actor的分数比较低，然后呢，Actor呢，再想办法去Maximize Reward Function，然后就反覆这个循环，最终你就会得到一个Reward Function。那这个，就是我们用Inverse Reinforcement Learning，Learn出来的Reward Function。

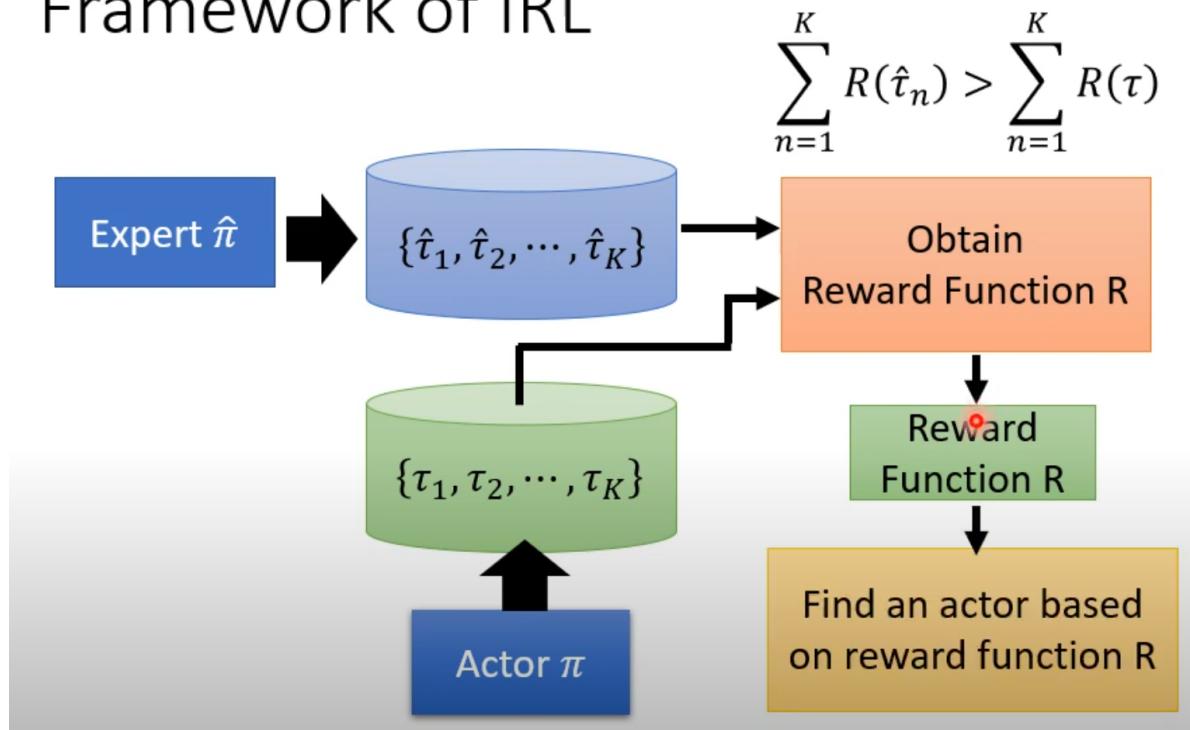
- Principle: ***The teacher is always the best.***

- Basic idea:

- Initialize an **actor**
- In each iteration
  - The **actor** interacts with the **environments** to obtain some trajectories.
  - Define a **reward function**, which makes the trajectories of the teacher better than the **actor**.
  - The **actor** learns to maximize the **reward** based on the new **reward function**.
- Output the **reward function** and the **actor** learned from the reward function

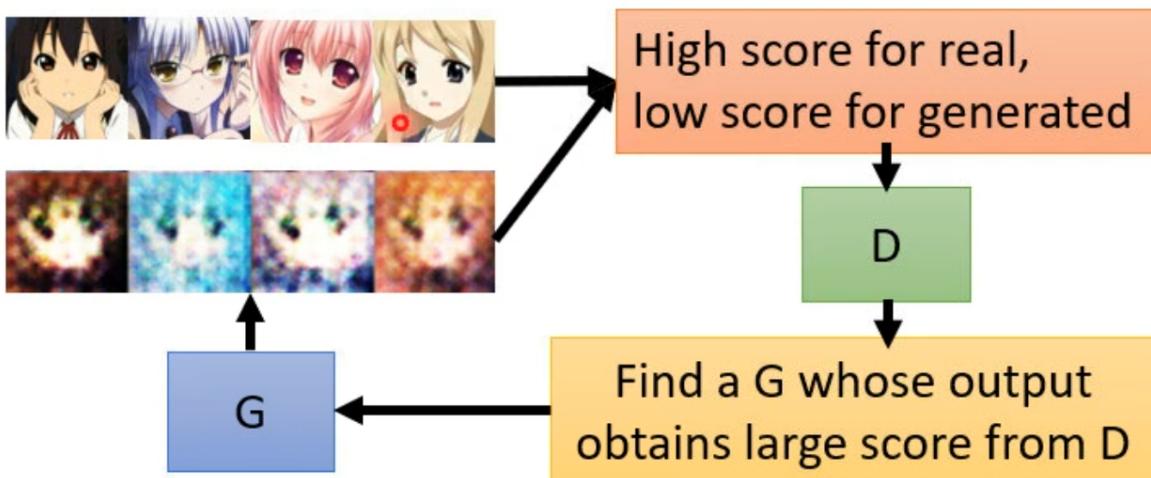
好那如果刚才那个Algorithm，你没有看的很懂的话，那这边是用图像化的方法，来讲一下Inverse Reinforcement Learning，那它的缩写呢，是IRL。好那现在呢，有Expert的Demonstration，写成 $\hat{\pi}$ ，有Actor跟环境的互动，写成 $\tau$ 。那接下来你要定一个Reward Function，这个RewardFunction呢，会给出 $\hat{\pi}$ ，也就是Expert的Demonstration比较高的分数，给 $\tau$ ，也就是你的Actor的Trajectory比较低的分数。

## Framework of IRL



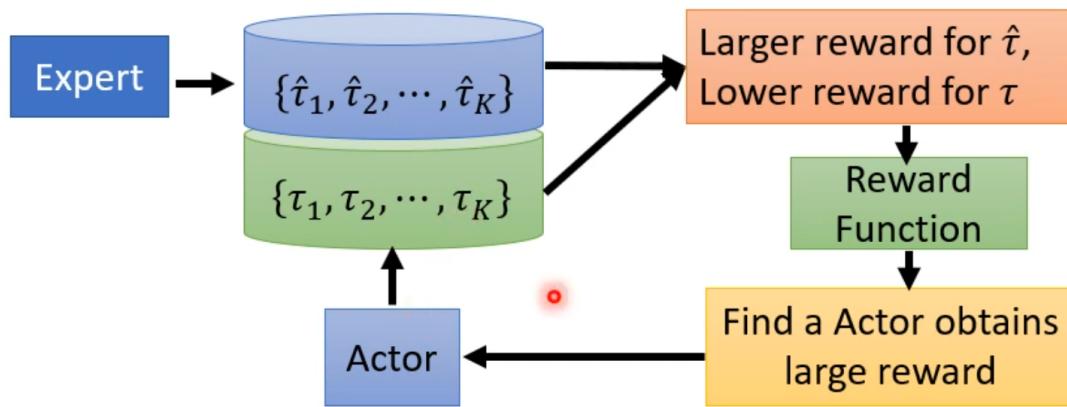
那有了这一个R以后，你再去训练你的Actor，去Maximize这个Reward Function，怎么训练Actor去Maximize这个，刚学出来的RewardFunction呢？这边你就要透过，ReinforcementLearning的方法。接下来，你有了新的Actor，新的Actor有新的行为，但这些新的行为，仍然要得到比老师低的分数，你会去更新你的Reward Function，让老师得到的分数，仍然高过于Actor得到的分数。然后就反覆反覆这一个回圈，反覆反覆这个循环，最终，你就可以把一个Reward Function Learn出来。

那这整个Framework，你听起来有没有觉得有点熟悉呢？我们可以把Actor，想像成是GAN，Generative Adversarial Network，里面的Generator。把RewardFunction，想像成是GAN里面的Discriminator，我们来很快复习一下GAN的Framework。



在GAN的Framework里面，你有一个Generator，它会产生比较差的图片，然后有一个Discriminator，它要想办法给真正的图片高分，Generator产生的图片低分，然后呢，你的这个Generator，会去想办法骗过Discriminator，产生新的图片，Discriminator又会Update它的参数，想办法去，评价好的图片跟Generator产生出来图片的差别。然后这个Discriminator，跟这个Generator，就会反覆地Update，那这个是GAN。

## IRL



Inverse Reinforcement Learning跟GAN，其实根本就是一样的东西。只是把Generator，跟Discriminator的名字换掉而已。Actor产生一些行为，然后你要去订一个Reward Function，给Expert的Trajectories高分，给Actor的Trajectories低分，然后接下来，Actor想办法去，在这个RewardFunction得到高分，那有了新的Actor，有了新的行为，RewardFunction又会被Update，想办法给Expert高分，给Actor低分，所以RewardFunction，完全可以对应到Discriminator，Actor可以对应到Generator。所以你会发现，GAN跟IRL，Inverse Reinforcement Learning。它们有异曲同工之妙，好像是同一个Framework，用不同的方法，不同的角度来描述。好，那像IRL这种方法，常常被用来训练机械手臂。那过去，在如果你不是用Reinforcement Learning，来训练机械手臂的话，可能看起来，是什么样子呢，以下又是从TheBigBangTheory，里面撷出来的一段[https://www.youtube.com/watch?v=75rZwxKBAf0&list=PLJV\\_el3uVTsMhtt7\\_Y6sgTHGHp1Vb2P2J&index=33&ab\\_channel=Hung-yiLee](https://www.youtube.com/watch?v=75rZwxKBAf0&list=PLJV_el3uVTsMhtt7_Y6sgTHGHp1Vb2P2J&index=33&ab_channel=Hung-yiLee) 19分36秒

好那这个影片想要告诉大家的事情是说，假设你今天想要用写程式的方法，来操控一个机械手臂，虽然对人来说，我们要伸自己的手臂来做什么事情，都是一件很简单的事情，但是你要把这么简单的行为程式化，把它写成程式，去操控机械手臂的每一个关节，做出某一些指定的动作，往往不是那么容易。那这个时候，你就可以使用，Inverse Reinforcement Learning的技术，你就示范给机器看，示范给它一次你要它做的行为，看看它能不能就借此学会，我们要它做的行为。所以以下，就是用Inverse Reinforcement Learning，其中的某一个技术达到的结果。

[https://www.youtube.com/watch?v=75rZwxKBAf0&list=PLJV\\_el3uVTsMhtt7\\_Y6sgTHGHp1Vb2P2J&index=34&ab\\_channel=Hung-yiLee](https://www.youtube.com/watch?v=75rZwxKBAf0&list=PLJV_el3uVTsMhtt7_Y6sgTHGHp1Vb2P2J&index=34&ab_channel=Hung-yiLee) 21分50秒

我这边是不是有点卡顿，我跳出来再跳进去好了，好那我们继续吧，好那就播一下这个影片，在教机器摆盘子，先示范给它看，这边会示范个20次，那这个是示范，这个是教机器那个倒东西，然后这个也是示范20次。好那这个影片是想要告诉大家说，未来我们可能可以用Demonstrate的方法，来教机器事情，好那事实上呢，如果你要教机械手臂一些行为，现在还有一个更潮的做法。那这个更潮的做法呢，是你直接给机器一个画面，然后让机器呢，做出这个画面中的行为，那这个部分我们就不细讲，我这边就是列举了一篇，NIPS的Paper，跟一篇ICML的Paper给大家参考。

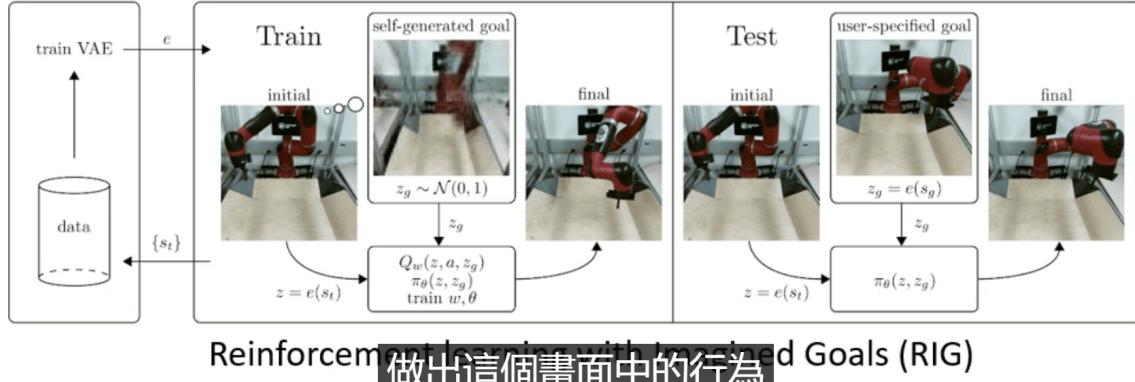
# To Learn More ...

Visual Reinforcement Learning with Imagined Goals, NIPS 2018

<https://arxiv.org/abs/1807.04742>

Skew-Fit: State-Covering Self-Supervised Reinforcement Learning,  
ICML 2020

<https://arxiv.org/abs/1903.03698>



那它的基本概念是说，就给机器一个画面，告诉它说，你就去达到这个目标，然后机器就会自己想办法达到目标。那怎么训练机器有这个能力，看到一个画面，就知道怎么达到目标呢，这个训练的过程也非常地有意思，这个训练过程是，机器会自己创造目标。它自己在心里呢，想像一些画面，然后呢，再想办法去达到这些画面，这个就好像是说，有人告诉你说，欸那你要念一个博班，你要拿一个博士学位，那你就会去想办法呢，拿到这个博士学位。那中间的过程是怎么样，不知道，你要自己想办法去Figure Out。可是怎么训练自己，有拿到博士学位的能力呢，你就会自己给自己先设定一些目标。比如说你先设定说，我要成为一个YouTuber，然后要做做做，做一些事情，然后成为一个YouTuber，你就知道说，嗯我有达成目标的能力，那再设定一些别的，别的各式各样的目标，然后都想办法去达成它，就可以培养自己达成目标的能力。那之后有人告诉你说，你现在的目标是拿一个博士学位，那你就会知道要怎么拿一个博士学位这样。

## Concluding Remarks

What is RL? (Three steps in ML)

Policy Gradient

Actor-Critic

Sparse Reward

No Reward: Learning from Demonstration

好那这个有关RL的部分呢，大概就，有关RL的部分呢，就讲到这边。

Q：好，有同学问说，使用IRL这类的方法的话，是不是就没办法找到比人类更好的方法，有没有办法让机器青出于蓝，欸我觉得这是一个好问题。

A：使用IRL这个做法，我们要注意一下，机器并不是去完全模仿人类的行为，所以机器的Solution，跟人的Solution，不一定会是一样的。所以，所以今天，如果我们要让机器，它得到的结果比人类更好的话，我觉得有一个可能的方法是，我们先用IRL，先Learn出一个Reward Function，然后在这个Reward Function上面，再加上额外的限制。举例来说，我们刚才看到的例子都是，你今天有示范给机器看，然后机器就可以Learn出一个Reward Function，知道说，你现在要叫它做的事情就是，摆盘子，那我们现在可以再加上新的Reward说，你摆盘子的速度要越快越好，如果你摆盘子的速度也很快的话，那你一样，你会得到额外的Reward，那这样也许就有机会让机器学一些，原来人类示范的时候，做不到的事情，所以我觉得，IRL还是有机会做得比人类更好的。