Reinforcement Learning P1 ： Basics

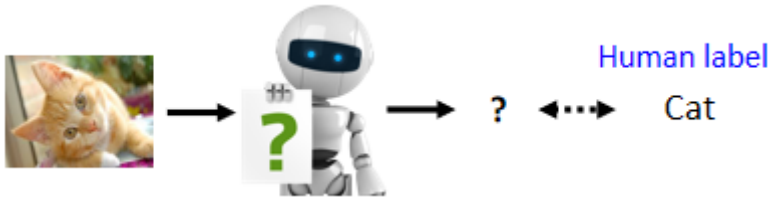
那这一堂课啊,我们要讲的是,Deep Reinforcement Learning,也就是 RL,那我想这个 RL 啊,Reinforcement Learning 啊,大家一定一点都不陌生,因为你知道很多很潮的应用,AlphaGo 等等,它背后呢,用的就是 RL 的技术,那 RL 可以讲的技术啊,非常非常地多,它不是在一堂课裡面可以讲得完的,我甚至觉得说,如果有人要把它开成一整个学期的课,可能也是有这麼多东西可以讲

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来,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

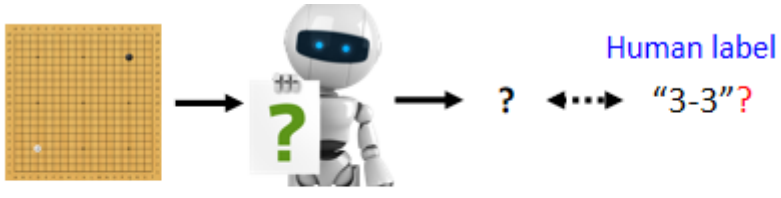


那在多数这门课讲到目前为止的技术,基本上都是基于 Supervised Learning 的方法,就算是我们在讲 Self Supervised Learning 的时候,我们其实也是,很类似 Supervised Learning 的方法,只是我们的 Label,不需要特别僱用人力去标记,它可以自动产生

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,但它是不是最好的答案呢,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

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

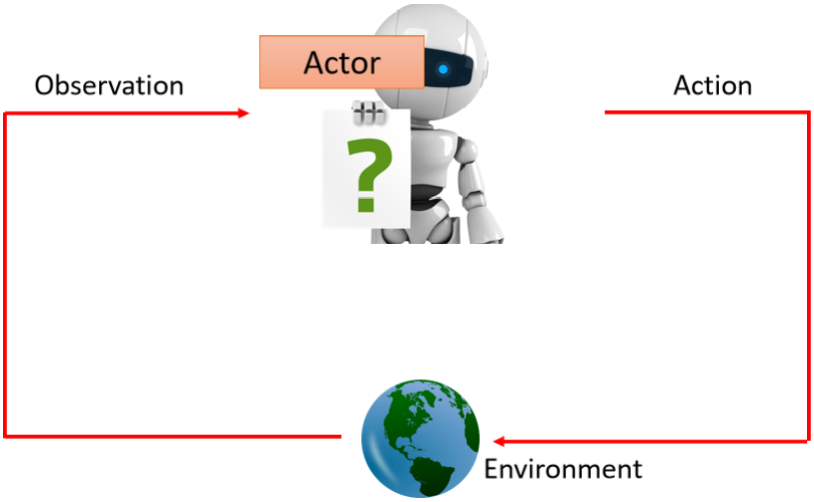
那我们这边选择了一个比较不一样的切入点,我要告诉你说,虽然如果你自己读 RL 的文献的话,你会觉得,哇 RL 很複杂哦,跟一般的 Machine Learning 好像不太一样哦,但是我这边要告诉你说,**RL 它跟我们这一门课学的 Machine Learning,是一样的框架**

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

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**



- 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 采取 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

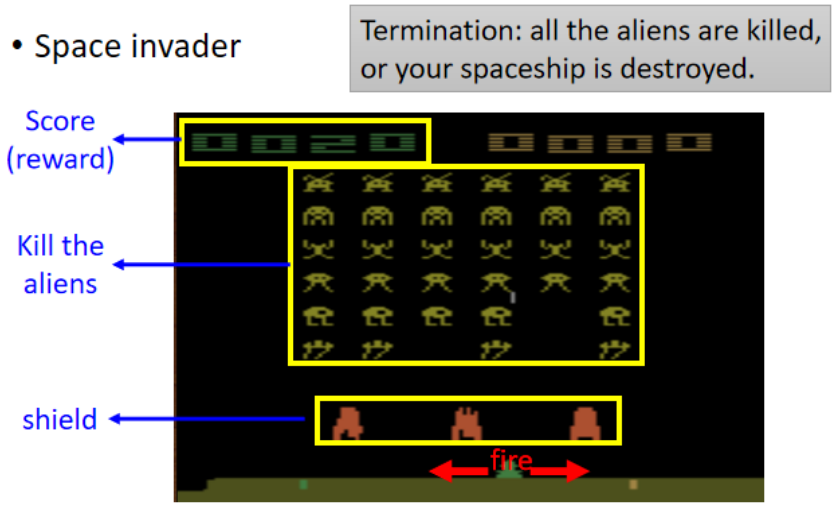
所以 **Actor 本身啊,它就是一个 Function**,其实 Actor,它就是我们要找的 Function,这个 Function 它的轮入,就是环境给它的 Observation,输出就是这个 Actor 要采取的 Action,而今天在这个互动的过程中呢,**这个 Environment,会不断地给这个 Actor 一些 Reward**,告诉它说,你现在采取的这个 Action,它是好的还是不好的

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



- 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,你可以採取的行为,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

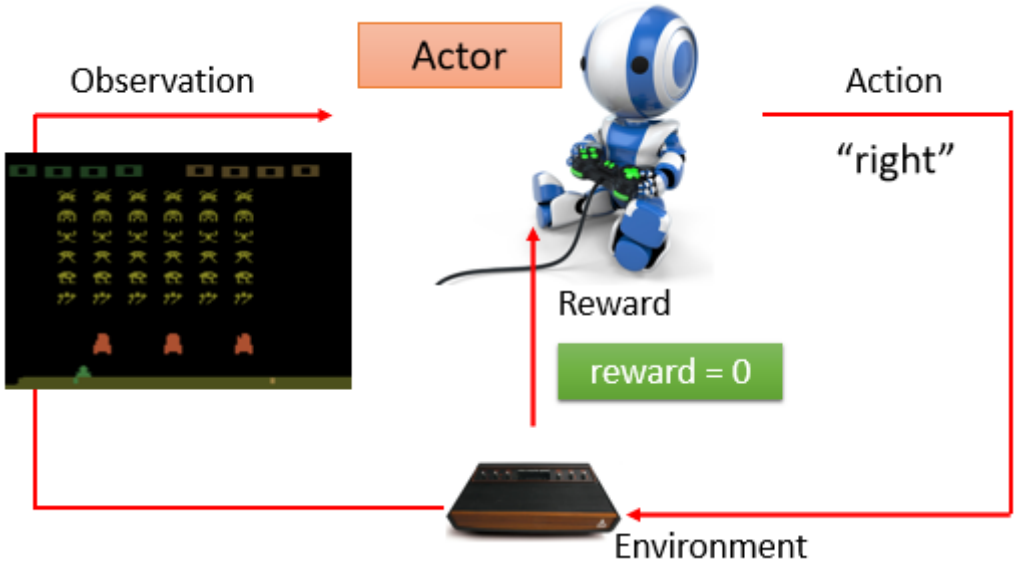
这些外星人

- 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

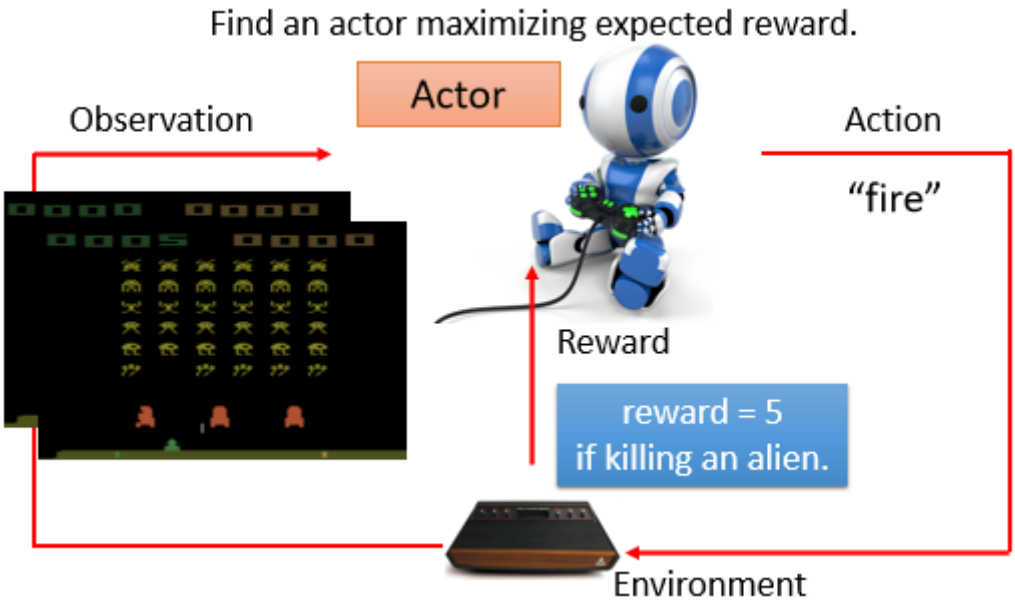


所以对 Actor 来说,它看到的,其实就跟人类在玩游戏的时候,看到的东西是一样的,就看到一个游戏的画面

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



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

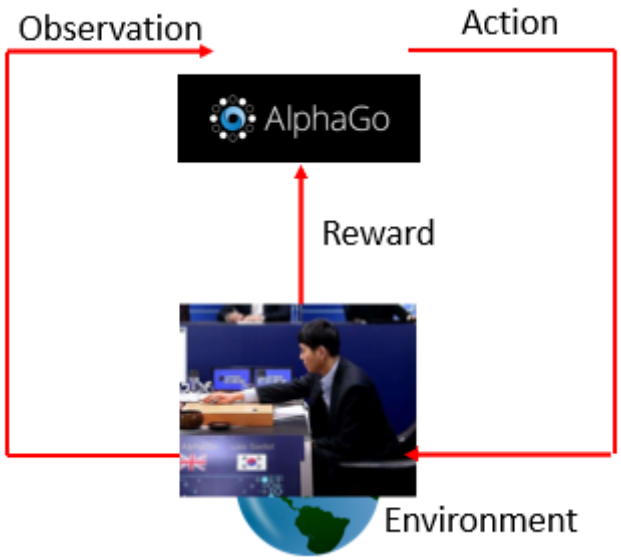
那这个呢,就是拿 Actor 去玩,玩这个 Space Invader 这个游戏的状况,好 那这个 Actor 呢,它想要学习的是什么呢,我们在玩游戏的过程中,会不断地得到 Reward,那在刚才例子裡面,做第一个行为的时候,向右的时候得到的是 0 分,做第二个行为,开火的时候得到的是 5 分,那接下来你採取了一连串行为,都有可能给你分数

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

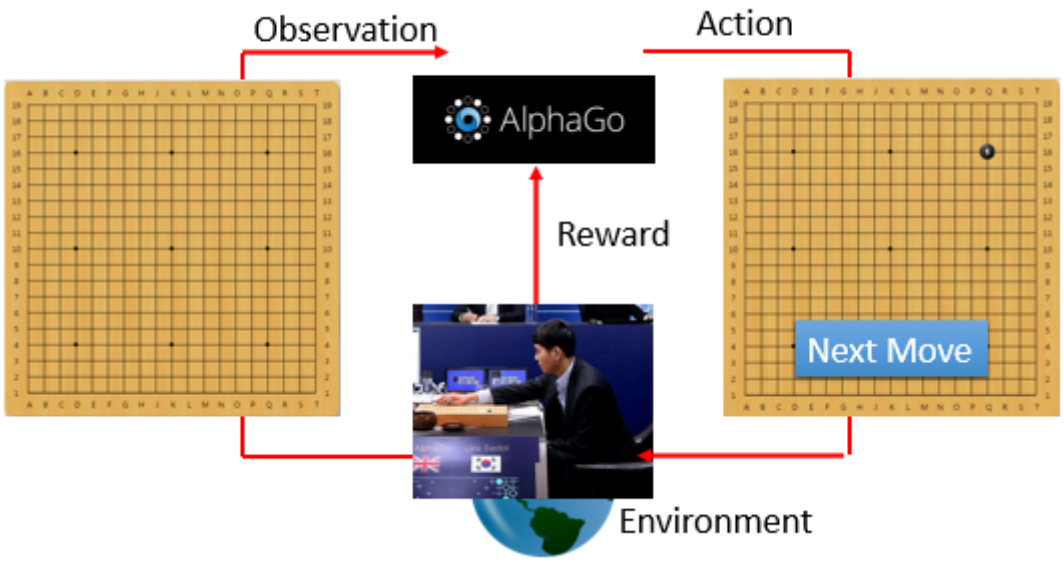
那如果今天你要让机器来下围棋,那你的 Actor 就是就是 AlphaGo,那你的环境是什么,你的环境就是 AlphaGo 的人类对手



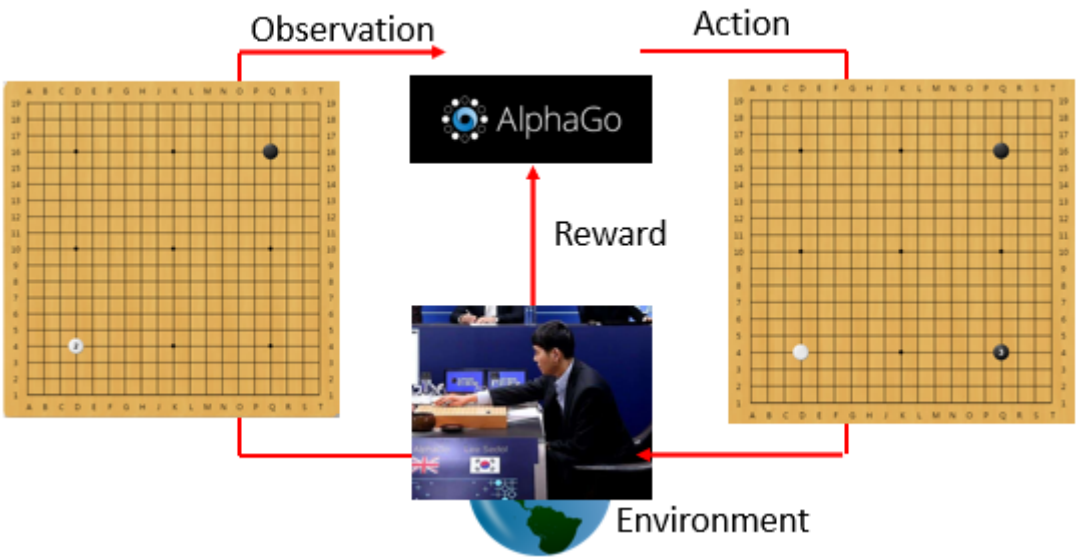
你的 Actor 的输入就是棋盘,棋盘上黑子跟白子的位置,那如果是在游戏的一开始,棋盘上就空空的,空空如也,上面什么都没有,没有任何黑子跟白子

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 19×19 个可能性,那这 19×19 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



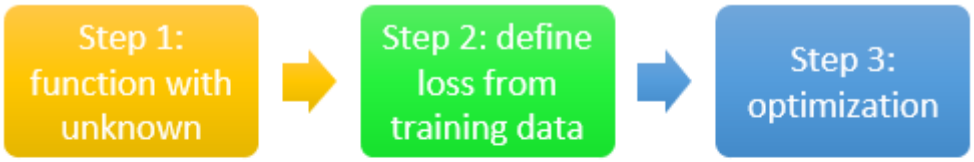
你就可以让机器做下围棋这件事情,好 那在这个,在这个下围棋这件事情裡面的 Reward,是怎麼计算的呢

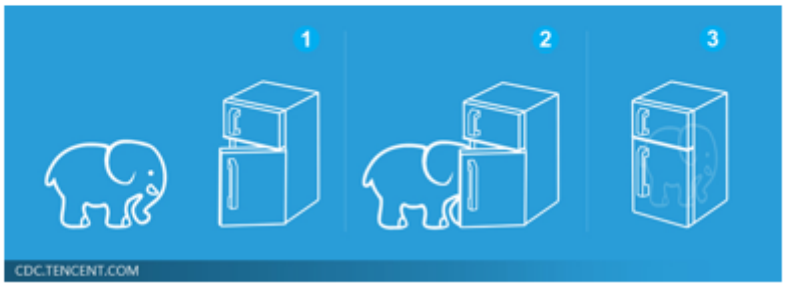
在下围棋裡面,你所採取的行为,几乎都没有办法得到任何 Reward,在下围棋这个游戏裡,在下围棋这件事情裡面呢,你会定义说,如果赢了,就得到 1 分,如果输了就得到 -1 分

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,它们之间的关系是什么





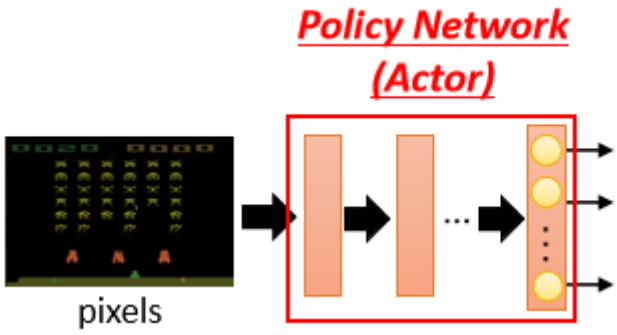
开学第一堂课就告诉你说,Machine Learning 就是三个步骤

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

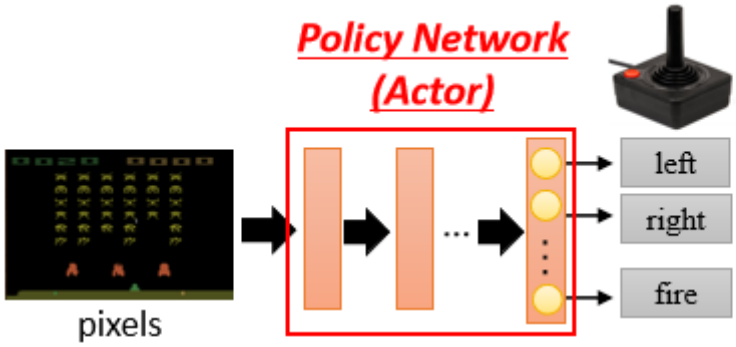
而 RL 其实也是一模一样的三个步骤

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,那今天我们都知道要用 Network,来当做这个 Actor,那这个 Network,其实就是一个很複杂的 Function



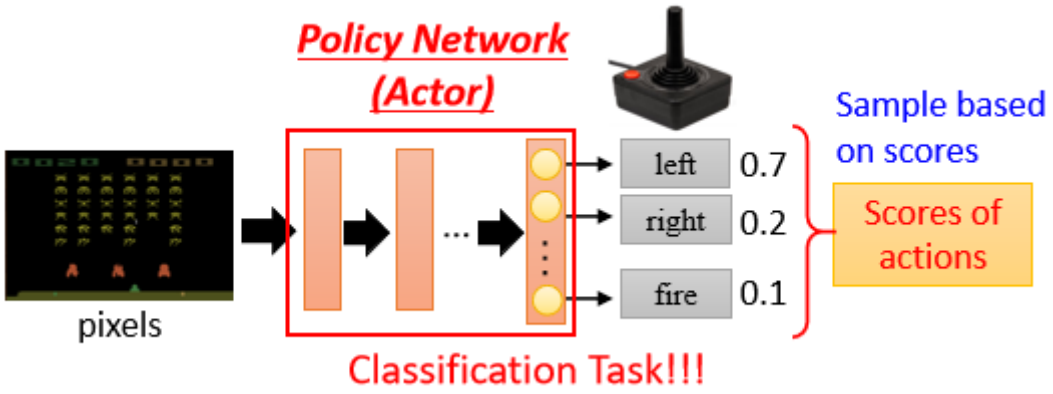
- Input of neural network: the observation of machine represented as a vector or a matrix
- Output neural network : each action corresponds to a neuron in output layer

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,你也会让它是 1



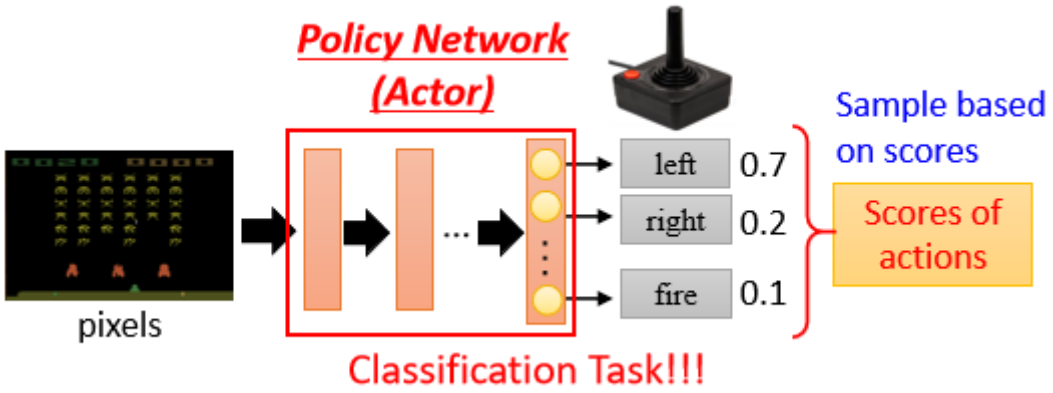
那至于这个 **Network 的架构呢,那你就可以自己设计了**,要设计怎麼样都行,比如说如果输入是一张图片,欸 也许你就会想要用 CNN 来处理

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,所以我们是让,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 那麼複杂,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 的架构是你可以自己设计的,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

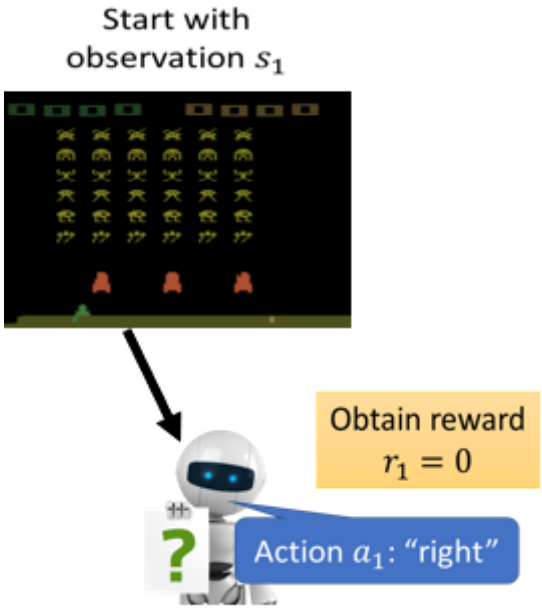
那你可能会问说,为什么不是用argmax呢,为什么不是看 Left 的分数最高,就直接向左呢,你也可以这麼做,但是在助教的程式裡面,还有多数 RL 应用的时候 你会发现,我们都是採取 Sample,

採取 **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”

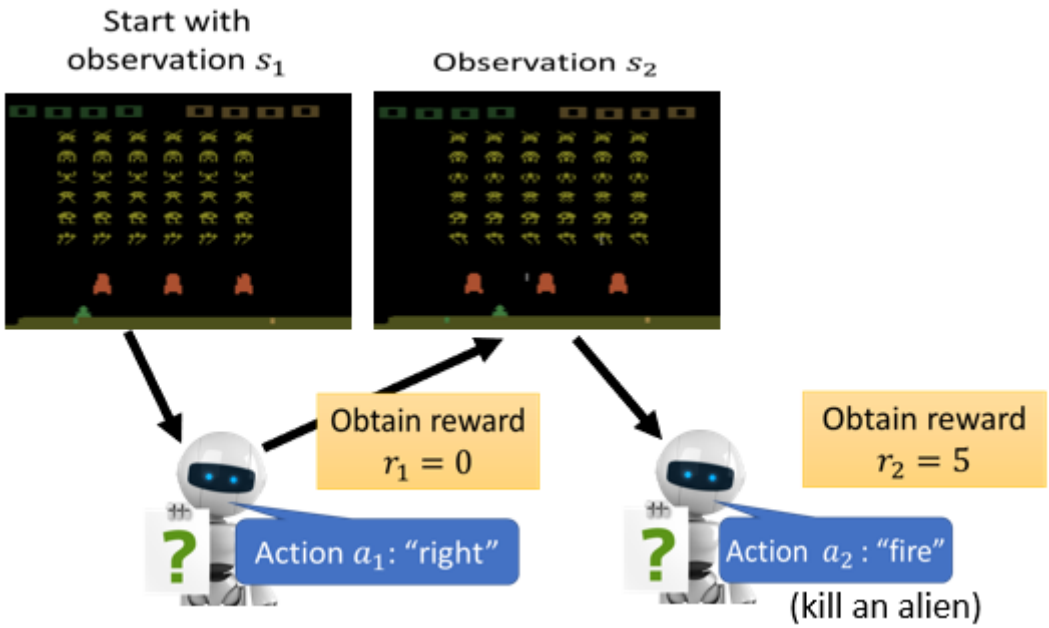
然后接下来第二步,我们要定义 Loss,在 RL 裡面,我们的 Loss 长得是什么样子呢,我们再重新来看一下,我们的机器跟环境互动的过程,那只是现在用不一样的方法,来表示刚才说过的事情



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

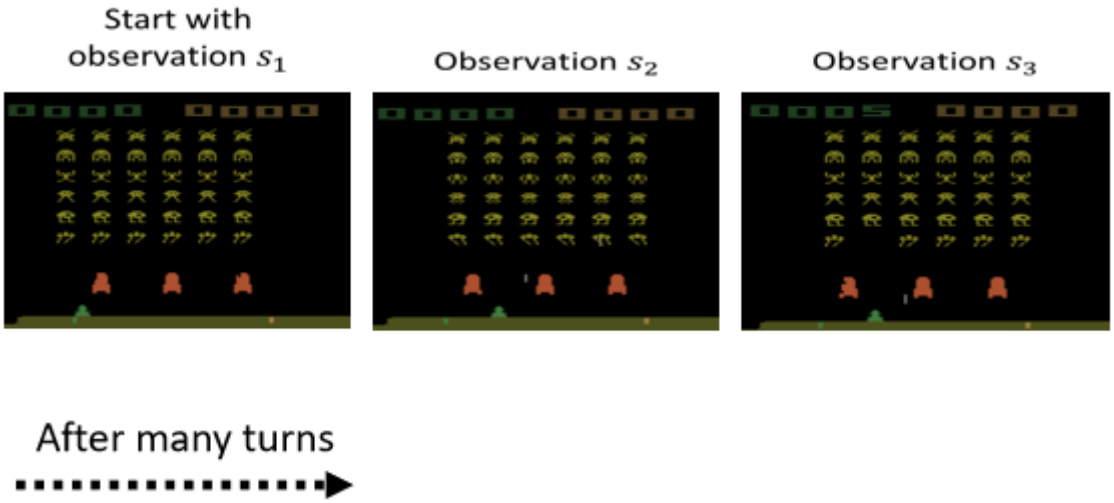
你的 Actor 那就输出了一个 Action,比如说向右,输入的游戏画面呢,我们叫它 s_1,然后输出的 Action 呢,就叫它 a_1

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



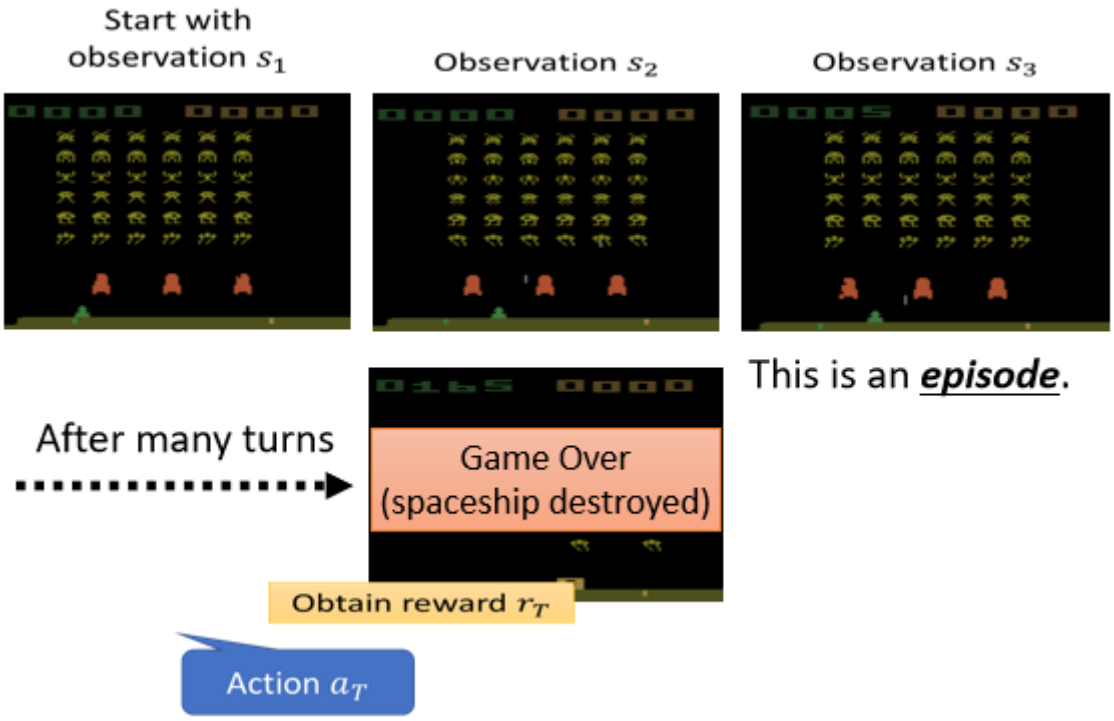
采取向右以后,会看到新的游戏画面,这个叫做 s_2,根据新的游戏画面 s_2,你的 Actor 会采取新的行为,比如说开火,这边用 a_2,来表示看到游戏画面 s_2 的时候,所採取的行为

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

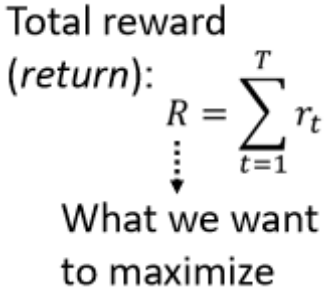


接下来你会看到新的游戏画面,那机器又会采取新的行为,那这个互动的过程呢,就会反覆持续下去,直到机器在采取某一个行为以后,游戏结束了,那什么时候游戏结束呢,就看你游戏结束的条件是什么嘛

举例来说,采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,机器会采取非常多的行为,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,



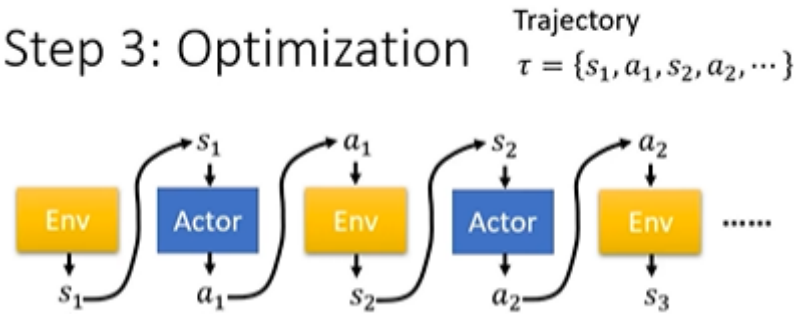
那这个 Total Reward 呢,就是从游戏一开始得到的 r_1,一直得,一直加,累加到游戏最后结束的时候,得到的 rt,假设这个游戏裡面会互动,T 次,那麼就得到一个 Total Reward,我们这边用 R,来表示 Total Reward,其实这个 Total Reward 又有另外一个名字啊,叫做 Return 啦,你在这个 RL 的文献上,常常会同时看到 Reward 跟 Return,这两个词会出现,那 Reward 跟 Return 其实有点不一样,Reward 指的是你採取某一个行为的时候,立即得到的好处,这个是 Reward,把整场游戏裡面所有的 Reward 通通加起来,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

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

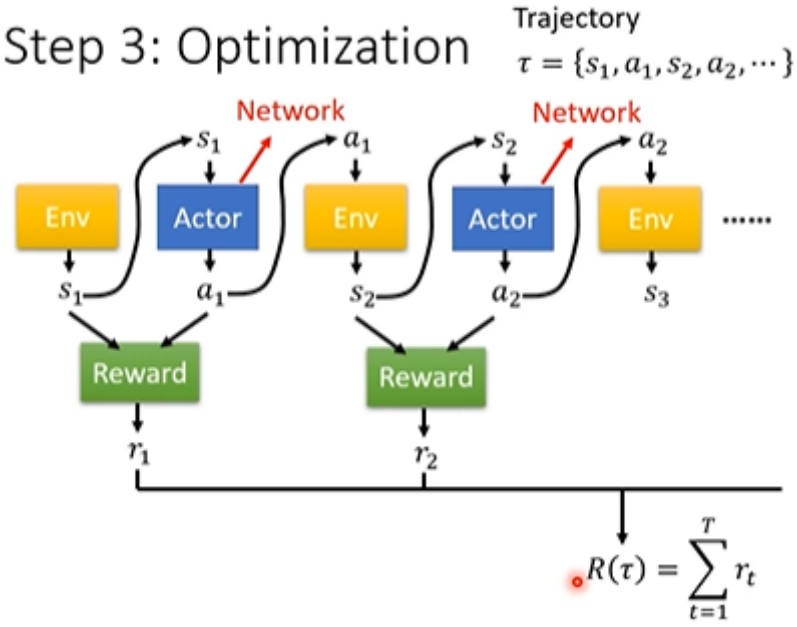


这个是你的环境,你的环境呢,输出一个 Observation,叫做 s_1

- 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 τ来表示 Trajectory



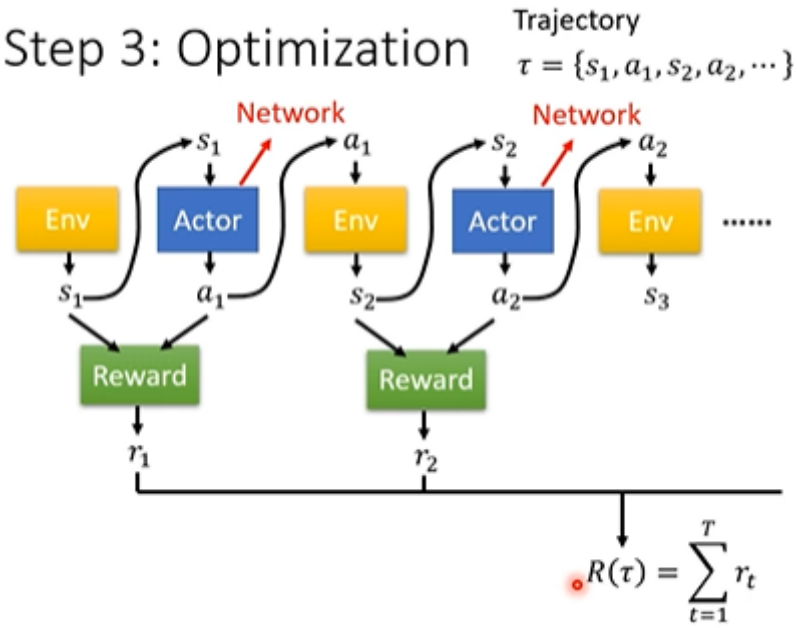
那根据这个互动的过程,Machine 会得到 Reward,你其实可以把 Reward 也想成是一个 Function,我们这边用一个绿色的方块来代表,这个 Reward 所构成的 Function

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

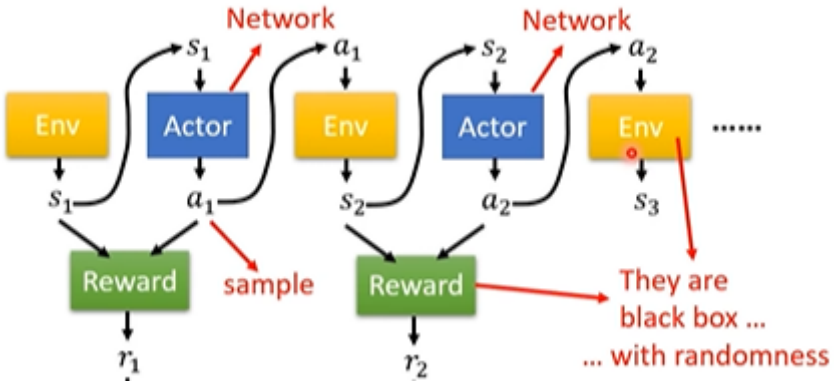
这个 Optimization 的问题是这个样子,你要去找一个 Network,其实是 Network 裡面的参数,你要去 Learn 出一组参数,这一组参数放在 Actor 的裡面,它可以让这个 R 的数值越大越好,就这样,结束了,整个 Optimization 的过程就是这样,你要去找一个 Network 的参数,让这边产生出来的 R 越大越好



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**



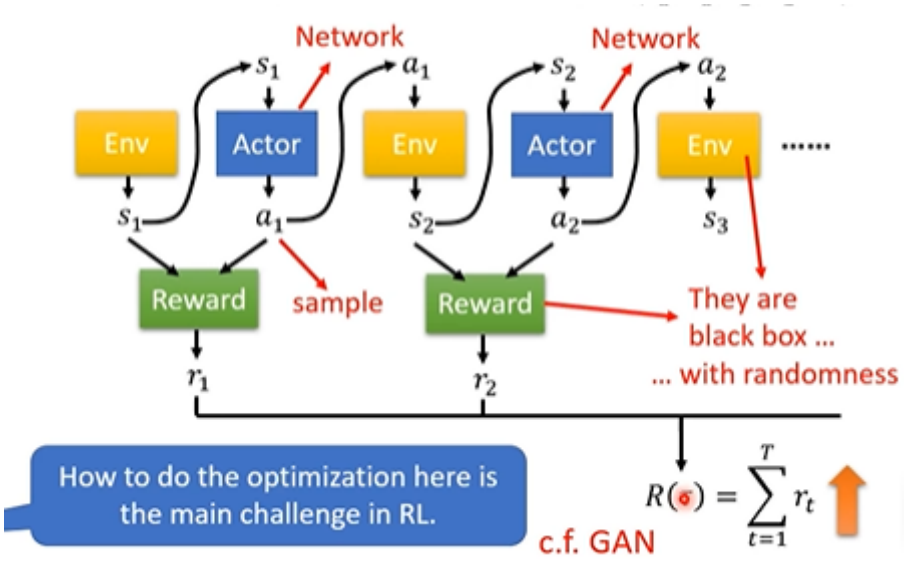
这个 a_1 **它是用 Sample 产生的**,你定同样的 s_1 每次产生的 a_1 不一定会一样,所以假设你把 Environment Actor 跟 Reward,合起来当做是一个巨大的 Network 来看待,这个 Network 可不是一般的 Network,**这个 Network 裡面是有随机性的**,这个 Network 裡面的某一个 Layer 是,每次产生出来结果是不一样的,**这个 Network 裡面某一个 Layer 是,它的输出每次都是不一样的**

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,但是到底是怎麼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,到底游戏机会怎麼样回应,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,你的对手会怎麼样回应,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,我们怎麼解这一个 Optimization 的问题,怎麼找到一组 Network 参数,可以让 R 越大越好

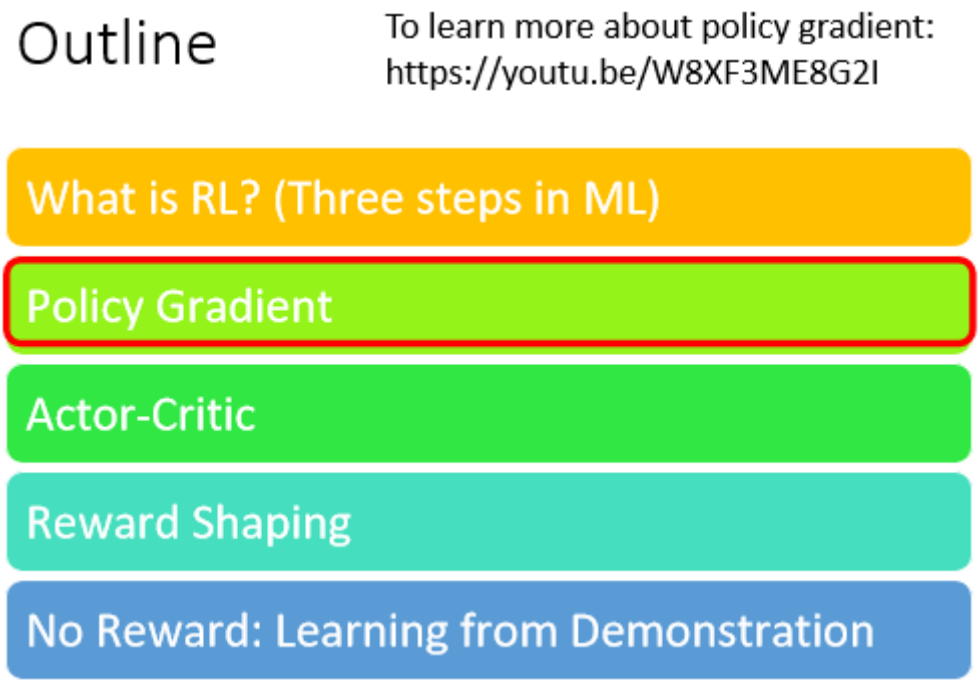
其实你再仔细想一想啊,这整个问题**跟 GAN 其实有异曲同工之妙**,它们有一样的地方,也有不一样的地方

- 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,我们也可以说这个 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 的问题裡面,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,在你怎麼 Minimize Loss,也就怎麼 Maximize Reward 的时候,跟之前我们学到的方法是不太一样的

Policy Gradient

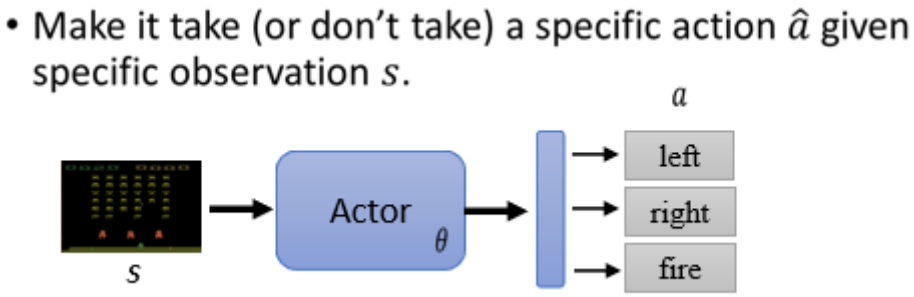
接下来啊,我们就要讲一个拿来解 RL,拿来做 Optimization 那一段常用的一个演算法,叫做 Policy Gradient



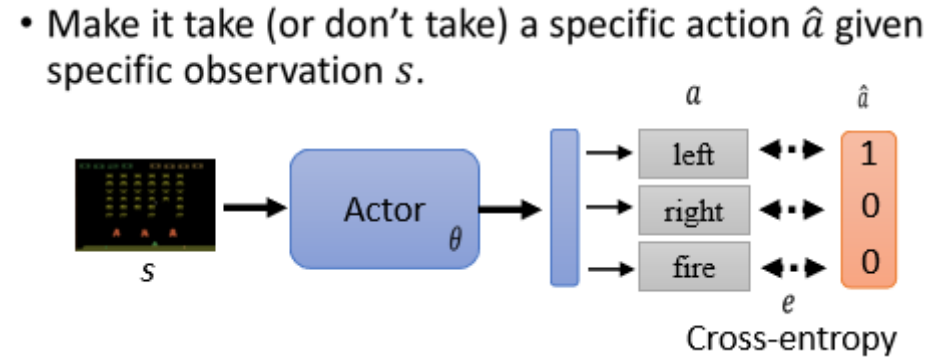
那如果你真的想知道,Policy Gradient 是哪裡来的,你可以参见过去上课的录影,对 Policy Gradient 有比较详细的推导,那今天我们是从另外一个角度,来讲 Policy Gradient 这件事情

How to control your actor

那在讲 Policy Gradient 之前,我们先来想想看,我们要怎麼操控一个 Actor 的输出,我们要怎麼让一个 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,我们怎麼让一个 Actor,它的输入是 s 的时候,它就要输出 Action $\hat{a}$ 呢

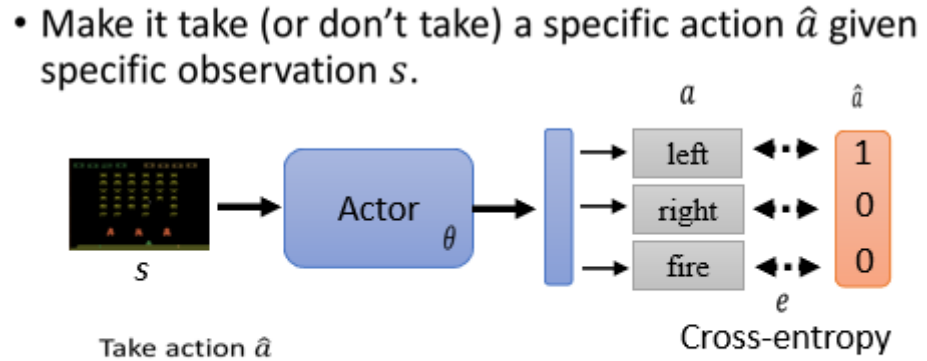


那你其实完全可以把它**想成一个分类的问题**,也就是说假设你要让 Actor 输入 s,输出就是 $\hat{a}$,假设 $\hat{a}$ 就是向左好了,假设你要让,假设你已经知道,假设你就是要教你的 Actor 说,看到这个游戏画面向左就是对的,你就是给我向左,那你要怎麽让你的 Actor 学到这件事呢



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

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



$$L = e$$

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

然后呢,你再去 Learn 一个 θ,你再去 Learn 一个 θ,然后这个 θ 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 $\hat{a}$

$$L = -e$$

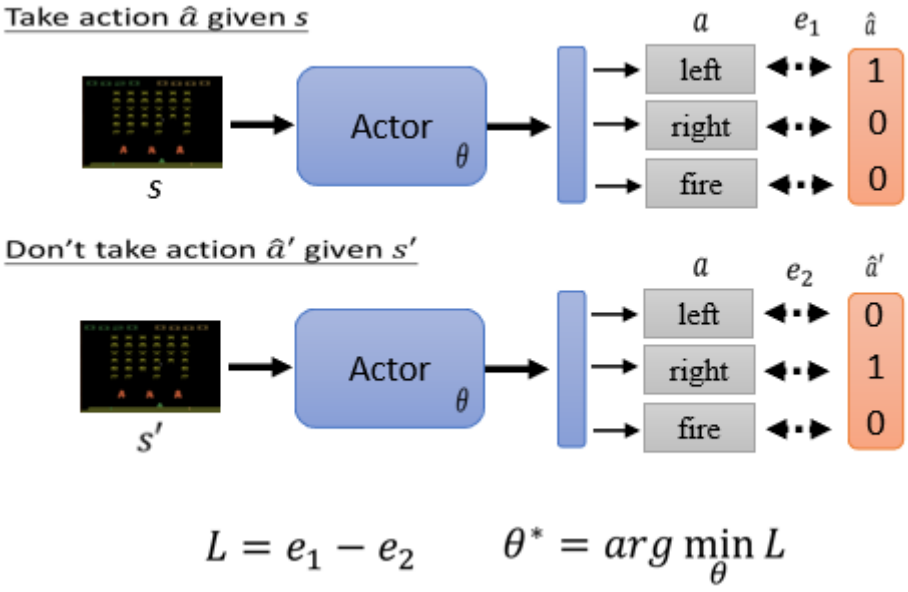
Cross-entropy

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

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,要怎麼做呢

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



$$L = e_1 - e_2$$

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

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 θ 去 Minimize Loss,得到 θ^*,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,

有一个同学问了一个非常好的问题

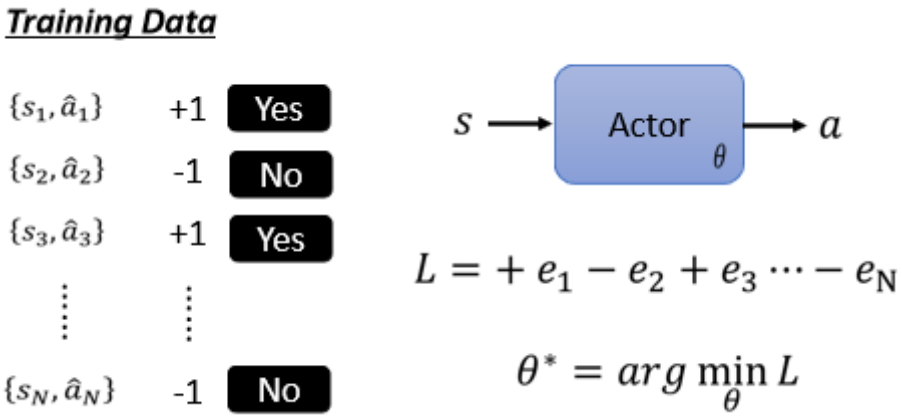
- **Q:**就是如果以 Alien 的游戏来说的话,因为只有射中 Alien 才会有 Reward,这样 Model 不是就会一直倾向于射击吗

  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 $\hat{a}2$

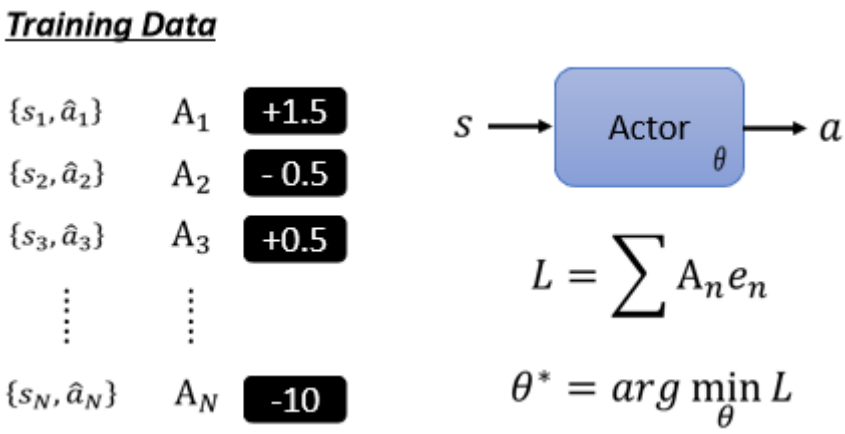


但可能会问说,欸 这个训练资料哪来的,这个我们等一下再讲训练资料哪来的

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,这是我们就用 ±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

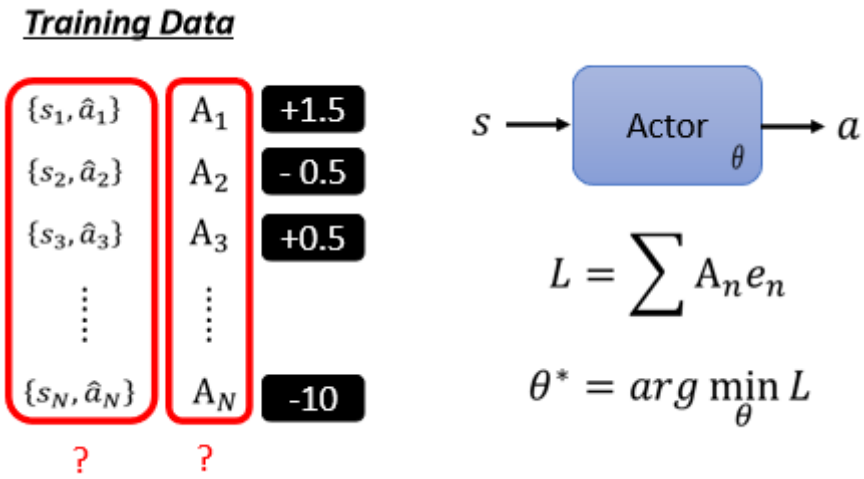
那比如说这边第一笔资料跟第三笔资料,我们分别是定 +1.5 跟 +0.5,就代表说我们期待机器看到 s_1 的时候,它可以做 $\hat{a}_1$,看到 s_3 的时候它可以做 $\hat{a}_3$,但是我们期待它看到 s_1 的时候,做 $\hat{a}_1$ 的这个期待更强烈一点,比看到 s_3 做 $\hat{a}_3$ 的期待更强烈一点

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 sN 的时候,不要做 $\hat{a}_N$,而且我们非常不希望,它在看到 sN 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 An 这一项,改成乘上 An 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麼大

所以我们**透过这个 An 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 θ,Train 下去你就找一个 θ^*,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,要怎麼定出这一个 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,怎麼产生这个 s 跟 a 的 Pair 呢,怎麼知道在 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,