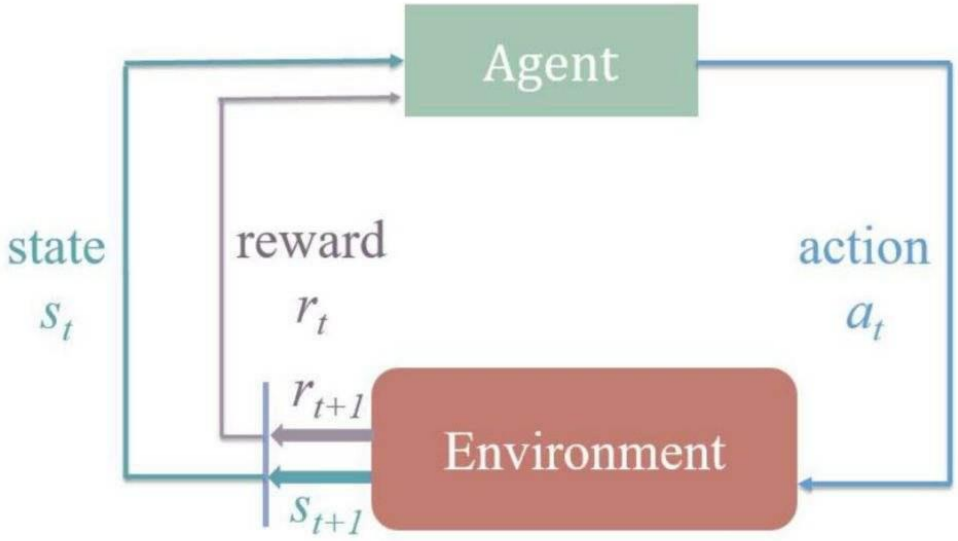
Hung-yi Lee 强化学习课程笔记



本文档为第一版，后续还会修改，修改后的文档会放入 Github 中

本文档是针对李宏毅老师深度学习课程(Reinforcement Learning)

视频做的笔记

编者：何志强

1603868203@qq.com

https://github.com/18279406017

2018.08.13

目录

1. **Policy gradient** 3
2. **Proximal Policy Optimization (PPO)** 9
3. **Q-Learning** 13
   - Monte- Carlo(MC) based approach 13
   - Temporal-difference (TD) approach 14

   Another Critic 15

   Some Tips For Q-learning 17

   ·Target Network 17

   ·Exploration 18

   ·Replay Buffer 18

   Advanced Tips 18

   ·Double DQN 18

   ·Dueling Network 19

   ·Prioritized Reply 20

   ·Multi-step 21

   ·Distributional Q-function 22

   ·Rainbow 22

   Q-learning for continuous Actions 23
4. **Actor-Critic** 24

   ·Advantage Actor-Critic (A2C) 24

   ·Asynchronous Advantage Actor-Critic (A3C) 26

   ·Pathwise derivative policy gradient 26
5. **Sparse Reward** 29
   - Reward shaping 29
   - Curriculum Learning 30
   - Hierarchical Reinforcement Learning 31
6. **Imitation Learning** 32

   ·Behavior Cloning 32

   ·Inverse Reinforcement Learning (IRL) 33

1. Policy gradient

在强化学习里面有三个基本要素：Actor(动作)；Env(环境)；Reward Function(奖励函数)。例如我们用强化学习去玩游戏，那么这个时候的 Actor 就是去操控游戏柄，Env 就是我们的游戏，Reward Function 就是我们在游戏中得到的分数。这里要注意一点就是这个 Enviroment 和 Reward Function 不是我们所能控制的，在强化学习智能体开始学习之前就已经事先给定了，智能体唯一能做的就是调整自己的策略，使得自己拿到更多的 Reward。Actor 是依据 Policy 来采取动作。因此 Policy 就是给一个外界的输入，然后输出 Actor。假设我们使用 deep learning 的技术来做 Reinforcement learning. 那么这个 Policy π is a network with parameter θ . Input : the observation of machine representd as a vector or a matrix. Output : each action corresponds to a neuron in output layer. 在这里要说一下这个 Input。这个 Input 很大程度上会影响我们的 Training。举例来说：比如在玩游戏的时候，如果游戏前后的画面是相关的，这个时候使用 RNN 来做连续画面的处理可能会更好，当然这个也会比较难处理。如果没有记错的话，在 AlphaGo 中一张棋盘是做成了 48 张图片(比如棋盘上只有白子、只有黑字等等)作为一个状态的输入。要让 Policy 看到什么画面这个是由我们自己决定的。

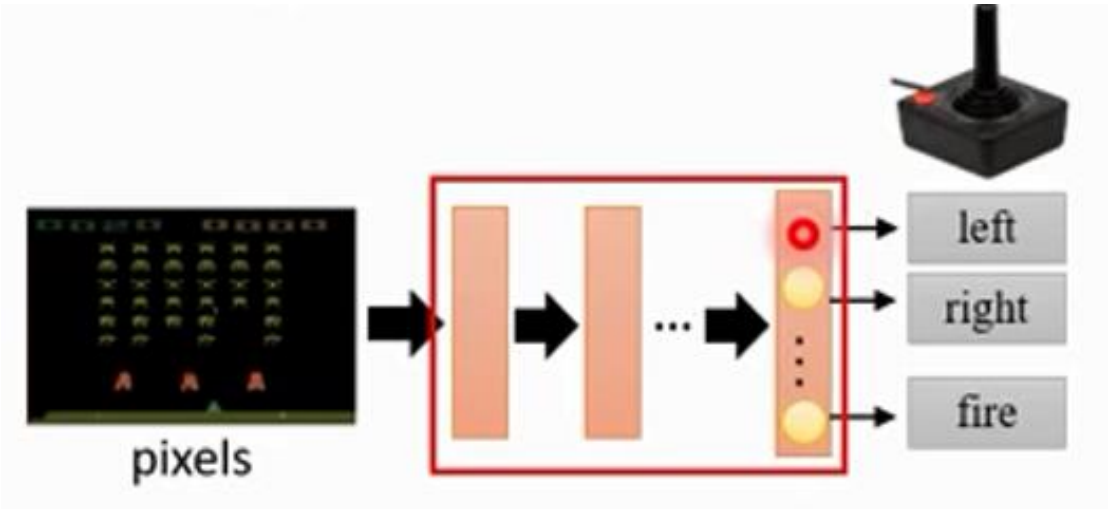


图 1-1 智能体决策图

以上面这张图为例的话我们的 Input 就是游戏的画面，Policy 就是 Network，Action 就是 Left、Right、Fire。当输入一张真实的游戏场景的时候，我们的 Policy 就会输出对应 Action 的分数，这个分数就是采取动作的概率大小，强化学习依据这个概率的大小决定自己的行为。

那我们的强化学习 Agent 是如何跟环境互动的呢？

首先我们的 Agent 会看到一个游戏的画面，这个游戏的画面我们使用 state S1 来表示，接下来我们的 Agent 看到这个游戏的初始画面后，依据它内部的 Policy 采取相应的 Action，假设是向右走一步，采取完这个 Action 后将会得

到一个 Reward。接下来就会看到新的游戏图像，也就是新的 Enviorment，或则叫做新的 state S2，这样一直重复下去，直到执行某个 Action 得到一个奖励后这个游戏结束了。一场游戏叫做一个 Episode，这个 Total Reward 如公式(1)所示：

$$R=\sum_{t=1}^{T}\mathrm{r}_t \tag{1-1}$$

其流程图如下图所示：

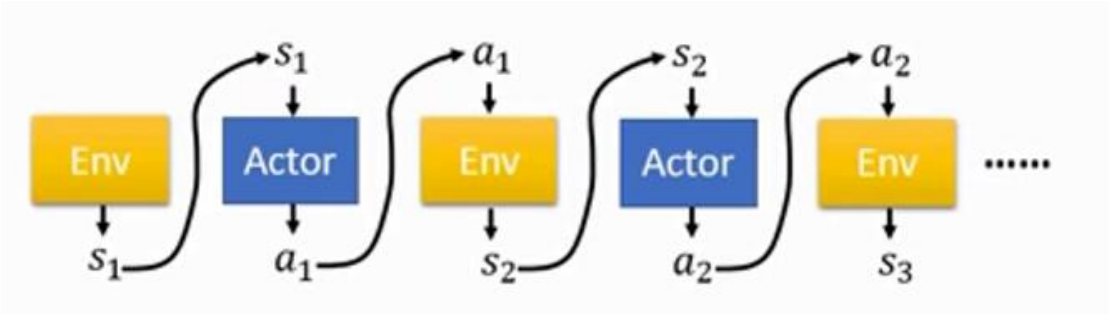


图 1-2 智能体动作序列图

我们把一场游戏中的 state S 和 action A 串起来叫做一个 Trajector τ。

$$\tau=\{s_1,a_1,s_2,a_2,\cdots,s_T,a_T\} \tag{1-2}$$

假设 Policy 的参数θ被给定的话，我们就可以计算某一个回合中 Trajector 发生的概率 $p_\theta(\tau)$。

$$\begin{aligned} &p_\theta(\tau)\\ &=p(s_1)p_\theta(a_1\mid s_1)p(s_2\mid s_1,a_1)p_\theta(a_2\mid s_2)p(s_3\mid s_2,a_2)\cdots\\ &=p(s_1)\prod_{t=1}^{T}p_\theta(a_t\mid s_t)p(s_{t+1}\mid s_t,a_t) \end{aligned} \tag{1-3}$$

上式对应的就是 environment 输出 S1 的几率乘以 policy 依据 S1 采取 a1 的几率乘以在 S1 下采取 a1 得到 S2 的几率 ••• 依次算下去即可得到某一个回合中 Trajector 发生的概率 $p_\theta(\tau)$。这里要说明几点：$p_\theta(a_t\mid s_t)$是 policy 依据 S1 采取 a1 的几率，是由参数θ所决定的；$p(s_{t+1}\mid s_t,a_t)$是 S1 下采取 a1 得到 S2 的几率，是由环境本身的几率设置有关系的，这一项我们是无法控制的。

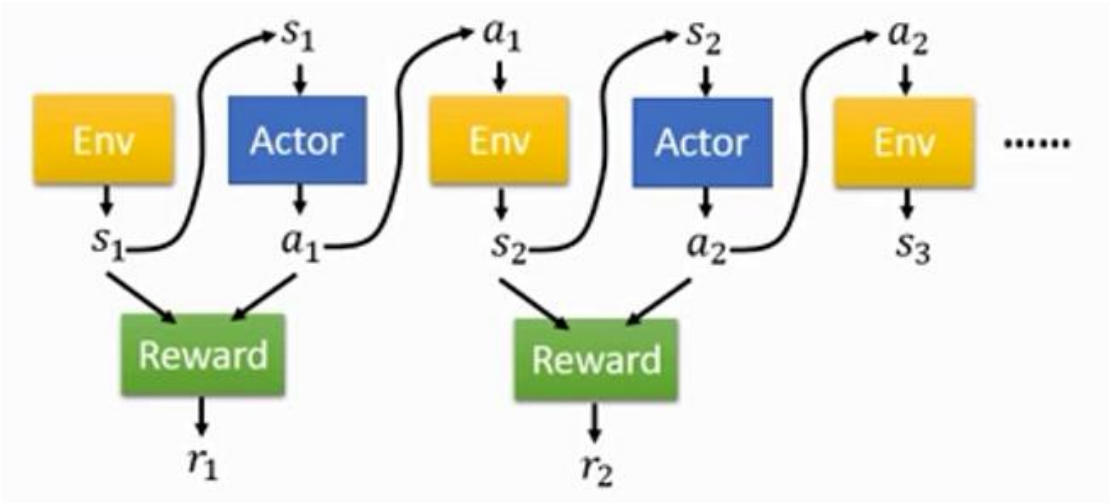


图 1-3 智能体动作奖励序列图

除了之前所说的 state 和 action 后还有一个 Reward Function，Reward Function 是由当前的 state 采取相应的 action 由 Enviroment 所给出的。我们将所有的 Reward 加起来就能得到公式(1)。我们需要做的是调整 Policy 中的参数θ使得这个 Total Reward 越大越好，但是这个 Total Reward 其实是一个随机变量，这个随机性是由于环境的不确定性所导致的。所以我们能够计算的是在某一组参数θ下我们得到的 Total Reward 的期望值。

那么这个期望值是怎么算的呢？

$$\bar{R}_\theta = \sum_\tau R(\tau) p_\theta(\tau) = E_{\tau \sim p_\theta(\tau)}[R(\tau)] \tag{1-4}$$

这个期望值就是穷举所有可能 Trajector，计算这个 Trajector 发生的概率和相应的 Total Reward 并将其相乘求和便得到了 Total Reward 的期望值，如公式(4)所示。

那么我们要做的事情就变成了最大化我们的期望 Reward。要最大化我们的期望 Reward 的话，我们就需要计算 Policy Gradient $\nabla \bar{R}_\theta$

$$
\begin{aligned}
\nabla \bar{R}_\theta &= \sum_\tau R(\tau)\nabla p_\theta(\tau) = \sum_\tau R(\tau) p_\theta(\tau)\frac{\nabla p_\theta(\tau)}{p_\theta(\tau)} \\
&= \sum_\tau R(\tau) p_\theta(\tau)\nabla \log p_\theta(\tau) \\
&= E_{\tau\sim p_\theta(\tau)}[R(\tau)\nabla \log p_\theta(\tau)] \\
&\approx \frac{1}{N}\sum_{n=1}^{N} R(\tau^n)\nabla \log p_\theta(\tau^n) \\
&= \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n)\nabla \log p_\theta(a_t^n \mid s_t^n)
\end{aligned}
\tag{1-5}
$$

这里假设我们在 s_t^n 状态下得到的奖励是正的，那么我们就需要增加在 s_t^n 状态下采取动作 a_t^n 的概率 $p_\theta(a_t^n \mid s_t^n)$。反之就要减少相应的概率。最总我们通过公式(6)更新整个网络。

$$
\theta \leftarrow \theta + \eta \nabla \bar{R}_\theta \tag{1-6}
$$

其中 η 是我们的 learning rate。其参数更新流程主要如下图所示：

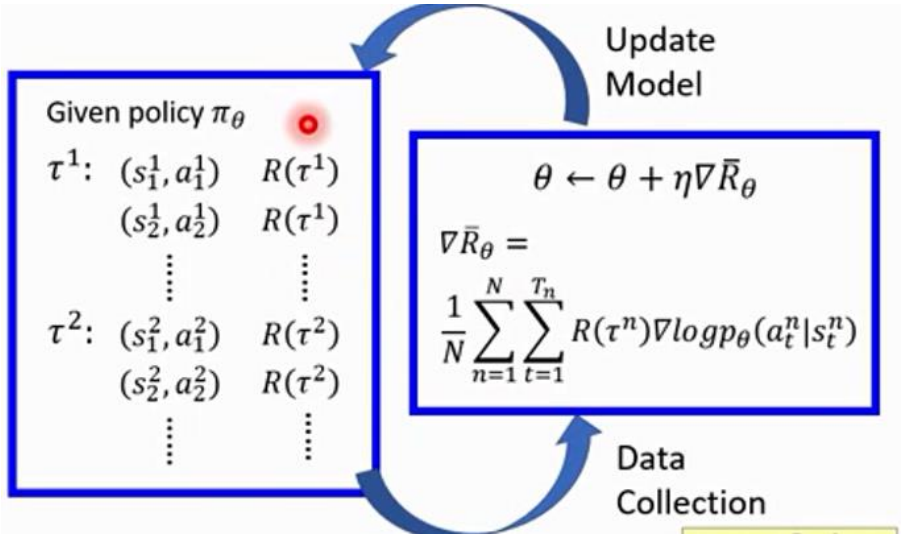


图 1-4 强化学习参数更新流程图

在这里我们可以将其想象成一个分类的问题，输入是一张图片，输出为动作，我们可以将动作分成三类：向左、向右、开火。

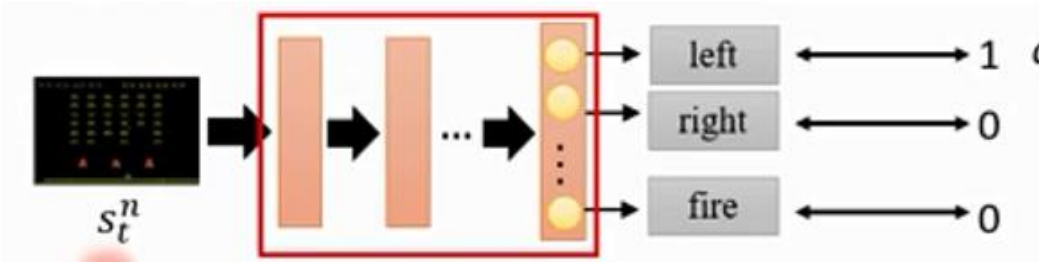


图 1-5 强化学习近似为分类问题图示

假设我们在 s_t^n sample 到向左 a_t^n，那么我们就告诉我们的 network 去调整你的参数，当你看到 s_t^n 后，你就向左。在一般的 classfication 问题中一般都会最小化损失函数来解决这样一个问题，但是在强化学习里面我们要 max 公式(5)

在实做的时候这里有些 Tip：

Tip1：Add a Baseline

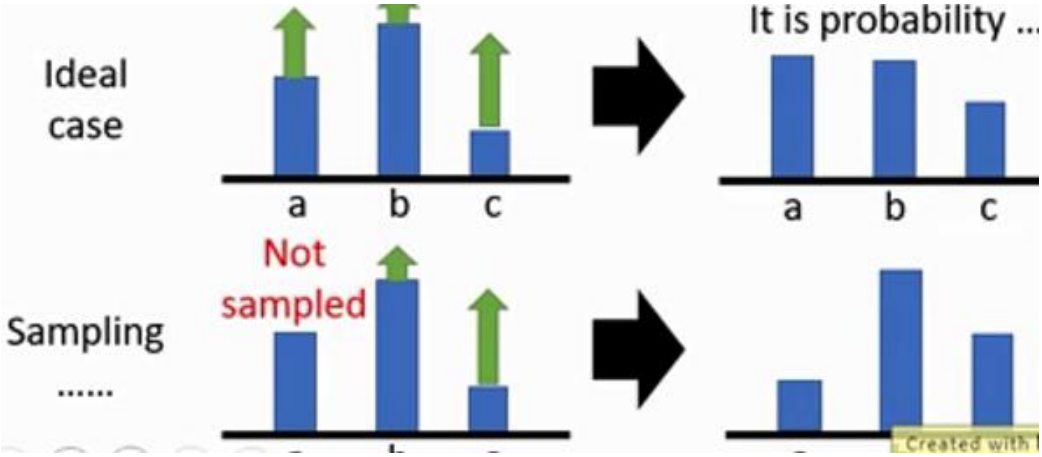


图 1-6 未加 Baseline 问题示意图

在我们在做采样的时候，我们可能有些 Action 是没有被 Sample 到的，以至于没有被 Sample 到的好动作没有被提高下次被采取的概率。为了解决这个问题我们希望我们的 Reward 不要总是正的，由此我们的更新公式(5)变为公式(7)

$$\nabla \bar{R}_\theta = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}(R(\tau^n)-b)\nabla \log p_\theta(a_t^n \mid s_t^n) \tag{1-7}$$

这个 b 的值我们可以拿所有 Total Reward 的平均值来定义。

Tip2：Assign Suitable Credit

由于之前是用某一个 Trajector 中的所有 Reward 作为更新的权重大小，但是这样并不是对每个 Action 都是公平的，如下图所示：

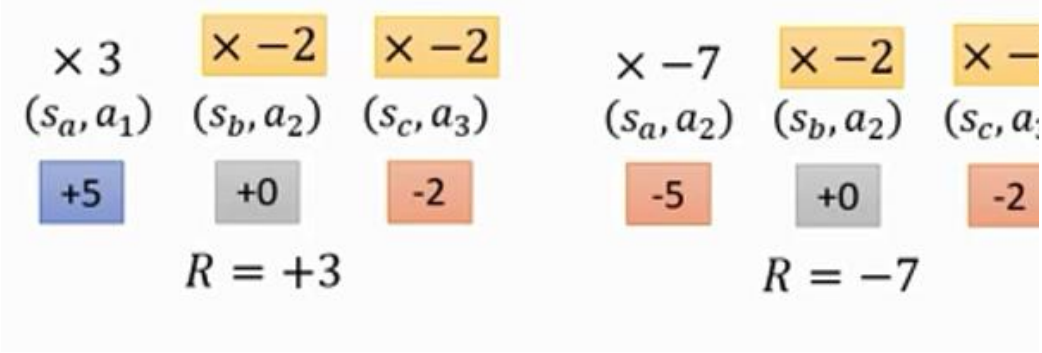


图 1-7 未加 Credit 问题示意图

采取公式(5)或者公式(7)这样的公式的话会将图 7 左面的 a3 的概率增大，这将会导致不好的行为被选中的概率。由此我们希望对每一个 action 前面都乘以一个 weight 使其能够真正反映每一个 action 到底是好还是不好。因此公式(7)可更新为公式(8)。

$$\nabla \bar{R}_\theta = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\sum_{t'=t}^{T_n} r_{t'}^n - b\right)\nabla \log p_\theta(a_t^n \mid s_t^n) \tag{1-8}$$

其中 $\sum_{t'=t}^{T_n} r_{t'}^n$ 表示的是从某一个动作开始之后得到的所有奖励。为了使得越后面的奖励对当前的奖励的影响干扰越小(因为越后面的奖励，可能是由越后面的动作所导致的，与当前的动作关系越少)，公式(8)可更新为公式(9)。

$$\nabla \bar{R}_\theta = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\sum_{t'=t}^{T_n} \gamma^{t'-t} r_{t'}^n - b\right)\nabla \log p_\theta(a_t^n \mid s_t^n) \tag{1-9}$$

其中 γ 为折扣因子，其值小于 1。上式得 $\sum_{t'=t}^{T_n} \gamma^{t'-t} r_{t'}^n - b$ 可表示为 Advantage Function 用 $A^\theta(s_t, a_t)$ 表示，因为实际上在算 Advantage Function 这一项的时候我们可以采取一个相对的值来计算这个 action 有多好，也就是说相对于其他的 action，当前的 action 会不会比其他的 action 好，是一个相对的值，这个时候我们可以采用 Critic 网络来评估这样一个值，即使这个 Critic 网络不好，但是 Critic 网络能够反映出一个相对的好，这样就够了。

2. Proximal Policy Optimization (PPO)

在学习 PPO 算法之前，我们先要了解什么是 On-Policy 什么是 Off-Policy。

On-Poilcy : The agent learned and the agent interacting with the environment is the same.

Off-Poilcy : the agent learned and the agent interacting with the environment is different.

简单来说就是：我们训练的智能体和我们与环境交互的智能体，是否是同一个智能体？如果是同一个智能体的话那么就是 On-Policy，如果不是同一个智能体的话那就是 Off-Policy。举个例子就是：一个智能体自己边玩游戏边学习那么这个叫做 On-Policy，如果是看别人玩来学习的话这个叫做 Off-Policy。之前说的 Policy Gradient 就是 On-Policy。

我们回顾一下之前的更新公式：

$$\nabla \bar{R}_\theta = E_{\tau \sim p_\theta(\tau)}[R(\tau)\nabla \log p_\theta(\tau)] \quad (2\text{-}1)$$

在(2-1)这个公式里面是对现在的 Policy θ 所 sample 出来的 Trajector τ 求期望。所以当我们 Policy 的参数从 θ 变为 θ^- 后，上述公式(2-1)就不对了，也就是之前 sample 出来的 data 更新完一次 θ 就不能用了。所以 Policy 会花很多时间来 sample data。

所以我们需要从 On-Poilcy 变为 Off-Policy，这样做的好处就是，我们能够用另外一个 Policy、另外一个 π_{θ^-} 去跟环境做互动，这样就可以做很多次 Policy Gradient。原文是这样说的：

- Use π_θ to collect data. When θ is updated, we have to sample training data again.
- Goal : Using the sample from π_{θ^-} to train θ. θ^- is fixed, so we can re-use the sample data.

在实现这样一个功能之前，我们需要介绍一下 Importance Sampling，Importance Sampling 不是只能用在 RL 上，它是一个通用的方法。假设我们有一个 function $f(x)$，我们要从 p 这个 distribution 中去 sample x，再把 x 带入到 $f(x)$ 中去计算其期望值。

$$E_{x\sim p}[f(x)] \approx \frac{1}{N}\sum_{i=1}^{N} f(x^i) \tag{2-2}$$

Where x^i is sampled form $p(x)$.

但是我们现在的问题是我们没有办法从 p 这个 distribution 中去 sample data，我们只能从另外一个 distribution q 中去 sample data。q 这个分布可以是任意分布，在多数情况下都是成立的。但是我们从 q 采样出来的数据不能带入公式(2-2),所以我们需要做一个修正，如公式(2-3)：

$$\begin{aligned} E_{x\sim p}[f(x)] &= \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx \\ &= E_{x\sim q}[f(x)\frac{p(x)}{q(x)}] \end{aligned} \tag{2-3}$$

由此即可将从 p 这个 distribution 中 sample data 转化为从 q 这个 distribution 中 sample data。公式(2-2)与(2-3)的不同在于公式(2-2)是从 p 这个 distribution 中 sample data，公式(2-3)是从 q 这个 distribution 中 sample data，并且 $f(x)$ 还乘了个 weight $\frac{p(x)}{q(x)}$，去修正这两个分布之间的差异。虽然在理论上可以把 p 换成任意的 q，但是在实做上 p 与 q 还是不能差太多。如果差太多的话会有一些问题，如下图所示。

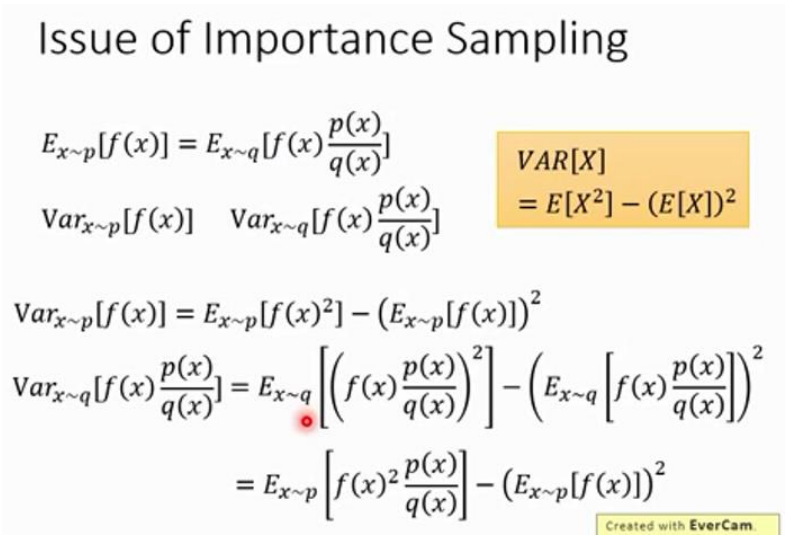


图 2-1 重要性采样的问题

但是我们 sample 的次数足够多的话就能够解决这个问题。

我们接下来要做的事情就是把 Importance sample 用在 Ploicy 里面使得 On-Policy 变为 Off-Policy

之前我们的更新公式(2-1)为

$$\nabla \bar{R}_\theta = E_{\tau \sim p_\theta(\tau)}[R(\tau)\nabla \log p_\theta(\tau)] \tag{2-1}$$

采用重要性采样后，我们就可以从θ^-中采样，这样能够使得数据被利用很多次，一直到θ^-被替换掉之后，之后我们的更新公式变为：

$$\nabla \bar{R}_\theta = E_{\tau \sim p_{\theta^-}(\tau)}[\frac{p_\theta(\tau)}{p_{\theta^-}(\tau)} R(\tau)\nabla \log p_\theta(\tau)] \tag{2-4}$$

在 Policy Gradient 中我们并不是给一整个 Trajector 的 Reward，我们的更新公式如公式(2-5)所示：

$$\begin{aligned}
&\nabla \bar{R}_\theta = E_{(s_t,a_t)\sim\pi_\theta}[A^\theta(s_t,a_t)\nabla \log p_\theta(a_t^n \mid s_t^n)] \\
&= E_{(s_t,a_t)\sim\pi_{\theta^-}}[\frac{p_\theta(s_t,a_t)}{p_{\theta'}(s_t,a_t)} A^{\theta^-}(s_t,a_t)\nabla \log p_\theta(a_t^n \mid s_t^n)] \\
&= E_{(s_t,a_t)\sim\pi_{\theta^-}}[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t \mid s_t)} \frac{p_\theta(s_t)}{p_{\theta'}(s_t)} A^{\theta^-}(s_t,a_t)\nabla \log p_\theta(a_t^n \mid s_t^n)]
\end{aligned} \tag{2-5}$$

也就是使用τ_θ去 sample 出(s_t,a_t)，然后计算期望，(2-5)中的第一个等式是之前的 Policy Gradient 的，第二个等式是采用重要性采样后的等式。第三个等式是第二个等式的一个变形，其中$\frac{p_\theta(s_t)}{p_{\theta'}(s_t)}$可以约掉，我们可以想象成我们在玩游戏的时候看到的游戏画面其实都是差不多的，不同的θ对 state 的影响都非常小。

我们就能够得到一个新的目标函数了，也就是我们需要去优化的目标函数，可以看成 Policy Gradient 的$A^\theta(s_t,a_t)$。

$$J^{\theta'}(\theta) = E_{(s_t,a_t)\sim\pi_{\theta^-}}[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t \mid s_t)} A^{\theta^-}(s_t,a_t)] \tag{2-6}$$

至此的话我们就可以将 On-Policy 变成 Off-Policy。之前讲的 Importance Sample 中，两个分布不能差太多，如何来使得这两个分布不能够差太多呢？这就是 PPO 做的事情。由此导出了 PPO 的算法跟新公式。

$$\begin{aligned}
&J_{ppo}^{\theta'}(\theta) = J^{\theta'}(\theta) - \beta KL(\theta,\theta') \\
&J^{\theta'}(\theta) = E_{(s_t,a_t)\sim\pi_{\theta^-}}[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t \mid s_t)} A^{\theta^-}(s_t,a_t)]
\end{aligned} \tag{2-7}$$

其中$\beta KL\left(\theta,\theta^{'}\right)$是$\theta$，和$\theta^{-}$动作输出的差距，用于衡量两个网络的相似度。最后我们希望这两个网络越像越好，不要差距太大。

PPO 有一个前身叫 TRPO(Trust Region Policy Optimization)，在 TRPO 中$KL\left(\theta,\theta^{'}\right)$被单独提出来，作为一个不等式：

$$KL\left(\theta,\theta^{'}\right)<\sigma \tag{2-8}$$

$KL\left(\theta,\theta^{'}\right)$并不是算$\theta$，和$\theta^{-}$参数之间的距离，而是他们动作上的距离。之所以为什么不采用参数上的变化，是因为参数不同输出的 output 是有可能相同的，而动作上的差距才是我们真正在意的。其算法流程如下图所示：

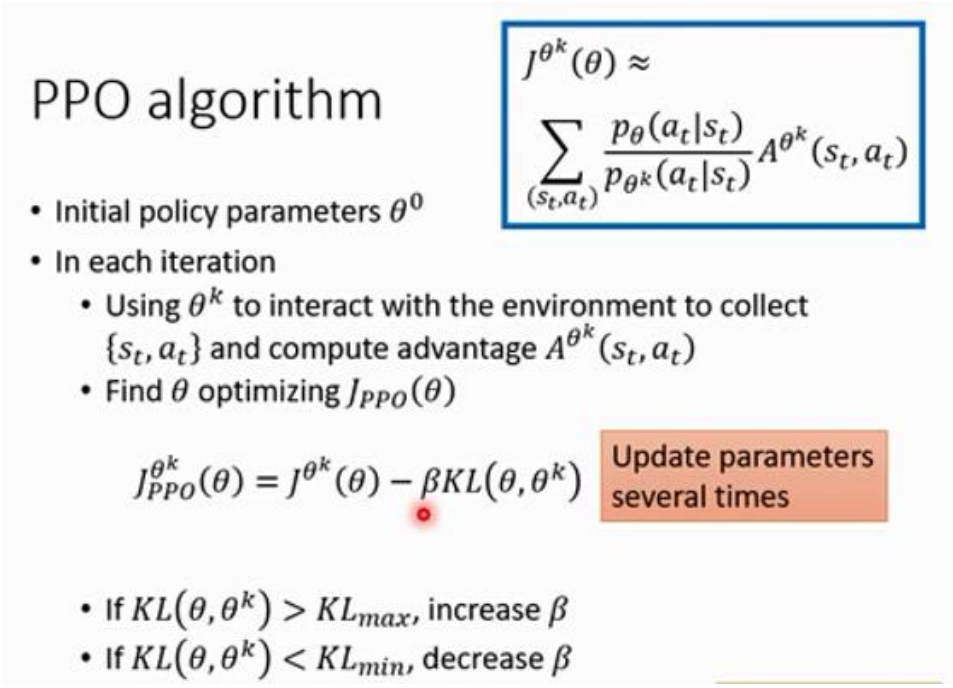


图 2-2　PPO 算法流程图

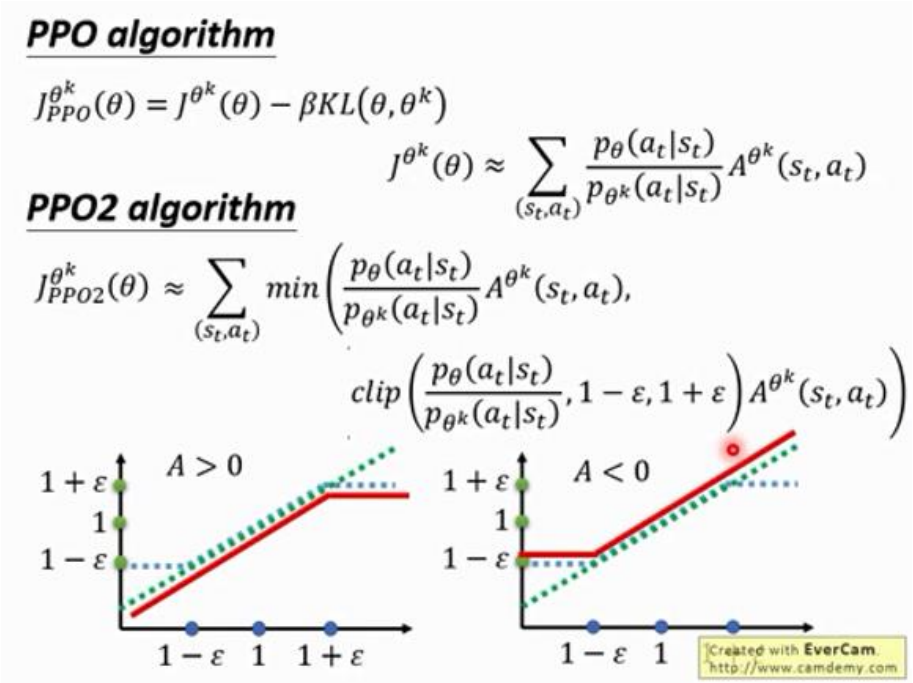


图 2-3　PPO2 算法流程图

图 2-3 可解释为：当 A 大于 0 时，这个动作是好的，但是我们不希望它更新地太多，那么就如图 2-3 左边那条红线一样，A 小于 0 时也是一样。这样的话就能够使得两个网络的参数θ，和θ^-不会差距太大。

3. Q-Learning

Q-learning 是一种 Value Based 的方法，在 Value Based 的方法里面强化学习学习的不是 Policy，而是 Critic，Critic 并不直接采取行为，做的事情是评价现在的行为有多好或则多不好。假设我们现在有一个 actor π，Critic 就是评价这个这个 actor 有多好或则多不好。

- State value function $V^{\pi}(s)$
  - When using actor π，the cumulated reward expects to be obtained after visiting state s

Critic 的输出判别值大小取决于两件事，state 和 动作策略π。其实是衡量某一个 actor 的好坏，而不是大体上衡量某一个 state 的好坏.

现在的问题就是我们如何来衡量这个 state value function $V^{\pi}(s)$？

主要有两种方法：

- Monte- Carlo(MC) based approach
  - The Critic watches π playing the game

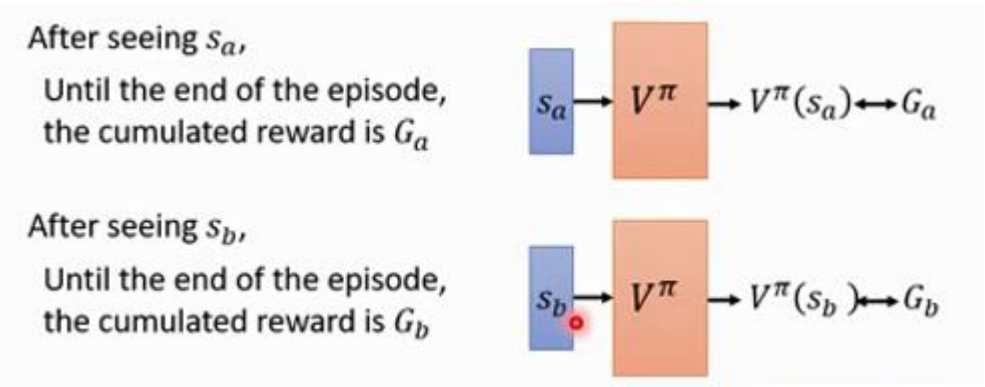


图 3-1 Critic-V-Function 网络工作流程图

其实 Critic 网络做的就是一个回归的问题，当输入状态 Sa 后 Critic 输出一个值，我们希望这个输出的值跟 Ga 越近越好。按照这种方法训练 Network 就可以了。这是第一种方法。在第一种方法中，我们要估计 Critic 的话我们需要将游戏玩完后去计算 Ga，然后进行 Critic 网络更新。但是有些游戏一个 episode 非常长，我们可能收集不到太多的 Reward。并且 Ga 含有很大的随机性，应为在相同的 state 下可能得到不同的 Ga，并且这个 Ga 的差别可能很大。

• Temporal-difference (TD) approach

TD 的方法是一种不需要将游戏玩完一个 Episode 就可以更新 state value 的方法。假设我们有某个 Policy 在 state s_t 采取了动作 a_t，给我们 reward r_t，接着 state 进入 state s_{t+1}。由此可得公式(3-1)

$$V^\pi(s_t) = V^\pi(s_{t+1}) + r_t \tag{3-1}$$

也就是说当前状态的 State Value 等于下一时刻的 State Value 加上当前 state 在当前策略 π 采取动作后获得的奖励值 r_t。在训练的时候依据图 3-2 Critic 训练流程图训练下去就可以了。

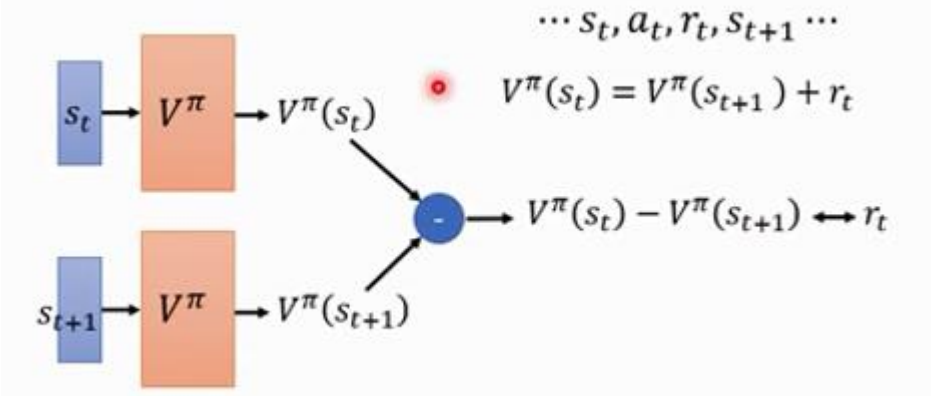


图 3-2 Critic V-Function 训练流程图

TD 的方法中只有 r_t 是一个随机变量，但是这个变化是要比 MC 方法中的 Ga 要小很多，但是这个 $V^{\pi}(s_t)$ 不见得会估计得很准。

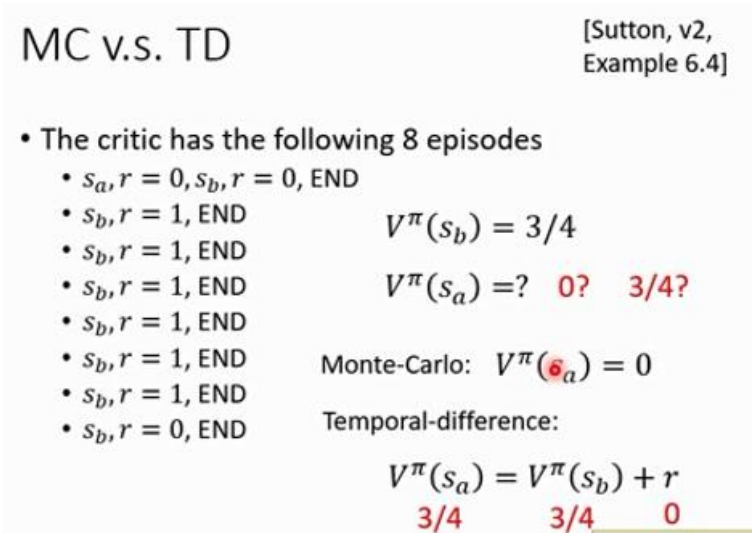


图 3-3 MC 与 TD 举例对比图

Another Critic

·State-action value function $Q^{\pi}(s,a)$

·When using actor π, the cumulated reward expects to be obtained after taking a at state s。

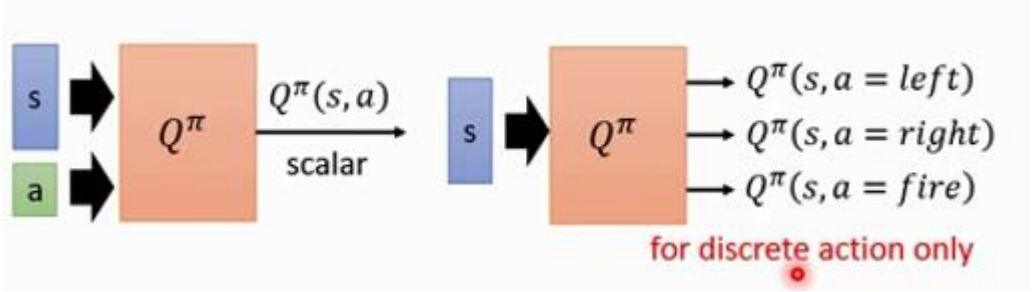


图 3-4 Critic-Q-Function 工作流程图

从表面上看，我们有了一个 state value function 后我们只能去评估一个 state 或则 state 和 action 两者的好坏。但是实际上有了 state value function 后我们就可以做 reinforcement learning。

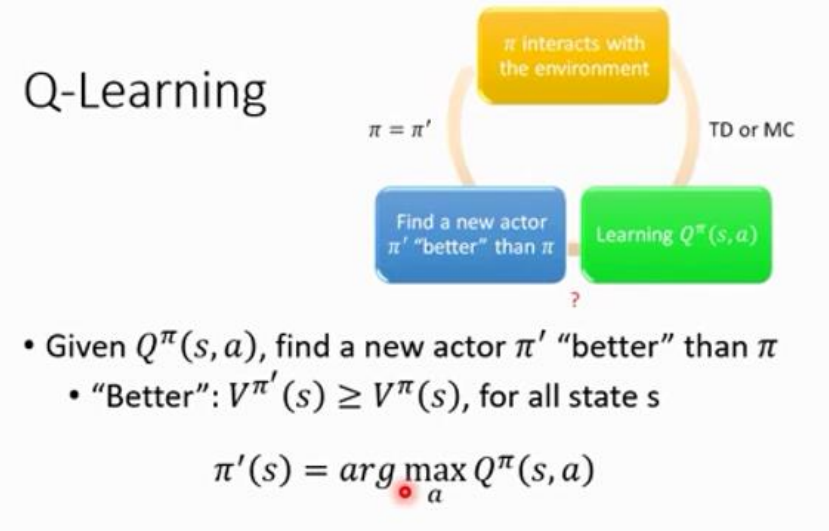


图 3-5 Q-Learning 工作流程图

这里有两点需要注意：

1. π' dose not have extra parameters. It depends on Q
2. Not suitable for continuous action a (solve it later).

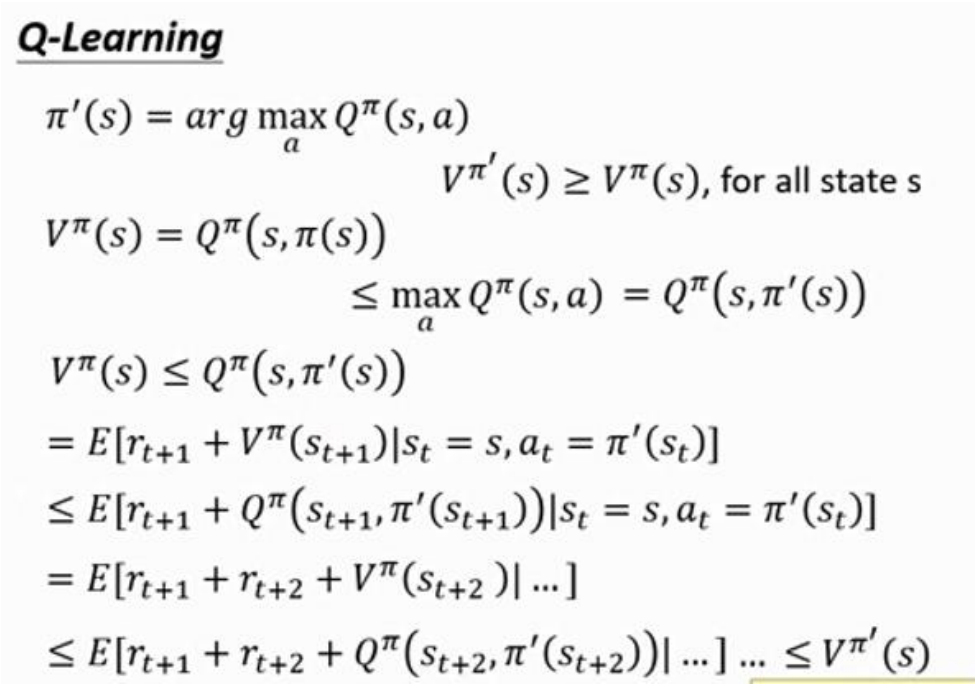
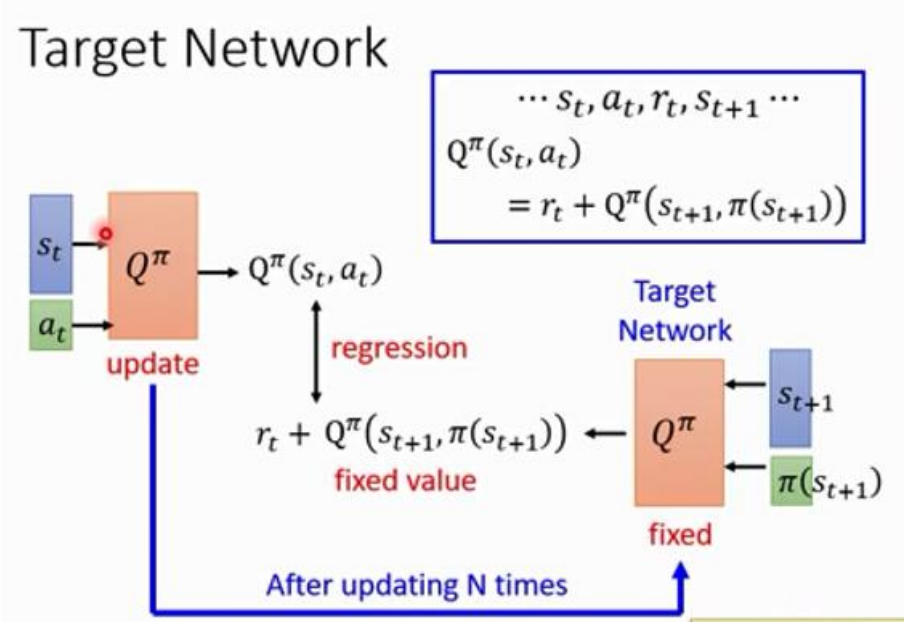


图 3-6 Q-Learning 相关公式证明图

Some Tips For Q-learning

·Target Network



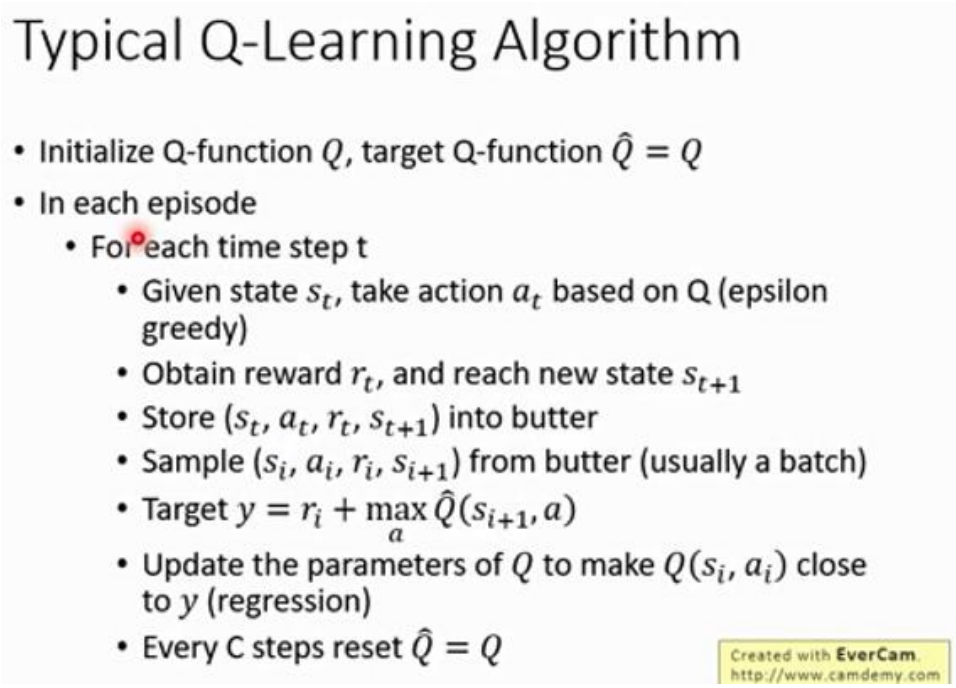
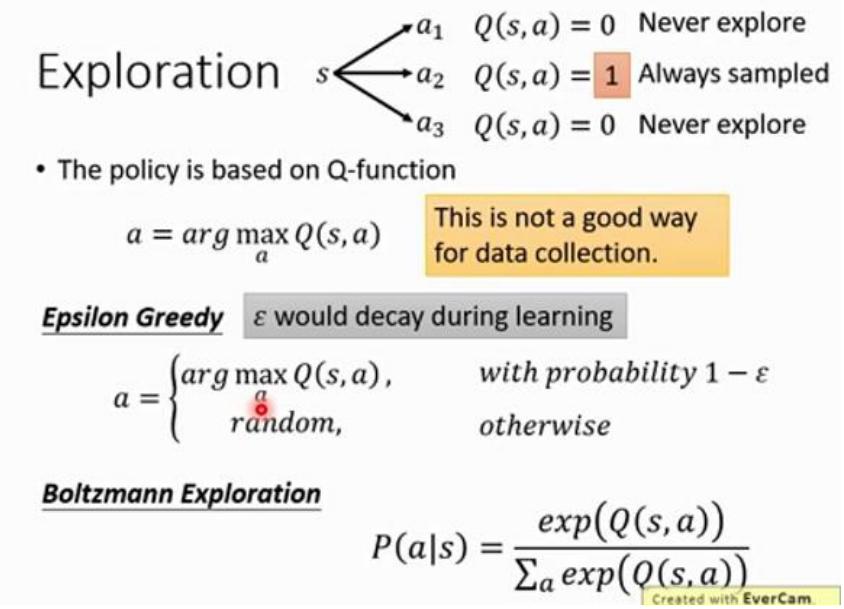


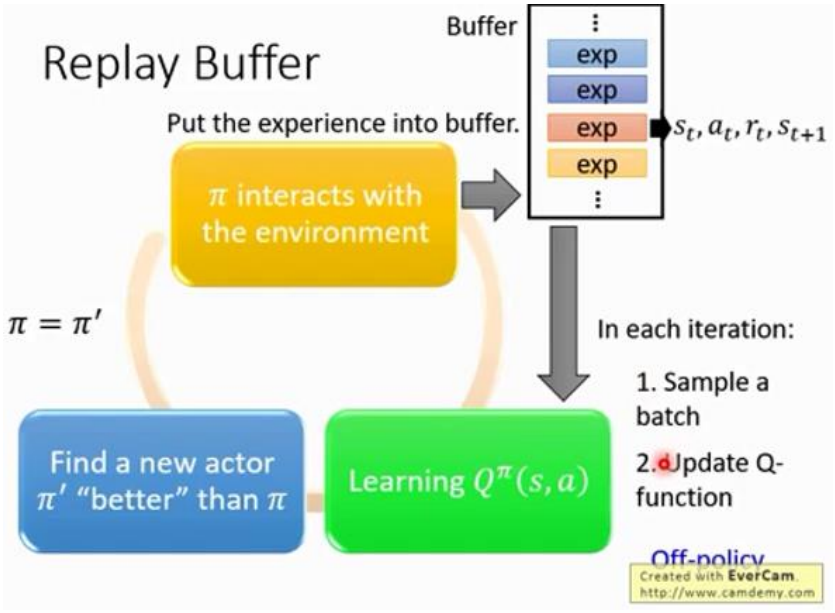
图 3-7 Q-Learning 算法更新图

从上图 3-7 可知我们可以采取 TD 的方法去更新整个网络，但是在实际处理过程中由于网络本身是动态的，这将导致我们的目标函数也是动态的，使得网络不好收敛，因此这个时候设立 Traget Network 就显得很有必要，能够加速网络的收敛。

·Exploration



·Replay Buffer



Advanced Tips

·Double DQN

在实做的时候，Q-learning 的估计值往往都是被高估(Over -eatimated)的，也就是说与实际的 Reward 相比，Q-learning 会高估了自己所能得到的奖励。为什么会总是被高估了呢？

这是由于我们在实际做的过程中，我们总是拿公式(3-2)去作为我们 Q-learning 的更新公式。而我们的 Q 值是被估计出来的，存在一定的误差，每次都取最大的话将会导致在实做的时候，输出的 Q 值被高估。

$$Q(s_t, a_t) \leftarrow r_t + \max_a Q(s_{t+1}, a) \tag{3-2}$$

Double 将评估在 t+1 时刻的状态下，选取动作 a 时的 Q 值换成了另外一个网络 Q',由此将公式(3-2)变换为公式(3-3)。

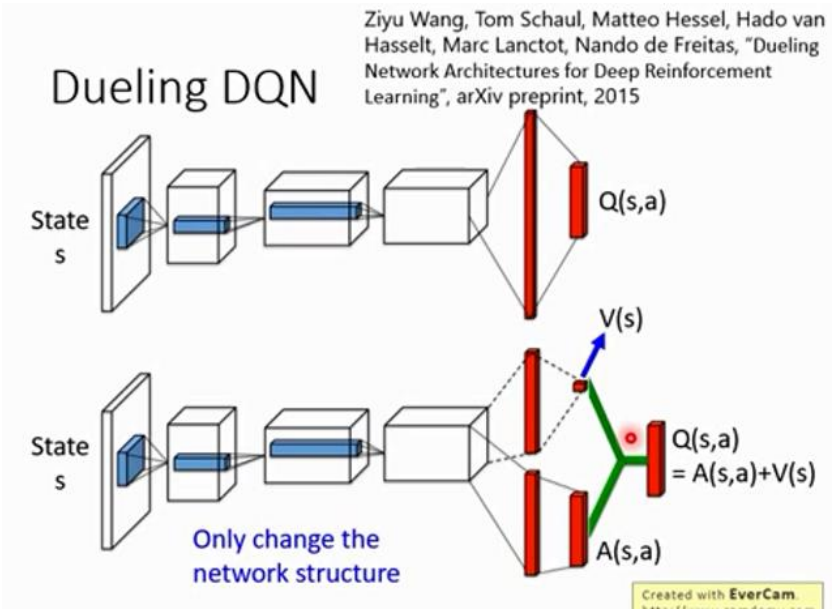
$$Q(s_t, a_t) \leftarrow r_t + Q'(s_{t+1}, \arg\max_a Q(s_{t+1}, a)) \tag{3-3}$$

If Q over-estimate a, so it is selected. Q' would give it proper value.

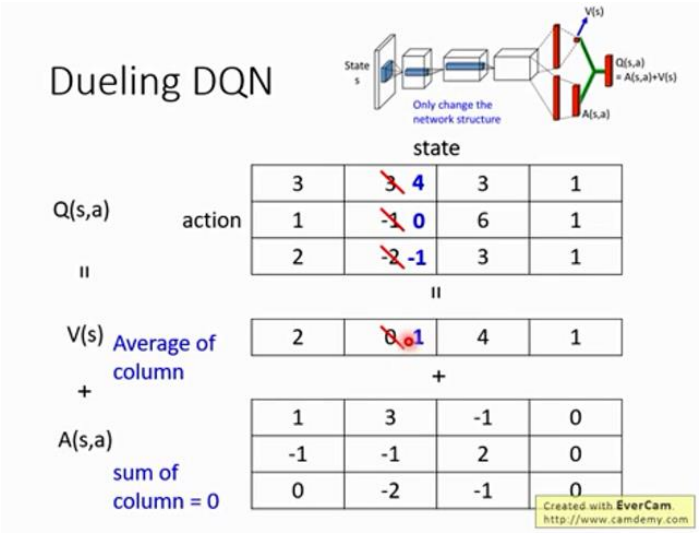
How about Q' overestimate? The action will not be selected by Q.

并且在实做的时候我们确实有两个 Q network，另一个 Q network 就是 Target Network。

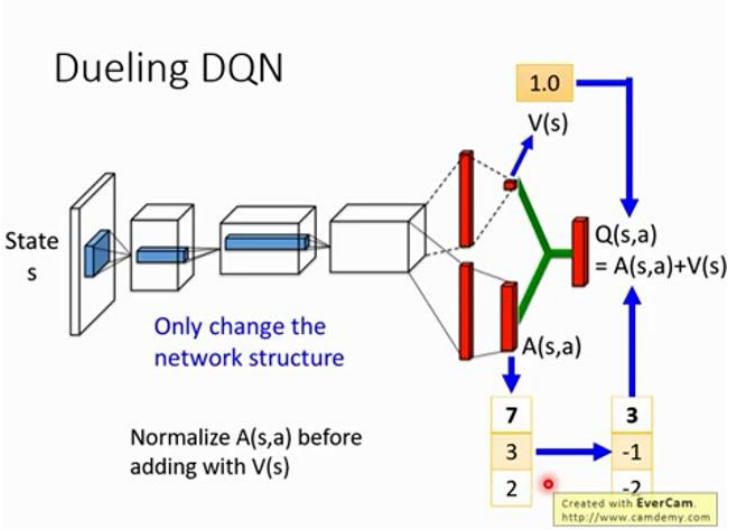
·Dueling Network



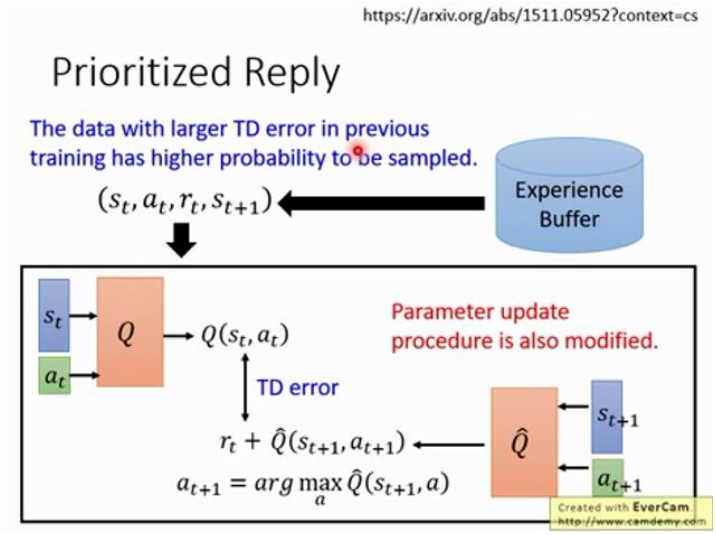
在 Dueling Network 中，只是改变了网络的架构，其改动的部分如上图所示。这样做的好处就是当我们想要改变 Q 的值的时候，可以通过改变 V 的值来做相应的改变，而 V 的值变化后，也将会带动所有 A 的 Q 值得改动，这样就会使得假设我们有一个动作没有被 sample 到，他的 Q 的值也是被改动了。



在实做的时候我们采取下图的方式：A 的值先经过一个 Normalize

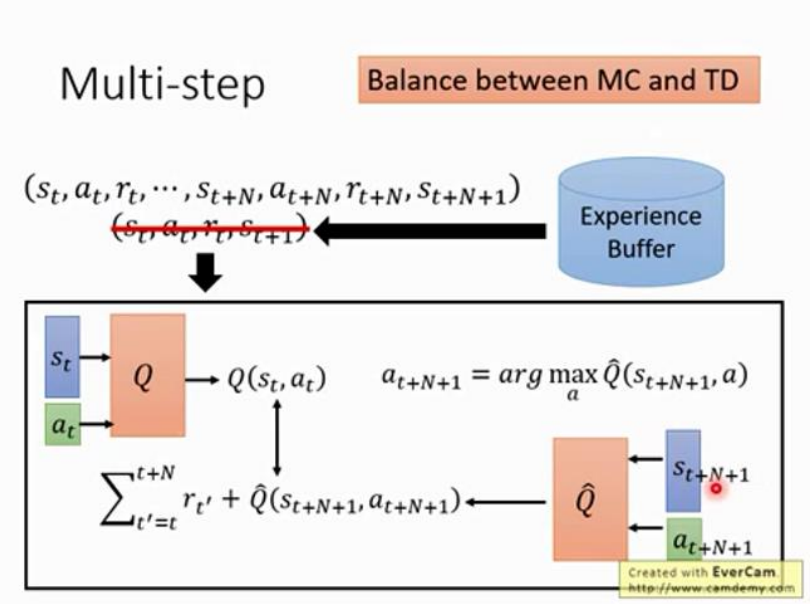


·Prioritized Reply

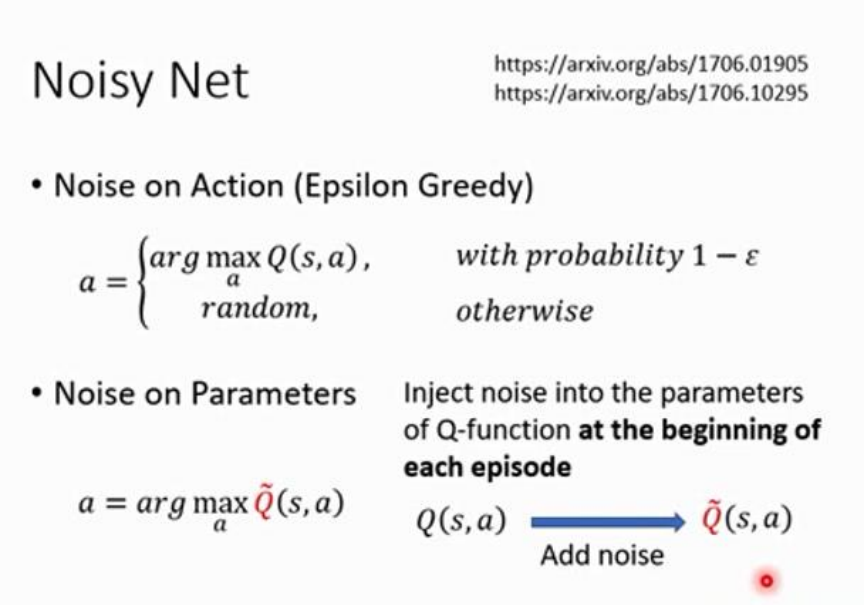


我们在从 Buffer 中采样的时候其实随机采样并不好，如果 TD Error 误差比较大的时候我们希望它被采样的次数增多。

·Multi-step



·Noisy Net



Noisy Net

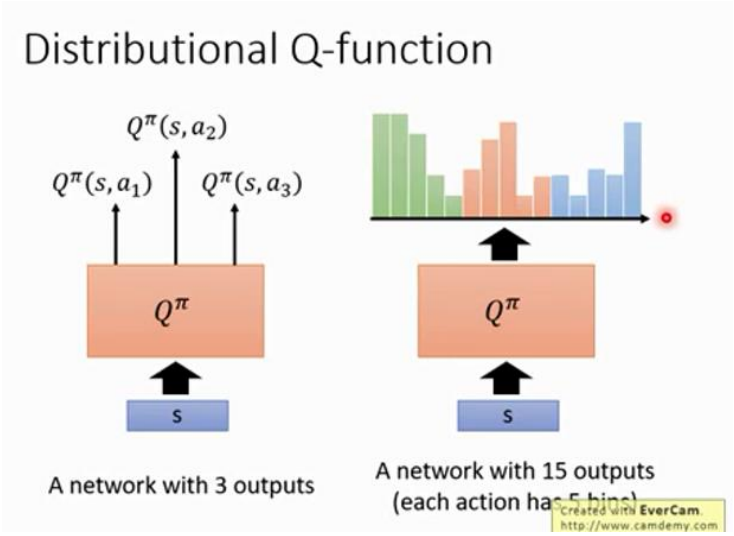
- Noise on Action
  - Given the same state, the agent may takes different actions.
  - No real policy works in this way

隨機亂試

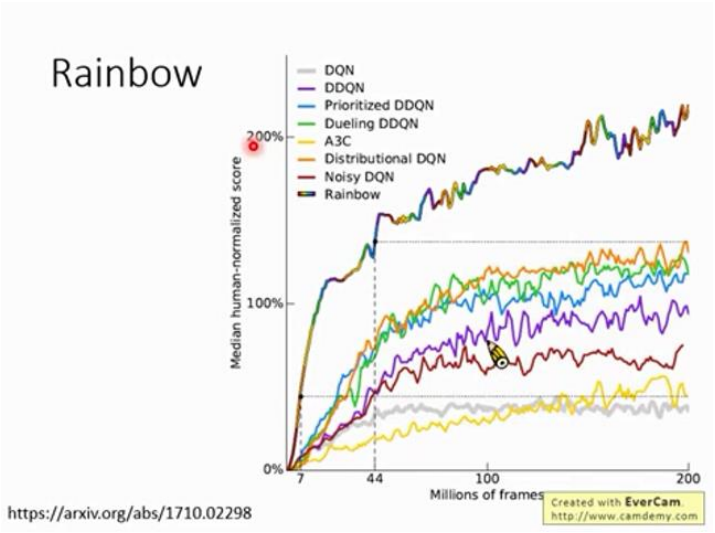
- Noise on Parameters
  - Given the same (similar) state, the agent takes the same action.
    - ⟶ State-dependent Exploration
  - Explore in a *consistent* way

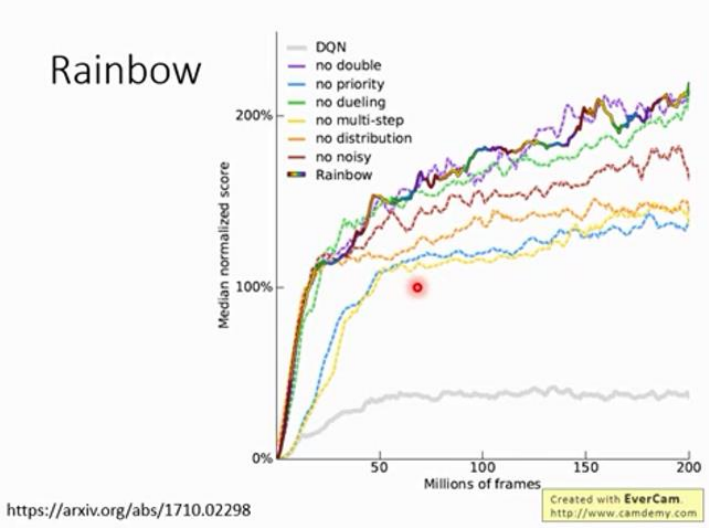
有系統地試

·Distributional Q-function



·Rainbow





Q-learning for continuous Actions

Q-learning 是不太好处理 continuous action 的，但是在很多时候我们的 action 都是 continuous 的 action 的。在 Q-learning 中动作的选取是由公式(3-4)所选取的。

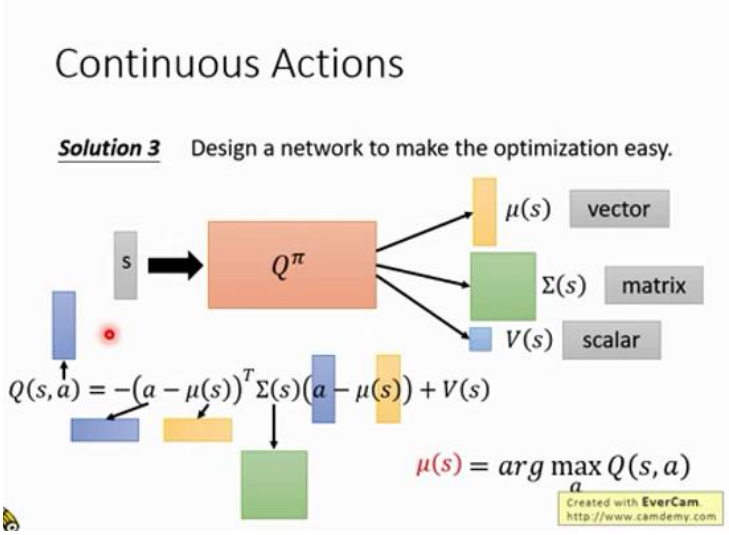
$$a = \arg\max_{a} Q(s,a) \tag{3-4}$$

Solution 1：Sample a set of actions : {a1, a2,· · ·, aN}, and see which action can obtain the largest Q Value.

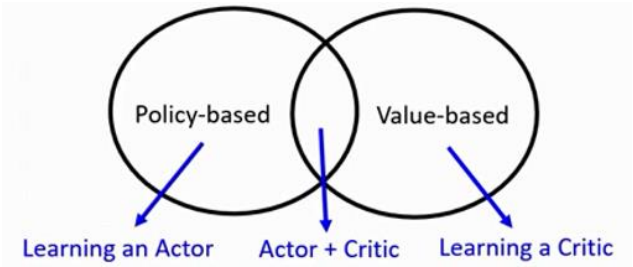
Solution2 : Using gradient ascent to solve the optimization problem.

上述第一种方法就是采取大量的采样将离散的动作近似连续化，第二种方法就是采取 gradient ascent 去找目标函数的最大值，但是这种方法的运算量是比较大的。

Solution3 ： Design a network to make the optimization easy.



Solution4 ： Using actor-critic algorithm.



4. Actor-Critic

• Advantage Actor-Critic (A2C)

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\sum_{t'=t}^{T_n}\gamma^{t'-t}r_{t'}^n - b\right)\nabla \log p_\theta(a_t^n \mid s_t^n) \qquad (4\text{-}1)$$

我们先回顾一下 Policy Gradient：在 Policy Gradient 中我们使用公式(4−1)去算 Gradient 并更新我们的参数θ，但是$\sum_{t'=t}^{T_n}\gamma^{t'-t}r_{t'}^n = G_t^n$是非常的不稳定的，导致其不稳定的原因主要是我们是在做一些 sample，拿这些 sample 计算出来的，其实是一个分布，主要是由于环境的不确定性导致的。由于我们在做 Policy Gradient 的时候我们只能做非常少量的 sample 这就会导致不稳定的现象的发生。因此我们希望$\sum_{t'=t}^{T_n}\gamma^{t'-t}r_{t'}^n$这一项能够变得稳定一点，那么我们是否可以用一个 Network 去估计 G 这一项的期望值？这样的话我们就可以用期望值去代替 G 值。

这里我们采用 Value Based 的方法，在 Value Base 的方法里面这里有两种方法：

• **State value function** $V^\pi(s)$

·When using actor π, the cumulated reward expects to be obtained after visiting state s

- **State-action value function** $Q^{\pi}(s,a)$

·When using actor π, the cumulated reward expects to be obtained after taking a at state s。

$$E[G_t^n] = Q^{\pi_\theta}(s_t^n, a_t^n) \tag{4-2}$$

我们知道 G 的期望值刚好就是$Q^{\pi}(s,a)$的定义。Baseline b 我们可以使用$V^{\pi_\theta}(s_t^n)$去代替。因此公式(4-1)可变为(4-3)。

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}(Q^{\pi_{\theta'}}(s_t^n, a_t^n) - V^{\pi_{\theta''}}(s_t^n))\nabla \log p_\theta(a_t^n \mid s_t^n) \tag{4-3}$$

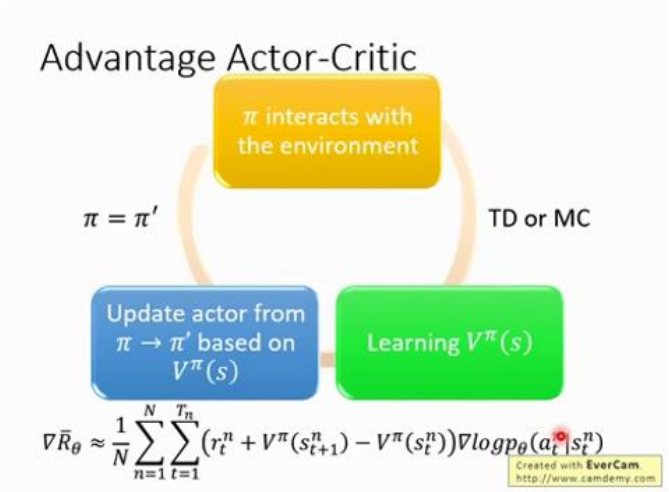
在上式(4-3)中我们需要去做两个 Network 分别估计 Q 值和 V 值，这样的话就会有两倍的风险去承担估测不准的误差。在实际操作中其实是不需要这样做的。我们其实可以用 V 的值来表示 Q 的值，如公式(4-4)所示。

$$\begin{aligned} Q^{\pi_\theta}(s_t^n, a_t^n) &= E[r_t^n + V^{\pi}(s_{t+1}^n)] \\ Q^{\pi_\theta}(s_t^n, a_t^n) &= r_t^n + V^{\pi}(s_{t+1}^n) \end{aligned} \tag{4-4}$$

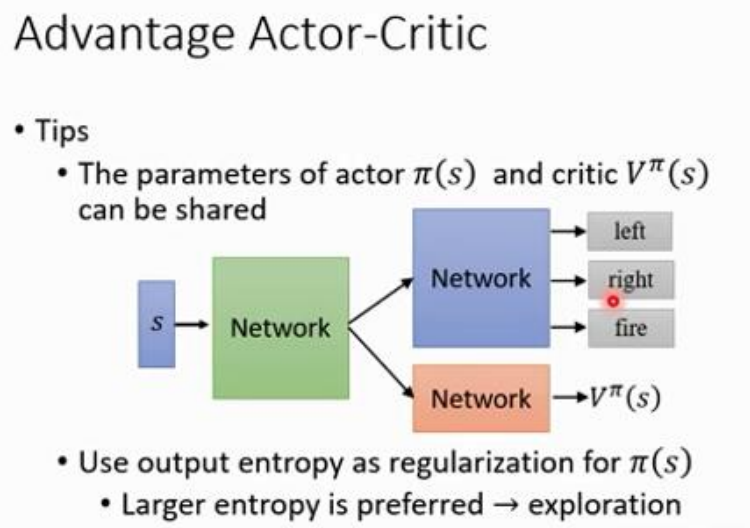
因此可得公式(4-5)

$$Q^{\pi_{\theta'}}(s_t^n, a_t^n) - V^{\pi_{\theta''}}(s_t^n) = r_t^n + V^{\pi}(s_{t+1}^n) - V^{\pi_{\theta''}}(s_t^n) \tag{4-4}$$

虽然这里得到的及时奖励 r 也是具有随机性的，但是相较于 G 来说这个随机性会比较好，因为 G 是所有 r 的总和，随机性更大。

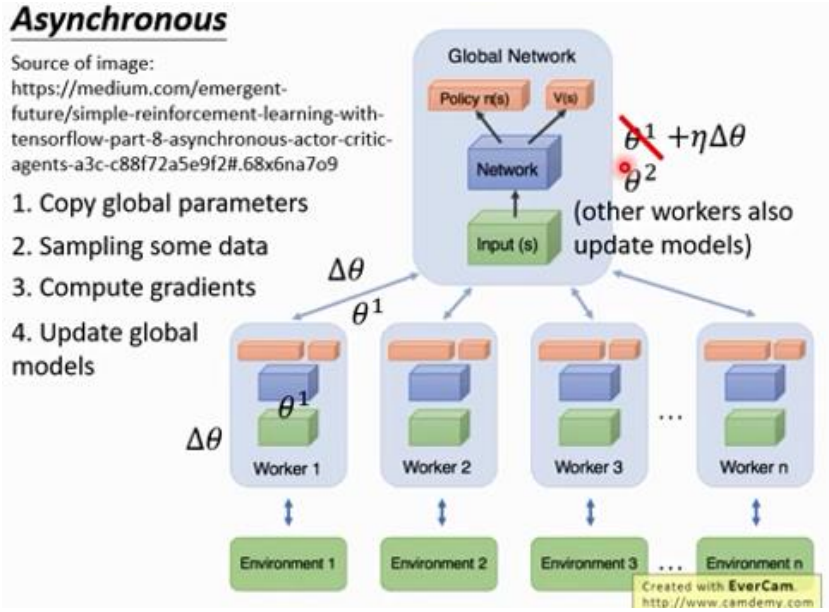


Tips



• Asynchronous Advantage Actor-Critic (A3C)

Reinforcement Learning 的一个问题就是它很慢，那么怎么加快它的速度呢？它就跟鸣人用影分身练武功一样，开很多的 worker，最后将学习到的资料汇总总结。



• Pathwise derivative policy gradient

在 Pathwise derivative policy gradient 里面，Critic 不只是告诉 Actor 采取的动作好或则不好，而是会直接告诉 Actor 采取什么样的动作才是好的。

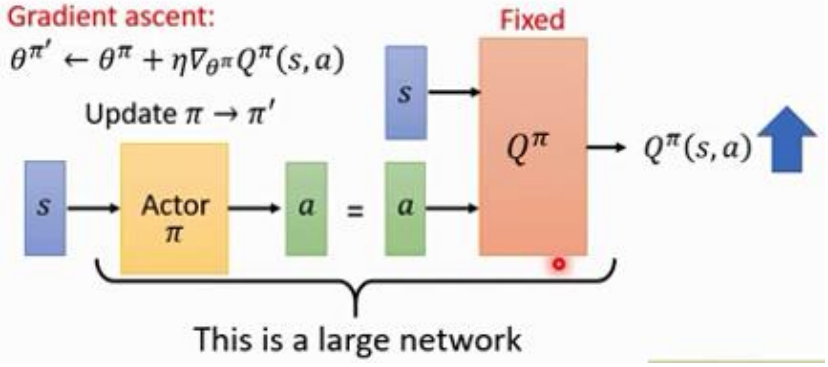
Pathwise Derivative Policy Gradient

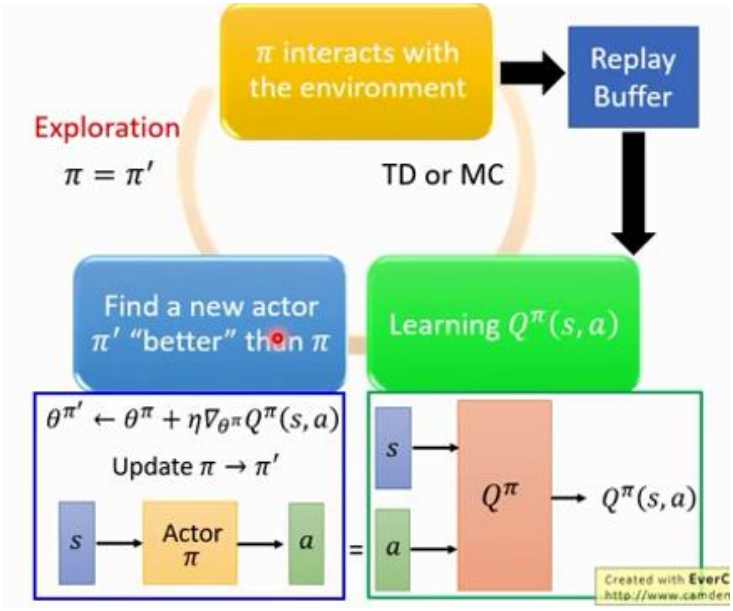
$\pi'(s) = arg\max_a Q^\pi(s,a)$ ← a is the output of an actor

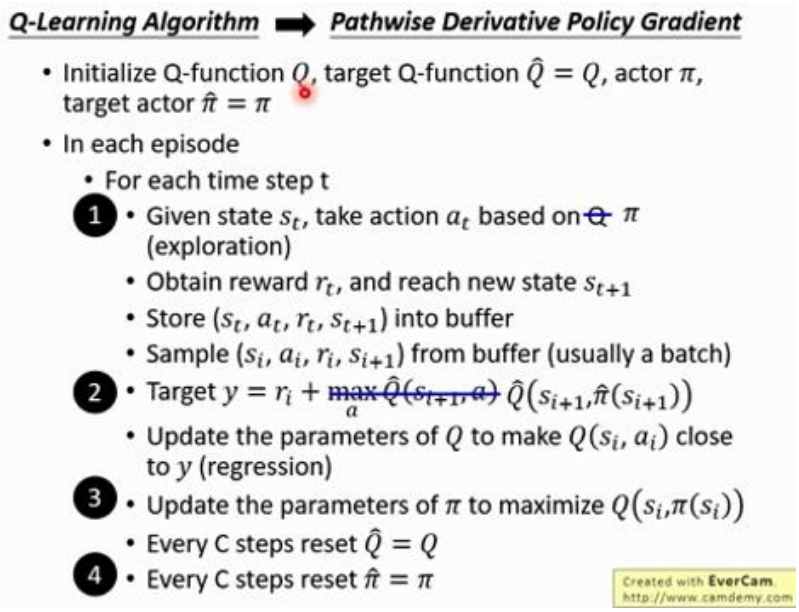
Gradient ascent:

$\theta^{\pi'} \leftarrow \theta^\pi + \eta \nabla_{\theta^\pi} Q^\pi(s,a)$

Update $\pi \rightarrow \pi'$







从上述可知，actor-critic 与 GAN 是非常像的，也有相关学者做了相关的学术研究，如下参考文献所示。

Pfau D, Vinyals O. Connecting Generative Adversarial Networks and Actor-Critic Methods[J]. 2016.

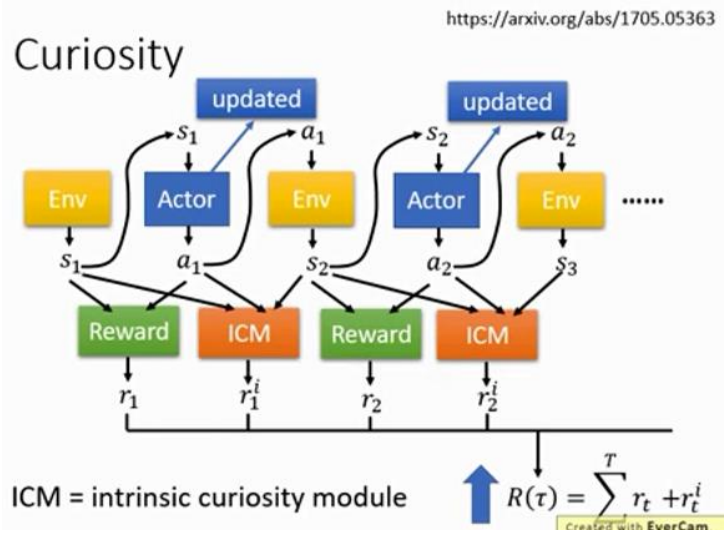
5. Sparse Reward

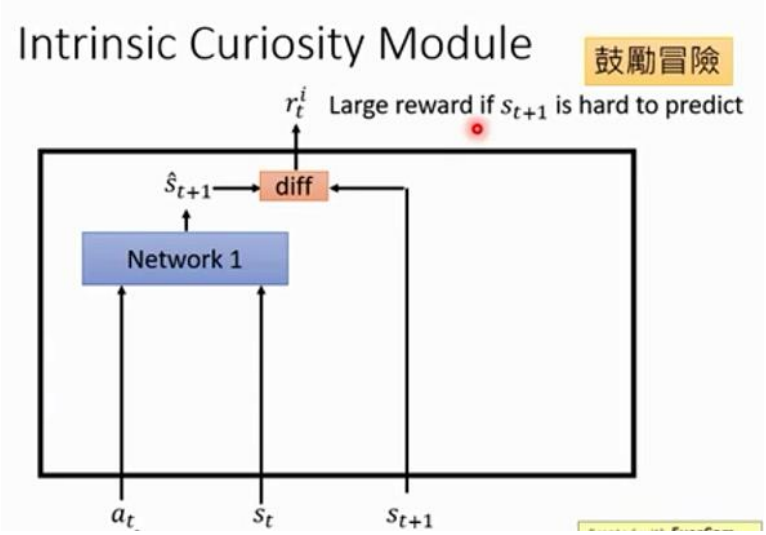
在我们实际训练强化学习智能体的大多数时候我们都没有办法获得 Reward 的，这样的话我们就很难训练这个 Agent。课程主要介绍了三个方法：

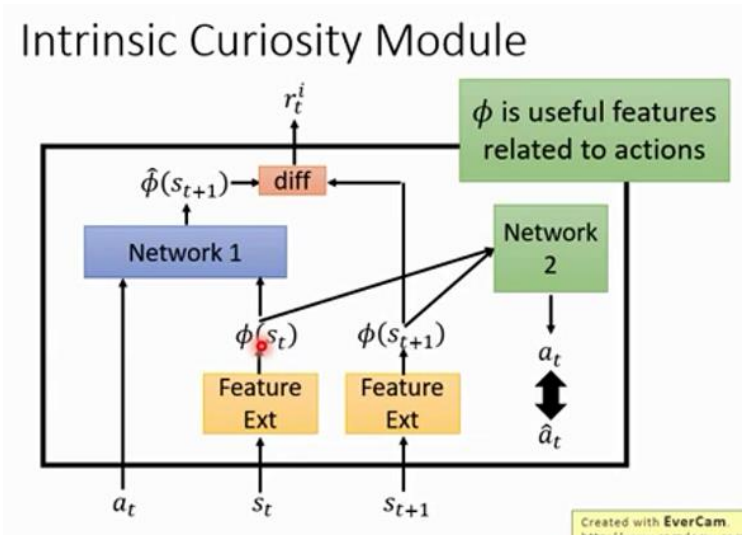
• Reward shaping

当环境只有最终的一个 Reward，为了能够使得 Reinforcement Agent 能够在这种情况下进行训练。我们会人为设置一些奖励去引导 Agent 去得到这个最终的奖励。举个例子来说呢就是：我们假设一个小孩是一个 Agent，他可以选择出去玩，这个时刻奖励是 1，但是在期末考试的时候成绩就会考不好，他的奖励可能就是-100；他也可以选择去念书，这个时刻奖励是-1，但是在期末考试的时候就有可能考很好，那么他的奖励就是+100。对于这种情况呢，小孩可能看不到那么远的事情，导致拿不到很高的奖励。这个时候就需要大人的引导，大人就跟小孩说，如果你现在坐下来念书，我就给你一个棒棒糖，那么这个时候学习的当前时刻的奖励就有可能是 Positive 的。

Reward shaping 的概念是一样的，我们给它一些 Reward，这些 Reward 并不是环境真正给他的 Reward，以此来引导 Machine 去做一些我们想要它做的事情。



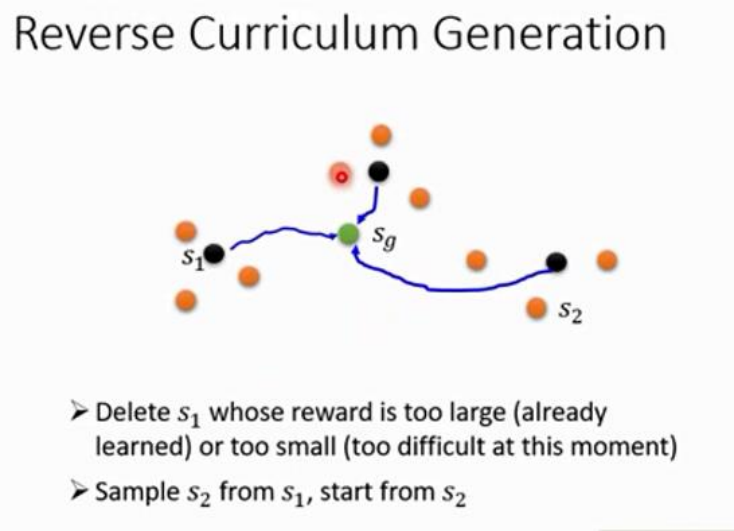




上图中的 Network 1 其实是另外 Train 出来的，它吃两个输入这个时刻的状态和这个状态下采取的动作，输出下一个状态，我们希望输出的下一个状态与真实的状态差距越大越好。但是有些动作是真的很难预测，但是却并不是非常重要，由此引来了下面的变种

- Curriculum Learning

Curriculum Learning 做的事情其实是给机器做规划，这样的一种方法也不是 Reinforcement Learning 所独有的，在 Machine Learning 和 Deep Learning 中也会经常被用到。那么所谓的 Curriculum Learning 其实是将喂给机器的数据做一个人为地排序，通常都是由简单到难。比如说教小孩子做微积分，做不对就打他一巴掌，这样的话就是打死他估计都学不起来，我们应该先教给他 99 乘法表，之后再慢慢一步一步教他。



• Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning 也叫做层次强化学习，所谓阶层次的强化学习，也就是说我们有好几个 Agent，有一些 Agent 负责高层次的东西，比如负责定目标，定完目标后再分配给其他的 Agent，去把相关的东西做完。这样的做法其实也是很合理的，比如说我们的人一生当中也不是时时刻刻在做决定。

6. Imitation Learning

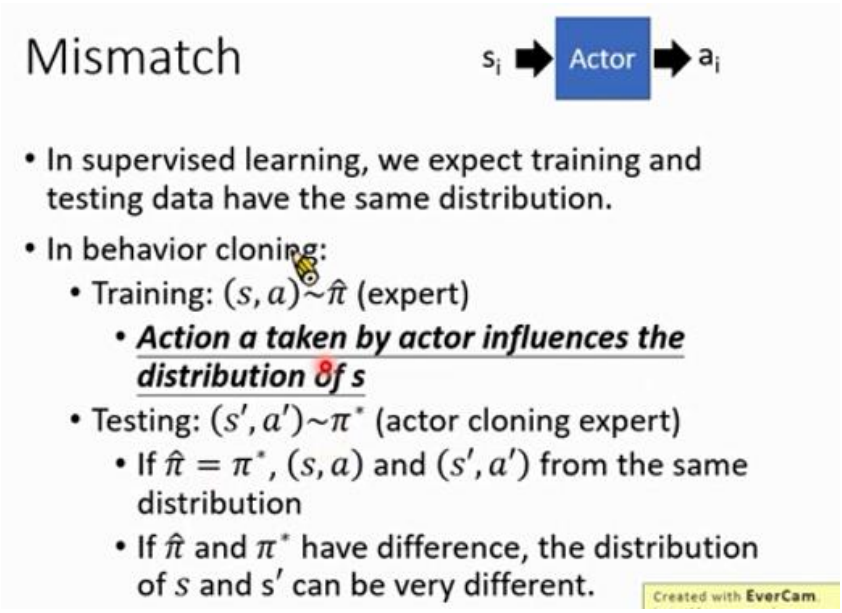
Imitation Learning 要讨论的问题是，假设我们今天的 Reward 都没有，那要如何解决这个问题呢？

- Imitation Learning
  - Also known as learning by demonstration,
  - Apprenticeship Learning
- An expert demonstrates how to solve the task
  - Machine can also interact with the environment, but cannot explicitly obtain reward
  - It is hard to define reward in some tasks.
  - Hand-crafted rewards can lead to uncontrolled behavior

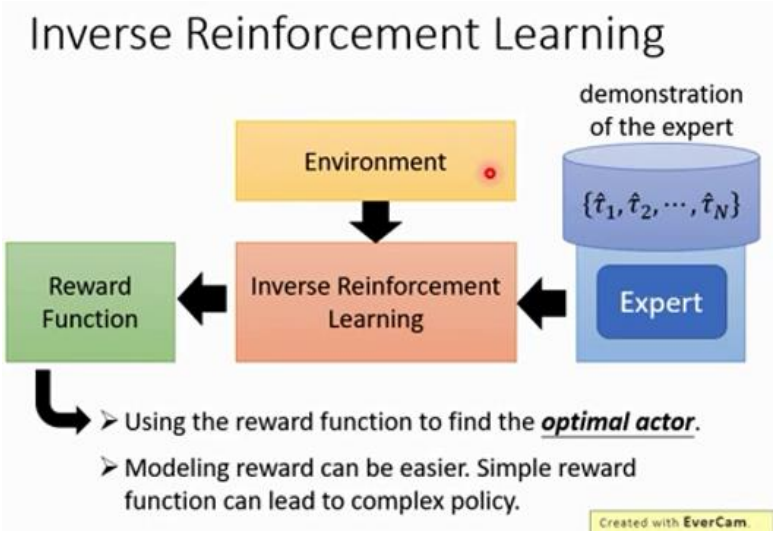
- Behavior Cloning

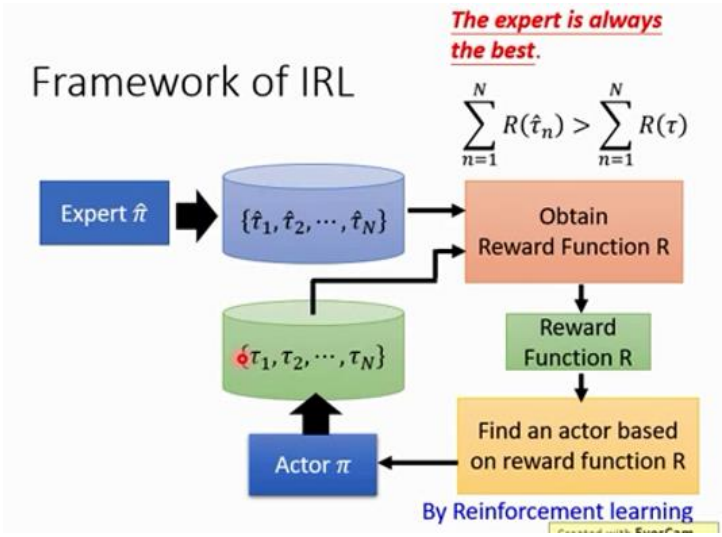
Behavior Cloning 与监督学习是一模一样的，举例来说就是在训练开自驾车的时候，我们可以收集到人开自驾车的资料，然后我们以 experts 的开车数据为机器的学习目标，让机器学地跟 experts 一模一样，那么我们如何让机器学习地跟 experts 一模一样呢？我们将其看作一个 Supervise Learning 的过程，我们先去收集很多数据(人在那样的情境下会采取什么样的行为)，接下来我们就 Learning 一个 Network，这个 Network 输入一个 state，输出一个 action，我们希望这个输出的 action 与 experts 采取的 action 越接近越好。虽然这种方法比较简单，但是这种方法会使得我们看过的 state 会是非常有限的。举例来说就是机器学习 experts 的经验开车，这个车从来都没有过快要撞墙的经验，因此当机器开到了快要撞墙的时候，机器不知道如何处理。

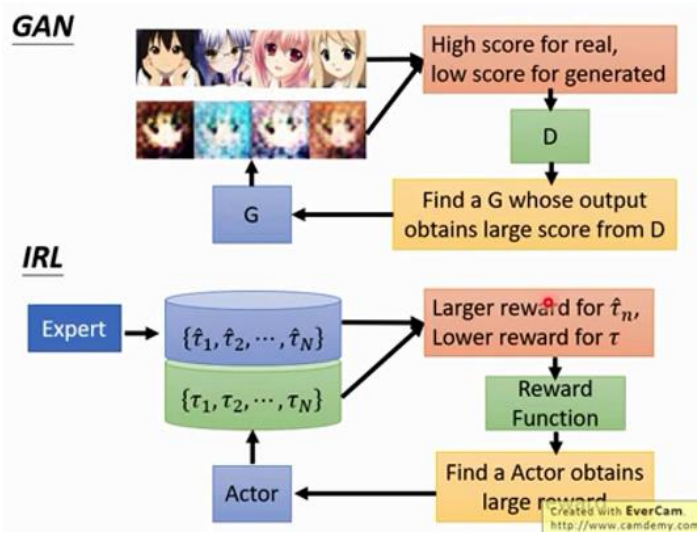
- Major problem: if machine has limited capacity, it may choose the wrong behavior to copy.
- Some behavior must copy, but some can be ignored.
  - Supervised learning takes all errors equally.



- Inverse Reinforcement Learning (IRL)







Stadie B C, Abbeel P, Sutskever I. Third-Person Imitation Learning[J]. 2017.