

深度强化学习笔记-台大李宏毅

李宏毅老师通过下面的地球跟机器人比喻RL (Reinforcement Learning) 过程是怎么回事。

地球是环境(environment)，代理(agent)用感测器去接收外接讯息，就像无人车在路上有六种以上装置感知外接讯息。

外边感知到了一杯水，它(agent)感知到讯息接着采取行动，它把水打翻了。因他的改变而外界有所改变，一摊水洒在地上。

接着外界（地球）给她了一个回馈：你刚刚的动作是不好的（Don't do that），所以机器人得到一个**负面回馈**。

Scenario of Reinforcement Learning



接着，机器人感测到地上有一滩水后，便采取行动——把地上水擦净，改变了外界的状态。

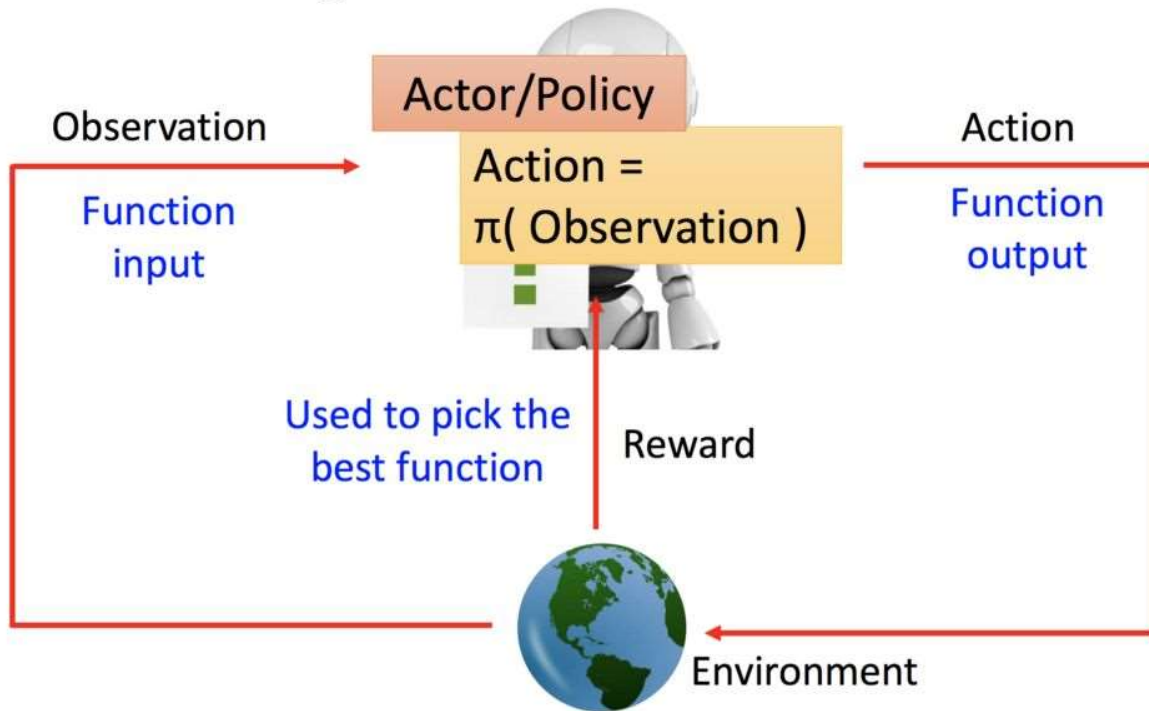
接着地球给了个回馈：干得好兄弟！这是一个**正面的奖励**，接着这个反馈机器人也接收起来了：我这个动作是好的。

Scenario of Reinforcement Learning



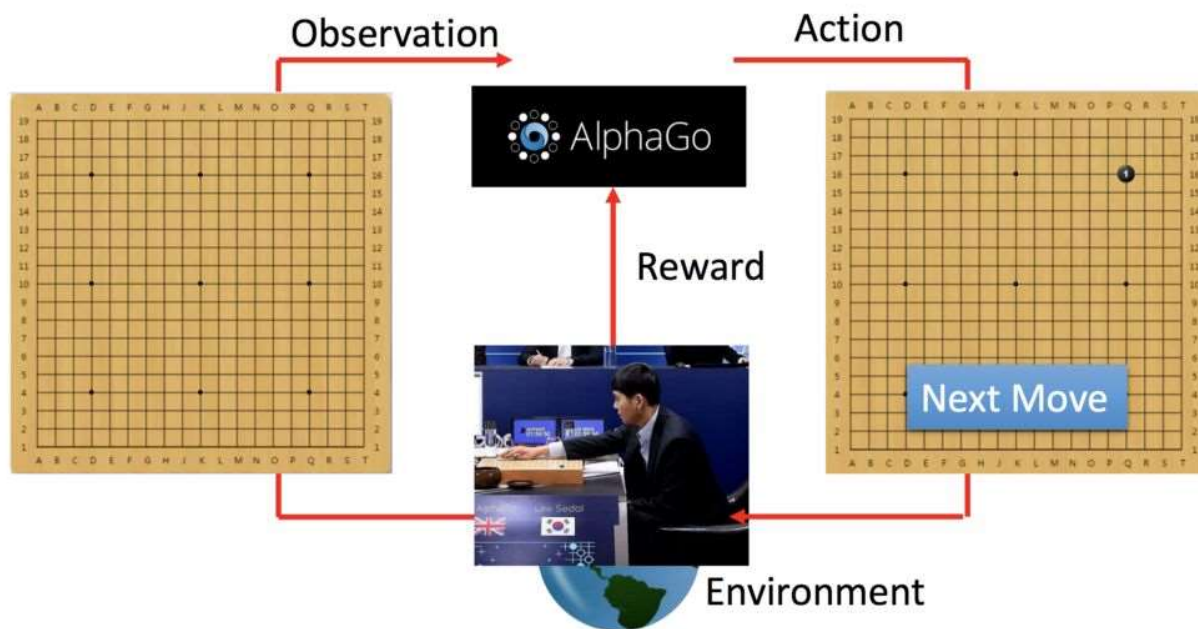
这里比喻机器的学习过程就是**找到一个函数**，函数的输入是外界（观察），而机器学习得目标就是要把这个函数（奖励）**最大化**。

Machine Learning ≈ Looking for a Function



这边举例阿法狗的学习过程。首先观测棋局（左），阿法狗下了一手。外部环境接收到了讯息，反馈给阿法狗。

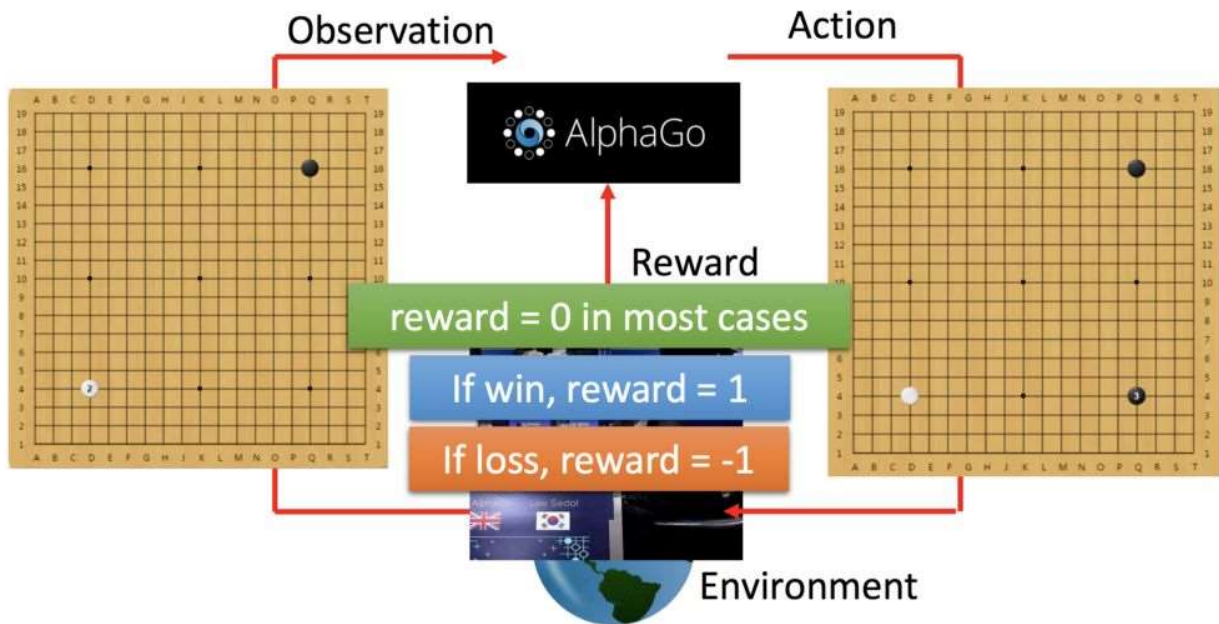
Learning to play Go



人类下了第一手，阿法狗观测棋盘，然后不断循环刚刚的步骤。整个过程奖励是0，直到棋局结束，才会产生1或0的奖励。

Learning to play Go

Agent learns to take actions maximizing expected reward.



假设是**监督式方法**让机器去学习，就会变成你教授5-5后，第二手教机器下3-3，一步一步的带下法。

但**强化学习**不一样，是到棋局结束才有奖励。

阿法狗的算法则是，监督式先学习许多的棋谱，然后才用强化学习去探索更多棋谱跟走法。

Learning to play Go

- Supervised: Learning from teacher



Next move:
"5-5"



Next move:
"3-3"

- Reinforcement Learning Learning from experience

First move → many moves → Win!
(Two agents play with each other.)

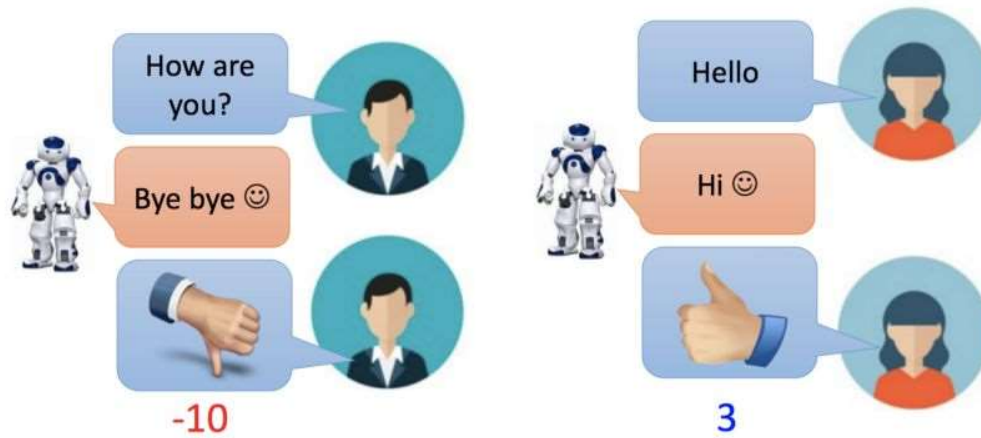
Alpha Go is supervised learning + reinforcement learning.

我们用语音机器人举例。一开始的监督则是从你一句我一句训练，然后根据动作奖励值，机器的目标就是要**最大化期望值**。

Learning a chat-bot

https://image.freepik.com/free-vector/variety-of-human-avatars_23-2147506285.jpg
http://www.freepik.com/free-vector/variety-of-human-avatars_766615.htm

- Machine obtains feedback from user



- Chat-bot learns to maximize the expected reward

如果像阿法狗一样，让两个机器人训练呢？那机器人就会不断的对话出很多的句子。

Learning a chat-bot

- Let two agents talk to each other (sometimes generate good dialogue, sometimes bad)



How old are you?



See you.



How old are you?



I am 16.



See you.



See you.



I though you were 12.



What make you think so?

产生的句子很多，也不可能一个一个去看，那就要采用监督式学习了。你可以制定一个规则，假如你希望一个机器人学习骂脏话，那就让输入的句子奖励都能得到正值，反之如果不希望，则加入规则，骂脏话的时候变的反馈负分。

Learning a chat-bot

- By this approach, we can generate a lot of dialogues.
- Use some pre-defined rules to evaluate the goodness of a dialogue



Machine learns from the evaluation

Deep Reinforcement Learning for Dialogue Generation
<https://arxiv.org/pdf/1606.01541v3.pdf>

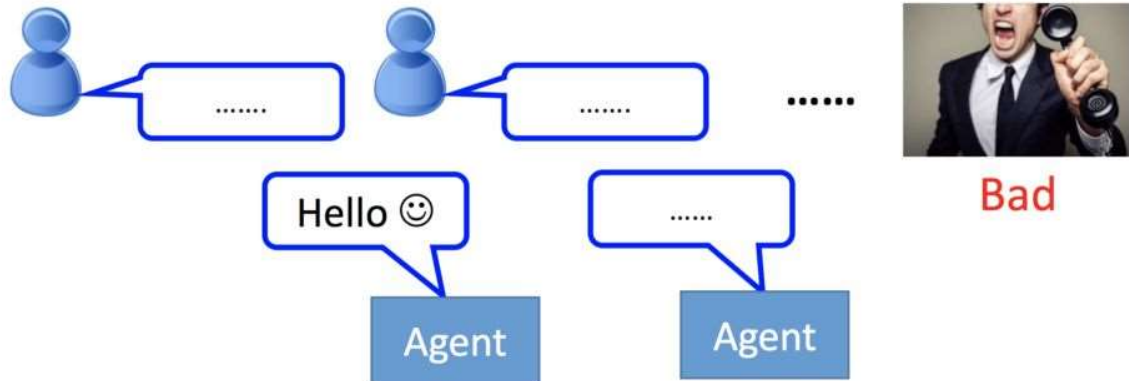
如果把语音机器人用监督和强化学习来比喻，非监督方式就是一句一句地教，强化学习就是**让机器自己去对话**，直到对方挂电话结束语音聊天。

Learning a chat-bot

- Supervised



- Reinforcement



以下是提供的两个RL环境，有空可以上去玩玩试试。接下来的内容大部分会以机器人玩游戏为主题做延伸。

Example: Playing Video Game

- Widely studies:

- Gym: <https://gym.openai.com/>
- Universe: <https://openai.com/blog/universe/>

Machine learns to play video games as human players

- What machine observes is pixels
- Machine learns to take proper action itself

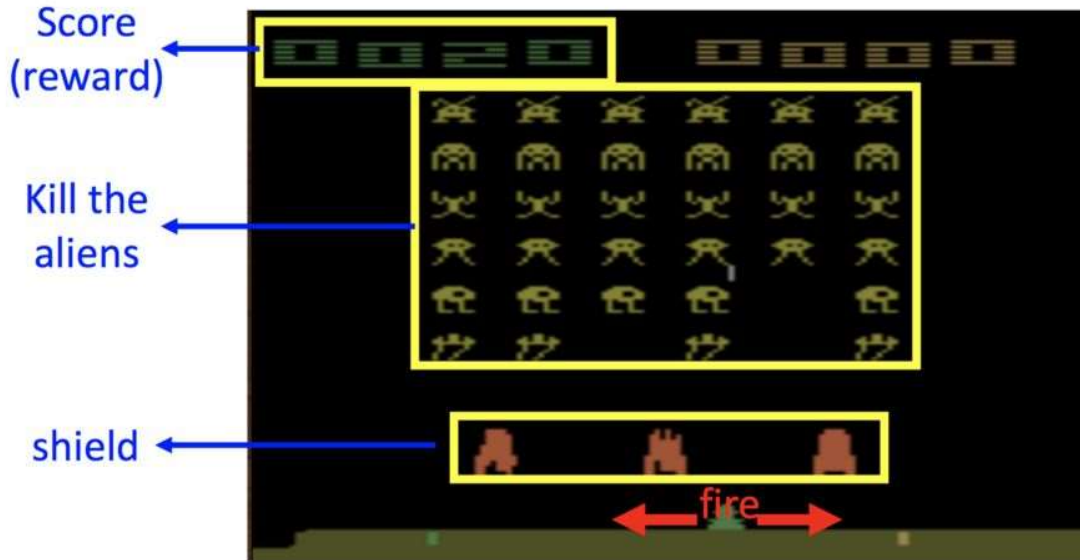


下面是一个用RL玩游戏的例子，左上方是已获得分数，中间是还没打完的怪，下方则是你可操作的动作，包括向左移动、向右移动以及开火。

Example: Playing Video Game

- Space invader

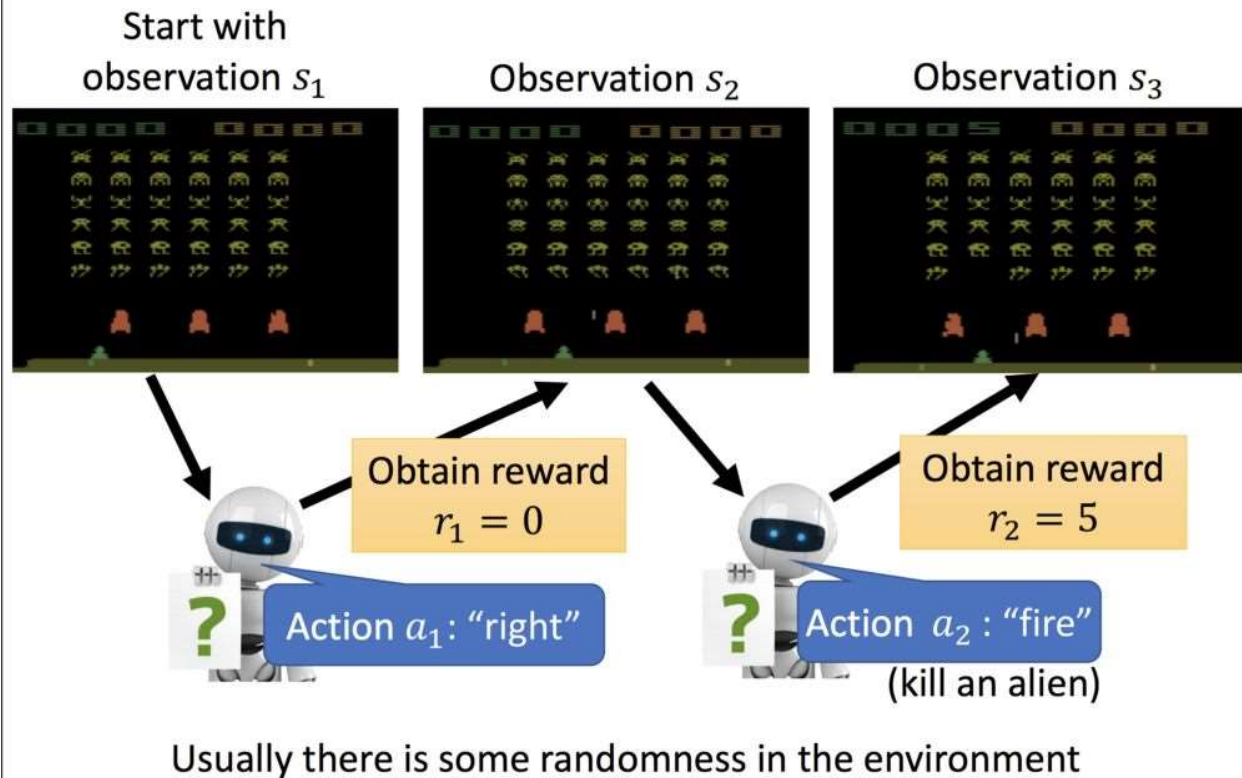
Termination: all the aliens are killed, or your spaceship is destroyed.



整个流程你可以这样了解[如何互通](#)。

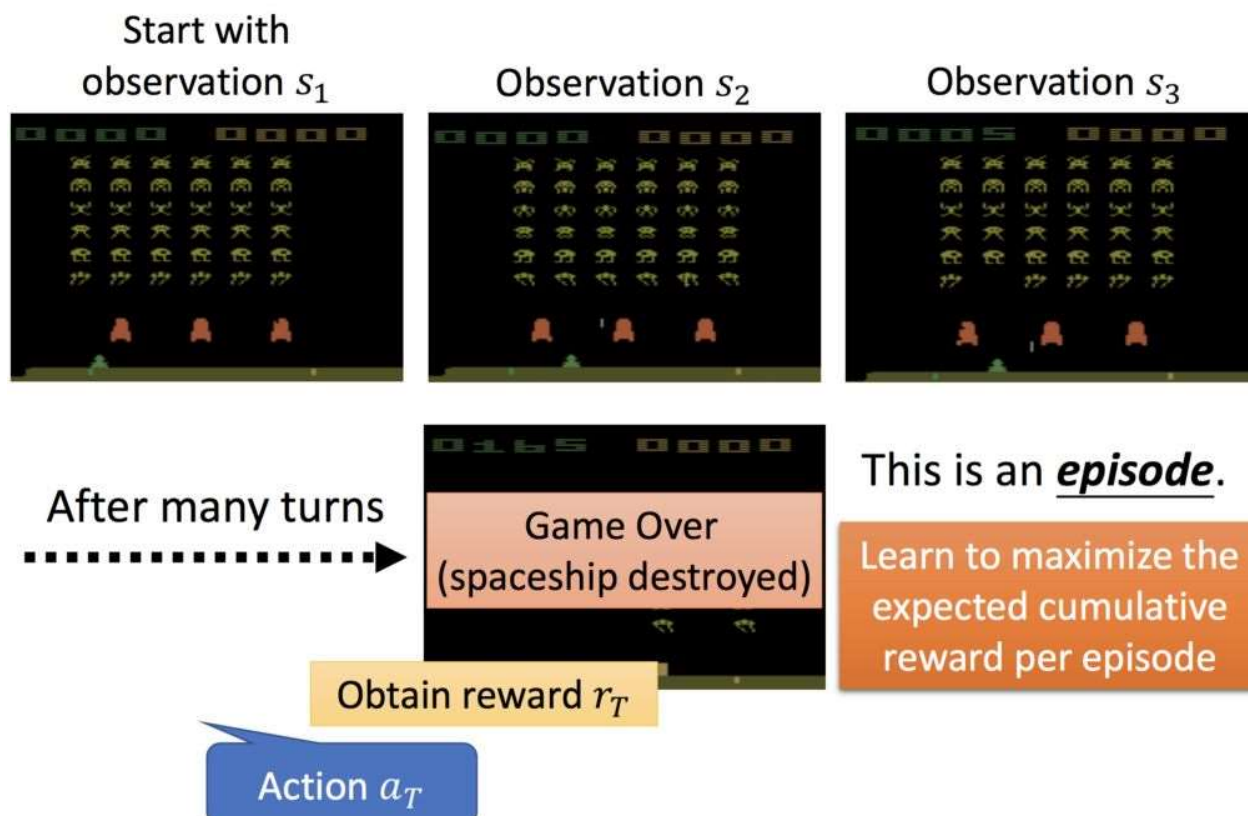
首先机器看到最左边的画面 (state s_1)，接着采取行动 (action a_1) 向右走一步，得到回馈 reward ($r_1 = 0$)，然后再接收状态资讯 (state s_2)，接着再选择开火 (action a_2)，然后环境给予他的回馈奖励 ($r_2 = 5$)， $s_1 \rightarrow a_1 \rightarrow r_1 \rightarrow s_2 \rightarrow a_2 \rightarrow r_2$ 。

Example: Playing Video Game



直到游戏结束，整个过程会得一个**累积的奖励**，游戏会以整个情节的奖励为目标，并按照目标最大化原则调整行为。

Example: Playing Video Game



目前强化学习有两个需要关注的特性。

首先是关于学习，有着奖励延迟的特性，你的机器人或许会知道开火跟得分有关系，但不能直接了解得分跟往右移动有什么关系，这样机器最后只会不断地开火。

再举个围棋的例子，在与环境对弈的过程，并不是每步都有明显的回馈说这步下得很好，有时早期的牺牲些区块，诱敌等战术都能让你在后面获得更好的期望利益，学习的对象是一连串的行为（轨迹），机器才能了解，有些没有及时奖励值也是很重要，目标是最大化整个过程的奖励。

另一个特性是，机器不是一开始便拥有标注好的资料，机器要跟环境持续做互动，改变环境获得反馈，玩许多次才会更新算法，过程整个这样持续。

Properties of Reinforcement Learning

- **Reward delay**
 - In space invader, only “fire” obtains reward
 - Although the moving before “fire” is important
 - In Go playing, it may be better to sacrifice immediate reward to gain more long-term reward
- Agent's actions **affect the subsequent data it receives**
 - E.g. Exploration



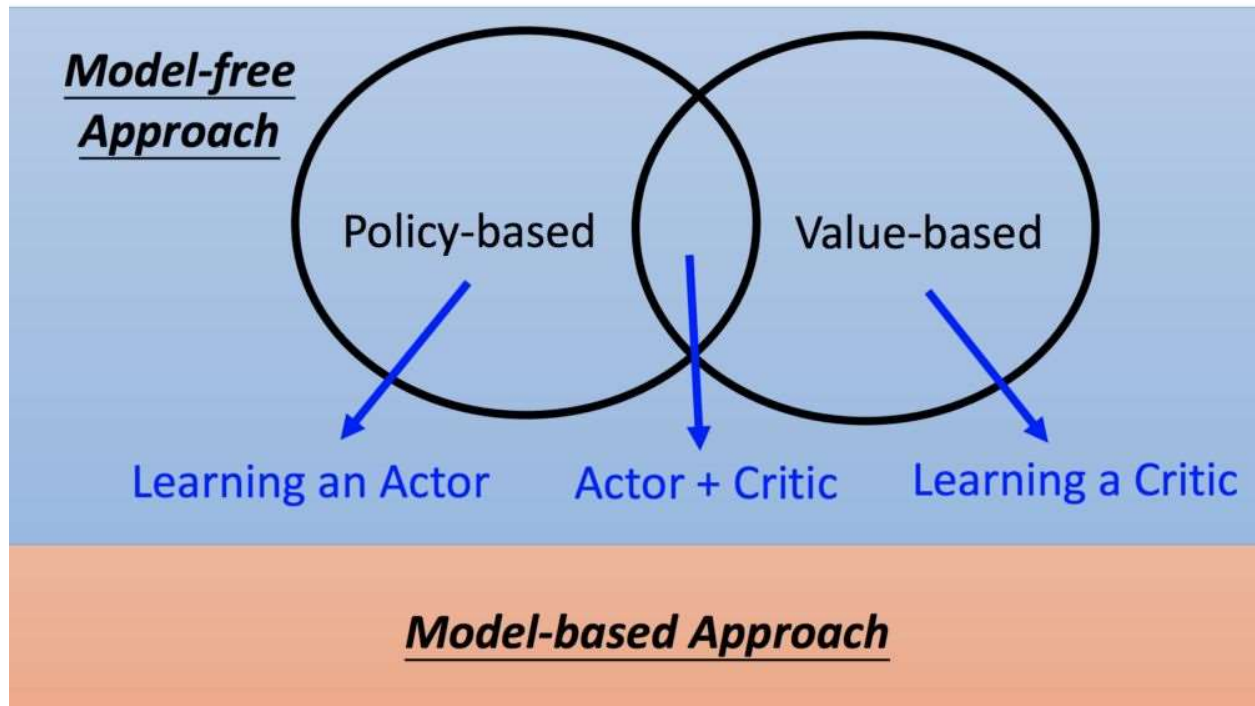
强化学习模型主要有两个，第一个是**模型的基础上**。

以围棋举例：你下一步后，机器便可以预想后面所有可能的棋步，然后推出胜率最大的下一步，但这是基于对规则与环境的充分理解，才有可能做到。

另外一个则是**无模型**，你并不是对环境很有着充分理解，基于这个产生两个方向，基于策略的和基于价值的，以及混杂的Actor+Critic。阿法狗可以参考，它是兼这三个类型使用。

Outline

Alpha Go: policy-based + value-based
+ model-based



接下来就开始介绍基于政策途径，如何得到一个好的Actor。

Policy-based Approach

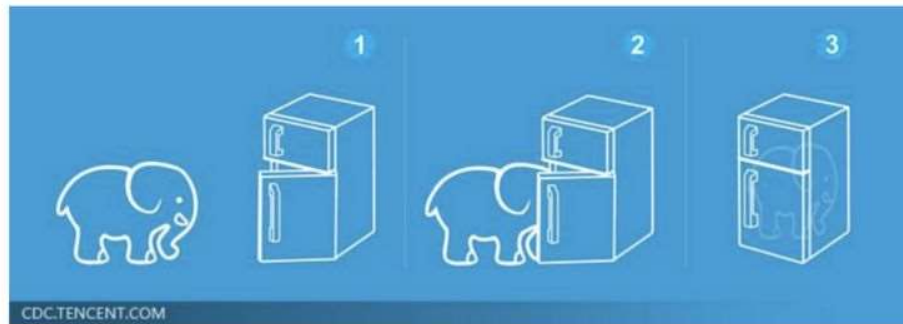
Learning an Actor

这边分三个部分介绍，RL导入NN(Neural Network)，如何定义好的函数及如何找出最好的。

Three Steps for Deep Learning



Deep Learning is so simple

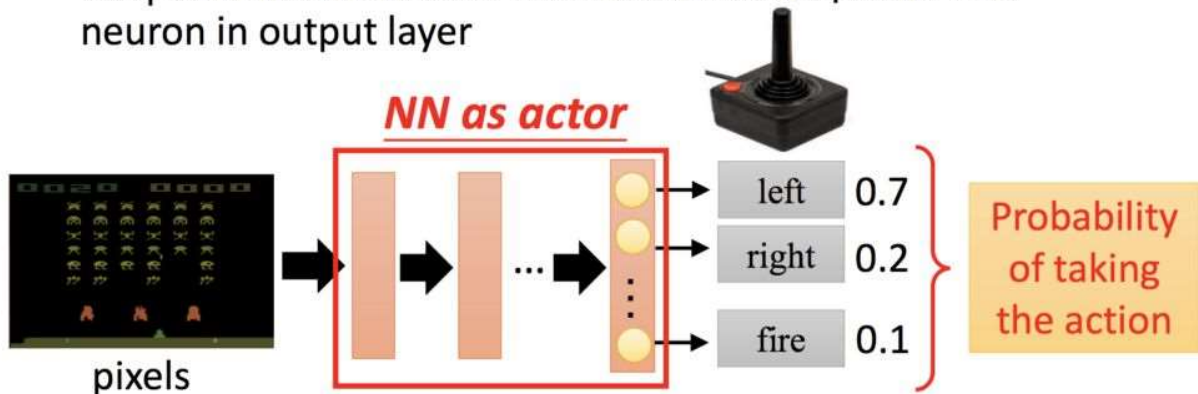


左下角可以看到整个游戏画面，进入NN会输出**三个维度**的结果，分别是三个动作的值。

其实过往RL就有些固定算法，例如Q-表，现在导入NN的原因是，原本的RL输入的内容必须比较固化，如果针对没看过的例子性能会较差，但NN优点就在于**泛化能力好**，就算画面没看到但仍会找到个看到且相似的画面，具有泛化特性。

Neural network as Actor

- 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



What is the benefit of using network instead of lookup table?

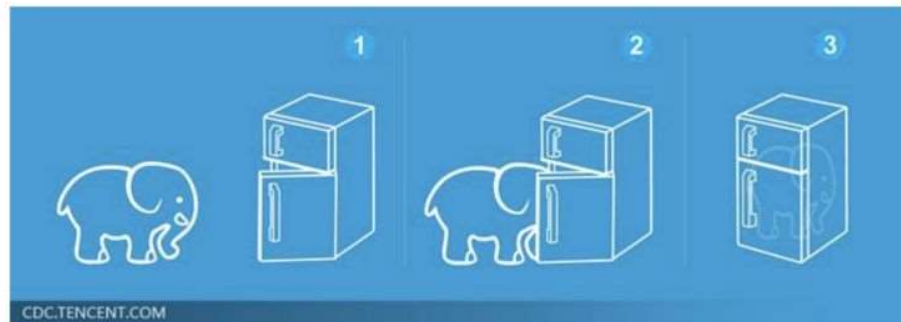
generalization

有了使用NN设计Actor的概念，接下来我们要来定义什么是好的函数。

Three Steps for Deep Learning



Deep Learning is so simple



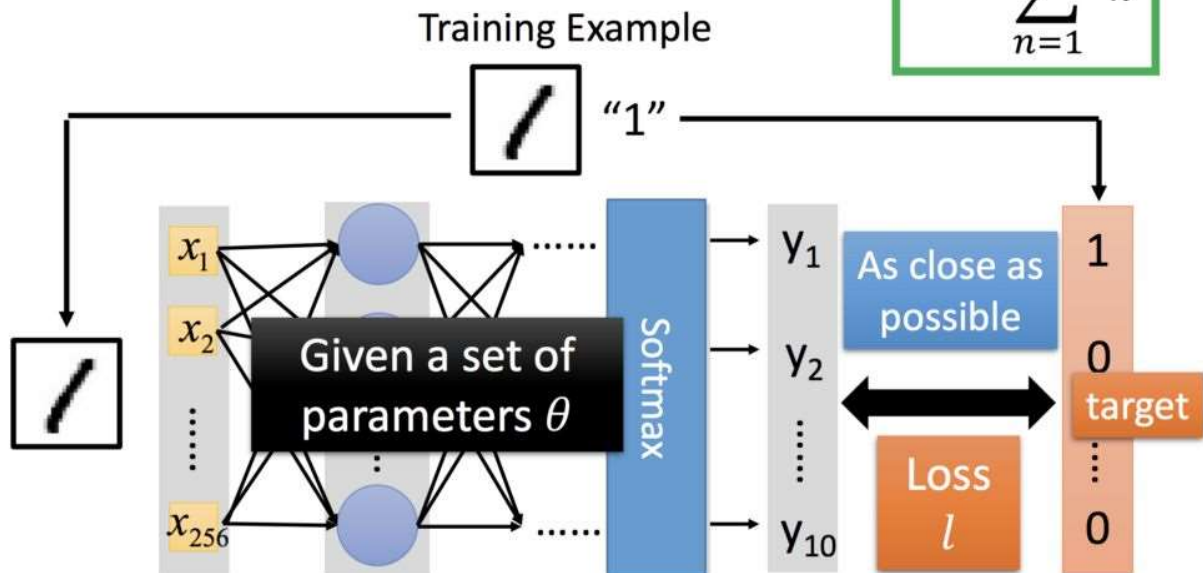
这是过去我们知道的分类问题：手写数字辨识经过神经网络，给定一个值，对照标签去评估损失。

Goodness of Actor

Total Loss:

$$L = \sum_{n=1}^N l_n$$

- Review: Supervised learning



函数 π (Actor) 会有一组参数 θ ，接着会先让Actor玩第一回游戏，整个过程（轨迹）结束会得到一个总奖励 R 。

对于相同的Actor来说，每次环境回馈的 R 并不一定相同，以及RL算法某些时刻会采取**随机的方式**选择策略，这是为了满足探索新的可能的需求。因为这些原因，我们会求 \mathcal{R} （注，应为右下角有下缀 θ ，暂时以 \mathcal{R} 代替）。求每次的机率与奖励，得期望值。

Goodness of Actor

- Given an actor $\pi_{\theta}(s)$ with network parameter θ
- Use the actor $\pi_{\theta}(s)$ to play the video game

- Start with observation s_1
- Machine decides to take a_1
- Machine obtains reward r_1
- Machine sees observation s_2
- Machine decides to take a_2
- Machine obtains reward r_2
- Machine sees observation s_3
-
- Machine decides to take a_T
- Machine obtains reward r_T

END

Total reward: $R_{\theta} = \sum_{t=1}^T r_t$

Even with the same actor,
 R_{θ} is different each time

Randomness in the actor
and the game

We define \bar{R}_{θ} as the
expected value of R_{θ}

\bar{R}_{θ} evaluates the goodness of an actor $\pi_{\theta}(s)$

我们知道想要的值是什么后，就先来求机率的公式。

首先定义 τ ，整个轨迹展开，求机率 $P(\tau|\theta)$ ，展开来后从第一项开始：环境初始状态 $P(S_1)$ ，在状态 (S_1) 状态下，基于 θ 所以采取的行动 (a_1) 中，接着基于 a_1 ，state1 (S_1) 过渡到状态state (S_2) ，中间所产生的奖励 (R_1) ，接着持续下去...

切到下方公式，除了 θ 外可拿掉，因为我们所关注的仅有参数。右下角是对于求出公式的理解
state1进入NN， $a = \text{文件的机率}$ 是0.7，另外则是对 $= 0.2$ ， $\text{left} = 0.1$ 的机率。

Goodness of Actor

We define \bar{R}_θ as the expected value of R_θ

$$\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_T, a_T, r_T\}$$

$$P(\tau|\theta) =$$

$$p(s_1)p(a_1|s_1, \theta)p(r_1, s_2|s_1, a_1)p(a_2|s_2, \theta)p(r_2, s_3|s_2, a_2) \dots$$

$$= p(s_1) \prod_{t=1}^T p(a_t|s_t, \theta)p(r_t, s_{t+1}|s_t, a_t)$$

Diagram illustrating the components of the probability distribution $P(\tau|\theta)$:

- $p(s_1)$ is labeled "not related to your actor" (yellow box).
- $p(a_t|s_t, \theta)$ is labeled "Control by your actor π_θ " (orange box).
- The diagram shows the state s_t entering the Actor π_θ , which outputs three actions: "left" (0.1), "right" (0.2), and "fire" (0.7).
- A grey box indicates the probability of the "fire" action: $p(a_t = \text{"fire"}|s_t, \theta) = 0.7$.

现在公式可表达每次的奖励值与机率，但延伸出另外一个问题：我们不太可能穷举所有的 τ ，找出所有可能性。

所以这边的替代方法是，让演员**玩N次游戏**，加起来后除N，作为奖励的期望值。

Goodness of Actor

- An episode is considered as a trajectory τ
 - $\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_T, a_T, r_T\}$
 - $R(\tau) = \sum_{t=1}^T r_t$
 - If you use an actor to play the game, each τ has a probability to be sampled
 - The probability depends on actor parameter θ :
 $P(\tau|\theta)$

$$\bar{R}_\theta = \sum_{\tau} R(\tau) P(\tau|\theta) \approx \frac{1}{N} \sum_{n=1}^N R(\tau^n)$$

Sum over all possible trajectory

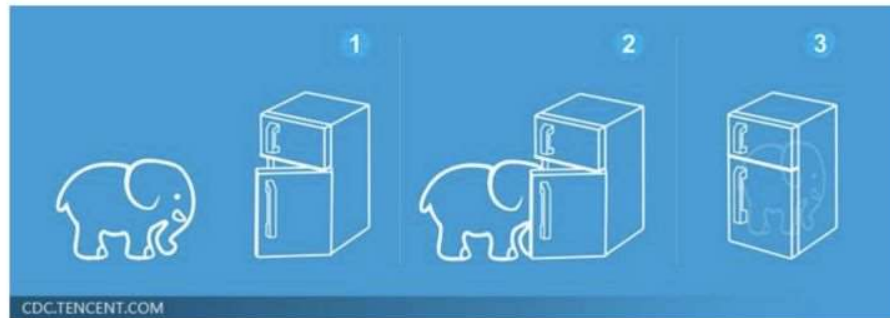
Use π_θ to play the game N times, obtain $\{\tau^1, \tau^2, \dots, \tau^N\}$
Sampling τ from $P(\tau|\theta)$ N times

接着我们要想办法找出最好的函数。

Three Steps for Deep Learning



Deep Learning is so simple



怎么定义出我们想要找的函数呢？只要 θ 能使得奖励最大化，便是我们想要的目标。这个一样需要**梯度**，右下是根据参数，我们要修改的 θ ，除了**权重**还包含偏差，右下角经过微分的向量，便是我们要更新的梯度。

Gradient Ascent

- Problem statement

$$\theta^* = \arg \max_{\theta} \bar{R}_{\theta}$$

- Gradient ascent

- Start with θ^0

- $\theta^1 \leftarrow \theta^0 + \eta \nabla \bar{R}_{\theta^0}$

- $\theta^2 \leftarrow \theta^1 + \eta \nabla \bar{R}_{\theta^1}$

-

$$\theta = \{w_1, w_2, \dots, b_1, \dots\}$$

$$\nabla \bar{R}_{\theta} = \begin{bmatrix} \partial \bar{R}_{\theta} / \partial w_1 \\ \partial \bar{R}_{\theta} / \partial w_2 \\ \vdots \\ \partial \bar{R}_{\theta} / \partial b_1 \\ \vdots \end{bmatrix}$$

这里如何去求梯度呢？

我们现在的目标是 $\nabla \mathcal{R}$ ，公式就是原本的**奖励乘机率**，但机率前面加sum，这样没法直接求值，这里先乘一个 τ 的机率以及除一个 τ 的机率。分子分母的部分带微分，然后左边的部分从sum所有轨迹替换成样本N次，蓝线部分就是 τ 的n次方的机率求log与 ∇ 。

Policy Gradient

$$\bar{R}_\theta = \sum_{\tau} R(\tau)P(\tau|\theta) \quad \nabla \bar{R}_\theta = ?$$

$$\nabla \bar{R}_\theta = \sum_{\tau} R(\tau) \nabla P(\tau|\theta) = \sum_{\tau} R(\tau) P(\tau|\theta) \frac{\nabla P(\tau|\theta)}{P(\tau|\theta)}$$

$R(\tau)$ do not have to be differentiable

It can even be a black box.

$$= \sum_{\tau} R(\tau) P(\tau|\theta) \nabla \log P(\tau|\theta) \quad \boxed{\frac{d \log(f(x))}{dx} = \frac{1}{f(x)} \frac{df(x)}{dx}}$$

$$\approx \frac{1}{N} \sum_{n=1}^N R(\tau^n) \nabla \log P(\tau^n|\theta)$$

Use π_θ to play the game N times,
Obtain $\{\tau^1, \tau^2, \dots, \tau^N\}$

τ 的机率求log, 怎么解? 这边一样从轨迹展开, 每一项带机率, 然后求值。我们求有关参数的项就好, 其他去掉, 就可找出值。

Policy Gradient

$$\nabla \log P(\tau|\theta) = ?$$

$$\bullet \tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_T, a_T, r_T\}$$

$$P(\tau|\theta) = p(s_1) \prod_{t=1}^T p(a_t|s_t, \theta) p(r_t, s_{t+1}|s_t, a_t)$$

$$\log P(\tau|\theta)$$

$$= \log p(s_1) + \sum_{t=1}^T \log p(a_t|s_t, \theta) + \log p(r_t, s_{t+1}|s_t, a_t)$$

$$\nabla \log P(\tau|\theta) = \sum_{t=1}^T \nabla \log p(a_t|s_t, \theta)$$

Ignore the terms
not related to θ

这就是整个**参数梯度更新**的方法。

下面的式子都可以与前面求得的带入。可以从物理方面去理解，如果你的回馈是正的，便可以改变参数，让其对这个state采取的行动机率提高，负值的话则反之。

Policy Gradient


$$\theta^{new} \leftarrow \theta^{old} + \eta \nabla \bar{R}_{\theta^{old}}$$

$$\begin{aligned} & \nabla \log P(\tau|\theta) \\ &= \sum_{t=1}^T \nabla \log p(a_t|s_t, \theta) \end{aligned}$$

$$\begin{aligned} \nabla \bar{R}_{\theta} &\approx \frac{1}{N} \sum_{n=1}^N R(\tau^n) \nabla \log P(\tau^n|\theta) = \frac{1}{N} \sum_{n=1}^N R(\tau^n) \sum_{t=1}^{T_n} \nabla \log p(a_t^n|s_t^n, \theta) \\ &= \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p(a_t^n|s_t^n, \theta) \end{aligned}$$

What if we replace $R(\tau^n)$ with r_t^n

If in τ^n machine takes a_t^n when seeing s_t^n in

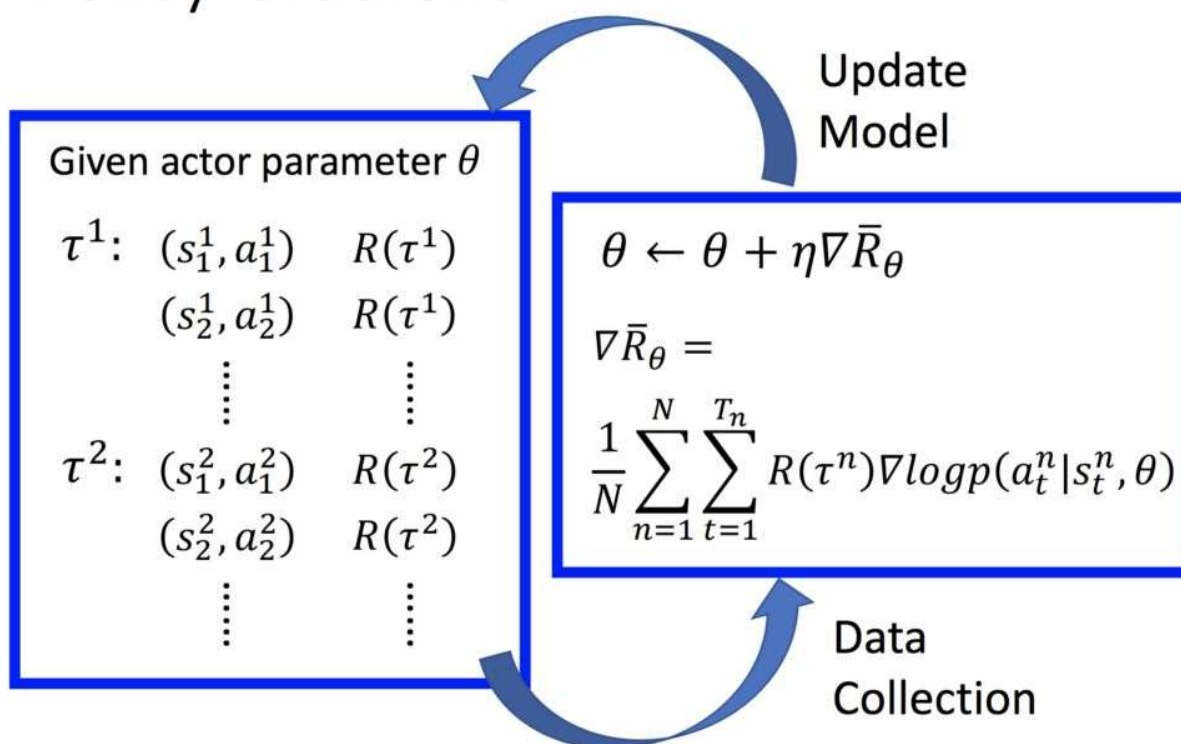
$R(\tau^n)$ is positive  Tuning θ to increase $p(a_t^n|s_t^n)$

$R(\tau^n)$ is negative  Tuning θ to decrease $p(a_t^n|s_t^n)$

It is very important to consider the cumulative reward $R(\tau^n)$ of the whole trajectory τ^n instead of immediate reward r_t^n

理解完公式，就是整个循环了。不过RL都是玩好几次游戏，再一次回头列出参数，比起其他AI应用，强化学习过程挺花时间的。

Policy Gradient



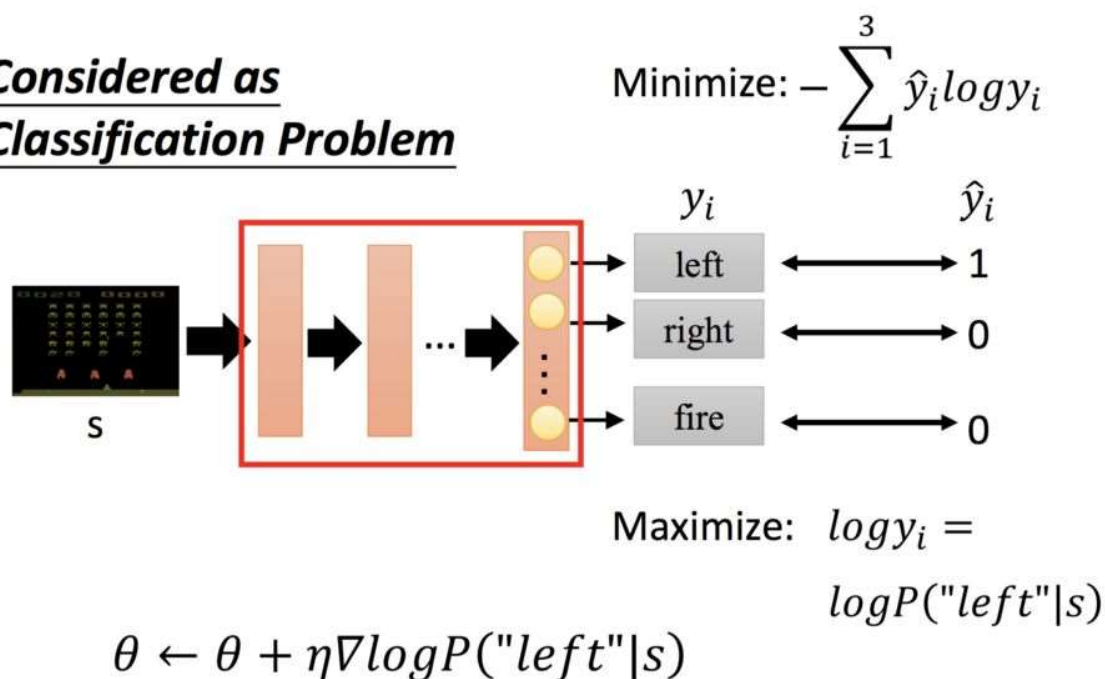
我们可以换个角度，用平常看到的**分类模型**来思考。

假设左边是游戏画面，输入到了神经网络中，输出了分别**三个维度**的动作，我们希望他这个画面产生的动作是往左边，值便给1。

过去的分类我们会用cross entropy计算，希望它最小化，这里的话则是希望这个机率最大化，针对状态采取的动作，便可以对参数做**梯度修正**。

Policy Gradient

Considered as
Classification Problem



其实这个公式把奖励拿掉，会发现跟分类模型差不多，状态1进入NN输出三维的资讯，左边的值为1，状态2进入NN，也是输出三维的资讯，值为1。

Policy Gradient

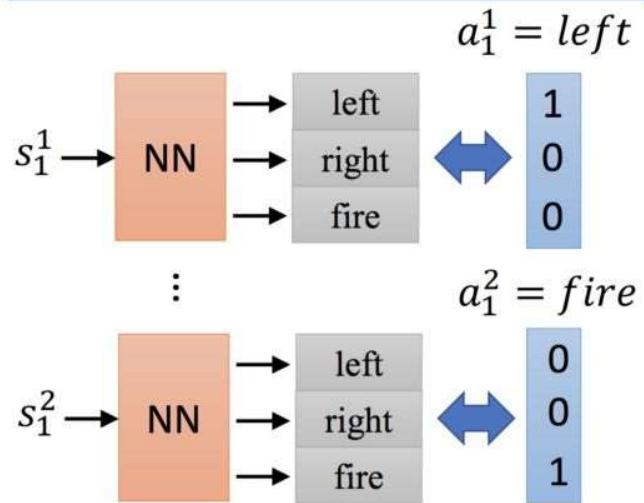
Given actor parameter θ

$\tau^1: (s_1^1, a_1^1) \quad R(\tau^1)$
 $(s_2^1, a_2^1) \quad R(\tau^1)$
 \vdots
 $\tau^2: (s_1^2, a_1^2) \quad R(\tau^2)$
 $(s_2^2, a_2^2) \quad R(\tau^2)$
 \vdots

$$\theta \leftarrow \theta + \eta \nabla \bar{R}_\theta$$

$$\nabla \bar{R}_\theta =$$

$$\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} \boxed{} \nabla \log p(a_t^n | s_t^n, \theta)$$



有无奖励的差别在哪里？

如果把奖励当作常数项，它实际上就是针对这个状态动作乘一个值，例如 τ 的奖励为2，则 s_1 至 a_1 就会产生两次，state2的奖励为1，则只会乘1。

Policy Gradient

Given actor parameter θ

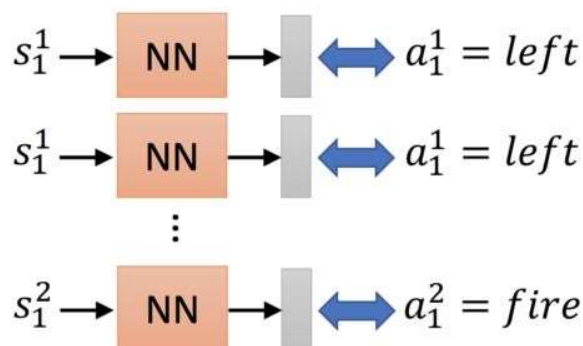
τ^1 :	(s_1^1, a_1^1)	$R(\tau^1)$	2
	(s_2^1, a_2^1)	$R(\tau^1)$	2
	\vdots	\vdots	
τ^2 :	(s_1^2, a_1^2)	$R(\tau^2)$	1
	(s_2^2, a_2^2)	$R(\tau^2)$	1
	\vdots	\vdots	

$$\theta \leftarrow \theta + \eta \nabla \bar{R}_\theta$$

$$\nabla \bar{R}_\theta =$$

$$\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p(a_t^n | s_t^n, \theta)$$

Each training data is weighted by $R(\tau^n)$



这边说明一个要注意的地方。理想上，A，B，C三个动作皆产生奖励然后修正。

你看到理想的地方，虽然幅度不一样，但其实都有调升，但因机率值关系，三者会再加起来当作分母，加起来总合一定会是1。

现在延伸的问题是，如果B，C有更新，但一个没有呢？

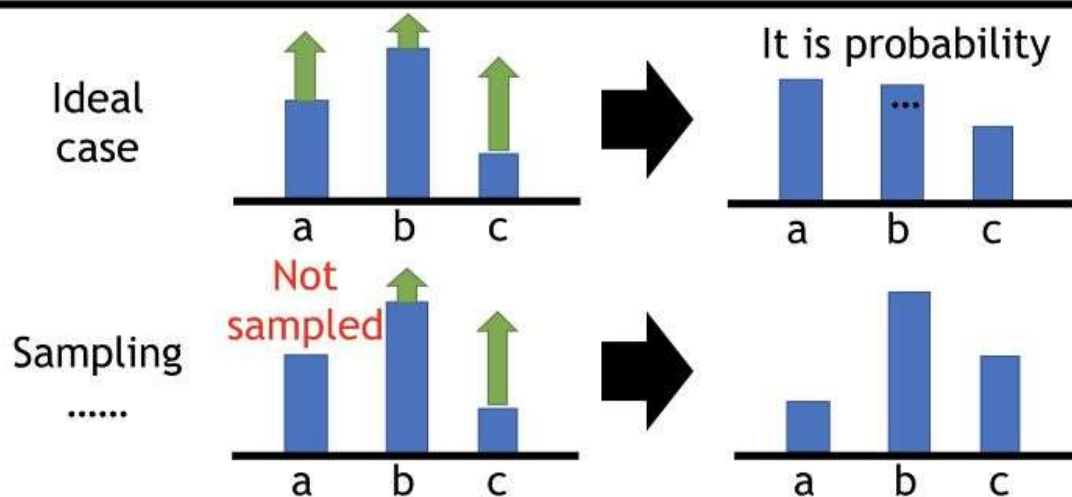
一个值会下降，因为他们最终会除总和的关系。那这应对的方法就是减去一个b值，这样奖励出来如果是正的，减去b值则有可能会变成正值，也可能是负值。

Add a Baseline

It is possible that $R(\tau^n)$ is always positive.

$$\theta^{new} \leftarrow \theta^{old} + \eta \nabla \bar{R}_{\theta^{old}}$$

$$\nabla \bar{R}_{\theta} \approx \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p(a_t^n | s_t^n, \theta)$$



接下来开始介绍基于价值，怎么去训练一个Critic。

Value-based Approach

Learning a Critic

Critic是什么呢？Critic并不会跟你的训练过程有直接关系，它要做的是评估一个Actor的好坏，好的Actor会由Critic挑出，Q-学习就是这样的方法。

Critic

- A critic does not determine the action.
- Given an actor π , it evaluates the how good the actor is

An actor can be found from a critic.

e.g. Q-learning



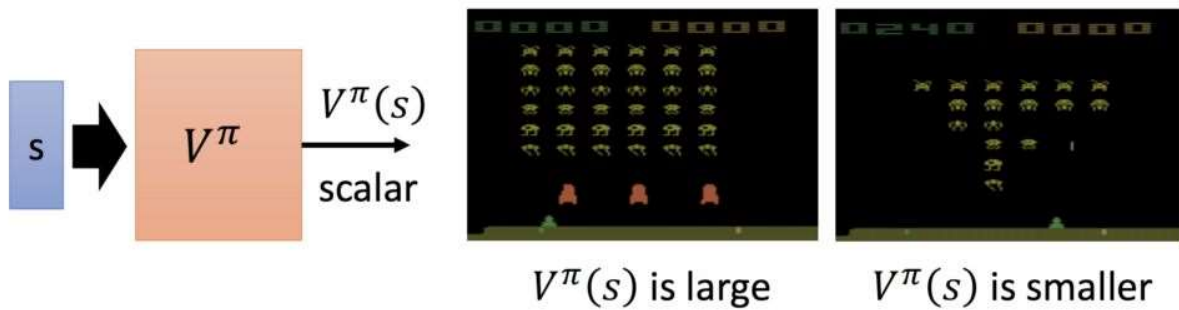
<http://combiboilersleeds.com/picaso/critics/critics-4.html>

评论的价值函数V是怎么评估一个值的呢？

V评估的方法是输入进你的前状态，然后给出后面会累积奖励的值。可以看下图理解，如果是游戏还没开始多久，画面上可得分的目标还挺多，V输出的值便会很大。如果目标已经被击落的差不多了V值便会比较小。但这前提是你的Actor够强，如果Actor在前面阶段便被射中，当然V值也会较小。

Critic

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



用棋灵王的故事举个例子。佐为 (Critic) 告诉阿光 (Actor) , 这个大马步飞的棋步不好, 理由是风险比较高。

但过了段时间阿光变强了, 佐为反而告诉他是好的, 因为现在阿光能力变好了, 这棋步虽较有风险, 但能带来好的获益。

Critic

V 以前的阿光(大馬步飛) = bad

V 變強的阿光(大馬步飛) = good



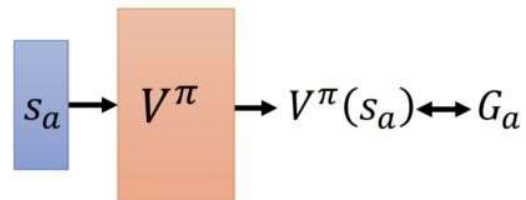
关于V的评估有两种方法，第一种是**Monte-Carlo**，MC就是对于你输入的状态，会把未来积累的奖励输出来。

How to estimate $V^\pi(s)$

- Monte-Carlo based approach
 - The critic watches π playing the game

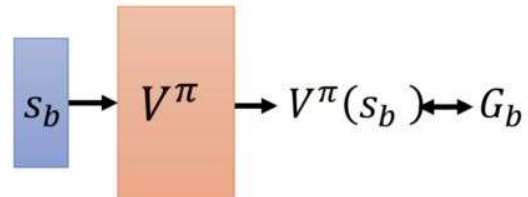
After seeing s_a ,

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

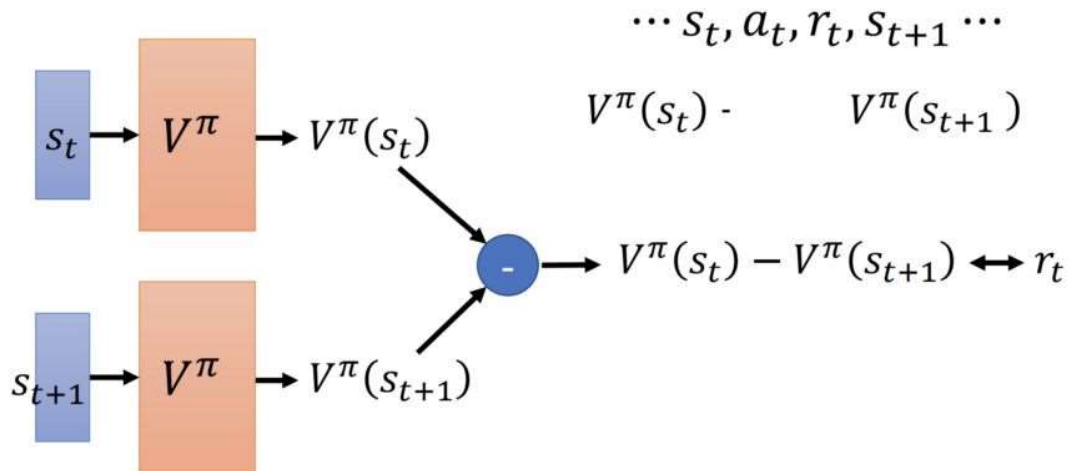


这里是第二个方法，**Temporal-difference**.

TD的做法是输入两个状态，接着从这两个状态中间求出reward。TD的场景比较偏重于，如果这个训练是较长比较少停止的，例如训练机器人走路，终局的奖励比较取得，使用这种取得两边输出的方法求奖励。

How to estimate $V^\pi(s)$

- Temporal-difference approach

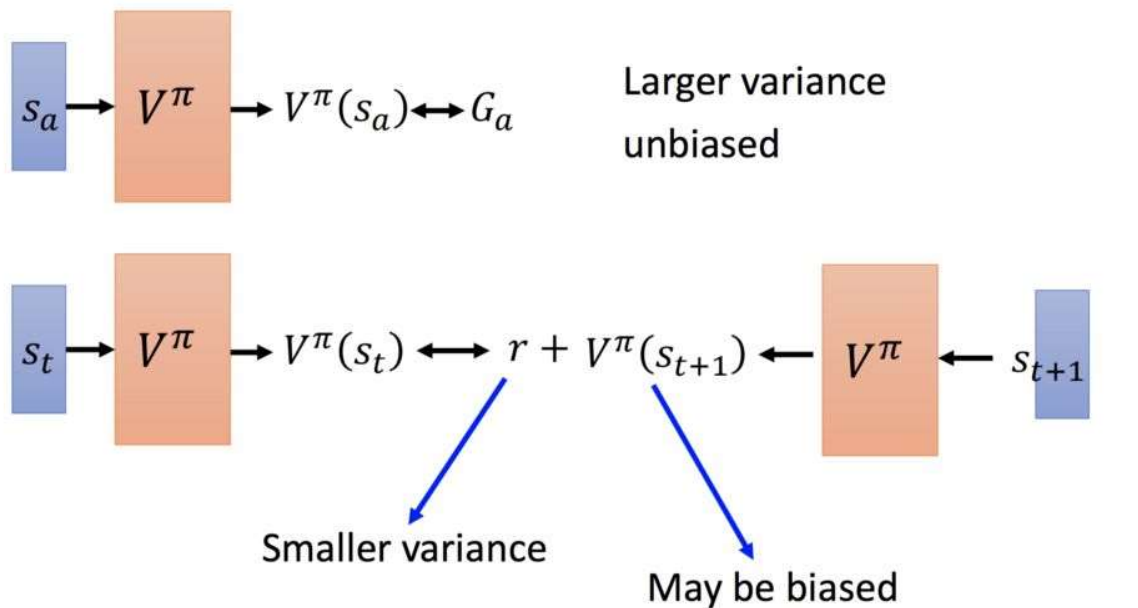


Some applications have very long episodes, so that delaying all learning until an episode's end is too slow.

MC跟TD的差别在于，MC的方法因为是累积的奖励，奖励本身因跟环境互动，与自身随机策略因素关系，较有**不确定性**，间接有着**高方差**。但相对的会有**无偏**的特性。

TD的方法因为直接求出**R**值，得到的方差会较小，但因为TD的V较为不确定的关系，值有可能是有偏也可能是无偏的。

MC v.s. TD



比较两者。先忽略动作，假设第一次的 τ 是状态a，奖励= 0，接着状态b，奖励= 0，结束。

另外有七次的 τ ，都是状态b，六次的奖励= 1，一次的奖励= 0。

这里评估 V （状态b）可以很快地得出 $6/8 = 3/4$ 的值，但 V （状态a）可以得出两个值，这就基于看是哪种评估方法。

如果是MC的话， V （状态a）最直接看出来就可求得为0.但如果是TD的话，可以看到下面公式
 V （状态b）+奖励= V （状态a）， V （状态b）= $3/4$ ，奖励是0，那个 V （状态a）不就也等于 $3/4$ 了吗？其实这两个都是对的，仅是方式不一样而已，再来也有可能样本不够充足，或许 V （状态a）是等于 $3/4$ 。

这边要注意的是，或许看到第一个 τ 会怀疑 V （状态b）是因为前面有状态a的关系，但TD的特性是**前后不会受到影响的**。

MC v.s. TD

[Sutton, v2,
Example 6.4]

- The critic has the following 8 episodes

- $s_a, r = 0, s_b, r = 0, \text{END}$

- $s_b, r = 1, \text{END}$

$$V^\pi(s_b) = 3/4$$

- $s_b, r = 1, \text{END}$

- $s_b, r = 1, \text{END}$

$$V^\pi(s_a) = ? \quad 0? \quad 3/4?$$

- $s_b, r = 1, \text{END}$

- $s_b, r = 1, \text{END}$

Monte-Carlo: $V^\pi(s_a) = 0$

- $s_b, r = 1, \text{END}$

Temporal-difference:

- $s_b, r = 0, \text{END}$

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

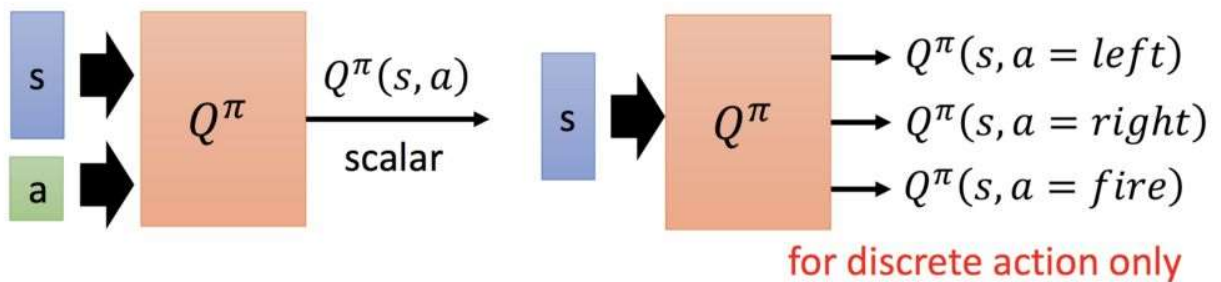
$$3/4 \quad 0 \quad 3/4$$

(The actions are ignored here.)

再来介绍另一个Critic，这就是有众所皆知的Q函数。跟前面两者的差别是，在输入的部分会加一个动作去计算值，右下角也是一样，只是在输出改成三维的资讯去求值，左右道理其实一样。

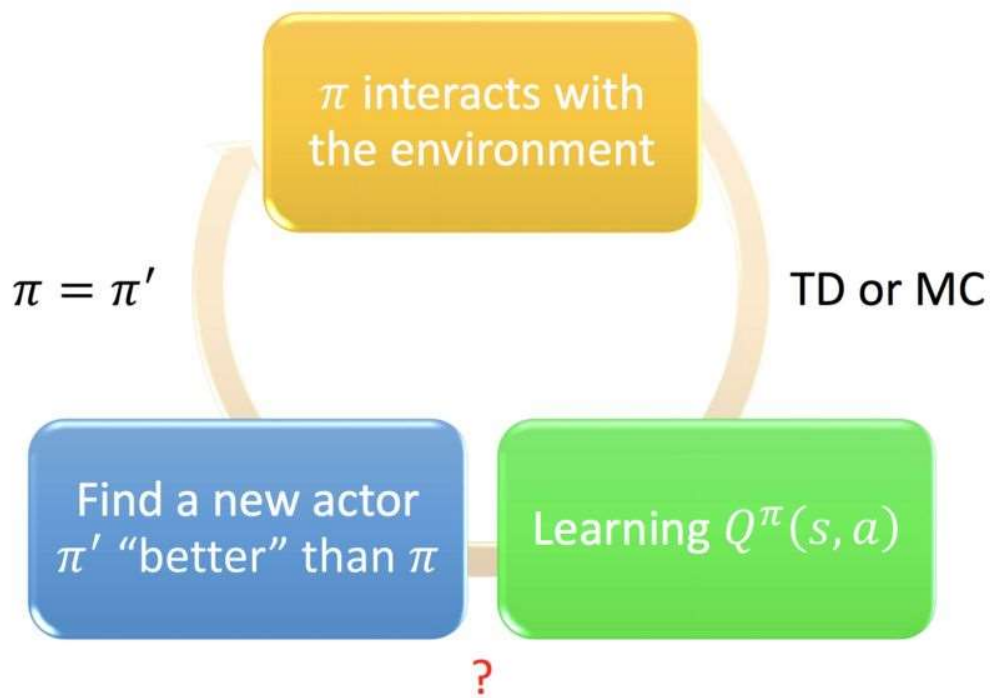
Another Critic

- State-action value function $Q^\pi(s, a)$
 - When using actor π , the *cumulated* reward expects to be obtained after seeing observation s and taking a



这边Actor指 π 。首先让 π 玩N次游戏，接着由TD或MC的方式求Q函数，然后在找从里面找一个好的 π' ，更新原本的 π 。

Q-Learning

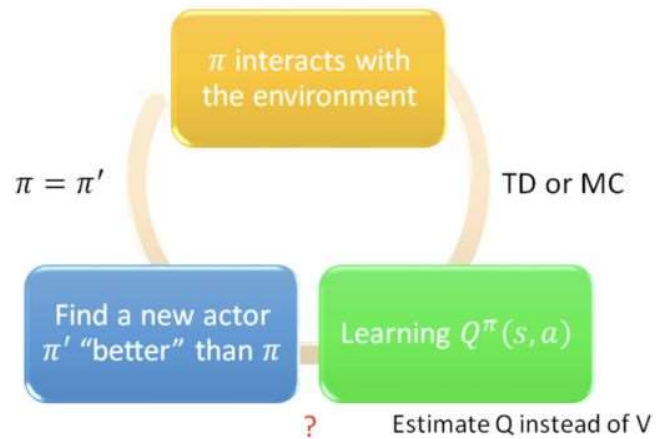


首先我们要定义，什么是更好的 π ？包括所有的状态，只要你 V 大于等于原本的，就是更好的演员。

根据 π 求得 Q 函数，输入一个 S ，穷举所有的动作，找寻使之最大值的动作求得 π' 。实际上 π' 就是由原本的 π 而来，没有额外的参数。

需要注意的是，更新 π' 如果是连续不断的动作，会让 Q 函数在计算上非常消耗时间，所以 Q 会比较适合在**可穷举**action的案例上。

Q-Learning



- Given $Q^\pi(s, a)$, find a new actor π' "better" than π
 - "Better": $V^{\pi'}(s) \geq V^\pi(s)$, for all state s

$$\pi'(s) = \arg \max_a Q^\pi(s, a)$$

- π' does not have extra parameters. It depends on Q
- Not suitable for continuous action a

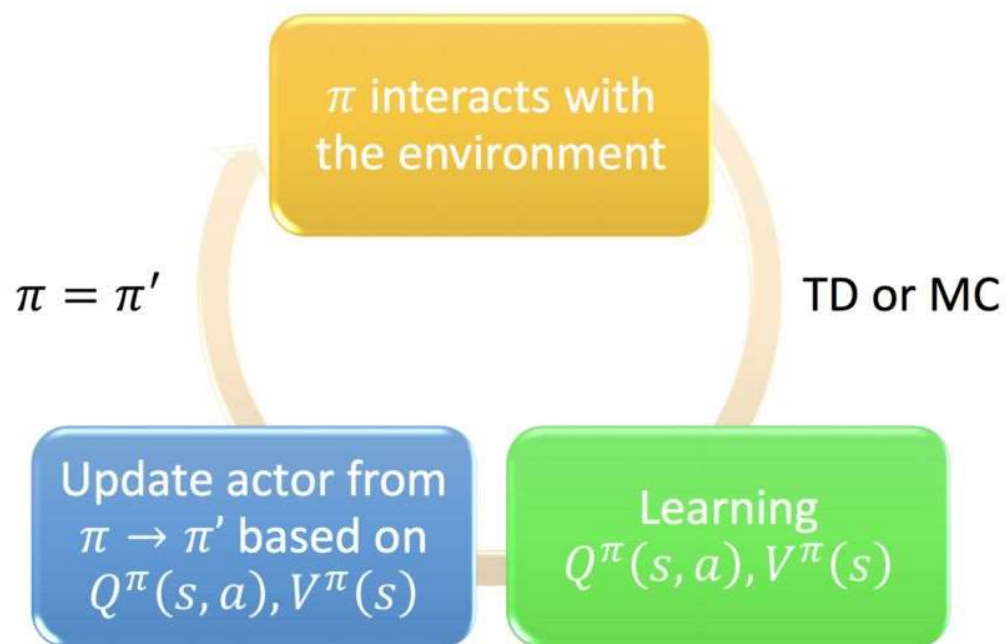
最后要介绍的是Actor和Critic的结合。

Deep Reinforcement Learning

Actor-Critic

跟上一个部分相比，找寻 π' 的会有Q函数(Q function)跟V函数(V function)功能， π' 部分不再是依靠 π 产生Q function穷举动作找出来的，而是会有个实质的数去最大化求值，因此便可以对应可连续动作做应用。

Actor-Critic

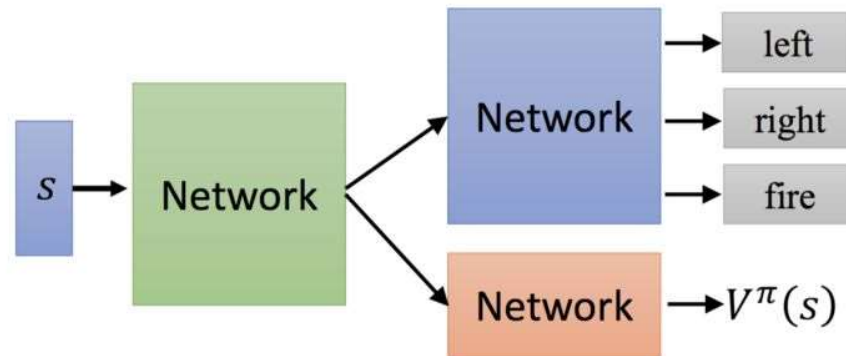


(这里没有李宏毅老师的讲解，作者猜测是 π 跟V function可以共享，把输出的值最大化。)

Actor-Critic

- Tips

- The parameters of actor $\pi(s)$ and critic $V^\pi(s)$ can be shared



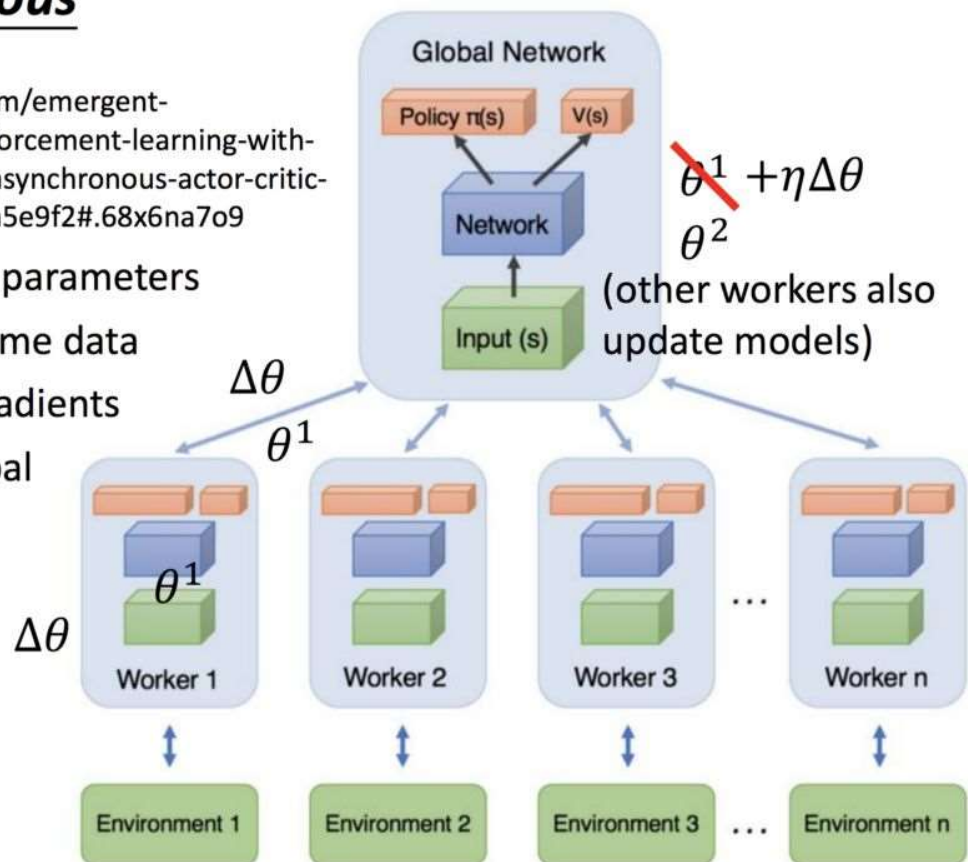
(这边老师也没说明，不过给了有关A3C部分的链接，有兴趣的朋友可以看看)

Asynchronous

Source of image:

<https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2#.68x6na7o9>

1. Copy global parameters
2. Sampling some data
3. Compute gradients
4. Update global models



李宏毅老师PPT网址:

[http://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2017/Lecture/RL%20\(v4\).pdf](http://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2017/Lecture/RL%20(v4).pdf)

本文经授权转载自Medium，感谢作者Ivan Lee授权。