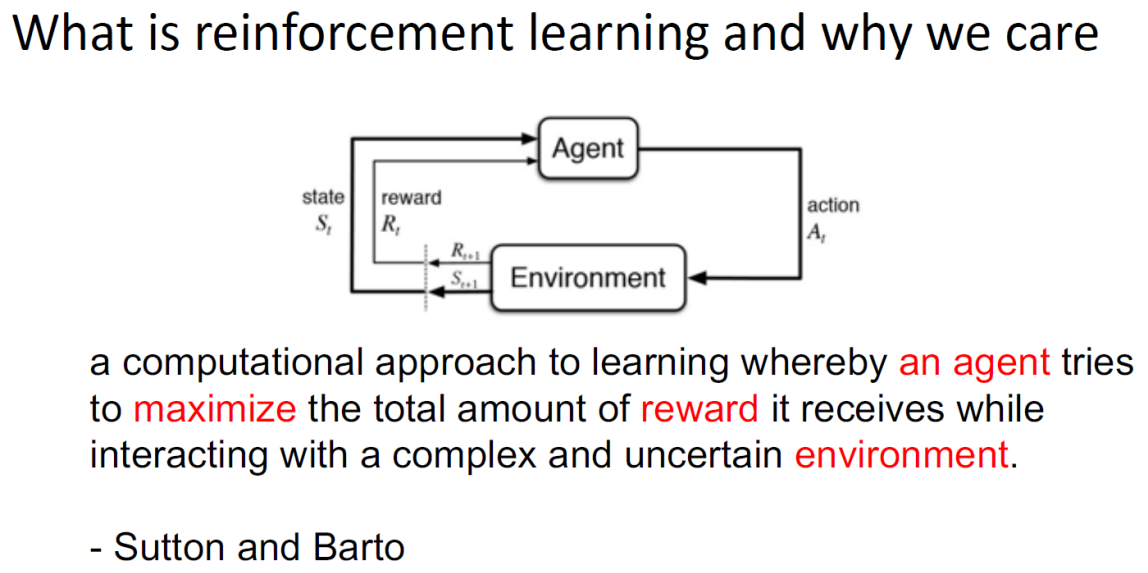
# 第一章 强化学习的基本介绍

## 强化概念的来源

在动物学习背景下，“强化”这个术语在Thorndike表达效力定律后得到了很好的应用。 在1927年巴甫洛夫关于条件反射的专著的英文译本中，首先出现在这种背景下： **巴甫洛夫将强化描述为由于动物接受刺激 - 一种强化剂 - 与另一种刺激或反应有适当的时间关系而加强行为模式。** 一些心理学家将强化的观点扩展到包括削弱和加强行为，并扩展强化者的想法，包括可能**忽略或终止**刺激。 要被认为是增强剂，强化或弱化必须在强化剂被撤回后持续存在；仅仅吸引动物注意力或刺激其行为而不产生持久变化的刺激物不会被视为强化物。

## 强化学习的基本概念




**强化学习是指与复杂、不确定的环境进行交互时，最大化从环境获得的累计奖励的一种方法。**

## 强化学习和监督学习的差别

- Sequential data as input (not i.i.d)

  （输入的是时序的数据而不是像监督学习那样输入的数据满足独立同分布的要求）
- The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them.

  （没有一个“监督者”，只有通过奖励信号来判断应该采取哪个动作）
- Trial-and-error exploration (balance between exploration and exploitation)

  （探索与利用之间的平衡）
- The data used to train the agent is collected through interactions with the environment by the agent itself (compared to supervised learning where you have a fixed dataset for instance). This dependence can lead to vicious circle: if the agent collects poor quality data (e.g., trajectories with no rewards), then it will not improve and continue to amass bad trajectories.

  （训练智能体的数据是智能体自身和环境交互得到的，而不像监督学习那样一开始有一个固定的数据集，**这样就可能带来一个恶性循环**：如果智能体收集到一些很差的数据，比如一些奖励为0的轨迹，那么智能体通过这些数据可能得不到提升，进一步地智能体又采集到一些质量很差的数据，陷入恶性循环）。

- **具有超人类的上限**：传统的机器学习算法依赖人工标注好的数据，从中训练好的模型的性能上限是产生数据的模型（人类）的上限；而强化学习可以从零开始和环境进行不断地交互，可以不受人类先验知识的桎梏，从而能够在一些任务中获得超越人类的表现，比如AlphaGo、AlphaZero的表现就超越了人类。
- **延迟性**：得到的结果很可能延迟于我们做出决策一段时间，有些糟糕的影响并不完全因为当前的决策错误导致的，可能由于前面某一个步骤造成了一个不可逆的错误。
- **非监督的**：我们通常只得到reward signal。每次系统的action只能得到代表这次行为的好坏的标量，比如是10 points，但是我们不知道他的最好的值是多少，就可以理解为一个老师给你打了10分，你其实不知道这是百分制的10分还是十分制的10分。

## 进化策略与强化学习

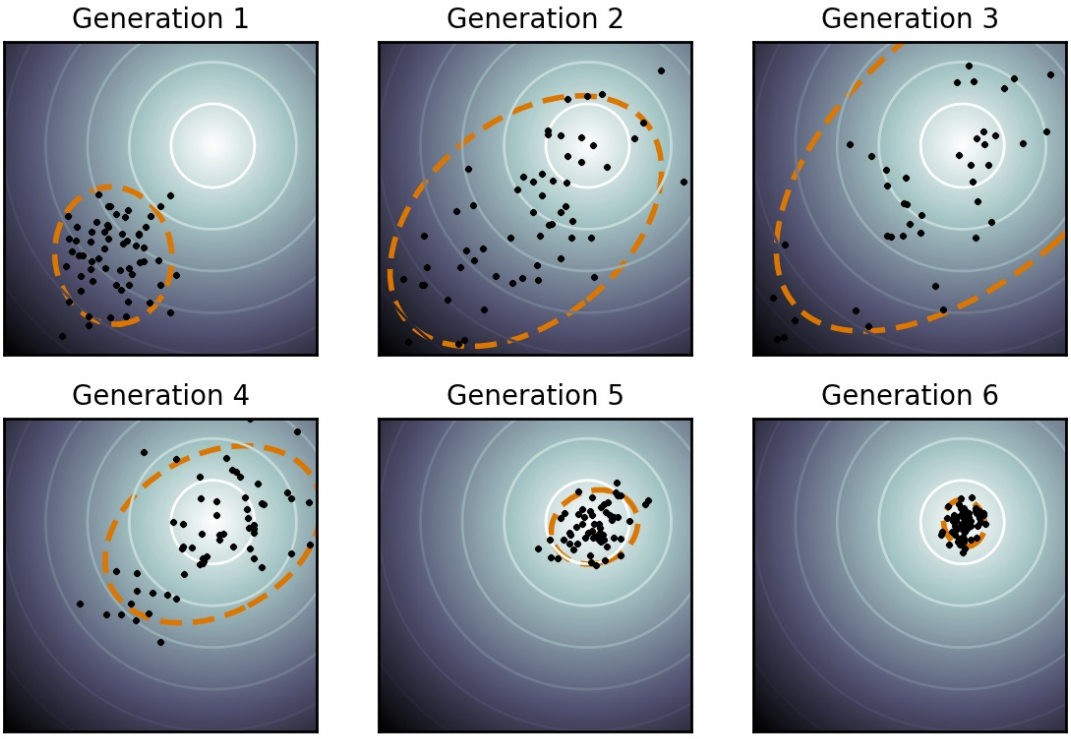
进化策略(Evolution Strategies, ES)反复迭代调整一个正态分布进行搜索来优化算法，是一种无梯度随机优化算法。进化策略中迭代的正态分布一般写成 $N\left(m_t, \sigma_t^2 C_t\right)$ , 包 含三个参数 $m_t, \sigma_t, C_t$ 。正态分布的参数所起的作用为：

- $m_t$ 均值，决定分布的中心位置; 在算法中，决定搜索区域;
- $\sigma_t$ 步长参数, 决定分布的整体方差(global variance); 在算法中，决定搜索范围的大小和强度。
- $C_t$ 协方差矩阵，决定分布的形状; 在算法中决定变量之间的依赖关系，以及搜索方向之间的相 对尺度(scale).

ES算法设计的核心就是如何对这些参数进行调整, 尤其是步长参数和协方差矩阵的调整, 以达到尽可能好的搜索效果。对这些参数的调整在ES算法的收敛速率方面有非常重要的影响。一般的, ES调整参数的基本思路是, 调整参数使得产生好解的概率逐渐增大（沿好的搜索方向进行搜索的 概率增大）。
进化策略在搜索中反复迭代以下步骤：

1. Sampling: 采样产生一个或者一组候选解(candidate solutions);
2. Evaluation：对新产生的解计算对应的目标函数值;
3. Selection：依据目标函数值选择部分或者全部解;
4. Update：使用选择的解更新分布参数.
   在进化算法中，一次完整的迭代称为一代 (generation)，一个候选解称为一个个体，计算目标计算目标函数值的过程称为评估。每次迭代产生的新的候选解称为子代（offspring），通过选择得到的用于产生子代的解称为父代（parent）。

如下图所示：




进化采取大量静态策略，每个策略在扩展过的较长时间内与环境的一个独立实例进行交互，然后选择最多收益的策略及其变种来产生下一代的策略，然后继续循环更替，**它在单个个体的生命周期内不进行学习，它没有利用强化学习中这种交互的特性，其实更像是一种策略搜索。**

为了评估策略，进化方法保持策略固定并且针对对手进行多场游戏，或者使用对手的模型模拟多场游戏。 胜利的频率给出了对该策略获胜的概率的无偏估计，并且可用于指导下一个策略选择。 **但是每次策略进化都需要多场游戏来计算概率，而且计算概率只关心最终结果，*每场游戏内*的信息被忽略掉了。** 例如，如果玩家获胜，那么游戏中的 *所有* 行为都会被认为是正确的，而不管具体移动可能对获胜至关重要。 甚至从未发生过的动作也会被认为是正确的！ 相反，值函数方法允许评估各个状态。

最后，进化和价值函数方法都在搜索策略空间，但价值函数学习会利用游戏过程中可用的信息。

## 强化学习中的基本概念

可以参考知乎上这篇文章的内容:

- Policy :A policy is the agent's behavior model . It is a map function from state/observation to action. Policy（策略）其实就是强化学习中最终要得到的模型，**策略是一个状态到动作的映射**，也就是说告诉Policy当前状态是什么样的，它会返回给你一个动作。
  - Stochastic policy: Probabilistic sample: $\pi(a \mid s)=P\left[A_t=a \mid S_t=s\right]$
  - Deterministic policy:$a^*=\arg\max_a \pi(a \mid s)$
- Agent：智能体，负责环境进行交互。
- Reward： reward是一个**量化的反馈信号。**它决定了对于智能体来说什么是好，什么是坏（短时间），**强化学习的最终目标其实就是最大化累积奖励。**而相比于reward，我们后面要学的**value function其实一个更长远角度的一个过程，但是估计价值其实是一个很难的过程。**

  举个例子来说，如果我们强化学习的应用场景是开直升飞机，那么如果直升飞机按照我们期望的轨迹飞行，我们就会给智能体一个正的奖励，如果它坠机了，我们就会给它一个负的奖励。当然，在实际问题中，具体奖励给多少，怎么给，其实是一个很复杂的问题。
- Action：智能体的动作。
- Goal：我们要实现的目标，对于强化学习任务本身来说，就是最大化累计奖励，所以在强化学习中很关键的一点就是保证最终要达到的目标和最大化累计奖励的目标是一致的: 智能体可能会学出一个策略去最大化累积奖励，但是根本不是我们想要的方式。

- Observation： 智能体观察到的东西，比如说直升机当前的位置和速度。
- Environment：环境。有的书里面会花很多篇幅去区分智能体和环境，去纠结哪个是属于智能体，哪个是属于环境，我觉得对于像我这样的初学者来说，不用过度纠结，你就把智能体想象成自己，把除你自己以外的所有东西都想象成环境。
- Trajectory：轨迹，可以理解成智能体从开始到结束是怎么一步一步过来的。
- Rollout: rollout在字典中的意思是：首次展示，滑跑。在强化学习中大家就可以理解成一次实验、一条轨迹。
- Value function （expected discounted sum of future rewards under a particular policy $\pi$, **we use it to quantify goodness/badness of states**（这里的有多好是用未来预期的收益来定义的），可以简单地理解为**价值函数就是来判断状态的好坏。**

  $v_\pi(s) \doteq \mathbb{E}_\pi\left[G_t \mid S_t=s\right]=\mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t=s\right]$, for all $s \in \mathcal{S}$
- Model: A model predicts what the environment will do next。注意这里的model不是我们最终学出来的做决策的模型，而是当前状态S和动作A到下一个动作S'的映射的模型，也就是告诉你现在如果你这么干你会得到什么后果的模型。
  - Predict the next state:$\mathcal{P}_{ss'}^a=\mathbb{P}\left[S_{t+1}=s' \mid S_t=s, A_t=a\right]$
  - Predict the next reward:$\mathcal{R}_s^a=\mathbb{E}\left[R_{t+1} \mid S_t=s, A_t=a\right]$
  - model-based 和model-free的差别就是知不知道状态转移矩阵和reward
- Q-function **(could be used to select among actions)**:
  $q_\pi(s, a) \doteq \mathbb{E}_\pi\left[G_t \mid S_t=s, A_t=a\right]=\mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t=s, A_t=a\right]$

  **Q函数其实是对（S，A）对的一个评价，也就是说从长期的角度来看（reward是短期），你在当前状态做某一个动作到底有多好。**所以如果我们知道了Q函数，其实就知道了在每一步到底应该怎么做了，最简单的就是取agmax选Q函数最大的动作，根据贝尔曼方程，我们如果知道V函数，我们可以求出Q函数，所以大家会看到说，我们如果知道了Q函数或者V函数，这个强化学习问题就是可解的。
- history是一个action、observation、reward的序列：

  $$H_t=A_1, O_1, R_1, \ldots, A_t, O_t, R_t$$

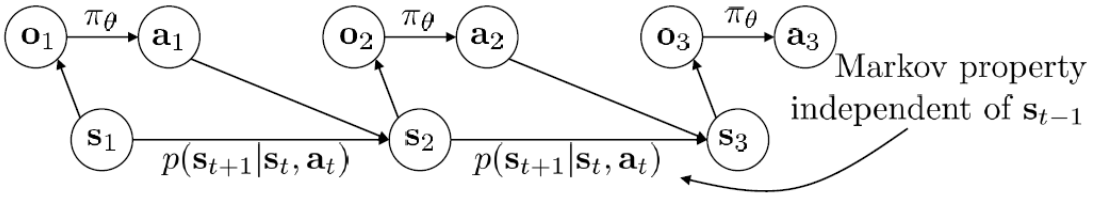
  history能够决定action接下来采取什么行动；实际上，agent做的就是从history到action的映射；对于环境而言，输入history和智能体产生的action，输出对应的observation和reward。
- State：是history的一个总结，用来决定下一步的action是什么，实际上，他是一个关于history的函数，我们可以这样表示：

  $$S_t=f\left(H_t\right), A_t=h\left(S_t\right)$$

State相对于history而言最大的好处在于他的信息量很少，实际上，我们也不需要得到所有的history来得到下一步的action。

在这里想讲一下state和observation的区别，也就是大家有时候会看到$\pi_\theta\left(\mathbf{a}_t \mid \mathbf{o}_t\right)$，有时候看到$\pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)$，这两个其实是存在区别的，用Sergey Levine来说就是，**States are the true configuration of the system and an observation is something that results from that state which may or may not be enough to deduce the state. State 反映了系统的真实的信息，observation是state表现出来的结果，并不能完全反映出state中的信息。**举个例子，有一只老虎在我们的面前，老虎在哪，速度多少这些就算是state，而我们观察的时候，这只老虎刚好被一棵树挡住了，也就是说我们的observation是一张被树挡住的老虎的照片，那么我们仅仅根据observation（照片）是不能决定我们现在要不要赶紧跑的。而这其实也说明了**当state满足马尔科夫性的时候，observation是不一定满足的**，也就是我们不能根据当前的这张照片来判断我们是不是该不该跑，但是结合之前的一些照片比如老虎一步步潜伏到树底下的照片，我们就可以知道，该跑了！

用下面这张图可以帮助比较好地理解Observation和State之间的关系：




- Bootstrapping: 指用上一次估计值来更新本次的估计值。 比如用上一次估计的表格中的下一状态s'的价值来更新此次估计的表格中当前状态s的价值。

  强化学习中还有很多基本的概念，这里只是一些基础的，大家现在不能完全理解也没关系，到后面就自然会理解了。大家可以通过宝藏李宏毅老师课程上的这张图来大致理解一下强化学习的大致过程以及其中的基本概念。


- Sequential Decision Making

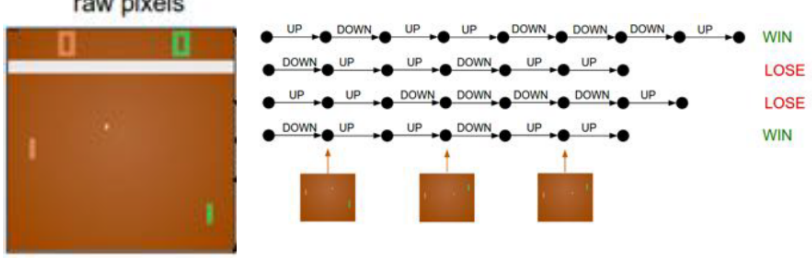
# Sequential Decision Making

- The history is the sequence of observations, actions, rewards.

$$H_t = O_1, R_1, A_1, ..., A_{t-1}, O_t, R_t$$

- What happens next depends on the history
- State is the function used to determine what happens next

$$S_t = f(H_t)$$




- Goal: select actions to maximize total future reward （最大化未来的累积奖励)。

- Actions may have long term consequences （动作在很长的时间后才发生作用）。
  Reward may be delayed （奖励存在延时）。
- It may be better to sacrifice immediate reward to gain more long-term reward （最好是牺牲眼前的短暂的奖励去获得更多的长期奖励）。
- Examples:
  - A financial investment (may take months to mature)
  - Refuelling a helicopter (might prevent a crash in several hours)
  - Blocking opponent moves (might help winning chances many moves from now)

### 全部可观和部分可观

在介绍部分客观和全部可观之前，想先介绍三个概念：

- Environment State： The environment state $S_t^e$ is the environment's private representation。 environment state指的是对环境信息的一个总结，他可能是一段0101的数字序列，用于产生对应的observation和reward，**他是环境的一种内部的状态**，对于agent来说environment state是不可见的，agent能看到的只有 environment state产生的observation和reward。
- Agent State: The agent state $S_t^a$ is the agent's internal representation。**是智能体的一些内部表示，比如说强化学习算法的实现。他是关于History的一个函数。**
- Information state：包含了history中所有有用的信息。如果它满足马尔科夫性，那么保留了$S_t$, 我们就可以将 $S_1, \dots S_{t-1}$ 全部丢掉。

有了这三个概念之后，理解全部客观和部分可观就更简单了：

# Sequential Decision Making

- Environment state and agent state

$$S_t^e = f^e(H_t) \quad S_t^a = f^a(H_t)$$

- **Full observability**: agent directly observes the environment state, formally as Markov decision process (MDP)

$$O_t = S_t^e = S_t^a$$

- **Partial observability**: agent indirectly observes the environment, formally as partially observable Markov decision process (POMDP)
  - Black jack (only see public cards), Atari game with pixel observation,

#### Fully Observable Environments

- 在完全可观察环境下，agent能够直接观察到environment state，也就是说：

$$O_t = S_t^a = S_t^e$$

- 这类问题被称为MDP问题，Markov decision process。

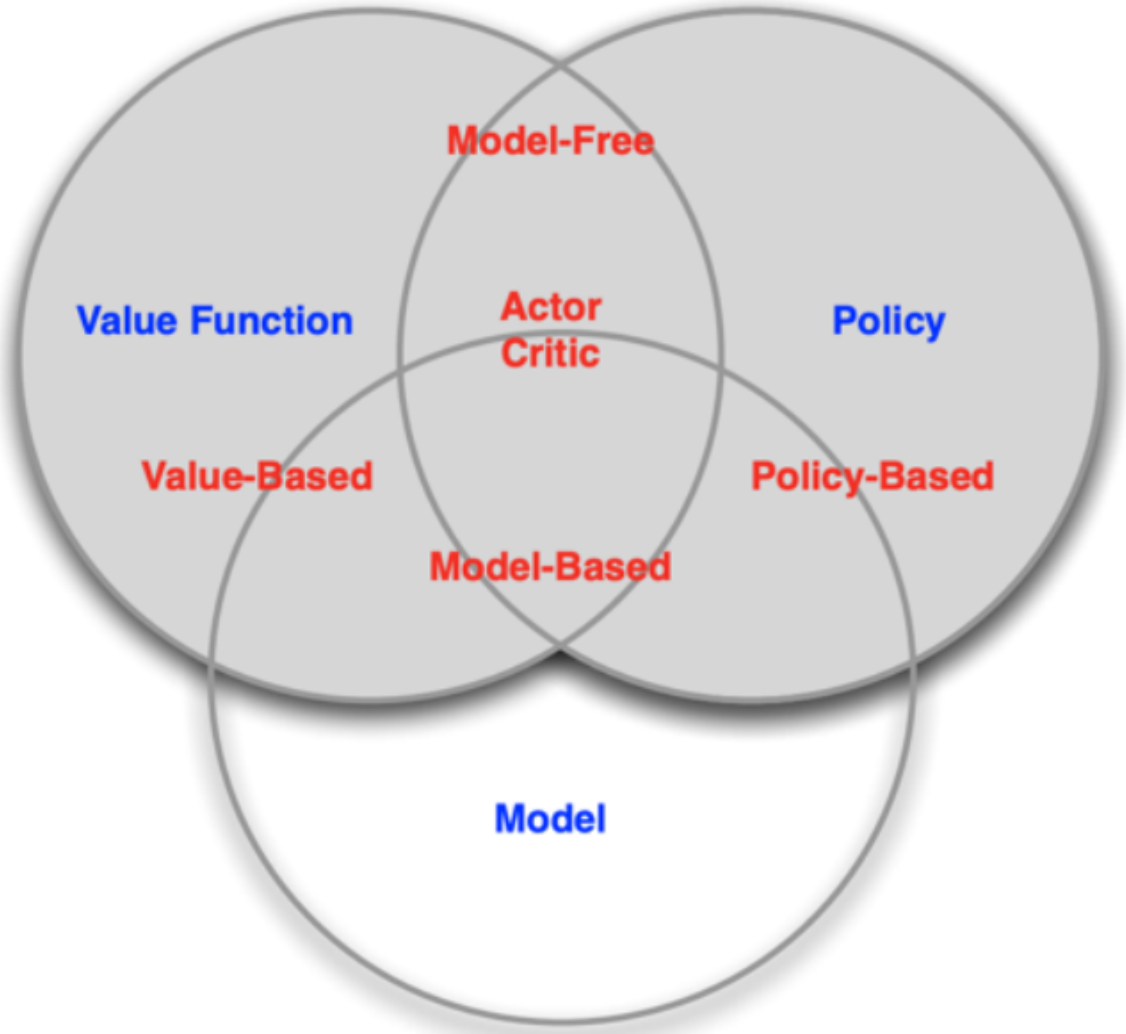
#### Partially Observable Environments

- 在部分可观察环境中，Observation state 不等于 environment state，我们只能看到部分信息，或者只能看到一些现象，比如在扑克游戏中，我们只能看到公开的牌面，看不到其他人隐藏的牌。
- 这个为题被称为POMDP问题：partially observable Markov decision process。
- 对于这个问题，有几种解决方案：

- 记住所有的历史状态：$S_t^a = H_t$
- 使用贝叶斯概率，我们可以创建一个Beliefs, $S_t^a = \left(P\left[S_t^e = s^1\right], \ldots, P\left[S_t^e = s^n\right]\right)$, 这个概率状态的向量组成当前的状态, 所以我们需要保存所 有状交的概率值
- 使用递归神经网络： $S_t^a = \sigma\left(S_{t-1}^a W_s + O_t W_o\right)$,将之前的状态 $S_{t-1}^a$ 与 $W_s$ 进行线性组合, 再加上最近的一次观察值的线性組合, 带入非线性函 数 $\sigma$, 得到新的状态

## Agent的分类

- Types of RL Agents based on What the Agent Learns
  - Value-based agent：我们使用Value function 来计算Policy，Policy是隐含的，不需要直接定义
  - Policy-based agent：每一个状态都映射到一个action，直接使用这个action做决策
  - Actor-Critic agent：综合使用Policy 和 value function，使用Policy，同时保留每次所得到的value function
- Types of RL agent on if there is a model
  - model-free：我们不知道环境的状态概率转移矩阵，不知道环境模型，只是直接根据value, function、policy来直接得出结论
  - model-based：我们知道环境的动态概率转移矩阵，会根据动态概率转移矩阵来预测我们做动作带来的影响从而帮助我们更好地决策




## Exploration和Exploitation

- Exploration: trying new things that might enable the agent to make better decisions in the future
- Exploitation: choosing actions that are expected to yield good reward given the past experience

- Often there may be an exploration-exploitation trade-off.(**When to explore and when to do exploitation?** )
- May have to sacrifice reward in order to explore & learn about potentially better policy
- 利用就是选择最高估计价值的动作而探索不是，短期来看利用是合理的，但是长期来看探索可能会带来总体收益的最大化，探索可以改善对非贪心动作的价值的估计。
- **Exploration(探索)**：倾向于探索环境中新的信息，比如说去没吃过的饭店吃饭。
- **Exploitation(利用)**：倾向于开发使用我们已经探测得到的最大reward，就像我们吃过海底捞了觉得海底捞好吃，以后就什么新的饭店也不去了就只吃海底捞。这么做相对来说确实是"安全"的，起码可以保证结果不至于太坏，但是可能我们永远就吃不到比海底捞更好吃的东西了。

## Planning和learning

- **Learning problem**：你的环境是未知的，你不能提前知道你的决策将对环境造成什么样的改变。我们需要通过不断地与环境交互，从而得知我们的action造成什么样的改变。
- **Planning Problem**：我们的工作环境是已知的，我们被告知了整个环境的运作规则的详细信息。智能体能够计算出一个完美的模型，并且在不需要与环境进行任何交互的时候进行计算。在这种情况下智能体不用实时地与环境交互就能知道未来环境，只需要知道当前的状态，就能够开始思考，来寻找最优解。

  **感觉Planning和Learning就像是model-based和model-free?**

## Prediction和Control

- **预测(Prediction)**:给你一个policy，agent得到，这个policy能够得到多少reward，这是一个预估未来的过程。
- **控制（Control）**：确定众多决策中，哪一个决策能够得到最多的奖励。

  要强调的是，这两者的区别就在于，预测问题是**给定一个policy**，我们要确定他的value function是多少。而控制问题，是在**没有policy的前提下**，我们要确定最优的value function以及对应的决策方案。

  实际上，这两者是递进的关系，在强化学习中，我们通过解决预测问题，进而解决控制问题：我们会通过先衡量某个策略的好坏再接着对这个策略进行优化。

## on-policy 和off-policy

首先，我们来区分一下Behavior Policy(行为策略) 和Target Policy（目标策略）：

- 行为策略是用来与环境交互产生数据的策略，即在训练过程中实际做决策；
- 而目标策略是通过行为策略采集得到的数据来进行不断地优化、学习的策略，当然，每隔一段时间行为策略会更新成目标策略来保证我们是用优化后的策略来与环境进行交互。

  那么为什么要有两个策略呢？这就是为了解决我们前面讲到的强化学习中的探索与利用的问题：我们**可以利用行为策略来保证探索性，提供多样化的数据；而不断地优化目标策略来保证利用。**

  **On-policy 的目标策略和行为策略是同一个策略**，其好处就是简单粗暴，直接利用枚举就可以优化其策略, 但这样的处理会导致策略其实是在学习一个局部最优, 因为On-policy的策略没办法很好地同时保持即探索又利用; **而Off-policy将目标策略和行为策略分开，**可以在保持探索的同时, 能求到全局最优值。但其难点在于：如何在一个策略下产生的数据来优化另外一个策略? 也可以从Buffer（数据缓冲区）的层面来理解On-policy和Off-policy的不同：**off-policy是将缓冲区数据每次采样出一部分，而on-policy可以看做一次性将缓冲区中所有数据采集出来并删除。**

## online和offline

注意这两个概念和on-policy与off-policy完全没有关系。

- **在线学习**：边玩边学---可能第一关你还没过，你已经学到了不少有用的技术；

- **离线学习**：你没头没脑地（没有学习）玩着游戏，等第一关过了或者死了，然后你仰天长叹，在脑海中回顾着刚刚发生的一切，在这个过程中回顾并学习更新着自己的策略p(action|state)，专业一点的解释可以看这篇推文。

## deterministic and stochastic

- Deterministic: given a state, the policy returns a certain action to take：给定一个状态，策略只会返回一个固定的动作。
- Stochastic: given a state, the policy returns a probability distribution of the actions (e.g., $40\%$ chance to turn left, $60\%$ chance to turn right) or a certain Gaussian distribution for continuous action。**给定一个状态，策略会返回一个动作的分布**，比如以40%的概率向左走，60%的概率向右走。常用的分布有用于离散动作空间的类别分布（Categorical Distribution）、用于连续动作空间的对角高斯分布（Diagonal Gaussian Distribution）
- A deterministic policy is easily beaten。一个确定性的策略是很容易输的，比如你和同学玩斗地主，每次你都一个炸之后留一张3在手里，那肯定多玩几次你同学看到你只剩一张牌是不会让你走的。

## Gym中的环境

强化学习中有很多的benchmark帮助大家很好地去测试自己算法性能，像Gym、MoJoCo等，MoJoCo是需要付费的，学生可以试用一年。可以一步一步来，先给大家介绍一下最简单的Gym环境，Gym里面其实有很多的环境可以供大家去使用，具体的一些细节我觉得大家最好就上Gym的gitlab上看就好了，这里主要是给大家讲一些基础的。

安装的话也很简单，就一个命令

```
pip install gym
```

或者用conda装也行，其实也是一行命令，把pip换成conda。

windows环境下用anaconda安装gym

gym的最基本的使用可以看我下面的几行代码，都有注释应该比较好理解。

- 基本使用

```
import gym

env=gym.make("MountainCar-v0") #创建对应的游戏环境
env.seed(1) # 可选，设置随机数，以便让过程重现
env=env.unwrapped  # 可选，为环境增加限制，对训练有利

# ----------------------动作空间和状态空间---------------------#
print(env.action_space)  # 动作空间，输出的内容看不懂
print(env.action_space.n)  # 当动作是离散的时，用该方法获取有多少个动作
# env.observation_space.shape[0] # 当动作是连续的时，用该方法获取动作由几个数来表示
print(env.action_space.low)  # 动作的最小值
print(env.action_space.high)  # 动作的最大值
print(env.action_space.sample())  # 从动作空间中随机选取一个动作
# 同理，还有 env.observation_space，也具有同样的属性和方法（.low和.high方法除外）
#------------------------------------------------------------#

for episode in range(100): #每个回合
    s=env.reset()  # 重新设置环境，并得到初始状态
    while True:  # 每个步骤
```

```
        env.render()  # 展示环境
        a=env.action_space.sample() # 智能体随机选择一个动作
        s_,r,done,info=env.step(a)   # 环境返回执行动作a后的下一个状态、奖励值、是否终止以及其他信息
        if done:
            break
```

下面以CartPole为例，当大家要用一个以前从来没有用过的新环境时，首先脑子里要大概有上面的强化学习的一套代码：

- 创建环境、
- 设置环境随机种子、
- 初始化环境、
- 给环境传动作并获取环境状态、
- 判断是不是终止.....

大概都会这么一些步骤。当然每个环境的调用是有差别的，比如说要知道他的动作空间是什么样的。

一般gym中的动作空间会有这几种：

- `Box`: A N-dimensional box that contains every point in the action space.
- `Discrete`: A list of possible actions, where each timestep only one of the actions can be used.
- `MultiDiscrete`: A list of possible actions, where each timestep only one action of each discrete set can be used.
- `MultiBinary`: A list of possible actions, where each timestep any of the actions can be used in any combination.

他的observation_space是怎么样的，像这些就直接去环境的源代码中看就好了，看一下文档注释大概就知道了。比如下面的CartPole中的文档注释：

- CartPole

```
"""
Description:
    A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum starts upright, and the goal is to prevent it from falling over by increasing and reducing the cart's velocity.
    一根杆子通过一个未驱动的接头连接到小车上，小车沿着无摩擦的轨道移动。钟摆开始直立，目标是通过增加或减少小车的速度来防止它倒下。

Source:
    This environment corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson

Observation:
    Type: Box(4)
    Num Observation                 Min         Max
    0   Cart Position             -4.8            4.8
    1   Cart Velocity             -Inf            Inf
    2   Pole Angle                 -24 deg        24 deg
    3   Pole Velocity At Tip      -Inf            Inf

Actions:
    Type: Discrete(2)
    Num Action
```

```
    0   Push cart to the left
    1   Push cart to the right

    Note: The amount the velocity that is reduced or increased is not
fixed; it depends on the angle the pole is pointing. This is because
the center of gravity of the pole increases the amount of energy needed
to move the cart underneath it

Reward:
    Reward is 1 for every step taken, including the termination step

Starting State:
    All observations are assigned a uniform random value in
[-0.05..0.05]

Episode Termination:
    Pole Angle is more than 12 degrees
    Cart Position is more than 2.4 (center of the cart reaches the edge
of the display)
    Episode length is greater than 200
    Solved Requirements
    Considered solved when the average reward is greater than or equal
to 195.0 over 100 consecutive trials.
"""
```

首先看了介绍就大概知道CartPole游戏是一个小车连着一根杆子，然后小车通过控制左右移动来防止杆子倒下。

然后他下面还告诉了我们CartPole游戏中observation返回的四个值的含义：小车的位置、小车的速度、杠的角度、杠的尖端的速度；以及action_spaces里面就是一个0，1，如果我们给step函数传0，小车就往左走；传1小车就往右走。然后还告诉了我们每坚持1步不倒，就会得到一个+1的reward；还告诉了我们环境的一些终止条件，比如说如果能坚持200步不倒环境就会终止。

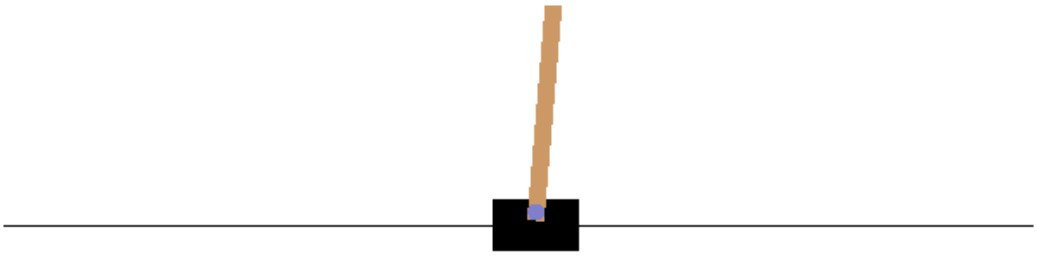
有了这些信息之后，我们可以先简单地写一些测试的代码,就看一下环境到底是怎么样的，动作要怎么传，observation到底是什么样的，环境render之后是什么样的。

```python
import gym

env = gym.make("CartPole-v0")
env.seed(1)
# ----------------------动作空间和状态空间--------------------#
print("action space:", env.action_space)  # 动作空间
print("The number of the action:", env.action_space.n)  # 当动作是离散的时，用该方法获取有多少个动作
for i in range(5):
    print("The testing action:", env.action_space.sample())  # 从动作空间中随机选取一个动作,可以多打印几次，看看动作空间都是什么样的
print(env.observation_space)  # 状态空间
print(env.observation_space.shape)  # 状态空间的形状
for i in range(5):
    print("The testing observation:", env.observation_space.sample())
 # 对状态空间进行采样，打印出来看一下

for episode in range(100):  # 每个回合
    s = env.reset()  # 重新设置环境，并得到初始状态
    while True:  # 每个步骤
        if episode % 50 == 0:
```

```
            env.render()  # 展示环境
        a = env.action_space.sample()  # 智能体随机选择一个动作
        s_, r, done, info = env.step(a)  # 环境返回执行动作a后的下一个状态、奖励值、是否终止以及其他信息
        if done:
            break

```

然后看一下结果：我们大概就知道环境大致是什么样的，它的动作空间和状态空间大概是什么量级。

```
action space: Discrete(2)
The number of the action: 2
The testing action: 0
The testing action: 0
The testing action: 0
The testing action: 1
The testing action: 0
observation space: Box(4,)
The shape of the observation: (4,)
The testing observation: [ 1.5918539e+00  1.8163429e+38 -3.2249892e-01 -5.7971208e+36]
The testing observation: [-2.9963758e+00 -2.3220294e+38 -4.0943721e-01 -1.1218629e+38]
The testing observation: [ 4.1609278e+00 -1.6202370e+37  1.0278270e-01 -2.1812878e+38]
The testing observation: [ 4.6125871e-01 -2.7374105e+38  3.6709017e-01  2.7989397e+38]
The testing observation: [-4.9687669e-01  1.3121045e+38  2.9005164e-01  1.6835726e+38]
```



- Frozen Lake

```
 """
    Winter is here. You and your friends were tossing around a frisbee at the park
    when you made a wild throw that left the frisbee out in the middle of the lake.
    The water is mostly frozen, but there are a few holes where the ice has melted.
    If you step into one of those holes, you'll fall into the freezing water.
    At this time, there's an international frisbee shortage, so it's absolutely imperative that
    you navigate across the lake and retrieve the disc.
```

```
    However, the ice is slippery, so you won't always move in the direction
you intend.
    The surface is described using a grid like the following

        SFFF
        FHFH
        FFFH
        HFFG

    S : starting point, safe
    F : frozen surface, safe
    H : hole, fall to your doom
    G : goal, where the frisbee is located

    The episode ends when you reach the goal or fall in a hole.
    You receive a reward of 1 if you reach the goal, and zero otherwise.

    """
```

同理Frozen Lake也是一样的去看源码中的注释。

## 强化学习的一些资源

这里主要给大家分享一些课程，书籍还有代码库。

**课程**

- CS 285 at UC Berkeley，这个课程是2020年底Sergey Levine大神在UC Berkeley教的课程，我觉得讲的特别详细，而且里面加了很多一些最新的研究成果。
- David Silver's course，这个课程是David Sliver在2015年出的课程，非常的经典，相信很多入门强化学习的同学都是看的这个课程，里面的很多东西课程刚开始听会比较难懂，可能得多听几遍。
- Berkeley's Deep RL Bootcamp：这个课我其实没看过，但是看了一下lectures，是一群大佬讲的这个课程，质量应该也很好。
- Stanford CS234: Reinforcement Learning ：这个课暂时没看过，不过看评价感觉挺好的。
- Stanford CS330: Multi-Task and Meta-Learning, 2019 ：Finn关于多任务学习和元学习的课程，强推。
- 李宏毅老师的机器学习课程：李宏毅老师的机器学习课程里面有一些也涉及到了强化学习的内容，李宏毅老师是用中文讲的课程，而且总是能把一个比较复杂的问题以一种很简单的方式讲出来。
- 港中文周博磊老师的课程：我觉得大家如果看英文课比较吃力的话建议大家可以先看周博磊老师的课程先入个门，这个课程的内容相对比较简单，比较好入门。
    - B站视频
    - 课程作业
    - RL demo代码

    课程的话暂时就只想到这么多，上面的这些课程我听了的大部分都记了笔记，大部分是直接记在课件上的，**大家如果有想要的也可以留言告诉我。**

**书籍**

- Sutton的强化学习书籍：这个算是强化学习里面最经典的书了，Sutton巨佬的作品，基本上不管是哪个老师上强化学习的课程都会推荐这本书，不过这本书里面将的大多数都是value-based的方法。
    - 英文版

    - 中文版，这个翻译其实还存在挺多问题的，有些语句和专业词汇翻译的也不太好，建议大家可以购买俞凯老师翻译的纸质版书籍。
    - 代码
    - 书中的提到的文献
    - sliders
- Algorithms for Reinforcement Learning, Szepesvari：很多经典的强化学习算法里面都有。
- 《强化学习精要：核心算法与TensorFlow 实现》：这本书是我偶然在实验室的一个书柜里面找到的，随手翻了一下发现写的挺好的，这本书和Sutton的书是我平时用的最多的了。

**代码库**

- OpenAI Spinning Up
- Baselines
- Stable-Baselines
- Ray/Rlib
- Pytorch-DRL
- rlpyt
- Tianshou

  这些库有些有tensorflow写的，有些用pytorch写的，具体的对比和测评可以看Tianshou的作者写的这个测评。

## 强化学习的学习建议

下面的很多建议我都是从Stable Baselines 上面的tips总结的，然后加上了自己一些粗浅的看法。

- Read about RL and Stable Baselines： 多看一些强化学习算法的Baseline代码，比如我上面提供的代码库
- Do quantitative experiments and hyperparameter tuning if needed. This factor, among others, explains that results in RL may vary from one run to another (i.e., when only the seed of the pseudo-random generator changes). For this reason, **you should always do several runs to have quantitative results**.： 因为强化学习算法跑出来的结果有时候会差特别多，所以应该多跑几组实验得到量化的结果。
- Good results in RL **are generally dependent on finding appropriate hyperparameters**. Recent algorithms (PPO, SAC, TD3) normally require little hyperparameter tuning, however, *don't expect the default ones to work* on any environment. 虽然最近的一些算法比如PPO、SAC、TD3不太需要调参，但是不要期待一个默认的超参数设置可以在所有环境下都能用，所有强化学习中的一些好的结果都需要一些超参数的调节。
- When applying RL to a custom problem, you should always normalize the input to the agent (e.g. using VecNormalize for PPO2/A2C) and look at common preprocessing done on other environments (e.g. for Atari, frame-stack, ...). Please refer to *Tips and Tricks when creating a custom environment* paragraph below for more advice related to custom environments. 应该根据智能体的不同，对输入进行规范化。
- Evaluate the performance using a separate test environment ： 有时候训练的时候会加入一些epsilon贪婪等来提升探索，但是在测试的时候应该用一个分开的环境直接对模型进行测试。
- As a general advice, to obtain better performances, you should augment the budget of the agent (number of training timesteps). 为了更好的效果，训练的久一点。

- In order to achieve the desired behavior, expert knowledge is often required to design an adequate reward function. This *reward engineering* (or *RewArt* as coined by Freek Stulp), necessitates several iterations. As a good example of reward shaping, you can take a look at Deep Mimic paper which combines imitation learning and reinforcement learning to do acrobatic moves. 加入一些专家经验去设计比较好的奖励。
- Reproducibility：Completely reproducible results are not guaranteed across Tensorflow releases or different platforms. Furthermore, results need not be reproducible between CPU and GPU executions, even when using identical seeds.In order to make computations deterministic on CPU, on your specific problem on one specific platform, you need to pass a `seed` argument at the creation of a model and set n_cpu_tf_sess=1 (number of cpu for Tensorflow session). If you pass an environment to the model using set_env(), then you also need to seed the environment first.设置随机种子增强结果的复现性。
- 没有哪个算法会适合所有的任务，那么平时怎么选择算法呢？ 首先第一个标准就是看看问题的动作空间是离散的还是连续的，比如像DQN只适合离散动作，SAC只适合连续的动作；还有就是你要不要并行化你的训练：具体怎么选看下面：
    - Discrete Actions - Single Process： DQN with extensions (double DQN, prioritized replay, ...) and ACER are the recommended algorithms. DQN is usually slower to train (regarding wall clock time) but is the most sample efficient (because of its replay buffer).**离散动作、不并行化**：DQN及其扩展，ACER等。DQN训练很慢，但是因为有replay buffer的存在样本利用率比较高。
    - Discrete Actions - Multiprocessed： You should give a try to PPO2, A2C and its successors (ACKTR, ACER). If you can multiprocess the training using MPI, then you should checkout PPO1 and TRPO. **离散动作、并行化**：PPO2、A2C及其对其改进（ACKER、ACER）等
    - Continuous Actions - Single Process： Current State Of The Art (SOTA) algorithms are SAC and TD3. Please use the hyperparameters in the RL zoo for best results. **连续动作、不并行化**：SAC、TD3。
    - Continuous Actions - Multiprocessed： Take a look at PPO2, TRPO or A2C. Again, don't forget to take the hyperparameters from the RL zoo for continuous actions problems (cf Bullet envs). If you can use MPI, then you can choose between PPO1, TRPO and DDPG. 连续动作、并行化: SAC、TD3。
- 在创建一个自定义的环境的时候也有一些tricks：
    - always normalize your observation space when you can, i.e., when you know the boundaries 规范化你的状态空间。
    - normalize your action space and make it symmetric when continuous (cf potential issue below). A good practice is to rescale your actions to lie in [-1, 1]. This does not limit you as you can easily rescale the action inside the environment 规范化你的动作空间，比如把你的动作缩放到[-1, 1]之间。
    - start with shaped reward (i.e. informative reward) and simplified version of your problem 使用一些shaped reward，开始的时候使用问题的简单形式。
    - debug with random actions to check that your environment works and follows the gym interface: 用随机动作去测试你的环境是否正常工作。
- 实现强化学习算法的一些trick：
    - Read the original paper several times 多看几遍原始论文。
    - Read existing implementations (if available) 看算法的现有的实现。
    - Try to have some "sign of life" on toy problems 先在简单的问题试起来。

- Validate the implementation by making it run on harder and harder envs (you can compare results against the RL zoo). You usually need to run hyperparameter optimization for that step. You need to be particularly careful on the shape of the different objects you are manipulating (a broadcast mistake will fail silently cf issue #75) and when to stop the gradient propagation　通过在越来越难的环境中运行来验证实现(可以将结果与RL zoo进行比较)

  下面列举了哪些环境是比较简单的，哪些环境是比较难的。

A personal pick (by @araffin) for environments with gradual difficulty in RL with continuous actions:

- **Pendulum (easy to solve)**
- **HalfCheetahBullet (medium difficulty with local minima and shaped reward)**
- **BipedalWalkerHardcore (if it works on that one, then you can have a cookie)**

in RL with discrete actions:

- **CartPole-v1 (easy to be better than random agent, harder to achieve maximal performance)**
- **LunarLander**
- **Pong (one of the easiest Atari game)**
- **other Atari games (e.g. Breakout)**