第一讲 基础概念

machine learning vs. reinforcement learning?

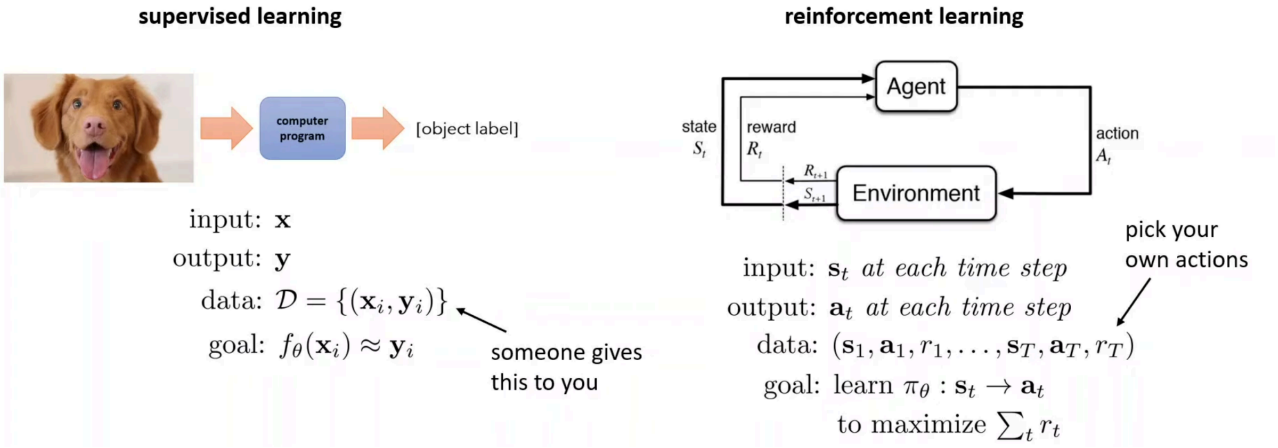
1.基本假设差异

机器学习基本假设：

1. 监督学习 数据是有标签的，即数据为（ 输入X，标签Y ）对
2. 不同数据对 XY 之间是相互独立的，即一个X对应的Y不会对其他X产生影响
3. 最后得到的函数f(X)逼近Y对于每一个X都是相同的，即学习到的函数或者网络对任何X都是有效的（同分布）

强化学习基本假设：

1. 数据无标签，只知道输入数据后的奖励(reward)
2. 并不假设数据是独立同分布的，即前一个输出会影响下一个输出



其中这里最后学习到的函数中有一些参数theta，可以视为神经网络的权重

2.目标导向不同

机器学习是数据驱动，上限为人类上限
强化学习是优化驱动，无上限

第二讲 数学符号 模仿学习

1.数学符号

在强化学习中，我们通常会定义以下概念：

- **状态 (State):** $\mathbf{s}_t$ - 环境的完整描述，有时是智能体无法直接感知的。

- **观测 (Observation):** $\mathbf{o}_t$ - 智能体从环境中获取的信息，可能是状态的局部或噪声表示。
- **动作 (Action):** $\mathbf{a}_t$ - 智能体对环境施加的影响。

策略 (π) 定义了智能体在给定状态或观测下采取行动的方式。

- **基于观测的策略:** $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ - policy。当状态不可完全观测时（例如图片中物体被遮挡），我们基于观测来决定行动。这里的 θ 代表了策略的参数，通常是神经网络的权重。
- **基于状态的策略 (完全可观测):** $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ - policy (fully observed)。如果智能体可以完全感知环境的状态，那么策略可以直接基于状态来决定行动。

环境的动态通常由状态转移概率描述：
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

这表示在当前状态 $\mathbf{s}_t$ 并采取动作 $\mathbf{a}_t$ 的条件下，下一个状态为 $\mathbf{s}_{t+1}$ 的概率。

马尔可夫性质 (Markov property)：
Markov property independent of $\mathbf{s}_{t-1}$
这意味着系统的未来只取决于当前状态和动作，而与过去的历史无关。

2. 模仿学习理论(Math theory)

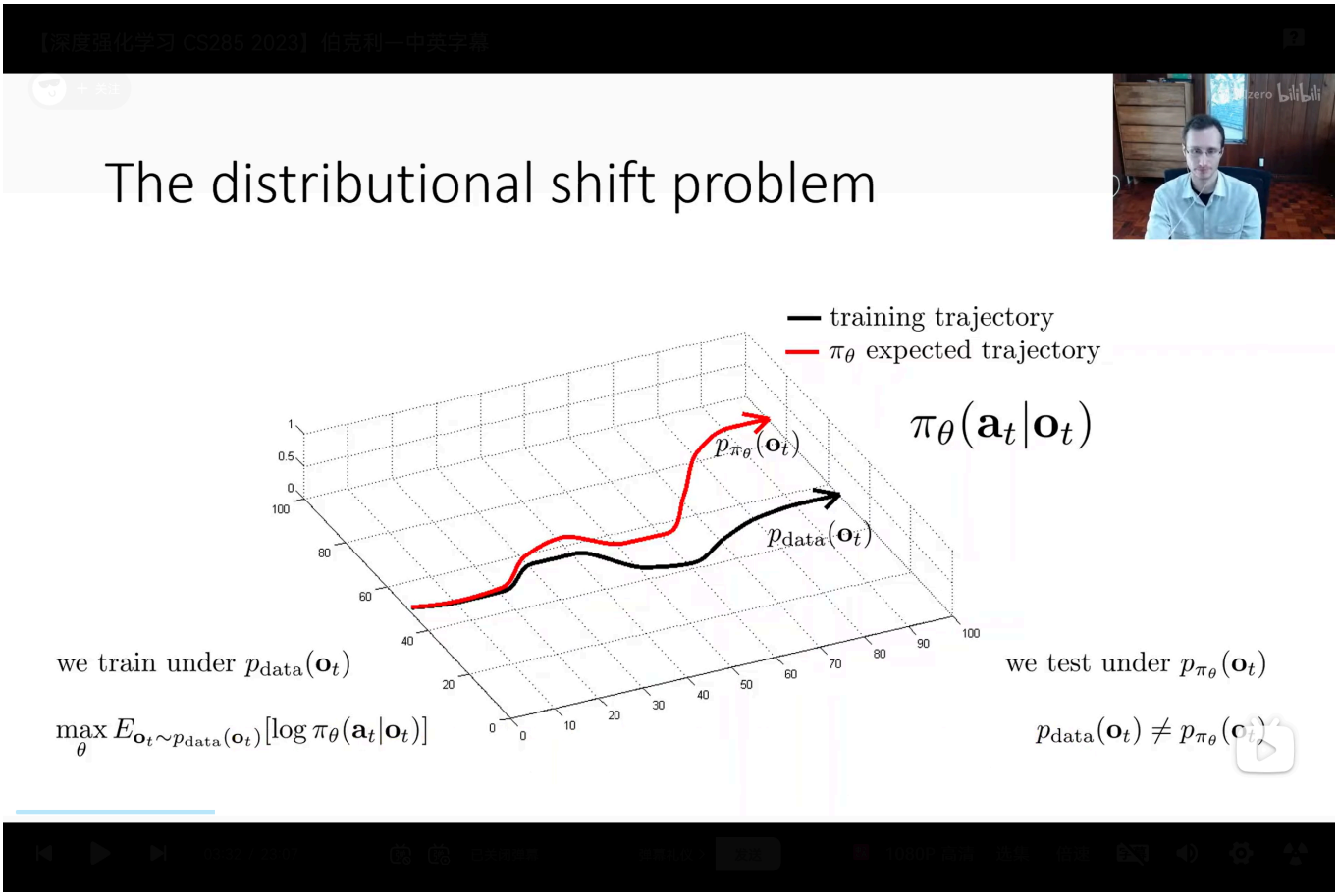
Why imitation learning fail?

2.1直观理解：分布漂移问题 distributional shift problem

train under $p_{data}(\mathbf{o}_t)$
goal:$\max_\theta E_{\mathbf{o}_t \sim p_{data}(\mathbf{o}_t)}[\log \pi_\theta(\mathbf{a}_t|\mathbf{o}_t)]$

test under $p_{\pi_\theta}(\mathbf{o}_t)$
expect:$p_{\pi_\theta}(\mathbf{o}_t) = p_{data}(\mathbf{o}_t)$
$p_{data}(\mathbf{o}_t) \neq p_{\pi_\theta}(\mathbf{o}_t)$
直观理解：微小的区别会导致后续更大的漂移，本质是因为数据对不是i.i.d（独立同分布）的，

未来受过去影响



2.2理论分析：算法层面O(ϵT^2)复杂度太高

形式化证明：

what makes a learned $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ good or bad?

如果如上图的 max 函数，在微小差别时如果做出显著不同的动作，这个函数无法有效区分，我们需要更好的奖励/损失函数

成本cost

$$c(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} 0 & \text{if } \mathbf{a}_t = \pi^*(\mathbf{s}_t) \\ 1 & \text{otherwise} \end{cases}$$

所以目标转换成：

Goal: minimize $E_{\mathbf{s}_t\sim p_{\pi_\theta}(\mathbf{s}_t)}[c(\mathbf{s}_t, \mathbf{a}_t)]$

2.2.1证明 前提for all $\mathbf{s} \in \mathcal{D}_{\text{train}}$ 即对于每一个真正在训练集中的state都有如下假设

assume: $\pi_\theta(\mathbf{a} \neq \pi^*(\mathbf{s})|\mathbf{s}) \leq \epsilon$ for all $\mathbf{s} \in \mathcal{D}_{\text{train}}$

即 对于训练集中每一个 $\mathbf{s}$,策略π_θ 都能很好地模仿人类专家的动作

$$E\left[\sum_t c(\mathbf{s}_t, \mathbf{a}_t)\right] \leq \epsilon T + (1-\epsilon)(\epsilon(T-1) + (1-\epsilon)(\ldots))$$

最后总成本的期望计算：
第一个t犯错了，那么之后每一个t都犯错，加和为ϵT
第一个t没犯错，在$1-\epsilon$ 概率下有ϵ 犯错，之后$T-1$ 都犯错，加和为$(1-\epsilon)(\epsilon T-1)$
$\cdots$
将$(1-\epsilon)$ 提取公因式得到上式
等式右侧在ϵ 很小时为T terms, each $O(\epsilon T)$
所以左侧期望上限为$O(\epsilon T^2)$

2.2.2证明 前提for $\mathbf{s} \sim p_{\text{train}}(\mathbf{s})$ 即对于任意s在分布中的采样

assume: $\pi_\theta(\mathbf{a} \neq \pi^*(\mathbf{s})|\mathbf{s}) \leq \epsilon$ for $\mathbf{s} \sim p_{\text{train}}(\mathbf{s})$
actually enough for $E_{p_{\text{train}}(\mathbf{s})}[\pi_\theta(\mathbf{a} \neq \pi^*(\mathbf{s})|\mathbf{s})] \leq \epsilon$
if $p_{\text{train}}(\mathbf{s}) \neq p_\theta(\mathbf{s})$: $p_\theta(\mathbf{s}_t) = (1-\epsilon)^t p_{\text{train}}(\mathbf{s}_t) + (1-(1-\epsilon)^t) p_{\text{mistake}}(\mathbf{s}_t)$

$$p_\theta(\mathbf{s}_t) = \underbrace{(1-\epsilon)^t p_{\text{train}}(\mathbf{s}_t)}_{\text{probability we made no mistakes}} + \underbrace{(1-(1-\epsilon)^t) p_{\text{mistake}}(\mathbf{s}_t)}_{\text{some other distribution}}$$

$$|p_\theta(\mathbf{s}_t) - p_{\text{train}}(\mathbf{s}_t)| = |(1-(1-\epsilon)^t)(p_{\text{mistake}}(\mathbf{s}_t) - p_{\text{train}}(\mathbf{s}_t))| \leq 2(1-\epsilon)^t \leq 2\epsilon t$$

$$\sum_t E_{p_\theta(\mathbf{s}_t)}[c_t] = \sum_t \sum_{\mathbf{s}_t} p_\theta(\mathbf{s}_t) c_t(\mathbf{s}_t) \leq \sum_t \sum_{\mathbf{s}_t} p_{\text{train}}(\mathbf{s}_t) c_t(\mathbf{s}_t) + |p_\theta(\mathbf{s}_t) - p_{\text{train}}(\mathbf{s}_t)| c_{\max} \leq \sum_t \epsilon + 2\epsilon t \leq \epsilon T$$

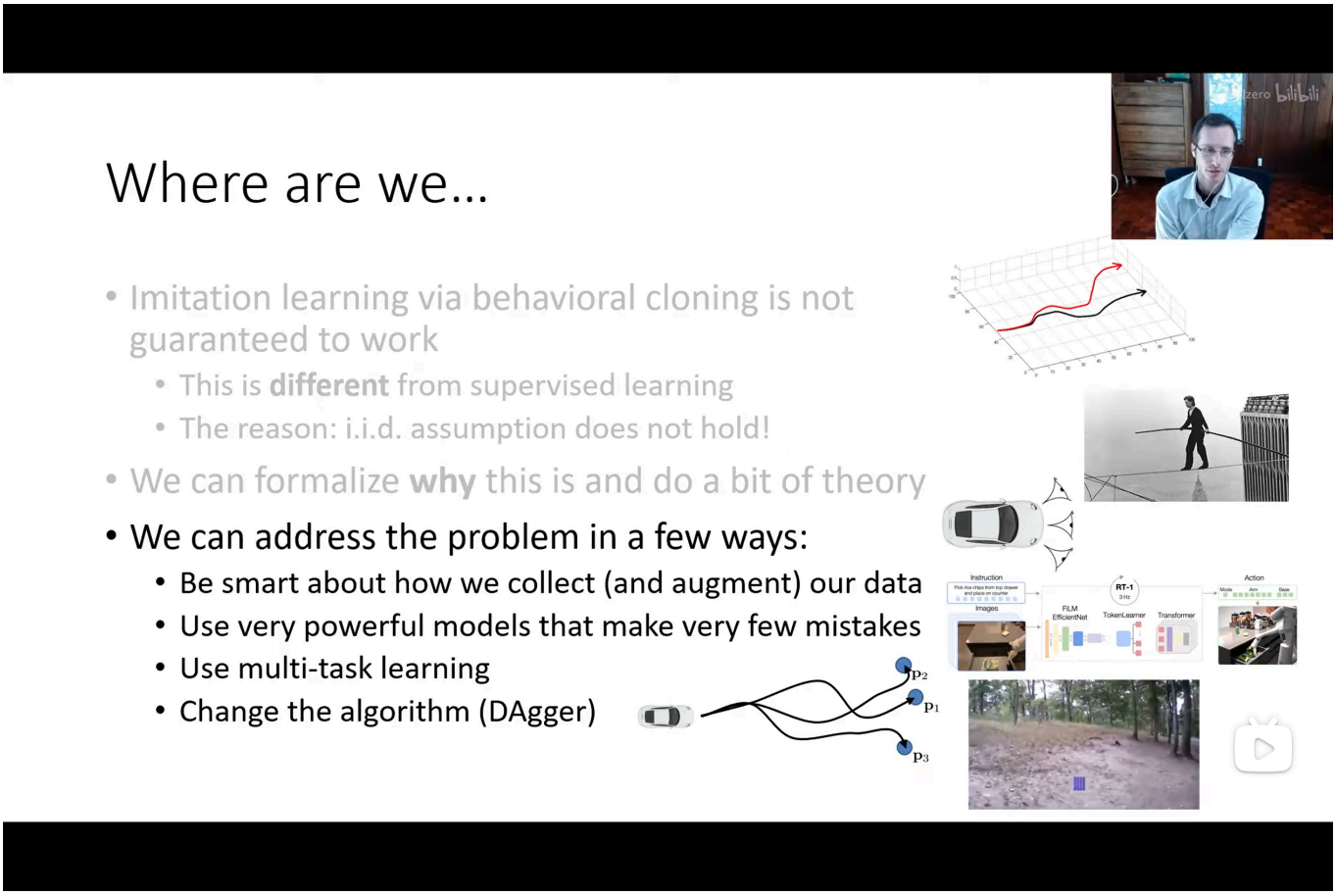
即结果最终为$O(\epsilon T^2)$

2.2.3不同前提概念区分

- 假设1：for all $\mathbf{s} \in \mathcal{D}_{\text{train}}$
  - **含义：** 这表示最初的假设是，上述的“犯错概率小于 ϵ”这个条件对**所有在训练数据集 D_{train}中的状态 s 都成立。**
  - 这通常意味着这个条件过于严格，或者在实际操作中很难保证。在实践中，我们通常无法保证对于训练数据中的每一个单一状态，策略都能完美地满足这个 ϵ 误差界限。更实际的假设是基于期望。
- 假设2：for $\mathbf{s} \sim p_{\text{train}}(\mathbf{s})$ 和 actually enough for $E_{p_{\text{train}}(\mathbf{s})}[\pi_\theta(\mathbf{a} \neq \pi^*(\mathbf{s})|\mathbf{s})] \leq \epsilon$
  - $p_{train}(\mathbf{s})$： 这是训练数据中状态的分布。简单来说，就是专家在收集训练数据时，他会经过某些状态，这些状态出现的频率和方式就构成了$p_{train}(\mathbf{s})$。
  - $\mathbf{s} \sim p_{train}(\mathbf{s})$： 表示状态 s 是从训练数据分布 **$p_{train}(\mathbf{s})$中抽样得到的。
  - *$E_{p_{\text{train}}(\mathbf{s})}[\pi_\theta(\mathbf{a} \neq \pi^*(\mathbf{s})|\mathbf{s})] \leq \epsilon$: 这是期望操作，表示对括号内的表达式在 $p_{train}(\mathbf{s})$ 分布下取平均值。
  - **整体含义：** 这句话的意思是，实际上，我们只需要保证**在训练数据分布 $p_{train}(\mathbf{s})$ 下，策略 π_θ 犯错的期望概率（平均错误率）小于或等于 ϵ 就足够了。
  - **为什么“足够”：** 这是因为模仿学习通常通过梯度下降等优化方法，最小化一个损失函数（如负对数似然），这个损失函数通常是关于训练数据分布上的期望。所以，我们通过

优化，只能保证在平均意义上（即期望值）满足某个性能指标，而不能保证对每个单独的训练样本都完美。

阶段总结1



3.解决方法概览

3.1不完美的数据可能比完美的数据更好

人为引入一些错误的校正数据可以帮助模仿学习减少分布漂移 i.e. nvidia的车 三个摄像头左摄像头标记向右转

3.2使用更强的模型

待补

3.3 multi-task learning 多任务学习

3.3.1 Goal-conditional behavioral cloning 目标条件行为克隆

goal：训练一个策略$\pi_\theta(\mathbf{a}|\mathbf{s},\mathbf{g})$ 这里$\mathbf{s}$ 时输入状态state，而$\mathbf{g}$ 是到达的目标
比如有三条路线，去向P_1, P_2, P_3 三个地方，通过调用人类专家最后一个state的状态，我可以确定路线通向哪里，也就是$\mathbf{g}$

原理：这样带来同样目标的更多数据，有一些是单一目标训练中你不会使用到的数据，比如去往P_1 不会用到去往P_2 的一些数据，但是带来更多数据丰富性，有望在你偏离专家演示时(分布漂移时)及时修正。即当你遇到在一个任务中不熟悉的场景时，可能此时是其他任务熟悉的场景，有助于你修正。
but! 这会在理论上带来新的问题(尽管实践上更好) 有两个分布漂移!

- **第一个分布漂移：** 策略在部署时遇到的**状态 s 的分布** $P_{\pi_\theta}(\mathbf{s})$ 会与训练时专家演示的状态分布 $P_{data}(\mathbf{s})$ 发生漂移。
- **第二个分布漂移：** 策略在部署时被赋予的**目标 g 的分布** $P_{inference}(\mathbf{g})$ 可能会与训练时演示数据中的目标分布 $P_{data}(\mathbf{g})$发生漂移。即实际部署时可能会遇到没见到的目标或者伪目标等。

3.4 DAgger： Dataset Aggregation

can we make $p_{\text{data}}(\mathbf{o}_t) = p_{\pi_\theta}(\mathbf{o}_t)$?
idea: instead of being clever about $p_{\pi_\theta}(\mathbf{o}_t)$, be clever about $p_{\text{data}}(\mathbf{o}_t)$!
goal: collect training data from $p_{\pi_\theta}(\mathbf{o}_t)$ instead of $p_{\text{data}}(\mathbf{o}_t)$
how? just run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ but need labels $\mathbf{a}_t$!

1. train $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, \ldots, \mathbf{o}_N, \mathbf{a}_N\}$
2. run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions $\mathbf{a}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$
   实际上，就是之前 i.e. {nvidia的车 三个摄像头左摄像头标记向右转 右摄像头标记向左转 中间摄像头标记直行，这样车向左时会自动向右转向，其余方向同理。} 这个例子的加强版，与其人工预期策略π_θ 可能会犯的错，不如直接运行π_θ ，收集错误数据，人工标注后并回原训练数据集。

strike how to prove DAgger has a bound? especially a linear bound?

第四讲 强化学习 基本概念

1.基础定义

1.1 MDP定义 马尔可夫决策过程

Markov decision process
$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$
$\mathcal{S}$ -- state space -- states $s \in \mathcal{S}$ (discrete or continuous)
$\mathcal{A}$ -- action space -- actions $a \in \mathcal{A}$ (discrete or continuous)
$\mathcal{T}$ -- transition operator (now a tensor!)
r -- reward function $r(s_t, a_t)$ 输入为$\mathbf{s_t}, \mathbf{a_t}$
$r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$

let $\mu_{t,i} = p(s_t = i)$
let $\xi_{t,k} = p(a_t = k)$
let $\mathcal{T}_{i,j,k} = p(s_{t+1} = j | s_t = i, a_t = k)$

$\mu_{t+1,i} = \sum_{j,k} \mathcal{T}_{i,j,k} \mu_{t,j} \xi_{t,k}$

整个公式的含义可以理解为：

为了计算在下一个时间步 (t+1) 系统处于某个特定状态 i 的总概率 $\mu_{t+1,i}$ ，我们需要考虑所有可能导致它发生的“路径”：

- **对于每一个可能的当前状态 j** (系统可能在时间 t 处于的状态)；
- **和每一个可能的动作 k** (智能体可能在时间 t 采取的动作)；
- **计算这条路径的概率：** (在时间 t 处于状态 j 的概率 $\mu_{t,j}$) 乘以 (在时间 t 采取动作 k 的概率 $xi_{t,k}$) 乘以 (从状态 j 采取动作 k 转移到状态 i 的概率$\mathcal{T}_{i,j,k}$)。
- **将所有这些路径的概率相加，** 就能得到在时间 t+1 处于状态 i 的总概率。

1.2 POMDP 部分可观察马尔可夫决策过程 partially observed Markov decision process

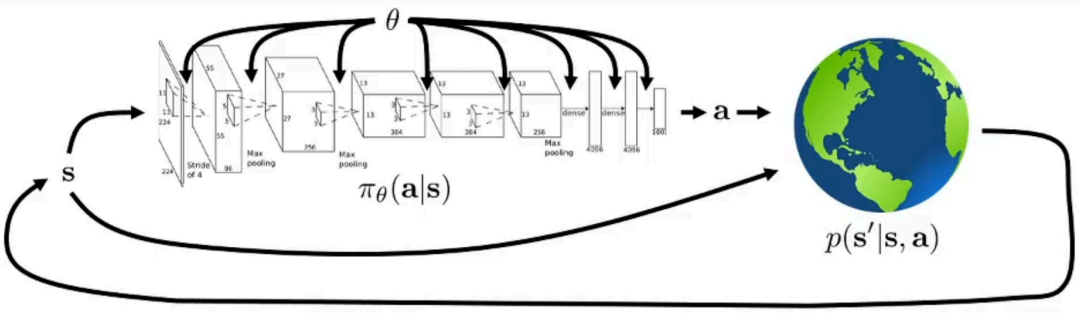
在现实世界中，不是所有state都已知，我们只能观测到部分observation，$\mathbf{o}_k$

$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{E}, r\}$
$\mathcal{S}$ -- state space -- states $s \in \mathcal{S}$ (discrete or continuous)
$\mathcal{A}$ -- action space -- actions $a \in \mathcal{A}$ (discrete or continuous)
$\mathcal{O}$ -- observation space -- observations $o \in \mathcal{O}$ (discrete or continuous)
$\mathcal{T}$ -- transition operator (now a tensor!)
$\mathcal{E}$ -- emission probability $p(o_t|s_t)$ 在$\mathbf{s}_t$ 下观测到$\mathbf{o}_t$ 的概率
r -- reward function $r(s_t, a_t)$ 输入为$\mathbf{s_t}, \mathbf{a_t}$
$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

注意！此时没有计算公式和算子！后续补充

2.强化学习的目标

2.1目标表达式



$$p_\theta(s_1, \mathbf{a}_1, \ldots, s_T, \mathbf{a}_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

说明

这里等式右边$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ 是收到训练策略π_θ 影响的，但是$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ 是与策略无关的，仅由环境决定
其中等式左边也可以写作：

$$\underbrace{p_\theta(s_1, \mathbf{a}_1, \ldots, s_T, \mathbf{a}_T)}_{p_\theta(\tau)}$$

其中τ 表示轨迹 trajectory

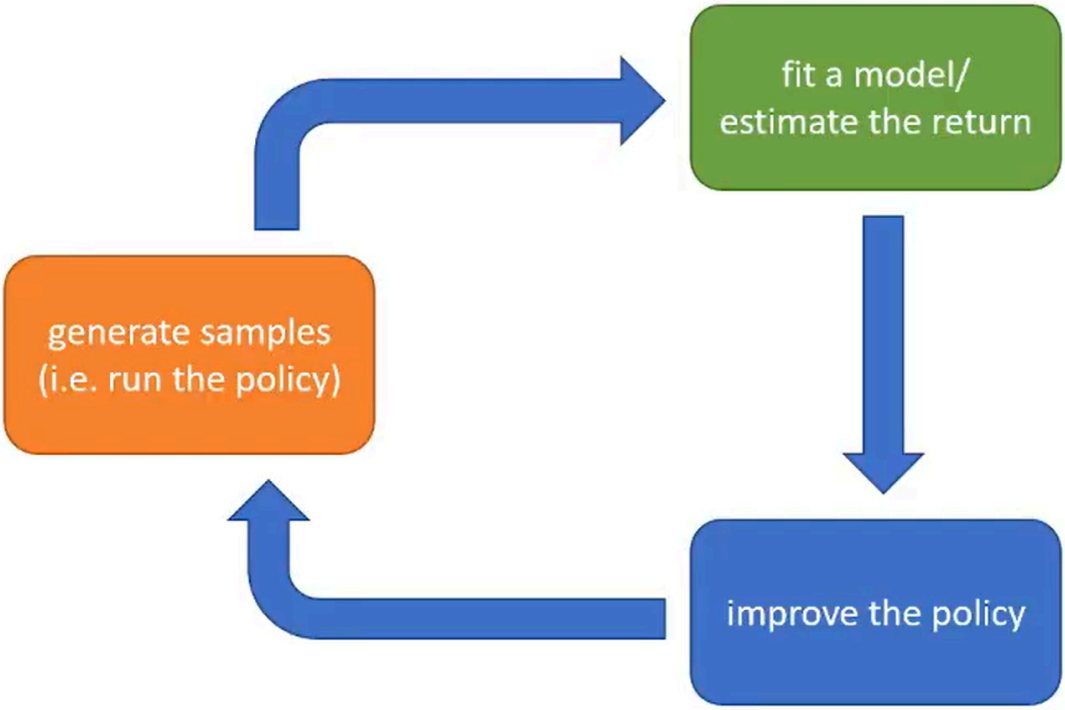
$$\theta^* = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, \mathbf{a}_t) \right]$$

而θ^* 就是我们希望训练出来的参数
$E_{\tau \sim p_\theta(\tau)}$ 是指对所有**可能轨迹 τ 的总奖励**求期望。其中我们考虑的轨迹 τ 并不是任意的轨迹，而是根据智能体当前遵循的策略（由参数 θ 定义）所产生的概率分布 $p_\theta(\tau)$ 来随机生成的轨迹

强化学习的目标是找到一套最优的策略参数 θ^*，使得智能体在遵循该策略 π_{θ^*} 时，能够最大化它在所有可能轨迹上获得的期望总奖励。

3.算法实现

每个算法都基本包含三部分：



不同算法这三部分的复杂度不同，侧重点不同

算法简介（直接学习策略/学习价值函数）

- **策略梯度：** 直接优化策略。
- **基于价值：** 学习价值函数来间接得到策略。
- **Actor-Critic：** 结合前两者，一个学习策略，一个评估策略。
- **基于模型：** 先学习环境模型，再用模型进行规划或辅助学习。

3.1 Policy gradient 策略梯度

3.1.1 有限时间步

$$\theta^* = \arg\max_\theta \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim p_\theta(s_t,a_t)}\left[r(s_t,a_t)\right]$$

记：$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim p_\theta(s_t,a_t)}\left[r(s_t,a_t)\right]$ ，我们怎么估计策略好坏？

$$J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t} r(s_{i,t},a_{i,t})$$

其中下标含义为：一共收集N条轨迹(Trajectory / Episode)，每个轨迹中有不同的时间步t，$s_{i,t}$ 为在第i 个轨迹中的第t 个时间步的状态

(strike!):

- **Rollout** 是一个**动作**，它产生**轨迹**。
- **轨迹** 是一个**数据结构**，记录了智能体与环境的一次完整交互。
- **Batch** 是一个**数据集合**，由多个轨迹或多个经验元组组成，用于高效地更新学习算法。

数学推导：

目标函数：

$$\theta^* = \arg\max_\theta J(\theta)$$

根据**期望**与**微分**关系将$J(\theta)$ 转变为可以求梯度的形式

$$J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\big[r(\tau)\big] = \int p_\theta(\tau) r(\tau) d\tau$$

$$\nabla_\theta J(\theta) = \nabla_\theta \int p_\theta(\tau) r(\tau) d\tau = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau$$

接下来化简$\nabla_\theta p_\theta(\tau)$ ，由微积分基本定理：

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \nabla_\theta \log p_\theta(\tau)$$

接下来

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

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

$$\nabla_\theta \log p_\theta(\tau) = \nabla_\theta \left(\log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \sum_{t=1}^{T} \log p(s_{t+1}|s_t, a_t) \right)$$

$$\nabla_\theta \log p(s_1) = 0, \quad \nabla_\theta \log p(s_{t+1}|s_t, a_t) = 0$$

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

带入有：

$$\int p_\theta(\tau) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) r(\tau) d\tau$$

将积分形式重新转换为期望形式有：

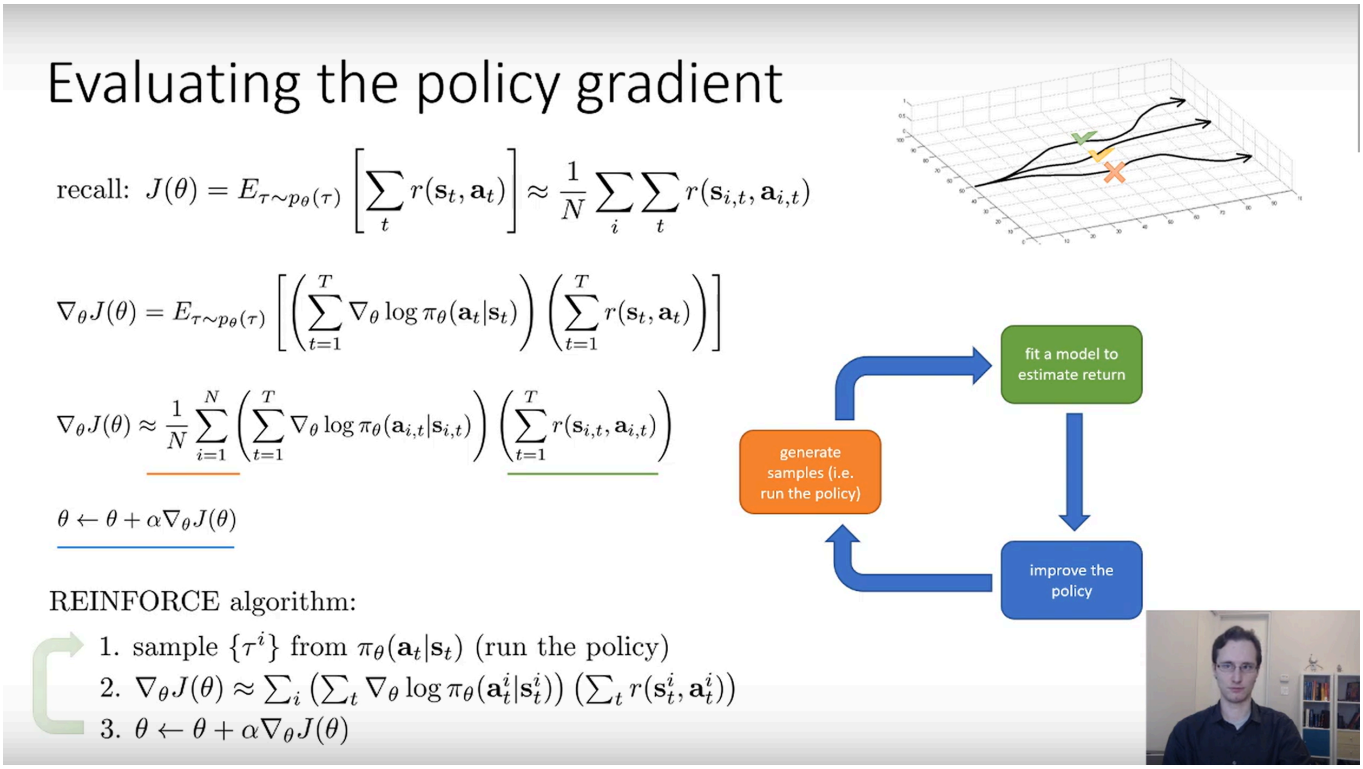
$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(\sum_{t=1}^{T} r(s_t,a_t)\right)\right]$$

由于不知道$p_\theta(\tau)$ 的形式，所以无法直接计算期望，转而使用蒙特卡洛近似，等于号变成**约等于**

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t} \mid s_{i,t})\right)\left(\sum_{t=1}^{T} r(s_{i,t},a_{i,t})\right)\right)$$

也就是随机采样很多(N)条轨迹，最终这些轨迹的**平均值**会收敛到**期望值**
更新策略：

$$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$$



按照强化学习算法的结构(右侧三个颜色方块)，在左边公式也用对应颜色标出

1. 橙色部分为采样，agent进行N 次采样获得轨迹
2. 绿色部分为评估，用$J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_t r(s_{i,t},a_{i,t})$ 来评估策略好坏
3. 蓝色部分为改进，引入学习率采用梯度上升(这里我们希望最大化θ 所以不是梯度下降)
   这也组成了基本的REINFORCE算法
   discount版本见策略梯度改进部分引入discount的笔记

(strike!)
与监督学习(最大似然估计的区别？)
最大似然中是最大化人类动作出现的概率，也就是让训练集中的动作出现概率尽可能大
而强化学习中引入了奖励信号，如果一个轨迹奖励信号是负数，那么策略更新就会减少这条轨迹

出现的概率，也就是说，策略梯度实现了追求"好"轨迹的目的，而不是仅仅拟合存在的所有轨迹

What did we just do?

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$

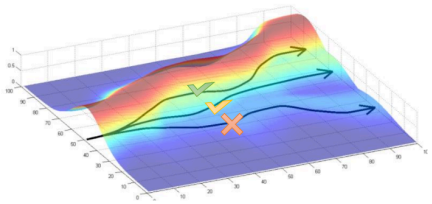
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\underbrace{\nabla_\theta \log \pi_\theta(\tau_i)}_{\sum_{t=1}^{T}\nabla_\theta \log_\theta \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} r(\tau_i)$$

maximum likelihood: $\nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \log \pi_\theta(\tau_i)$

good stuff is made more likely

bad stuff is made less likely

simply formalizes the notion of “trial and error”!



REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i,\mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

但是！！！
策略梯度方法中：梯度方差较大，导致训练不稳定，难以直接应用
接下来尝试解决方差大的问题

利用因果关系减少方差

因果关系：在时间 t' 的策略不能影响在时间 t 时的奖励，如果 $t < t'$

$$\text{原始公式：}\ \nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\cdot\left(\sum_{t'=1}^{T} r(s_{t'}^i,a_{t'}^i)\right)$$

$$\text{改进后公式：}\ \nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\cdot\left(\sum_{t'=t}^{T} r(s_{t'}^i,a_{t'}^i)\right)$$

区别：奖励求和起始下标变化，从当t' 从t 开始求和时，在t 时间步之前的奖励不再影响策略的改进，这样总的求和数值减少，方差变小

$$\text{与Q相联系：}\ \hat{Q}_{i,t} = \sum_{t'=t}^{T} r(s_{t'}^i,a_{t'}^i) \qquad \text{称为reward to go！！！}$$

这里$\hat{Q}$ 是指这是单样本估计，**单样本估计也是导致方差大的主要因素！！！** 当使用单独的样本来替代整体数据分布得出的期望时，难免有随机波动，进而导致单样本数据之间有大波动，方差大!

Baseline 减少方差

引入一个基线(baseline)

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p_\theta(\tau_i) \cdot \big(r(\tau_i) - b\big)$$

我们可以选择不同的baseline
这里使用：当前批次所有轨迹的平均回报

$$b = \frac{1}{N} \sum_{i=1}^{N} r(\tau_i)$$

下面证明引入b后方差减小的基础上，我们训练的目标——期望没有改变(unbiased)
为了证明：$\mathbb{E}_{\tau \sim p_\theta}\big[\nabla_\theta \log p_\theta(\tau) \cdot b\big] = 0$

$$\begin{aligned}
\mathbb{E}\big[\nabla_\theta \log p_\theta(\tau) \cdot b\big] &= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) \cdot b \, d\tau \\
&= \int \nabla_\theta p_\theta(\tau) \cdot b \, d\tau \\
&= b \cdot \nabla_\theta \int p_\theta(\tau) \, d\tau \qquad \text{概率分布积分为1} \\
&= b \cdot \nabla_\theta 1 \\
&= 0
\end{aligned}$$

所以，只要b不是策略 π_θ 的直接函数），我们就可以放心地从回报中减去它，而不会改变梯度估计的正确方向

$$b^* = \frac{\mathbb{E}\big[\big(\nabla_\theta \log p_\theta(\tau)\big)^2 \cdot r(\tau)\big]}{\mathbb{E}\big[\big(\nabla_\theta \log p_\theta(\tau)\big)^2\big]}$$

如果不使用平均基线，而是最优基线呢？怎么求出最优基线是多少？
分析方差：

$$\mathrm{Var}[x] = \mathbb{E}[x^2] - \big(\mathbb{E}[x]\big)^2$$

$$\mathrm{Var} = \mathbb{E}_{\tau \sim p_\theta(\tau)}\big[\big(\nabla_\theta \log p_\theta(\tau) \cdot (r(\tau) - b)\big)^2\big] - \Big(\mathbb{E}_{\tau \sim p_\theta(\tau)}\big[\nabla_\theta \log p_\theta(\tau) \cdot (r(\tau) - b)\big]\Big)^2$$

由于引入b后期望不变(unbiased)，即
$\mathbb{E}_{\tau \sim p_\theta(\tau)}\big[\nabla_\theta \log p_\theta(\tau) \cdot (r(\tau) - b)\big] = \mathbb{E}_{\tau \sim p_\theta(\tau)}\big[\nabla_\theta \log p_\theta(\tau) \cdot r(\tau)\big]$
所以对b求导后，Var第二项求导为0

$$\frac{d\,\mathrm{Var}}{db} = \frac{d}{db}\,\mathbb{E}\big[g(\tau)^2 \cdot (r(\tau) - b)^2\big]$$

$$\frac{d}{db}\Big(\mathbb{E}\big[g(\tau)^2 r(\tau)^2\big] - 2b\,\mathbb{E}\big[g(\tau)^2 r(\tau)\big] + b^2\,\mathbb{E}\big[g(\tau)^2\big]\Big)$$

令导数为零，极值条件

$$-2\,\mathbb{E}\big[g(\tau)^2 r(\tau)\big] + 2b\,\mathbb{E}\big[g(\tau)^2\big] = 0$$

$$b = \frac{\mathbb{E}\big[g(\tau)^2 r(\tau)\big]}{\mathbb{E}\big[g(\tau)^2\big]}$$

此处可以看到最优bseline是与轨迹有关的，所以实际应用上时每次轨迹的baseline需要重新估计。因此实践中一般不用最优基线

On-Policy vs. Off-Policy

On-Policy

以上的策略梯度是同策略的！即用于估计梯度的样本必须是当前正在优化的策略π_θ 生成的，所以在每次更新策略之后必须重新获得样本，效率低、成本高

Off-Policy

我们希望能够利用过去的旧策略生成的数据来训练新策略。引入**重要性采样**

引理

如果想计算：$\mathbb{E}_{x\sim p(x)}[f(x)]$ 但是采样来自另一个分布$q(x)$ 可以通过配凑系数来调整为：

$$\mathbb{E}_{x\sim p(x)}[f(x)] = \mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)} f(x)\right]$$

我们在强化学习中目标为最大化$J(\theta)$ ，但是只有之前策略$\bar{\pi}$ 生成的数据，服从分布$\bar{p}(\tau)$

$$J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}[r(\tau)]$$

$$J(\theta) = \mathbb{E}_{\tau\sim \bar{p}(\tau)}\left[\frac{p_\theta(\tau)}{\bar{p}(\tau)} r(\tau)\right]$$

其中重要性权重为$\frac{p_\theta(\tau)}{\bar{p}(\tau)}$

$$\text{轨迹概率链式法则展开有：} p(\tau) = p(s_1)\prod_{t=1}^{T}\pi(a_t|s_t)p(s_{t+1}|s_t,a_t)$$

因为初始概率$p(s_1)$ 和环境转移概率$p(s_{t+1}|s_t,a_t)$ 与策略无关，所以可以消去，得到：

$$\frac{p_\theta(\tau)}{\bar{p}(\tau)} = \prod_{t=1}^{T}\frac{\pi_\theta(a_t|s_t)}{\bar{\pi}(a_t|s_t)}$$

使用重要性采样后的目标函数：这里θ' 是正在优化的策略(目标策略)，θ 是过去的策略

$$J(\theta') = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\frac{p_{\theta'}(\tau)}{p_\theta(\tau)} r(\tau)\right]$$

$$\nabla_{\theta'} J(\theta') = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\frac{p_{\theta'}(\tau)}{p_\theta(\tau)} \nabla_{\theta'} \log p_{\theta'}(\tau) r(\tau) \right]$$

$$\nabla_{\theta'} J(\theta') = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\left(\prod_{t=1}^{T} \frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \right) \left(\sum_{t=1}^{T} \nabla_{\theta'} \log \pi_{\theta'}(a_t|s_t) \right) r(\tau) \right]$$

但是此时，$\left(\prod_{t=1}^{T} \frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \right)$ 随着T的增加会指数级增长或者衰减，导致方差极大！！！

利用因果关系优化：得到理论上比较精确的估计

$$\nabla_{\theta'} J(\theta') = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \left(\prod_{t'=1}^{t} \frac{\pi_{\theta'}(a_{t'}|s_{t'})}{\pi_\theta(a_{t'}|s_{t'})} \right) \nabla_{\theta'} \log \pi_{\theta'}(a_t|s_t) \left(\sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \right) \right]$$

但是仍然有指数级增长或者衰减的问题，需要进一步近似以供实际应用

1. 直接将$\left(\prod_{t=1}^{T} \frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \right)$ 近似为只有一个时间步的$\frac{\pi_{\theta'}(a_{i,t}|s_{i,t})}{\pi_\theta(a_{i,t}|s_{i,t})}$ 具体可行性分析在后续笔记出现
2. 将期望运算转化为蒙特卡洛近似，使用Σ 来方便计算

$$\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{\pi_{\theta'}(a_{i,t}|s_{i,t})}{\pi_\theta(a_{i,t}|s_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(a_{i,t}|s_{i,t}) \hat{Q}_{i,t}$$

若想在pytorch等自动微分工具中实现梯度下降，需要构造出一个标量损失函数(没有期望运算参与)，这样对这个损失函数求导正好数值上等于策略梯度：

$$\tilde{J}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log \pi_\theta(a_{i,t}|s_{i,t}) \hat{Q}_{i,t}$$

通过最小化损失函数$L(\theta) = -\tilde{J}(\theta)$ 即可实现最大化$J(\theta)$

策略梯度实际应用(On-Policy)：

Policy gradient with automatic differentiation

Pseudocode example (with discrete actions):

Policy gradient:

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# q_values – (N*T) x 1 tensor of estimated state-action values
# Build the graph:
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, q_values)
loss = tf.reduce_mean(weighted_negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

$$\tilde{J}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\hat{Q}_{i,t}$$

q_values

策略梯度问题问题：

1. 高方差，如前文所示
2. On-Policy采样效率低
3. 优化器难以选择Adam还行，后续会有更好的
4. 策略的不同参数对策略输出动作概率的影响程度不同、且差异大。后续引入**自然策略梯度 (Natural Policy Gradient)** 解决

对策略梯度进行优化！

因为单样本$\hat{Q}$ 会导致方差大，我们引入了更准确的Q 作为奖励函数和$V(s)$ 作为baseline，二者合在一起称为优势函数$A(s,a)$ 使得估计更加准确

真实预期的未来奖励Q:

$Q(\mathbf{s}_t,\mathbf{a}_t)=\sum_{t'=t}^{T}E_{\pi_\theta}[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t,\mathbf{a}_t]$

引入$V(s)$ 作为基线，衡量了某个状态的“好坏程度”

$V^\pi(\mathbf{s}_t)=E_{\mathbf{a}_t\sim\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}\left[Q^\pi(\mathbf{s}_t,\mathbf{a}_t)\right]$

最后以上二者形成的优势函数：

$A^\pi(\mathbf{s}_t,\mathbf{a}_t)=Q^\pi(\mathbf{s}_t,\mathbf{a}_t)-V^\pi(\mathbf{s}_t)$

最后策略梯度优化为：

$\nabla_\theta J(\theta)\approx\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})A^\pi(\mathbf{s}_{i,t},\mathbf{a}_{i,t})$

按照之前的强化学习算法三个组成部分，我们优化了奖励函数部分，那么在评估模型部分应该学习哪个量Q，V，A?

Value function fitting

$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t\right]$

$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t)]$

$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t)$

$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) A^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$

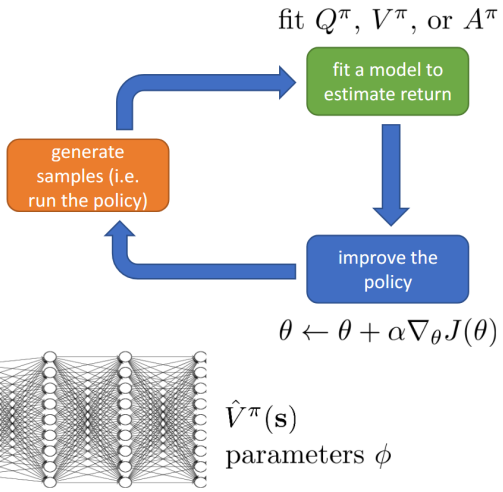
fit *what* to *what*?

Q^π, V^π, A^π?

$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \sum_{t'=t+1}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t\right]$

$A^\pi(\mathbf{s}_t, \mathbf{a}_t) \approx r(\mathbf{s}_t, \mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t)$

let's just fit $V^\pi(\mathbf{s})$!



由图中左下，Q与V之间有关系，与MDP中一样！所以我们在fit阶段就学习$V^\pi(s)$ 这个量！
使用一个神经网络来逼近$V^\pi(s)$ 这个量，记神经网络为$V_\phi^\pi(\mathbf{s}_i)$
使用监督回归来学习这个神经网络，，损失函数为：

$$L(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$

其中y_i 为 $y_{i,t} = \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$ 是关于单样本s_t 的未来奖励(单次样本轨迹采样)，尽管不如多次样本轨迹采样(只是因为多次采样更能靠近真实值)

$$V^\pi(s_t) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})$$

但是实践中通常不能回到相同的状态s_i 所以多次基于相同状态采样不现实。

但实际上还能更好！
可以使用之前已经优化好的$V_\phi^\pi(\mathbf{s}_{i,t+1})$ (这里没有空格是因为针对所有状态而言的，图中是针对状态$\mathbf{s}_{i,t+1}$而言的

Can we do better?

ideal target: $y_{i,t} = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_{i,t}\right] \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + V^{\pi}(\mathbf{s}_{i,t+1}) \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \underbrace{\hat{V}_{\phi}^{\pi}(\mathbf{s}_{i,t+1})}$

Monte Carlo target: $y_{i,t} = \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$ directly use previous fitted value function!

training data: $\left\{\left(\mathbf{s}_{i,t}, \underbrace{r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}_{\phi}^{\pi}(\mathbf{s}_{i,t+1})}_{y_{i,t}}\right)\right\}$

supervised regression: $\mathcal{L}(\phi) = \frac{1}{2} \sum_{i} \left\| \hat{V}_{\phi}^{\pi}(\mathbf{s}_i) - y_i \right\|^2$

sometimes referred to as a “bootstrapped” estimate

后续算法都是基于这个算式进行的，该方法称为自举(bootstrapped)
这个算式好处在于

1. 每个时间步都可以更新
2. 减少方差(相比于原先的整条轨迹 $y_{i,t}$ 依赖所有随机性——动作和状态转移，这里只和一步的奖励r和下一个状态的价值估计有关)

但是坏处在于

1. 依赖正在学习的网络 V_{ϕ}^{π} ，可能会导致不稳定

引入折扣因子discount

像MDP一样引入discount来解决当T趋于无穷或者比较大的情况
MDP中discount：CS188笔记
这里除了右上角是actor-critic算法见(下部分笔记)之外，其余的公式都是基于蒙特卡洛估计的算法(比如REINFORCE)

Aside: discount factors for policy gradients

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s}_{i,t+1})$$

$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}^{\pi}_{\phi}(\mathbf{s}_i) - y_i \right\|^2$$

with critic:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \overbrace{\left(r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s}_{i,t+1}) - \hat{V}^{\pi}_{\phi}(\mathbf{s}_{i,t}) \right)}^{\hat{A}^{\pi}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}$$

what about (Monte Carlo) policy gradients?

option 1: $$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

option 2: $$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} \gamma^{t-1} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

not the same!

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \gamma^{t-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

(later steps matter less)

我们通常使用option1，因为option2在靠后的阶段对梯度仍然乘了折扣因子导致最后的策略尽管正确但也不重要，这不是我们想要的，也就是过分强调初始策略、过分忽视后期策略

Which version is the right one?

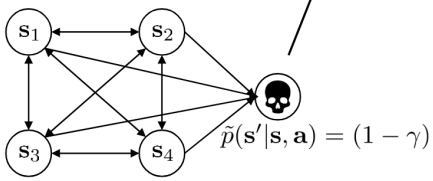
option 1: $$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

this is what we actually use...

why?

option 2: $$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \gamma^{t-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

later steps don't matter if you're dead!



Further reading: Philip Thomas, Bias in natural actor-critic algorithms. ICML 2014

Actor-Critic 演员评论家算法

基本思想：

1. Actor（演员）
   Actor 是策略函数，记作 $\pi_\theta(a|s)$，负责根据当前状态 s 按照参数化策略选择动作 a。 它的参数 θ 通过 **策略梯度方法** 进行更新。

2. Critic（评论家）

Critic 是价值函数估计器，用于评估当前状态 s 或状态-动作对 (s,a)的价值，常用的两种形式：

状态值函数 $\hat{V}_{\phi}^{\pi}(s)$

状态-动作值函数 $\hat{Q}_{\phi}^{\pi}(s,a)$

这里的参数 ϕ 通过最小化预测值与实际回报之间的误差进行更新（通常是平方误差）。

优势函数（Advantage Function）

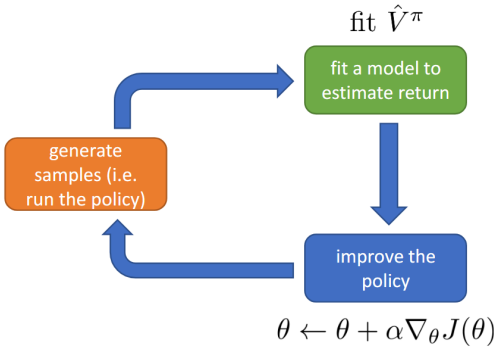
优势函数$A^{\pi}(s,a)$ 衡量在状态 s 下，采取动作 a 比该状态的平均水平好多少（尽管这里是单样本，蒙特卡洛采样保证了随着采样增多会向平均值靠拢）。

Actor 利用 Critic 提供的优势函数来更新策略，从而更有效地区分好的和差的动作。

An actor-critic algorithm

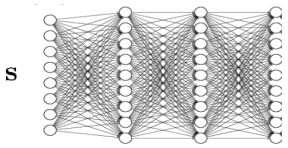
batch actor-critic algorithm:

1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
2. fit $\hat{V}_{\phi}^{\pi}(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^{\pi}(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \hat{V}_{\phi}^{\pi}(\mathbf{s}_i') - \hat{V}_{\phi}^{\pi}(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^{\pi}(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



$$y_{i,t} \approx \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$$

$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_{\phi}^{\pi}(\mathbf{s}_i) - y_i \right\|^2$$



$$V^{\pi}(\mathbf{s}_t) = \sum_{t'=t}^{T} E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t]$$

引入discount后的最终算法：

Actor-critic algorithms (with discount)

batch actor-critic algorithm:

1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
2. fit $\hat{V}^\pi_\phi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update $\hat{V}^\pi_\phi$ using target $r + \gamma \hat{V}^\pi_\phi(\mathbf{s}')$
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}') - \hat{V}^\pi_\phi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

这两种的区别
batch：
先采集一批数据再进行更新
online：
每一个时间步都进行更新

具体设计架构

1. actor和critic使用两个神经网络或者使用两个不同的头但共享大部分底层网络层
2. 使用**并行**来工作，分为同步并行和异步并行
   - **同步并行**：所有工作者收集数据，然后同步地将梯度汇总到主参数服务器进行更新。
   - **异步并行（Asynchronous Actor-Critic, A3C）**：每个工作者独立地与环境交互、计算梯度，然后异步地更新共享的全局参数。这允许更快的训练，但也可能导致一些梯度过时的问题。
3. 普通actor-critic算法仍然是On-Policy的，能不能改造成Off-Policy?

off-policy actor-critic

引入经验缓冲区(replay buffer)
但是会有一些问题，体现在

1. 计算V 时，使用到了s′ 但是这个s‘应该是由当前最新策略产生的而不是调用经验缓冲区中元组(s, a, s', r) 中的s’(这里s‘是由旧策略产生的！)。所以从学习V转变为学习Q，但是$Q(s'_i, a'_i)$ 中的a'_i 同理必须是最新策略π_θ 产生的，也就是图中所示：$a'_i \sim \pi_\theta(a'_i|s'_i)$
损失函数也进行相应调整$\mathcal{L}(\phi) = \frac{1}{N}\sum_i \left\|\hat{Q}^\pi_\phi(s_i, a_i) - y_i\right\|^2$

Fixing the value function

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in $\mathcal{R}$
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer $\mathcal{R}$
3. update $\hat{V}^\pi_\phi$ using targets $y_i = r_i + \gamma\hat{V}^\pi_\phi(\mathbf{s}'_i)$ for each $\mathbf{s}_i$
4. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma\hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
5. $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i)\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
6. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

not the right target value

not the action π_θ would have taken!

~~$V^\pi(\mathbf{s}_t) = \sum_{t'=t}^T E_{\pi_\theta}[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t]$~~

$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^T E_{\pi_\theta}[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t]$

"total reward we get if we take $\mathbf{a}_t$ in $\mathbf{s}_t$...
... and then follow the policy π"

where does this come from?

3. update $\hat{Q}^\pi_\phi$ using targets $y_i = r_i + \gamma\hat{V}^\pi_\phi(\mathbf{s}'_i)$ for each $\mathbf{s}_i, \mathbf{a}_i$
$= r_i + \gamma\hat{Q}^\pi_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$

$\mathcal{L}(\phi) = \frac{1}{N}\sum_i \left\|\hat{Q}^\pi_\phi(\mathbf{s}_i, \mathbf{a}_i) - y_i\right\|^2$

not from replay buffer $\mathcal{R}$!

$\mathbf{a}'_i \sim \pi_\theta(\mathbf{a}'_i|\mathbf{s}'_i)$

$V^\pi(\mathbf{s}_t) = \sum_{t'=t}^T E_{\pi_\theta}[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t] = E_{\mathbf{a}\sim\pi(\mathbf{a}_t|\mathbf{s}_t)}[Q(\mathbf{s}_t, \mathbf{a}_t)]$

2. 计算梯度时使用到了经验缓冲区中，由旧策略产生的(s, a, s', r) 的a，所以也是重新对a进行采样
以当前最新策略π_θ ，s_i 为条件产生新动作$a^\pi_i \sim \pi_\theta(a|s_i)$

Fixing the policy update

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in $\mathcal{R}$
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer $\mathcal{R}$
3. update $\hat{Q}^\pi_\phi$ using targets $y_i = r_i + \gamma\hat{Q}^\pi_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$ for each $\mathbf{s}_i, \mathbf{a}_i$
4. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = Q(\mathbf{s}_i, \mathbf{a}_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
5. $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i)\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
6. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

not the action π_θ would have taken!
use the same trick, but this time for $\mathbf{a}_i$ rather than $\mathbf{a}'_i$!
sample $\mathbf{a}^\pi_i \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)$
$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}^\pi_i|\mathbf{s}_i)\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}^\pi_i)$

not from replay buffer $\mathcal{R}$!

higher variance, but convenient
why is higher variance OK here?

in practice: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}^\pi_i|\mathbf{s}_i)\hat{Q}^\pi(\mathbf{s}_i, \mathbf{a}^\pi_i)$

3. 但是还有无法解决的问题：s_i 并没有从$p_\theta(s)$ 的分布中来，这也是正常的因为有很多来自于经验缓冲区的数据，实际上不会造成我们最优策略失效，因为相当于引入了很多原本不会出现的情况，这些情况最优策略也能处理好，也就是在更大的数据分布下获得最优策略

What else is left?

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in $\mathcal{R}$
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer $\mathcal{R}$
3. update $\hat{Q}^\pi_\phi$ using targets $y_i = r_i + \gamma \hat{Q}^\pi_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$ for each $\mathbf{s}_i, \mathbf{a}_i$
4. $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}^\pi_i|\mathbf{s}_i) \hat{Q}^\pi(\mathbf{s}_i, \mathbf{a}^\pi_i)$ where $\mathbf{a}^\pi_i \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Is there any remaining problem?

$\mathbf{s}_i$ didn't come from $p_\theta(\mathbf{s})$

nothing we can do here, just accept it

intuition: we want optimal policy on $p_\theta(\mathbf{s})$
but we get optimal policy on a *broader* distribution

n-step returns

在整体设计架构上：可以平衡variance和bias
价值函数估计组成的优势函数有较低方差(因为学习到的V较为平滑，不会特别随机)，但是如果价值函数学习较差，会有偏差
蒙特卡洛估计构成的优势函数没有偏差，但是基于单步预测未来的折扣奖励之和是单样本采样，随机性大，方差大
通过引入n步回报平衡两者

$$\hat{A}^\pi_n(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}^\pi_\phi(\mathbf{s}_t) + \gamma^n \hat{V}^\pi_\phi(\mathbf{s}_{t+n})$$

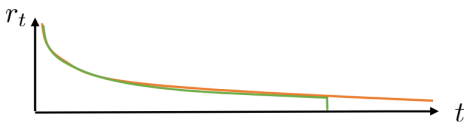
Eligibility traces & n-step returns

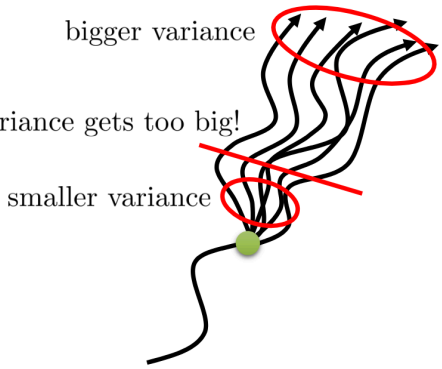
$$\hat{A}^\pi_{\text{C}}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \hat{V}^\pi_\phi(\mathbf{s}_{t+1}) - \hat{V}^\pi_\phi(\mathbf{s}_t)$$

$$\hat{A}^\pi_{\text{MC}}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}^\pi_\phi(\mathbf{s}_t)$$

+ lower variance
- higher bias if value is wrong (it always is)
+ no bias
- higher variance (because single-sample estimate)

Can we combine these two, to control bias/variance tradeoff?



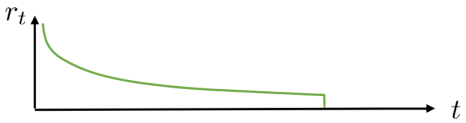


$$\hat{A}^\pi_n(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}^\pi_\phi(\mathbf{s}_t) + \gamma^n \hat{V}^\pi_\phi(\mathbf{s}_{t+n})$$

choosing $n > 1$ often works better!

GAE n-step进阶版

Generalized advantage estimation



Do we have to choose just one n?

Cut everywhere all at once!

$$\hat{A}^\pi_n(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}^\pi_\phi(\mathbf{s}_t) + \gamma^n \hat{V}^\pi_\phi(\mathbf{s}_{t+n})$$

$$\hat{A}^\pi_{\text{GAE}}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{n=1}^{\infty} w_n \hat{A}^\pi_n(\mathbf{s}_t, \mathbf{a}_t)$$

weighted combination of n-step returns

How to weight? Mostly prefer cutting earlier (less variance) $w_n \propto \lambda^{n-1}$ exponential falloff

$$\hat{A}^\pi_{\text{GAE}}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma((1-\lambda)\hat{V}^\pi_\phi(\mathbf{s}_{t+1}) + \lambda(r(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) + \gamma((1-\lambda)\hat{V}^\pi_\phi(\mathbf{s}_{t+2}) + \lambda r(\mathbf{s}_{t+2}, \mathbf{a}_{t+2}) + \dots)$$

$$\hat{A}^\pi_{\text{GAE}}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} (\gamma\lambda)^{t'-t} \delta_{t'} \qquad \delta_{t'} = r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}_{t'+1}) - \hat{V}^\pi_\phi(\mathbf{s}_{t'})$$

similar effect as discount!

option 1: $$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

remember this?
discount = variance reduction!

1. 核心问题引入："我们必须只选择一个 n 吗？" (Do we have to choose just one n?)

n 步优势函数：

$$\hat{A}^\pi_n(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}^\pi_\phi(\mathbf{s}_t) + \gamma^n \hat{V}^\pi_\phi(\mathbf{s}_{t+n})$$

解释：n 步优势函数通过选择不同的 n 来平衡偏差和方差。但是，选择一个最佳的 n 往往需要手动调优，而且这个最佳 n 可能在不同任务或学习阶段有所不同。
GAE 的答案：`Cut everywhere all at once!` (一次性在所有地方截断！)。这意味着我们不只选择一个 n，而是结合所有可能的nn。

2. GAE 的思想：n 步优势函数的加权组合

公式：

$$\hat{A}^{\pi}_{GAE}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{n=1}^{\infty} w_n \hat{A}^{\pi}_{n}(\mathbf{s}_t, \mathbf{a}_t)$$

解释：GAE 是不同 n 值下的 n 步优势函数 $\hat{A}^{\pi}_{n}(\mathbf{s}_t, \mathbf{a}_t)$的**加权组合**。这允许我们同时利用短期回报的低方差和长期回报的低偏差特性。

如何加权？（`How to weight?`）

权重衰减规律：

$$w_n \propto \lambda^{n-1}$$

解释：权重呈**指数衰减**。这意味着较短的 n 步估计（方差较小但偏差可能较高）会获得更高的权重，而较长的 n 步估计（方差较高但偏差较低）会获得较低的权重。这再次体现了偏差-方差的权衡。

3. GAE 的两种等价形式

GAE 可以被写成两种等价的、更易于计算的形式：

a) 递归形式 (Recursive Form)：

$$\hat{A}^{\pi}_{GAE}(\mathbf{s}_t, \mathbf{a}_t) = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \dots$$

其中单步 TD 误差为：

$$\delta_t = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s}_{t+1}) - \hat{V}^{\pi}_{\phi}(\mathbf{s}_t)$$

更紧凑写为：

$$\hat{A}^{\pi}_{GAE}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

4. GAE 与折扣因子的关系

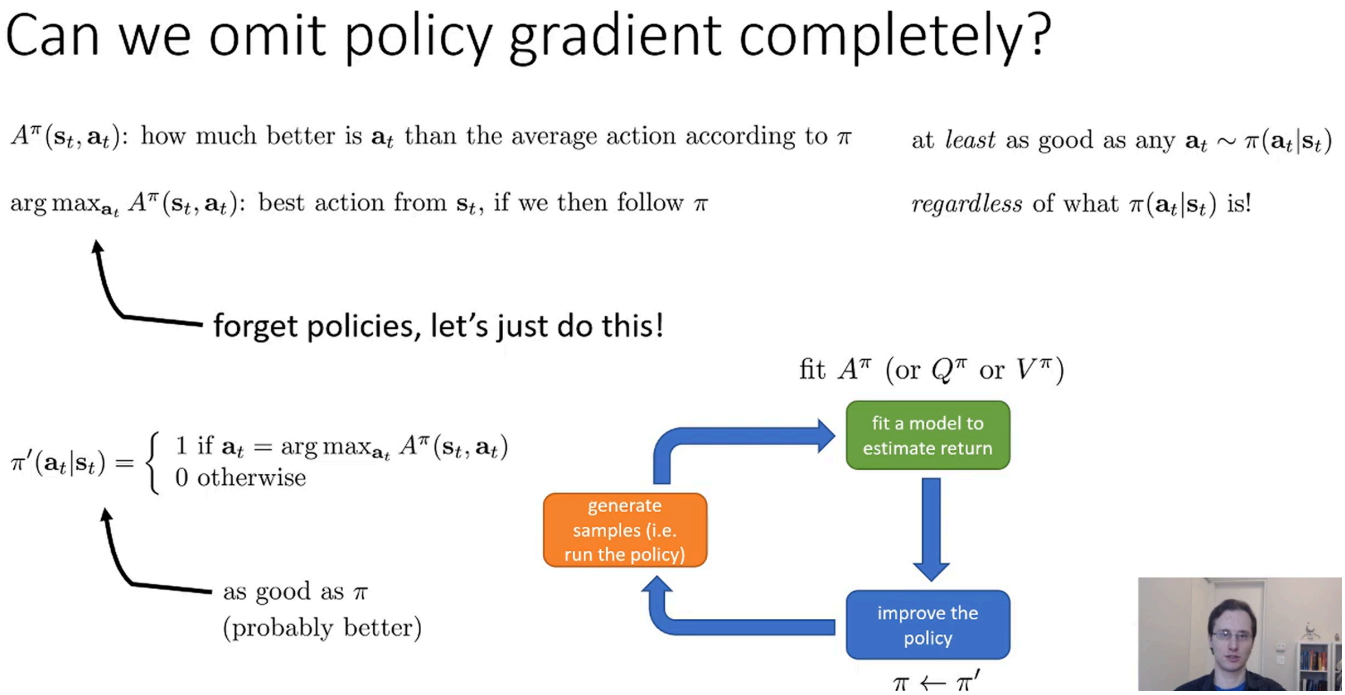
幻灯片左下方再次强调了折扣因子与方差削减的关系：

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log\ \pi_\theta(a_{i,t}|s_{i,t})\left(\sum_{t'=t}^{T}\gamma^{t'-t}r(s_{i,t'},a_{i,t'})\right)$$

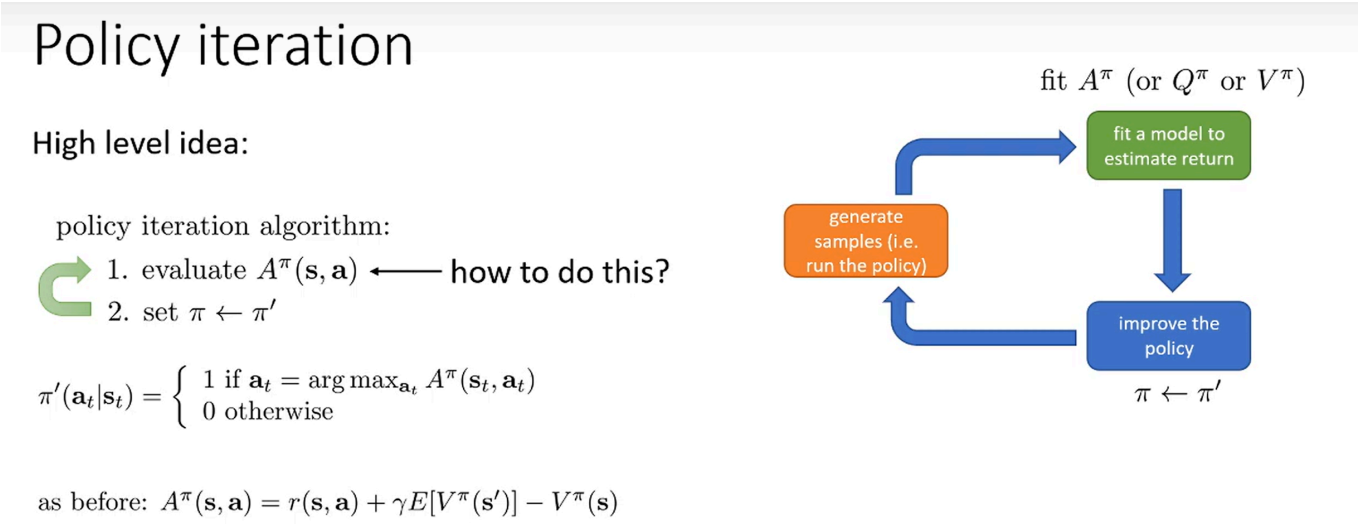
解释：折扣因子 γ 不仅用于处理无限时间步任务，它本身也具有**方差削减**的作用。因为 $\gamma < 1$使得未来遥远的、不确定的奖励对当前回报的贡献越来越小，从而减少了回报估计的波动性。GAE中的 λ 参数与 γ结合，进一步增强了这种方差削减的效果。

策略迭代 Policy-Iteration

抛弃显式地策略梯度迭代，直接迭代优势函数A，每次只需要找到最大A的动作，就能够造出最优策略



策略迭代算法!



Fitted Q-iterition与Q-Learning

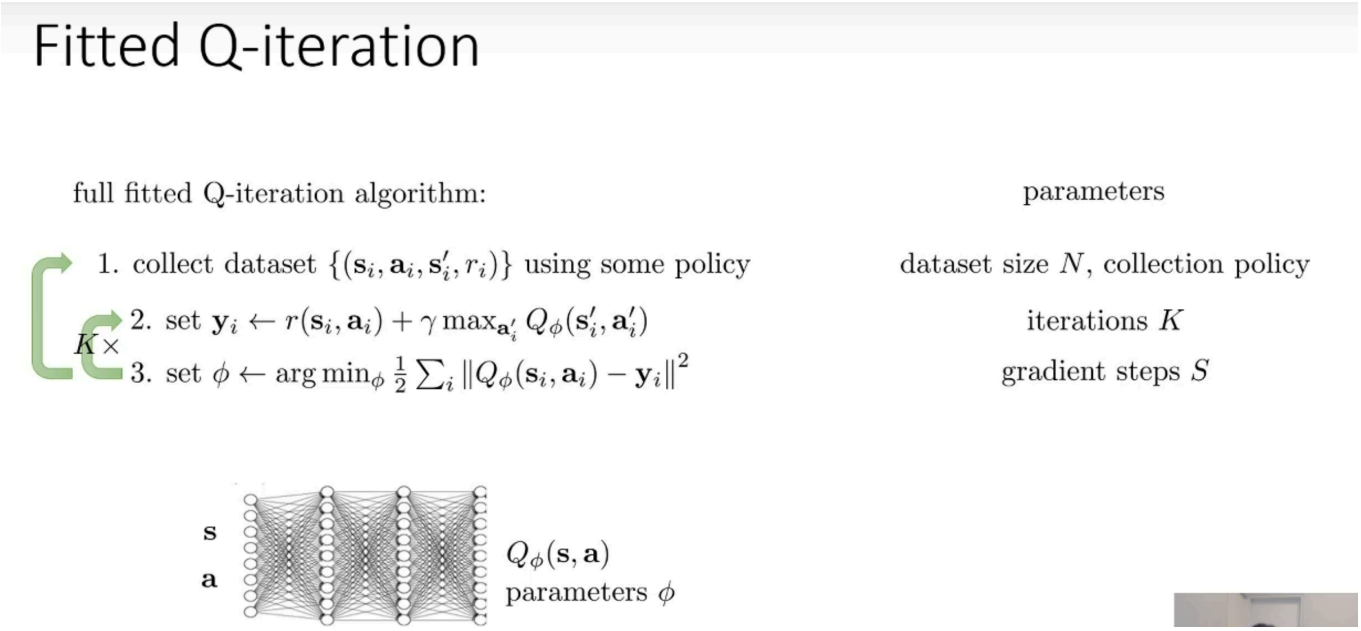
这里iterition和learning的侧重点就是：

1. iterition：off-policy、批处理
2. learning：online、单步更新（通常使用差分形式）

Fitted Q-iterition

首先核心是学习一个神经网络用来描述$Q_\phi(s,a)$ 其中ϕ 是学习到的参数

在常规MDP 价值迭代CS188/CS188笔记.md中
价值迭代方法$Q(s,a) \leftarrow E_{s' \sim p(s'|s,a)}[V^*(s')]$ ，但是在应用中我们不知道环境转移概率$p(s'|s,a)$ 和理论上的期望值，所以在拟合Q迭代中对目标项采取一定替代：
$y_i \leftarrow r(s_i,a_i) + \gamma \max_{a_i'} Q_\phi(s_i',a_i')$
$\max_{a_i'} Q_\phi(s_i',a_i')$ 是对$E[V_\phi(s_i')]$ 的近似，通过取max 技巧规避了环境转移概率的介入，直接选择最好的未来奖励赋值给y_i
使用最小二乘学习参数ϕ
这里还有一些参数K、S可以调整

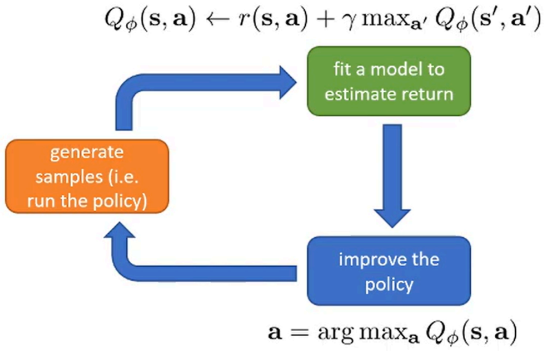


Q迭代算法核心图示：

这里隐式包含了策略-贪婪策略，选择$arg\ max_a Q_\phi(s,a)$ 的动作就是Q迭代算法中的策略

Review

- Value-based methods
  - Don't learn a policy explicitly
  - Just learn value or Q-function
- If we have value function, we have a policy
- Fitted Q-iteration



Q-Learning

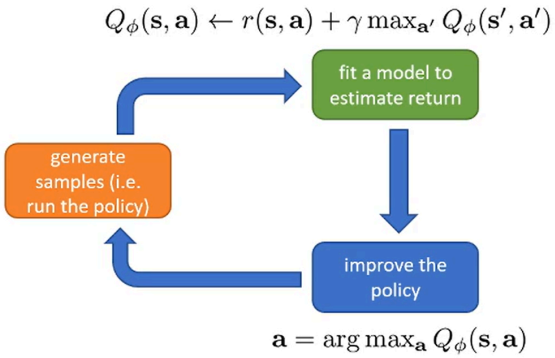
这里是单时间步采样和更新，但是是off-policy的！！！因为采样池包含了过去策略产生的动作

Online Q-learning algorithms

full fitted Q-iteration algorithm:

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

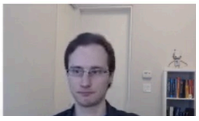
($K\times$)



off policy, so many choices here!

online Q iteration algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$



对Q-learning使用探索策略：

试想一下如果从第一步开始就每次都直接选择当下最好的动作，很可能陷入局部最优等其他情

况。引入exploration 给出探索度——也就是选择其他动作的可能性。后续会详细介绍

Exploration with Q-learning

online Q iteration algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

final policy:

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 \text{ otherwise} \end{cases}$$

why is this a bad idea for step 1?

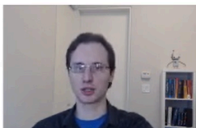
$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1-\epsilon \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ \epsilon/(|\mathcal{A}|-1) \text{ otherwise} \end{cases}$$

"epsilon-greedy"

$$\pi(\mathbf{a}_t|\mathbf{s}_t) \propto \exp(Q_\phi(\mathbf{s}_t, \mathbf{a}_t))$$

"Boltzmann exploration"

We'll discuss exploration in detail in a later lecture!



但是基础的Q-Learning也有一些问题需要改进：

1. 获取的样本是高度相关的，比如在 step t 采取的 action t，在 steo t+1 采取的action t+1很大程度可能会与上一个action相似，不符合梯度优化的数据前提（随机性）
2. 第三步更新参数ϕ 时，没有对整个式子求梯度。
   括号中一共两项：$Q_\phi(s_i, a_i)$ 和 $\mathbf{y}_i$ ，但是$\mathbf{y}_i$ 也与参数ϕ 有关，但是在这里没有使用链式法则也对$\mathbf{y}_i$ 关于ϕ 求导，也就是红色圈出的，$\mathbf{y}_i$部分没有正确参与梯度，这样会导致训练的时候，参数更新的目标一直在变，很难稳定、收敛

What's wrong?

online Q iteration algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

these are correlated!

isn't this just gradient descent? that converges, right?

Q-learning is *not* gradient descent!

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$$

no gradient through target value

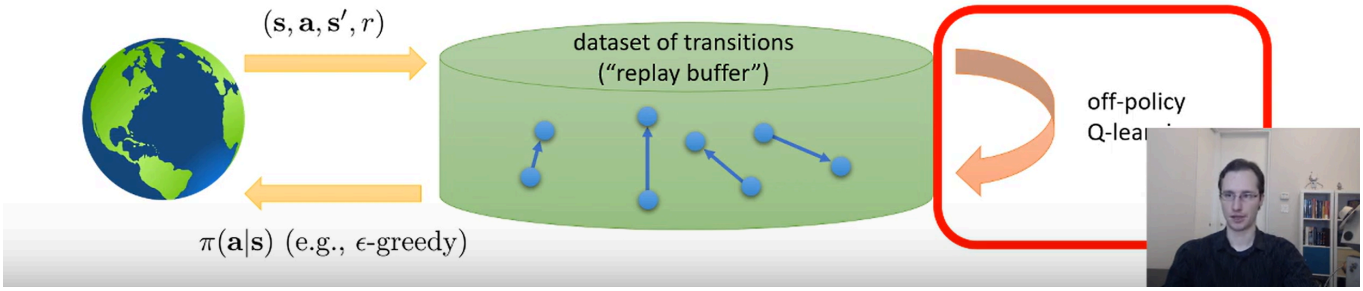
这里改进第一点，数据不是i.i.d。
使用之前介绍过的replay buffer(经验缓冲区)

Putting it together

full Q-learning with replay buffer:

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$
2. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$ ($K\times$)
3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

K = 1 is common, though larger K more efficient



这里改进第二点，目标($\mathbf{y}_i$)更新频率快
使用target network
通过构造外层大循环和引入一个新的神经网络来保存参数，减缓目标(targets)更新的频率

Q-Learning with target networks

Q-learning with replay buffer and target network:

1. save target network parameters: $\phi' \leftarrow \phi$ ($N\times$)
2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$
3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$ ($K\times$)
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

targets don't change in inner loop!

supervised regression

DQN 融合上述Q-Learning的所有技巧得出的经典算法

上部算法为通用结构
取$K = 1$ 即可得到底部算法

“Classic” deep Q-learning algorithm (DQN)

Q-learning with replay buffer and target network:

1. save target network parameters: $\phi' \leftarrow \phi$
2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$ ($N\times$)
3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$ ($K\times$)
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

“classic” deep Q-learning algorithm:

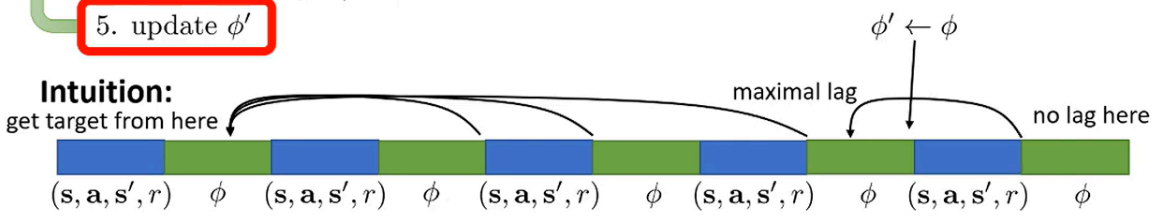
1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. update ϕ': copy ϕ every N steps

但是这里是固定的N steps才更新ϕ' 所以对于第N步和第N+1步的ϕ来说，目标变化仍然很大，可以引入动态的N steps来使得每一步更新都是变化较小的目标值，如下所示

Alternative target network

“classic” deep Q-learning algorithm:

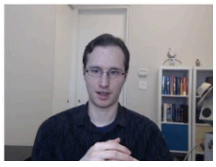
1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. update ϕ'



Feels weirdly uneven, can we always have the same lag?

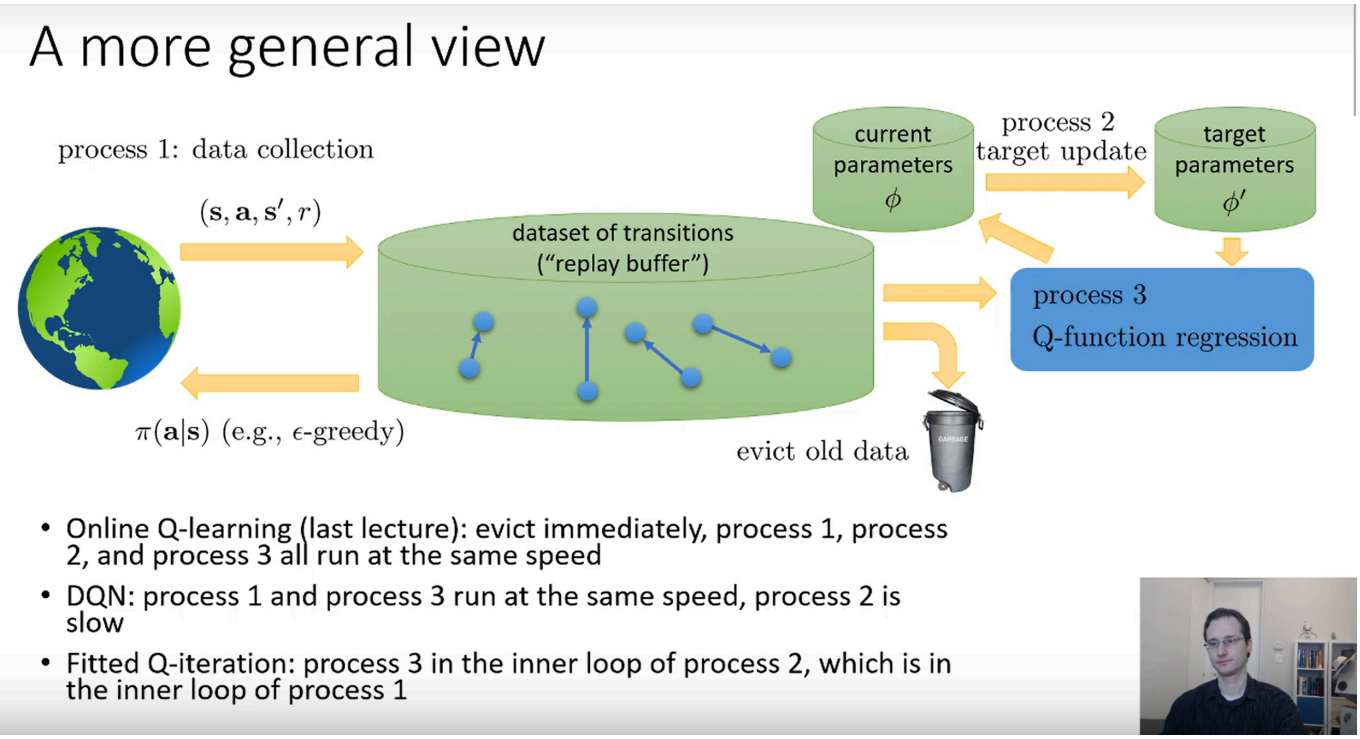
Popular alternative (similar to Polyak averaging):

5. update ϕ': $\phi' \leftarrow \tau\phi' + (1-\tau)\phi$ $\tau = 0.999$ works well



整体架构

不同的Q-Learning或者iterition算法只是调整这里不同过程发生的速率，实际上，调整不同过程的速率帮助神经网络或者对函数的估计收敛！



实践改进：Double Q-Learning and Multi-Step retruns

Overestimation in Q-learning

实践中发现：估计的Q函数值高于实际获得的最终总回报，解决这个问题，引入Double Q-Learning

Double Q-learning

Q学习的目标值（Target Value）的计算方式导致了过高估计的问题。
标准 Q-learning 的目标值 y_j 定义为：

$$y_j = r_j + \gamma \max_{a'_j} Q_{\phi'}(s'_j, a'_j)$$

拆分成两个部分：

1. 动作选择部分：$a^*_j = arg\max_{a'_j} Q_{\phi'}(s'_j, a'_j)$
2. 价值评估部分：$Q_{\phi'}(s'_j, a^*_j) = Q_{\phi'}(s'_j, \max_{a'_j} Q_{\phi'}(s'_j, a'_j))$

   此时这两个部分都是由ϕ'来进行的

数学解释：期望的最大值与最大值的期望

设 X_1 和 X_2 是两个随机变量。数学上有一个重要的不等式：

$$E[\max(X_1, X_2)] \geq \max(E[X_1], E[X_2])$$

这表示最大值的期望总是大于或等于期望的最大值。
左侧表示现在的y_j，先对动作选择取 max 再进行监督学习(对应求期望操作) **这是大于** 右侧先对各种动作的价值评估取期望后再求 max 的（也就是带有误差的真实Q函数值）

在 Q 学习中，$Q_{\phi'}(s', a')$ 是对真实 Q 值的估计，本身存在噪声（不完美）。当我们对这些带有噪声的 Q 值进行 max 操作时，会倾向于选择那些被**随机高估**的动作。
此时动作选择部分$\max_{a'_j} Q_{\phi'}(s'_j, a'_j)$已经比较大

注意！这是在参数ϕ'下选择的，也就是说ϕ'有可能会错误地将Q函数值估计偏大，那么，对动作选择部分选出来的动作a_j^*再进行价值评估$Q_{\phi'}(s'_j, a_j^*) = Q_{\phi'}(s'_j, \max_{a'_j} Q_{\phi'}(s'_j, a'_j))$，$\phi'$参数更有可能将Q函数估计偏大，在学习$\phi'$参数时，$y_j$由于错误的估计偏大，导致学习到的$Q_{\phi'}$也偏大，这样恶性循环导致了最终的overestimate！！！
所以
避免使用同一个 Q 函数来选择动作和评估其价值。

Double Q-learning

$E[\max(X_1, X_2)] \geq \max(E[X_1], E[X_2])$

note that $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}') = Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}'))$

value *also* comes from $Q_{\phi'}$ action selected according to $Q_{\phi'}$

if the noise in these is decorrelated, the problem goes away!

idea: don't use the same network to choose the action and evaluate value!

"double" Q-learning: use two networks:

$$Q_{\phi_A}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_B}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi_A}(\mathbf{s}', \mathbf{a}'))$$

$$Q_{\phi_B}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_A}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi_B}(\mathbf{s}', \mathbf{a}'))$$

if the two Q's are noisy in *different* ways, there is no problem

在具体实践中：

Double Q-learning in practice

where to get two Q-functions?

just use the current and target networks!

standard Q-learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}'))$

double Q-learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi}(\mathbf{s}', \mathbf{a}'))$

just use current network (not target network) to evaluate action

still use target network to evaluate value!

Multi-Step returns

原始目标价值$y_{j,t}$有两个问题：

1. 学习前期，Q函数很差，导致y是由r主要构成的，优点：unbiased，更真实；缺点：方差大
2. 学习后期，Q函数占比大，优点：低方差；缺点：有误差，奖励信号传播慢
   引入n -step奖励，跟之前一样，结合二者优点

Multi-step returns

Q-learning target: $y_{j,t} = r_{j,t} + \gamma \max_{\mathbf{a}_{j,t+1}} Q_{\phi'}(\mathbf{s}_{j,t+1}, \mathbf{a}_{j,t+1})$

these are the only values that matter if $Q_{\phi'}$ is bad!

these values are important if $Q_{\phi'}$ is good

where does the signal come from?

Q-learning does this: max bias, min variance

remember this?

Actor-critic: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\left(r(\mathbf{s}_{i,t},\mathbf{a}_{i,t}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}_{i,t+1}) - \hat{V}^\pi_\phi(\mathbf{s}_{i,t})\right)$

+ lower variance (due to critic)
- not unbiased (if the critic is not perfect)

Policy gradient: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\left(\left(\sum_{t'=t}^{T}\gamma^{t'-t} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right) - b\right)$

+ no bias
- higher variance (because single-sample estimate)

can we construct multi-step targets, like in actor-critic?

$y_{j,t} = \sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{j,t'} + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi'}(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N})$

N-step return estimator

但是！因为涉及到多个时间步的奖励r，导致这种方法是on-policy的，可以引入一些解决手段

(重要性采样等)

Q-learning with N-step returns

$$y_{j,t} = \frac{\sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{j,t'} + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi'}(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N})}{\text{this is supposed to estimate } Q^\pi(\mathbf{s}_{j,t}, \mathbf{a}_{j,t}) \text{ for } \pi}$$

+ less biased target values when Q-values are inaccurate

+ typically faster learning, especially early on

- only actually correct when learning on-policy

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 \text{ otherwise} \end{cases}$$

why?

we need transitions $\mathbf{s}_{j,t'}, \mathbf{a}_{j,t'}, \mathbf{s}_{j,t'+1}$ to come from π for $t' - t < N - 1$

(not an issue when $N = 1$)

how to fix?

- ignore the problem
  - often works very well
- cut the trace – dynamically choose N to get only on-policy data
  - works well when data mostly on-policy, and action space is small
- importance sampling

For more details, see: "Safe and efficient off-policy reinforcement learning." Munos et al. '16

Q-learning with continuous actions 处理连续动作的Q-Learning

连续动作空间中，对动作取max操作有很大困难，可以使用不同的解决办法

Q-learning with continuous actions

What's the problem with continuous actions?

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 \text{ otherwise} \end{cases}$$

this max

$$\text{target value } y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$$

this max

particularly problematic (inner loop of training)

How do we perform the max?

SAC

见hw3

DDPG

通过再学习一个网络来拟合取最大值这个操作

Option 3: learn an approximate maximizer

DDPG (Lillicrap et al., ICLR 2016)

"deterministic" actor-critic
(really approximate Q-learning)

$\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a}) = Q_\phi(\mathbf{s}, \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a}))$

idea: train another network $\mu_\theta(\mathbf{s})$ such that $\mu_\theta(\mathbf{s}) \approx \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$

how? just solve $\theta \leftarrow \arg\max_\theta Q_\phi(\mathbf{s}, \mu_\theta(\mathbf{s}))$ $\quad \frac{dQ_\phi}{d\theta} = \frac{d\mathbf{a}}{d\theta}\frac{dQ_\phi}{d\mathbf{a}}$

new target $y_j = r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \mu_\theta(\mathbf{s}'_j)) \approx r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j))$

这里更新参数θ是链式法则推导出的：

$$\nabla_\theta Q_\phi(s_i, \mu_\theta(s_i)) = \left.\frac{\partial Q_\phi(s,a)}{\partial a}\right|_{s=s_i,\, a=\mu_\theta(s_i)} \cdot \left.\frac{\partial \mu_\theta(s)}{\partial \theta}\right|_{s=s_i}$$

Q-learning with continuous actions

Option 3: learn an approximate maximizer

DDPG:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \mu_{\theta'}(\mathbf{s}'_j))$ using *target* nets $Q_{\phi'}$ and $\mu_{\theta'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. $\theta \leftarrow \theta + \beta \sum_j \frac{d\mu}{d\theta}(\mathbf{s}_j)\frac{dQ_\phi}{d\mathbf{a}}(\mathbf{s}_j, \mu(\mathbf{s}_j))$
6. update ϕ' and θ' (e.g., Polyak averaging)

Q相关算法实践上tricks

Simple practical tips for Q-learning

- Q-learning takes some care to stabilize
  - Test on easy, reliable tasks first, make sure your implementation is correct

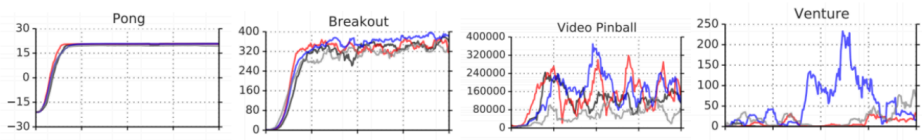


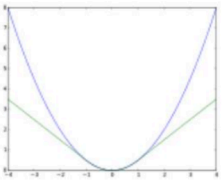
Figure: From T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized experience replay". *arXiv preprint arXiv:1511.05952* (2015), Figure 7

- Large replay buffers help improve stability
  - Looks more like fitted Q-iteration
- It takes time, be patient – might be no better than random for a while
- Start with high exploration (epsilon) and gradually reduce

Advanced tips for Q-learning

- Bellman error gradients can be big; clip gradients or use Huber loss

$$L(x) = \begin{cases} x^2/2 & \text{if } |x| \leq \delta \\ \delta|x| - \delta^2/2 & \text{otherwise} \end{cases}$$



- Double Q-learning helps *a lot* in practice, simple and no downsides
- N-step returns also help a lot, but have some downsides
- Schedule exploration (high to low) and learning rates (high to low), Adam optimizer can help too
- Run multiple random seeds, it's very inconsistent between runs

Model-Based rl

learn the transition dynamic then figure out how to choose acitons

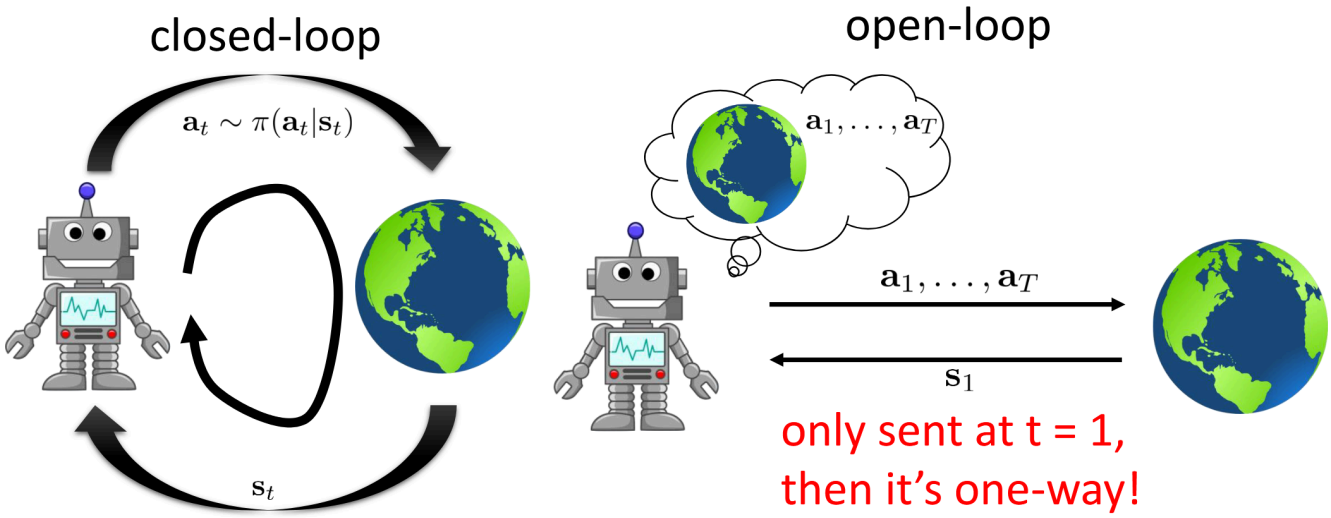
Lecture 10:已知转移概率，应该怎么做？经典算法

开环 vs. 闭环 规划

在确定性策略中，开环与闭环没有什么区别，但是在不确定性策略中，由于采取动作后环境有反馈，此时开环并不能接收到这些反馈以改进策略，所以我们在规划时尽量选择闭环规划

Aside: terminology

what is this "loop"?



Open-Loop Planning

随机射击（Random Shooting）：

随机生成一整条动作序列 a1,a2,...,aT ，评估，重复上述流程

Stochastic optimization

abstract away optimal control/planning:

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg\max_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \underbrace{J(\mathbf{a}_1, \ldots, \mathbf{a}_T)}_{\text{don't care what this is}} \qquad \mathbf{A} = \arg\max_{\mathbf{A}} J(\mathbf{A})$$

simplest method: guess & check "random shooting method"

1. pick $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from some distribution (e.g., uniform)
2. choose $\mathbf{A}_i$ based on $\arg\max_i J(\mathbf{A}_i)$

CEM：cross entropy

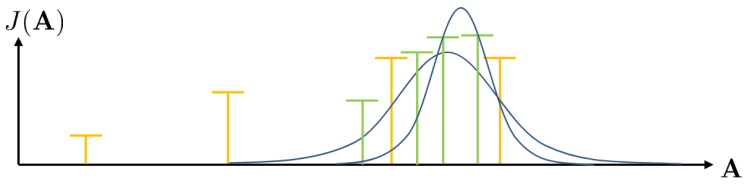
具体流程：

1. **采样**：从当前的参数化分布（如高斯分布$N(\mu, \Sigma)$）中采样 N 条完整的动作序列 $\{a_1, a_2, \cdots, a_T\}$。
2. **评估**：使用**已知的动力学模型** $s_{t+1} = f(s_t, a_t)$ ，从当前状态 s_0 开始，**模拟**每条动作序列产生的完整轨迹 $\tau = (s_0, s_1, \cdots, s_T)$
3. **计算回报**：对每条轨迹计算其**总回报** $R(\tau) = \sum_{t=1}^{T} r(st, at)$
4. **筛选**：选出回报最高的前 $k\%$ 的序列（称为“精英样本”）
5. **更新分布**：用这些精英样本重新估计高斯分布的均值 μ 和协方差 Σ （例如，取样本均值和样本协方差）。
6. **重复**：回到步骤1，用更新后的分布进行下一轮采样。

Cross-entropy method (CEM)

1. pick $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from some distribution (e.g., uniform)
2. choose $\mathbf{A}_i$ based on $\arg\max_i J(\mathbf{A}_i)$

can we do better?



typically use Gaussian distribution

see also: CMA-ES (sort of like CEM with momentum)

cross-entropy method with continuous-valued inputs:

1. sample $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from $p(\mathbf{A})$
2. evaluate $J(\mathbf{A}_1), \ldots, J(\mathbf{A}_N)$
3. pick the *elites* $\mathbf{A}_{i_1}, \ldots, \mathbf{A}_{i_M}$ with the highest value, where $M < N$
4. refit $p(\mathbf{A})$ to the elites $\mathbf{A}_{i_1}, \ldots, \mathbf{A}_{i_M}$

但是以上两种方法都有优势与局限性：

1. 算法简单
2. 并行处理，现在硬件系统计算效率高
3. 维度必须较低，一旦维度较大，采样效率极低
4. 系统如果有噪声导致执行偏离，失效

MCTS 蒙特卡洛树搜索

适合规则明确(已知动力学模型)但状态空间巨大的场景

Discrete case: Monte Carlo tree search (MCTS)

generic MCTS sketch

1. find a leaf s_l using TreePolicy(s_1)
2. evaluate the leaf using DefaultPolicy(s_l)
3. update all values in tree between s_1 and s_l

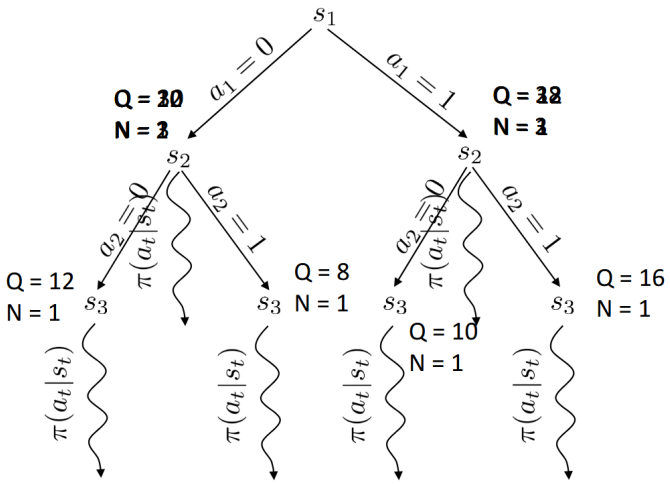
take best action from s_1

UCT TreePolicy(s_t)

if s_t not fully expanded, choose new a_t

else choose child with best Score(s_{t+1})

$$\text{Score}(s_t) = \frac{Q(s_t)}{N(s_t)} + 2C\sqrt{\frac{2\ln N(s_{t-1})}{N(s_t)}}$$



Paper reading:

Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis, Colton. (2012). A Survey of Monte Carlo Tree Search Methods.

接下来通过一系列version的model-based reinforcement learning来逐渐完善MBRL

首先，我们model-based的底层逻辑是：学习环境转移函数，进而根据已知转移函数再做其他的工作

1.MBRL-0.5

这里的step 3：plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions是指已知转移函数我们可以根据这个转移函数来进行下一步操作比如使用上面的“规划”算法如CEM、MCTS等找到较好的动作序列，也可能是用已知转移函数生成更多数据到replay buffer中增强model-free learning方法的能力(后文介绍)

总之，这里version=0.5主要确立了MBRL的整体框架

Why learn the model?

If we knew $f(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{s}_{t+1}$, we could use the tools from last week.

(or $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ in the stochastic case)

So let's learn $f(\mathbf{s}_t, \mathbf{a}_t)$ from data, and *then* plan through it!

model-based reinforcement learning version 0.5:

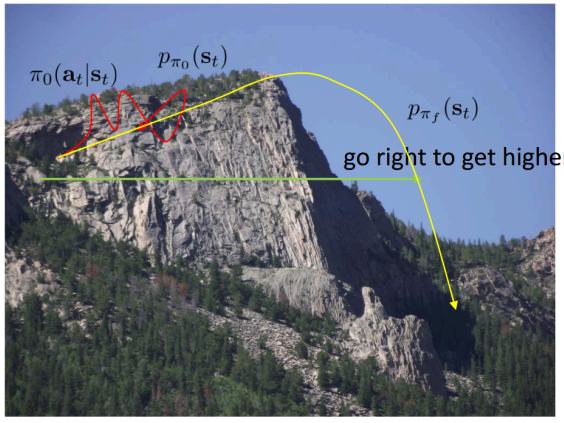
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions

动态模型 $f(\mathbf{s}, \mathbf{a})$ 的目标是预测状态转移 $s' = f(s, a)$ 。
它是在基础策略 π_0 的数据分布 $p_{\pi_0}(s_t)$ 上训练的，我们使用动态模型$f(s, a) 进行规划并优化策略时，得到的优化后策略 $\pi_f(a_t|s_t)$ 的数据分布为 $p_{\pi_f}(s_t)$。

但是这里有一些问题：

1. 初始策略 π_0无法探索环境的所有状态空间，导致初始数据分布 $p_{\pi_0}(s_t)$ 有限且不完整
2. 而优化后的策略 π_f行为与 π_0不同，导致 $p_{\pi_f}(s_t) \neq p_{\pi_0}(s_t)$ ，这种分布偏移导致 f 预测不可靠，因为f 是在 π_0下训练的，在 π_f 下表现不佳

Does it work?



No!

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions

$$p_{\pi_f}(\mathbf{s}_t) \neq p_{\pi_0}(\mathbf{s}_t)$$

- Distribution mismatch problem becomes exacerbated as we use more expressive model classes

比如这里，初始策略是在左侧山峰中寻找最高处，f 在左侧进行训练，可能会认为越向右越高，但是和实际不同导致预测错误。因为实际上不是越向右越高，也就是 $p_{\pi_f}(s_t) \neq p_{\pi_0}(s_t)$

2.MBRL-1.0

很自然想到，在策略$\pi_f(s_t)$执行过程中将新的数据分布加载进训练 f 的数据集中

Can we do better?

can we make $p_{\pi_0}(\mathbf{s}_t) = p_{\pi_f}(\mathbf{s}_t)$?

where have we seen that before? need to collect data from $p_{\pi_f}(\mathbf{s}_t)$

model-based reinforcement learning version 1.0:

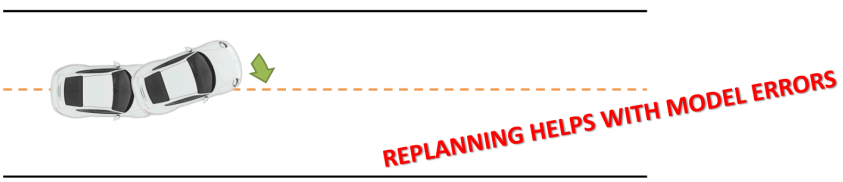
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute those actions and add the resulting data $\{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_j\}$ to $\mathcal{D}$

这里是f 直接规划出整个动作序列，应用后再添加新数据到D 中，可以用重规划改进

3.MBRL-1.5

由于直接规划出整个动作序列有犯错的概率，所以可以每N steps后重新进行规划

Can we do better?



model-based reinforcement learning version 1.5:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute the first planned action, observe resulting state $\mathbf{s}'$ (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset $\mathcal{D}$

every N steps

This will be on HW4!

但是，纯MBRL有一个致命的问题：动态模型(学习转移函数)容易过拟合！
我们怎么避免模型的过度自信？采用不确定性感知：Uncertainty-Aware RL
理论基础：区分两种不确定性：

1. **偶然不确定性 (Aleatoric Uncertainty)**：环境固有的、无法消除的噪声（如传感器噪声）。
2. **认知不确定性 (Epistemic Uncertainty)**：对模型本身的不确定性，源于数据不足

解决方法：

1. use output entropy
2. estimate model uncertainty
    1. 使用**Bootstrap Ensembles**：训练多个（如5-10个）独立的动态模型。**认知不确定性**由这些模型预测结果的**方差**来衡量。模型们“意见分歧”越大，不确定性越高。讲义指出，通过随机初始化和SGD的随机性，通常无需显式重采样数据集。
    2. 使用**贝叶斯神经网络 (BNNs)**：将模型权重视为随机变量，通过变分推断等方法学习权重的后验分布。预测时对所有可能的权重进行积分，自然地包含了认知不确定性。笔记见下

如果使用Bootatrap Ensembles方法，在规划plan环节，可以计算其在所有模型上的期望奖励，并引入 风险敏感的目标等

How to plan with uncertainty

Before: $J(\mathbf{a}_1, \ldots, \mathbf{a}_H) = \sum_{t=1}^{H} r(\mathbf{s}_t, \mathbf{a}_t)$, where $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$

Now: $J(\mathbf{a}_1, \ldots, \mathbf{a}_H) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{H} r(\mathbf{s}_{t,i}, \mathbf{a}_t)$, where $\mathbf{s}_{t+1,i} = f_i(\mathbf{s}_{t,i}, \mathbf{a}_t)$

distribution over deterministic models

In general, for candidate action sequence $\mathbf{a}_1, \ldots, \mathbf{a}_H$:

Step 1: sample $\theta \sim p(\theta|\mathcal{D})$

Step 2: at each time step t, sample $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)$

Step 3: calculate $R = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

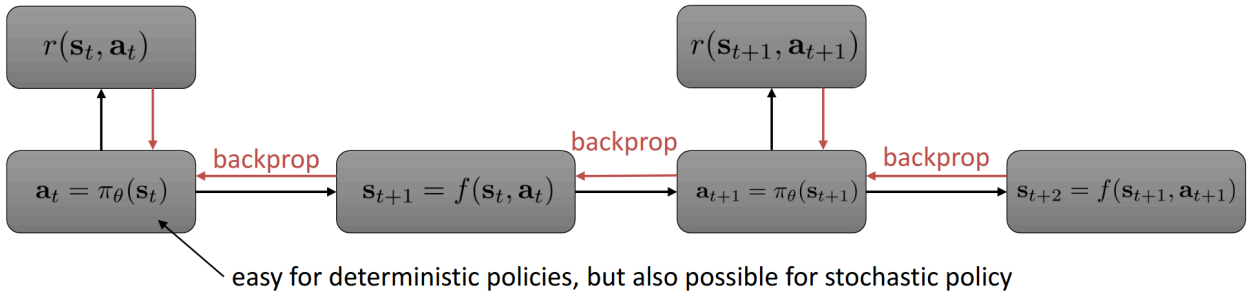
Step 4: repeat steps 1 to 3 and accumulate the average reward

Other options: moment matching, more complex posterior estimation with BNNs, etc.

4.MBRL-2.0

如果我们不使用学习f之后使用规划(plan)算法来产生动作，而是直接创建一个新的策略 π_θ 来产生action呢?

Backpropagate directly into the policy?



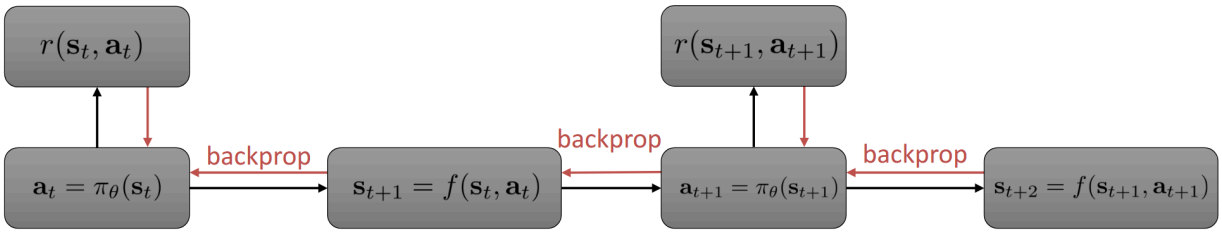
model-based reinforcement learning version 2.0:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. backpropagate through $f(\mathbf{s}, \mathbf{a})$ into the policy to optimize $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
4. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to $\mathcal{D}$

但是就像时序相关的反向传播问题一样，这里s_{t+1}由策略 π_θ和 a_t组成，而 a_t又与策略 π_θ 组成，

所以反向传播时累积到初始状态时的策略梯度大，更新效果差

What's the problem with backprop into policy?



- Similar parameter sensitivity problems as shooting methods
  - But no longer have convenient second order LQR-like method, because policy parameters couple all the time steps, so no dynamic programming
- Similar problems to training long RNNs with BPTT
  - Vanishing and exploding gradients
  - Unlike LSTM, we can't just "choose" a simple dynamics, dynamics are chosen by nature

所以我们使用Model-Based RL来产生更多数据训练Model-Free RL

两种梯度计算方式：

1. 策略梯度Policy Gradient：
   ∇_θJ(θ)≈N∑_{i=1}^N∑_{t=1}^T∇_θlog π_θ(a_{i,t}|s_{i,t})\hat Q_{i,t}^π
2. 反向传播Backprop Gradient(路径传播 pathwise gradient)：

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \frac{d\mathbf{a}_t}{d\theta} \frac{d\mathbf{s}_{t+1}}{d\mathbf{a}_t} \left(\sum_{t'=t+1}^{T} \frac{dr_{t'}}{d\mathbf{s}_{t'}} \left(\prod_{t''=t+2}^{t'} \frac{d\mathbf{s}_{t''}}{d\mathbf{a}_{t''-1}} \frac{d\mathbf{a}_{t''-1}}{d\mathbf{s}_{t''-1}} + \frac{d\mathbf{s}_{t''}}{d\mathbf{s}_{t''-1}} \right) \right)$$

其中反向传播中有连续Jacobi矩阵乘积，容易梯度消失或者梯度爆炸，接下来都是用策略梯度方法

5.MBRL-2.5

Model-based RL via policy gradient

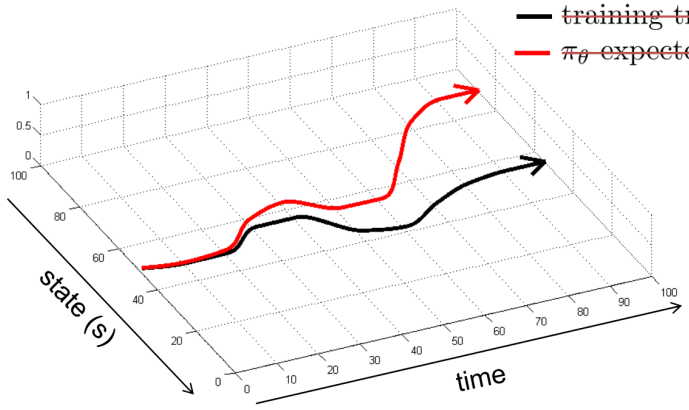
model-based reinforcement learning version 2.5:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. use $f(\mathbf{s}, \mathbf{a})$ to generate trajectories $\{\tau_i\}$ with policy $\pi_\theta(\mathbf{a}|\mathbf{s})$
4. use $\{\tau_i\}$ to improve $\pi_\theta(\mathbf{a}|\mathbf{s})$ via policy gradient
5. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to $\mathcal{D}$

What's a potential **problem** with this approach?

这个方法引入了训练策略π_θ部分，但是就像模仿学习一样会有问题，即

The curse of long model-based rollouts



How quickly does error accumulate?

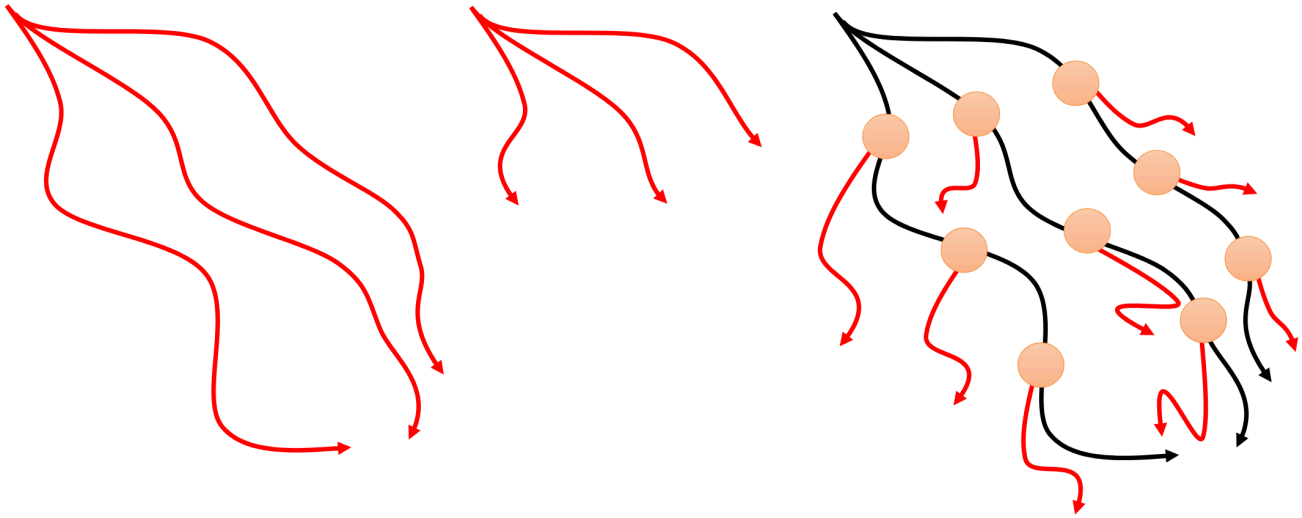
$$\mathcal{O}(\epsilon T^2)$$

rollouts太长

6.MBRL-3.0

怎么获得短rollouts?

How to get away with **short** rollouts?



- huge accumulating error

+ much lower error
- never see later time steps

+ much lower error
+ see all time steps
- wrong state distribution

实践证明是第三种最好:

Model-based RL with **short** rollouts

model-based reinforcement learning version 3.0:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. pick states $\mathbf{s}_i$ from $\mathcal{D}$, use $f(\mathbf{s}, \mathbf{a})$ to make *short* rollouts from them
4. use *both* real and model data to improve $\pi_\theta(\mathbf{a}|\mathbf{s})$ with *off-policy RL*
5. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to $\mathcal{D}$

将step 4中的off-policy RL 换成Q-Learning:
这里的step1-7都是在一次和环境交互的step中进行的，也就是说，动力学模型model $\hat{p}(s'|s,a)$ 每次和环境交互时就更新一次

Model-free optimization with a model

Dyna

online Q-learning algorithm that performs model-free RL with a model

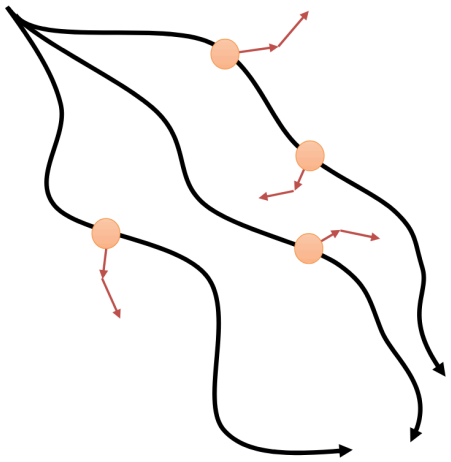
1. given state s, pick action a using exploration policy
2. observe s' and r, to get transition (s, a, s', r)
3. update model $\hat{p}(s'|s, a)$ and $\hat{r}(s, a)$ using (s, a, s')
4. Q-update: $Q(s, a) \leftarrow Q(s, a) + \alpha E_{s',r}[r + \max_{a'} Q(s', a') - Q(s, a)]$
5. repeat K times:
   6. sample $(s, a) \sim \mathcal{B}$ from buffer of past states and actions
   7. Q-update: $Q(s, a) \leftarrow Q(s, a) + \alpha E_{s',r}[r + \max_{a'} Q(s', a') - Q(s, a)]$

Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming.

整体架构

General "Dyna-style" model-based RL recipe

1. collect some data, consisting of transitions (s, a, s', r)
2. learn model $\hat{p}(s'|s, a)$ (and optionally, $\hat{r}(s, a)$)
3. repeat K times:
   4. sample $s \sim \mathcal{B}$ from buffer
   5. choose action a (from $\mathcal{B}$, from π, or random)
   6. simulate $s' \sim \hat{p}(s'|s, a)$ (and $r = \hat{r}(s, a)$)
   7. train on (s, a, s', r) with model-free RL
   8. (optional) take N more model-based steps



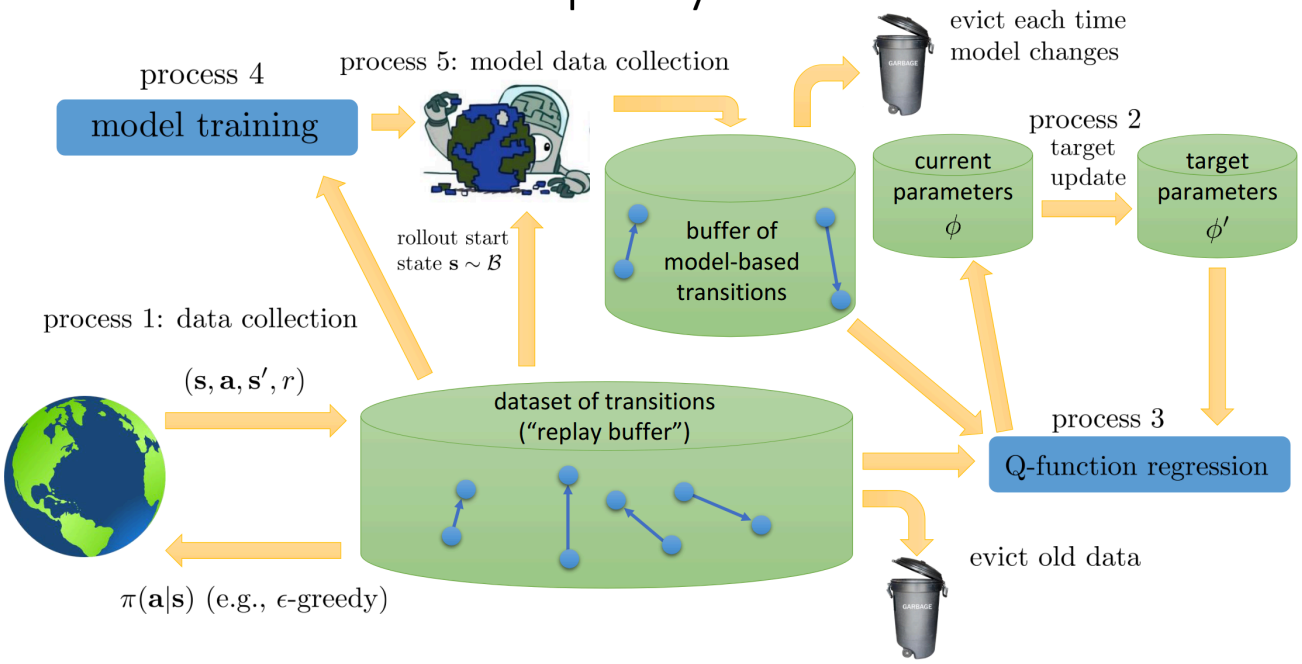
+ only requires short (as few as one step) rollouts from model

+ still sees diverse states

Model-accelerated off-policy RL

基于 Model-Based RL 生成更多数据来加速 off-policy RL学习的方法：

Model-accelerated off-policy RL



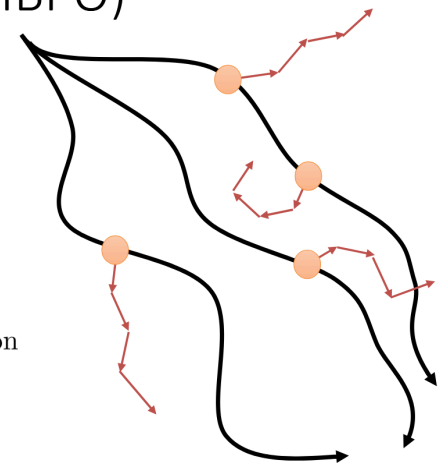
除了上述算法，也有很多衍生算法：

Model-Based Acceleration (MBA)
Model-Based Value Expansion (MVE)
Model-Based Policy Optimization (MBPO)

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. use $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j\}$ to update model $\hat{p}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$
4. sample $\{\mathbf{s}_j\}$ from $\mathcal{B}$
5. for each $\mathbf{s}_j$, perform model-based rollout with $\mathbf{a} = \pi(\mathbf{s})$
6. use all transitions $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$ along rollout to update Q-function

+ why is this a *good* idea?

- why is this a *bad* idea?



Gu et al. Continuous deep Q-learning with model-based acceleration. '16
Feinberg et al. Model-based value expansion. '18
Janner et al. When to trust your model: model-based policy optimization. '19

Successor Representations 后继表示

引入：评估一个策略的好坏，即计算： $J(\pi) = E_{s \sim p(s_1)}[V^{\pi}(s_1)]$

$$V^{\pi}(s_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} E_{p(s_{t'}|s_t)} E_{\mathbf{a}_{t'} \sim \pi(\mathbf{a}_{t'}|\mathbf{s}_{t'})}[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})]$$

为了后续推导方便，可以将基于 s 与 a 的reward简记为： $E_{\mathbf{a}_{t'}\sim\pi(\mathbf{a}_{t'}|\mathbf{s}_{t'})}[r(\mathbf{s}_{t'},\mathbf{a}_{t'})]=r(\mathbf{s}_{t'})$
所以：

$V^{\pi}(s_t)=\sum_{t'=t}^{\infty}\gamma^{t'-t}E_{p(s_{t'}|s_t)}[r(\mathbf{s}_{t'})]$
将期望展开为求和形式：

$$=\sum_{t'=t}^{\infty}\gamma^{t'-t}\sum_{\mathbf{s}}p(\mathbf{s}_{t'}=\mathbf{s}|\mathbf{s}_t)r(\mathbf{s})$$

$$=\sum_{\mathbf{s}}\left(\sum_{t'=t}^{\infty}\gamma^{t'-t}p(\mathbf{s}_{t'}=\mathbf{s}|\mathbf{s}_t)\right)r(\mathbf{s})$$

这样，我们就可以通过定义一个分布来简化式子：其中前面的系数 $(1-\gamma)$ 是用来归一化的(确保分布和为1，证明：求和号内为等比数列，乘1-q即可归一)

$$p_{\pi}(\mathbf{s}_{\text{future}}=\mathbf{s}|\mathbf{s}_t)=(1-\gamma)\sum_{t'=t}^{\infty}\gamma^{t'-t}p(\mathbf{s}_{t'}=\mathbf{s}|\mathbf{s}_t)$$

$V^{\pi}(\mathbf{s}_t)=\frac{1}{1-\gamma}\sum_{\mathbf{s}}p_{\pi}(\mathbf{s}_{\text{future}}=\mathbf{s}|\mathbf{s}_t)r(\mathbf{s})$
$\mu_i^{\pi}(s_t)$ ：在策略 π 下，从当前状态 s_t 转移到特定未来状态 i 的概率
$\mu_i^{\pi}(\mathbf{s}_t)=p_{\pi}(\mathbf{s}_{\text{future}}=i|\mathbf{s}_t)$
所以我们的价值函数 V 可以用向量点积的形式表示：

$$V^{\pi}(\mathbf{s}_t)=\mu^{\pi}(\mathbf{s}_t)^T\vec{r}$$

所以接下来我们只需要更新学习——后继表示： $\mu^{\pi}(\mathbf{s}_t)$ ，与奖励无关但与策略有关
类比传统价值函数 V 的贝尔曼方程：$V^{\pi}(\mathbf{s}_t)=r(\mathbf{s}_t)+\gamma\mathbb{E}_{a_t\sim\pi(a_t|\mathbf{s}_t),\mathbf{s}_{t+1}\sim p(\mathbf{s}_{t+1}|\mathbf{s}_t,a_t)}[V^{\pi}(\mathbf{s}_{t+1})]$，也可以得出后继表示的贝尔曼方程迭代：

$$\mu_i^{\pi}(\mathbf{s}_t)=(1-\gamma)\delta(\mathbf{s}_t=i)+\gamma\mathbb{E}_{a_t\sim\pi(a_t|\mathbf{s}_t),\mathbf{s}_{t+1}\sim p(\mathbf{s}_{t+1}|\mathbf{s}_t,a_t)}[\mu_i^{\pi}(\mathbf{s}_{t+1})]$$

其中

$$\delta(\mathbf{s}_t=i)=\begin{cases}1 & \text{if } \mathbf{s}_t=i\\ 0 & \text{otherwise}\end{cases}$$

Successor representations

$$\mu_i^\pi(\mathbf{s}_t) = (1-\gamma)\sum_{t'=t}^{\infty}\gamma^{t'-t}p(\mathbf{s}_{t'}=i|\mathbf{s}_t)$$

$$= \underbrace{(1-\gamma)\delta(\mathbf{s}_t=i) + \gamma E_{\mathbf{a}_t\sim\pi(\mathbf{a}_t|\mathbf{s}_t),\mathbf{s}_{t+1}\sim p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}[\mu_i^\pi(\mathbf{s}_{t+1})]}$$

like a Bellman backup with "reward" $r(\mathbf{s}_t) = (1-\gamma)\delta(\mathbf{s}_t=i)$

in practice, we can use vectorized backups for all i at once

A few issues...

- Not clear if learning successor representation is easier than model-free RL
- How to scale to large state spaces?
- How to extend to continuous state spaces?

所以下面来讨论如果有一些特征 features 来描述不同的state时，可以对$V^\pi(\mathbf{s}_t) = \mu^\pi(\mathbf{s}_t)^T\vec{r}$进行简化：

假设对于不同的状态 s 我们有一些特征： $\phi_j(\mathbf{s})$ ，与后继表示 μ_s^π 加权相乘后得到 $\psi_j^\pi(\mathbf{s}_t)$

$$\psi_j^\pi(\mathbf{s}_t) = \sum_{\mathbf{s}}\mu_{\mathbf{s}}^\pi(\mathbf{s}_t)\phi_j(\mathbf{s})$$

写成向量形式： $\psi_j^\pi(\mathbf{s}_t) = \mu^\pi(\mathbf{s}_t)^T\vec{\phi}_j$

现在假设奖励函数 r 对于不同的特征features有一个权重向量 $\mathbf{w}$

$$\text{if } r(\mathbf{s}) = \sum_j \phi_j(\mathbf{s})w_j = \phi(\mathbf{s})^T\mathbf{w}$$

$$\text{then } V^\pi(\mathbf{s}_t) = \psi^\pi(\mathbf{s}_t)^T\mathbf{w}$$

$$= \sum_j \psi_j^\pi(\mathbf{s}_t)w_j$$

$$= \sum_j \mu^\pi(\mathbf{s}_T)^T\vec{\phi}_j\mathbf{w}$$

$$= \mu^\pi(\mathbf{s}_T)^T\sum_j\vec{\phi}_j\mathbf{w} = \mu^\pi(\mathbf{s}_t)^T\vec{r}$$

这样发现价值函数最终和奖励函数可以共用一套权重 $\mathbf{w}$ ！！！

后继特征也有类似的贝尔曼更新：

Successor features

$$\mu_i^\pi(\mathbf{s}_t) = (1-\gamma)\delta(\mathbf{s}_t = i) + \gamma E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t), \mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}[\mu_i^\pi(\mathbf{s}_{t+1})]$$

$$\psi_j^\pi(\mathbf{s}_t) = \phi_j(\mathbf{s}_t) + \gamma E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t), \mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}[\psi_j^\pi(\mathbf{s}_{t+1})]$$

special case with
$\phi_i(\mathbf{s}_t) = (1-\gamma)\delta(\mathbf{s}_t = i)$

can also construct a "Q-function-like" version:

$$\psi_j^\pi(\mathbf{s}_t, \mathbf{a}_t) = \phi_j(\mathbf{s}_t) + \gamma E_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t), \mathbf{a}_{t+1} \sim \pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})}[\psi_j^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})]$$

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) \approx \psi^\pi(\mathbf{s}_t, \mathbf{a}_t)^T \mathbf{w} \qquad \text{when } r(\mathbf{s}_t) \approx \phi(\mathbf{s}_t)^T \mathbf{w}$$

接下来就可以利用后继特征来估计 Q 值和学习策略了
这里是因为价值函数和奖励函数可以共用一套权重 $\mathbf{w}$
所以后续可以先更新学习 ψ，再和 r 监督学习权重 $\mathbf{w}$，最后给出策略

Using successor features

Idea 1: recover a Q-function very quickly

1. Train $\psi^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ (via Bellman backups)
2. Get some reward samples $\{\mathbf{s}_i, r_i\}$
3. Get $\mathbf{w} \leftarrow \arg\min_{\mathbf{w}} \sum_i ||\phi(\mathbf{s}_i)^T \mathbf{w} - r_i||^2$
4. Recover $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) \approx \psi^{\pi}(\mathbf{s}_t, \mathbf{a}_t)^T \mathbf{w}$

Is this the **optimal** Q-function?

$$\pi'(\mathbf{s}) = \arg\max_{\mathbf{a}} \psi^{\pi}(\mathbf{s}, \mathbf{a})^T \mathbf{w}$$

Equivalent to **one step** of policy iteration

Better than nothing, but **not** optimal

Using successor features

Idea 2: recover many Q-functions

1. Train $\psi^{\pi_k}(\mathbf{s}_t, \mathbf{a}_t)$ for many policies π_k (via Bellman backups)
2. Get some reward samples $\{\mathbf{s}_i, r_i\}$
3. Get $\mathbf{w} \leftarrow \arg\min_{\mathbf{w}} \sum_i ||\phi(\mathbf{s}_i)^T \mathbf{w} - r_i||^2$
4. Recover $Q^{\pi_k}(\mathbf{s}_t, \mathbf{a}_t) \approx \psi^{\pi_k}(\mathbf{s}_t, \mathbf{a}_t)^T \mathbf{w}$ for every π_k

$$\pi'(\mathbf{s}) = \arg\max_{\mathbf{a}} \max_{k} \psi^{\pi_k}(\mathbf{s}, \mathbf{a})^T \mathbf{w}$$

Finds the highest reward policy in **each state**

Barreto et al. **Successor Features for Transfer in Reinforcement Learning**. 2016.

接下来将后继特征用于连续动作空间：
问题是： $\delta(s_t = i)$ 一直为0， 而我们还无法使用概率密度替代概率来表示后继表示 μ
通过将问题转化为一个分类器 classifier来判断 $\mathbf{s}_{\text{future}}$ 是不是从 $\mathbf{s}_t, \mathbf{a}_t$ 在策略 π 下生成出来的

Continuous successor representations

$$\mu_i^\pi(\mathbf{s}_t) = (1-\gamma)\delta(\mathbf{s}_t = i) + \gamma E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t), \mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}[\mu_i^\pi(\mathbf{s}_{t+1})]$$

always zero for any sampled state if states are continuous

Framing successor representation as *classification*:

$$p^\pi(F = 1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}}) = \frac{p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t)}{p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t) + p^\pi(\mathbf{s}_{\text{future}})}$$

binary classifier

$F = 1$ means $\mathbf{s}_{\text{future}}$ is a future state from $\mathbf{s}_t, \mathbf{a}_t$ under π

$$\mathcal{D}_+ \sim p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t) \qquad \mathcal{D}_- \sim p^\pi(\mathbf{s})$$

有关贝叶斯分类器的推导:

Continuous successor representations

$$\mathcal{D}_+ \sim p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t) \qquad \mathcal{D}_- \sim p^\pi(\mathbf{s})$$

$$p^\pi(F = 1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}}) = \frac{p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t)}{p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t) + p^\pi(\mathbf{s}_{\text{future}})}$$

$$p^\pi(F = 0|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}}) = \frac{p^\pi(\mathbf{s}_{\text{future}})}{p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t) + p^\pi(\mathbf{s}_{\text{future}})}$$

$$\frac{p^\pi(F = 1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}})}{p^\pi(F = 0|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}})} = \frac{p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t)}{p^\pi(\mathbf{s}_{\text{future}})}$$

$$\frac{p^\pi(F = 1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}})}{p^\pi(F = 0|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}})} p^\pi(\mathbf{s}_{\text{future}}) = p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t)$$

constant independent of $\mathbf{a}_t, \mathbf{s}_t$

算法实现与应用：

The C-Learning algorithm

$$\mathcal{D}_+ \sim p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t) \qquad \mathcal{D}_- \sim p^\pi(\mathbf{s})$$

$$p^\pi(F=1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}}) = \frac{p^\pi(F=1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}})}{p^\pi(F=1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}}) + p^\pi(\mathbf{s}_{\text{future}})}$$

To train:

1. Sample $\mathbf{s} \sim p^\pi(\mathbf{s})$ (e.g., run policy, sample from trajectories)
2. Sample $\mathbf{s} \sim p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t)$ (e.g., pick $\mathbf{s}_{t'}$ where $t' = t+\Delta$, $\Delta \sim \text{Geom}(\gamma)$)
3. Update $p^\pi(F=1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s})$ using SGD with cross entropy loss

This is an **on policy** algorithm

Could also derive an **off policy** algorithm

Eysenbach, Salakhutdinov, Levine. **C-Learning: Learning to Achieve Goals via Recursive Classification**. 2020.

探索与利用 Exploration and exploitation

基础概念

当奖励稀疏时，同时缺少中间奖励，传统强化学习很难知道哪些是好的动作进而完成任务

Montezuma's revenge

- Getting key = reward
- Opening door = reward
- Getting killed by skull = nothing (is it good? bad?)
- Finishing the game only weakly correlates with rewarding events
- We know what to do because we **understand** what these sprites mean!

随着问题的逐渐复杂化，问题从可被定义和处理的情景逐渐转向不可被处理的情形，但是我们可

以借鉴小规模可处理问题的思路来尝试处理大规模问题

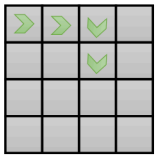
What makes an exploration problem tractable?

multi-arm bandits

contextual bandits

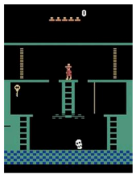
can formalize exploration as POMDP identification

policy learning is trivial even with POMDP



small, finite MDPs

can frame as Bayesian model identification, reason explicitly about value of information



large or infinite MDPs

optimal methods don't work

...but can take inspiration from optimal methods in smaller settings

use hacks

用bandits为例简要引入概念

$\mathcal{A} = \{\text{pull arm}\}$

$r(\text{pull arm}) = ?$

$\mathcal{A} = \{\text{pull}_1, \text{pull}_2, \ldots, \text{pull}_n\}$

$r(a_n) = ?$

assume $r(a_n) \sim p(r|a_n)$

unknown *per-action* reward distribution!

单臂老虎机： $\mathcal{A} = \{pullarm\}$ 这是动作空间，奖励 $r(\text{pull arm})$ 未知，可以通过探索来进行估计

多臂老虎机： $\mathcal{A} = \{pull_1, pull_2, \cdots, pull_n\}$ ，奖励 $r(a_n)$ 未知，可以通过探索先估计 $p(r|a_n)$ 再采样出 $r(a_n)$

用数学符号来定义问题：

假设 $r(a_i) \sim p_{\theta_i}(r_i)$，这里指每一个拉杆都有一个自己的奖励参数 θ_i，奖励服从这个奖励参数分布，这个奖励参数服从大的先验分布 $p(\theta)$ 。比如这里奖励参数分布是伯努利分布，同时每个 θ_i 都服从 $p(\theta)$

这就可以定义为一个POMDP问题：

其中真实状态$\mathbf{s} = [\theta_1, \ldots, \theta_n]$

由于真实状态 s 不可观测，智能体必须维护一个**信念状态**，即它对真实状态（所有 θ_i）的后验概率分布$\hat{p}(\theta_1, \ldots, \theta_n)$。这个信念状态会根据每次拉臂观察到的奖励进行更新

但是当n很大时维护信念状态很难，后续笔记会有具体算法来解决问题如UCB、Thompson Sampling

How can we define the bandit?

assume $r(a_i) \sim p_{\theta_i}(r_i)$

e.g., $p(r_i = 1) = \theta_i$ and $p(r_i = 0) = 1 - \theta_i$

$\theta_i \sim p(\theta)$, but otherwise unknown

this defines a POMDP with $\mathbf{s} = [\theta_1, \ldots, \theta_n]$

belief state is $\hat{p}(\theta_1, \ldots, \theta_n)$

- solving the POMDP yields the optimal exploration strategy
- but that's overkill: belief state is huge!
- we can do very well with much simpler strategies

how do we measure goodness of exploration algorithm?

regret: difference from optimal policy at time step T: $\mathrm{Reg}(T) = TE[r(a^\star)] - \sum_{t=1}^{T} r(a_t)$

expected reward of best action (the best we can hope for in expectation)

actual reward of action actually taken

怎么判断探索算法的好坏呢？引入一个Regret值：

$Reg(T) = TE[r(a^*)] - \sum_{t=1}^{T} r(a_t)$

Three Classes of Exploration Methods 理论说明

Optimistic exploration 乐观探索

UCB (Upper Confidence Bound)

Optimistic exploration

keep track of average reward $\hat{\mu}_a$ for each action a

exploitation: pick $a = \arg\max \hat{\mu}_a$

optimistic estimate: $a = \arg\max \hat{\mu}_a + C\sigma_a$

some sort of variance estimate

intuition: try each arm until you are *sure* it's not great

example (Auer et al. Finite-time analysis of the multiarmed bandit problem):

$$a = \arg\max \hat{\mu}_a + \sqrt{\frac{2\ln T}{N(a)}}$$

number of times we picked this action

$\mathrm{Reg}(T)$ is $O(\log T)$, provably as good as any algorithm

$$a = \arg\max_a \left(\hat{\mu}_a + \sqrt{\frac{2\ln T}{N(a)}} \right)$$

其中：

- $\hat{\mu}_a$ 是动作 a 的平均奖励估计，
- T 是总时间步数，
- $N(a)$ 是动作 a 被选择的次数。

概率匹配/后验采样（Probability Matching / Posterior Sampling）

Thompson sampling

Probability matching/posterior sampling

assume $r(a_i) \sim p_{\theta_i}(r_i)$

this defines a POMDP with $\mathbf{s} = [\theta_1, \ldots, \theta_n]$

belief state is $\hat{p}(\theta_1, \ldots, \theta_n)$

this is a *model* of our bandit

idea: sample $\theta_1, \ldots, \theta_n \sim \hat{p}(\theta_1, \ldots, \theta_n)$

pretend the model $\theta_1, \ldots, \theta_n$ is correct

take the optimal action

update the model

- This is called posterior sampling or Thompson sampling
- Harder to analyze theoretically
- Can work very well empirically

See: Chapelle & Li, "An Empirical Evaluation of Thompson Sampling."

信息增益（Information Gain）

$IG(z, y \mid a) = E_y[\mathcal{H}(\hat{p}(z)) - \mathcal{H}(\hat{p}(z) \mid y) \mid a]$。它衡量了执行动作 a 后，我们对潜在变量 z （例如，臂的真实参数 θ_a ）的不确定性（熵）平均减少了多少 。

Information gain

Bayesian experimental design:

say we want to determine some latent variable z (e.g., z might be the optimal action, or its value)

which action do we take?

let $\mathcal{H}(\hat{p}(z))$ be the current entropy of our z estimate

let $\mathcal{H}(\hat{p}(z)|y)$ be the entropy of our z estimate after observation y (e.g., y might be $r(a)$)

the lower the entropy, the more precisely we know z

$$\mathrm{IG}(z, y) = E_y[\mathcal{H}(\hat{p}(z)) - \mathcal{H}(\hat{p}(z)|y)]$$

typically depends on action, so we have $\mathrm{IG}(z, y|a)$

理论总结，接下来是实际应用到RL算法中

Recap: classes of exploration methods in deep RL

- Optimistic exploration:
  - new state = good state
  - requires estimating state visitation frequencies or novelty
  - typically realized by means of exploration bonuses
- Thompson sampling style algorithms:
  - learn distribution over Q-functions or policies
  - sample and act according to sample
- Information gain style algorithms
  - reason about information gain from visiting new states

Exploration in Deep RL 具体实践实现

1. Optimistic exploration in RL

这里的 $\mathcal{B}(N(\mathbf{s}))$是指以 $N(\mathbf{s})$ 为自变量的bonus函数
我们用 r^+ 来替换平常的 r 奖励函数

Optimistic exploration in RL

UCB: $a = \arg\max \hat{\mu}_a + \sqrt{\frac{2 \ln T}{N(a)}}$

"exploration bonus"

lots of functions work, so long as they decrease with $N(a)$

can we use this idea with MDPs?

count-based exploration: use $N(\mathbf{s}, \mathbf{a})$ or $N(\mathbf{s})$ to add *exploration bonus*

use $r^+(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \mathcal{B}(N(\mathbf{s}))$

bonus that decreases with $N(\mathbf{s})$

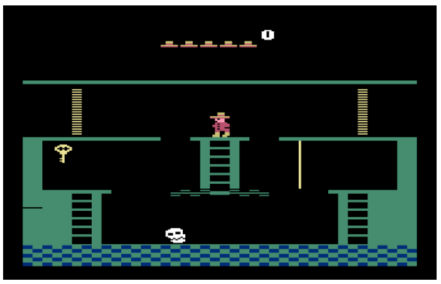
use $r^+(\mathbf{s}, \mathbf{a})$ instead of $r(\mathbf{s}, \mathbf{a})$ with any model-free algorithm

+ simple addition to any RL algorithm

- need to tune bonus weight

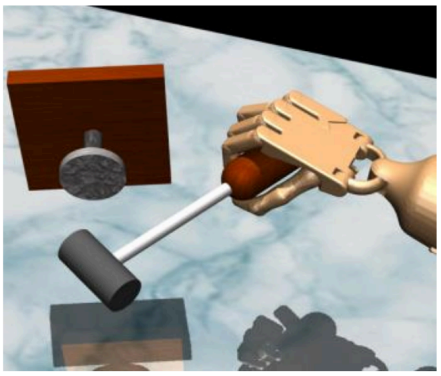
但是 $N(\mathbf{s})$是什么？在连续空间中没有两个状态是相同的！
我们引入一个密度模型来进行伪计数：

What kind of model to use?



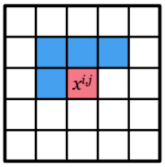
$$p_\theta(\mathbf{s})$$

need to be able to output densities, but doesn't necessarily need to produce great samples



opposite considerations from many popular generative models in the literature (e.g., GANs)

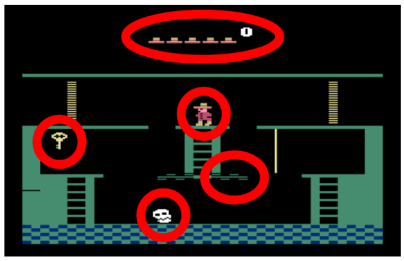
Bellemare et al.: "CTS" model: condition each pixel on its top-left neighborhood



Other models: stochastic neural networks, compression length, EX2

a density model $p_\theta(\mathbf{s}, \mathbf{a})$中 θ 是这个密度模型的参数，s 或者 (s , a)对是这个模型的输入

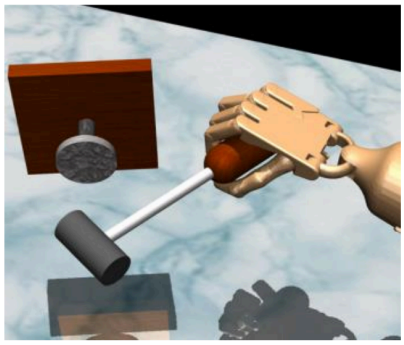
Fitting generative models



idea: fit a density model $p_\theta(\mathbf{s})$ (or $p_\theta(\mathbf{s}, \mathbf{a})$)

$p_\theta(\mathbf{s})$ might be high even for a new $\mathbf{s}$
if $\mathbf{s}$ is similar to previously seen states

can we use $p_\theta(\mathbf{s})$ to get a "pseudo-count"?



if we have small MDPs
the true probability is:

$$P(\mathbf{s}) = \frac{N(\mathbf{s})}{n}$$

count

probability/density

total states visited

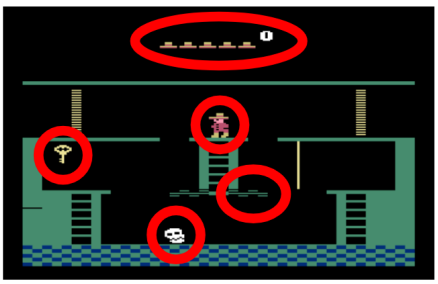
after we see $\mathbf{s}$, we have:

$$P'(\mathbf{s}) = \frac{N(\mathbf{s}) + 1}{n + 1}$$

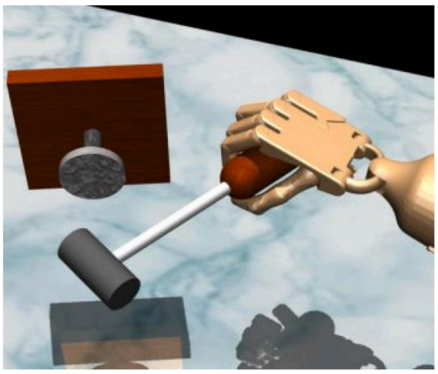
can we get $p_\theta(\mathbf{s})$ and $p_{\theta'}(\mathbf{s})$ to obey these equations?

训练这个密度模型通常使用最大似然估计的方法，我们通过两次训练，可以获得两个参数 θ 和 θ' 进而用这两个模型的估计值解出次数估计值 $\hat{N}$ 。即伪估计!

Exploring with pseudo-counts



fit model $p_\theta(\mathbf{s})$ to all states $\mathcal{D}$ seen so far
take a step i and observe $\mathbf{s}_i$
fit new model $p_{\theta'}(\mathbf{s})$ to $\mathcal{D} \cup \mathbf{s}_i$
use $p_\theta(\mathbf{s}_i)$ and $p_{\theta'}(\mathbf{s}_i)$ to estimate $\hat{N}(\mathbf{s})$
set $r_i^+ = r_i + \mathcal{B}(\hat{N}(\mathbf{s}))$ ← "pseudo-count"



how to get $\hat{N}(\mathbf{s})$? use the equations

$$p_\theta(\mathbf{s}_i) = \frac{\hat{N}(\mathbf{s}_i)}{\hat{n}} \qquad p_{\theta'}(\mathbf{s}_i) = \frac{\hat{N}(\mathbf{s}_i)+1}{\hat{n}+1}$$

two equations and two unknowns!

$$\hat{N}(\mathbf{s}_i) = \hat{n} p_\theta(\mathbf{s}_i) \qquad \hat{n} = \frac{1 - p_{\theta'}(\mathbf{s}_i)}{p_{\theta'}(\mathbf{s}_i) - p_\theta(\mathbf{s}_i)} p_\theta(\mathbf{s}_i)$$

Bellemare et al. "Unifying Count-Based Exploration…"

在获得了 $\hat{N}$ 后，我们挑选合适的bonus函数就可以利用前文所述的 r^+ 训练Deep RL了

What kind of bonus to use?

Lots of functions in the literature, inspired by optimal methods for bandits or small MDPs

UCB: $\mathcal{B}(N(\mathbf{s})) = \sqrt{\frac{2 \ln n}{N(\mathbf{s})}}$

MBIE-EB (Strehl & Littman, 2008): $\mathcal{B}(N(\mathbf{s})) = \sqrt{\frac{1}{N(\mathbf{s})}}$ ← this is the one used by Bellemare et al. '16

BEB (Kolter & Ng, 2009): $\mathcal{B}(N(\mathbf{s})) = \frac{1}{N(\mathbf{s})}$

更新颖的方法：解决连续空间的探索难题

1. EX2

Implicit density modeling with exemplar models

$p_\theta(\mathbf{s})$ need to be able to output densities, but doesn't necessarily need to produce great samples

Can we explicitly compare the new state to past states?

Intuition: the state is **novel** if it is **easy** to distinguish from all previous seen states by a classifier

for each observed state $\mathbf{s}$, fit a classifier to classify that state against all past states $\mathcal{D}$, use classifier error to obtain density

$$p_\theta(\mathbf{s}) = \frac{1 - D_{\mathbf{s}}(\mathbf{s})}{D_{\mathbf{s}}(\mathbf{s})}$$

probability that classifier assigns that $\mathbf{s}$ is "positive"
positives: $\{\mathbf{s}\}$
negatives: $\mathcal{D}$

Fu et al. "EX2: Exploration with Exemplar Models…"

Implicit density modeling with exemplar models

hang on… aren't we just checking if $\mathbf{s} = \mathbf{s}$?

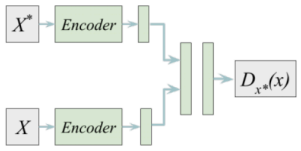
if $\mathbf{s} \in \mathcal{D}$, then the optimal $D_{\mathbf{s}}(\mathbf{s}) \neq 1$

in fact: $D^\star_{\mathbf{s}}(\mathbf{s}) = \frac{1}{1 + p(\mathbf{s})}$ ⟹ $p_\theta(\mathbf{s}) = \frac{1 - D_{\mathbf{s}}(\mathbf{s})}{D_{\mathbf{s}}(\mathbf{s})}$

in reality, each state is unique, so we *regularize* the classifier

isn't one classifier per state a bit much?

train one *amortized* model: single network that takes in exemplar as input!



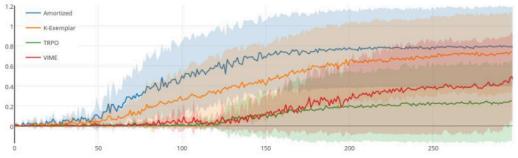


Figure 9: DoomMyWayHome+

Fu et al. "EX2: Exploration with Exemplar Models…"

2. 直接拟合神经网络

Heuristic estimation of counts via errors

$p_\theta(\mathbf{s})$

need to be able to output densities, but doesn't necessarily need to produce great samples

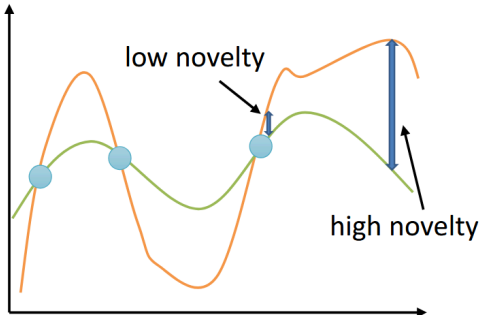
…and doesn't even need to output great densities

…just need to tell if state is **novel** or not!

let's say we have some **target** function $f^\star(\mathbf{s}, \mathbf{a})$

given our buffer $\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i)\}$, fit $\hat{f}_\theta(\mathbf{s}, \mathbf{a})$

use $\mathcal{E}(\mathbf{s}, \mathbf{a}) = ||\hat{f}_\theta(\mathbf{s}, \mathbf{a}) - f^\star(\mathbf{s}, \mathbf{a})||^2$ as bonus

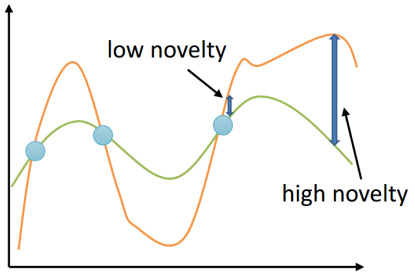


Heuristic estimation of counts via errors

let's say we have some **target** function $f^\star(\mathbf{s}, \mathbf{a})$

given our buffer $\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i)\}$, fit $\hat{f}_\theta(\mathbf{s}, \mathbf{a})$

use $\mathcal{E}(\mathbf{s}, \mathbf{a}) = ||\hat{f}_\theta(\mathbf{s}, \mathbf{a}) - f^\star(\mathbf{s}, \mathbf{a})||^2$ as bonus



what should we use for $f^\star(\mathbf{s}, \mathbf{a})$?

one common choice: set $f^\star(\mathbf{s}, \mathbf{a}) = \mathbf{s}'$ – i.e., next state prediction

- also related to information gain, which we'll discuss next time!

this will be in HW5!

even simpler: $f^\star(\mathbf{s}, \mathbf{a}) = f_\phi(\mathbf{s}, \mathbf{a})$, where ϕ is a *random* parameter vector

Burda et al. Exploration by random network distillation. 2018.

2.Posterior Sampling in Deep RL

这部分详细解释了**汤普森采样**如何从多臂老虎机扩展到深度强化学习。
核心问题: 在深度RL中，我们应该采样什么？如何表示分布？

- **采样对象**: 在多臂老虎机中采样的是奖励参数 θ_i。在MDP（马尔可夫决策过程）中，其类比是 **Q-函数（Q-function）**。因此，我们采样的是Q-函数 Q 。

- **流程**:
  1. 从 Q-函数分布p(Q) 中采样一个 Q-函数 Q 。
  2. 根据这个采样的 Q-函数在一个 episode 中行动 。
  3. 更新 Q-函数分布p(Q) 。
- **Q-learning 的异策略优势**: “由于 Q-learning 是异策略的，我们不关心收集数据时使用了哪个 Q-函数。” 这意味着无论哪个采样的 Q-函数生成了经验数据，这些数据都可以用来更新所有 Q-函数的信念分布。
  但是!
- **如何表示函数上的分布?** : 这是关键挑战。如何表示一个函数上的概率分布?
- **解决方案：Bootstrap (引导法)**
  - **思想**: 给定一个数据集 D，通过有放回地重采样（resample with replacement）N 次，得到 N 个不同的数据集 $D_1, \ldots, D_N$ 。
  - **模型训练**: 在每个数据集 D_i 上训练一个独立的模型 f_{θ_i} 。
  - **采样**: 当需要从 $p(\theta)$ 采样时，随机选择 $i \in [1, \ldots, N]$ 并使用 f_{θ_i} 。
  - **挑战**: 训练 N 个大型神经网络是很昂贵的。
  - **Bootstrapped DQN 的实现**: 为了解决这个问题，Bootstrapped DQN (Osband et al. 2016) 使用一个
    共享网络（Shared network），并在其顶部连接多个**独立的“头”（Head K）**。
    - 共享网络学习通用的特征表示。
    - 每个“头”学习一个独立的 Q-函数。
    - 训练时，每个头在原始数据的不同重采样版本上（或通过对损失函数进行遮蔽）进行更新。
    - 在探索时，随机激活一个头来生成动作。
- **为什么这种方法有效?*
  - **与随机动作的对比**: 传统的 ϵ-greedy 探索通过随机动作进行探索，可能导致智能体在环境中“振荡”或“不连贯”地移动，无法达到一个有意义的或连贯的状态 。
  - **随机 Q-函数的探索**: 通过随机采样一个 Q-函数，智能体可以在一个完整的 episode 中**坚持一个随机但内部一致的策略** 。这意味着智能体不会在每一步都随机行动，而是会根据一个固定的（尽管是随机采样的）Q-函数进行决策，从而能够更深入、更连贯地探索环境。
  - **优点**: 这种方法不改变原始奖励函数 ， 并且在许多 Atari 游戏上表现良好，特别是在《蒙特祖玛的复仇》这种稀疏奖励游戏中-

Posterior sampling in deep RL

Thompson sampling:

$\theta_1, \ldots, \theta_n \sim \hat{p}(\theta_1, \ldots, \theta_n)$

$a = \arg\max_a E_{\theta_a}[r(a)]$

What do we sample?

How do we represent the distribution?

bandit setting: $\hat{p}(\theta_1, \ldots, \theta_n)$ is distribution over *rewards*

MDP analog is the Q-function!

1. sample Q-function Q from $p(Q)$
2. act according to Q for one episode
3. update $p(Q)$

since Q-learning is off-policy, we don't care which Q-function was used to collect data

how can we represent a distribution over functions?

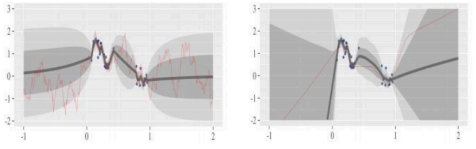
Osband et al. "Deep Exploration via Bootstrapped DQN"

Bootstrap

given a dataset $\mathcal{D}$, resample with replacement N times to get $\mathcal{D}_1, \ldots, \mathcal{D}_N$

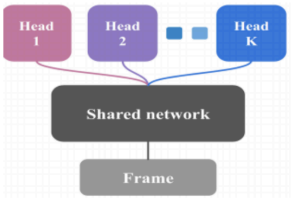
train each model f_{θ_i} on $\mathcal{D}_i$

to sample from $p(\theta)$, sample $i \in [1, \ldots, N]$ and use f_{θ_i}



(b) Gaussian process posterior (c) Bootstrapped neural nets

training N big neural nets is expensive, can we avoid it?



Osband et al. "Deep Exploration via Bootstrapped DQN"

3.Information Gain in Deep RL

Reasoning about information gain (approximately)

Generally intractable to use exactly, regardless of what is being estimated

A few approximations:

prediction gain: $\log p_{\theta'}(\mathbf{s}) - \log p_{\theta}(\mathbf{s})$ (Schmidhuber '91, Bellemare '16)

intuition: if density changed a lot, the state was novel

variational inference: (Houthooft et al. "VIME")

IG can be equivalently written as $D_{\mathrm{KL}}(p(z|y)\|p(z))$

learn about *transitions* $p_\theta(s_{t+1}|s_t, a_t)$: $z = \theta$

$y = (s_t, a_t, s_{t+1})$

$$D_{\mathrm{KL}}(p(\theta|h, s_t, a_t, s_{t+1})\|p(\theta|h))$$

model parameters for $p_\theta(s_{t+1}|s_t, a_t)$

history of all prior transitions

newly observed transition

intuition: a transition is more informative if it causes belief over θ to change

idea: use variational inference to estimate $q(\theta|\phi) \approx p(\theta|h)$

given new transition (s, a, s'), update ϕ to get ϕ'

Reasoning about information gain (approximately)

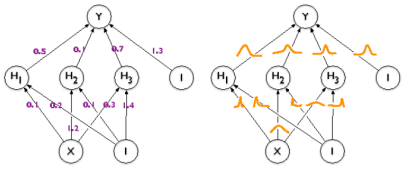
VIME implementation:

IG can be equivalently written as $D_{\mathrm{KL}}(p(\theta|h, s_t, a_t, s_{t+1})\|p(\theta|h))$

model parameters for $p_\theta(s_{t+1}|s_t, a_t)$

history of all prior transitions

newly observed transition



$q(\theta|\phi) \approx p(\theta|h)$ specifically, optimize variational lower bound $D_{\mathrm{KL}}(q(\theta|\phi)\|p(h|\theta)p(\theta))$

represent $q(\theta|\phi)$ as product of independent Gaussian parameter distributions

with mean ϕ (see Blundell et al. "Weight uncertainty in neural networks")

given new transition (s, a, s'), update ϕ to get ϕ'

i.e., update the network weight means and variances

use $D_{\mathrm{KL}}(q(\theta|\phi')\|q(\theta|\phi))$ as approximate bonus

$$p(\theta|\mathcal{D}) = \prod_i p(\theta_i|\mathcal{D})$$

$$p(\theta_i|\mathcal{D}) = \mathcal{N}(\mu_i, \sigma_i)$$

ϕ

Houthooft et al. "VIME"

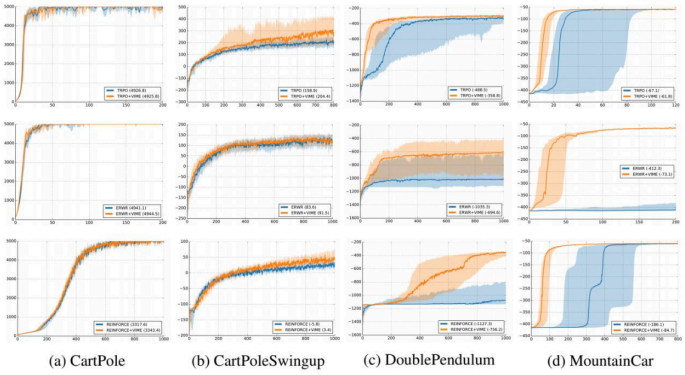
Reasoning about information gain (approximately)

VIME implementation:

IG can be equivalently written as $D_{\mathrm{KL}}(p(\theta|h, s_t, a_t, s_{t+1})\|p(\theta|h))$

$q(\theta|\phi) \approx p(\theta|h)$ specifically, optimize variational lower bound $D_{\mathrm{KL}}(q(\theta|\phi)\|p(h|\theta)p(\theta))$

use $D_{\mathrm{KL}}(q(\theta|\phi')\|q(\theta|\phi))$ as approximate bonus



(a) CartPole (b) CartPoleSwingup (c) DoublePendulum (d) MountainCar

Houthooft et al. “VIME”

Approximate IG:

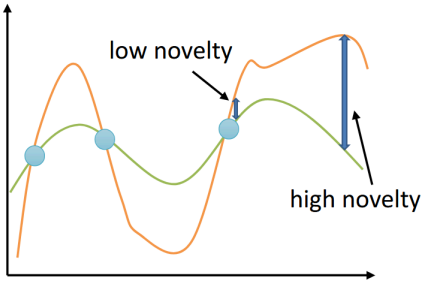
+ appealing mathematical formalism

- models are more complex, generally harder to use effectively

Exploration with model errors

$D_{\mathrm{KL}}(q(\theta|\phi')\|q(\theta|\phi))$ can be seen as change in network (mean) parameters ϕ

if we forget about IG, there are many other ways to measure this



Stadie et al. 2015:

- encode image observations using auto-encoder
- build predictive model on auto-encoder latent states
- use model error as exploration bonus

Schmidhuber et al. (see, e.g. “Formal Theory of Creativity, Fun, and Intrinsic Motivation):

- exploration bonus for model error
- exploration bonus for model gradient
- many other variations

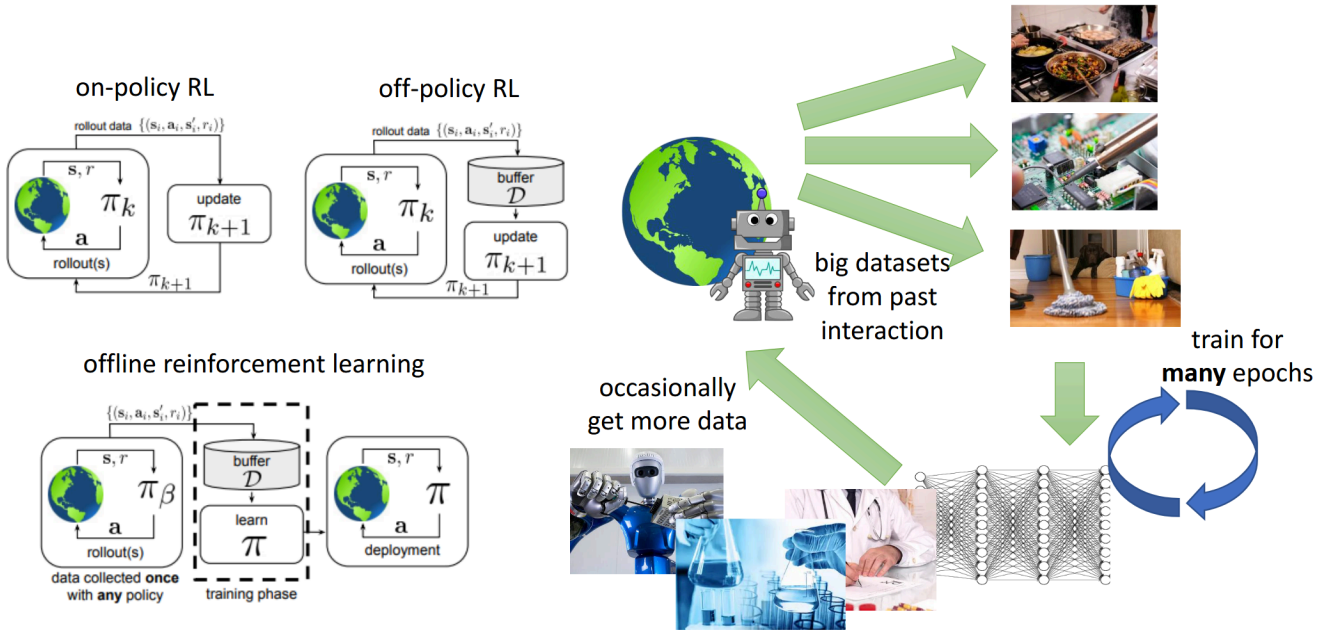
Many others!

Offline RL

Basic introduction

在Online RL中我们在训练过程中都与环境交互，更新了数据。我们希望像现在的监督学习一样，有一个大的数据集之后就可以完全依据这个数据集进行训练而无需与环境交互

Can we develop **data-driven** RL methods?



Levine. Kumar. Tucker. Fu. **Offline Reinforcement Learning: Tutorial. Review. and Perspectives on Open Problems.** '20

这张图展示了offline RL和on-policy、off-policy之间的区别

Formally offline-RL definition：

$\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ 这里并不假设 s_i 与 s'_i 是trajectories中的连续状态

$\mathbf{s} \sim d^{\pi_\beta}(\mathbf{s})$ **状态 s 是从策略 π_β 所诱导的状态分布 $d^{\pi_\beta}(s)$ 中采样得到的。这里 π_β 通常不知道，是一个随机策略**

$\mathbf{a} \sim \pi_\beta(\mathbf{a}|\mathbf{s})$ 动作转移，由policy π_β 和当前状态 s 决定

$\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ 状态转移函数

$r \leftarrow r(\mathbf{s}, \mathbf{a})$ 奖励函数

RL objective: $\max_\pi \sum_{t=0}^{T} \mathbb{E}_{\mathbf{s}_t \sim d^\pi(\mathbf{s}), \mathbf{a}_t \sim \pi(\mathbf{a}|\mathbf{s})} \left[\gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right]$

difference between off-policy

使用数据集来学习策略我们之前在off-policy evaluation (OPE) 中也提到过：

given $\mathcal{D}$, estimate $J(\pi) = E_\pi \left[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right]$

但是，在offline RL中

given $\mathcal{D}$ 学习"最好"的策略，**这里最佳策略并不是整个要解决的MDP问题的最优解，而是在这个数据集 $\mathcal{D}$ 上的最优解！！！**

我们在off-policy（比如模仿学习）中学习到的策略通常不是最优的，如果数据中有好有坏，可能获得的最终策略是一个平均水平的policy

但是offline RL 中不仅可以学习到数据中最好的策略，甚至可以做的更好：从很多次优策略中学

习到更优策略!

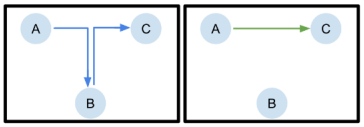
What do we expect offline RL methods to do?

Bad intuition: it's like imitation learning

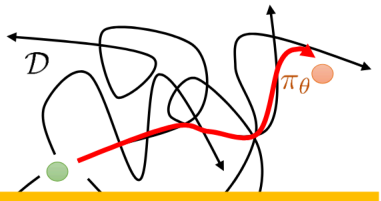
Though it can be shown to be **provably** better than imitation learning even with optimal data, under some structural assumptions!

See: Kumar, Hong, Singh, Levine. Should I Run Offline Reinforcement Learning or Behavioral Cloning?

Better intuition: get order from chaos



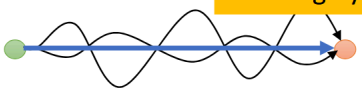
"Macro-scale" stitching



If we have algorithms that properly perform dynamic programming, we can take this idea much further and get near-optimal policies from highly suboptimal data

But this is just the clearest example!

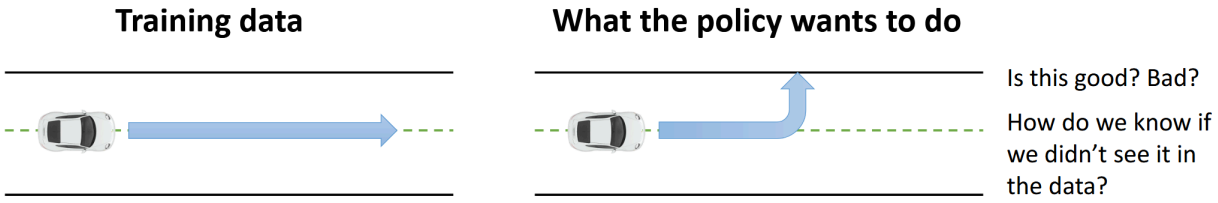
"Micro-scale" stitching:



KEY！！！Why is offline RL hard?

实际上问题在于我们不知道一些没出现在数据集里的动作是否是好的，在online RL 中我们可以直接尝试这个动作，看看后续Q values的更新，但是在offline RL 中我们无法判断，即counterfactural queries 反事实查询

Fundamental problem: counterfactual queries



Online RL algorithms don't have to handle this, because they can simply **try** this action and see what happens

Offline RL methods must somehow account for these unseen ("out-of-distribution") actions, ideally in a safe way

…while still making use of generalization to come up with behaviors that are better than the best thing seen in the data!

14

Levine, Kumar, Tucker, Fu. **Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems.** '20

数学上的解释即分布偏移 Distribution shift 与模仿学习中的类似：

行为策略（Behavior Policy） π_β：生成数据集 $\mathcal{D}$ 的策略。

新策略（New Policy） π_{new}：模型正在学习或评估的策略

无论更新Q values时使用 $\max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}')$ 还是 $E_{\mathbf{a}' \sim \pi_{\text{new}}}[Q(\mathbf{s}', \mathbf{a}')]$ 我们都是用 π_β 来采样动作，用正

在学习的策略 π_{new} 来评估和更新，一旦新策略找到一个可以使估计 Q values变大的动作(但是实际上可能并不是好的，offline RL 无法验证)，那么就会一直高估 Q 同时实际效果还不好

Where do we suffer from distribution shift?

~~$Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \max_{\mathbf{a}'} Q(\mathbf{s}',\mathbf{a}')$~~

$Q(\mathbf{s},\mathbf{a}) \leftarrow \underbrace{r(\mathbf{s},\mathbf{a}) + E_{\mathbf{a}'\sim\pi_{\text{new}}}[Q(\mathbf{s}',\mathbf{a}')]}_{y(\mathbf{s},\mathbf{a})}$

what is the objective?

$\min_Q E_{(\mathbf{s},\mathbf{a})\sim\pi_\beta(\mathbf{s},\mathbf{a})}\left[(Q(\mathbf{s},\mathbf{a}) - y(\mathbf{s},\mathbf{a}))^2\right]$

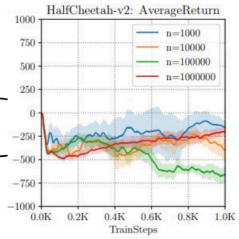
behavior policy　　target value

expect good accuracy when $\pi_\beta(\mathbf{a}|\mathbf{s}) = \pi_{\text{new}}(\mathbf{a}|\mathbf{s})$

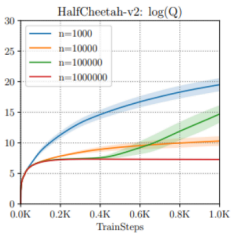
how often does *that* happen?

even *worse*: $\pi_{\text{new}} = \arg\max_\pi E_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})]$

(what if we pick $\mathbf{x}^\star \leftarrow \arg\max_{\mathbf{x}} f_\theta(\mathbf{x})$?)



how well it does



how well it *thinks* it does (Q-values)

Kumar, Fu, Tucker, Levine. **Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction.** NeurIPS '19

怎么解决？见下面Explicit policy constraint methods

1. **约束新策略**：BC CQL AWAC
2. **惩罚对未见动作的过度自信。**IQL

Policy Constraint

intuition：我们可以通过限制 π_{new} 和 π_β 的差异来减少这两个策略之间的分布差异，比如通过约束KL-散度来确保 π_{new} 与 π_β 之间差别不会太大
但是有两个问题

1. 不知道 π_β

2. 实际策略 π_β 分布并不理想，有一些问题(见下)

How do prior methods address this?

$$Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + E_{\mathbf{a}'\sim\pi_{\text{new}}}[Q(\mathbf{s}',\mathbf{a}')]$$

$$\pi_{\text{new}}(\mathbf{a}|\mathbf{s}) = \arg\max_{\pi} E_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] \text{ s.t. } D_{\text{KL}}(\pi\|\pi_\beta) \leq \epsilon$$

This solves distribution shift, right?

No more erroneous values?

Issue 1: we usually don't know the behavior policy $\pi_\beta(\mathbf{a}|\mathbf{s})$

- human-provided data
- data from hand-designed controller
- data from many past RL runs
- all of the above

Issue 2: this is both *too pessimistic* and *not pessimistic enough*

"policy constraint" method

very old idea (but it had no single name?)

Todorov et al. [passive dynamics in linearly-solvable MDPs]

Kappen et al. [KL-divergence control, etc.]

trust regions, covariant policy gradients, natural policy gradients, etc.

used in some form in recent papers:

Fox et al. '15 ("Taming the Noise…")

Fujimoto et al. '18 ("Off Policy…")

Jaques et al. '19 ("Way Off Policy…")

Kumar et al. '19 ("Stabilizing…")

Wu et al. '19 ("Behavior Regularized…")

Levine, Kumar, Tucker, Fu. **Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems.** '20

像右上角图像中所示：其中蓝色线为 π_β ，橙色线为不采用constrain拟合的 π_{new} ，紫色描的浅绿线是在KL-constraint下获得的最好策略 π ，红色描的深绿色线是实际上我们根据 $\mathcal{D}$ 能获得最好策略

所以，单纯使用KL-Divergence来约束策略，仍然会保留一些不好动作的的出现概率。但是由于实现起来KL散度比较容易，接下来我们还是使用KL散度来进行一些算法实现

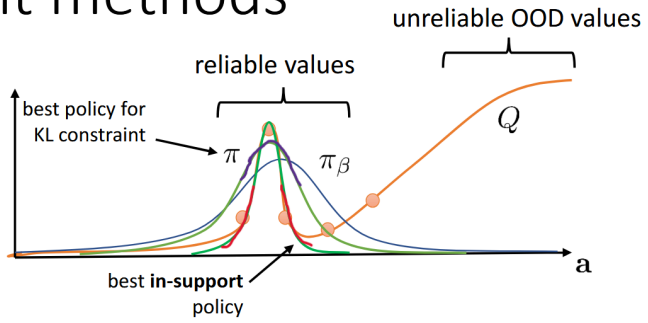
Explicit policy constraint methods

What kinds of constraints can we use?

KL-divergence: $D_{\text{KL}}(\pi\|\pi_\beta)$

+ easy to implement (more on this later)

- not necessarily what we want



support constraint: $\pi(\mathbf{a}|\mathbf{s}) \geq 0$ only if $\pi_\beta(\mathbf{a}|\mathbf{s}) \geq \epsilon$

can approximate with MMD

- significantly more complex to implement

+ much closer to what we really want

For more information, see:

Levine, Kumar, Tucker, Fu. **Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems.** '20

Kumar, Fu, Tucker, Levine. **Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction.** '19

Wu, Tucker, Nachum. **Behavior Regularized Offline Reinforcement Learning.** `19

一共有三种方法可以实现constraints，前两种实践效果都不好

1. 在actor参数更新时加入 D_{KL} 项，这里引入拉格朗日乘子 λ 是用来计算有 D_{KL} 参与的极值求解
2. 直接在奖励函数 r 中加入 D_{KL} 惩罚项

How do we implement constraints?

Lagrange multiplier

easy to compute and differentiate for Gaussian or categorical policies

1. Modify the actor objective

~~$\theta \leftarrow \arg\max_\theta E_{\mathbf{s}\sim D}\left[E_{\mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})]\right]$~~

$$\theta \leftarrow \arg\max_\theta E_{\mathbf{s}\sim D}\left[E_{\mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a}) + \lambda \log \pi_\beta(\mathbf{a}|\mathbf{s})] + \lambda\mathcal{H}(\pi(\mathbf{a}|\mathbf{s}))\right]$$

$$D_{\mathrm{KL}}(\pi\|\pi_\beta) = E_\pi[\log\pi(\mathbf{a}|\mathbf{s}) - \log\pi_\beta(\mathbf{a}|\mathbf{s})] = -E_\pi[\log\pi_\beta(\mathbf{a}|\mathbf{s})] - \mathcal{H}(\pi)$$

2. Modify the reward function

$$\bar{r}(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) - D(\pi,\pi_\beta)$$

simple modification to directly penalize divergence
also accounts for **future** divergence

See: Wu, Tucker, Nachum. **Behavior Regularized Offline Reinforcement Learning.** `19

3. AWAC

AWAC

KEY：显式约束策略(actor)来减少不好动作的发生

在优势函数中引入权重来放大好动作的概率，减小坏动作的的概率，实践表现较好

这里由第一行可以直接解出第二行的最优策略

$$\pi_{\mathrm{new}}(\mathbf{a}|\mathbf{s}) = \arg\max_\pi E_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] \text{ s.t. } D_{\mathrm{KL}}(\pi\|\pi_\beta) \le \epsilon$$

$$\pi^\star(\mathbf{a}|\mathbf{s}) = \frac{1}{Z(\mathbf{s})}\pi_\beta(\mathbf{a}|\mathbf{s})\exp\left(\frac{1}{\lambda}A^\pi(\mathbf{s},\mathbf{a})\right)$$

straightforward to show via duality

See also:
Peters et al. (REPS)
Rawlik et al. ("psi-learning")
…many follow-ups

approximate via **weighted** max likelihood!

$$\pi_{\mathrm{new}}(\mathbf{a}|\mathbf{s}) = \arg\max_\pi E_{(\mathbf{s},\mathbf{a})\sim\pi_\beta}\left[\log\pi(\mathbf{a}|\mathbf{s})\overbrace{\frac{1}{Z(\mathbf{s})}\exp\left(\frac{1}{\lambda}A^{\pi_{\mathrm{old}}}(\mathbf{s},\mathbf{a})\right)}^{w(\mathbf{s},\mathbf{a})}\right]$$

samples from dataset $\mathbf{a}\sim\pi_\beta(\mathbf{a}|\mathbf{s})$

critic can be used to give us this

Peng*, Kumar*, Levine. **Advantage-Weighted Regression.** '19

Nair, Dalal, Gupta, Levine. **Accelerating Online Reinforcement Learning with Offline Datasets.** '20

下图是理论优化：与普通Q-Learning不同的是，更新critic时：更新 Q 值时Q-Learning使用贪婪策略，即用 $\arg\max$ 更新，而这里使用 $E_{\mathbf{a}'}\sim\pi_{\mathrm{new}}$ 更新。更新actor时有KL-Divergence的约束条件

$$Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + E_{\mathbf{a}'\sim\pi_{\text{new}}}[Q(\mathbf{s}',\mathbf{a}')]$$

$$\pi_{\text{new}}(\mathbf{a}|\mathbf{s}) = \arg\max_{\pi} E_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] \text{ s.t. } D_{\text{KL}}(\pi\|\pi_\beta) \leq \epsilon$$

实际训练时更新见下图，由critic和actor组成，训练时引入weighted权重

$\mathcal{L}_C(\phi)$ 是critic的损失函数 MSE， $Q_\phi(\mathbf{s},\mathbf{a})$ 是critic当前的状态估计，

$r(\mathbf{s},\mathbf{a}) + \gamma E_{\mathbf{a}'\sim\pi_\theta(\mathbf{a}'|\mathbf{s}')}[Q_\phi(\mathbf{s}',\mathbf{a}')]$ 是Bellman 目标（Target value），注意这里的动作 $\mathbf{a}'$是从actor中采样得来的

$L_A(\theta)$ 是actor的损失函数 加权的最大似然（Weighted Maximum Likelihood），$\log\pi_\theta(\mathbf{a}|\mathbf{s})$ 是标准的模仿学习损失，鼓励新策略模仿数据集中的动作，这里的 $A^{\pi_{\text{old}}}(\mathbf{s},\mathbf{a})$ 是指在上一轮的critic参数下计算 $A = Q - V$

$$\mathcal{L}_C(\phi) = E_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim D}\left[\left(Q_\phi(\mathbf{s},\mathbf{a}) - (r(\mathbf{s},\mathbf{a}) + \gamma E_{\mathbf{a}'\sim\pi_\theta(\mathbf{a}'|\mathbf{s}')}[Q_\phi(\mathbf{s}',\mathbf{a}')])\right)^2\right]$$

$$\mathcal{L}_A(\theta) = -E_{(\mathbf{s},\mathbf{a})\sim\pi_\beta}\left[\log\pi_\theta(\mathbf{a}|\mathbf{s})\frac{1}{Z(\mathbf{s})}\exp\left(\frac{1}{\lambda}A^{\pi_{\text{old}}}(\mathbf{s},\mathbf{a})\right)\right]$$

1. $\phi \leftarrow \phi - \alpha\nabla_\phi\mathcal{L}_C(\phi)$
2. $\theta \leftarrow \theta - \alpha\nabla_\theta\mathcal{L}_A(\theta)$

IQL Implict Q-Learning

KEY：隐式约束 Q 值来使没见过的数据(OOD data)的 Q 值估计变小

AWAC中 Q 值更新为 $Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + E_{\mathbf{a}'\sim\pi_{\text{new}}}[Q(\mathbf{s}',\mathbf{a}')]$ 如果使用神经网络来拟合 $V(\mathbf{s}')$ 来替换对 Q 取期望的估计

那我们就可以用一种是损失函数来更新 V

$$V \leftarrow \arg\min_V \frac{1}{N}\sum_{i=1}^{N}\ell(V(\mathbf{s}_i), Q(\mathbf{s}_i,\mathbf{a}_i))$$

这里 $Q(\mathbf{s_i},\mathbf{a_i})$ 中的 a_i 都是从初始策略 π_β 中采样的，也就是说，V 更新时会像初始数据集中a_i决定的Q 来更新，也就是隐式地加强了对初始数据中action的权重，进而减少OOD数据参与update

这里使用什么损失函数呢？如果是MSE，则最终如右侧概率分布图所示，最终 V 会收敛到在 π_β 下的期望 Q 值，而我们希望收敛到能在 π_β 下获得的最好policy，所以使用expectile损失函数，这种函数时加权损失，在值较大的损失较少，进而可以让 V 收敛到高分位值

Can we **also** avoid all OOD actions in the Q update?

$$Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \underbrace{E_{\mathbf{a}'\sim\pi_{\text{new}}}[Q(\mathbf{s}',\mathbf{a}')]}_{V(\mathbf{s}')}$$

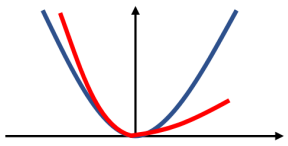
$V(\mathbf{s}')$ ← just another neural network

$$V \leftarrow \arg\min_V \frac{1}{N}\sum_{i=1}^{N} \ell(V(\mathbf{s}_i), Q(\mathbf{s}_i,\mathbf{a}_i))$$

e.g., MSE loss $(V(\mathbf{s}_i) - Q(\mathbf{s}_i,\mathbf{a}_i))^2$

this action comes from π_β not from π_{new}

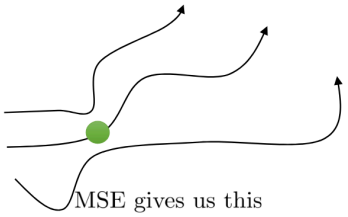
expectile: $\ell_2^\tau(x) = \begin{cases} (1-\tau)x^2 & \text{if } x > 0 \\ \tau x^2 & \text{else} \end{cases}$

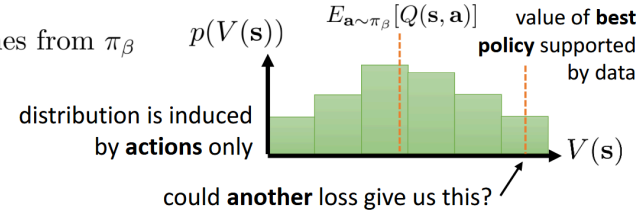


$$V(\mathbf{s}) \leftarrow \max_{\mathbf{a}\in\Omega(\mathbf{s})} Q(\mathbf{s},\mathbf{a})$$

$$\Omega(\mathbf{s}) = \{\mathbf{a} : \pi_\beta(\mathbf{a}|\mathbf{s}) \geq \epsilon\}$$

if we use ℓ_2^τ for big τ





Kostrikov, Nair, Levine. **Offline Reinforcement Learning with Implicit Q-Learning.** '21

IQL算法就是隐式更新 Q 来避免OOD动作的采取，论文中证明如果使用expectile损失函数当 τ 足够大时会有： $V(\mathbf{s}) \leftarrow \max_{\mathbf{a}\in\Omega(\mathbf{s})} Q(\mathbf{s},\mathbf{a})$ $\Omega(\mathbf{s}) = \{\mathbf{a} : \pi_\beta(\mathbf{a}|\mathbf{s}) \geq \epsilon\}$

进而有：$Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \max_{\mathbf{a}'\in\Omega(\mathbf{s}')} Q(\mathbf{s}',\mathbf{a}')$

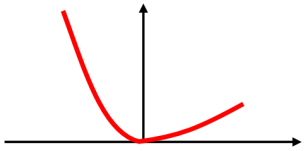
最终隐式策略就是：

$\pi_{\text{new}}(\mathbf{a}|\mathbf{s}) = \delta(\mathbf{a} = \arg\max_{\mathbf{a}\in\Omega(\mathbf{s})} Q(\mathbf{s},\mathbf{a}))$

Implicit Q-learning (IQL)

Q-learning with *implicit* policy improvement

$$Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + V(\mathbf{s}') \qquad V \leftarrow \arg\min_V \frac{1}{N}\sum_{i=1}^{N} \ell_2^\tau(V(\mathbf{s}_i), Q(\mathbf{s}_i,\mathbf{a}_i))$$



$$V(\mathbf{s}) \leftarrow \max_{\mathbf{a}\in\Omega(\mathbf{s})} Q(\mathbf{s},\mathbf{a})$$

$$\Omega(\mathbf{s}) = \{\mathbf{a} : \pi_\beta(\mathbf{a}|\mathbf{s}) \geq \epsilon\}$$

if we use ℓ_2^τ for big τ

$$Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \max_{\mathbf{a}'\in\Omega(\mathbf{s}')} Q(\mathbf{s}',\mathbf{a}')$$

"implicit" policy

$$\pi_{\text{new}}(\mathbf{a}|\mathbf{s}) = \delta(\mathbf{a} = \arg\max_{\mathbf{a}\in\Omega(\mathbf{s})} Q(\mathbf{s},\mathbf{a}))$$

CQL Conservative Q-learning

通过修改Q-learning的损失函数，显式地惩罚模型对未见动作（OOD actions）的高Q值预测，特指OOD的高 Q 预测，既不是单纯惩罚OOD 也不是单纯惩罚 high Q ，而是惩罚OOD中的high Q !

这里 $\hat{Q}^\pi$ 代表正在学习的 Q value， Q^π 指真正的 Q value

A *better* bound: always pushes Q-values down; push up on (**s**, **a**) samples in data

$$\hat{Q}^\pi = \arg\min_Q \max_\mu \alpha E_{\mathbf{s}\sim D, \mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] - \alpha E_{(\mathbf{s},\mathbf{a})\sim D}[Q(\mathbf{s},\mathbf{a})] + E_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim D}\left[\left(Q(\mathbf{s},\mathbf{a}) - (r(\mathbf{s},\mathbf{a}) + E_\pi[Q(\mathbf{s}',\mathbf{a}')])\right)^2\right] \Big\} \mathcal{L}_{\mathrm{CQL}}(\hat{Q}^\pi)$$

no longer guaranteed that $\hat{Q}^\pi(\mathbf{s},\mathbf{a}) \le Q^\pi(\mathbf{s},\mathbf{a})$ *for all* $(\mathbf{s},\mathbf{a})$

but guaranteed that $E_{\pi(\mathbf{a}|\mathbf{s})}[\hat{Q}^\pi(\mathbf{s},\mathbf{a})] \le E_{\pi(\mathbf{a}|\mathbf{s})}[Q^\pi(\mathbf{s},\mathbf{a})]$ *for all* $\mathbf{s} \in D$

$$\mathcal{L}_{CQL}(\hat{Q}^\pi) = \underbrace{\alpha\mathbb{E}_{s\sim\mathcal{D},a\sim\mu(a|s)}[Q(s,a)]}_{\text{1. Push down big Q-values}} - \underbrace{\alpha\mathbb{E}_{s,a\sim\mathcal{D}}[Q(s,a)]}_{\text{2. Push up on in-data samples}} + \underbrace{\mathbb{E}_{s,a,s'\sim\mathcal{D}}[(Q(s,a)-y(s,a))^2]}_{\text{3. Regular Bellman Error}}$$

第一项：Push down big Q-values

- **公式**：$\alpha\mathbb{E}_{s\sim\mathcal{D},a\sim\mu(a|s)}[Q(s,a)]$
- **含义**：这一项是CQL最核心的部分。它试图**最小化** Q 函数在**一个特定策略 μ 下的期望值**。这里 μ 的选择见下文
  - $\mu(a|s)$ 是一个专门设计的策略，它被用来**探索那些高Q值的动作**。它试图找到Q值最高的动作，然后对它进行惩罚。
- **直觉推导**：通过最小化这一项，我们**显式地惩罚**那些可能被Q函数高估的动作。

第二项：Push up on in-data samples

- **公式**：$-\alpha\mathbb{E}_{s,a\sim\mathcal{D}}[Q(s,a)]$
- **含义**：这一项是**最大化** Q 函数在**数据集中**的期望值。
- **直觉推导**：通过最大化这一项，我们**鼓励**Q函数对数据集中出现的动作给出更高的Q值。
- **与第一项的关系**：这两项结合起来形成一个**正则化项**：$\alpha(\mathbb{E}_{OOD}[Q] - \mathbb{E}_{in-data}[Q])$。这个正则化项的目的是**拉开**OOD动作的Q值和in-data动作的Q值，迫使OOD动作的Q值被压低。

第三项：Regular Bellman Error

- **公式**：$\mathbb{E}_{s,a,s'\sim\mathcal{D}}[(Q(s,a)-y(s,a))^2]$
- **含义**：这是标准的**贝尔曼误差**。这里的 $y(s,a)$ 是贝尔曼目标，例如 $r(s,a) + \gamma\max_{a'} Q(s',a')$ 。
- **直觉推导**：这一项确保模型仍然在学习正确的动态和价值。

如何选择惩罚策略 $\mu(a|s)$?
这里要推导出最好的策略 μ ，在 $\hat{Q}^\pi$ 中引入关于策略 μ 的正则化项 $\mathcal{R}(\mu)$
我们一般使用最大熵正则化，最优选择是$\mu(\mathbf{a}\mid\mathbf{s}) \propto \exp\left(\frac{1}{\alpha}Q(s,a)\right)$
从而在计算$\alpha\mathbb{E}_{s\sim\mathcal{D},a\sim\mu(a|s)}[Q(s,a)]$ 时，就使用：

$$E_{\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] = \log \sum_{a} \exp(Q(\mathbf{s},\mathbf{a}))$$

离散动作就直接只用贪婪策略，连续的动作就用重要性采样来估计。这里因为我们已经推导出最优解，就可以直接使用重要性采样估计来计算，而不用引入新参数后用策略梯度来更新参数了

Conservative Q-learning (CQL)

regularization

$$\hat{Q}^{\pi} = \arg\min_{Q} \max_{\mu} \alpha E_{\mathbf{s}\sim D,\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] - \alpha E_{(\mathbf{s},\mathbf{a})\sim D}[Q(\mathbf{s},\mathbf{a})] - \mathcal{R}(\mu)$$
$$+ E_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim D}\left[\left(Q(\mathbf{s},\mathbf{a}) - (r(\mathbf{s},\mathbf{a}) + E_{\pi}[Q(\mathbf{s}',\mathbf{a}')])\right)^2\right] \Big\} \mathcal{L}_{\mathrm{CQL}}(\hat{Q}^{\pi})$$

common choice: $\mathcal{R} = E_{\mathbf{s}\sim D}[\mathcal{H}(\mu(\cdot|\mathbf{s}))]$ maximum entropy regularization

optimal choice: $\mu(\mathbf{a}|\mathbf{s}) \propto \exp(Q(\mathbf{s},\mathbf{a}))$

$$E_{\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] = \log \sum_{\mathbf{a}} \exp(Q(\mathbf{s},\mathbf{a}))$$

for discrete actions: just calculate directly

for continuous actions: use importance sampling to estimate $E_{\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})]$

Kumar, Zhou, Tucker, Levine. **Conservative Q-Learning for Offline Reinforcement Learning.** '20

CQL的训练流程：

1. Update $\hat{Q}^\pi$ w.r.t. $\mathcal{L}_{\text{CQL}}(\hat{Q}^\pi)$ using $\mathcal{D}$
2. Update policy π

if actions are discrete:

$$\pi(\mathbf{a}|\mathbf{s}) = \begin{cases} 1 \text{ if } \mathbf{a} = \arg\max_{\mathbf{a}} \hat{Q}(\mathbf{s}, \mathbf{a}) \\ 0 \text{ otherwise} \end{cases}$$

if actions are continuous:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \sum_i E_{\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)}[\hat{Q}(\mathbf{s}_i, \mathbf{a})]$$

1. **更新 Critic**：使用 CQL 损失函数 $\mathcal{L}_{CQL}$，从离线数据集 $\mathcal{D}$ 中采样，并使用梯度下降来更新 Q 函数 $\hat{Q}^\pi$ 的参数。
2. **更新 Policy**：根据更新好的 Q 函数 $\hat{Q}^\pi$，来更新策略 π。
   - **离散动作**：策略可以直接是贪婪策略，即选择Q值最大的动作。
   - **连续动作**：策略通常是使用**策略梯度**（policy gradient）来更新。这里的策略梯度是基于 CQL 学习到的 Q 函数，而不是完整的蒙特卡洛回报。

offline RL via Importance Sampling

就像off-policy一样，使用两个策略来分别实行动作和更新策略，在offline RL 中也可以通过重要性采样来更新策略

Estimating the returns

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}\nabla_\theta \gamma^t \log \pi_\theta(\mathbf{a}_{t,i}|\mathbf{s}_{t,i})\underbrace{\left(\prod_{t'=t}^{T}\frac{\pi_\theta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})}{\pi_\beta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})}\right)\hat{Q}(\mathbf{s}_{t,i},\mathbf{a}_{t,i})}_{\sum_{t'=t}^{T}\left(\prod_{t''=t}^{t'}\frac{\pi_\theta(\mathbf{a}_{t'',i}|\mathbf{s}_{t'',i})}{\pi_\beta(\mathbf{a}_{t'',i}|\mathbf{s}_{t'',i})}\right)\gamma^{t'-t}r_{t',i}} \quad \leftarrow E_{\pi_\theta}\left[\sum_{t'=t}^{T}\gamma^{t'-t}r_{t'}\right] \approx \sum_{t'=t}^{T}\gamma^{t'-t}r_{t',i}$$

but this is *still* exponential!

To avoid exponentially exploding importance weights, we **must** use value function estimation!

imagine we knew $Q^{\pi_\theta}(\mathbf{s},\mathbf{a})$

第一个约等于式子是高指数性的，记 $\rho_{t'} = \frac{\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\pi_\beta(\mathbf{a}_{t'}|\mathbf{s}_{t'})}$

The doubly robust estimator

$$\begin{aligned}
V^{\pi_\theta}(\mathbf{s}_0) &\approx \sum_{t=0}^{T}\left(\prod_{t'=0}^{t}\frac{\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\pi_\beta(\mathbf{a}_{t'}|\mathbf{s}_{t'})}\right)\gamma^t r_t \\
&= \sum_{t=0}^{T}\left(\prod_{t'=0}^{t}\rho_{t'}\right)\gamma^t r_t \\
&= \rho_0 r_0 + \rho_0\gamma\rho_1 r_1 + \rho_0\gamma\rho_1\gamma\rho_2 r_2 + \dots \\
&= \rho_0(r_0 + \gamma(\rho_1(r_1 + \gamma(\rho_2(r_2 + \gamma\dots))))) \\
&= \bar{V}^T \qquad \text{where } \bar{V}^{T+1-t} = \rho_t(r_t + \gamma\bar{V}^{T-t})
\end{aligned}$$

$$\bar{V}_{\text{DR}}^{T+1-t} = \hat{V}(\mathbf{s}_t) + \rho_t(r_t + \gamma\bar{V}_{\text{DR}}^{T-t} - \hat{Q}(\mathbf{s}_t,\mathbf{a}_t))$$

this is exponential!

doubly robust estimation (bandit case)

$$V_{\text{DR}}(s) = \hat{V}(s) + \rho(s,a)(r_{s,a} - \hat{Q}(s,a))$$

model or function approximator

这个doubly robust 估计器与offline RL 无关，但是是解决重要性采样问题和纯价值估计的一种方法(回收前文重要性采样伏笔)

offline RL via Linear Fitted Value Functions

与现代的deep rl 不同，但是是经典的基于值的线性逼近方法
待补

Model-Based Offline RL

MOPO

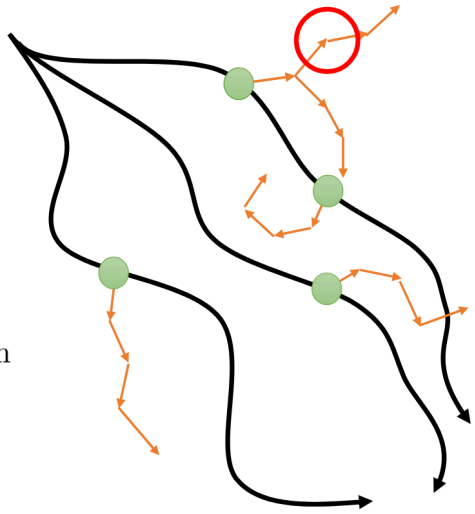
MOPO: Model-Based Offline Policy Optimization

solution: "punish" the policy for exploiting

$$\tilde{r}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) - \lambda u(\mathbf{s}, \mathbf{a})$$

uncertainty penalty

...and then use any existing model-based RL algorithm



MOPO：使用**模型（Model-based）** 来生成数据，并用**不确定性**来惩罚生成的数据，从而解决分布偏移。

- **MOPO的流程**：
  1. **模型学习**：首先，用离线数据集 D 训练一个**环境模型**，使其能够预测状态转移 $p(s\prime \mid s, a)$ 。
  2. **数据生成**：在训练策略时，智能体从离线数据集中采样一个状态 s 和动作 a。
  3. **模型rollout**：使用训练好的环境模型来生成**新的、合成的数据** $(s, a, s\prime, r)$。
  4. 不确定性惩罚：这是MOPO最关键的部分。我们不是直接使用模型生成的数据，而是修改其奖励。

     $r\ (s, a) = r(s, a) - \lambda u(s, a)$
     - $r(s, a)$ 是环境给出的真实奖励。
     - $u(s, a)$ 是模型在预测状态转移时的**不确定性**。这里可以使用之前学习到的模型不确定性衡量方法 比如 集成(ensembles)
     - λ 是一个超参数，控制惩罚的强度。

COMBO（Conservative Model-Based RL）

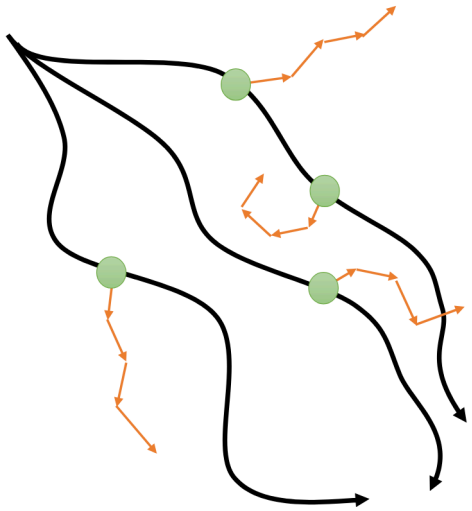
COMBO: Conservative Model-Based RL

Basic idea: just like CQL minimizes Q-value of policy actions, we can minimize Q-value of model state-action tuples

state-action tuples from the model

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \beta \left(\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\rho(\mathbf{s},\mathbf{a})}[Q(\mathbf{s},\mathbf{a})] - \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q(\mathbf{s},\mathbf{a})] \right)$$
$$+\frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim d_f}\left[\left(Q(\mathbf{s},\mathbf{a}) - \widehat{\mathcal{B}}^{\pi}\hat{Q}^{k}(\mathbf{s},\mathbf{a}) \right)^2 \right]. \qquad (4)$$

Intuition: if the model produces something that looks clearly different from real data, it's easy for the Q-function to make it look bad



COMBO：结合了**模型**和**保守性**思想，在模型生成的数据上应用类似CQL的惩罚，以获得更鲁棒的离线学习。

- **COMBO的流程**：
  1. **数据混合**：训练策略时，它使用一个**混合数据集** $Dmix$，这个数据集包含了**真实的离线数据** D 和**模型生成的数据** $Dmodel$。
  2. CQL损失：它在整个混合数据集 $Dmix$ 上，应用一个类似CQL的损失函数。

  $Q^{k+1} = \arg\min \beta E_{s,a\sim\rho(s,a)}[Q(s,a)] - E_{s,a\sim D}[Q] + \dots$
  - 这里的 ρ 是一个用来**惩罚**的策略，类似于CQL中的 μ。
  - 这里的**惩罚**不仅作用于模型生成的数据，也作用于真实数据。

Trajectory Transformer（轨迹Transformer）

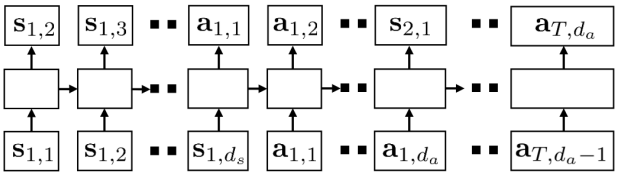
Trajectory Transformer

Basic ideas:

1. train a joint state-action model:
   $p_\beta(\tau) = p_\beta(s_1, a_2, \ldots, s_T, a_T)$
2. use a big expressive model (a Transformer)

Trajectory Transformer making accurate predictions to hundreds of steps

The model:



How to do control:

beam search, but use $\sum_t r(\mathbf{s}_t, \mathbf{a}_t)$ instead of probability

1. given current sequence, sample next tokens from model
2. store top K tokens with highest cumulative reward
3. move on to next token

Why does this work?

generating high-probability trajectories avoids out-of-distribution states & actions

using a really big model works well in offline mode (lots of compute, captures complex behavior policies)

Janner, Li, Levine. **Reinforcement Learning as One Big Sequence Modeling Problem**, 2021.

- **核心思想**：将强化学习问题，特别是**离线强化学习**，完全视为一个**序列建模问题。**
- **基本思路**：
  1. **数据建模**：训练一个联合状态-动作模型$p_\beta(\tau) = p_\beta(s_1, a_2, \ldots, s_T, a_T)$。这个模型将整个轨迹（序列）作为输入，并学习其联合概率分布。
  2. **大模型**：使用一个强大的**Transformer**架构来完成这个序列建模任务。Transformer天然适合处理序列数据，并能捕捉序列中长期的依赖关系。
- **模型结构**：图中左下角展示了模型的输入。它是一个序列，其中每个“token”可以是状态 s 或动作 a。模型会自回归地（Autoregressively）预测下一个 token。
- **如何控制**：
  - 在部署阶段，我们不再像传统的RL那样使用一个策略网络来选择动作。
  - 我们使用**束搜索（beam search）**。
  - 不是像传统语言模型那样根据概率选择最高分的序列，而是根据**累积回报（cumulative reward）** $\sum_t r(s_t, a_t)$ 来选择最高分的序列。
  - 模型会生成多个可能的轨迹，我们选择累积回报最高的那个。
  - **避免分布外（OOD）**：Transformer的自回归特性使得它只会生成在数据集中见过的高概率轨迹。因此，它**天然地避免了OOD状态和动作**，从而解决了分布偏移问题。

选择算法？

Which offline RL algorithm do I use?

If you want to *only* train offline…

Conservative Q-learning	+ just one hyperparameter	+ well understood and widely tested
Implicit Q-learning	+ more flexible (offline + online)	- more hyperparameters

If you want to *only* train offline and finetune online

Advantage-weighted actor-critic (AWAC)	+ widely used and well tested
Implicit Q-learning	+ seems to perform much better!

If you have a good way to train models in your domain

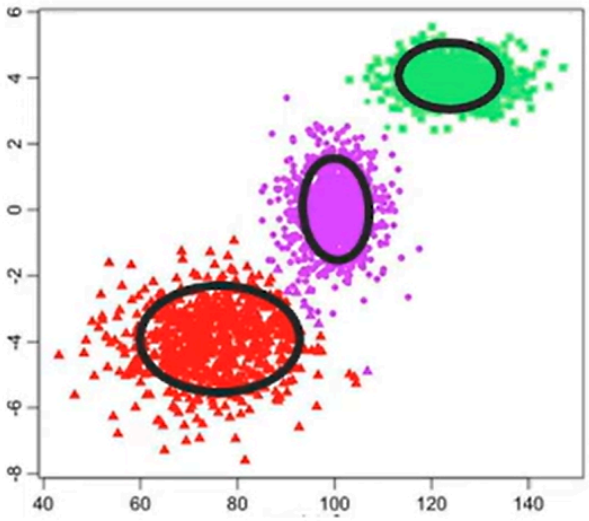
COMBO	+ similar properties as CQL, but benefits from models
	- not always easy to train a good model in your domain!
Trajectory transformer	+ very powerful and effective models
	- extremely computationally expensive to train and evaluate

Variation Inference 变分推理

基础概念basic concepts

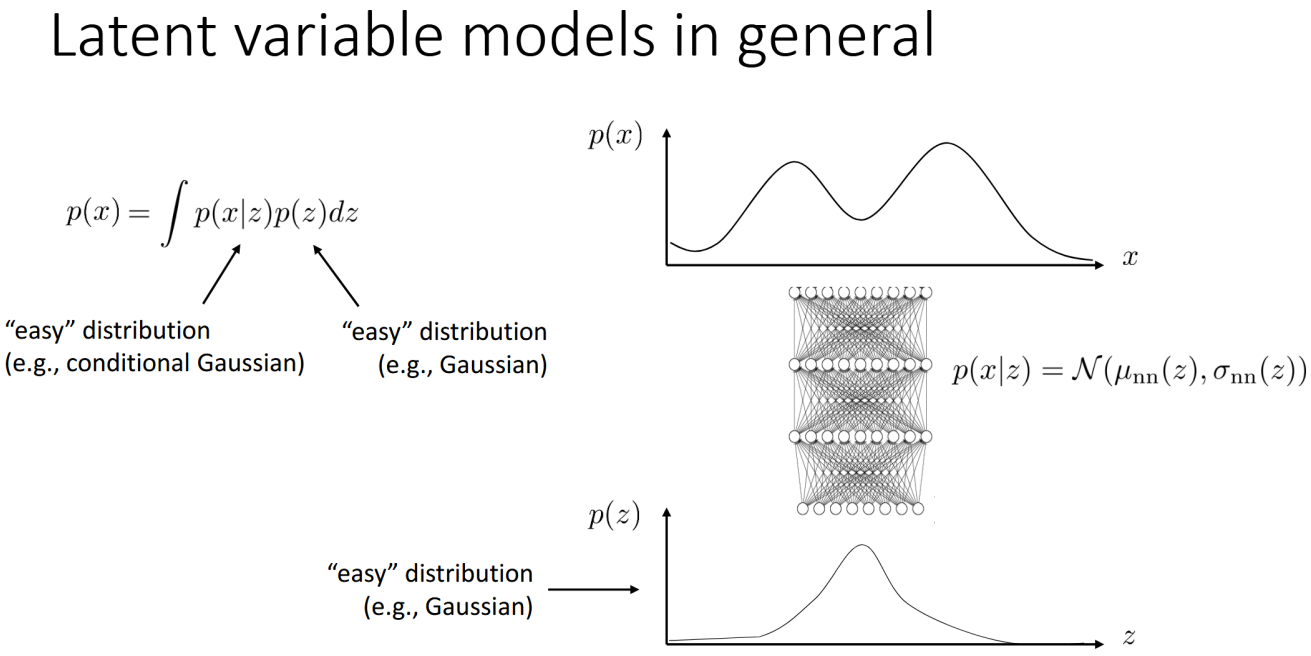
Probability models: 建模 $p(x)$ $p(y|x)$ in RL policy $\pi_\theta(a|s)$ is an example
Latent variable models:

$$p(x) = \sum_z p(x|z)p(z)$$



in this case z is the latent variable which is a category
in general : if $p(x)$ itself is conplicated we can depart $p(x) = \sum_z p(x|z)p(z)$ which $p(x|z)$ and $p(z)$

is easy to calculate. like the folloeing picture we assume $p(z)$ is a simple Gaussian distribution and $p(x|z)$ is also a simple distribution which use neural network to learn the mean and std.



How to train latent variable models?

the model : $p_\theta(x)$

the data : $\mathcal{D} = \{x_1, x_2, \cdots x_N\}$

在生成模型中，我们的目标是让模型能够生成真实的数据。为此，我们希望最大化观测数据 x 的 **边缘似然（Marginal Likelihood）** 也就是提高 $p_\theta(x)$

maxinum likelihood fit : $\theta \leftarrow \arg\max_\theta \frac{1}{N} \sum_i \log p_\theta(x_i)$

already have : $p(x) = \int p(x|z)p(z)dz$

put $p(x)$ into the fitting :

$\theta \leftarrow \arg\max_\theta \frac{1}{N} \sum_i \log \left(\int p_\theta(x_i|z)p(z)dz \right)$ which is untractable

so we try to use estimating the log-likelihood:

使用后验分布 $p(z|x_i)$，但是也难以计算，引入后文variational inference and VAE

Estimating the log-likelihood

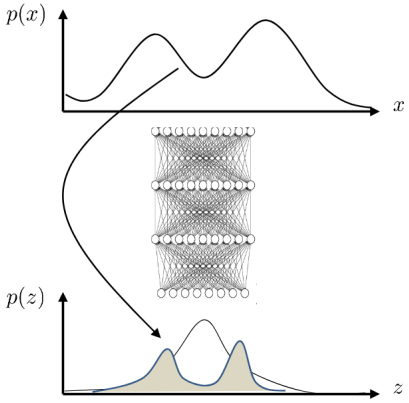
alternative: *expected* log-likelihood:

$$\theta \leftarrow \arg\max_\theta \frac{1}{N}\sum_i E_{z\sim p(z|x_i)}[\log p_\theta(x_i,z)]$$

but... how do we calculate $p(z|x_i)$?

intuition: "guess" most likely z given x_i, and pretend it's the right one

...but there are many possible values of z so use the distribution $p(z|x_i)$



variational approximation

想要计算 $p_\theta(x)$ 就要计算后验分布 $p(z|x_i)$，引入variational approximation估计后验分布
首先引入entropy & KL散度概念

1.Entropy

$$\mathcal{H}(p) = -E_{x\sim p(x)}[\log p(x)] = -\int_x p(x)\log p(x)\,dx$$

intuition: how random is the random variable

2.KL-Divergence

$$D_{\mathrm{KL}}(q\|p) = E_{x\sim q(x)}\left[\log\frac{q(x)}{p(x)}\right] = E_{x\sim q(x)}[\log q(x)] - E_{x\sim q(x)}[\log p(x)] = -E_{x\sim q(x)}[\log p(x)] - \mathcal{H}(q)$$

intuition: how different between two distributions?

现在目标是：提高generat model能力就要提高 $\log p_\theta(x)$，通过证明 $\log p_\theta(x)$ 有下界，就可以通过优化下界来优化 $\log p_\theta(x)$
变分近似variational approximation引入可学习的 $q_\phi(z|x)$ 来逼近 $p(z|x)$ 并且构造下界，以下为证明

现在我们假设对于每一个datapoint x_i 都有一个可学习的 z 分布函数 $q_i(z) = \mathcal{N}(\mu_i, \sigma_i)$
proof 1.

$$\log p(x_i) = \log \int_z p(x_i|z)p(z)$$

$$= \log \int_z p(x_i|z)p(z)\frac{q_i(z)}{q_i(z)}$$

$$= \log E_{z\sim q_i(z)}\left[\frac{p(x_i|z)p(z)}{q_i(z)}\right]$$

$$\geq E_{z\sim q_i(z)}\left[\log \frac{p(x_i|z)p(z)}{q_i(z)}\right]$$

$$= E_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)] - \mathcal{H}(q_i(z)) = \mathcal{L}_i(p, q_i)$$

其中不等式使用：use Jensen inequality: $\log E[y] \geq E[\log y]$ log为凸函数

proof 2.

$$\log p(x_i) \geq \overbrace{E_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)] + \mathcal{H}(q_i)}^{\mathcal{L}_i(p, q_i)}$$

what makes a good $q_i(z)$? intuition: $q_i(z)$ should approximate $p(z|x_i)$

approximate in what sense? compare in terms of KL-divergence: $D_{\mathrm{KL}}(q_i(z)\|p(z|x))$

why?

$$D_{\mathrm{KL}}(q_i(x_i)\|p(z|x_i)) = E_{z\sim q_i(z)}\left[\log \frac{q_i(z)}{p(z|x_i)}\right] = E_{z\sim q_i(z)}\left[\log \frac{q_i(z)p(x_i)}{p(x_i, z)}\right]$$

$$= -E_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)] + E_{z\sim q_i(z)}[\log q_i(z)] + E_{z\sim q_i(z)}[\log p(x_i)]$$

$$= -E_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)] - \mathcal{H}(q_i) + \log p(x_i)$$

$$= -\mathcal{L}_i(p, q_i) + \log p(x_i)$$

$$\log p(x_i) = D_{\mathrm{KL}}(q_i(z)\|p(z|x_i)) + \mathcal{L}_i(p, q_i)$$

$$\log p(x_i) \geq \mathcal{L}_i(p, q_i)$$

通过计算KL散度可以得到精确等式：

$$\log p(x_i) = \mathcal{L}_i(p, q_i) + D_{KL}(q_i(z)||p(z|x_i))$$

所以我们想要增大 $\log p(x_i)$ 就要增大 $\mathcal{L}_i$ 的同时减小 q_i 与 p 之间的 D_{KL} ， 也就是减少 $q_i(z)$ 与 $p(z|x_i)$ 之间的差异

How do we use this?

$$\log p(x_i) \geq \overbrace{E_{z\sim q_i(z)}[\log p_\theta(x_i|z) + \log p(z)] + \mathcal{H}(q_i)}^{\mathcal{L}_i(p,q_i)}$$

$\theta \leftarrow \arg\max_\theta \frac{1}{N}\sum_i \log p_\theta(x_i)$ ~~(crossed out)~~

$\theta \leftarrow \arg\max_\theta \frac{1}{N}\sum_i \mathcal{L}_i(p, q_i)$

for each x_i (or mini-batch):

calculate $\nabla_\theta \mathcal{L}_i(p, q_i)$:

sample $z \sim q_i(z)$

$\nabla_\theta \mathcal{L}_i(p, q_i) \approx \nabla_\theta \log p_\theta(x_i|z)$

$\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}_i(p, q_i)$

update q_i to maximize $\mathcal{L}_i(p, q_i)$ ← how?

let's say $q_i(z) = \mathcal{N}(\mu_i, \sigma_i)$

use gradient $\nabla_{\mu_i} \mathcal{L}_i(p, q_i)$ and $\nabla_{\sigma_i} \mathcal{L}_i(p, q_i)$

gradient ascent on μ_i, σ_i

Amortized Variational Inference

基本的变分估计有参数量大的问题——当datapoints变大时， $q_i(z)$对每一个i 都要学习两个分布参数 (mean & std)

我们可以进行改进，用一个deep neural network 学习一个大分布 $q_\phi(z|x)$ 其中输入为 x_i

What's the problem?

for each x_i (or mini-batch):

calculate $\nabla_\theta \mathcal{L}_i(p, q_i)$:

sample $z \sim q_i(z)$

$\nabla_\theta \mathcal{L}_i(p, q_i) \approx \nabla_\theta \log p_\theta(x_i|z)$

$\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}_i(p, q_i)$

update q_i to maximize $\mathcal{L}_i(p, q_i)$

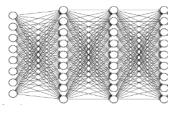
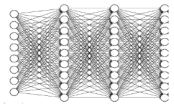
let's say $q_i(z) = \mathcal{N}(\mu_i, \sigma_i)$

use gradient $\nabla_{\mu_i} \mathcal{L}_i(p, q_i)$ and $\nabla_{\sigma_i} \mathcal{L}_i(p, q_i)$

gradient ascent on μ_i, σ_i

How many parameters are there? $|\theta| + (|\mu_i| + |\sigma_i|) \times N$

intuition: $q_i(z)$ should approximate $p(z|x_i)$ what if we learn a *network* $q_i(z) = q(z|x_i) \approx p(z|x_i)$?

z  $p_\theta(x|z)$ x  $q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$

Amortized variational inference

for each x_i (or mini-batch):

calculate $\nabla_\theta \mathcal{L}(p_\theta(x_i|z), q_\phi(z|x_i))$:

sample $z \sim q_\phi(z|x_i)$

$\nabla_\theta \mathcal{L} \approx \nabla_\theta \log p_\theta(x_i|z)$

$\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}$

$\phi \leftarrow \phi + \alpha \nabla_\phi \mathcal{L}$

$$q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$$

look up formula for entropy of a Gaussian

$$\mathcal{L}_i = E_{z \sim q_\phi(z|x_i)}[\log p_\theta(x_i|z) + \log p(z)] + \mathcal{H}(q_\phi(z|x_i))$$

$$J(\phi) = E_{z \sim q_\phi(z|x_i)}[r(x_i, z)]$$

can just use policy gradient!

What's wrong with this gradient?

$$\nabla J(\phi) \approx \frac{1}{M} \sum_j \nabla_\phi \log q_\phi(z_j|x_i) r(x_i, z_j)$$

这里解决的问题是怎么计算关于 ϕ 的梯度，在 $\mathcal{H}$ 中如果是Gussian分布可以直接用mean和std得到

所以只需要考虑 $\mathcal{L}$ 前面期望部分，这个期望部分和 ϕ 的联系是：采样与 ϕ 有关，但是计算式和 ϕ 无关，联想到policy gradient中也有类似的情况($J(\theta)$)可以使用蒙特卡洛估计来求解，但是就像在policy gradient笔记中提到的，这种方法方差较大

使用重参数化技巧：

将Gausssian分布重新写为两个由神经网络参数 ϕ 直接输出的参数 $\mu_\phi(x_i)$ $\sigma_\phi(x_i)$ 和随机采样的 ϵ 进行线性组合得到，这样可以直接对参数 ϕ 求梯度微分

注意这里的 $\frac{1}{M}$ 是对 M 个 ϵ 的期望的蒙特卡洛估计，而不是像policy gradient一样对 $J(\phi)$ 中的 $r(x_i, z)$ 或者 $\mathcal{L}_i$ 中的 $[\log p_\theta(x_i|z) + \log p(z)]$ 一样进行蒙特卡洛估计

The reparameterization trick

Is there a better way?

$$J(\phi) = E_{z\sim q_\phi(z|x_i)}[r(x_i, z)]$$
$$= E_{\epsilon\sim\mathcal{N}(0,1)}[r(x_i, \mu_\phi(x_i) + \epsilon\sigma_\phi(x_i))]$$

$$q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$$

$$z = \mu_\phi(x) + \epsilon\sigma_\phi(x)$$

$$\epsilon \sim \mathcal{N}(0,1)$$

independent of ϕ!

estimating $\nabla_\phi J(\phi)$:

sample $\epsilon_1, \ldots, \epsilon_M$ from $\mathcal{N}(0,1)$ (a single sample works well!)

$$\nabla_\phi J(\phi) \approx \frac{1}{M}\sum_j \nabla_\phi r(x_i, \mu_\phi(x_i) + \epsilon_j\sigma_\phi(x_i))$$

most autodiff software (e.g., TensorFlow) will compute this for you!

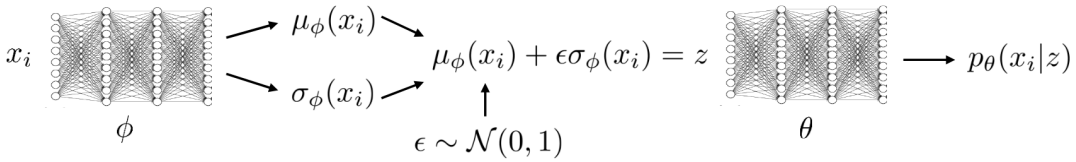
下图从网络流动的角度分析，其中KL散度对Gaussian分布有特定的数学表达式(用mean和std表示，所以可以直接求导)，而期望项使用重参数化技巧，最下面就是从 x_i 流动到 $p_\theta(x_i|z)$ 的过程

Another way to look at it…

$$\mathcal{L}_i = E_{z\sim q_\phi(z|x_i)}[\log p_\theta(x_i|z) + \log p(z)] + \mathcal{H}(q_\phi(z|x_i))$$
$$= E_{z\sim q_\phi(z|x_i)}[\log p_\theta(x_i|z)] + \underbrace{E_{z\sim q_\phi(z|x_i)}[\log p(z)] + \mathcal{H}(q_\phi(z|x_i))}$$
$$-D_{\mathrm{KL}}(q_\phi(z|x_i)\|p(z))$$

this often has a convenient analytical form (e.g., KL-divergence for Gaussians)

$$= E_{z\sim q_\phi(z|x_i)}[\log p_\theta(x_i|z)] - D_{\mathrm{KL}}(q_\phi(z|x_i)\|p(z))$$
$$= E_{\epsilon\sim\mathcal{N}(0,1)}[\log p_\theta(x_i|\mu_\phi(x_i) + \epsilon\sigma_\phi(x_i))] - D_{\mathrm{KL}}(q_\phi(z|x_i)\|p(z))$$
$$\approx \log p_\theta(x_i|\mu_\phi(x_i) + \epsilon\sigma_\phi(x_i)) - D_{\mathrm{KL}}(q_\phi(z|x_i)\|p(z))$$



总结：注意重参数化技巧只能在连续空间使用

Reparameterization trick vs. policy gradient

- Policy gradient
  - Can handle both discrete and continuous latent variables
  - High variance, requires multiple samples & small learning rates
- Reparameterization trick
  - Only continuous latent variables
  - Very simple to implement
  - Low variance

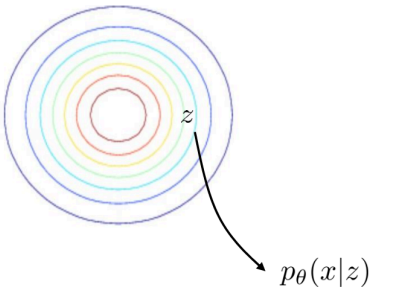
$$J(\phi) \approx \frac{1}{M} \sum_j \nabla_\phi \log q_\phi(z_j|x_i) r(x_i, z_j)$$

$$\nabla_\phi J(\phi) \approx \frac{1}{M} \sum_j \nabla_\phi r(x_i, \mu_\phi(x_i) + \epsilon_j \sigma_\phi(x_i))$$

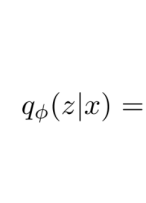
VAE

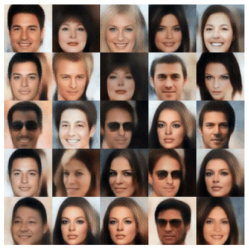
使用上文的amortized variational inference思想的实例就是VAE (variational autoencoder)
这是一个生成模型，有encoder和decoder组成，其中 $q_\phi(z|x)$ 是encoder，产生尽可能逼近真实后验分布 $p(z|x)$ 而 $p_\theta(x|z)$ 是decoder，针对产生的 z 生成尽可能真实的datapoint x
训练过程和上文一样，只不过这里可能 x 是图像的pixels， z 是一个潜在特征向量比如64dim 或128dim等等等

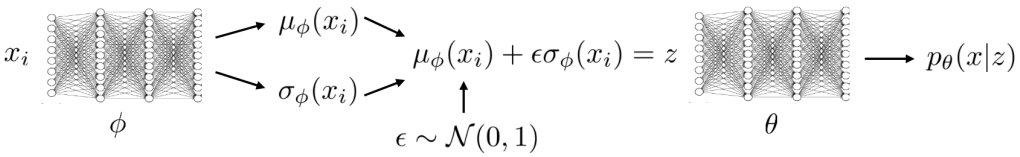
The *variational* autoencoder



$q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$ ϕ θ $p_\theta(x|z) = \mathcal{N}(\mu_\theta(z), \sigma_\theta(z))$







$$\max_{\theta,\phi} \frac{1}{N} \sum_i \log p_\theta(x_i|\mu_\phi(x_i) + \epsilon\sigma_\phi(x_i)) - D_{\mathrm{KL}}(q_\phi(z|x_i)\|p(z))$$

Variational Inference in Deep RL

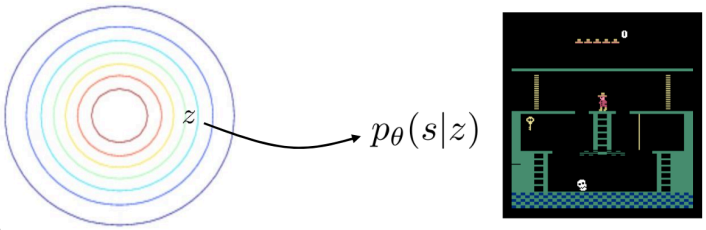
使用VAE来获得潜在空间中的特征向量 z 来更好的表示RL中原本的输入状态 s

s 包含了很多无用的信息，而使用VAE来提取更本质的潜在表示 z 进而更好的优化Q函数

Example applications

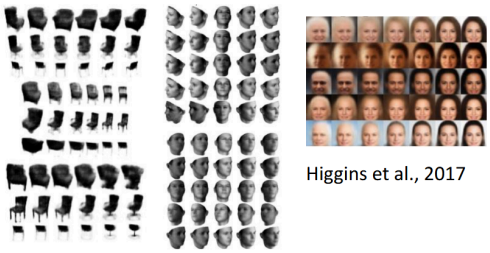
Representation learning

z is a *representation* of s



1. Train VAE on states in replay buffer $\mathcal{R}$
2. Run RL, using z as the state instead of s

Why is this a good idea?



Higgins et al., 2017

Sample algorithm

1. Collect transition (s, a, s', r), add to $\mathcal{R}$
2. Update $p_\theta(s|z)$ and $q_\phi(z|s)$ w/ batch from $\mathcal{R}$
3. Update $Q(z, a)$ w/ batch from $\mathcal{R}$

This also provides a great way to use prior data!

条件生成模型的潜在空间利用，

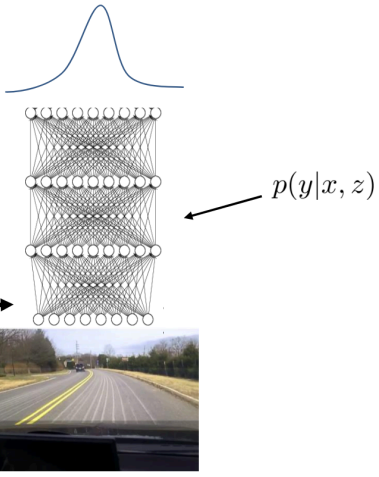
在给定某些条件 x 下的数据 y。例如，在自动驾驶中，我们想根据输入的车道线、路标等信息 x，预测出未来的场景 y

结合**强化学习框架**，我们可以用这个预测模型来训练一个策略。机器人可以通过“想象”（即从条件模型中采样不同的未来轨迹）来评估不同动作的后果，从而更好地做出决策。

Conditional models

$$\mathcal{L}_i = E_{z\sim q_\phi(z|x_i,y_i)}[\log p_\theta(y_i|x_i,z) + \log p(z|x_i)] + \mathcal{H}(q_\phi(z|x_i,y_i))$$

just like before, only now generating y_i
and *everything* is conditioned on x_i



at test time:

$z \sim p(z|x_i)$

$y \sim p(y|x_i, z)$