多智能体强化学习实训
Multi-agent Reinforcement Learning

Lecture 5:
Policy Gradient

教师：张寅 zhangyin98@zju.edu.cn

助教：邓悦 devindeng@zju.edu.cn

王子瑞 ziseoiwong@zju.edu.cn

李奕澄 yichengli@zju.edu.cn

浙江大学计算机学院

课程大纲

腾讯开悟

- **课程回顾**
- 策略梯度
- 策略梯度算法
- Actor-Critic算法

课程回顾

- **蒙特卡罗方法** $V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$
- **时序差分方法** $V(s_t) \leftarrow V(s_t) + \alpha\big(r_t + \gamma V(s_{t+1}) - V(s_t)\big)$
  - **SARSA方法**

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

  - **Q-learning方法**

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

课程回顾

- 基于随机梯度下降（SGD）的值函数近似
- 目标：最小化值函数近似值与真实值之间的均方误差

$$J(\theta) = \mathbb{E}_\pi[\frac{1}{2}(V^\pi(s) - V_\theta(s))^2]$$

- 误差减小的梯度方向

$$-\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_\pi[(V^\pi(s) - V_\theta(s))\frac{\partial V_\theta(s)}{\partial \theta}]$$

- 单次采样进行随机梯度下降

$$\theta \leftarrow \theta - \alpha\frac{\partial J(\theta)}{\partial \theta} = \theta + \alpha(V^\pi(s) - V_\theta(s))\frac{\partial V_\theta(s)}{\partial \theta}$$

课程回顾

- 将策略参数化$\pi_\theta(a|s)$
- 对于确定性的策略$\mathrm{a} = \pi_\theta(\mathrm{s})$
- 对于随机策略$\pi_\theta(\mathrm{a}|\mathrm{s}) = \mathrm{P}(\mathrm{a}|\mathrm{s}, \theta)$
  - θ是策略的参数
  - 将可见的已知状态泛化到未知的状态上

课程回顾

- 策略参数化的优点
  - 具有更好的收敛性质
  - 在高维度或连续的动作空间中更有效
  - 能够学习出随机策略

- 策略参数化的缺点
  - 通常会收敛到局部最优而非全局最优
  - 评估一个策略通常不够高效并具有较大的方差

课程大纲

- 课程回顾
- **策略梯度**
- 策略梯度算法
- Actor-Critic算法

Reinforce算法

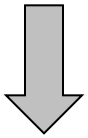
强化学习方法对比

基于价值的强化学习方法：

- 学习价值函数
- 利用价值函数导出策略
- 更高的样本训练效率
- 通常仅适用于具有离散动作的环境

基于策略的强化学习方法：

- 不需要价值函数
- 直接学习策略
- 在高维或连续动作空间场景中更加高效
- 适用任何动作类型的场景
- 容易收敛到次优解



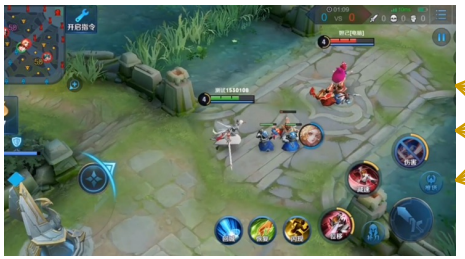
基于演员-评论家的方法：

- 结合两者优势
- 将在后面的课程详细介绍

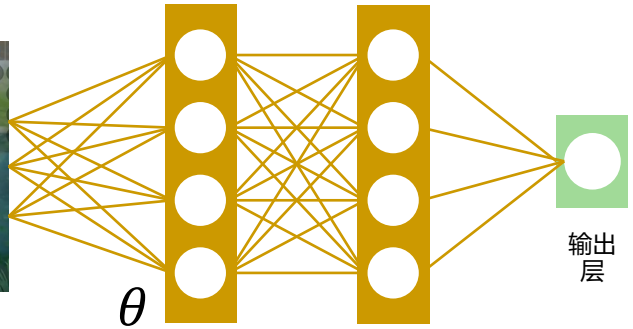
参数化策略

- 基于策略的强化学习方法方法直接搜索最优策略π^*
- 通常做法是参数化策略 π_θ，并利用无梯度或基于梯度的优化方法对参数进行更新

  ✓无梯度优化可以有效覆盖低维参数空间，但基于梯度的训练仍然是首选，因为其具有更高的采样效率



输入状态 s



使用一技能“将进酒”的概率

$\pi_\theta(a|s)$

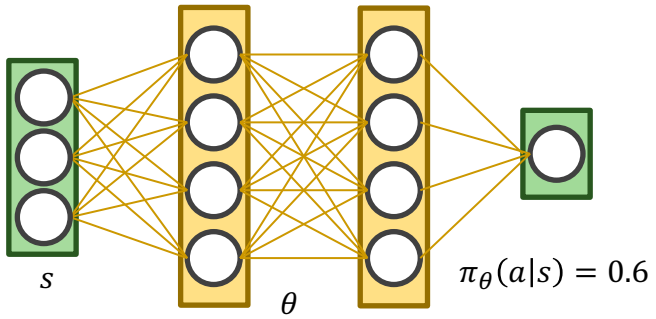
参数化策略与表格形策略

策略表征方式不同也会带来不同的性质：

	a_1	a_2
s_1	0.7	0.3
s_2	0.4	0.6

- 采取某个动作的概率的计算方式不同
  - 表格型策略：状态 s 上采取动作 a 的概率直接查表可得
  - 参数化策略：通过给定的函数结构和参数计算 $\pi_\theta(a|s)$

- 策略的更新方式不同
  - 表格型策略：直接修改表格中对应的条目
  - 参数化策略：通过更新参数 θ 对策略进行更新



参数化策略与表格形策略

最优策略的定义不同

- 表格型策略：称一个策略 π^* 是最优的，若它能最大化每个状态对应的值函数

$$\pi^*(s) = \text{argmax}_{a\epsilon\mathcal{A}} q_\pi(s,a), s\epsilon\mathcal{S}$$

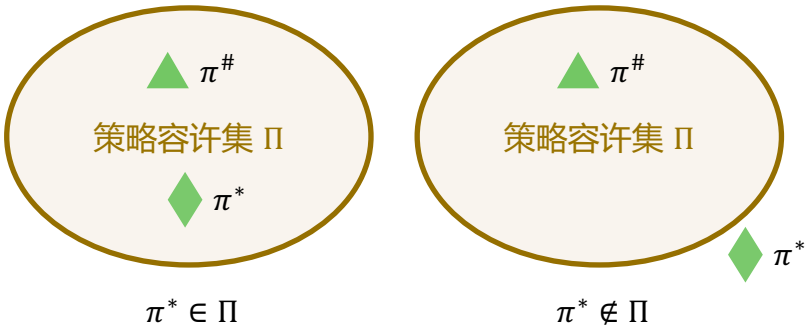
- 参数化策略：称一个策略 $\pi^\#$ 是最优的，若它能最大化一个给定的标量指标

$$\pi^\#(s) = \text{argmax}_\pi J\big(\pi(s)\big), \pi \in \Pi$$

- 将参数化策略 π_θ 所能表示的所有策略组成的集合记为策略容许集 $\Pi = \{\pi_\theta : \theta \in \Theta\}$， Θ 为参数 θ 的取值范围

✓若 $\pi^* \in \Pi$，则有 $J\left(\pi^\#\right) = J(\pi^*)$

✓若 $\pi^* \notin \Pi$，则策略 π^* 比 $\pi^\#$ 更优



基本思想

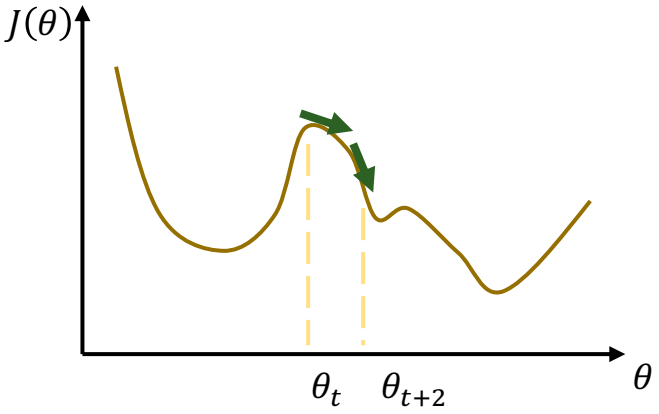
- 基本思想：
  - 利用目标函数定义策略优劣性：$J(\theta) = J(\pi_\theta)$
  - 对目标函数进行优化，以寻找最优策略

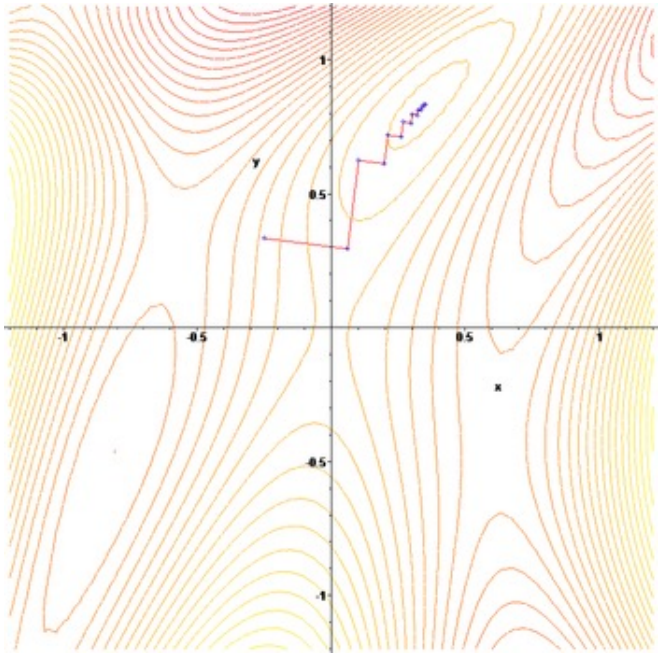
- 问题：
  - 目标函数 $J(\theta)$ 如何设计？
  - 该目标函数关于参数的优化方向（如梯度 $\nabla_\theta J(\theta)$）如何计算？

优化方向

1. 目标函数不可微分时：使用无梯度算法进行最优参数搜索

2. 目标函数可微分时：利用基于梯度的优化方法寻找最优策略 $\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta J(\theta_t)$





目标函数

1. 平均值目标：$\boldsymbol{J(\theta) = \mathbb{E}_s\left[v_{\pi_\theta}(s)\right] = \sum_{s\in\mathcal{S}} d(s) v_{\pi_\theta}(s)}$

2. 平均奖励目标：$\boldsymbol{J(\theta) = \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) r(s,a)}$

3. 平均轨迹回报目标：$\boldsymbol{J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_t r(s_t, a_t)\right]}$

- $d(s)$ 是状态分布，满足 $\sum_{s\in\mathcal{S}} d(s) = 1$
- p_θ 是轨迹的分布：$p_\theta(s_1, a_1, \ldots, s_T) = p(s_1)\prod_{t=1}^{T-1}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$
- 策略 π 越好（即参数 θ 越好），则对于所有状态 s，状态价值 $v_{\pi_\theta}(s)$ 的均值应当也越大
- 这些目标函数排除了状态 s 和动作 a 的因素，只依赖于策略参数 θ

状态分布

- 策略无关的状态分布
  - 在这类情况下，目标函数关于参数的梯度通常更好算
  - 一个简单的做法是取 $d(s)$ 为均匀分布，即每个状态都有相同的权重 $1/|\mathcal{S}|$
  - 另一种做法是把权重集中分配给一部分状态集合。例如，在一些任务中，一个回合只从状态 s_0 开始，那么可以设置为：$d(s_0)=1, d(s\neq s_0)=0$
- 策略相关的状态分布
- 在这种情况下，通常选用稳态状态分布
- $d(s)$ 是稳态状态分布：若对一个状态转移 $s\to a\to s'$，满足：

$$\boldsymbol{d(s')=\sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} p(s'|s,a)\cdot\pi_\theta(a|s)\cdot d(s)}$$

无梯度优化

- 基于策略的强化学习算法实际就是优化目标函数以寻找最优参数 $\theta^* = \text{argmax}\, J(\theta)$ 的方法
- 当目标函数不可微分或者难以微分时，通常可以使用一些无梯度的启发式优化方法：
  - 有限差分方法、交叉熵方法、遗传算法等

计算策略梯度

- 当目标函数可微分时，可以使用一些基于梯度的算法：如梯度下降法、拟牛顿法等
- 但在此之前，需要先计算出目标函数关于策略参数的梯度。

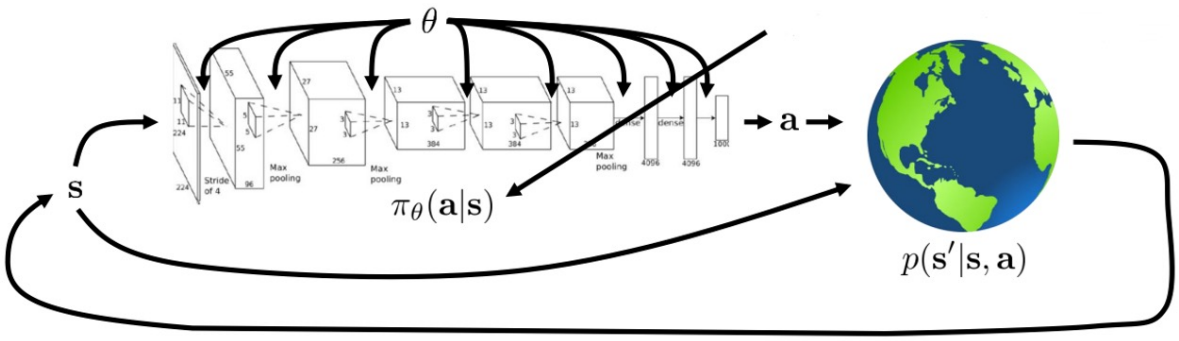
- **最大化平均轨迹回报目标函数：**

$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

  - τ 为策略 π_θ 采样而来的轨迹 $\{s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_T\}$

计算策略梯度

状态完全可观测



$$\boldsymbol{\theta^* = arg\,max_{\theta}\, \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_t r(s_t, a_t)\right]}$$

目标函数 $J(\theta)$

$$\boldsymbol{p_\theta(s_1, a_1, \ldots, s_T) = p(s_1)\prod_{t=1}^{T-1}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)}$$

计算策略梯度

- 记 $G(\tau)=\sum_t r(s_t,a_t)$，平均轨迹回报目标的策略梯度为：

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

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

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

$$\nabla_\theta J(\theta)=\int \frac{p_\theta(\tau)}{p_\theta(\tau)}\nabla_\theta p_\theta\, G(\tau)d\tau$$

$$\nabla_\theta J(\theta)=\int p_\theta(\tau)\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)}G(\tau)d\tau$$

$$\nabla_\theta J(\theta)=\int p_\theta(\tau)\nabla_\theta \log p_\theta(\tau)\,G(\tau)d\tau$$

$$\nabla_\theta J(\theta)=\mathbb{E}_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)\,G(\tau)]$$

计算策略梯度

关于$\nabla_\theta \log p_\theta(\tau)$的简化：

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

其中 $p(s_1)$，$p(s_{t+1}|s_t, a_t)$ 与参数 θ 无关，因此 $\nabla_\theta \log p(s_1) = 0$，$\nabla_\theta \sum_{t=1}^{T} \log p(s_{t+1}|s_t, a_t) = 0$

计算策略梯度

- 对于平均值目标函数 $J(\theta) = \sum_{s\in\mathcal{S}} d(s) v_{\pi_\theta}(s)$ 和平均奖励目标函数

$J(\theta) = \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) r(s,a)$，则有如下策略梯度：

$$\nabla_\theta J(\theta) \propto \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|s) q_\pi(s,a)$$
$$= \mathbb{E}_{s,a}\left[\nabla_\theta \ln \pi_\theta(a|s) q_\pi(s,a)\right]$$

$$\nabla_\theta \pi_\theta(a|s)$$
$$= \frac{\pi_\theta(a|s)}{\pi_\theta(a|s)} \nabla_\theta \pi_\theta(a|s)$$
$$= \pi_\theta(a|s) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}$$
$$= \pi_\theta(a|s) \nabla_\theta \ln \pi_\theta(a|s)$$

- 需要注意，由于需要计算 $\ln \pi_\theta(a|s)$，需要保证对所有的状态 s 和动作 a，有 $\pi_\theta(a|s) > 0$
  - 一个简单有效的方法是在网络最后输出层使用 softmax 函数，以保证所有动作有正概率值：

$$\pi_\theta(a|s) = \frac{exp(h_\theta(s,a))}{\sum_{a'\in\mathcal{A}} exp(h_\theta(s,a'))}$$

策略梯度优化

- 不同的目标函数可以得到如下类似的策略梯度：

$$\nabla_\theta J(\theta) = \mathbb{E}_{s,a}\left[\nabla_\theta \ln \pi_\theta(a|s) q_\pi(s,a)\right]$$

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

- 利用这些策略梯度，我们可以对策略进行优化：

$$\theta_{t+1} \leftarrow \theta_t + \nabla_\theta J(\theta)$$

- 使用不同的策略梯度，以及不同的近似方法，我们可以得到各种各样的基于策略梯度的强化学习算法，如REINFORCE、DDPG、PPO等等，这些方法将在后续课程中进行介绍

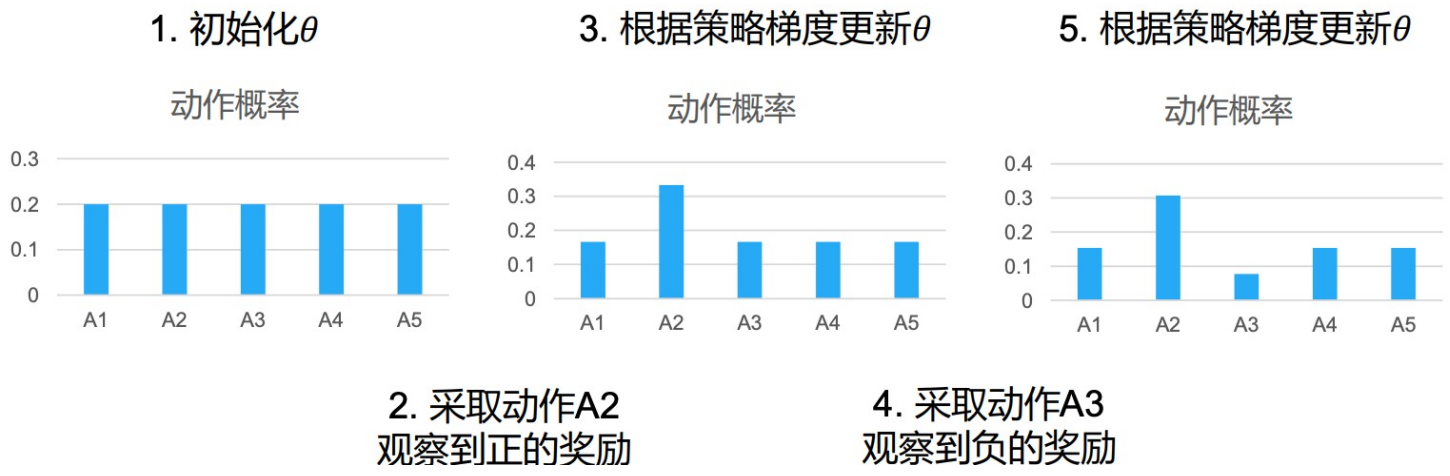
课程大纲

腾讯开悟

- 课程回顾
- 策略梯度
- **策略梯度算法**
- Actor-Critic算法

Reinforce算法

- 对于随机策略$\pi_\theta(a|s) = P(a|s,\theta)$
- 直觉上我们应该
  - 降低带来低价值/奖励的动作出现的概率
  - 提高带来高价值/奖励的动作出现的概率
- 一个离散动作空间维度为5的例子



Reinforce算法

基于蒙特卡洛法计算策略梯度

- 上一章中我们推导出了策略梯度：

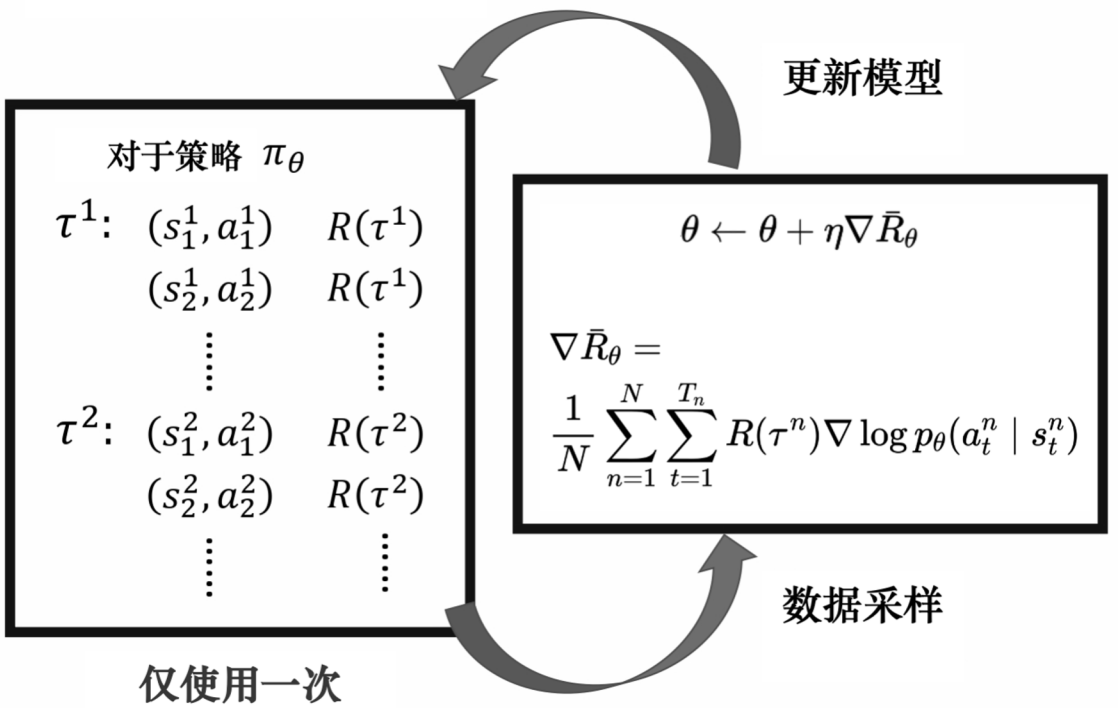
$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta log\, \pi_\theta(a_t|s_t)\, R(\tau) \right]$$

- 在实践中，我们可以通过蒙特卡洛方法进行估计

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T^n} R(\tau^n) \nabla_\theta log\, \pi_\theta(a_t^n|s_t^n)$$

- 据此，我们可以得到 REINFORCE 算法

Reinforce算法



REINFORCE 算法：

1. 利用策略 $\pi_\theta(a|s)$ 采样轨迹 $\{\tau_i\}$
2. 计算梯度 $\nabla_\theta J(\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T^n} R(\tau^n) \nabla_\theta \log \pi_\theta(a_t^n | s_t^n)$
3. 更新参数 $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

分类问题？

强化学习与分类问题对比：

- **都是输入状态，输出要采取的行为**
- 分类问题（监督学习）：假设有带标签的训练数据，随后利用极大似然法进行优化

s_t, a_t

$\{(s_t, a_t)\}$

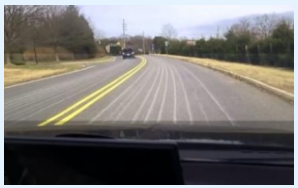
监督学习

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$
$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T^n} \nabla_\theta \log p_\theta(a_t^n | s_t^n)$$

- 强化学习：没有标签，只能通过试错的方式与环境交互获取奖励，以替代监督信息进行训练。

s_t, a_t

r_t, s_{t+1}



$\{(s_t, a_t, r_t, s_{t+1})\}$

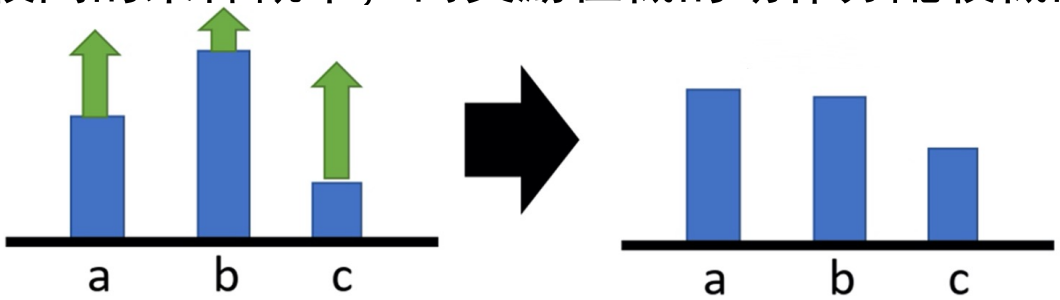
强化学习

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$
$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T^n} R(\tau^n) \nabla_\theta \log p_\theta(a_t^n | s_t^n)$$

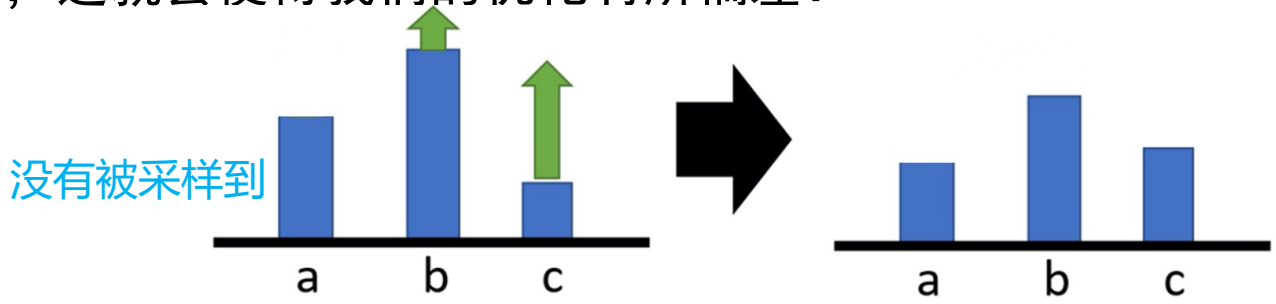
添加baseline

训练可能存在偏差：

- 考虑只有正奖励的场景，在理想情况下进行优化，最终能够使得奖励值高的动作分配较高的采样概率，而奖励值低的动作分配较低的采样概率：



- 但在实际场景中我们可能并不能采样到所有的动作，如只能采样到 b 和 c ，这就会使得我们的优化有所偏差：



添加baseline

- 我们可以将奖励函数减去一个基线 b，使得 $R(\tau)-b$ 有正有负
  - ✓ 如果 $R(\tau)>b$，就让采取对应动作的概率提升
  - ✓ 如果 $R(\tau)<b$，就让采取对应动作的概率降低

$$\nabla R_\theta = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T^n}(R(\tau^n)-b)\nabla_\theta log\, p_\theta(a_t^n|s_t^n)$$

- 一般来说，可以使用奖励的平均值作为基线，$b=\frac{1}{N}\sum_{i=1}^{N}R(\tau)$。
  - ✓ 在训练时记录 $R(\tau)$ 的值，并维护 $R(\tau)$ 的平均值

添加baseline

- 减去一个基线并不会影响原梯度的期望值

$$\nabla R_\theta = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T^n}(R(\tau^n) - b)\nabla_\theta log\, p_\theta(a_t^n|s_t^n)$$

$$\begin{aligned}
\mathbb{E}[\nabla_\theta\, log\, p_\theta(\tau)b] &= \int p_\theta(\tau)\nabla_\theta \log p_\theta(\tau)\mathrm{b}\, d\tau \\
&= \int \nabla_\theta p_\theta(\tau) b d\tau \\
&= b\int \nabla_\theta p_\theta(\tau) d\tau \\
&= b\nabla_\theta \int p_\theta(\tau) d\tau \\
&= b\nabla_\theta 1 = 0
\end{aligned}$$

实现策略梯度

- 利用自动求导工具实现策略梯度算法

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T^n} R(\tau^n)\nabla_\theta log\, \pi_\theta(a_t^n|s_t^n)$$

→ 难以计算

- 在实际应用时，都会采用Pytorch、Tensorflow 中的自动求导工具辅助求解
- 参照分类问题的损失函数
  - ✓ 分类问题的目标函数（极大似然法）：$J(\theta) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T^n} \log p_\theta(a_t^n|s_t^n)$
  - ✓ 策略梯度的目标函数：$J(\theta) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T^n} R(\tau^n)\log \pi_\theta(a_t^n|s_t^n)$
- 将策略梯度的目标函数视为极大似然法的目标函数一个利用累积奖励进行加权的版本。

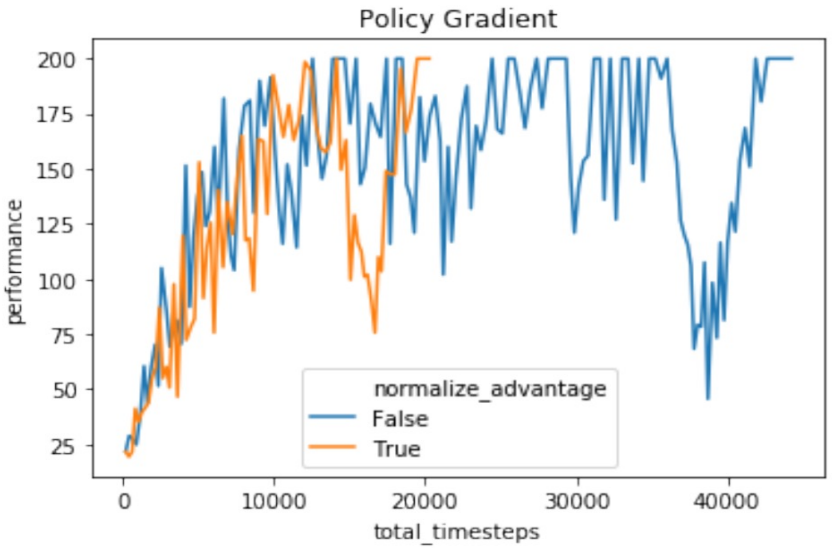
策略梯度存在问题

策略梯度算法在样本利用率以及稳定性上存在缺陷：

- 由于策略梯度算法为同策略算法，因此样本利用率较低

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

- 较大的策略更新或不适宜的更新步长会导致训练的不稳定
  - ✓ 在监督学习中，训练数据具有独立同分布的性质
  - ✓ 而在强化学习中，不适宜的更新步长 → 坏策略 → 低质量的数据
  - ✓ 可能难以从糟糕的策略中恢复，进而导致性能崩溃



离策略梯度

策略梯度是同策略算法，估计策略梯度所用的数据需要由当前策略采集

$$\boldsymbol{\theta^* = arg\,max_{\theta} J(\theta)}$$

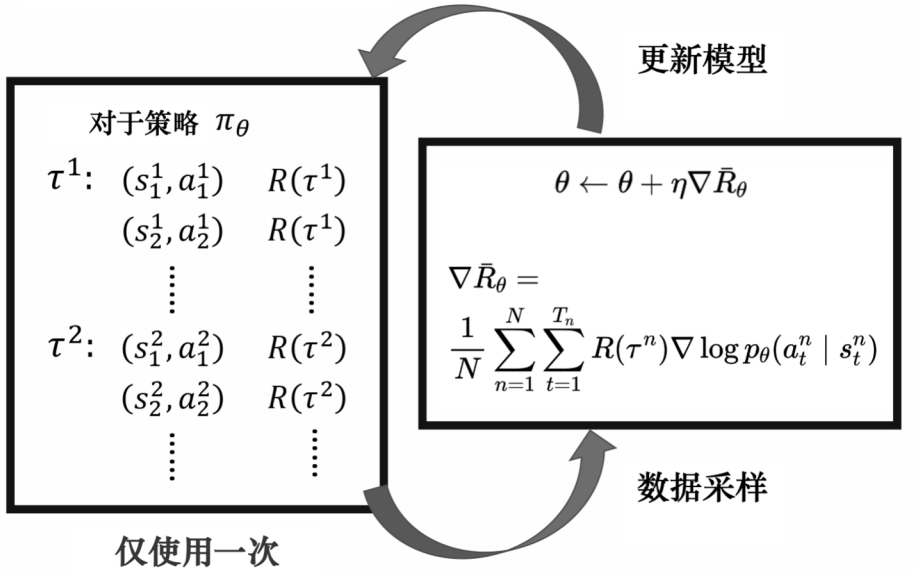
$$\boldsymbol{J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[r(\tau)]}$$

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

REINFORCE 算法：

1. 利用策略 $\pi_\theta(a|s)$ 采样轨迹 $\{\tau_i\}$
2. 计算梯度 $\nabla_\theta J(\theta) =$

   $\frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n)\nabla_\theta \log \pi_\theta(a_t^n|s_t^n)$
3. 更新参数 $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$



每次更新只能使参数发生很小的改变，直接应用梯度下降方法相当低效

离策略梯度

根据重要性采样利用异策略样本

$$\theta^* = arg\,\max_{\theta} J(\theta)$$
$$J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}[r(\tau)]$$

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

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

$$\frac{p_\theta(\tau)}{p_{\theta'}(\tau)} = \frac{p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)}{p(s_1)\prod_{t=1}^{T}\pi_{\theta'}(a_t|s_t)p(s_{t+1}|s_t,a_t)}$$

$$= \frac{\prod_{t=1}^{T}\pi_\theta(a_t|s_t)}{\prod_{t=1}^{T}\pi_{\theta'}(a_t|s_t)}$$

重要性采样

$$\mathbb{E}_{X\sim P}[f(X)] = \sum P(X)f(X)$$
$$= \sum Q(X)\frac{P(X)}{Q(X)}f(X)$$
$$= \mathbb{E}_{X\sim Q}[\frac{P(X)}{Q(X)}f(X)]$$

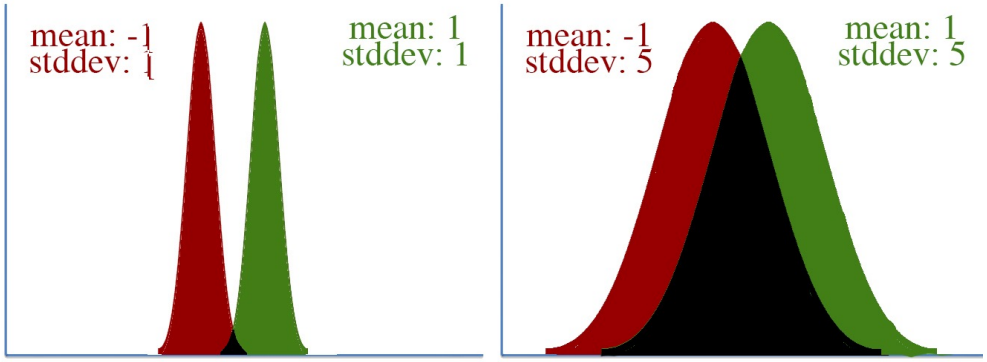
自然策略梯度

- 假设现在我们拥有一个神经网络与一个损失函数
- 梯度下降方法使参数朝着梯度方向发生微小改变，使得损失函数发生尽可能大的变化
- 如何定义“微小改变”？
  - ✓ 在标准梯度下降中，基于欧式距离度量改变量

$$distance = sqrt(a^2 + b^2)$$

- 考虑以下两对高斯分布
  - ✓ 同样是将均值从 −1 调整到 1 ，

    左图分布的改变要明显大于右图



自然策略梯度

$$d^* = arg\max_d J(\theta + d), s.t. KL(\pi_\theta || \pi_{\theta+d}) = c$$

- KL 散度用于衡量两个分布的接近程度

$$KL(\pi_\theta || \pi_{\theta'}) = \mathbb{E}_{\pi_\theta}[log\, \pi_\theta] - \mathbb{E}_{\pi_{\theta'}}[log\, \pi_{\theta'}]$$

- 将更新前后策略的 KL 散度限定为一个常数 c，确保策略分布以一个常量速度更新，而不受策略的参数化形式影响。
- 可以将 KL 散度进行二阶泰勒展开：

$$KL(\pi_\theta || \pi_{\theta+d}) \approx \frac{1}{2} d^T F d$$

其中 F 为 **Fisher Information Matrix**，即 $\mathbb{E}_{\pi_\theta}[\nabla log\, \pi_\theta \nabla log\, \pi_\theta{}^T]$

自然策略梯度

$$d^* = arg\max_d J(\theta + d), s.t. KL(\pi_\theta || \pi_{\theta+d}) = c$$

- 基于拉格朗日乘数法，并使用一阶泰勒展开对 $J(\theta + d)$ 进行近似，代入 KL 散度的二阶泰勒展开近似，可转化为如下形式，

$$\begin{aligned} d^* &= arg\max_d J(\theta + d) - \lambda(KL(\pi_\theta || \pi_{\theta+d}) - c) \\ &\approx arg\max_d J(\theta) + \nabla_\theta J(\theta)^T d - \frac{1}{2}\lambda d^T F d + \lambda c \end{aligned}$$

- 为了求解该最大化问题，可以令上式关于 d 的导数为 0，得到 natural policy gradient：

$$d = \frac{1}{\lambda} F^{-1} \nabla_\theta J(\theta)$$

- Natural policy gradient 是一种二阶优化方法，更新更加准确。此外，其更新效果也与模型的参数化形式无关

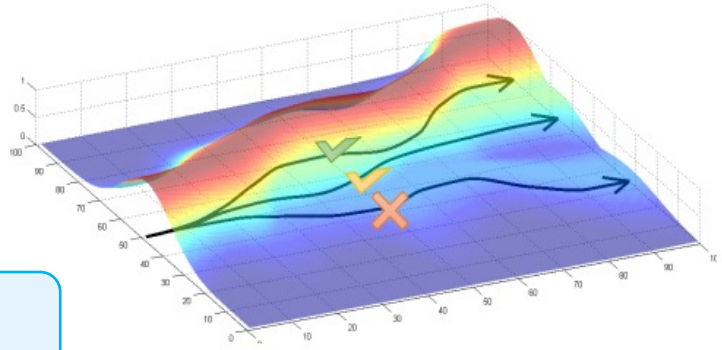
$$\theta_{t+1} = \theta_t + \alpha F^{-1} \nabla_\theta J(\theta)$$

课程大纲

腾讯开悟

- 课程回顾
- 策略梯度
- 策略梯度算法
- **Actor-Critic算法**

回顾Reinforce

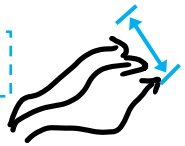


1. 使用当前策略 π_θ 采样N条经验轨迹 $\{\tau_i\}$

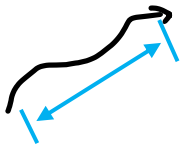
对第t步做对数极大似然估计

2. 计算梯度 $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\left(\sum_{t'=t}^{T} r(s_{t'}^i, a_{t'}^i)\right)\right)$

对N条轨迹进行平均



在每一条轨迹内累计回报



3. 更新策略参数 $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

改进策略

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

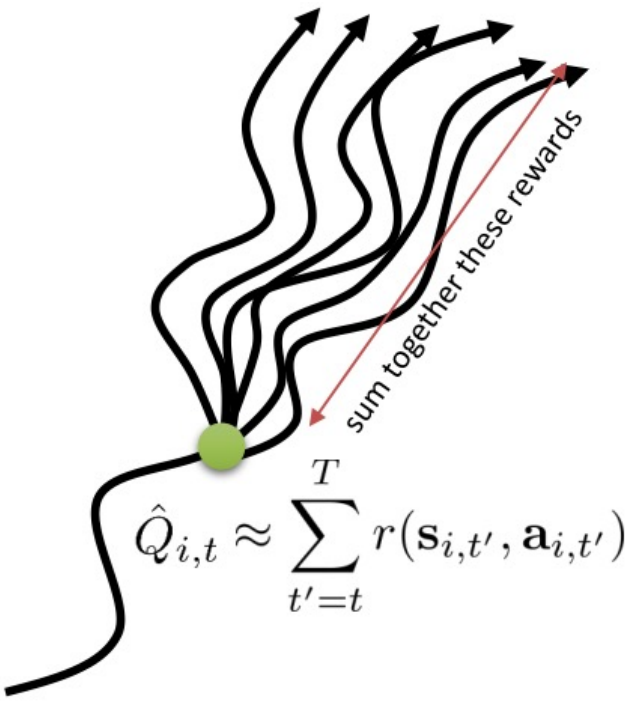
改进

方差较大

使用动作价值估计$\hat{Q}^\pi$近似轨迹回报期望$\mathbb{E}_{\pi_\theta}\ \sum_{t'=t}^{T} r(s_{t'}^i, a_{t'}^i)$：

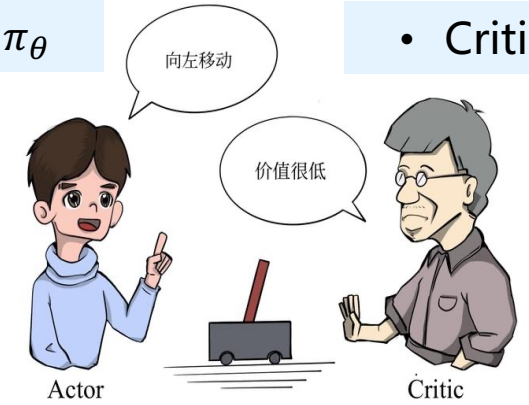
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \hat{Q}^\pi(s_t^i, a_t^i)\right)$$

$$\hat{Q}^\pi(s_t^i, a_t^i) \approx \sum_{t'=t}^{T}\mathbb{E}_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t, a_t]$$

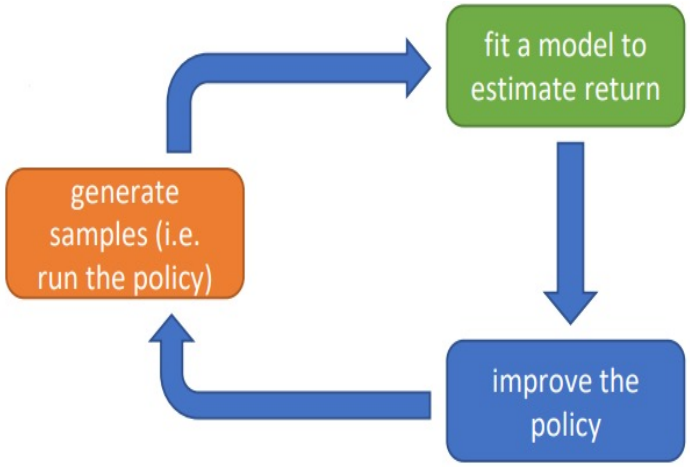


Actor-Critic方法

- Actor: π_θ
- Critic: $\hat{Q}^\pi$



$$\hat{Q}^\pi(s_{t'}^i, a_{t'}^i) = \sum_{t'=t}^{T} r(s_{t'}^i, a_{t'}^i)$$



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

1. 使用当前策略 π_θ 在环境中进行采样

2. 策略提升：$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\hat{Q}^\pi(s_t^i, a_t^i)\right)$

对第t步做对数极大似然估计

Actor Learning

对N条轨迹进行平均

动作值函数估计(Critic)

3. .拟合当前策略的动作值函数：$\hat{Q}^\pi(s_{t'}^i, a_{t'}^i) \approx \sum_{t'=t}^{T} r(s_{t'}^i, a_{t'}^i)$

Critic Learning

Advantage Actor-Critc

- Actor-Critic算法的策略梯度

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\hat{Q}^\pi(s_t^i,a_t^i)\right)$$

- 进一步对策略梯度进行改进

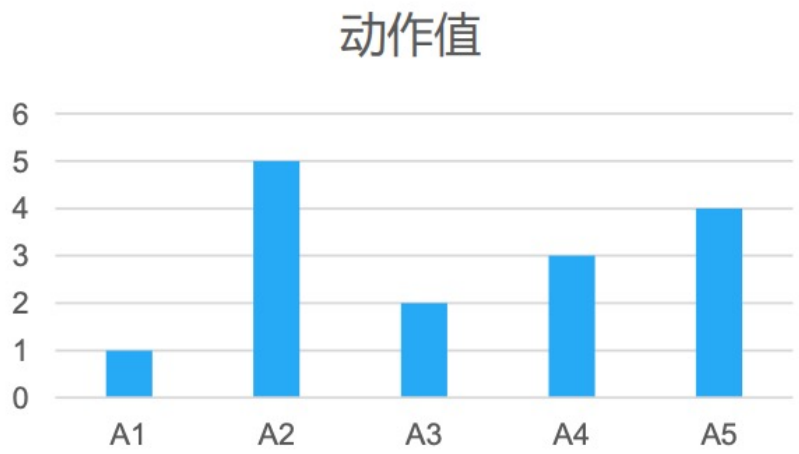
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\hat{A}^\pi(s_t^i,a_t^i)\right)$$

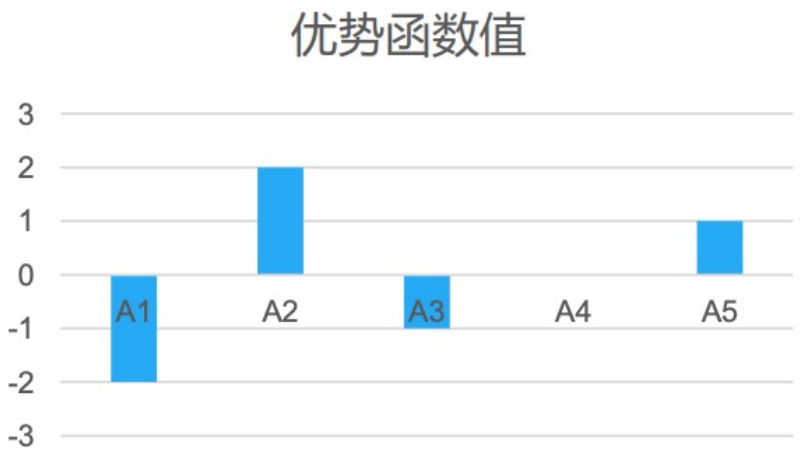
“优势函数”：$\hat{A}^\pi(s_t^i,a_t^i) = \hat{\mathrm{Q}}^\pi(s_t^i,a_t^i) - \hat{\mathrm{V}}^\pi(s_t^i)$

A2C算法

- 思想：通过减去一个基线值来标准化评论家的打分
  - 降低较差动作概率，提高较优动作概率
  - 进一步降低方差
- 优势函数（Advantage Function）

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





神经网络拟合

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\hat{A}^\pi(s_t^i,a_t^i)\right)$$
$$=\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)[\hat{Q}^\pi(s_t^i,a_t^i)-\hat{V}^\pi(s_t^i)]\right)$$

- 用两个神经网络分别拟合 $\hat{Q}^\pi$ 和 $\hat{V}^\pi$ ？

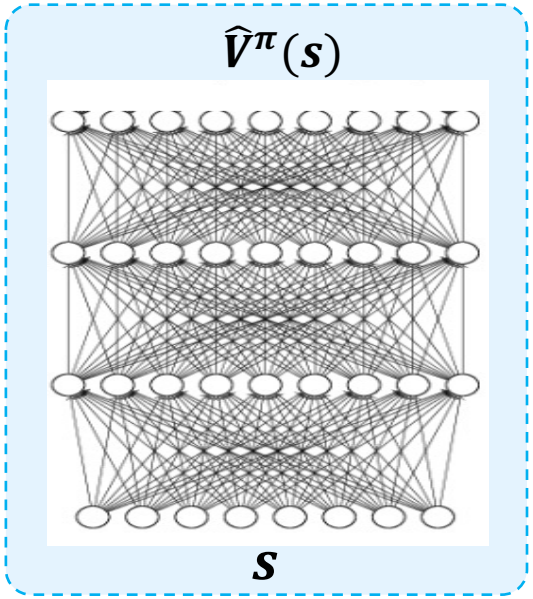
$$Q^\pi(s_t,a_t)=\mathbb{E}_\pi[R(s_t,a_t)+\gamma V(s_{t+1})]$$

神经网络拟合

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)[\hat{Q}^\pi(s_t^i,a_t^i)-\hat{V}^\pi(s_t^i)]\right)$$

$$\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)[R(s_t^i,a_t^i)+\gamma\hat{V}^\pi(s_{t+1}^i)-\hat{V}^\pi(s_t^i)]\right)$$

- 只需用一个神经网络拟合 $\hat{V}^\pi$



神经网络拟合

深度A2C算法

问题：状态值估计$\hat{V}^{\pi}$能否与策略 π_θ 共用网络？

1. 使用当前策略 π_θ 采样N条经验轨迹 $\{\tau_i\}$

2. 估计优势函数：$\hat{A}^{\pi}(s_t^i, a_t^i) = R(s_t^i, a_t^i) + \gamma\hat{V}^{\pi}(s_{t+1}^i) - \hat{V}^{\pi}(s_t^i)$

Advantage Computing

3. 策略提升：$\theta \leftarrow \theta + \alpha\frac{1}{N}\sum_i\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\hat{A}^{\pi}(s_t^i, a_t^i)\right)$

对第t步做对数极大似然估计

对N条轨迹进行平均

优势函数估计

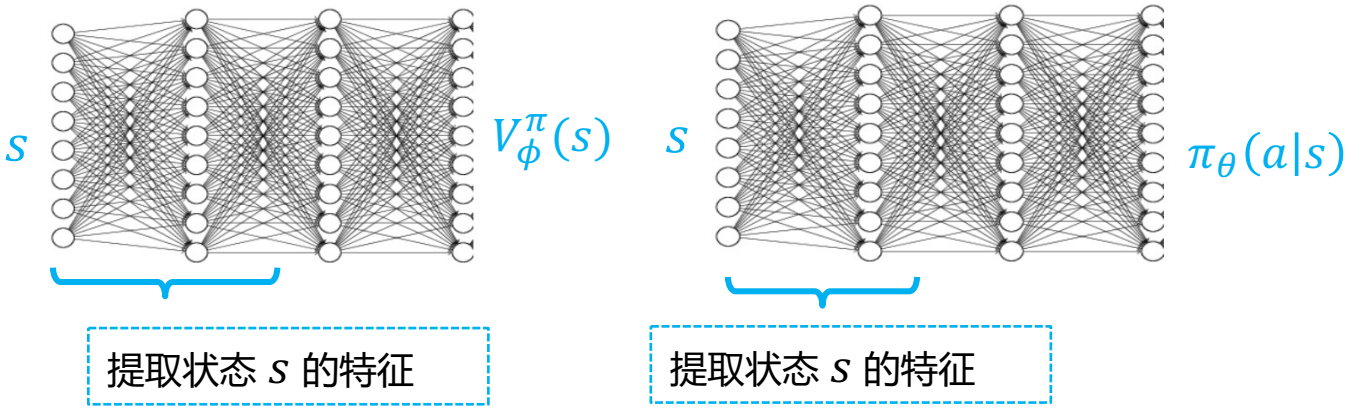
Actor Learning

4. 拟合当前策略的状态值函数：$\hat{V}^{\pi}(s_t) \approx \sum_{t'=t}^{T}\mathbb{E}_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t]$

Critic Learning

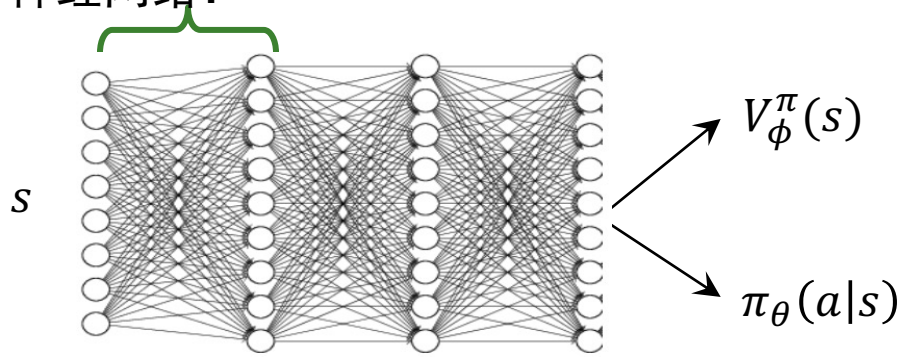
神经网络拟合

方案一：实现两个神经网络分别对价值函数$\hat{V}^{\pi}$和策略π_θ进行拟合：



- 优点：简单且训练稳定
- 缺点：训练效率较低

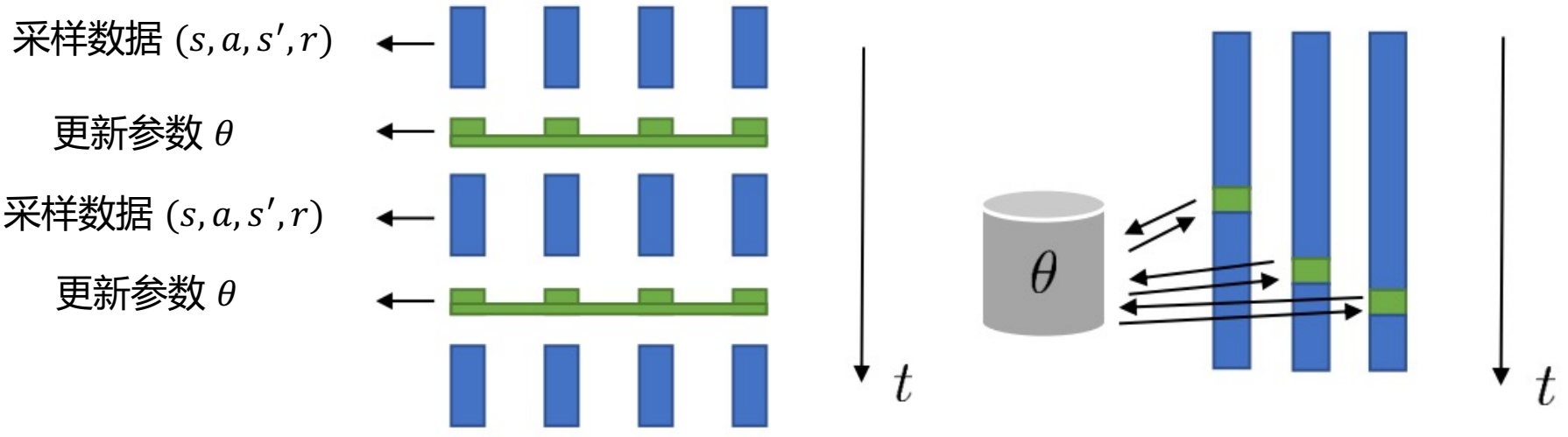
方案二：共享前几层特征提取网络，只使用一个神经网络：



- 优点：减小了参数量，因此训练效率高
- 缺点：由于策略与值所需的状态特征有一定差异，因此训练稳定性上要差于方案一

批量更新

- 问题：利用单个样本进行更新：更新方差较大，训练稳定性差
- 解决方案：获得一个批次的数据后再进行更新，分为同步和异步两种方法



同步更新　　异步更新

批量更新

批量更新的A2C算法

1. 采取动作 $a \sim \pi_\theta(a|s)$ ，与环境交互得到样本数据 (s, a, s', r) ，并保存至经验缓存 $\mathcal{R}$
2. 从经验缓存 $\mathcal{R}$ 中采样一个批次的数据 $\{s_i, a_i, r_i, s_i'\}$
3. 使用目标 $r_i + \gamma V_\phi^\pi(s_i')$ 更新价值函数 $V_\phi^\pi(s_i)$
4. 估算优势函数 $\hat{A}^\pi(s_i, a_i) = r(s_i, a_i) + \gamma V_\phi^\pi(s_i') - V_\phi^\pi(s_i)$
5. 估算策略梯度 $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i\left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta\left(a_t^i|s_t^i\right)\hat{A}^\pi(s_t^i, a_t^i)\right)$
6. 更新策略 $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

- 优点：充分利用旧策略采集的数据，减小同步时间，提高样本利用率

异步学习？

- 缺点：由于 $V_\phi^\pi(s_i')$ 中的 s_i' 并非当前策略 π_θ 所收集，无法准确估算优势值 A^π

改进批量更新

改进后的批量A2C算法

1. 采取动作 $a \sim \pi_\theta(a|s)$ ，与环境交互得到样本数据 (s, a, s', r)，并保存至经验缓存 $\mathcal{R}$
2. 从经验缓存 $\mathcal{R}$ 中采样一个批次的数据 $\{s_i, a_i, r_i, s_i'\}$
3. 使用目标 $r_i + \gamma Q_\phi^\pi(s_i', a_i')$ 更新价值函数 $Q_\phi^\pi(s_i', a_i')$
4. 从当前策略采样：$a_i^\pi \sim \pi_\theta(\cdot|s_i)$
5. 计算策略梯度 $\nabla_\theta J(\theta) \approx$

$$\frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(a_i^\pi|s_i) Q_\phi^\pi(s_i, a_i^\pi)$$

6. 更新策略 $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

对离策略问题的**解决方案**：在估值时避免使用经验池里的动作a_t^i和下一步状态s_i'：

- 将 $\hat{A}^\pi(s_t^i, a_t^i)$，即$r(s_i, a_i) + \gamma V_\phi^\pi(s_i') - V_\phi^\pi(s_i)$，换成$Q_\phi^\pi(s_t^i, a_t^i)$
- 将$Q_\phi^\pi(s_t^i, a_t^i)$中的a_t^i换成$a_i^\pi \sim \pi_\theta(\cdot|s_i)$，即根据状态$s_i$重新从当前策略中采样

A3C算法

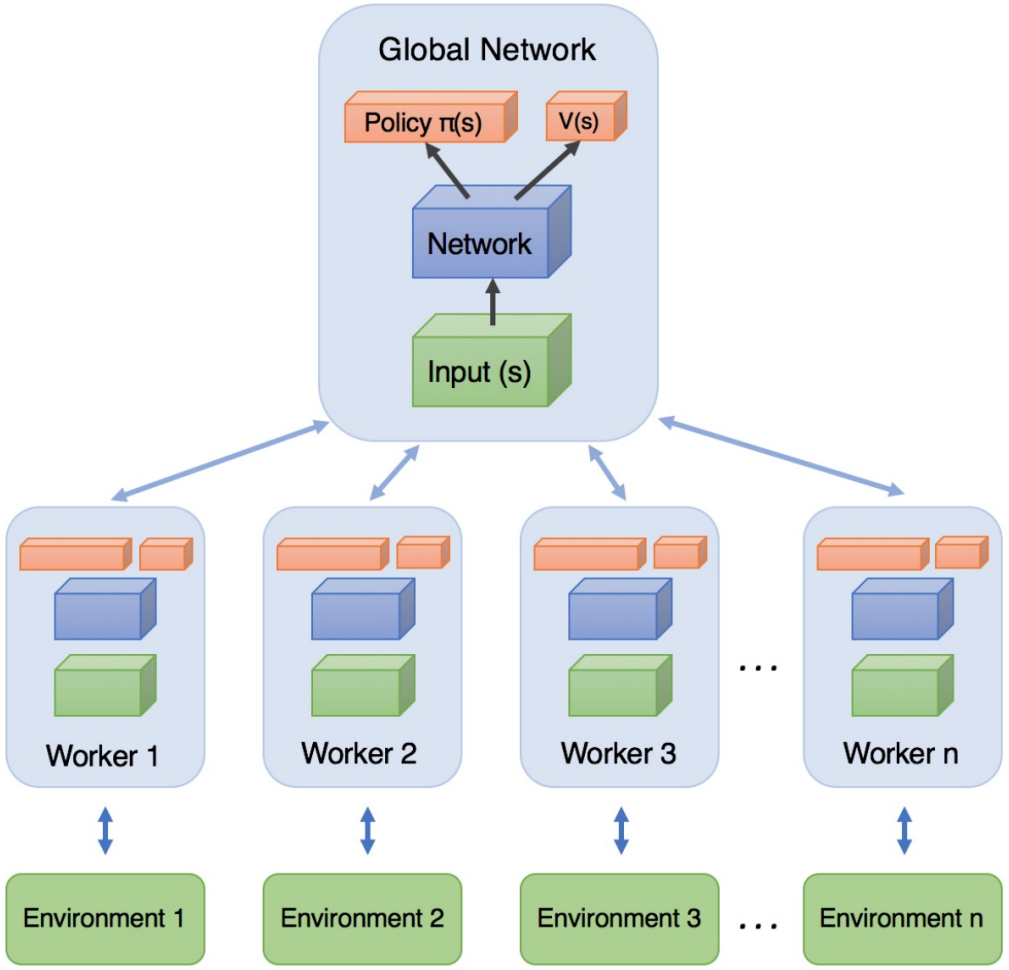
- A3C代表异步（Asynchronous）A2C
  - 该算法涉及并行执行一组异步并行的环境
  - 与A2C一样使用优势函数

$$\nabla_{\theta'} \log \pi(a_t|s_t;\theta') A(s_t,a_t;\theta_v)$$

$$A(s_t,a_t;\theta_v) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v)$$

A3C算法

- 异步的Actor-Critic方法能够充分利用多核CPU资源采样环境的经验数据，利用GPU资源异步地更新网络，这有效提升了强化学习算法的训练效率



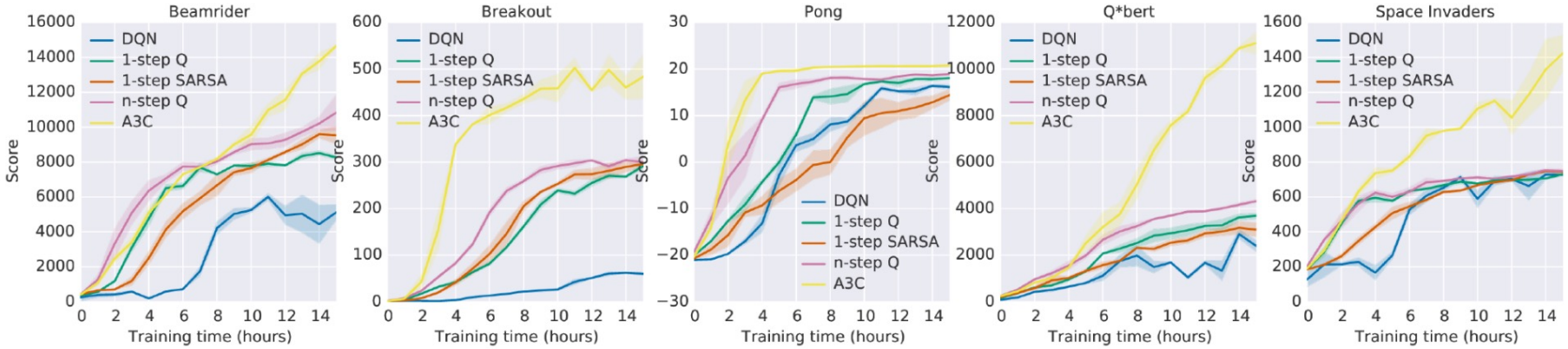
A3C算法

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ *and* θ_v *and global shared counter* $T = 0$
// Assume thread-specific parameter vectors θ' *and* θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial \left(R - V(s_i;\theta'_v)\right)^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

A3C算法

- A3C对比实验



Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

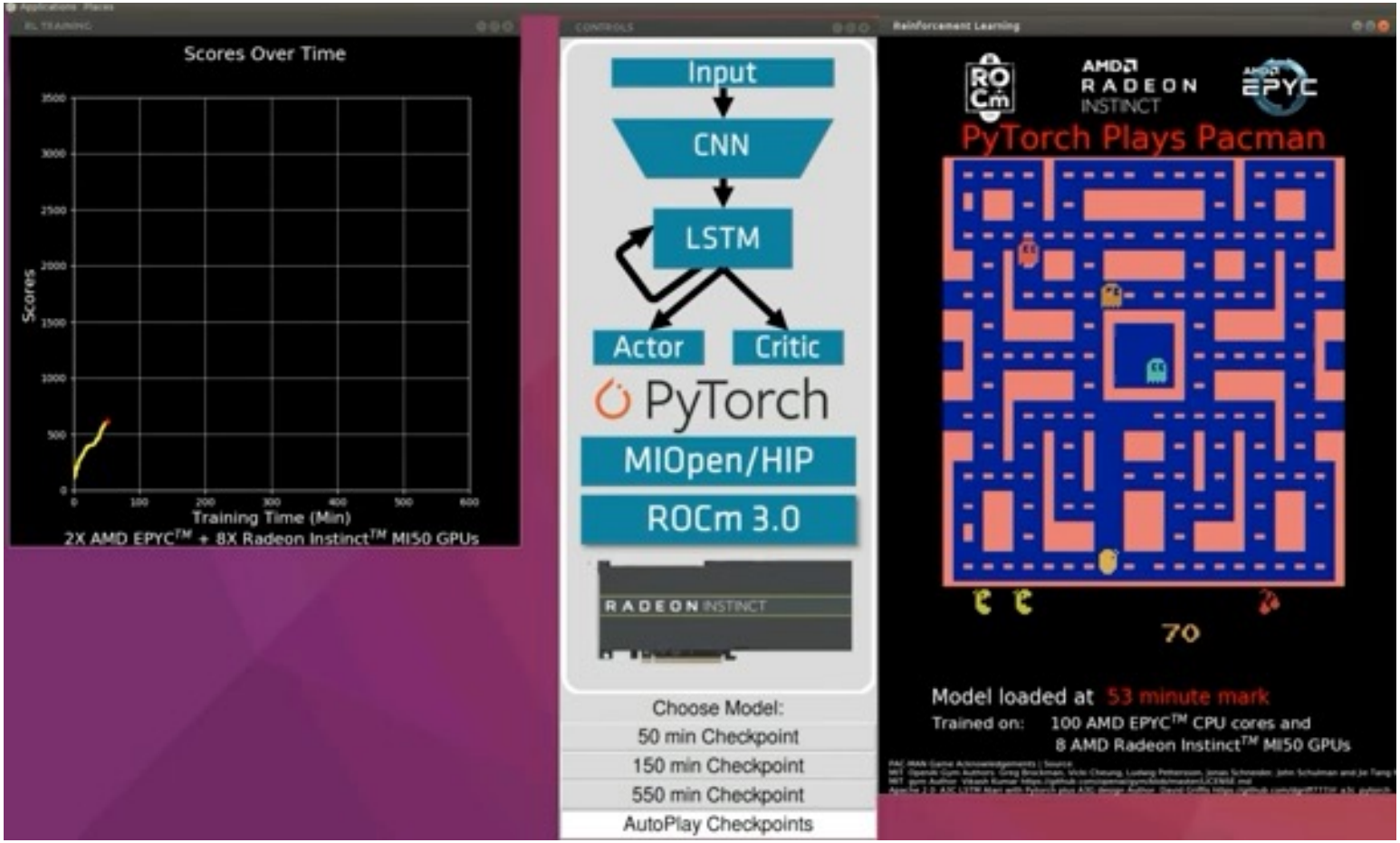
Nvidia K40 GPUs

16 CPU cores and no GPU

Mean and median human-normalized scores on 57 Atari games

A3C算法

- 实验案例


Scores Over Time
Scores
Training Time (Min)
2X AMD EPYC™ + 8X Radeon Instinct™ MI50 GPUs
Input
CNN
LSTM
Actor
Critic
PyTorch
MIOpen/HIP
ROCm 3.0
RADEON INSTINCT
Choose Model:
50 min Checkpoint
150 min Checkpoint
550 min Checkpoint
AutoPlay Checkpoints
AMD RADEON INSTINCT
EPYC
PyTorch Plays Pacman
70
Model loaded at 53 minute mark
Trained on: 100 AMD EPYC™ CPU cores and
8 AMD Radeon Instinct™ MI50 GPUs

A3C算法

- 实验案例

提问环节

谢 谢