多智能体系统与强化学习

主讲人：高阳、杨林、杨天培

https://reinforcement-learning-2025.github.io/

第六讲：深度强化学习

端到端学习

杨　林

大 纲

深度强化学习概述

Deep Q-learning（DQN）

Proximal Policy Optimization (PPO)

Asynchronous Advantage Actor-Critic (A3C)

大 纲

深度强化学习概述

Deep Q-learning（DQN）

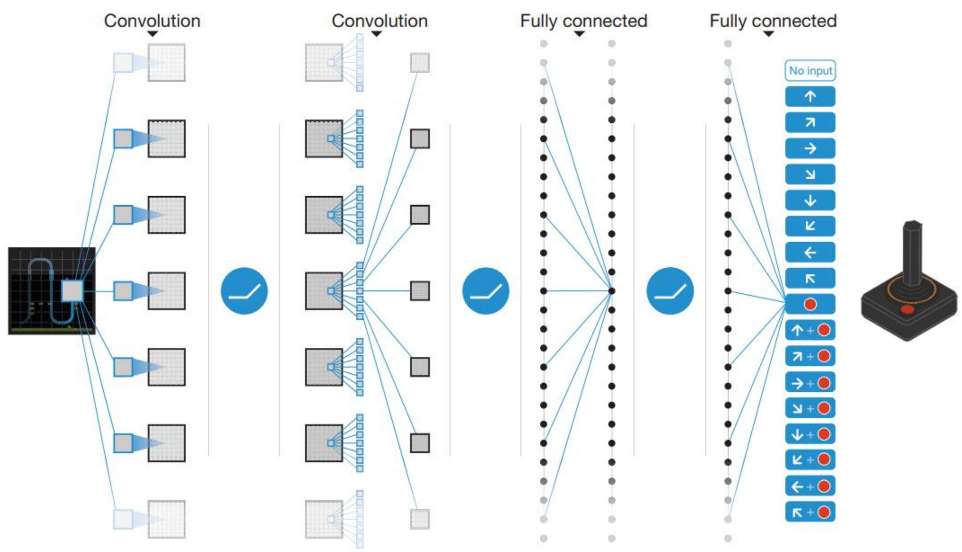
Proximal Policy Optimization (PPO)

Asynchronous Advantage Actor-Critic (A3C)

深度强化学习

◻ 深度强化学习:

- ✓ 利用深度神经网络进行价值函数和策略近似
- ✓ 从而使强化学习算法能够以端到端的方式解决复杂问题



深度Q网络直接从原始像素输入中学习游戏策略，在Atari 2600的多个游戏中超越人类玩家水平，标志着深度强化学习的正式诞生

Volodymyr Mnih, Koray Kavukcuoglu, David Silver et al.
Playing Atari with Deep Reinforcement Learning. NIPS 2013 workshop.

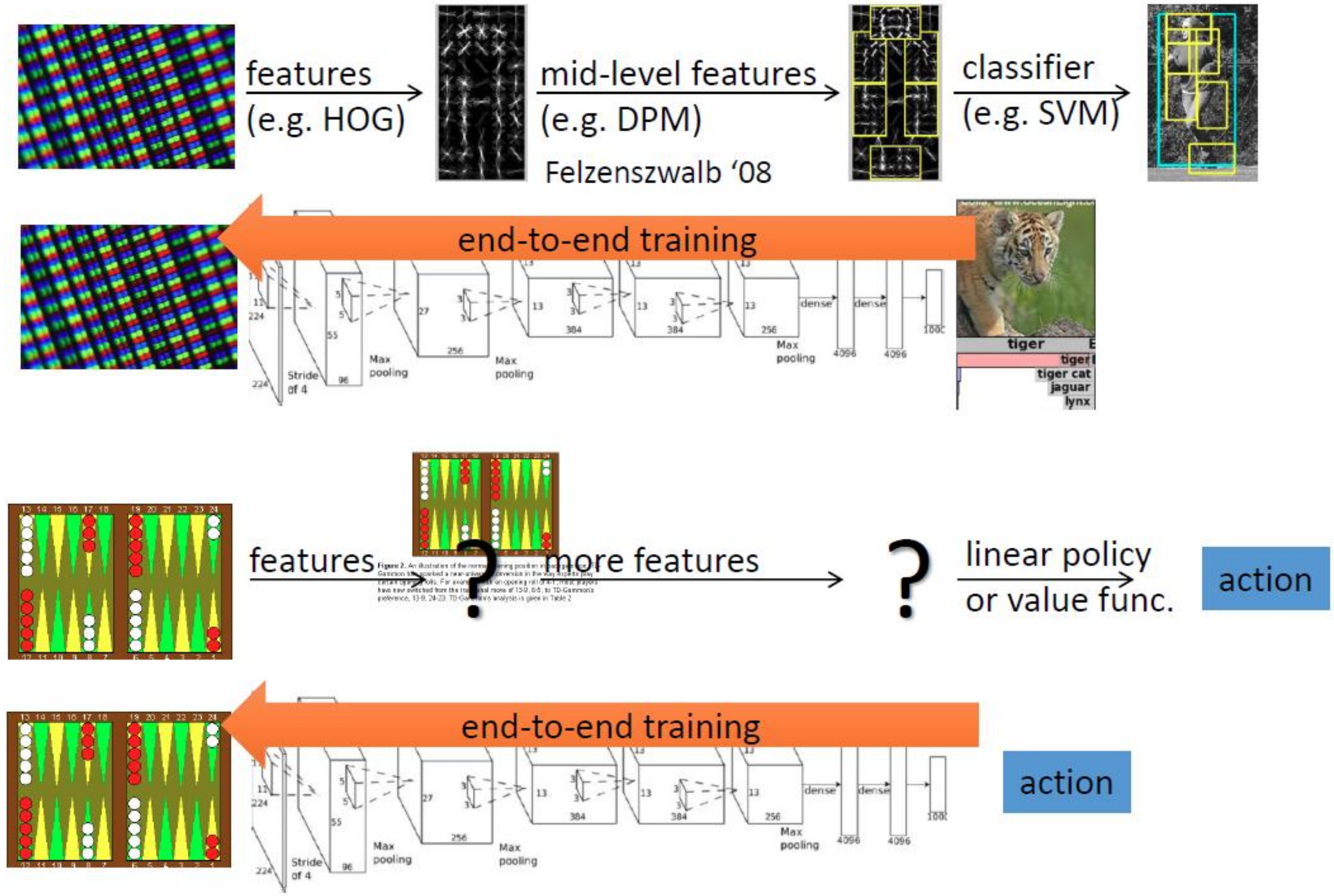
端到端强化学习

标准（传统）
计算机视觉

深度学习

标准（传统）
强化学习

深度强化学习



深度强化学习使强化学习算法能够以端到端的方式解决复杂问题

Slide from Sergey Levine. http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-1.pdf

深度强化学习带来的关键变化

- 假如将深度学习（DL）和强化学习（RL）结合在一起会发生什么？
  - 价值函数和策略现在变成了深度神经网络
  - 相当高维的参数空间
  - 难以稳定地训练
  - 容易过拟合
  - 需要大量的数据
  - 需要高性能计算
  - CPU（用于收集经验数据）和GPU（用于训练神经网络）之间的平衡
  - …

这些新的问题促进着深度强化学习算法的创新

大 纲

深度强化学习概述

Deep Q-learning（DQN）

Proximal Policy Optimization (PPO)

Asynchronous Advantage Actor-Critic (A3C)

回顾：值函数估计

- **目标：**寻找$\boldsymbol{w}$最小化值函数**估计值**与**真实值**之间的均方误差

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}\left[\left(V^{\pi}(s) - \hat{V}(s,\boldsymbol{w})\right)^2\right]$$

真实值　估计值

- 误差减少的方向

$$-\frac{1}{2}\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \mathbb{E}_{\pi}\left[\left(V^{\pi}(s) - \hat{V}(s,\boldsymbol{w})\right)\nabla_{\boldsymbol{w}}\hat{V}(s,\boldsymbol{w})\right]$$

TD：时序差分误差；
MC：轨迹误差

- 采样梯度下降

$$\Delta\boldsymbol{w} = -\frac{1}{2}\alpha\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \alpha\left(V^{\pi}(s) - \hat{V}(s,\boldsymbol{w})\right)\nabla_{\boldsymbol{w}}\hat{V}(s,\boldsymbol{w})$$

回顾：状态–动作值函数估计

- 同样，对动作-状态值函数进行估计

$$\hat{Q}(s,a,w) \approx Q^{\pi}(s,a)$$

- 最小均方误差

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}\left[\left(Q^{\pi}(s,a) - \hat{Q}(s,a,\boldsymbol{w})\right)^2\right]$$

- 在单个样本上进行随机梯度下降

$$\Delta \boldsymbol{w} = -\frac{1}{2}\alpha\nabla_{\boldsymbol{w}}J(\boldsymbol{w}) = \alpha\left(Q^{\pi}(s,a) - \hat{Q}(s,a,\boldsymbol{w})\right)\nabla_{\boldsymbol{w}}\hat{Q}(s,a,\boldsymbol{w})$$

TD：TD target

Q-learning

Q-learning的优化目标

- ✓ 深度 Q 学习旨在最小化目标函数 / 损失函数：

$$J(w) = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{Q}(S', a, w) - \hat{Q}(S, A, w)\right)^2\right]$$

Q-learning的TD目标

其中 (S, A, R, S') 是随机变量

- ✓ 上式实际是 Bellman 最优性误差，$\hat{Q}$逼近

$$Q(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a \in \mathcal{A}(S_{t+1})} Q(S_{t+1}, a) | S_t = s, A_t = a\right], \forall s, a$$

- ✓ 从期望的角度来看，$R + \gamma \max_{a \in \mathcal{A}(S')} \hat{Q}(S', a, w) - \hat{Q}(S, A, w)$ 取最优策略时

Q-learning策略优化的数学原理

证明： $\pi'(s) = \underset{a}{\operatorname{argmax}}\, Q^{\pi}(s,a) \quad V^{\pi'}(s) \geq V^{\pi}(s)$，对所有$s$

$$V^{\pi}(s) = Q^{\pi}\big(s,\pi(s)\big) \leq \max_{a} Q^{\pi}(s,a) = Q^{\pi}\big(s,\pi'(s)\big)$$

重复利用上式

$$V^{\pi}(s) \quad \leq Q^{\pi}\big(s,\pi'(s)\big)$$

$$= E[r_{t+1} + V^{\pi}(s_{t+1}) | s_t = s, a_t = \pi'(s_t)]$$

$$\leq E[r_{t+1} + Q^{\pi}\big(s_{t+1},\pi'(s_{t+1})\big) | s_t = s, a_t = \pi'(s_t)]$$

$$= E[r_{t+1} + r_{t+2} + V^{\pi}(s_{t+2}) | \ldots]$$

$$\leq E[r_{t+1} + r_{t+2} + Q^{\pi}\big(s_{t+2},\pi'(s_{t+2})\big) | \ldots] \qquad \ldots \leq V^{\pi'}(s)$$

Q-learning的梯度更新

◻ Deep Q-learning

- ✓ 最早且最成功的算法之一，它将深度神经网络引入强化学习（RL）。
- ✓ 神经网络的作用是作为一个非线性函数逼近器。

$$\Delta \boldsymbol{w} = (R_{t+1} + \gamma \max_{a} \hat{Q}(s, a, \boldsymbol{w_t}) - \hat{Q}(s_t, a_t, \boldsymbol{w_t}))\nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t, \boldsymbol{w_t})$$

DQN梯度更新会有什么问题？

DQN的梯度更新

☐ 梯度计算逼近

- ✓ 期望中两项都含有参数w：

$$J(w)=\mathbb{E}\left[\left(R+\gamma\max_{a\in\mathcal{A}(S')}\hat{Q}(S',a,w)-\hat{Q}(S,A,w)\right)^2\right]$$

- ✓ 将第一二项之和看作常数，即 $R_{t+1}+\gamma\max_{a\in\mathcal{A}(s_{t+1})}\hat{Q}(s_{t+1},a,w_t)=\text{constant}$：

$$w_{t+1}$$

$$=w_t+\alpha_t\left[R_{t+1}+\gamma\max_{a\in\mathcal{A}(s_{t+1})}\hat{Q}(s_{t+1},a,w_t)-\hat{Q}(s_t,a_t,w_t)\right]\nabla_w\hat{Q}(s_t,a_t,w_t)$$

存在追逐不稳定目标问题

DQN的过高估计问题

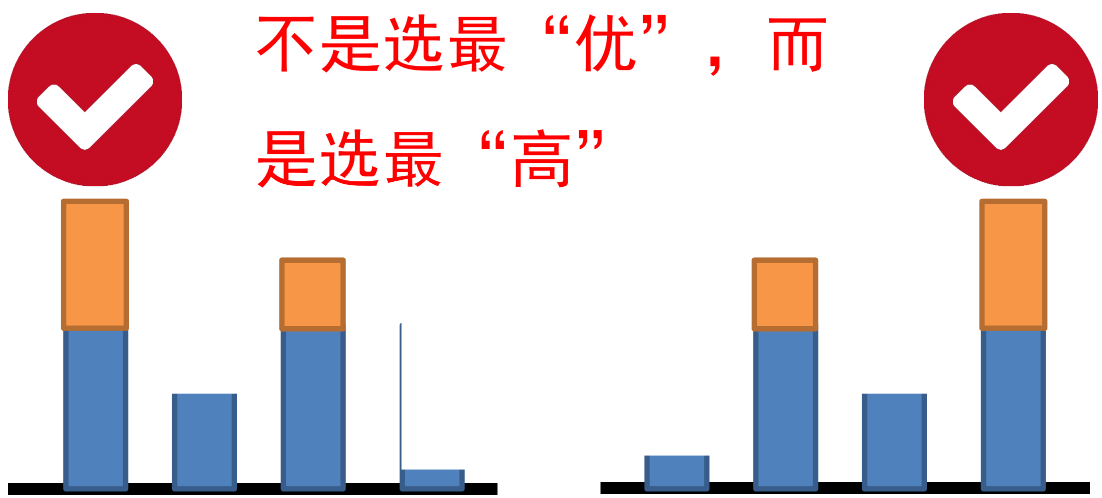
◻过高估计问题

✓ max运算引入高方差

✓ Q值通常被高估 $Q(s_t, a_t) \longleftrightarrow r_t + \max_a Q(s_{t+1}, a)$

$$J(w) = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w) - \hat{q}(S, A, w)\right)^2\right]$$

$Q(s_{t+1}, a)$



双网络结构

- 在线网络：代表 $\hat{q}(s, a, w)$, 时刻更新
- 目标网络：代表 $\hat{q}(s, a, w_T)$

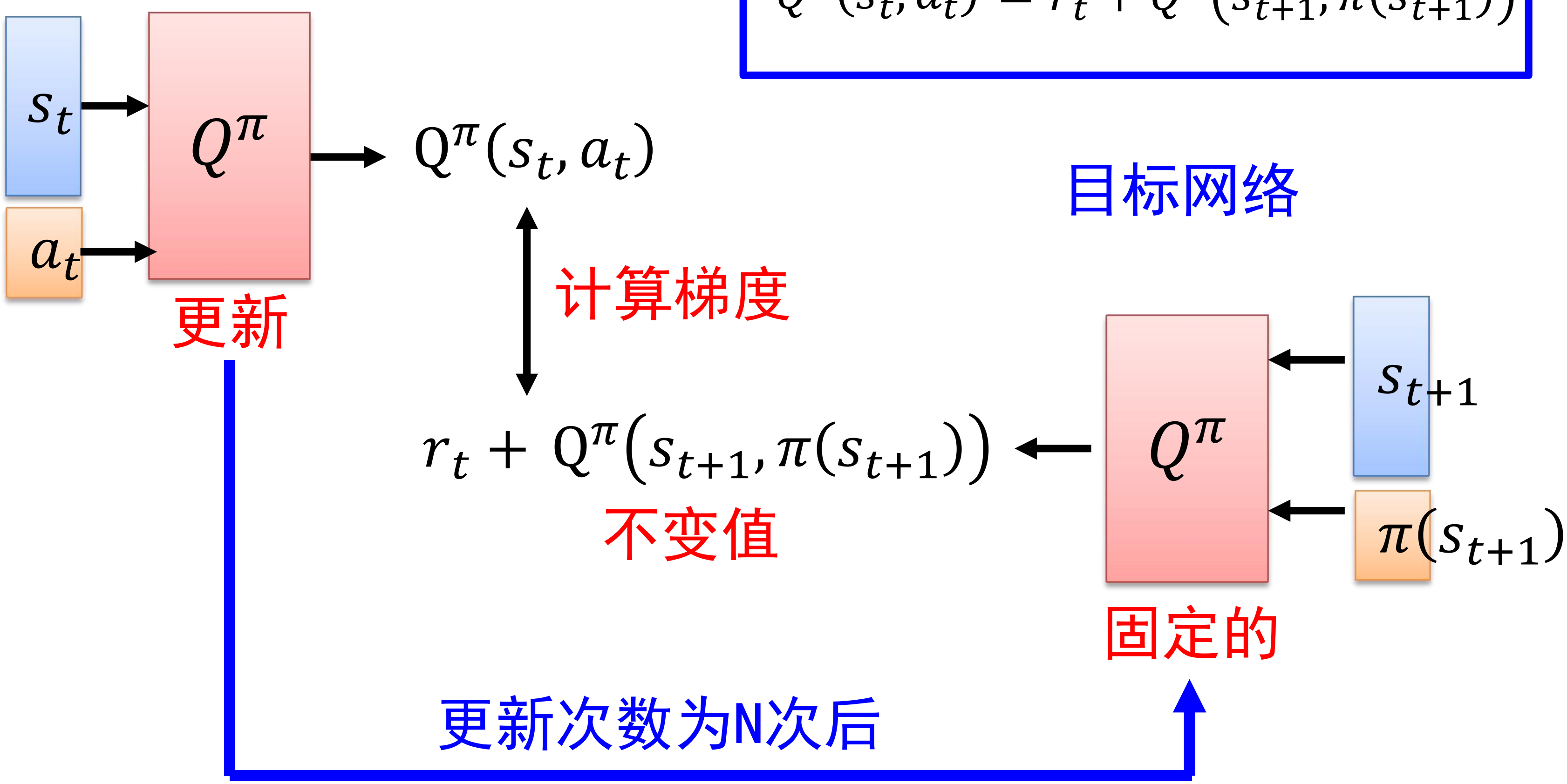
$$J(w) = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}(S', a, w_T) - \hat{q}(S, A, w)\right)^2\right]$$

- 打破时序相关性
- 避免追逐移动目标，降低高方差的影响

目标网络

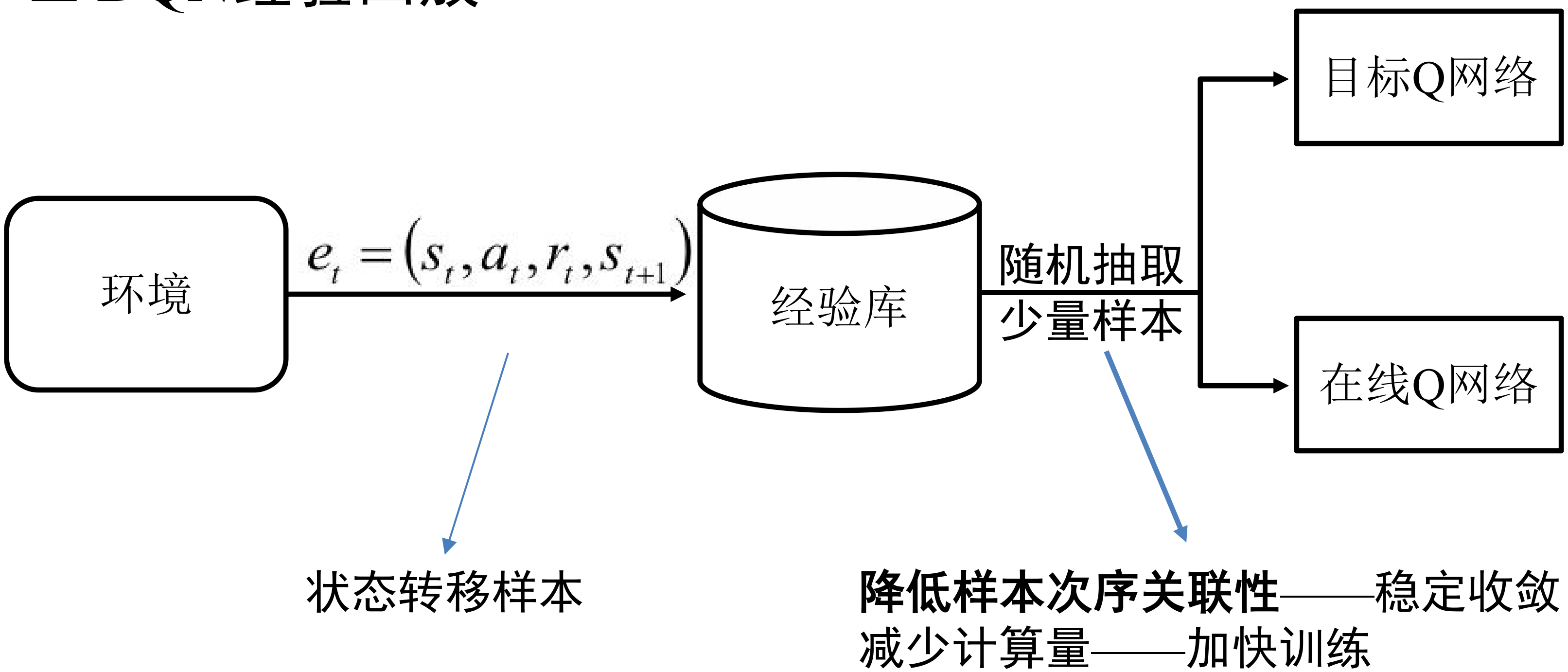
$$\cdots s_t, a_t, r_t, s_{t+1} \cdots$$

$$Q^\pi(s_t, a_t) = r_t + Q^\pi\big(s_{t+1}, \pi(s_{t+1})\big)$$



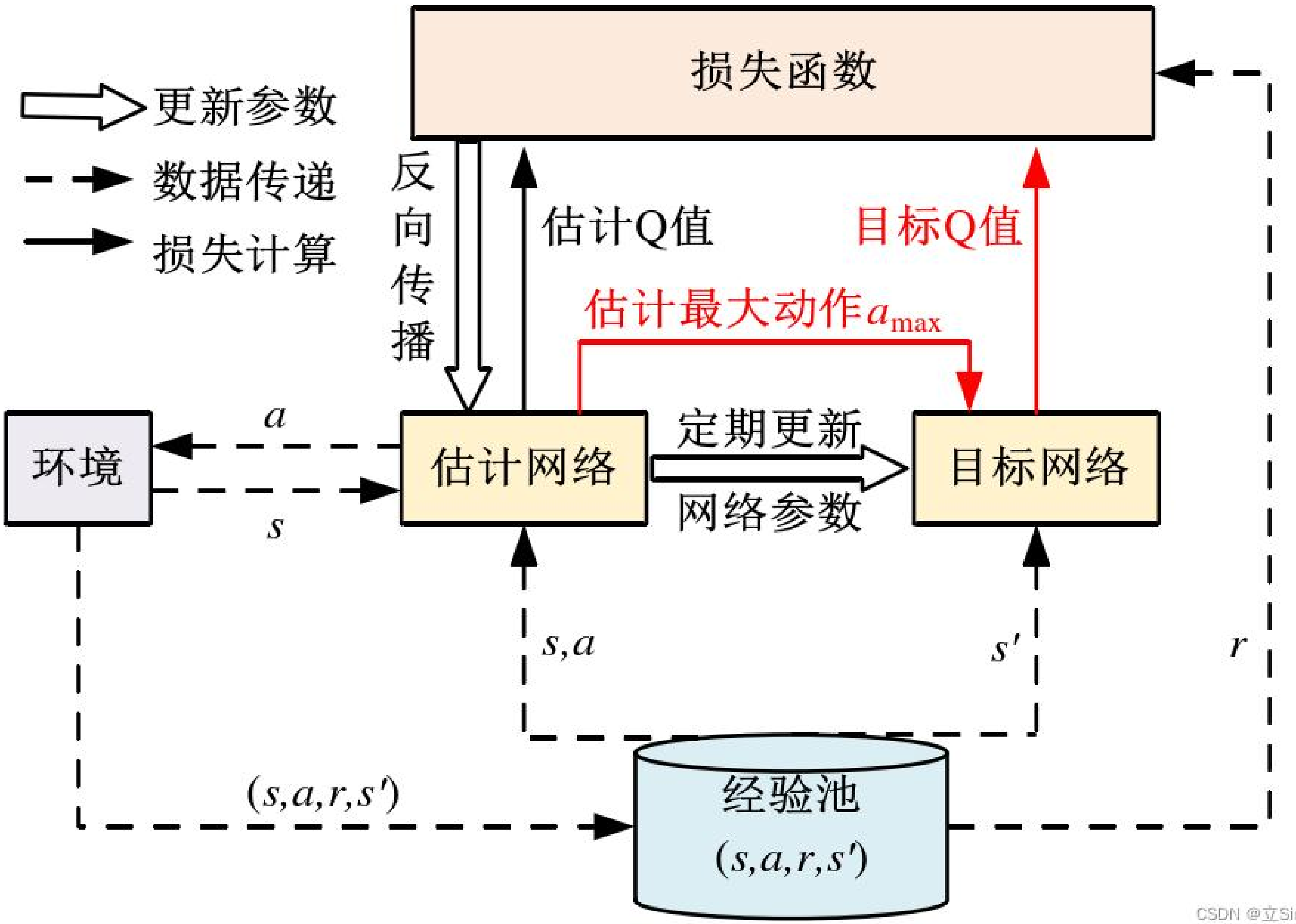
经验回放

- DQN经验回放



- 在每一次选择动作后，存储样本$\langle s, a, r, s' \rangle$到样本池中
- 每次迭代从样本池中随机均匀抽取样本更新Q网络参数

Q-learning 算法过程图



Deep Q-learning经验回放

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$

采样

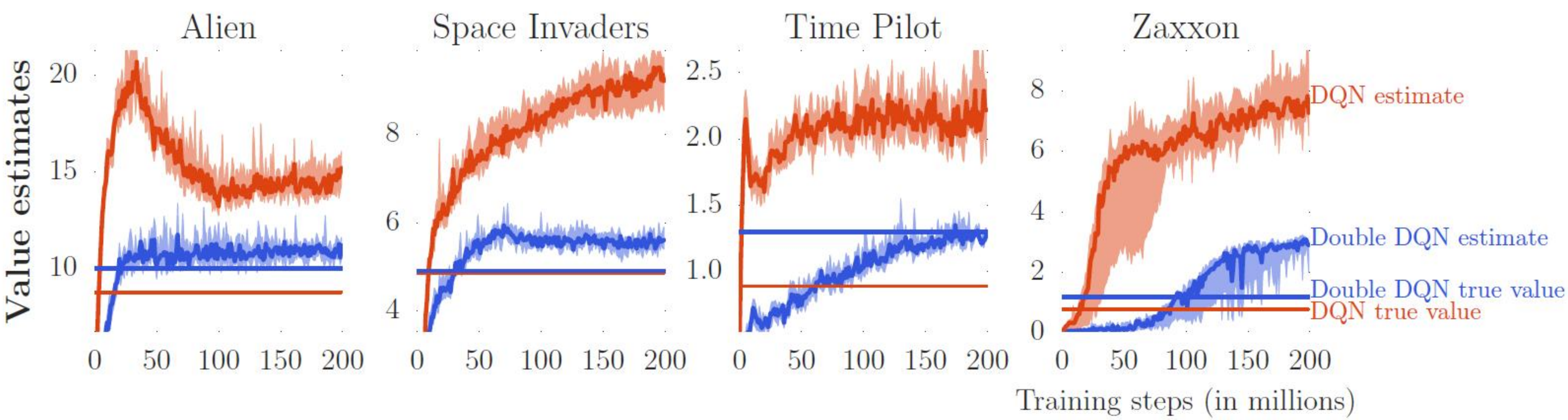
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation $\boxed{3}$

经验回放

  end for
end for

实验效果

- 缓解Q值通常被高估问题



大 纲

深度强化学习概述

Deep Q-learning（DQN）

Proximal Policy Optimization (PPO)

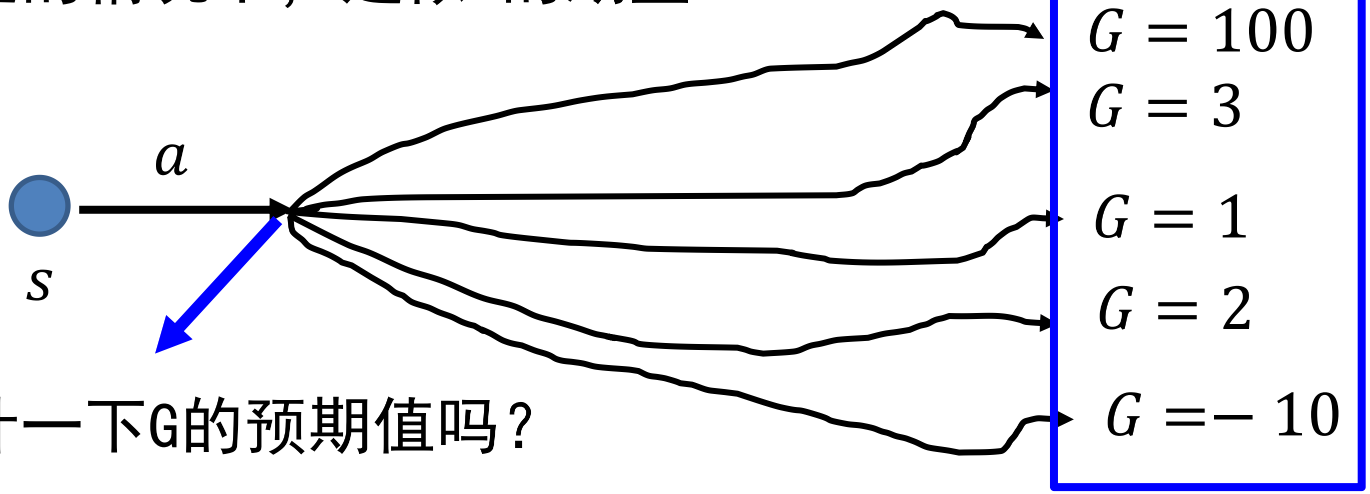
Asynchronous Advantage Actor-Critic (A3C)

回顾：策略梯度

$$\nabla J_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\Big(\underline{\sum_{t'=t}^{T_n}\gamma^{t'-t}r_{t'}^n} - \underline{\overset{\text{baseline}}{b}}\Big)\nabla\log\pi_\theta(a_t^n|s_t^n)$$

G_t^n：通过互动获得 非常不稳定

在样本充足的情况下，近似G的期望



我们能估计一下G的预期值吗？

如何将采样与行动解耦？

PPO：从 on-policy 到 off-policy

On-policy：学习到的智能体和与环境交互的智能体是相同的
Off-policy：学习到的智能体和与环境交互的智能体是不同的

开车上路

通过观看视频学习经验

PPO：从 on-policy 到 off-policy

$$\nabla_\theta J(\theta) = \mathbb{E}_{a\sim\pi_\theta(a)}[Q^{\pi_\theta}(a)\nabla_\theta \log \pi_\theta(a)]$$

- 问题：使用 π_θ 去收集数据。 当θ更新时，我们需要再次去采样训练数据

行动和样本耦合

- 目标：使用从 $\pi_{\theta'}$得到的采样数据去训练θ。θ'是固定的，因此我们可以重复使用样本数据

高样本效率与并行化支持

重要性采样

$\mathbb{E}_{x\sim p}[f(x)]$

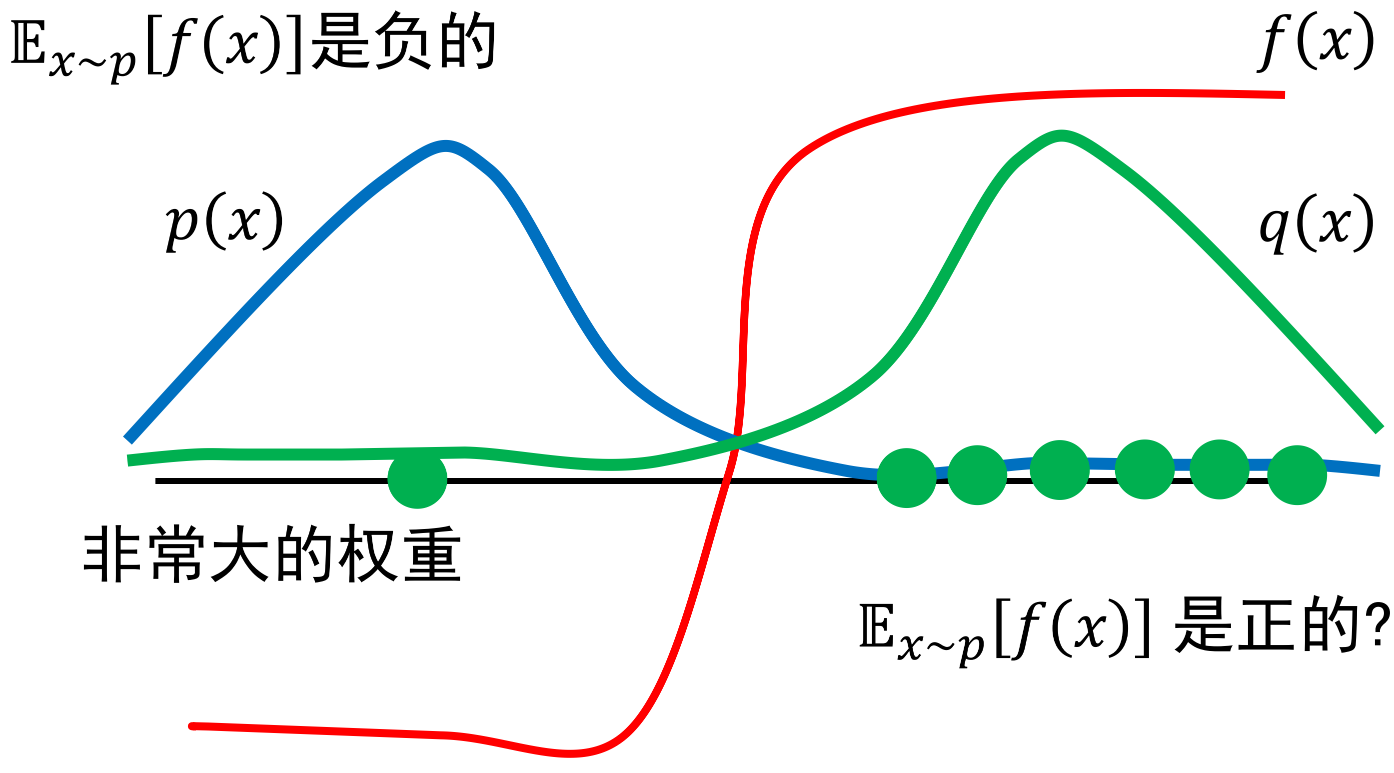
我们只能从 $q(x)$处采样获得 x^i

$$= \int f(x)p(x)dx \ = \int f(x)\frac{p(x)}{q(x)}q(x)dx \ = E_{x\sim q}[f(x)\boxed{\frac{p(x)}{q(x)}}]$$

重要性权重

重要性采样的偏差问题

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



PPO：On-policy → Off-policy

更新梯度

$$= \mathbb{E}_{(s_t,a_t)\sim\pi_\theta}[A^\theta(s_t,a_t)\nabla\log\pi_\theta(a_t^n|s_t^n)]$$

$\boxed{A^{\theta'}(s_t,a_t)}$ 通过离策略数据采样获得

$$= \mathbb{E}_{(s_t,a_t)\sim\pi_{\theta'}}[\frac{p_\theta(s_t,a_t)}{p_{\theta'}(s_t,a_t)}\cancel{A^{\theta}(s_t,a_t)}\nabla\log\pi_\theta(a_t^n|s_t^n)]$$

$$= \mathbb{E}_{(s_t,a_t)\sim\pi_{\theta'}}[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}\cancel{\frac{p_\theta(s_t)}{p_{\theta'}(s_t)}}A^{\theta'}(s_t,a_t)\nabla\log\pi_\theta(a_t^n|s_t^n)]$$

回顾 $\nabla f(x) = f(x)\nabla\log f(x)$

$$J^{\theta'}(\theta) = \mathbb{E}_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}A^{\theta'}(s_t,a_t)\right]$$

被优化的目标函数

PPO：On-policy → Off-policy

$$\nabla_\theta J(\theta) = \underline{\mathbb{E}_{a\sim\pi_\theta(\tau)}}[A^{\pi_\theta}(a)\nabla\log\ \pi_\theta(a)]$$

- ✓ 使用π_θ收集数据。当θ更新时，我们需要重新采样训练数据。
- ✓ 目标：使用从$\pi_{\theta'}$得到的采样数据去训练θ。θ'是固定的，因此我们可以重复使用样本数据。

$$\nabla_\theta J(\theta) = \underline{\mathbb{E}_{a\sim\pi_{\theta'}(\tau)}}\left[\frac{p_\theta(a|s)}{p_{\theta'}(a|s)}\underline{A^{\pi_{\theta'}}(a)}\nabla\log\pi_\theta(a)\right]$$

- ✓ 从θ'处采样数据。
- ✓ 使用数据去训练θ多次。

重要性采样

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

PPO：增加约束

PPO / TRPO

θ不能和θ'非常不同

应该约束行为而不是约束参数

Proximal Policy Optimization (PPO)

$$J_{PPO}^{\theta'}(\theta) = J^{\theta'}(\theta) - \beta \mathrm{KL}(\theta, \theta')$$

$$J^{\theta'}(\theta) = \mathbb{E}_{(s_t, a_t) \sim \pi_{\theta'}} \left[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)} A^{\theta'}(s_t, a_t) \right]$$

TRPO (Trust Region Policy Optimization)

$$J_{TRPO}^{\theta'}(\theta) = \mathbb{E}_{(s_t, a_t) \sim \pi_{\theta'}} \left[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)} A^{\theta'}(s_t, a_t) \right] \quad \text{s.t. } \mathrm{KL}(\theta, \theta') < \delta$$

PPO：增加约束

$$J^{\theta^k}(\theta) \approx \sum_{(s_t,a_t)} \frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)} A^{\theta^k}(s_t,a_t)$$

- 初始化策略参数θ^0
- 在每轮迭代中
  - ✓ 使用θ^k 去与环境交互来收集 $\{s_t,a_t\}$ 和计算优点函数 $A^{\theta^k}(s_t,a_t)$
  - ✓ 寻找θ来优化 $J_{PPO}(\theta)$

$$J_{PPO}^{\theta^k}(\theta) = J^{\theta^k}(\theta) - \beta \mathrm{KL}(\theta,\theta^k)$$

多次更新参数

- 若$\mathrm{KL}(\theta,\theta^k) > \mathrm{KL}_{\max}$，增加$\beta$
- 若$\mathrm{KL}(\theta,\theta^k) < \mathrm{KL}_{\min}$，减少$\beta$

自适应KL惩罚机制

PPO：增加约束

PPO算法

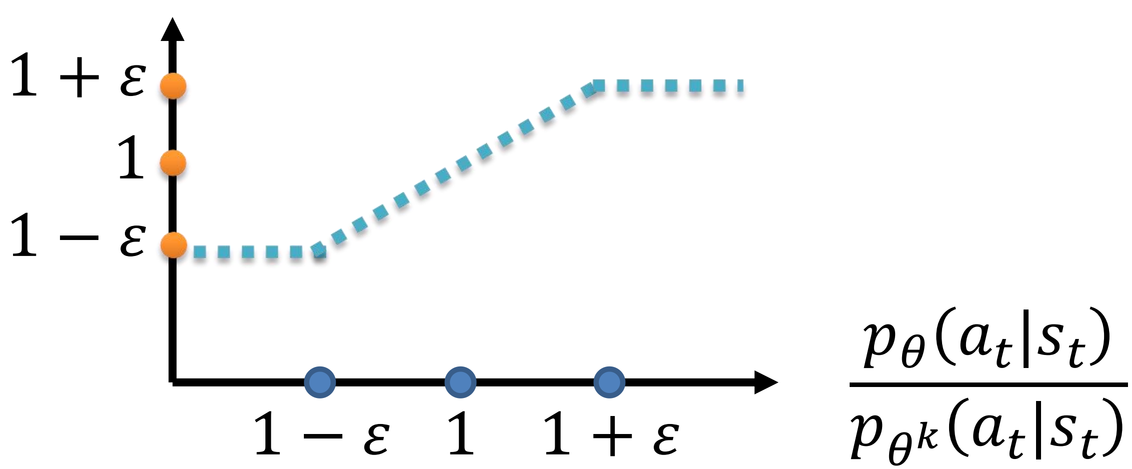
$$J_{PPO}^{\theta^k}(\theta) = J^{\theta^k}(\theta) - \beta \mathrm{KL}(\theta, \theta^k)$$

$$J^{\theta^k}(\theta) \approx \sum_{(s_t, a_t)} \frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)} A^{\theta^k}(s_t, a_t)$$

PPO2算法

$$J_{PPO2}^{\theta^k}(\theta) \approx \sum_{(s_t, a_t)} \min\left(\frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)} A^{\theta^k}(s_t, a_t), \mathrm{clip}\left(\frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right) A^{\theta^k}(s_t, a_t)\right)$$

$$\approx \sum_{(s_t, a_t)} \min\left(\mathrm{clip}\left(\frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right) A^{\theta^k}(s_t, a_t)\right)$$



PPO：增加约束

PPO算法

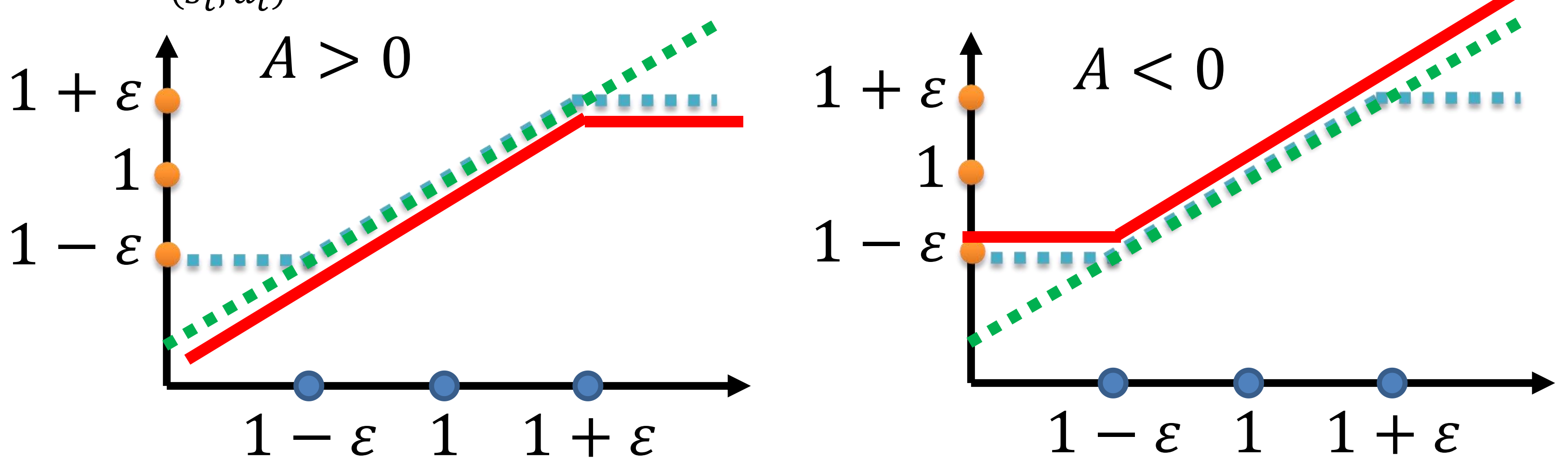
$$J_{PPO}^{\theta^k}(\theta) = J^{\theta^k}(\theta) - \beta KL(\theta, \theta^k)$$

$$J^{\theta^k}(\theta) \approx \sum_{(s_t, a_t)} \frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)} A^{\theta^k}(s_t, a_t)$$

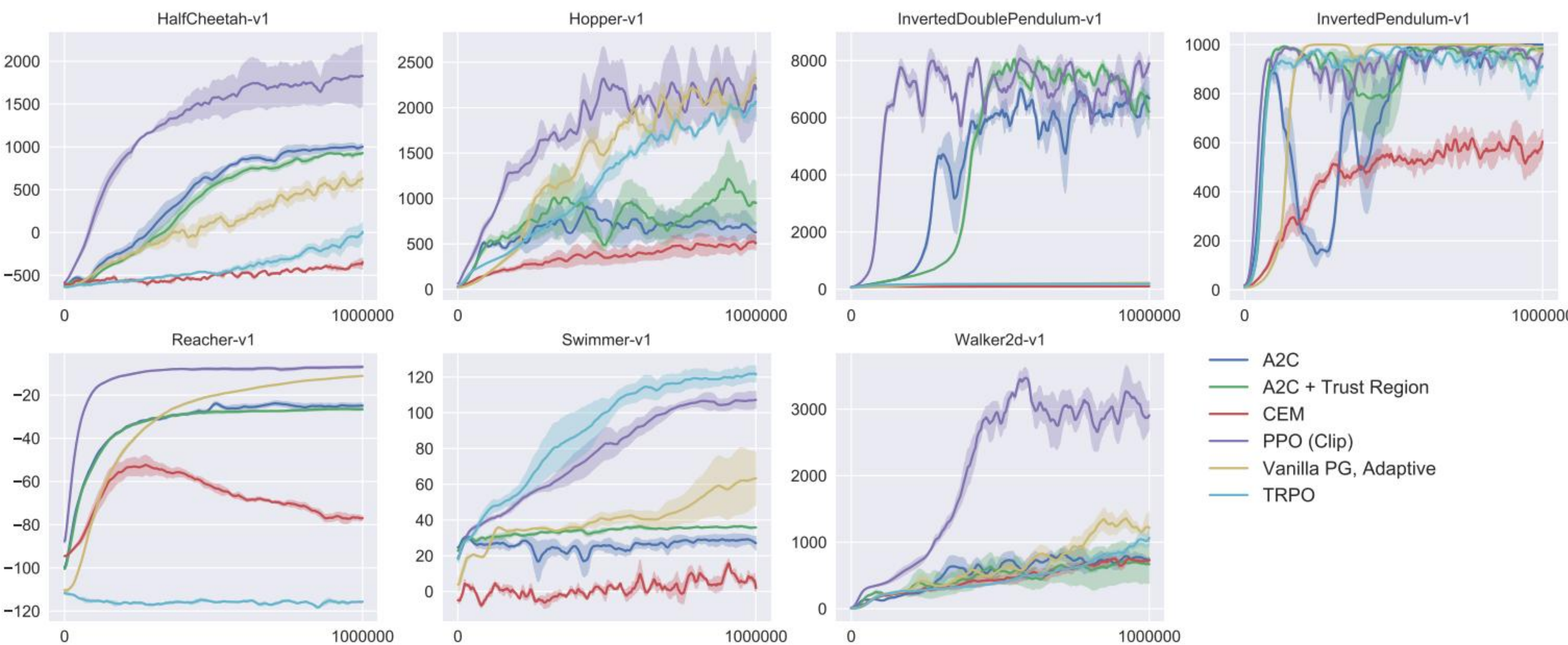
PPO2算法

$$J_{PPO2}^{\theta^k}(\theta) \approx \sum_{(s_t, a_t)} \min\left(\frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)} A^{\theta^k}(s_t, a_t), \mathrm{clip}\left(\frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right) A^{\theta^k}(s_t, a_t)\right)$$

$$\approx \sum_{(s_t, a_t)} \min\left(\mathrm{clip}\left(\frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right) A^{\theta^k}(s_t, a_t)\right)$$



PPO实验对比



John Schulman, et al. Proximal Policy Optimization Algorithms. 2017.

PPO小结

- **优点：**

收敛稳定、计算效率高、样本利用率高、适用于离散 & 连续动作空间

- **缺点：**

超参数敏感、探索能力有限、无法保证严格的单调性（可能导致性能不稳定）

适用环境：

自动驾驶、机器人控制、金融交易等。

大 纲

深度强化学习概述

Deep Q-learning（DQN）

Proximal Policy Optimization (PPO)

Asynchronous Advantage Actor-Critic (A3C)

回顾：Actor-Critic

$Q^{\pi_\theta}(s_t^n, a_t^n) - V^{\pi_\theta}(s_t^n)$

$V^{\pi_\theta}(s_t^n)$

基线

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\boxed{\left(\underline{\sum_{t'=t}^{T_n}\gamma^{t'-t}r_{t'}^n} - \underline{b}\right)}\nabla \log \pi_\theta(a_t^n|s_t^n)$$

G_t^n：通过互动获得

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

回顾：Advantage Actor-Critic

$$Q^{\pi}(s_t^n, a_t^n) - V^{\pi}(s_t^n)$$

估计两个网络？我们只能估计一个。

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

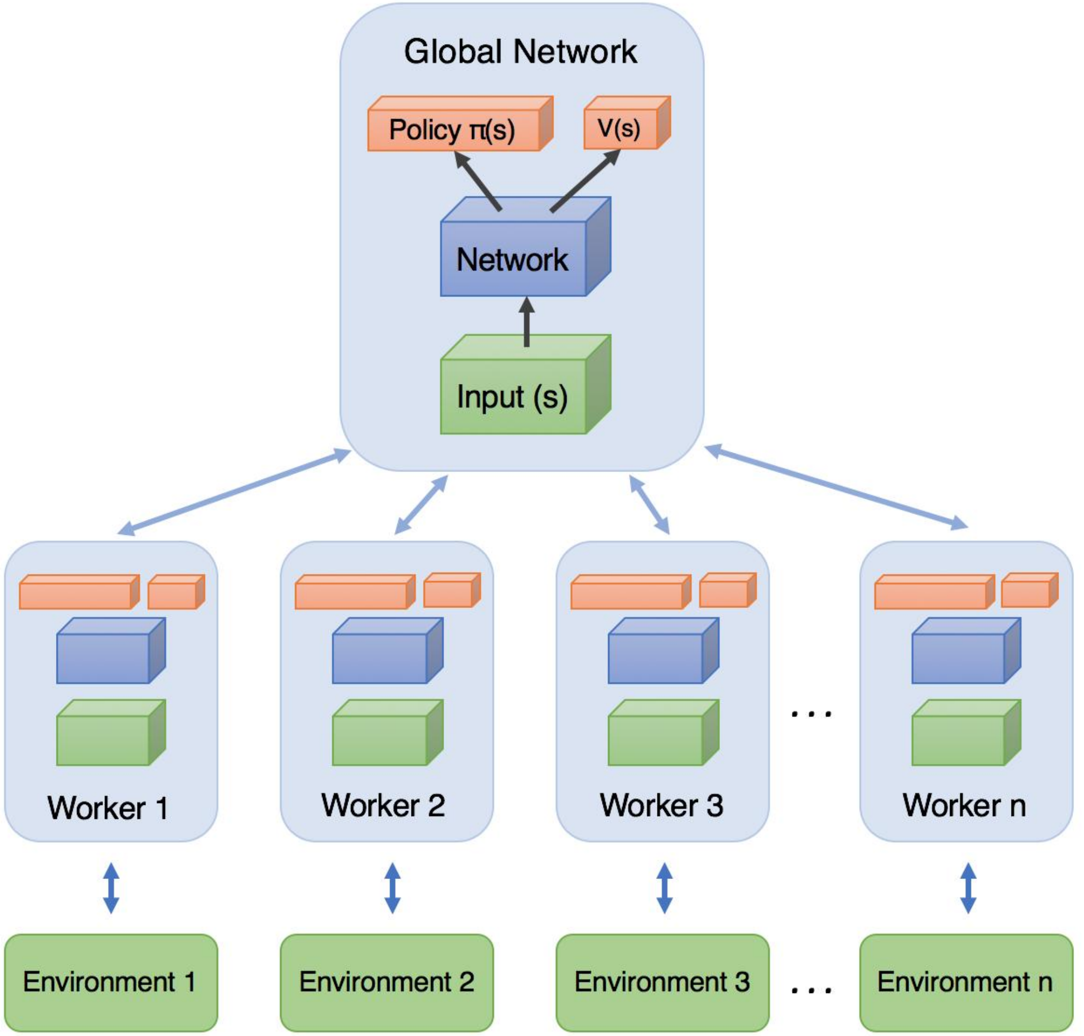
仅估计状态值

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

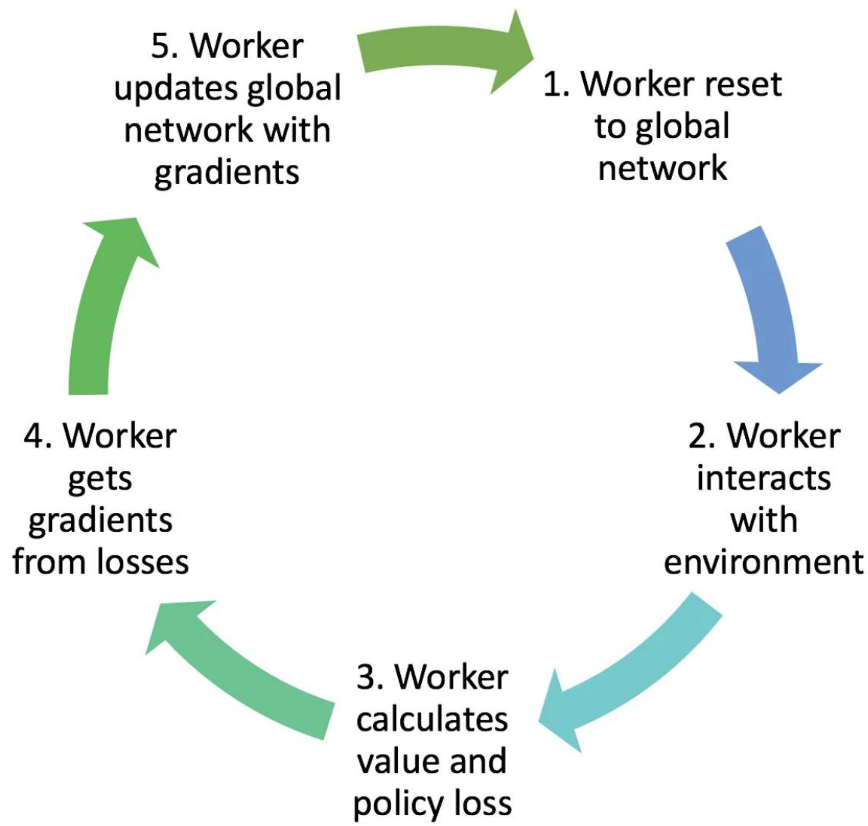
A3C: 异步A2C方法

每个Agent：

1. 复制全局参数
2. 抽样一些数据
3. 计算梯度
4. 更新全局模型



A3C算法每个worker agent训练流程



1. 每个worker从global network复制参数
2. 不同的worker与环境去做互动
3. 不同的worker计算出各自的gradient
4. 不同的worker把各自的gradient传回给global network
5. global network接收到gradient后进行参数更新

A3C算法的梯度更新

A3C算法策略梯度计算公式：

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta (a_t|s_t) A(s_t, a_t)]$$

其中$A(s_t, a_t)$ 是优势函数，用于衡量当前动作相较于平均策略的优越性：

$$A(s_t, a_t) = Q_t(s_t, a_t) - V^{\pi_\theta}(s_t)$$

A3C算法价值函数更新公式：

Critic 负责评估状态的值函数 $\hat{V}(s_t)$ ，它的损失函数定义为：

$$L_V(\theta_v) = \left(R - \hat{V}(s_t;\theta_v)\right)^2$$ 其中，$R = r_t + \hat{V}(s_{t+1};\theta_v)$ for TD（0）

通过最小化均方误差（MSE）来更新值函数的参数θ：

$$\nabla_{\theta_v} L_V(\theta_v) = \nabla_{\theta_v}\left(R - \hat{V}(s_t;\theta_v)\right)^2$$

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** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$

异步参数更新

  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 $T > T_{max}$

Volodymyr Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. NIPS 2016.

带熵奖励的A3C算法的梯度更新

- 总损失函数：

A3C 采用异步更新，因此多个worker进程并行采样并计算梯度，最终的损失函数可以写为：

$$L = L_{\text{policy}}(\theta) + \lambda L_V(\theta_v) - \beta H(\pi)$$

其中：

➢ $L_{\text{policy}} = -\log \pi_\theta(a_t|s_t) A(s_t, a_t)$ （负奖励）

➢ L_V 是值函数的损失

➢ $H(\pi)$ 是策略的熵，目的是鼓励探索，防止策略过早收敛到次优解

➢ λ 和 β 是超参数，用于平衡值函数损失和熵奖励

带熵奖励的A3C算法的梯度更新

◻ A3C算法异步更新：

A3C 的梯度更新是在每个线程中计算并累积，在每个片段开始时，线程特定参数 θ' 和 θ'_v 与全局参数 θ 和 θ_v 同步。

对于每个时间步 i（从 t − 1 到 $\mathbf{t_{start}}$ ）：

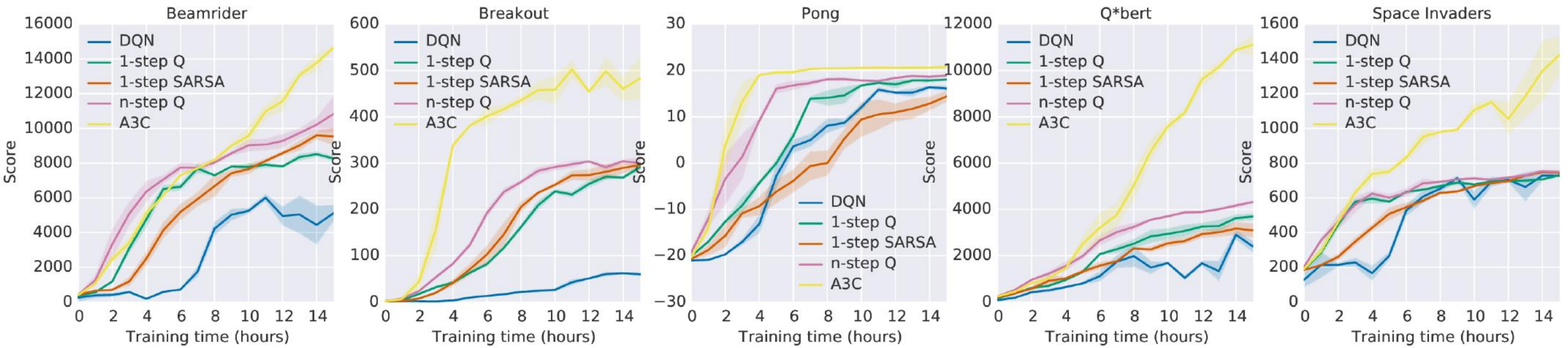
✓ 计算策略梯度 和值函数梯度，累积到

$$d\theta \leftarrow d\theta + \nabla_{\theta'}\log\pi(a_i|s_i;\theta')\big(R - V(s_i;\theta'_v)\big) + \beta\nabla_{\theta'}H\big(\pi(s_i;\theta')\big)$$

$$d\theta_v \leftarrow d\theta_v + \frac{\partial\Big(R - V(s_i;\theta'_v)\Big)^2}{\partial\theta'_v}$$

✓ 异步更新：将累积的 $d\theta$ 和 $d\theta_v$ 应用于全局参数 θ 和 θ_v

A3C对比实验



a single Nvidia K40 GPU while the asynchronous methods were trained using 16 CPU cores

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

Volodymyr Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. NIPS 2016.

谢 谢 ！