强化学习

第九讲：深度强化学习

教师：赵冬斌　朱圆恒　张启超

中国科学院大学
中国科学院自动化研究所

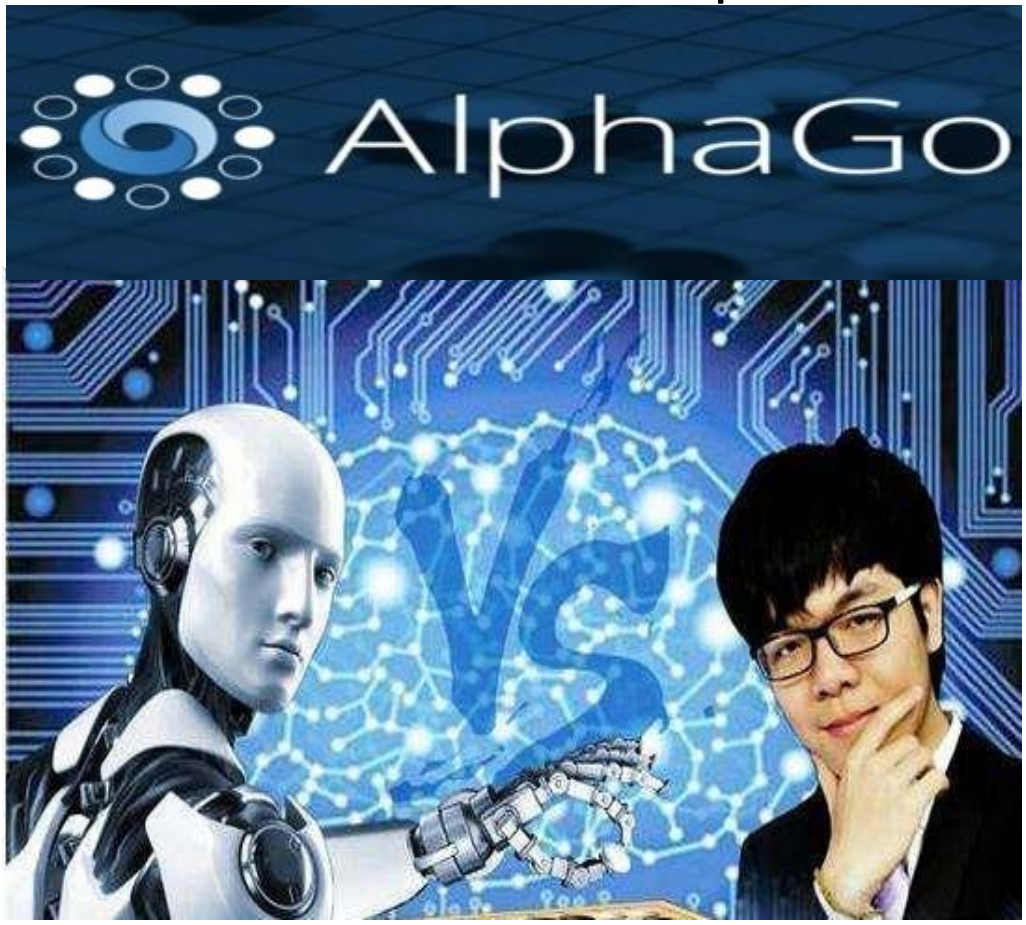
June 6, 2021

课程目录

深度强化学习的发展

- 讲到深度强化学习会想到的是?

2016年李世石 VS AlphaGo



深度强化学习的发展

- 单智能体感知决策

2013 NIPS，DeepMind提出Atari视频游戏的深度强化学习算法DQN，得分超过人类水平

神经信息处理系统大会 (Conference and Workshop on Neural Information Processing Systems) CCF-A类会议

2013-2015 → **2016-2017** → **2017-2018** → **2019-2020**

2015 Nature，DeepMind提出Atari视频游戏的深度强化学习算法DQN，得分超过人类高级水平

深度强化学习的发展

- 完全信息零和博弈

2016 Nature，DeepMind的AlphaGo以4:1的大比分战胜了世界围棋顶级选手李世石

2013-2015 → 2016-2017 → 2017-2018 → 2019-2020

- 蒙特卡洛树搜索
- Actor-Critic
- 自我博弈

2017 Nature，DeepMind的AlphaGo Zero不依赖人类的棋谱数据，自我博弈学习达到人类顶级水平

深度强化学习的发展

- 非完全信息双人零和博弈
  - 蒙特卡洛搜索
  - 虚拟遗憾最小化算法（CFR）

  …

2017 Science, 加拿大阿尔伯特大学开发的DeepStack，世界上第一个在“1对1无限注德州扑克”上击败了职业扑克玩家的计算机程序

2013-2015 → **2016-2017** → **2017-2018** → **2019-2020**

2017-2018, 卡内基梅隆大学开发的Libratus，1V1德扑冷扑大师系列人机大战，先后取胜，获得NIPS 2017最佳论文

深度强化学习的发展

- 非完全信息多人混合博弈

2019 Science, 谷歌第一视角多个体合作游戏，雷神之锤

2019 Science, CMU的六人桌德扑Bot Pluribus

2019.5, OpenAI Five，Dota2的人机大战

2013-2015 → **2016-2017** → **2017-2018** → **2019-2020**

- 多智能体深度强化学习
- 递归神经网络

…

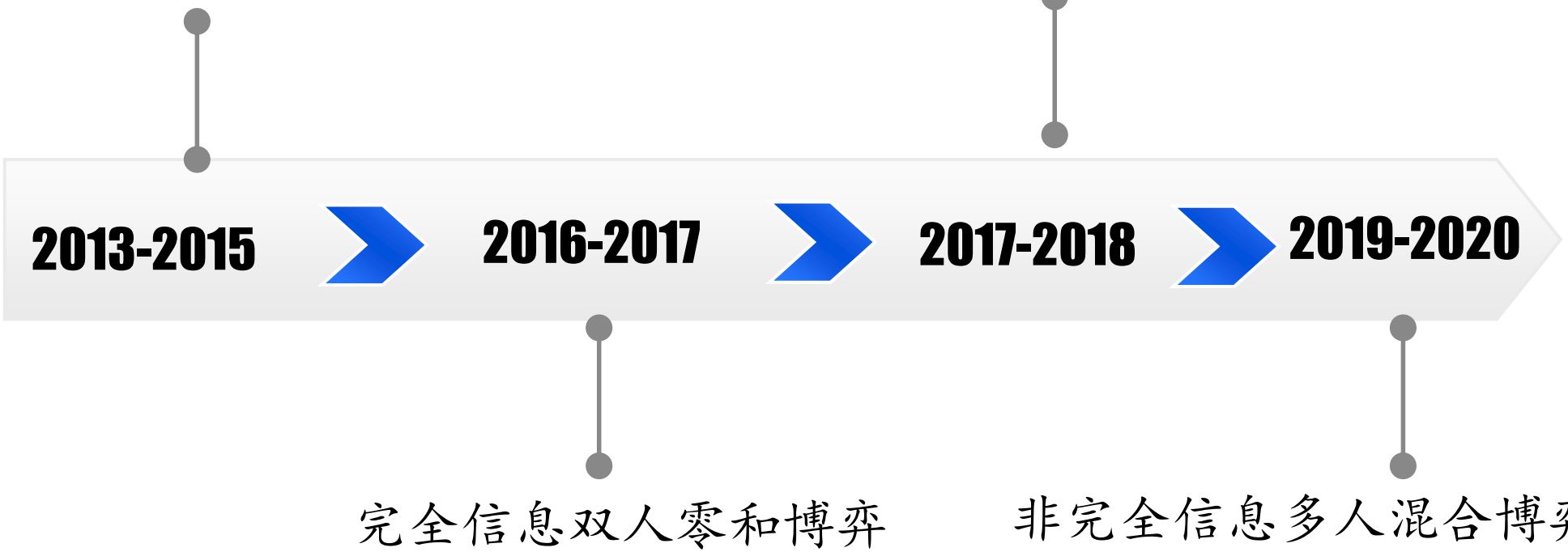
2019.8,微软麻将AI Suphx, 10段

2019.11 Nature，谷歌Alpha Star，星际争霸II达到大师级水平

2019.12，腾讯绝悟AI击败王者荣耀顶尖职业玩家

深度强化学习的发展

深度强化学习近年来的发展趋势



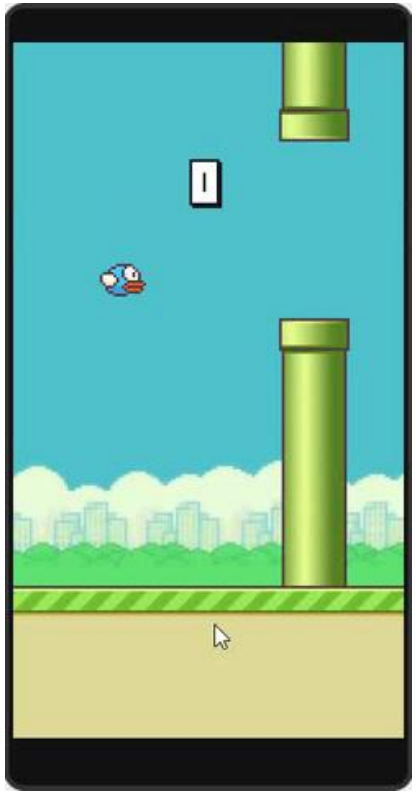
有很多算法及应用方面的问题还有待大家去探索解决

深度强化学习的定义

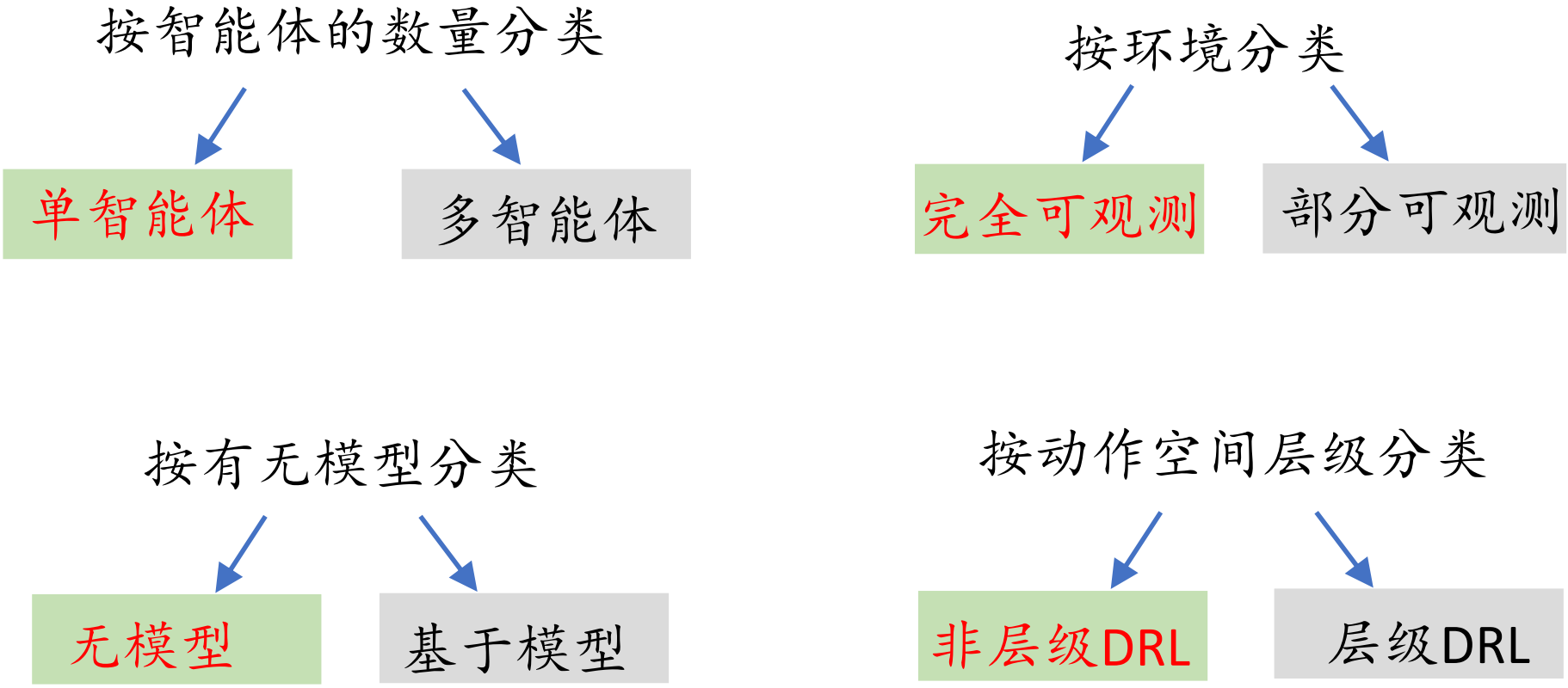
深度强化学习是指什么？

高维输入状态到策略的映射模型

- 字面理解：深度学习+强化学习 = 深度强化学习
- 本质上：使用深度神经网络作为强化学习的函数近似器
- 针对复杂高维原始数据(图像、视频等)输入的决策问题，深度学习强大的特征提取能力，可将原始数据表征为和问题相关的特征

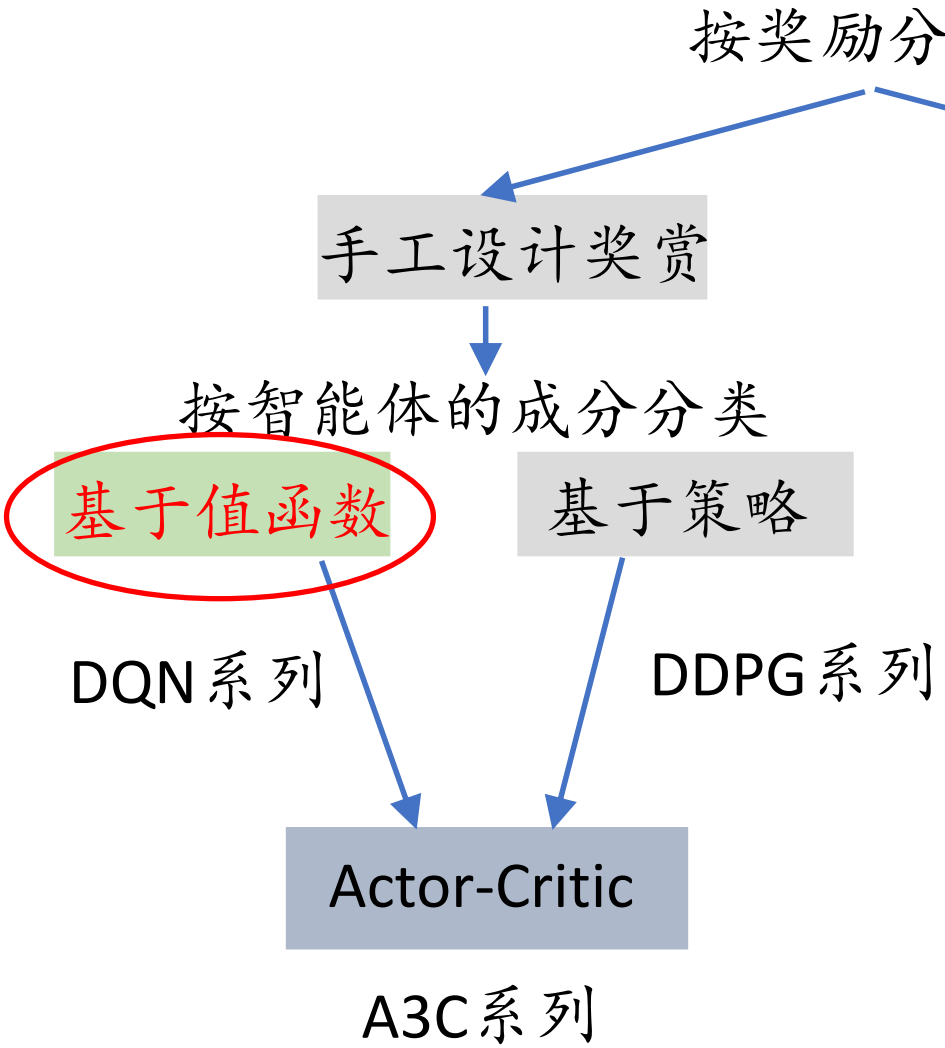


深度强化学习的分类



课程介绍的内容均是针对完全可观测环境下的单智能体无模型DRL方法

深度强化学习的分类



- 经验回放
- 目标网络
- 重要性采样
- …

课程目录

基于Q函数的深度强化学习

学习目标

- 基于Q函数的深度强化学习

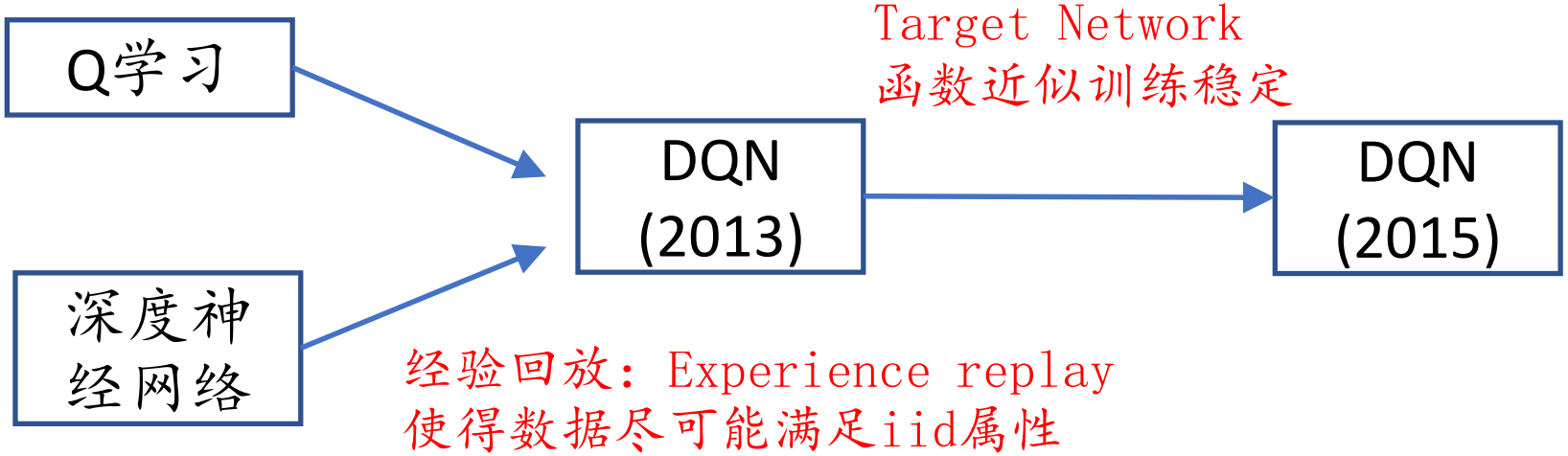
1. DQN(Deep Q Learning)
2. Double DQN
3. Prioritized Experience Replay
4. Dueling DQN

- 如何将深度网络与Q学习结合？
- 离散空间的几类深度Q学习的改进方法

理解如何基于深度网络的复杂函数逼近器应用Q学习

参考：CS 285: Deep Reinforcement Learning, Decision Making, and Control, Sergey Levine

DQN (Deep Q Learning)



- 首次将最火热的深度神经网络与强化学习结合，解决复杂高维输入的决策控制问题，确保训练过程的稳定性
- 证明了能够通过raw pixels 解决游戏问题
- 可适用于绝大多数的Atari游戏

1. Playing Atari with Deep Reinforcement Learning (NIPS2013)
2. Human-level control through deep reinforcement learning (Nature2015)

DQN (Deep Q Learning)

- Q学习回顾

传统Q学习步骤：

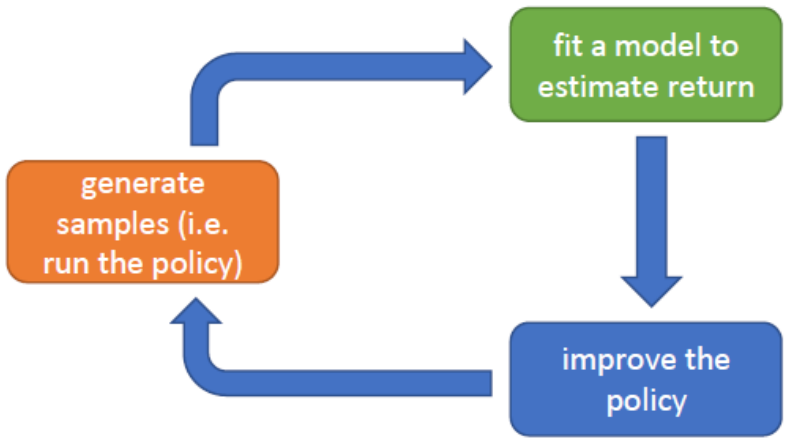
1. 利用某些策略收集数据集 $\{(s_i, a_i, s_i', r_i)\}$

2. 估计Q值 $y_i \leftarrow r(s_i, a_i) + \gamma max_{a_i'} Q_\varphi(s_i', a_i')$

3. 更新参数 $\varphi \leftarrow \arg\min_\varphi \frac{1}{2}\Sigma_i \left\| Q_\varphi(s_i, a_i) - y_i \right\|^2$

N

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



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

DQN (Deep Q Learning)

- 当Q学习遇见深度神经网络

传统在线Q学习步骤：

N

1. 初始执行动作a_i，得到观测数据 (s_i, a_i, s_i', r_i)
2. 估计Q值 $y_i \leftarrow r(s_i, a_i) + \gamma max_{a'} Q_\varphi(s_i', a_i')$
3. 更新参数 $\varphi \leftarrow \varphi - \alpha \frac{dQ_\varphi}{d\varphi}(s_i, a_i)(Q_\varphi(s_i, a_i) - y_i)$

- 每一次只用1个样本训练网络，存在很大方差，Batchsize=1
- 样本存在相关性
- 算法很难保证收敛性

DQN (Deep Q Learning)

- 当Q学习遇见深度神经网络

问题1：样本不满足iid条件，时间相关性很强

在线Q学习步骤：

1. 初始执行动作a_i，得到观测数据$\{(s_i, a_i, s_i', r_i)\}$
2. 估计Q目标值 $y_i \leftarrow r(s_i, a_i) + \gamma max_{a'} Q_\varphi(s_i', a_i')$
3. 更新参数 $\varphi \leftarrow \varphi - \alpha \frac{dQ_\varphi}{d\varphi}(s_i, a_i)(Q_\varphi(s_i, a_i) - y_i)$

问题2：

- Q学习理论上无法证明收敛，因为更新并非是梯度下降

$y_i \leftarrow r(s_i, a_i) + \gamma max_{a'} Q_\varphi(s_i', a_i')$

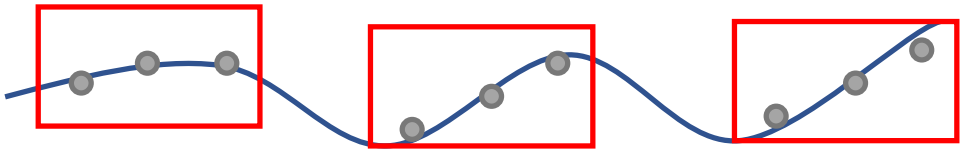
目标值同样依赖参数φ，一直在变化；

计算梯度时却忽略了该项；

DQN (Deep Q Learning)

在线Q学习步骤：

1. 初始执行动作a_i，得到观测数据 $\{(s_i, a_i, s_i', r_i)\}$
2. 估计Q目标值 $y_i \leftarrow r(s_i, a_i) + \gamma max_{a'} Q_\varphi(s_i', a_i')$
3. 更新参数 $\varphi \leftarrow \varphi - \alpha \frac{dQ_\varphi}{d\varphi}(s_i, a_i)(Q_\varphi(s_i, a_i) - y_i)$

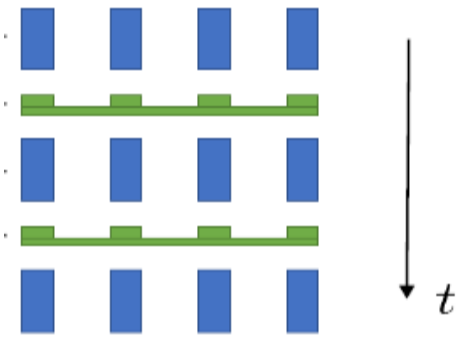


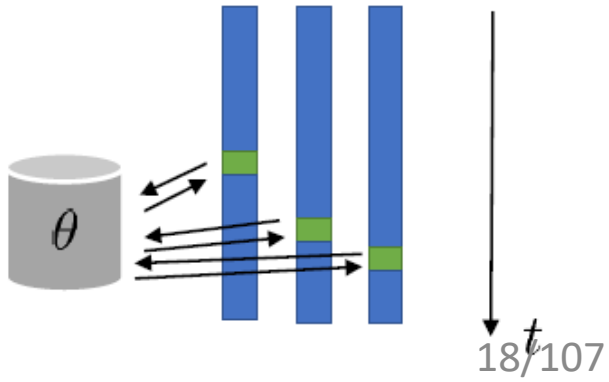
过拟合到局部数据

同步并行-on policy

异步并行

收集 $\{(s_i, a_i, s_i', r_i)\}$
更新参数 φ
收集 $\{(s_i, a_i, s_i', r_i)\}$
更新参数 φ





DQN (Deep Q Learning)

在线Q学习步骤：

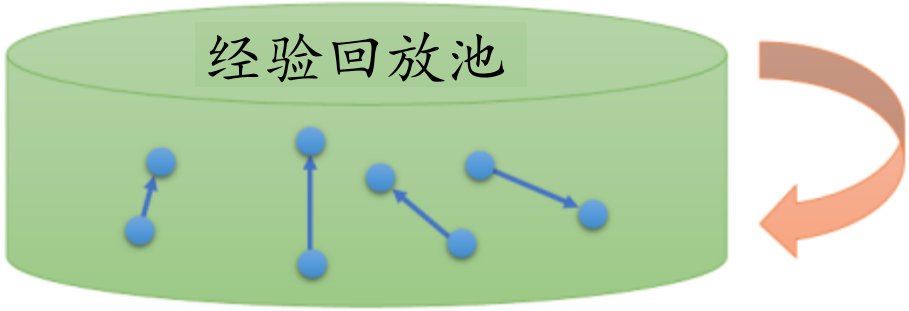
1. 初始执行动作a_i，得到观测数据 $\{(s_i, a_i, s_i', r_i)\}$
2. 估计Q目标值 $y_i \leftarrow r(s_i, a_i) + \gamma max_{a'} Q_\varphi(s_i', a_i')$
3. 更新参数 $\varphi \leftarrow \varphi - \alpha \frac{dQ_\varphi}{d\varphi}(s_i, a_i)(Q_\varphi(s_i, a_i) - y_i)$

传统Q学习步骤：

1. [redacted]

N

2. 估计Q值 $y_i \leftarrow r(s_i, a_i) + \gamma max_{a_i'} Q_\varphi(s_i', a_i')$
3. 更新参数 $\varphi \leftarrow \arg\min_\varphi \frac{1}{2} \Sigma_i \| Q_\varphi(s_i, a_i) - y_i \|$



DQN (Deep Q Learning)

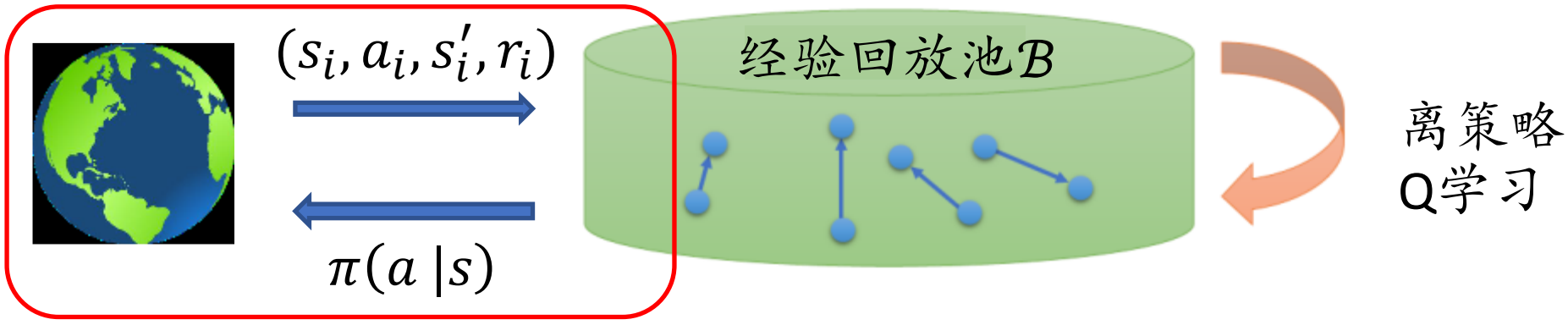
经验回放下的Q学习步骤：

1. 在经验池 $\mathcal{B}$ 采样batch数据 $\{(s_i, a_i, s_i', r_i)\}$
2. 更新参数 $\varphi \leftarrow \varphi - \alpha \sum_i \frac{dQ_\varphi}{d\varphi}(s_i, a_i)(Q_\varphi(s_i, a_i) - [r(s_i, a_i) + \gamma max_{a'} Q_\varphi(s_i', a_i')])$

- 样本相关性大幅降低
- Batch训练降低方差

- 如何得到经验池数据？
  周期性存储并更新经验池数据

利用历史策略收集样本，存入样本回放池，执行2-3步迭代



DQN (Deep Q Learning)

经验回放下的Q学习步骤：

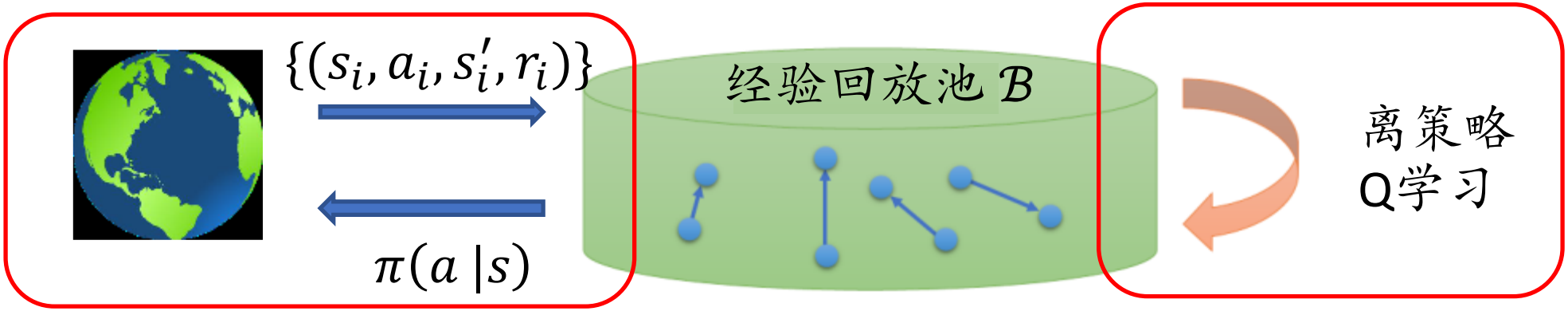
1. 利用某些策略收集样本 $\{(s_i, a_i, s_i', r_i)\}$，加入样本池 $\mathcal{B}$

N

2.在经验池 $\mathcal{B}$ 采样batch数据

3. 更新参数 $\varphi \leftarrow \varphi - \alpha \sum_i \frac{dQ_\varphi}{d\varphi}(s_i, a_i)(Q_\varphi(s_i, a_i) - \left[r(s_i, a_i) + \gamma max_{a_i'} Q_\varphi(s_i', a_i')\right])$

通过经验回放缓解了数据iid的问题



这就是2013年DQN最初的结构

DQN (Deep Q Learning)

经验回放下的Q学习步骤：

1. 利用某些策略收集样本 $\{(s_i, a_i, s_i', r_i)\}$，加入样本池 $\mathcal{B}$

N

2. 在经验池 $\mathcal{B}$ 采样batch数据

3. 更新参数 $\varphi \leftarrow \varphi - \alpha \sum_i \frac{dQ_\varphi}{d\varphi}(s_i, a_i)(Q_\varphi(s_i, a_i) - [r(s_i, a_i) + \gamma max_{a_i'} Q_\varphi(s_i', a_i')$

没有目标值中的梯度传递，
而目标值又一直在变化

导致DQN的训练不稳定

DQN (Deep Q Learning)

经验回放+目标网络结合的Q学习步骤：

1. 保存目标Q网络参数$\varphi' \leftarrow \varphi$

N

2.利用某些策略收集样本 $\{(s_i, a_i, s_i', r_i)\}$，加入样本池$\mathcal{B}$

K

3.在经验池$\mathcal{B}$ 采样batch数据

4. 更新参数$\varphi \leftarrow \varphi - \alpha \sum_i \frac{dQ_\varphi}{d\varphi}(s_i, a_i)(Q_\varphi(s_i, a_i) - \left[r(s_i, a_i) + \gamma max_{a_i'} Q_{\varphi'}(s_i', a_i')\right])$

内层类似于一个回归任务

内层迭代的时候目标值不再发生改变

DQN (Deep Q Learning)

经典DQN算法：

1. 利用ε-greedy策略执行动作a_i，收集样本$\{(s_i, a_i, s_i', r_i)\}$，加入经验池$\mathcal{B}$

2.在经验池$\mathcal{B}$ 中采样batch数据$\{(s_j, a_j, s_j', r_j)\}$

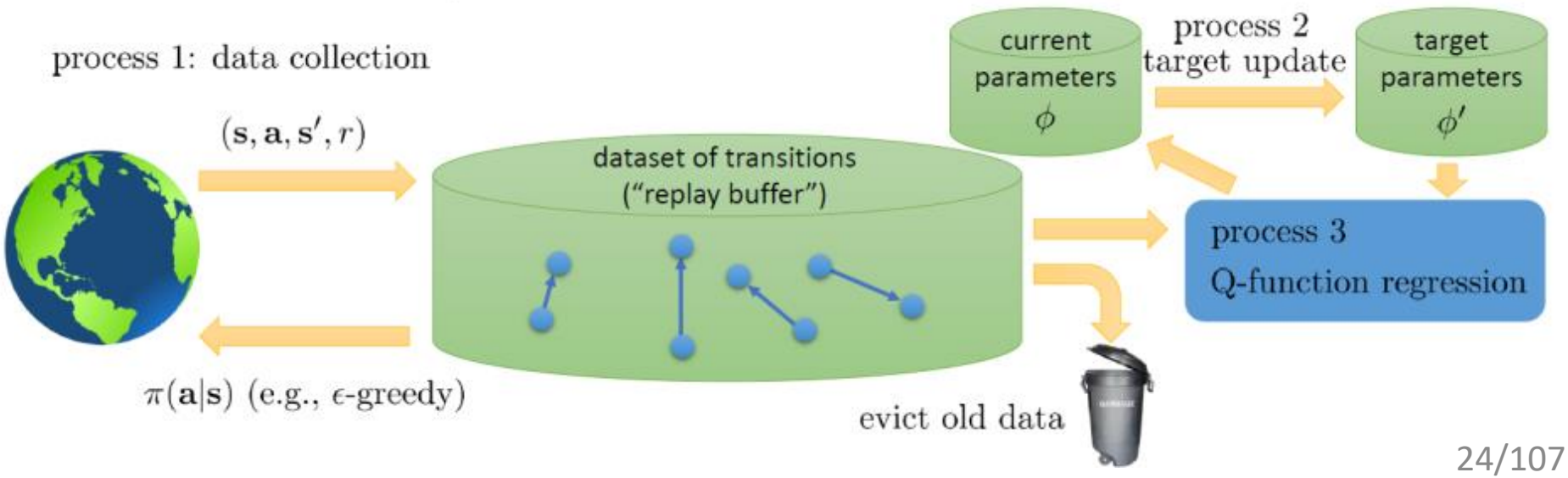
3. 计算目标网络估计值 $y_j \leftarrow \begin{cases} y_j \leftarrow r_j ; j\text{为终止时刻} \\ r(s_j,a_j)+\gamma max_{a'} Q_{\varphi'}(s_j', a_j') \end{cases}$

4. 更新Q网络参数 $\varphi \leftarrow \varphi - \alpha \sum_j \frac{dQ_\varphi}{d\varphi}(s_j,a_j)(Q_\varphi(s_j,a_j) - y_j)$

5. 更新目标网络参数 $\varphi' \leftarrow \varphi$

K=1

N=1



DQN (Deep Q Learning)

这就是2015年DQN的伪代码

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-}=\theta$

For episode $=1, M$ **do**

  Initialize sequence $s_1=\{x_1\}$ and preprocessed sequence $\phi_1=\phi(s_1)$

  For $t=1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t=\text{argmax}_a Q(\phi(s_t),a;\theta)$

    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 D

    Sample random minibatch of transitions $(\phi_j,a_j,r_j,\phi_{j+1})$ from D

    Set $y_j=\begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j+\gamma \max_{a'} \hat{Q}(\phi_{j+1},a';\theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j-Q(\phi_j,a_j;\theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q}=Q$

  End For

End For

初始化

步骤1

步骤2-3

步骤4-5

DQN-多步回报

- DQN的多步回报版本

- 标准DQN $y_j \leftarrow r(s_j, a_j) + \gamma Q_{\varphi'}(s'_j, \text{argmax} Q_{\varphi'}(s'_j, a'_j))$

如果$Q_{\varphi'}$逼近器很差，
r的好坏影响会很大

如果$Q_{\varphi'}$逼近器很好，
s'_j和a'_j的值又非常关键

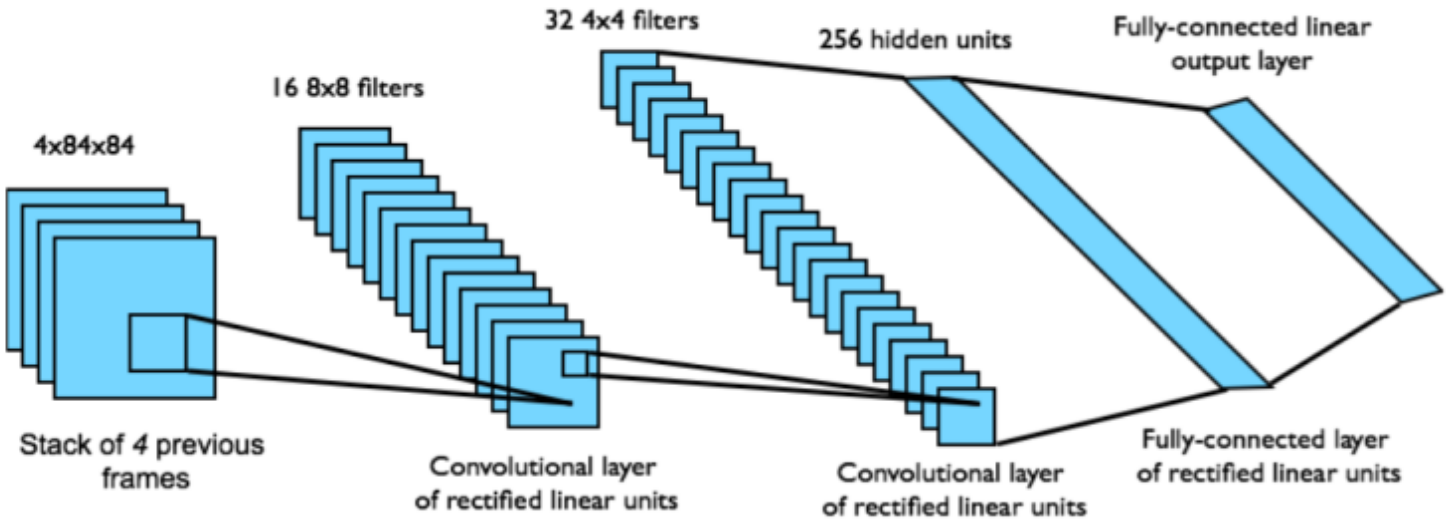
想要得到更低的方差，N步回报估计器

$$y_{j,t} \leftarrow \sum_{t'=t}^{t+N-1} \gamma^{t-t'} r(s_{j,t'}, a_{j,t'}) + \gamma^N \max_{a_{j,t+N}} Q_{\varphi'}(s'_{j,t+N}, a'_{j,t+N}))$$

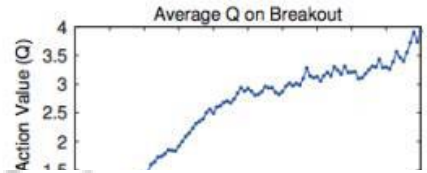
DQN (Deep Q Learning)

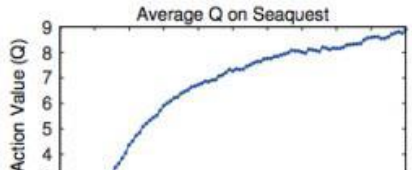
DQN in Atari

- 以图片s为状态的端到端学习(估计Q(s,a))
- 输入状态s是最近4帧的原始像素图像
- 输出为离散动作空间对应的Q(s,a)
- 奖赏值为每次的得分

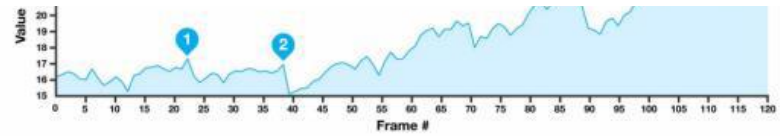


DQN (Deep Q Learning)





Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0



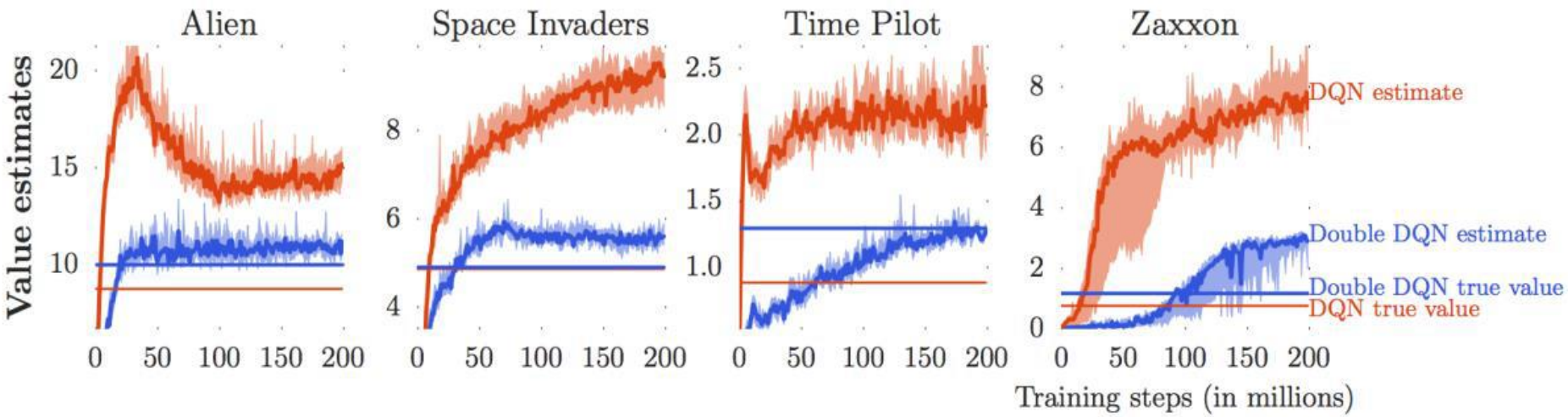
问题思考：1.Q值估计是否准确？

2. 经验池均匀采样的效率是否好？

源码：sites.google.com/a/deepmind.com/dqn/

DQN (Deep Q Learning)

- 问题1：DQN中Q值估计的准确么？



答案是：Q值估计并不准确

Double DQN

- DQN中的过估计问题

目标网络估计值：$y_i \leftarrow r(s_i, a_i) + \gamma max_{a_i'} Q_{\varphi'}(s_i', a_i')$

$$max_{a_i'} Q_{\varphi'}(s_i', a_i') = Q_{\varphi'}\left(s_i', \underset{a_i'}{\text{argmax}}\, Q_{\varphi'}(s_i', a_i')\right)$$

目标Q网络值估计　动作的选择

- TD目标值中的max操作，将引入一个正向的偏差，导致下一时刻的目标值存在过估计

假设存在两个随机变量X_1, X_2：

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

X_1的期望为0.5　X_2的期望为1，左项的期望不小于1

Double DQN

- DQN中的过估计问题

$$max_{a_i'} Q_{\varphi'}(s_i', a_i') = Q_{\varphi'}\left(s_i', \underset{a_i'}{\operatorname{argmax}}\, Q_{\varphi'}(s_i', a_i')\right)$$

目标网络值估计　　动作的选择

如果这两项不再相关，将大大减轻过估计问题

解决思路：使用不同的网络来分别计算目标Q网络值和选择动作

- Double DQN：利用两个网络

$$Q_{\varphi_A}(s, a) \leftarrow r(s, a) + \gamma Q_{\varphi_B}\left(s', \underset{a_i'}{\operatorname{argmax}} Q_{\varphi_A}(s', a')\right)$$

$$Q_{\varphi_B}(s, a) \leftarrow r(s, a) + \gamma Q_{\varphi_A}\left(s', \underset{a_i'}{\operatorname{argmax}} Q_{\varphi_B}(s', a')\right)$$

Double DQN

- DQN中的过估计问题

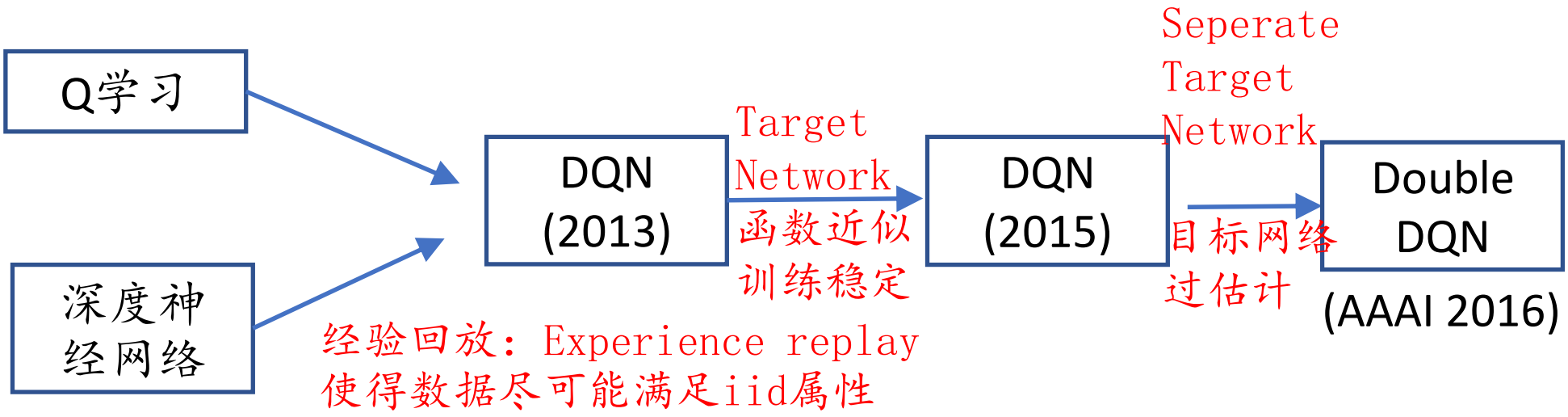
  如何利用DQN得到两个Q网络呢？

  思路：使用当前Q网络Q_φ和目标Q网络$Q_{\varphi'}$

  - 标准DQN $y_j \leftarrow r(s_j, a_j) + \gamma Q_{\varphi'}(s'_j, \text{argmax} Q_{\varphi'}(s'_i, a'_i))$
  - Double DQN $y_j \leftarrow r(s_j, a_j) + \gamma Q_{\varphi'}(s'_j, \text{argmax} Q_{\varphi}(s'_i, a'_i))$

  ■ 利用当前Q网络来选择动作；
  ■ 利用目标Q网络值来估计目标值

Double DQN



Algorithm 1 Double Q-learning

Initialize Q^A, Q^B, s
repeat
  Choose a, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
  Choose (e.g. random) either UPDATE(A) or UPDATE(B)
  if UPDATE(A) **then**
    Define $a^* = \arg\max_a Q^A(s',a)$
    $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*) - Q^A(s,a)\right)$
  else if UPDATE(B) **then**
    Define $b^* = \arg\max_a Q^B(s',a)$
    $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s',b^*) - Q^B(s,a))$
  end if
  $s \leftarrow s'$
until end

- Q_{φ_A}和Q_{φ_B}两个网络是相对独立的，一般不同一时刻更新
- 每次随机选择一个网络更新

Prioritized Experience Replay (ICLR 2016)

- 问题2： 经验池均匀采样的效率是否足够好？

  - DQN 算法的一个重要改进是Experience Replay
  - 训练时从经验池中均匀采样

Prioritized Experience Replay 就是维护了一个带优先级的经验回放

- 每个样本的价值不同，采样的优先级应该不同
- 利用TD 误差去衡量优先级，TD误差大样本价值高，从而剔除价值低的样本
- 经验池的样本保持“先入先出”原则

TD 误差： $|r(s_i,a_i)+\gamma max_{a_i'}Q_{\varphi'}(s_i',a_i')-Q_{\varphi}(s_i,a_i)|$

Prioritized Experience Replay (ICLR 2016)

优先级经验回放带来的问题：

- TD 误差对噪声敏感
- TD 误差小的样本长时间得不到不更新
- 过分关注TD 误差大的样本，丧失了样本多样性

- 使用某种分布采样样本(优先级高的样本被长期选择)，会引入偏差Bias

Prioritized Experience Replay (ICLR 2016)

优先级确定与根据优先级采样

（1）优先级确定方法（TD误差）

比例优先化：$p_i = |\delta_i| + \epsilon$

通过加入一个小的噪音项，增加多样性，确保非零

排序优先化 $p_i = \frac{1}{\text{rank}(i)}$

（2）优先级采样概率

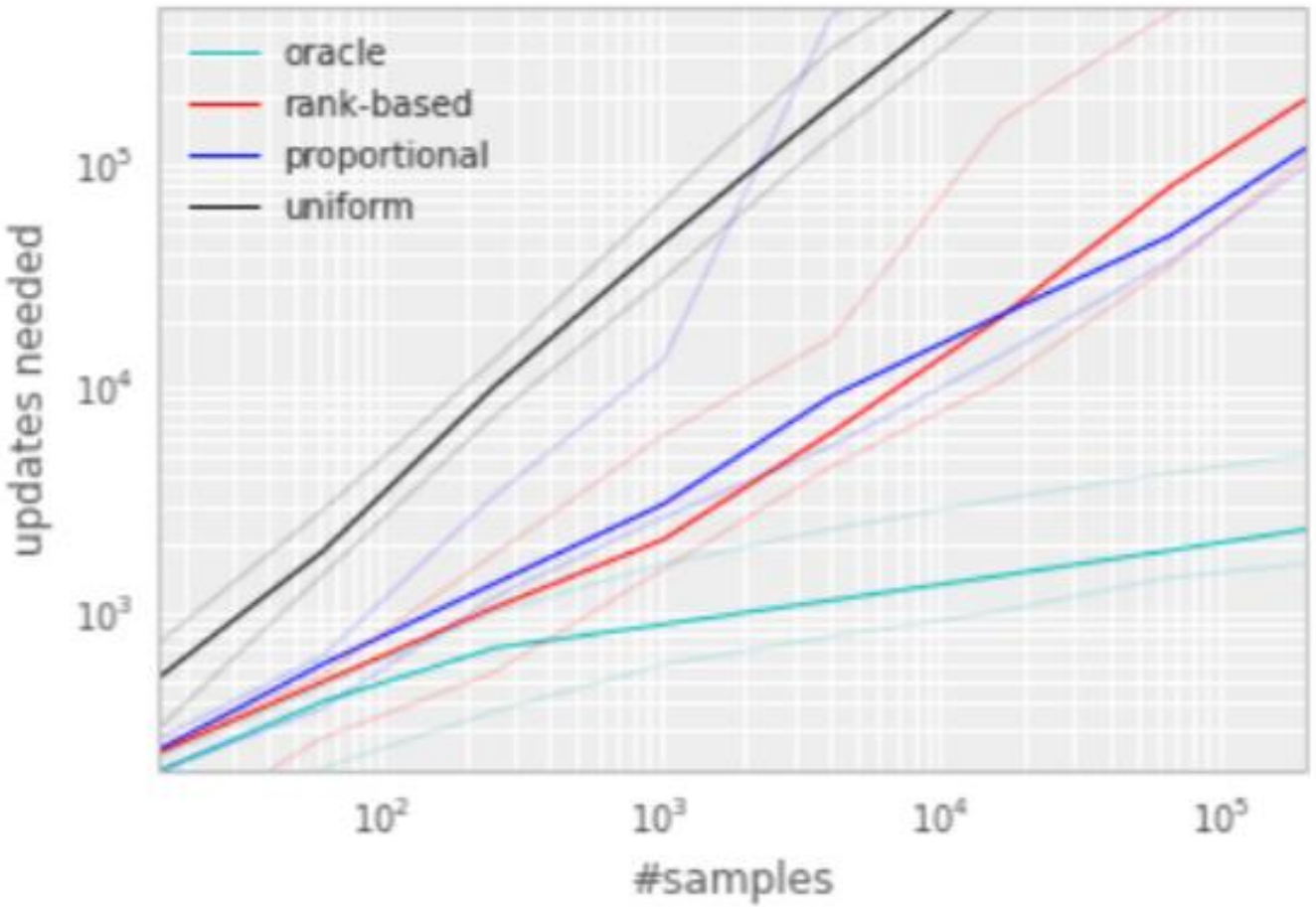
随机优先化

采样概率 α--- 样本优先级使用程度系数

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

α系数的选择决定了优先级的使用程度，α=0为标准DQN均匀采样

Prioritized Experience Replay (ICLR 2016)



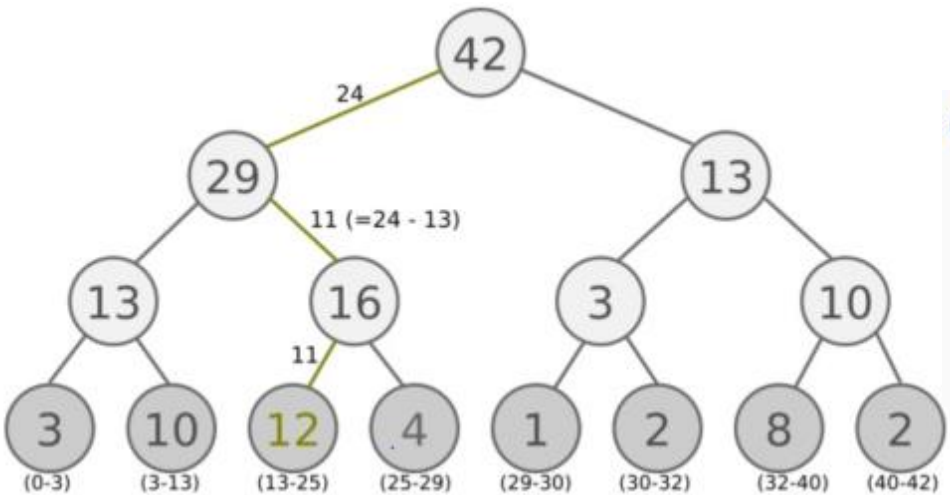
Prioritized Experience Replay (ICLR 2016)

每一次都需要更新经验池中所有样本的优先级，遍历整个经验池选择优先值高的样本，计算量较大的问题

存储、采样方式：Sum-Tree

[0-7] [7-14] [14-21]
[21-28] [28-35] [35-42]

就将根节点的总的优先值除以batch_size，划分为batch_size个区间




```
def retrieve(n, s):
  if n is leaf_node: return n

  if n.left.val >= s: return retrieve(n.left, s)
  else: return retrieve(n.right, s - n.left.val)
```

Prioritized Experience Replay (ICLR 2016)

优先级高的样本具有更高利用率带来的“有偏”

引入重要性采样权重来平衡“有偏”问题

重要性采样退火 $w(i) = \left(\frac{1}{N}\frac{1}{P(i)}\right)^{\beta}$

一般需要归一化

$$\Delta \leftarrow \Delta + w(i)\delta_i \nabla_\theta Q(s_{i-1}, a_{i-1})$$

β: $0 \rightarrow 1$

前期注重优先级高的样本的利用率
后期注重无偏性

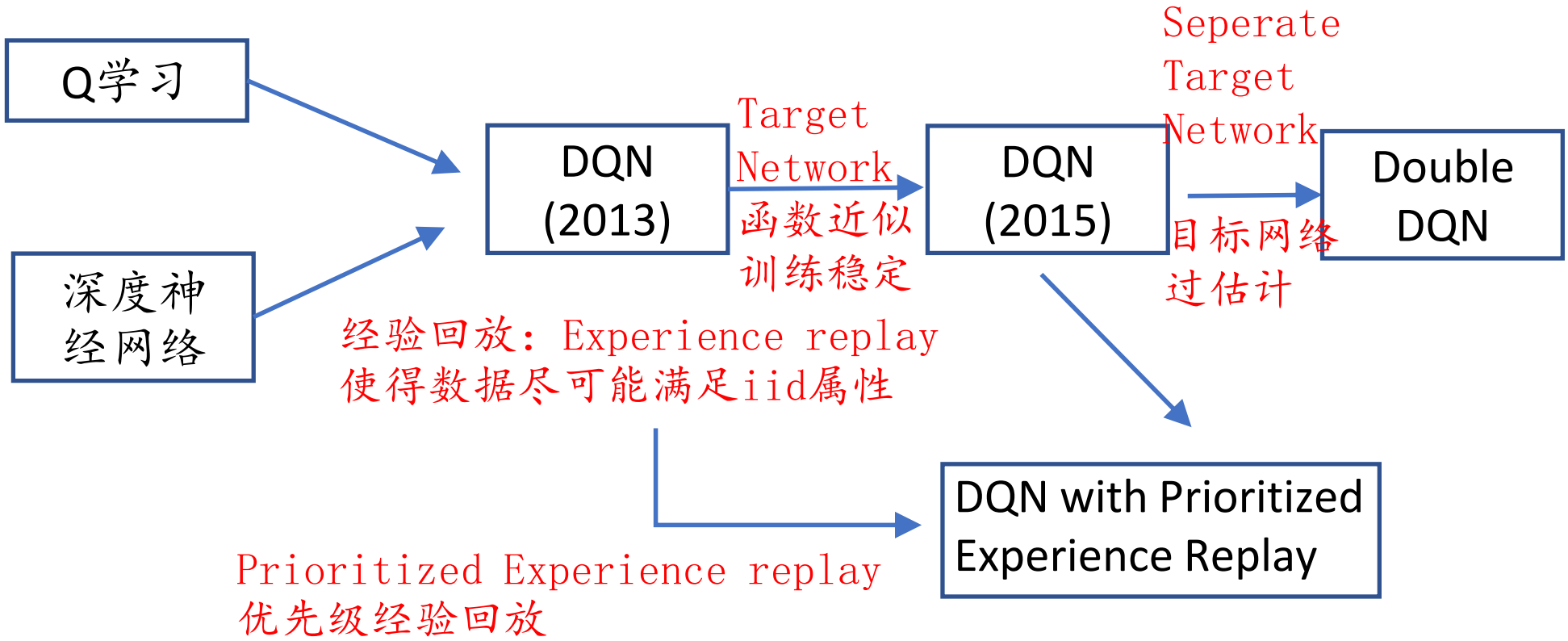
经典的强化学习的场景下，更新的无偏性是训练最后接近收敛最重要的部分

Prioritized Experience Replay (ICLR 2016)

Algorithm 1 Double DQN with proportional prioritization

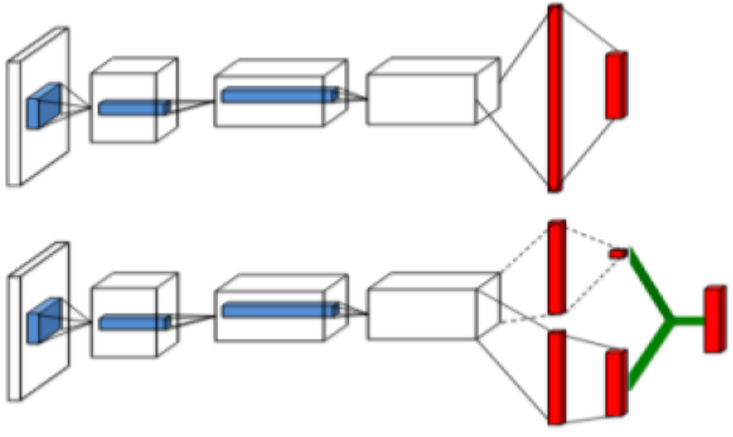
Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ **to** T **do**
  Observe S_t, R_t, γ_t
  Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
  if $t \equiv 0 \mod K$ **then**
    for $j = 1$ **to** k **do**
      Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
      Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
      Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
      Update transition priority $p_j \leftarrow |\delta_j|$
      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
    end for
    Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
    From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
  end if
  Choose action $A_t \sim \pi_\theta(S_t)$
end for

Prioritized Experience Replay (ICLR 2016)



Dueling-DQN(ICML 2016)

- 将Q 函数分解成V 函数和优势(A) 函数



•特征提取层参数共享，
•在DQN全连接层将网络输出一分为二

$$Q(s,a) = V(s) + A(s,a)$$

- 不依赖动作的值函数$V(s)$
- 依赖动作的优势函数$A(s,a)$

- 优势函数的定义为：

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

优化神经网络的结构来优化算法

Dueling-DQN(ICML 2016)

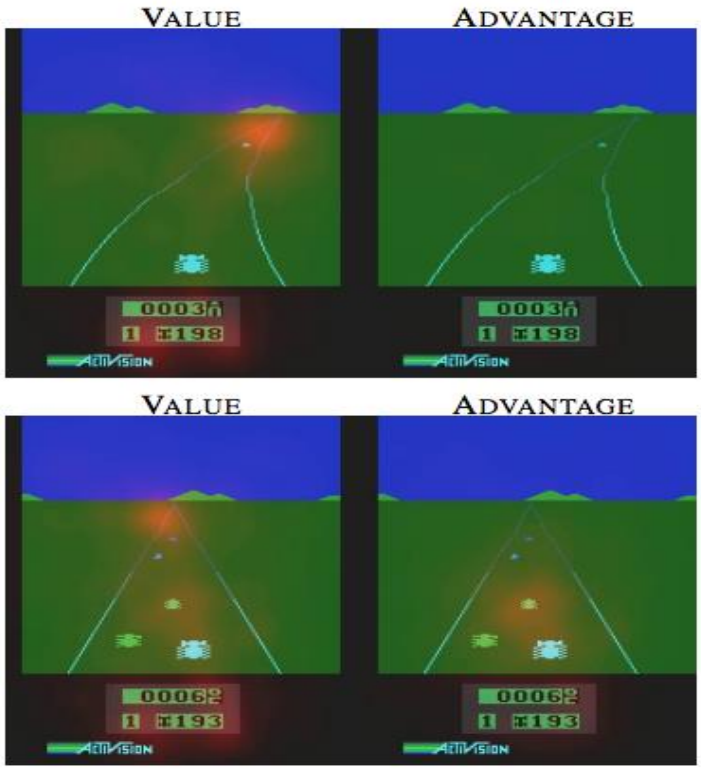
- 为什么将Q网络分解成动作依赖和动作不依赖的两个网络

  - 对于很多状态并不需要估计每个动作的值，增加了V函数的学习机会
  - V函数的泛化性能好，当有新动作加入时，并不需要重新学习
  - 减少了Q 函数由于状态和动作维度差导致的噪声和突变

Dueling-DQN(ICML 2016)

值函数：当前方没有车辆时，重点关注道路状态，动作变化对Q值影响不大，应该着重学习V函数

优势函数：当周围车辆密集时，动作变化对Q值影响增大，次数优势函数的学习可有助于估计更准确的Q函数



Dueling-DQN(ICML 2016)

原始公式：

$$Q(s,a) = V(s) + A(s,a)$$

改进版本：

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a' \in |\mathcal{A}|} A(s,a';\theta,\alpha) \right)$$

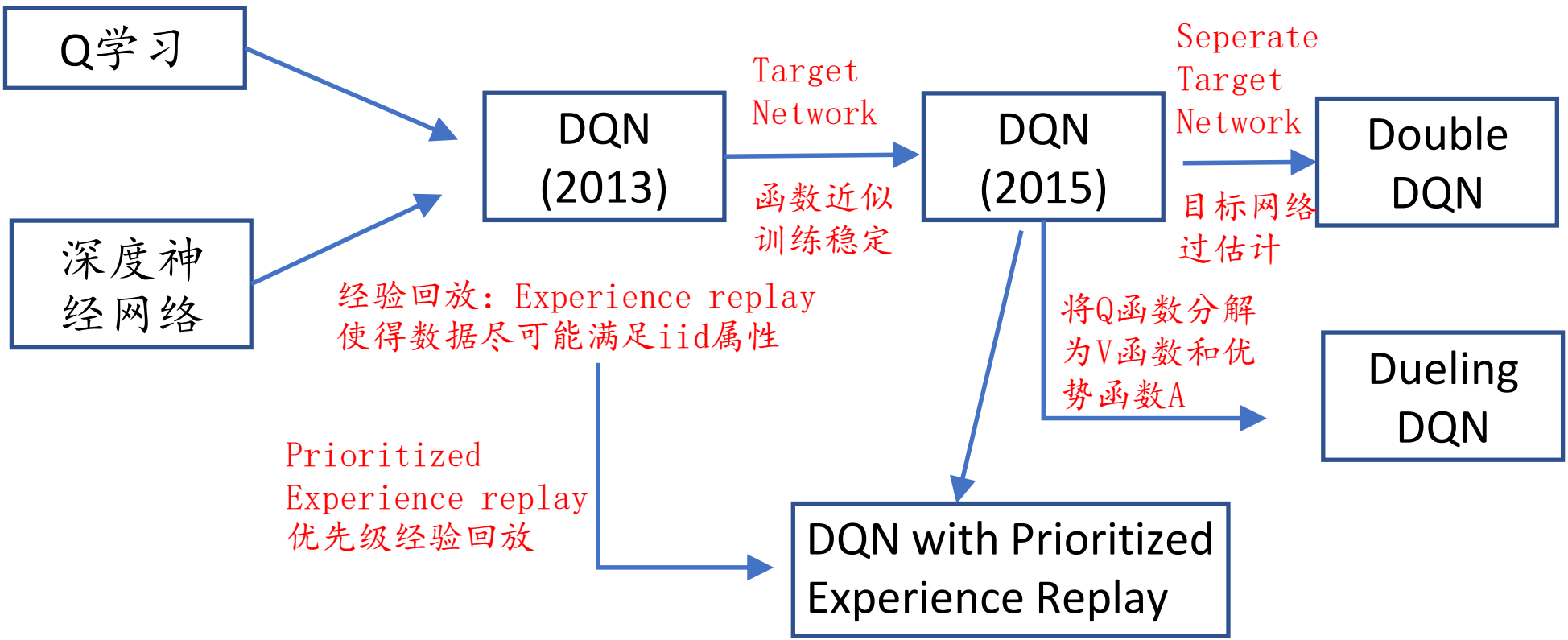
$$a^* = \arg\max_{a' \in \mathcal{A}} Q(s,a';\theta,\alpha,\beta)$$

$$Q(s,a^*;\theta,\alpha,\beta) = V(s;\theta,\beta)$$

平均值版本：

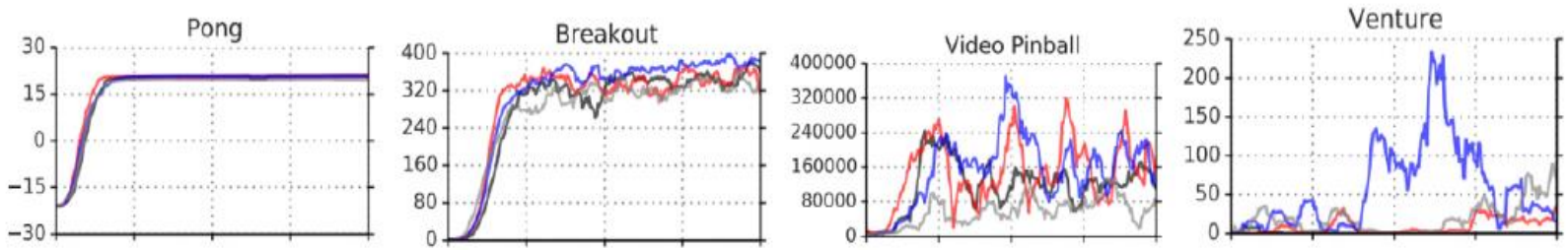
$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a';\theta,\alpha) \right)$$

Dueling-DQN(ICML 2016)



训练技巧

1. 算法确定后，首先在简单可信的任务上训练(比如Atrai)，确保你的配置是正确的



2. 一般情况下，经验池越大，越有助于提升稳定性

3. 收敛过程曲折震荡，需要保持耐心(有时性能比随机动作还糟)

4. 初始epsilon值可以设置大一些，逐渐减小

总结

1. DQN(Deep Q learning)
   经验回放
   目标网络

2. Double DQN
   利用两个网络解决过估计问题

3. Prioritized Experience Replay
   利用TD误差对样本池数据进行筛选

4. Dueling DQN
   Q函数分解为值函数V和优势函数A

课程目录

基于策略的深度强化学习

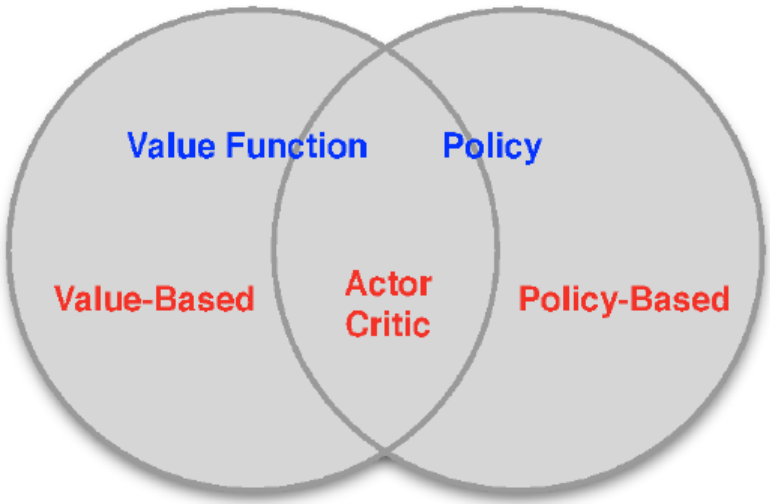
回顾

- 基于值函数的RL
  1. 学习值/动作-值函数逼近器

$$V_\theta(s) \approx V^\pi(s)$$

$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

  2. 隐式的策略（ε-greedy等）



- 基于策略的RL
  没有价值函数
  学习策略逼近器 $\pi_\theta(s,a) = \mathrm{p}[a|s,\theta]$

- 基于Actor-Critic的RL
  学习价值逼近器
  学习策略逼近器

基于策略的深度强化学习

目标：找到最优策略$\pi_\theta(s,a)$，即找到最好的参数θ

如何评估策略$\pi_\theta(s,a)$的好坏？

针对episodic环境：利用

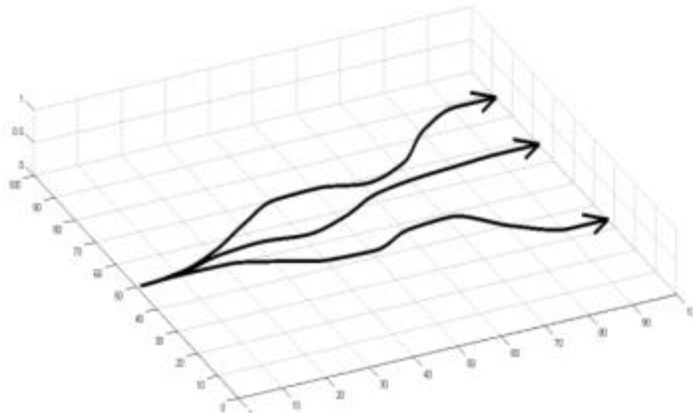
$$J(\theta) = E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i\sum_t r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})$$

用N条样本近似期望回报

第i条轨迹上的样本累加奖赏

$$\theta^\star = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right]}_{J(\theta)}$$

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



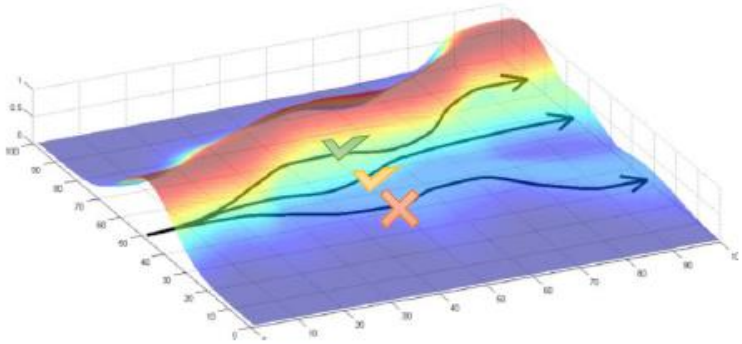
优化问题：寻找最大化$J(\theta)=J^{\pi_\theta}$的策略参数θ，与基于值函数不同的是，这里的θ直接就是策略的参数

REINFORCE

- 策略梯度：基于梯度的优化方法

$$J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[r(\tau)] = \int \pi_\theta(\tau) r(\tau) d\tau$$

$$\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)$$



$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau = \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau = E_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

得分函数

$$\pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) = \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \nabla_\theta \pi_\theta(\tau)$$

$$\nabla_\theta \left[\log p(\mathbf{s}_1) + \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) + \log p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \right]$$

梯度

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

REINFORCE

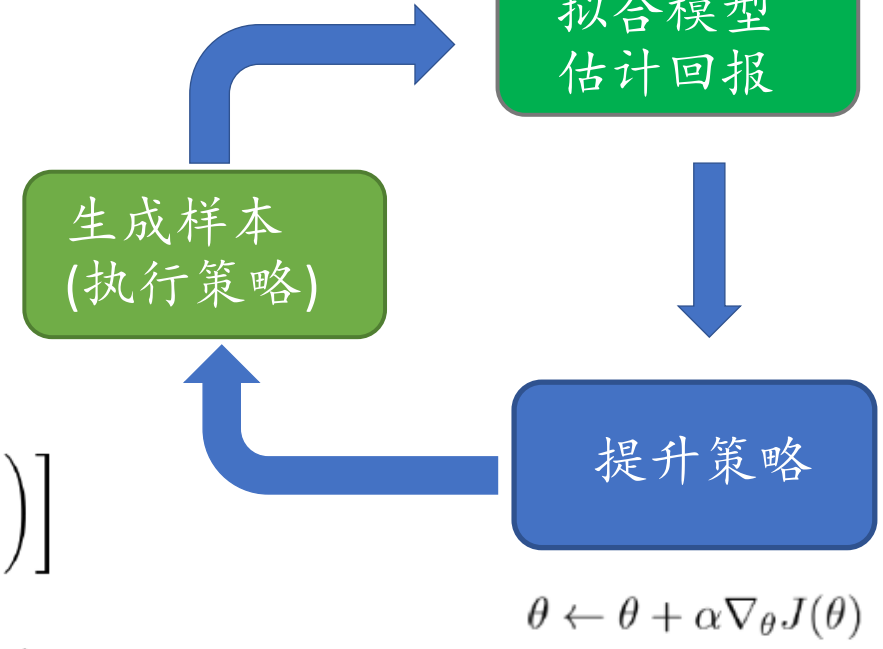
- 策略梯度

$t' < t$ 时刻之前的奖励与t时刻的策略无关

REINFORCE 算法：

1. 基于策略$\pi_\theta(s,a)$采样轨迹样本
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) \left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}^i, \mathbf{a}_{t'}^i)\right)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

$$\hat{Q}^\pi(\mathbf{x}_t, \mathbf{u}_t) = \sum_{t'=t}^{T} r(\mathbf{x}_{t'}, \mathbf{u}_{t'})$$



$$J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N} \sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

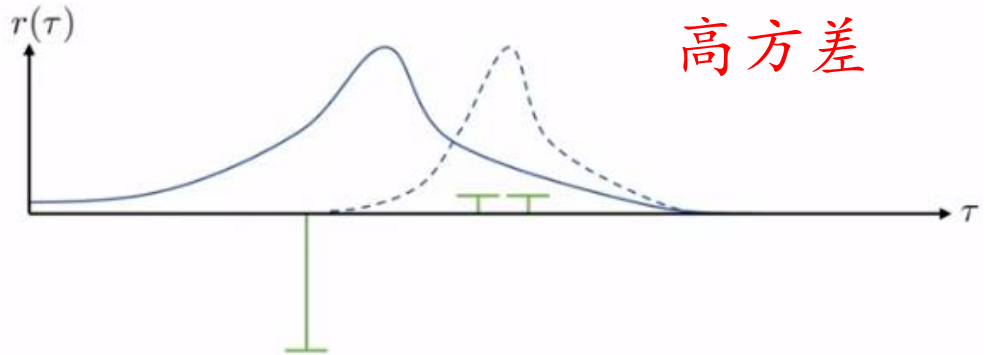
$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

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

REINFORCE

蒙特卡洛策略梯度(REINFORCE)

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \log \pi_\theta(\tau) r(\tau)$$



减少方差的技巧

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

- $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \log \pi_\theta(\tau)\,[r(\tau) - b]$ $\quad b = \frac{1}{N}\sum_{i=1}^{N} r(\tau)$
- $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\left(\sum_{t'=t}^{T} r(s_{i,t'},a_{i,t'})\right) \rightarrow \hat{Q}_{i,t}$

Actor-Critic

- 蒙特卡洛策略梯度法具有高方差
- 使用critic来评估动作 -价值函数

$$Q_{\mathbf{w}}(s,a) \approx Q^{\pi_\theta}(s,a)$$

- Actor-critic 算法包含两组参数

Critic 更新动作 -价值函数参数 $\mathbf{w}$

Actor 更新策略参数 θ, 更新方向由 critic 提供

- Actor-critic 算法使用的是近似策略梯度

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_{\mathbf{w}}(s,a)]$$

Actor-Critic

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, G_t] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, Q_{\mathbf{w}}(s,a)] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, A_{\mathbf{w}}(s,a)] && \text{优势Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, \delta] && \text{TD Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, \delta e] && \text{TD}(\lambda)\ \text{Actor-Critic}
\end{aligned}
$$

1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t.
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
4. $Q^\pi(s_t, a_t)$: state-action value function.
5. $A^\pi(s_t, a_t)$: advantage function.
6. $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$: TD residual.

AsyncRL

- 如何将基于策略的RL 方法和深度神经网络有效结合在一起?
- DQN 使用经验回放技术, 但是标准的策略梯度(PG) 方法是在策略的
  - 需要根据当前策略采样数据
  - 也存在比较好的离策略方法

  例如ACER(Wang et al., 2016)和PGQL(O'Donoghue et al., 2016)

- 思路:

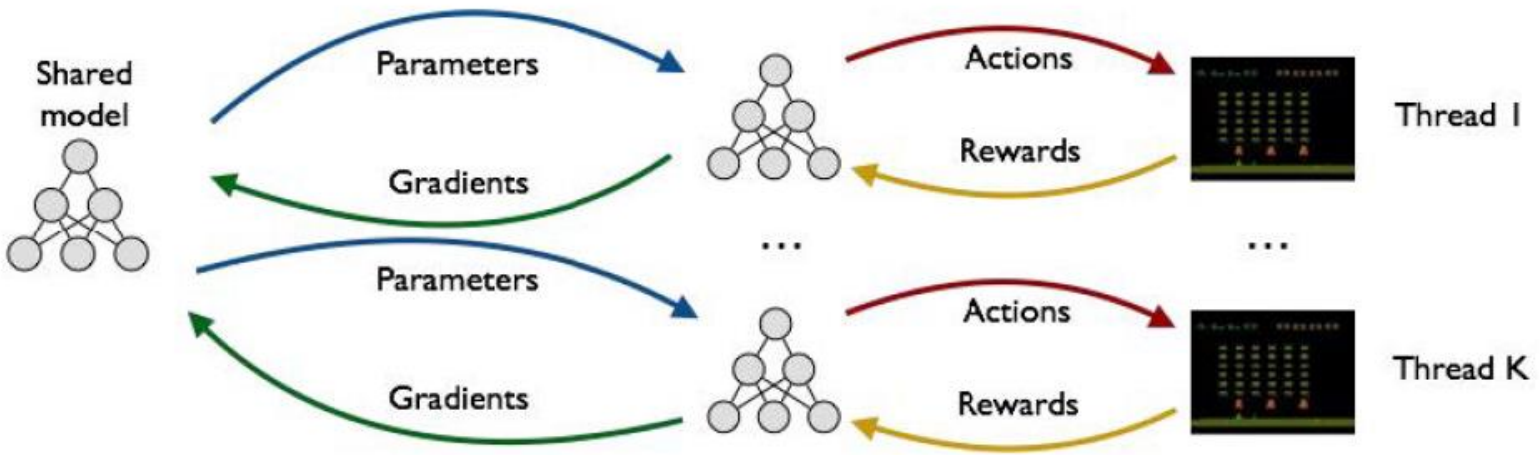
  经验回放（1.离策略 2.占用大量内存）

  并行训练（1.在策略 2.占用线程资源）

减少数据相关性

AsyncRL

- RL 智能体的异步训练
  - 多个 Actor 使用CPU 线程和参数共享实现并行学习
  - 在线Hogwild!-式的异步更新 (Recht et al., 2011, Lian et al., 2015)
  - 没有回放? 多个 Actor 并行学习同样具有稳定学习过程的效果
  - 可以使用的 RL 算法: 在策略/离策略, 基于价值/基于策略



异步：不同的线程在不同时间达到“更新点”的时候会独立地更新梯度

AsyncRL

异步的1-步Q学习

- 多个 actor 并行计算 1-步更新

$$y \leftarrow r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

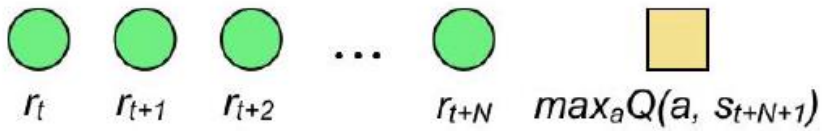
$$\Delta\theta \leftarrow \Delta\theta + \frac{\partial (y - Q(s, a; \theta))^2}{\partial \theta}$$

- 梯度会在更新之前像使用 minibatch 一样累积

AsyncRL

异步的N-步Q学习

- 均匀的混合使用 1 到 N 步的奖励和来实现 Q 学习算法



$$y \leftarrow \sum_{k=0}^{N-1} \gamma^k r_{t+k} + \gamma^N \max_{a'} Q(s_{t+N}, a'; \theta^-)$$

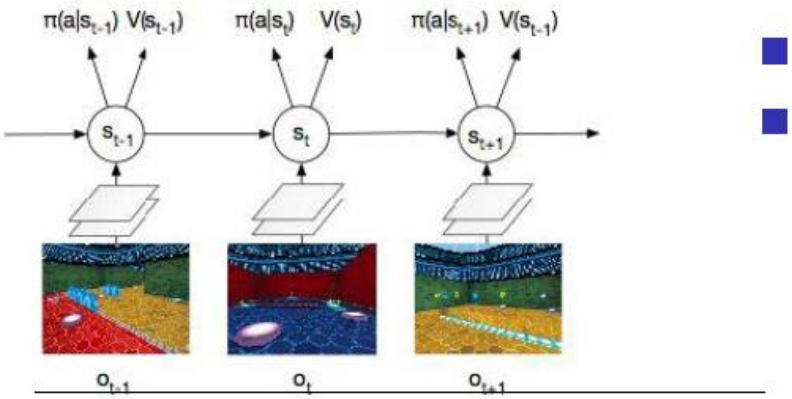
$$\Delta\theta \leftarrow \Delta\theta + \frac{\partial (y - Q(s_t, a_t; \theta))^2}{\partial \theta}$$

- 对 "Incremental multi-step Q-learning" (Peng & Williams, 1995) 工作的扩展

Async Advantage Actor-Critic (A3C)

- 智能体学习一个策略和一个状态的价值函数
- 使用 n-步回报来减少方差
- 对策略梯度乘上一个优势的估计

$$\nabla_\theta \log \pi(a_t|s_t,\theta)\left(\sum_{k=0}^{N}\gamma^k r_{t+k}+\gamma^{N+1}V(s_{t+N+1})-V(s_t)\right)$$



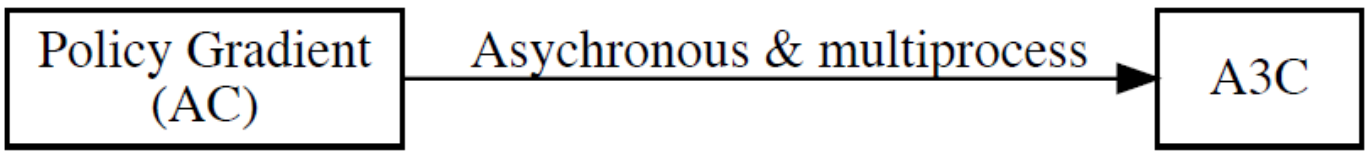
- 对价值使用 n-步 TD 学习算法
- 相当于是最小化

$$\left(\sum_{k=0}^{N}\gamma^k r_{t+k}+\gamma^{N+1}V(s_{t+N+1};\theta^-)-V(s_t;\theta)\right)$$

"Asynchronous Methods for Deep Reinforcement Learning", Mnih et al. (2016)

Async Advantage Actor-Critic (A3C)

- 异步训练架构
  - 通过创建多个agent 在多个环境执行异步学习构建batch
    - 来自不同环境的样本无相关性，有助于满足iid条件
    - 直觉上不同线程使用了不同的探索策略，增加了探索量



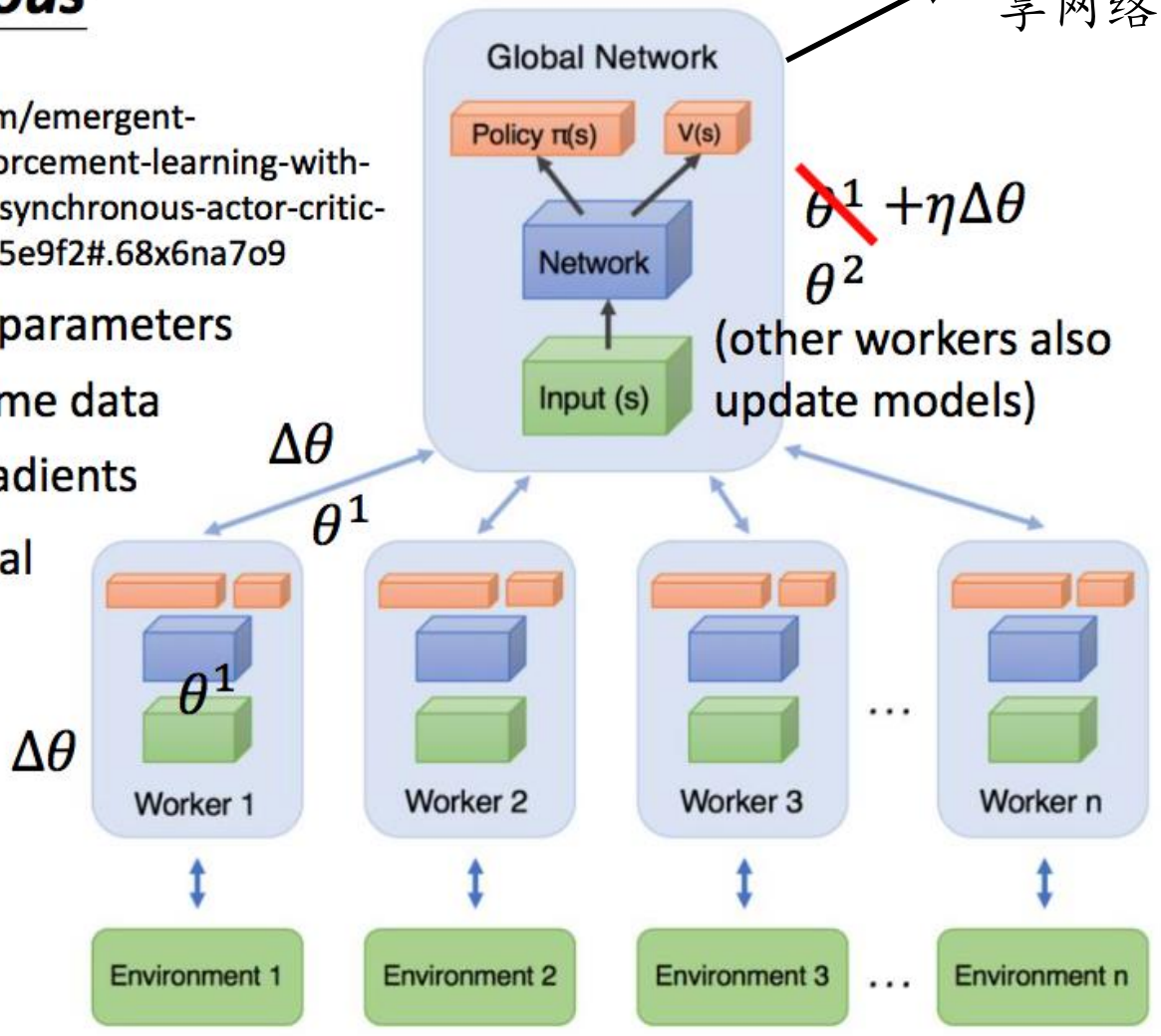
- 网络结构优化

Async Advantage Actor-Critic (A3C)

Asynchronous

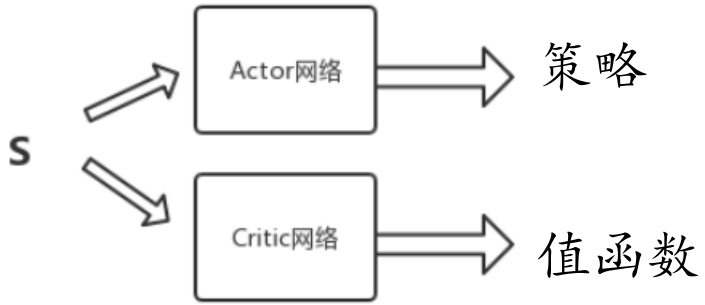
Source of image:
https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2#.68x6na7o9

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

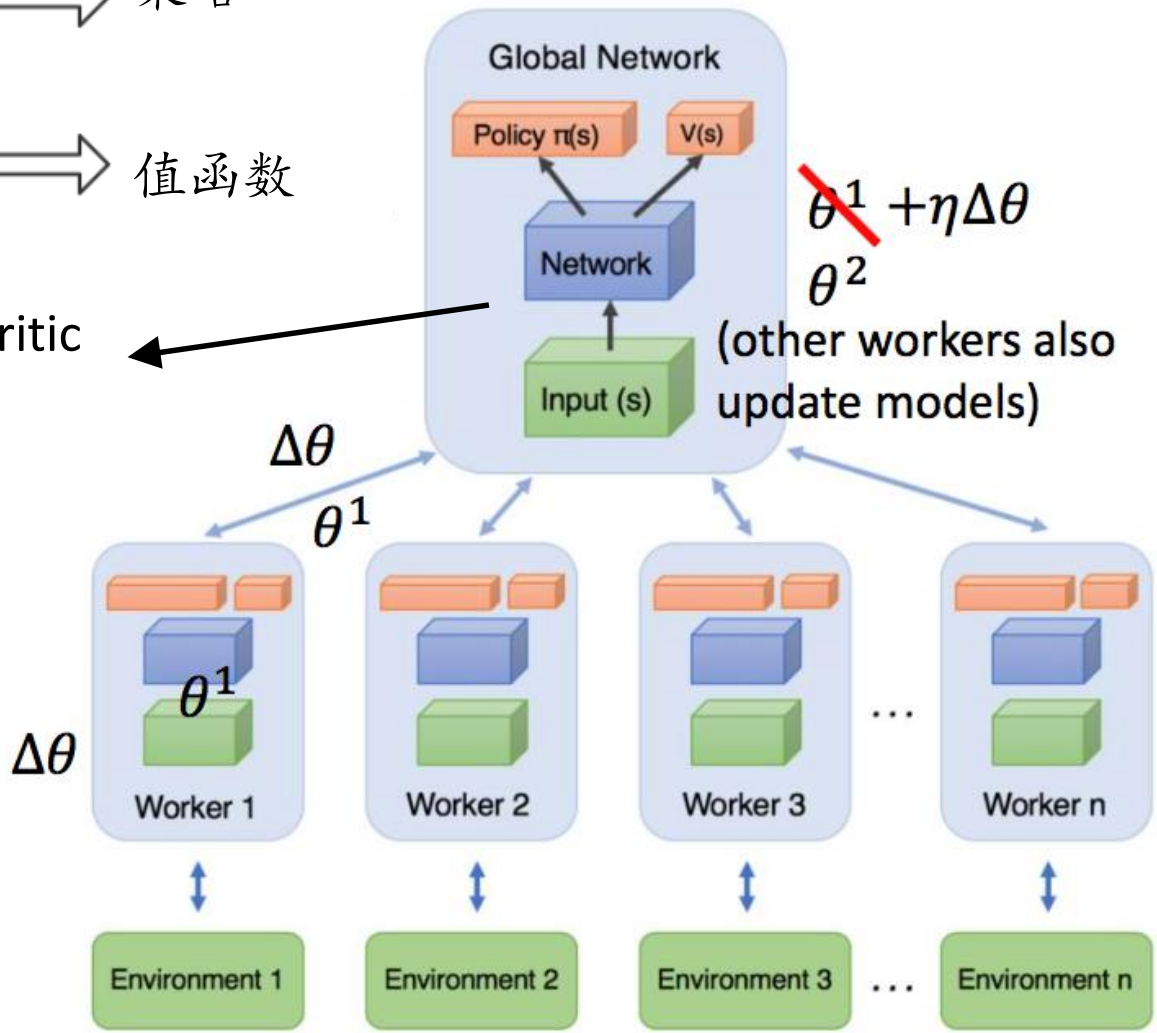


异步更新

Async Advantage Actor-Critic (A3C)



共享Actor-Critic网络权重



n步T-D学习

优势函数

Async Advantage Actor-Critic (A3C)

Async Advantage Actor-Critic (A3C)

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: $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i;\theta'_v))^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

多线程数据采集

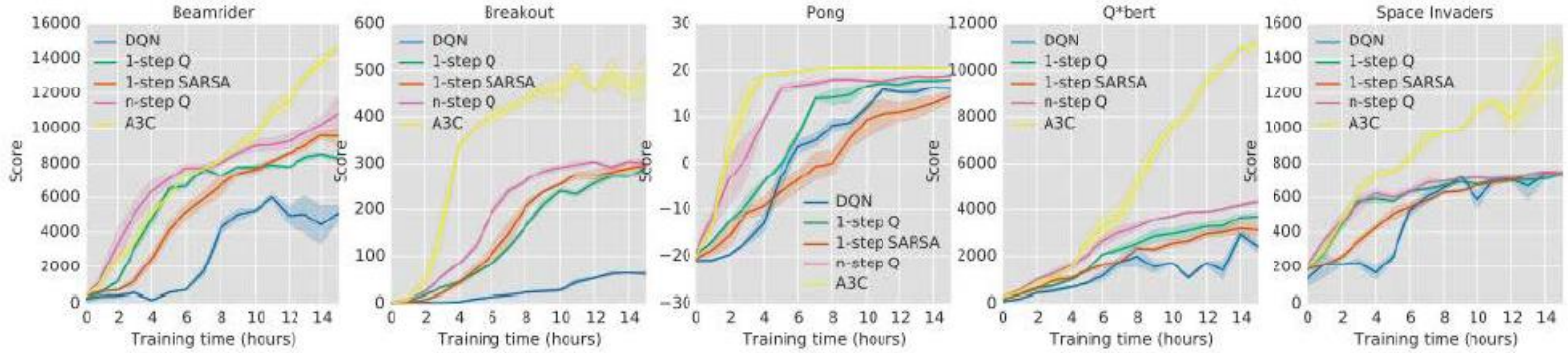
累计梯度

异步训练

Async Advantage Actor-Critic (A3C)

A3C的学习效率

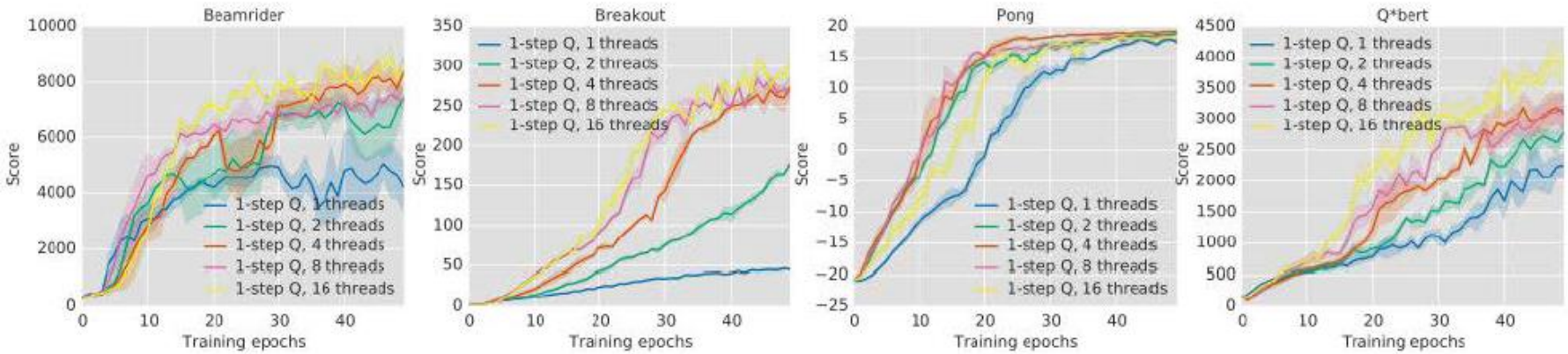
- 异步算法在 16 核 CPU 上运行 vs DQN 在 K40 GPU 上运行
- n-步算法要比 1-步算法快很多
- A3C 算法要比基于价值的算法好很多



Async Advantage Actor-Critic (A3C)

异步训练的数据利用率

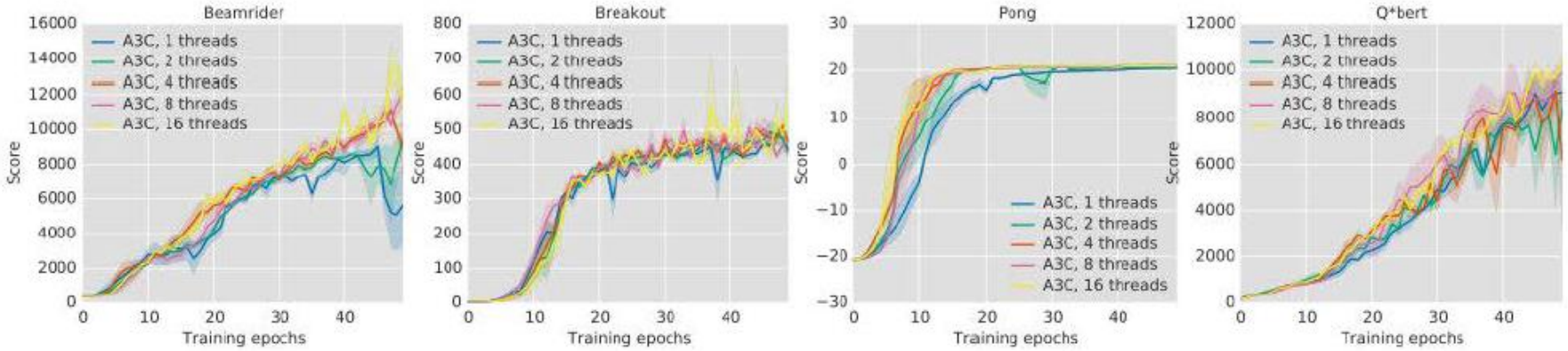
- 使用更多的线程和使用并行训练能够提升数据利用率
  - 1 个线程（蓝色）vs16 个线程（黄色）



Async Advantage Actor-Critic (A3C)

A3C的数据利用率

- 没有明显的数据利用率的提升. 并行训练获得的是次线性的速率提升
  - 1 个线程 (蓝色)vs 16 个线程 (黄色)



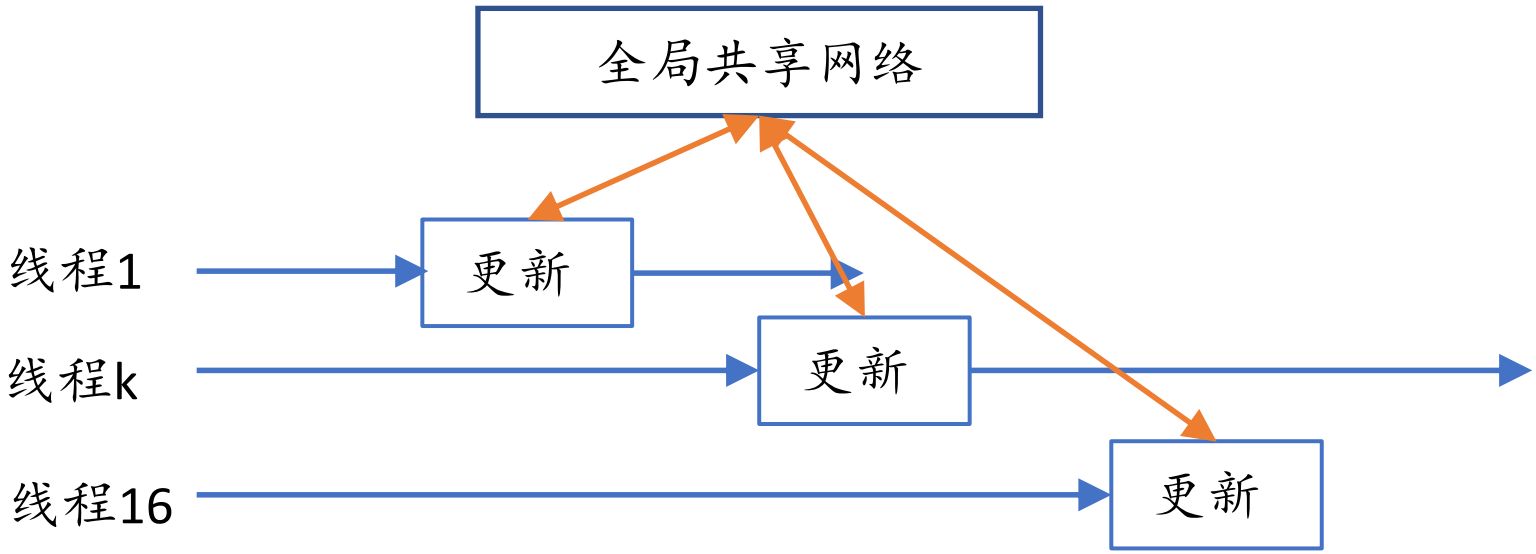
Async Advantage Actor-Critic (A3C)

A3C的Atari结果

Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorilla	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%

Advantage Actor Critic(A2C)

A3C异步更新是否是使得模型表现更好的关键?

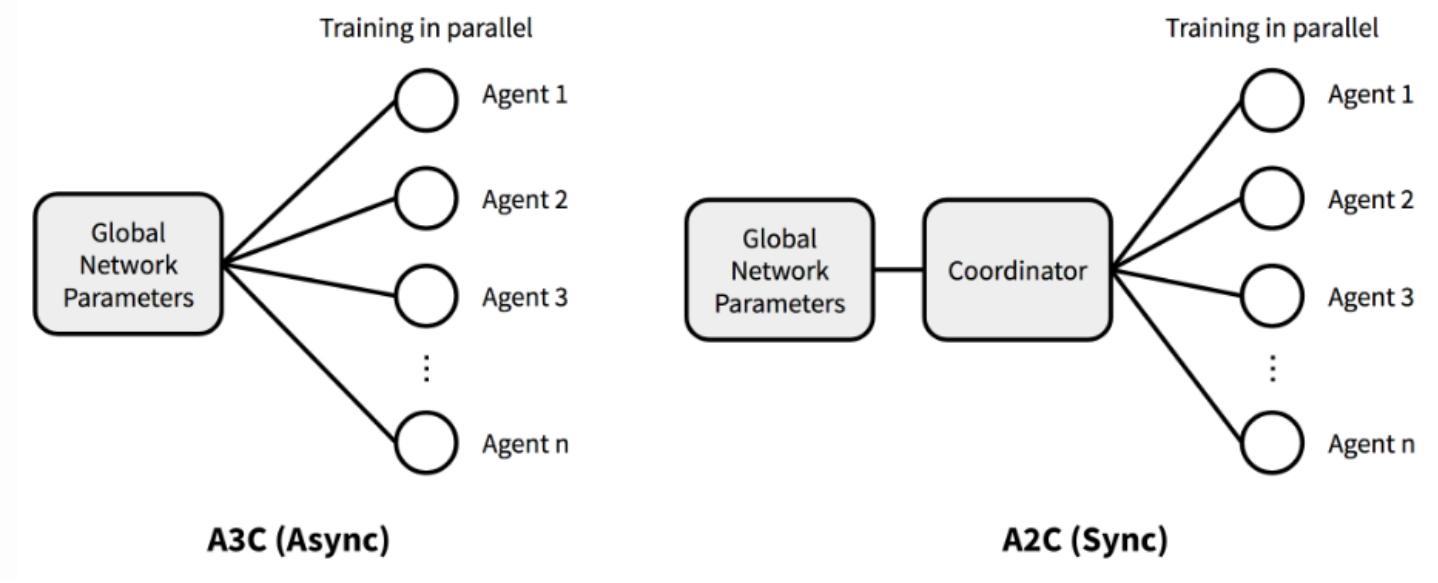


有时不同线程中的Actor将使用不同版本的策略，因此累积更新的方向将不是最优的。

https://openai.com/blog/baselines-acktr-a2c/

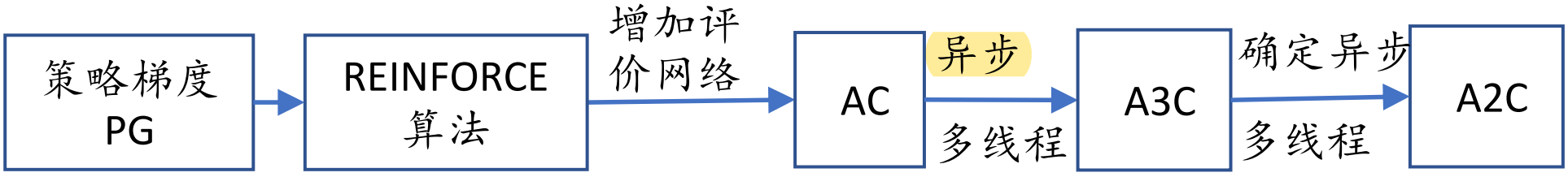
Advantage Actor Critic(A2C)

- A2C是一种确定的异步更新的方法
- 等待各个线程完成自己的任务，再计算各个线程的梯度平均值，然后对参数进行更新
- 每一次迭代中并行的actor将均执行同一策略，这种训练方式理论上会收敛更快
- A2C已被证明在能够实现与A3C相同或更好的性能得同时，更有效地利用GPU，并且能够适应更大的批量batch_size大小



Advantage Actor Critic(A2C)

A3C、A2C



Off-Policy PG

前面所提的策略梯度方法均是在策略的

$$J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[r(\tau)]$$

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

真实物理系统的在策略学习效率低

是否可以使用离策略的方法更新？

离策略方法的优势：

1. 离策略方法可重复利用历史样本，样本利用率高

2. 行为策略收集的样本与目标策略不同，带来更好的探索性

Off-Policy PG

行为策略$\bar{\pi}(\tau)$

目标策略$\pi_\theta(\tau)$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

$$J(\theta) = E_{\tau\sim\bar{\pi}(\tau)}\left[\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)}r(\tau)\right]$$

重要性采样

$$\begin{aligned} E_{x\sim p(x)}[f(x)] &= \int p(x)f(x)dx \\ &= \int \frac{q(x)}{q(x)}p(x)f(x)dx \\ &= \int q(x)\frac{p(x)}{q(x)}f(x)dx \\ &= E_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right] \end{aligned}$$

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

$$\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)} = \frac{\cancel{p(\mathbf{s}_1)}\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}}{\cancel{p(\mathbf{s}_1)}\prod_{t=1}^{T}\bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)\cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}} = \frac{\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^{T}\bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)}$$

Off-Policy PG

行为策略$\bar{\pi}(\tau)$

目标策略$\pi_\theta(\tau)$

$$\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)} = \frac{\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^{T}\bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)}$$

$$J(\theta) = E_{\tau\sim\bar{\pi}(\tau)}\left[\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)}r(\tau)\right]$$

$$\nabla_\theta J(\theta) = E_{\tau\sim\bar{\pi}(\tau)}\left[\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)}\nabla_\theta \log \pi_\theta(\tau) r(\tau)\right]$$

$$= E_{\tau\sim\bar{\pi}(\tau)}\left[\frac{\prod_{t=1}^{T}\pi_\theta(a_t|s_t)}{\prod_{t=1}^{T}\bar{\pi}(a_t|s_t)}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(\sum_{t=1}^{T} r(s_t,a_t)\right)\right]$$

$$\approx E_{\tau\sim\bar{\pi}(\tau)}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\prod_{t'=1}^{t}\frac{\pi_\theta(a_{t'}|s_{t'})}{\bar{\pi}(a_{t'}|s_{t'})}\right)\left(\sum_{t'=t}^{T} r(s_{t'},a_{t'})\left(\prod_{t''=t}^{T}\frac{\pi_\theta(a_{t''}|s_{t''})}{\bar{\pi}(a_{t''}|s_{t''})}\right)\right)\right]$$

未来动作并不影响当前权重

随着T的增加计算量指数增长

忽略该项，经典策略迭代算法

Off-Policy PG

结合On-Policy PG与重要性采样的改进版本

On-Policy PG更新公式：

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

$$(s_{i,t},a_{i,t}) \sim \pi_\theta(s_t,a_t)$$

Off-Policy PG更新公式：

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\prod_{t'=1}^{t}\frac{\pi_\theta(a_{t'}|s_{t'})}{\bar{\pi}(a_{t'}|s_{t'})}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\sum_{t'=t}^{T} r(s_{i,t'},a_{i,t'})$$

$$\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\frac{\pi_\theta(s_{i,t})}{\bar{\pi}(s_{i,t})}\frac{\pi_\theta(a_{i,t}|s_{i,t})}{\bar{\pi}(a_{i,t}|s_{i,t})}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\sum_{t'=t}^{T} r(s_{i,t'},a_{i,t'})$$

$$(s_{i,t},a_{i,t}) \sim \bar{\pi}(s_t,a_t)$$

当利用$\pi_{\theta_{\text{old}}}(\tau)$作为行为策略$\bar{\pi}(\tau)$时，
可忽略相似策略的状态概率分布不匹配性

Off-Policy PG

行为策略$\bar{\pi}(\tau)=\pi_{\theta_{\text{old}}}(\tau)$

目标策略$\pi_\theta(\tau)$

$$J(\theta)=\sum_{t=1}^{T} E_{(s_t,a_t)\sim\pi_\theta(s_t,a_t)}[r(s_t,a_t)]$$

$$=\sum_{t=1}^{T} E_{s_t\sim\pi_\theta(s_t)}[E_{a_t\sim\pi_\theta(a_t|s_t)}[r(s_t,a_t)]]$$

$$J(\theta) = E_{\tau\sim\bar{\pi}(\tau)}\left[\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)} r(\tau)\right]$$

$$J(\theta)=\sum_{t=1}^{T} E_{s_t\sim\pi_{\theta_{\text{old}}}(s_t)}[\frac{\pi_\theta(s_t)}{\pi_{\theta_{\text{old}}}(s_t)} E_{a_t\sim\pi_{\theta_{\text{old}}}(a_t|s_t)}[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} r(s_t,a_t)]]$$

https://medium.com/@jonathan_hui/rl-policy-gradients-explained-9b13b688b146

Off-Policy PG

优化目标 $J(\theta)=E_{\tau\sim\pi_{\theta_{\text{old}}}(\tau)}\left[\frac{\pi_\theta(\tau)}{\pi_{\theta_{\text{old}}}(\tau)}r(\tau)\right]$

或者 $J(\theta)=E_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}r(s_t,a_t)\right]$

Off-policy梯度优化

$$\nabla J(\theta)=E_{\tau\sim\pi_{\theta_{\text{old}}}(\tau)}\left[\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(a_t|s_t)\left(\prod_{t'=1}^{t}\frac{\pi_\theta(a_{t'}|s_{t'})}{\bar{\pi}(a_{t'}|s_{t'})}\right)\left(\sum_{t'=t}^{T}r(s_{t'},a_{t'})\right)\right]$$

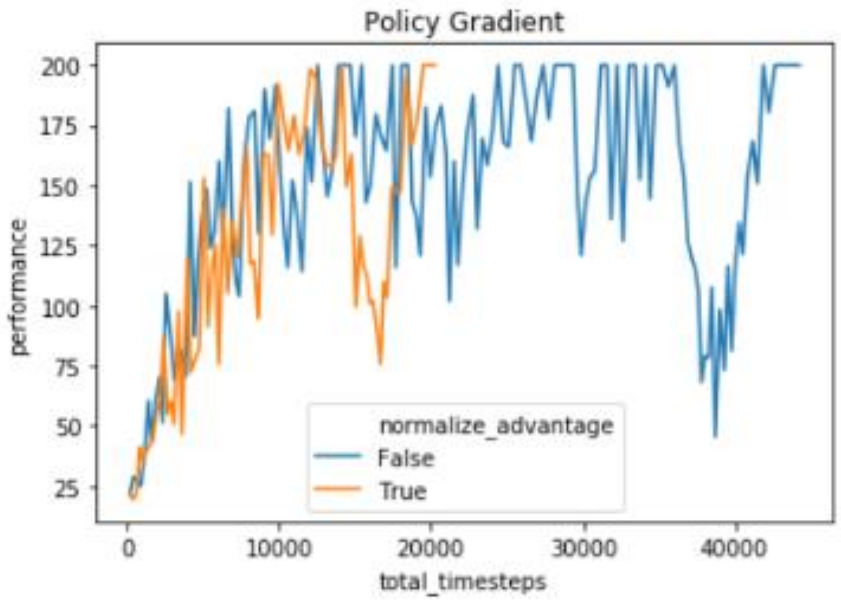
$$\theta=\theta+\alpha\nabla_\theta J(\theta)$$

Off-Policy PG

Trust region policy optimization (TRPO)

步长更新 $\theta = \theta + \alpha \nabla_\theta J(\theta)$

- 策略梯度算法的更新步长很重要
  步长太小，导致更新效率低下
  步长太大，导致参数变动太大，不易收敛,同样存在复合误差，导致可能无法从bad policy恢复



Trust region policy optimization (TRPO)

策略梯度目标： 寻找最大化$J(\theta) = J^{\pi\theta}$的策略参数$\theta$

如何选择一个合适的步长，使得每次更新得到的新策略所实现的回报值单调不减？

信赖域(Trust Region) 方法是更高级的步长更新方法，指在该区域内更新，策略所实现的回报值单调不减

Gradient ascend

Trust region

Trust region policy optimization (TRPO)

自然策略梯度 – 2001 Nips

$$J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$D_{\mathrm{KL}}(\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)) \leq \epsilon$$

梯度更新

$$\theta' = \theta + \alpha \mathbf{F}^{-1} \nabla_\theta J(\theta)$$

二阶泰勒级数展开

$$D_{\mathrm{KL}}(\pi_{\theta'} \| \pi_\theta) \approx (\theta' - \theta)^T \mathbf{F} (\theta' - \theta)$$

$$\mathbf{F} = E_{\pi_\theta}[\log \pi_\theta(\mathbf{a}|\mathbf{s}) \log \pi_\theta(\mathbf{a}|\mathbf{s})^T]$$

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

Fisher信息矩阵：可以基于样本估计得到

$$\alpha = \sqrt{\frac{2\epsilon}{\nabla_\theta J(\theta)^T \mathbf{F}^{-1} \nabla_\theta J(\theta)}}$$

Trust region policy optimization (TRPO)

自然策略梯度需要计算F的逆，计算昂贵

TRPO估计 $x = \mathbf{F}^{-1}\nabla_\theta J(\theta)$ 通过求解线性方程 $\mathbf{F}x = g$， $g = \nabla_\theta J(\theta)$

$$求解 Ax = b$$

$$等价于求解 \quad x = \arg\max_x f(x) = \frac{1}{2}x^T Ax - b^T x$$

$$由于 \quad f'(x) = Ax - b = 0$$

那么我们可以通过优化二次方程式

$$\min_x \frac{1}{2}x^T Fx - g^T x$$

利用共轭梯度方法来求解，避免计算F矩阵的逆

Trust region policy optimization (TRPO)

对于深度神经网络，参数θ维度很高，导致求F矩阵的逆困难

TRPO基于共轭梯度提出更高效的求解方法

1: **for** iteration=1, 2, . . . **do**
2: Run policy for T timesteps or N trajectories
3: Estimate advantage function at all timesteps
4: Compute policy gradient g
5: Use CG (with Hessian-vector products) to compute $F^{-1}g$ where F is the Fisher information matrix
6: Do line search on surrogate loss and KL constraint
7: **end for**

- 求解带约束的优化问题，利用自然梯度
- 实际求解是利用了共轭梯度 + 线性搜索的方法，避免求自然梯度

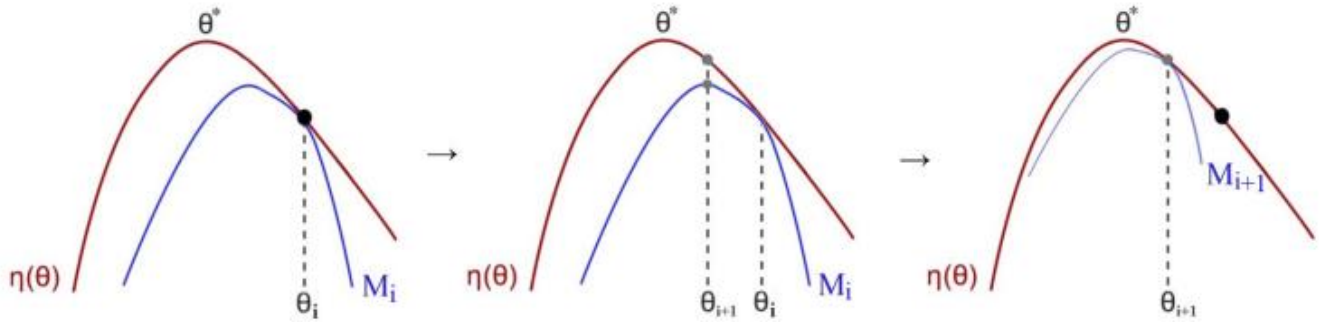
Trust region policy optimization (TRPO)

TRPO：

$$\theta' \leftarrow \arg\max_{\theta'} \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

使得 $D_{\mathrm{KL}}(\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)) \leq \epsilon$

- 重要性采样
- 优势函数
- 自然策略梯度
- 共轭梯度法



理论证明：

TRPO 论文提供了2页的数学证明

TRPO每次策略更新后，新策略可以确保目标函数单调递增

Trust region policy optimization (TRPO)

优势

Trust region policy optimization (TRPO)

不足

- TRPO的稳定性问题

1. 每次计算F矩阵是非常昂贵的

$$\mathbf{F} = E_{\pi_\theta}[\log \pi_\theta(\mathbf{a}|\mathbf{s}) \log \pi_\theta(\mathbf{a}|\mathbf{s})^T]$$

2. 二阶共轭梯度使得实施起来更复杂

TRPO的性能在某些任务上不如DQN

	B. Rider	*Breakout*	*Enduro*	*Pong*	*Q*bert*	*Seaquest*	*S. Invaders*
Random	354	1.2	0	−20.4	157	110	179
Human (Mnih et al., 2013)	7456	31.0	368	−3.0	18900	28010	3690
Deep Q Learning (Mnih et al., 2013)	4092	168.0	470	20.0	1952	1705	581
UCC-I (Guo et al., 2014)	5702	380	741	21	20025	2995	692
TRPO - single path	1425.2	10.8	534.6	20.9	1973.5	1908.6	568.4
TRPO - vine	859.5	34.2	430.8	20.9	7732.5	788.4	450.2

Proximal Policy Optimization (PPO)

TRPO:

$$\text{优化目标} \quad \max imize_{\theta}\ E_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}A(s_t,a_t)\right]$$

$$\text{约束} \quad D_{kl}\left[\pi_{\theta_{old}}(a_t|s_t),\pi_\theta(a_t|s_t)\right]\le\epsilon$$

不受约束形式

$$\max imize_{\theta}\ E_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}A(s_t,a_t)\right]\text{-}\beta E_t\left[D_{kl}\left[\pi_{\theta_{old}}(a_t|s_t),\pi_\theta(a_t|s_t)\right]\right]$$

PPO：自适应调整KL 惩罚因子，确保策略在trust region内更新

Proximal Policy Optimization (PPO)

Algorithm 4 PPO with Adaptive KL Penalty

Input: initial policy parameters θ_0, initial KL penalty β_0, target KL-divergence δ
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta||\theta_k)$$

  by taking K steps of minibatch SGD (via Adam)
  if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \geq 1.5\delta$ **then**
    $\beta_{k+1} = 2\beta_k$
  else if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \leq \delta/1.5$ **then**
    $\beta_{k+1} = \beta_k/2$
  end if
end for

与TRPO性能相当，但是使用SGD求解速度更快

Proximal Policy Optimization (PPO)

PPO with clipping

$$L_t(\theta) = \min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)$$

PPO的目标函数是去取原始值与截断版本的之间的较小值

如果新老策略的重要性比值在$[1-\varepsilon, 1+\varepsilon]$范围之外，目标函数将会被截断

$\text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)$ 将策略比值约束在$[1-\varepsilon, 1+\varepsilon]$范围内

ϵ常常取值为0.2

Openai blog(https://blog.openai.com/openai-baselines-ppo/)

Proximal Policy Optimization (PPO)

通过截断操作使得策略变化更加平稳

$$L^{CLIP}(\theta) = \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t, \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)$$

当优势函数为正时，鼓励该动作a

$$L(\theta;\theta_{old}) = \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, (1+\epsilon)\right)\hat{A}_t$$

当优势函数为负时，不鼓励该动作a

$$L(\theta;\theta_{old}) = \max\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, (1-\epsilon)\right)\hat{A}_t$$

Proximal Policy Optimization (PPO)

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0, clipping threshold ϵ

for $k = 0, 1, 2, \ldots$ **do**

Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$

Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

Compute policy update

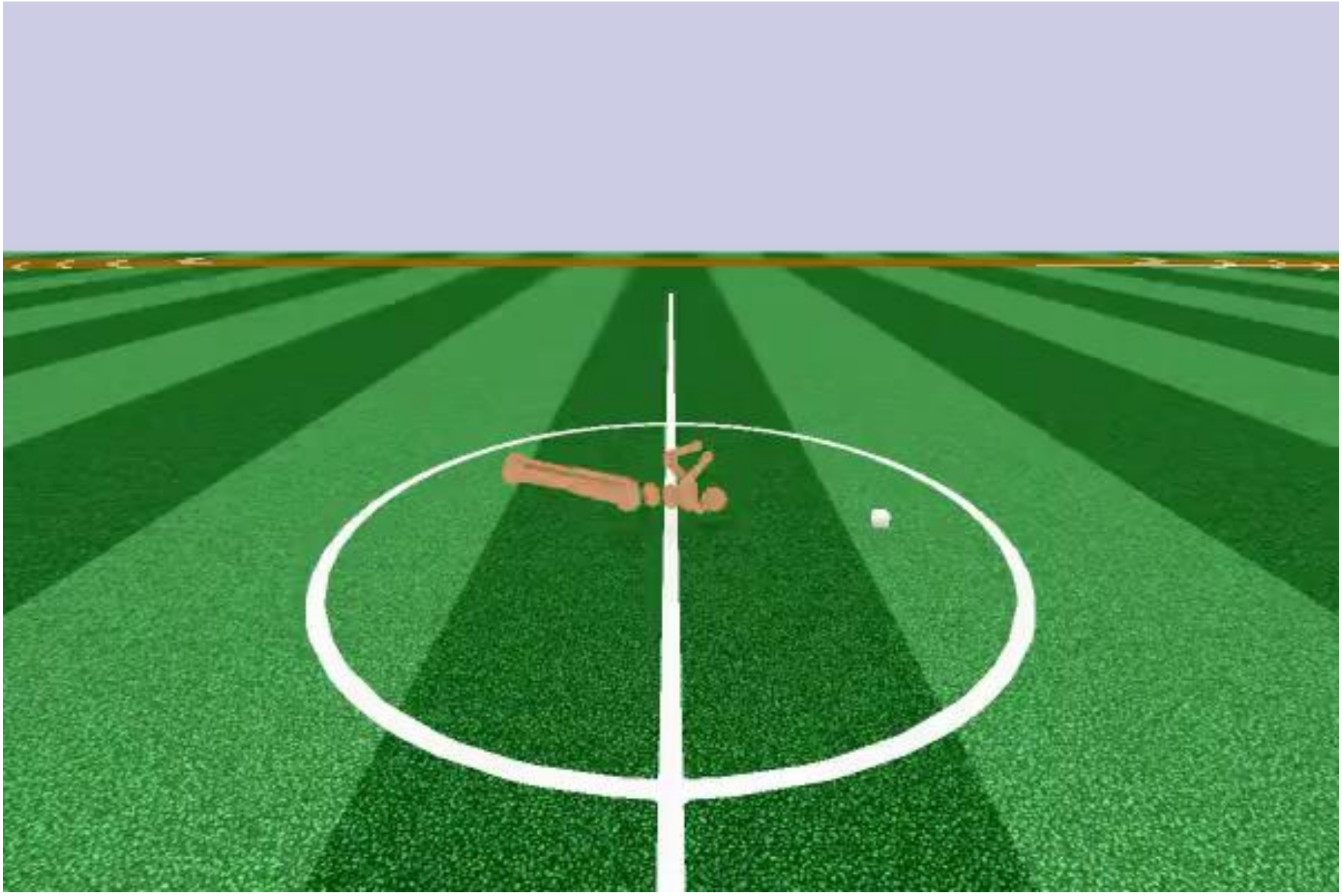
$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

by taking K steps of minibatch SGD (via Adam), where

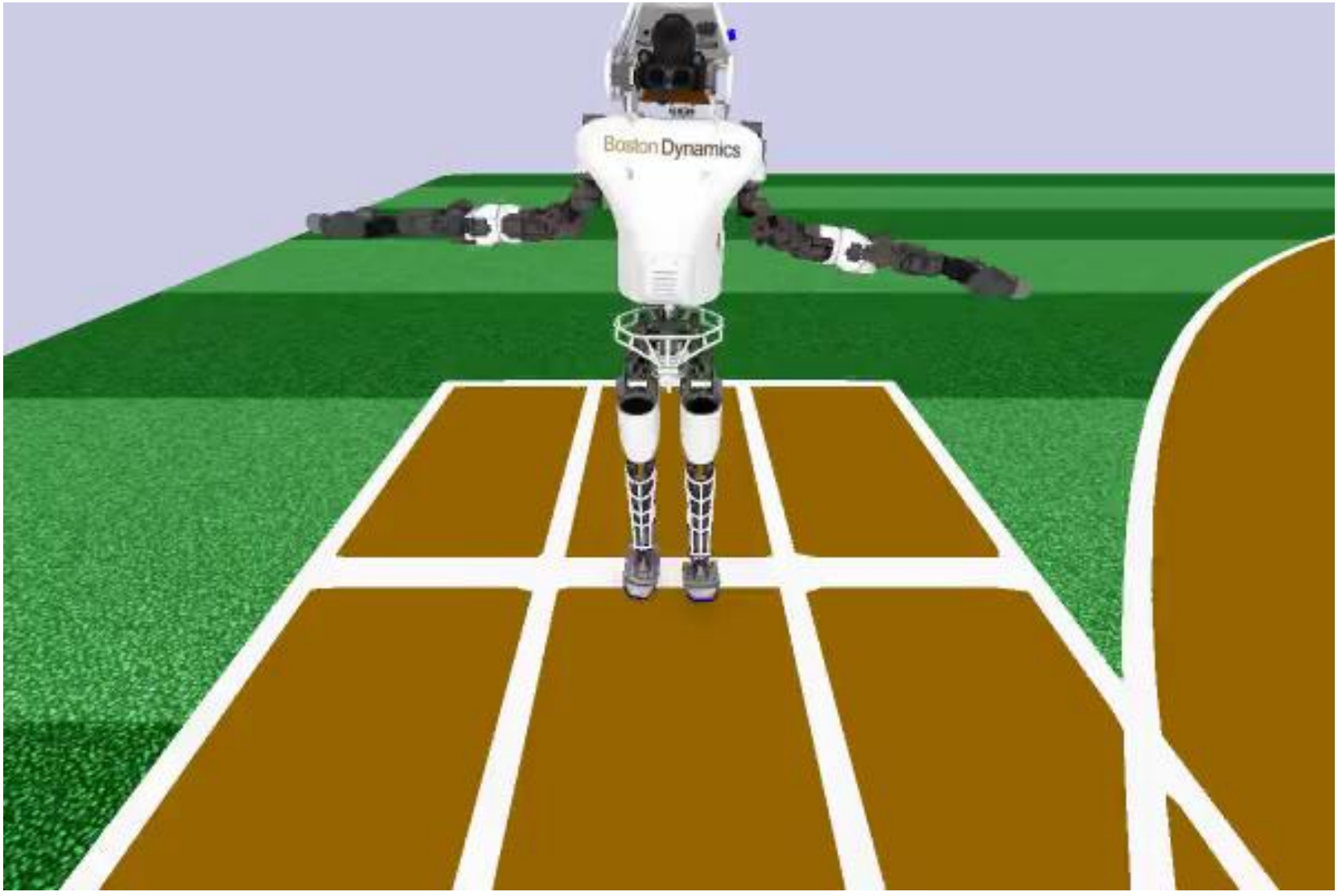
$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k} \left[\sum_{t=0}^{T} \left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}) \right] \right]$$

end for

Proximal Policy Optimization (PPO)



Proximal Policy Optimization (PPO)



Proximal Policy Optimization (PPO)

其他信赖域算法

ACKTR: Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation

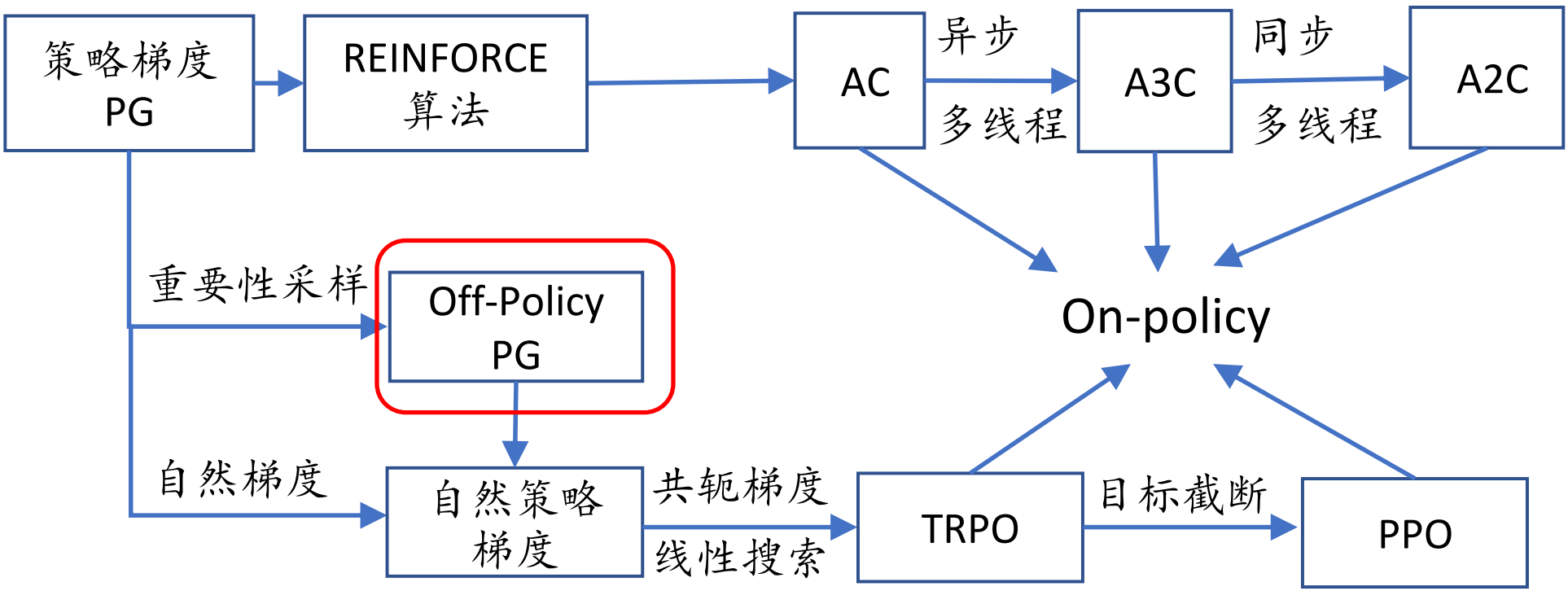
ACER: Sample Efficient Actor-Critic with Experience Replay

GAE: High-Dimensional Continuous Control Using Generalized Advantage Estimation

...

Proximal Policy Optimization (PPO)

TRPO、PPO



Deterministic policy gradient (DPG)

随机性策略函数：$\pi_\theta(a|s)$

- 给定当前状态下在动作空间上的概率分布
- 难以用到高维和连续动作空间

确定性策略梯度： $a = \pi_\theta(s)$

- 直接给定当前状态下在确定性的动作
- 找到使 Q 函数最大的动作

$$\nabla_\theta Q(s, \pi_\theta(s)) = \nabla_a Q(s, a)\big|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)$$

$$a^* = arg\max_a Q^*(s, a) \qquad a^* = arg\max_a Q^*(s, a) \approx Q_\theta\big(s, \pi_\theta(s)\big)$$

Deterministic policy gradient (DPG)

1: 定义 Q 函数逼近器 $Q_{\mathbf{w}}(s,a)$，策略逼近器 $\pi_\theta(s,a)$，初始化 $\mathbf{w}$，θ，$s_t = s_0$，$t = 0$
2: **repeat**
3: 采样动作 $a_t = \pi_\theta(s_t) + \mathcal{N}_t$ 并执行，观测 r_{t+1}, s_{t+1}
4: 计算 TD 误差：$\delta = r_t + \gamma Q_{\mathbf{w}}(s_{t+1}, \pi_\theta(s_{t+1})) - Q_{\mathbf{w}}(s_t, a_t)$
5: 更新 Critic：$\mathbf{w} \leftarrow \mathbf{w} + \beta\delta\nabla_{\mathbf{w}} Q_{\mathbf{w}}(s_t, a_t)$
6: 更新 Actor：$\theta \leftarrow \theta + \alpha\nabla_a Q_{\mathbf{w}}(s_t, a)\big|_{a=\pi_\theta(s_t)} \nabla_\theta \pi_\theta(s_t)$
7: $t \leftarrow t + 1$
8: **until**

- $\mathcal{N}_t$ 是 exploration noise (e.g. 高斯噪声)，避免参数陷入局部解

Deep Deterministic policy gradient (DDPG)

- 结合了DQN和DPG，DQN用于高维输入离散动作空间，DPG用于低维输入连续动作空间
- 使用了DQN 的两种技术：Experience Replay 和Target Network
  - 对于critic和actor均有Target Network，采用软更新方式

$$\theta' \leftarrow \tau\theta+(1-\tau)\theta' \qquad w' \leftarrow \tau w+(1-\tau)w' \qquad \tau \ll 1$$

θ'为目标actor网络参数　　w'为目标critic网络参数

- 为了充分探索，利用添加噪声产生探索性动作

$$\pi'_\theta(s) = \pi_\theta(s) + \mathcal{N} \qquad \mathcal{N}\text{为噪声}$$

Continuous Control with Deep Reinforcement Learning (ICRL2016)

Deep Deterministic policy gradient (DDPG)

Actor当前网络：负责策略网络参数θ的迭代更新，根据当前状态s选择当前执行的确定性动作a，用于和环境交互生成s', r

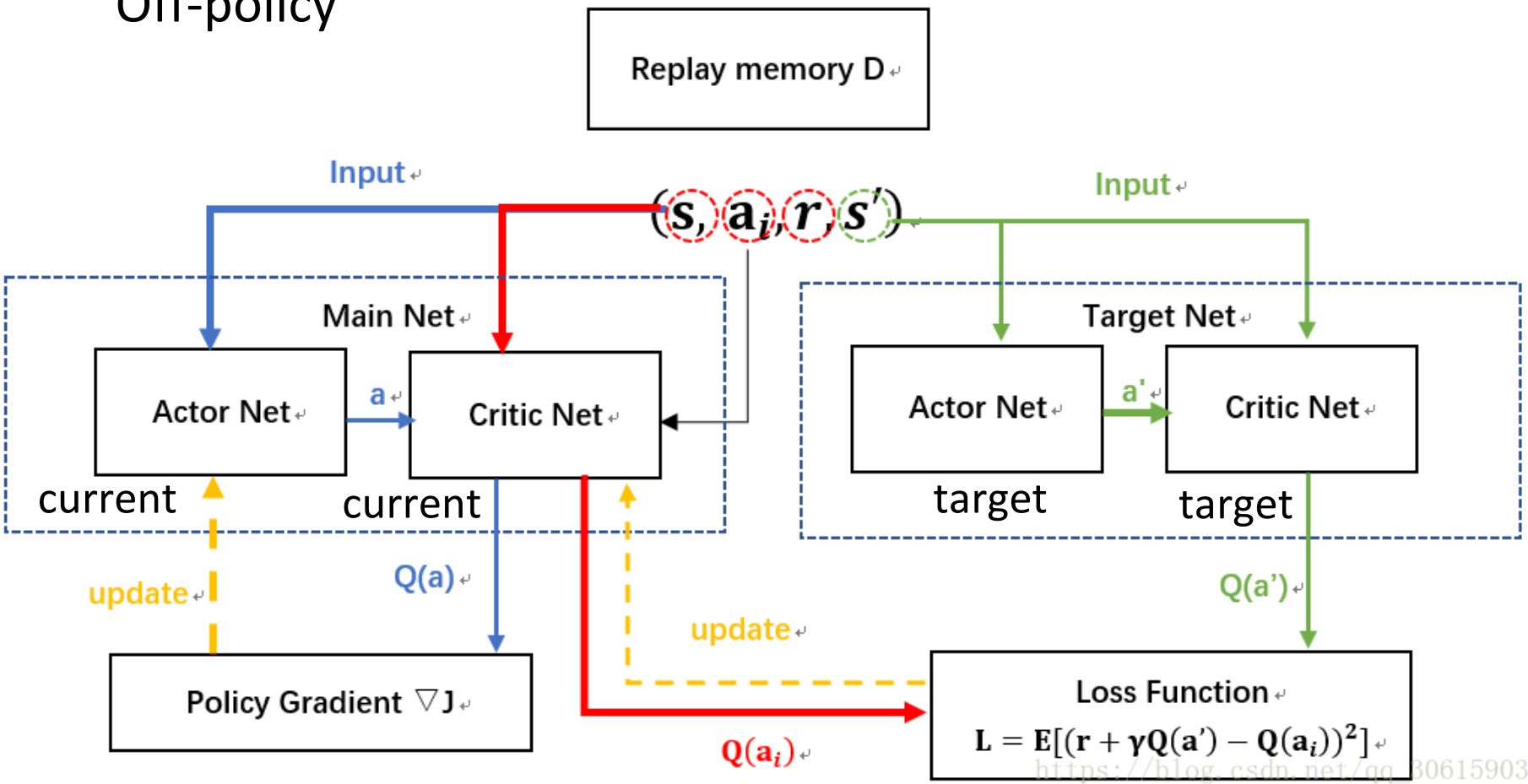
Actor目标网络：负责根据经验回放池中采样的下一状态s'选择下一最优动作a'，估计target Q值，网络参数$\theta' \leftarrow \tau\theta+(1-\tau)\theta'$

Critic当前网络：负责价值网络参数w的迭代更新,计算当前Q值$Q_w(s,a)$

Critic目标网络：负责计算target Q值$Q(s',a')$, 网络参数$w' \leftarrow \tau w+(1-\tau)w'$

Deep Deterministic policy gradient (DDPG)

Off-policy



Deep Deterministic policy gradient (DDPG)

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

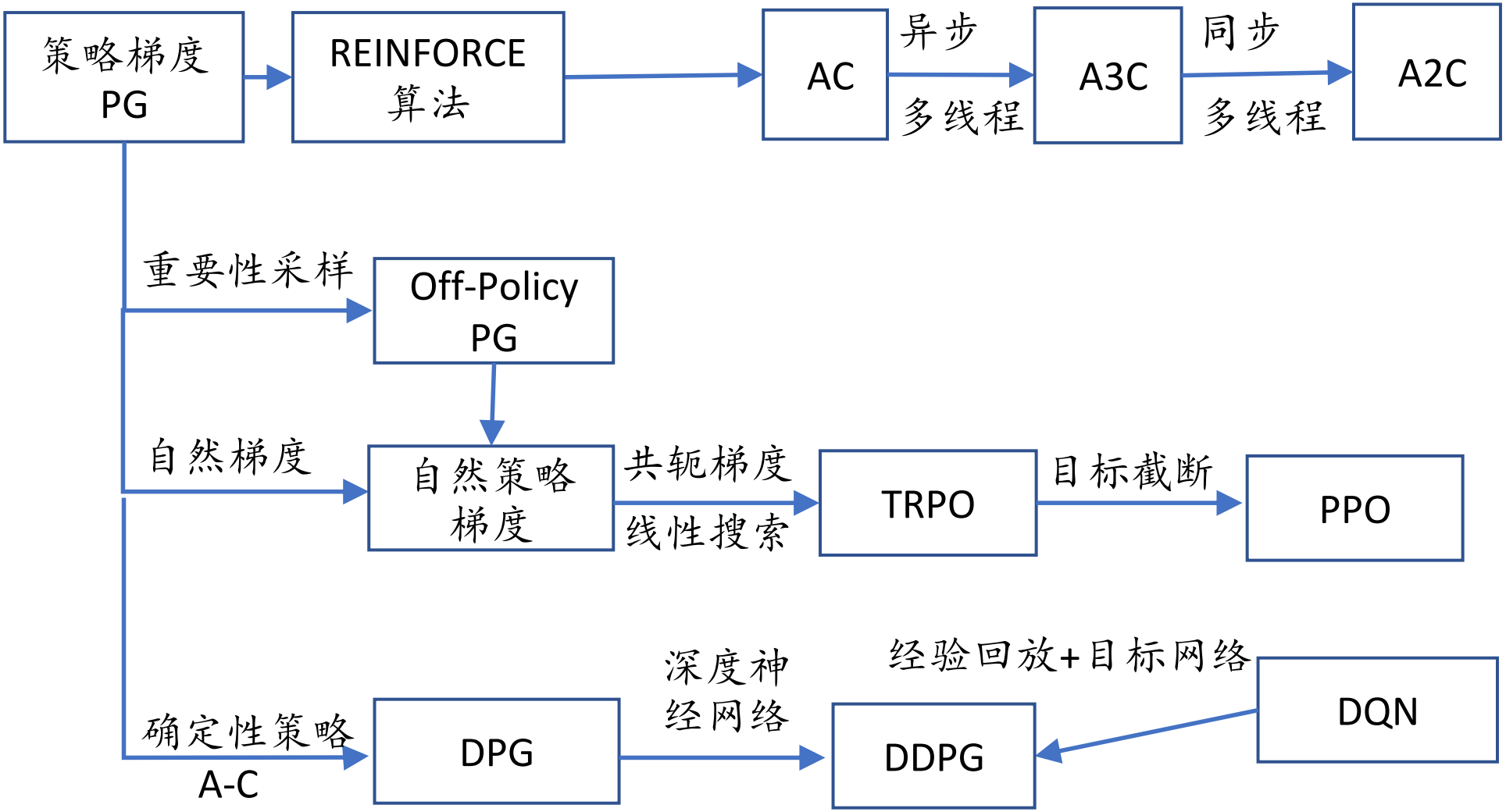
    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Deep Deterministic policy gradient (DDPG)

DPG、DDPG



Soft Actor-Critic (SAC)

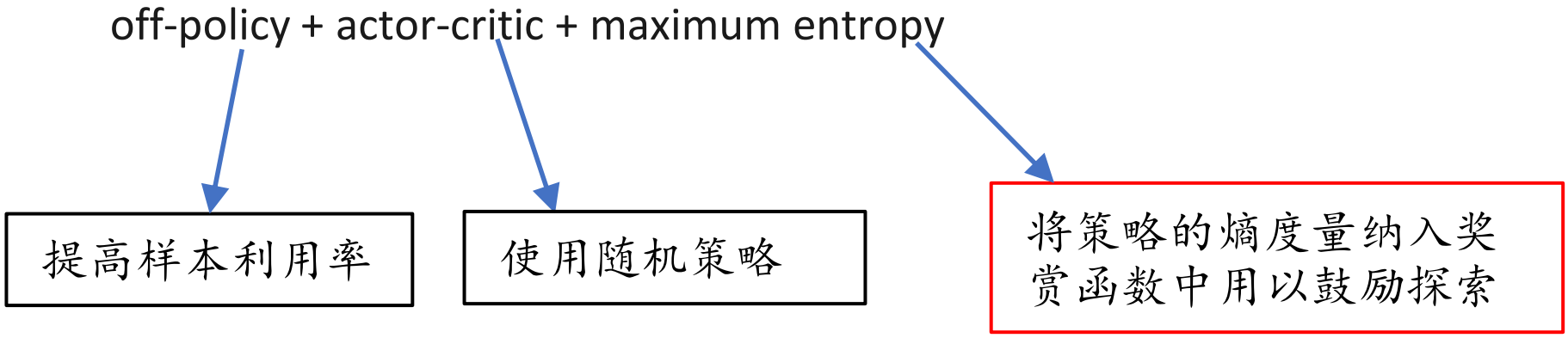
On-policy：A3C/TRPO/PPO等

面临样本利用率低的问题，用完一批样本即丢弃

Off-policy：off-policy PG/DDPG等

样本利用率提高，对超参数敏感，训练不稳定，收敛性较差

Soft Actor-Critic (SAC)



是遵循最大熵强化学习框架的off-policy actor-critic模型

三大关键组成部分

- 包含策略网络和值函数网络的actor-critic架构，利用随机策略；
- 异策略使其能够复用历史收集的数据从而实现高采样有效性；
- 熵最大化以使得训练稳定并鼓励探索。

Soft Actor-Critic (SAC)

策略的训练目标是同时最大化期望累积回报以及策略的熵度量

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\theta}} [r(s_t,a_t) + \alpha \mathcal{H}(\pi_\theta(.|s_t))]$$

$\mathcal{H}(.)$表示熵度量/熵正则项 $H(\pi(\cdot|s')) = -\mathbb{E}_a \log \pi(a'|s')$

α称为温度参数用以控制熵正则项的重要度，α越大策略的随机性越强

熵最大化使得策略在训练过程中

- 对未知空间进行更多的探索（使policy输出的action分布更均匀）
- 捕获近似最优策略的多种模式
- 提高学习速率和稳定性

Soft Actor-Critic (SAC)

soft 状态值函数定义：

$$V(s_t) = \mathbb{E}_{a_t \sim \pi}[Q(s_t, a_t) - \alpha \log \pi(a_t|s_t)]$$

soft 状态-动作值函数定义：

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \rho_\pi(s)}[V(s_{t+1})]$$

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{(s_{t+1}, a_{t+1}) \sim \rho_\pi}[Q(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1}|s_{t+1})]$$

在奖赏中加入策略熵项，对Bellman方程推导均产生了影响，变成soft-Bellman的形式

Soft Actor-Critic (SAC)

Soft-Critic

- 2个状态值函数，参数分别为 ψ 和 $\bar{\psi}$
- 1个状态-动作Q函数，参数为 ω

Target值函数参数 $\bar{\psi}$ 不进行训练，同样采用软更新方式：

$$\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$$

Current值函数参数 ψ 的训练方式为，最小化均方误差：

策略熵

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}}\left[\frac{1}{2}\left(V_\psi(s_t) - \mathbb{E}[Q_w(s_t, a_t) - \log \pi_\theta(a_t|s_t)]\right)^2\right]$$

计算梯度：

$$\nabla_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t)\left(V_\psi(s_t) - Q_w(s_t, a_t) + \log \pi_\theta(a_t|s_t)\right)$$

Soft Actor-Critic (SAC)

Soft-Critic

- 2个状态值函数，参数分别为 ψ 和 $\bar{\psi}$
- 1个状态-动作Q函数，参数为 ω

soft 状态-动作值函数训练：　通过最小化软贝尔曼残差来训练

$$J_Q(w) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}[\frac{1}{2}\left(Q_w(s_t,a_t) - (r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim\rho_\pi(s)}[V_{\bar{\psi}}(s_{t+1})])\right)^2]$$

计算梯度：

$$\nabla_w J_Q(w) = \nabla_w Q_w(s_t,a_t)\left(Q_w(s_t,a_t) - r(s_t,a_t) - \gamma V_{\bar{\psi}}(s_{t+1})\right)$$

为目标值网络参数，与DQN类似，为了稳定训练的目的

Soft Actor-Critic (SAC)

Soft Policy Iteration

Soft policy evaluation：

固定策略π，使用Bellman方程更新Q值直到收敛

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{(s_{t+1}, a_{t+1}) \sim \rho_\pi} [Q(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1} | s_{t+1})]$$

Soft policy improvement:

更新策略π

$$\begin{aligned} \pi_{\text{new}} &= \arg\min_{\pi' \in \Pi} D_{\text{KL}} \left(\pi'(. | s_t) \| \frac{\exp(Q^{\pi_{\text{old}}}(s_t, .))}{Z^{\pi_{\text{old}}}(s_t)} \right) \\ &= \arg\min_{\pi' \in \Pi} D_{\text{KL}} \left(\pi'(. | s_t) \| \exp(Q^{\pi_{\text{old}}}(s_t, .) - \log Z^{\pi_{\text{old}}}(s_t)) \right) \end{aligned}$$

可以确保 $Q^{\pi_{\text{new}}}(s_t, a_t) \geq Q^{\pi_{\text{old}}}(s_t, a_t)$

证明策略通过最小化KL散度来训练，能够保证policy improvement

Soft Actor-Critic (SAC)

Soft Actor：1个actor网络π_θ，参数为θ　　最小化KL散度来训练

$$\pi_{\text{new}} = \arg\min_{\pi' \in \Pi} D_{\text{KL}}\left(\pi'(.|s_t) \| \frac{\exp(Q^{\pi_{\text{old}}}(s_t,.))}{Z^{\pi_{\text{old}}}(s_t)}\right)$$

$$= \arg\min_{\pi' \in \Pi} D_{\text{KL}}\left(\pi'(.|s_t) \| \exp(Q^{\pi_{\text{old}}}(s_t,.) - \log Z^{\pi_{\text{old}}}(s_t))\right)$$

Π 是潜在策略的集合，可以是高斯混合分布簇

$Z^{\pi_{\text{old}}}(s_t)$ 是用于正则化分布的配分函数

更新目标为：

重参数技巧：$a_t = f_\theta(\epsilon_t; s_t)$

$$J_\pi(\theta) = \nabla_\theta D_{\text{KL}}\left(\pi_\theta(.|s_t) \| \exp(Q_w(s_t,.) - \log Z_w(s_t))\right)$$

$$= \mathbb{E}_{a_t \sim \pi}\left[-\log\left(\frac{\exp(Q_w(s_t,a_t) - \log Z_w(s_t))}{\pi_\theta(a_t|s_t)}\right)\right]$$

$$= \mathbb{E}_{a_t \sim \pi}[\log \pi_\theta(a_t|s_t) - Q_w(s_t,a_t) + \log Z_w(s_t)]$$

Soft Actor-Critic (SAC)

Soft Actor：1个actor网络π_θ，参数为θ

$$J_\pi(\theta) = \mathbb{E}_{a_t \sim \pi}[\log \pi_\theta(a_t|s_t) - Q_w(s_t, a_t) + \log Z_w(s_t)]$$

重参数技巧：$a_t = f_\theta(\epsilon_t; s_t)$

用输入随机噪声的随机性代替采样的随机性

计算梯度：

$$\nabla_\theta J_\pi(\theta) = \nabla_\theta \log \pi_\theta(a_t|s_t) + \left(\nabla_{a_t} \log \pi_\theta(a_t|s_t) - \nabla_{a_t} Q_\omega(s_t, a_t)\right) \nabla_\theta f_\theta(\epsilon_t; s_t)$$

Soft Actor-Critic (SAC)

Algorithm 1 Soft Actor-Critic

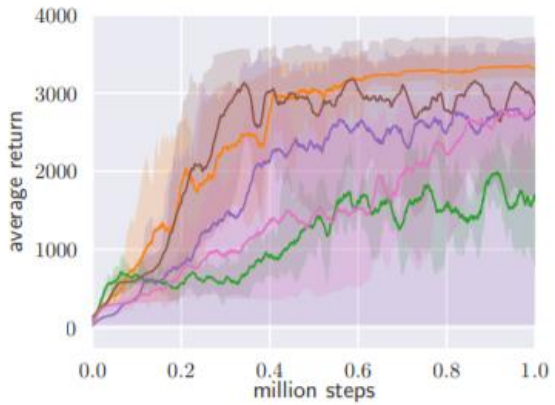
Inputs: The learning rates, λ_π, λ_Q, and λ_V for functions π_θ, Q_w, and V_ψ respectively; the weighting factor τ for exponential moving average.

Initialize parameters θ, w, ψ, and $\bar{\psi}$.
for each iteration **do**
  (In practice, a combination of a single environment step and multiple gradient steps is found to work best.)
  for each environment setup **do**
    $a_t \sim \pi_\theta(a_t|s_t)$
    $s_{t+1} \sim \rho_\pi(s_{t+1}|s_t, a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1}\}$
  for each gradient update step **do**
    $\psi \leftarrow \psi - \lambda_V \nabla_\psi J_V(\psi)$.
    $w \leftarrow w - \lambda_Q \nabla_w J_Q(w)$.
    $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta J_\pi(\theta)$.
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$).

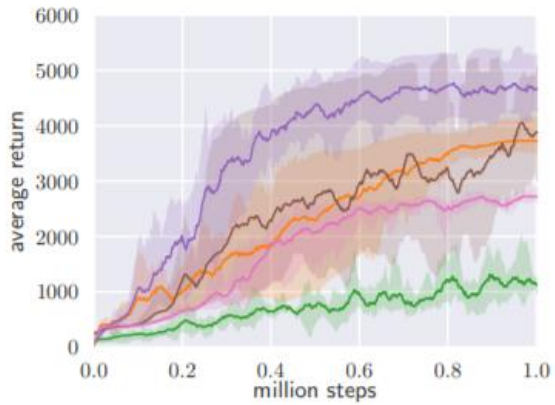
https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#sac

Soft Actor-Critic (SAC)

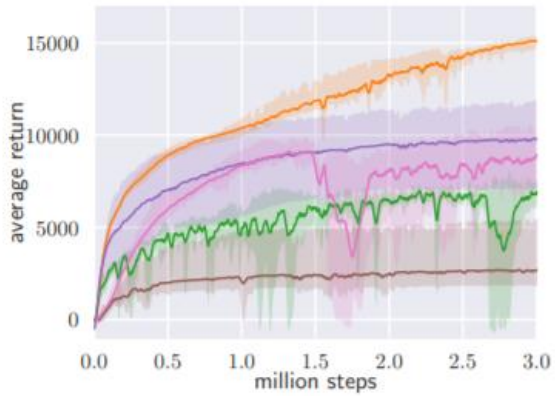
实验结果



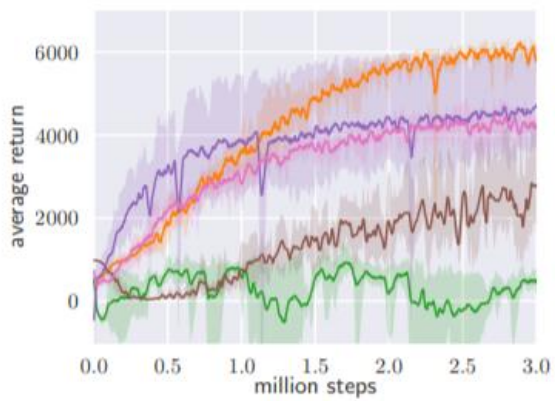
(a) Hopper-v1



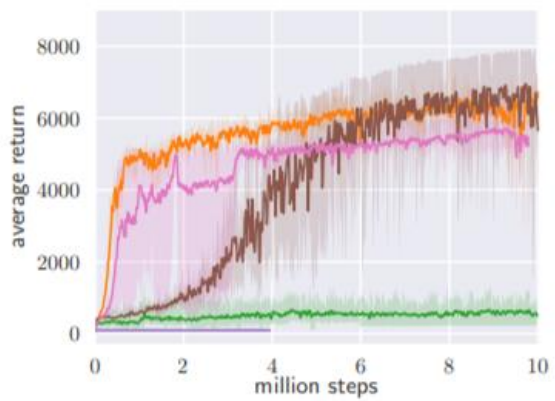
(b) Walker2d-v1



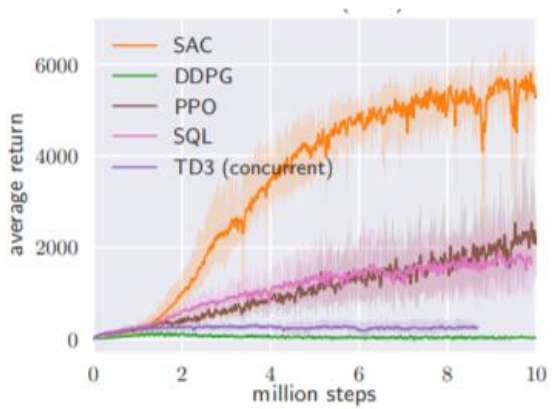
(c) HalfCheetah-v1



(d) Ant-v1



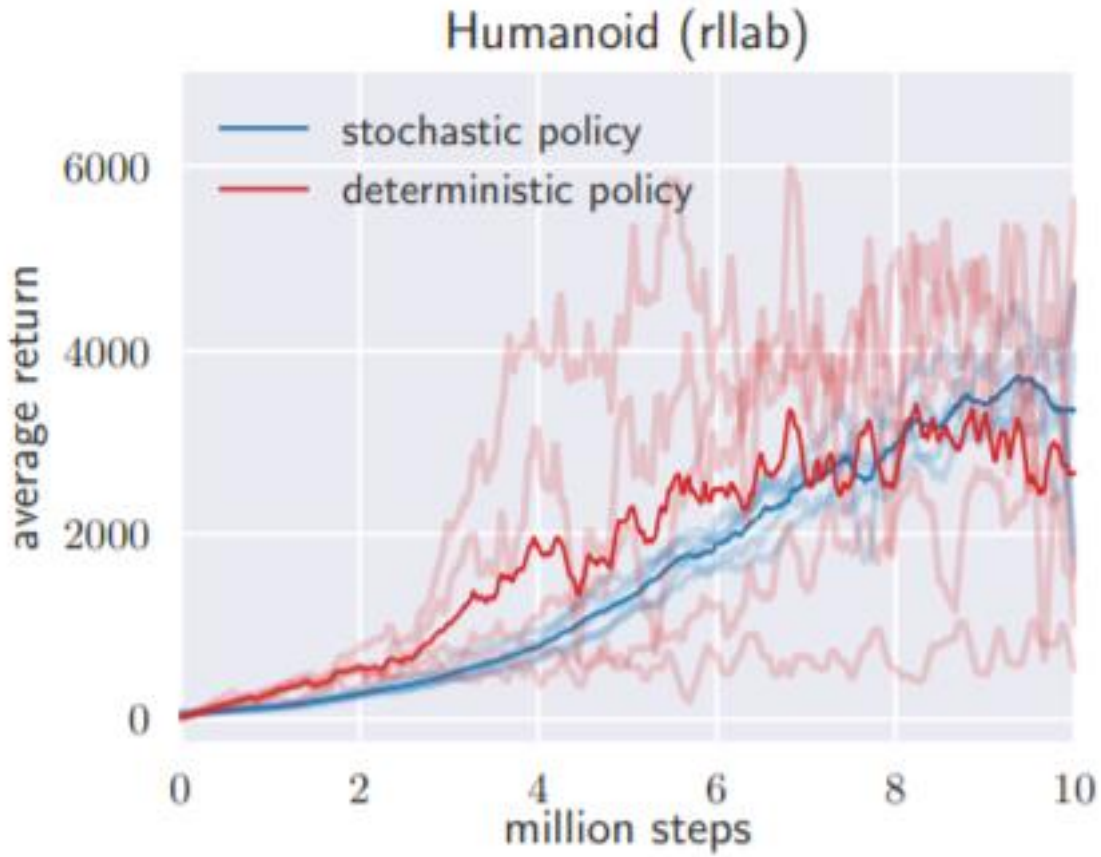
(e) Humanoid-v1



(f) Humanoid (rllab)

Soft Actor-Critic (SAC)

随机性策略VS 确定性策略



Soft Actor-Critic (SAC)

针对温度参数难以选择的问题

$$\max_{\pi_0,\ldots,\pi_T} \mathbb{E}\Big[\sum_{t=0}^{T} r(s_t,a_t)\Big] \text{s.t. } \forall t, \boxed{\mathcal{H}(\pi_t) \geq \mathcal{H}_0}$$

给策略加入最小熵的约束

$$h(\pi_T) = \mathcal{H}(\pi_T) - \mathcal{H}_0 = \mathbb{E}_{(s_T,a_T)\sim\rho_\pi}[-\log \pi_T(a_T|s_T)] - \mathcal{H}_0$$

$$f(\pi_T) = \begin{cases} \mathbb{E}_{(s_T,a_T)\sim\rho_\pi}[r(s_T,a_T)], & \text{if } h(\pi_T) \geq 0 \\ -\infty, & \text{otherwise} \end{cases}$$

$$\Downarrow$$

$$L(\pi_T,\alpha_T) = f(\pi_T) + \alpha_T h(\pi_T)$$

$$\Downarrow$$

$$\max_{\pi_T} f(\pi_T) = \min_{\alpha_T \geq 0} \max_{\pi_T} L(\pi_T,\alpha_T)$$

Soft Actor-Critic (SAC)

SAC with Automatically Adjusted Temperature

Algorithm 1 Soft Actor-Critic

Input: θ_1, θ_2, ϕ ▷ Initial parameters

$\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$ ▷ Initialize target network weights

$\mathcal{D} \leftarrow \emptyset$ ▷ Initialize an empty replay pool

for each iteration **do**

  for each environment step **do**

    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$ ▷ Sample action from the policy

    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ ▷ Sample transition from the environment

    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$ ▷ Store the transition in the replay pool

  end for

  for each gradient step **do**

    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters

    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update policy weights

    $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$ ▷ Adjust temperature

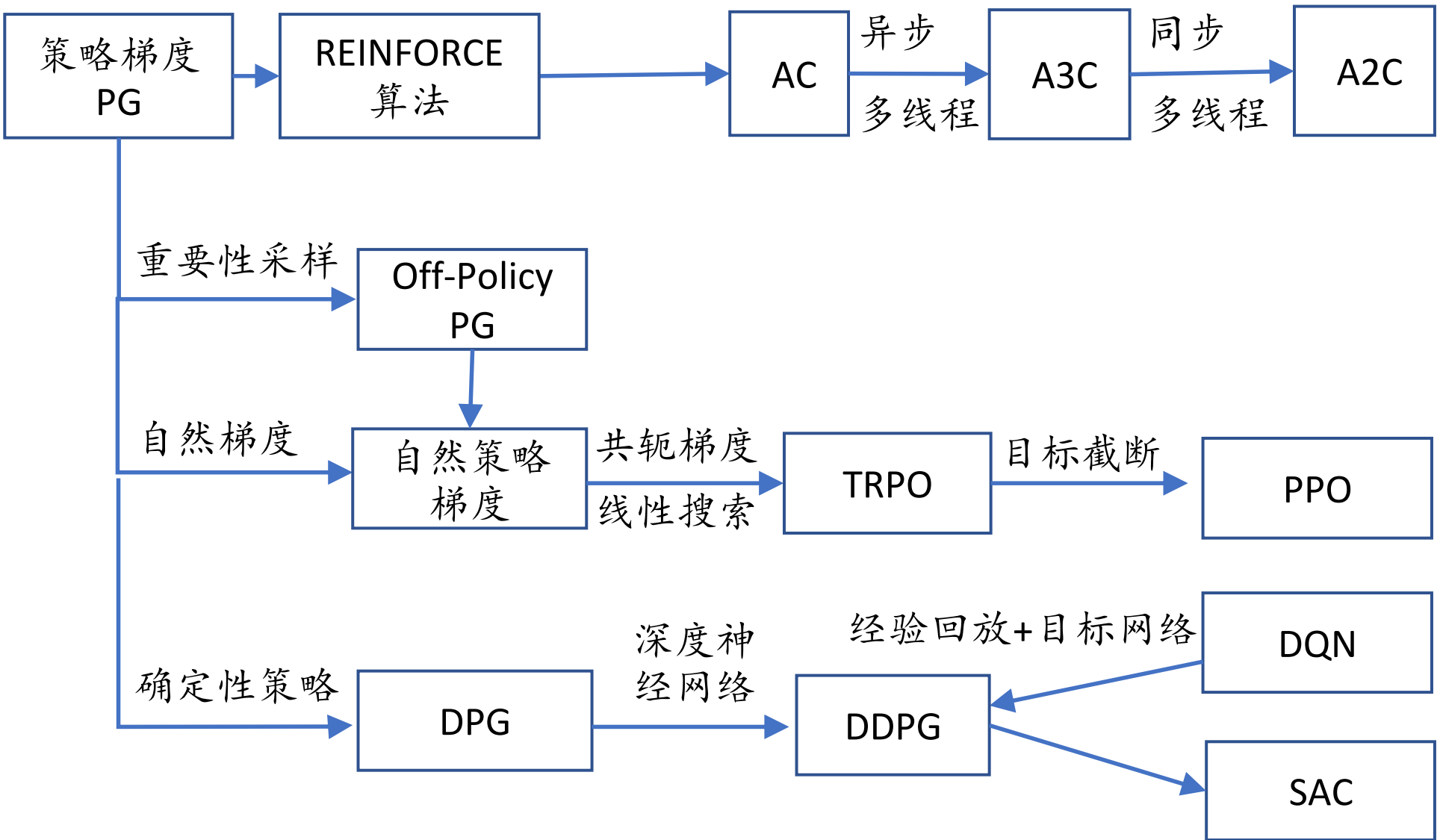
    $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target network weights

  end for

end for

Output: θ_1, θ_2, ϕ

Soft Actor-Critic (SAC)



Soft Actor-Critic (SAC)

- 尽量减少方差并保持偏差不变以稳定训练过程
- 减少数据相关性的办法:异步并行和经验回放
- 带有熵正则的回报函数 A3C
- Actor-Critic可以共享网络的低层参数然后两个输出头分别为策略和值函数
- 可以学习一个确定性的策略而不是一个随即策略： DDPG
- 在策略更新上施加步长约束: TRPO/PPO
- 最大化策略的熵度量从而鼓励探索: SAC

DRL算法实现

SpinningUp from OpenAI: https://spinningup.openai.com/

OpenAI
Spinning Up

Stable-baseline: https://stable-baselines.readthedocs.io/