法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

关注 小象学院

第9讲　从AC到A3C

强化学习

主讲人：叶梓

上海交通大学博士

主要研究方向：机器学习、深度学习、人工智能

本章内容

- PG的问题与AC的思路
- AC类方法的发展历程
- Actor-Critic基本原理
- DPG方法
- DDPG方法
- A3C方法
- 案例

回忆一下REINFORCE

- 原始的 Policy Gradient 往往采用的回合更新，也就是要到一轮结束后才能进行更新。其更新过程如下：

  - Update parameters by stochastic gradient ascent
  - Using policy gradient theorem
  - Using return v_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

function REINFORCE
  Initialise θ arbitrarily
  for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
    for $t = 1$ to $T - 1$ **do**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
    end for
  end for
  return θ
end function

PG的问题与AC方法的思路

- 基本PG方法采用的是回合更新（类似于蒙特卡罗法），但这样有点问题，最后的结果好并不能说明其中每一步都好。
- 因此一个很直观的想法就是能不能抛弃回合更新的做法，而采用单步更新？
- Actor-Critic正是如此。
- 要采用单步更新，意味着我们需要为每一步都即时做出评估。Actor-Critic 中的 Critic 负责的就是评估这部分工作，而 Actor 则是负责选择出要执行的动作。

Actor-only和Critic-only

两类方法各有利弊，而AC算法结合了两者的优点。

critic-only 方法

- 基于value estimation，广泛应用于各种领域，但有一些缺点使它的应用受到局限：
- 1） 难以应用到随机型策略和连续的动作空间。
- 2） value function的微小变化会引起策略变化巨大，从而使训练无法收敛。尤其是引入函数近似（FA）后，虽然算法泛化能力提高了，但也引入了bias，从而使得训练的收敛性更难以保证。

actor-only 方法

- 通过将策略参数化，从而直接学习策略。这样做的好处是与前者相比拥有更好的收敛性，以及适用于高维连续动作空间及随机策略。
- 缺点包括梯度估计variance比较高，且容易收敛到非最优解。
- 另外因为每次梯度的估计不依赖以往的估计，意味着无法充分利用老的信息。

Actor-Critic通俗的解释

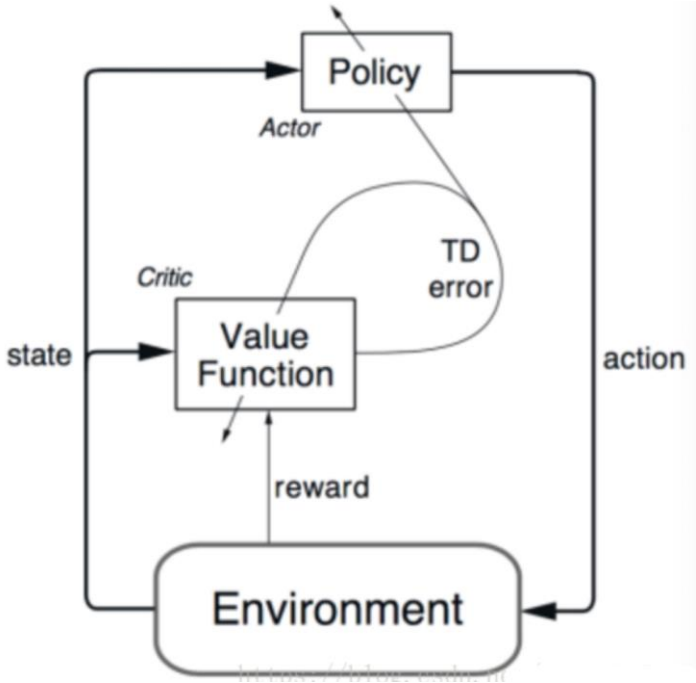
- Actor看到游戏目前的state，做出一个action。
- Critic根据state和action两者，对actor刚才的表现打一个分数。
- Actor依据critic（评委）的打分，调整自己的策略（actor神经网络参数），争取下次做得更好。
- Critic根据系统给出的reward来调整自己的打分策略（critic神经网络参数）。
- 一开始actor随机表演，critic随机打分。但是由于reward的存在，critic评分越来越靠谱，actor表现越来越好。

Actor-Critic基本原理

- 使用蒙特卡洛策略梯度方法使用了长期回报作为状态价值的估计，它虽然是无偏的，但是噪声却比较大，也就是变异性（方差）较高。
- 从原理上说：Actor-Critic算法就是牺牲一定的偏差来减小方差的方法。
- 其核心思想是通过一个单独的模型来估计轨迹的累积回报，而不再用采样完一条轨迹后的真实累积回报。
- 另一个好处就是，Actor-Critic算法可以在每一步进行模型的更新，而不用等到一个回合交互结束后才能更新一次模型。

Actor-Critic基本原理

- Actor-Critic（AC）方法其实是policy-based和value-based方法的结合。
- 在AC框架中，actor负责policy gradient学习策略，而critic负责policy evaluation估计value function。
- 可以看到：
  - 一方面actor学习策略，而策略更新依赖critic估计的value function；
  - 另一方面critic估计value function，而value function又是策略的函数。



Actor-Critic策略梯度（状态行为值）

- 基于Actor-Critic策略梯度学习分为两部分内容：
- 1. Critic：参数化的状态行为价值函数$Q_w(s, a)$

$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$$

带来bias

- 2. Actor：按照Critic部分得到的价值引导策略函数参数θ的更新。
- 这样，Actor-Critic算法遵循的是一个近似的策略梯度：

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q_w(s,a)\right]$$

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s,a)\ Q_w(s,a)$$

基于状态行为值的Actor-Critic

- 基于行为价值critic的简单actor-critic算法如下，
- 使用线性价值函数来近似状态行为价值函数：

$$Q_w(s,a) = \phi(s,a)^\top w$$

function QAC

Initialise s, θ

Sample $a \sim \pi_\theta$

for each step **do**

Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$

Sample action $a' \sim \pi_\theta(s', a')$

$\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$

$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$

$w \leftarrow w + \beta \delta \phi(s, a)$

$a \leftarrow a', s \leftarrow s'$

end for

end function

Actor：策略网络，参数θ

Critic：Q值网络，参数w

按策略获得下一个动作

Critic给出Q的估计，得到TD error

更新actor的参数

更新critic的参数

输出：优化的参数

基于状态值的Actor-Critic

One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

$$\delta = r_t(s_t, a_t) + v_\omega(s_{t+1}) - v_\omega(s_t)$$
$$\omega_{t+1} = \omega_t + \alpha^\omega \delta_t \nabla_\omega v_\omega(s_t)$$
$$\theta_{t+1} = \theta_t + \alpha^\theta \nabla_\theta \log \pi_\theta(a_t|s_t)\delta_t$$

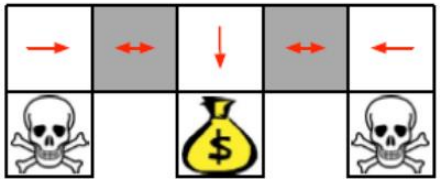
初始化A、C这两个网络

计算TD error

Actor-Critic策略梯度

- 用特征的线性组合来近似$Q_w(s, a)$进而求解策略梯度的方法引入了偏差，一个有偏差的价值函数指导下得到的策略梯度不一定能最后找到较好的解决方案。
- 注意如果其中的Q函数用真实的累积回报来估计，那就是REINFORCE算法。但结合了FA，即$Q_w(s,a)$后，会引入bias。
- 为此Sutton还证明了如果FA是compatible的，那就不会有bias。
- 不过如果我们小心设计近似的$Q_w(s, a)$函数，是可以避免引入偏差的，这样我们相当于遵循了准确的策略梯度。

兼容近似函数



- 兼容近似函数定理，如下：

Theorem (Compatible Function Approximation Theorem)

If the following two conditions are satisfied:

1. *Value function approximator is compatible to the policy*

$$\nabla_w Q_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)$$

梯度相同，即不存在重名

2. *Value function parameters w minimise the mean-squared error*

$$\varepsilon = \mathbb{E}_{\pi_\theta}\left[(Q^{\pi_\theta}(s,a) - Q_w(s,a))^2\right]$$

参数w使得误差的平方最小

Then the policy gradient is exact,

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

则策略梯度是准确的

改进思路

- 实践中，使用$Q_w(s,a)$来计算策略目标函数的梯度并不能保证每次都可行。
- 对于随机变量X，其方差$DX=EX^2-(EX)^2$，如果EX^2比较小的话，那么方差就会小了。自然想到给X减去一个值。
- 如果先去掉状态的变异性的影响，这样状态行为值的变异性可否小一点？

“基线”函数

- “基线”的基本思想是从策略梯度里抽出一个基准函数B(s)，要求这一函数仅与状态有关，与行为无关，这样才不会不改变梯度本身。
- B(s)的特点是能在不改变状态行为价值期望的同时降低其Variance。
- 当B(s)具备上述特点时，下面的推导成立：

$$\begin{aligned}\mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)B(s)\right] &= \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s)\sum_a \nabla_\theta \pi_\theta(s,a)B(s) \\ &= \sum_{s\in\mathcal{S}} d^{\pi_\theta} B(s)\nabla_\theta \sum_{a\in\mathcal{A}} \pi_\theta(s,a) \\ &= 0\end{aligned}$$

上述推导过程的解释

- 策略函数对数的梯度与基线函数乘积的期望，可以表示为策略函数梯度与B(s)的乘积对所有状态及行为分布求和的形式。

$$\mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) B(s)\right] = \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_{a} \nabla_\theta \pi_\theta(s,a) B(s)$$

- 由于B(s)与行为无关，可以将其从针对行为a的求和中提出来，同时也可以把梯度从求和符号中提出来：
  - 梯度的和等于和的梯度

$$= \sum_{s\in\mathcal{S}} d^{\pi_\theta} B(s) \nabla_\theta \sum_{a\in\mathcal{A}} \pi_\theta(s,a)$$

和为1

- 而后一项求和符号中的内容的含义为：对所有行为的策略函数求和，这一求和根据策略函数的定义肯定是1，而常数的梯度是0。

“基线”函数

- 原则上，和行为无关的函数都可以作为B(s)。一个很合适的B(s)就是基于当前状态的状态价值函数：

$$B(s) = V^{\pi_\theta}(s)$$

- 可以通过使用一个advantage function，定义为：

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

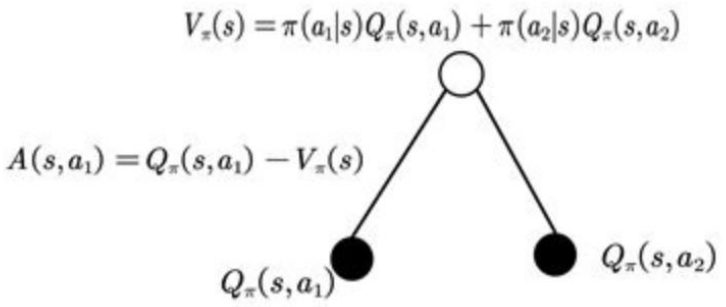
- 这个“优势”函数的现实意义在于，当个体采取行为a离开s状态时，究竟比该状态s总体平均价值要好多少。这样，原目标函数可以写成：

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

- 现在目标函数梯度的意义就变成：为了得到这个“优势”函数，应该如何改变策略参数。

“基线”函数

- 值函数V(s)可以理解为在该状态下所有可能动作所对应的动作值函数乘以采取该动作的概率的和。
- 或者说，值函数V(s)是该状态下所有动作值函数关于动作概率的平均值。而动作值函数Q(s,a)是单个动作所对应的值函数，Q(s,a)- V(s)能评价当前动作值函数相对于平均值的大小。
- 所以，这里的优势指的是动作值函数相比于当前状态的值函数的优势。



“基线”函数

- Advantage 函数可以明显减少状态价值的变异性，因此算法的Critic部分可以去估计advantage函数而不是仅仅估计状态行为价值函数。
- 在这种情况下，需要两个近似函数也就是两套参数，一套用来近似状态值函数，一套用来近似状态行为值函数，以便计算advantage函数，并且通过TD学习来更新这两个价值函数，数学表示如下：

$$V_v(s) \approx V^{\pi_\theta}(s)$$
$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$$
$$A(s,a) = Q_w(s,a) - V_v(s)$$

使用“基线”来减少方差

- 先看看TD误差的计算公式。

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

- 可以得到：TD误差的无偏估计是advantage函数，这同样是根据状态行为值函数的定义推导成立的，即：

$$\begin{aligned}\mathbb{E}_{\pi_\theta}\left[\delta^{\pi_\theta}|s,a\right] &= \mathbb{E}_{\pi_\theta}\left[r + \gamma V^{\pi_\theta}(s')|s,a\right] - V^{\pi_\theta}(s) \\ &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \\ &= A^{\pi_\theta}(s,a)\end{aligned}$$

“基线”函数

- 根据上面，就可以使用TD误差来计算策略梯度：

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

- 实际运用时，使用一个近似的TD误差，即用状态函数的近似函数来代替实际的状态函数：

$$\delta_v = r + \gamma V_v(s') - V_v(s)$$

- 这样做的好处就是，只需要一套参数描述状态价值函数，而不再需要针对行为价值近似函数了。

针对Critic过程

$$w \leftarrow w + \beta\delta\phi(s,a)$$

- 可通过计算不同时间范围内（步长）的TD误差来更新状态价值函数$V_\theta(s)$，此时的Critic过程可以根据时间范围的的长短（步长的多少）来分为：
- MC - 直至Episode结束： $\Delta\theta = \alpha(v_t - V_\theta(s))\phi(s)$
- TD(0) - 一步： $\Delta\theta = \alpha(r + \gamma V(s') - V_\theta(s))\phi(s)$
- TD(λ)的前向视角 - 需要至Episode结束：

$$\Delta\theta = \alpha(v_t^\lambda - V_\theta(s))\phi(s)$$

- TD(λ)的后向视角 - 实时，具备近时记忆功能：

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$
$$e_t = \gamma\lambda e_{t-1} + \phi(s_t)$$
$$\Delta\theta = \alpha\delta_t e_t$$

针对Actor过程

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

- 同样在Actor过程中也可以把时间范围考虑进去用来更新参数，具体公式如下：
- 策略梯度可以表示为。

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

- MC - 直至Episode结束：

$$\Delta\theta = \alpha(v_t - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- TD(0) - 一步：

$$\Delta\theta = \alpha(r + \gamma V_v(s_{t+1}) - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

针对Actor过程

- TD(λ)的前向视角 - 需要至Episode结束：

$$\Delta\theta = \alpha(v_t^{\lambda} - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- TD(λ)的后向视角 - 实时，具备频率记忆和近时记忆功能：

$$\delta = r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)$$
$$e_{t+1} = \lambda e_t + \nabla_\theta \log \pi_\theta(s, a)$$
$$\Delta\theta = \alpha\delta e_t$$

策略梯度方法总结

- 有许多形式各异的策略梯度函数的形式，都是用随机梯度上升算法；

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ v_t\right] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)\right] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^w(s,a)\right] && \text{Advantage Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta\right] && \text{TD Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta e\right] && \text{TD}(\lambda)\ \text{Actor-Critic} \\
G_\theta^{-1}\nabla_\theta J(\theta) &= w && \text{Natural Actor-Critic}
\end{aligned}
$$

- 同样Critic部分使用策略评估来实现，可以使用MC或者TD，TD(λ)等去估计状态价值函数V、行为价值函数Q或advantage函数A等。

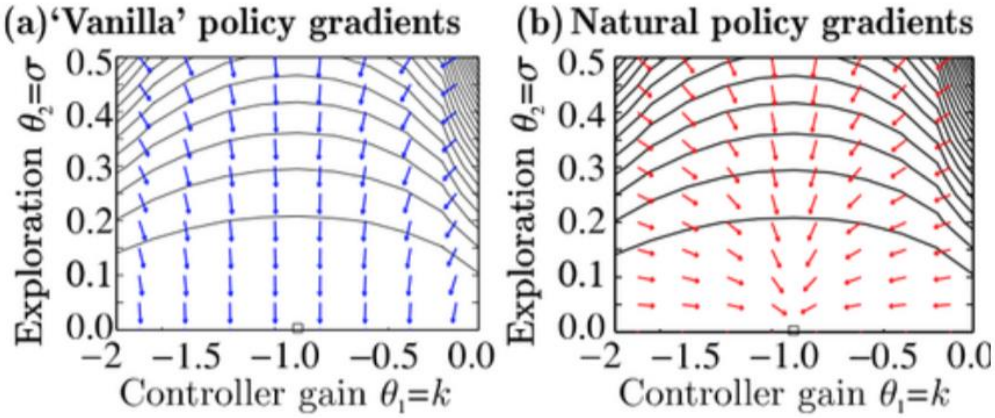
natural policy gradient

- 当以小而固定的数量改变策略时，它能找到最接近原始梯度的上升方向。
- NPG就是在传统的梯度计算的基础上加入Fisher information。

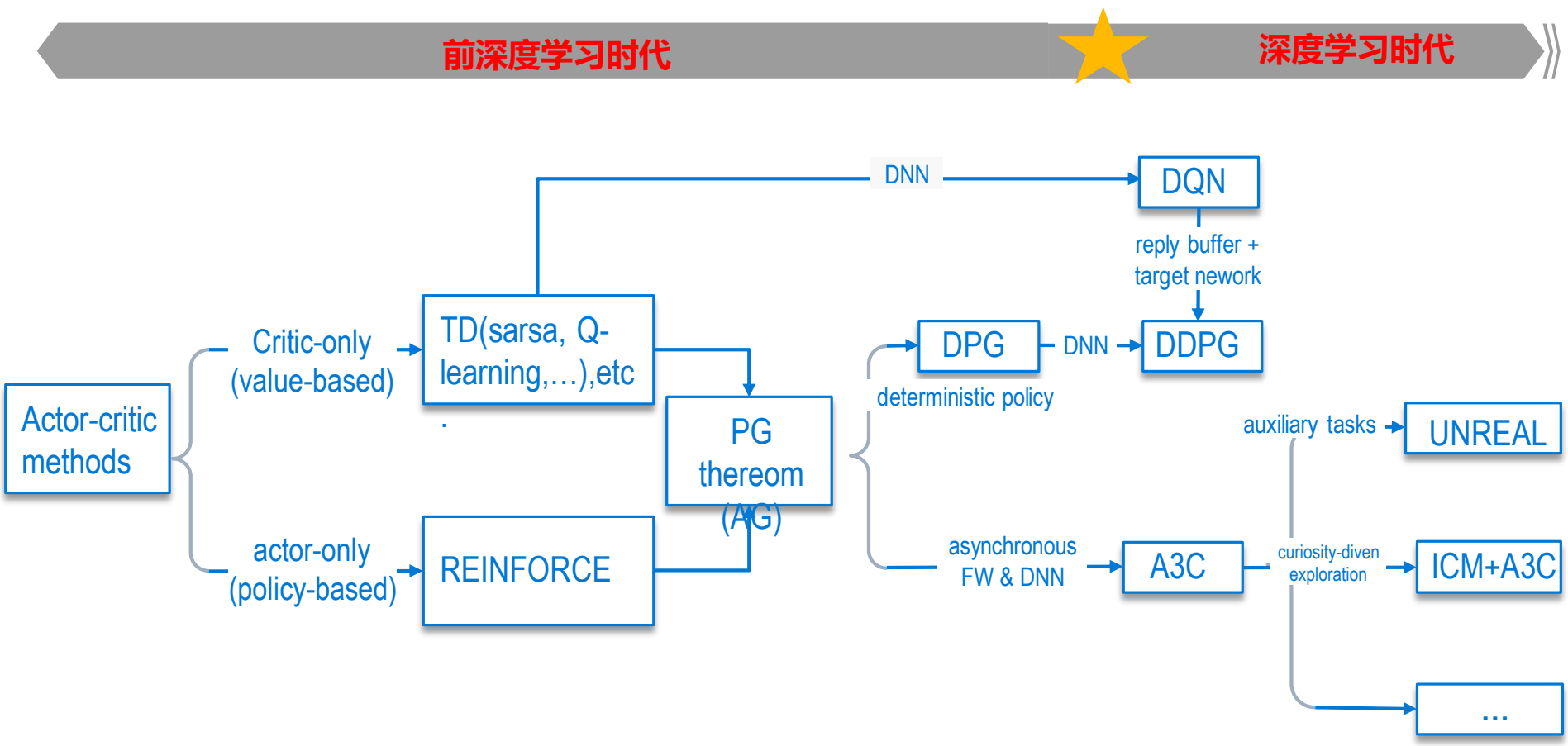
$$\nabla_\theta^{nat}\pi_\theta(s,a) = G_\theta^{-1}\nabla_\theta\pi_\theta(s,a)$$

FIM

$$G_\theta = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)^T\right]$$



Actor-Critic相关技术路线



DPG方法

- DPG的论文主要提出了用于连续动作空间的deterministic policy gradient(DPG)定理和学习确定性目标策略（Deterministic target policy）的off-policy AC类算法。
- 回忆一下：
  - 之前的stochastic policy（随机策略）$\pi_\theta(a|s)$ 定义为动作的分布。
  - 而deterministic policy（确定策略）则为状态到动作的映射$a=\mu_\theta(s)$。
- 其背后的基本思想是，虽然policy可以用概率分布表示，但算法想要的其实就是一个动作而已。

DPG方法

- 强化学习问题的目标函数，就是累计折扣奖励的定义，即状态满足ρ^{π}分布上的累计奖励

$$\nabla_{\theta} J(\pi_{\theta}) = \int_{S} \rho^{\pi}(s) \int_{A} \pi_{\theta}(a|s) r(s,a) da ds$$
$$= E_{s\sim\rho^{\pi}, a\sim\pi_{\theta}}[r(s,a)]$$

- 策略梯度就是沿着使目标函数变大的方向调整策略的参数，它被定义为：

$$\nabla_{\theta} J(\pi_{\theta}) = \int_{S} \rho^{\pi}(s) \int_{A} \nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s,a) da ds$$
$$= E_{s\sim\rho^{\pi}, a\sim\pi_{\theta}}[\nabla_{\theta} \log \pi_{\theta}(a|s) Q^{\pi}(s,a)]$$

- 据上式，目标函数的梯度主要与策略梯度和值函数的期望有关，策略梯度并不依赖于状态分布上的梯度。

DPG方法

- DPG将PG框架推广到deterministic policy（确定策略）。提出DPG的论文证明了deterministic policy gradient不仅存在，而且是model-free形式，同时也是action-value function的梯度，形式简单。
- Deterministic Policy Gradient（确定策略的梯度，DPG）定理给出了目标函数对于策略参数的梯度：

$$\nabla_\theta J(\mu_\theta) = \int_S \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)} ds$$
$$= E_{s\sim\rho^\mu}[\nabla_\theta \mu_\theta(s) \nabla Q^\mu(s,a)|_{a=\mu_\theta(s)}]$$

- 并且证明了它是之前stochastic policy gradient定理在policy的variance趋向于0时的极限。

DPG方法

$$E_{x\sim p(x)}[f(x)] = \int_x p(x)f(x)\mathrm{d}x$$

- 比较一下这两个公式：

$$\nabla_\theta J(\pi_\theta) = \int_S \rho^\pi(s) \int_A \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) da ds$$
$$= E_{s\sim\rho^\pi, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)]$$

随机策略的梯度

$$\nabla_\theta J(\mu_\theta) = \int_S \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)} ds$$
$$= E_{s\sim\rho^\mu}[\nabla_\theta \mu_\theta(s) \nabla Q^\mu(s,a)|_{a=\mu_\theta(s)}]$$

确定策略的梯度

- DPG和随机策略梯度SPG差异在于随机策略梯度中多一个log项，同时期望也不一样；
- 本质上源于随机策略需要再**加一层策略π的期望**，则策略网络π的梯度相当于DPG中的策略梯度再除以策略π，数学转化成log(π)的导数了。

d(log(x))=(1/x)dx

DPG方法

- 上述证明意味着compatible FA，natural gradients等一众理论都可以应用到DPG场景。
- 另外，在高维动作空间问题中，stochastic policy gradient定理中的梯度内积需要在高维空间中采样，而deterministic policy gradient不需要积分，有解析解。
- 这些都是引入deterministic policy后带来的好处。
- 但deterministic policy又会导致exploration不足。

DPG方法

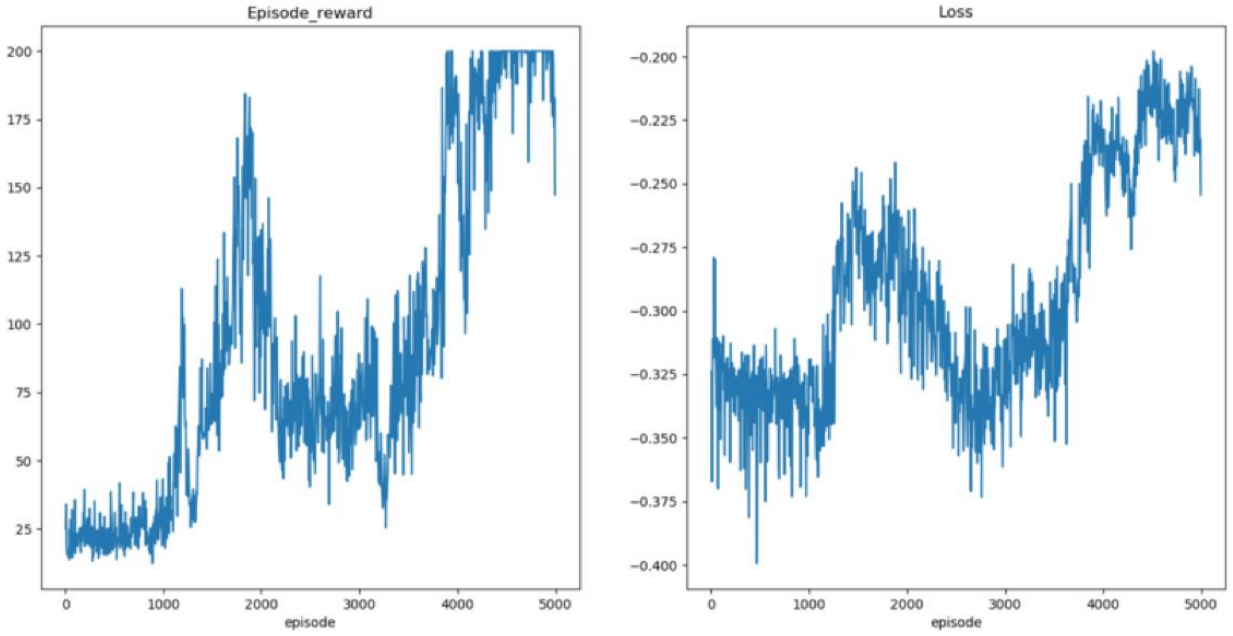
- 以下定理说明，满足一定条件时，可以找到一类兼容函数逼近器$Q_w(s,a)$，它使得梯度$\nabla_a Q_\mu(s,a)$可以用$\nabla_a Q_w(s,a)$替代，而不会影响确定性政策梯度。

Theorem 3. *A function approximator $Q^w(s,a)$ is compatible with a deterministic policy $\mu_\theta(s)$, $\nabla_\theta J_\beta(\theta) = \mathbb{E}\left[\nabla_\theta \mu_\theta(s)\, \nabla_a Q^w(s,a)|_{a=\mu_\theta(s)}\right]$, if*

1. $\nabla_a Q^w(s,a)|_{a=\mu_\theta(s)} = \nabla_\theta \mu_\theta(s)^\top w$ *and*
2. *w minimises the mean-squared error, $MSE(\theta,w) = \mathbb{E}\left[\epsilon(s;\theta,w)^\top \epsilon(s;\theta,w)\right]$ where $\epsilon(s;\theta,w) = \nabla_a Q^w(s,a)|_{a=\mu_\theta(s)} - \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}$*

DPG方法

- 根据实验，DPG训练起来并不稳定，模型参数初始化对训练效果也有着较大的影响，需要多次尝试。
- 有时reward收敛一段时间后又会快速下降，出现周期性的变化，从图中也可以看出训练过程的不稳定。



DPG方法

- 该论文根据DPG定理论文提出了deterministic AC算法。先是基于on-policy(Sarsa)和off-policy（Q-learning）情况提出了deterministic AC算法，然后基于OffPAC算法提出了OPDAC（off-policy deterministic actor-critic algorithm）算法。
- OPDAC算法将前面的OffPAC中的梯度公式扩展到deterministic policy，给出off-policy deterministic policy gradient：

$$\nabla_\theta J_\beta(\mu_\theta) \approx \mathbb{E}_{s\sim\rho^\beta}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}]$$

- 但它们都有收敛性问题（因为FA引入的bias和off-policy引入的不稳定性），于是论文最后结合compatible FA和gradient TD给出了COPDAC-GQ算法。

DDPG方法

- 近年来随着DL的兴起，PG进入DL时代就成了必然趋势了。value-based方法已和DQN结合产生了DQN，而DL在其中其实主要充当了FA的角色。
- 而另一方面PG中也需要FA，那PG和DL的结合也很自然了。
- DDPG算法是一种model-free, off-policy, actor-critic的DRL算法。
- 原始的DQN无法应用到连续动作空间。直觉上当然可以通过离散化动作空间解决，但会导致维度灾难（curse of dimensionality）。
- DDPG基于DPG算法，它将AC方法与DQN结合。

DDPG方法

- DDPG主要解决高维连续动作控制问题，最大的特点就是简单。原因在于DDPG使用了以下两大技术，DQN同时再加上batch normalization。
- 回顾一下DQN的理论意义，之前，学术界普遍认为大型非线程FA用于学习value function或者action-value function理论上收敛不可保证，且实际学习效果也不稳定。
- 但不使用大型非线性FA，算法又扩展不到大型状态空间，只能在小规模场景中用。DQN证明了用非线程FA来表示value function也可以稳定收敛。

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

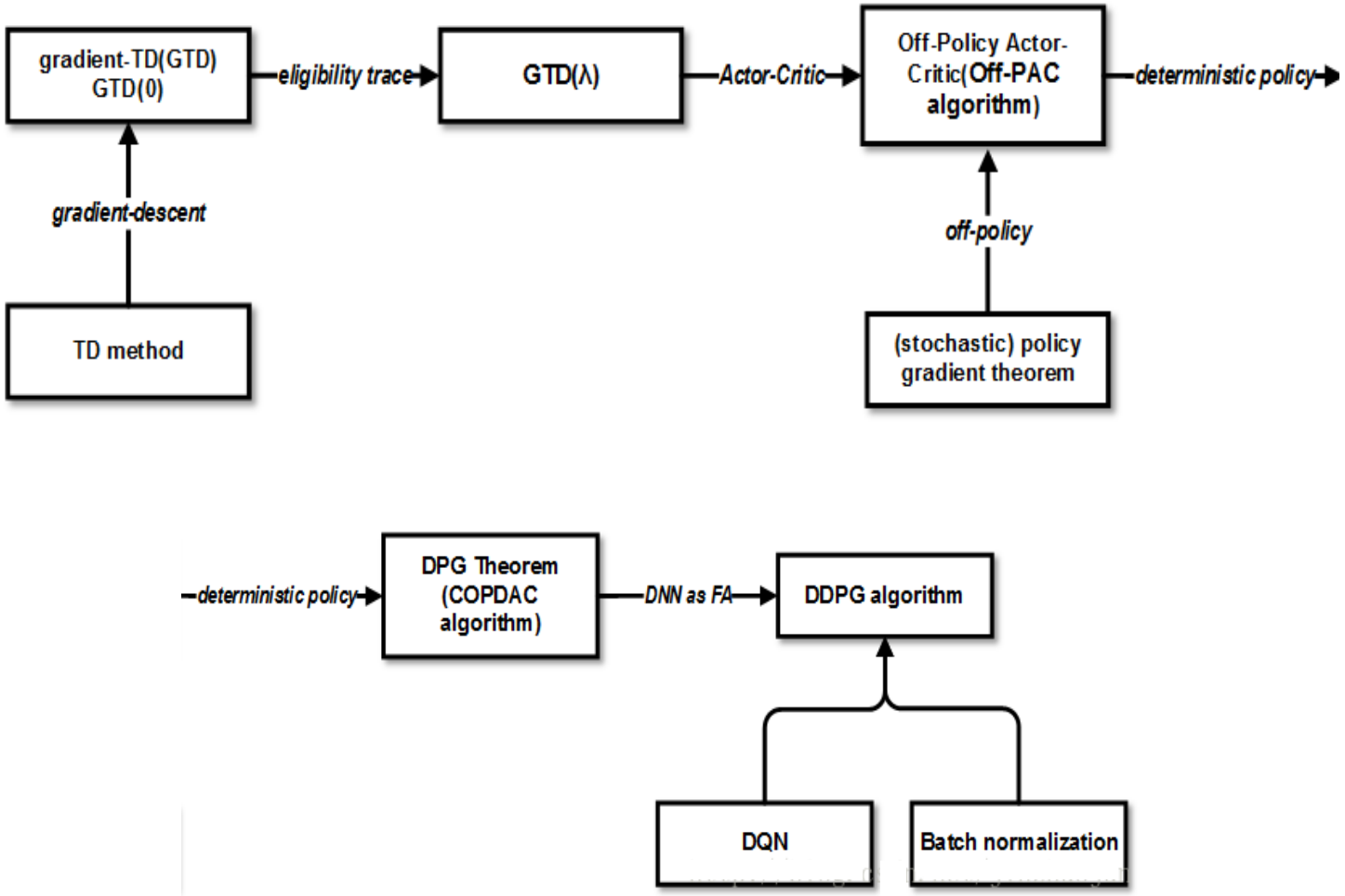
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

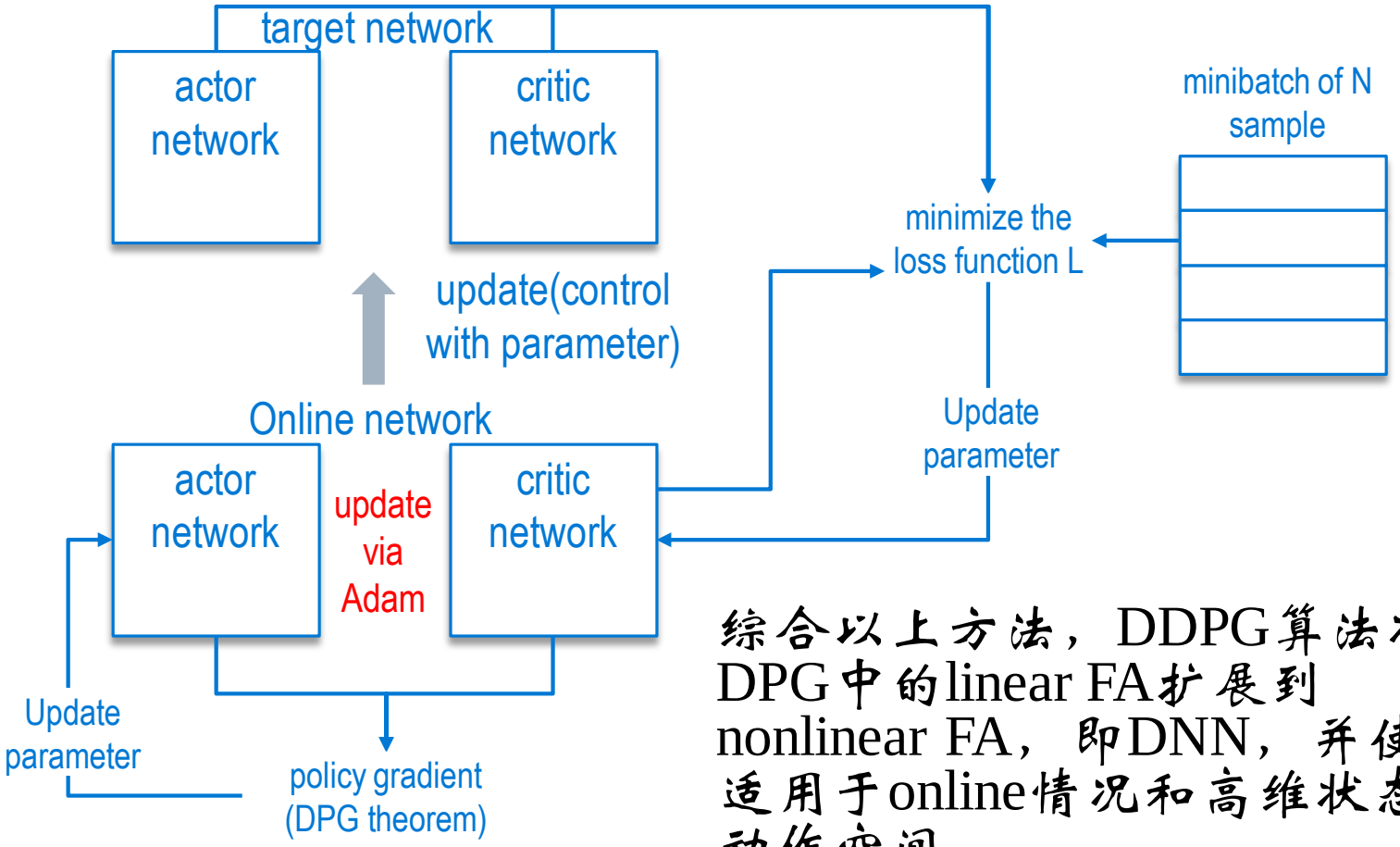
$$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- 在每次SGD时，通过mini-batch来对相应的activation做规范化操作，使得结果（输出信号各个维度）的均值为0，方差为1.
- 最后的“scale and shift”操作则是为了让因训练所需而“刻意”加入的BN能够有可能还原最初的输入。

DDPG方法的由来


gradient-TD(GTD)
GTD(0)
eligibility trace
GTD(λ)
Actor-Critic
Off-Policy Actor-Critic(Off-PAC algorithm)
deterministic policy
gradient-descent
TD method
off-policy
(stochastic) policy gradient theorem
deterministic policy
DPG Theorem (COPDAC algorithm)
DNN as FA
DDPG algorithm
DQN
Batch normalization

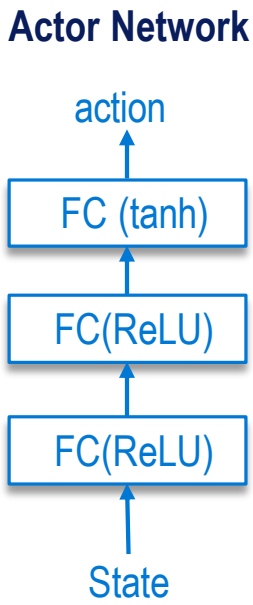
DDPG方法

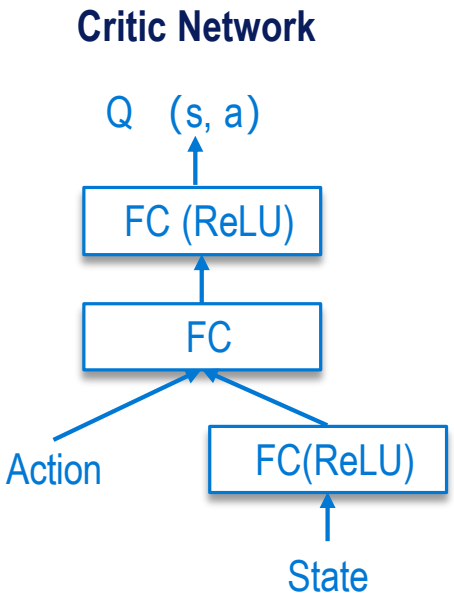


综合以上方法，DDPG算法将DPG中的linear FA扩展到nonlinear FA，即DNN，并使之适用于online情况和高维状态和动作空间。

DDPG方法

- 另外还有就是连续动作空间中的exploration问题。这里使用了加一个noise process中产生的noise sample到actor policy来得到探索性的 behavior policy。
- 算法中actor和critic均用DNN表示，分为称为actor network和critic network。
- 它们分别是deterministic policy μ和value-action function Q的FA，参数分别为θ^Q和θ^μ。





DDPG方法

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

复制到target_N

确定策略加一点噪声

Replay buffer

critic的目标函数

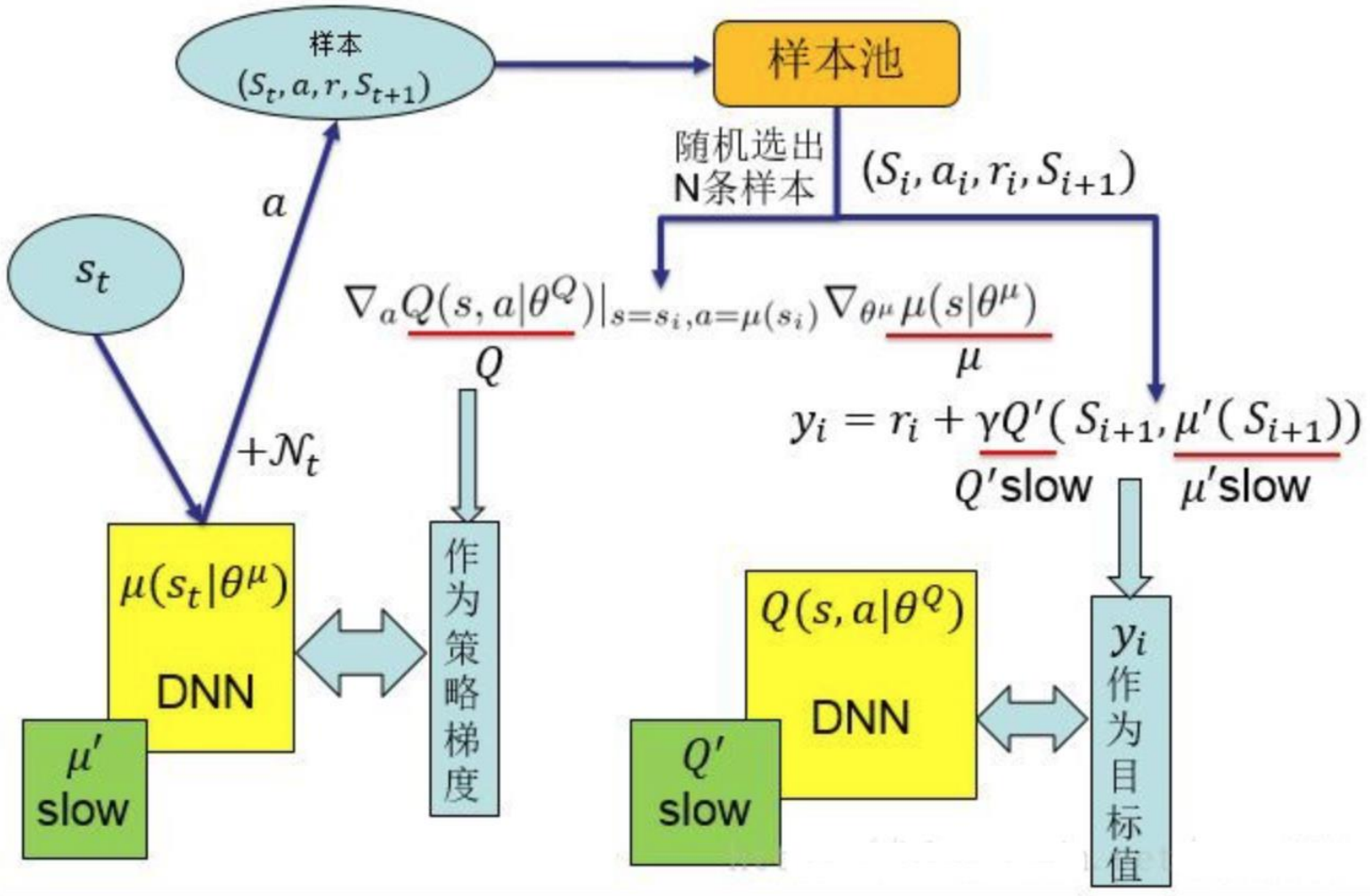
DPG

target_N参数微调一点点

DDPG方法

- 而当输入为image（即raw pixels）时，按套路需要加卷积层和全连接层（3层Conv，2层FC）来学习特征。
- 这里用的深度网络优化方法为Adam（DL中的主流优化算法之一）。
- 在迭代更新过程中，先积累experience replay buffer直到达到minibatch指定个数，然后根据sample分别更新两个DNN。
- 先更新critic，通过loss函数L更新参数θ^Q。然后，通过critic得到Q函数相对于动作a的梯度，因为在actor的更新公式（也就是DPG定理）中会用到，然后应用actor更新公式更新参数。
- 刚才得到的θ^μ和θ^Q，对于θ的更新，会按比例（通过参数τ）更新到target network。这个target network会在下一步的训练中用于predict策略和Q函数值。

DDPG总结



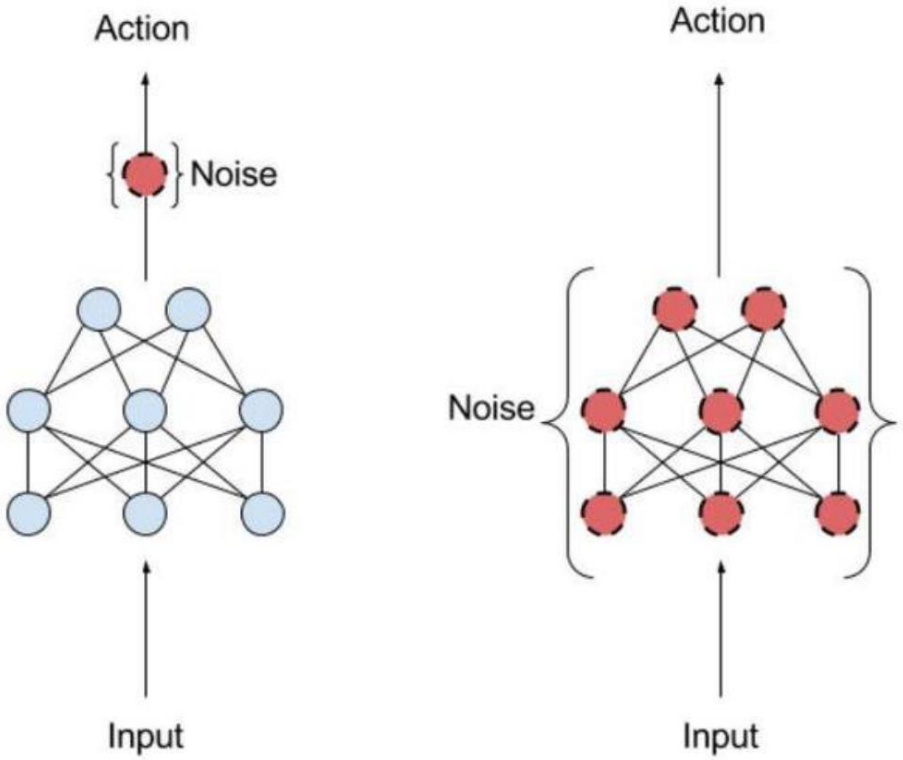
小象学院 ChinaHadoop.cn

DDPG学习中的小trick

- 传统的DQN采用的是一种被称为‘hard’模式的target-net网络参数更新：
  - 每隔一定的步数就将eval-net中的网络参数赋值过去，
- 与传统的DQN不同的是，在DDPG中，采用的是一种‘soft’模式的target-net网络参数更新：
  - 每一步都对target-net网络中的参数更新一点点，
  - 这种参数更新方式经过试验表明可以大大的提高学习的稳定性。

DDPG学习中的小trick

- DDPG 如何对动作进行探索？解决方案是在参数空间或动作空间中添加噪声。



在参数空间上添加噪声比在动作空间上添加要好得多。

A3C方法

- A3C算法（asynchronous advantage actor-critic），不仅适用于离散也适用于连续动作空间的控制问题。
- A3C是2016提出的方法，到现在已有不少基于该算法的改进，但基本思想和框架没有大改，如今仍是最牛的DRL算法之一。
- 传统经验认为：online的RL算法在和DNN简单结合后会不稳定。主要原因是观察数据往往波动很大且前后sample相互关联。
- 像TRPO方法通过将经验数据batch，或者像DQN中通过experience replay memory对之随机采样，这些方法有效解决了前面所说的两个问题，但是也将算法限定在了off-policy方法中。

A3C方法

- A3C提出了另一种思路，即通过创建多个agent，在多个环境实例中并行且异步的执行和学习。
- 另外，将value function的估计作为baseline可以使得PG方法有更低的variance。
- 它还有个潜在的好处是不那么依赖于GPU或大型分布式系统。A3C可以跑在一个多核CPU上。总得来说，A3C更多是工程上的设计和优化。
- 通过这种方式，在DNN下，解锁了一大批online/offline的RL算法（如Sarsa, AC, Q-learning）。

A3C方法

- A3C方法可以将one-step Sarsa, one-step Q-learning, n-step Q-learning都扩展至多线程异步架构。注意这几种方法是完全不同的：
  - AC是on-policy的policy搜索方法，而Q-learning是off-policy value-based方法。
  - 这体现了该框架的通用性。
- 简单地说，每个线程都有agent运行在环境的拷贝中，每一步生成一个参数的梯度，多个线程的这些梯度累加起来，一定步数后一起更新共享参数。
- A3C 算法和DDPG类似，通过 DNN 拟合 policy function 和 value function的估计。

A3C与DDPG的区别

- 但是不同点在于：
  - A3C 中有多个agent 对网络进行 asynchronous update，这样带来了样本间的相关性较低的好处，
  - 因此A3C 中也没有采用 Experience Replay的机制；这样A3C 就能支持 online 的训练模式了。
  - A3C 有两个输出，其中一个softmax output 作为policy $\pi(a_t|s_t;\theta)$，而另一个linear output为 value function $V(s_t;\theta_v)$，其余layer都是共享的。
  - A3C 中的Policy network 的评估指标采用的是上面比较了多种评估指标的论文中提到的Advantage Function(即A值)，而不是 DDPG 中单纯的 Q 值。
  - A3C没使用deterministic policy。

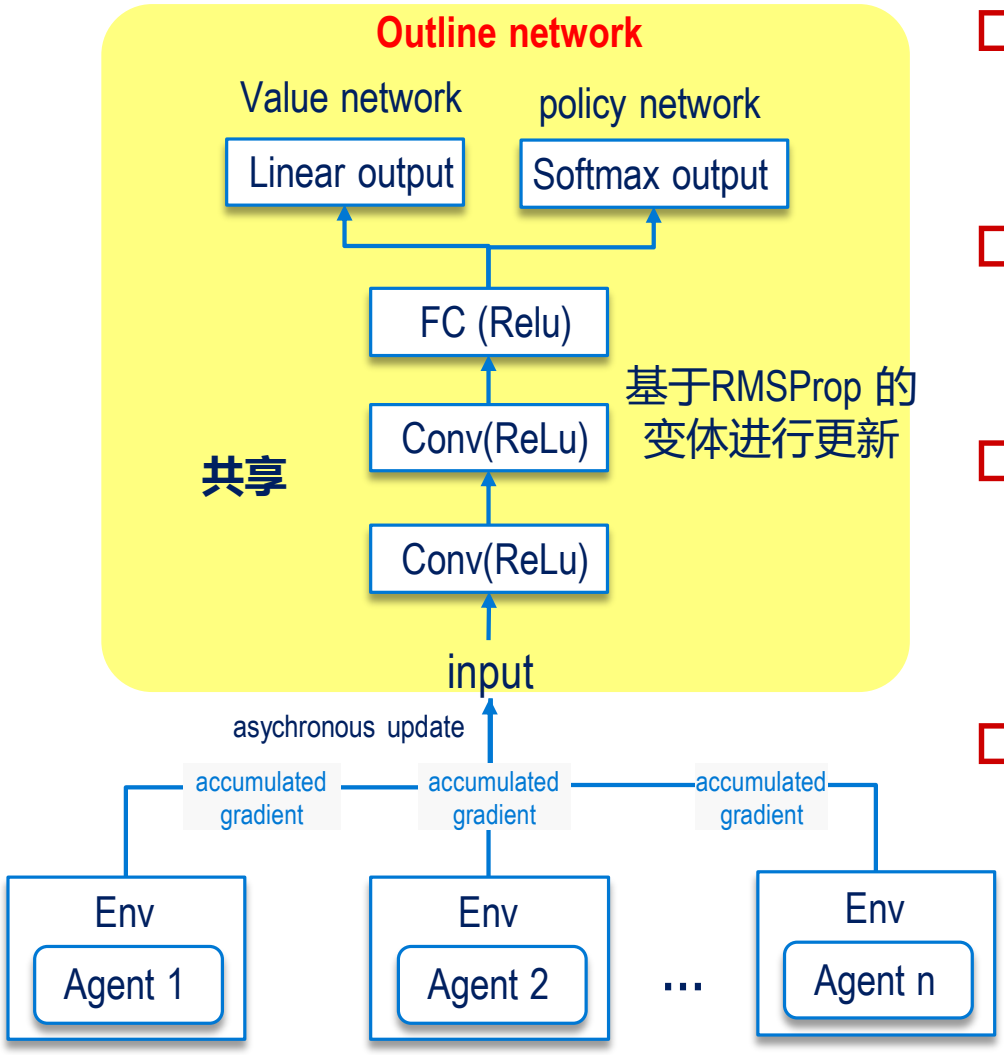
A3C方法的优点

- 它运行在单个机器的多个CPU线程上，而非使用parameter server的分布式系统
  - 这样就可以避免通信开销和利用lock-free的高效数据同步方法。
- 多个并行的actor可以有助于exploration。在不同线程上使用不同的探索策略，使得经验数据在时间上的相关性很小
  - 这样不需要DQN中的experience replay也可以起到稳定学习过程的作用，意味着学习过程可以是on-policy的。
- 好处还包括更少的训练时间。

A3C方法

- 另外，A3C还运用了一个古老的技巧，即将policy的entropy加到目标函数可以避免收敛到次优确定性解。
- 直观上，加上该正则项后目标函数更鼓励找entropy大的，即形状“扁平”的分布，这样就不容易在训练过程中聚集到某一个动作上去。
- 在优化方法上，A3C使用了基于RPMProp的一种变体。

A3C方法总体架构



- 作者在Atari 2600，TORCS，MoJoCo等平台上做了一系列的实验：
- 实验证明在Atari的一些游戏中n-steps方法比one-step方法快。
- 另外policy-based advantage actor-critic方法比value-based method的效果更好。
- 实验中通过6个游戏学习了超参数，然后在其它57个游戏中固定这些参数，证明了算法的通用性。

A3C方法更新算法

- Actor网络和Critic网络各自更新的方式：

  - Estimate state-value function

$$V(s, \mathbf{v}) \approx \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \ldots | s\right]$$

  - Q-value estimated by an n-step sample

$$q_t = r_{t+1} + \gamma r_{t+2} \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}, \mathbf{v})$$

  - Actor is updated towards target

$$\frac{\partial l_u}{\partial \mathbf{u}} = \frac{\partial \log \pi(a_t | s_t, \mathbf{u})}{\partial \mathbf{u}} (q_t - V(s_t, \mathbf{v}))$$

  - Critic is updated to minimise MSE w.r.t. target

$$l_v = (q_t - V(s_t, \mathbf{v}))^2$$

A3C方法

- A3C还讨论了算法的scalability，即多线程方法的可扩展性。
- 结果显示当工作线程增多时，算法可以获得显著的加速。16个线程的时候，就加速一个数量级了。而且在一些算法中（如one-step Q-learning和Sarsa）还达到了超过线性的加速比。
  - 这一现象可以用多线程减少了one-step方法的有偏性来解释。
- 总之，从实验数据可以看到，A3C方法，无论是离散还是连续动作空间问题，效果都远好于前面说的各种方法。

A2C方法

- A3C算法表现十分优异，但是其中的异步更新是否是必要的？
- 凭直觉，异步或者同步更新并不是决定算法优劣的主要因素。那么为什么不尝试使用同步更新方法呢？
- 这就是A2C方法的来源。OpenAI的官方博客中也提到A2C的效果优于A3C。
- 可以在Baseline项目中a2c文件夹下看到A2C的实现。

相关论文

- Sutton
  - 《Policy Gradient Methods for Reinforcement Learning with Function Approximation》
- ICML 2014 DeepMind
  - 《Deterministic Policy Gradient Algorithms》
- ICLR 2016 DeepMind
  - 《Continuous Control with Deep Reinforcement Learning》
- ICML 2016 DeepMind
  - 《Asynchronous Methods for Deep Reinforcement Learning》

参考资料

- https://blog.csdn.net/jinzhuojun/article/details/72851548
- https://blog.csdn.net/dukuku5038/article/details/84670464
- https://blog.csdn.net/weixin_37895339/article/details/74612572
- https://www.jianshu.com/p/e037d42ab6b1
- https://blog.csdn.net/gsww404/article/details/80403150
- https://www.jianshu.com/p/6fe18d0d8822
- https://blog.csdn.net/gsww404/article/details/80403150

问答互动

在所报课的课程页面，
1、点击“全部问题”显示本课程所有学员提问的问题。
2、点击“提问”即可向该课程的老师和助教提问问题。

联系我们

小象学院：互联网新技术在线教育领航者

– 微信公众号：**小象学院**