2020 11 26 What Matters On-Policy Deep Actor Critic Method: A Large Scale Study

Chen Gong

26 December 2020

这是一篇调参的文章，在 ICLR 上取得了 (7,7,9,9) 的超高得分。小编还想这又是哪的大作，在 arxiv 上一查，不出意料，是 Google brain 的论文。之前小编没有详细的阅读此类文章，而在实际工程和算法实现过程中，强化学习的参数极度的敏感，关于参数的设置非常的重要，错误的参数设置可能导致算法完全不 work。参数的设置一直是一个大问题，而且由于参数选择通常没有在文献中广泛讨论，导致算法在论文上的效果的和实际实现之间有着巨大的差异（小编就碰到过好几回，满满的都是泪）。这无疑极度的拉后了强化学习的发展，所以作者提出，全面的分析强化学习参数的影响对于推动强化学习的进程有着非常重要的作用！本文中，作者在五种不同复杂性的连续控制环境中训练了 25 万个智能体，并对 on-policy 的 Actor-Critic 算法实现提供了深刻的见解和实用的建议。通过这篇文章不仅可以系统的梳理 on-policy 的相关算法，还能通过参数的变化，进一步理解算法的本质。非常值得阅读!

1 Conference Name

The International Conference on Learning Representations (ICLR 2021)

2 强化学习基础背景

首先，要了解这些参数到底有哪些，它们如何在强化学习中起作用。一些简单的基础，如价值函数，策略梯度这些东西，本文就不做过多的描述了。

2.1 数据收集和循环优化过程

强化学习算法通常使用当前策略和环境进行交互。通常，我们会设置 `num_envs (C1)` 个环境，每一次迭代中，所有环境同步运行，从当前策略函数中样得到动作，直到收集到 `iteration_size (C2)` 次转移为止。这意味着每个环境将收集到了 `iteration_size (C2) / num_envs (C1)` 次转移。然后，我们执行 `num_epochs (C3)` 次批梯度更新，在每个片段中，把数据拆分成 `batch_size (C4)` 的大小。实际这样的做法在 A2C 等 on-policy 算法中非常常见。增加重复采样次数，使得其不再严格的是 On-Policy 算法，并且会增加采样复杂度，在优化的过程中消耗更多的计算资源。

2.2 优势函数估计

令 V 为一些策略的近似价值函数，$V \approx V^{\pi}$。作者在 on-policy actor critic 算法中实验了三种最常用的优势函数估计方法。为什么要引入优势函数呢？优势函数是为了减小方差。**强化学习中不管是做值优化还是策略优化，如果使用基于梯度的优化方法，对目标函数的梯度基本都是关于状态空间和动作空间的期望的形式，但是我们无法遍历状态空间和动作空间，只能使用采样作为近似的梯度，每次采样得到的梯度其实就是一个随机变量，是状态和动作这两个随机变量的函数。所谓方差大，是指这个随机变量方差大**。而样本有限，一般来说估计越准，方差越大，如果是对累积回报的无偏估计（即采样轨迹 accumulate reward 的期望值数学上等于其真实值），那么方差应该就是方差很大；而方差大会使得训练不稳定，结果难以复现。

1. **N-Step return**

$$\hat{V}_t^{(N)} = \left(\sum_{i=t}^{t+N-1} \gamma^{i-t} r_i \right) + \gamma^N V\left(s_{t+N}\right) \approx V^{\pi}\left(s_t\right) \tag{1}$$

参数 N 控制着偏差和方差间的平衡，N 值越大估计值更接近真实的回报，具有更小的偏差和更大的方差。

2. **通用优势估计 (GAE)**

GAE 基本上可以算是目前强化学习算法中最常用的优势函数估计方法了，GAE 中结合了多步回报：

$$\hat{V}_t^{\mathrm{GAE}} = (1-\lambda) \sum_{N>0} \lambda^{N-1} \hat{V}_t^{(N)} \approx V^{\pi}\left(s_t\right) \tag{2}$$

其中，$0 < \lambda < 1$ 是其中的超参数 `C8`，控制着偏差和方差的平衡。优势估计函数的计算为：

$$\hat{A}_t^{\mathrm{GAE}} = \hat{V}_t^{\mathrm{GAE}} - V\left(s_t\right) \approx A^{\pi}\left(s_t, a_t\right) \tag{3}$$

其可以计算出在线性时间内某一片段中遇到的所有状态的估计值。**GAE 中考虑的是带有折扣因子形式的价值函数和状态价值函数，从而组成带有折扣因子形式的优势函数。然后将无穷个时刻的优势函数累积求和，和 $TD(\lambda)$ 非常相似，不过 $TD(\lambda)$ 是对值函数的估计，而 GAE 是对优势函数的估计。本质上，λ 和 γ 都是控制方差和精度，其中 γ 体现了“短视”到“长视”的折中，而 λ 平衡的是要精度还是方差**。具体推导，请看`https://zhuanlan.zhihu.com/p/139097326`。

3. **V-trace** $(\lambda, \bar{c}, \bar{\rho})$：

V-trace $(\lambda, \bar{c}, \bar{\rho})$ 是 GAE 的扩展，类似于 PPO，其中引入了截断的重要性采样，以说明当前策略和经验策略之间差距不能过大，这个和 PPO 的思想出发点是一样的。其中，λ 参数和 GAE 中是一样的，而 $\bar{c}, \bar{\rho}$ 也是两种不同参数的截断域。其中，迭代方程为：

$$v_s \overset{\text{def}}{=} V\left(x_s\right) + \sum_{t=s}^{s+n-1} \gamma^{t-s} \left(\prod_{i=s}^{t-1} c_i \right) \delta_t V \tag{4}$$

其中，时间差分为：$\delta_t V \overset{\text{def}}{=} \rho_t(r_t + \gamma V(x_{t+1}) - V(x_t))$；并且，$\rho_t \overset{\text{def}}{=} \min\left(\bar{\rho}, \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)}\right), c_i \overset{\text{def}}{=} \min\left(\bar{c}, \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)}\right)$。这样做有什么好处呢？每次价值函数 $V_\theta(x)$ 都是向 v_S 上更新的，收敛到的价值韩式是介于 V^{π} 和 V^{μ} 之间的某个价值函数，我们将此价值函数记为 $V^{\pi_{\bar{\rho}}}$，该价值函数对应的策略为：

$$\pi_{\bar{\rho}}(a \mid x) \overset{\text{def}}{=} \frac{\min(\bar{\rho}\mu(a \mid x), \pi(a \mid x))}{\sum_{b \in A} \min(\bar{\rho}\mu(b \mid x), \pi(b \mid x))} \tag{5}$$

而 $\bar{c}, \bar{\rho}$ 是避免 important sampling 采样发散，而加上的上界，控制每一步更新的速度。从公式 (5) 中可以看出，ρ 决定了收敛的不动点的位置。而 c_i 显然决定着更新的幅度，所有可以控制收敛的速度。详细内容请参考论文 IMPALA。这篇文章还是很不错的，IMPALA 是一个大规模强化学习训练的框架，具有较高的性能（high throughput）、较好的扩展性（scalability）和较高的效率（data-efficiency）。在大规模计算的框架下，采样和策略更新会有一些错位（不再是完全的 on-policy），在这种情况下，文章通过 V-trace 技术来完成地使用 off-policy 样本进行训练。V-trace 保证了在 off-policy 情况下的稳定的效果，加上对 replay buffer 的充分应用，使最终 actor 和 learner 可以异步进行，为数据吞吐量的提升提供了保障。

2.3 策略损失

令 π 为当前时刻需要优化的策略，μ 为采样策略，也就是获得样本的策略，比如之前经验生成的策略。此外，令 $\hat{A}_t^\pi$ 和 $\hat{A}_t^\mu$ 为策略 π 和策略 μ 在 t 时刻对优势函数的估计。我们可以用下面的策略损失函数来优化策略。

- **Policy Gradient**，其损失函数为：

$$\mathcal{L}_{\mathrm{PG}} = -\log \pi\left(a_t \mid s_t\right) \hat{A}_t^\pi \tag{6}$$

  当 $\hat{A}_t^\pi$ 是无偏估计，那么 $\nabla_\theta \mathcal{L}_{\mathrm{PG}}$，是假设从当前策略 π 中获得的经验的，对策略梯度的无偏估计。

- **V-trace**，其损失函数被定义为：

$$\mathcal{L}_{\mathrm{V-trace}}^{\bar{\rho}} = \mathrm{sg}\left(\rho_t\right) \mathcal{L}_{\mathrm{PG}} \tag{7}$$

  其中，$\rho_t = \min\left(\frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}, \bar{\rho}\right)$ 是被截断的重要性权值，sg 是 `stop_gradient` 算子，其中 $\bar{\rho}$ 是超参数 (`C15`)。如果 $\hat{\rho} = \infty$，不管采样策略 μ 怎么样，$\nabla_\theta \mathcal{L}_{\mathrm{V-trace}}^{\bar{\rho}}$ 都是对策略梯度的无偏估计。这个其实很好理解，无偏还是有偏都是由优势函数的估计所确定的。

- **Proximal Policy Optimization (PPO)**：

$$\mathcal{L}_{\mathrm{PPO}}^{\epsilon} = -\min\left[\frac{\pi\left(a_t \mid s_t\right)}{\mu\left(a_t \mid s_t\right)} \hat{A}_t^\pi, \mathrm{clip}\left(\frac{\pi\left(a_t \mid s_t\right)}{\mu\left(a_t \mid s_t\right)}, \frac{1}{1+\epsilon}, 1+\epsilon\right) \hat{A}_t^\pi\right] \tag{8}$$

  其中，ϵ 是超参数 (`C16`)，PPO 的损失会鼓励策略采取比平均水平更好的动作，而 clipping 则会限制策略在个例数据上因获得过大的回报，从而对策略进行的过大的改变，而导致训练的不稳定。PPO 就是 TRPO 的简化版，这个操作太简单了，而同样产生了很好的效果，是目前大规模运用的强化学习算法。而在原始版本的 PPO 中，用的并不是 $\frac{1}{1+\epsilon}$，而是 $1-\epsilon$，这两种用法都可以，但是作者觉得 $\frac{1}{1+\epsilon}$ 更加对 (小编怀疑，就是作者看前者比较爽而已)。

- **Advantage-Weighted Regression (AWR)**

$$\mathcal{L}_{\mathrm{AWR}}^{\beta, \omega_{\max}} = -\log \pi\left(a_t \mid s_t\right) \min\left(\exp\left(A_t^\mu / \beta\right), \omega_{\max}\right) \tag{9}$$

  当 $\omega_{\max} = \infty$ 时 (`C17`) 时，目标函数退化成 $-\log \pi\left(a_t \mid s_t\right) \exp\left(A_t^\mu / \beta\right)$。注意，AWR 是和之前的策略损失对比，AWR 中计算的是采样策略的优势函数 A_t^μ，并不是当前时刻的策略的优势函数 A_t^π，所以 AWR 更多的在 off-policy 的算法中提及。而文章中说到，AWR 是对 π 在 $\mathrm{KL}(\pi|\mu) < \epsilon$

条件下的近似优化，其中 ϵ 和 β 是相关的。我就非常纳闷，这个是怎么得到的，看了一下`https://zhuanlan.zhihu.com/p/99205485` 中的解读，才理解了，大家可以详细的阅读这篇 blog，大致做法就是使用拉格朗日乘子法，令导数等于零而求解出的。

- **On-Policy Maximum a Posteriori Policy Optimization (V-MPO)**，这个策略损失函数和 AWR 中的基本一致，只有少数几个不同点，

  1. 将 `exp` 函数换成 `softmax` 函数，并且没有对 $\omega_{\max}$ 的 clipping 操作。

  2. 对于每一个批次中只使用优势值为前 50% 的样本;

  3. 将 β 视为拉格朗日乘子，并且自适应的进行调整。

  4. 增加软约束项 $\mathrm{KL}(\mu|\pi) < \epsilon$，之前增加的是约束是 $\mathrm{KL}(\pi|\mu) < \epsilon$。关于这个约束的详细描述在 2.6中。但是在实验中，并没有直接把这个约束作为 Policy loss 的一部分，而是分别考虑 $\mathrm{KL}(\mu|\pi) < \epsilon$ 和 $\mathrm{KL}(\pi|\mu) < \epsilon$。看看那个效果好，(小编觉得这个改进纯属多此一举，哈哈哈哈!)。

- **Repeat Positive Advantages (RPA)**:

$$\mathcal{L}_{\mathrm{RPA}} = -\log \pi\left(a_t \mid s_t\right)\left[A_t > 0\right] \tag{10}$$

  其中，$[\cdot]$ 是判别函数，比如，$[P] = 1$ 如果 P 为真，其他情况 $[P] = 0$。这是，作者在本文中新提出的策略损失函数。事实上这个损失函数，是 AWR 和 V-MPO 的极限形式。实际上，当 $\beta \to 0$，则公式 (5) 等价于 $\omega_{\max}\mathcal{L}_{RPA}$。并且当 $\epsilon \to 0$ 时，V-MPO 将退化成 RPA。

2.4 Handling of Timesteps

控制时间步数最重要的参数是折扣因子 γ，这个懂一点强化学习的朋友都知道。实际上，还有一种控制时间步数的方法为 $frame\ skip$ (`C16`)，主要是出现在 Atari 游戏中。帧跳跃是什么意思呢? k 步帧跳跃，意味着每 k 帧运行一次，之间没有运行的帧使用最近的动作来作用于环境，并且将收到的奖励相加。在使用帧跳跃时，同时也要适当的调整折扣因子，应该将 γ 改为 γ^k，因为其中实际上有 k 步被隐藏了。

很多强化学习环境中都有步数限制，也就是环境运行到一定步数之后就强制终止。但是，剩余环境的步数并不包含在策略中，特别是在非马尔可夫环境中，可能会使学习更加困难。作者提出了两种处理方法，1. 将最后的转移看为终止状态的转移，最后时刻状态的价值函数目标等于最后时刻获得的奖励；2. 尊重事实，我们的确不知道接下来会发生什么。在这种情况下，我们将最后状态的优势设置为 0，并将其值函数目标设置为值函数当前的值，实际上就相当于直接丢掉后面的状态。这个选择为 (`C22`)。

2.5 优化器

优化器指的是梯度下降法用到的优化器。实验中探究了两种在强化学习中广泛使用的优化器：Adam 和 RMSProp。对于每一种优化器，探究了学习率 (`C24, C25`)，动量 (`C26, C27`)，为了增加数值稳定性增加的参数 ϵ(`C28, C29`)。其他的参数，全部和 TensorFlow 中的默认参数保持一致。最后，通过设置 `Learning rate decay (C31)` ，可以线性的调整学习率，它将最终学习率定义为初始学习率的一个分数 (比如，0 对应着衰减到 0，1 对应着不衰减)。

2.6 策略正则化

本文中考虑了三种不同的正则化模型 (32)。

- 没有正则化项，简单明了。
- 惩罚项，使用固定强度的正则化器 R(eg. SAC)，比如，在 loss 项后面增加固定的参数的 αR。
- 约束项，这个是强加一个软约束 (e.g TRPO) 来限制每一次更新的步长 $R < \epsilon$，以免步长过大而导致训练不稳定。令 α 为拉格朗日数乘项。那么根据拉格朗日算法，可以将其表示为拉格朗日算子 $\alpha \cdot \text{sg}(\epsilon - R)$，并且加在 loss 函数后面，其中 sg 表示 `stop_gradient` 算子。实际应用中，一般令 $\alpha = \exp\{c \cdot p\}$，其中，$c$ 是控制 α 的调整速度作者的所有实验中 $c = 10$，p 是训练出来的参数，初始值等于 0。在每一次梯度更新时，为了保证更新速度不会太快或者太慢，通常将 p 的值 clip 到 $[\log(10^{-6})/10, \log(10^{6})/10]$。

下面将介绍，通常使用在惩罚项和约束项中的正则化器，

- 熵正则化项：$\text{H}(\pi(\cdot|s))$，鼓励策略尝试不同的动作。
- $\text{KL}(\mu(\cdot \mid s) \| \pi(\cdot \mid s))$，用行为策略和目标策略间的 KL 散度来防止采取给定行动的概率过快下降。
- $\text{KL}(\pi(\cdot \mid s) \| \mu(\cdot \mid s))$，和前一个类似，不过是为了其概率增加得更快。
- $\text{KL}(\text{ref}(\cdot \mid s) \| \pi(\cdot \mid s))$，其中 ref 是固定的分布，在实验中我们都是使用 $\text{ref} = \mathcal{N}(0, 1)$。这样的正则化项鼓励策略尝试所有的动作。
- 解耦 $\text{KL}(\mu(\cdot \mid s) \| \pi(\cdot \mid s))$。对于一个高斯分布，我们可以将 $\text{KL}(\mu(\cdot \mid s) \| \pi(\cdot \mid s))$ 分解成一个依赖于分布平均值变化的项和另一个依赖于标准差变化的项：$\text{KL}(\mu(\cdot \mid s) \| \pi(\cdot \mid s)) = \text{KL}(\mu(\cdot \mid s) || \zeta(\cdot \mid s)) + \text{KL}(\zeta(\cdot \mid s) || \pi(\cdot \mid s))$，其中 $\zeta(\cdot \mid s)$ 是和 $\mu(\cdot \mid s)$ 有着相同的均值，和 $\pi(\cdot \mid s)$ 有着相同的标准差的高斯分布。所以，不需要直接对 $\text{KL}(\mu(\cdot \mid s) \| \pi(\cdot \mid s))$ 进行优化，可以对 $\text{KL}(\mu(\cdot \mid s) || \zeta(\cdot \mid s))$ 和 $\text{KL}(\zeta(\cdot \mid s) || \pi(\cdot \mid s))$ 分别进行优化，来调整侧重点。这样的技术在 V-MPO 中得到使用，而且 $\text{KL}(\mu(\cdot \mid s) || \zeta(\cdot \mid s))$ 的值比 $\text{KL}(\zeta(\cdot \mid s) || \pi(\cdot \mid s))$ 的值低一个数量级。同样，$\zeta(\cdot \mid s)$ 也可以和 $\mu(\cdot \mid s)$ 有着相同的标准差，和 $\pi(\cdot \mid s)$ 有着相同的均值的高斯分布。

虽然，可以将上述各项的任意线性组合加到损失中，但文章中在每个实验中只使用单一的正则化器。总的来说，所有这些正则化模式的组合和不同的超参数选择在下图中有详细描述。

Table 1: Choices pertaining to regularization.

Choice	Name
C32	Regularization type
C33	Regularizer (in case of penalty)
C34	Regularizer (in case of constraint)
C35	Threshold for KL($\mu\|\|\pi$)
C36	Threshold for KL($\pi\|\|\mu$)
C37	Threshold for KL(ref$\|\|\pi$)
C38	Threshold for mean in decoupled KL($\mu\|\|\pi$)
C39	Threshold for std in decoupled KL($\mu\|\|\pi$)
C40	Threshold for entropy $H(\pi)$
C41	Regularizer coefficient for KL($\mu\|\|\pi$)
C42	Regularizer coefficient for KL($\pi\|\|\mu$)
C43	Regularizer coefficient for KL(ref$\|\|\pi$)
C44	Regularizer coefficient for mean in decoupled KL($\mu\|\|\pi$)
C45	Regularizer coefficient for std in decoupled KL($\mu\|\|\pi$)
C46	Regularizer coefficient for entropy

图 1: 正则化项的选择

2.7 神经网络架构

文中使用多层感知机 (MLP) 来表示策略和价值函数。要么使用分离的单独的网络作为策略函数和价值函数，或者策略网络和价值网络共享特征提取层 (C47)。我们考虑了不同宽度的共享 MLP(C48)，策略 MLP(C49) 和价值 MLP(C50)，和不同深度的共享 MLP(C51)，策略 MLP(C52) 和价值 MLP(C53)。并且在使用共享 MLP 的情况下，我们进一步增加了超参数 Baseline cost (shared) (C54) 来记录 value loss 对目标函数整体梯度的贡献。这一点很重要，因为 MLP 的共享层的变化会同时影响与策略和价值函数相关的损失项。进一步将考虑不同的激活函数 (C55) 和神经网络初始器 (C56)。

2.8 动作分布参数化

策略输出的是一个分布，策略函数表示的是从状态到动作分布的映射，记为：$\pi(a|s)$。那么，我们需要找到一个方法来表示动作分布。实际应用中，我们将确定动作分布的参数形式。比如在绝大多数 RL 的连续控制问题中，我们用高斯分布来表示动作分布，然后从中采样即可得到样本动作。但是，在运行过程中，仍有几点需要选择。

- 动作的方差可否为网络输出的一部分，或者是独立的。在后一种情况下，标准差仍然是可学习的，但对于每个动作都是一样的。
- 高斯分布是用一个平均值和一个非负的标准差来参数化的。那么用什么函数将网络输出的负值转化为标准差 (C60)。我们考虑了指数化 (e.g. PPO, TRPO) 和 softplus 函数 ($\text{softplus}(x) = \log(e^x + 1)$)。
- 是否应该在标准差中增加一个小值，以避免非常小的值。
- 大多数连续控制环境都希望输出动作在范围 $[-1, 1]$，但是高斯分布并不能做到这一点。有两种常见的解决这个问题的方法，1. 最简单的方法是使用 clip 函数，将动作的值剪到环境要求的范围

中。2. 第二种方法则是使用 `tanh` 函数，将动作分布的值限制在区间内。这样我们就对目标的动作的分布 $p_\theta(x)$ 做了变量替换。值得一提的是在流模型的思路里经常使用这一套东西，建立简单分布与复杂分布之间的联系，通过对简单分布进行优化来间接优化复杂分布。使用了 tanh 函数后，$u = \tanh(x)$，其中 x 是从策略分布 $p_\theta(x)$ 中采样出的样本。那么现在问题来了，我们如何通过优化 $p_u(u)$ 来优化 $p_\theta(x)$? 需要建立二者之间的联系。

而实际上并不难，推导过程如下所示：

$$\begin{aligned} \int_u p_u(u)du &= \int_x p_\theta(x)dx \\ |p_u(u)du| &= |p_\theta(x)dx| \\ p_u(u) &= \frac{p_\theta(x)}{du/dx} \\ p_u(u) &= \frac{p_\theta(x)}{\tanh'(x)} \end{aligned} \tag{11}$$

所以有：$\log p_u(u) = \log p_\theta(x) - \log \tanh'(x)$。而其中对 $\tanh'(x)$ 与 θ 无关，所以，$\nabla_\theta \tanh'(x) = 0$。并且，$u = \tanh(x)$ 并不会影响对 KL 散度的计算，

$$\begin{aligned} \mathrm{KL}\left(U_1, U_2\right) =& \mathbb{E}_{u \leftarrow U_1} \log U_1(u) - \log U_2(u) \\ =& \mathbb{E}_{x \leftarrow X_1} \left(\log X_1(x) - \log \tanh'(x)\right) - \left(\log X_2(x) - \log \tanh'(x)\right) \\ =& \mathbb{E}_{x \leftarrow X_1} \log X_1(x) - \log X_2(x) \\ =& \mathrm{KL}\left(X_1 \| X_2\right) \end{aligned} \tag{12}$$

那么，唯一影响求解的地方就是熵正则化项，$H(U) = -\mathbb{E}_u \log p_u(u) = \mathbb{E}_x \left[-\log p_\theta(x) + \log \tanh'(x)\right]$。相比原算法，增加了一项 $\log \tanh'(x)$，这一项可以帮助惩罚采取极端动作，防止 `tanh` 过大和梯度损失的策略。

总结来看，我们利用如下方式对动作分布进行参数化，

$$T_u\left(\mathcal{N}\left(x_\mu, T_\rho\left(x_\rho + c_\rho\right) + \epsilon_\rho\right)\right) \tag{13}$$

其中，

1. x_μ 是策略网络的输出；
2. x_ρ 是策略网络输出的一部分，或者是单独的可学习参数 (每个动作维度一个)；
3. ϵ_ρ(C62) 是控制标准差超参数最小化的系数；
4. T_ρ 是保证方差大于 0 的函数 ($\mathbb{R} \to \mathbb{R}_{\leq 0}$)；
5. T_u 将动作映射到合理的区间 ($\mathbb{R} \to [-1, 1]$)；
6. c_ρ 是一个常数，用来控制初始化标准差，$c_\rho = T_\rho^{-1}\left(i_\rho - \epsilon_\rho\right)$，其中 i_ρ 是初始化标准差 (C61)。

2.9 数据归一化和 CLIPPPING

这两个内容通常在 RL 的公开发表物中没有过多的讨论，所以大家经常使用各种各样的实现归一化的方法，通常可以总结为以下几点：

- 对于观测状态的归一化 (C64)。基于之前所有观测到的状态，我们保留经验数据的均值 o_μ 和标准差 o_ρ。并通过减去经验均值和除以最大值 $(o_\rho, 10^{-6})$ 来归一化观察结果。这将导致所有的神经网络输入的均值和标准差都接近于 0。此外，我们可以选择性地将归一化观测状态通过超参数 $o_{\max}$ 剪到 $[-o_{\max}, o_{\max}]$ 以内 (C65)。
- 对于观测状态的归一化 (C64)。同样，记录目标价值函数的经验均值 v_μ 和标准差 v_ρ。而价值函数网络输出的结果 $V_{\text{out}} = (\hat{V} - v_\mu / \max(v_\rho, 10^{-6}))$。根据，神经网络的输出值，我们就可以得到最终的预测值为：$\hat{V} = v_\mu + V_{\text{out}} \max(v_\rho, 10^{-6})$。
- 对于 minbatch 的归一化 C67，我们通过将每个小批量的优函数势减去它们的平均值，然后除以它们的标准差，来得到保策略损失。
- 梯度 clipping，在将梯度反馈给优化器之前重新调整它，这样它的 L2 范数就不会超过期望的阈值。

3 Thesis statement

本文的主要贡献是，深入研究基础参数对强化学习最终性能的影响。1. 对于每种深度 AC 强化学习算法，研究其中超过 50 种参数的选择；2. 我们进行了大规模的实验（训练了超过 250000 个 agent），涵盖了训练过程的不同方面；3. 对于实验结果进行分析，为训练深度 AC 算法中的参数设置提供实用的见解和建议。

随着大规模的实验，作者也发现了一些很有意思的事，或许之前在其他实验中忽略的。比如，策略的初始化，非常影响算法性能。并且，将初始动作分布，均值设置为 0，并且方差较小的情况下，可以提高训练速度。

3.1 研究设置

本文中，考虑的是 on-policy deep actor-critic reinforcement learning for continuous control。那么，对于一个算法，如何决定需要研究那些参数呢？作者用了以下三种方法：

1. 重新研究了之前的工作和流行的代码，比如 OpenAI-baseline，确定了一系列常见的选择，比如：不用的损失函数；网络结构的选择，比如初始化方法；启发式的技巧，比如：梯度截断，和其对应的参数。
2. 先制定一个简单的 AC 算法，然后根据需求，不断在其上面增加新的参数和 tricks。
3. 文章中验证，当所有参数选择和 OpenAI-baseline 中 PPO 算法的选择一致时，可以获得和 OpenAI-baseline 类似的效果。这说明算法的主体是没有问题的，而且可以通过这样的方式来单一变量，研究问题。

3.2 Policy Loss

文章中研究了不同的策略误差函数，vanilla Policy Gradient (PG), V-trace, PPO, AWR, V-MPO, RPA 即为限定情况下的 AWR ($\beta \to 0$) 和 $V - MPO(\epsilon_n \to 0)$ 的结合体。实验的目的是为了帮助大家

更好的理解策略损失函数在 AC 算法中的作用。我们的目标并不是非要找出哪个损失函数一定比另一个损失函数要好，因为有的损失函数就是在某些特定环境下用的（比如，V-trace 是大规模运算中使用的，其本来就是为了拟合一个近似的策略。）

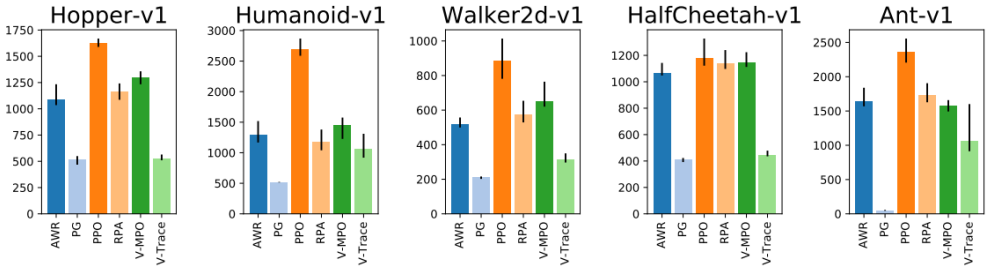

Figure 1: Comparison of different policy losses (○14).

图 2: 矩阵与列向量的乘法

从此图中可以看出，我们观察到 PPO 在五个环境上都实现了最好的效果。并且，作者发现 PG 和 V-trace 几乎在所有的任务下都是表现最差的。这是因为它们不太适合于处理这样在一次迭代中数据变成 off-Policy 的问题。在 AWR 中，β 参数比 $\omega_{\max}$ 参数更重要。对于 PPO 来说，截断阈值 ϵ，作者发现通常来说，$\epsilon = 0.2$ 或者 $\epsilon = 0.3$ 时，在绝大多数环境中可以取得较好的效果。但是对于某些环境，$\epsilon = 0.1$ 或者 $\epsilon = 0.5$ 可以获得更好的效果。

推荐：使用 PPO 损失函数，开始令截断阈值 $\epsilon = 0.25$，然后可以尝试更大或者更小的值。

3.3 网络框架

文章中研究了策略和价值网络对实验效果的影响，考虑了网络的大小，结构，激活函数，权重初始化方法。

作者发现分开策略和动作网络，不共享同一套特征提取层可以获得更好的效果。并且作者在实验中发现，策略网络的宽度和环境的复杂度有关，太大或者太小都会对实验结果产生较大的影响，而价值网络的宽度对实验结果没有太大的影响。**在一些环境中，最好令价值网络的宽度大于策略网络**。策略网络和价值网络中，令隐藏层数目为 2 都可以获得较好的效果，并且 `tanh` 激活函数比 `relu` 函数更好。

比较有趣的是，作者发现策略网络的初始化方法对训练性能有较大的影响。较好的方法，是令策略网络输出的动作分布均值为 0，方差比较小（作者发现对于大多数环境，初始方差敏感，最好将动作的分布设置为 0.5）。至于其他参数的选择则没有那么的重要，比如作者实验中没有发现策略网络对于每个状态方差不一样和所有状态使用同一个方差有什么区别。

推荐：将策略网络的最后一层。使用 `Softplus` 函数将网络的输出转换为动作的标准差，并输入加上非负的偏置，来减小初始化动作的方差，并且尽可能的调整此偏置。如果网络的深度不是非常深，尽可能所有函数都用 `tanh`。分开策略和动作网络，并且使用较宽的策略价值网络，而调节策略网络的宽度，它可能需要比价值网络的宽度更小。

3.4 归一化和截断

本文研究了不同的归一化技术，包括对观测变量的归一化，价值函数的归一化，每一个 minibatch 的优势函数的归一化；还有对梯度和观察变量的截断。

作者发现对输入状态的归一化对实验性能有较大的提升。比较惊奇的是，价值函数的归一化也会较大的影响实验性能，但是影响不一定是正面的。作者怀疑价值函数的归一化会影响价值函数拟合的速度，从而影响其性能。最终，作者发现对梯度和观测变量的截断并不会对最终性能产生多大的影响。

推荐：可以经常性的使用对观测变量的归一化，而尝试对价值函数的归一化，万一可以取得较好的效果。而对梯度的截断可能并没有什么用，但是也可试试。

3.5 优势估计

我们比较了较为常见的优势估计方法：N-step，GAE 和 V-trace。并且尝试在 Value 的学习中用 Huber loss 来代替 MSE。

作者发现GAE 和 V-trace 可以获得比 N-step 更好的性能。这指示联系多个时间步有利于价值估计。并且，作者没有发现 GAE 和 V-trace 之间有什么较大的区别。并且，作者发现当 $\lambda = 0.9$ 时效果较好。并且 PPO 那样的截断方法会降低性能。同样，作者发现 Huber loss 在所有环境中表现都比 MSE 差。

推荐：优势估计使用 $\lambda = 0.9$ 的 GAE。但是，即不使用 Huber loss，也不使用 PPO 风格的价值函数截断。

3.6 训练设置

我们探究了有关于数据收集和 minibatch 处理的选择，包括并行环境数；每次迭代中计算的转移（transitions）数；传送数据的次数；minibatch 大小；如何将数据分割成 minibatch。

对于最后一个选择，如何将如何将数据分割成 minibatch? 除了标准的做法之外，作者还提出了一种对原来的 PPO 算法的小修改。原始的 PPO 算法中，在每一个策略迭代步中，将片段拆成独立的 transition，并将其随机的分配给不同的 minibatch。所以，当数据的时间结构被破坏时，这样的就基本没有用了。所以，在每一轮迭代开始之前，需要计算优势函数值，然后用于 minibatch 的策略和价值函数的更新。这可以使每个 minibatch 中的样本多样性更高，但是会使用较老的优势估计。为了解决这个问题，作者提出在每次数据传统开始时就计算优势，而不是在每一次迭代的时候计算。

显然，多次重复采样提高样本的复杂度是非常重要的。通常，因为模型比较简单，计算可能比较简单，但是如果并行环境数量上升，在某些环境上性能会大大的下降。而强化学习很多时候计算性能被限制，是因为 CPU 采样能力不够。如果 CPU 充足的话，在更多的环境中进行训练可以使训练更快。而且，增加 batch 大小，并不会损害样本的复杂性。另一方面说，每次迭代中 transitions 的数量对最终性能的影响非常大。最后，作者比较了不同的 minibatch 处理方法，表明之前的优势值确实会影响算法的性能，每次测试开始的时候，重新计算优势，可以缓解这个问题，并且取得较好的效果。简单的说，就是提前计算优势函数，因为很多优势函数的计算都和未来 n 个时刻的状态有关。（具体的理解，小伙伴就得看代码了。）

推荐：多次使用之前的采样，可以提高性能。将每个 transition 分配给不同的 minibatch 时，将 transition 随机打乱，并在每次数据传递时计算优势函数。在 CPU 资源充足的情况下，尽可能

的多使用并行环境，并提高批处理的大小（这两种方法都可以降低样本的复杂度）。如果可能，可以适当的调整每次迭代中的 transition 数目。

3.7 时间步处理

本文中研究了关于时间步的处理方法，有折扣因子，帧跳跃，如何处理由于时间步长带来的片段终止。

作者的实验发现，折扣因子 γ 对实验性能有很大的影响，当 $\gamma = 0.99$ 时，所有环境中都能表现较好。帧跳跃只在 2/5 个环境中起作用。实验中发现由于时间步长限制而扔掉的片段，并不会对最终性能产生影响，这可能是由于我们在所有环境中，时间步设置较大的原因。

推荐：折扣因子 γ 对最终的实验性能影响非常大，建议从 $\gamma = 0.99$ 时开始。可以尝试帧跳跃技术。当限制时间步较大的时候，不需要考虑处理片段舍弃的部分。

3.8 优化器

文章中主要研究了两种优化器，Adam 和 RMSprop。

优化器的选择带来的性能差异似乎比较小，没有一个优化器可以在所有环境上始终优于另一个优化器。不出意外，学习率对性能的影响非常大。Adam 优化器中设置的 0.0003 在绝大多数任务上效果较好。Adam 优化器中动量会对实验效果有一定的影响，而在 RMSprop 中，动量基本没影响。对学习率线性的衰减到 0，在 4/5 个环境中都取得了较好的性能。

推荐：使用 Adam 优化器，并且令动量 $\beta_1 = 0.9$，调整学习率为 0.0003（默认）。对学习率线性的衰减，可能会适当的提高性能，但是并没有那么重要。

3.9 正则化

本文中研究了不同策略的正则化方法，包括直接作为惩罚项和软约束的方法。本文中考虑了，熵正则化项，当前策略和标准正态分布 $\mathcal{N}(0, 1)$ 之间的 KL 散度，当前策略和行为策略之间的 KL 散度，还有对偶形式的 KL 散度，这些我们都在 2.6 节中有详细的描述。

作者在实验中，并没有发现任何证据表明，所研究的正则化项对环境得到的 agent 有显著的提高。除了 HalfCheeta 环境，所有的约束都有帮助（特别是熵正则化项）。然而，性能的提升很大程度上与我们软约束中设定的 ϵ 值无关，这意味着约束的效果是初始的高强度惩罚引起的，期望的约束没什么太大的效果。令作者出乎意料的是，除了 HalfCheeta 环境，正则项几乎没有对性能产生效果。我们推测了正则项在实验中不太起作用的原因：1. PPO 策略损失中，已经对置信区间进行了截断，这使得 KL 惩罚或约束变得有点多余；2. 我们之前详细的分析过初始动作分布参数的设置，这实际上保证了好的探索性，这使得熵和软约束变得有些多余了。

4 Conclusion and Discussion

此篇文章的总体阅读难度不大，此篇文章带给我的帮助还挺大的。主要是两个方面，1. 我对 On-Policy Deep Actor Critic RL algorithms 的发展进程有了更深一步的思考。而且，我也重新温习了一下 GAE，V-trace，PPO，AWR 等里程碑式的研究成果，巩固了强化学习的基础知识；**2. 了解了各参数对强化学习的影响，这对我的工程能力一定有着很大的帮助。而且，意外之喜，发现了通过控制**

初始动作分布的方差可以达到控制探索开发的作用，这也算个非常重要的 trick 了。而且，很多文章中加千奇百样的正则化，可能没有效果，如果把初始动作方差调大一点，或者不用 PPO 那样截断的方法，效果提升会更明显。当然，工程能力的提高，需要我们在之后的实验中，反复思考，阅读此文章，工程能力才会真正的突飞猛进。

总之，无论对于强化学习小白还是有一定基础的学生，此文章都是可读的。