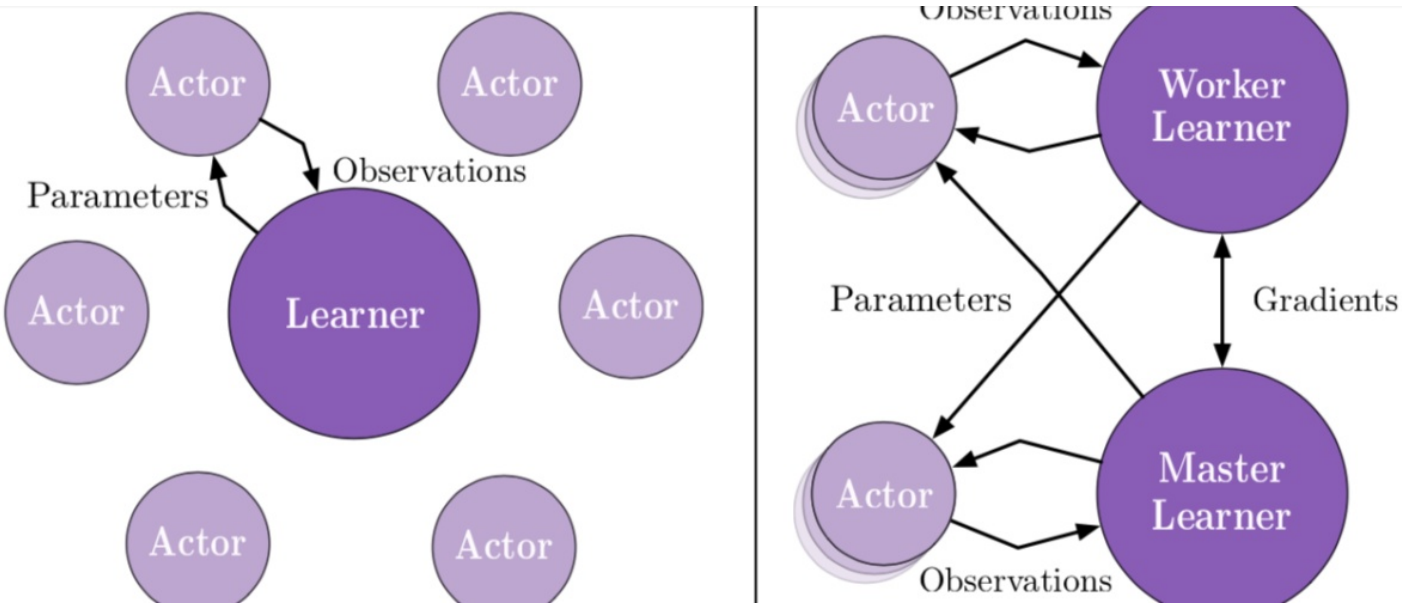


AlphaStar之IMPALA

starimpact
计算视觉

前两天DeepMind终于爆出了在星际2上的最新进展，我是看着羡慕（详见：AlphaStar: Mastering the Real-Time Strategy Game StarCraft II | DeepMind）。里面提到了应用到的各种技术：Transformer, LSTM, pointer network, novel off-policy actor-critic rl, self-imitation learning, policy distillation等等。前面几个网络结构都可以搜到中文，但是后面若干算法就比较难找了。在这里，我先说一下novel off-policy actor-critic rl，即IMPALA。

强化学习需要大量在线得到的样本。这就涉及到其中一个问题：产生的样本怎么处理？如果是on-policy的RL算法，样本用过一次就扔掉，十分浪费（千辛万苦采到的样本，就用一次？太浪费了！哪个监督学习里的样本不用个百八十次的~）。如果off-policy的RL算法就可以做到以前的样本反复利用（有RL基础的同学可以马上联想到Deep Q-Learning的Replay Buffer）。

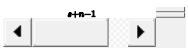
涉及到的第二个问题：如何高效的产生样本？当然是并行或分布式啦！Actor和Learner各干各的，谁都不要等谁。Actor们（不只一个）不断地将采到的样本放到Replay Buffer里，并周期性地从Learner那拿到最新的参数。Learner不断地从Replay Buffer里拿样本过来训练自己的参数。要达到这样的目的，也只有off-policy的方法可以这么干了。而如果是on-policy的方法就要遵循：Actor放样本到buffer->Learner取样本训练参数->Actor取最新的参数->Actor执行动作收集样本->Actor放样本到buffer，这个过程按顺序来的，无法并行。

涉及到的第三个问题：off-policy的方法是可以让样本收集和学习变成并行，还可以利用老样本，但是那些比较老的样本就这么直接拿来更新当前的参数，也会产生利用效率不高的问题（可以理解成并不能有效提升当前的Agent水平）。

好啦，谁能解决上面三个问题？IMPALA的V-trace！

IMPALA这么怪的名字？**IMP**ortance weighted **A**ctor-**L**earner **A**rchitecture！大家都以为是六个单词的首字母~现在的简写真是没有底线了^_^。

先上一下V-trace target本尊：


--------(1)

非常复杂，看不懂，有没有？看看里面一些变量的定义是什么：

$$\delta_t V = \rho_t(r_t + \gamma V_{s_{t+1}} - V_{s_t})$$

括号里的表达式是Temporal Difference，然而外面的 ρ_t 是什么含义了？

$$\rho_t = min(\bar{\rho}, \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)})$$

还有其它变量的定义：

$$c_t = min(\bar{c}, \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)})$$

$$\bar{\rho} \geq \bar{c}$$

看到 ρ_t 和 c_t 的表达式，是不是联想起什么了？这就是Importance **S**ampling。从策略 μ 中采样，更新当前的策略 π 。只不过加上了最大值的限制，不能超过 $\bar{\rho}$ 和 $\bar{c}$ ，也就是说进行了截断处理，一般情况下这两个值可以设置成1啦。

要进一步理解 ρ_t 的意义可以看下面的表达式（为什么会出现这样的表达式，这个要看原论文）：

$$\pi_{\bar{\rho}}(a|x) = \frac{min(\bar{\rho}\mu(a|x), \pi(a|x))}{\sum_{b\in a} min(\bar{\rho}\mu(b|x), \pi(b|x))}$$

$\pi_{\bar{\rho}}$ 是一个介于 π 和 μ 的中间态的策略。为什么这么定义了？如果当 $\bar{\rho}=\infty$ ，那么 $\pi_{\bar{\rho}}$ 就会变成策略 π ，如果 $\bar{\rho} \Rightarrow 0$ （是接近于0，不是等于），那么 $\pi_{\bar{\rho}}$ 就会变成策略 μ 。（所以 $\bar{\rho}$ 越大，那么off-policy学习的bias就越小，相应的variance就越大。）

c_i 的乘积表示 $\delta_t V$ 在时刻 t 影响前面时刻 s 的值函数更新的强弱程度。π 和 μ 差距越大，那么off-poliocy越明显，那么这个乘积的variance就越大。这里用了截断方法来控制这种variance。

$\bar{\rho}$ 影响的是要收敛到什么样的值函数，而 $\bar{c}$ 影响的是收敛到这个值函数的速度。（突然间出来了值函数，有什么样的策略就有什么样的值函数，二者是对应的，就这么理解吧。）

要注意的是， v_s 在on-policy的情况下（即 $\pi=\mu$ ），并让 $c_i=1$ ，且 $\rho_t=1$ ，就会变成下式：

$$v_s = V(x_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s}(r_t + \gamma V(x_{t+1}) - V(x_t)) = \sum_{t=s}^{s+n-1} \gamma^{t-s} r_t + \gamma^n V(x_{s+n}) \quad \text{--------(2)}$$

这个式子是Bellman Target (类似于TD Target)。也就是说式(1)在on-policy的特殊情况下就变成了式(2)。于是，同样的算法可以把off和on两种policy通吃了。

V-trace targets 可以用迭代的方式进行计算（让人联想到back view TD(λ)，实际上在某些条件下确实也可以将它转化到TD(λ)，具体看论文）：

$$v_s = V(x_s) + \delta_s V + \gamma c_s (v_{s+1} - V(x_{s+1}))$$

----------我是可爱的分割线----------

其实重点基本都讲完了，下面讲怎么用 v_s 。这实际上就是actor-crtitic的标准训练方法了。

首先，更新critic时用的梯度：

$$(v_s - V_\theta(x_s))\Delta_\theta V_\theta(x_s)$$

然后，更新actor时用的梯度：

$$\rho_s \Delta_\omega log\pi_\omega(a_s|x_s)(r_s + \gamma v_{s+1} - V_\theta(x_s))$$

为了防止过早的收敛，为actor的梯度加一个policy entropy(熵)的惩罚：

$$-\Delta_\omega \sum_a \pi_\omega(a|x_s) log\pi_\omega(a|x_s)$$

----------我是可爱的分割线----------

下面简单看看实验部分。先看看网络是啥样子（做了大小网络实验）：

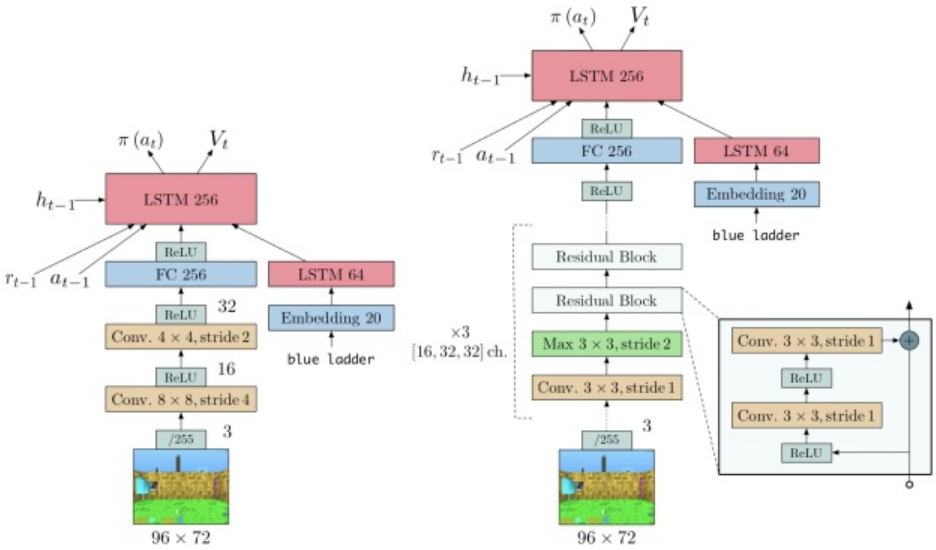


Figure 3. Model Architectures. **Left:** Small architecture, 2 convolutional layers and 1.2 million parameters. **Right:** Large architecture, 15 convolutional layers and 1.6 million parameters.

再看看Replay Buffer对比实验：

	Task 1	Task 2	Task 3	Task 4	Task 5
Without Replay					
V-trace	46.8	32.9	**31.3**	**229.2**	**43.8**
1-Step	**51.8**	**35.9**	25.4	215.8	43.7
ε-correction	44.2	27.3	4.3	107.7	41.5
No-correction	40.3	29.1	5.0	94.9	16.1
With Replay					
V-trace	47.1	**35.8**	**34.5**	**250.8**	**46.9**
1-Step	**54.7**	34.4	26.4	204.8	41.6
ε-correction	30.4	30.2	3.9	101.5	37.6
No-correction	35.0	21.1	2.8	85.0	11.2

Tasks: `rooms_watermaze`, `rooms_keys_doors_puzzle`, `lasertag_three_opponents_small`, `explore_goal_locations_small`, `seekavoid_arena_01`

在应用完整V-trace的情况下Replay Buffer的提升作用明显。如果去掉了off-policy的correction部分，那么就会产生负作用，可以看上面的No-correction行。

最后看看训练速度，数据吞吐率：

Architecture	CPUs	GPUs[1]	FPS[2]	
Single-Machine			Task 1	Task 2
A3C 32 workers	64	0	6.5K	9K
Batched A2C (sync step)	48	0	9K	5K
Batched A2C (sync step)	48	1	13K	5.5K
Batched A2C (sync traj.)	48	0	16K	17.5K
Batched A2C (dyn. batch)	48	1	16K	13K
IMPALA 48 actors	48	0	17K	20.5K
IMPALA (dyn. batch) 48 actors[3]	48	1	21K	24K
Distributed				
A3C	200	0	46K	50K
IMPALA	150	1	80K	
IMPALA (optimised)	375	1	200K	
IMPALA (optimised) batch 128	500	1	250K	

[1] Nvidia P100 [2] In frames/sec (4 times the agent steps due to action repeat). [3] Limited by amount of rendering possible on a single machine.

Table 1. Throughput on `seekavoid_arena_01` (task 1) and `rooms_keys_doors_puzzle` (task 2) with the shallow model in Figure 3. The latter has variable length episodes and slow restarts. Batched A2C and IMPALA use batch size 32 if not otherwise mentioned.

所以，CPU搞多点去跑actors，GPU搞一个用来跑learner就行啦~

论文中还做了一些与A3C，A2C效果的对比，基本上是对超参更不敏感，在大多数任务中能达到更好的效果，当然吞吐量也是好得很。

----------我是可爱的分割线----------

对于星际这种搜索空间超大的任务来说，选择off-policy的方法是必然的。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。

在IMPALA中，我们同样看到了policy entropy的身影。它似乎已经变成了RL可以稳定训练的一个必备组成。在soft q learning和soft actor critic等一些任务中，我们都可以看到它的优秀表现。

和同样是off-policy的sac(bair出品)来比，谁又更有优势了？sac可是能直接在真机上提升采样效率的方法。

编辑于 2019-02-17

强化学习 (Reinforcement Learning)

文章被以下专栏收录