Distributed Proximal Policy Optimization
(by Google Deep Mind)

Distributed: 与A3C策略一致，采用并行的分布式训练。以降低参数之间的相关性，提高model的收敛性和稳定性。

Proximal Policy: 采用加入KS惩罚项的方法，限制了new policy的更新幅度 ⇒ 降低了model对 step size/lr 敏感度，提高了鲁棒性。

PPO: (by OpenAI) 近端策略优化

特点: 现有最顶尖的RL算法之一 ⇒

对比ACER. TRPO
① 在众多问题上表现优异 (more general)
② 算法简单，易于实现、调试 (simpler to implement)

对比A3C. DDPG
③ 采样效率更高 (better sample complexity)
④ 鲁棒性好 (Robustness)

前身: TRPO (Trust Region Policy Optimization)

同级算法: ACER (Actor Critic with Experience Replay)
缺: 与PPO表现相近，但实现十分复杂，且范化性差(与参数共享、dropout不兼容)

why PPO? ⇒

Policy Gradient类的算法 (Simple PG、AC、A3C、DDPG...)
(i) 对超参数敏感，鲁棒性差。
① learning rate
② step size. ⇒ ①、②过大: Policy跳动太大，难以收敛
①、②太小: 学习速度缓慢。

(ii) 采样效率低下，学习简单任务即需要百万级iteration

Algorithm 1 Proximal Policy Optimization (adapted from [8])

for $i \in \{1, \cdots, N\}$ do — empirical collection

Run policy π_θ for T timesteps, collecting $\{s_t, a_t, r_t\}$

Estimate advantages $\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t)$ ⇒ Advantage = (V_- − V) (类似于 td-error)

⇓ discount reward to estimate V_-

$\pi_{\text{old}} \leftarrow \pi_\theta$

for $j \in \{1, \cdots, M\}$ do — actor update steps

Update Actor ⇐

$J_{PPO}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \hat{A}_t - \lambda \mathrm{KL}[\pi_{old}|\pi_\theta]$ ⇒ KL penalty

⇒ 在迭代过程中保持不变，此为 PPO 核心思想

Update θ by a gradient method w.r.t. $J_{PPO}(\theta)$

end for

→ KL regularization coefficient

for $j \in \{1, \cdots, B\}$ do — critic update steps

Update critic ⇐

$L_{BL}(\phi) = -\sum_{t=1}^{T} \left(\sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t)\right)^2$ ⇒ c-loss — td-error

Update ϕ by a gradient method w.r.t. $L_{BL}(\phi)$

end for

if $\mathrm{KL}[\pi_{old}|\pi_\theta] > \beta_{high}\mathrm{KL}_{target}$ then

$\lambda \leftarrow \alpha\lambda$

else if $\mathrm{KL}[\pi_{old}|\pi_\theta] < \beta_{low}\mathrm{KL}_{target}$ then — 权衡因子

$\lambda \leftarrow \lambda/\alpha$ — 调整因子 (scaling term)

end if

⇒ Adaptive KL penalty (因为 KL penalty 自适应能力差)

end for

Update Actor

KL penalty:

$$L^{KL}(\theta) = \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}\hat{A}_t - \lambda \cdot KL[\pi_{old}|\pi_\theta]\right]$$

Expect over T time steps. — in time step t

Clipped surrogate objective:

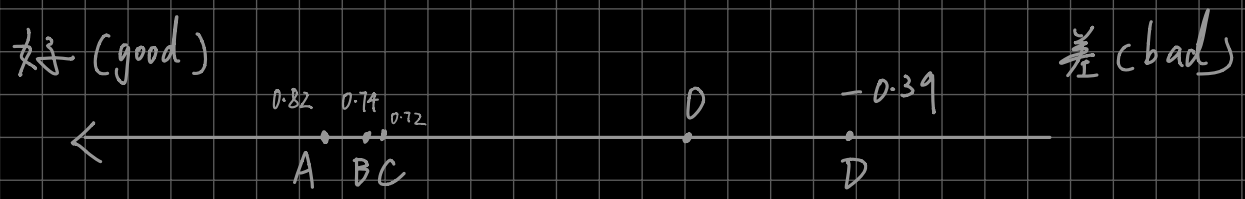
$$L^{CLIP}(\theta) = \hat{E}_t\left[\min\left(r_t(\theta)\cdot\hat{A}_t,\ \mathrm{clip}\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon\right)\cdot\hat{A}_t\right)\right]$$

$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}$ — r_t 区间 — 超参数，0.1~0.2

△ CLIP method is better than KL method，因为①它与 GD 兼容 ②简化了实现 ⇒ 移除了 KL 惩罚项（和 adaptive KL 部分）

Experiment by OpenAI:

Continuous control benchmark:



A: Clipping
B: Adaptive KL Penalty
C: Fixed KL Penalty
D: No clipping or penalty

Atari Benchmark:

average episode reward over

	作为baseline，为1	ACER	PPO
all training:	1	18	30
last 100 episode:	1	28	19

PPO在全局上性能要优于ACER

ACER在训练最后能达到更好的效果

Distributed PPO

⇓

data collection & gradient calculation are distributed over workers

Important points:

- Pull: Averaging gradients from workers
- Push: Push parameters synchronously
- Normalization:

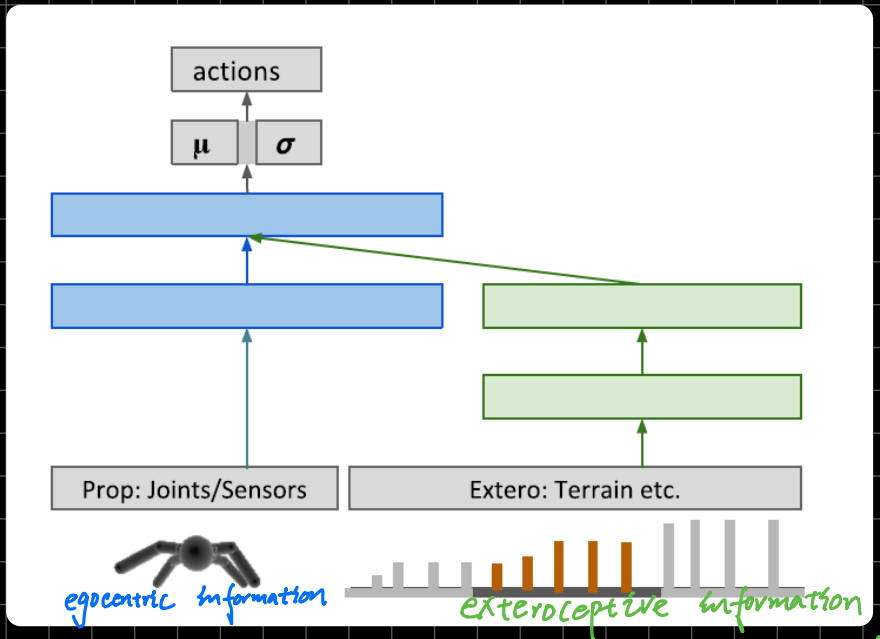
observation: $s_t \Rightarrow \frac{s_t - w}{\sigma}$

reward: $r_t \Rightarrow \frac{r_t}{\sigma}$

advantage: $A_t \Rightarrow$ per batch norm

} T time steps

Network Architecture:



(environment & task related)

Aim to achieve a seperation of basic locomotion skills and terrain perception & navigation

Algorithm 2 Distributed Proximal Policy Optimization (chief)

```
for i ∈ {1, ··· , N} do                          (Episode number)
    for j ∈ {1, ··· , M} do                      (global update steps; number of workers)
        Wait until at least W − D gradients wrt. θ are available   (threshold)
        average gradients and update global θ
    end for                                      (update actor)
    for j ∈ {1, ··· , B} do
        Wait until at least W − D gradients wrt. φ are available
        average gradients and update global φ
    end for                                      (update critic)
end for
```

Algorithm 3 Distributed Proximal Policy Optimization (worker)

for $i \in \{1, \cdots, N\}$ **do** — number of data points per worker

  for $w \in \{1, \cdots, T/K\}$ **do** — number of time steps (因为有多个 worker, 因此一次 empirical collection 降为 K 即可)

    Run policy π_θ for K timesteps, collecting $\{s_t, a_t, r_t\}$ for $t \in \{(i-1)K, \ldots, iK-1\}$

    Estimate return $\hat{R}_t = \sum_{t=(i-1)K}^{iK-1} \gamma^{t-(i-1)K} r_t + \gamma^K V_\phi(s_{iK})$

    Estimate advantages $\hat{A}_t = \hat{R}_t - V_\phi(s_t)$

    Store partial trajectory information

  end for

  $\pi_{\text{old}} \leftarrow \pi_\theta$

  for $m \in \{1, \cdots, M\}$ **do** — 与 PPO 一样, 在更新时使用所有的 data point 计算 $J(\theta)$

    $J_{PPO}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \hat{A}_t - \lambda \mathrm{KL}[\pi_{old}|\pi_\theta] - \xi \max(0, \mathrm{KL}[\pi_{old}|\pi_\theta] - 2\mathrm{KL}_{target})^2$ — KL penalty item; threshold; additional penalty term

    if $\mathrm{KL}[\pi_{old}|\pi_\theta > 4\mathrm{KL}_{target}$ **then** — Early Stopping

      break and continue with next outer iteration $i+1$

    end if

    Compute $\nabla_\theta J_{PPO}$

    send gradient wrt. to θ to chief

    wait until gradient accepted or dropped; update parameters — improve stability

  end for

  for $b \in \{1, \cdots, B\}$ **do**

    $L_{BL}(\phi) = -\sum_{t=1}^{T} (\hat{R}_t - V_\phi(s_t))^2$ ⇒ critic loss

    Compute $\nabla_\phi L_{BL}$

    send gradient wrt. to ϕ to chief

    wait until gradient accepted or dropped; update parameters — ⇒ wait for global update finished, then update local parameters.

  end for

  if $\mathrm{KL}[\pi_{old}|\pi_\theta] > \beta_{high}\mathrm{KL}_{target}$ **then**

    $\lambda \leftarrow \tilde{\alpha}\lambda$

  else if $\mathrm{KL}[\pi_{old}|\pi_\theta] < \beta_{low}\mathrm{KL}_{target}$ **then**

    $\lambda \leftarrow \lambda/\tilde{\alpha}$ — λ is shared among workers. (But λ update is based on local data)

  end if

end for

Implementation

Hyper Parameters:

N: number of episode T: timesteps

γ: decay rate λ: KL factor

$\begin{cases} M: \text{action update steps} \\ B: \text{critic update steps} \end{cases}$

$\begin{cases} B_{low}, B_{high} \\ \alpha \end{cases}$

for $i \in \{1, \cdots N\}$ do $\Rightarrow$ training episode

Empirical Collection $\Rightarrow$ For T timesteps, collect $\{s_t, a_t, r_t\}$ from interation with RL environment

Calculate $\hat{A}_t$ for $t \in \{1, \cdots T\}$

$$\hat{A}_t = \sum_{t' > t} \gamma^{t'-t} \cdot r_t - \underbrace{V_\phi(s_t)}_{\Rightarrow \text{critic network}}$$

$\pi_{old} \leftarrow \pi_\theta$ $\Rightarrow$ update π_{old}

for $j \in \{1, \cdots M\}$ do $\Rightarrow$ action update steps

$$J(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \cdot \hat{A}_t - \lambda \cdot KL(\pi_\theta | \pi_{old})$$

Calculate $\nabla_\theta J(\theta)$ and update $J(\theta)$

for $j \in \{1, \cdots B\}$ do $\Rightarrow$ critic update steps

$$L(\phi) = -\sum_{i=1}^{T} (\hat{A}_t - V_\phi(s_t))^2$$

Calculate $\nabla_\theta L(\phi)$ and update $L(\phi)$

If $KL(\pi_\theta | \pi_{old}) > \beta_{high} \cdot KL_{target}$:

$$\lambda = \lambda \cdot \alpha$$

Else if $KL(\pi_\theta | \pi_{old}) < \beta_{low} \cdot KL_{target}$:

$$\lambda = \lambda / \alpha$$

可能原因：

① env. seed 无法复现，每次会不同 ✓
但达到的 reward 上限类似

② env. seed 无设置 ✗

③ KL-divergence 不满足交换律：✗

KL-divergence (old-p^i, p^i) ✓

KL-divergence (p^i, old-p^i) ✗

④ Kernel-initializer 与 Constant initializer ✗
影响不大

⑤ update 中 adv 是否进行了 norm 操作。 ✓