1707.02286 - Emergence of Locomotion Behaviours in Rich Environments

- **Author: Yunqiu Xu**
- Similar work from OpenAI: 1707.06347 - Proximal Policy Optimization Algorithms
- PPO have some of the benefits of TRPO, but much simpler to implement, more general, and have better sample complexity (empirically)
- Other reference:
    - https://zhuanlan.zhihu.com/p/30138538
    - https://www.leiphone.com/news/201707/A2TWlxblaBFl8aod.html
    - https://morvanzhou.github.io/tutorials/machine-learning/reinforcement-learning/6-4-DPPO/

- Previous methods:
    - Q-learning with function approximation: fails on many simple problems and is poorly understood
    - Vanilla policy gradient: poor data effiency and robustness
        - Sparse reward: step size is prone to be too large
        - Get target by sampling: maybe the target is not the best(local optima)
    - TRPO:
        - Constraint: the KL divergence of old policy and new policy should not exceed threshold
        - Complicated, not compatible with architectures that include noise or parameter sharing
- PPO from OpenAI:
    - Improve the current state of affairs by introducing an algorithm that attains the data efficiency and reliable performance of TRPO, while using only first-order optimization
    - PPO put constraint into loss function:
        - If new policy is in wrong opt direction : still need to optimize

- If new policy is in right opt direction but with too large learning rate (much different from old policy) : stop optimizing
- If new value is worse than old (difference from target): still need to optimize
- If new value is better than old but go too far (much different from old): stop optimizing

- PPO from DeepMind: Distributed PPO

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

---

for $i \in \{1, \cdots, N\}$ **do**
  Run policy π_θ for T timesteps, collecting $\{s_t, a_t, r_t\}$
  Estimate advantages $\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t)$
  $\pi_{\text{old}} \leftarrow \pi_\theta$
  for $j \in \{1, \cdots, M\}$ **do**
    $J_{PPO}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \hat{A}_t - \lambda \text{KL}[\pi_{old}|\pi_\theta]$
    Update θ by a gradient method w.r.t. $J_{PPO}(\theta)$
  end for
  for $j \in \{1, \cdots, B\}$ **do**
    $L_{BL}(\phi) = -\sum_{t=1}^{T} (\sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t))^2$
    Update ϕ by a gradient method w.r.t. $L_{BL}(\phi)$
  end for
  if $\text{KL}[\pi_{old}|\pi_\theta] > \beta_{high} \text{KL}_{target}$ **then**
    $\lambda \leftarrow \alpha \lambda$
  else if $\text{KL}[\pi_{old}|\pi_\theta] < \beta_{low} \text{KL}_{target}$ **then**
    $\lambda \leftarrow \lambda / \alpha$
  end if
end for

  - Estimate Advantage:
    - $sum_{t'>t} \gamma^{t'-t} r_{t'}$: expected future return approximated with a sample rollout
    - $V_\psi(s_t)$: learned approximation with parameters ψ
  - Similar to AC: Actor try to maximize J_{PPO}, critic try to minimize L_{BL}
  - Scaling term $\alpha > 1$:
    - If new policy is much different from old $\rightarrow$ similar to large learning rate $\rightarrow$

hard to converge

        - If KL-divergence significantly different from the target KL (we do not want to see this) , increase its importance in J_{PPO}
        - Controls the adjustment of the KL-regularization coefficient

- Some details for DPPO:
    - D: sets a threshold for the number of workers whose gradients must be available to update the parameters
    - M, B: the number of sub-iterations with policy and baseline updates given a batch of datapoints.
    - T: the number of data points collected per worker before parameter updates are computed
    - K(for RNNs): the number of time steps for computing K-step returns and truncated backprop through time

Algorithm 2 Distributed Proximal Policy Optimization (chief)

for $i \in \{1, \cdots, N\}$ **do**
    for $j \in \{1, \cdots, M\}$ **do**
        Wait until at least $W - D$ gradients wrt. θ are available
        average gradients and update global θ
    end for
    for $j \in \{1, \cdots, B\}$ **do**
        Wait until at least $W - D$ gradients wrt. ϕ are available
        average gradients and update global ϕ
    end for
end for

Algorithm 3 Distributed Proximal Policy Optimization (worker)

for $i \in \{1, \cdots, N\}$ **do**
  for $w \in \{1, \cdots T/K\}$ **do**
    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
  $\pi_{\mathrm{old}} \leftarrow \pi_\theta$
  for $m \in \{1, \cdots, M\}$ **do**
    $J_{PPO}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \hat{A}_t - \lambda \mathrm{KL}[\pi_{old}|\pi_\theta] - \xi \mathrm{max}(0, \mathrm{KL}[\pi_{old}|\pi_\theta] - 2\mathrm{KL}_{target})^2$
    if $\mathrm{KL}[\pi_{old}|\pi_\theta > 4\mathrm{KL}_{target}$ **then**
      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
  end for
  for $b \in \{1, \cdots, B\}$ **do**
    $L_{BL}(\phi) = -\sum_{t=1}^{T} (\hat{R}_t - V_\phi(s_t))^2$
    Compute $\nabla_\phi L_{BL}$
    send gradient wrt. to ϕ to chief
    wait until gradient accepted or dropped; update 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**
    $\lambda \leftarrow \lambda/\tilde{\alpha}$
  end if
end for