12강.Actor-Critic: PPO

Contents

Advantage
GAE

Advantage function

- $A^{\pi}(s, a) = Q^{\pi}(s, a) - \boxed{V^{\pi}(s)}$

$$\longrightarrow V^{\pi}(s) = \sum_{\mathrm{a}} \pi(\mathrm{a}|\mathrm{s}) Q^{\pi}(s, a)$$

- If A(s, a1) is +, relatively good action, if – bad action
  - Mean advantage is 0:

$$E^{\pi}[A^{\pi}(s, a)|s] = \sum_{a} \pi(\mathrm{a}|\mathrm{s})[Q^{\pi}(s, a) - V^{\pi}(s)] = V^{\pi}(s) - V^{\pi}(s) = 0$$

Advantage function

- The mean gradient is the same, but the sample mean gradient variance is low because $Q^{\pi\theta}(s,a)$-V $^{\pi\theta}(s) < Q^{\pi\theta}(s,a)$

- $\nabla_\theta \mathrm{J}(\theta) = \sum_a \pi_\theta(a|s)\, Q^{\pi\theta}(s,a)\ \nabla_\theta log\pi_\theta(a|s)$
  $= \sum_a \pi_\theta(a|s)\, [Q^{\pi\theta}(s,a)\text{-V}^{\pi\theta}(s)]\ \nabla_\theta log\pi_\theta(a|s)$

- $\nabla_\theta \mathrm{J}(\theta) \approx \frac{1}{n}\sum_{(s,a,r,s')} sg\,(Q^{\pi\theta}(s,a))\ \nabla_\theta log\pi_\theta(a|s)$ High variance
  $\approx \frac{1}{n}\sum_{(s,a,r,s')} sg\,(Q^{\pi\theta}(s,a) - \text{V}^{\pi\theta}(s))\ \nabla_\theta log\pi_\theta(a|s)$ Low variance

Advantage function

- Adv. expressed by the value function:

  $A^{\pi}(s, a) = \boxed{r + V^{\pi}(s') - V^{\pi}(s)}$

  $\longrightarrow$ 1-step TD error!

- Adv. expressed by the action-value function:

  $A^{\pi}(s, a) = Q^{\pi}(s, a) - \sum_b \pi(b|s)\, Q^{\pi}(s, b)$

  or

  $A^{\pi}(s, a) = r + \gamma Q^{\pi}(s', a') - \sum_b \pi(b|s)\, Q^{\pi}(s, b)$

Advantage function

- Actor loss expressed by the value function Adv.:

$$L_{actor}(\theta) = -\frac{1}{n}\sum_{(s,a,r,s')} sg\,(\boxed{r + \gamma\, V^{\pi\theta}(s') -}\; V^{\pi\theta}(s))\; log\pi_\theta(a|s)$$

$$\nabla_\theta L_{actor}(\theta) \approx \hat{E}\;[A^{\pi\theta}(s,a)\;\nabla_\theta log\pi_\theta(a|s)]$$

Could be n-step TD target!

- Actor loss expressed by the action-value function Adv.:

$$L_{actor}(\theta) = -\frac{1}{n}\sum_{(s,a,r,s')} sg(Q^{\pi}(s, a) - \sum_b \pi(b|s)\, Q^\pi(s,b))\; log\pi_\theta(a|s)$$

or

$$L_{actor}(\theta) = -\frac{1}{n}\sum_{(s,a,r,s')} sg\boxed{(r + \gamma Q^{\pi}(s', a')} - \sum_b \pi(b|s)\, Q^\pi(s,b))\; log\pi_\theta(a|s)$$

Could be n-step TD target!

Advantage function

Initialize θ of the parameterized policy π_θ and value function V_θ and learning rate η.

While True

Collect a trajectory of length L, $\tau = (s_t, a_t, r_{t+1}, \cdots, s_{t+L_1}, a_{t+L-1}, r_{t+L}, s_{t+L})$.

$L_{critic}(\theta) = L_{actor}(\theta) = L_{exp}(\theta) = 0$

For $i \in 0 : L - 1$

$L_{critic}(\theta) \leftarrow L_{critic}(\theta) + \left(r_{t+i} + \gamma \cdot sg(V_\theta(s_{t+1+i})) - V_\theta(s_{t+i})\right)^2$

$L_{actor}(\theta) \leftarrow L_{actor}(\theta) - sg\left(r_{t+i} + \gamma V_\theta(s_{t+1+i}) - V_\theta(s_{t+i})\right) \log \pi_\theta(a_{t+i}|s_{t+i})$

$L_{exp}(\theta) \leftarrow L_{exp}(\theta) - H(\pi_\theta(\cdot|s_{t+i}))$

$L_{total}(\theta) = c_1 L_{critic}(\theta) + c_2 L_{actor}(\theta) + c_3 L_{exp}(\theta)$

$\theta \leftarrow \theta - \eta \nabla_\theta L_{total}(\theta)$

Advantage function

Initialize θ of the parameterized policy π_θ and value function V_θ and learning rate η.

While True

Collect a trajectories of length L, $\{\tau_1, \tau_2, \cdots, \tau_n\}$.

$L_{critic}(\theta) = L_{actor}(\theta) = L_{exp}(\theta) = 0$

For $\tau \in \{\tau_1, \tau_2, \cdots, \tau_n\}$

For $i \in 0 : L-1$

$$L_{critic}(\theta) \leftarrow L_{critic}(\theta) + \left(r_{t+1+i} + \gamma \cdot sg(V_\theta(s_{t+1+i})) - V_\theta(s_{t+i})\right)^2$$

$$L_{actor}(\theta) \leftarrow L_{actor}(\theta) - \left(r_{t+1+i} + \gamma \cdot sg(V_\theta(s_{t+1+i})) - V_\theta(s_{t+i})\right) \log \pi_\theta(a_{t+i}|s_{t+i})$$

$$L_{exp}(\theta) \leftarrow L_{exp}(\theta) - H(\pi_\theta(\cdot|s_{t+i}))$$

$$L_{total}(\theta) = c_1 L_{critic}(\theta) + c_2 L_{actor}(\theta) + c_3 L_{exp}(\theta)$$

$$\theta \leftarrow \theta - \eta/n \nabla_\theta L_{total}(\theta)$$

Multiple trajectories from paralleled environments

GAE

- $A_n^{\pi}(s_t, a_t) = \underbrace{r_{t+1} + \gamma r_{t+2} + \ldots + r_n + V^{\pi}(s_{t+n})}_{\text{N-step TD target}} - V^{\pi}(s_t))$

  $= \underbrace{\delta_t^{(n)}}_{\text{N-step TD error}}$

- $A_n(s_t, a_t) = r_{t+1} + \gamma r_{t+2} + \ldots + r_n + V_{\theta}^{\pi}(s_{t+n}) - V_{\theta}^{\pi}(s_t)$
  - True value function is unknown. We use learnt value function
  - As n grows, the estimation become closer to the true value
  - But variance grows because $r_{t+1} < r_{t+1} + \gamma r_{t+2} + \ldots + r_n$
  - Mixing various n size, we can find a sweet spot that outputs both precise estimation and low variance

  ➔ TD(lambda)

GAE

- $A_{GAE(\gamma,\lambda)}(s_t, a_t) = (1-\lambda)(\underbrace{A_1(s_t, a_t)}_{\delta_t} + \lambda \underbrace{A_2(s_t, a_t)}_{\delta_t + \lambda\delta_{t+1}} + \lambda^2 \underbrace{A_3(s_t, a_t)}_{\delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+1}} + \ldots)$
    - $A_n(st, at) = \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+1} + \ldots + \gamma^{n-1}\delta_{t+n}$
    - We want $\lambda + \lambda^2 + \ldots + \lambda^{n-1}$ to be 1, so multiply $(1-\lambda)$

- $A_{GAE(\gamma,\lambda)}(s_t, a_t) = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \cdots$
    - If $\lambda = 0$: 1-step
    - If $\lambda \to 1$: As λ grows, $A_{GAE(\gamma,\lambda)}(s_t, a_t)$ is $\underbrace{\text{closer to the true Return}}_{\text{lower depencence on value function}}$

GAE

Initialize θ of the parameterized policy π_θ and value function V_θ and learning rate η.

While True

Collect a trajectory of length L, $\tau = (s_t, a_t, r_{t+1}, \cdots, s_{t+L_1}, a_{t+L-1}, r_{t+L}, s_{t+L})$.

$L_{critic}(\theta) = L_{actor}(\theta) = L_{exp}(\theta) = 0$

For $i \in 0 : L-1$

$L_{critic}(\theta) \leftarrow L_{critic}(\theta) + \left(r_{t+i} + \gamma \cdot sg(V_\theta(s_{t+1+i})) - V_\theta(s_{t+i})\right)^2$

$L_{actor}(\theta) \leftarrow L_{actor}(\theta) - sg\left(\underline{A_{GAE(\gamma,\lambda)}(s_{t+i}, a_{t+i})}\right) \log \pi_\theta(a_{t+i}|s_{t+i})$

$L_{exp}(\theta) \leftarrow L_{exp}(\theta) - H(\pi_\theta(\cdot|s_{t+i}))$

$L_{total}(\theta) = c_1 L_{critic}(\theta) + c_2 L_{actor}(\theta) + c_3 L_{exp}(\theta)$

$\theta \leftarrow \theta - \eta \nabla_\theta L_{total}(\theta)$

GAE

Initialize θ of the parameterized policy π_θ and value function V_θ and learning rate η.

While True

Collect a trajectories of length L, $\{\tau_1, \tau_2, \cdots, \tau_n\}$.

$L_{critic}(\theta) = L_{actor}(\theta) = L_{exp}(\theta) = 0$

For $\tau \in \{\tau_1, \tau_2, \cdots, \tau_n\}$

For $i \in 0 : L - 1$

$$L_{critic}(\theta) \leftarrow L_{critic}(\theta) + \left(r_{t+1+i} + \gamma \cdot sg(V_\theta(s_{t+1+i})) - V_\theta(s_{t+i})\right)^2$$

$$L_{actor}(\theta) \leftarrow L_{actor}(\theta) - \underline{sg\left(A_{GAE(\gamma,\lambda)}(s_{t+i}, a_{t+i})\right)} \log \pi_\theta(a_{t+i}|s_{t+i})$$

$$L_{exp}(\theta) \leftarrow L_{exp}(\theta) - H(\pi_\theta(\cdot|s_{t+i}))$$

$$L_{total}(\theta) = c_1 L_{critic}(\theta) + c_2 L_{actor}(\theta) + c_3 L_{exp}(\theta)$$

$$\theta \leftarrow \theta - \eta/n \nabla_\theta L_{total}(\theta)$$

Monte-Carlo PG
Off-policy PG
Off-policy AC

Monte-Carlo Policy Gradient

- $J(\theta) = \sum_a \pi_\theta(a|s)\, Q^{\pi\theta}(s,a)$

※ Off-policy $\begin{cases} \pi_b : \text{collects data} \\ \pi_t : \text{learns target policy} \end{cases}$

- Objective :
  - e = (s_0, a_0, r_1, s_1, s_{T-1}, a_{T-1}, r_T, s_T)
  - $p(e|s_0) = \pi_\theta(a_0|s_0)p(s_1|s_0, a_0)\ \pi_\theta(a_1|s_1)p(s_2|s_1, a_1)\ \dots\ \pi_\theta(a_{T-1}|s_{T-1})p(s_T|s_{T-1}, a_{T-1})$
  - $R(e)$ = r1 + γr2 + … + $\gamma^{n-1} r_T$

$$J(\theta|s_0) = \sum_e p_\theta(e|s_0)\, R(e)$$

Monte-Carlo Policy Gradient

- $J(\theta|s_0) = \sum_e (p_\theta(s_1|s_0)\, \pi_\theta(a_0|s_0)$

$$p_\theta(s_2|s_0)\, \underline{sg}(\pi_\theta(a_0|s_0))$$

stop gradient

$$\vdots$$

$$p_\theta(s_T|s_0)\, sg(\pi_\theta(a_0|s_0)))\ R(e)$$

※ $\nabla_\theta \pi_\theta(a_0|s_0)$
$= \pi_\theta(a_0|s_0) \log \nabla_\theta \pi_\theta(a_0|s_0)$

- $\nabla_\theta J(\theta|s_0) = \sum_e \pi_\theta(a_0|s_0) p(e|s_0, a_0)\, R(e)\ \nabla_\theta log \pi_\theta(a_0|s_0)$

$$\approx \hat{E}\ [R(e) \nabla_\theta log \pi_\theta(a_0|s_0)]$$

- $\nabla_\theta J(\theta|s_t)\ \approx \hat{E}\ [R(e_t) \nabla_\theta log \pi_\theta(a_t|s_t)]$

Off-policy Policy Gradient

※ Off-policy

$$\begin{cases} \pi_b : \text{behavior policy that collects data} \\ \pi_\theta : \text{learns target policy} \end{cases}$$

- $J(\theta|s_0) = \sum_e p_\theta(e|s_0)\, R(e)$
  $= \sum_e p_b(e|s_0) \boxed{\frac{p_\theta(e|s_0)}{p_b(e|s_0)}} R(e)$

Importance sampling ratio

- $\nabla_\theta J(\theta|s_0) = \sum_e p_b(e|s_0) \boxed{\frac{p_\theta(e|s_0)}{p_b(e|s_0)}} R(e)\ \nabla_\theta log\pi_\theta(a_0|s_0)$
  $\approx \hat{E}\ [\boxed{\frac{p_\theta(e|s_0)}{p_b(e|s_0)}} R(e)\nabla_\theta log\pi_\theta(a_0|s_0)]$

$$\frac{\pi_\theta(a_0|s_0)\cancel{p(s_1|s_0,a_0)}\ \ldots\ \pi_\theta(a_{T-1}|s_{T-1})\cancel{p(s_T|s_{T-1},a_{T-1})}}{\pi_\theta(a_0|s_0)\cancel{p(s_1|s_0,a_0)}\ \ldots\ \pi_\theta(a_{T-1}|s_{T-1})\cancel{p(s_T|s_{T-1},a_{T-1})}}$$

$$\frac{\pi_\theta(a_0|s_0)\ \ldots\ \pi_\theta(a_{T-1}|s_{T-1})}{\pi_\theta(a_0|s_0)\ \ldots\ \pi_\theta(a_{T-1}|s_{T-1})}$$

→ This value could be a very big!!
instable learning problem due to high variance.

Off-policy Actor-Critic

※ Off-policy

$\begin{cases} \pi_{\theta'}\text{: old policy that collected data} \\ \pi_\theta\text{: learns target policy} \end{cases}$

- Actor Objective(1-step TD):
- $J(\theta) = \sum_a \sum_{s'} p(s'|s,a)\pi_\theta(a|s)\left(r + \gamma V^{\pi\theta}(s')\right)$

$$\approx \hat{\mathrm{E}}^{\pi\theta}[r + \gamma V^{\pi\theta}(s')]$$

$$\approx \hat{\mathrm{E}}^{\pi\theta'}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}(r + \gamma V^{\pi\theta}(s'))\nabla_\theta \log\pi_\theta(a|s)\right]$$

- $\nabla_\theta J(\theta|s_0) = \sum_a \sum_{s'} p(s'|s,a)\pi_\theta(a|s)\left(r + \gamma V^{\pi\theta}(s')\right)\nabla_\theta log\pi_\theta(a|s)$

$$\approx \hat{\mathrm{E}}^{\pi\theta}\left[(r + \gamma V^{\pi\theta}(s'))\nabla_\theta log\pi_\theta(a|s)\right]$$

$$= \sum_a \sum_{s'} p(s'|s,a)\pi_{\theta'}(a|s)\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}\left(r + \gamma V^{\pi\theta}(s')\right)\nabla_\theta log\pi_\theta(a|s)$$

$$\approx \hat{\mathrm{E}}^{\pi\theta'}\left[\boxed{\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}}(r + \gamma V^{\pi\theta}(s'))\nabla_\theta \log\pi_\theta(a|s)\right]$$

This value could be a very big!!
instable learning problem due to high variance.

Off-policy Actor-Critic

※ Off-policy

$\begin{cases} \pi_{\theta'}: \textit{old policy} \text{ that collected data} \\ \pi_\theta: \text{learns target policy} \end{cases}$

- What about Critic Objective?
- $L(\theta) = \sum_a \sum_{s'} p(s'|s,a)\pi_\theta(a|s)\left((r+\gamma \cdot \mathrm{sg}(V^{\pi\theta}(s'))) - V^{\pi\theta}(s)\right)^2$

$$\approx \hat{E}^{\pi\theta}[\left((r+\gamma \cdot \mathrm{sg}(V^{\pi\theta}(s'))) - V^{\pi\theta}(s)\right)^2]$$

$$= \sum_a \sum_{s'} p(s'|s,a)\pi_{\theta'}(a|s)\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}\left(r+\gamma \cdot \mathrm{sg}(V^{\pi\theta}(s')) - V^{\pi\theta}(s)\right)^2$$

$$\approx \hat{E}^{\pi\theta'}[\boxed{\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}}\left(r+\gamma \cdot \mathrm{sg}(V^{\pi\theta}(s')) - V^{\pi\theta}(s)\right)^2]$$

→ This value could be a very big!!
instable learning problem due to high variance.

Proximal Policy Optimization(PPO)

Overview

- PPO problem statement:
  - Q-learning is tricky,
  - vanilla PG is low in sample efficiency and unstable in learning
  - TRPO is stable but rather complex and unable to share parameters btw. value and policy

- PPO uses Off-policy data and update multiple times per sample (high sample efficiency)

Overview

- PPO updates the policy conservatively

- 'conservative' means to stop the policy change too fast or dramatically
  - Importance Sampling ratio could be a very big!!
    (instable learning problem due to high variance)
  - Updating multiple times per sample worsen learning instabilities
  - An actor learning w/o well learnt critic adds instability
    (sub-optima convergence)

Method

- $L^{pg}(\theta) = \hat{E}_t\left[log\pi_\theta(a_t|st)\hat{A}_t\right]$
  - Multiple time updates should look good for learning
  - But, in fact, empirically, multiple updates cause dramatic change in policy that lead to instability in learning

- $\nabla_\theta L^{pg}(\theta) = \hat{E}_t\left[\nabla_\theta log\pi_\theta(a_t|st)\hat{A}_t\right]$
  - π_θ: stochastic policy
  - $\hat{A}_t$: advantage estimator at time step t

Method

- KL-divergence (Discrete distribution):

$$KL[p,q] = \sum_{x} p(x) \log \frac{p(x)}{q(x)} \geq 0$$

  - measure of how much two distributions are different
  - 0 if p = q. The greater KL-divergence is, the bigger the difference is

→ Want to increase value accuracy as much as possible
while updating the policy very conservatively

Method

- TRPO Objective that limit the amount of change in Policy:

$$\underset{\theta}{\text{maximize}}\ \hat{E}_t[\frac{\pi_\theta(at|st)}{\pi_{\theta old}(a_t|s_t)}\hat{A}_t]$$

※ $\nabla_\theta \pi_\theta(a_0|s_0)$
$= \pi_\theta(a_0|s_0)\log\nabla_\theta \pi_\theta(a_0|s_0)$

$$\longrightarrow\ \nabla_\theta J(\theta) = \hat{E}\ [\frac{\nabla_\theta \pi(a_t|s_t)}{\pi_{\theta\text{old}}(a_t|s_t)}\hat{A}_t]$$

$$\longrightarrow\ \nabla_\theta J(\theta) = \hat{E}^{\pi\theta\text{old}}[\frac{\pi(a_t|s_t)}{\pi_{\theta\text{old}}(at|s_t)}\hat{A}_t\nabla_\theta log\pi_\theta(a_0|s_0)] \longleftarrow$$

Equal to off-policy PG

$$subject\ to\ \hat{E}_t\left[KL[\pi_{\theta old}(\cdot\,|s_t), \pi_\theta(\cdot\,|s_t)]\right] \leq\ \delta$$

$\longrightarrow$ Allow the new policy to change by δ from the old policy

Objective

- TRPO:

$$\underset{\theta}{\text{maximize}}\ \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}\hat{A}_t\ \underbrace{-\ \beta KL[\pi_{\theta old}(\cdot\,|s_t),\pi_\theta(\cdot\,|s_t)]}\right]\ with\ \underbrace{\beta \geq 0}$$

Stop a new policy become too different from the behavior policy $\pi_{\theta old}$

Setting β is difficult

- PPO:

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

if $rt(\theta) \leq 1-\varepsilon$,or $rt(\theta) \geq 1+\varepsilon$
No gradient(no learning),

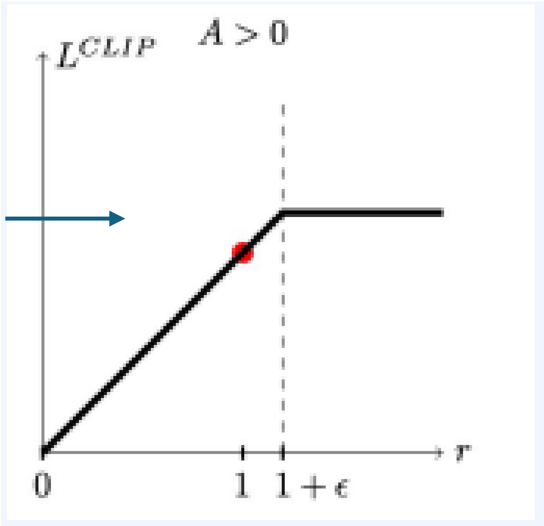
$$with\ r_{t(\theta)} = \frac{\pi_{\theta(a_t|s_t)}}{\pi_{\theta old}(a_t|s_t)}\ \ ,1 > \ \varepsilon\ > 0$$

Objective

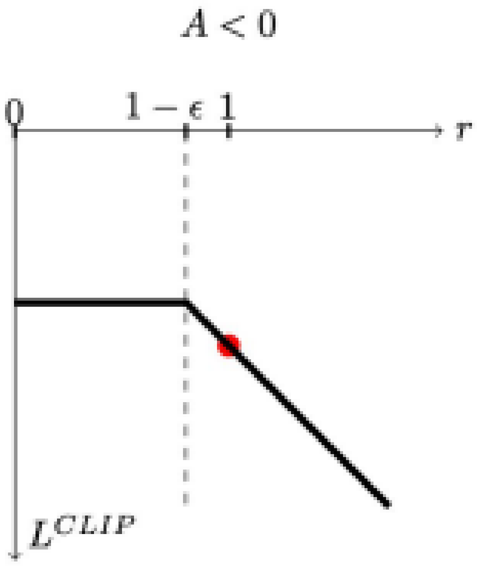
$$L^{clip}(\theta) = \hat{E}_t\left[\min(r_t(\theta)\,\hat{A}_t,\ clip(r_t(\theta), 1-\varepsilon, 1+\epsilon)\hat{A}_t\right]$$

$$with\ r_{t(\theta)} = \frac{\pi_{\theta(a_t|s_t)}}{\pi_{\theta old}(a_t|s_t)}\ ,1 > \ \varepsilon\ > 0$$

Loss does not grow = No gradient



Limit an action prob. to grow only (1 $\pm\varepsilon$) times greater or smaller than previous policy (control policy to change only$\pm\varepsilon$ ratio)



Objective

Maximize constraint off-policy gradient

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right]$$

Minimize loss

bonus

$$L_t^{VF}(\theta) = (V_\theta(s_t) - V_t^{targ})^2$$

S : Entropy $\hat{A}_t = A_t^{GAE(\gamma,\lambda)}$

Algorithm 1 PPO, Actor-Critic Style

Parallel learning

```
for iteration=1, 2, ... do
    for actor=1, 2, ..., N do
        Run policy π_θold in environment for T timesteps
        Compute advantage estimates Â_1, ..., Â_T
    end for
    Optimize surrogate L wrt θ, with K epochs and minibatch size M ≤ NT
    θ_old ← θ
end for
```

Results

- Performance comparison

No clipping or penalty:	$L_t(\theta) = r_t(\theta)\hat{A}_t$
Clipping:	$L_t(\theta) = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta)), 1-\epsilon, 1+\epsilon)\hat{A}_t$
KL penalty (fixed or adaptive)	$L_t(\theta) = r_t(\theta)\hat{A}_t - \beta\,\text{KL}[\pi_{\theta_{\text{old}}}, \pi_\theta]$

- Test for 7 robotic task in Mujoco (continuous domain!)
- 1M step updates
- Test with multiple β, d_{targ}, various ε
- Policy: 64 hidden unit, 2 layer MLP, tanh activation, with output of Gaussian (mean, std)

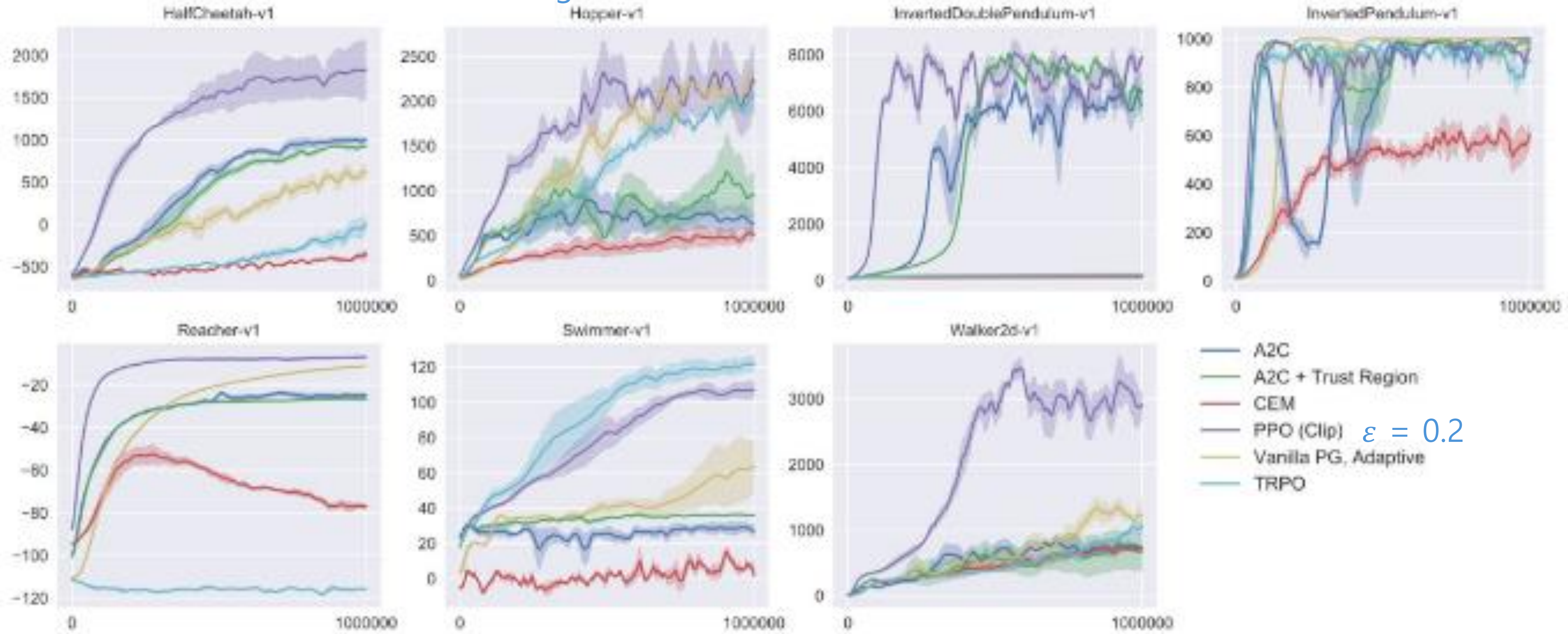
Results

- No parameter sharing, no entropy bonus
- 3 random seed
- avg. score for the last 100 before the end
- Normalization 0~1
- No clipping & KL penalty is -0.39
- Clipping score is 0.82

algorithm	avg. normalized score
No clipping or penalty	-0.39
Clipping, $\epsilon = 0.1$	0.76
Clipping, $\epsilon = 0.2$	**0.82**
Clipping, $\epsilon = 0.3$	0.70
Adaptive KL $d_{\text{targ}} = 0.003$	0.68
Adaptive KL $d_{\text{targ}} = 0.01$	0.74
Adaptive KL $d_{\text{targ}} = 0.03$	0.71
Fixed KL, $\beta = 0.3$	0.62
Fixed KL, $\beta = 1.$	0.71
Fixed KL, $\beta = 3.$	0.72
Fixed KL, $\beta = 10.$	0.69

Results

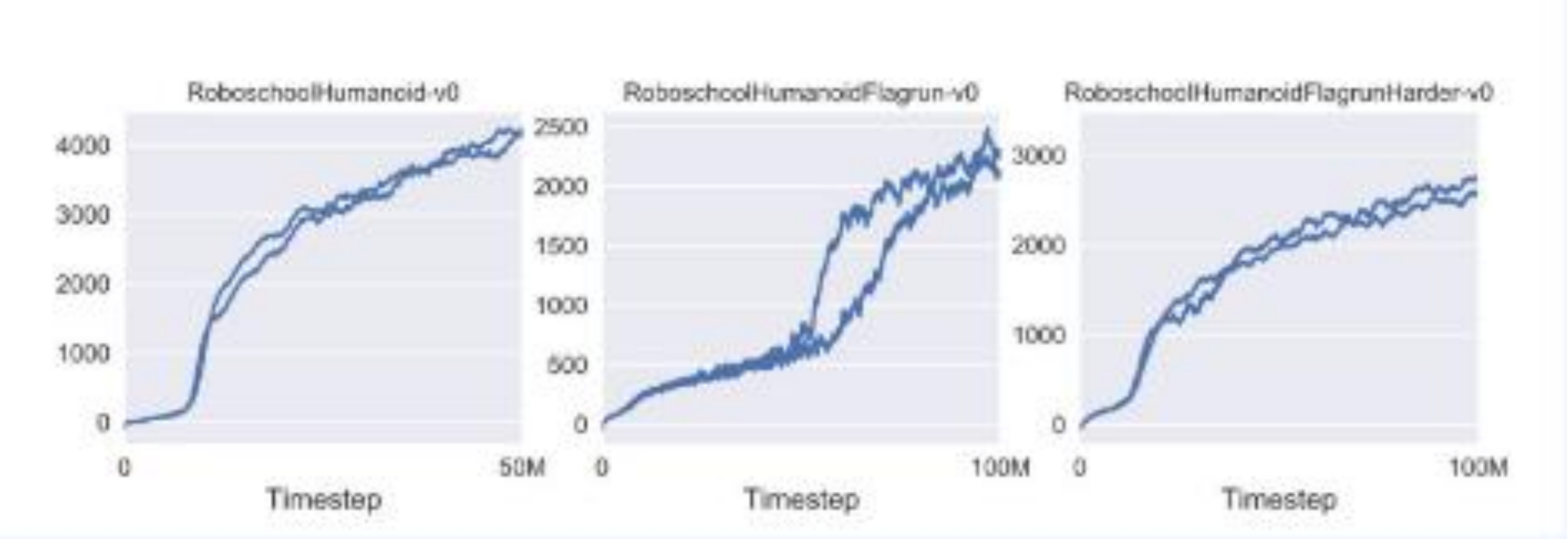
PPO shows sample efficiency
and stable learning



$\varepsilon = 0.2$

Results

- High-dimensional continuous control test of PPO: 3D humanoid
    1. RoboschoolHumannoid: moving forward
    2. RoboschoolHumanoidFlagrun: moving forward with moving target every 200 timestep
    3. RoboschoolHumanoidFlagranHarder: throw a cube to the robot tumble and stand-up again



Results

- Learning in Atari domain:

Hyperparameter	Value
Horizon (T)	128
Adam stepsize	$2.5 \times 10^{-4} \times \alpha$
Num. epochs	3
Minibatch size	32×8
Discount (γ)	0.99
GAE parameter (λ)	0.95
Number of actors	8
Clipping parameter ϵ	$0.1 \times \alpha$
VF coeff. c_1 (9)	1
Entropy coeff. c_2 (9)	0.01

The number of games out of 48

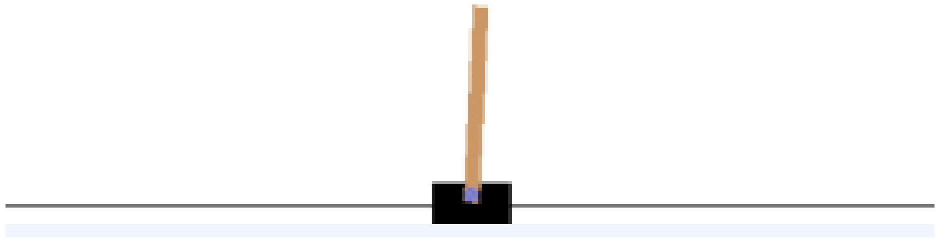
	A2C	ACER	PPO	Tie
(1) avg. episode reward over all of training	1	18	**30**	0
(2) avg. episode reward over last 100 episodes	1	**28**	19	1

Code Ex.1

- configuration.py
- utils.py
- agent.py
- train.py
- eval.py

Code Ex.1

- ‘Cartpole’ Env.: move a cart (black) such that it balances a pendulum (brown) without moving too far from the center.
- State:
  - The agent observes current position and velocity of the cart, as well as angle and velocity of the pole
  - cart position, cart velocity, pole angle, pole angular velocity
- Action:
  - 0: push the cart left
  - 1: push the cart right
- Reward:+1 for every step



configuration.py

```
from utils import AttrDict

config = AttrDict(
    gamma=0.99,
    lam=0.95,
    eps_clip=100,
    k_epoch=4,
    lr=1e-4,
    c1=1,
    c2=0.5,
    c3=1e-3,
    num_env=8,
    seq_length=16,
    batch_size=64,
    minibatch_size=16,
    hidden_size=128,
    train_env_steps=1000000,
    num_eval_episode=100,
)
```

Critic loss constant
actor loss constant
Exploration loss constant (entropy loss constant)
Parallel environment to collect trajectories
Length of a trajectory
Learning from a batch that contains 64 trajectories

train.py

```
import torch
import datetime

import numpy as np

from utils import create_env
from agent import Agent
from configuration import config
from collections import deque
from torch.utils.tensorboard import SummaryWriter


if __name__ == '__main__':
    env_list = [create_env(config) for _ in range(config.num_env)]
    agent = Agent(env_list[0], config)
    agent.set_optimizer()
    assert config.batch_size % config.num_env == 0

    dt_now = datetime.datetime.now()
    logdir = f"logdir/{dt_now.strftime('%y-%m-%d_%H-%M-%S')}"
    writer = SummaryWriter(logdir)

    score_que = deque([], maxlen=config.num_eval_episode)
    count_step_que = deque([], maxlen=config.num_eval_episode)

    score = 0
    count_step = 0
    s_list = [env.reset() for env in env_list]
    score_list = [0 for env in env_list]
    count_step_list = [0 for env in env_list]
```

Vector environments
One agent

train.py

```
num_iteration = int(config.train_env_steps / config.num_env / config.seq_length)
for step_iteration in range(num_iteration):
    for i_env in range(config.num_env):
        env = env_list[i_env]
        s = s_list[i_env]
        for _ in range(config.seq_length):
            pi, a = agent.action(s)
            s_next, r, done, info = env.step(a)
            agent.add_to_batch(s, pi, a, r, s_next, done)
            s = s_next
            score_list[i_env] += r
            count_step_list[i_env] += 1

            s_list[i_env] = s
            if done:
                s = env.reset()
                s_list[i_env] = s

                score_que.append(score_list[i_env])
                count_step_que.append(count_step_list[i_env])

                score_list[i_env] = 0
                count_step_list[i_env] = 0

                break
```

Parallelized environments shares config.train_env_steps load

One agent plays in multiple environments

Trajectory rollout until seq_length

pi is behavior policy
a is an action sampled
from pi

Put the behavior policy
into a trajectory in batch

agent.py

```
def calc_return_seq_tensor(gamma, reward_seq_tensor):
    seq_length, n_batch = reward_seq_tensor.shape
    gamma_seq = gamma * torch.ones(reward_seq_tensor.shape)
    return_seq = torch.zeros(seq_length, n_batch)  # Initialize return sequence

    for t in range(seq_length):
        gamma_seq_from_t = gamma_seq[t:, :]  # (n_seq, n_batch)
        powers = torch.arange(seq_length - t).unsqueeze(-1).repeat(1, n_batch)  # (n_seq, n_batch)
        gamma_power_seq_from_t = torch.pow(gamma_seq_from_t, powers)  # (n_seq, n_batch)
        reward_seq_from_t = reward_seq_tensor[t:, :]  # (n_seq, n_batch)
        g_t = torch.sum(reward_seq_from_t * gamma_power_seq_from_t, dim=0)  # (n_batch)
        return_seq[t, :] = g_t

    return return_seq
```

GAE(Generalized advantage function)

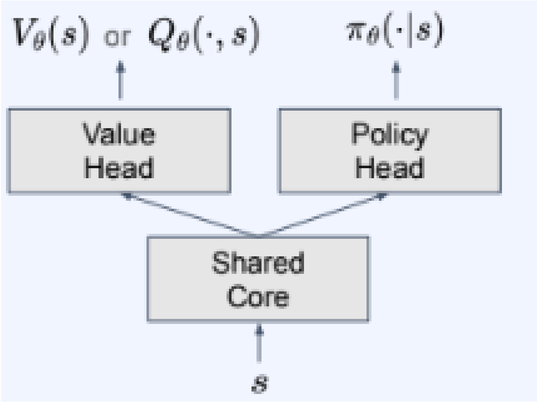
agent.py

```
class Agent(nn.Module):
    def __init__(self, env, config):
        super().__init__()
        self.config = config

        d_state = env.observation_space.shape[0]
        n_action = env.action_space.n

        self.encoder = nn.Sequential(
            nn.Linear(d_state, self.config.hidden_size),
            nn.ELU(),
            nn.Linear(self.config.hidden_size, self.config.hidden_size),
            nn.ELU()
        )
        self.value_head = nn.Sequential(
            nn.Linear(self.config.hidden_size, self.config.hidden_size),
            nn.ELU(),
            nn.Linear(self.config.hidden_size, 1)
        )
        self.policy_head = nn.Sequential(
            nn.Linear(self.config.hidden_size, self.config.hidden_size),
            nn.ELU(),
            nn.Linear(self.config.hidden_size, n_action),
            nn.Softmax(dim=-1)
        )

        self.batch = deque([], maxlen=config.batch_size)  # list of episodes
```



batch size is 64 trajectories
Each trajectory has seq. length 16

agent.py

```python
def crate_trajectory(self):
    trajectory = {
        'state': list(),
        'pi_old': list(),
        'action': list(),
        'reward': list(),
        'state_next': list(),
        'done': list(),
    }
    return trajectory

def add_to_batch(self, s, pi_old, a, r, s_next, done):
    if (
            len(self.batch) == 0
            or len(self.batch[-1]['state']) == self.config.seq_length
    ):
        trajectory = self.crate_trajectory()
        self.batch.append(trajectory)

    if not done:
        length_to_append = 1
    else:
        # When the trajectory is done before it is full, append the last data until the end
        length_to_append = self.config.seq_length - len(self.batch[-1]['state'])

    for _ in range(length_to_append):
        self.batch[-1]['state'].append(s)
        self.batch[-1]['pi_old'].append(pi_old)
        self.batch[-1]['action'].append(a)
        self.batch[-1]['reward'].append(r)
        self.batch[-1]['state_next'].append(s_next)
        self.batch[-1]['done'].append(done)
```

A trajectory dictionary

We'll record the behavior policy every time step the agent act.

Add a sample to the batch

When a seq. is full
Create a new seq.
Add the seq. to the batch

agent.py

```
def set_optimizer(self):
    self.optim = torch.optim.Adam(
        self.parameters(),
        lr=self.config.lr
    )

def forward(self, x):
    h_enc = self.encoder(x)
    value = self.value_head(h_enc)
    pi = self.policy_head(h_enc)
    return pi, value

def action(self, x):
    # used when sampling an action for a state
    with torch.no_grad():
        x = torch.from_numpy(x).float().reshape(1, -1)
        pi, value = self.forward(x)
        a = torch.distributions.Categorical(pi).sample().item()
        pi = pi.numpy().squeeze(0)  # (n_action)
    return pi, a
```

Action sample using pi distribution

agent.py

```python
def train(self):
    for k in range(self.config.k_epoch):
        minibatch = random.sample(self.batch, self.config.minibatch_size)
        state_seq_array = np.array([trajectory['state'] for trajectory in minibatch])  # (n_batch, n_seq, *dim_state)
        pi_old_seq_array = np.array([trajectory['pi_old'] for trajectory in minibatch])  # (n_batch, n_seq, n_action)
        action_seq_array = np.array([trajectory['action'] for trajectory in minibatch], dtype=np.int64)  # (n_batch, n_seq)
        reward_seq_array = np.array([trajectory['reward'] for trajectory in minibatch])  # (n_batch, n_seq)
        state_next_seq_array = np.array([trajectory['state_next'] for trajectory in minibatch])  # (n_batch, n_seq, *dim_state)
        done_seq_array = np.array([trajectory['done'] for trajectory in minibatch])  # (n_batch, n_seq)

        state_seq_tensor = torch.from_numpy(
            state_seq_array
        ).float().transpose(0, 1)  # (n_seq, n_batch, *dim_states)
        pi_old_seq_tensor = torch.from_numpy(pi_old_seq_array).transpose(0, 1)  # (n_seq, n_batch, n_action)
        action_seq_tensor = torch.from_numpy(action_seq_array).transpose(0, 1)  # (n_seq, n_batch)
        reward_seq_tensor = torch.from_numpy(reward_seq_array).float().transpose(0, 1)  # (n_seq, n_batch)
        state_next_seq_tensor = torch.from_numpy(
            state_next_seq_array
        ).float().transpose(0, 1)  # (n_seq, n_batch, *dim_states)
        done_seq_tensor = torch.from_numpy(done_seq_array).float().transpose(0,1)  # (n_seq, n_batch)

        # mask for updating policy, until the transition that its done is True
        update_mask = done_seq_tensor.roll(1, dims=0)  # (n_seq, n_batch)
        update_mask[0, :] = 0  # (n_seq, n_batch)
        update_mask = 1 - update_mask  # (n_seq, n_batch)
```

Train repeating k epoch with minibatch_size

To insert into NN.

Mask for updating policy

agent.py

```python
pi, value = self.forward(state_seq_tensor)  # (n_seq, n_batch, n_action), (n_seq, n_batch, 1)
_, value_next = self.forward(state_next_seq_tensor)  # (n_seq, n_batch, 1)
value = value.squeeze(-1)  # (n_seq, n_batch)
value_next = value_next.squeeze(-1)  # (n_seq, n_batch)

delta = reward_seq_tensor + self.config.gamma * (1 - done_seq_tensor) * value_next - value
gae = calc_return_seq_tensor(self.config.lam * self.config.gamma, update_mask * delta.detach())

pi_chosen = pi.gather(dim=-1, index=action_seq_tensor.unsqueeze(-1))  # (n_seq, n_batch, 1)
pi_chosen = pi_chosen.squeeze(-1)  # (n_seq, n_batch)

pi_old_chosen = pi_old_seq_tensor.gather(dim=-1, index=action_seq_tensor.unsqueeze(-1))  # (n_seq, n_batch, 1)
pi_old_chosen = pi_old_chosen.squeeze(-1)  # (n_seq, n_batch)

value_target = (
    reward_seq_tensor
    + self.config.gamma * (1 - done_seq_tensor) * value_next.detach()
)  # (n_seq, n_batch)

loss_critic = torch.mean(update_mask * (value_target - value) ** 2)

r = pi_chosen / pi_old_chosen
loss_actor = -torch.mean(
    update_mask * torch.min(
        gae * r,
        gae * torch.clip(r, 1 - self.config.eps_clip, 1 + self.config.eps_clip)
    )
)
```

[P(a1),p(a2)], $V(s_t)$

$V(s_{t+1})$

$\hat{A}_t$ is GAE that uses N-step TD target

Value target

Critic loss(L^{VF})

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}$$

Actor loss $L^{clip}(\theta) = \hat{E}_t\left[\min(rt(\theta)\,\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\epsilon)\hat{A}_t)\right]$

agent.py

Entropy bonus

```
loss_exp = -torch.mean(
    update_mask
    * torch.sum(-pi * torch.log(pi + 1e-15), dim=-1)  # (n_seq, n_batch, n_action) -> (n_seq, n_batch)
)
```

Add all loss and make actor – critic loss

```
loss = self.config.c1 * loss_critic + self.config.c2 * loss_actor + self.config.c3 * loss_exp

loss_critic_avg = loss_critic * self.config.seq_length * self.config.minibatch_size / update_mask.sum()
entropy_avg = -loss_exp * self.config.seq_length * self.config.minibatch_size / update_mask.sum()
```

Gradient descent

```
self.optim.zero_grad()
loss.backward()
self.optim.step()
```

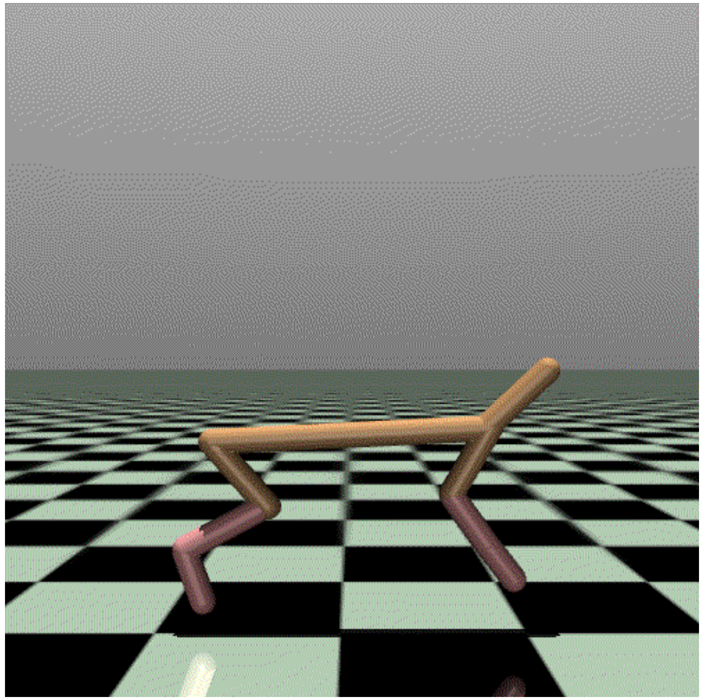
Throw away the batch

```
self.batch.clear()

return loss_critic_avg.detach().item(), entropy_avg.detach().item()
```

Code Ex.2

- Upload 'proximal_policy_optimizatin.ipynb' file onto Colab
- The notebook includes 'Mujoco' environment, 'Half Cheetah'
- State(17):
  - (body parts) position,
  - (roter, axis) angle or velocity
- Action(6):
  - Torque on 6 roters (-1~1)
- Reward:
  - Forward_reward: moving-forward reward
  - ctrl_cost: penalty in taking too large action



Continuous Action

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, \ldots$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

end for

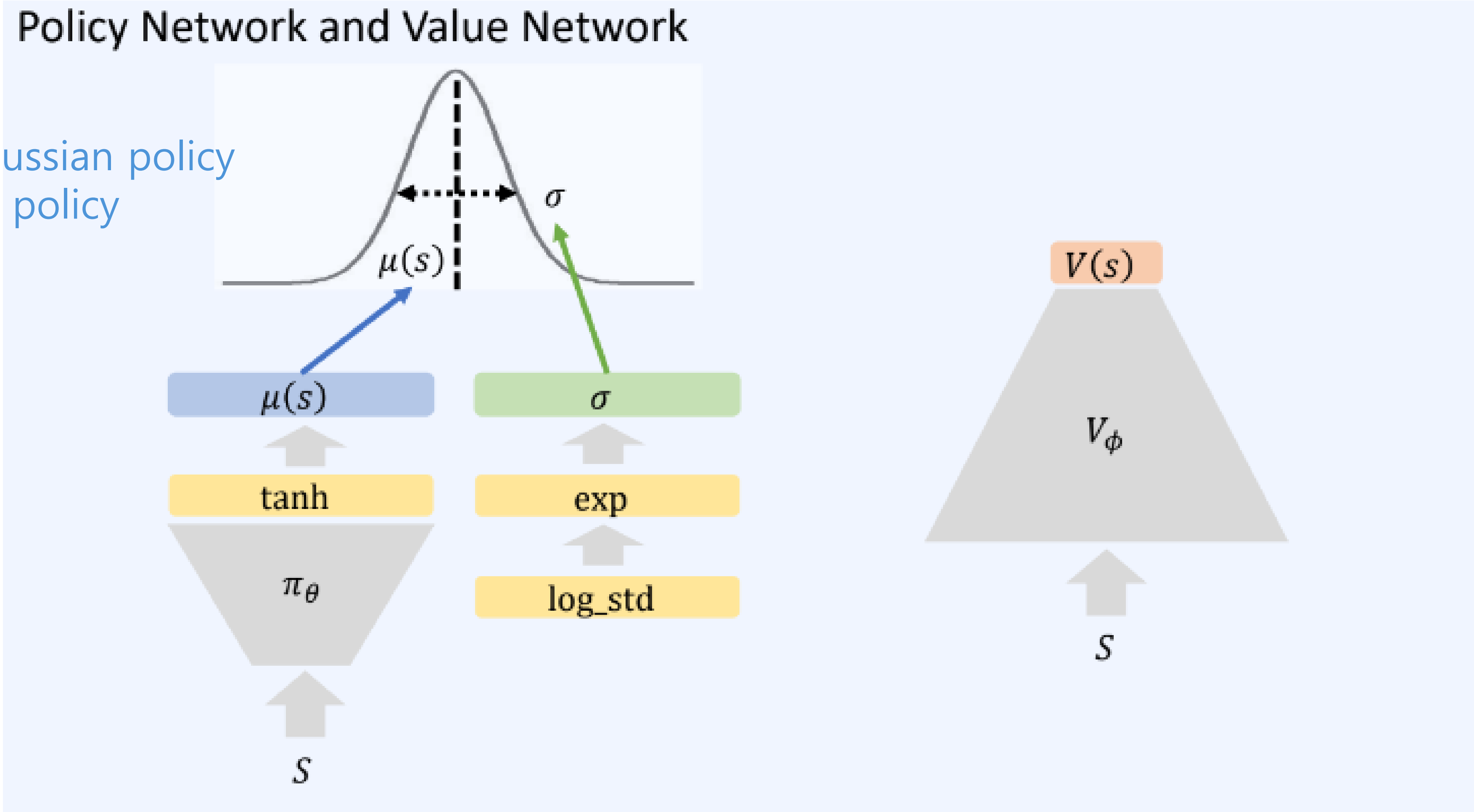
https://spinningup.openai.com/en/latest/algorithms/ppo.html#background

2 NN, θ, φ

(old) behavior policy

Continuous Action

PPO uses Gaussian policy
as stochastic policy

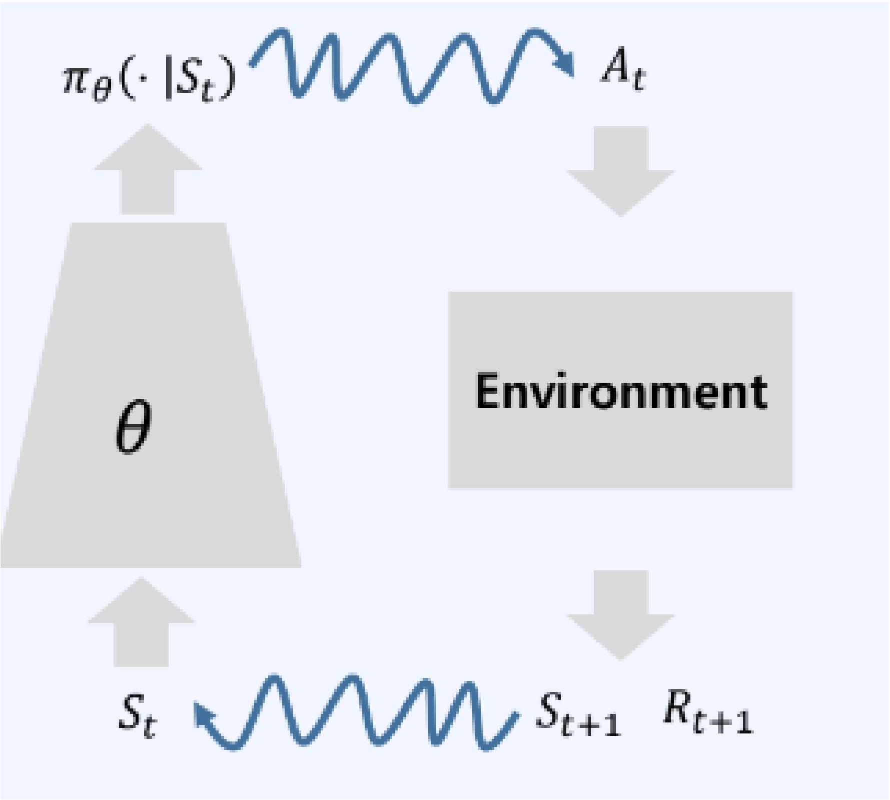


Collecting episodes

- Initial sampling
  - $S_0 \sim d(\cdot)$
- Action sampling from Policy
  - $\mu_t = \pi_\theta(s)$
  - $\varepsilon \sim N(o, I)$:Noisiness for exploration
  - $a_t = \mu + \sigma\varepsilon_t$:Reparameterization trick

learnt std.

- Simulation
  - $s_{t+1}, r_{t+1}, d_{t+1}$ = env.step(a_t)
  - Next_state
  - Reward
  - Done, terminal information



Loss function

- Return Definition:
  - $G_t = \sum_{k=0} \gamma^k R_{t+1+k}$
  - recursive: $G_t = R_{t+1} + \gamma \mathrm{G}_{t+1}$
- Return Calculation (Backward):
  - $G_{t+1} = 0$
  - From t = T to t= 0, compute $G_t = R_{t+1} + \gamma \mathrm{G}_{t+1}$
- Generalized Advantage Estimation(GAE):
  - $\delta_t = R_{t+1} + \gamma V(s_{t+1}) - V(st)$
  - $\mathrm{A}_\mathrm{t}^{(\lambda)} = \sum_{k=0} (\gamma\lambda)^k \delta_{t+k} = \delta_t + \gamma\lambda \sum_{k=0} (\gamma\lambda)^k \delta_{t+1+k} = \delta_t + \gamma\lambda\ \mathrm{A}_{\mathrm{t+1}}^{(\lambda)}$

Policy Loss

There's no differentiation of φ network (use no grad or detach()) or *old*(recorded old π_θ)

$$B = \{(S_t, A_t, R_{t+1}, S_{t+1})\}$$

Get TD-error

$$\delta_t = R_{t+1} + \gamma \boxed{V_\phi(s_{t+1})} - \boxed{V_\phi(s_t)}$$

$$\delta_0, \delta_1, \delta_2, \dots, \delta_t, \dots, \delta_T$$

$$r_i = \frac{\boxed{\pi_\theta}(a_i|s_i)}{\boxed{\pi_{old}}(a_i|s_i)}$$

Get GAE using TD-error

$$A_t^{(\lambda)} = \delta_t + \gamma\lambda A_{t+1}^{(\lambda)}$$

$$A_0^{(\lambda)}, A_t^{(\lambda)}, A_2^{(\lambda)}, \dots, A_t^{(\lambda)}, \dots, A_{T+1}^{(\lambda)}$$

Differentiation of θ network will be performed

$$\min_\theta \frac{1}{N}\sum_i \min(\boxed{r_i} A_i^{(\lambda)}, \boxed{[r_i]}_{1-\epsilon}^{1+\epsilon} A_i^{(\lambda)})$$

$[r_i]_{1-\epsilon}^{1+\epsilon}$: Clipped Ratio

Value Loss

$$B = \{(S_t, A_t, R_{t+1}, S_{t+1})\}$$

$$G_t = \sum \gamma^k R_{t+k}$$

$$\min_{\phi} \frac{1}{N} \sum_i \frac{1}{2} \left(G_i - \boxed{V_\phi(s_i)}\right)^2$$

Differentiation of φ network will be performed