106062109 黃中鈺

- **What algorithm have you implemented?**

  I first implement original Actor-Critic version, then PPO(Proximal Policy Optimization).

  Here's the rewards comparison:

Actor-Critic	PPO
Episode 20 length: 154 reward: -396.04188633687227	Episode 20 avg length: 79 reward: -137
Episode 40 length: 118 reward: -267.82300554392606	Episode 40 avg length: 86 reward: -183
Episode 60 length: 150 reward: -517.5924279392121	Episode 60 avg length: 90 reward: -160
Episode 80 length: 153 reward: -423.0069357130834	Episode 80 avg length: 78 reward: -178
Episode 100 length: 113 reward: -326.1209005643042	Episode 100 avg length: 80 reward: -135
Episode 120 length: 98 reward: -242.49694728043406	Episode 120 avg length: 80 reward: -124
Episode 140 length: 89 reward: -449.4194637776465	Episode 140 avg length: 83 reward: -148
Episode 160 length: 87 reward: -482.7429554841181	Episode 160 avg length: 83 reward: -130
Episode 180 length: 69 reward: -202.55485482418368	Episode 180 avg length: 84 reward: -125
Episode 200 length: 81 reward: -72.84484324908954	Episode 200 avg length: 83 reward: -115
Episode 220 length: 153 reward: -67.53457492940767	Episode 220 avg length: 82 reward: -112
Episode 240 length: 98 reward: -55.19903008630845	Episode 240 avg length: 84 reward: -98
Episode 260 length: 142 reward: -71.64755129128199	Episode 260 avg length: 90 reward: -87
Episode 280 length: 114 reward: -154.9368873101258	Episode 280 avg length: 105 reward: -84
Episode 300 length: 164 reward: -94.65030034212754	Episode 300 avg length: 110 reward: -86
Episode 320 length: 114 reward: -65.86661435346787	Episode 320 avg length: 141 reward: -72
Episode 340 length: 240 reward: -142.0397544048803	Episode 340 avg length: 154 reward: -50
Episode 360 length: 113 reward: -7.190940598384961	Episode 360 avg length: 222 reward: -43
Episode 380 length: 90 reward: -13.850279174129543	Episode 380 avg length: 234 reward: 0
Episode 400 length: 999 reward: -31.896496560252377	Episode 400 avg length: 209 reward: 13
Episode 420 length: 533 reward: -167.72595909472014	Episode 420 avg length: 255 reward: 15
Episode 440 length: 107 reward: 15.139900602529238	Episode 440 avg length: 235 reward: 39
Episode 460 length: 109 reward: -2.381383773087257	Episode 460 avg length: 234 reward: 67
Episode 480 length: 999 reward: 25.760769990764015	Episode 480 avg length: 208 reward: 37
Episode 500 length: 98 reward: 24.396068747841376	Episode 500 avg length: 266 reward: 43
Episode 520 length: 84 reward: -8.112633434886545	Episode 520 avg length: 252 reward: 68
Episode 540 length: 105 reward: -10.359116183660918	Episode 540 avg length: 231 reward: 86
Episode 560 length: 352 reward: 15.981948831060397	Episode 560 avg length: 233 reward: 77
Episode 580 length: 133 reward: -26.934258157859848	Episode 580 avg length: 221 reward: 83
Episode 600 length: 87 reward: -30.692083237474	Episode 600 avg length: 247 reward: 68
Episode 620 length: 132 reward: 42.808327701430144	Episode 620 avg length: 252 reward: 84
Episode 640 length: 265 reward: 55.447502891734544	Episode 640 avg length: 222 reward: 42
Episode 660 length: 661 reward: 58.21749017306778	Episode 660 avg length: 224 reward: 76
Episode 680 length: 211 reward: 70.35205757349966	Episode 680 avg length: 180 reward: 52
Episode 700 length: 119 reward: 0.3060176177746371	Episode 700 avg length: 209 reward: 59
Episode 720 length: 124 reward: 6.983869297775692	Episode 720 avg length: 166 reward: 37
Episode 740 length: 125 reward: 24.21127854731189	Episode 740 avg length: 180 reward: 50
Episode 760 length: 186 reward: 159.17501666524296	Episode 760 avg length: 152 reward: 32
Episode 780 length: 480 reward: 77.06971429338077	Episode 780 avg length: 202 reward: 19
Episode 800 length: 194 reward: -60.57491026973844	Episode 800 avg length: 197 reward: 33
Episode 820 length: 310 reward: 139.91903861701533	Episode 820 avg length: 195 reward: 81
Episode 840 length: 155 reward: 93.73931904323548	Episode 840 avg length: 213 reward: 80
Episode 860 length: 185 reward: 164.68134109540782	Episode 860 avg length: 265 reward: 101
Episode 880 length: 253 reward: 135.43161183693707	Episode 880 avg length: 249 reward: 103
Episode 900 length: 397 reward: 130.396809300553	Episode 900 avg length: 268 reward: 124
Episode 920 length: 159 reward: 98.11736578930024	Episode 920 avg length: 269 reward: 137
Episode 940 length: 135 reward: 118.98843261917996	Episode 940 avg length: 264 reward: 110
Episode 960 length: 175 reward: 198.11829463544026	Episode 960 avg length: 259 reward: 99
Episode 980 length: 136 reward: 130.3790677522589	Episode 980 avg length: 248 reward: 95
Episode 1000 length: 372 reward: 155.0528370037614	Episode 1000 avg length: 268 reward: 137
Episode 1020 length: 202 reward: 193.112741614541	Episode 1020 avg length: 262 reward: 120
Episode 1040 length: 177 reward: 125.84665034747836	Episode 1040 avg length: 280 reward: 156
Episode 1060 length: 205 reward: 181.51091753426118	Episode 1060 avg length: 263 reward: 120
	Episode 1080 avg length: 291 reward: 151

I think PPO is much more stable than AC but it improves slower.

- **How do you implement the algorithm? (I will only discuss PPO version)**
  1. Read the PPO paper https://arxiv.org/abs/1707.06347
  2. Use Pytorch

ActorCritic part:

Action layer:

Sequential layers:Use Linear layers and Tanh activation function for first 2 times, and Linear layer with Softmax regression on output layer.

Critic layer:

Sequential layers:Use Linear layers and Tanh activation function for first 2 times, and Linear layer on output layer.

PPO part:

Optimizer: Adam

Policy(actor-critic) update:

Step 1: Use Monte Carlo estimate state rewards.

Step 2: Normalize the rewards

Step 3: Convert list of actions, states and log probabilities to tensor for later computation.

Step 4: Optimize policy for K epochs(K is 4)

Step 5: In each epoch:

Evaluate old actions and values

Find ratio (the explanation and code below)

But as we refine the current policy, the difference between the current and the old policy is getting larger. The variance of the estimation will increase. So, say for every 4 iterations, we synchronize the second network with the current policy again.

$$\pi_{\theta_{k+1}}(a_t|s_t) \longleftarrow \pi_\theta(a_t|s_t)$$

In PPO, we compute a ratio between the new policy and the old policy:

$$r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_k}(a_t|s_t)$$

Code:

```
for i in range(self.K_epochs):
    logprobs, state_val, d_entropy = self.policy.evaluation(old_states, old_actions)
    ratios = torch.exp(logprobs-old_logprob.detach())
```

Find surrogate loss(the explanation and code below)

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0, clipping threshold ϵ
for $k = 0, 1, 2, \dots$ **do**
Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k})\right]\right]$$

end for

- Clipped Objective
  - New objective function: let $r_t(\theta) \doteq \pi_\theta(a_t|s_t)/\pi_{\theta_k}(a_t|s_t)$. Then

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k})\right]\right]$$

    where ϵ is a hyperparameter (maybe $\epsilon = 0.2$)
  - Policy update is $\theta_{k+1} = \arg\max_\theta \mathcal{L}_{\theta_k}^{CLIP}(\theta)$

Code:

```
advantages = rewards - state_val.detach()
surrogate1 = ratios*advantages
surrogate2 = torch.clamp(ratios, 1-self.clip, 1+self.clip)*advantages
loss = -torch.min(surrogate1, surrogate2)+0.5*self.MseLoss(state_val, rewards)-0.01*d_entropy
```

Take gradient step:

```
self.optimizer.zero_grad()
loss.mean().backward()
self.optimizer.step()
```

Example:

```
for input, target in dataset:
    optimizer.zero_grad()
    output = model(input)
    loss = loss_fn(output, target)
    loss.backward()
    optimizer.step()
```

Step 6: Copy new weights into old policy then start another epoch again.

- **Anything you have tried?**

1. I tried to run gym environment on colab, but showed error.

   I searched on google and solved it by adding this:

```
!apt-get install xvfb
!pip install pyvirtualdisplay
!pip install Pillow
from pyvirtualdisplay import Display
display = Display(visible=0, size=(1400, 900))
display.start()
```

(since there's no window for colab to display the environment)

- **What hyper parameters have you tried?**

Reference:

https://docs.google.com/spreadsheets/d/1fNVfqgAifDWnTq-4izPPW_CVAUu9FXl3dWkqWIXz04o/edit#gid=0

1. Clipping maintain 0.2

```
ppo = PPO(state_dim, 4, 64, 0.002, (0.9, 0.999), 0.99, 4, 0.2)
```

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

(PPO paper)

2. Discount gamma maintain 0.99
3. Epoch set as 4 (3 is also fine but need more time to converge)
4. Alter learning rate: I tried 0.003, 0.002, 0.0001(learn too slow) and 0.0005(slow) and finally chose 0.002.
5. Minibatches set as 2000 `update_timestep = 2000`
   I also tried 3000, but the result is bad.

- **How you design the reward function?**

Expected discounted reward

The expected discounted reward η is calculated as:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right], \text{ where } s_0 \sim \rho_0(s_0),\ a_t \sim \pi(a_t|s_t),\ s_{t+1} \sim P(s_{t+1}|s_t, a_t).$$

Alternatively, we can compute the reward of a policy using another policy. This lays down the foundation of comparing two policies.

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a).$$

where ρ_π be the (unnormalized) discounted visitation frequencies where $s_0 \sim \rho_0$

$$\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \ldots,$$

Code:

```
for reward, is_terminal in zip(reversed(memory.rewards), reversed(m
    if is_terminal:
        discounted_reward = 0
    discounted_reward = reward + (self.gamma * discounted_reward)
    rewards.insert(0, discounted_reward)
```

(gamma is the discount factor)

- **Observation**

  I found out that although PPO is much efficient (reward gains quickly in the beginning, but it's easily to stuck in the local minimum. Original Actor-critic version is not stable but can get to higher reward quickly.

  I found out that PPO stuck in local optimum quite often (AC version does not).