CSE291 Course Project Part I Report

Wenshuo Zang, Zhaofang Qian

February 26, 2023

The results link: https://drive.google.com/drive/folders/12-EF52mNqfiQgLc0iGvZ2sLgkiIA-N1r?usp=sharing

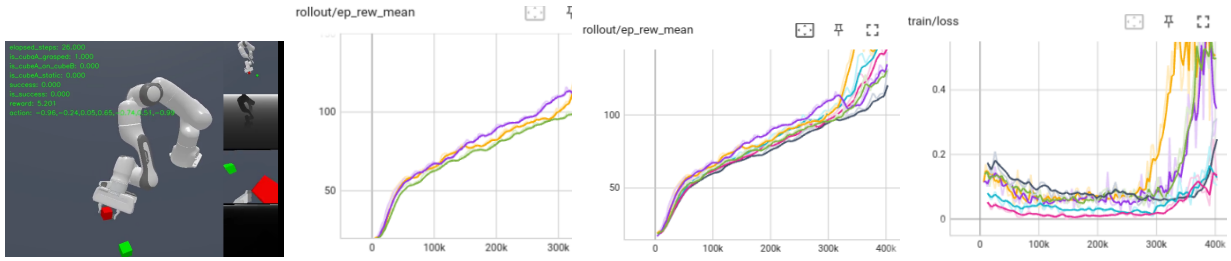
1 LiftCube

In this task, the agent of a seven-joint manipulator is trained by Proximal Policy Optimization(PPO) [1]. We started with the parameters given by the tutorials of Maniskill2, and the results are shown in the first row of table 1. The initial training took a little bit long, around twenty minutes. To increase the speed, the number of environments(env) is set to eight, since they contributes to the same Q-network. But the result doesn't not manifest any improvement. The reason is that the number of steps for each env is equal to total rollout steps divided by number of envs. Each env seemed to be lack of steps to collect enough info for time difference computation. Either increasing the total rollout steps or changing back. To verify the rest of parameters, we decided to only have two environments. Then we tried to increase the batch size, finding that the larger batch size, the higher the slope of mean reward pre epsoide is, indicating that this reward curve would increase faster to reach the goal. Just as shown in fig 1(b), the purple curve is 512 batch size and others are 128 and 256 instead. Hence the 512 batch size was selected.

At this time, we didn't realize the most important parameter, reward or discounted factor, should be tuned first, as instructor mentioned during the class. We only can change γ to impact the rewards. The large factors weighted more future returns, while the small factors focused on more "myoptic" results. We increased γ to 0.95, and got a similar result, which indicates that the large discounted factor failed in converging the neural networks. After a few experiments, when the factor is 0.7, PPO can completely handle the task. In this case, the training loss remained at a low level in blue curves of Fig 1(d). This Python library implements a GAE+PPO method. We wanna to see how the GAE factor λ impacts results. Due to the limitation of samples, the advantage function has to balance the variance and bias. The smaller λ manifests smaller variance but larger bias. We played with different λ values from 0.95 to 0.999 and found the best is the default value. All results are presented in table.1 for the convenience of comparsion.

Test No.	γ	batch_size	env_num	samples	steps	epochs	λ	Success
1	0.85	256	2	400k	4096	15	0.95	0.5
2	0.85	256	8	400k	4096	15	0.95	0.3
3	0.85	512	2	400k	4096	15	0.95	0.4
4	0.95	512	2	400k	4096	15	0.95	0.5
5	0.70	512	2	400k	4096	15	0.95	1.0
6	0.70	512	2	400k	4096	15	0.97	0.9

Table 1: Parameters of PPO in LiftCube-v1

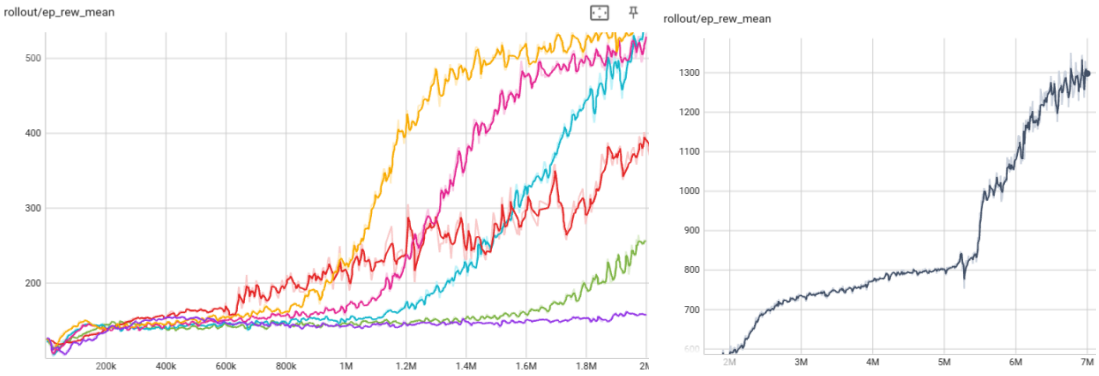


(a) (b) (c) (d)

Figure 1: (a)PPO StackCube in local optima (b) PPO avg_reward various batch sizes (c) PPO avg_reward (d) PPO train loss

2 StackCube

This task consists of three movements, which are lifting the cube, moving to the top of the other, and dropping the cube. After finishing each, the agent would get rewards, but easily stuck in local optima by using PPO. We frequently observed in videos that the manipulator stayed static all the time immediately one of the movements was done. Since the exploration of PPO algorithm relied on action samples correspongding to the most recent stochastic polices, both the initial conditions and the training processes determined the sampled actions' randomness. Then, the policy network gradually lost the randomness and instead exploited the rewards as much as it could. Therefore, another algorithm, Soft Actor-Critic(SAC) [2], was applied to prevent the agent from trapped into local optima.



(a) (b)

Figure 2: (a)SAC Average rewards per eposide (Test 1-6)
(b)SAC Average rewards per eposide of the successful tuning (Test 7)

SAC is a stochastic off-policy method incorporating entropy regularization and clipped double-Q trick. First, we tried the default parameters provided by stable_baseline3 and got a bad result, as shown in the first row of table 2. The most important two key parameters are reward(or discounted factor γ) and update interval (the ratio to env_steps) and every tuning should start with these two. After increasing γ to 0.95, the model doesn't converge at all and the rewards remained relatively stable at a low value, as the green curve shown in figure 2(a). Then, we tried a smaller discounted factor, 0.85 and the eposide mean reward curve increased a little bit. After several trials, the best γ is 0.9, and the comparsion are presented in table 2. Next, we tried different update interval correspongding to the update_freq in table. This parameter controls updating the model every "train_freq" steps. The suitable update interval is 1 after some experiments. Because when increasing the frequency to 2, the rewards didn't not grow gradually, just like the cyan color one in figure 2(a). So far, we have tuned the dominating

hyperparameters. Subsequently, we move to some special parameters of SAC.

The value function of SAC is given by

$$V^{\pi}(s) = E_{\tau\sim\pi}(Q(s_t, a_t) + \alpha H(\pi(\cdot|s_t)))$$

If we write the above in the supervised learning convention, it is equivalent to

$$\mathcal{L}_{actor} = \lambda_1 \mathcal{L}_{reward} + \lambda_2 \mathcal{L}_{entropy}$$

, where $\mathcal{L}_{reward}$ allows actor's policy gains larger and larger returns from critic, and $\mathcal{L}_{entropy}$ increases the entropy of the policy of actor. As we know, the entropy can helps the exploration so we have to balance it with exploitation. Thus, reward scaling is introduced to meet the requirements. The parameter, entropy_coeff is equal to the inverse of reward scaling. When increasing it, the total returns grows faster but suffered from stability issues. From the red curve corresponding to test 3 in figure 2(a), the reward is shaking all the time, indicating too much random exploration. Therefore, we should tune it down. The desired value is 0.08. In this case, the reward incremental is very stable.

Test No.	γ	batch_size	entropy_coeff	samples	update_freq	Success Rate
1	0.70	512	0.08	2M	1	0
2	0.85	512	0.08	2M	2	0
3	0.85	512	0.15	4M	1	0
4	0.95	256	0.08	4M	1	0
5	0.90	512	0.08	4M	1	0
6	0.90	1024	0.08	4M	1	0.1
7	0.90	2048	0.08	7M	1	1.0

Table 2: Parameters of SAC in StackCube-v1

Additionally, we also did the experiments about the batch size. Although the large batch size demonstrates faster growth rates of rewards, as the yellow, pink, cyan curves corresponding to test 4,5,6 in figure 2(a), they finally reached a similar value. So the choice of batch size should be considered in terms of the RAM of our devices. However, we still couldn't get a good results, so the last method is to increase the samples. We increased number of samples to 7 million and finally got the best results, shown in figure 2(b). We can see that there are two slews at 2.5M and 5.5M samples. It means the agent successfully tackled the last two movements mention above.

3 PegInsertion

In this task, we can just use SAC with parameters of StackCube. But the rewards increased slowly and slowly. We tried the 8 million and 10 million samples and got a not high successful rate. We estimated that to achieve 100 percent rate, the samples should be at least 20 million.

4 TurnFaucet

In this task, we can just use SAC with parameters of StackCube. The samples could be around 4 millions and models converge very fast.

5 PushChair

This task is very tricky, and we didn't get any good results. Most of times the rewards are negative. One possible solution we proposed is to train each chari individually with very large sample number. We would like to try it for the future work.

References

[1] Schulman, John and Wolski, Filip and Dhariwal, Prafulla and Radford, Alec and Klimov, Oleg, "Proximal Policy Optimization Algorithms", "https://arxiv.org/abs/1707.06347"

[2] Haarnoja, Tuomas and Zhou, Aurick and Abbeel, Pieter and Levine, Sergey, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor", "https://arxiv.org/abs/1801.01290"