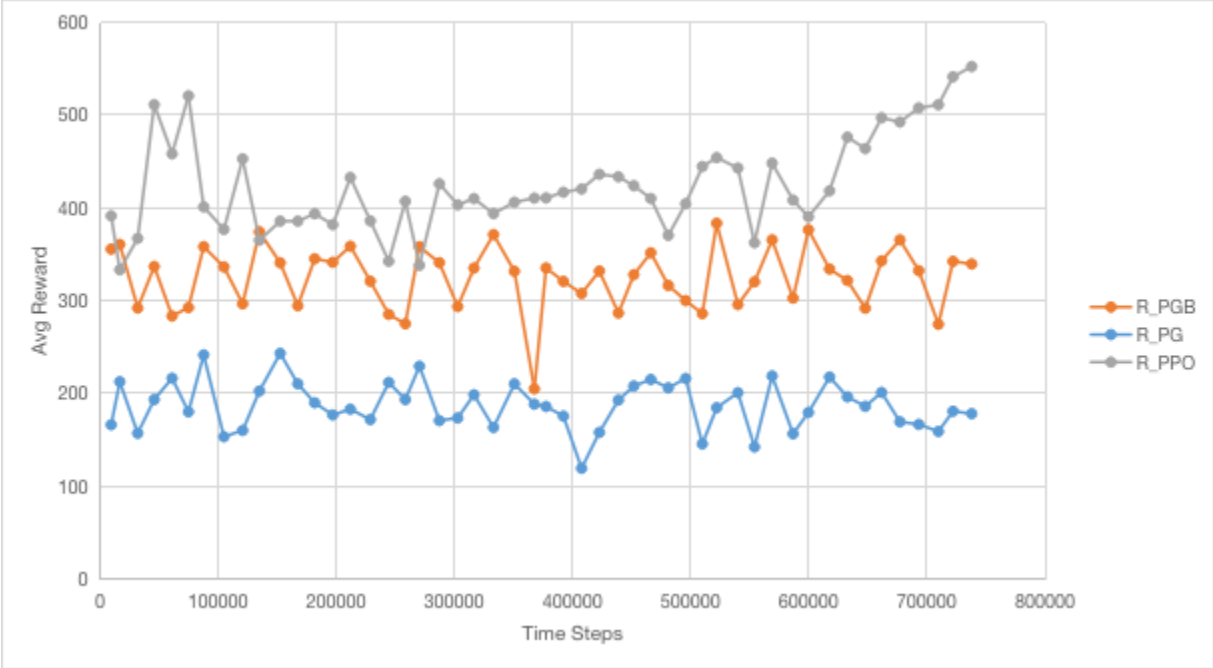
COSC 689 Assignment 1

Zirong Chen

1 Experiment results



Brief explanation:

- I applied highcharts for visualization, which I think will give a clearer figure than matplotlib.
- The X-axis here means the **accumulated time steps** the system took in the environment.
- The Y-axis here means **the average reward of each trajectory** collected during one epoch.
- Each point represents the **[TimeSteps, AvgReward]** tuple during every training epochs.

2 Analysis

Here are some interesting results I found during the experiment:

- Policy Gradient-based models **are very sensitive to the hyper-parameters**. Here is the table of my observations(results might differ due to different settings, I ran several experiments, but I got different results every time. So I just picked one which is able to represent the influences best):

	Vanilla PG(Reward)	PG with Baseline(Reward)
Learning_rate = 1e-3	$\sim$ 70 per epoch	$\sim$ 100 per epoch
Leraning_rate = 1e-5	$\sim$ 150 per epoch	$\sim$ 300 per epoch

- From the experiments, we can easily tell that generally speaking, the performances from PG, PG w/ B, and PPO can be ranked as PG $<$ PG w/ B $<$ PPO. Compared with PG, PG w/ B has **lower variance and better performance**. But from my observation, throughout the whole training process, both PG and PG w/ B are struggling and seem to be trapped at one certain level(PG: $\sim$ 150 AvgReward; PG w/ B: $\sim$ 300 AvgReward). There is barely noticeable evidence that PG or PG w/ B can actually do a better job as the epoch go up(at least in this experiment, with 50 epochs and 20 trajectories per epoch). But PPO is actually 'learning' comparing to PG and PG w/ B, it performs at the same level as PG w/ B does at the beginning, but as the epoch goes up, the performance of **PPO is increasing stably** and I think its performance can be even better if the epochs are bigger.

- After experiments, I find the performance from **PPO can be hurt by adding the number of trajectories per epoch**(at least in this task). Here is the comparison(results might differ due to different settings, I ran several experiments, but I got different results every time. So I just picked one which is able to represent the influences best):

Number of trajectories per epoch	AvgReward from PPO
5	$\sim$ 450 per epoch(almost 550 at the last)
20	$\sim$ 350 per epoch(almost 450 at the last)
25	$\sim$ 150 per epoch

- According to the above table, I ran PPO with 5 trajectories per epoch and 200 epochs to better present its performance. After that, I took an average of 4 AvgRewards and the sum of 4 accumulated TimeSteps as the [AvgReward, TimeSteps] tuple. Even using PPO with 20 trajectories per epoch, we will still get much better results than using PG or PG w/ B.

- The loss of different algorithms differs a lot. Although the loss might not be directly relevant to the AvgReward, I think it is still interesting to list. Here is the table:

	PG	PG w/ B	PPO
The loss at first	~ 5	~ 420 for pi; ~ 13 for val	~ 15
The loss at last	~ 0	~ 400 for pi; ~ 10 for val	~ 12

3 Notes

3.1 How to run the system

- PG: **python hw_A.py –algo pg –epoch [epochs]**
- PG w/ B: **python hw_A.py –algo pgb –epoch [epochs]**
- PPO: **python hw_A.py –algo ppo –epoch [epochs]**
- The num_of_trajectories are set to 20(see the line 23) by default, in order to achieve a better performance of PPO, please set it to 5 or 10.
- The results used for visualization are stored as '**[algo_name].txt**'.
- Running PPO is really time-consuming because I did not do optimization at the matrix multiplications. So it might take 5-10mins to run each epoch.

3.2 Confusions

When trying to improve performance, I find the covariance of action distribution is not updated while doing gradient descent/ascent.