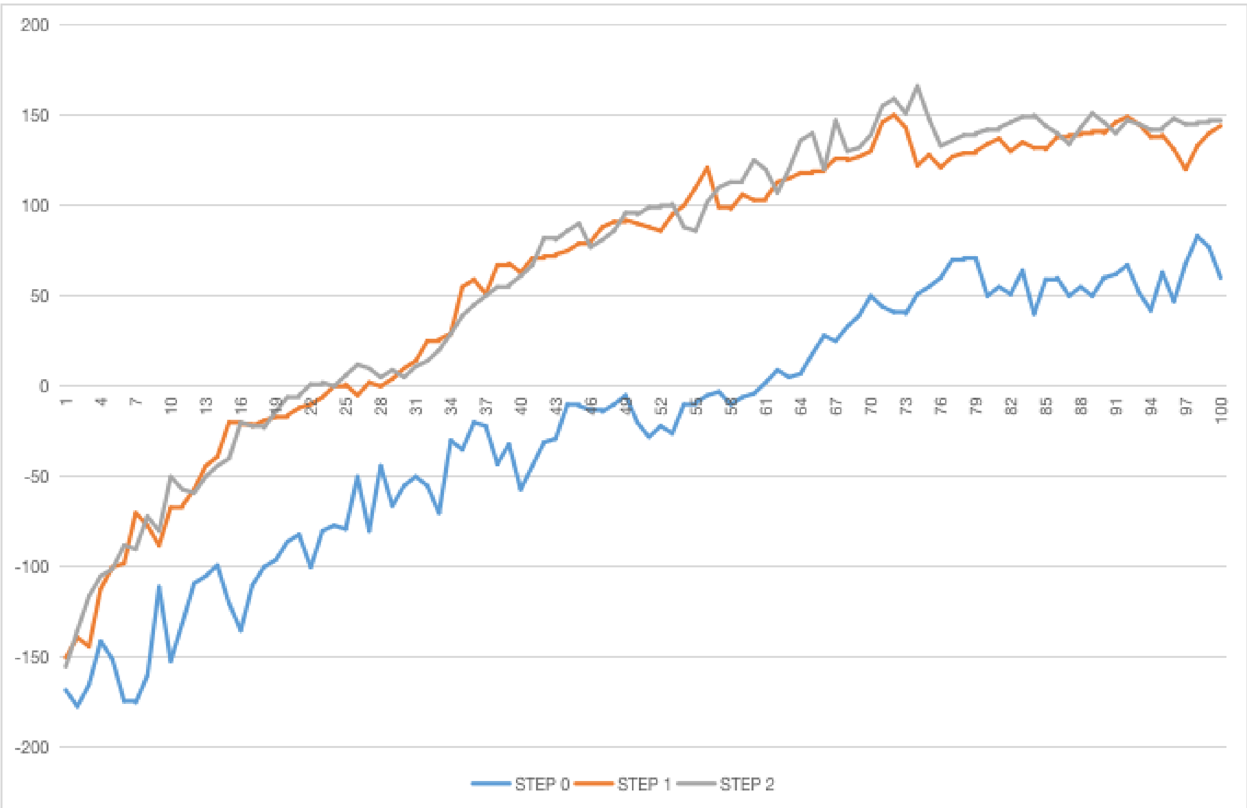
COSC 689 Assignment 2

Zirong Chen

1 Experiment results



	Return[All]	Return[Forward]	Return[Backward]
STEP 0	~ 65	~ 70	~ 48
STEP 1	~ 140	~ 150	~ 140
STEP 2	~ 145	~ 140	~ 145

Brief explanation:

- I applied highcharts for visualization, which I think will give a clearer figure than matplotlib.
- The X-axis here means the **iterations of Meta-Learning** the system took for this task.
- The Y-axis here means **the Average Returns after each Meta-Learning iteration** collected during training.
- Each point represents the **[Epochs, AvgReward]** tuple during every training epoch.
- Due to the limitation of computational resources on my personal laptop, only **100** epochs were experimented with to meet the deadline.

2 Analysis

Here are some interesting results I found during the experiment:

- Running experiments for Meta-Learning is **super time-consuming**, I tried several tricks to make the training faster(tested for STEP1 only). And here are the records of training time reduced each epoch. And in the end, every epoch still took me **about 50 mins** to train (it might be due to my 8G RAM).

	Time per epoch
STEP 0	$\sim$ 40mins
STEP 1	$\sim$ 55mins
STEP 2	$\sim$ 68mins

Trick	Training time each epoch **reduced** by (for STEP 1 only)
No trick applied	N/A
Shared models for Policy and Value nets[1]	$\sim$ 5 mins
Estimate return$*$ from the end[2]	$\sim$ 3 mins

[1] Trick 1 can be found at **Policy_agent.py** from line 29 to line 61.

$*$ This trick was not implemented for the whole process of estimation due to limited time. [2] Trick 2 can be found at **Meta_learner.py** from line 112 to line 118.

- Hyperparameter settings:
  - Inner learning_rate: 0.1
  - Outer learning_rate: 1e-4
  - Discount factor: 0.99
  - Number of trajectories: 20
  - Clip ratio: 0.2
  - Lambda(PPO): 0.9
  - c1(PPO): 0.5

After experiments, I find that hyperparameters, especially leraning_rates, can affect the results a lot. Thus, Meta-Learning is super sensitive to hyperparameter settings. Here are some interesting findings:

Inner learning_rate	Impacts(for STEP 0 only)
1e-5	Learning slowly and only gained $\sim$ 40 rewards after 60 epochs[1]
1e-3	Learning a little bit more efficient, $\sim$ 50 achieved[2]
0.1	Seem unreasonable, but this one works the best $\sim$ 65 achieved

[1] I manually stopped the experiment once the returns were dropping continuously in three epochs.

[2] This is the most recent return I got, my laptop ran out of battery at that time, and that was after 81 epochs. According to its trend, it might continue to increase.

Outer learning_rate	Impacts(for STEP 0 only)
0.1	This one is the worst, the performance increased sharply after 10 epochs($\sim$ 60) and began to drop($\sim$ 30 after 60 epochs).
0.01	Although smaller learning_rate can avoid bigger stepsize during gradient descent/ascent, this one was still too big for this task.
1e-4	This one is the most proper one after the first two attempts. And it gave $\sim$ 65 at the end, and showed a stable increase in returns.

Due to the limited time, I could not do experiments more specifically in hyperparameter settings.

- Running time taken for different numbers of steps can be found as STEP 0 $<$ STEP 1 $<$ STEP 2. And the overall performance can be found as STEP 0 $<<$ STEP 1 $\approx <$ STEP 2. So I would say STEP 2 is not very efficient because of the time spent. However, there do exist some improvements in STEP 2 compared with STEP 1. And STEP 2 and 1 converges faster than STEP 0.

3 Notes

3.1 How to run the system

- STEP 0: **python hw2_A.py –num_of_steps 0**
- STEP 1: **python hw2_A.py –num_of_steps 1**
- STEP 2: **python hw2_A.py –num_of_steps 2**
- Running this work is really time-consuming.
- Best policies(98th STEP 0; 138th STEP 1; 150th STEP 2) are stored.

3.2 Confusions

- The learning_rate is fixed, as can be found at **update_parameters()** in **policy_agent.py**. So I wonder if I can try dynamic learning rates in this task. Since the returns are still increasing at the end, a dynamic learning rate might lead to a better result.