

# CSE5100 Homework 3

Weikai Rao 519139, Longhan Lin 529150

November 2, 2025

## 1 Problem 1 Policy Gradient

### 1.1 Graphs

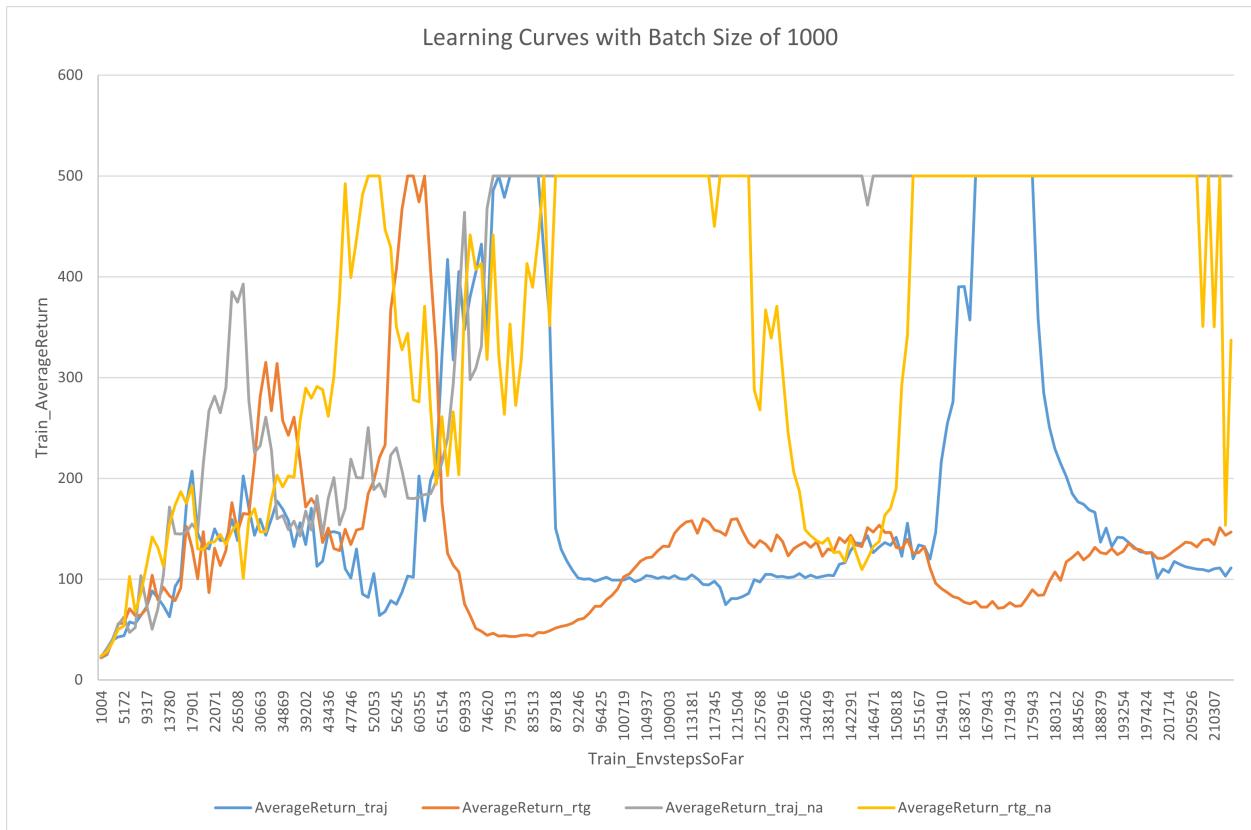


Figure 1: The learning curves (average return vs. number of environment steps) for the experiments running with batch size of 1000

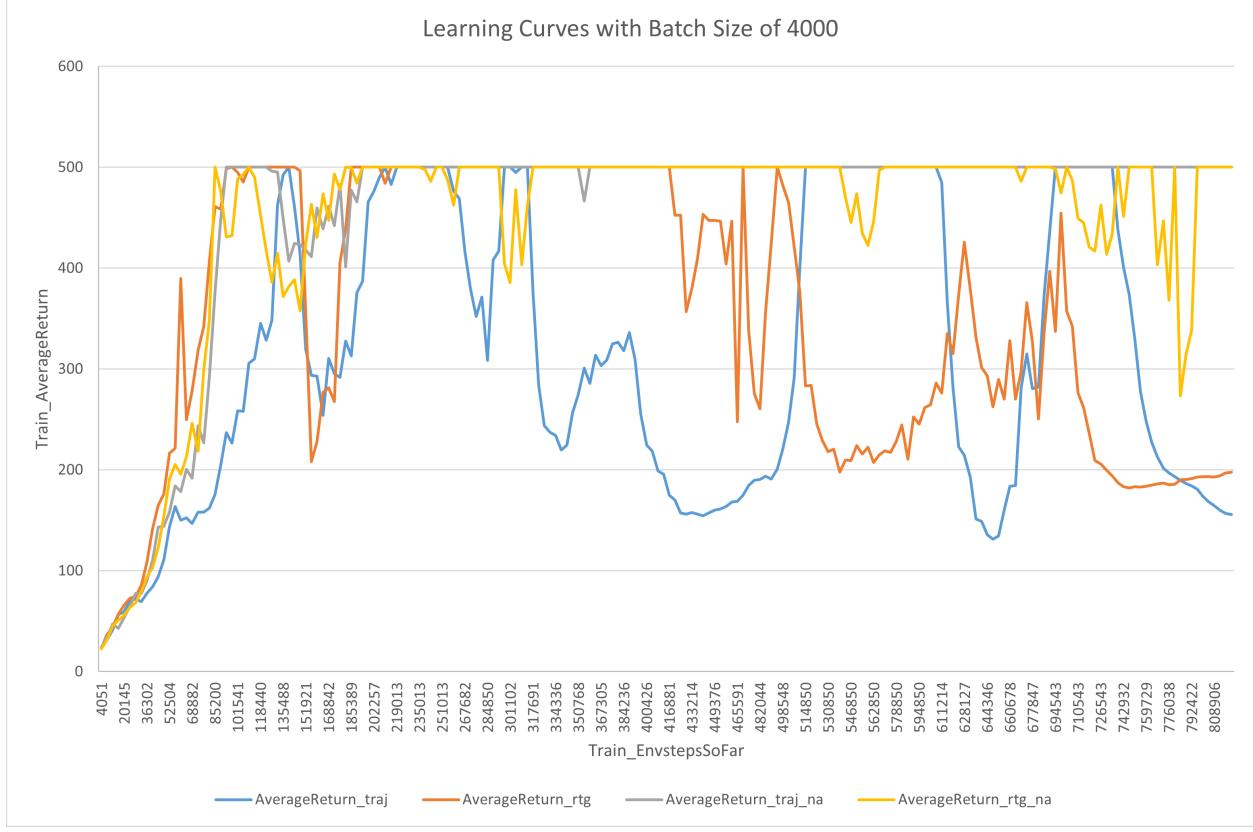


Figure 2: The learning curves for the experiments running with batch size of 4000

## 1.2 Questions

Command lines used:

- Batch size 1000:

```
python run.py --env_name CartPole-v1 -n 200 -b 1000 --exp_name cartpole
python run.py --env_name CartPole-v1 -n 200 -b 1000 -rtg --exp_name cartpole_rtg
python run.py --env_name CartPole-v1 -n 200 -b 1000 -na --exp_name cartpole_na
python run.py --env_name CartPole-v1 -n 200 -b 1000 -rtg -na --exp_name cartpole_rtg_na
```

- Batch size 4000:

```
python run.py --env_name CartPole-v1 -n 200 -b 4000 --exp_name cartpole_lb
python run.py --env_name CartPole-v1 -n 200 -b 4000 -rtg --exp_name cartpole_lb_rtg
python run.py --env_name CartPole-v1 -n 200 -b 4000 -na --exp_name cartpole_lb_na
python run.py --env_name CartPole-v1 -n 200 -b 4000 -rtg -na --exp_name cartpole_lb_rtg_na
```

### 1.2.1 Which value estimator has better performance without advantage normalization: the trajectory-centric one, or the one using reward-to-go? Why?

The one using reward-to-go has better performance without advantage normalization. Because RTG exploits causality and reduces the variance of the policy gradient, so it learns faster and reaches 500 quicker.

### **1.2.2 Did advantage normalization help?**

Yes. The average return without normalization are much more unstable, and it drops down below 500 frequently. But with normalized advantage, the two learning curves stay at 500 most of the time after reaching 500.

### **1.2.3 Did the batch size make an impact?**

Yes, the bigger batch size makes the learning curves reach 500 faster.

## 2 Problem 2 Neural Network Baseline

### 2.1 Graphs

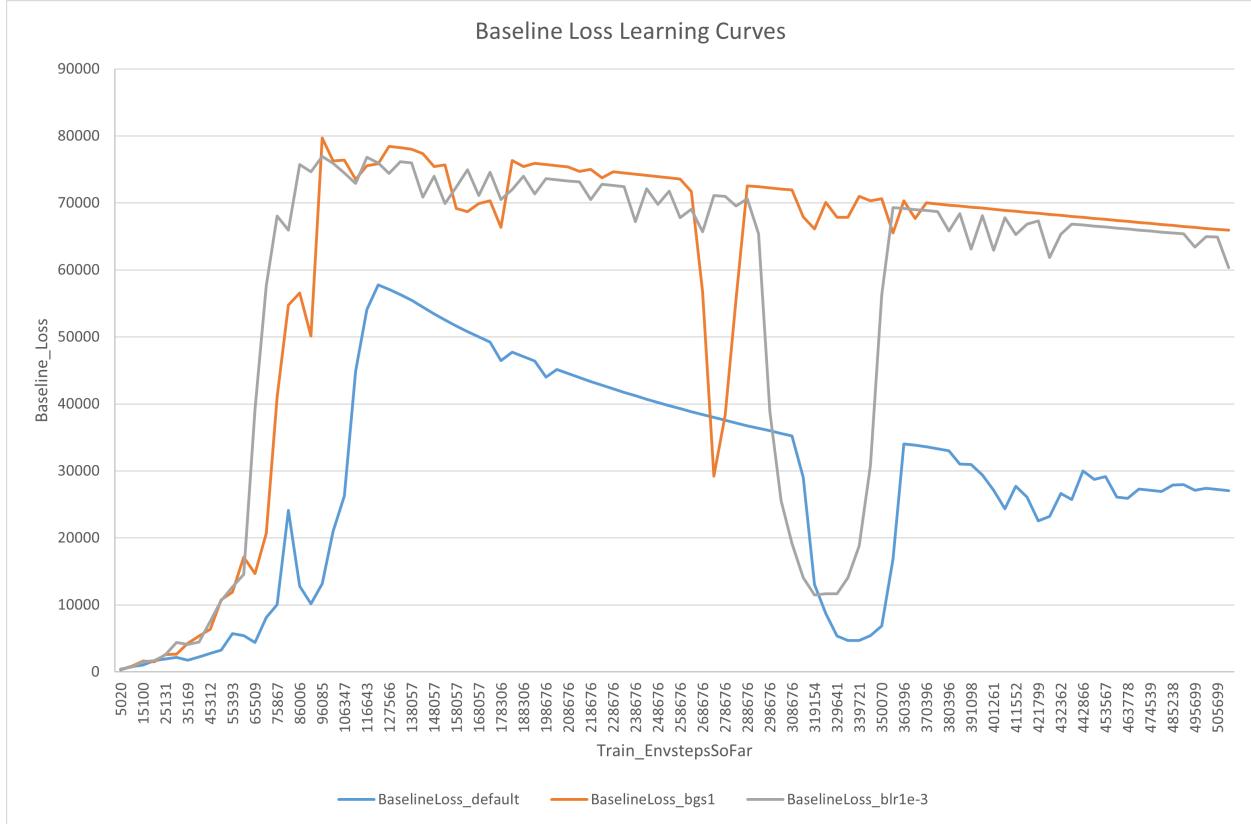


Figure 3: The baseline loss learning curves for the experiments running with batch size of 5000

Command lines used:

```
# baseline off
python run.py --env_name CartPole-v1 -n 100 -b 5000 -rtg --exp_name cartpole_rtg_no_baseline
# baseline on (default bgs=5, blr=5e-3)
python run.py --env_name CartPole-v1 -n 100 -b 5000 -rtg -na --use_baseline --exp_name
↪ cartpole_na_rtg_baseline
# fewer baseline steps (bgs=1)
python run.py --env_name CartPole-v1 -n 100 -b 5000 -rtg -na --use_baseline -bgs 1 --exp_name
↪ cartpole_na_rtg_baseline_bgs1
# lower baseline learning rate (blr=1e-3)
python run.py --env_name CartPole-v1 -n 100 -b 5000 -rtg -na --use_baseline -blr 1e-3 --exp_name
↪ cartpole_na_rtg_baseline_blr1e-3
```

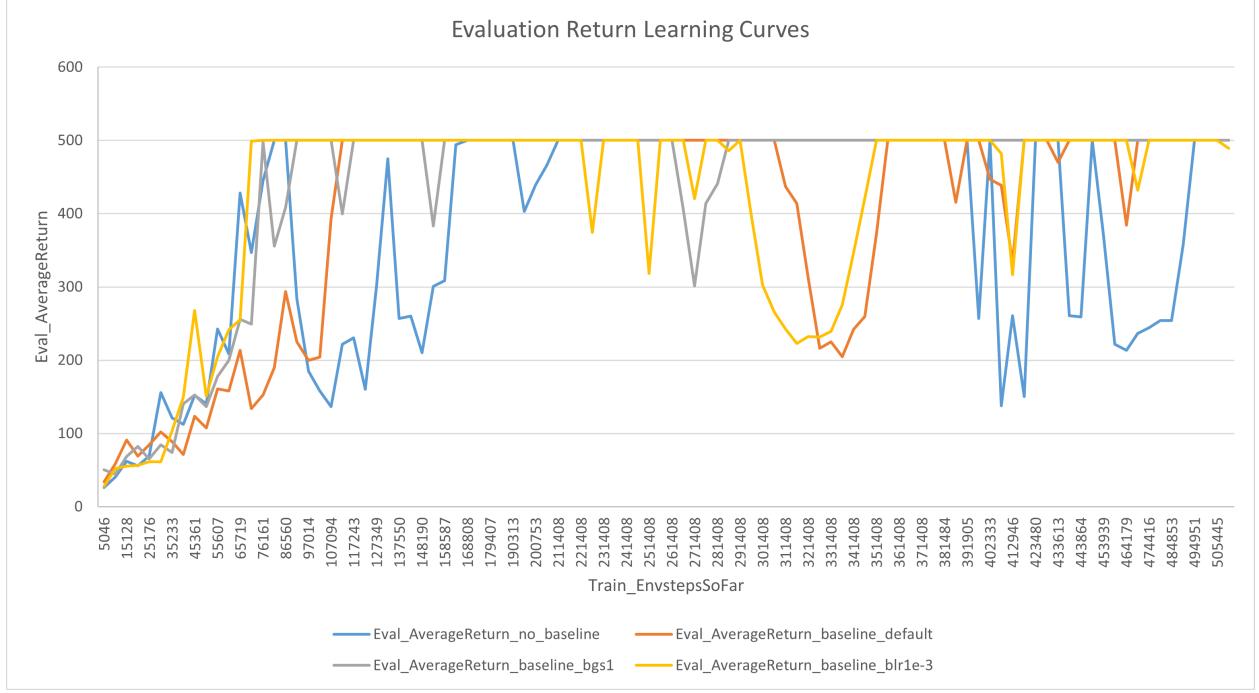


Figure 4: The evaluation return learning curves for the experiments running with batch size of 5000

## 2.2 Questions

**2.2.1 Run another experiment with a decreased number of baseline gradient steps (-bgs in command line) and/or baseline learning rate (-blr in command line). How does this affect (a) the baseline learning curve and (b) the performance of the policy?**

- (a) Fewer baseline steps (bgs=1) and lower baseline learning rate (blr=1e-3) both make the baseline loss stays higher and drops slower than the default baseline (bgs=5, blr=5e-3).
- (b) Fewer baseline steps (bgs=1) and lower baseline learning rate (blr=1e-3) let the variance higher, they reach 500 faster but they drops and fluctuate more frequently than the default baseline.

**2.2.2 How does the command line argument -na influence the performance? Why is that the case? (5 pts)**

The command line argument -na normalize the advantage and help stabilize the learning curve. Because it will lower the variance and gradient scale and make the learning more stable.