

EXPERIMENT 1

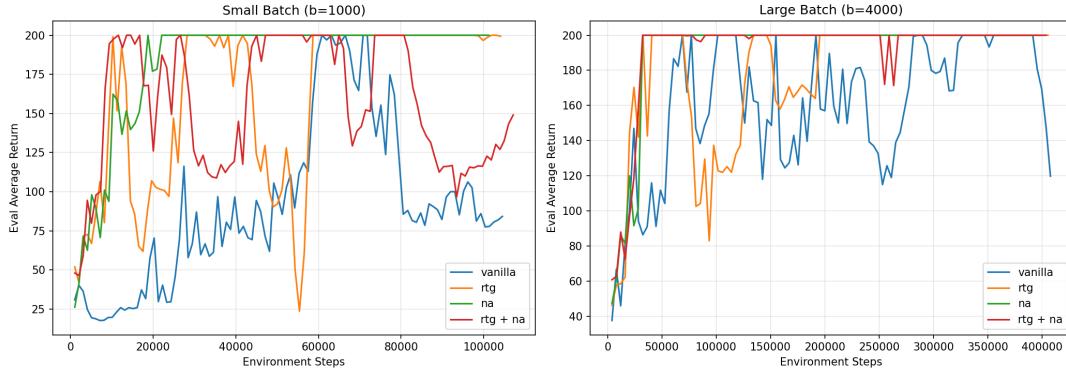


Figure 1: Eval Return vs. Env Steps

1. Without **advantage normalization**, the **reward-to-go** value estimator performs better than the **trajectory-centric** one.
2. **advantage normalization** help the policy to achieve the same performance in **less environment step**.
3. Actually the training with **larger batch size** achieves the same performance in **more environment steps** but **less training iteration steps**.
4. The command line configuration i use is totally the same as the one given in homework document.

EXPERIMENT 2

1.

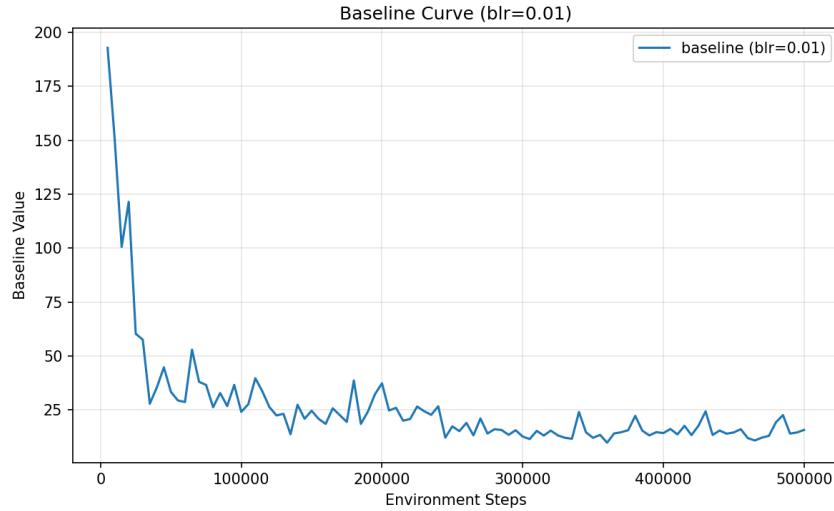


Figure 2: Baseline Loss vs. Env Steps with baseline learning rate=0.01

2.

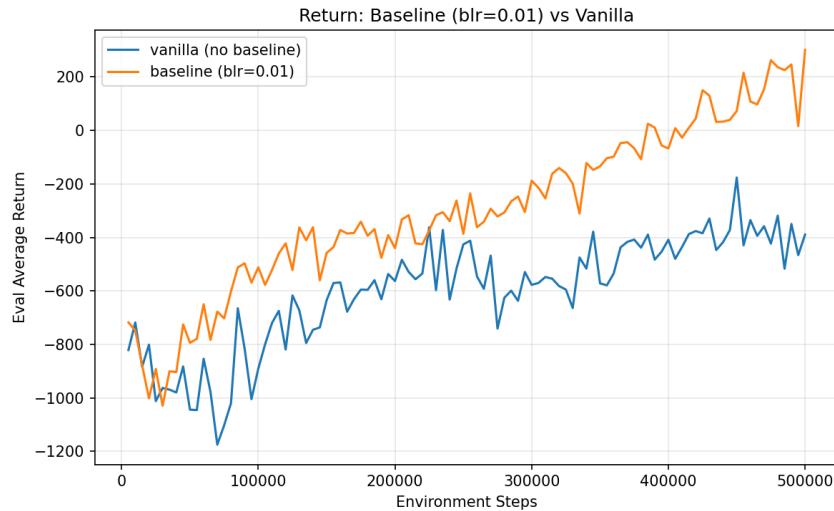
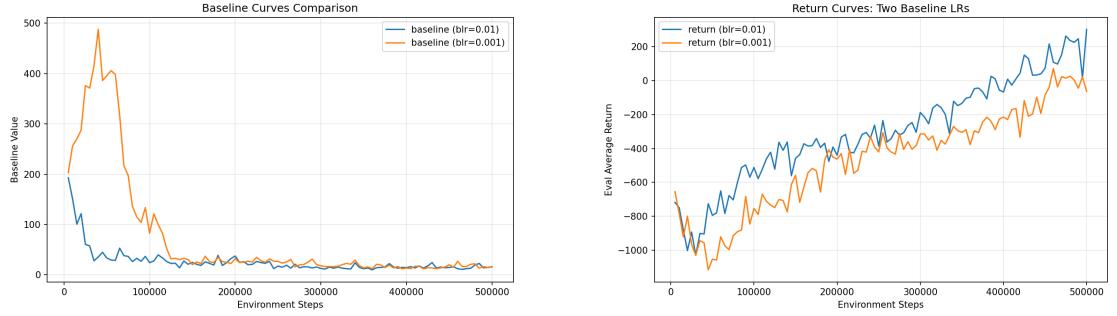


Figure 3: Average Return vs. Env Steps with baseline learning rate=0.01

3.



(a) Baseline Loss with different baseline learning rate

(b) Baseline Return with different baseline learning rate

Compared to **baseline learning rate = 0.01**, training with **baseline learning rate 0.001** causes slower **baseline loss converging** and slower /textbf{average} return increasing.

EXPERIMENT 3

1.

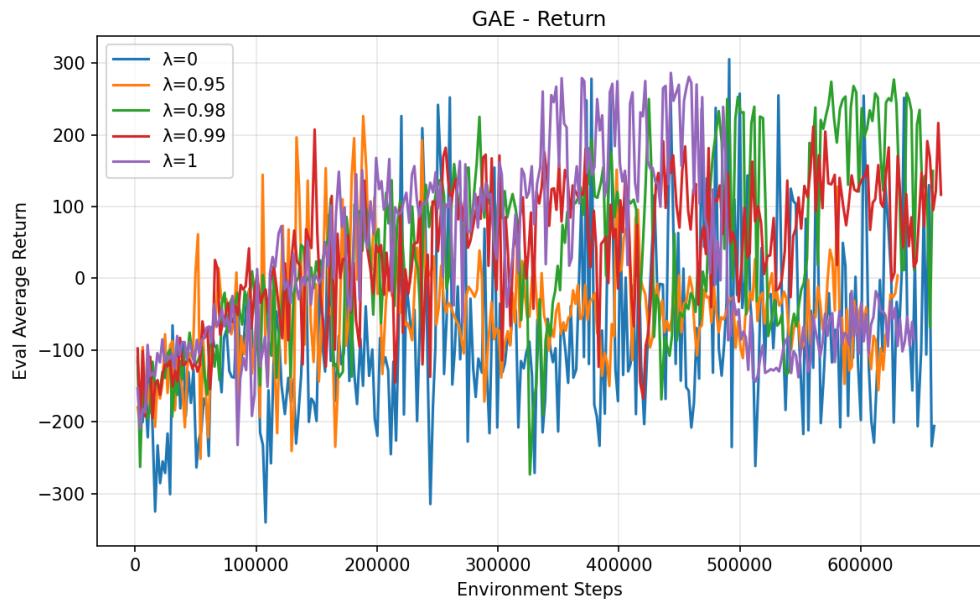


Figure 5: Average Return vs. Env Steps with different GAE lambda

Increasing λ makes the policy to perform better, but too big λ makes the training unstable in the later stage.

2. $\lambda = 0$ correspond to $\hat{A}(s_t, a_t) = r(s_t, a_t) + \gamma \hat{V}(s_{t+1}) - \hat{A}(s_t)$, which means we totally use prediction from critic as the estimation of state value. In the beginning of our training, the critic prediction is totally biased, which causes the low performance of the policy. But the prediction of critic network has low variance, which leads to lower oscillation.

$\lambda = 1$ correspond to $\hat{A}(s_t, a_t) = \sum_{t'=t}^T \gamma^{t'-t} r(s_{t'}, a_{t'}) - \hat{V}(s_t)$, which means we totally use the following steps' rewards from environment to estimate the advantage of a_t . This causes high variance because every step is stochastic but unbiased.

In the figure 5, the blue curve ($\lambda = 0$)'s *eval_return* struggles near -100 because of huge bias caused by predicting the value totally. And other four curves obviously perform better than the blue one. However, due to the high variance from stochastic step reward, sometimes the return falls down violently.

EXPERIMENT 4

1. Hyperparameters

parameter	value
iteration	100
batch size	500
learning rate	0.01
discount	0.99
GAE lambda	0.98
reward to go	true
use baseline	true
baseline learning rate	0.01
baseline gradient steps	5
n layers	2
layer size	64

Table 1: Hyperparameters

2. Return Curves

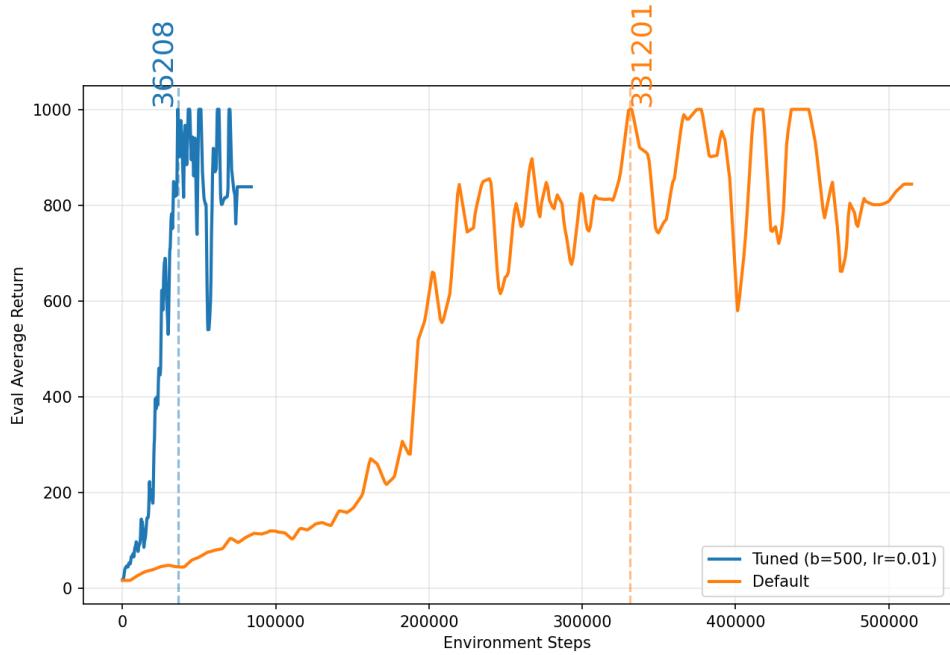


Figure 6: Eval Average Return-Env Steps(default VS. self-define)