Part 3

1.Average return for cartpole experiments

Figure 1: Experiments run with cartpole

Figure 2: Experiments run with large batch cartpole

Did advantage normalization help?
- Yes, normalizing advantages allows the advantages to stabilize, keeping the average return stable once it peaks around 250.

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

Did batch Size make an impact
it made the reward converge faster, so it helps with making the average return more reliable.

Part 4.2

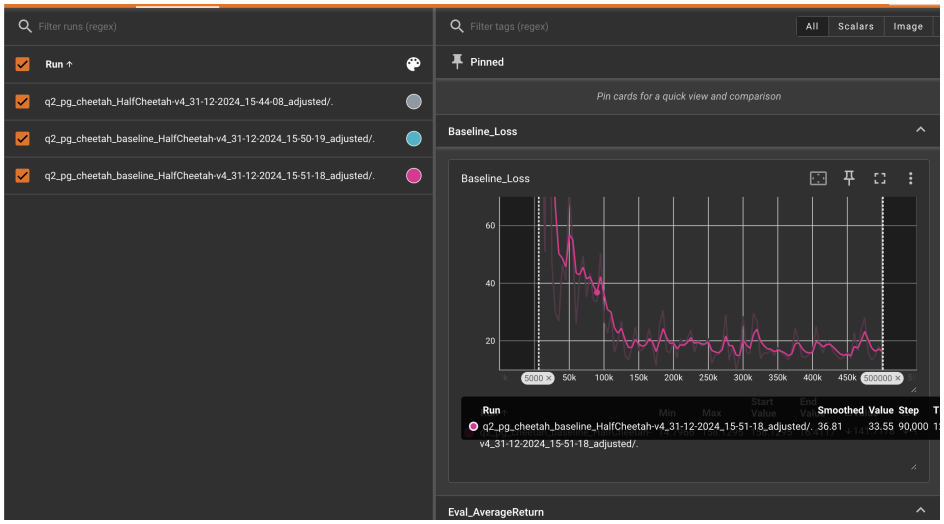

Figure 3: Baseline Loss for Cheetah With and without baseline

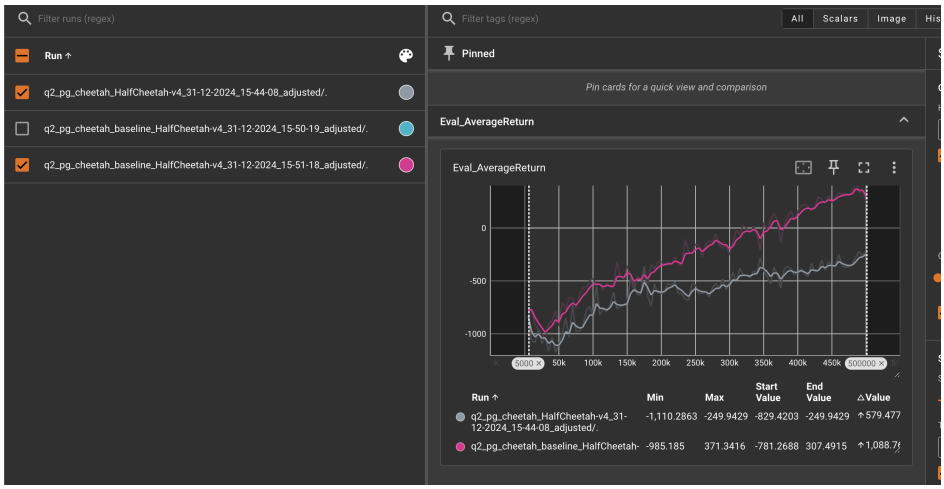

Figure 4: Return for Cheetah with and without baseline

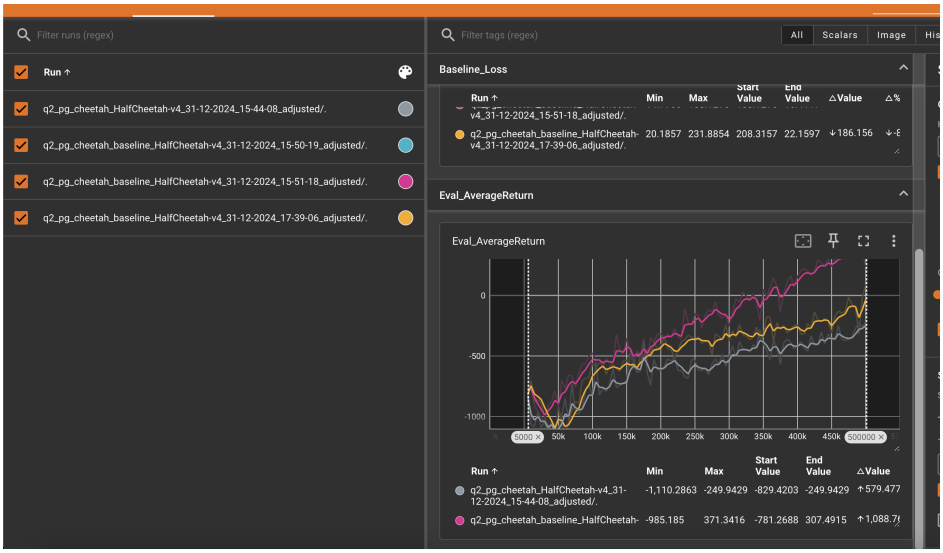

Figure 5: Return for Cheetah with higher blr and bgs (purple)

yellow line is case using baseline, lower blr and fewer bgs, as you can see it has lower average return compared to the baseline with higher blr and bgs (purple). Grey is no baseline

Part 5: Implementing GAE

```
\python cs285/scripts/run_hw2.py \
--env_name LunarLander-v2 --ep_len 1000 \
--discount 0.99 -n 300 -l 3 -s 128 -b 2000 -lr 0.001 \
--use_reward_to_go --use_baseline --gae_lambda 0 \
--exp_name lunar_lander_lambda0

python cs285/scripts/run_hw2.py \
--env_name LunarLander-v2 --ep_len 1000 \
--discount 0.99 -n 300 -l 3 -s 128 -b 2000 -lr 0.001 \
--use_reward_to_go --use_baseline --gae_lambda 0.95 \
--exp_name lunar_lander_lambda0.95

python cs285/scripts/run_hw2.py \
--env_name LunarLander-v2 --ep_len 1000 \
--discount 0.99 -n 300 -l 3 -s 128 -b 2000 -lr 0.001 \
--use_reward_to_go --use_baseline --gae_lambda 0.98 \
--exp_name lunar_lander_lambda0.98

python cs285/scripts/run_hw2.py \
--env_name LunarLander-v2 --ep_len 1000 \
--discount 0.99 -n 300 -l 3 -s 128 -b 2000 -lr 0.001 \
--use_reward_to_go --use_baseline --gae_lambda 0.99 \
--exp_name lunar_lander_lambda0.99

python cs285/scripts/run_hw2.py \
--env_name LunarLander-v2 --ep_len 1000 \
--discount 0.99 -n 300 -l 3 -s 128 -b 2000 -lr 0.001 \
--use_reward_to_go --use_baseline --gae_lambda 1 \
```

```
--exp_name lunar_lander_lambda1
```

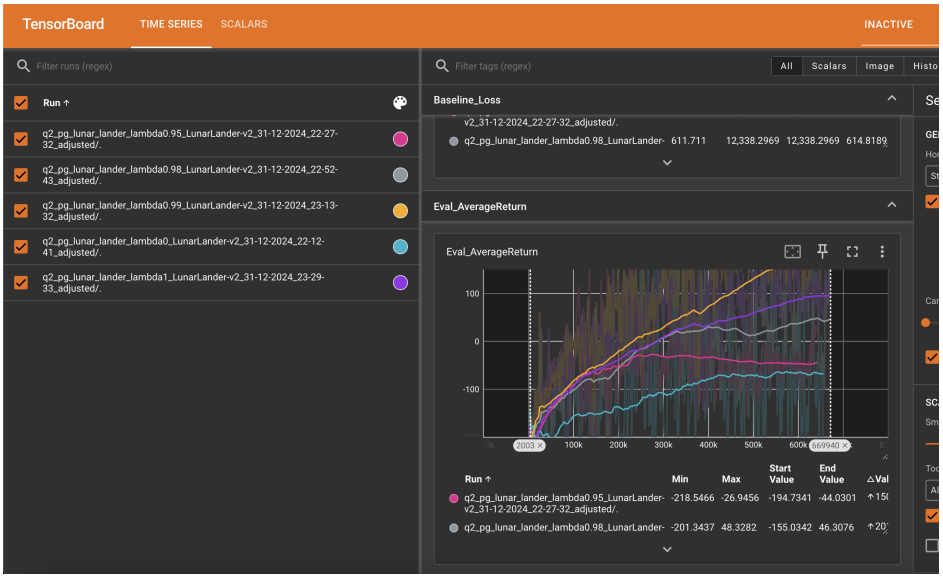

Figure 6: Effect of λ on LunarLander EvalReturn. Orange is $\lambda = 0.99$, blue is $\lambda = 0$

higher lambda represents weighing the reward of future actions in calculating the advantage the same as the current reward. if lambda is 0 that means we only consider the current action and its reward set whereas if lambda is 1 we weigh future rewards at a timestep the same as the current reward

Part 6: Hyperparameters and Sample Efficiency

Default, yellow on the graph

```
python cs285/scripts/run_hw2.py --env_name InvertedPendulum-v4 -n 100 \
--exp_name pendulum_default_s5 \
-rtg --use_baseline -na \
--batch_size 5000 \
--seed 5
```

lr 0.02, grey on the graph

```
python cs285/scripts/run_hw2.py \
--env_name InvertedPendulum-v4 --exp_name pendulum_0.02lr \
-n 100 --batch_size 5000 --seed 5 \
-lr 0.02 --discount 0.98 \
-rtg --use_baseline -na
```

lr 0.04, blue on the graph

```
python cs285/scripts/run_hw2.py \
--env_name InvertedPendulum-v4 --exp_name pendulum_0.04lr \
-n 100 --batch_size 6000 --seed 5 \
-lr 0.04 --discount 0.97 \
-rtg --use_baseline -na
```

lr 0.03, red on the graph

```
python cs285/scripts/run_hw2.py \
--env_name InvertedPendulum-v4 --exp_name pendulum_0.03lr \
-n 100 --batch_size 6000 --seed 5 \
-lr 0.03 --discount 0.98 \
-rtg --use_baseline -na
```

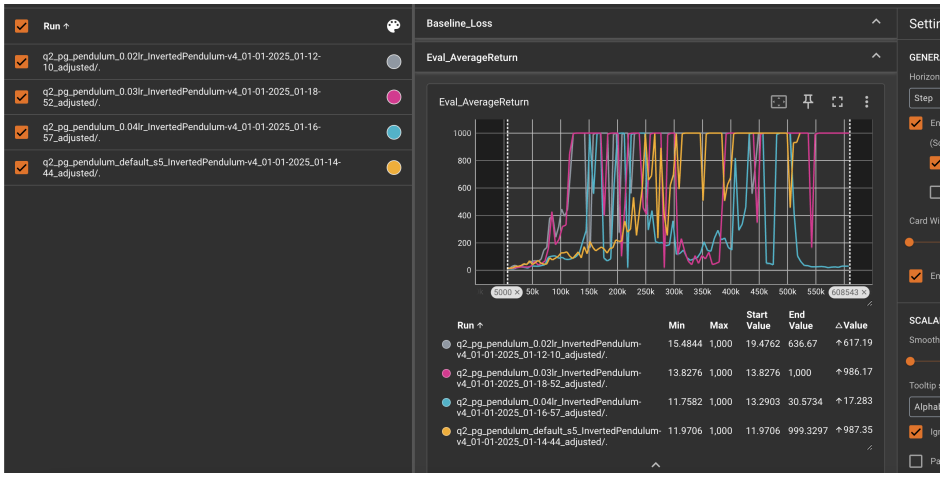

Figure 7: Parameters for optimal performance on Inverted Pendulum