Assignment 2: Policy Gradient

Andrew ID: `mukaiy`
NOTE: Please do **NOT** change the sizes of the answer blocks or plots.

5 Small-Scale Experiments

5.1 Experiment 1 (Cartpole) – [25 points total]

5.1.1 Configurations

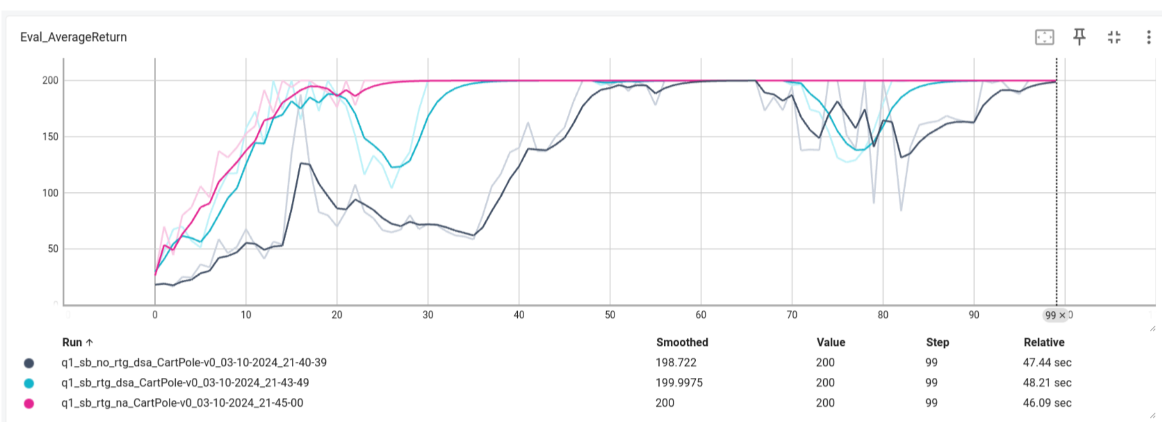
Q5.1.1

```
python rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
    -dsa --exp_name q1_sb_no_rtg_dsa

python rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
    -rtg -dsa --exp_name q1_sb_rtg_dsa

python rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
    -rtg --exp_name q1_sb_rtg_na

python rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 5000 \
    -dsa --exp_name q1_lb_no_rtg_dsa

python rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 5000 \
    -rtg -dsa --exp_name q1_lb_rtg_dsa

python rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 5000 \
    -rtg --exp_name q1_lb_rtg_na
```

5.1.2 Plots

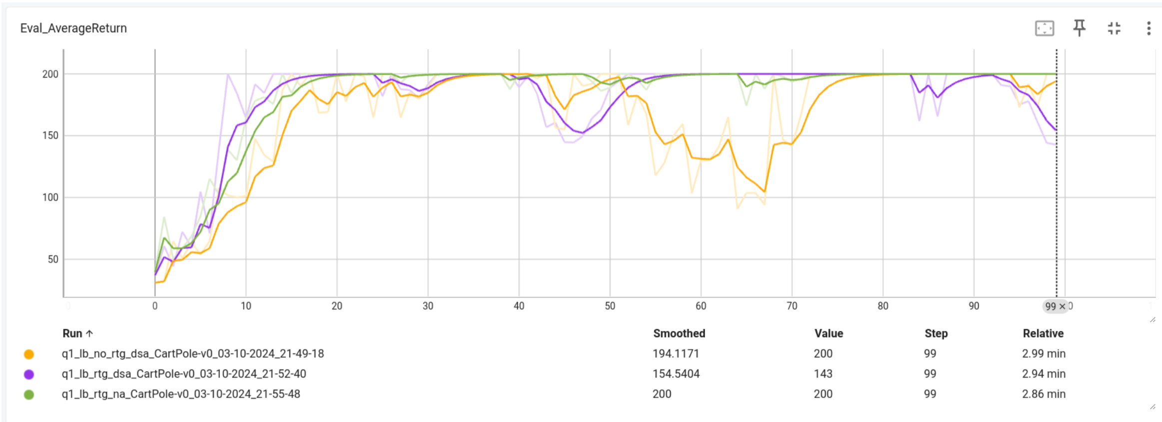
5.1.2.1 Small batch – [5 points]

Q5.1.2.1



5.1.2.2 Large batch – [5 points]

Q5.1.2.2



5.1.3 Analysis

5.1.3.1 Value estimator – [5 points]

Q5.1.3.1

Obviously reward-to-go because of causality and smaller variance.

5.1.3.2 Advantage standardization – [5 points]

Q5.1.3.2

Yes, it does help. It helps to stabilize the learning process by reducing the variance of the advantage estimates.
One thing I don't understand is why changing the bias has no significant impact on training, since standardization changes the mean. Each reward is a function of both state and action, so moving the mean shouldn't be just like adding a state-dependent bias, e.g., $V(s)$, right?
I guess it's because this process only "slightly" biases the advantage estimates, pretty much like the effect of not having accurate $V(s)$ in the actor-critic method.

5.1.3.3 Batch size – [5 points]

Q5.1.3.3

I would say that batch size marginally helps to improve the policy faster, and I think the improvement is non-linear, which is why we don't see the time-to-plateau gets $\frac{1}{5}$ed.
But more data does help the experiment without reward-to-go to converge faster, since it typically needs more rollouts, i.e., more state/action/reward tuples, because it typically has higher variance.

5.2 Experiment 2 (InvertedPendulum) – [15 points total]

5.2.1 Configurations – [5 points]

Q5.2.1

```
python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 \
    --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 10000 -lr 0.001 -rtg \
    --exp_name q2_b_10000_r_0.001

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 \
    --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 1000 -lr 0.001 -rtg \
    --exp_name q2_b_1000_r_0.001

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 \
    --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 1000 -lr 0.01 -rtg \
    --exp_name q2_b_1000_r_0.01

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 \
    --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 1000 -lr 0.05 -rtg \
    --exp_name q2_b_1000_r_0.05

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 \
    --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 1000 -lr 0.1 -rtg \
    --exp_name q2_b_1000_r_0.1

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 \
    --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 100 -lr 0.1 -rtg \
    --exp_name q2_b_100_r_0.1

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 \
    --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 2000 -lr 0.01 -rtg \
    --exp_name q2_b_2000_r_0.01

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 \
    --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 500 -lr 0.01 -rtg \
    --exp_name q2_b_500_r_0.01

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 \
    --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 800 -lr 0.01 -rtg \
    --exp_name q2_b_800_r_0.01
```

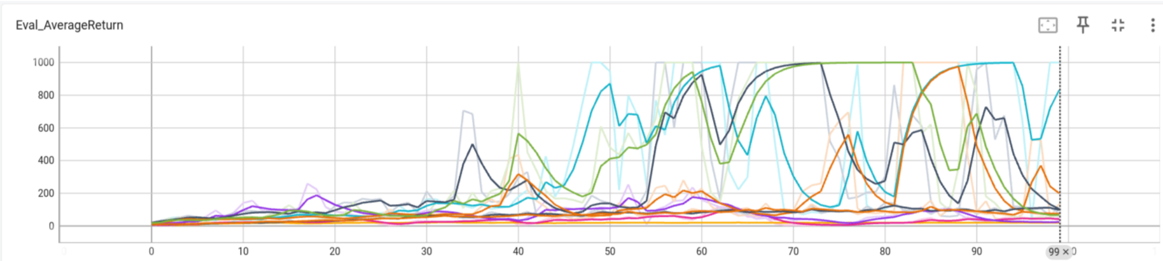
5.2.2 smallest b* and largest r* (same run) – [5 points]

Q5.2.2

b* = 500
r* = 0.01

5.2.3 Plot – [5 points]

Q5.2.3



Run ↑	Smoothed	Value	Step	Relative
q2_b_10000_r_0.001_InvertedPendulum-v4_03-10-2024_22-39-20	97.6726	76.8333	99	6.704 min
q2_b_1000_r_0.001_InvertedPendulum-v4_03-10-2024_22-37-33	74.8903	72.8333	99	56.62 sec
q2_b_1000_r_0.01_InvertedPendulum-v4_03-10-2024_22-47-56	832.143	1,000	99	1.049 min
q2_b_1000_r_0.05_InvertedPendulum-v4_03-10-2024_22-51-43	24.3166	26.875	99	55.25 sec
q2_b_1000_r_0.1_InvertedPendulum-v4_03-10-2024_22-50-01	19.4487	18.9545	99	55.48 sec
q2_b_100_r_0.1_InvertedPendulum-v4_03-10-2024_22-49-13	42.6378	38.1818	99	21.05 sec
q2_b_2000_r_0.01_InvertedPendulum-v4_03-10-2024_22-59-32	103.7933	61.1111	99	1.636 min
q2_b_500_r_0.01_InvertedPendulum-v4_03-10-2024_22-53-51	66.8982	66.8571	99	47.76 sec
q2_b_800_r_0.01_InvertedPendulum-v4_03-10-2024_22-55-17	199.8111	135.6667	99	52 sec

7 More Complex Experiments

7.1 Experiment 3 (LunarLander) – [10 points total]

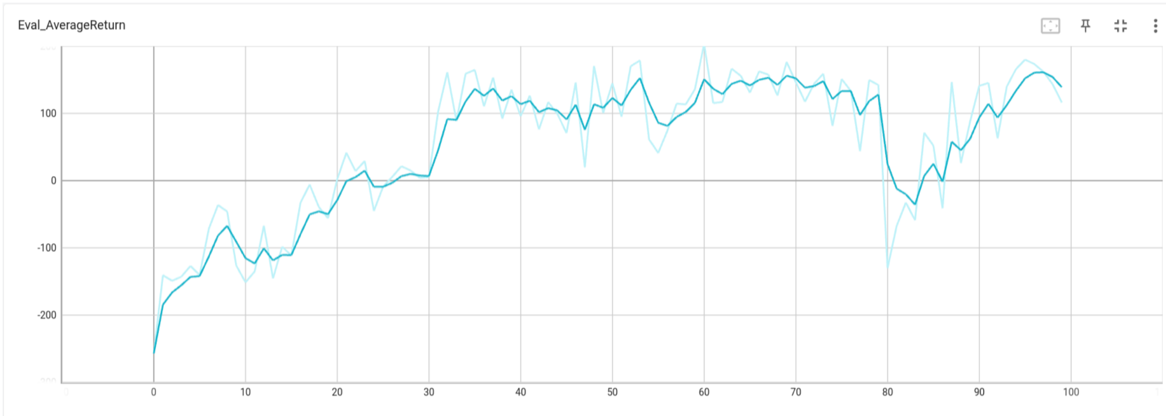
7.1.1 Configurations

Q7.1.1

```
python rob831/scripts/run_hw2.py \
    --env_name LunarLanderContinuous-v4 --ep_len 1000 \
    --discount 0.99 -n 100 -l 2 -s 64 -b 10000 -lr 0.005 \
    --reward_to_go --nn_baseline --exp_name q3_b10000_r0.005
```

7.1.2 Plot – [10 points]

Q7.1.2



7.2 Experiment 4 (HalfCheetah) – [30 points]

7.2.1 Configurations

Q7.2.1

```
python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 10000 -lr 0.02 \
    --exp_name q4_search_b10000_lr0.02
python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 30000 -lr 0.02 -rtg \
    --exp_name q4_search_b30000_lr0.02_rtg
python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 50000 -lr 0.02 --nn_baseline \
    --exp_name q4_search_b50000_lr0.02_nnbaseline
python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 10000 -lr 0.01 \
    --exp_name q4_search_b10000_lr0.01
python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 30000 -lr 0.01 -rtg \
    --exp_name q4_search_b30000_lr0.01_rtg
python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 50000 -lr 0.01 --nn_baseline \
    --exp_name q4_search_b50000_lr0.01_nnbaseline
python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 10000 -lr 0.005 \
    --exp_name q4_search_b10000_lr0.005
python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 30000 -lr 0.005 -rtg \
    --exp_name q4_search_b30000_lr0.005_rtg
python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 50000 -lr 0.005 --nn_baseline \
    --exp_name q4_search_b50000_lr0.005_nnbaseline
```

7.2.2 Plot – [10 points]

Q7.2.2

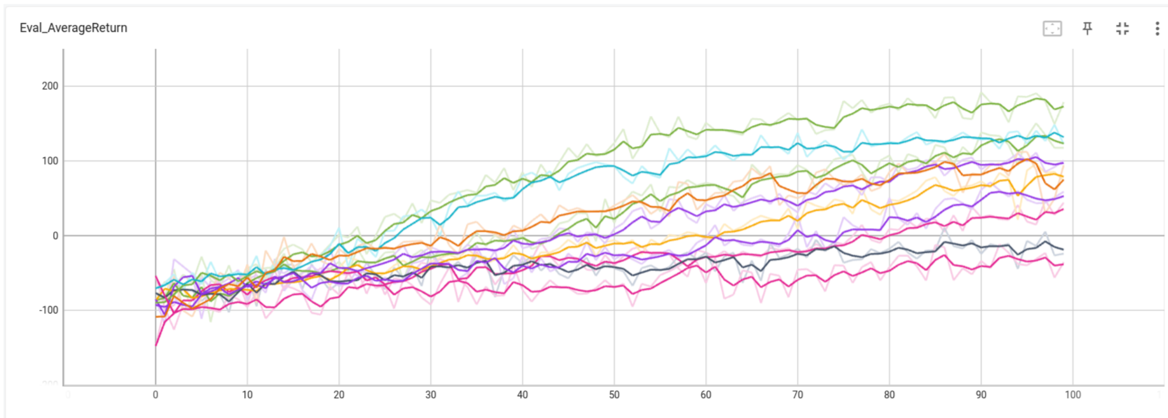


Figure 1: Brute-force search for the best batch size and learning rate

7.2.3 (Optional) Optimal b* and r* – [3 points]

Q7.2.3

b* = 50000
r* = 0.02

7.2.4 (Optional) Plot – [10 points]

Q7.2.4

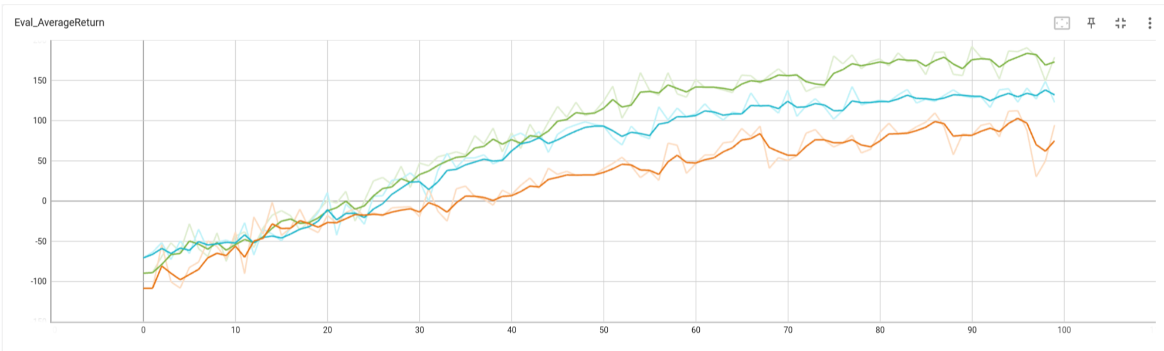


Figure 2: Learning rate = 0.02

7.2.5 (Optional) Describe how b* and r* affect task performance – [7 points]

Q7.2.5

First of all, there's still rooms for improvement if we train more iterations.
Larger batch and learning rate help to improve the policy faster.
I personally thinks learning rate affects more because all the configurations with learning rate = 0.02 achieved above 0 evaluation average return.

7.2.6 (Optional) Configurations with optimal b* and r* – [3 points]

Q7.2.6

```
python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 50000 -lr 0.02 \
    --exp_name q4_b50000_r0.02

python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 50000 -lr 0.02 -rtg \
    --exp_name q4_b50000_r0.02_rtg

python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 50000 -lr 0.02 --nn_baseline \
    --exp_name q4_b50000_r0.02_nnbaseline

python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 50000 -lr 0.02 -rtg --nn_baseline \
    --exp_name q4_b50000_r0.02_rtg_nnbaseline
```

7.2.7 (Optional) Plot for four runs with optimal b* and r* – [7 points]

Q7.2.7

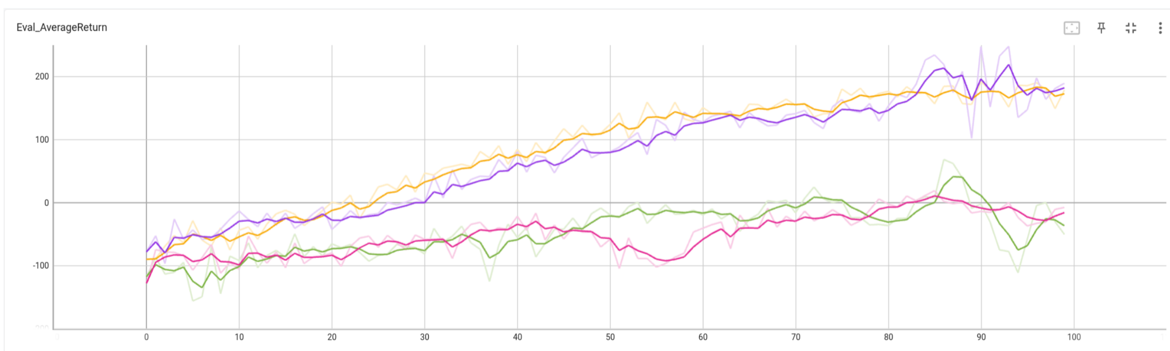


Figure 3: orange: rtg + baseline, violet: rtg, magenta: baseline, green: nothing

Reward-to-go boosts so much.

8 Implementing Generalized Advantage Estimation

8.1 Experiment 5 (Hopper) – [20 points]

8.1.1 Configurations

Q8.1.1

```
# λ ∈ [0, 0.95, 0.99, 1]
python rob831/scripts/run_hw2.py \
    --env_name Hopper-v4 --ep_len 1000 \
    --discount 0.99 -n 300 -l 2 -s 32 -b 2000 -lr 0.001 \
    --reward_to_go --nn_baseline --action_noise_std 0.5 --gae_lambda <λ> \
    --exp_name q5_b2000_r0.001_lambda<λ>
```

8.1.2 Plot – [13 points]

Q8.1.2

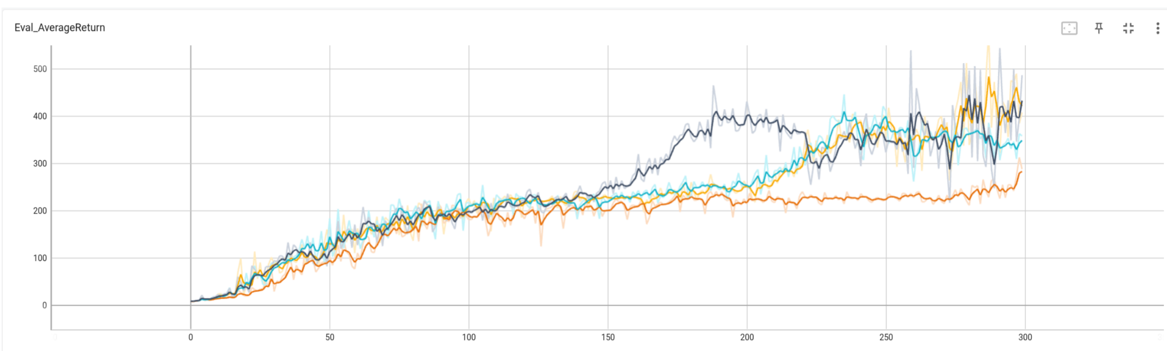


Figure 4: black: $\lambda = 0.95$, yellow: $\lambda = 1$, cyan: $\lambda = 0.99$, orange: $\lambda = 0$

8.1.3 Describe how λ affects task performance – [7 points]

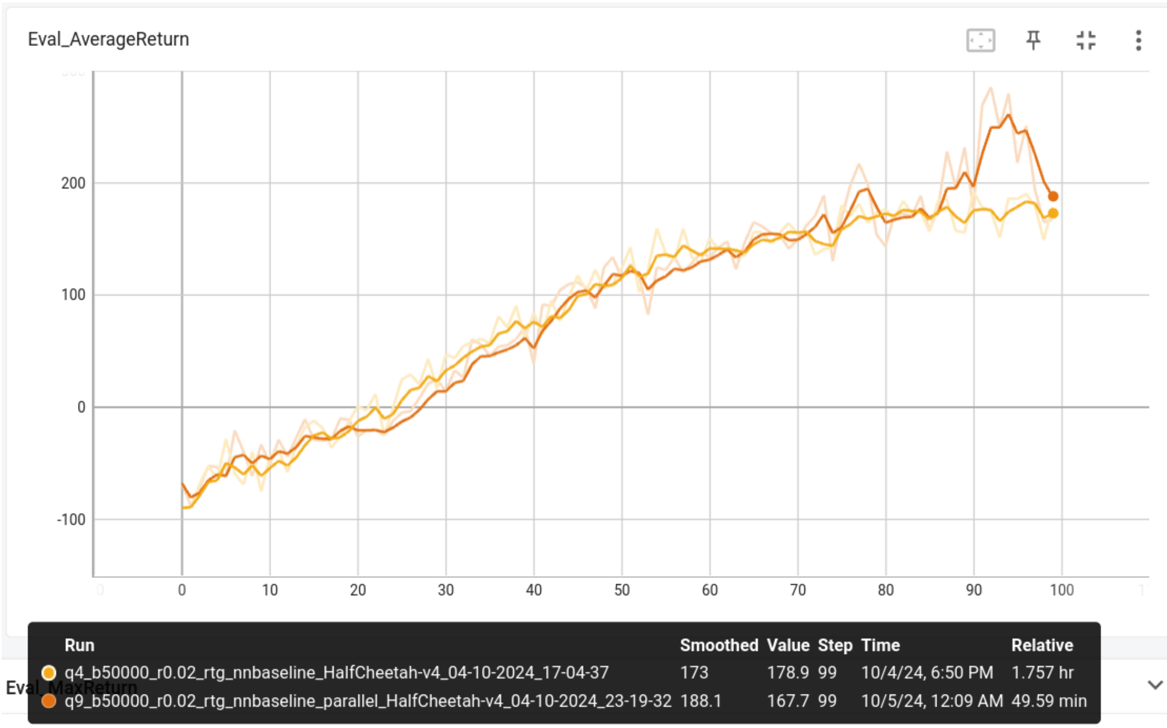
Q8.1.3

Lower λ values means we trust each individual rollouts over the value function (baseline), which typically has higher variance, or more noise.
Higher λ values means the opposite.
GAE's pretty clever in that it maps the input from $n \in [0, T - t - 1]$ to $\lambda \in [0, 1]$, so λ is basically the percentage of how much remaining rollouts to use, and how much we trust the reward-to-go. But it's non-linear, which is probably why we need to optimize around [0.9, 1]

9 Bonus! (optional)

9.1 Parallelization – [15 points]

Q9.1

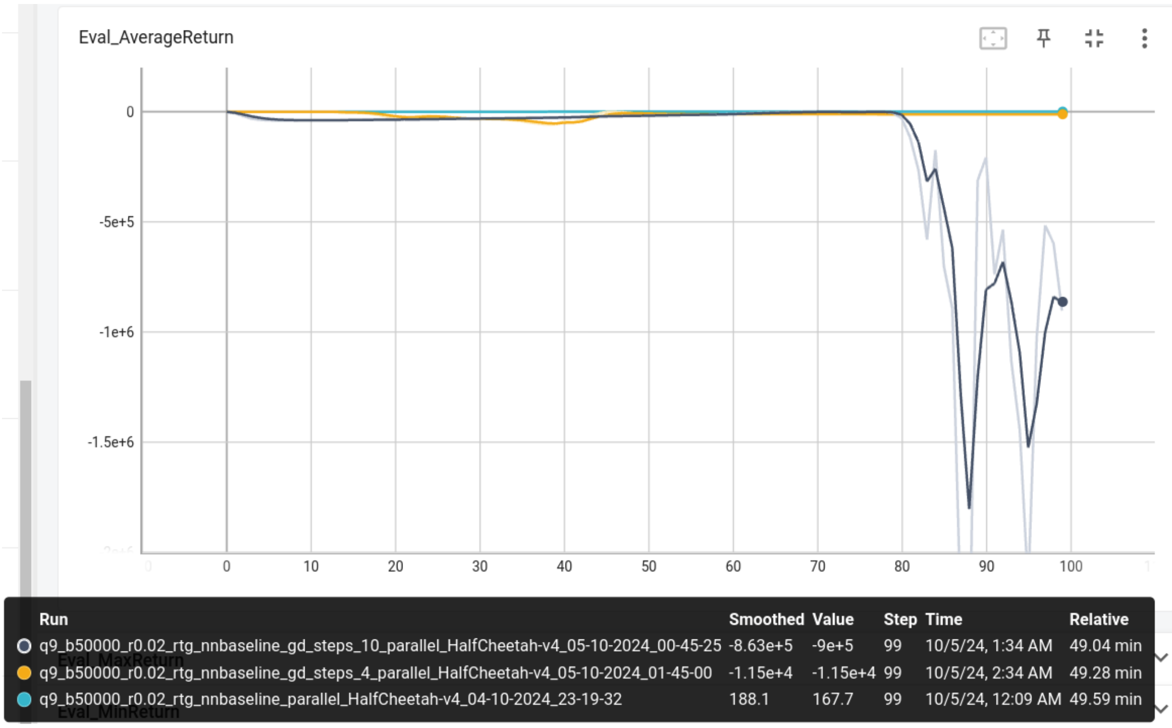


num_worker_threads is adaptive according to $\frac{\text{min\_timesteps\_per\_batch}}{\text{average\_path\_length}}$, capped by mp.cpu_count() - 1 I'm training on a laptop with 32 CPU hardware threads.
Difference in training time: $\frac{1.757\text{hr}}{49.59\text{min}} \approx 2$

```
python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 50000 -lr 0.02 -rtg --nn_baseline --parallel \
    --exp_name q9_b50000_r0.02_rtg_nnbaseline_parallel
```

9.2 Multiple gradient steps – [5 points]

Q9.1



Basically on-policy vs off-policy, off-policy without re-evaluating the policy to get up-to-date action and reward will result in negative effects.
Even if actions and rewards are re-evaluated after each gradient step, $p_\theta(s)$ still won't change, resulting in a larger exploration space, which makes the policy more robust (covers wider range) but less optimized (for the states it's supposed to optimize).

```
python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 50000 -lr 0.02 -rtg --nn_baseline --parallel --num_agent_train_steps_per_iter
↪   4 \
    --exp_name q9_b50000_r0.02_rtg_nnbaseline_gd_steps_4_parallel

python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b 50000 -lr 0.02 -rtg --nn_baseline --parallel --num_agent_train_steps_per_iter
↪   10 \
    --exp_name q9_b50000_r0.02_rtg_nnbaseline_gd_steps_10_parallel
```