Assignment 2: Policy Gradient

Andrew ID: `jihyunki`
Collaborators: `Write the Andrew IDs of your collaborators here (if any).`
NOTE: Please do **NOT** change the sizes of the answer blocks or plots.

5 Small-Scale Experiments

5.1 Experiment 1 (Cartpole) – [5 points total]

5.1.1 Configurations

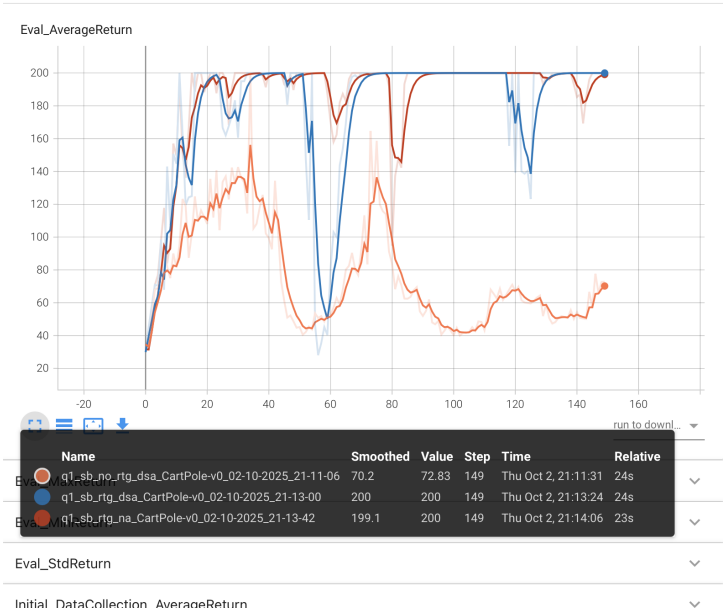
Q5.1.1

```
python rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 150 -b 1500 \
    -dsa --exp_name q1_sb_no_rtg_dsa

python rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 150 -b 1500 \
    -rtg -dsa --exp_name q1_sb_rtg_dsa

python rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 150 -b 1500 \
    -rtg --exp_name q1_sb_rtg_na

python rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 150 -b 6000 \
    -dsa --exp_name q1_lb_no_rtg_dsa

python rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 150 -b 6000 \
    -rtg -dsa --exp_name q1_lb_rtg_dsa

python rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 150 -b 6000 \
    -rtg --exp_name q1_lb_rtg_na
```

5.1.2 Plots

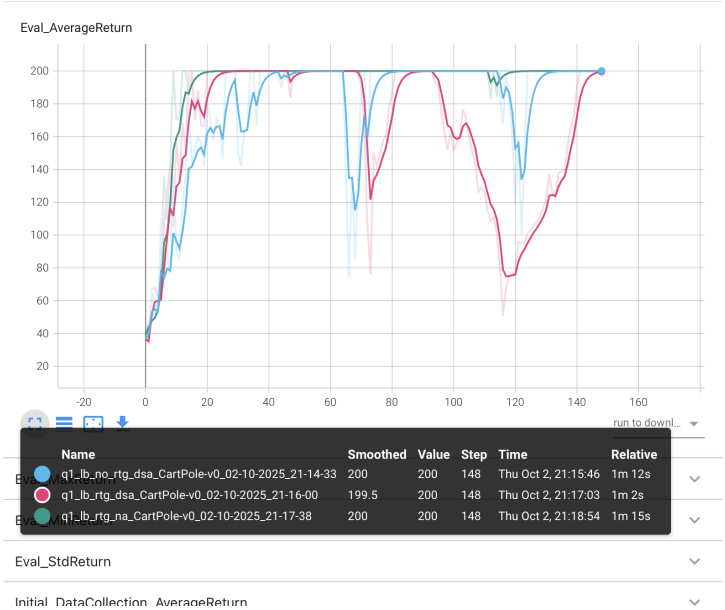
5.1.2.1 Small batch – [1 points]

Q5.1.2.1



5.1.2.2 Large batch – [1 points]

Q5.1.2.2



5.1.3 Analysis

5.1.3.1 Value estimator – [1 points]

Q5.1.3.1

The reward-to-go estimator shows better performance compared to the trajectory-centric estimator without advantage standardization. Across both small-batch (b=1500) and large-batch (b=6000) settings, reward-to-go policies converge more quickly to the optimal return and maintain higher stability, while trajectory-centric estimators learn more slowly and often fluctuate at lower performance. This confirms that reward-to-go provides a more informative gradient signal, enabling faster and more stable learning.

5.1.3.2 Advantage standardization – [1 points]

Q5.1.3.2

Advantage standardization improves training stability across both small and large batch settings. With standardization, the learning curves become smoother and exhibit reduced variance, indicating that normalizing advantages helps stabilize updates and prevents divergence. This leads to more consistent convergence toward the optimal return.

5.1.3.3 Batch size – [1 points]

Q5.1.3.3

Smaller batches lead to noisier learning curves and less consistent final returns, while larger batches provide more reliable gradient estimates and smoother convergence. Larger batch runs consistently reach higher and more stable returns, demonstrating the advantage of using greater batch sizes in this setting.

5.2 Experiment 2 (InvertedPendulum) – [4 points total]

5.2.1 Configurations – [1.5 points]

Q5.2.1

```
# Sweep used to explore b and r
for B in 800 1000 1500 2000; do
  for LR in 0.001 0.003 0.01 0.03; do
    /content/py310/bin/python -m rob831.scripts.run_hw2 \
      --env_name InvertedPendulum-v4 \
      --ep_len 1000 --discount 0.92 \
      -n 100 -l 2 -s 64 \
      -b $B -lr $LR -rtg \
      --exp_name q2_b${B}_r${LR}
  done
done

# Final chosen run (b* = 1000, r* = 0.01)
/content/py310/bin/python -m rob831.scripts.run_hw2 \
  --env_name InvertedPendulum-v4 \
  --ep_len 1000 --discount 0.92 \
  -n 100 -l 2 -s 64 \
  -b 1000 -lr 0.01 -rtg \
  --exp_name q2_b1000_r0.01
```

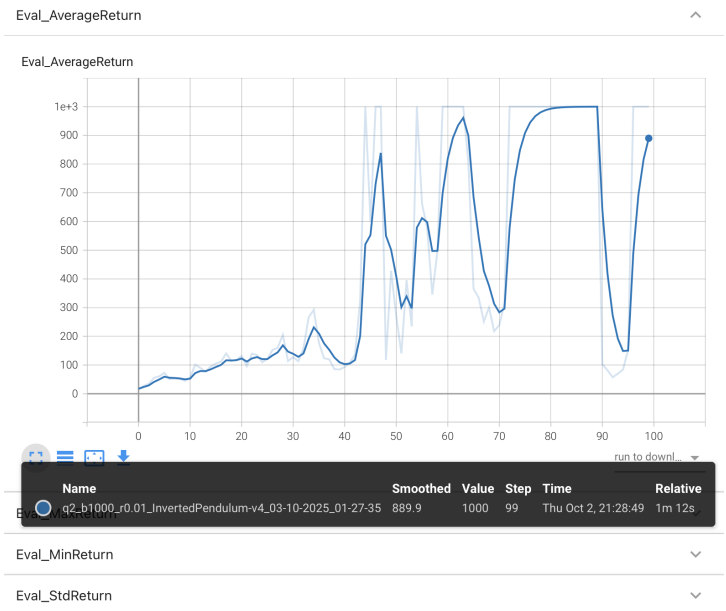
5.2.2 smallest b* and largest r* (same run) – [1.5 points]

Q5.2.2

The smallest batch size and largest learning rate combination that still reached the optimal score within 100 iterations was b = 1000* and r = 0.01*. This configuration converged to the maximum reward of 1000 in fewer than 100 iterations, demonstrating fast and stable learning.

5.2.3 Plot – [1 points]

Q5.2.3



7 More Complex Experiments

7.1 Experiment 3 (LunarLander) – [1 points total]

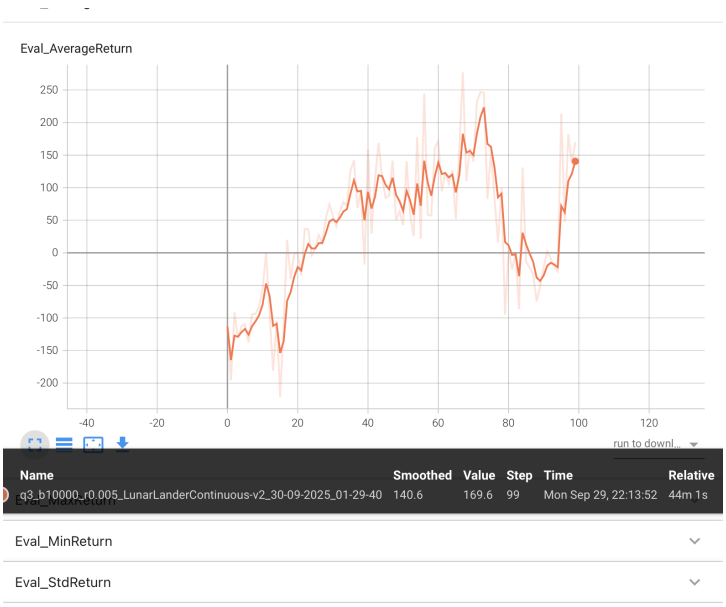
7.1.1 Configurations

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 – [1 points]

Q7.1.2



7.2 Experiment 4 (HalfCheetah) – [1 points]

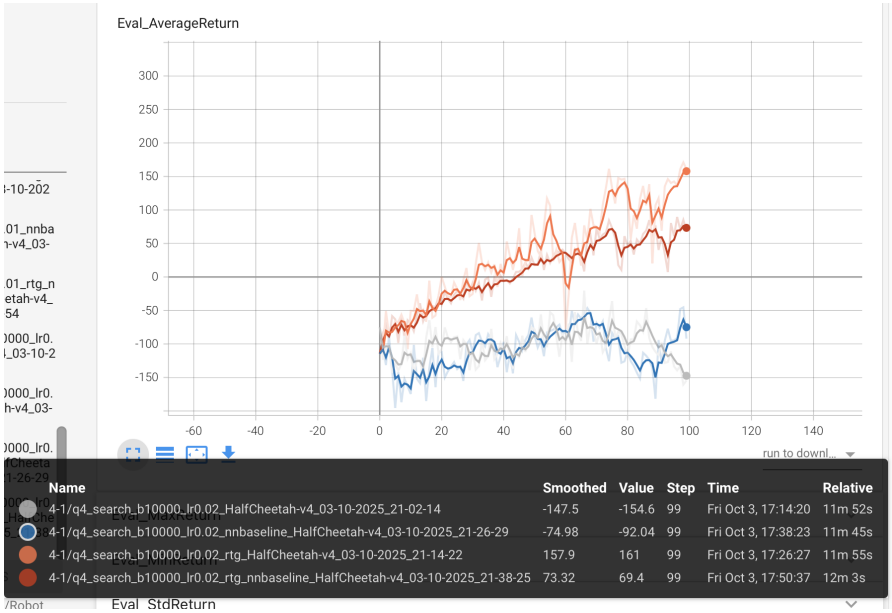
7.2.1 Configurations

Q7.2.1

```
# Search space: batch size {15000, 35000, 55000}, learning rate {0.005, 0.01, 0.02}
for B in 15000 35000 55000; do
  for LR in 0.005 0.01 0.02; do
    /content/py310/bin/python -m rob831.scripts.run_hw2 \
      --env_name HalfCheetah-v4 --ep_len 150 \
      --discount 0.95 -n 100 -l 2 -s 32 \
      -b $B -lr $LR -rtg --nn_baseline \
      --exp_name q4_search_b${B}_lr${LR}_rtg_nnbaseline
  done
done
```

7.2.2 Plot – [1 points]

Q7.2.2



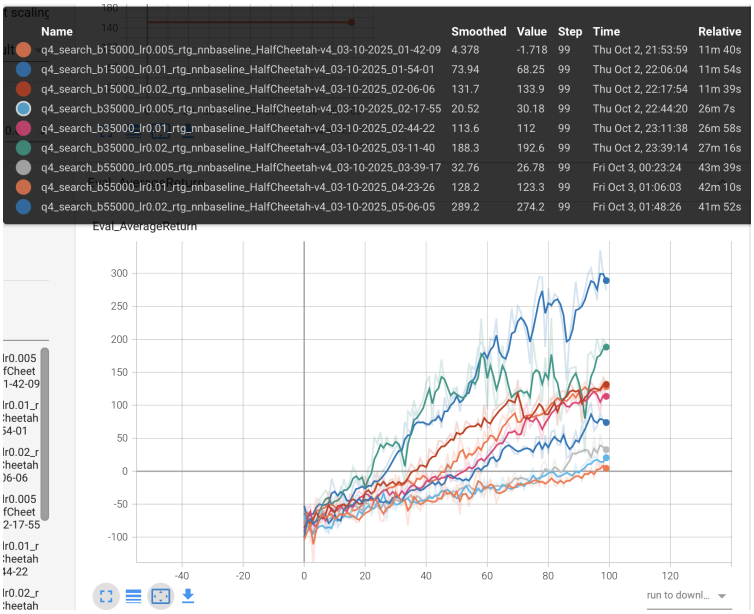
7.2.3 (bonus) Optimal b* and r* – [0.5 points]

Q7.2.3

I found b*=35000, r*=0.01

7.2.4 (bonus) Plot – [0.5 points]

Q7.2.4



7.2.5 (bonus) Describe how b* and r* affect task performance – [0.5 points]

Q7.2.5

Larger batches ($b = 35\text{k}–55\text{k}$) produced smoother curves and faster, more reliable gains, while $b = 15\text{k}$ was noticeably noisier and tended to plateau lower. The learning rate showed the usual bias–variance trade-off: $r = 0.005$ was too conservative (slow improvement and lower final returns), whereas $r = 0.02$ often became unstable with higher variance and occasional regressions. The mid-range $r = 0.01$ consistently yielded the best performance and stability. Taken together, the results indicate an optimal setting around $\boxed{b^{=}35{,}000,\ r^{=}0.01}$, which balances sample efficiency and stable policy updates and attains the highest returns in the plot.

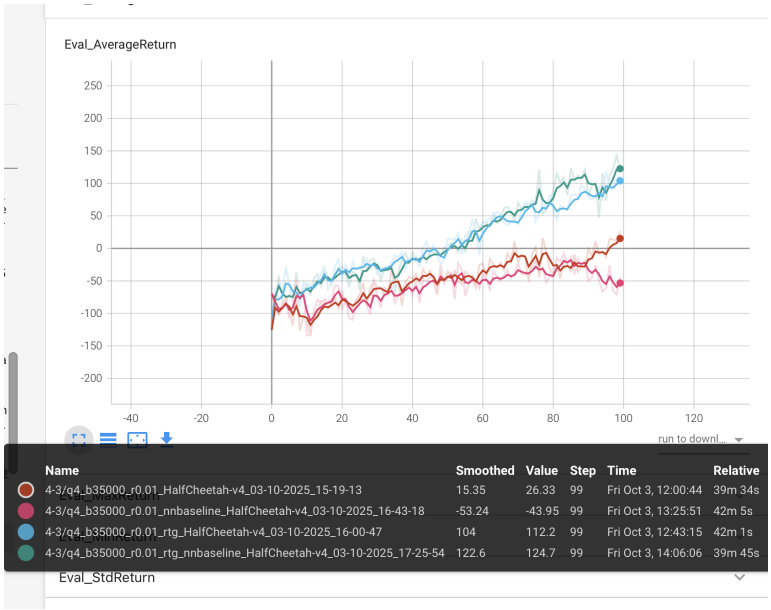
7.2.6 (bonus) Configurations with optimal b* and r* – [0.5 points]

Q7.2.6

```
# Run HalfCheetah-v4 with b*=35000, r*=0.01

/content/py310/bin/python -m rob831.scripts.run_hw2 --env_name HalfCheetah-v4 --ep_len 150 \
  --discount 0.95 -n 100 -l 2 -s 32 -b 35000 -lr 0.01 --exp_name q4_b35000_r0.01

/content/py310/bin/python -m rob831.scripts.run_hw2 --env_name HalfCheetah-v4 --ep_len 150 \
  --discount 0.95 -n 100 -l 2 -s 32 -b 35000 -lr 0.01 -rtg --exp_name q4_b35000_r0.01_rtg

/content/py310/bin/python -m rob831.scripts.run_hw2 --env_name HalfCheetah-v4 --ep_len 150 \
  --discount 0.95 -n 100 -l 2 -s 32 -b 35000 -lr 0.01 --nn_baseline --exp_name q4_b35000_r0.01_nnbaseline

/content/py310/bin/python -m rob831.scripts.run_hw2 --env_name HalfCheetah-v4 --ep_len 150 \
  --discount 0.95 -n 100 -l 2 -s 32 -b 35000 -lr 0.01 -rtg --nn_baseline --exp_name q4_b35000_r0.01_rtg_nnbaseline
```

7.2.7 (bonus) Plot for four runs with optimal b* and r* – [0.5 points]

Q7.2.7



8 Implementing Generalized Advantage Estimation

8.1 Experiment 5 (Hopper) – [4 points]

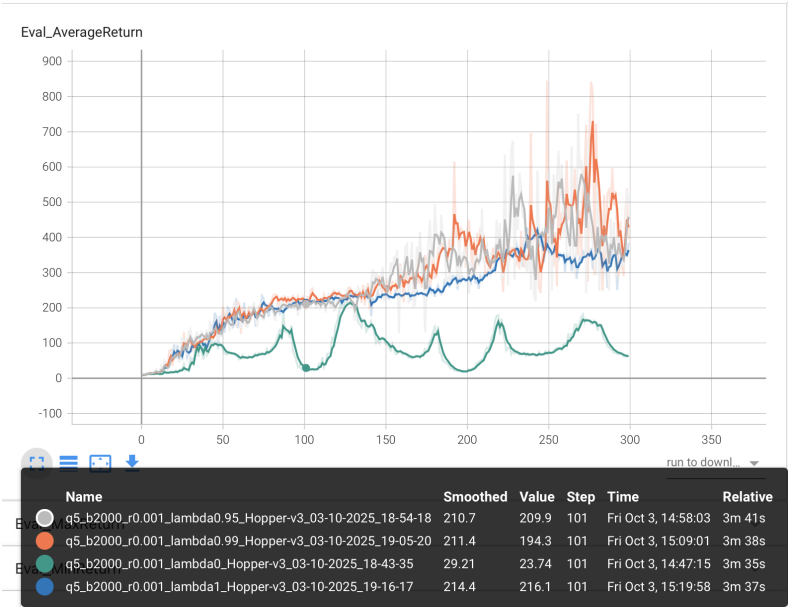
8.1.1 Configurations

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 – [2 points]

Q8.1.2



8.1.3 Describe how λ affects task performance – [2 points]

Q8.1.3

The choice of λ strongly impacts both stability and final performance. At $\lambda = 0$, the policy fails to learn effectively, oscillating with low returns below 250. Increasing λ improves performance: $\lambda = 0.95$ and $\lambda = 0.99$ achieve the highest returns, with $\lambda = 0.99$ reaching peaks above 600 but at the cost of higher variance and instability, while $\lambda = 0.95$ provides steadier learning with returns in the 400–500 range. In contrast, $\lambda = 1$ produces more stable but lower returns around 300–350, suggesting reduced variance but limited performance. Overall, moderate-to-high λ values (0.95–0.99) strike the best balance between bias and variance, yielding faster learning and higher average returns compared to extreme values of λ.

9 More Bonus!

9.1 Parallelization – [1.5 points]

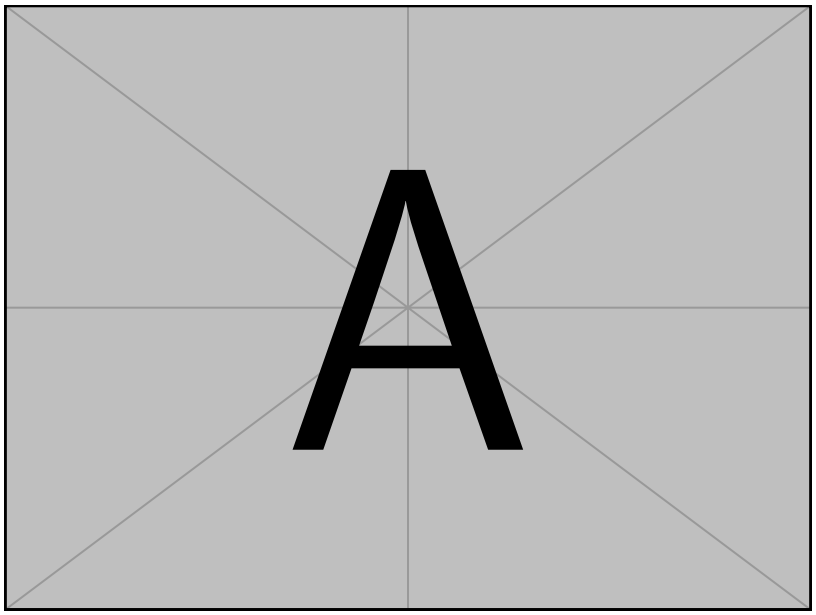
Q9.1

Difference in training time:

```
python rob831/scripts/run_hw2.py \
```

9.2 Multiple gradient steps – [1 points]

Q9.1




```
python rob831/scripts/run_hw2.py \
```