Assignment 2: Policy Gradient

Andrew ID: `dameria`
Collaborators: `parthsin, ankitagg, Gen AI`
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

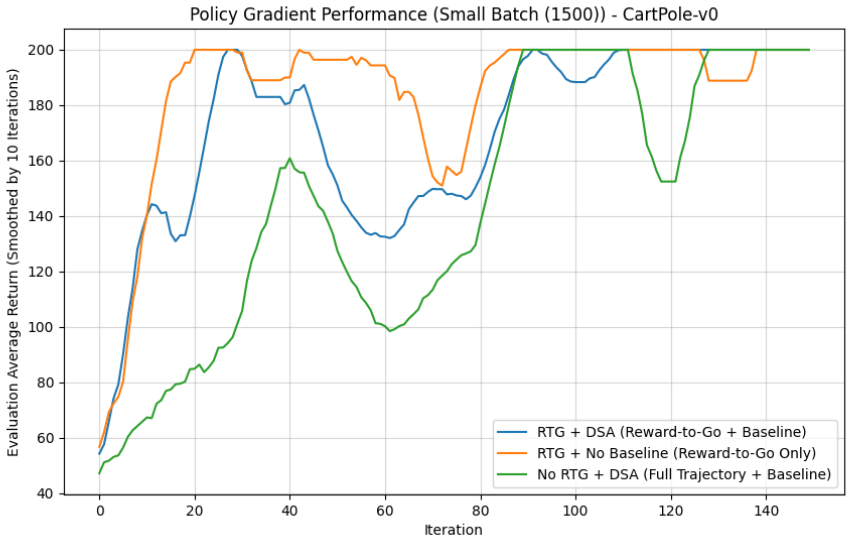
Q5.1.1

```
python3 rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 150 -b 1500 -rtg --exp_name q1_sb_rtg_na

python3 rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 150 -b 1500 -dsa --exp_name q1_sb_no_rtg_dsa

python3 rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 150 -b 1500 -rtg -dsa --exp_name q1_sb_rtg_dsa

python3 rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 150 -b 6000 -rtg --exp_name q1_lb_rtg_na

python3 rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 150 -b 6000 -dsa --exp_name q1_lb_no_rtg_dsa

python3 rob831/scripts/run_hw2.py --env_name CartPole-v0 -n 150 -b 6000 -rtg -dsa --exp_name q1_lb_rtg_dsa
```

5.1.2 Plots

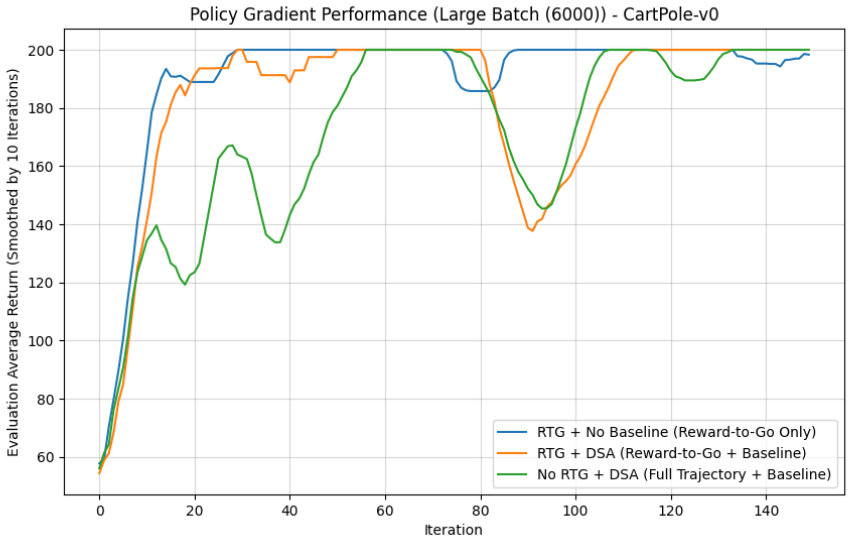
5.1.2.1 Small batch – [1 points]

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 RTG + No Baseline curve consistently achieves the highest final return (near the maximum of 200) and is generally the most stable performer in both Small and Large Batch plots.

5.1.3.2 Advantage standardization – [1 points]

Q5.1.3.2

Yes, advantage standardization helped significantly for improving training stability, particularly when used with small batch sizes. Its benefit was less pronounced with large batch sizes because they inherently reduce the variance in the advantage estimates, making the normalization less critical.

5.1.3.3 Batch size – [1 points]

Q5.1.3.3

Yes, the batch size made a noticeable impact, primarily on stability and variance. The Large Batch (6000) size produced much smoother and more reliable training curves, leading to longer periods of optimal performance compared to the Small Batch (1500).

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

5.2.1 Configurations – [1.5 points]

Q5.2.1

```
python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 --discount 0.92 --n_iter 100 \
--n_layers 2 --size 64 --reward_to_go --exp_name q2_b10000_r0.01 --batch_size 10000 --learning_rate 0.01

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 --discount 0.92 --n_iter 100 \
--n_layers 2 --size 64 --reward_to_go --exp_name q2_b20000_r0.01 --batch_size 20000 --learning_rate 0.01

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 --discount 0.92 --n_iter 100 \
--n_layers 2 --size 64 --reward_to_go --exp_name q2_b10000_r0.005 --batch_size 10000 --learning_rate 0.005

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 --discount 0.92 --n_iter 100 \
--n_layers 2 --size 64 --reward_to_go --exp_name q2_b30000_r0.01 --batch_size 30000 --learning_rate 0.01

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 --discount 0.92 --n_iter 100 \
--n_layers 2 --size 64 --reward_to_go --exp_name q2_b15000_r0.008 --batch_size 15000 --learning_rate 0.008
```

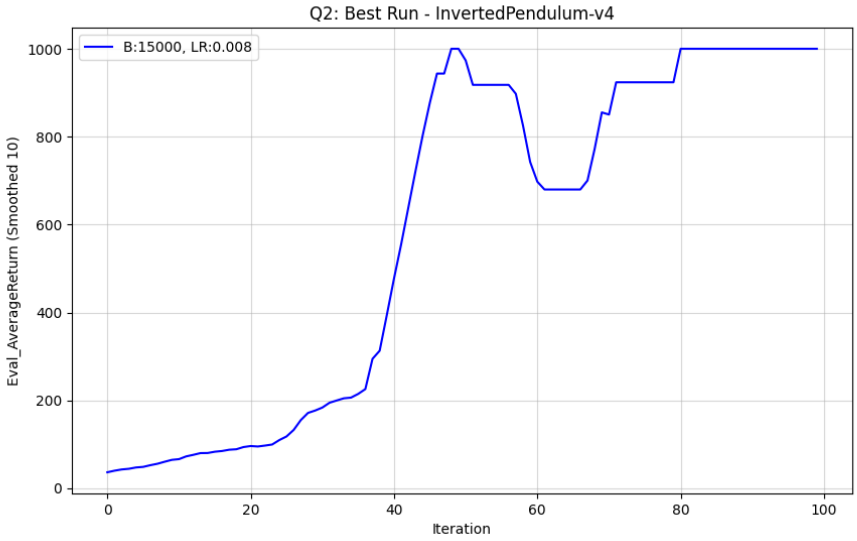
5.2.2 smallest b* and largest r* (same run) – [1.5 points]

Q5.2.2

The best results came from b* = 15000 and r* = 0.008

5.2.3 Plot – [1 points]

Q5.2.3



7 More Complex Experiments

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

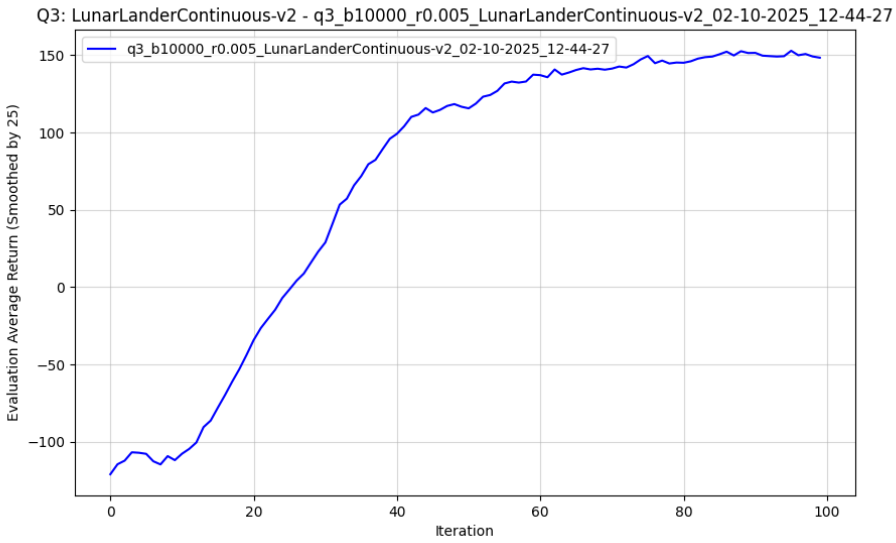
7.1.1 Configurations

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]

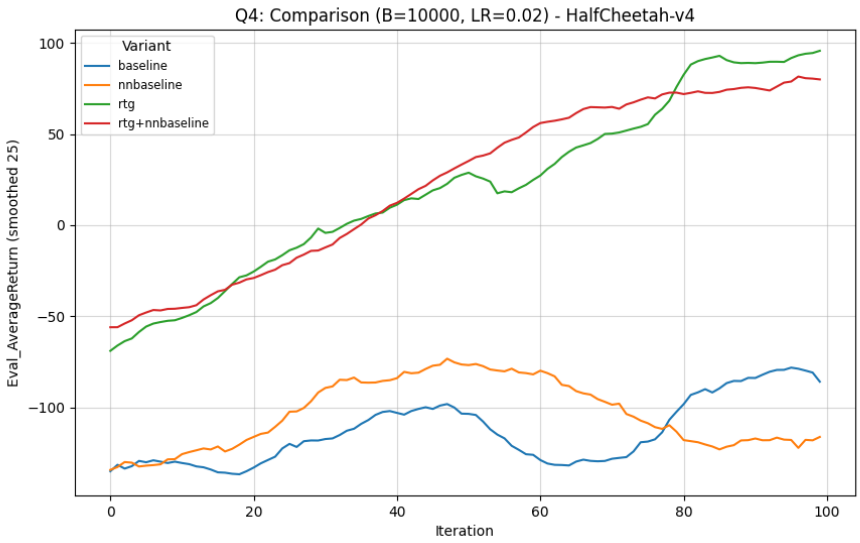
7.2.1 Configurations

Q7.2.1

```
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 10000 -lr 0.02 -rtg \
    --exp_name q4_search_b10000_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 10000 -lr 0.02 --nn_baseline \
    --exp_name q4_search_b10000_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 0.02 -rtg --nn_baseline \
    --exp_name q4_search_b10000_lr0.02_rtg_nnbaseline
```

7.2.2 Plot – [1 points]

Q7.2.2



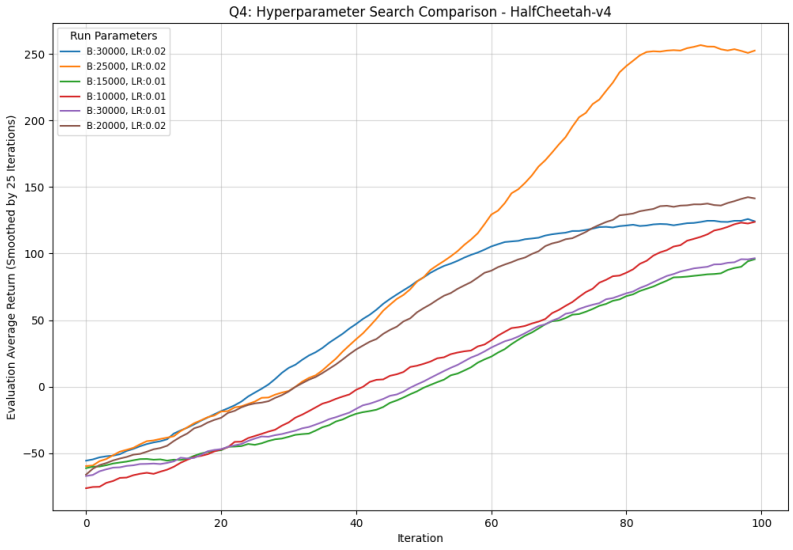
7.2.3 Optimal b* and r* – [0.5 points]

Q7.2.3

Optimal b* = 25000, r* = 0.02

7.2.4 Plot – [0.5 points]

Q7.2.4



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

Q7.2.5

Large batch sizes improved final performance, but only when paired with a high learning rate. While large batches promote stability by reducing variance, the highest performance requires accelerating convergence with a suitable LR.
The higher LR: 0.02 was critical for achieving both the fastest and highest final returns across various batch sizes. The lower LR:0.01 consistently resulted in poorer performance, underscoring its role in setting the performance ceiling and convergence speed. It is noteworthy that too high of an LR can lead to overshooting and missed minima.

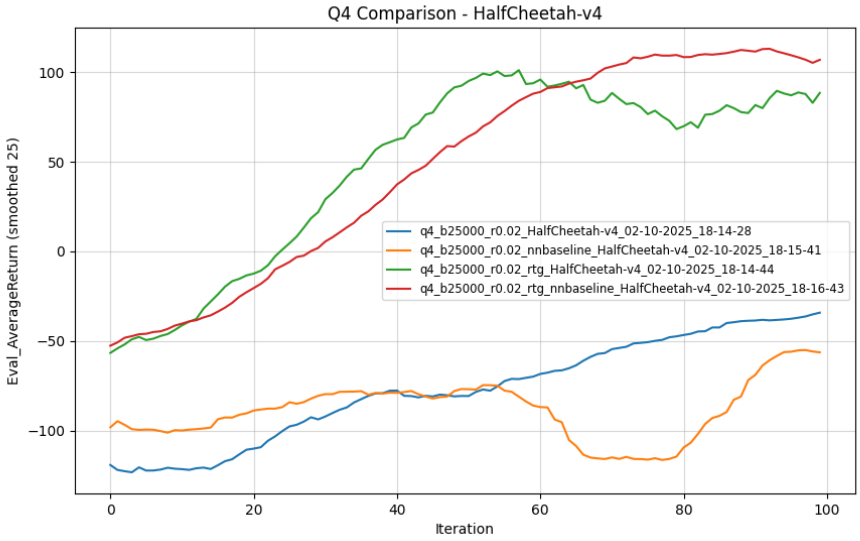
7.2.6 Configurations with optimal b* and r* – [0.5 points]

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 25000 -lr 0.02 \
    --exp_name q4_b250000_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 25000 -lr 0.02 -rtg \
    --exp_name q4_b250000_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 25000 -lr 0.02 --nn_baseline \
    --exp_name q4_b250000_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 25000 -lr 0.02 -rtg --nn_baseline \
    --exp_name q4_b250000_r0.02_rtg_nnbaseline
```

7.2.7 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]

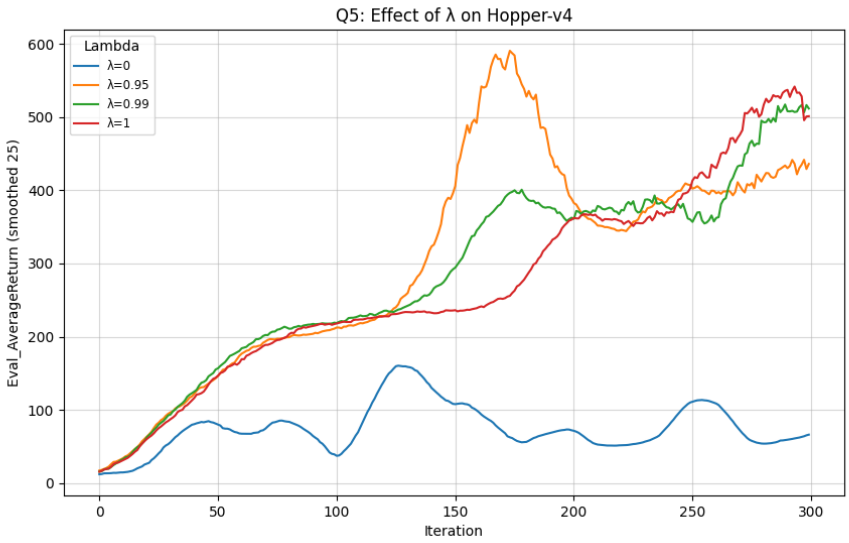
8.1.1 Configurations

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 significantly impacts learning, balancing stability and speed of convergence.
1. When λ=0, the network fails to learn because the baseline network is not being utilized.
2. Values of λ=0.95 and λ=0.99 yield competitive performance, with 0.95 having higher peaks, but 0.99 achieving a higher final converged value.
The highest value, λ=1, results in the most stable learning with less dramatic overshoots and peaks.

9 More Bonus!(DID NOT ATTEMPT)

9.1 Parallelization – [1.5 points]

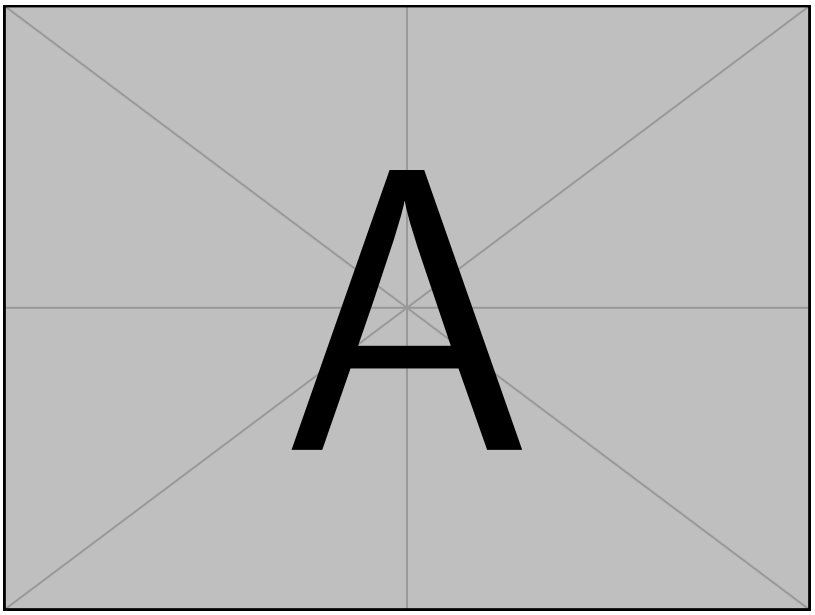
Q9.1

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 \
```