Assignment 2: Policy Gradient

Andrew ID: `rlokosso`
Collaborators: `None`

5 Small-Scale Experiments

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

5.1.1 Configurations

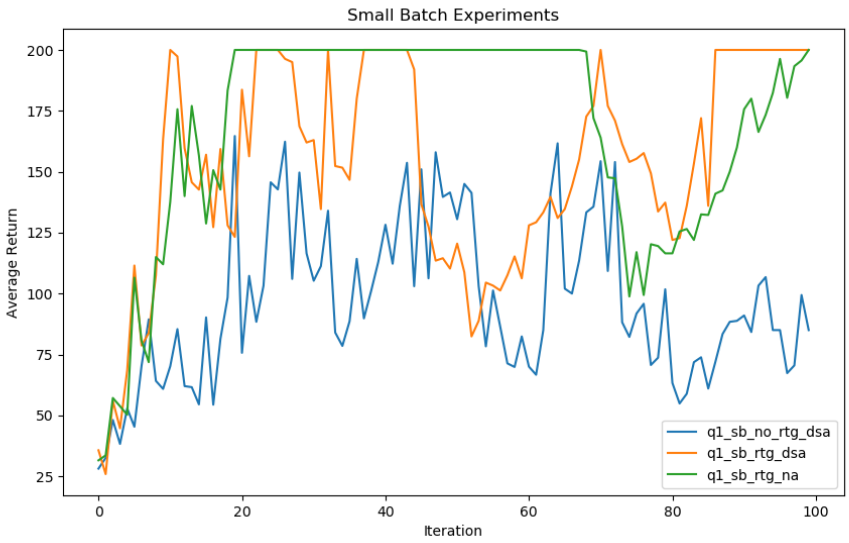
Q5.1.1

```
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

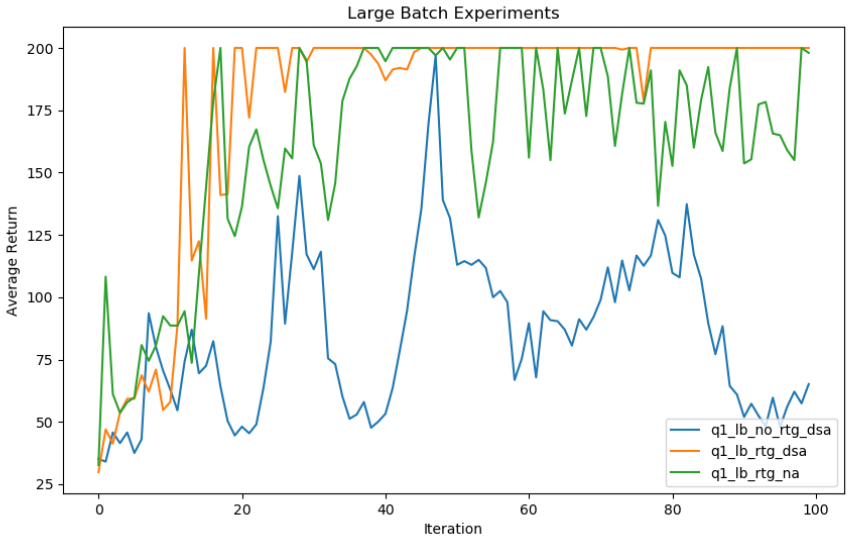
5.1.2.1 Small batch – [5 points]

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

The reward-to-go estimator has better performance without advantage-standardization. In both small and large batch experiments, the reward-to-go method (orange line) achieves higher average returns and shows more stability compared to the trajectory-centric approach (blue line).

5.1.3.2 Advantage standardization – [5 points]

Q5.1.3.2

Yes, advantage standardization definitely helped. In both graphs, the method using reward-to-go with advantage standardization (green line) achieves the highest and most stable average returns. This improvement is particularly evident in the large batch experiments, where it consistently outperforms the other methods.

5.1.3.3 Batch size – [5 points]

Q5.1.3.3

Yes, the batch size made a significant impact. The large batch experiments demonstrate:

- More stable learning curves with less variance between iterations.
- Higher peak performances, especially for the reward-to-go method with advantage standardization.
- Clearer differentiation between the effectiveness of each method.

In conclusion, larger batch sizes lead to more stable and better overall performance, particularly when combined with reward-to-go and advantage standardization.

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 1000
↪  -lr 0.01 -rtg --exp_name q2_b1000_r1e_2

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_b1000_r1e_1

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 70
↪  -lr 0.03 -rtg --exp_name q2_b70_r3e_2

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 60
↪  -lr 0.03 -rtg --exp_name q2_b60_r3e_2

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 50
↪  -lr 0.03 -rtg --exp_name q2_b50_r3e_2

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 55
↪  -lr 0.03 -rtg --exp_name q2_b55_r3e_2

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 53
↪  -lr 0.03 -rtg --exp_name q2_b53_r3e_2

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 54
↪  -lr 0.03 -rtg --exp_name q2_b54_r3e_2

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 55
↪  -lr 0.03 -rtg --exp_name q2_b55_r3e_2

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 55
↪  -lr 0.04 -rtg --exp_name q2_b55_r4e_2

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 55
↪  -lr 0.03 -rtg --exp_name q2_b55_r3e_2
```

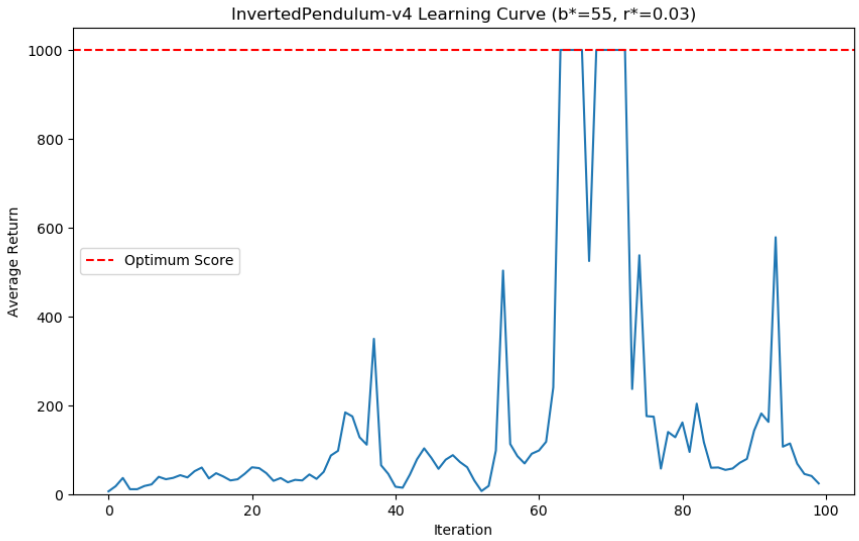
5.2.2 smallest b* and largest r* (same run) – [5 points]

Q5.2.2

After running the experiments with the InvertedPendulum-v4 environment, I found the following optimal parameters:
Smallest batch size (b*): 55
Largest learning rate (r*): 0.03

5.2.3 Plot – [5 points]

Q5.2.3



7 More Complex Experiments

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

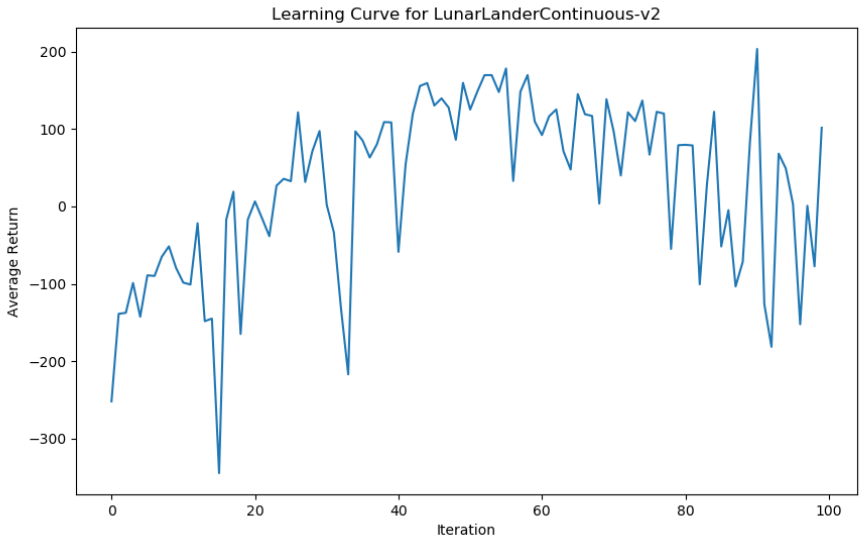
7.1.1 Configurations

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]

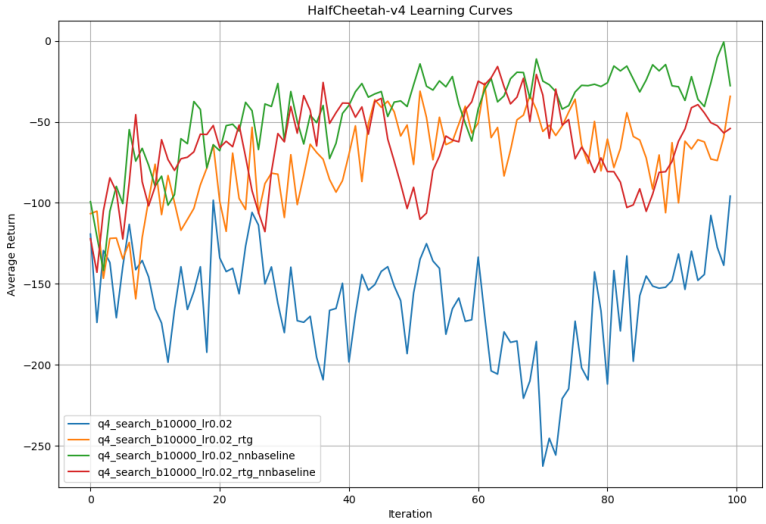
7.2.1 Configurations

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 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 – [10 points]

Q7.2.2



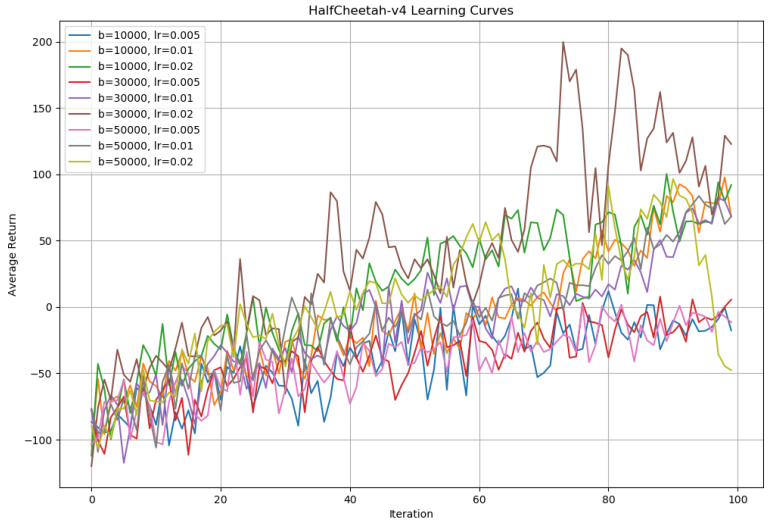
7.2.3 (Optional) Optimal b* and r* – [3 points]

Q7.2.3

Optimal parameters: batch_size (b*) = 30000, learning_rate (r*) = 0.02
For this HalfCheetah-v4 task, larger batch sizes paired with higher learning rates generally led to better and more stable performance. This suggests that for this complex control task, having more data per iteration (larger batch size) allows for more accurate gradient estimates, while higher learning rates enable the policy to adapt more quickly to this information. The optimal combination of b=30000 and lr=0.02 likely provides the best trade-off between stable learning and rapid policy improvement for this specific environment and training setup.

7.2.4 (Optional) Plot – [10 points]

Q7.2.4



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

Q7.2.5

The optimal batch size of 30000 and learning rate of 0.02 significantly enhanced task performance across all methods, particularly for the combined reward-to-go (RTG) and neural network (NN) baseline approach. This large batch size likely provided stable gradient estimates, allowing for more reliable policy updates. The relatively high learning rate enabled rapid adaptation without causing instability, especially evident in the RTG and RTG+NN baseline methods. These parameters allowed the algorithm to efficiently process a substantial amount of experience per iteration while making significant policy improvements.

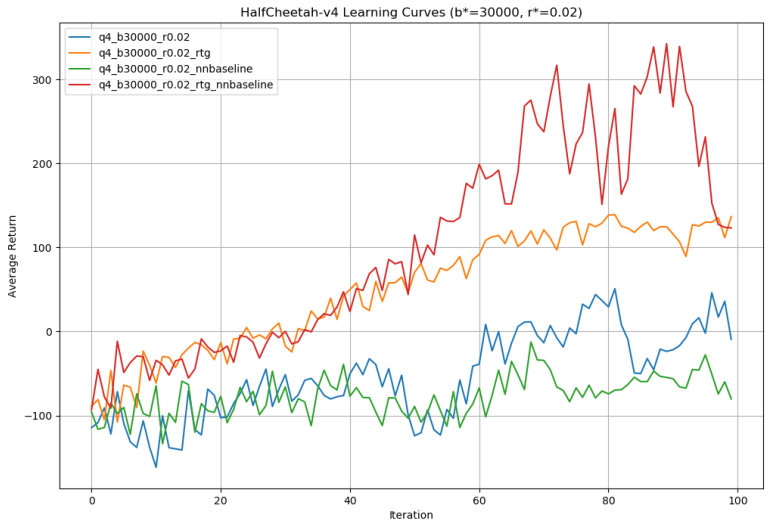
7.2.6 (Optional) Configurations with optimal b* and r* – [3 points]

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 <b*> -lr <r*> \
    --exp_name q4_b<b*>_r<r*>

python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b <b*> -lr <r*> -rtg \
    --exp_name q4_b<b*>_r<r*>_rtg

python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b <b*> -lr <r*> --nn_baseline \
    --exp_name q4_b<b*>_r<r*>_nnbaseline

python rob831/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b <b*> -lr <r*> -rtg --nn_baseline \
    --exp_name q4_b<b*>_r<r*>_rtg_nnbaseline
```

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

Q7.2.7



8 Implementing Generalized Advantage Estimation

8.1 Experiment 5 (Hopper) – [20 points]

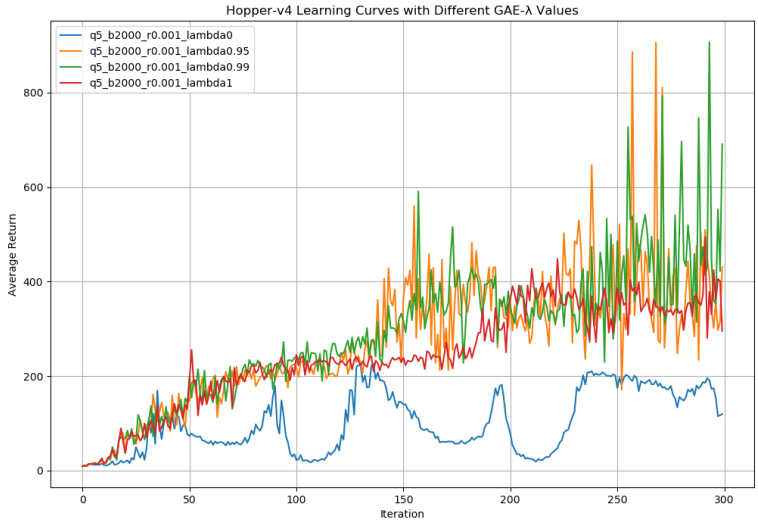
8.1.1 Configurations

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



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

Q8.1.3

The GAE-λ parameter significantly impacted performance on the Hopper-v4 task. Higher λ values (0.95, 0.99, 1.0) consistently outperformed $\lambda = 0$, with $\lambda = 0.99$ and $\lambda = 0.95$ showing the best results, often exceeding 600 in average returns. $\lambda = 1.0$ (vanilla baseline) performed well but with slightly lower peaks. $\lambda = 0$ severely underperformed, rarely exceeding 200. This indicates that incorporating longer-term rewards through higher λ values is crucial for effective learning in this complex, noisy environment, likely due to more accurate advantage estimation.

9 Bonus! (optional)

9.1 Parallelization – [15 points]

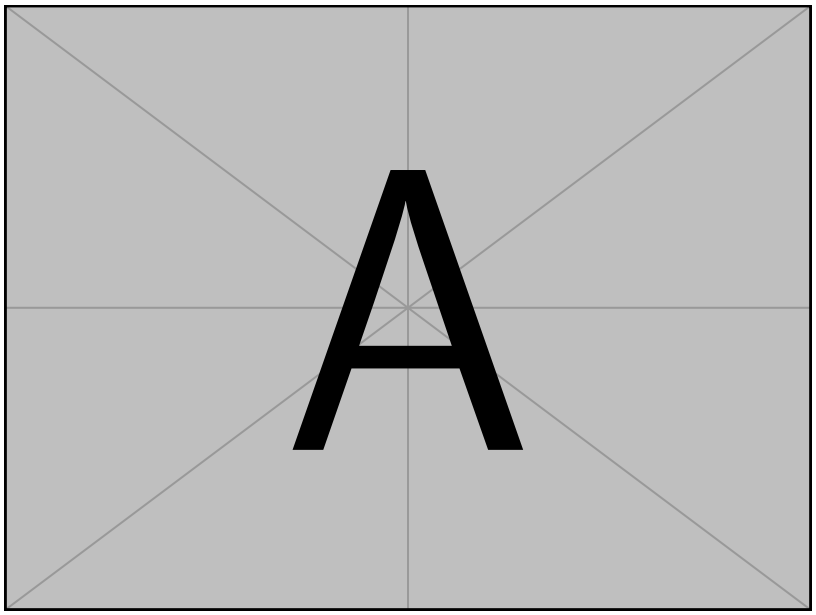
Q9.1

Difference in training time:

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

9.2 Multiple gradient steps – [5 points]

Q9.1




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