Assignment 2: Policy Gradient

Andrew ID: vtambe
Collaborators: pvenkat2
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

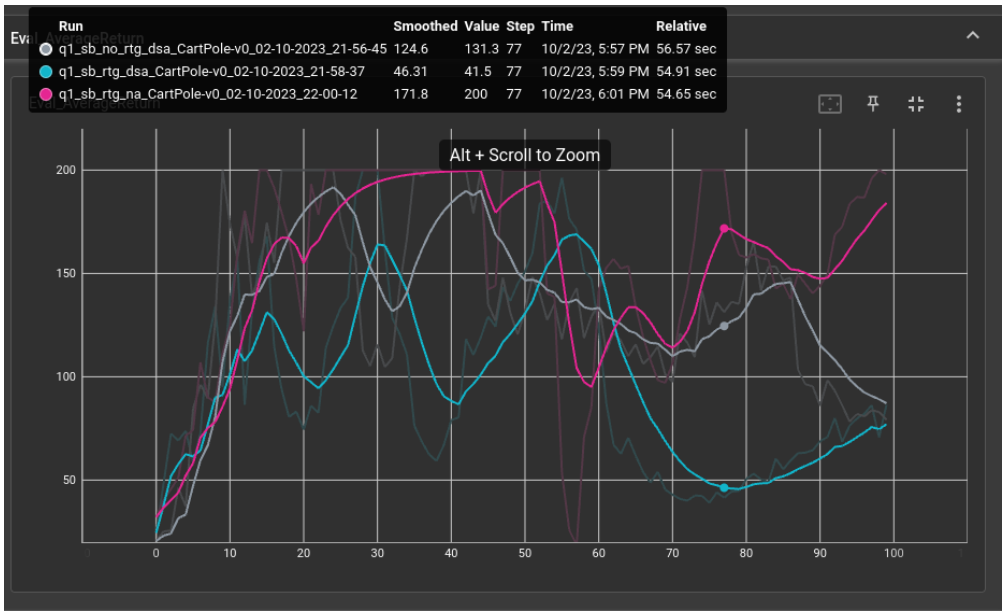
Q5.1.1

```
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

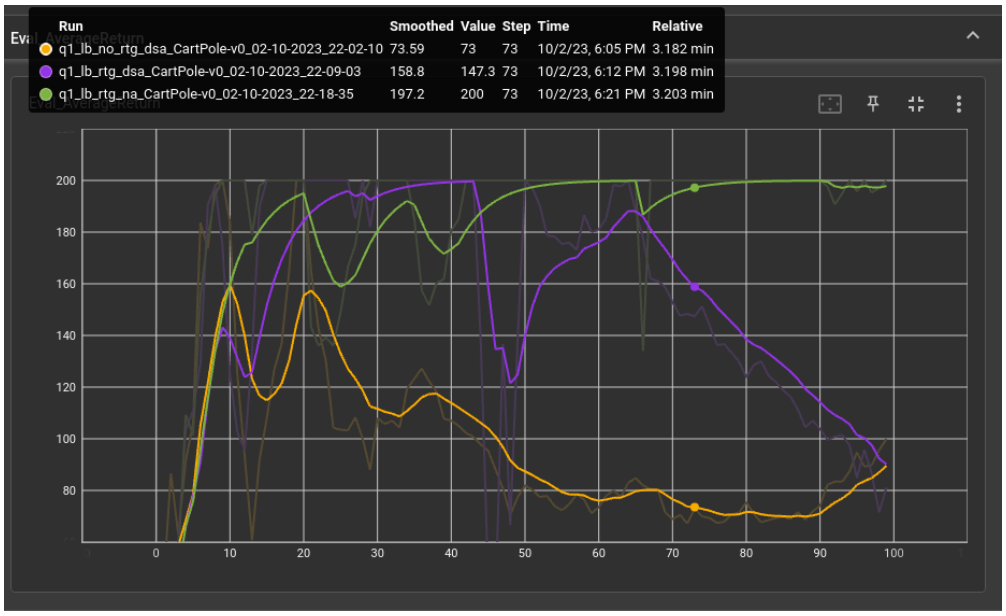
5.1.2.1 Small batch – [5 points]

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

As seen from the plots in Q5.1.3.2 the value estimator trained using **reward-to-go performs better** than the trajectory-centric one. The reason for this is that we can make a markokvian assumption that all the past rewards can be implicitly represented by the current reward estimate thus not needing to sum over them. We can get a better estimate of the expected reward (reward-to-go) by summing the current and estimated future rewards. This has the additional benefit of reduced computation.

5.1.3.2 Advantage standardization – [5 points]

Q5.1.3.2

Yes, advantage standardization helps in training the value estimator. The reason for this is that this **prevents large variation** in advantage values by keeping it zero mean and 1 standard deviation, reducing the possibilities of noisy gradients, and thus allowing the network to learn faster.

5.1.3.3 Batch size – [5 points]

Q5.1.3.1

Yes, the experiments with larger batch size converge faster than the experiments with smaller batch size. This is because of how the policy gradient algorithm is designed - which requires a trajectory of data predicted by the same policy in order to do backpropagation on it. This data is then discarded - which makes it very data inefficient. Thus, increasing batch size results in an increase in the amount of data available during every interaction of training and leads to faster convergence of the network.

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 0.005 -rtg --exp_name q2_b1000_r0.005

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 \
    --discount 0.9 -n 100 -l 2 -s 64 -b 2500 -lr 0.005 -rtg --exp_name q2_b2500_r0.005

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 \
    --discount 0.9 -n 100 -l 2 -s 64 -b 5000 -lr 0.005 -rtg --exp_name q2_b5000_r0.005


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_r0.01

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 \
    --discount 0.9 -n 100 -l 2 -s 64 -b 2500 -lr 0.01 -rtg --exp_name q2_b2500_r0.01

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 \
    --discount 0.9 -n 100 -l 2 -s 64 -b 5000 -lr 0.01 -rtg --exp_name q2_b5000_r0.01

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 0.02 -rtg --exp_name q2_b1000_r0.02

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 \
    --discount 0.9 -n 100 -l 2 -s 64 -b 2500 -lr 0.02 -rtg --exp_name q2_b2500_r0.02

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 --ep_len 1000 \
    --discount 0.9 -n 100 -l 2 -s 64 -b 5000 -lr 0.02 -rtg --exp_name q2_b5000_r0.02
```

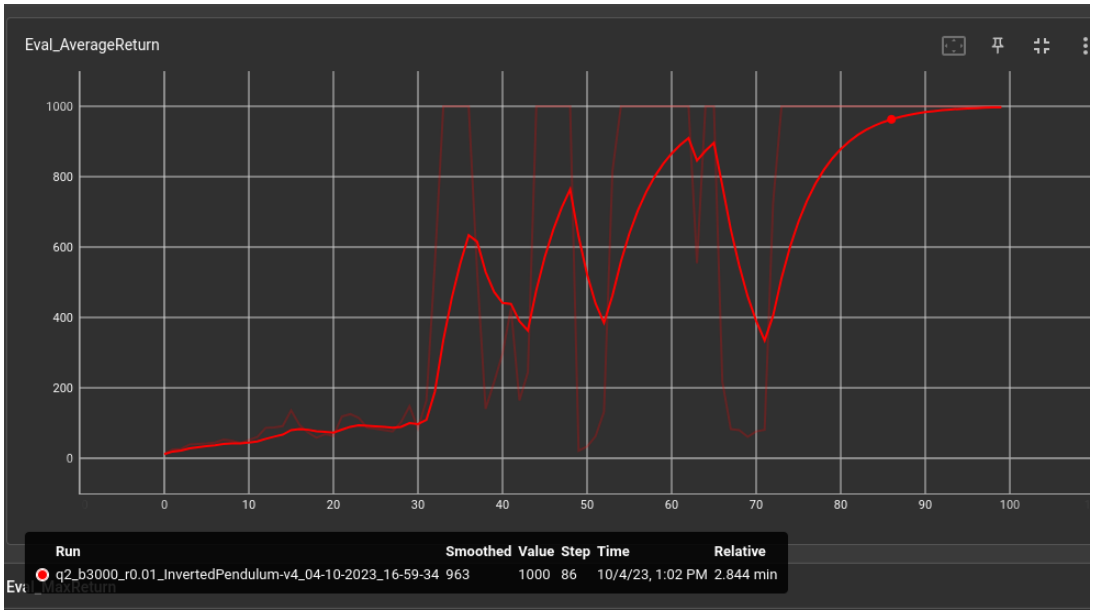
5.2.2 smallest b* and largest r* (same run) – [5 points]

Q5.2.2

Smallest batch size (**b***) = 3000
Largest learning rate (**r***) = 0.01

5.2.3 Plot – [5 points]

Q5.2.3



Smoothing = 0.8 | Batch size = 3000 | Learning Rate = 0.01

7 More Complex Experiments

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

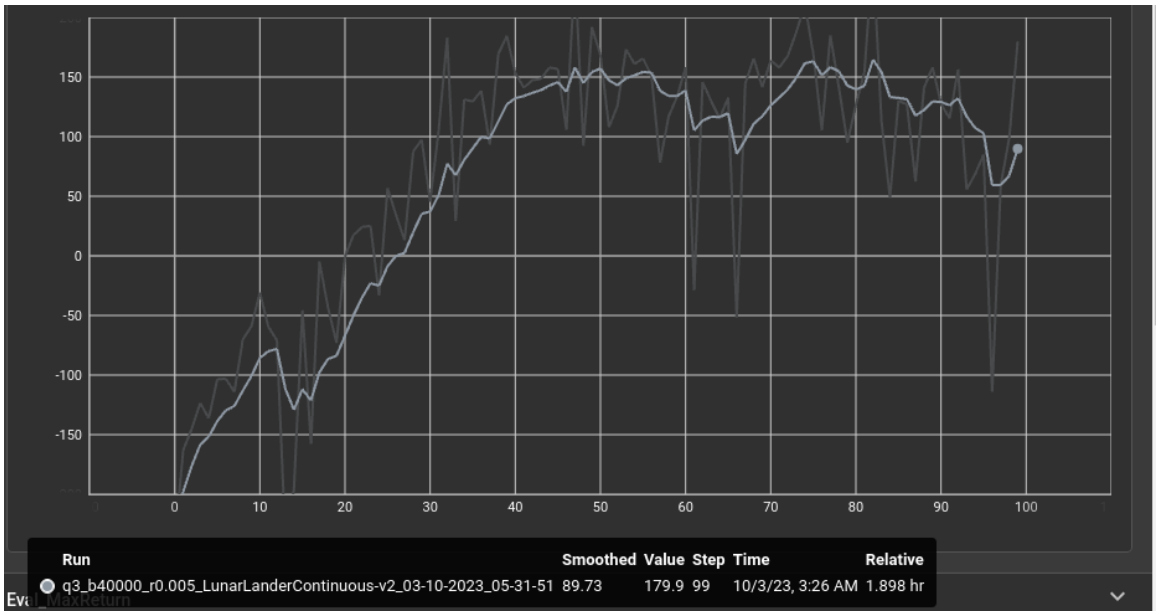
7.1.1 Configurations

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 40000 -lr 0.005 \
    --reward_to_go --nn_baseline --exp_name q3_b40000_r0.005
```

7.1.2 Plot – [10 points]

Q7.1.2



Smoothing = 0.8 | Batch size = 40000 | Learning Rate = 0.005

7.2 Experiment 4 (HalfCheetah) – [30 points]

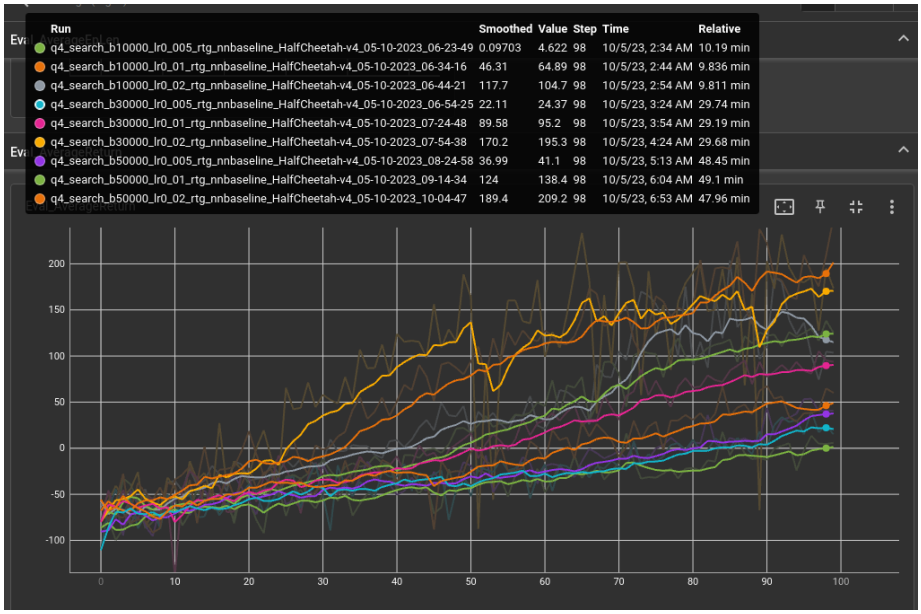
7.2.1 Configurations

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 0.005 -rtg --nn_baseline --exp_name q4_search_b10000_lr0_005_rtg_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 -rtg --nn_baseline --exp_name q4_search_b10000_lr0_01_rtg_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

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 --nn_baseline --exp_name q4_search_b30000_lr0_005_rtg_nnbaseline

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 --nn_baseline --exp_name q4_search_b30000_lr0_01_rtg_nnbaseline

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 --nn_baseline --exp_name q4_search_b30000_lr0_02_rtg_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 0.005 -rtg --nn_baseline --exp_name q4_search_b50000_lr0_005_rtg_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 0.01 -rtg --nn_baseline --exp_name q4_search_b50000_lr0_01_rtg_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_search_b50000_lr0_02_rtg_nnbaseline
```

7.2.2 Plot – [10 points]

Q7.2.2



Smoothing = 0.8 | Batch size = [10000, 30000, 50000] | Learning Rate = [0.005, 0.01, 0.02]

7.2.3 Optimal b* and r* – [3 points]

Q7.2.3

Optimal batch size (**b***) = 50000
Optimal learning rate (**r***) = 0.02

7.2.4 Describe how b* and r* affect task performance – [7 points]

Q7.2.4

b* - A larger batch size implies more data available during training leading to faster convergence to the optimal value estimator - this results in better performance on the required task. In this particular case b* was set to 50000.
r* - The learning rate is highly dependent on the task and the loss landscape of the RL problem. In this particular case having a high reward prevents the agent from getting stuck in local minimas and helps converge to a global optima.

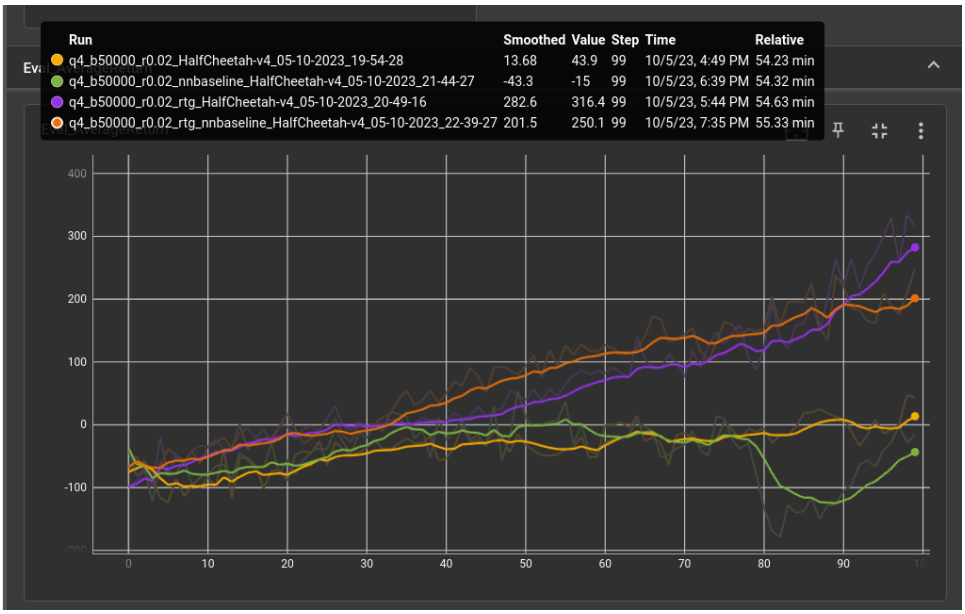
7.2.5 Configurations with optimal b* and r* – [3 points]

Q7.2.5

```
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_b<b*>_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
```

7.2.6 Plot for four runs with optimal b* and r* – [7 points]

Q7.2.6



Smoothing = 0.8 | Batch size = 50000 | Learning Rate = 0.02

8 Implementing Generalized Advantage Estimation

8.1 Experiment 5 (Hopper) – [20 points]

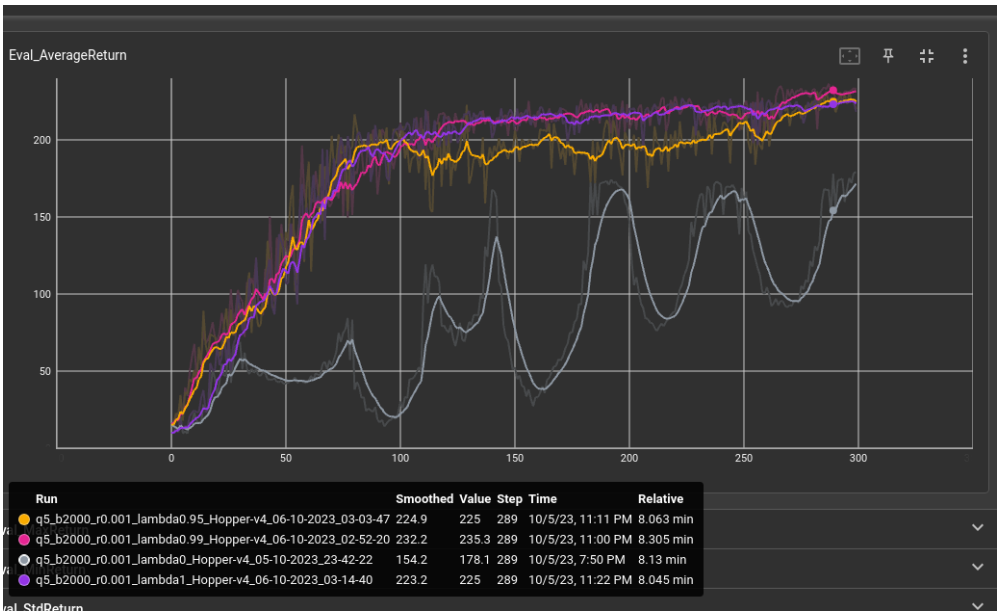
8.1.1 Configurations

Q8.1.1

```
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 0 --exp_name q5_b2000_r0.001_lambda0
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 0.95 --exp_name q5_b2000_r0.001_lambda0.95
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 0.99 --exp_name q5_b2000_r0.001_lambda0.99
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 1 --exp_name q5_b2000_r0.001_lambda1
```

8.1.2 Plot – [13 points]

Q8.1.2



Smoothing = 0.8 | λ = [0, 0.95, 0.99,1]

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

Q8.1.3

The value of λ determines how strongly we weigh the n-step advantage estimates and is tuned to get a bias-variance trade-off. A higher value of λ means that the trained network will be biased towards A_n whereas a low value of λ means that it will be less biased towards the n-step advantage estimate however this will result in a higher variance.
As we see from the above plots the network where λ was set to 0 has a very high variance and $\lambda = 1$ has a very low variance but is biased towards the A_n. The best results are obtained for the plot with $\lambda =$ **0.99.**

9 Bonus! (optional)

9.1 Parallelization – [15 points]

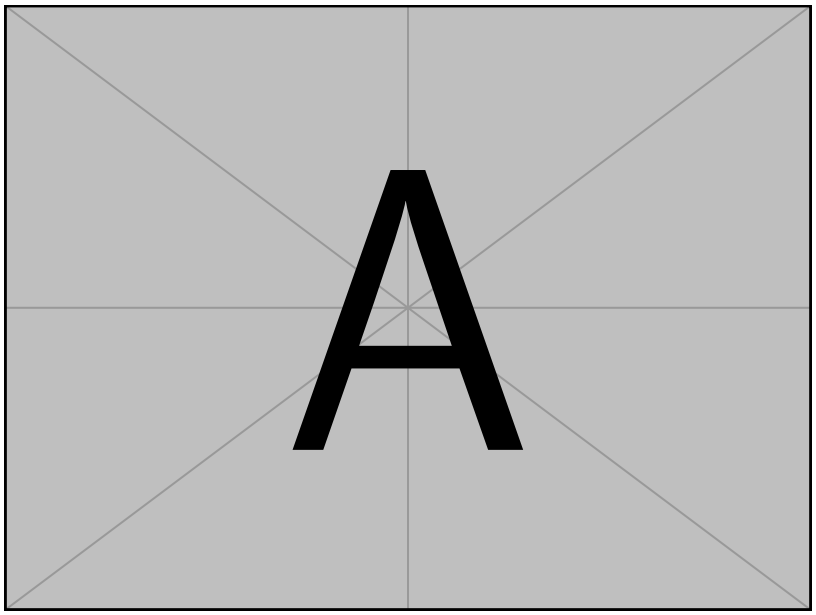
Q9.1

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 \
```