Assignment 2: Policy Gradient

Andrew ID: sghatge
Collaborators: None
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

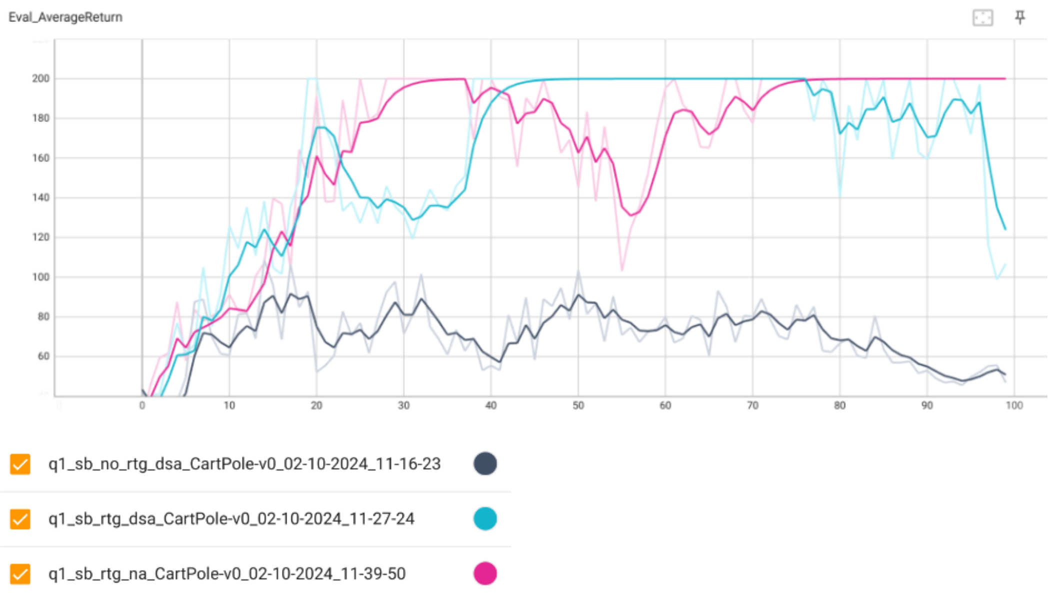
Q5.1.1

```
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

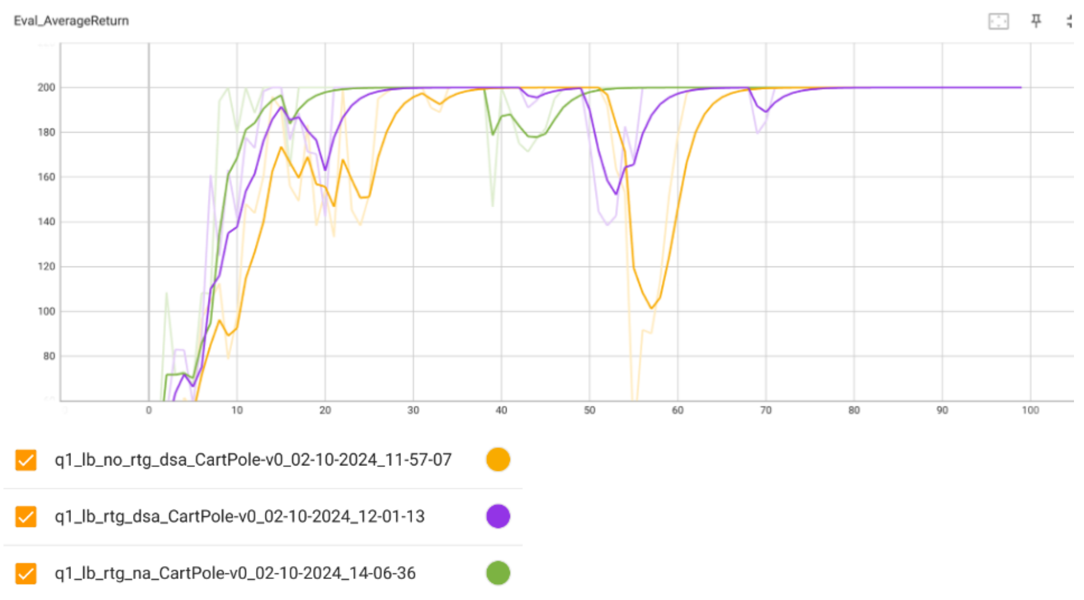
5.1.2.1 Small batch – [5 points]

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

In both the case, i.e. for small batch sizes and large batch sizes, the reward-to-go has better performance without advantage-standardization. This is because of less variance in the data. The policy converges faster because it gives more importance to the immediate rewards rather than the past rewards.

5.1.3.2 Advantage standardization – [5 points]

Q5.1.3.2

From the graphs above we can see that the performance has drastically improved for small batch size. This is because small batch sizes tend to have higher variance in advantage estimates. Thus with advantage standardization the variance decreases and the policy converges at the right value. Large batch sizes however have less variance because of more data, so advantage standardization does not affect it much.

5.1.3.3 Batch size – [5 points]

Q5.1.3.3

Batch size did make an impact. In small batches, the gradients are more prone to high variance because of less data. This leads to noisy updates, making training inconsistent. Large batch size helps to reduce this variance by averaging gardients over more data points, this giving more stable updates. The training process for large batch size converges more smoothly and consistently.

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 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 2000 -lr 0.01 -rtg \
    --exp_name q2_b2000_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 3000 -lr 0.05 -rtg \
    --exp_name q2_b3000_r0.05

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 \
    --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 20000 -lr 0.001 -rtg \
    --exp_name q2_b20000_r0.001

python rob831/scripts/run_hw2.py --env_name InvertedPendulum-v4 \
    --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 20000 -lr 0.01 -rtg \
    --exp_name q2_b20000_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 30000 -lr 0.01 -rtg \
    --exp_name q2_b30000_r0.01
```

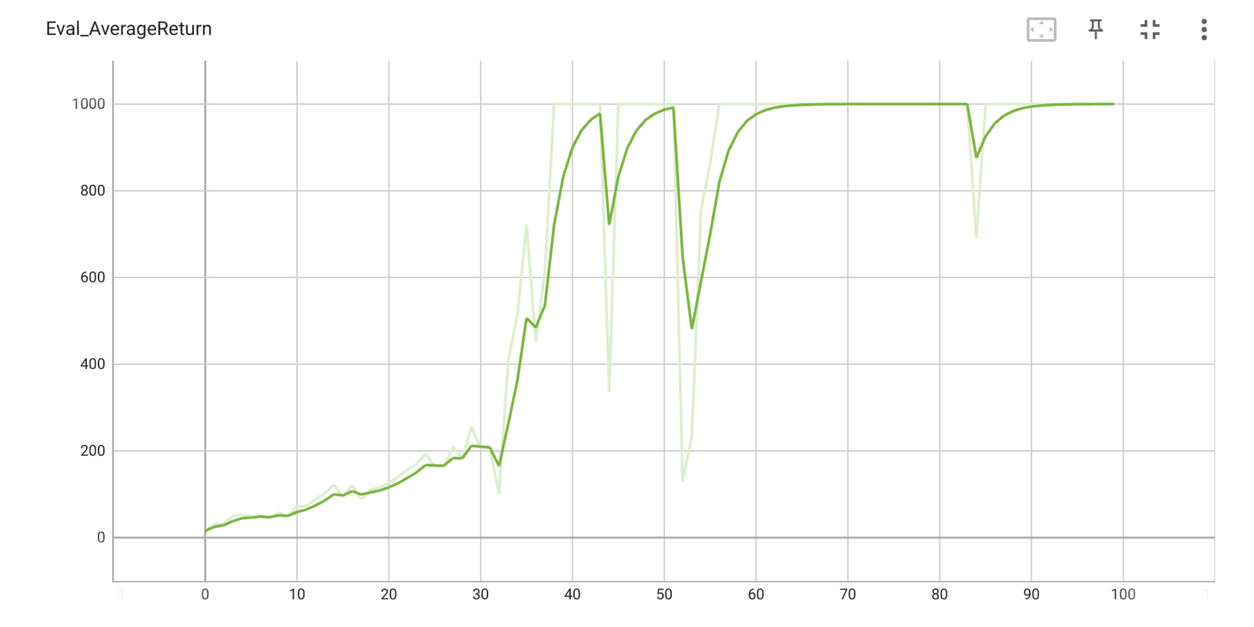
5.2.2 smallest b* and largest r* (same run) – [5 points]

Q5.2.2

The smallest batch size and the largest learning rate which got the optimum score is -
b* = 30000
r* = 0.01

5.2.3 Plot – [5 points]

Q5.2.3



7 More Complex Experiments

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

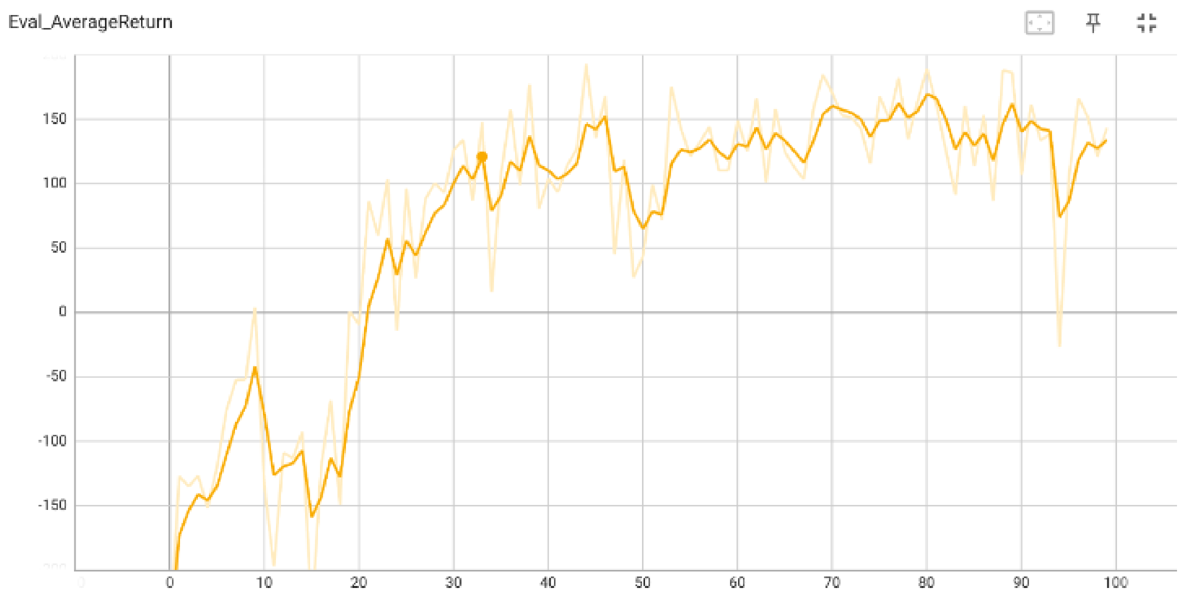
7.1.1 Configurations

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]

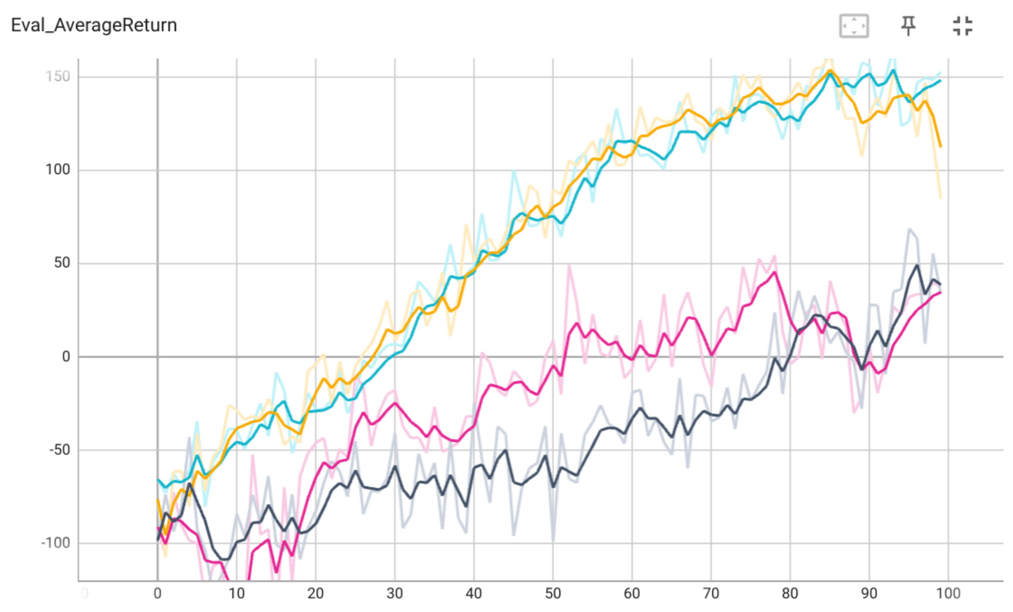
7.2.1 Configurations

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

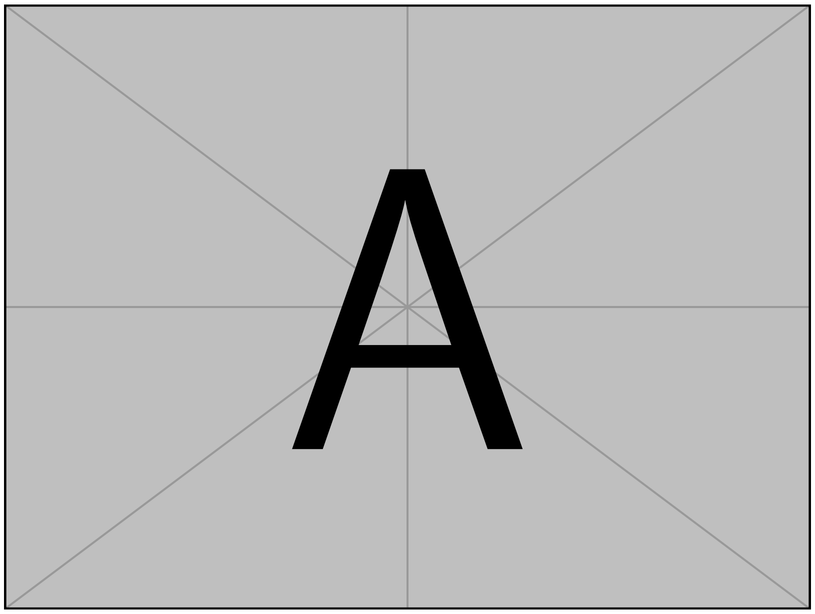


7.2.3 (Optional) Optimal b* and r* – [3 points]

Q7.2.3

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

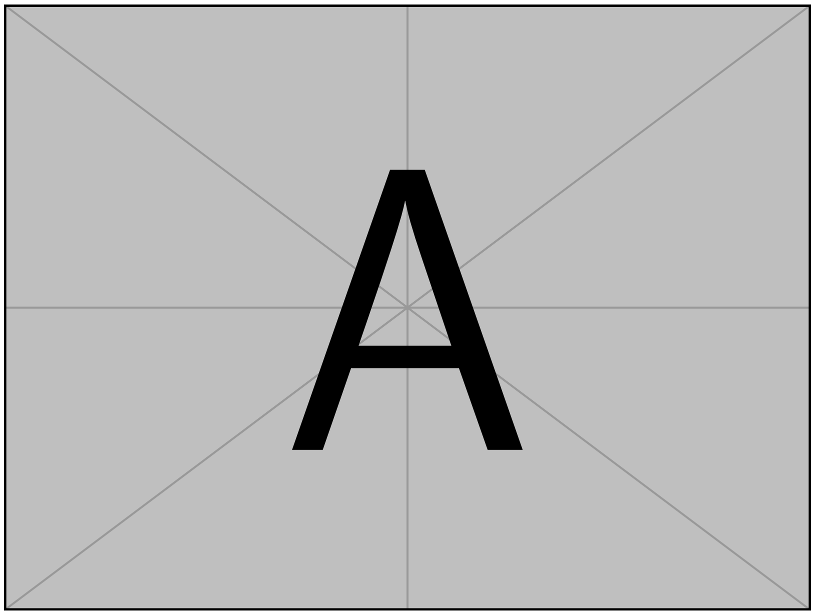
7.2.6 (Optional) Configurations with optimal b* and r* – [3 points]

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]

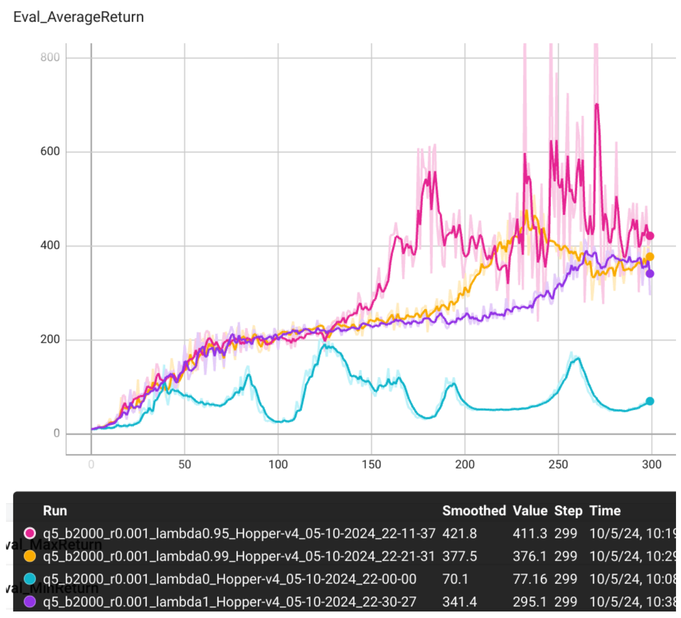
8.1.1 Configurations

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



	Run	Smoothed	Value	Step	Time
●	q5_b2000_r0.001_lambda0.95_Hopper-v4_05-10-2024_22-11-37	421.8	411.3	299	10/5/24, 10:1
●	q5_b2000_r0.001_lambda0.99_Hopper-v4_05-10-2024_22-21-31	377.5	376.1	299	10/5/24, 10:2
●	q5_b2000_r0.001_lambda0_Hopper-v4_05-10-2024_22-00-00	70.1	77.16	299	10/5/24, 10:0
●	q5_b2000_r0.001_lambda1_Hopper-v4_05-10-2024_22-30-27	341.4	295.1	299	10/5/24, 10:3

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

Q8.1.3

For $\lambda = 0$ the network does not learn at all because we are not using the baseline network.
For $\lambda = 0.99$ we see that the network is kind of stable but it does not converge to the desired value.
For $\lambda = 1$ we see it being the most stable but converges slowly
For $\lambda = 0.95$ this value converges to the desired value

9 Bonus! (optional)

9.1 Parallelization – [15 points]

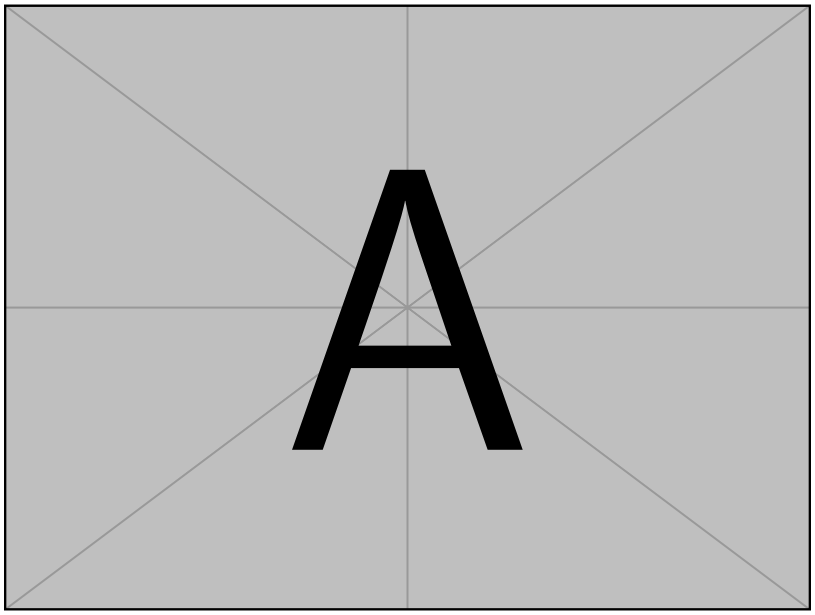
Q9.1

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 \
```