Homework 2

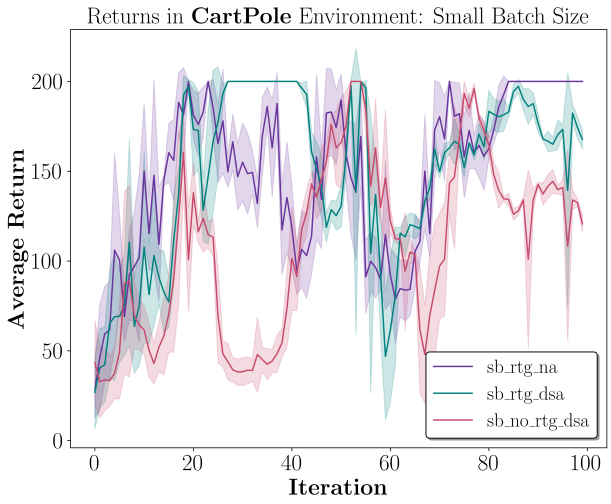
Author:
Reece D. Huff

CS 285: Deep Reinforcement Learning

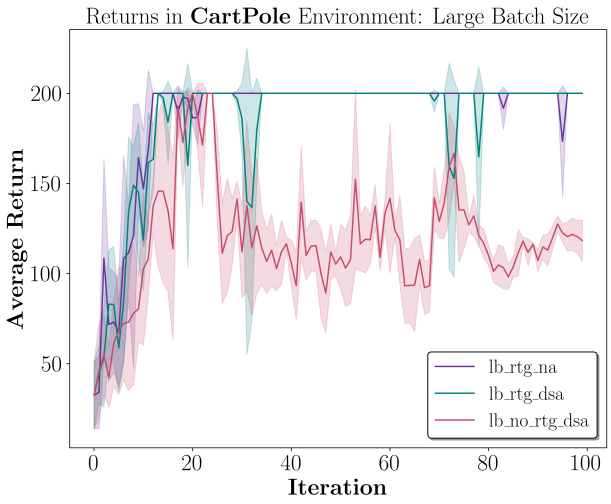
Dr. Sergey Levine

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Experiment 1



(a) Small batch experiments.



(b) Large batch experiments.

Figure 1: All of the results from Experiment 1.

- Which value estimator has better performance without advantage-standardization: the trajectory-centric one, or the one using reward-to-go?

  The one that uses reward-to-go outperforms the trajectory-centric one when the batch size is large, and almost always outperforms when the batch size is small.

- Did advantage standardization help?

  Advantage standardization generally helped. When the batch size is large, the performance is similar irrespective of advantage standardization, and when the batch size is small, the advantage standardization generally helps.

- Did the batch size make an impact?

  Yes, absolutely. Having a large batch size makes a huge positive impact on performance. However, interestingly, a large batch size does not seem to help when using the trajectory-centric algorithm without advantage standardization.

`experiment1.sh`

```
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
-dsa --exp_name q1_sb_no_rtg_dsa
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
-rtg -dsa --exp_name q1_sb_rtg_dsa
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 \
-rtg --exp_name q1_sb_rtg_na
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 5000 \
-dsa --exp_name q1_lb_no_rtg_dsa
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 5000 \
-rtg -dsa --exp_name q1_lb_rtg_dsa
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 5000 \
-rtg --exp_name q1_lb_rtg_na
```

Experiment 2

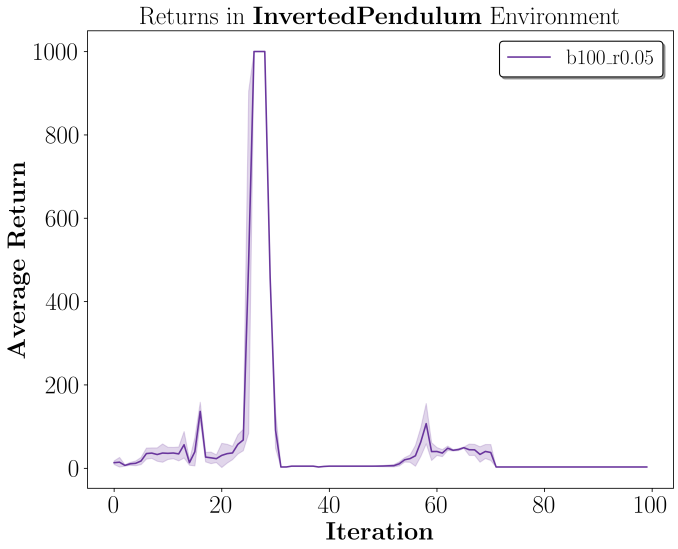


Figure 2: An example of a run reaching the maximum of 1000 mean return. As stated in the legend, this run corresponds to a batch size of 100 and learning rate of 0.05.

As shown in the bash script below, I tested `batch_size` $\in$ [100, 500, 1000] and `learning_rate` $\in$ [0.02, 0.03, 0.05].

experiment2.sh

```
for i in 100 500 1000
do
  for j in 0.02 0.03 0.05
        do
    python cs285/scripts/run_hw2.py --env_name InvertedPendulum-v4 \
    --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b $i -lr $j -rtg \
    --exp_name q2_b${i}_r${j}
  done
done
```

Experiment 3

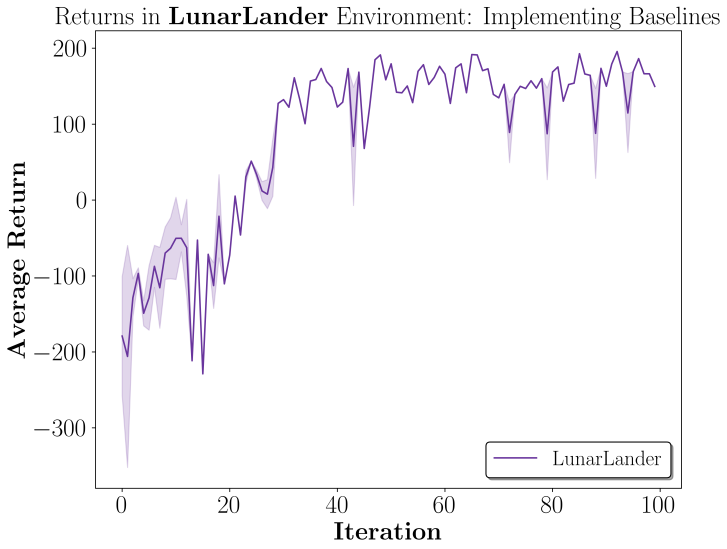


Figure 3: Validating the performance of the RL algorithm with neural network baselines implemented in the **LunarLander** environment.

experiment3.sh

```
python cs285/scripts/run_hw2.py \
--env_name LunarLanderContinuous-v2 --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
```

Experiment 4

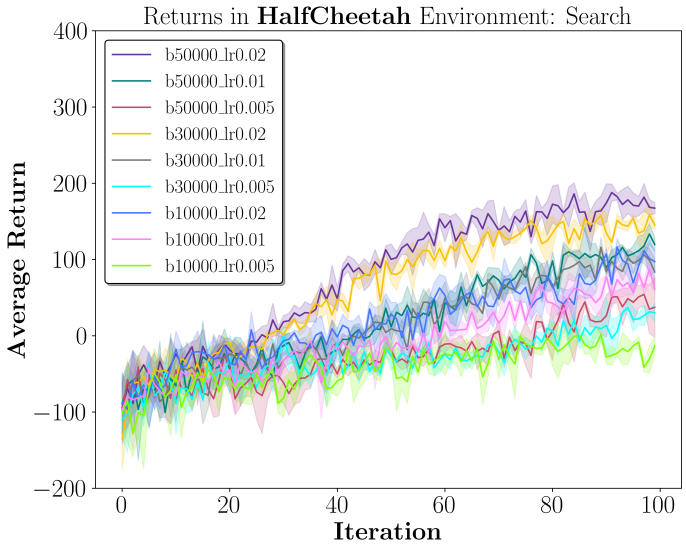


Figure 4: Searching for the optimal batch size and learning rate in the **HalfCheetah** environment.

- Describe in words how the batch size and learning rate affected task performance.
  Increasing batch size and learning rate both improve the performance.

`experiment4.sh`

```
for i in 10000 30000 50000
do
        for j in 0.005 0.01 0.02
        do
                python cs285/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150
    \
    --discount 0.95 -n 100 -l 2 -s 32 -b $i -lr $j -rtg --nn_baseline \
    --exp_name q4_search_b${i}_lr${j}_rtg_nnbaseline
        done
done
```

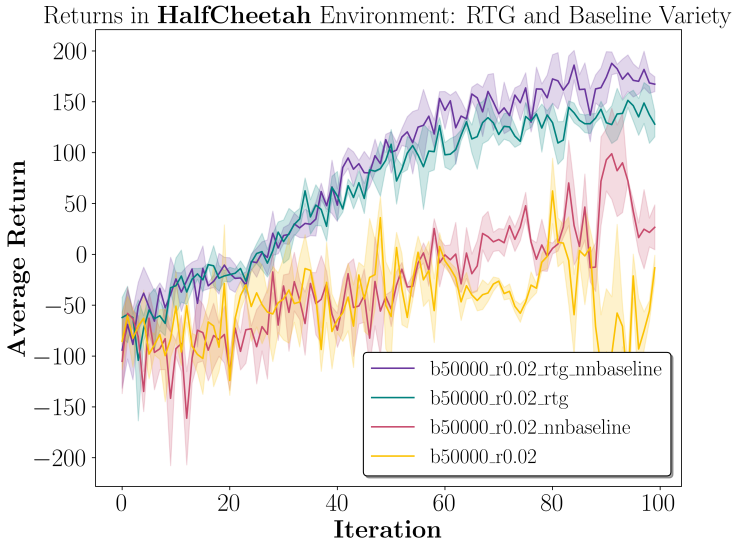


Figure 5: Using the optimal hyperparameters (batch size of 50000 and learning rate of 0.02) in the **HalfCheetah** environment as the baseline and reward-to-go is varied.

experiment4_optimal.sh

```
for i in 50000
do
  for j in 0.02
  do
    python cs285/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b $i -lr $j \
    --exp_name q4_b${i}_r${j}
    python cs285/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b $i -lr $j -rtg \
    --exp_name q4_b${i}_r${j}_rtg
    python cs285/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b $i -lr $j --nn_baseline \
    --exp_name q4_b${i}_r${j}_nnbaseline
    python cs285/scripts/run_hw2.py --env_name HalfCheetah-v4 --ep_len 150 \
    --discount 0.95 -n 100 -l 2 -s 32 -b $i -lr $j -rtg --nn_baseline \
    --exp_name q4_b${i}_r${j}_rtg_nnbaseline
  done
done
```

Experiment 5

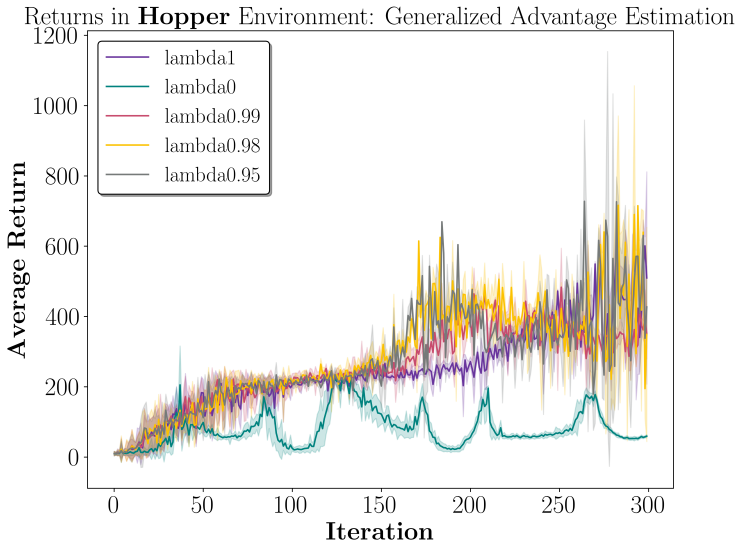


Figure 6: Performance in the **Hopper** environment using generalized advantage estimation (GAE) as λ is varied.

- Describe in words how λ affected task performance.

  Generally speaking, increasing λ improves the performance in the **Hopper** environment. However, it is not always so clear. For example, it appears that $\lambda = 0.95$ is the highest performer.

`experiment5.sh`

```
for i in 0 0.95 0.98 0.99 1
do
  python cs285/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 $i \
  --exp_name q5_b2000_r0.001_lambda$i
done
```