Homework 2

Juanwu Lu (3037432593)

(M.Sc. Civil Engineering, UC Berkeley)

Experiment 1 CartPole

Answers:

- The **reward-to-go** estimator has a better performance without advantage-standardization. Compare the green with orange curves in both figure 1(a) and (b), reward-to-go value estimators converge faster and are more stable across the training process.
- From my experiment results, advantage standardization helps in small batch experiments, but does not in large batch experiments.
- From my experiment results, batch size did make an impact to the training, with a larger batch size helps stablize the training.

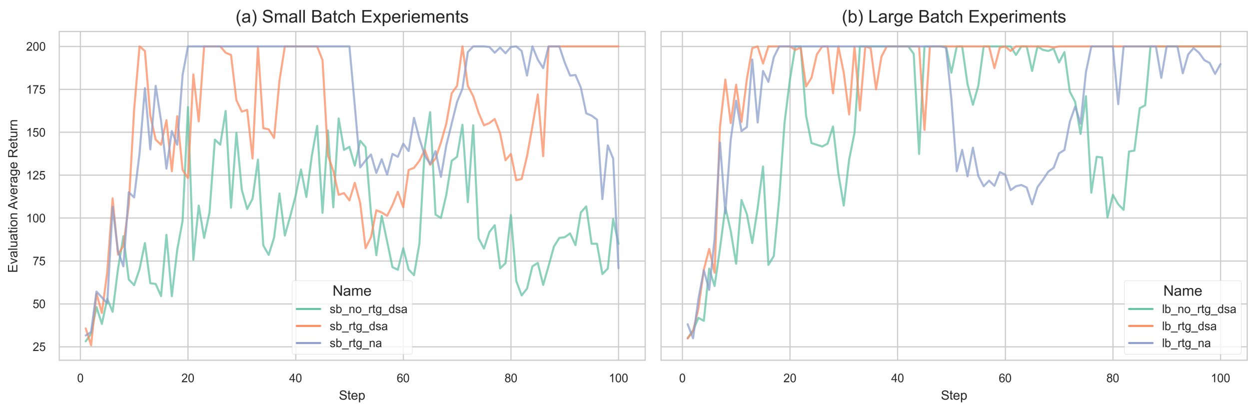

Figure 1. Visualization of learning curves for (a) small batch experiments and (b) large batch experiments.

Command-line Codes

```
echo 'Running small batch w/o reward_to_go w/ standardized_advatages';
python $1 --env_name CartPole-v0 -n 100 -b 1000 -dsa --exp_name q1_sb_no_rtg_dsa;

echo 'Running small batch w/ reward_to_go w/ standardized_advatages';
python $1 --env_name CartPole-v0 -n 100 -b 1000 -rtg -dsa --exp_name q1_sb_rtg_dsa;

echo 'Running small batch w/ reward_to_go w/o standardized_advatages';
python $1 --env_name CartPole-v0 -n 100 -b 1000 -rtg --exp_name q1_sb_rtg_na;
```

```
echo 'Running large batch w/o reward_to_go w/ standardized_advatages';
python $1 --env_name CartPole-v0 -n 100 -b 5000 -dsa --exp_name q1_lb_no_rtg_dsa;

echo 'Running large batch w/ reward_to_go w/ standardized_advatages';
python $1 --env_name CartPole-v0 -n 100 -b 5000 -rtg -dsa --exp_name q1_lb_rtg_dsa;

echo 'Running large batch w/ reward_to_go w/o standardized_advatages';
python $1 --env_name CartPole-v0 -n 100 -b 5000 -rtg --exp_name q1_lb_rtg_na;
```

Experiment 2 InvertedPendulum

Answers:

From my experiments, the optimal setting combination is `b*=500` and `r*=0.01`. Using this setting, I obtain a learning curve as shown in figure 2. Although this settings reaches a best score 1000 the fastest, the average return is unstable and shows occasion extreme decays.

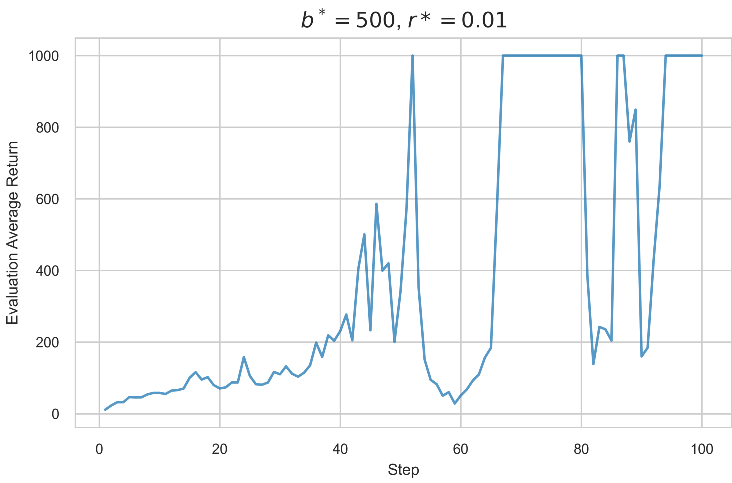

Figure 2. Learning curve with optimal settings.

Command-line Codes

```
echo "Searching for optimal batch and learning rate...";
for BATCH in 500 1000 2500 5000 7500
do
    for LR in 0.005 0.001 0.005 0.01 0.05
    do
        echo "Now running on batch_size=${BATCH}, learning rate=${LR}."
        NAME="q2_b${BATCH}_r${LR}";
        python $1 --env_name InvertedPendulum-v4 --ep_len 1000 --discount 0.9 -n 100 -l 2 -s
            64 -b $BATCH -lr $LR -rtg --exp_name $NAME;
    done
done
```

Experiment 3 LunarLander

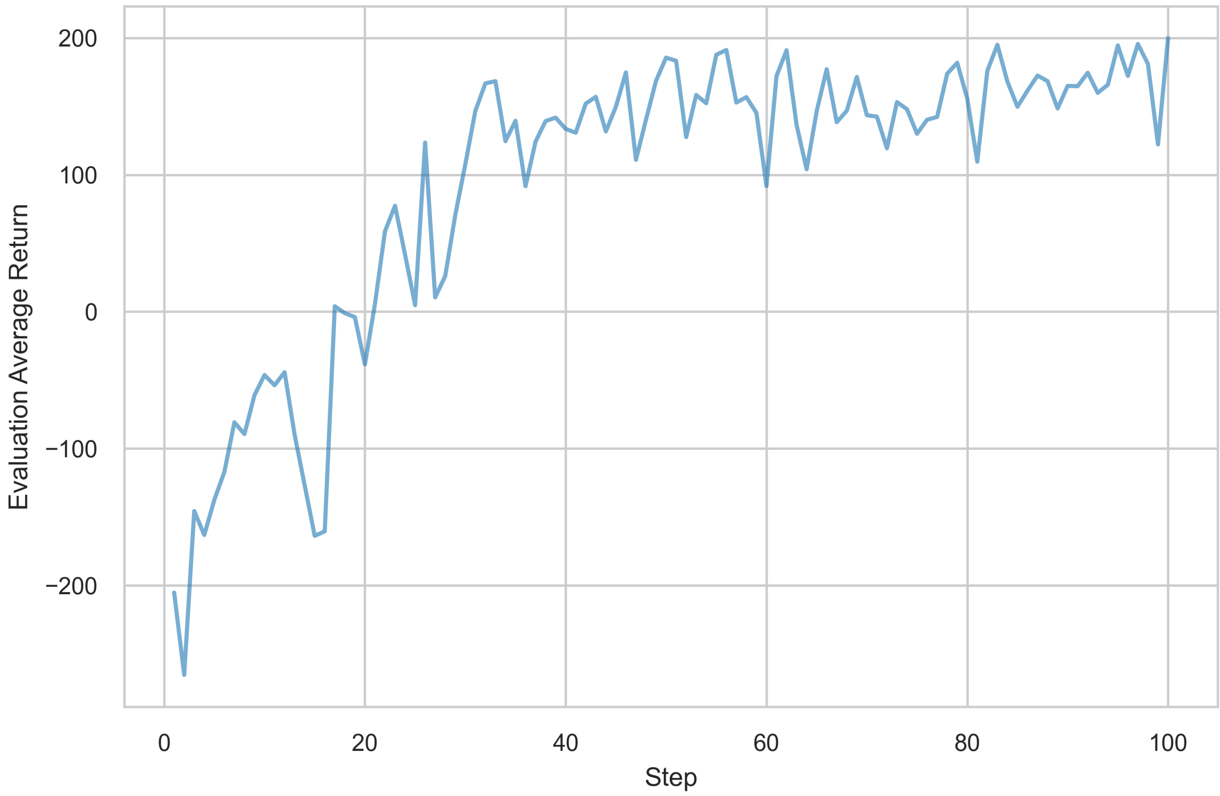

Figure 3. Learning curves for the LunarLander-v2.

Command-line Codes

```
echo "Running LunarLander with reward-to-go estimator.";
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 HalfCheetah

Answer:

The results from trying out different settings are shown in the following figure 4. From my experiments, the optimal settings are `b*=30000` and `lr*=0.02`.

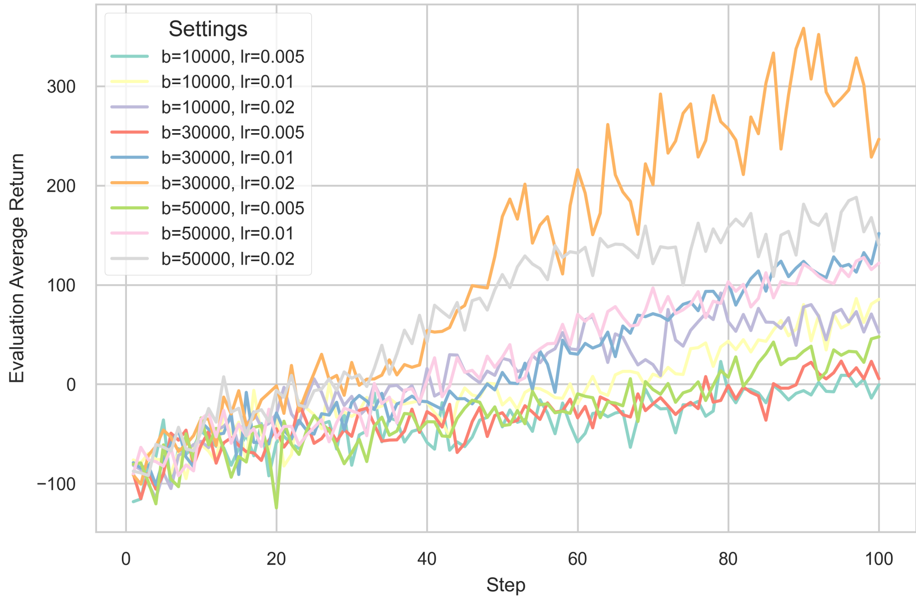

Figure 4. Learning curves for the `HalfCheetah` experiments of different settings.

Using the optimal settings to investigate the effect of reward-to-go and value estimator network, the results are shown in the following figure 5. The switching from vanilla policy gradients to a reward-to-go formulation has more significant acceleration to the training converge than the value estimator.

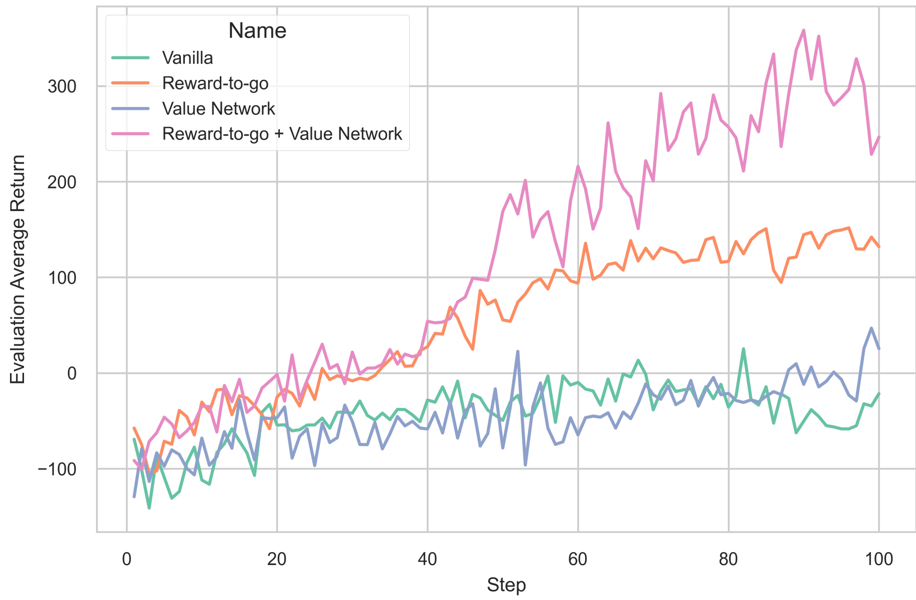

Figure 5. Learning curves of different policy gradient settings under optimal batch size and learning rate.

```
echo "Search for optimal batch_size and learning rate for HalfCheetah.";
for BATCH in 10000 30000 50000
do
    for LR in 0.005 0.01 0.02
    do
        echo "Now running on batch_size=${BATCH}, learning_rate=${LR}.";
        NAME="q4_search_b${BATCH}_lr${LR}_rtg_nnbaseline";
        python $1 --env_name HalfCheetah-v4 --ep_len 150 --discount 0.95 -n 100 -l 2 -s 32 -b
            $BATCH -lr $LR -rtg --nn_baseline --exp_name $NAME;
    done
done


BATCH=$2
LR=$3

echo "Running with optimal batch_size and learning rate for HalfCheetah.";
echo "Running baseline";
python $1 --env_name HalfCheetah-v4 --ep_len 150 --discount 0.95 -n 100 -l 2 -s 32 -b $BATCH
    -lr $LR --exp_name q4_b${BATCH}_r${LR};
echo "Running reward-to-go";
python $1 --env_name HalfCheetah-v4 --ep_len 150 --discount 0.95 -n 100 -l 2 -s 32 -b $BATCH
    -lr $LR -rtg --exp_name q4_b${BATCH}_r${LR}_rtg;
echo "Running with baseline neural network";
python $1 --env_name HalfCheetah-v4 --ep_len 150 --discount 0.95 -n 100 -l 2 -s 32 -b $BATCH
    -lr $LR --nn_baseline --exp_name q4_b${BATCH}_r${LR}_nnbaseline;
echo "Running with all";
python $1 --env_name HalfCheetah-v4 --ep_len 150 --discount 0.95 -n 100 -l 2 -s 32 -b $BATCH
    -lr $LR -rtg --nn_baseline --exp_name q4_b${BATCH}_r${LR}_rtg_nnbaseline
```

Experiment 5 HopperV4

Answers:

The averaged evaluation returns with respect to training steps given different λ settings is as shown in figure 6. Results show that the evaluation performance at the same timestep increases with a λ increasing from 0.00 to 1.00, meaning reducing bias has benefits on the trainig procedure.

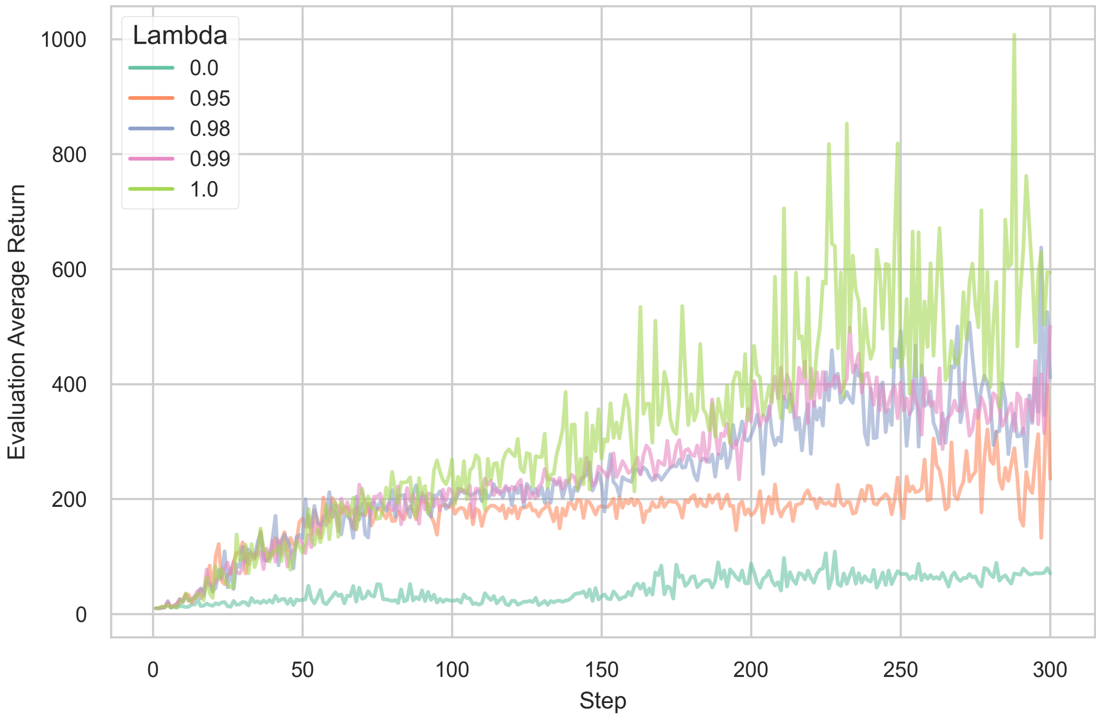

Figure 6. Learning curves for the `Hopper-v4` with different λ settings.

Command-line Codes

```
echo "Search for the optimal GAE lambda setting...";
for LAMBDA in 0.00 0.95 0.98 0.99 1.00
do
    echo "Now running on gae_lambda = ${LAMBDA}.";
    NAME="q5_b2000_r0.001_lambda${LAMBDA}";
    python $1 --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 $LAMBDA
        --exp_name $NAME;
done
```