1 Experiment 1

1.1 Small Batch Experiments

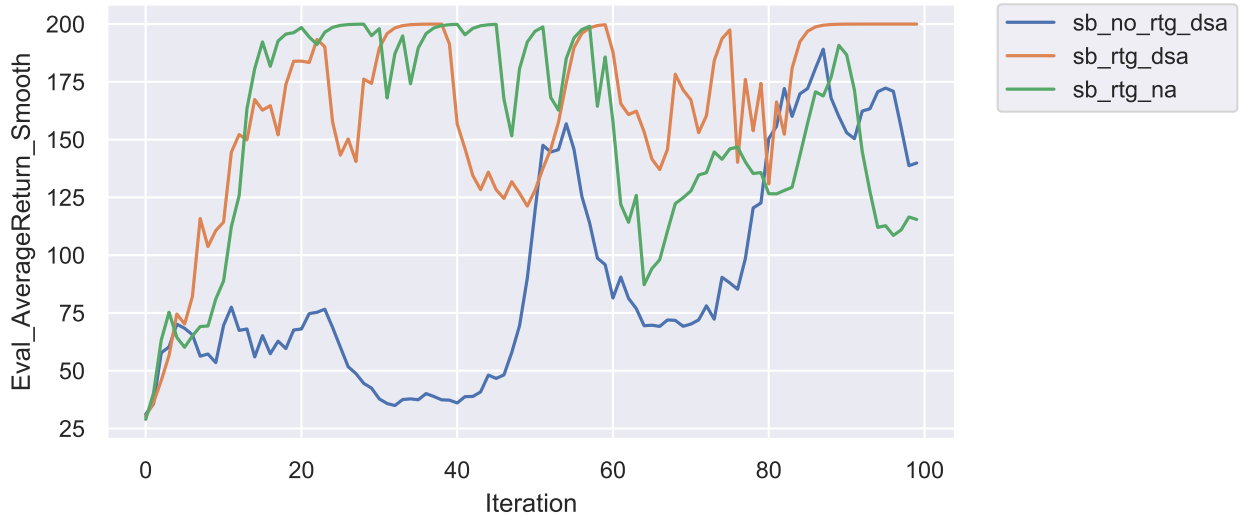

Figure 1: Small Batch Experiments

Listing 1: Exact command line configurations

```
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
```

The blue line used the trajectory-centric value estimator and did not standardize the advantages, which performed the worst. The orange line used the reward-to-go value estimator and did not standardize advantages, while the green line used the reward-to-go and did standardize advantages. The orange and green lines performed similarly, which seems to indicate that advantage standardization did not help that much.

1.2 Large Batch Experiments

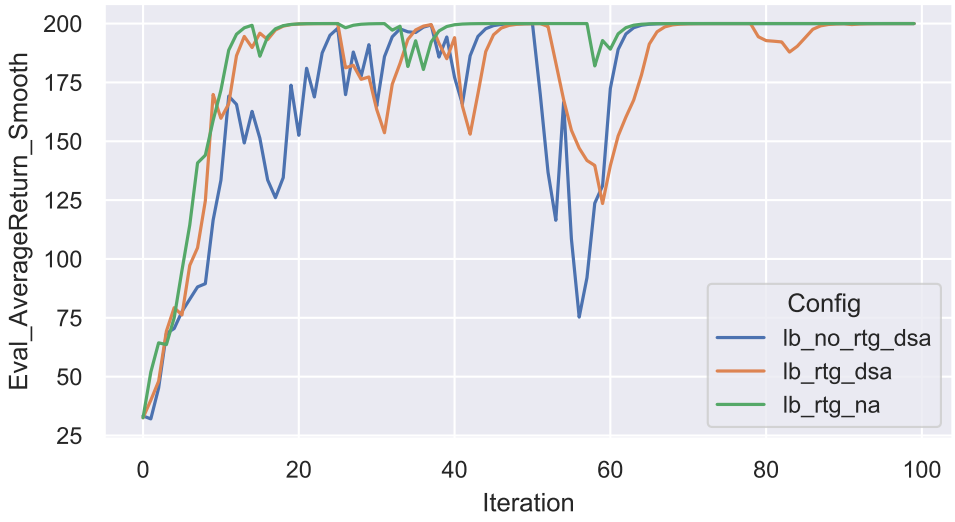

Figure 2: Large Batch Experiments

In Figure 2, we can see the learning curve of the large batch experiments. The following commands were run:

Listing 2: Exact command line configurations

```
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
```

The blue line had the trajectory-centric value estimator and did not standardize advantages. The orange line used the reward-to-go and did not standardize advantages, while the green line used the reward-to-go and did standardize advantages.

Without advantage standardization, the reward-to-go had a better performance. In both the large and small batch experiments, the reward-to-go value estimator converged in fewer iterations than the trajectory-centric estimator. Advantage standardization did not seem to help, the reward-to-go experiments with and without advantage standardization seem to converge at roughly the same rate. Larger batch sizes significantly helped the policy, as the experiments with larger batch sizes converged in fewer iterations than the small batch size experiments.

2 Experiment 2: InvertedPendulum

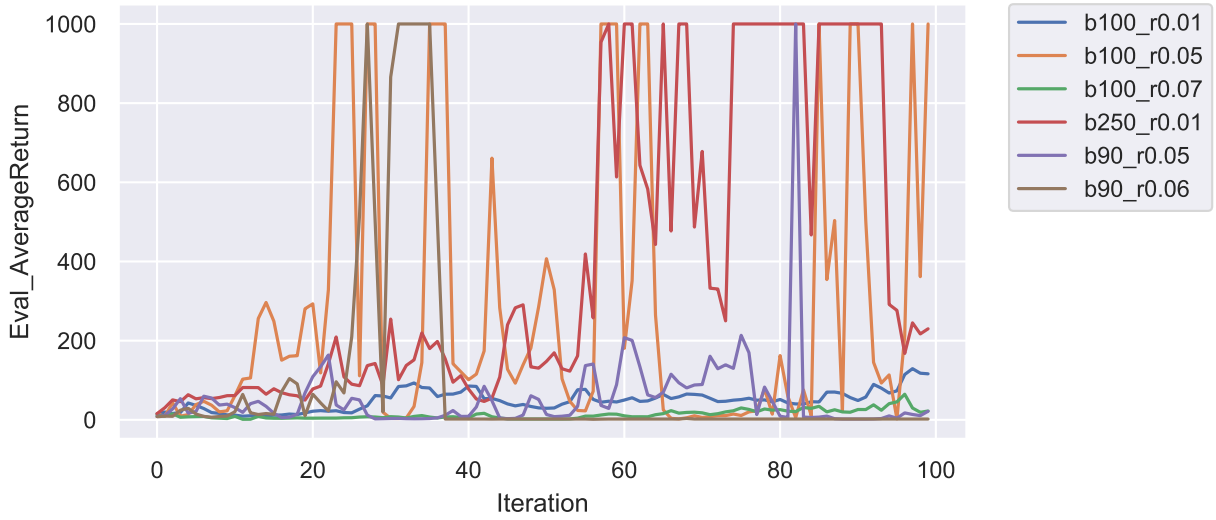

Figure 3: LunarLanderContinuous-v2

Listing 3: Exact command line configurations

```
python cs285/scripts/run_hw2.py --env_name InvertedPendulum-v2 --ep_len 1000 --discount 0.9
    -n 100 -l 2 -s 64 -b <b*> -lr <r*> -rtg --exp_name q2_b<b*>_r<r*>
```

The smallest batch size and largest learning rate I was able to get to converge to the optimum was 100 and 0.05 respectively. I used the above command except with `<b*>` replaced with 100 and `<r*>` replaced with 0.05. However, the policy was still quite unstable, and deviations from those values would cause the policy to not converge at all. A batch size of 90 and learning rate of 0.05 did not converge and a batch size of 100 with a learning rate of 0.07 did not converge either.

3 Experiment 3: LunarLander

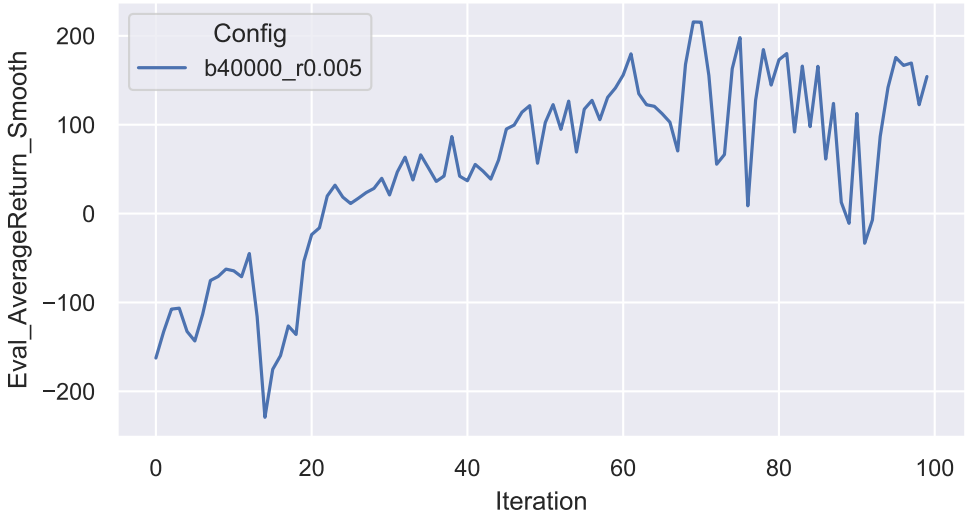

Figure 4: LunarLanderContinuous-v2

This experiment was run with the following command:

Listing 4: Exact command line configurations

```
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
```

4 Experiment 4: HalfCheetah

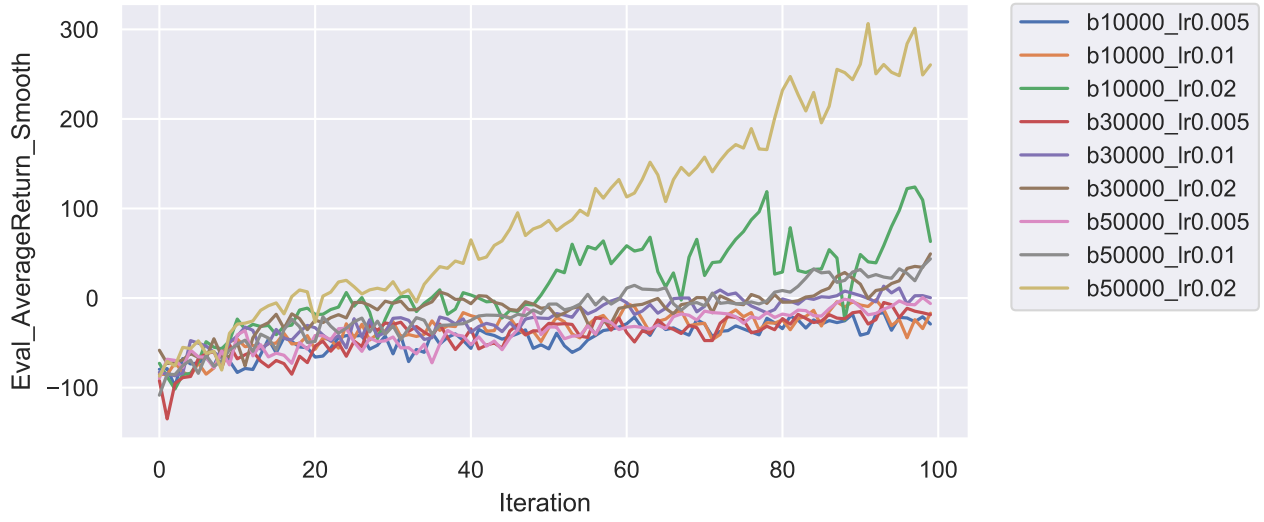

Figure 5: HalfCheetah Hyperparameter Search

Listing 5: Exact command line configurations

```
1 python cs285/scripts/run_hw2.py --env_name HalfCheetah-v2 --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
2 python cs285/scripts/run_hw2.py --env_name HalfCheetah-v2 --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
3 python cs285/scripts/run_hw2.py --env_name HalfCheetah-v2 --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
4 python cs285/scripts/run_hw2.py --env_name HalfCheetah-v2 --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
5 python cs285/scripts/run_hw2.py --env_name HalfCheetah-v2 --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
6 python cs285/scripts/run_hw2.py --env_name HalfCheetah-v2 --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
7 python cs285/scripts/run_hw2.py --env_name HalfCheetah-v2 --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
8 python cs285/scripts/run_hw2.py --env_name HalfCheetah-v2 --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
9 python cs285/scripts/run_hw2.py --env_name HalfCheetah-v2 --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
```

As seen in the graph, the yellowish-green line, which corresponds with `batch_size=50000` and `lr=0.02` performed the best. Batch size alone did not have the same impact as a properly tuned learning rate. Of the experiments with `batch_size=50000`, only the one with `lr=0.02` performed noticeably better than the other experiments. The other experiments with that batch size performed on par with the other experiments. Interestingly, the experiment with `batch_size=10000` and `lr=0.02` performed better than nearly all of the other experiments (but not as well as the one with `batch_size=50000`), which indicates that a properly tuned learning rate has a larger effect on the performance of the policy. Since `batch_size=50000` and `lr=0.02`

performed the best, I ran the next four experiments with those hyperparameter values, which is shown in the next figure.

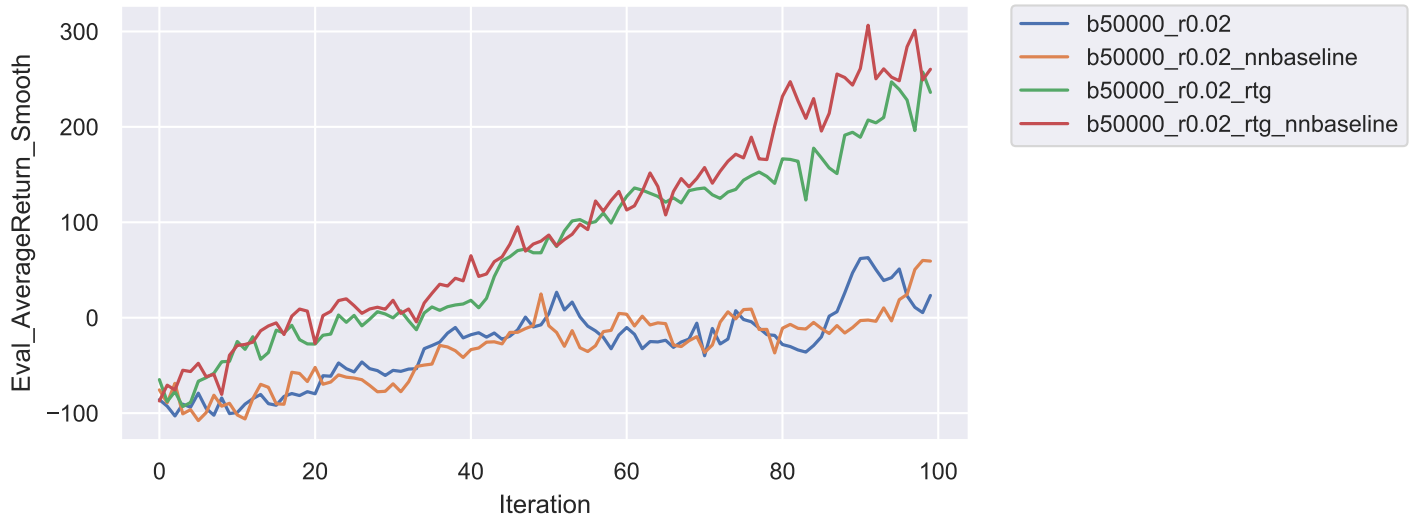

Figure 6: HalfCheetah Experiments

From the figure, we can see that the reward-to-go value estimator had a significant effect on the performance of the policy gradient. The two experiments with reward-to-go performed significantly better than the experiments that did not use it, and the baseline only improved the policy slightly (maybe even not at all).