Homework #3

Mahan Fathi

March 15, 2022

1 Question 1

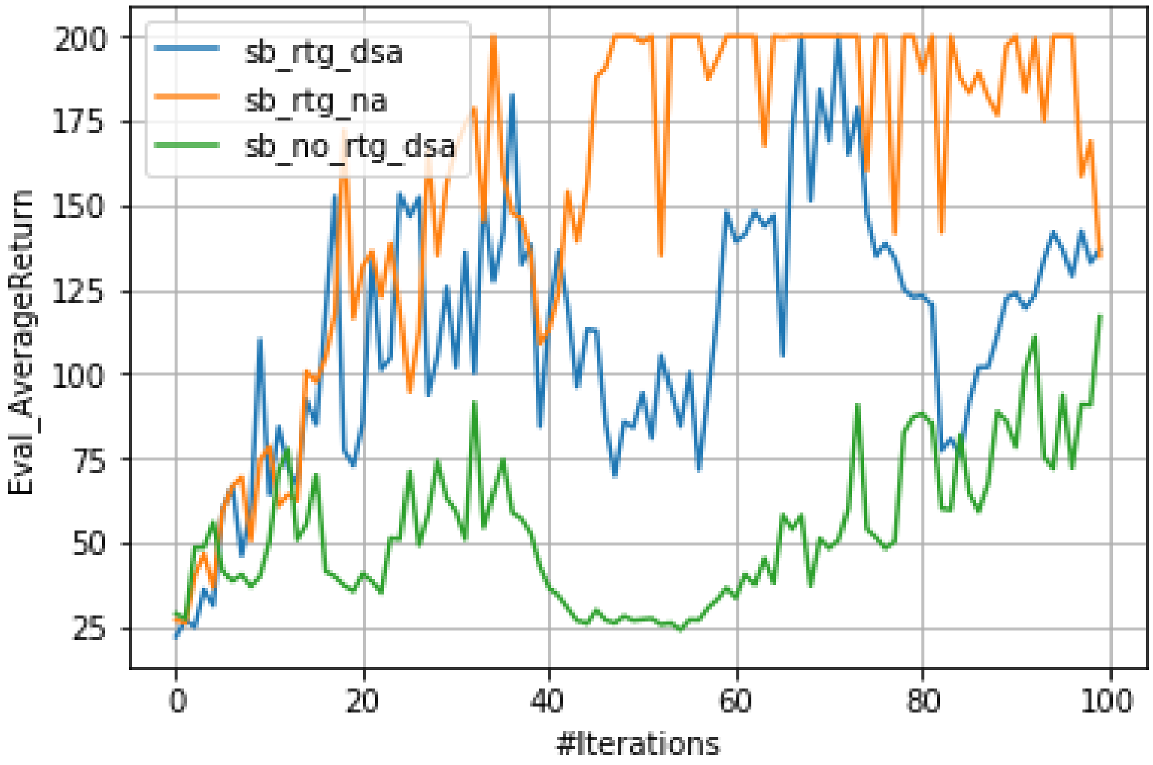


Figure 1: **Q1** `CartPole-v0` with short `batch_size=1000`

- Questions:
  - `rtg` is outperforming naive REINFORCE as expected. Using reward-to-go reduces the variance of the gradient estimator, at no cost (unbiased).
  - Standardizing helps regardless of the batch size.
  - Using a larger batch size helps a lot with the convergence rate – 5x `batch_size` roughly speeds up convergence by 2x.

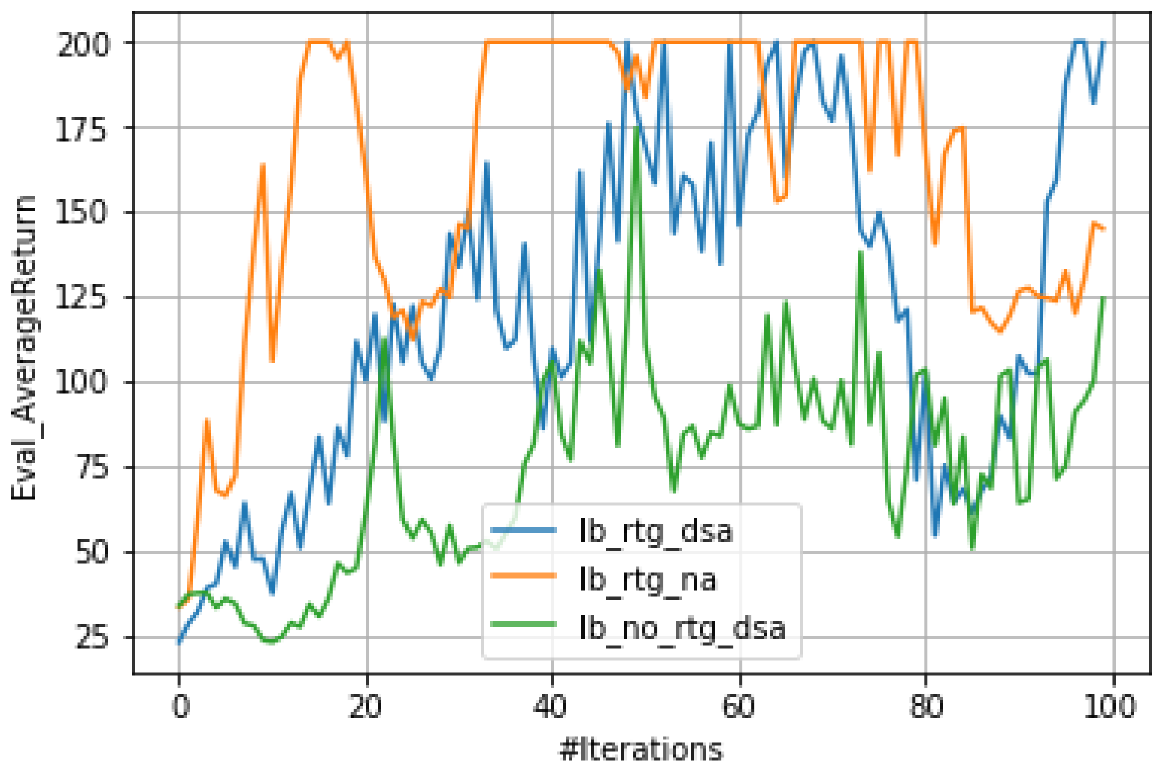
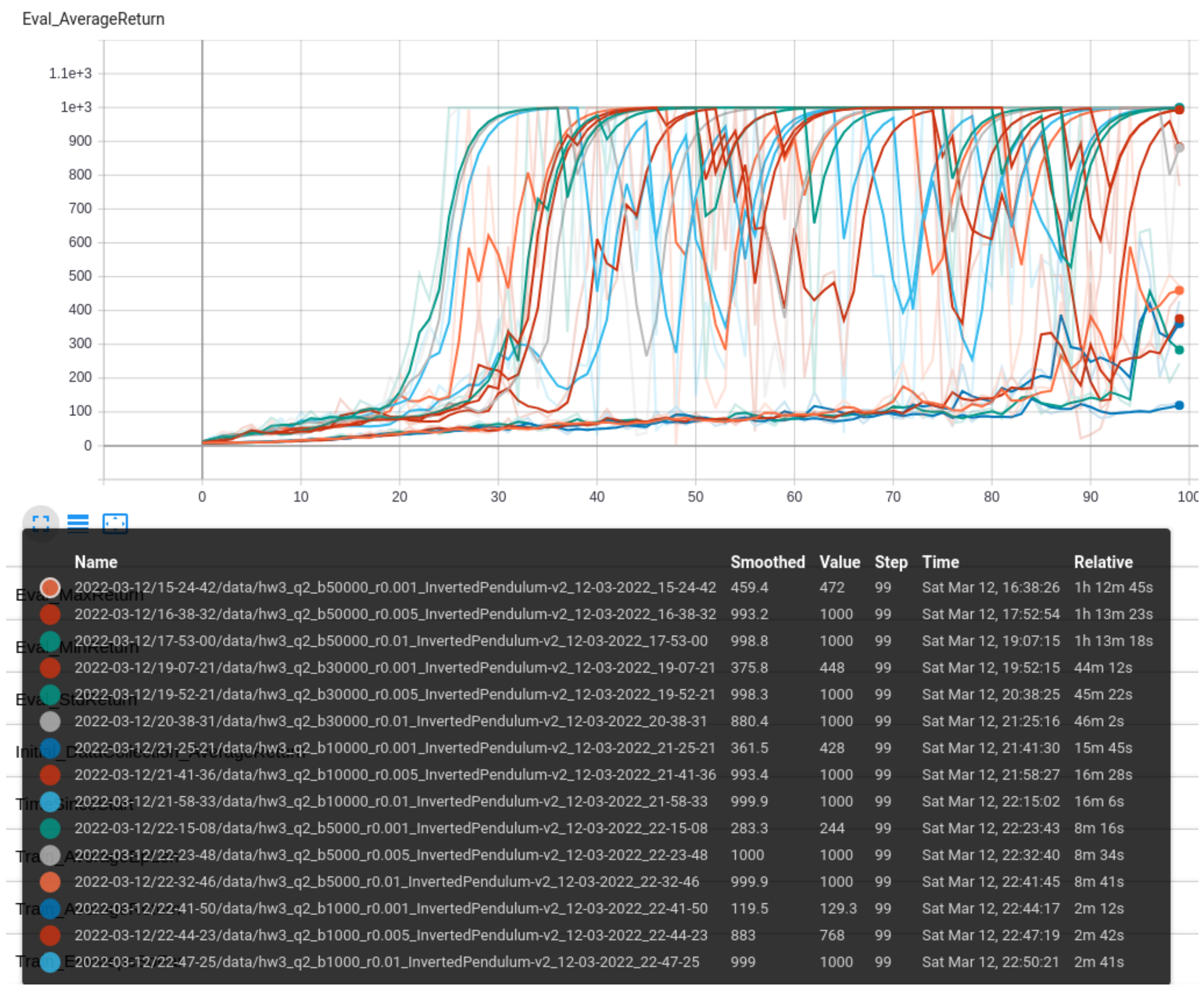


Figure 2: **Q1** `CartPole-v0` with large `batch_size=5000`

```
python run_hw3.py env_name=CartPole-v0 n_iter=100 batch_size=1000 rl_alg=reinforce \
    estimate_advantage_args.standardize_advantages=false \
    exp_name=q1_sb_no_rtg_dsa estimate_advantage_args.reward_to_go=false
python run_hw3.py env_name=CartPole-v0 n_iter=100 batch_size=1000 rl_alg=reinforce \
    estimate_advantage_args.standardize_advantages=false \
    exp_name=q1_sb_rtg_dsa estimate_advantage_args.reward_to_go=true
python run_hw3.py env_name=CartPole-v0 n_iter=100 batch_size=1000 rl_alg=reinforce \
    estimate_advantage_args.standardize_advantages=true \
    exp_name=q1_sb_rtg_na estimate_advantage_args.reward_to_go=true
python run_hw3.py env_name=CartPole-v0 n_iter=100 batch_size=5000 rl_alg=reinforce \
    estimate_advantage_args.standardize_advantages=false \
    exp_name=q1_lb_no_rtg_dsa estimate_advantage_args.reward_to_go=false
python run_hw3.py env_name=CartPole-v0 n_iter=100 batch_size=5000 rl_alg=reinforce \
    estimate_advantage_args.standardize_advantages=false \
    exp_name=q1_lb_rtg_dsa estimate_advantage_args.reward_to_go=true
python run_hw3.py env_name=CartPole-v0 n_iter=100 batch_size=5000 rl_alg=reinforce \
    estimate_advantage_args.standardize_advantages=true \
    exp_name=q1_lb_rtg_na estimate_advantage_args.reward_to_go=true
```

Listing 1: **Q1** Run commands

2 Question 2



	Name	Smoothed	Value	Step	Time	Relative
●	2022-03-12/15-24-42/data/hw3_q2_b50000_r0.001_InvertedPendulum-v2_12-03-2022_15-24-42	459.4	472	99	Sat Mar 12, 16:38:26	1h 12m 45s
●	2022-03-12/16-38-32/data/hw3_q2_b50000_r0.005_InvertedPendulum-v2_12-03-2022_16-38-32	993.2	1000	99	Sat Mar 12, 17:52:54	1h 13m 23s
●	2022-03-12/17-53-00/data/hw3_q2_b50000_r0.01_InvertedPendulum-v2_12-03-2022_17-53-00	998.8	1000	99	Sat Mar 12, 19:07:15	1h 13m 18s
●	2022-03-12/19-07-21/data/hw3_q2_b30000_r0.001_InvertedPendulum-v2_12-03-2022_19-07-21	375.8	448	99	Sat Mar 12, 19:52:15	44m 12s
●	2022-03-12/19-52-21/data/hw3_q2_b30000_r0.005_InvertedPendulum-v2_12-03-2022_19-52-21	998.3	1000	99	Sat Mar 12, 20:38:25	45m 22s
●	2022-03-12/20-38-31/data/hw3_q2_b30000_r0.01_InvertedPendulum-v2_12-03-2022_20-38-31	880.4	1000	99	Sat Mar 12, 21:25:16	46m 2s
●	2022-03-12/21-25-21/data/hw3_q2_b10000_r0.001_InvertedPendulum-v2_12-03-2022_21-25-21	361.5	428	99	Sat Mar 12, 21:41:30	15m 45s
●	2022-03-12/21-41-36/data/hw3_q2_b10000_r0.005_InvertedPendulum-v2_12-03-2022_21-41-36	993.4	1000	99	Sat Mar 12, 21:58:27	16m 28s
●	2022-03-12/21-58-33/data/hw3_q2_b10000_r0.01_InvertedPendulum-v2_12-03-2022_21-58-33	999.9	1000	99	Sat Mar 12, 22:15:02	16m 6s
●	2022-03-12/22-15-08/data/hw3_q2_b5000_r0.001_InvertedPendulum-v2_12-03-2022_22-15-08	283.3	244	99	Sat Mar 12, 22:23:43	8m 16s
●	2022-03-12/22-23-48/data/hw3_q2_b5000_r0.005_InvertedPendulum-v2_12-03-2022_22-23-48	1000	1000	99	Sat Mar 12, 22:32:40	8m 34s
●	2022-03-12/22-32-46/data/hw3_q2_b5000_r0.01_InvertedPendulum-v2_12-03-2022_22-32-46	999.9	1000	99	Sat Mar 12, 22:41:45	8m 41s
●	2022-03-12/22-41-50/data/hw3_q2_b1000_r0.001_InvertedPendulum-v2_12-03-2022_22-41-50	119.5	129.3	99	Sat Mar 12, 22:44:17	2m 12s
●	2022-03-12/22-44-23/data/hw3_q2_b1000_r0.005_InvertedPendulum-v2_12-03-2022_22-44-23	883	768	99	Sat Mar 12, 22:47:19	2m 42s
●	2022-03-12/22-47-25/data/hw3_q2_b1000_r0.01_InvertedPendulum-v2_12-03-2022_22-47-25	999	1000	99	Sat Mar 12, 22:50:21	2m 41s

Figure 3: **Q2** `InvertedPendulum-v2` search

Table 1: **Q2** results

*	`batch_size`	`learning_rate`
Fastest convergence	50000	0.01
min `batch_size`	1000	-
min `learning_rate`	-	0.01
min `lr` and `bs`	1000	0.01

Since `InvertedPendulum-v2` uses a continuous reward function (penalizes for deviations from the center, i.e. a properly shaped reward), we can use much lower batch sizes.

```
for batch_size in 50000 30000 10000 5000 1000; do
    for learning_rate in 0.001 0.005 0.01; do
        echo "BATCH_SIZE=$batch_size, LR=$learning_rate"
        python run_hw3.py env_name=InvertedPendulum-v2 \
            n_iter=100 ep_len=1000 computation_graph_args.n_layers=3 \
            computation_graph_args.size=64 \
            train_batch_size=$batch_size batch_size=$batch_size \
            batch_size_initial=$batch_size \
            computation_graph_args.learning_rate=$learning_rate \
            estimate_advantage_args.standardize_advantages=true \
            exp_name=q2_b${batch_size}_r${learning_rate} \
            estimate_advantage_args.reward_to_go=true rl_alg=reinforce
    done
done
```

Listing 2: **Q2** Run commands

3 Question 3

My lunar lander does not converge sadly, and I tried so many things and trust my implementation, but I have just given up. In the best cases it would get close to zero, and then just collapses afterwards.

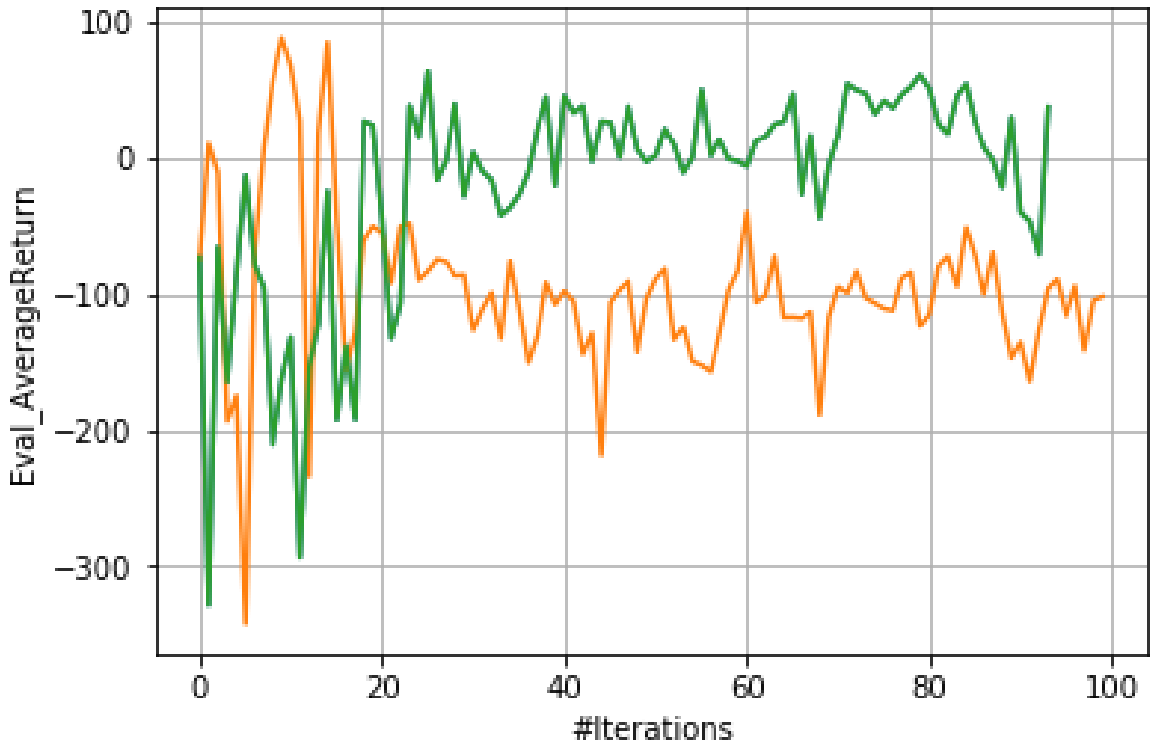


Figure 4: **Q3** Failed attempts at `LunarLanderContinuous-v2`

```
python run_hw3.py env_name=LunarLanderContinuous-v2 no_gpu=True n_iter=100 \
    ep_len=1000 computation_graph_args.n_layers=2 computation_graph_args.size=64 \
    estimate_advantage_args.discount=0.99 batch_size=40000 train_batch_size=40000 \
    batch_size_initial=40000 computation_graph_args.learning_rate=0.005 \
    estimate_advantage_args.nn_baseline=true \
    estimate_advantage_args.standardize_advantages=false \
    exp_name="q3_b100000_r0.003" estimate_advantage_args.reward_to_go=true \
    rl_alg=reinforce estimate_advantage_args.gae_lambda=false seed=7
# as a desperate measure I also tried changing the seed :(
```

Listing 3: **Q3** Run commands

4 Question 4

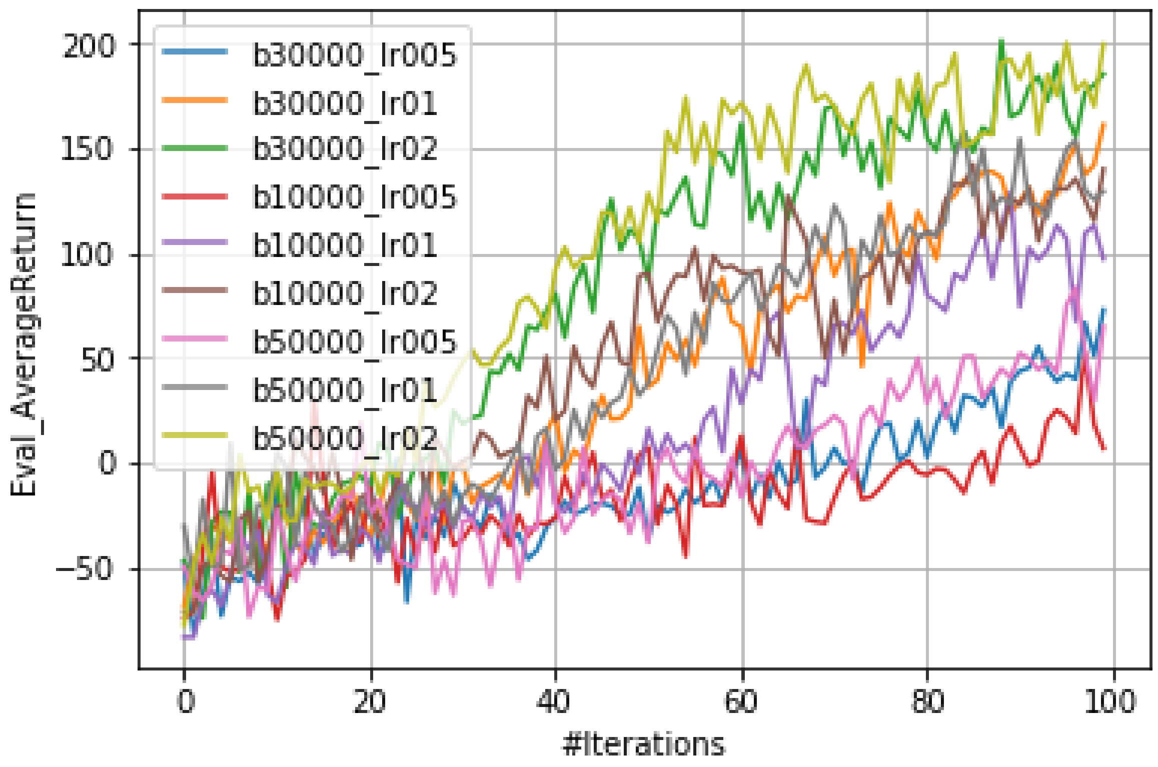


Figure 5: **Q4** Hyperparam search for `HalfCheetah-v2`

Increasing the batch size does not necessarily helps with the convergence rate, but it checks out with the overall pattern. Using a larger bach size leads to lower variance in general. As to how the learning rate affects the convergence, it basically boils down to the magnitude of the step to take in the direction suggested based on the data, which is of size `batch_size`. So given the right direction, i.e. assuming we have minimal variance in the gradient estimator, we can treat the problem as a supervised learning problem, and 0.02 seems like the perfect step size for that.

```
for batch_size in 30000 10000 50000; do
    for learning_rate in 0.005 0.01 0.02; do
        echo "BATCH_SIZE=$batch_size, LR=$learning_rate"
        python run_hw3.py env_name=HalfCheetah-v2 n_iter=100 ep_len=150 \
            computation_graph_args.n_layers=2 computation_graph_args.size=32 \
            estimate_advantage_args.discount=0.95 train_batch_size=$batch_size \
            batch_size=$batch_size batch_size_initial=$batch_size \
            computation_graph_args.learning_rate=$learning_rate \
            estimate_advantage_args.nn_baseline=true \
            estimate_advantage_args.standardize_advantages=true \
            exp_name="q4_search_b${batch_size}_lr${learning_rate}_rtg_nnbaseline" \
            estimate_advantage_args.reward_to_go=true rl_alg=reinforce
    done
done

batch_size=50000
learning_rate=0.02
python run_hw3.py env_name=HalfCheetah-v2 n_iter=100 ep_len=150 \
    computation_graph_args.n_layers=2 computation_graph_args.size=32 \
    estimate_advantage_args.discount=0.95 train_batch_size=$batch_size \
    batch_size=$batch_size batch_size_initial=$batch_size \
    computation_graph_args.learning_rate=$learning_rate \
    estimate_advantage_args.nn_baseline=false \
    estimate_advantage_args.standardize_advantages=true \
    exp_name="q4_b${batch_size}_lr${learning_rate}" \
    estimate_advantage_args.reward_to_go=false rl_alg=reinforce

python run_hw3.py env_name=HalfCheetah-v2 n_iter=100 ep_len=150 \
    computation_graph_args.n_layers=2 computation_graph_args.size=32 \
    estimate_advantage_args.discount=0.95 train_batch_size=$batch_size \
    batch_size=$batch_size batch_size_initial=$batch_size \
    computation_graph_args.learning_rate=$learning_rate \
    estimate_advantage_args.nn_baseline=false \
    estimate_advantage_args.standardize_advantages=true \
    exp_name="q4_b${batch_size}_lr${learning_rate}_rtg" \
    estimate_advantage_args.reward_to_go=true rl_alg=reinforce
```

Listing 4: **Q4** Run commands (1/2)

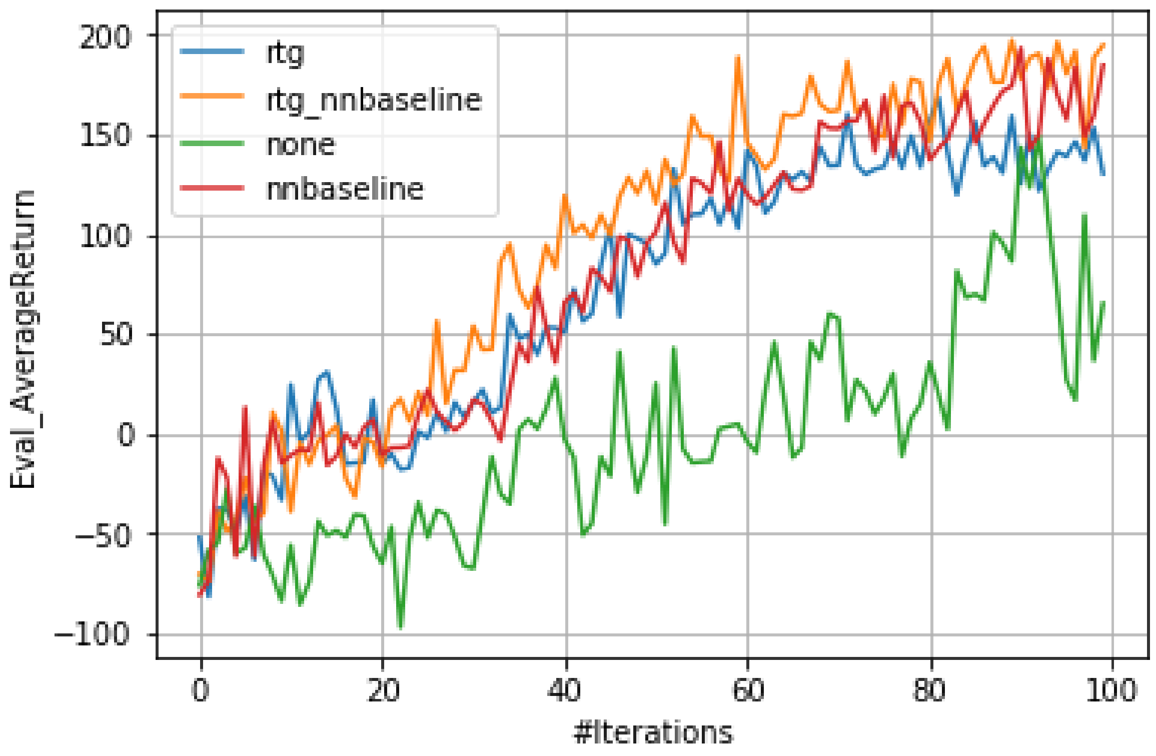


Figure 6: **Q4** `HalfCheetah-v2` with/without `rtg/nnbaseline`

```
python run_hw3.py env_name=HalfCheetah-v2 n_iter=100 ep_len=150 \
    computation_graph_args.n_layers=2 computation_graph_args.size=32 \
    estimate_advantage_args.discount=0.95 train_batch_size=$batch_size \
    batch_size=$batch_size batch_size_initial=$batch_size \
    computation_graph_args.learning_rate=$learning_rate \
    estimate_advantage_args.nn_baseline=true \
    estimate_advantage_args.standardize_advantages=true \
    exp_name="q4_b${batch_size}_lr${learning_rate}_nnbaseline" \
    estimate_advantage_args.reward_to_go=false rl_alg=reinforce

python run_hw3.py env_name=HalfCheetah-v2 n_iter=100 ep_len=150 \
    computation_graph_args.n_layers=2 computation_graph_args.size=32 \
    estimate_advantage_args.discount=0.95 train_batch_size=$batch_size \
    batch_size=$batch_size batch_size_initial=$batch_size \
    computation_graph_args.learning_rate=$learning_rate \
    estimate_advantage_args.nn_baseline=true \
    estimate_advantage_args.standardize_advantages=true \
    exp_name="q4_b${batch_size}_lr${learning_rate}_rtg_nnbaseline" \
    estimate_advantage_args.reward_to_go=true rl_alg=reinforce
```

Listing 5: **Q4** Run commands (2/2)

5 Question 5

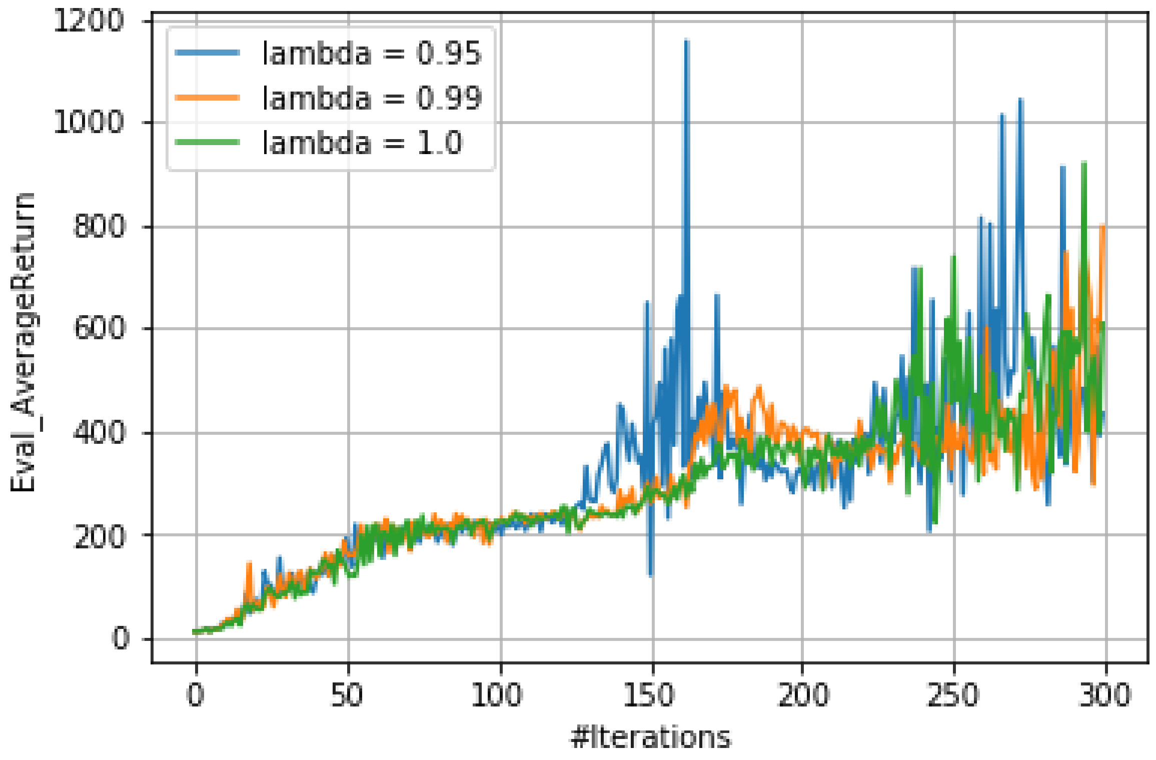


Figure 7: **Q5** GAE-λ search for `Hopper-v2`

```
batch_size=2000
learning_rate=0.001
for lambda in 0.95 0.99 1.0; do
    python run_hw3.py env_name=Hopper-v2 n_iter=300 ep_len=1000 \
        computation_graph_args.n_layers=2 computation_graph_args.size=32 \
        estimate_advantage_args.discount=0.99 train_batch_size=$batch_size \
        batch_size=$batch_size batch_size_initial=$batch_size \
        computation_graph_args.learning_rate=$learning_rate \
        estimate_advantage_args.nn_baseline=true \
        estimate_advantage_args.standardize_advantages=true \
        estimate_advantage_args.gae_lambda=$lambda \
        exp_name="q5_b${batch_size}_r${learning_rate}_lambda${lambda}" \
        estimate_advantage_args.reward_to_go=true rl_alg=reinforce
done
```

Listing 6: **Q5** Run commands

6 Question 6

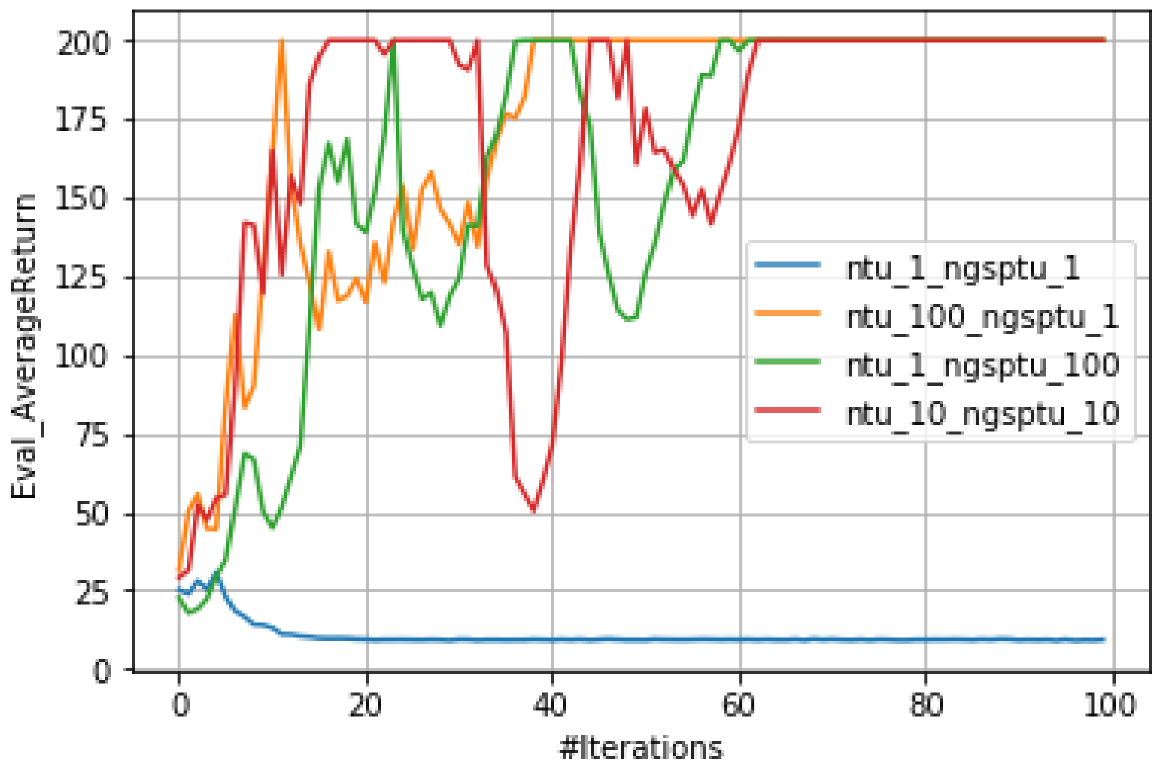


Figure 8: **Q6** `#ntu/#ngsptu` search for `CartPole-v0`

Verdict: `ntu, ngsptu = 10, 10` works best.

```
batch_size=1000
for ntu in 1 10 100; do
    for ngsptu in 1 10 100; do
        python run_hw3.py env_name=CartPole-v0 n_iter=100 \
            rl_alg=ac train_batch_size=$batch_size \
            batch_size=$batch_size batch_size_initial=$batch_size \
            estimate_advantage_args.nn_baseline=true \
            estimate_advantage_args.standardize_advantages=true \
            estimate_advantage_args.reward_to_go=true \
            exp_name="q6_ac_${ntu}_${ngsptu}" \
            computation_graph_args.num_target_updates=$ntu \
            computation_graph_args.num_grad_steps_per_target_update=$ngsptu
    done
done
```

Listing 7: **Q6** Run commands

7 Question 7

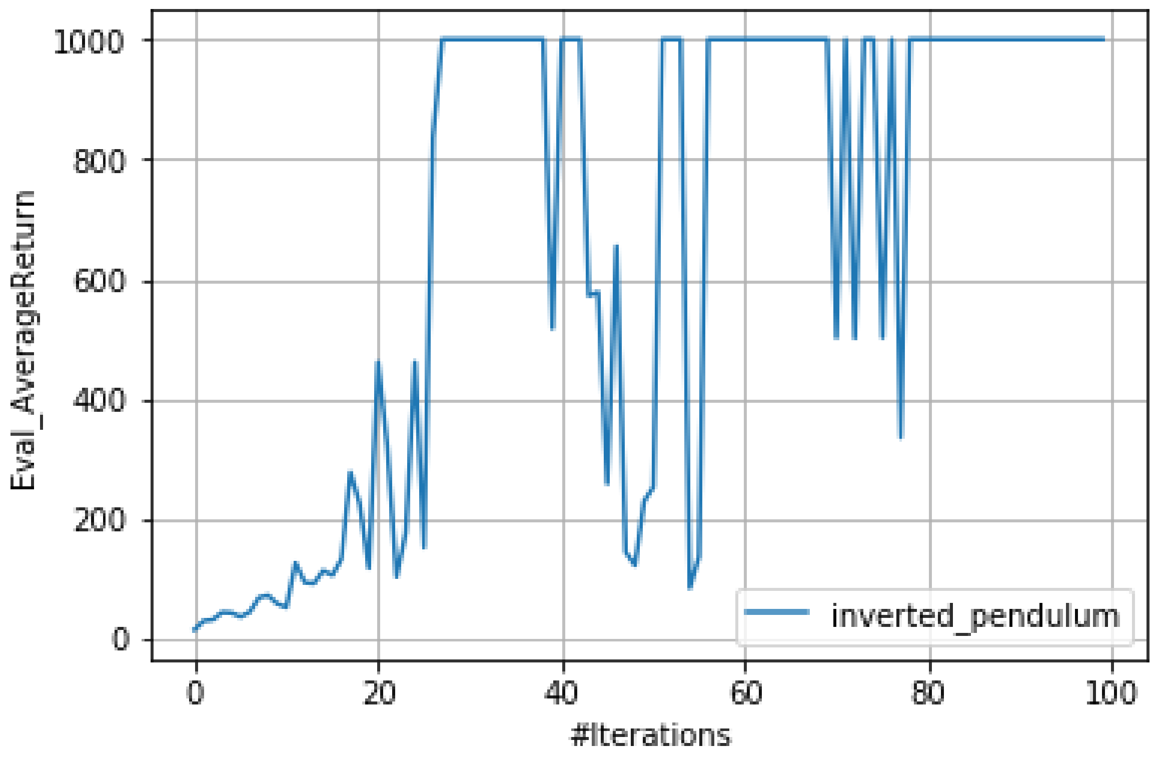


Figure 9: **Q7** AC for `InvertedPendulum-v2`

```
batch_size=5000
python run_hw3.py env_name=InvertedPendulum-v2 n_iter=100 ep_len=1000 \
    computation_graph_args.n_layers=2 computation_graph_args.size=64 \
    train_batch_size=$batch_size batch_size=$batch_size \
    batch_size_initial=$batch_size \
    computation_graph_args.learning_rate=0.01 \
    estimate_advantage_args.nn_baseline=true \
    estimate_advantage_args.standardize_advantages=true \
    exp_name="q7_10_10" estimate_advantage_args.reward_to_go=true \
    rl_alg=ac computation_graph_args.num_target_updates=10 \
    computation_graph_args.num_grad_steps_per_target_update=10
```

Listing 8: **Q7** Run commands (1/2)

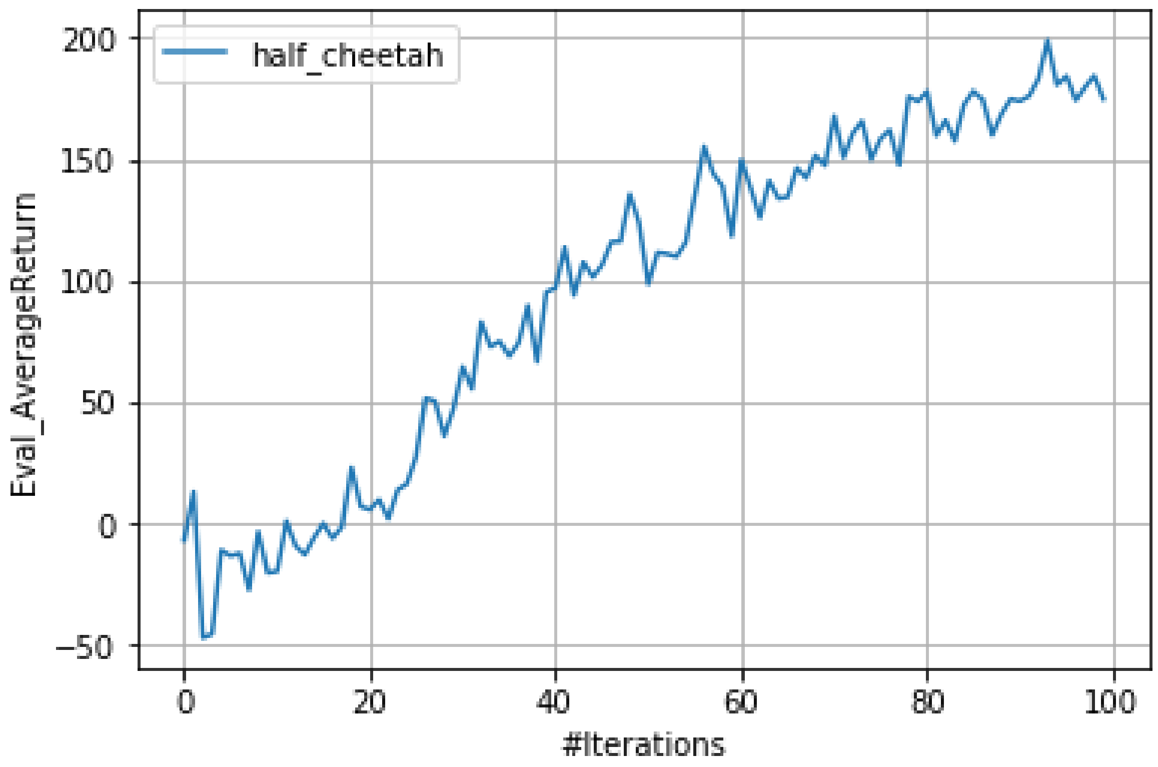


Figure 10: **Q7** AC for `HalfCheetah-v2`

```
batch_size=30000
python run_hw3.py env_name=HalfCheetah-v2 n_iter=100 ep_len=150 \
    computation_graph_args.n_layers=2 computation_graph_args.size=32 \
    train_batch_size=$batch_size batch_size=$batch_size \
    batch_size_initial=$batch_size computation_graph_args.learning_rate=0.02 \
    eval_batch_size=1500 estimate_advantage_args.discount=0.9 \
    estimate_advantage_args.nn_baseline=true \
    estimate_advantage_args.standardize_advantages=true \
    exp_name="q7_10_10" estimate_advantage_args.reward_to_go=true \
    rl_alg=ac computation_graph_args.num_target_updates=10 \
    computation_graph_args.num_grad_steps_per_target_update=10
```

Listing 9: **Q7** Run commands (2/2)

8 Question 8

Description of the implementation: I am using dyna models to imagine full-blown trajectories; first I train the models using randomly sampled data from the replay buffer, then I sample recent data, which gives me full trajectories that I can use to form policy gradient updates, however, I also concatenate these trajectories with the imagined ones via dyna models, which are also reward labled using `self.env.get_reward()`. The results are not great, but this was my understanding of dyna.

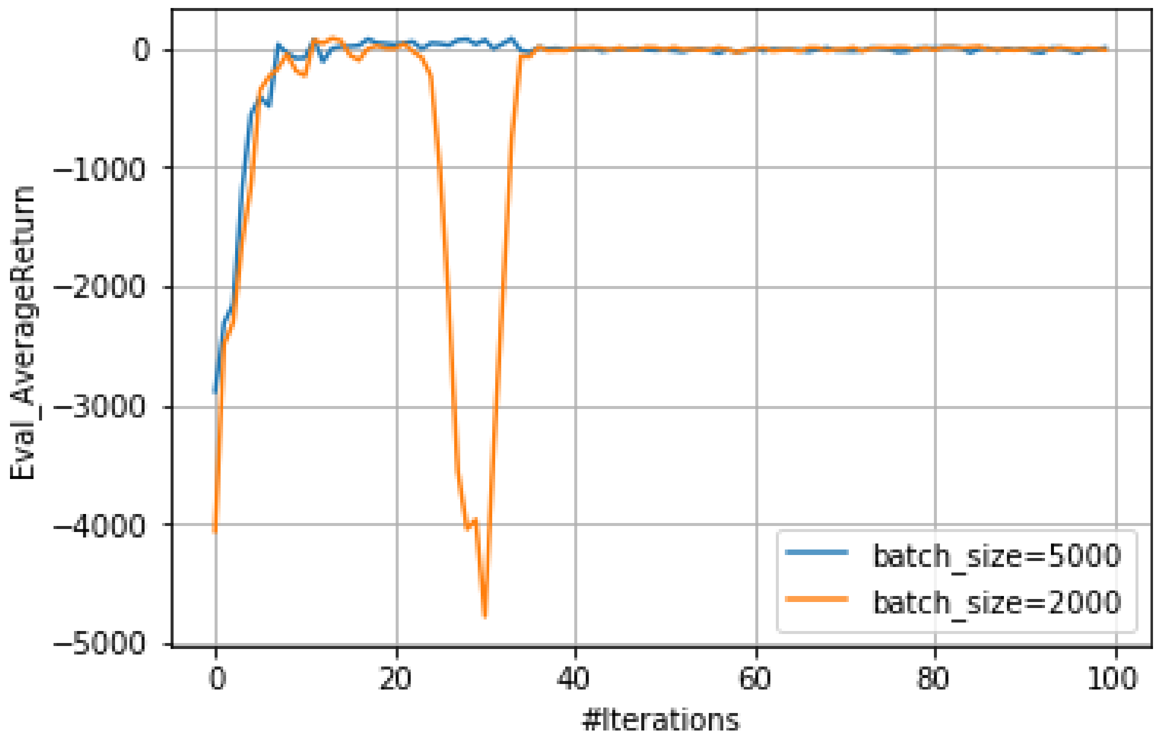


Figure 11: **Q8** Dyna for `HalfCheetah-v2`

```
batch_size=5000
python run_hw3.py env_name=cheetah-ift6163-v0 n_iter=100 \
    computation_graph_args.n_layers=2 computation_graph_args.size=32 \
    estimate_advantage_args.discount=0.95 train_batch_size=$batch_size \
    batch_size=$batch_size batch_size_initial=$batch_size \
    computation_graph_args.learning_rate=0.01 \
    estimate_advantage_args.nn_baseline=true \
    estimate_advantage_args.standardize_advantages=true \
    exp_name="q8_cheetah_n500-arch1x32" \
    estimate_advantage_args.reward_to_go=true rl_alg=dyna

batch_size=2000
python run_hw3.py env_name=cheetah-ift6163-v0 n_iter=100 \
    computation_graph_args.n_layers=2 computation_graph_args.size=32 \
    estimate_advantage_args.discount=0.95 train_batch_size=$batch_size \
    batch_size=$batch_size batch_size_initial=$batch_size \
    computation_graph_args.learning_rate=0.01 \
    estimate_advantage_args.nn_baseline=true \
    estimate_advantage_args.standardize_advantages=true \
    exp_name="q8_cheetah_n500-arch1x32" \
    estimate_advantage_args.reward_to_go=true rl_alg=dyna
```

Listing 10: **Q8** Run commands