Homework 4

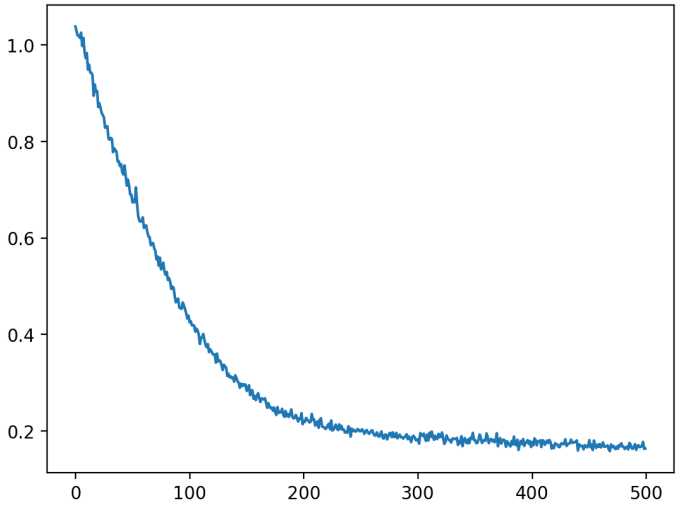
Author:
Reece D. Huff

CS 285: Deep Reinforcement Learning

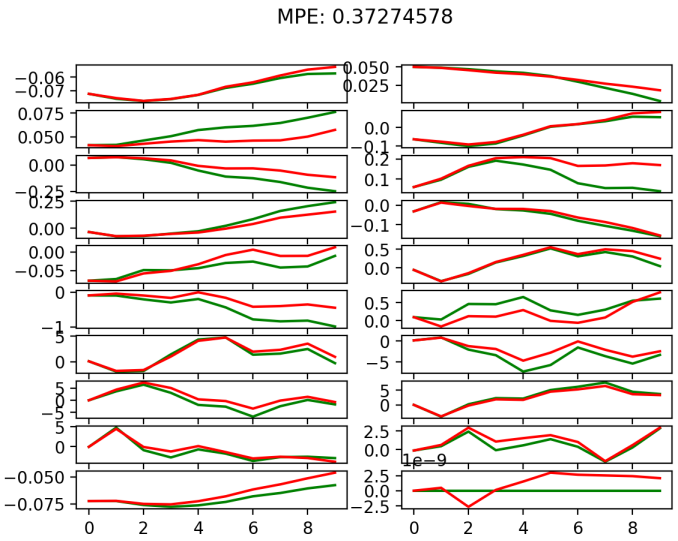
Dr. Sergey Levine

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

Problem 1

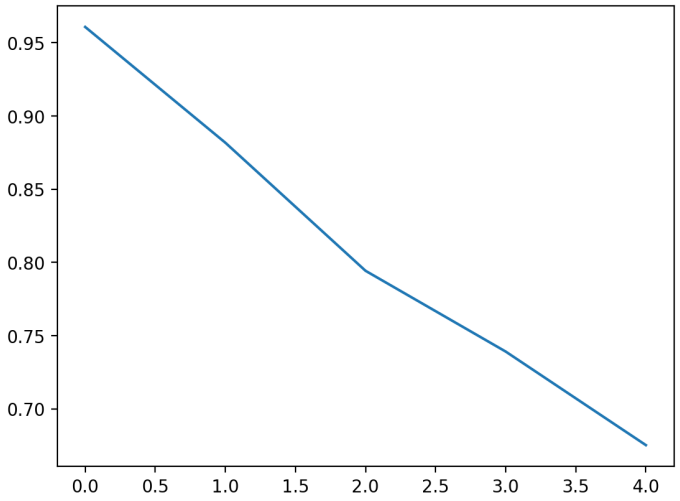


(a) Losses during the training of the neural network. x-axis is iterations; y-axis is MSE loss.

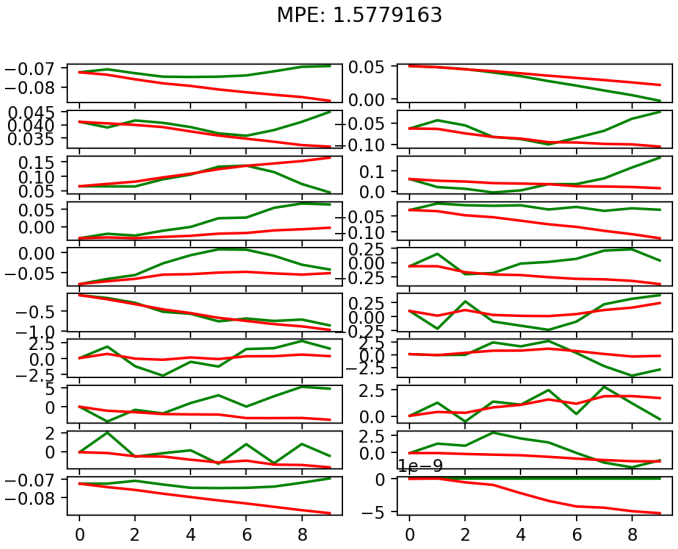


(b) Model predictions and ground truth dynamics. Model prediction error (MPE) in the title.

Figure 1: Model-based learning with a **small** neural network (1x32) trained with **many steps** ($n = 500$).

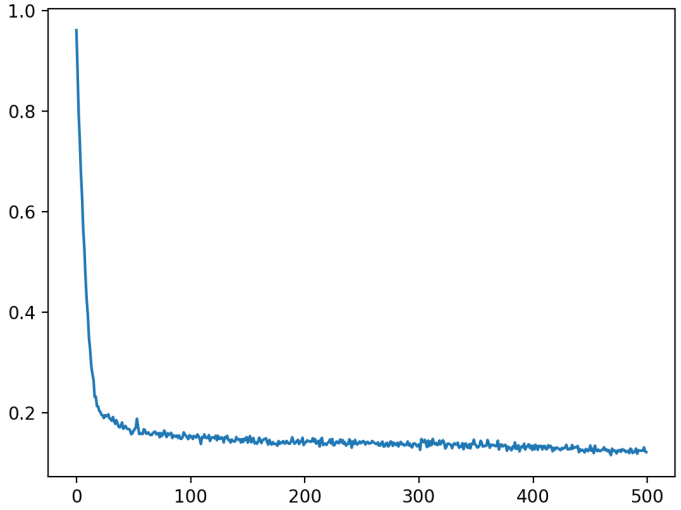


(a) Losses during the training of the neural network. x-axis is iterations; y-axis is MSE loss.

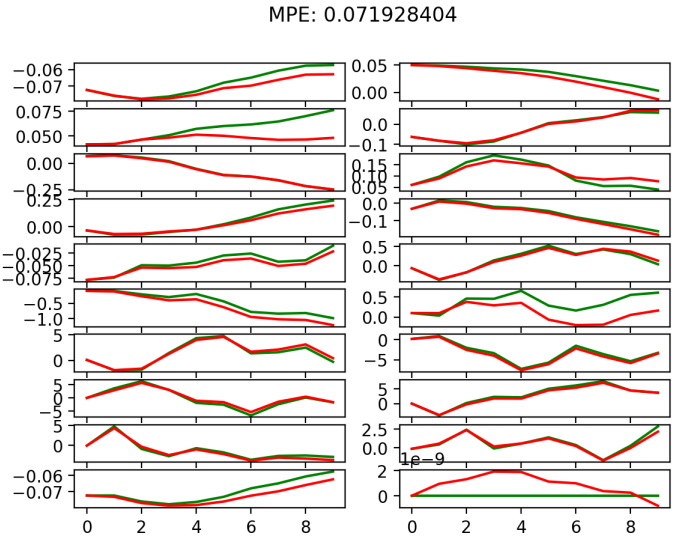


(b) Model predictions and ground truth dynamics. Model prediction error (MPE) in the title.

Figure 2: Model-based learning with a **large** neural network (1x32) trained with **few steps** ($n = 5$).



(a) Losses during the training of the neural network. x-axis is iterations; y-axis is MSE loss.



(b) Model predictions and ground truth dynamics. Model prediction error (MPE) in the title.

Figure 3: Model-based learning with a **large** neural network (1x32) trained with **many steps** ($n = 500$).

Comments: Perhaps unsurprisingly, the best performing framework is the one that trains a large, powerful neural network with many training steps. The frameworks have downsides in that the framework with the smaller, less-predictive neural network is incapable generalizing well regardless of the number of training steps, and the framework with the large neural network is trained but cutoff before convergence is reached.

`problem1.sh`

```
python cs285/scripts/run_hw4_mb.py --exp_name q1_cheetah_n500_arch1x32 --env_name
    cheetah-cs285-v0 --add_sl_noise --n_iter 1 --batch_size_initial 20000 --
    num_agent_train_steps_per_iter 500 --n_layers 1 --size 32 --scalar_log_freq -1 --
    video_log_freq -1 --mpc_action_sampling_strategy 'random'

 python cs285/scripts/run_hw4_mb.py --exp_name q1_cheetah_n5_arch2x250 --env_name
    cheetah-cs285-v0 --add_sl_noise --n_iter 1 --batch_size_initial 20000 --
    num_agent_train_steps_per_iter 5 --n_layers 2 --size 250 --scalar_log_freq -1 --
    video_log_freq -1 --mpc_action_sampling_strategy 'random'

python cs285/scripts/run_hw4_mb.py --exp_name q1_cheetah_n500_arch2x250 --env_name
    cheetah-cs285-v0 --add_sl_noise --n_iter 1 --batch_size_initial 20000 --
    num_agent_train_steps_per_iter 500 --n_layers 2 --size 250 --scalar_log_freq -1 --
    video_log_freq -1 --mpc_action_sampling_strategy 'random'
```

Problem 2

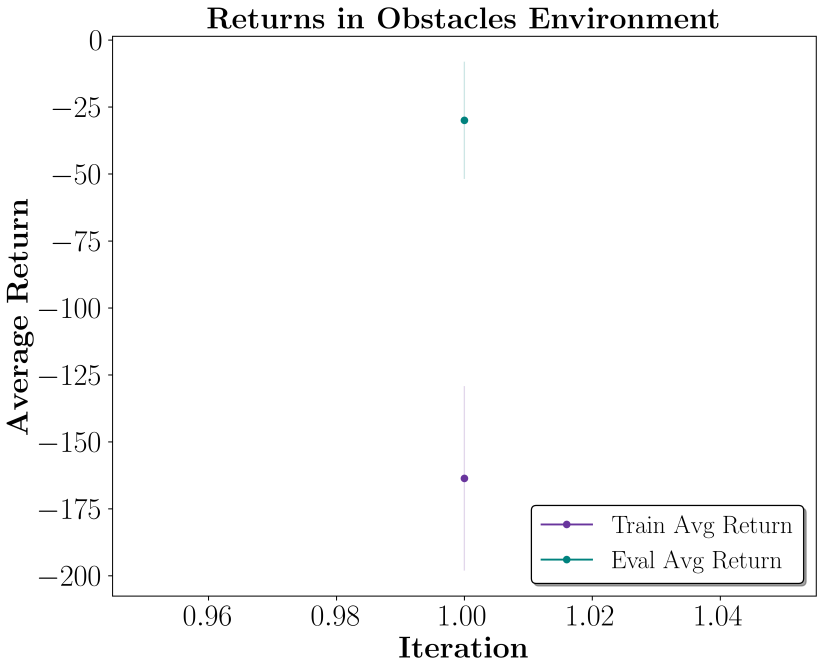


Figure 4: Testing our MBRL framework for just one iteration.

`problem2.sh`

```
python cs285/scripts/run_hw4_mb.py --exp_name q2_obstacles_singleiteration --env_name
    obstacles-cs285-v0 --add_sl_noise --num_agent_train_steps_per_iter 20 --n_iter 1
    --batch_size_initial 5000 --batch_size 1000 --mpc_horizon 10 --video_log_freq -1
    --mpc_action_sampling_strategy 'random'
```

Problem 3

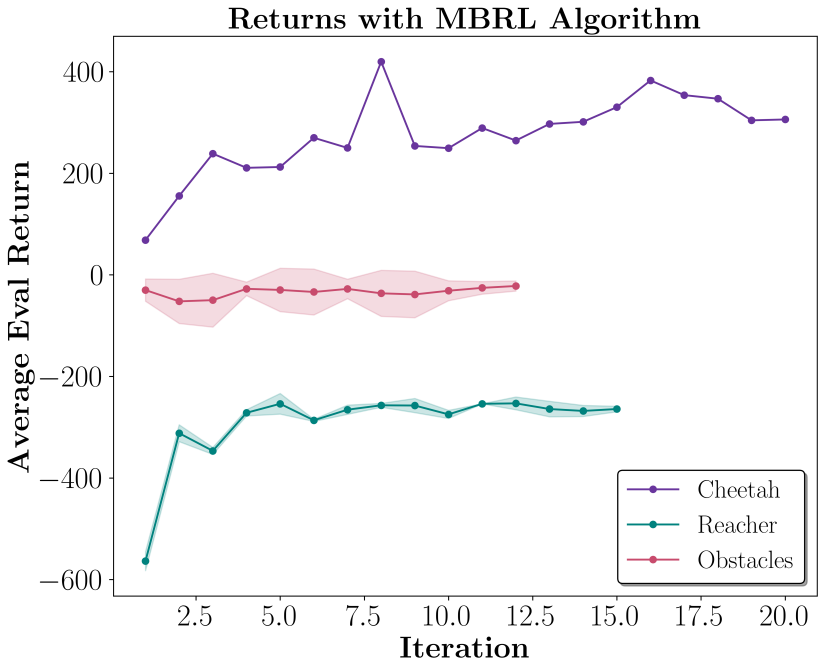


Figure 5: Testing our MBRL framework in the Cheetah, Reacher, and Obstacles environments.

`problem3.sh`

```
python cs285/scripts/run_hw4_mb.py --exp_name q3_obstacles --env_name obstacles-cs285-
    v0 --add_sl_noise --num_agent_train_steps_per_iter 20 --batch_size_initial 5000 --
    batch_size 1000 --mpc_horizon 10 --n_iter 12 --video_log_freq -1 --
    mpc_action_sampling_strategy 'random'

python cs285/scripts/run_hw4_mb.py --exp_name q3_reacher --env_name reacher-cs285-v0
    --add_sl_noise --mpc_horizon 10 --num_agent_train_steps_per_iter 1000 --
    batch_size_initial 5000 --batch_size 5000 --n_iter 15 --video_log_freq -1 --
    mpc_action_sampling_strategy 'random'

python cs285/scripts/run_hw4_mb.py --exp_name q3_cheetah --env_name cheetah-cs285-v0
    --mpc_horizon 15 --add_sl_noise --num_agent_train_steps_per_iter 1500 --
    batch_size_initial 5000 --batch_size 5000 --n_iter 20 --video_log_freq -1 --
    mpc_action_sampling_strategy 'random'
```

Problem 4

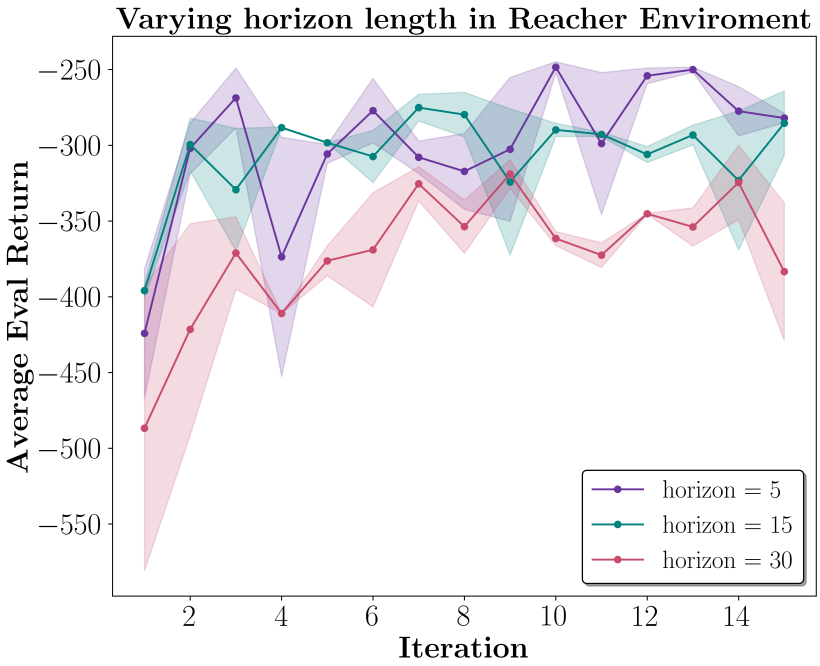


Figure 6: Varying the horizon length in the Reacher environment. It is interesting to note that the shorter horizon length generally decreases performance, suggesting that the additional steps harm the framework. Although there is a lot a randomness in the system, therefore averaging over random seeds may be needed to make a definitive conclusion.

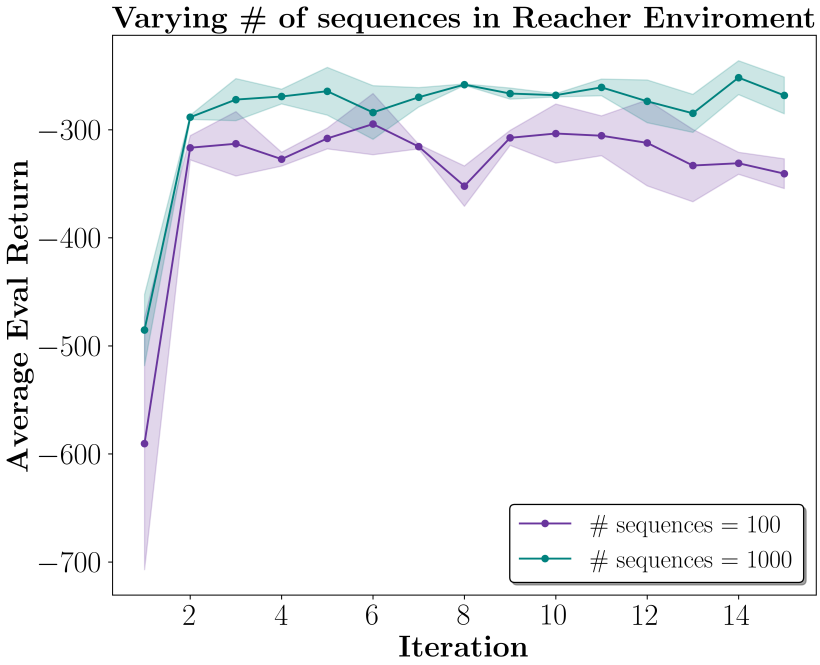


Figure 7: The number of sequences varied in the Reacher environment. It is perhaps no surprise that the more sequences, the higher performance.

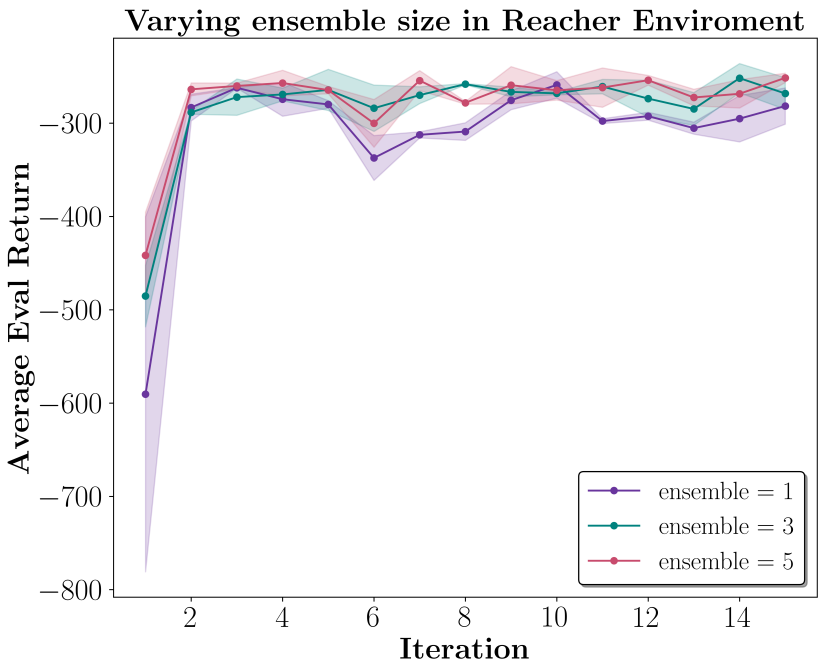


Figure 8: The ensemble size is varied in the Reacher environment. The ensemble size does not have a huge impact on performance. However, generally speaking, the larger the ensemble, the higher the performance.

`problem4.sh`

```
python cs285/scripts/run_hw4_mb.py --exp_name q4_reacher_horizon5 --env_name reacher-
    cs285-v0 --add_sl_noise --mpc_horizon 5 --mpc_action_sampling_strategy 'random' --
    num_agent_train_steps_per_iter 1000 --batch_size 800 --n_iter 15 --video_log_freq
    -1 --mpc_action_sampling_strategy 'random'

python cs285/scripts/run_hw4_mb.py --exp_name q4_reacher_horizon15 --env_name reacher-
    cs285-v0 --add_sl_noise --mpc_horizon 15 --num_agent_train_steps_per_iter 1000 --
    batch_size 800 --n_iter 15 --video_log_freq -1 --mpc_action_sampling_strategy '
    random'

python cs285/scripts/run_hw4_mb.py --exp_name q4_reacher_horizon30 --env_name reacher-
    cs285-v0 --add_sl_noise --mpc_horizon 30 --num_agent_train_steps_per_iter 1000 --
    batch_size 800 --n_iter 15 --video_log_freq -1 --mpc_action_sampling_strategy '
    random'

python cs285/scripts/run_hw4_mb.py --exp_name q4_reacher_numseq100 --env_name reacher-
    cs285-v0 --add_sl_noise --mpc_horizon 10 --num_agent_train_steps_per_iter 1000 --
    batch_size 800 --n_iter 15 --mpc_num_action_sequences 100 --
    mpc_action_sampling_strategy 'random'

python cs285/scripts/run_hw4_mb.py --exp_name q4_reacher_numseq1000 --env_name reacher
    -cs285-v0 --add_sl_noise --mpc_horizon 10 --num_agent_train_steps_per_iter 1000 --
    batch_size 800 --n_iter 15 --video_log_freq -1 --mpc_num_action_sequences 1000 --
    mpc_action_sampling_strategy 'random'

python cs285/scripts/run_hw4_mb.py --exp_name q4_reacher_ensemble1 --env_name reacher-
    cs285-v0 --ensemble_size 1 --add_sl_noise --mpc_horizon 10 --
    num_agent_train_steps_per_iter 1000 --batch_size 800 --n_iter 15 --video_log_freq
    -1 --mpc_action_sampling_strategy 'random'

python cs285/scripts/run_hw4_mb.py --exp_name q4_reacher_ensemble3 --env_name reacher-
```

```
    cs285-v0 --ensemble_size 3 --add_sl_noise --mpc_horizon 10 --
    num_agent_train_steps_per_iter 1000 --batch_size 800 --n_iter 15 --video_log_freq
    -1 --mpc_action_sampling_strategy 'random'

python cs285/scripts/run_hw4_mb.py --exp_name q4_reacher_ensemble5 --env_name reacher-
    cs285-v0 --ensemble_size 5 --add_sl_noise --mpc_horizon 10 --
    num_agent_train_steps_per_iter 1000 --batch_size 800 --n_iter 15 --video_log_freq
    -1 --mpc_action_sampling_strategy 'random'
```

Problem 5

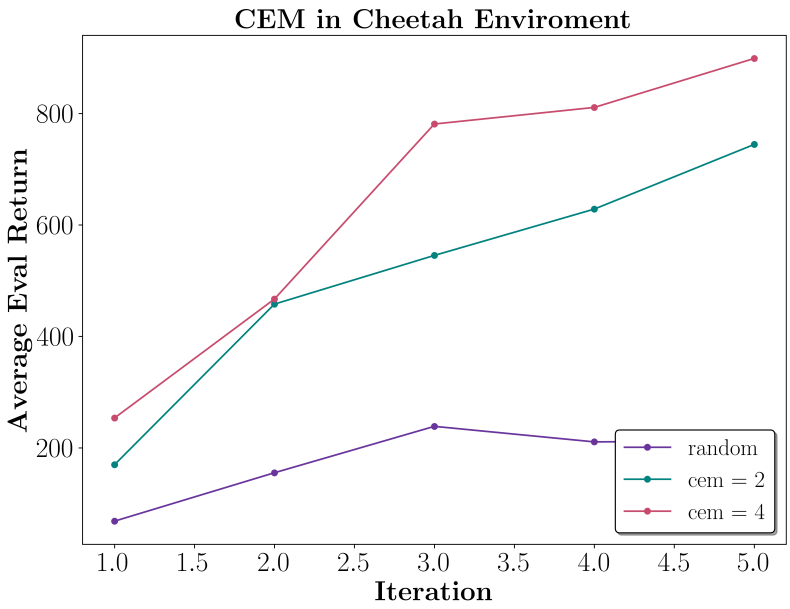


Figure 9: Random shooting vs. CEM in the Cheetah environment. We see that random shooting is the lowest performer. Including CEM greatly improves performance, and having more CEM steps increases performance; however, it comes with a large computational cost.

`problem5.sh`

```
python cs285/scripts/run_hw4_mb.py --exp_name q5_cheetah_random --env_name 'cheetah-
    cs285-v0' --mpc_horizon 15 --add_sl_noise --num_agent_train_steps_per_iter 1500 --
    batch_size_initial 5000 --batch_size 5000 --n_iter 5 --video_log_freq -1 --
    mpc_action_sampling_strategy 'random'

python cs285/scripts/run_hw4_mb.py --exp_name q5_cheetah_cem_2 --env_name 'cheetah-
    cs285-v0' --mpc_horizon 15 --add_sl_noise --num_agent_train_steps_per_iter 1500 --
    batch_size_initial 5000 --batch_size 5000 --n_iter 5 --video_log_freq -1 --
    mpc_action_sampling_strategy 'cem' --cem_iterations 2

python cs285/scripts/run_hw4_mb.py --exp_name q5_cheetah_cem_4 --env_name 'cheetah-
    cs285-v0' --mpc_horizon 15 --add_sl_noise --num_agent_train_steps_per_iter 1500 --
    batch_size_initial 5000 --batch_size 5000 --n_iter 5 --video_log_freq -1 --
    mpc_action_sampling_strategy 'cem' --cem_iterations 4
```

Problem 6

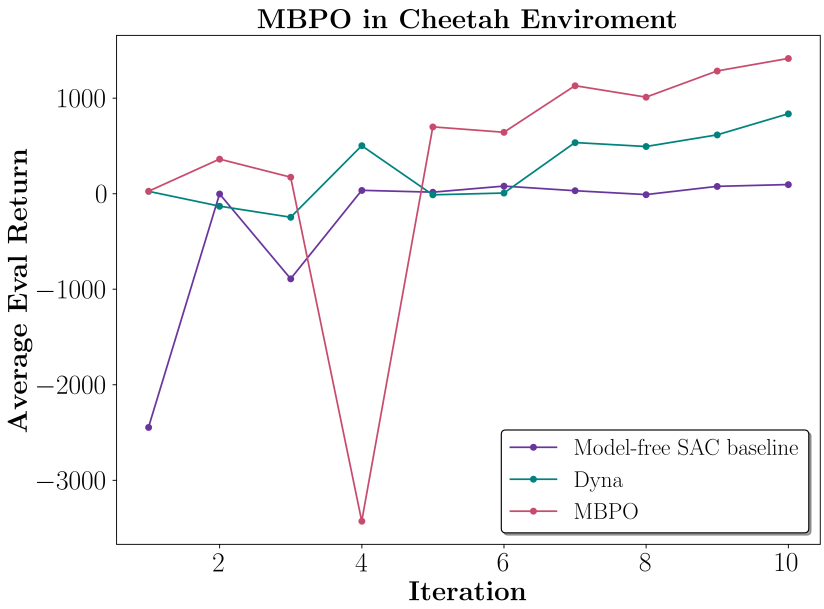


Figure 10: Model-free SAC, Dyna, and MBPO in the Cheetah environment. It is awesome to see how much MBPO improves performance, and more broadly how including rollouts maximizes returns.

problem6.sh

```
python cs285/scripts/run_hw4_mbpo.py --exp_name q6_cheetah_rlen0 --env_name 'cheetah-
    cs285-v0' --add_sl_noise --num_agent_train_steps_per_iter 1500 --
    batch_size_initial 5000 --batch_size 5000 --n_iter 10 --video_log_freq -1 --
    sac_discount 0.99 --sac_n_layers 2 --sac_size 256 --sac_batch_size 1500 --
    sac_learning_rate 0.0003 --sac_init_temperature 0.1 --sac_n_iter 1000 --
    mbpo_rollout_length 0

python cs285/scripts/run_hw4_mbpo.py --exp_name q6_cheetah_rlen1 --env_name 'cheetah-
    cs285-v0' --add_sl_noise --num_agent_train_steps_per_iter 1500 --
    batch_size_initial 5000 --batch_size 5000 --n_iter 10 --video_log_freq -1 --
    sac_discount 0.99 --sac_n_layers 2 --sac_size 256 --sac_batch_size 1500 --
    sac_learning_rate 0.0003 --sac_init_temperature 0.1 --sac_n_iter 5000 --
    mbpo_rollout_length 1

python cs285/scripts/run_hw4_mbpo.py --exp_name q6_cheetah_rlen10 --env_name 'cheetah-
    cs285-v0' --add_sl_noise --num_agent_train_steps_per_iter 1500 --
    batch_size_initial 5000 --batch_size 5000 --n_iter 10 --video_log_freq -1 --
    sac_discount 0.99 --sac_n_layers 2 --sac_size 256 --sac_batch_size 1500 --
    sac_learning_rate 0.0003 --sac_init_temperature 0.1 --sac_n_iter 5000 --
    mbpo_rollout_length 10
```