Assignment 4: Model-Based RL

1 MBRL Problem 1

1.1 What you will implement

Collect a large dataset by executing random actions. Train a neural network dynamics model on this fixed dataset. The implementation that you will do here will be for training the dynamics model.

1.2 What code files to fill in

The starter code for this assignment can be found at

- `cs285/agents/model_based_agent.py`: up to (and including) `update_statistics`.
- `cs285/scripts/run_hw4.py`: everything except for `collect_mbpo_rollout` at the top of the file.

1.3 What command to run

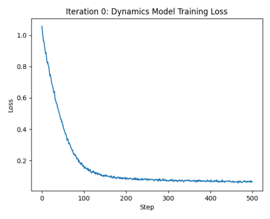
```
python cs285/scripts/run_hw4.py -cfg experiments/mpc/halfcheetah_0_iter.yaml
```

This config will only run the first iteration without actually evaluating the policy, meaning it will only train the ensemble of dynamics models. The code will produce plots inside your logdir that illustrate the full learning curve of the dynamics models. For the first command, the loss should go below 0.2 by iteration 500.

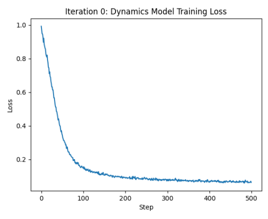
Modify `experiments/mpc/halfcheetah_0_iter.yaml` to change some hyperparameters. Try at least two other configurations of hyperparameters that affect dynamics model training (e.g., number of layers, hidden size, or learning rate).

1.4 What to submit:

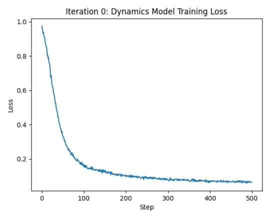
For this question, submit the learning curve for 3 runs total: the initial run with provided hyperparameters as well as 2 of your own. Note that for these learning curves, we intend for you to just copy the png images produced by the code.



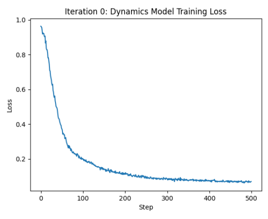
(a) 1 Layer



(b) 2 Layers

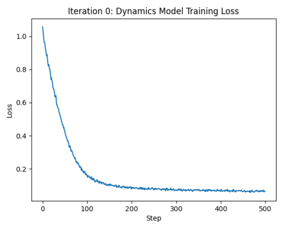


(c) 3 Layers

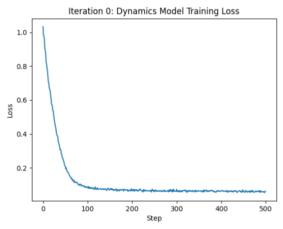


(d) 4 Layers

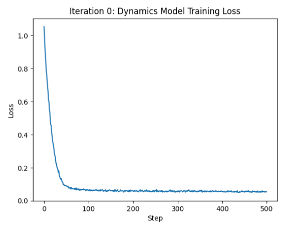
Figure 1: Effect of Number of Layers (hidden size = 32)



(a) Hidden = 32



(b) Hidden = 64



(c) Hidden = 128

Figure 2: Effect of Hidden Size (1 layer)

2 MBRL Problem 2

2.1 What you will implement

Action selection using your learned dynamics model and a given reward function.

2.2 What code files to fill in

- `cs285/agents/model_based_agent.py`: the rest of the file, except for the CEM strategy in `get_action`.

2.3 What commands to run

```
python cs285/scripts/run_hw4.py -cfg experiments/mpc/obstacles_1_iter.yaml
```

Recall the overall flow of our training loop. We first collect data with our policy (which starts as random), we train our model on that collected data, we evaluating the resulting MPC policy (which now uses the trained model), and repeat. To verify that your MPC is indeed doing reasonable action selection, run one iteration of this process using the command above. This will evaluate your MPC policy, but not use it to collect data for future iterations. Look at `eval_return`, which should be greater than -70 after one iteration.

2.4 What to submit:

Submit this run as part of your run logs, and report your `eval_return`.

```
########################
logging outputs to /home/siqi/Desktop/homework_fall2023-main/hw4/cs285/scripts/../../data/
    obstacles-cs285-v0_obstacles_single_l2_h250_mpcrandom_horizon10_actionseq1000_09
    -01-2026_03-37-12
########################
Using GPU id 0


********** Iteration 0 ************
Collecting data...
Training agent...
100%|| 20/20 [00:00<00:00, 47.44it/s]
Evaluating 20 rollouts...
Average eval return: -32.893217120463
```

3 MBRL Problem 3

3.1 What you will implement

MBRL algorithm with on-policy data collection and iterative model training.

3.2 What code files to fill in

None. You should already have done everything that you need, because `run_hw4.py` already aggregates your collected data into a replay buffer. Thus, iterative training means to just train on our growing replay buffer while collecting new data at each iteration using the most newly trained model.

3.3 What commands to run

```
python cs285/scripts/run_hw4.py -cfg experiments/mpc/obstacles_multi_iter.yaml
```

```
python cs285/scripts/run_hw4.py -cfg experiments/mpc/reacher_multi_iter.yaml
```

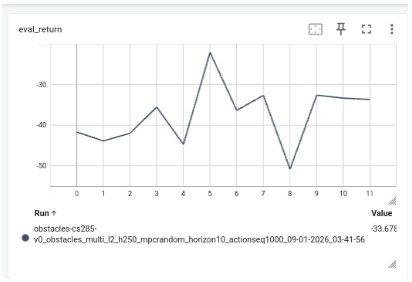
```
python cs285/scripts/run_hw4.py -cfg experiments/mpc/halfcheetah_multi_iter.yaml
```

You should expect rewards of around -25 to -20 for the obstacles env, rewards of around -300 to -250 for the reacher env, and rewards of around 250-350 for the cheetah env.

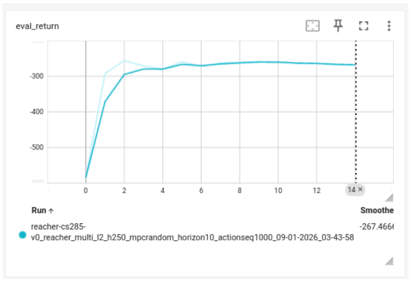
3.4 What to submit:

Submit these runs as part of your run logs, and include the return plots in your pdf.

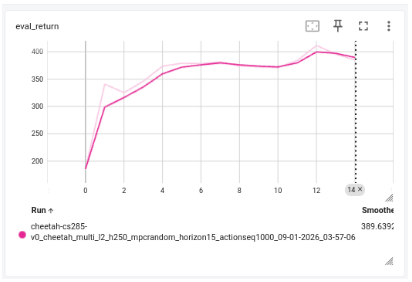
3.5 Results



(a) Obstacles



(b) Reacher



(c) HalfCheetah

Figure 3: Multi-iteration MBRL performance across three environments. Returns improve over iterations as the model is retrained with on-policy data.

4 MBRL Problem 4

4.1 What you will implement

You will compare the performance of your MBRL algorithm as a function of three hyperparameters: the number of models in your ensemble, the number of random action sequences considered during each action selection, and the MPC planning horizon.

4.2 What code files to fill in

None.

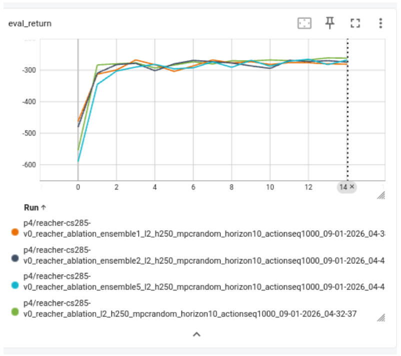
4.3 What commands to run

```
python cs285/scripts/run_hw4.py -cfg experiments/mpc/reacher_ablation.yaml
```

Modify (or make copies of) the YAML file to ablate each of the hyperparameters. For each hyperparameter, do at least 1 run with it increased and 1 with it decreased from the default (so 7 runs total). Make sure to keep the other hyperparameters the same when studying the effect of one of them.

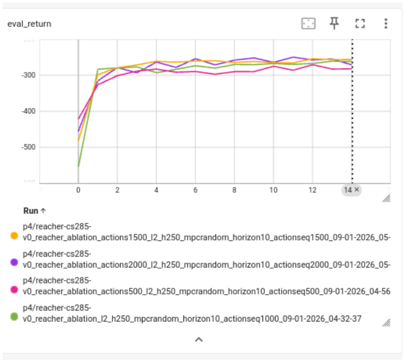
4.4 What to submit:

1. Submit these runs as part of your run logs.
2. Include the following plots (as well as captions that describe your observed trends) of the following:
   - effect of ensemble size
   - effect of the number of candidate action sequences
   - effect of planning horizon



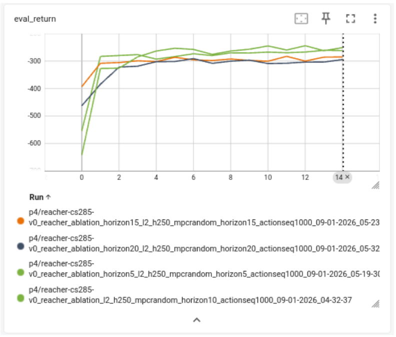
● p4/reacher-cs285-v0_reacher_ablation_ensemble1_l2_h250_mpcrandom_horizon10_actionseq1000_09-01-2026_04-3
● p4/reacher-cs285-v0_reacher_ablation_ensemble2_l2_h250_mpcrandom_horizon10_actionseq1000_09-01-2026_04-4
● p4/reacher-cs285-v0_reacher_ablation_ensemble5_l2_h250_mpcrandom_horizon10_actionseq1000_09-01-2026_04-4
● p4/reacher-cs285-v0_reacher_ablation_l2_h250_mpcrandom_horizon10_actionseq1000_09-01-2026_04-32-37

(a) Effect of ensemble size



● p4/reacher-cs285-v0_reacher_ablation_actions1500_l2_h250_mpcrandom_horizon10_actionseq1500_09-01-2026_05-
● p4/reacher-cs285-v0_reacher_ablation_actions2000_l2_h250_mpcrandom_horizon10_actionseq2000_09-01-2026_05-
● p4/reacher-cs285-v0_reacher_ablation_actions500_l2_h250_mpcrandom_horizon10_actionseq500_09-01-2026_04-56
● p4/reacher-cs285-v0_reacher_ablation_l2_h250_mpcrandom_horizon10_actionseq1000_09-01-2026_04-32-37

(b) Effect of the number of candidate action sequences



● p4/reacher-cs285-v0_reacher_ablation_horizon15_l2_h250_mpcrandom_horizon15_actionseq1000_09-01-2026_05-23
● p4/reacher-cs285-v0_reacher_ablation_horizon20_l2_h250_mpcrandom_horizon20_actionseq1000_09-01-2026_05-32
● p4/reacher-cs285-v0_reacher_ablation_horizon5_l2_h250_mpcrandom_horizon5_actionseq1000_09-01-2026_05-19-3(
● p4/reacher-cs285-v0_reacher_ablation_l2_h250_mpcrandom_horizon10_actionseq1000_09-01-2026_04-32-37

(c) Effect of planning horizon

Figure 4: Ablation Study: i) Action sequence length affects optimization difficulty. Long or short action sequences seem both lead to higher initial return. ii) Ensemble size affects model uncertainty & conservatism. Large ensemble size leads to lower initial return, but the final results would not be superior. iii) Planning horizon affects model bias accumulation. Low horizon seems lead to lower initial evaluation return but higher final return and faster convergence.

- Short sequences reduce optimization difficulty and model-error accumulation, leading to stable early performance. Long sequences increase exploration and occasionally discover high-reward trajectories, but suffer from higher variance and reduced reliability later in training.
- Larger ensembles reduce initial performance due to increased conservatism from model disagreement. Early uncertainty biases planning toward safer actions. As models improve, this effect weakens, but larger ensembles do not improve final performance, indicating diminishing returns once uncertainty is sufficiently mitigated.
- Short horizons produce lower initial returns due to limited foresight, but enable faster convergence and higher final performance. Reduced rollout error and more frequent replanning improve optimization stability, highlighting a trade-off between early performance and long-term learning efficiency.

5 MBRL Problem 5

5.1 What you will implement

You will compare the performance of your MBRL algorithm with action selecting performed by random-shooting (what you have done up to this point) and CEM.

Because CEM can be much slower than random-shooting, we will only run MBRL for 5 iterations for this problem. We will try two hyperparameter settings for CEM and compare their performance to random-shooting.

5.2 What code files to fill in

- `cs285/agents/model_based_agent.py`: the CEM action selection strategy.

5.3 What commands to run

```
python cs285/scripts/run_hw4.py -cfg experiments/mpc/halfcheetah_cem.yaml
```

You should expect rewards around 800 or higher when using CEM on the cheetah env. Try a `cem_iterations` value of both 2 and 4, and compare results.

5.4 What to submit:

1. Submit these runs as part of your run logs.
2. Include a plot comparing random shooting (from Problem 3) with CEM, as well as captions that describe how CEM affects results for different numbers of sampling iterations (2 vs. 4).

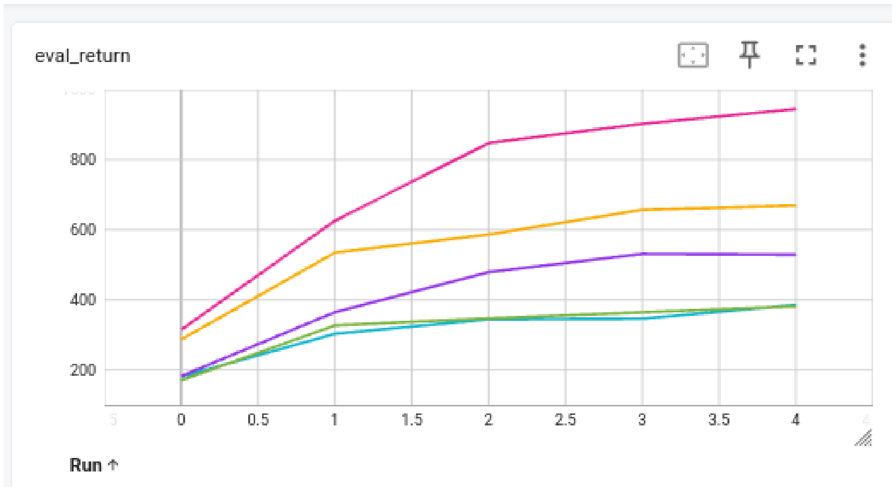


Figure 5: Comparison of CEM vs Random Shooting on HalfCheetah: The plot shows evaluation returns across 5 MBRL iterations. CEM with 4 iterations (pink) achieves the best performance, reaching approximately 900 reward. CEM with 3 iterations (orange) reaches around 650-700, while CEM with 2 iterations (purple) performs similar or better than random shooting baseline (green) at around 100-400. The results demonstrate that CEM significantly outperforms random shooting when using sufficient iterations (4), with performance scaling with the number of CEM iterations used for action selection.

6 MBRL Problem 6

6.1 What you will implement

In this homework you will also be implementing a variant of MBPO. Another way of leveraging the learned model is through generating additional samples to train the policy and value functions. Since RL often requires many environment interaction samples, which can be costly, we can use our learned model to generate additional samples to improve sample complexity. In MBPO, we build on your SAC implementation from HW3 and use the learned model you implemented in the earlier questions for generating additional samples to train our SAC agent. We will try three settings:

1. Model-free SAC baseline: no additional rollouts from the learned model.
2. Dyna (technically "dyna-style" - the original Dyna algorithm is a little different): add single-step rollouts from the model to the replay buffer and incorporate additional gradient steps per real world step.
3. MBPO: add in 10-step rollouts from the model to the replay buffer and incorporate additional gradient steps per real world step.

6.2 What code files to fill in

- `cs285/scripts/run_hw4.py`: the `collect_mbpo_rollout` function at the top of the file.

6.3 What commands to run

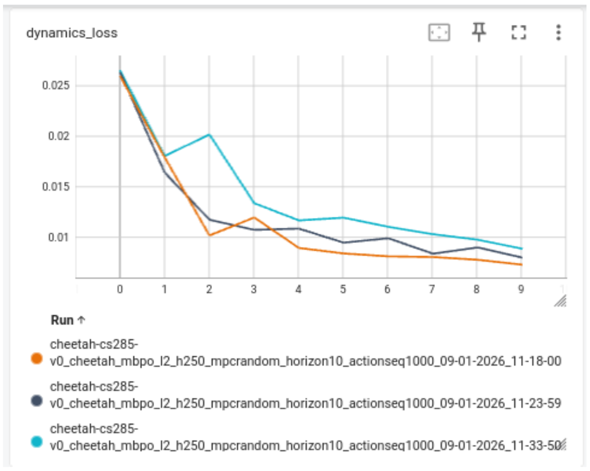
```
python cs285/scripts/run_hw4.py -cfg experiments/mpc/halfcheetah_mbpo.yaml --
    sac_config_file experiments/sac/halfcheetah_clipq.yaml
```

Edit `experiments/sac/halfcheetah_clipq.yaml` to change the MBPO rollout length. The model-free SAC baseline corresponds to a rollout length of 0, The Dyna-like algorithm corresponds to a rollout length of 1, and full MBPO corresponds to a rollout length of 10.

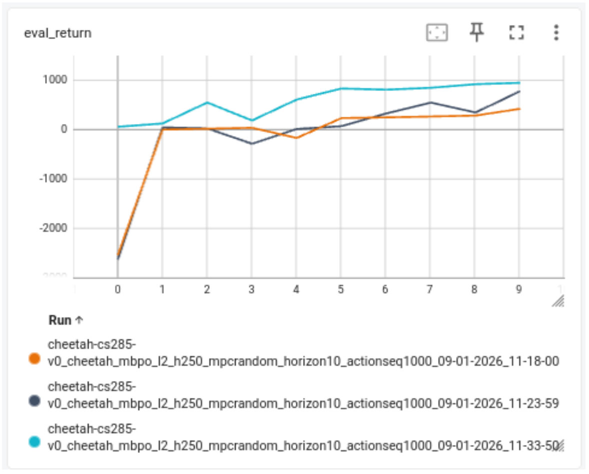
You should be able to reach returns around 700 or higher with full MBPO with a rollout length of 10.

6.4 What to submit:

1. Submit these 3 runs as part of your run logs.
2. Include a plot to show a comparison between the 3 runs, and explain any trends you see.



(a) The plot shows dynamics loss over training iterations.



(b) The plot shows evaluation returns over training iterations.

Figure 6: MBPO Training Progress: Comparison of model-free SAC baseline, Dyna-style (1-step), and full MBPO (10-step) on HalfCheetah environment. Large rollout length helps.