University of California Berkeley

Computer Science Department

CS 294-112

HW4: Model-Based RL

Submitted by
Cristobal Pais

CS 294-112
Oct 24th
Fall 2018

HW4: Model-Based RL

Cristobal Pais

Oct 24th 2018

1 Problem 1: Dynamics prediction

The command used to generate the results of this section is:

```
python main.py q1
```

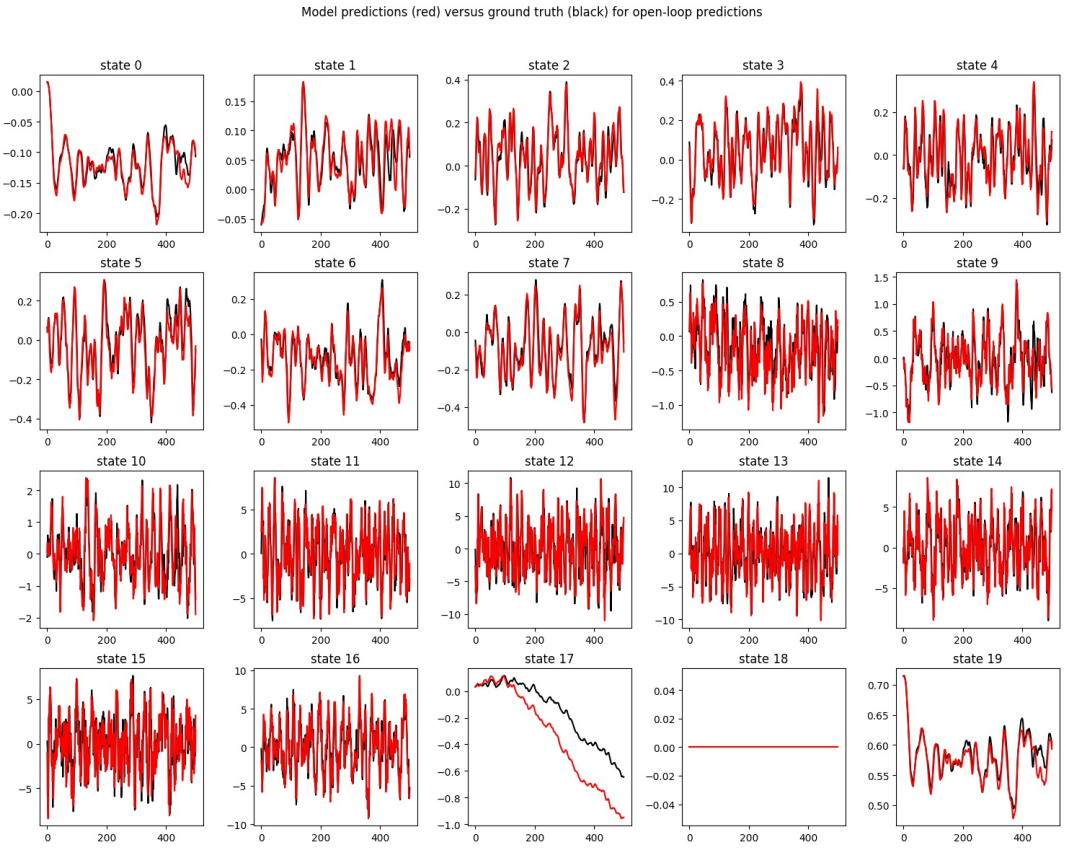

Figure 1: Model prediction versus observable states for open-loop predictions. From the results, it can be seen how accurate the model prediction is among all states, obtaining the worst result for state 17. The explanation behind this pattern could lie in the fact that the model is not able to clearly identify the evolution of all the states from the randomly sampled data (gathered with a random policy, off-policy approach), introducing some error when estimating the next state based on the current information of the system, as well as our f_θ function not being able to capture the real dynamics of the model (NN could be too simple). From the results (among several predictions), state 17 predictions follow a similar pattern: similar shape to the ground truth evolution but an error factor is always deviating the predictions. An additional reason could be the complexity of the state being approximated: different states represent different aspects of the environment and thus, their complexity and approximation/estimation error could be different depending on how flexible and expressive our model is.

2 Problem 2: Model-Based controller

The command used to generate the results of this section is:

```
python main.py q2
```

Results are summarized in Table 1 where the average and standard deviation of the returns for the random policy and our model-based controller trained on the randomly gathered data are shown. From them, we can see that both implementations achieve values close to the expectations (-160 for the random policy, and around 0 for the model-based controller).

Metric	Random Policy	Model-Based Controller
Return AVG	-156.788	7.077
Return STD	31.833	30.425

Table 1: Returns AVG and STD from the random policy and the trained model-based controller using randomly gathered data (random dataset D).

3 Problem 3a: Model-Based RL

The commands used to generate the results and figures of this section are (using an extra script named P3Plotter for simplicity):

```
python main.py q3
python P3Plotter.py
```

3.1 Returns vs Iteration

In Figure 2, the average return versus iteration number when running our model-based reinforcement learning with on-policy data collection is shown. From the results, the model is able to reach the expected average return level of 300 by the 10th iteration.

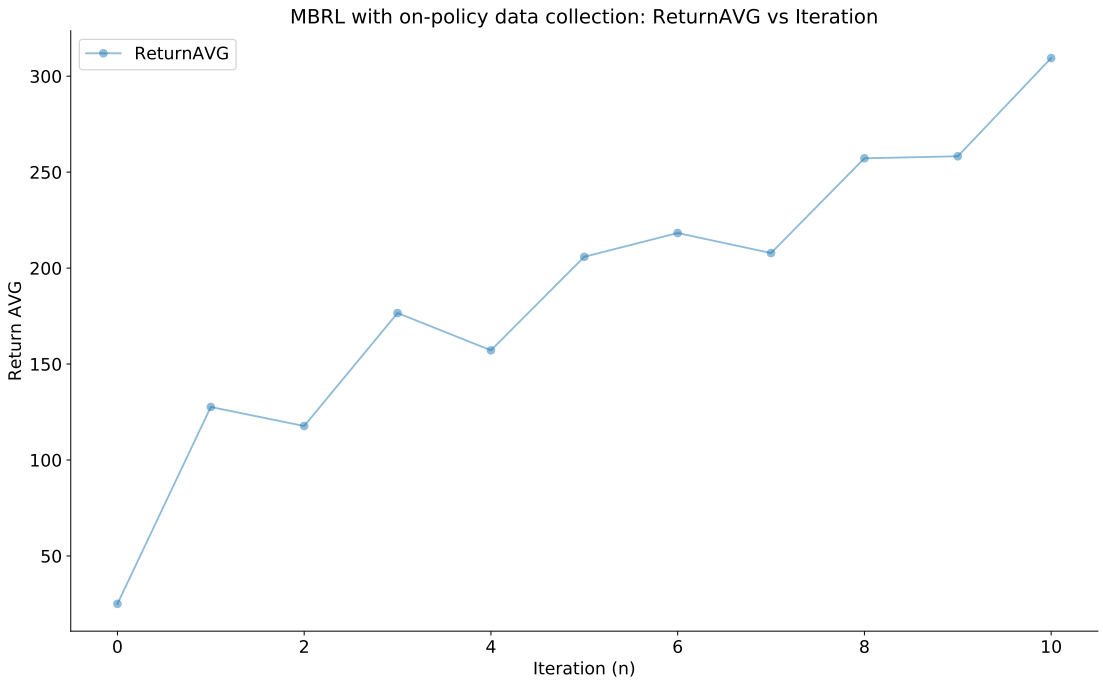

Figure 2: Returns versus iterations using the model-based RL approach. The expected average return of 300 is achieved within the 10 iterations, showing an increasing pattern.

4 Problem 3b: Hyperparameters

4.1 MPC horizon

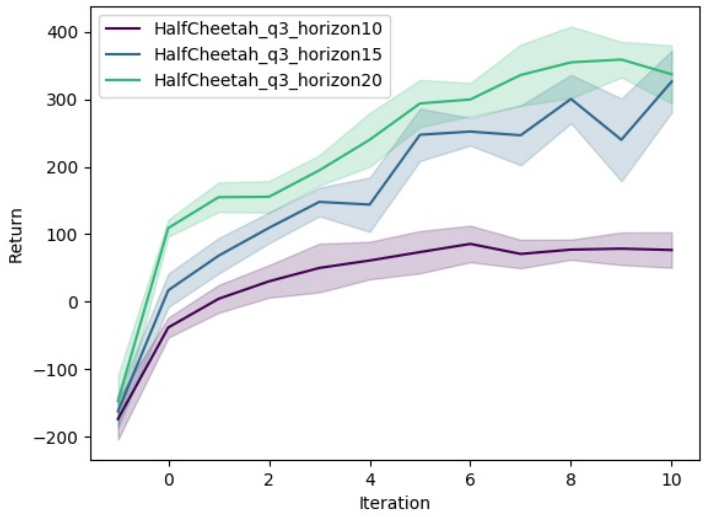

Figure 3: Multiple MPC horizons comparison. As expected, the larger the horizon, the better the performance since we are generating larger trajectories of random actions and evaluating them (eq. 6) allowing us to select actions that lead to better returns in the long-run term.

4.2 Number of randomly sampled action sequences

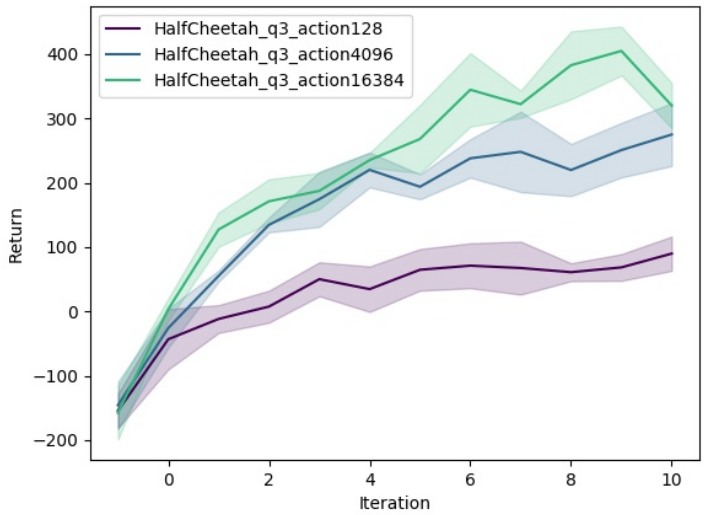

Figure 4: Different number of sampled action sequences (at random) performance comparison. Following the previous discussion, increasing the number of random action sequences generated gives us more possibilities for exploration, obtaining better average performance than the 128 and 4096 cases. However, running times are also increased, being necessary to analyze the inherent trade-off between solution quality and computational time required.

4.3 Number of neural layers

Note: Results include the bug detected by a classmate and published in Piazza, where the number of layers (nn_layers) never gets passed into the Model-Based Policy.

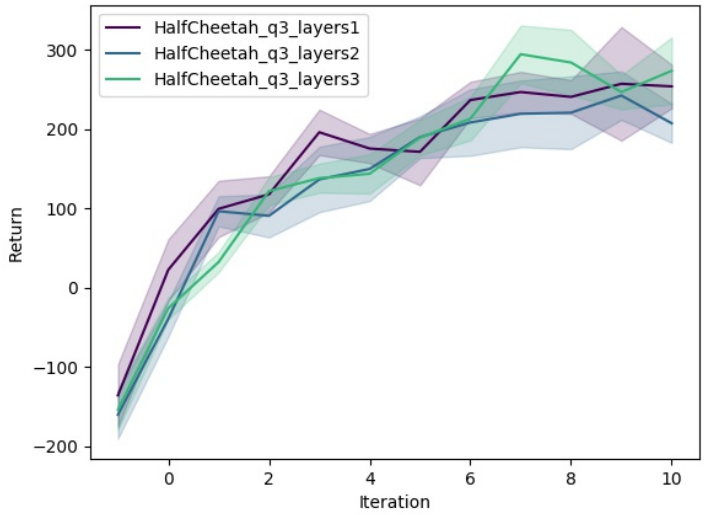

Figure 5: Different number of hidden layers inside the neural network (learned dynamic model) performance comparison. In this case, the impact on the model's performance is not as explicit as in both previous cases, mainly because adding extra layers to the dynamic model function approximation (f_θ) do not significantly improves it, and thus, performance is mainly dependent on the number of random actions and/or the horizon length. If these values are kept constant, there is no improvement in the performance of our policy.

5 Appendix

The commands used for generating the outputs for Question 3b are the following:

```
python main.py q3 --exp_name horizon10
                  --mpc_horizon 10
python main.py q3 --exp_name horizon15
                  --mpc_horizon 15
python main.py q3 --exp_name horizon20
                  --mpc_horizon 20
python plot.py --exps HalfCheetah_q3_horizon10
                      HalfCheetah_q3_horizon15
                      HalfCheetah_q3_horizon20
               --save HalfCheetah_q3_mpc_horizon
```

```
python main.py q3 --exp_name action128
                  --num_random_action_selection 128
python main.py q3 --exp_name action4096
                  --num_random_action_selection 4096
python main.py q3 --exp_name action16384
                  --num_random_action_selection 16384
python plot.py --exps HalfCheetah_q3_action128
                      HalfCheetah_q3_action4096
                      HalfCheetah_q3_action16384
               --save HalfCheetah_q3_actions
```

```
python main.py q3 --exp_name layers1
                  --nn_layers 1
python main.py q3 --exp_name layers2
                  --nn_layers 2
python main.py q3 --exp_name layers3
                  --nn_layers 3
python plot.py --exps HalfCheetah_q3_layers1
                      HalfCheetah_q3_layers2
                      HalfCheetah_q3_layers3
               --save HalfCheetah_q3_nn_layers
```

References

[1] Nagabandi, A., Kahn, G., Fearing, R. S., & Levine, S. (2017). Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. arXiv preprint arXiv:1708.02596.