CS294-112 Deep Reinforcement Learning HW4: Model-Based RL

Due October 24th, 11:59 pm

1 Introduction

The goal of this assignment is to get experience with model-based reinforcement learning. Model-based reinforcement learning consists of two main parts: learning a dynamics model, and using a controller to plan and execute actions that minimize a cost function. You will be implementing both the learned dynamics model and the controller in this assignment. This assignment is based on this paper.

2 Model-Based Reinforcement Learning

We will now provide a brief overview of model-based reinforcement learning (MBRL), and the specific type of MBRL you will be implementing in this homework. Please see Lecture 11: Model-Based Reinforcement Learning (with specific emphasis on the slides near page 12) for additional details.

MBRL consists primarily of two aspects: (1) learning a dynamics model and (2) using the learned dynamics models to plan and execute actions that minimize a desired cost function.

2.1 Dynamics Model

In this assignment, you will learn a neural network dynamics model of the form:

$$\hat{\Delta}_{t+1} = f_\theta(\mathbf{s}_t, \mathbf{a}_t) \tag{1}$$

such that

$$\hat{\mathbf{s}}_{t+1} = \mathbf{s}_t + \hat{\Delta}_{t+1} \tag{2}$$

in which the neural network f_θ encodes the difference between the next state and the current state (given the current action that was executed).

You will train f_θ by performing gradient descent on the following objective:

$$\mathcal{L}(\theta) = \sum_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \in \mathcal{D}} \|(\mathbf{s}_{t+1} - \mathbf{s}_t) - f_\theta(\mathbf{s}_t, \mathbf{a}_t)\|_2^2 \tag{3}$$

$$= \sum_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \in \mathcal{D}} \|\Delta_{t+1} - \hat{\Delta}_{t+1}\|_2^2 \tag{4}$$

2.2 Action Selection

Given the learned dynamics model, we now want to select and execute actions that minimize a specified cost function. Ideally, you would calculate these actions by solving the following optimization:

$$\mathbf{a}^* = \arg\min_{\mathbf{a}_{t:\infty}} \sum_{t'=t}^{\infty} c(\hat{\mathbf{s}}_{t'}, \mathbf{a}_{t'}) \;\text{ s.t. }\; \hat{\mathbf{s}}_{t'+1} = \hat{\mathbf{s}}_{t'} + f_\theta(\hat{\mathbf{s}}_{t'}, \mathbf{a}_{t'}). \tag{5}$$

However, solving Eqn. 5 is impractical for two reasons: (1) planning over an infinite sequence of actions is generally difficult and (2) the learned dynamics model is inaccurate, so planning far into the future will also be inaccurate.

Instead, we will solve the following optimization:

$$\mathbf{A}^* = \arg\min_{\{\mathbf{A}^{(0)}, \ldots, \mathbf{A}^{(K-1)}\}} \sum_{t'=t}^{t+H-1} c(\hat{\mathbf{s}}_{t'}, \mathbf{a}_{t'}) \;\text{ s.t. }\; \hat{\mathbf{s}}_{t'+1} = \hat{\mathbf{s}}_{t'} + f_\theta(\hat{\mathbf{s}}_{t'}, \mathbf{a}_{t'}), \tag{6}$$

in which $\mathbf{A}^{(k)}$ are each a random action sequence of length H. What Eqn. 6 says is to consider K random action sequences of length H, predict the future states for each action sequence using the learned dynamics model f_θ, evaluate the total cost of each candidate action sequence, and select the action sequence with the lowest cost.

However, Eqn. 6 only plans H steps into the future. This is problematic because we would like our agent to execute long-horizon tasks. We therefore adopt a model predictive control (MPC) approach, in which we solve Eqn. 6, execute the first action, proceed to the next state, resolve Eqn. 6 at the next state, and repeat this process.

2.3 On-Policy Data Collection

Although MBRL is in theory off-policy—meaning it can learn from any data—in practice it will perform badly if you don't have on-policy data. We can

therefore use on-policy data collection to improve the policy's performance. This is summarized in the algorithm below:

Algorithm 1 Model-Based Reinforcement Learning with On-Policy Data

Run base policy $\pi_0(\mathbf{a}_t, \mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})\}$
while *not done* **do**
  Train f_θ using $\mathcal{D}$ (Eqn. 4)
  $\mathbf{s}_t \leftarrow$ current agent state
  for $t = 0$ *to* T **do**
    $\mathbf{A}^* \leftarrow$ by solving Eqn. 6 at state $\mathbf{s}_t$
    $\mathbf{a}_t \leftarrow$ first action in $\mathbf{A}^*$
    Execute $\mathbf{a}_t$ and proceed to next state $\mathbf{s}_{t+1}$
    Add $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ to $\mathcal{D}$
  end
end

3 Code

You will implement the MBRL algorithm described in the previous section.

3.1 Installation

Obtain the code from https://github.com/berkeleydeeprlcourse/homework/tree/master/hw4. In addition to the installation requirements from previous homeworks, install additional required packages by running:

```
pip install -r requirements.txt
```

3.2 Overview

You will modify two files:

- model_based_policy.py
- mode_based_rl.py

You should also familiarize yourself with the following files:

- main.py
- utils.py
- run_all.sh

All other files are optional to look at.

4 Implementation

Problem 1

What you will implement: The neural network dynamics model and train it using a fixed dataset consisting of rollouts collected by a random policy.

Where in the code to implement: All parts of the code where you find

```
### PROBLEM 1
### YOUR CODE HERE
```

Implementation details are in the code.

How to run:

```
python main.py q1
```

What will be outputted: Plots of the learned dynamics model's predictions vs. ground truth will be saved in the experiment folder.

What will a correct implementation output: Your model's predictions should be similar to the ground-truth for the majority of the state dimensions.

Problem 2

What will you implement: Action selection using your learned dynamics model and a given cost function.

Where in the code to implement: All parts of the code where you find

```
### PROBLEM 2
### YOUR CODE HERE
```

Implementation details are in the code.

How to run:

```
python main.py q2
```

What will be outputted: The log.txt file saved in the experiment folder will tell you the ReturnAvg and ReturnStd for the random policy versus your model-based policy.

What will a correct implementation output: The random policy should achieve a ReturnAvg of around -160, while your model-based policy should achieve a ReturnAvg of around 0.

Problem 3a

What will you implement: MBRL with on-policy data collection.

Where in the code to implement: All parts of the code where you find

```
### PROBLEM 3
### YOUR CODE HERE
```

Implementation details are in the code.

How to run:

```
python main.py q3
```

What will be outputted: The log.txt / log.csv file saved in the experiment folder will tell you the ReturnAvg for each iteration of on-policy data collection.

What will a correct implementation output: Your model-based policy should achieve a ReturnAvg of around 300 by the 10th iteration.

Problem 3b

What will you implement: You will compare the performance of your MBRL algorithm when varying three hyperparameters: the number of random action sequences considered during actions selection, the MPC planning horizon, and the number of neural network layers for the dynamics model.

Where in the code to implement: There is nothing additional to implement.

How to run:

```
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 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 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
```

What will be outputted: Three plots will be saved to the plots/ folder.

Extra Credit

Implement one of the following improvements:

(i) Instead of performing action selection using random action sequences, use the Cross Entropy Method (CEM). (See pseudo-code on Wikipedia.) How much does CEM improve performance?

(ii) The current loss function is only for one-step (i.e., we train on (s_t, a_t, s_{t+1}) tuples). Extend the loss function to be multi-step (i.e., train on $(s_t, a_t, ..., a_{t+N-1}, s_{t+N})$) for $N > 1$. How much do multi-step losses improve performance?

5 PDF Deliverable

You can generate all results needed for the deliverables by running:

```
bash run_all.sh
```

Please provide the following plots and responses on the specified pages.

Problem 1 (page 1)

(a) Provide a plot of the dynamics model predictions when the predictions are mostly accurate.

(b) For (a), for which state dimension are the predictions the most inaccurate? Give a possible reason why the predictions are inaccurate.

Problem 2 (page 2)

(a) Provide the ReturnAvg and ReturnStd for the random policy and for your model-based controller trained on the randomly gathered data.

Problem 3a (page 3)

(a) Plot of the returns versus iteration when running model-based reinforcement learning.

Problem 3b (page 4)

(a) Plot comparing performance when varying the MPC horizon.

(b) Plot comparing performance when varying the number of randomly sampled action sequences used for planning.

(c) Plot comparing performance when varying the number of neural network layers for the learned dynamics model.

[optional] Extra credit (page 5)

(a) Plot comparing performance of either CEM to random for action selection, or multi-step loss training to single-step loss training.

6 Submission

Turn in both parts of the assignment on Gradescope as one submission. Upload the zip file with your code to **HW4 Code**, and upload the PDF of your report to **HW4**.