Project 2: Model-Based Reinforcement Learning

24-784 Special Topics: Trustworthy AI Autonomy

Prof. D. Zhao

Due: Friday Mar 3rd, 23:59:59 EST

- Your online version and its timestamp will be used for assessment.
- We will use Gradescope to grade. The link is on the panel of CANVAS.
- Submit **your_code.py** or **your_code.ipynb** or **your_code.zip** to Gradescope under **Project 2 - Programming** and your solutions in **.pdf** format to **Project 2**. Insert the generated images and plots in the **.pdf**. We will randomly test your code and manually check all answers.
- Please familiarize yourself with Python programming, if you have never used it previously. We only accept Python for the coding of all howeworks. To that end, reviewing the recorded Python recitation at the beginning of this course is highly recommended. Also check out the Pytorch tutorial listed at the Reference section.
- We advise you to start with the assignment early. All the submissions are to be done before the respective deadlines. For information about available credits for late days, refer to the syllabus in CANVAS.
- Running the whole Colab file ideally will take less than 15 mins
- The accompanying Google Colab Notebook is available at: https://colab.research.google.com/drive/11AFxoA-VQwZYCt3Qs4N4uI4hEu9tpmxw?usp=sharing

1 Introduction

In this homework you are going to experiment model-based reinforcement learning (MBRL) algorithms using **PyTorch**[1] and **GPyTorch**[2]. You will experiment with the **parking-v0**[3]. Your goal is to park in a given space with the appropriate heading using two different models: neural network (NN), and Gaussian Process (GP).

For this homework, our problem is a goal-conditioned continuous control task where an agent drives a car by controlling the gas pedal and steering angle and must park in a given location with the appropriate heading. You are provided a Colab Notebook file which will help you install necessary packages. The task terminates if the vehicle successfully parks from an initial location to the parking point while achieving an appropriate orientation or if the agent moves beyond the parking area. For the parking task, the state consists of positions, velocity, and heading angle while the action consists of acceleration and steering.

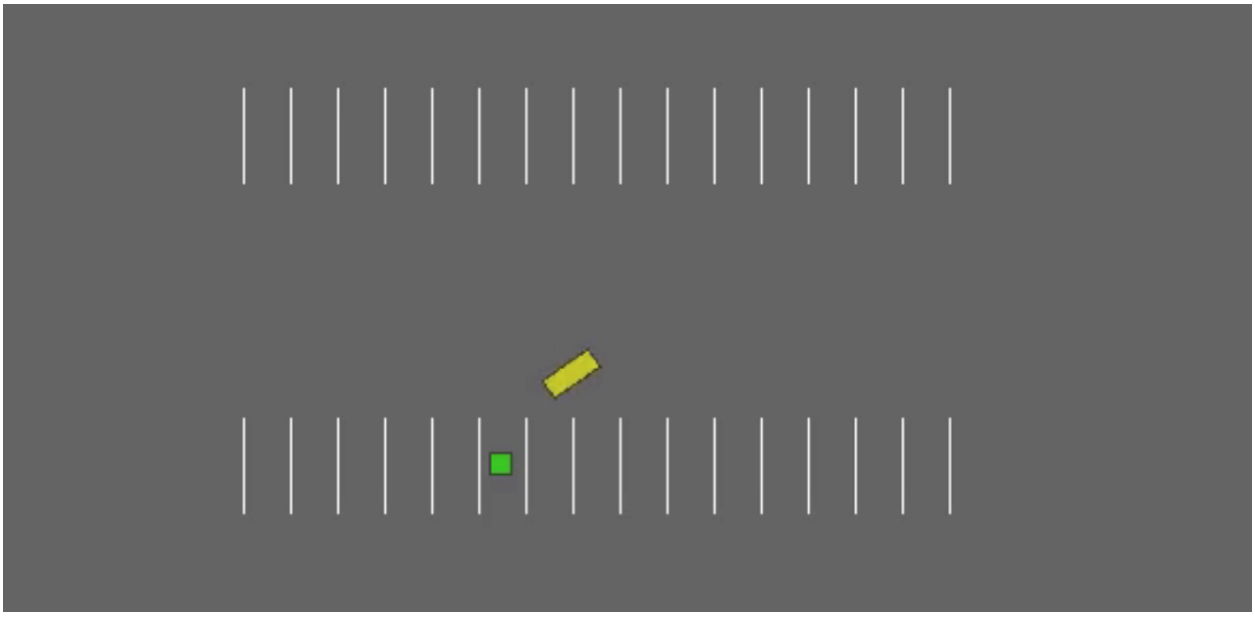

Figure 1: The rendered image from the parking-v0 environment

2 Model

There are two main components of MBRL: (1) learning a dynamics model which output a state or a distribution over states given a state and action pair $(\mathbf{s}, \mathbf{a})$, (2) using the learned dynamics model to do planning and select the best action that minimize a predefined cost function.

[1]PyTorch tutorial: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
[2]GPyTorch's documentation: https://docs.gpytorch.ai/en/stable/
[3]Environment documentation: https://github.com/eleurent/highway-env

2.1 Learning dynamics model

MBRL presumes an underlying but unknown dynamic f that governs the state transitions of the environment, say from $\mathbf{s}_t$ at time t, to $\mathbf{s}_{t+1}$ at time $t+1$ after taking action $\mathbf{a}_t$:

$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t).$$

A model f_θ is then used to replace f in the decision-making process to help decide the course of actions $\mathbf{a}_1, \cdots, \mathbf{a}_n$ that maximize the reward. Hence, we want to use a model f_θ that is accurate (to help predict the outcome) and tractable (to get the actions quickly). In this homework, we consider the following models:

Neural Networks model: We now design a model to represent the system dynamics. We choose a structured model inspired from linear time invariant systems

$$\dot{x} = f_\theta(x, u)$$

where the (x, u) notation comes from the Control Theory community and represents the state and action (s, a) in our setting. That is, we consider a model

$$\hat{\mathbf{s}}_{t+1} = \mathbf{s}_t + \Delta\hat{\mathbf{s}}_t \tag{1}$$
$$\Delta\hat{\mathbf{s}}_t = f_\theta(\mathbf{s}_t, \mathbf{a}_t). \tag{2}$$

In this part of the homework, the model f_θ is a fully connected neural network with one hidden layer. This is essentially a regression task, so we can use MSE loss:

$$L_{MSE}(\theta) = \frac{1}{n}\sum_{t=1}^{n} \|\hat{\mathbf{s}}_{t+1} - \mathbf{s}_{t+1}\|_2^2 \tag{3}$$

Gaussian Process model: In this assignment, we assume that the output of the dynamics model is a Gaussian distribution over states. Hence, we learn the mean parameter $\mu_\theta(\mathbf{s}_t, \mathbf{a}_t)$ and covariance Σ_θ:

$$\hat{\mathbf{s}}_{t+1} = \mathbf{s}_t + \Delta\hat{\mathbf{s}}_t \tag{4}$$
$$\Delta\hat{\mathbf{s}}_t \sim \mathcal{N}(\mu_\theta(\mathbf{s}_t, \mathbf{a}_t), \Sigma_\theta) \tag{5}$$
$$(\mu_\theta(\mathbf{s}_t, \mathbf{a}_t), \Sigma_\theta) = f_\theta(\mathbf{s}_t, \mathbf{a}_t) \tag{6}$$

To learn these models, we assume a history dataset $\mathcal{D} = \{[\mathbf{s}_t, \mathbf{a}_t, \Delta\mathbf{s}_t]_{t=1}^n\}$ is maintained, where $\Delta\mathbf{s}_t = \mathbf{s}_{t+1} - \mathbf{s}_t$ is the true state increment. The training of f_θ aims at minimizing negative log-likelihood over the collected data $\mathcal{D}$ with gradient descent. Formally, the objective to minimize is

$$L_{NLL}(\theta) = \sum_{t=1}^{n} \log(|\Sigma_\theta|) + (\Delta\mathbf{s}_t - \mu_\theta(\mathbf{s}_t, \mathbf{a}_t))^\intercal \Sigma_\theta^{-1} (\Delta\mathbf{s}_t - \mu_\theta(\mathbf{s}_t, \mathbf{a}_t)) \tag{7}$$

2.2 Model Predictive Control with Cross Entropy Method

This section introduces Model Predictive Control (MPC) which aims to obtain a policy $\pi(\mathbf{s}_t, \mathbf{a}_t)$ using a learned dynamics model f_θ. The objective is to maximize the accumulated reward w.r.t a sequence of actions $\mathcal{A} = \{\mathbf{a}_t, ..., \mathbf{a}_{t+H}\}$, where H is the planning horizon. Formally MPC aims to solve the following optimization problem:

$$\arg \max_{\mathbf{a}_t, ..., \mathbf{a}_{t+H}} \quad \mathbb{E}\left[\sum_{t'=t}^{t+H} \gamma^t \hat{r}_{t'}\right], \ \hat{\mathbf{s}}_{t'+1} = \hat{\mathbf{s}}_{t'} + f_\theta(\hat{\mathbf{s}}_{t'}, \mathbf{a}_{t'}), \ \hat{\mathbf{s}}_t = \mathbf{s}_t \tag{8}$$

where γ is the discount factor and $\hat{r}_{t'} = r(\hat{\mathbf{s}}_{t'}, \mathbf{a}_{t'})$ is the reward using predicted state $\hat{\mathbf{s}}_{t'}$ at step t'. Note that the prediction may not be exactly the same as the future reward $r_{t'} = r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$, but still meaningful if $\hat{\mathbf{s}}_{t'} \approx \mathbf{s}_{t'}$ for all $t' > t$ in the planning horizon. This amounts to maximize the discounted sum of the reward of the predicted future states during the planning horizon.

You are provided with a popular random shooting algorithm to solve (8), the Cross Entropy Method (CEM). At time step t, K random sequences of actions $\mathcal{A}$ are sampled from a diagonal Gaussian distribution $\mathbf{a}_{t:t+H} \sim \mathcal{N}(\mu_{t:t+H}, \sigma^2_{t:t+H})$. The first element of the sequence with maximum expected rewards are executed. The mean and variance of the sample distribution are then re-fitted based on the elites of the sampled trajectories. This process is repeated at each time step. The code is provided in the Colab notebook for this homework.

3 Resources

We provide a Colab notebook to help you set up the environment. The main packages to use is gym, the highway environment, and GPyTorch, which can be installed by running

```
!pip install gym
!pip install git+https://github.com/eleurent/highway-env#egg=highway-env
!pip install gpytorch
```

Other standard packages like Numpy, Matplotlib, and PyTorch are also required for plotting, basic arithmetic, and running the models. Below are the main calls that you would need. More details are available in the documentation.
In addition, we will also need virtualdisplay to visualize the behaviour of the agent, installed using the following command:

```
!pip install pyvirtualdisplay
```

Other settings will be added in the notebook file.

4 Problem Set

Exercise 1. Understanding the Parking-v0 environment.

1. Run one episode with random actions. Check the observation dictionary returned by the environment. Store the observations (states) and rewards over time! What is your target state (desired goal) $\mathbf{s}^*$? What is your final state $\mathbf{s}_n$ at the 100-th time step? Plot the values of all the states over time!

2. Calculate the reward of the environment manually. The reward r_t is defined as the negative of the weighted Euclidean norm of $|s_t - s^*|$. That is, for some weight $\beta = [\beta^{(1)}, \cdots, \beta^{(d)}]$,
$$r_t = -\sqrt{\sum_{i=1}^{d} |s_t^{(i)} - s^{(i)}| \ \beta^{(i)}},$$
where d is the dimension of the state variables and $\beta = [1, 0.3, 0, 0, 0.02, 0.02]$. (Hint: Compare your manual calculation with the environment reward to check if you are correct!)

3. Plot the positions (x and y coordinate) of the vehicle! What is the reward at the final time step in the episode? Show the rendered image of the environment at the final time step in the episode!

Exercise 2. Model Based Reinforcement Learning with Neural Network.

1. Construct a NN model with one hidden layer with random initialization (the structure is given in the Colab notebook). Implement the forward pass. Perform one run of episode and plot the untrained NN predictions and the true values of all the states over time!

2. Create a batch of experiences dataset $\mathcal{D}$ with $n = 2000$ using the provided function. Train a NN model that consists of one hidden layer with 128 nodes and ReLU activation (the structure is already provided in the Colab notebook). Use this dataset with training and validation ratio 70%-30% using Adam optimizer with learning rate 0.01 and training epoch 1,000. Plot the MSE loss for training and validation set!

3. Perform one run of episode again and plot the trained NN predictions and the true values of all the states over time!

4. Implement the CEM planner. Combine the NN model with CEM planner. Evaluate the performance of your model when planning using a time horizon $H = 5$ and $K = 10$ possible action sequences only for $n = 20$ timesteps. Plot the rewards over time!

Exercise 3. Model Based Reinforcement Learning with Gaussian Process.

1. Construct and train the provided GP model. Train the model using dataset $\mathcal{D}$ from 2.2 with training and validation ratio 20%-80% using Adam optimizer with learning rate 0.2 and training epoch 15. Note that GP is more sample-efficient than NN, so we need fewer training samples for GP. More training samples will also slow down the speed. Plot the NLL loss for training and validation set!

2. Perform one run of episode again for only $n = 20$ timesteps and plot the trained GP predictions and the true values of all the states over time!

3. Combine the GP model with planning. Evaluate the performance of your model when planning using a time horizon $H = 5$ and $K = 10$ possible action sequences only for $n = 20$ timesteps. Plot the rewards over time!

4. Compare NN and GP with the following metrics: final reward, prediction error, and computation time!

5 Reference

1. CMU 24784 Spring 2023 Project 2 Google Colab Notebook setup, https://colab.research.google.com/drive/11AFxoA-VQwZYCt3Qs4N4uI4hEu9tpmxw?usp=sharing

2. PyTorch tutorial, https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

3. GPyTorch documentation, https://docs.gpytorch.ai/en/stable/

4. Environment documentation, https://github.com/eleurent/highway-env