Homework 5

CMU 10-403: Deep Reinforcement Learning and Control
(Spring 2020)
OUT: Apr. 20, 2020
DUE: May. 1, 2020 by 11:59pm

Instructions: START HERE

- **Collaboration policy:** You may work in groups of up to three people for this assignment. It is also OK to get clarification (but not solutions) from books or online resources after you have thought about the problems on your own. You are expected to comply with the University Policy on Academic Integrity and Plagiarism[1].

- **Late Submission Policy:** You are allowed a total of 10 grace days for your homeworks. However, no more than 3 grace days may be applied to a single assignment. Any assignment submitted after 3 days will not receive any credit. Grace days do not need to be requested or mentioned in emails; we will automatically apply them to students who submit late. We will not give any further extensions so make sure you only use them when you are absolutely sure you need them. See the Assignments and Grading Policy here for more information about grace days and late submissions: https://cmudeeprl.github.io/Spring202010403website/logistics/

- **Submitting your work:**

  - **Gradescope:** Please write your answers and copy your plots into the provided LaTeX template, and upload a PDF to the GradeScope assignment titled "Homework 5." Additionally, upload your code (as a zip file) to the GradeScope assignment titled "Homework 5: Code." Each team should only upload one copy of each part. Regrade requests can be made within one week of the assignment being graded.

[1] https://www.cmu.edu/policies/

Collaborators

Please list your name and Andrew ID, as well as those of your collaborators.

Environment

For this assignment, you will work with the `Pushing2D-v1` environment, note that this environment is slightly **different** from the one you used in the previous homework. In particular, the state is now 10 dimensional and includes the velocity for the pusher and the box. To recap, in order to make the environment using `gym`, you can use the following code:

```
import gym
import envs

env = gym.make("Pushing2D-v1")
```

Then you can interact with the environment as you usually do. Similarly to Homework 4, the environment is considered "solved" once the percent of success (i.e., the box reaches the goal within the episode) reaches 90%. A `render` option is provided in the step function to provide a visualization.

1 Model-Based Reinforcement Learning with PETS [90pts]

For this section, you will implement a model-based reinforcement learning (MBRL) method called **PETS** which stands for probabilistic ensemble and trajectory sampling [1].

There are 3 main components to MBRL with PETS:

1. Probabilistic ensemble of networks: you will be using probabilistic networks that output a distribution over the resulting states given a state and action pair.

2. Trajectory sampling: propagate hallucinated trajectories through time by passing hypothetical state-action pairs through different networks of the ensemble.

3. Planning with model predictive control: Use the trajectory sampling method along with a cost function to perform planning and select good actions.

We will start with the third component. You will first implement CEM and MPC and test it on a ground truth dynamics. Then, you will replace the ground truth dynamics with a learned, probabilistic ensemble dynamics model. It is recommended to go through all the questions before you start the implementation.

1.1 Planning with Cross Entropy Method(CEM) [20 pts]

Before learning the model, let us first understand how do we obtain a policy from a dynamics model $p(s_{t+1}|s_t, a)$. One approach is to train a model-free policy on the imagined trajectories generated by the model. Another simpler approach is to do random shooting, where an action

Algorithm 1 Cross Entropy Method (CEM)

1: **procedure** CEM(population size M, # elites e, # iters I, initial mean μ, initial std σ, plan horizon T, action dimensionality A)
2: **for** i in $1 : I$:
3: Sample M action sequences $a_{1:T,m}$ according to μ and σ from normal distribution
4: where $a_{1:T,m}, \mu \in \mathbb{R}^{A \times T}$ and $\sigma \in \mathbb{R}^{AT \times AT}$ is a diagonal covariance matrix.
5: **for** m in $1 : M$:
6: Sample a trajectory $\tau_m = (s_1, a_1, ..., a_T, s_{T+1})_m$ using the dynamics
7: model $p(s_{t+1}|s_t, a_t)$ and the action sequence $a_{1:T,m}$ where $s_t \in \mathbb{R}^S, a_t \in \mathbb{R}^A$.
8: Calculate the cost of $a_{1:T,m}$ based on τ_m.
9: Update μ and σ using the top e action sequences.
10: **return:** μ
11: **end procedure**

sequence $a_{t:t+\tau}$ is optimized on the model to minimize a cost, which often works well. The Cross Entropy Method (CEM) is one of the popular random shooting algorithms. To make things precise, we show CEM in Algorithm 1. Note that when updating the variance of the actions, we assume that the actions across time are independent.

Cost Function In Line 8 of Algorithm 1, we need a cost function to evaluate the fitness of different states and action pairs. Defining the right cost function is often the hardest part of getting model-based reinforcement learning to work, since the action selection and resulting trajectories from CEM depend on the cost function. For this homework, you will be using the following cost function:

$$\text{cost}(\textbf{pusher}, \textbf{box}, \textbf{goal}, \textbf{a}) = d(\text{pusher}, \text{box}) + 2d(\text{box}, \text{goal}) + 5\left|\frac{\text{box}_x}{\text{box}_y} - \frac{\text{goal}_x}{\text{goal}_y}\right| \quad (1)$$

where $d(p, q)$ is the Euclidean distance between points $p \in \mathbb{R}^2$ and $q \in \mathbb{R}^2$. This function is already implemented in `obs_cost_fn()` of *mpc.py*. Given this state based cost, you can get the cost of a trajectory by summing up all the cost of all the states within one trajectory.

Policy from CEM When sampling trajectories using a policy with CEM, given a state, we can run Algorithm 1 and generate an action sequence of length T, which we can use in the future T time steps. On top of CEM, one thing can be done to give better planning results is Model Predictive Control (MPC), where we only use the first action in the planned action sequence and re-run CEM for each time step. The pseudocode is shown in Algorithm 2. MPC proceeds by starting with an initial μ and σ and use that as input to CEM. Then we take the updated μ from CEM and execute the action in the first time step in the environment to get a new state that we use for MPC in the next time step. We then update the μ that is used for the next timestep to be the μ from CEM for the remaining steps in the plan horizon and initialize the last time step to 0. Finally, we return all the state transitions gathered so that they can be appended to D.

Dynamics Model For Problem 1.1, use the ground truth dynamics, which can be accessed through the `env.get_nxt_state` function, which takes in the current state and action and generate the next state.

The hyper-parameters are summarized in Table 1.

Parameter	Value
Population size M	200
Action dimensionality A	2
State dimensionality S	8
# elites e	20
# iters I	5
# plan horizon T	5
Initial mean μ	$\mathbf{0} \in \mathbb{R}^{A \times T}$
Initial std σ	$0.5\mathbf{I} \in \mathbb{R}^{AT \times AT}$

Table 1: Summarized hyper-parameters.

Algorithm 2 Generating an episode using MPC

procedure MPC(env, plan horizon T)
  `transitions` $= []$
  $s =$ `env.reset()`
  $\mu = \mathbf{0}$, $\sigma = 0.5\mathbf{I}$
  while not done:
    $\mu =$CEM(200, 20, 5, μ, σ)
    $a = \mu[0, :]$
    $s' =$ `env.step(a)`
    `transitions.append`(s, a, s')
    $s = s'$
    $\mu = \mu[1:T]$`.append(0)`
  return: `transitions`
end procedure

1. **(5 pts)** Before you begin to implement, we recommend going through how `run.py` works in a top-down manner. See how each component of the codes work toghether. For this question, implement CEM (`solve` function) in `cem.py`. You also need to implement `act` and `predict_next_state_gt` functions in `mpc.py`. Then test it on the `Pushing2D-v1` environment. The CEM policy you implement will also be used later for planning over a learned dynamics model. For all the questions in 1.1, we provide the starter code in the `ExperimentGTDynamics` class in `run.py`. All the hyper-parameters are provided in the code. Report the percentage of success over 50 episodes.

2. **(5 pts)** Instead of CEM, plan with random action sequences where each action is generated independently from a normal distribution $\mathcal{N}(\mathbf{0}, 0.5\boldsymbol{I})$, where $\boldsymbol{I}$ is an identity matrix. Use the same number of trajectories for planning as CEM, i.e. for each state, sample M*I trajectories of length T and pick the best one. Implement this random planning method (`solve` function) in `randopt.py`. Report the percentage of success over 50 episodes. How does its performance compare to CEM?

3. **(10 pts)**

   (a) (2 pts) Implement MPC for sampling trajectories in the function `act` in `mpc.py`.

   (b) (5 pts) We provide another environment `Pushing2DNoisyControl-v1`, where a control noise is added to the algorithm. Test the two algorithm CEM and MPC+CEM on both environments and report the percentage of success over 50 episodes for each of them.

   (c) (3 pts) Which algorithm performs better on which environments? Why? Discuss the pros and cons of MPC.

1.2 Probabilistic Ensemble and Trajectory Sampling (PETS) [70 pts]

Probabilistic Ensemble

Now, we will first try to learn a single probabilistic dynamics model but will show the algorithm in the form of learning the probabilistic ensemble.

You are provided starter code in `model.py` that specifies that model architecture that you will be using for each member of the ensemble. Specifically, each network is a fully connected network with 3-hidden layers, each with 400 hidden nodes. If you have trouble running this network, a smaller network may work as well, but may require additional hyperparameter tuning. The starter code also includes a method for calculating the output of each network, which will return the mean and log variance of the resulting states.

The training routine for the ensemble is shown in Algorithm 3. Consider defining a list of operations that you can run simultaneously with a single call to `Session.run()`. This will help speed up training significantly.

Algorithm 3 Training the Probabilistic Ensemble

procedure TRAIN(data D, # networks N, # epochs E)
    Sample $|D|$ transitions from D for each network (sample with replacement).
    for e in $1 : E$:
        for n in $1 : N$:
            Shuffle the sampled transitions for network n.
            Form batches of size **128**.
            Loop through batches and take a gradient step of the loss for each batch.
end procedure

Single probabilistic network (40 pts) Implement and train a single probabilistic network for 100 epochs on transitions from 1000 randomly sampled episodes and answer the following questions. You need to implement functions in `model.py` and `predict_next_state_model` function in `mpc.py`.

Note: As the dynamics model is now probabilistic, when planning over the model using CEM (i.e. when generating trajectories from the dynamics model), you should sample P trajectories and use the average cost of the P trajectories. In this homework, we recommend using $P = 6$.

1. (5 pts) The loss that you should use to train each network is the negative log likelihood of the actual resulting state under the predicted mean and variance from the network. Given state transition pairs s_t, a_t, s_{t+1}, assuming the output distribution of the network is a Gaussian distribution $\mathcal{N}(\mu_\theta(s_n, a_n), \Sigma_\theta)$, where μ_θ and Σ_θ are outputs of the network. Derive the loss function $\mathcal{L}_{\text{Gauss}}$ for training the network. You will note that both the μ and the Σ depend on the input state and action.

2. (15 pts) Plot the loss and RMSE vs number of epochs trained for the single network.

3. (5 pts) Combine your model with planning using randomly sampled actions + MPC. Evaluate the performance of your model when planning using a time horizon of 5 and 1000 possible action sequences. Do this by reporting the percent successes on 50 episodes.

4. (10 pts) Combine your model with planning using CEM+MPC. Evaluate the performance of your model when planning using a time horizon of 5, a population of 200, 20 elites, and 5 epochs. Do this by reporting the percent successes on 50 episodes.

5. (5 pts) Which planning method performs better, random or CEM? How did MPC using this model perform in general? When is the derived policy able to succeed and when does it fail?

Trajectory Sampling

With an ensemble of probabilistic dynamics model, how do we we sample trajectories from the ensemble model? One approach called TS1 sampling is to randomly pick one of the dynamics model at each time step of the trajectories. Algorithm 4 shows how TS1 determines which network to use for each time step. It is only one component of model predictive control and will be combined with the model and the planning method. Implement TS1 in `predict_next_state_model` function in `mpc.py`.

Similar to single probabilistic case, we will sample P trajectories or particles, each of which represents a trajectory of length T.

Algorithm 4 Trajectory Sampling with TS1

procedure TS1(# networks N, # particles P, plan horizon T)
    Initialize array S of dimension $P \times T$ to store network assignments for each particle.
    for p in $1 : P$:
        Randomly sample a sequence s of length T where $s \in \{1, \ldots, N\}^T$.
        Set $S[p, :] = s$.
end procedure

Algorithm 5 MBRL with PETS

procedure MBRL(# of epochs I)
    Initialize empty data array D and initialize probabilistic ensemble (PE) of models.
    Sample 100 episodes from the environment using random actions and store into D.
    Train ensemble of networks for 10 epochs.
    repeat for I epochs:
        Sample 1 episode using MPC and latest PE and add to D.
        Train PE for 5 epochs over D.
end procedure

MBRL with PETS (30 pts).

6. (5 pts) Describe in detail your implementation of PETS. Include how you implement CEM, MPC, TS1. Include any additional hyper-parameters you need to tune other than the default hyper-parameters. Your response should walk the reader through your submitted code so that he/she will understand the key components of your implementation and be able to run your code with different arguments.

7. (10 pts) Run Algorithm 5 for 500 epochs (i.e., collect 100 initial episodes and then 500 episodes using MPC). Plot the loss and RMSE vs number of epochs trained. This can take hours to run. Read though all the questions below before you start running this experiment.

8. (10 pts) Every 50 epochs, test your model with both CEM+MPC and random actions+MPC on 20 episodes and report the percent of successes. Plot this as a function of the number of epochs of PETS. Which planning algorithm performs better? Combine this result with the observation in the single dynamics model and the ground truth dynamics cases, and discuss the comparison between CEM+MPC and random actions + MPC. Make sure to include both your observations and your explaination.

9. (5 pts) What are some limitations of MBRL? Under which scenarios would you prefer MBRL to policy gradient methods like the ones you implemented in the previous homeworks?

2 Theoretical Questions [10 pts]

1. (5 pts) **Aleatoric vs epistemic uncertainty.** Aleatoric uncertainty is also known as statistical uncertainty and represents the random variability that is unpredictable, such as the outcome of a dice roll. Aleatoric uncertainty cannot be resolved by gathering more information. Epistemic uncertainty, also known as systematic uncertainty, which can be resolved by gathering more information. In the case of model-based RL, aleatoric uncertainty refers to the uncertainty due to the stochasticity in the environment and epistemic refers to the model error due to the capacity of the neural network or the difficulty in optimization. In the PETS framework, describe how we can measure the aleatoric and epistemic uncertainty. Make sure your description is specific such that a reader would be able to write practical algorithms to measure the two uncertainties based on your description.

2. (5 pts) What is the failure mode if the aleatoric uncertainty is not considered? What is the failure mode if the epistemic uncertainty is not considered? In other word, describe in what ways a model-based agent will fail in these two cases.

Important Implementation Advice

It takes quite a while to run the experiments in this homework. Please plan accordingly and get started early! Please turn your homework in on time to facilitate grading! Again, to make debugging easier, you should implement your code piecewise and test each component as you build up your implementation, i.e., test the probabilistic ensemble, test the trajectory sampling, test the cross entropy method, test the cost calculation, test the MPC, etc.

References

[1] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.