ECE433/COS435 Introduction to RL
Assignment 2: MPC, CEM; Imitation Learning
Spring 2025

Fill me in

Your name here.

Due February 17, 2025

Collaborators

Fill me in

Please fill in the names and NetIDs of your collaborators in this section.

Instructions

Writeups should be typeset in Latex and submitted as PDF. You can work with whatever tool you like for the code, but **please submit the asked-for snippet and answer in the solutions box as part of your writeup. We will only be grading your writeup.**

Grading. Questions 1 will collectively be worth total 20 points. Question 2 is optional; you will receive a 10-point bonus if you answer it correctly. Question 3-5 will also be worth 80 points, making the total score 100 points.

Question 1 (20 points): MPC with CEM for Cart-Pole Control

You are required to implement a Model Predictive Control (MPC) approach for the CartPole-v1 environment. The starter code and complete instructions can be found on linked **Google Colab here**.

Within each MPC step, use the Cross-Entropy Method (CEM) to search over a horizon H for the best sequence of actions. Execute only the first action from that sequence in the real environment, then re-plan at the next step. You need to implement them from scratch.

1. Write two functions from scratch, with the input arguments provided for each function in parentheses:
   - `cross_entropy_method(mean_probs, horizon, num_actions, sample_size, elite_frac, cem_iterations, evaluate_fn)`
   - `mpc_control(env, horizon, cem_iterations, sample_size, elite_frac, gamma)`
2. `cross_entropy_method(...)`
   - Maintains or updates a probability distribution over action sequences (length H). For simplicity here, we only fit the mean of action sequence distributions, i.e., `mean_probs`, which provides its initial values as input argument and should be returned as output after the whole update process. Please note that it should be a distribution over sequences of horizon H instead of distribution over actions at one step.
   - We consider a simple 1-dimensional discrete action space with `num_actions` actions. As an example, if `num_actions`=2, the two actions in the action space are integers: 0 and 1. If `num_actions`=4, the four actions in the action space are integers: 0, 1, 2 ,3.
   - Samples multiple candidates according to `samle_size`, evaluates them with `evaluate_fn` which yields one return for one sequence, selects "elite" sequences with high returns according to the `elite_frac` which is the ratio of "elite" sequences over the all sampled sequences, and refits the distribution over the "elite" sequences.
   - Iterates above steps to fit the distributions for `cem_iterations` steps. Returns the best sequence found over all update iterations and the final `mean_probs` (which describes the final distribution).
3. `mpc_control(env, horizon, cem_iterations, sample_size, elite_frac, gamma)`
   - On each environment step, calls the above CEM to get an action sequence of length H from the current state.
   - Executes the first action in the real environment, obtains next state, and continues until done.
   - The `env` is an environment instantiation with two functions provided:

     At the beginning, the environment is reset and yields an observation.

     ```
     obs, _ = env.reset()
     ```

     At each step, the environment is taking an action and transits to next state (with an observation and a reward yielded), and the "done" is a boolean value describing whether the environment simulation finishes:

     ```
     obs_next, rew, done, _, _ = env.step(action)
     ```

   - You will need to use subroutines `cross_entropy_method(...)` (as you write in last question) and `evaluate_fn`. The `evaluate_fn` is provided and you can use directly:

     ```
     def evaluate_fn(seq):
         temp_env = gym.make('CartPole-v1', render_mode=None)
         temp_env.reset()
         total_ret = 0.0
         discount_factor = 1.0
         for action in seq:
             # step
             s_next, rew, done_, _, _ = temp_env.step(action)
     ```

```
                total_ret += discount_factor * rew
                discount_factor *= gamma
                if done_:
                    break
            return total_ret
```

Question 2 (10-point bonus – optional): Zero-Order Policy Search using CEM

Now that you've seen how to use CEM for searching action sequences in a MPC example, let's re-use the same concept to search directly for a policy parameter vector. This is often called **"Zero-Order Policy Search,"** because we do not require gradients of the policy but use a zero-order optimization technique. We only need to evaluate how good each parameter vector is.
In this question, you need to:

- Define a parametric policy for CartPole, which is an environment with a 4-dimensional state space $\mathbb{R}^4$.
- Reuse your CEM approach to search over $w \in \mathbb{R}^4$ as policy parameters.
- Evaluate each sampled w by running a full episode, gathering total reward, picking elites, and updating the distribution.
- After some iterations, obtain a final w. Evaluate that policy and report the average return over multiple episodes.

Below is a skeleton with TODOs for you to fill in.

Solution

```
def choose_action(w, obs):
    """
    A simple linear policy for CartPole:
      action = 1 if w dot obs > 0 else 0.
    w, obs: shape (4,)
    """
    ### YOUR CODE HERE ###
    pass


def rollout_episode(env, w):
    """
    Run one full episode in 'env' using param 'w'.
    Return total (undiscounted) reward.
    """
    ### YOUR CODE HERE ###
    pass


def train_cem_policy(env, dim=4, cem_iterations=10, sample_size=50, elite_frac=0.2):
    """
    TODOs: Zero-Order Policy Search with CEM.
      1) Maintain param distribution (mean, std) in R^dim
      2) For iteration in [1..cem_iterations]:
         - sample 'sample_size' param vectors
         - evaluate each, get total reward
         - pick top 'elite_frac', update distribution
      3) Return best param or final mean
    """

    # Example init (feel free to change it)
    mean = np.zeros(dim)
```

```
    std = np.ones(dim)*2.0
    best_w = mean.copy()
    best_score = -1e9

    ### YOUR CODE HERE ###

    pass


def eval_policy(env, w, episodes=5):
    """
    TODOs: Evaluate policy param 'w' over multiple episodes
    and return average total reward.
    """
    ### YOUR CODE HERE ###
    pass
```

Question 3. (20 points). Flap Like How Flappy Sr. Taught You!

This is an introduction to implement Behavioral Cloning. The starter code and complete instructions can be found on linked **Google Colab here**.

(a) Policy Evaluation

Paste the **entire cell** implementing policy evaluation below.

Solution

```
####################
#YOUR CODE HERE
####################

```

Paste the **entire cell** for evaluating a policy that chooses actions uniformly at random below.

Solution

```
####################
#YOUR CODE HERE
####################

```

(b) Defining a Policy

Paste the **entire cell** defining a policy class over discrete actions below.

Solution

```
####################
#YOUR CODE HERE
####################

```

(c) Setting Up Behavioral Cloning

Paste the **entire cell** defining a behavioral cloning training loop below.

Solution

```
####################
#YOUR CODE HERE
####################

```

(d) Behavioral Cloning and Evaluation

Paste the **entire cell** applying behavioral cloning to `flappy_sr_notes.mat` below.

Solution

```
####################
#YOUR CODE HERE
####################

```

Question 4 (35 points): Floppy the Sloppy Ruins the Day

This is an introduction to Filtered Behavioral Cloning.

(a) What is Left???

Paste the **entire cell** applying behavioral cloning to `vandalized_notes.mat` below.

Solution

```
####################
#YOUR CODE HERE
####################

```

(b) Array of Hope???

Paste the **entire cell** defining a reweighed behavioral cloning loss below.

Solution

```
####################
#YOUR CODE HERE
####################

```

Paste the **entire cell** defining a training loop using the reweighed behavioral cloning loss below.

Solution

```
####################
#YOUR CODE HERE
####################

```

(c) Filtering Strategy I: Trajectory-Level Reweighting???

Paste the **entire cell** implementing the trajectory-level reweighting scheme below.

Solution

```
####################
#YOUR CODE HERE
####################

```

Paste the **entire cell** applying the reweighting scheme above to Filtered BC.

Solution

```
####################
#YOUR CODE HERE
####################

```

(d) Filtering Strategy II: Truncated Future Return???

Paste the **entire cell** implementing the truncated future return reweighting scheme below.

Solution

```
####################
#YOUR CODE HERE
####################

```

Paste the **entire cell** applying the reweighting scheme above to Filtered BC.

Solution

```
####################
#YOUR CODE HERE
####################

```

Question 5 (25 points): Short-Answer Questions

For this problem, we will ask you a few questions regarding the experiments you ran to hopefully further develop your intuition for BC/Filtered BC. **Limit your answer to each part to 2–3 sentences. You**

only need to choose 5 questions to answers. Note that these questions are found throughout Problem 2 in the provided `.ipynb` file.

1. What does the result of the experiment in Problem 2(a) tell you about running behavioral cloning on noisy/low-quality datasets? Intuitively, why does this happen?
(Hint: Think about what the BC loss is optimizing.)

Solution

(Your answer here.)

2. Why does the trajectory-level reweighting scheme in Problem 2(c) work? How does it affect what the BC loss is doing?

Solution

(Your answer here.)

3. What is the effect of the temperature on the weighing scheme in Problem 2(c)?
(Hint: As a starting point, think about what happens to the softmax function as $\alpha \to 0$. To make it even easier to think about, consider applying the softmax to two fixed values a, b with $a > b$ as you take this limit.)

Solution

(Your answer here.)

4. One could consider a version of the reweighting scheme from Problem 2(c), where we remove the softmax and simply define the weight for τ_i as $R(\tau_i)$. While this may work in certain circumstances, can you think of some potential pitfalls of such a weighing scheme?
(Hint: Think about the values the reward function could take in all kinds of environments).

Solution

(Your answer here.)

5. Let us explore the effect of the hyperparameter T on the weighing scheme in Problem 2(d). Give a succinct description of the weights when $T = 1$. Do you expect this to work well in general? Why or why not?

Solution

(Your answer here.)

6. Can you think of a reason why one can set T in Problem 2(d) relatively small in Flappy Bird and still obtain decent performance?

Solution

(Your answer here.)

7. What are the risks of making an error when reconstructing Flappy Sr.s flight path? How does this compare to real-world applications like self-driving cars?

Solution

(Your answer here.)

8. How can filtering strategies prevent a self-driving car from learning dangerous driving habits from human demonstrations?

Solution

(Your answer here.)

9. What ethical considerations arise in behavioral cloning when using data from competitors?

Solution

(Your answer here.)