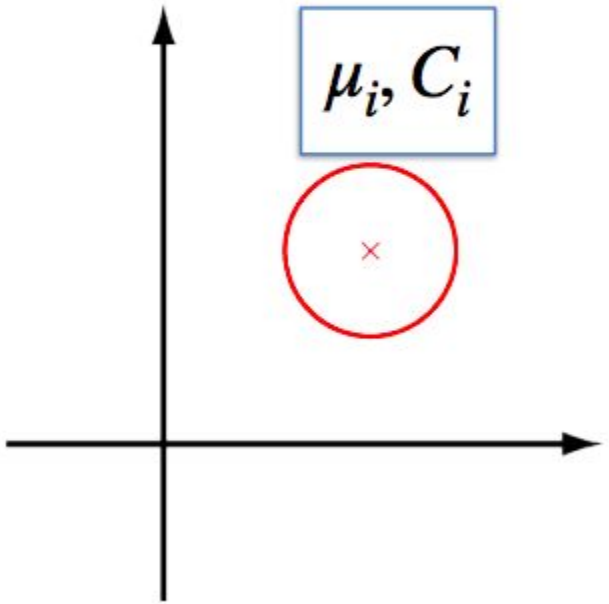
Recitation 7

Homework 3: CMA-ES, Behaviour Cloning

Question 1: CMA-ES

CMA-ES Algorithm



- Sample
- Select elites
- Update mean
- Update covariance
- iterate

CMA-ES Algorithm

$$\mu_{t+1} \leftarrow \frac{1}{\text{elite size}} \sum_{i=1}^{\text{elite size}} \theta_t^{(i)}, \qquad \Sigma_t \leftarrow \text{Cov}\left(\theta_t^{(1)}, \cdots, \theta_t^{(\text{elite size})}\right) + \epsilon I,$$

Question 1: FAQ

- Runtime less than 1 minute
- For part 3 “mean sample reward” refers to the average across all parameters, not just elite parameters
- For part 2 and 3:
  - Policy is calculated from a 5-d parameter vector
  - Reward/fitness is from sampling policy and acting in environment

$$\pi(a = \text{LEFT} \mid s) = s \cdot w + b,$$

```
def _get_action(s, params):
  w = params[:4]
  b = params[4]
  p_left = _sigmoid(w @ s + b)
  a = np.random.choice(2, p=[p_left, 1 - p_left])
  return a
```

Question 2: BC, DAGGER

1. Behavior Cloning (BC)

Input: State

Output: action

Purpose: Learning a policy in a **supervised** manner, trained by data generated from expert

Procedures:

1. Generate episodes of data [state, action] with expert policy
2. Train the student policy with the generated data

That's it!

Discussion: What's the problem of BC?

2. DAgger: Dataset Aggregation

Procedures:

generate _dataset( ):

1. Initialize D = [ ]
2. Generate episodes of data D_i = [s1, a_s1, … sn, a_sn] with **student** policy
3. Ask the **expert** to relabel the generated data D_i* = [s1, a_e1, … sn, a_en]
4. Data aggregation: D = D U D_i*

Train the student policy with the aggregated data D

Discussion: How would you compare BC with DAgger?

Question 3: GCBC

Tasks - Implement the step() function

- Implement the step() function. The vector of scalar action a is defined as self.act_set[a]. The dynamics is to attempt to move current position self.s towards the next position by adding the action vector unless the next position is the wall (it will stay at original position).
- The goal is fixed through the transition.
- The reward is constant zero and not used in this problem.
- The episode will end in the two scenarios:
    - next position is exactly the goal: success
    - time is over self.t == self.T: failure
- Please differentiate the two scenarios in info as you will use info in evaluation function evaluate_gc.
- The render_map() visualizes the map.

Tasks - Implement any shortest-path algorithm

- Implement any shortest-path algorithm on this environment. Since this is a classic maze problem, you can exactly solve it by any shortest-path search algorithms, such as breadth-first search (BFS) and Dijkstra's algorithm.
- After you implement the expert algorithm, please collect N=1000 expert trajectories in this environment, with expert_trajs as a list of expert state trajectories, and expert_actions as the corresponding list of expert action sequences. For each trajectory, you can use env.sample_sg() sample the starting state s and the goal g uniformly over the valid grids s.t. $s \neq g$.

Tasks - Implement GCBC Expert Policy

- For vanilla GCBC, implement `generate_behavior_cloning_data()` and `train()` like BC.
    - State should include goal for GCBC, rather than the state itself as in BC.
- For expert relabelling trick, implement `generate_relabel_data()`.
    - For non-terminal state s, any future state is labeled as goal. You can think this as data augmentation.
        - For a sequence of state s1, s2, …, sT and the goal g.
            - For s1, the original state is (s1, g). After relabeling, we have (s1, s2), (s1, s3), …, (s1,g).
            - For s2, the original state is (s2, g). After relabeling, we have (s2, s3), (s2, s4), …, (s2,g).
            - ...

Tasks - Implement GCBC Random Policy

- Generate Random _trajs. Random_goal is achieved each time we reset the environment, since the start state and the goal state is sampled. Random_action means the action is sampled as well, rather than the expert’s action as in the previous question.