Exercise 7

Tommaso Brumani - 100481325
ELEC-E8125 - Reinforcement Learning

November 21, 2022

Task 1

A Cross-Entropy Method model was implemented by adding the following lines to the **cem.py** file:

```
# Task 1 Implement CEM
########## Your code starts here. ##########

# 1. select actions, note plan horizon and number of samples and action dimensionality
size = (self.num_samples,self.plan_horizon,self.action_dim)
actions = np.random.normal(mean, std, size=size)

# 2. evaluate actions by computing values for your actions
## use parallel(delayed(rollout_simulator)(model, action) for each sample
action_values = parallel(delayed(rollout_simulator)(self.model, sample_act) for sample_act in actions)

# 3. select top actions (elite actions) in samples (highest returns)
partitioned_action_indices = np.argpartition(action_values, -self.num_topk)
top_action_indices = partitioned_action_indices[-self.num_topk:]
top_actions = actions[top_action_indices]

# 4. compute new mean and std, note that we used momentum for mean
_mean, _std = np.mean(top_actions, axis=0), np.std(top_actions, axis=0)
```

The resulting performance plot is reported in Figure 1.

Figure 1: Training performance plot for the CEM model.

Question 1.1

Increasing the number of samples might result in a longer running time for each iteration, as more information has to be processed for every update of the model, but it could also lead to improved performance, and possibly faster convergence, as the variance of the updates is reduced as a result of the averaging over a larger sample set.

Question 1.2

Compared to model-free methods such as DDPG, CEM has a few advantages, such as, for example, being more data efficient (as many real-world models behave linearly in the local proximity) and being more portable to other tasks employing the same model.
At the same time, it has the disadvantage of being more computationally demanding (having to learn the entire system dynamics for the problem at hand) and could obtain worse performances than model-free methods when dealing with problems that can be simulated digitally rather than physically, as its trajectory optimization is far more complex.

Task 2

Question 2.1

The probability of reaching the goal state (assuming the agent follows the random policy) is that the agent chooses the option to 'go right' at each location. This corresponds to choosing one of two options $N-1$ times, and thus the probability is:

$$P = \frac{1}{2^{N-1}}$$

If N is large, a DQN following ϵ-greedy policy will initially balance exploration and exploitation, but given the very small likelihood of reaching the goal through random exploration it will likely end up converging to a policy that always chooses to go left (as it is the locally optimal thing to do) without ever reaching the goal.

Question 2.2

Monte Carlo Tree Search is a search method for optimal decision-making that works by iteratively building a search tree in which every node is a multi-armed bandit.
The algorithm is divided into 4 phases:

1) Selection:
   From start node, actions are chosen to walk down the tree until a leaf node is reached. From here a promising child node is determined by balancing exploration and exploitation following a multi-armed bandit technique (such as UCB1).

2) Expansion:
   An action is selected and the child node is created for that action. This will be the starting point for the simulation.

3) Simulation:
   One or more random roll-outs are carried out by taking random actions until the end of the episode (or until a fixed horizon). The value of the explored state is determined as a mean return of the various simulations.

4) Backup:
   Simulated values are backpropagated to the root node, updating the values of nodes along the way through Monte Carlo estimation.

Question 2.3

Using actor-critic reduces the high variance associated with the use of Monte Carlo estimates of the value function, in which the return is calculated by summing a large number of random variables.