Written Assignment 2

● Graded

Student

Anirud N

Total Points

17.75 / 22 pts

Question 1

Q1

3 / 3 pts

✓ **– 0 pts** Correct

– 1 pt Advantages not complete.

– 1 pt Disadvantages not complete.

– 1 pt Please explain why for each points mentioned.

– 0.5 pts Advantage partially incorrect

– 0.5 pts disadvantage not complete both points are similar

– 2 pts Copied

– 2 pts Advantage and disadvantages are incorrect

Question 2

Q2

4 / 4 pts

✓ **– 0 pts** Correct

– 2 pts Partial Marks

– 4 pts Incorrect

Question 3

Q3

2 / 2 pts

✓ **– 0 pts** Correct

– 1 pt Wrong Faster Learning

– 1 pt Wrong slower learning

Question 4

Q4

3 / 3 pts

✓ **– 0 pts** Correct

Question 5

Q5 0.75 / 1 pt

- **– 0 pts** Correct
- **– 0.25 pts** Wrong 1
- ✓ **– 0.25 pts** Wrong 2
- **– 1 pt** Incorrect

Question 6

Q6 1 / 3 pts

- **– 0 pts** Correct
- ✓ **– 1 pt** Incorrect answer / It depends
- ✓ **– 1 pt** Incorrect explanation
- **– 1 pt** Insufficient explanation
- **– 1.5 pts** Incorrect explanation

Question 7

Q7 3 / 3 pts

- ✓ **– 0 pts** Correct
- **– 1 pt** Partially incorrect
- **– 0.5 pts** Show why A^n_t can be written as sum of deltas
- **– 1.5 pts** Partially incorrect
- **– 3 pts** Not attempted
- **– 0.5 pts** Missed few terms in the proof
- **– 2 pts** Incorrect
- **– 1 pt** Show those rearrangement. Its not so clear from the given proof.
- **– 2 pts** steps not clearly shown
- **– 3 pts** steps not shown

Question 8

Q8

1 / 3 pts

- **– 0 pts** Correct
- **– 1.5 pts** partially incorrect
- **– 2 pts** did not show model learning
- **– 1 pt** please show the model learning loss function.
- **– 1 pt** steps not shown clearly
- ✓ **– 2 pts** incorrect model learning
- **– 0.5 pts** for learning model, where did you get the data.
- **– 1.5 pts** pseudo code not shown
- **– 1.5 pts** where did you use update_model / learned_model ?
- **– 2 pts** difficult to understand
- **– 3 pts** not attempted
- **– 3 pts** copied
- **– 1 pt** partially incorrect
- **– 1.5 pts** incorrect model learning.

CS6700 : Reinforcement Learning
Written Assignment #2

Deadline: 3 May 2024, 11:59 pm

Name: Anirud N **Roll Number: CE21B014**

- This is an individual assignment. Collaborations and discussions are strictly prohibited.
- Be precise with your explanations. Unnecessary verbosity will be penalized.
- Check the Moodle discussion forums regularly for updates regarding the assignment.
- Type your solutions in the provided LaTeX template file.
- **Please start early.**

1. (3 marks) Ego-centric representations are based on an agent's current position in the world. In a sense the agent says, I don't care where I am, but I am only worried about the position of the objects in the world relative to me. You could think of the agent as being at the origin always. Comment on the suitability (advantages and disadvantages) of using an ego-centric representation in RL.

Solution: Ego-centric actions refer to actions performed from the agent's perspective, which are based on the environment from the agent's point of view and the agent's immediate surroundings. Allocentric, does not depend on the agent's point of view for actions. The actions and decisions would be taken based on a larger environment and a larger space.

- Ego-centric representations focus on simplifying the state space representations and reducing complexity, especially when the state space is high-dimensional.
- Ego-centric leads to reduced input, faster computation.
- Since the environment is viewed from the agent's point of view, the agent is the Origin, and all coordinates are based on the agent's position. As the agent position changes, the entire coordinate space changes, and it might lead to instability. Any error in the assumption of the agent's position with its actual position might lead to suboptimal solutions
- Ego-centric representations usually help with immediate rewards, for example, obstacle detection and avoidance or picking up an object in front of the agent. But in cases where we need to consider the entire state space, for example, route planning from one place to another - Ego Centric representations might not be good as we require a broader representation of the state space to account for global information.

2. (4 marks) One of the goals of using options is to be able to cache away policies that caused interesting behaviors. These could be to rare state transitions, or access to a new part of the state space, etc. While people have looked at generating options from frequently occurring states in a goal-directed trajectory, such an approach would not work in this case, without a lot of experience. Suggest a method to learn about interesting behaviors in the world while exploring. [*Hint: Think about pseudo rewards.*]

Solution: Introducing pseudo-rewards is one way to encourage picking up interesting behaviors.

Questions assigned to the following page: 3, 4, and 2

- Define "**Interesting Behaviours**": Provide a definition for the interesting, valuable, or rare behaviors that you wish to explore
- Implement Pseudo Rewards: Define functions for pseudo rewards that map functions with states as input and pseudo rewards as output. These pseudo-rewards should encourage the agent to explore these states. Modify your current RL Algorithm to add these pseudo rewards along with the actual rewards.
- Train the RL agent based on **maximising** the **actual rewards + pseudo reward**. Update the policies accordingly
- As the Training time and exploration increases, and as the agent learns more, gradually decrease the pseudo rewards with time. This ensures that the agent converges to optimal actions and policy over time and it does not get affected by the "interesting states" that we want to explore.

3. (2 marks) Consider a navigation task from a fixed starting point to a fixed goal in an arbitrarily complex(with respect to number of rooms, walls, layout, etc) grid-world in which apart from the standard 4 actions of moving North, South, East and West you also have 2 additional options which take you from fixed points A_1 to A_2 and B_1 to B_2. Now you learn that in any optimal trajectory from the starting point to the goal, you never visit a state that belongs to the set of states that either of the options can visit. Now would you expect your learning to be faster or slower in this domain by including these options in your learning? If it is case dependant, give a potential reason of why in some cases it could slow down learning and also a potential reason for why in some cases it could speed up learning.

Solution: It can either speed up or slow down; both are possible, depending on the case.

Slow Learning : When the option and the optimal actions are very mutually exclusive at each state, and there is less overlap. This case, the agent spends more time to learn, explore and evaluate these options which are irrelevant and not important to reach the final goal state. It might lead to slow learning as a result.

Faster Learning In cases when the options have a lot of overlap with the optimal actions,it could speed up the learning. Even if the options don't contribute to the optimal trajectory, if they are close enough, They could provide more valuable information about the state space and the environment. It might help the agent learn the outcome of a few nearby actions more quickly, and thus helps in comparison of these options with the optimal action which would be chosen as the agent learns over time.

4. (3 marks) This question requires you to do some additional reading. Dietterich specifies certain conditions for safe-state abstraction for the MaxQ framework. I had mentioned in class that even if we do not use the MaxQ value function decomposition, the hierarchy provided is still useful. So, which of the safe-state abstraction conditions are still necessary when we do not use value function decomposition.

Solution: This question has been answered after reading the paper **State Abstraction in MAXQ Hierarchical RL** by Dietterich and its inspired from the paper.
The following safe-state abstraction condition be useful when there is no MaxQ framework:

Questions assigned to the following page: 4 and 5

Subtask Irrelevance Segmenting the state space corresponding to different subtasks is essential. It ensures that each subtask operates independently of others.

Let M_i be a subtask of MDP M. A set of state variables Y is irrelevant to subtask i if the state variables of M can be partitioned into two sets X and Y such that for any stationary abstract hierarchical policy π executed by the descendants of M_i, the following two properties hold:

(a) The state transition probability distribution $P_\pi(s', N_i|s, j)$ for each child action j of M_i can be factored into the product of two distributions:

$$P_\pi(x', y', N_i|x, y, j) = P_\pi(x', N_i|x, j) \cdot P_\pi(y'|x, y, j),$$

where x and x' give values for the variables in X, and y and y' give values for the variables in Y.

(b) For any pair of states $s_1 = (x, y_1)$ and $s_2 = (x, y_2)$ and any child action j, $V^\pi(j, s_1) = V^\pi(j, s_2)$.

for example: In the Taxi problem, the source and destination of the passenger are irrelevant to the Navigate(t) subtask-only the target t and the current taxi position are relevant.

Leaf Irrelevance A set of state variables Y is irrelevant for a primitive action a if for any pair of states s_1 and s_2 that differ only in their values for the variables in Y,

$$\mathbb{E}[P(s_1' \mid s_1, a) \cdot R(s_1' \mid s_1, a)] = \mathbb{E}[P(s_2' \mid s_2, a) \cdot R(s_2' \mid s_2, a)].$$

This means that the expected values are same, only the state variables differ. This condition is satisfied by the primitive actions North, South, East, and West in the taxi task, where all state variables are irrelevant because R is constant.

Termination Let M_j be a child task of M_i with the property that whenever M_j terminates, it causes M_i to terminate too. Then the completion cost $C(i, s, j) = 0$ and does not need to be represented. This is a particular kind of funnel action - it funnels all states into terminal states for M_i.

For example, in the Taxi task, in all states where the taxi is holding the passenger, the Put subroutine will succeed and result in a terminal state for Root. This is because the termination predicate for Put (i.e., that the passenger is at his or her destination location) implies the termination condition for Root (which is the same). This means that $C(\text{Root}, s, \text{Put})$ is uniformly zero, for all states s where Put is not terminated.

5. (1 mark) In any model-based methods, the two main steps are **Planning** and **Model Update**. Now suppose you plan at a rate of F_P *(F_P times per time-step)* and update the model at a rate of F_M, compare the performance of the algorithm in the following scenarios:

   1. $F_P \gg F_M$
   2. $F_P \ll F_M$

Solution:

Case 1 : $F_p >> F_m$ The planning rate is much higher than the model update rate.- Agent plans more frequently than updation. This leads to instances when the agent decisions are based on the "Outdated" models/representations of the state space, leading to suboptimal decisions.

Questions assigned to the following page: 6 and 7

Case 2 $F_p << F_m$ Model updates are much more frequent than planning. This might lead to instability as the model keeps changing over time very frequently. Since the planning is infrequent, the agent may have difficulty to capture the planning from the model due to the instability of the model. It may not lead to convergence and leads to sub-optimal decisions

6. (3 marks) In the class, we discussed 2 main learning methods, policy gradient methods and value function methods. Suppose that a policy gradient method uses a class of policies that do not contain the optimal policy; and a value function based method uses a function approximator that can represent the values of the policies of this class, but not that of the optimal policy. Which method would you prefer and why?

Solution: The idea of policy gradient approach is to modify policy parameters directly instead of estimating the action values. It gives the probabilities of performing each action from the action space in each state. .
In a real setting, the environment is usually stochastic and noisy, with non-differentiable action and continuous state spaces. In such situations, the Policy Gradient approach is better. In this setting, the policy gradient approach converges to a sub-optimal policy that might be as close as possible to the optimal policy. They also do not have any approximation when compared to Value function methods. They are more sample efficient.
In terms of exploration, the Policy gradient approach is better
Value function methods estimate the value of each state or state-action pair and can guide the agent toward better decisions. However, the function approximations may lead to incorrect values, thereby poor exploration and leading to a sub-optimal solution farther than the optimal solution.

In a deterministic setting and a well-defined space of states and actions, Value function based method might perform well but in a realistic setting with noise and uncertainties, I feel Policy Gradient methods is better

7. (3 marks) The generalized advantage estimation equation ($\hat{A}_t^{GAE}$) is defined as below:

$$\hat{A}_t^{GAE} = (1-\lambda)\left(\hat{A}_t^1 + \lambda \hat{A}_t^2 + \lambda^2 \hat{A}_t^3 + ...\right)$$

where, $\hat{A}_t^n$ is the n-step estimate of the advantage function and $\lambda \in [0,1]$.

Show that

$$\hat{A}_t^{GAE} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

where δ_t is the TD error at time t, i.e.

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

Solution: The Advantage function is given as:

$$A_t^n = \delta_t + \gamma\delta_{t+1} + ... + \gamma^n \delta + t + n$$

Questions assigned to the following page: 7 and 8

substituting for A_n^t in the below equation

$$\hat{A}_t^{GAE} = (1-\lambda)\left(\hat{A}_t^1 + \lambda \hat{A}_t^2 + \lambda^2 \hat{A}_t^3 + ...\right)$$

gives

$$\hat{A}_t^{GAE} = (1-\lambda)\left(\delta_t + \lambda(\delta_t + \gamma\delta_{t+1}) + \lambda^2(\delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2})...\right)$$

Rearranging the terms and substituting the Infinite GP formula ie

$$1 + \lambda + \lambda^2 ...... = \frac{1}{1-\lambda}$$

gives

$$\hat{A}_t^{GAE} = \frac{(1-\lambda)}{(1-\lambda)}\left(\delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + ...\right)$$

which could be written as

$$\hat{A}_t^{GAE} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

8. (3 marks) In complex environments, the Monte Carlo Tree Search (MCTS) algorithm may not be effective since it relies on having a known model of the environment. How can we address this issue by combining MCTS with Model-Based Reinforcement Learning (MBRL) technique? Please provide a pseudocode for the algorithm, describing the loss function and update equations.

Solution:

MTCS + MBRL

Define the Environment
Initialise Q values $Q(s,a)$
Initialise a model M - Random model

While $Numberofepisodes < Threshold$

- Select an action using any action selection policy (e.g., epsilon greedy or softmax) with the current Q value estimates - Select the Node;
- If not terminal - expand the node
- Using the current Model M, rollout trajectories to the terminal state - Monte Carlo
- While $(currentnode! = root)$:
  - Update the Q values associated with each edge that reaches this node, by updating the count of state visits
- Loop through each of the state action pair generated when we simulated till the terminal state

Question assigned to the following page: 8

  - Make Predictions - using your current model M, predict the next state and reward for the current state action pair
  - if s- current state, a, action, r- reward and s' next state for the (s,a).
  - Define the Loss function as:

$$L = r + \gamma(\max_{a'}(Q(s', a')) - Q(s, a))$$

  - Backpropagate this Loss, and update M

- Loop through each of the state action pair generated when we simulated till the terminal state
  - Update action value estimates using the rollout reward:

$$Q(s, a) = Q(s, a) + \alpha(rr + \gamma \max_{a} Q(s', a') - Q(s, a))$$

    rr referes to rollout rewards