CS6700 : Reinforcement Learning
Written Assignment #2

Deadline: 3 May 2024, 11:59 pm

Name: Janmenjaya Panda **Roll Number:** ME20B087

- 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: The *egocentric representation* refers to a state representation where the agent perceives and interprets the environment solely based on its own perspective or local observations. Formally, an egocentric representation can be defined as follows:

Given an MDP $\langle \mathcal{S}, \mathcal{A}, R, P \rangle$, an egocentric representation is essentially a map $f : \mathcal{S} \to \Sigma$, which maps the state s in $\mathcal{S}$ in the global coordinate space to a point σ in the local coordinate space Σ wrt the agent so that the state of the agent in the global coordinate system at any given instance is always mapped to the origin in the local coordinate frame. We may now discuss about the advantages and the disadvantages of such representation system.

Advantages of Ego-centric representation: There are several (practical) advantages of the ego- centric representation system.

1. **Simplification of state space:** Egocentric representations often lead to a simpler state space compared to other representations. By focusing on the agent's own perspective, the state space can be reduced, which can result in faster learning and more efficient exploration.
2. **Spatial awareness:** Egocentric representations naturally encode spatial relationships between the agent and other objects in the environment. This can facilitate the learning of navigation tasks and help the agent develop a better understanding of its surroundings.
3. **Flexibility:** Egocentric representations can be more flexible in dynamic environments where the agent's position changes frequently. Since the representation is centered around the agent, it can easily adapt to changes in the agent's position without requiring significant modifications to the state space.
4. **Incomplete information of state space:** Often time, there are possibilities that we may not aware of the complete state space available to the agent; thus, modelling the environment in a global coordinate frame with an incomplete information of the environment may lead to complexity. Consequently, it is helpful to look at the environment from the agents point of view in an ego-centric represetation of the (partially available) state space.

Disadvantages of Ego-centric representation: But, there are severals disadvantages as well, as discussed below.

1. **Limited generalization:** Egocentric representations may struggle to generalize the learning/ result across different environments or tasks where the agent's perspective is not consistent. This can result in difficulties when transferring learned policies to new scenarios.
2. **Egocentric bias:** Egocentric representations may introduce biases in the agent's perception, as it only considers information relative to its own position. This can lead to suboptimal decision-making if important environmental cues are overlooked.
3. **Computational complexity:** Depending on the complexity of the environment and the number of objects or entities, maintaining an egocentric representation may require significant computational resources as it may keep on changing with the agents movement or its interaction with the environment. This can impact the scalability of the approach to larger or more complex environments.
4. **Modelling a non-inertial reference frame:** Egocentric representations may struggle to accurately consider all pseudo forces and capture the dynamics of non-inertial reference frames, particularly in scenarios with rapid accelerations or complex motion patterns. This sensitivity to motion dynamics can lead to inaccuracies in the agent's perception of the environment and consequently resulting in a substandard learning.

In summary, we may conclude the following:

The ego-centric representation provides several advantages such as simplification of the state space, spatial awareness of the agent, flexibility in dynamic environment and is suitable when the information of the entire state space is incomplete.

However, this representation is difficult to generalize, may introduce ego-centric bias, has higher computational complexity and is difficult to implement for a non-inertial environment.

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: One method to learn about interesting behaviors in the world while exploring is to use pseudo rewards. Pseudo rewards are artificial rewards designed to encourage exploration and capture specific behaviors or states of interest that may not be directly incentivized by the original reward function. Here's a suggested approach to using pseudo rewards for learning about interesting behaviors:

1. **Identify Behaviors of Interest:** Define the behaviors or states of interest that you want the agent to explore and learn about. These could be rare state transitions, accessing new parts of the state space, encountering novel situations, etc.
2. **Design Pseudo Reward Signals:** Define pseudo reward signals that encourage the agent to exhibit the behaviors of interest. Pseudo rewards should be crafted to guide the agent towards exploring and experiencing these behaviors. They should be carefully designed to avoid incentivizing undesirable behaviors.

3. **Incorporate Pseudo Rewards into Learning:** Augment the original reward function with the pseudo reward signals. During exploration, the agent receives both the original rewards from the environment and the pseudo rewards designed to encourage interesting behaviors.
4. **Update Policies Based on Combined Rewards:** Train the agent's policy using learning algorithms that take into account both the original rewards and the pseudo rewards. This encourages the agent to explore and learn about the behaviors of interest while still optimizing for the original task objectives.
5. **Adapt Pseudo Rewards Over Time:** Monitor the agent's behavior and adjust the pseudo reward signals as needed. As the agent gains experience and learns more about the environment, the relevance of certain behaviors may change, requiring updates to the pseudo rewards.
6. **Evaluate and Refine:** Continuously evaluate the effectiveness of the pseudo rewards in guiding exploration and learning about interesting behaviors. Refine the pseudo reward signals based on the agent's performance and the behaviors observed during exploration.

Conclusively, we may state the following:

By incorporating pseudo rewards into the learning process, the agent can effectively learn about and explore interesting behaviors in the environment, even in the absence of prior experience or knowledge about these behaviors. This approach enables the agent to discover novel states, transitions, or patterns that may not be immediately apparent from the original reward structure, facilitating more robust and adaptive learning.

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: Including the additional options A_1 to A_2 and B_1 to B_2 in the learning process could have different effects on learning speed depending on various factors.

1. **Faster Learning:**
   - If the additional options lead to shortcuts or bypass certain obstacles, they could potentially speed up learning. This is because the agent can learn to utilize these options to reach the goal more efficiently, avoiding unnecessary detours.
   - In scenarios where the grid-world is highly complex with many obstacles and dead ends, the additional options might provide alternative routes that are easier to traverse, leading to faster learning.
2. **Slower Learning:**
   - If the additional options lead to confusion or complicate the learning process, they might slow down learning. For instance, if the agent has difficulty distinguishing between when to use the standard movements and when to use the additional options, it could lead to suboptimal exploration and longer convergence times.

- In cases where the grid-world layout is relatively simple and the standard movements already provide efficient paths to the goal, adding more options could introduce unnecessary complexity without significant benefits, potentially slowing down learning.

In summary, the impact of including the additional options in the learning process depends on how effectively the agent can utilize them to improve navigation efficiency in the given grid-world environment. If the options offer valuable shortcuts or alternative routes without complicating the learning process, they are likely to speed up learning. However, if they add unnecessary complexity or confusion, they might slow down learning instead.

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: We follow the paper entitled 'State Abstraction in MAXQ Hierarchical Reinforcement Learning' by Thomas G. Dietterich [1] published in 1999 to answer the problem.

Let $M := \langle \mathcal{S}, \mathcal{A}, P, R \rangle$ be an MDP. Let H be a DAG defined over subtasks $\{M_0, M_1, M_2, \dots, M_k\}$. We state the conditions required for the safe-state abstractions with the assumption that the state s of the MDP is represented as a vector of state variables.

A state abstraction can be defined for each combination of subtask M_i and child action j by identifying a subset X of the state variables that are relevant and defining the value function and the policy using only these relevant variables. Such value functions and policies are said to be abstract.

The necessary conditions are as follows:

1. **Subtask Independence:** Consider a subtask M_i within the MDP M. A set of state variables Y is deemed *irrelevant* to subtask i if the state variables of M can be divided into two sets X and Y, satisfying the following two conditions for any stationary abstract hierarchical policy π executed by M_i's descendants, that are:

   (a) The state transition probability distribution $\mathbb{P}_\pi(s', N | s, j)$ for each child action j of M_i can be expressed as the product of two distributions:

   $$\mathbb{P}_\pi(x', y', N | x, y, j) = \mathbb{P}_\pi(x', N | x, j) \cdot \mathbb{P}_\pi(y' | x, y, j), \tag{1}$$

   where x and x' represent values for the variables in X, and y and y' represent values for the variables in Y; and

   (b) For any pair of states $s_1 = (x, y_1)$ and $s_2 = (x, y_2)$ and any child action j, it holds that:

   $$V_\pi(j, s_1) = V_\pi(j, s_2). \tag{2}$$

2. **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, it holds that:

   $$\sum_{s_1'} \mathbb{P}\left(s_1' | s_1, a\right) R\left(s_1' | s_1, a\right) = \sum_{s_2'} \mathbb{P}\left(s_2' | s_2, a\right) R\left(s_2' | s_2, a\right) \tag{3}$$

3. **Result Distribution Irrelevance:** A set of state variables Y_j is deemed *inconsequential* for the outcome distribution of primitive action j if, for all abstract policies π executed by M_j and its descendants in the hierarchy, the following condition prevails:

   For all pairs of states s_1 and s_2 that differ solely in their values for the state variables in Y_j, it holds that:
   $$\forall s' \in \mathcal{S},\ \mathbb{P}_\pi(s'|s_1, j) = \mathbb{P}(s'|s_2, j). \tag{4}$$

4. **Termination:** Suppose that M_j is a child task of M_i with the property that whenever M_j terminates it results the task M_i to terminate too. Then the completion cost $C(i, s, j) = 0$ and does not required to be represented.

5. **Shieling:** Consider subtask M_i and let s be a state such that for all paths from the root of the DAG down to M_i, there exists a subtask that is terminated. Then no C values need to be represented for subtask M_i in state s, because it can never be executed in s.

Conclusively, we state that the necessary safe-state abstraction conditions are: subtask independence, leaf irrelevance, result distribution irrelevance, termination and shielding.

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: Followingly we compare the performance of the algorithm in the scenarios where the planning rate F_P is much greater than the model update rate F_M and where F_P is much smaller than F_M.

1. $\mathbf{F_P \gg F_M}$: In this scenario, planning is performed frequently compared to model updates. This means that the agent spends more time exploring and considering potential actions based on the current model of the environment. The frequent planning allows the agent to make informed decisions and adapt its policy rapidly to changes in the environment. However, since the model is not updated as frequently, there's a risk of the model becoming outdated and inaccurate over time. This scenario may lead to a high computational burden due to frequent planning iterations, but it can be effective in environments where the dynamics change slowly, and the model remains relatively stable between updates.

2. $\mathbf{F_P \ll F_M}$: In this scenario, model updates are performed much more frequently compared to planning. This means that the agent focuses more on refining its model of the environment rather than actively planning and taking actions. While frequent model updates ensure that the agent's model remains accurate and up-to-date, the limited planning may result in suboptimal decision-making. The agent may not have enough time to explore and consider different actions thoroughly before the next model update. This scenario may be more computationally efficient since planning is performed less frequently, but it may lead to slower learning and adaptation, especially in dynamic environments where rapid decision-making is necessary.

In summary, the performance of the algorithm depends on the balance between planning and model update rates. A higher planning rate relative to the model update rate allows for more rapid adaptation to changes in the environment but may lead to computational challenges and potential inaccuracies in the model. Conversely, a higher model update rate relative to the planning rate ensures a more accurate model but may result in slower decision-making and adaptation. The optimal balance between these rates depends on the specific characteristics of the environment and the learning task at hand.

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: Choosing between policy gradient methods and value function methods depends on various factors including the characteristics of the problem at hand, such as the structure of the state and action spaces, the level of stochasticity, and the complexity of the environment.

If the action and state space is continuous or if there's high stochasticity in the environment, policy gradient methods tend to be more suitable. This is because policy gradient methods directly optimize the policy by updating the parameters in the direction of higher rewards, without needing to compute value functions explicitly. They can handle continuous action spaces and stochastic environments more effectively because they work directly with the policy and do not require the computation of value functions, which can be challenging in such settings.

However, if we have a scenario where a value function approximator can represent the values of the policies in consideration, but not necessarily the optimal policy, then a value function based method might be preferable. Value function methods, such as Q-learning or SARSA, estimate the value of being in a particular state and taking a particular action. If the function approximator can accurately represent the values of the policies under consideration, even if it doesn't capture the optimal policy, then value function methods can still converge to a good solution. Additionally, value function methods often offer more stability in learning and convergence guarantees, especially in settings where the environment is not highly stochastic or the action and state spaces are not continuous.

In summary, the preference between policy gradient methods and value function methods depends on the specific characteristics of the problem, and each method has its own strengths and weaknesses. If the action and state space is continuous or there's high stochasticity, policy gradient methods may be more appropriate. However, if a value function approximator can accurately represent the values of the policies under consideration, value function methods can still converge to a good solution with stability and convergence guarantees.

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: Recall the definition of the advantage function $A : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where $\mathcal{S}$ and $\mathcal{A}$ stands for the observation space and the action space.

$$A(s_t, a_t) := Q(s_t, a_t) - V(s_t)$$

The n-step estimation of the advantage function goes as follows:

$$\begin{aligned}\hat{A}^n(s_t, a_t) &: = \hat{Q}^n(s_t, a_t) - V(s_t) \\ &\approx r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n V(s_{t+n}) - V(s_t) \\ &= \left(\sum_{l=0}^{n-1} \gamma^l r_{t+l}\right) + \gamma^n V(s_{t+n}) - V(s_t)\end{aligned}$$

Proposition 0.1.

$$\hat{A}^n(s_t, a_t) = \sum_{l=0}^{n-1} \gamma^l \delta_{t+l}$$

Proof. We shall prove this using induction on n. For the base case, aka $n = 1$, the agent follows the single step estimation, thus,

$$\hat{A}^1(s_t, a_t) = r_t + \gamma V(s_{t+1}) - V(s_t) = \delta_t$$

By induction hypothesis, suppose that the statement holds true for $n = k$, for some $k \in \mathbb{N}$ with $k \geqslant 1$, that is, it holds that:

$$\hat{A}^k(s_t, a_t) = \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}$$

With the above supposition we shall show that, it holds true for $n = k + 1$. Observe that,

$$\begin{aligned}\hat{A}^{k+1}(s_t, a_t) &= \left(\sum_{l=0}^{k} \gamma^l r_{t+l}\right) + \gamma^{k+1} V(s_{t+k+1}) - V(s_t) \\ &= \left(\sum_{l=0}^{k-1} \gamma^l r_{t+l}\right) + \gamma^k V(s_{t+k}) - V(s_t) \\ &+ \gamma^k r_{t+k} + \gamma^{k+1} V(s_{t+k+1}) - \gamma^k V(s_{t+k}) \\ &= \hat{A}^k(s_t, a_t) + \gamma^k r_{t+k} + \gamma^{k+1} V(s_{t+k+1}) - \gamma^k V(s_{t+k}) \\ &= \sum_{l=0}^{k-1} \gamma^l \delta_{t+l} + \gamma^k \left[r_{t+k} + \gamma V(s_{t+k+1}) - V(s_{t+k})\right] \\ &= \sum_{l=0}^{k-1} \gamma^l \delta_{t+l} + \gamma^k \delta_{t+k} = \sum_{l=0}^{k} \gamma^l \delta_{t+l}\end{aligned}$$

This completes the proof. □

Lemma 0.2.

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

Proof. Observe that,

$$\begin{aligned}\hat{A}_t^{GAE} &= (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\hat{A}_t^n \\ &= (1-\lambda)\sum_{n=1}^{\infty}\left(\lambda^{n-1}\sum_{l=0}^{n-1}\gamma^l \delta_{t+l}\right)\end{aligned}$$

Observe that, for any specific l, the term $\gamma^l \delta_{t+l}$ is followed starting from $n = l$ to $n \to \infty$. Thus,

$$\begin{aligned}\hat{A}_t^{GAE} &= (1-\lambda)\sum_{l=0}^{\infty}\left(\gamma^l \delta_{t+l}\sum_{n=l}^{\infty}\lambda^l\right) \\ &= (1-\lambda)\sum_{l=0}^{\infty}\left(\gamma^l \delta_{t+l}\frac{\lambda^l}{1-\lambda}\right) \\ &= \sum_{l=0}^{\infty}(\gamma\lambda)^l \delta_{t+l}\end{aligned}$$

This completes the proof. □

Thus, we conclude that

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

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: Combining Monte Carlo Tree Search (MCTS) with Model-Based Reinforcement Learning (MBRL) techniques can indeed address the limitation of MCTS in complex environments where a known model of the environment is not available. This combination allows the agent to learn a model of the environment from interactions and then use this learned model within the MCTS framework to guide exploration and decision-making.

The Alogrithm 1 capture the method to combine MCTS with MBRL. The idea is there are two phases: sampling and model update. In sampling there are further two phases: sample from the true environment and sample from the experience (aka already sampled past trajectories); these samples overall will give the estimated distribution of state-action pairs, (s, a). Our goal is to fit a model $\theta(s, a)$, which is essentially a functional approximation for the distribution of (s, a) using these sampled data. One possibility is to obtain θ

such that the KL distance between the model θ and the sampled distribution α is minimized. Observe that all these samplings are done until goal state is reached and the value function is approximated simply by taking average over sampled returns.

Algorithm 1 : MCTS with MBRL

```
 1: Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$; # of MCTS: $T$; planning iteration: $N$; epsilon parameter $\epsilon$
 2: Initialization: Initialize state value function $Q(s,a)$, policy $\pi(s,a)$, returns $\mathcal{R}(s,a)$, and model
    $\theta(s,a)$ arbitrarily for all $(s,a) \in \mathcal{S} \times \mathcal{A}$; an empty dataset $\mathcal{D}$
 3: for $k$ in $\{1, 2, \ldots T\}$ do
 4:     Choose a state $s_0$ in $\mathcal{S}$ and an action $a_0$ in $\mathcal{A}$ such that $\pi(s,a) > 0$.
 5:     Generate a training dataset $\mathcal{D}^k = \{(s_i, a_i, r_i, s_i')\}_{i=1}^n$ by completing an episode $E$ through
    interacting the true environment $\mathcal{E}$ starting from $s_0$ with action $a_0$ following the policy $\pi$.
 6:     Update $D$ by appending $D^k$
 7:     for each $(s,a)$ in $\mathcal{S} \times \mathcal{A}$ in $E$ do
 8:         $G \leftarrow$ return following the first occurrence of $(s,a)$
 9:         Append $G$ to $\mathcal{R}(s,a)$
10:         $Q(s,a) \leftarrow \text{avg}\,(\mathcal{R}(s,a))$
11:     end for
12:     Let $\alpha(s,a)$ be the distribution of the $(s', r)$ in $\mathcal{D}$
13:     Learn $\theta(s,a)$ as the following optimization problem:
            $\min_{\theta} \; KL\;(\theta(s,a) \,||\, \alpha(s,a)\,)$
            such that $\; s \in \mathcal{S}$ and $a \in \mathcal{A}$
14:     for $i$ in $\{1, 2, \ldots N\}$ do
15:         Sample a trajectory $F$ from the combined dataset $\mathcal{D}$ using policy $\pi$; let the dataset
    corresponding to this trajectory be $\mathcal{D}_i^k$
16:         Update $\mathcal{D}$ by appending $D_i^k$ to it
17:         for each $(s,a)$ in $\mathcal{S} \times \mathcal{A}$ in $F$ do
18:             $G \leftarrow$ return following the first occurrence of $(s,a)$
19:             Append $G$ to $\mathcal{R}(s,a)$
20:             $Q(s,a) \leftarrow \text{avg}\,(\mathcal{R}(s,a))$
21:         end for
22:     end for
23:     for each $s$ in the episode $\mathcal{E}$ do $\pi(s,a) = \begin{cases} 1 - \epsilon + \frac{1}{|\mathcal{A}|} & \text{if } a = \arg\max_{a \in \mathcal{A}} Q(s,a) \\ \frac{1}{|\mathcal{A}|}\epsilon & \text{otherwise} \end{cases}$ [1]
24:     end for
25: end for
```

The policy that is learnt here is ϵ-greedy, which might be subtituated with other known policies.

[1] Observe that, this is another way of represetning ϵ-greedy algorithm by specifying the probablity of each action a to be chosen at a particular state s. In case, the action space is continuous, $\pi(s,a)$ will have a PDF. analogous to the PMF discussed above (with the asumption that A is a discrete space.)

References

[1] Thomas G. Dietterich. State abstraction in maxq hierarchical reinforcement learning. Advances in Neural Information Processing Systems, 11:1008–1014, 1999.