CS6700 : Reinforcement Learning
Written Assignment #2

Deadline: 9 May 2024, 11:59 am

Name: Shuvrajeet Das **Roll Number: CS23E001**

- 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:
In an egocentric representation, the number of states is reduced because the system only considers its immediate surroundings, which helps streamline computations. Additionally, this approach enables rapid learning to avoid hazardous states with high negative rewards, such as falling off a cliff, and to navigate toward goal states with positive rewards when nearby.
However, it's essential to acknowledge that egocentric representations prioritize immediate rewards over long-term ones. Consequently, they struggle to converge when higher gamma values are imposed, hindering their ability to effectively learn and plan for future outcomes.

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: Similar to the concept of Dyna Q+ discussed in our class, we can introduce pseudo-rewards for infrequent action sequences. These pseudo-rewards would increase based on the time since the sequence was last visited. Consequently, the longer it takes to revisit a sequence, the higher the exploration bonus it receives. This approach allows us to identify rare sequences effectively.
For instance, consider a grid world problem where there exists a short route to the goal surrounded by pits with high negative rewards. Despite being the optimal path, this route is rarely taken. Our proposed method would assign high pseudo-rewards to such infrequent action sequences, enabling us to recognize and prioritize them in the learning process.

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:

Adding the options to move from A1 to A2 and from B1 to B2 can have varying impacts on the speed of learning in this navigation task.

In one scenario (Case 1), if these options significantly reduce the complexity of the problem or the search space, learning might accelerate. For example, if they enable the agent to bypass obstacles or lengthy paths, it can expedite the discovery of optimal trajectories. Consequently, the agent can concentrate on learning efficient routes without the need to explore redundant states. Conversely, if these additional options lead to suboptimal paths or introduce ambiguity, they could decelerate learning. Longer paths or increased complexity introduced by these options might compel the agent to spend more time exploring and learning less efficient strategies. In such instances, the agent may need to invest extra effort in understanding how and when to utilize these options effectively without compromising optimality.

In another scenario (Case 2), if the agent never encounters states A1 and B1 (or A2 and B2) where invoking an option is optimal, learning could be slower compared to situations where the agent encounters relevant states. Without encountering these states, the agent might miss opportunities to learn from them, which could hinder the learning process.

Therefore, the impact of including these options on learning speed hinges on how well the agent can leverage them to navigate the environment and whether they simplify or complicate the task.

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:

In Ditterich's paper, five conditions are outlined for ensuring safe state abstraction. However, when we don't employ value function decomposition, only two of these conditions remain essential to uphold the hierarchy: *Subtask Irrelevance* and *Leaf Irrelevance.* These conditions are crucial for maintaining the hierarchy without the need for complete functions.

On the other hand, the remaining three conditions - Result Distribution Irrelevance, Termination, and Shielding - are primarily employed to eliminate the necessity of maintaining complete functions. Consequently, in cases where value function decomposition is not utilized, these conditions become unnecessary.

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:

1. $FP \gg FM$:
   - The algorithm heavily prioritizes planning, performing numerous planning iterations per time-step.
   - However, infrequent model updates may lead to outdated or inaccurate models.
   - Despite extensive planning, the algorithm's adaptability to changes in the environment may be compromised.
2. $FP \ll FM$:
   - The algorithm focuses on frequent model updates, ensuring accurate representations of the environment.
   - Limited planning iterations may hinder exploration and action sequence diversity.
   - Although the model is updated regularly, the algorithm's performance may suffer due to suboptimal action selection and slower convergence towards an optimal policy.

In summary, achieving an optimal balance between planning and model update rates is essential for the algorithm to effectively learn and adapt to the environment.

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:
Preferred Approach: Value Function Method

1. Value function methods possess inherent exploration capabilities, allowing them to iteratively refine policies within their defined policy space. Through continuous updates to the value function and subsequent action selection based on these values, this method has the potential to unveil policies that perform well, even if they aren't globally optimal.
2. Value function methods offer a guarantee of improvement within the limitations of their function approximators. They converge towards the best policy representable by the chosen function approximator, ensuring a continual refinement process.
3. Policy gradient methods face a significant challenge as their policy class often excludes the globally optimal policy. This constraint severely restricts their ability to explore and discover effective policies.

Caveats:

1. The efficacy of the value function method hinges on the quality of the chosen function approximator. A suboptimal choice may struggle to accurately represent the values within its designated policy space, hindering the discovery of an effective policy.
2. Even with value function methods, the integration of a robust exploration strategy is vital. This ensures that the method explores various policies within its representable class and avoids becoming trapped in local optima.

In summary, considering the limitations of policy gradient methods in this context, the value function method emerges as a more promising approach for discovering effective policies within its representable class. However, the effectiveness of both methods ultimately depends on the specific problem domain and the appropriateness of the chosen function approximations.

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: To derive the expression for $\hat{A}^t_{GAE}$ in terms of TD errors, we'll use the definition of the Generalized Advantage Estimation (GAE) and the definition of TD error. Let's start with the definition of GAE:

$$\hat{A}_t(s_{0:\infty}, a_{0:\infty}) = Q_t(s_{t:\infty}, a_{t:\infty}) - b_t(s_{0:t}, a_{0:t-1}) \tag{2}$$

Also,

$$\begin{aligned}
\hat{A}_t^{(n)} &= \sum_{i=0}^{n-1} r_{t+i} + \gamma^n V(s_{t+n}) - V(s_t) \\
&= r_t + \gamma V(s_{t+1}) - V(s_t) + \gamma \left(\sum_{i=1}^{n-1} r_{t+i+1} + \gamma^{n-1} V(s_{t+n}) - V(s_{t+1}) \right) \\
&= \delta_t + \gamma \left(\sum_{i=0}^{n-2} r_{t+i+1} + \gamma^{n-1} V(s_{t+n}) - V(s_{t+1}) \right) \\
&= \sum_{i=0}^{n-1} \gamma^i \delta_{t+i}
\end{aligned} \tag{3}$$

$$\begin{aligned}
\hat{A}_t^{GAE(\gamma,\lambda)} &:= (1-\lambda)\left(\sum_{n=0}^{\infty}\lambda^n \hat{A}_t^{(n+1)}\right) \\
&= (1-\lambda)\left(\sum_{n=0}^{\infty}\lambda^n \sum_{i=0}^{n}\gamma^i \delta_{t+i}\right) \\
&= (1-\lambda)\left(\sum_{i=0}^{\infty}\gamma^i \delta_{t+i} \sum_{n=i}^{\infty}\lambda^n\right) \\
&= (1-\lambda)\left(\sum_{i=0}^{\infty}\gamma^i \delta_{t+i}\lambda^i \sum_{n=0}^{\infty}\lambda^n\right) \\
&= (1-\lambda)\left(\sum_{i=0}^{\infty}(\gamma\lambda)^i \delta_{t+i}\frac{1}{1-\lambda}\right) && \text{since } \lambda < 1 \\
&= \sum_{i=0}^{\infty}(\gamma\lambda)^i \delta_{t+i} && (4)
\end{aligned}$$

Hence it has been shown that $\hat{A}_t^{GAE}$ can be expressed as an infinite sum of discounted TD errors.

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:

1. *Employ Model for Initial Exploration:*
   - Establish a dedicated model of the environment employing advanced techniques from Model-Based Reinforcement Learning (MBRL).
   - When conducting simulations using Monte Carlo Tree Search (MCTS), rather than relying solely on random rollouts, harness the capabilities of the trained model to simulate rollouts, thus facilitating the acquisition of preliminary insights into the environment.
2. *Augment Model with Insights from MCTS Exploration:*
   - As MCTS traverses the environment, encountering diverse states and transitions,
   - Leverage the wealth of data accumulated during MCTS simulations, including rewards and subsequent states, to iteratively refine and enhance the MBRL model.

Algorithm: MCTS with MBRL

1. Initialize model parameters either randomly or based on prior knowledge.
2. Initialize the Monte Carlo Tree Search (MCTS) tree with a root node representing the current state.
3. Repeat until convergence:

(a) Employ MCTS to select an action:

   i. For each simulation:

      A. Traverse the tree using Upper Confidence Bound for Trees (UCT) to select nodes until a leaf node is reached.

      B. Expand the leaf node if unvisited previously, utilizing the learned model to predict next states and rewards.

      C. Conduct rollouts from the leaf node to a terminal state employing either a default policy or a learned policy.

      D. Propagate the observed rewards up the tree to update visit counts and action values.

(b) Gather data for model learning:

   i. Collect trajectories (state, action, next state, reward) from MCTS simulations.

(c) Update the model utilizing MBRL techniques:

   i. Minimize the loss function between predicted next states and rewards from the learned model and actual outcomes observed during simulation.

   ii. Update model parameters leveraging gradient descent or other optimization methods.

(d) Update the MCTS tree with the learned model:

   i. Employ the updated model to guide tree traversal and action selection in subsequent MCTS simulations.

Loss function for model learning:

$$\mathcal{L}(\theta) = \sum_i \left\| \hat{S}^i_{t+1} - S^i_{t+1} \right\|^2 + \left\| \hat{R}^i_t - R^i_t \right\|^2$$

where $\hat{S}^i_{t+1}$ and $\hat{R}^i_t$ are the predicted next states and rewards respectively, S^i_{t+1} and R^i_t are the actual next states and rewards observed during simulation, and θ represents the model parameters.

Update equations for model learning:

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha \nabla_\theta \mathcal{L}(\theta)$$

where α is the learning rate, and $\nabla_\theta \mathcal{L}(\theta)$ is the gradient of the loss function with respect to the model parameters θ.