CS6700 : Reinforcement Learning
Written Assignment #2

Topics: Adv. Value-based methods, POMDP, HRL **Deadline**: 30 April 2023, 11:59 pm
Name: Shivaram V **Roll Number: BE20B032**

- 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 LaTeXtemplate file.
- **Please start early.**

1. (3 marks) Recall the four advanced value-based methods we studied in class: Double DQN, Dueling DQN, Expected SARSA. While solving some RL tasks, you encounter the problems given below. Which advanced value-based method would you use to overcome it and why? Give one or two lines of explanation for 'why'.

   (a) (1 mark) Problem 1: In most states of the environment, choice of action doesn't matter.

   Solution: If the choice of action doesn't matter in most of the states of the environment, the best advanced value-based method is **Expected SARSA**. Unlike Q-learning, Expected SARSA learns the expected value of taking an action in a given state, taking into account the probabilities of each possible action being taken. This makes it better suited to environments where the choice of action may not matter in some states, as it will still learn the expected value of taking an action in those states.

   (b) (1 mark) Problem 2: Agent seems to be consistently picking sub-optimal actions during exploitation.

   Solution: Double DQN is the method of choice if the agent consistently picks sub-optimal actions during exploitation. It consists of two value functions, one to select actions and the other to evluate the actions hence reducing the correlations between them. This approach can lead to more accurate value estimates, which in turn can lead to better action selection during exploitation.

   (c) (1 mark) Problem 3: Environment is stochastic with high negative reward and low positive reward, like in cliff-walking.

   Solution: Duelling DQN can be used to overcome in environment which is stochastic with high negative reward and low positive reward. Dueling DQN is a value-based algorithm that separates the estimation of the state-value function and the state-dependent action advantage function, allowing the agent to better handle environments with a mix of high negative and low positive rewards. By decoupling the value and advantage functions, Dueling DQN can learn to distinguish between states where it is best to take an action and those where the action does not matter. In the case of cliff-walking, where there are high negative rewards and low positive rewards, this approach can be particularly useful as it allows the agent to prioritize exploration of the positive regions of the environment while avoiding the negative regions.

2. (4 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:
Advantages:

- Ego-centric representations are particularly useful in tasks where the agent needs to navigate and interact with objects in the environment. By focusing on the position of objects relative to the agent's current position, the agent can more easily reason about how to move and interact with objects.
- Ego-centric representations can simplify the state space by reducing the number of variables needed to represent the state. By focusing only on the agent's position and the positions of objects relative to the agent, the agent can ignore other factors that may be less relevant to the task at hand.

Disadvantages:

- Ego-centric representations may not be suitable for all tasks and environments. In some cases, it may be important for the agent to reason about the global state of the environment, rather than just its local position relative to the agent.
- Ego-centric representations can lead to a biased view of the environment, as the agent only considers the part of the environment that is visible from its current position. This can make it difficult for the agent to reason about long-term consequences of its actions, particularly if those consequences are not immediately visible from its current position.

3. (12 marks) Santa decides that he no longer has the memory to store every good and bad deed for every child in the world. Instead, he implements a feature-based linear function approximator to determine if a child gets toys or coal. Assume for simplicity that he uses only the following few features:

   - Is the child a girl? (0 for no, 1 for yes)
   - Age? (real number from $0 - 12$)
   - Was the child good last year? (0 for no, 1 for yes)
   - Number of good deeds this year
   - Number of bad deeds this year

   Santa uses his function approximator to output a real number. If that number is greater than his good threshold, the child gets toys. Otherwise, the child gets coal.

   (a) (4 marks) Write the full equation to calculate the value for a given child (i.e., $f(s, \vec{\theta}) = \ldots$), where s is a child's name and $\vec{\theta}$ is a weight vector $\vec{\theta} = (\theta(1), \theta(2), \ldots, \theta(5))^{\mathrm{T}}$. Assume child s

is described by the features given above, and that the feature values are respectively written as ϕ_s^{girl} , ϕ_s^{age} , ϕ_s^{last} , ϕ_s^{good} , and ϕ_s^{bad} .

Solution:

$$f(s,\vec{\theta}) = \vec{\theta} \cdot \phi_s \quad (1)$$

$$f(s,\vec{\theta}) = \theta(1) \cdot \phi_s^{girl} + \theta(2) \cdot \phi_s^{age} + \theta(3) \cdot \phi_s^{last} + \theta(4) \cdot \phi_s^{good} + \theta(5) \cdot \phi_s^{bad} \quad (2)$$

(b) (4 marks) What is the gradient $\left(\nabla_{\vec{\theta}} f(s,\vec{\theta})\right)$? I.e. give the vector of partial derivatives

$$\left(\frac{\partial f(s,\vec{\theta})}{\partial \theta(1)}, \frac{\partial f(s,\vec{\theta})}{\partial \theta(2)}, \cdots, \frac{\partial f(s,\vec{\theta})}{\partial \theta(n)}\right)^{\mathrm{T}}$$

based on your answer to the previous question.

Solution:

$$\nabla_{\vec{\theta}} f(s,\vec{\theta}) = (\phi_s^{girl}, \phi_s^{age}, \phi_s^{last}, \phi_s^{good}, \phi_s^{bad})^\top \quad (3)$$

$$\nabla_{\vec{\theta}} f(s,\vec{\theta}) = \phi_s \quad (4)$$

(c) (4 marks) Using the feature names given above, describe in words something about a function that would make it impossible to represent it adequately using the above linear function approximator. Can you define a new feature in terms of the original ones that would make it linearly representable?

Solution: Any function which has higher order in the features Age, number of good deeds and number of bad deeds, can't be representable by the above linear function approximator. To make it linearly representable, we have to include polynomial degrees of the above said features, which makes it linearly representable in the new features.

4. (5 marks) We typically assume tabula rasa learning in RL and that beyond the states and actions, you have no knowledge about the dynamics of the system. What if you had a partially specified approximate model of the world - one that tells you about the effects of the actions from certain states, i.e., the possible next states, but not the exact probabilities. Nor is the model specified for all states. How will you modify Q learning or SARSA to make effective use of the model? Specifically, describe how you can reduce the number of *real* samples drawn from the *world*.

Solution: If we have a partially specified approximate model of the world, we can modify Q-learning or SARSA to take advantage of the model and reduce the number of real samples drawn from the world by using a technique called planning.

The basic idea of planning is to use the approximate model to simulate the effects of actions and generate a sequence of states and rewards that the agent could encounter if it takes a particular action from a particular state. The agent can then use these simulated trajectories to update its value function and improve its policy without actually having to interact with the environment.

This allows the agent to reduce the number of real samples it needs to collect from the environment, making learning more efficient.

Here are the steps to modify Q-learning or SARSA to incorporate planning using a partially specified approximate model:

- At each time step, the agent selects an action to take from its current state according to its current policy.
- The agent uses the approximate model to simulate the effects of the selected action from the current state. The model provides a set of possible next states and the associated rewards, but not the exact probabilities.
- The agent uses the approximate model to simulate the effects of the selected action from the current state. The model provides a set of possible next states and the associated rewards, but not the exact probabilities.
- For each possible next state, the agent updates its Q-value estimate using the Bellman equation and the rewards and Q-values associated with that state.
- The agent updates its policy based on the updated Q-values.
- The agent repeats this process for a fixed number of planning steps, using the updated Q-values to generate new simulated trajectories and refine its value estimates and policy.

By using planning with the approximate model, the agent can reduce the number of real samples it needs to collect from the environment, making learning more efficient. However, the quality of the approximate model can have a significant impact on the effectiveness of this approach, as errors in the model can lead to incorrect value estimates and policies. Therefore, it is important to carefully evaluate the accuracy and reliability of the model before using it in planning.

5. (4 marks) We discussed Q-MDPs in the class as a technique for solving the problem of behaving in POMDPs. It was mentioned that the behavior produced by this approximation would not be optimal. In what sense is it not optimal? Are there circumstances under which it can be optimal?

Solution: Q-MDPs are a technique for solving the problem of behaving in partially observable Markov decision processes (POMDPs). In Q-MDPs, the agent uses its current belief state (a probability distribution over the states) to select an action that maximizes the expected Q-value, which is the expected sum of discounted rewards obtained by following a particular action from a particular belief state.

The behavior produced by Q-MDPs is not optimal in the sense that it does not guarantee the agent will always choose the best action at every step. This is because the agent's belief state is an approximation of the true state, and the agent may not always choose the best action based on its belief state.

However, there are circumstances under which the behavior produced by Q-MDPs can be optimal. Specifically, if the agent's belief state is close to the true state and the POMDP is sufficiently "simple" (i.e., has a relatively small number of states and actions), then the behavior produced by Q-MDPs can be close to optimal. In these cases, the approximation error introduced by using the belief state instead of the true state may be small enough that the agent's behavior is still effective.

Additionally, Q-MDPs can be combined with other techniques, such as online planning or value function approximation, to improve the performance of the agent even further. By using Q-MDPs in combination with other techniques, the agent can often achieve near-optimal behavior even in complex POMDPs.

6. (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 the MaxQ framework, safe-state abstraction is used to enable hierarchical planning by dividing the state space into safe and unsafe states. Dietterich specifies several conditions for safe-state abstraction in MaxQ, which include:

- The set of safe states should be closed under the dynamics of the environment.
- The probability of transitioning from a safe state to an unsafe state should be low.
- The probability of transitioning from an unsafe state to a safe state should be high.

When not using value function decomposition, the hierarchy provided by the MaxQ framework can still be useful in enabling hierarchical planning. However, the conditions for safe-state abstraction are still necessary to ensure that the hierarchical planning is effective and safe.

For example, if the set of safe states is not closed under the dynamics of the environment, then the agent may encounter situations where it transitions to an unsafe state from a safe state, which can lead to failures or suboptimal behavior. Similarly, if the probability of transitioning from a safe state to an unsafe state is high, then the agent may need to spend additional resources avoiding such transitions, which can reduce the efficiency of the planning process.

Therefore, even when not using value function decomposition, it is important to ensure that the conditions for safe-state abstraction are met in order to achieve effective and safe hierarchical planning.

7. (4 marks) One of the goals of using options is to be able to cache away policies that caused interesting behaviors. These could be 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 way to learn about interesting behaviors in the world while exploring is to use novelty-based exploration. The basic idea behind novelty-based exploration is to encourage the agent to explore novel states or actions that it has not encountered before. This can be achieved by defining a novelty measure that quantifies the novelty of a state or an action.

One common way to define a novelty measure is to use the idea of information gain. Specifically, the novelty of a state or an action can be measured by the reduction in uncertainty that it provides about the environment. For example, one can use the entropy of the state distribution or the action distribution as a measure of uncertainty, and the reduction in entropy as a measure of novelty.

Once a novelty measure is defined, the agent can use it to guide its exploration. One way to do this is to use an exploration policy that biases the agent towards exploring novel states or actions.

For example, the agent can use an ϵ-greedy exploration policy, where with probability ϵ it chooses a random action, and with probability (1- ϵ) it chooses the action that maximizes the Q-value.

In addition to using novelty-based exploration to learn about interesting behaviors, the agent can also use it to discover new options. Specifically, the agent can define a new option each time it encounters a novel state or action, and use it to cache away the policy that caused the interesting behavior. Over time, the agent can accumulate a library of options that it can use to efficiently navigate the state space and solve the task.