Sumeet Shanbhag

Student ID : 641020714

Reinforcement Learning Assignment 6

1. What is "planning" in the context of reinforcement learning?

1) **Model of the Environment**: The agent's model predicts what the next state and reward will be, given the current state and action taken. This model can be learned from past experience or provided by a designer. It enables the agent to forecast the outcomes of its actions without actually performing them, which is essential for planning.
2) **Methods for Planning**: With the model, the agent uses planning algorithms to determine the best action sequence. These methods include:
    - **Dynamic Programming**: Techniques like Value Iteration and Policy Iteration are applied when the model of the environment is fully known. These algorithms iteratively evaluate and improve policies.
    - **Monte Carlo Tree Search (MCTS)**: This method is especially useful when the state space is too large for exhaustive search. It builds a search tree using random simulations and focuses on the most promising parts of the search space.
    - **Heuristic Search**: Algorithms like A* or other heuristic-based methods are used when the search space is large, and we have a heuristic to estimate the distance to the goal.
3) **Purpose of Planning**: The goal of planning is twofold—predicting future rewards (prediction) and identifying the best policy for maximizing those rewards (control). By simulating different action paths, the agent can evaluate the potential long-term outcomes before committing to a specific action in the real environment.

    Planning is computationally challenging because the number of possible futures the agent can consider grows exponentially with the number of steps it looks ahead. This is known as the curse of dimensionality, and various techniques are used to mitigate this issue, such as approximations or limiting the search depth. Planning enables the agent to learn policies more efficiently by leveraging the structure of the environment. It can lead to more robust decision-making since the agent can consider a wider range of possibilities and their implications.

2. What is the difference between Dyna-Q and Dyna-Q+ algorithms?

The difference between the two are as follows:

1) **Exploration Mechanism:**
    - Dyna-Q primarily relies on external exploration strategies, such as ϵ-greedy, where it occasionally selects a random action instead of the current best-known action. This method does not inherently distinguish between rarely and frequently visited states.
    - Dyna-Q+ enhances this by adding an intrinsic motivation factor. It assigns a bonus to the estimated rewards for state-action pairs that have not been tried in a while. The bonus grows over time, which naturally encourages the agent to revisit and explore less familiar or neglected areas.
2) **Adaptability to Environmental Changes:**
    - In Dyna-Q, learning from the model is static; it doesn't inherently adjust when the environment changes unless new real experiences are encountered that lead to model updates.
    - Dyna-Q+ adjusts more dynamically to environmental changes because the exploration bonus can lead to re-sampling of parts of the environment that may have changed. This can be crucial in non-stationary environments where the dynamics can change over time.
3) **Model-Based Updates:**
    - Both algorithms use simulated experiences to update value estimates, but Dyna-Q+'s model includes the exploration bonus as a component of the reward, making its simulated experiences slightly different from those of Dyna-Q. The bonus acts as a proxy for potential new information that could be gained from re-visiting an underexplored state-action pair.
4) **Optimizing Exploration:**
    - Dyna-Q does not explicitly optimize for exploration; instead, it depends on the chosen policy (like ϵ-greedy) to balance exploration with exploitation.
    - Dyna-Q+ implicitly optimizes for exploration by prioritizing the exploration of state-action pairs that may become more valuable over time due to the lack of recent visits.

3. Model-based RL methods suffer more bias than model-free methods. Is this statement correct? Why or why not?

The statement that model-based RL methods suffer more bias than model-free methods is not completely correct; the bias in either method depends on various factors as seen below:

1) **Model-Based RL**:
    - In model-based RL, the agent uses a learned model of the environment to simulate outcomes and make decisions. If the model is incorrect or incomplete, this can introduce bias because the agent is planning based on flawed predictions about the environment. The agent's policy is only as good as the model it's based on; any errors in the model will propagate through to the policy.

- However, if the model accurately captures the environment dynamics, model-based methods can be highly efficient and less biased in their estimations because they can generalize from fewer interactions with the environment.
- ***Model-based methods might have a lower variance because they can use their model to reason about unobserved parts of the environment, but if the model is wrong, they can be biased.***

2) **Model-Free RL**:
   - Model-free methods learn a policy or value function directly from interactions with the environment without an explicit model of the environment's dynamics. These methods can be less biased towards an incorrect model since they do not rely on one, but they can still be biased in other ways.
   - The bias in model-free methods often comes from the initial assumptions of the policy or value function approximation. For example, using function approximation (like neural networks) can introduce bias based on the chosen architecture or the initial weights.
   - Additionally, model-free methods can be sample-inefficient since they need a lot of data to learn good estimates of the policy or value function. During early learning, when the amount of data is limited, these methods can be biased by the initial experiences.
   - ***Model-free methods might have a higher variance because they rely on actual samples from the environment, which can be noisy and inconsistent, especially in the early stages of learning.***

The level of bias in model-based versus model-free methods depends on the quality of the model and the learning algorithm. An inaccurate model can introduce significant bias in model-based methods, while model-free methods can be biased by their initial configurations and the need for extensive data to converge. Therefore, the statement can be correct in situations where the model used in a model-based method is significantly inaccurate, but it's not a rule that applies universally across all scenarios and applications.

4. Model-based RL methods are more sample efficient. Is this statement correct? Why or why not?

Yes, model-based reinforcement learning (RL) methods are often more sample efficient than model-free methods.

1) **Simulated Experience**: Model-based methods use a learned model to simulate interactions with the environment. This means they can generate additional data for learning without extra real-world interactions, making better use of each sample from the real environment.
2) **Planning**: These methods can use planning algorithms to determine the best actions by considering future outcomes. This lookahead ability allows the agent to learn from potential future events as well as past experiences.

3) **Generalization**: A model can generalize from seen to unseen states. Thus, learning is not strictly limited to the sampled states, and the agent can infer outcomes in unexplored parts of the state space.
4) **Efficient Exploration**: With a model, the agent can potentially explore more strategically in the actual environment, seeking out the most informative experiences.

However, the sample efficiency advantage of model-based methods comes with the assumption that the model is reasonably accurate. If the model is wrong, it can lead to poor decision-making and negate the benefits of sample efficiency. Additionally, model-based methods can be computationally intensive since they involve maintaining and updating a model as well as planning over it.

5. What are the 4 steps of the MCTS algorithm? How does MCTS balances exploration/exploitation?

Monte Carlo Tree Search (MCTS) is a heuristic search algorithm used for making decisions in certain types of games or problems. The algorithm balances exploration of uncharted areas of the search tree with exploitation of known promising areas. The four main steps of the MCTS algorithm are:

1) **Selection**: Starting from the root node, the algorithm selects child nodes until it reaches a node that represents a non-terminal state that has not been fully expanded. The selection is typically guided by a policy that balances exploration and exploitation, such as the Upper Confidence Bound applied to Trees (UCT).
2) **Expansion**: Once a non-terminal node has been selected and if it is not fully expanded (i.e., not all possible moves have been tried), then one of the possible moves is chosen and a new child node is added to the tree.
3) **Simulation**: From the new node, a simulation (also called a rollout) is performed to play out a game or sequence of actions randomly or using a lightweight policy until a terminal state is reached. This results in an outcome that provides a sample value for the state.
4) **Backpropagation**: The results of the simulation are then propagated back up the tree. Each node visited during the Selection phase has its value and visit count updated based on the simulation outcome.

Balancing Exploration and Exploitation:
In MCTS, the balance between exploration and exploitation is typically managed by the Upper Confidence Bound (UCB) formula, particularly the Upper Confidence bounds for Trees (UCT) variant. The UCB formula uses both the success rate of the node (exploitation) and the number of times the node has been visited (exploration). The exploitation part prefers nodes with a high average win rate, indicating that the path is promising based on past simulations. The exploration part prefers nodes that have been visited less often, encouraging the algorithm to explore new or less-travelled paths that may lead to better outcomes. A mathematical constant often referred to as C is used to adjust the balance; increasing C will prioritize exploration, while decreasing it will prioritize exploitation. This

balance is crucial to ensure the algorithm does not become stuck in a local optimum and continues to search for potentially better strategies.

6. **The non-planning method looks particularly poor in Figure 8.3 because it is a one-step method; a method using multi-step bootstrapping would do better. Do you think one of the multi-step bootstrapping methods from Chapter 7 could do as well as the Dyna method? Explain why or why not.**

   A multi-step bootstrapping method is expected to perform better than a 1-step method. However, it is unlikely to outperform the Dyna method because the Dyna method updates the value functions many more times. For an episode of length 'i', the multi-step method only updates values 'i' times, whereas the Dyna method updates values 'i*j' times if it performs 'j' planning steps.
   Also, multi-step bootstrapping method is model-free and cannot use the environment's learned model to improve its training data. On the other hand, Dyna is more sample-efficient and works better in probabilistic and changing environments. In simpler terms, Dyna is a more effective and efficient approach compared to multi-step bootstrapping method, especially in changing environments.

7. **Why did the Dyna agent with exploration bonus, Dyna-Q+, perform better in the first phase as well as in the second phase of the blocking and shortcut experiments?**

   The Dyna-Q+ algorithm is an improvement on the Dyna-Q algorithm, which helps an agent to explore new actions in the environment by providing an exploration bonus during action selection. This encourages the agent to try out actions that it hasn't used before, potentially leading to better results.
   In the first phase, in both examples, Dyna-Q+ finds the optimal policy much faster than Dyna-Q, that's why its slope rate quickly fixed to the optimal one and creates the gap when Dyna-Q was struggling. It is because Dyna-Q+ is so good at exploration in finding the best policy. In second phase, both algorithms find the new optimal policy in the blocking example because both can handle the situation where the environment gets worse. Again, we can see Dyna-Q+ is much faster. In the shortcut example, Dyna-Q cannot find new policy because it is very hard for a vanilla method to keep exploratory across many steps. Dyna-Q+ is forced to explore further and creates the huge gap.