COMP 138 RL: Programming Assignment 3

Shivprasad Shivratri

Dyna-Q: A conventional ϵ-greedy policy, which plans with a model that uses only previously received states and rewards.

Dyna-Q+: A conventional ϵ-greedy policy, which plans with a model that gives additional value to the action proportionate to the time passed since they were last selected. This approach adds random value to the value function.

Modified-Dyna-Q+: A novel policy that adds value to each action before selection, in the same way as Dyna Q+, then selects the value maximising action. This combines exploration and exploitation unlike the above two that keep these regimes separate.

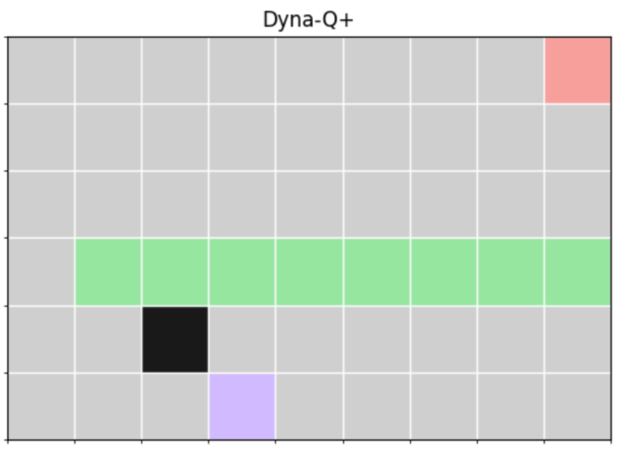

Figure 1: Blocking Maze Game

Design and Functionality of TabularDynaQ Class:

The `TabularDynaQ` class is a Python implementation of the Dyna-Q family of algorithms. The core components of the implementation are described below.

Model Representation An internal model of the environment is represented by a nested array, `self.model`, which stores transitions of the form $(s, a) \to (s', r)$ after being observed through real interaction. This model is used during planning steps to simulate experience.

Action Selection The agent selects actions using an ϵ-greedy policy:

- With probability ϵ, a random action is chosen to promote exploration.

- With probability $1 - \epsilon$, the agent selects the action with the highest estimated Q-value.

For methods like Dyna-Q+, an exploration bonus is added to the Q-values during planning, calculated as:

$$Q'(s, a) = Q(s, a) + k\sqrt{\tau(s, a)}$$

where $\tau(s, a)$ is the number of timesteps since the state-action pair was last tried, and k is a scaling hyperparameter.

Real Experience Update At each step, the agent interacts with the real environment, takes action a in state s, receives reward r, and transitions to s'. The Q-value is updated using the standard Q-learning rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

The experience is also stored in the model: `model[s][a] = (s’, r)`.

Model-Based Planning After each real interaction, the agent performs n planning steps:

1. Randomly sample a previously visited state-action pair (s, a).
2. Retrieve the stored transition (s', r) from the model.
3. Perform a Q-learning update using the simulated experience.

This allows the agent to reinforce learning even without new external input, significantly accelerating convergence.

Exploration Time Tracking For Dyna-Q+ variants, the class tracks the timestamp `self.time` and stores the last visited time for each (s, a) in `self.times`. This facilitates computing the exploration bonus used during action selection and planning.

Maze Dynamics and Environment Adaptation The agent operates within a dynamic maze environment. After a predetermined number of steps, the environment is altered (e.g., walls are added or removed). The algorithm must detect and adapt to these changes. Dyna-Q+ variants handle such non-stationary environments more robustly due to their tendency to revisit underexplored transitions during planning.

Archive of the code:

Clicking here shall redirect you to the Github Repository for Programming Assignment #3

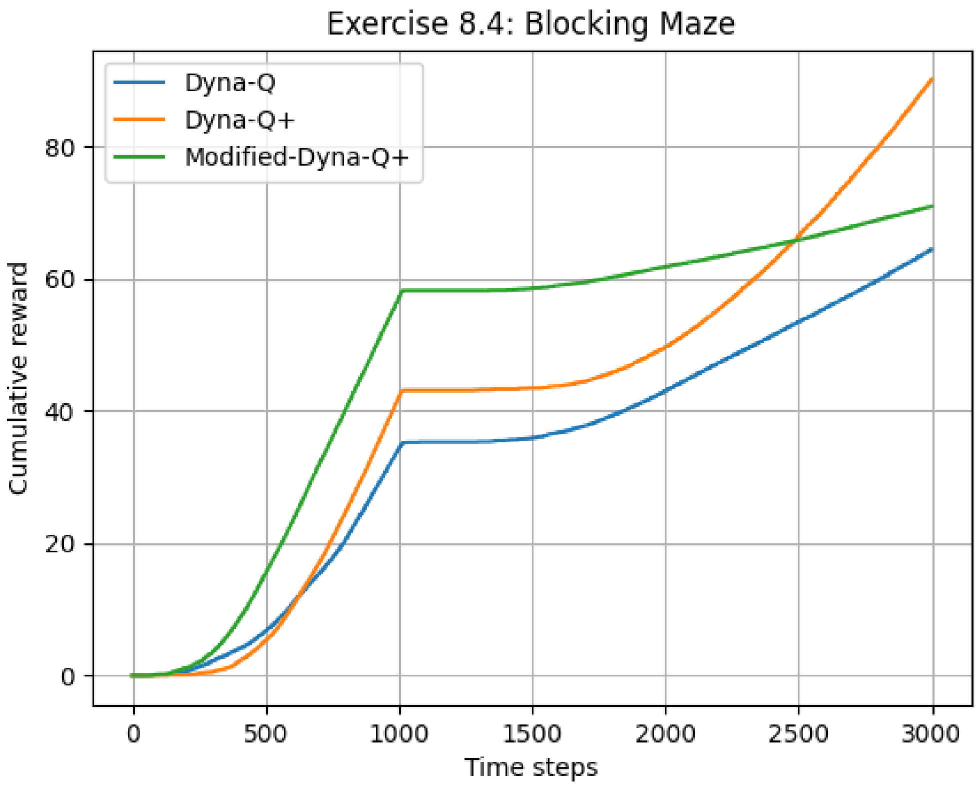

Figure 2: Cumulative reward for three model-based agents on the **Blocking Maze** task after 3000 timesteps

Results and analysis:

Figure 2 illustrates and compares the learning dynamics of three different reinforcement learning agents: Dyna-Q, Dyna-Q+, and Modified-Dyna-Q+, each operating within a gridworld environment that undergoes a structural change—namely, a wall switch—after timestep 1000. The figure highlights how each algorithm adapts (or fails to adapt) to this non-stationary aspect of the environment.

Initially, Dyna-Q demonstrates fast learning progress which is largely due to its ability to leverage a learned model of the environment to perform planning updates in addition to real-world experience. This planning component allows Dyna-Q to quickly simulate transitions and propagate value information, enabling it to identify a near-optimal policy efficiently in the early phases of learning. However, Dyna-Q's performance significantly degrades after the wall switch occurs at timestep 1000. Since its model is updated only through direct experience and does not inherently encourage exploration of unvisited or rarely visited transitions, it becomes slow to recognize and adapt to the new maze configuration. Consequently, it continues to act based on outdated environmental dynamics, leading to a period of suboptimal performance.

Dyna-Q+, in contrast, incorporates a mechanism for optimistic exploration within its planning updates. Specifically, it augments its model-based reward estimates with a bonus that increases with the amount of time since a state-action pair was last visited. This encourages the agent to periodically explore less recently visited transitions, including during simulated planning steps. As a result, while Dyna-Q+ learns slightly more slowly than Dyna-Q in the initial phase—due to the added exploration noise and a broader distribution of experience—it is much more effective at detecting and adapting to the wall switch. Once the environmental change occurs, Dyna-Q+ is more likely to re-explore previously optimal but now blocked routes, and to discover the new optimal path on the other side of the wall. Thus, it ultimately converges to a more effective long-term policy, balancing exploration during planning and exploitation during real interaction.

Modified-Dyna-Q+ presents a different set of characteristics. Before the wall switch, it is able to identify the optimal policy. The Modified-Dyna-Q+ does not incorporate any form of exploration bonus in its planning steps; exploration is performed solely through real environment interactions. Consequently, during planning, the agent reinforces knowledge only from state-action pairs it has already visited, lacking any simulated incentive to explore alternative or novel strategies. Following the wall switch, this limitation becomes more pronounced. Without directed exploration during planning, Modified-Dyna-Q+ must rely entirely on serendipitous real-world interactions to uncover the environmental change. Unless it stumbles across a series of actions that lead it to explore beyond the known region—actions that typically have low value estimates—it remains stuck in a suboptimal policy for an extended period. This makes its adaptability after the change significantly worse compared to Dyna-Q+.

In summary, while Dyna-Q exhibits a fast initial learning due to efficient value propagation via planning, its inflexibility to dynamic changes limits its long-term performance. Modified-Dyna-Q+, although capable of eventually finding optimal policies, does so inefficiently, particularly in the face of environmental change, due to its lack of exploratory planning. Dyna-Q+, by integrating exploration directly into its planning mechanism through the use of time-based reward bonuses, strikes the most effective balance. It demonstrates slower initial learning, but superior adaptability and overall long-term performance. It is important to note, however, that these outcomes can vary depending on factors such as model initialization, the choice of hyperparameters, and especially the initialization of τ, which governs the magnitude of exploration bonuses in Dyna-Q+. In some experimental runs, Modified-Dyna-Q+ fails to find the optimal policy even before the wall switch occurs, underlining its sensitivity to initial conditions and the limitations of relying solely on real-environment exploration.

Extra Question:

To further investigate the adaptability of the agents, we conducted an additional experiment in which the environment was reverted to its original configuration at 2000 timesteps, after the initial wall switch. This allowed us to observe how effectively each algorithm could re-adapt to a previously learned optimal policy after a period of environmental change.

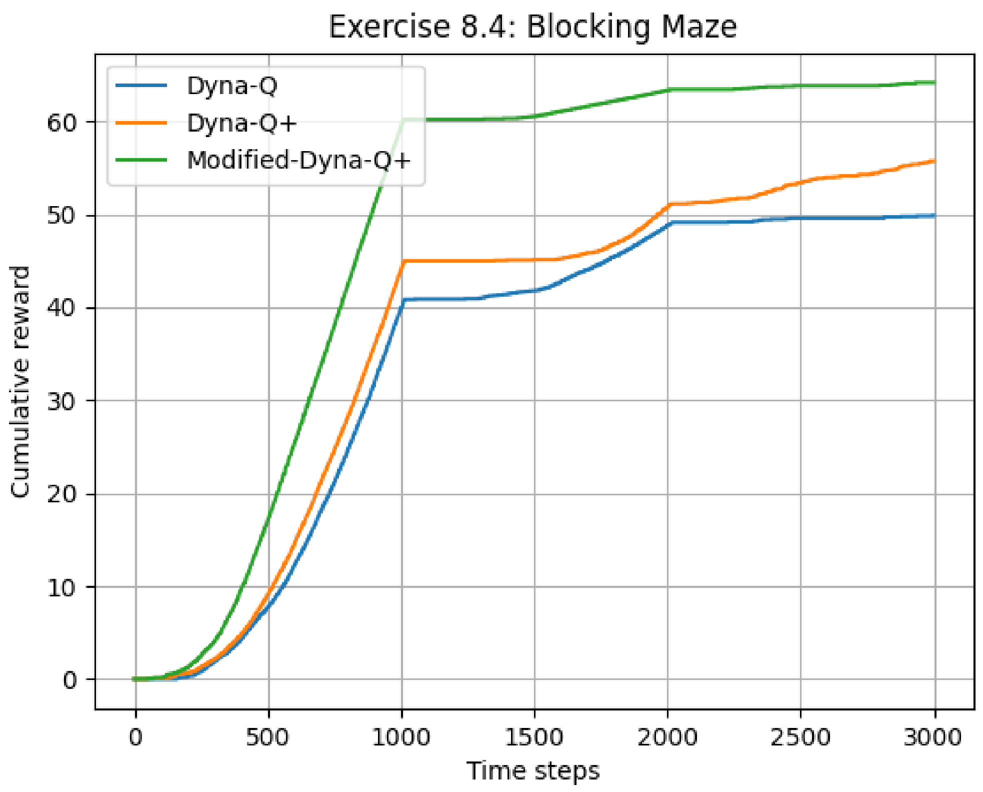

Figure 3: Cumulative reward for three model-based agents on the **Blocking Maze** task after switching to original configuration at 2000 timesteps, after the initial wall switch.

The experimental results reveal a marked difference in performance after 2000 timesteps, where the Dyna-Q+ algorithm accumulated a significantly greater cumulative reward compared to Dyna-Q and Modified-Dyna-Q+. This observation underscores the potential advantages of the Dyna-Q+ approach, particularly its mechanism for encouraging exploration, leading to improved performance.