Assignment 3: Model-based reinforcement learning

Simone de Vos Burchart (s1746995) & Hao Chen (s3990788)
Group Number: 80

1 Introduction

The RL methods are broadly classified into model-free and model-based approaches. Both methods rely on interaction with the environment to receive feedback (reward) and adjust it for future decisions. In the previous assignment, we investigated a few model-free methods, such as Q-learning and SARSA, which learn value functions or policies directly without constructing any model of the environment.

In contrast, model-based reinforcement learning aims to learn a predictive model of the environment's dynamics (the transition probabilities and reward functions). Thus the agent can perform planning by simulating state transitions and rewards internally, effectively augmenting real experience with synthetic data to accelerate policy learning.

In this report, we focus on two MBRL agents: the Dyna agent, which integrates real experience (using the strategy to explore the environment) with simulated planning updates using a learned model. And the Prioritized Sweeping (PS) agent, which extends Dyna Agent by prioritizing updates for the most impactful state-action pairs (the highest Q-value of the state-action pair). We evaluate and compare these agents under the same experimental setting, including identical planning budgets (1, 3 and 5) and environmental stochasticity levels (wind = 0.9 and 1.0). This allows for a fair comparison of their learning efficiency and performance characteristics.

1.1 Default parameters

Throughout the following sections, we will run experiments with various agents in the Windy Gridworld. Unless stated otherwise, they use the following default parameters: n_timesteps = 10001, eval_interval = 25, n_repetitions = 20, gamma = 1.0, learning_rate = 0.2, epsilon = 0.1, priority_cutoff = 0.01, smoothing_window = 5.

2 Dyna

The Dyna agent combines model-free Q-learning with model-based planning. This means that the model will first interact with the real environment to gather experience, and then the agent will use the experience to simulate updates (based on the parameter *n_planning_update*). In this way, Q-values are updated not only based on actual experience but also on simulated future outcomes, which extends the agent's ability to propagate reward information more efficiently throughout the state space.

2.1 Methodology

Since the Dyna agent is based on Q-learning with model-based planning, the agent uses an e-greedy policy to select actions.

$$\pi_\epsilon(a) = \begin{cases} 1-\epsilon, & \text{if } a = \text{argmax}_{b\in\mathcal{A}} Q(b) \\ \frac{\epsilon}{|\mathcal{A}|-1}, & \text{otherwise} \end{cases} \tag{1}$$

where $\mathcal{A}$ is the set of actions.

After the agent interacts with the real world (from the current state, take an action, get a reward, and go to the next state). The agent will store the transition (s, a, s') and the total received reward $R(s, a, s')$, which helps the agent to build an internal model of the environment. These are used to compute empirical estimates of the transition probability and expected reward:

$$\hat{P}(s'|s,a) = \frac{n(s,a,s')}{\sum_{s'} n(s,a,s')}, \quad \hat{R}(s,a,s') = \frac{R_{\text{sum}}(s,a,s')}{n(s,a,s')} \tag{2}$$

where $\hat{P}$(s'|s,a) is the estimated transition probability from the current state to the next state by taking action a. And n(s,a,s') is the number of times the agent has observed transitioning to s' after taking action a to the current state s. This $\sum_{s'} n(s,a,s')$ is the total number of times the agent has observed transitioning to a possible state after taking action a to the current state. This $\hat{R}(s,a,s')$ is the estimated reward received when transitioning from the current state to the next state by taking action a. This $R_{\text{sum}}(s,a,s')$ is the cumulative sum of rewards received for transitions from the current state to the next state via action a.

Once the agent gets the current state, reward, and next state, it will start updating the Q-value:

$$Q(S_t, A_t) \mathrel{+}= \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1,a}) - Q(S_t, A_t) \right] \tag{3}$$

where $Q(S_t, A_t)$ is the learned action-value function applied to state S_t and action A_t, R_{t+1} is the observed reward for taking action A_t in the state S_t, α is the learning rate, γ is the discount parameter, and $Q(S_{t+1}, a)$ is the Q-value for taking action a in the next state S_{t+1}. The key difference from model-free agents is that the Dyna agent performs planning. During the planning phase, the agent does not simulate from arbitrary (s, a) pairs. It will be based on the state that already visited and previous actions taken by this state. It randomly gets a state that has already been visited. And then it uses the estimated reward from equation **??** to update the Q-values (same formula as Q-learning 3). This process allows the agent to reinforce and propagate learned value information more efficiently across known parts of the environment.

2.2 Result

We evaluate the Dyna agent on the Windy Gridworld environment with the addition of *n_planning_updates* $\in \{1, 3, 5\}$ wind proportions of 0.9 (stochastic) and 1.0 (deterministic). The horizontal axis represents the timestep, and the vertical axis shows the episode return. Each configuration was repeated 20 times with 10001 timesteps per run. The agent was evaluated every 250 steps over 30 episodes using a greedy policy in a new Windy Gridworld environment (so that learning progress would not be compromised).

Figure 1 shows that under the stochastic environment, the Dyna agent outperforms the Q-learning baseline in all cases. The higher the number of planning increases, the faster the agent's performance improves. The agent with planning = 5 reaches an episode return of 50 at 3500 timesteps. At the same time, the agent with planning = 3 achieves only 25 episode returns, and the agent with planning = 1 has the same return as the Q-learning baseline, which is approximately -100. By combining with figure 2, we can observe the stochastic environment having a huge impact on the Dyna agent, especially for the final performance (after timestep 8500), where the randomness of the environment leads to the fact that the

Dyna agent does not end up performing as well as the Q-learning baseline.

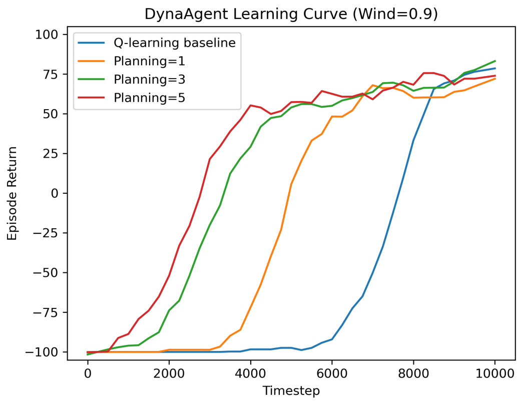


Figure 1: the learning curves of the Dyna agent under a stochastic environment with different number of planning times

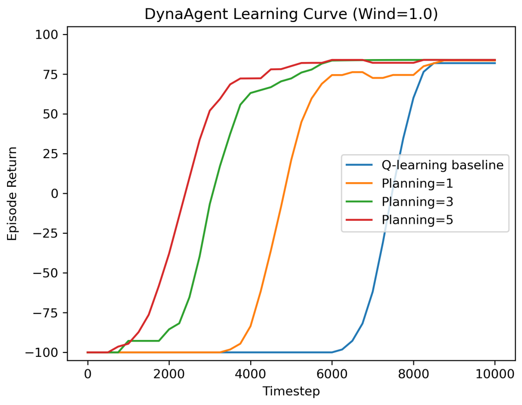


Figure 2: the learning curves of the Dyna agent under a deterministic environment with different number of planning times

To better quantify and compare the performance of different configurations, we computed the average episode return across the evaluation interval using a custom averaging function. The results are summarized in Table 1, as shown. In both stochastic and deterministic environments, the configuration with 5 planning updates consistently achieves the highest average return.

3 Prioritized Sweeping Agent

The Prioritized Sweeping (PS) agent is an enhanced version of the Dyna agent. The PS agent improves the efficiency of learning by prioritizing which state-action pairs to update. This means that it focuses more on the transitions that are expected to have a large impact on the Q-value. Instead of updating all past experiences equally, it uses a priority queue to focus computational resources on updates that are likely to lead to significant changes. This approach results in faster and more sample-efficient learning.

Planning Updates	**Wind**	
	0.9	**1.0**
Q-learning baseline	-53.29	-51.64
1	-14.90	-6.19
3	13.48	25.86
5	**22.83**	**38.47**

Table 1: Average return for the Dyna agent evaluations across planning updates and wind probabilities.

3.1 Methodology

Since the Prioritized Sweeping agent is built upon the Dyna agent, it shares many core strategies and mechanisms. Both agents use the e-greedy policy to select actions, where the agent explores with probability ϵ and exploits the current knowledge (chooses the action with the highest Q-values). They use the same way to compute the estimated transition probability $\hat{P}$(s'|s,a) and expected reward 2 based on experience (current state, action, and next state).

However, after obtaining a transition (s, a, r, s'), the PS agent does not immediately update the Q-value like the Dyna agent. Instead, it introduces the Prioritized Sweeping Extension, which will compute a priority value:

$$p = \left| r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right| \tag{4}$$

This priority value in (4) is actually the absolute value of the Temporal Difference Error (TDE), which measures the discrepancy between the current estimate and the new target value for Q(s,a). A large TDE implies that the transition (s, a, s') leads to a significant potential change in the value function. Therefore, we use self.queue to store transitions with significant TDE and filter out those with small values using a cutoff threshold (self.priority_cutoff = 0.01). The priority is stored as a negative number because Python's PriorityQueue pops the smallest item first, so by inserting (-priority, (s,a)), we ensure that the transition with the highest actual priority is retrieved first.

The core of the Prioritized Sweeping agent lies in how it performs planning updates based on the priority queue instead of randomly sampling a state-action pair and basing an estimated reward on that to update the Q-values. The PS agent selects the most informative (highest priority) state-action pair from the queue to update the Q-value (formula 3). After the Q-value update, the algorithm searches for all state-action pairs that could potentially lead to the updated state s. Each of these predecessor pairs is then evaluated using the same priority formula (Equation 4). If their priority exceeds the threshold, they are also added to the queue for future updates.

3.2 Result

We evaluate the performance of the Prioritized Sweeping agent under the same conditions as the experimental conditions of Dyna agent.

From figure 3, we can observe the Q-learning baseline achieves the highest return (approximately 55), but it learns more slowly in the beginning. Planning = 3 and planning = 5 both converge faster than Q-learning, but after reaching 52, planning = 3 plateaus, while planning = 5 performance is worse than Q-learning in the final. Planning = 1 shows the slowest learning and lowest return overall.

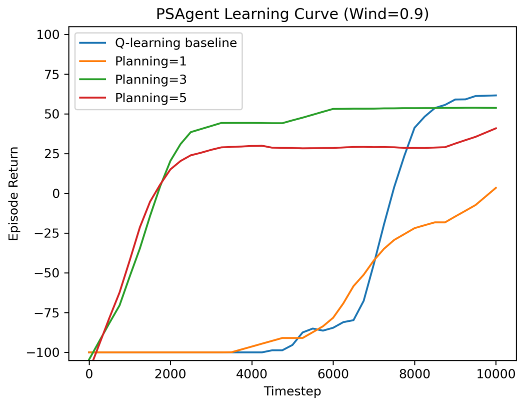


Figure 3: the learning curves of the PS agent under a stochastic environment with different number of planning times

For figure 4, we can observe the planning = 5 line learns the fastest, achieving max return (80) in less than 2000 timesteps. Planning = 3 also converges quickly, though slightly slower than planning 5. Q-learning baseline takes much longer to reach the same level of return (approximately 8000 timesteps). Planning = 1 performs the worst, but it starts improving earlier than Q-learning, its return plateaus around 70 and never reaches the maximum.

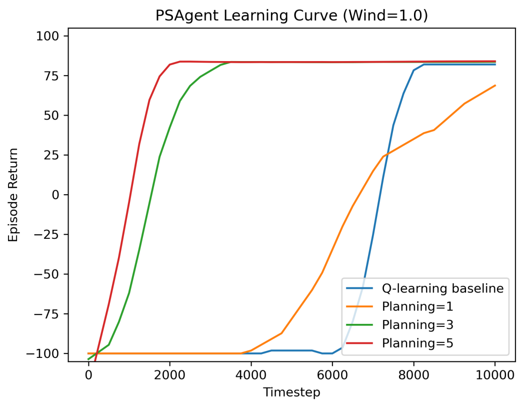


Figure 4: the learning curves of the PS agent under a deterministic environment with different number of planning times

Overall, in stochastic environments, higher planning steps help the agent adapt faster and converge earlier, but over-planning (e.g., Planning=5) may lead to premature convergence, suggesting a trade-off between speed and stability. In the deterministic environment, the higher the planning steps, the better the performance. From table 2, Planning = 3 offers a good balance between convergence speed and final return in most cases.

Planning Updates	Wind	
	0.9	**1.0**
Q-learning baseline	-53.64	-46.29
1	-69.00	-42.12
3	**28.14**	53.85
5	13.70	**63.93**

Table 2: Average return for the PS agent evaluations across planning updates and wind probabilities.

4 Comparison and conclusions

In this section we will compare the best agents from the the results of the previous sections, and the Q-learning baseline. We ran comparison experiments with the default parameters unless stated otherwise.

For the **deterministic** environment (wind=1.0), the best Dyna agent and the best PS agent both had 5 planning updates. Figure 5 shows the results of this experiment. This figure used no smoothing, to prevent the curves from starting below -100. We can see in the graph that the PS agent reached the limit of the episode returns the quickest at around 1500 timesteps. The Dyna agent learned a little slower, reaching the limit after around 3000 timesteps. Q-learning required about 8000 timesteps, and its return was still slightly lower until the end. We can conclude from this experiment that more planning updates certainly help to speed up learning and that Prioritized Sweeping speeds up learning slightly more compared to Dyna.

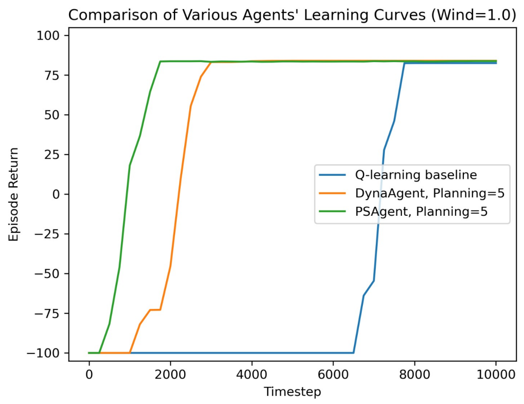


Figure 5: Comparison of learning curves for three agents under deterministic environment with Q-learning baseline, Dyna agent with planning steps = 5, and PS agent with planning steps = 5 (without smoothing)

For the **stochastic** environment (wind=0.9) the best Dyna agent had 5 planning updates and the best PS agent had 3 planning updates. Figure 6 shows the results of this experiment. Here we again

Agent	Average Return
Q-learning baseline	-46.81
Dyna (planning=5)	43.86
Prioritized Sweeping (planning=5)	64.25

Table 3: Average return over all evaluations for each agent configuration with wind=1.0.

use a smoothing window of 5, which makes it appear like the agents have a return below -100 at the start, but this is not the case. The Q-learning agent's performance is quite similar to the deterministic environment in terms of learning speed, although the returns in the end are a little lower (as expected). The PS agent learned more slowly, and after about 2500 timesteps learning slowed even more. It continued to improve but not as much as the other agents. The Dyna agent learned slightly slower than the PS agent at the start, but towards the end reached a higher return. Based on the average over all evaluation steps shown in 4, the PS agent is the best, but perhaps with a bigger range of timesteps, the Dyna agent might be better.

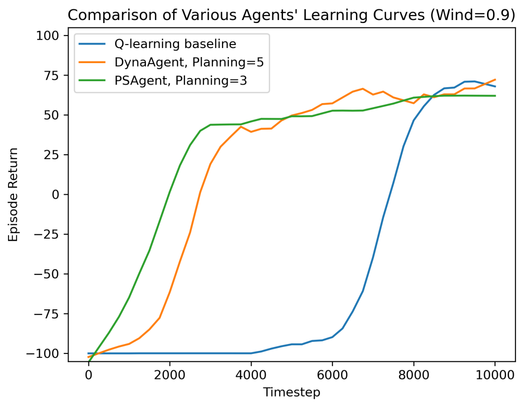


Figure 6: Comparison of learning curves for three agents under stochastic environment with Q-learning baseline, Dyna agent with planning steps = 5, and PS agent with planning steps = 5 (with smoothing = 5)

Agent	Average Return
Q-learning baseline	-51.44
Dyna (planning=5)	17.49
Prioritized Sweeping (planning=3)	27.39

Table 4: Average return over all evaluations for each agent configuration with wind=0.9.

4.1 Runtime

The runtime of a single experiment varies across agents, number of planning updates and wind proportions. Table 5 shows the number of seconds to run a single experiment with the default parameters as defined in Section 1.1, except for n_repetitions = 1.

We can see that Q-learning always has the shortest runtime, and more generally, that fewer planning updates is always faster than more planning updates for a given agent and wind proportion. For Dyna a one update increase adds a little more than 1s to a single experiment. For PS this increase is much smaller and does not appear to be as linear.

Planning updates	Wind=0.9		Wind=1.0	
	Dyna	PS	Dyna	PS
0	**1.15**	–	**1.02**	–
1	2.06	1.66	2.09	1.19
3	4.41	**2.39**	4.21	1.63
5	**6.66**	2.36	**6.60**	**2.24**

Table 5: Timing results (in seconds) for single experiments (n_repetitions = 1) of different agent configurations grouped by planning updates and wind condition. Dashes indicate no data. Dyna with 0 planning updates is the Q-learning baseline. Bold values highlight the agents compared in the previous section.

Figure 5 showed the comparison of the best agent configuration in the deterministic (wind=1.0) environment. The PS agent (planning updates = 5) reached the limit of the returns well before the Dyna agent (planning updates = 5), and as its runtime is about 3 times shorter, this is definitely the superior agent of the two, in this setting. The Q-learning agent is more than twice as fast as the PS agent, but it takes about four times as long to reach a good return, so again ultimately the PS agent is the quickest to learn.

It's similar for the stochastic environment (wind=0.9). Figure 6 showed the comparison between Q-learning, Dyna (planning updates = 5) and PS (planning updates = 3). PS had slightly better returns than Dyna, and being almost 3 times as fast is the best choice out of the two. Q-learning's runtime was again the fastest but not enough to compensate for the much delayed increase in returns.

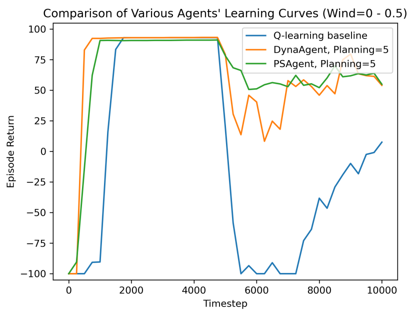
5 Reflection

Model-based reinforcement learning methods like Dyna and Prioritized Sweeping improve sample efficiency by using a learned model for planning. This allows agents to learn faster after they have an interaction with the environment. However, they rely on the accuracy of the learned model thus, in stochastic environments, over-planning can lead to unstable or suboptimal result.

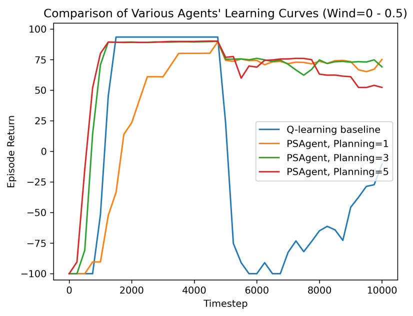
Compared to Dyna, PS focuses updates on transitions with large temporal difference errors, which helps the agent learn more quickly. It also requires less runtime than Dyna for the same number of planning steps. And in both environment (deterministic and stochastic environment), PS showed better performance.

Initializing Q(s,a) = 0 while the environment has a negative default reward may reduce the incentive for exploration, since all unseen actions appear equally unattractive, leading the agent to exploit suboptimal known path prematurely, see 3.

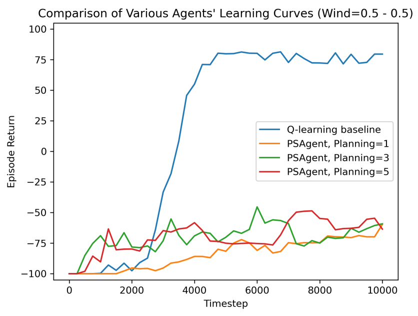
In the future, we can try to combine PS agent with model-free methods (such as SARSA), which may improve robustness. This hybrid approach could help the PS agent avoid instability or suboptimal performance in a stochastic environment. Our appendix also shows the possibility of training agents in deterministic setting before exposing them to stochastic elements.



(a)



(b)



(c)

Figure 7: Comparisons of various agents in various (changing) wind conditions. (a) and (b) show results from environments where the initial wind proportion was 0.0, and after 5000 timesteps changed to 0.5. (c) shows results where the wind proportion was always 0.5. No smoothing was applied.

6 Appendix - Investigating Adaptability

For real-world applications, it is a valuable quality for agents to be adaptable to changes in its environment. Therefore, we set up an extra experiment where we can look into how well our agents adapt to changes in the wind proportion. This time we have a starting wind proportion, which lasts for the first 5000 timesteps, and then we change the wind to a second wind proportion, which lasts until the end. We also did not apply smoothing to these graphs.

First we compared the best agent configurations from the deterministic environment. We start with wind proportion 1.0, and end with 0.5. Figure 8 shows the average returns of the three agents.

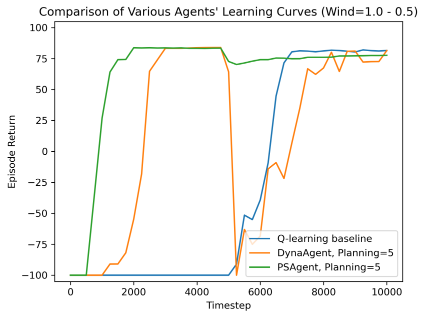


Figure 8: Returns of various agents with wind proportion decreasing from 1.0 to 0.5 halfway

The results are fascinating. The PS agent with 5 planning updates is only minimally affected by the change in environment, while the Dyna agent with 5 planning updates looks like all its progress is undone. The Q-learning baseline also seems mostly unaffected. This result really shows the value of PS beyond just the efficiency as pointed out in Section 4.

Next we compared PS agents with different numbers of planning updates. Figure 9 shows the differences in their episode returns.

These results are quite similar to the results in Section 3, with only a minor adjustment as the environment changes the wind. Again we see that the PS agent with 3 or 5 planning updates are quite similar, and far superior to 1 planning update.

Finally we repeated the two experiments above for a different change in the wind: from 0.0 at the start, to 0.5 at the end.

Figure 7a shows that once again, the Dyna agent's episode return is more affected by the change in wind than the PS agent. However,

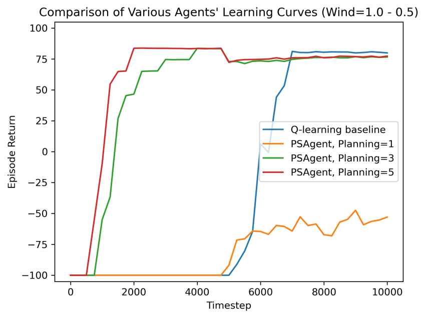


Figure 9: Returns of various agents with wind proportion decreasing from 1.0 to 0.5 halfway

the effect is much smaller than in the previous experiment. Both agents show some limited learning again after the wind changed. We also see that the Q-learning agent learns much faster in the no-wind environment than in previous experiments, which we attribute to the smaller number of steps needed to reach the goal.

In Figure 7b we see that the PS agent with 3 or 5 planning updates are again quite similar, but in this environment the PS agent with 1 planning update performed very well too, much better than the previous environment.

The observation that intrigues us the most is the difference in the curve of the PS agent with 1 planning update across the two experiments. In both wind change experiments, the wind proportion was 0.5 in the second half of the timesteps. However, the returns in the second halves were very different. It prompted us to run another experiment where the wind proportion was always 0.5, see 7. In this environment, all PS agents had very low returns, and slow learning. This has interesting implications. It suggests that pretraining in a deterministic environment (wind = 0.0, and 1.0 in the case of 3 and 5 planning updates) before transitioning to a stochastic one (wind = 0.5) enables the agent to build a more reliable and complete model of the environment. This model can then be leveraged through planning when stochasticity is introduced. In contrast, when the agent starts learning directly in the stochastic setting, it seems to struggle to form a consistent model early on, decreasing the impact of planning later. This highlights the value of early structure and consistency for model-based agents, where a strong initial model can provide lasting benefits even when the environment becomes stochastic.