Reinforcement Learning

Asal Asgari
UiT the Arctic University of Norway
Narvik, Norway
aas047@uit.no

***Abstract*—Reinforcement Learning (RL) is a framework for defining and solving a problem where an agent learns to take actions that maximize reward. The agent determines the appropriate action by observing the state of the world. A reward is given to the agent when it executes an action, and it uses this information to improve its future actions. In this study we compare two model-free and model-based RL methods for a grid world problem. The mentioned methods are Q-learning and Dyna-Q, we compare the performance of each method with a series of graphs through a number of expriments and episodes.**

***Index Terms*—Reinforcement Learning; Dyna-Q Learning; Q-Learning**

I. Introduction

Reinforcement Learning [1] is a type of machine learning technique in which a learning agent learns, over time, to behave optimally in a certain environment by interacting continuously in the environment. The agent during its course of learning experiences various different situations in the environment it is in. These are called states. The agent while being in that state may choose from a set of allowable actions which may fetch different rewards(or penalties). The learning agent over time learns to maximize these rewards so as to behave optimally at any given state it is in [1].

There are four essential elements of a reinforcement learning model besides the agent and the environment [1]: a policy, a reward, a value function, and a model of the environment.

A policy determines how an agent behaves at a specific point in time. A mapping between environmental conditions and actions is a process of linking environmental conditions to the activities performed in the environment by agents.

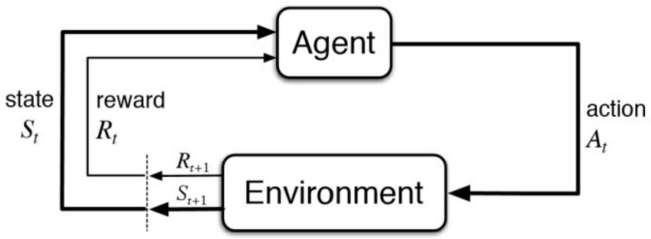

Fig. 1. Understanding the basics of Reinforcement Learning [1]

At each time step, the agent's behaviors result in a reward. Ultimately, the agent seeks to maximize its reward. Consequently, the reward differentiates between positive and negative outcomes for the agent.

A state's value is the total accumulated quantity of prizes that the agent may anticipate receiving in the future if it begins in that condition. A state's values reflect the expected future benefits it will generate and the long-term attractiveness of the state.

The environment model is another significant component of several reinforcement learning systems. In this mechanism, environmental behaviors are mimicked and predictions can be made about how those behaviors will be interpreted by the environment. By using this model, agents will be able to forecast the next reward if they take action. This will enable them to plan their actions based on the expected results of future environmental changes.

The question of whether the agent has access to a model of the environment is a critical branching point in an RL algorithm. The main upside to having a model is that it allows the agent to plan by think-

ing ahead. Using this method, the agent can explore possible options and make a choice between them. As a result of planning ahead, agents can formulate a learning policy. One downside is that agents rarely have access to ground-truth models of their environment. In this case, an agent must learn the model purely from experience, which presents several challenges. The biggest challenge is that agents can exploit bias in the model, resulting in agents that perform well when compared with learned models but behave poorly in reality. Algorithms that use a model are called model-based methods, and those that don't are called model-free methods.

In this report we compare a model based(Dyna-Q) and a model free(Q-learning) algorithm in the form of plots and graphs.

A. Q Learning

Q-Learning [1] is an off-policy value-based method that uses a Temporal-Difference approach to train its action-value function. Instead of modeling the whole environment, these algorithms learn directly from the environment's dynamics. Furthermore, they update their estimates based on previous estimates, so they don't wait for the final results. Temporal-Difference Learning is represented in the following formula: where α is a learning rate. This

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

means that this approach will wait until next time step $t+1$ and use reward and estimated value from that time step to update the value of the time step t. This is the difference between the actual reward $(r+\gamma * V(s'))$ and the expected reward $V(s)$ multiplied by the learning rate α.

As we take the difference between the actual and predicted values, this is the equivalent of an error. As the TD error depends on the next reward and state, it is not available until one timestep later. As we iterate, we will try to minimize this error. Temporal-Difference is performed like this [1]:

1) Initialize value for each state from the set of states S arbitrary: $V(s) = n, \forall s \in s$.
2) Pick the action a, from the set of actions defined for that state $A(s)$ defined by the policy π.
3) Perform action a
4) Observe reward R and the next state s'
5) Update value for the state using the formula: $V(s) \longleftarrow V(s) + \alpha(R + \gamma V(s')V(s))$
6) Repeat steps 2-5 for each time step until the terminal state is reached
7) Repeat steps 2-6 for each episode

As you could see Temporal-Difference Learning is based on estimated values based on other estimations. Q-Learning takes it one step further. It is estimating the aforementioned value of taking action a in state s under the policy. Basically, decisions in this approach are based on estimations of state-action pairs, not state-value pairs. The simplest form of it is called one-step Q-Learning and it is defined by the formula [1]:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

A Q-Value can be viewed as the quality of an action taken from a particular state-action combination. However, the policy only determines which state-action pairs are visited and updated. In Q-learning, the optimal action value is directly estimated based on the max operator used to select the optimal Q-value in the TD target, which is the cumulative future reward the agent would receive if it behaved optimally regardless of the policy it currently follows. This is why Q-learning is sometimes referred to as off-policy TD learning. Each of these Q-Values is stored within a matrix, which consists of rows for states and columns for actions [1].

In general, the Q-Values tend to converge after a sufficiently long time and enough random exploration of actions, serving as an action-value function for our agent. It is important to note that sometimes we add additional constraints to prevent overfitting [1].

The above equation is similar to the TD prediction update rule with a subtle difference. Below are the steps involved in Q-learning:

1) First, initialize the Q function to some arbitrary value

2) Take an action from a state using epsilon-greedy policy () and move it to the new state
3) Update the Q value of a previous state by following the update rule
4) Repeat steps 2 and 3 till we reach the terminal state

We use the Epsilon Greedy policy to define whether we should try new actions and perhaps come up with a better solution, or use the path that we have already learned. This parameter defines the relationship between exploration of new options and exploiting already learned options.

B. Dyna-Q Learning

By a model of the environment we mean anything that an agent can use to predict how the environment will respond to its actions. Given a state and an action, a model produces a prediction of the resultant next state and next reward. If the model is stochastic, then there are several possible next states and next rewards, each with some probability of occurring. Some models produce a description of all possibilities and their probabilities; these we call distribution models. Other models produce just one of the possibilities, sampled according to the probabilities; these we call sample models [1].

Given a starting state and a policy, a sample model could produce an entire episode, and a distribution model could generate all possible episodes and their probabilities. In either case, we say the model is used to simulate the environment and produce simulated experience.

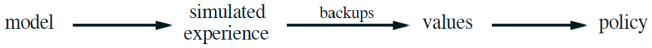

Fig. 2. Models and Planning [1]

Dyna-Q is a simple architecture integrating the major functions needed in an online planning agent. Within a planning agent, there are at least two roles for real experience: it can be used to improve the model (to make it more accurately match the real environment). This is called model-learning. It can be used to directly improve the value function and policy using different kinds of reinforcement learning methods. This is called direct reinforcement learning.

Dyna-Q algorithm integrates both direct RL and model learning, where planning is one-step tabular Q-planning, and learning is one-step tabular Q-learning.

Dyna-Q includes all of the processes shown in Fig 3. planning, acting, model-learning, and direct RL, All occurring continually. The planning method is the random-sample one-step tabular Q-planning method given in Figure 4. The direct RL method is one-step tabular Q-learning in Figure 5. The model-learning method is also table-based and assumes the world is deterministic.

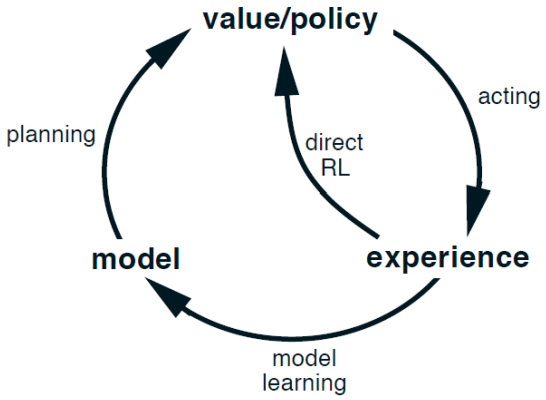

Fig. 3. Relationships among learning, planning, and acting [1]

Do forever:
1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(s)$, at random
2. Send S, A to a sample model, and obtain
   a sample next reward, R, and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S':
   $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Fig. 4. Random-sample one-step tabular Q-planning [1]

In Fig.5 Model(s; a) denotes the contents of the model(predicted next state and reward) for state-action pair (s; a). If (e) and (f) were omitted in this algorithm, the remaining algorithm would be one-step tabular Q-learning.

II. Related Work

Path finding on a static grid is well-studied and well-known in the AI, planning, and robotics communities [2] with many methods and algorithms

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \epsilon$-greedy(S,Q)
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Fig. 5. Dyna-Q Algorithm [1]

proposed so far. Previous studies primarily focused on the heuristic search in the state space induced by the grid cells and are based on the well-known A* algorithm proposed in 1968 [3]. Among the most common algorithms that solve the task online, e.g. without any preprocessing, one can name IDA* [4], ARA [5], JPS [6], Theta* [7] etc., all of which are apparently heuristic search algorithms.

In one study [8], they focused on the best path selection approach based on a hybrid improved A* algorithm and reinforcement learning for intelligent driving vehicles under the conditions of limits including height, width and weight, accidents, and congestions and they proved that the HARL algorithm has advantages in path length, path quality, and algorithm efficiency.

Another study [9] compares direct reinforcement learning (no explicit model) and model-based reinforcement learning on a simple task. They prove that model-based reinforcement learning is more data efficient than direct reinforcement learning, model-based reinforcement learning finds better trajectories, plans, and policies, and model-based reinforcement learning handles changing goals more efficiently.

Our study compares the performance of model-free and model-based learning for a path-finding task in a grid world. Similar to [9] we found that model-based learning performed more efficiently than the model-free algorithm.

III. Method

A. tools

For this project, Python 3.7 was selected as the programming language. A number of software libraries were used in implementing the algorithms, including NumPy for handling data and Matplotlib for rendering graphs and plots.

B. Problem

The problem we are solving for this task is a grid world of 10x10 containing a set of walls, traps and a goal state. Whereas moving between normal cells has a cost of -5 and going into traps has a cost of -500,-100. We cannot iterate through the walls and our goal state contain a reward of 1000. It is possible to move in all four directions, but not beyond the limits of the grid world. Our ultimate goal is to find the shortest path with the maximum accumilated reward from an initial state to the goal state, while comparing the performance of model-free(Q-learning) and the model-base(Dyna-Q) methods. We have applied epsilon-greedy policy to generate values and demonstrate policy iteration to find the optimal (executable) policy for the agent. The agent can go forward, backward, left, right with an equal probability (p=0,25). When the agent hits the wall it stays in its original state until next action.

In addition we are going to test and compare these two algorithms on a bounty hunter problem where The big reward is associated with the agent for catching a bank robber. The robber has a mixed strategy of moving between two cells and in other word we are training our agent on an environment where the goal changes with a probability of 35%. For comparison we have recorded and mentioned

	1	2	3	4	5	6	7	8	9	10
A									-500	
B										
C										
D										
E										
F										+1000
G										
H										
J							-100			
I										

Fig. 6. Gridworld

relevant statistics in the result section.

C. Dyna-Q

I will explain each function from the assignment independently and will explain how they relate to each other.

1) Class Grid: The grid class builds the environment described in the problem and calculates the reward and the next state when an action has been taken while in a particular state. This is defined by a series of conditional statements. Our grid has 100 cells and 4 possible actions to choose from. Whenever a given action leads us to a wall instead of updating the state and reward we keep our next state as our previous state. This means we stop and wait for the next action to define our state.

2) def epsilon_greedy: To be able to both explore and exploit we are using an epsilon greedy approach to choose our next action from the set of possible actions. we generate a random value if this value was bigger than our epsilon(which is defined as 0.05 in our case) we choose the best action possible, else we choose to take a random action.

3) def q_learning: As shown in formula for Q-learning in introduction, we read the previous state-action value and we update our state-action value table according to the formula and our parameters. where $Q(s, a)$ is the value function for action a at state s, α is the learning rate, r is the reward, and γ is the temporal discount rate.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

4) def dynamic_q_planner: Dyna-Q agent is implemented by adding a planning routine to a Q-learning agent: Once the agent has learned from observed experiences in the real world, it is allowed to perform k planning steps. As the model proceeds through these k steps, it generates simulated experiences by random sampling from previous state-action pairs. The agent then learns from this simulated experience, again using the same Q-learning rule that we implemented in the q_learning function. Therefore, Dyna-Q agents (via Q-learning) learn by acting from real experience, and planning from simulated experience.

A major advantage of Dyna-Q is that planning occurs simultaneously with the agent's interaction with the environment. This means that new information gained from the interaction can influence the model and for that matter affect planning.

We will sample a random state-action pair from those we've experienced (called visited in code), use our model to simulate the experience of taking that action in that state, and update our value function using Q-learning with these simulated state, action, reward, and next state outcomes. Furthermore, we want to run this planning step k times, which can be obtained from tuning[3].

5) def learn_environment: This function is where we call every other function written for the algorithm to run in order and where we initialize our primary values. We run our algorithm for a number of episodes for the purpose of collecting data and training. For each episode we have a maximum number of steps we can take, so that each episode would have a definite ending point.

D. Q-Learning

The Q-learner algorithm uses all the same functions as Dyna-Q with a single difference that we do not run the dynamic_q_planner function with the model so we only use the q-learner function for updating our state-action value matrix. This algorithm can be run simply by setting tuning[3] to zero.

E. Bounty Hunter with moving robber

For this section we use both Dyna-Q and Q-learning algorithm for comparing purposes to see if we have a changing position in our goal which method performs more efficiently and reach the goal in a smaller amount of steps. The only difference in code for this part is that we change the class grid structure to fit the moving goal, as

well as adding an extra $wait$ action to choose from for our agent.

IV. Result

I have divided the result into two segments. First part is comparing model-based and model free learning question a and b according to the assignment. The second section mentions results for for question d.

A. Dyna-Q vs Q-Learning

Figure 7 illustrate the difference between Q-learning(0 planning step) and Dyna-Q learning with 1,10, and 100 as planning steps. From this figure we can see that Q-learning starts with a higher number of steps(Around 60) during the first episodes and then it starts getting going down from episode 40. which means from episode 40 Q-learning alogrithm needed fewer steps to be able to reach the goal. However even having 1 planing step(Dyna-Q) makes a huge difference in number of steps that the agent needs to take from the very beginning(around 25 steps to goal) and with 10 and 100 level of planning we can see that the number of steps needed is even smaller and down to approximatly 20 steps during the whole experiment.

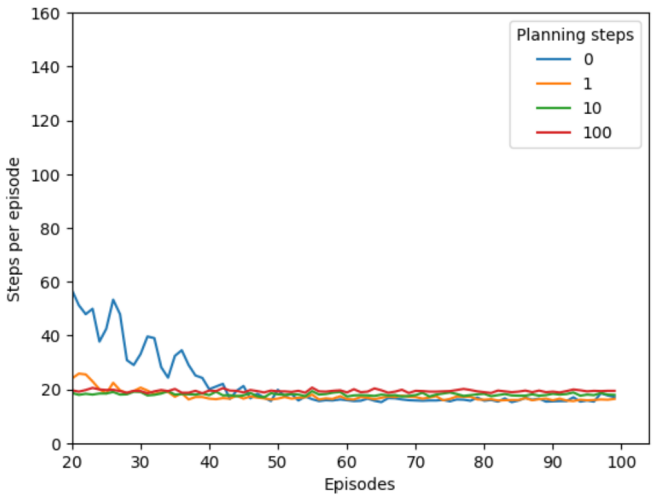

Fig. 7. Steps Taken per episode with different planning steps

we ran both algorithms for 500 episodes with maximum 1000 steps per episodes, with 10 planning steps for Dyna-Q and compared the results through figures 8 to 15.

Figure 8. and 9. shows the amount of reward gained through episodes in Dyna-Q and Q-learning. As we can see from these two figures Dyna-Q algorithm starts getting convergent to a high reward earlier on (from episode 20) comparing to Q-learning where starts converging from episode 80. Meaning with Dyna-Q we can reach our optimum result sooner that Q-learning.

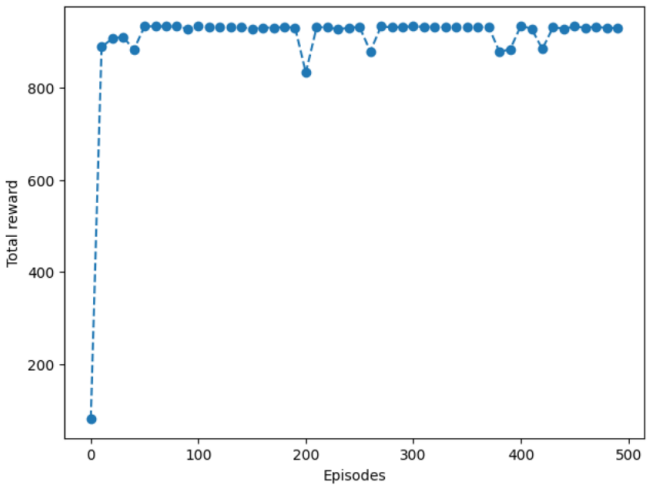

Fig. 8. Total amount of reward gained through each episode, Dyna-Q(model-based)

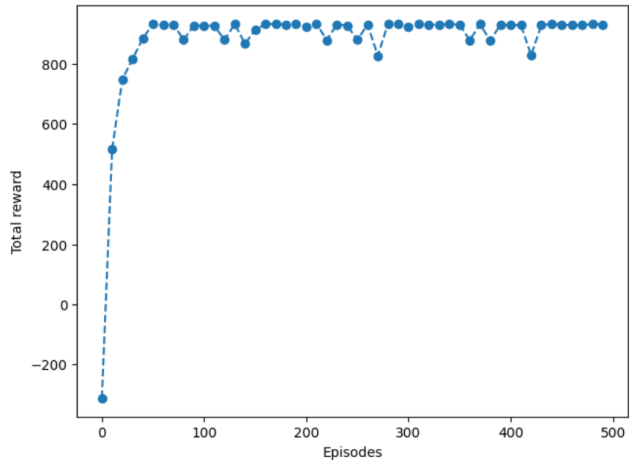

Fig. 9. Total amount of reward gained through each episode, Q-learning(model-free)

Comparing figure 10 and 11 shows us that with Dyna-Q the value function(maximum value per state) has a more precise definition for each cells.

Cells that are closer to the goal have a relatively higher value than the cells further away. Moreover, the cell that are close to traps or walls either have a value of zero or small value assigned to them meaning taking them is not suggested through the model. Whereas in the Q-learning the spectrum of distance and value are missing and most cells' values are defined close to zero.

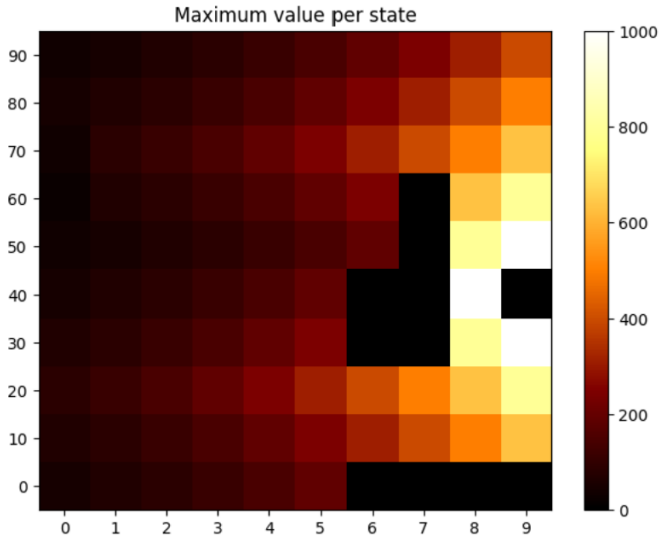

Fig. 10. Maximum value per state, Dyna-Q(model-based)

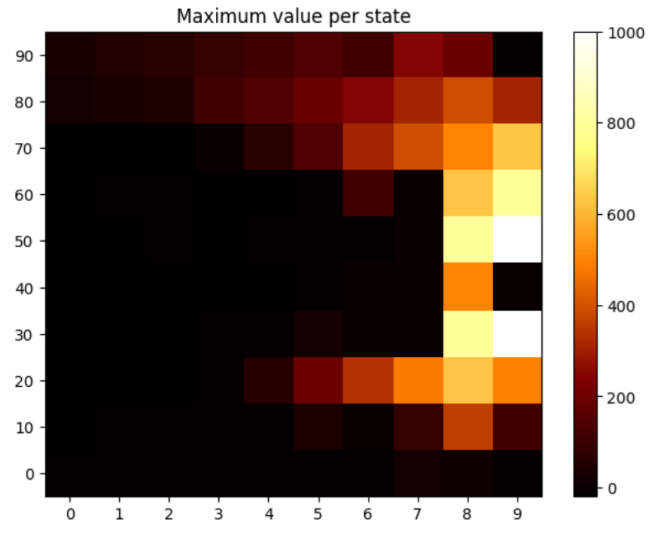

Fig. 11. Maximum value per state, Q-learning(model-free)

Figure 12 and 13 shows the maximum value/probability action in each states, comparing these two figures we underestand that some of the maximum actions does not make sense in the Q-learning method and overall the Dyna-Q algorithm has a more logical representation in regard of our world. For intstance in figure 13 cell 99 maximum action is going right however going right would be a boundry issue and since the goal is a few cell right bellow this cell the logical call would be going down where it is as well suggested in the figure 12.

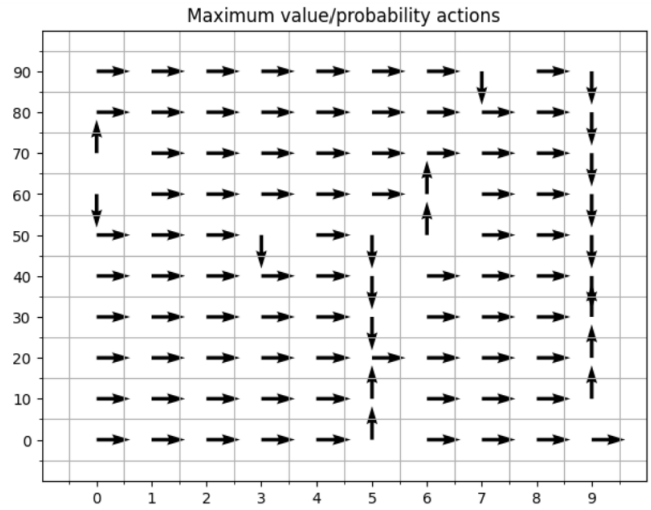

Fig. 12. Maximum value/probability actions per state, Dyna-Q(model-based)

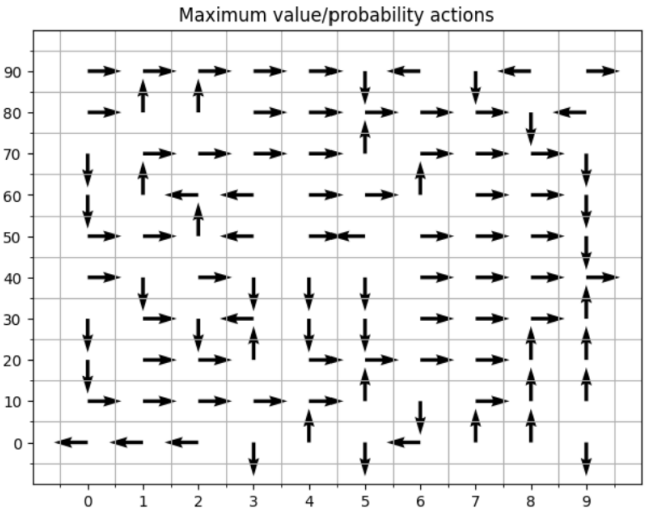

Fig. 13. Maximum value/probability actions per state, Q-learning(model-free)

B. Bounty Hunter

I have run the bounty hunter problem both in Q-learning and Dyna-Q algorithm. according to Figure 14, in the Dyna-Q algorithm, we start with a smaller amount of steps for reaching the goal in early episodes. However, we can see that in this algorithm we have more peaks and fluctuations where the agent struggles to find the goal compared to the Q-Learning algorithm at that very exact moment. Q-learning started off with needing more steps to find the goal in from episode 0 to 5 but after that on average Q-learning founded the goal more quickly than the Dyna-Q algorithm. For example around episode 20 Dyna-Q algorithm needed 1000 steps to find the goal whereas throughout the whole program Q-learning algorithm found the goal with an average of 15 steps. Therefore, in this case, the Q-learning algorithm acted much more efficiently compared to Dyna-Q and is the recommended algorithm for when we are dealing with an unstable goal state.

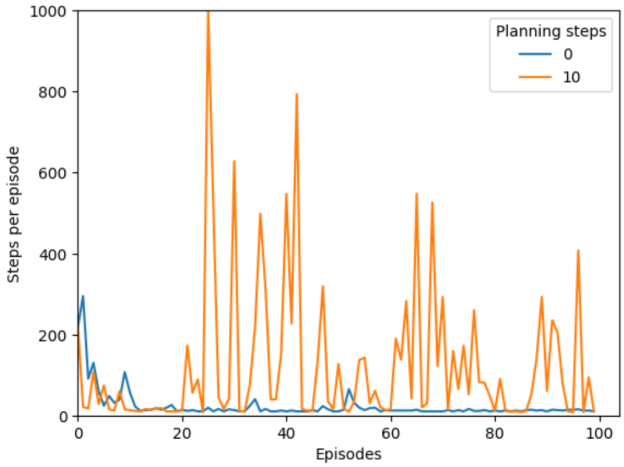

Fig. 14. Steps Taken per episode with different planning steps

V. Discussion

The purpose of this assignment was to develop and evaluate two RL methods for developing an agent to determine the most efficient path with the highest reward to the goal while preferably avoiding traps and walls in a grid world of size 10x10. As depicted in Figs 7-13 in the results section, the model-based technique is more efficient at finding convergence and finding the most suitable model in a shorter time compared to the model-free version where the goal state is constant. Dyna-Q's algorithm presented better metrics values and was more accurate. This means that under normal conditions this model-based algorithm is superior to Q-learning.

In contrast we saw that in situation where our goal is changing it's place with a predefined probability, Q-learning has an easier time to adapt to the situation and find the goal with the smaller amount of steps needed.

For future work comparing the deep Q-learning algorithm with these two methods could give us a new perspective and change our preferred solution for these types of problems.

VI. Conclusion

In this paper, we examined two well-known reinforcement learning methods, Dyna-Q and Q-Learning, for training an agent to find the best path to the goal state while optimizing the reward. The algorithms have been implemented in Python and their only difference is an extra planning step based on the model of the environment and applied to the Q-table. You may switch the algorithm from Model-based to Model-free by setting tuning[3] to zero.

The Dyna-Q approach has more precision and speed in finding the goal and converging to a minimum number of steps required in each episode than the other approach, so it is recommended for grid world problems where the goal is stationary, while Q-learning was more adaptable to problems with moving goals or not changing environments.

References

[1] Richard S. Sutton and Andrew G. Barto,"Reinforcement Learning: An Introduction"

[2] P. Yap. Grid-based path-finding. In Proceedings of 15th Conference of the Canadian Society for Computational Studies of Intelligence, pages 44–55, 2002.

[3] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics, 4(2):100–107, 1968

[4] R. E. Korf. Linear-space best-first search. Artificial Intelligence, 62(1):41–78, 1993.

[5] M. Likhachev, G. J. Gordon, and S. Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. Advances in Neural Information Processing Systems, 2003

[6] D. Harabor and A. Grastien. An optimal any-angle pathfinding algorithm. In ICAPS 2013, 2013

[7] A. Nash, K. Daniel, S. Koenig, and A. Felner. Theta*: Any-angle path planning on grids. In Proceedings of the National Conference on Artificial Intelligence, page 1177, 200

[8] Xiaohuan Liu, Degan Zhang, Ting Zhang, Yuya Cui, Lu Chen Si Liu; Novel best path selection approach based on hybrid improved A* algorithm and reinforcement learning, Applied Intelligence volume 51, pages9015–9029 (2021)

[9] Christopher G. Atkeson and Juan Carlos Santamaria; A Comparison of Direct and Model-Based Reinforcement Learning;College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280