Planning and Learning with Tabular Methods

Bolivar Enrique Solarte Pardo

ID: 106061858

May 1, 2019

1. Introduction

The present report refers to homework 3 for EE5510 System Theory course, 2019 spring semester. The objective of this assignment is to implement the Tabular Dyna-Q method. In the following sections, detail about their implementation, python code, and analysis over their performance is presented.

2. Implementation details

This implementation considers the following premises:

1. The size, obstacle states, starting state, goal state and actions for the maze environment is described in figure 1.
2. For the present model, 10 runs with 50 episodes are set for each test. Furthermore, each test consider a different number of planning states over the learning model, i.e [0, 5, 50]
3. For this assignment we defined two models, the first one is the maze environment introduced in figure 1. The second one is an internal model for planning states. This latest is the only model allowed for modification.

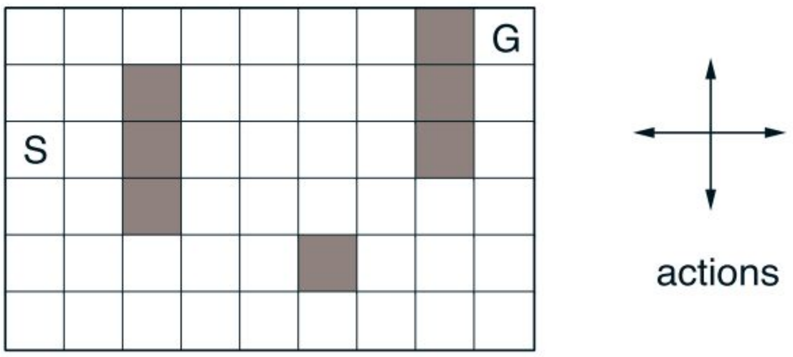

Figure 1.Simple maze environment from Figure 8.2 ***[Sutton et. al. 2018]***

2.1 Dyna Q-Learning Implementation

The Dyana Q-learning implementation is based on equation (1), which indeed is the updating rule for the Q-learning model.

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

The Dyna Q details as same as its implementation are described as following:

1. Based on the current state, an action is taken by using a ε-greedy policy. ***(line 8)***

2. The next state as same as a reward are evaluated from the maze environment ***(line 11)***
3. The updating rule for the state-action value ***Q*** is evaluated at line 14 to 16 based on the current state, selected action, next state and reward
4. The current state, selected action, next state, and reward are stored in the internal model, which is later used for planning. ***(line 19)***
5. A planning loop is set based on a defined number of planning states for each test which was previously described in section 2. This loop considers (1) Random selection of the stored values from the internal model ***(line 23)***; (2) Updating Q-learning rule using the internal model ***(line 26-28).*** It's important to note, in this loop the states, actions, and reward stored in the internal model are used to update the state-action value ***Q*** previous described.
6. The number of step until reaching the goal is counted in order to evaluate how the planning states using the internal model affects the learning behavior.

```
1 def dyna_q(args, q_value, model, maze):
2     s_0 = maze.START_STATE
3     steps = 0
4
5     while s_0 not in maze.GOAL_STATES:
6
7         # get action
8         a = choose_action(s_0, q_value, maze, args.epislon)
9
10        # take action
11        s_1, reward = maze.step(s_0, a)
12
13        # Q-Learning update rule
14        target = reward + args.gamma * np.max(q_value[s_1[0], s_1[1], :])
15        error = target - q_value[s_0[0], s_0[1], a]
16        q_value[s_0[0], s_0[1], a] = q_value[s_0[0], s_0[1], a] + args.alpha * error
17
18        # feed the internal model
19        model.store(s_0, a, s_1, reward)
20
21        # Planning from the internal model
22        for t in range(0, args.plan_step):
23            s_0i, a_i, s_1i, reward_i = model.sample()
24
25            # Q-Learning update rule using internal Model
26            target = reward_i + args.gamma * np.max(q_value[s_1i[0], s_1i[1], :])
27            error = target - q_value[s_0i[0], s_0i[1], a_i]
28            q_value[s_0i[0], s_0i[1], a_i] = q_value[s_0i[0], s_0i[1], a_i] + args.
    alpha * error
29
30        s_0 = s_1
31        steps += 1
32
33    return steps
```

3. Experiments and Analysis

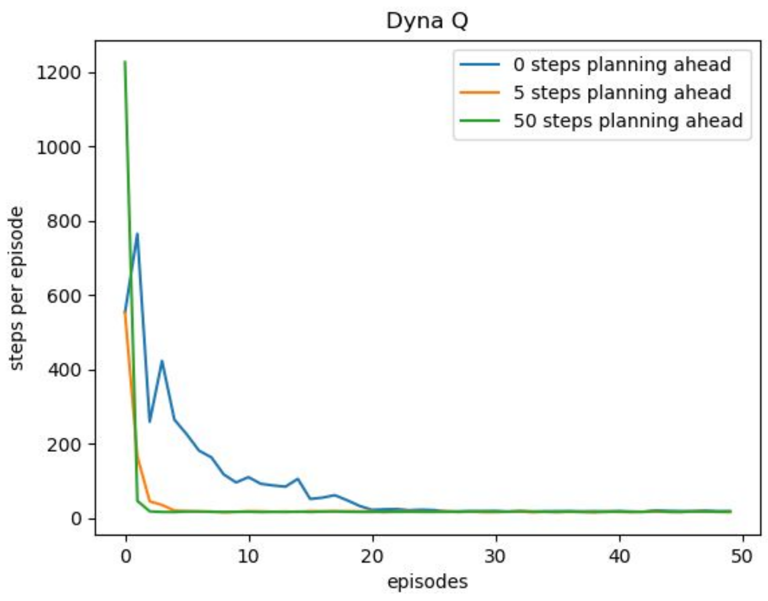

Figure 2. Tabular Dyna Q performance using different planning states

3.1 How the planning states affect learning performance?

In figure 1 three plots are presented; each of them was evaluated under the same simulation parameters but different step planning. From this fact, we can conclude the step planning speeds up the learning process. It means, for large the number of step planning fewer episodes are needed to find a state-action function. The main drawback for a large number of step planning is the computational time required.