Dynamic Planning in Dyna-Q for Faster Training

Aman Palariya **Rishabh Jain** **Tanmay Aeron**
Dept. of Computer Science and Engineering
Indian Institute of Technology Ropar, India
{2019csb1068, 2019csb1286, 2019csb1124}@iitrpr.ac.in

Abstract

Planning based Reinforcement Learning algorithms have excellent convergence guarantees. But the model of the environment is not available to us. So the only way to understand the environment is through experience. Dyna-Q uses both learning and planning to help train RL agents faster than just learning. It achieves this by building a model of the environment as it observes real state-action pairs. When the training starts, the model of the environment is not a good representative of the original environment; it improves as more state-action pairs are observed. In this paper, we propose ways to interlace planning with learning to achieve better training times.

1. Introduction

In a Reinforcement Learning problem, when the model of the environment is known, several methods like value iteration, policy iteration have been devised. They give near optimal results in very less training time. But in the real world, model of the environment is not known [3]. Most of the model-less Reinforcement learning algorithms focus on learning policy by learning optimal state-action value functions. One other way to learn optimal policy can be by learning model of the environment and then using model-based methods to learn model of the environment. One bottleneck with both these methods is the cost of interacting with the real environment.

Sutton[4] came up with the idea of integrating planning and learning. Sutton proposed Dyna-Q architecture in which he integrated planning and learning steps. In Dyna-Q architecture, model of the environment is also learnt. After every step of learning, state-action values are updated fixed number of time using (state, action, reward, next-state) generated by model of the environment. Model is learnt simultaneously with the learning steps.

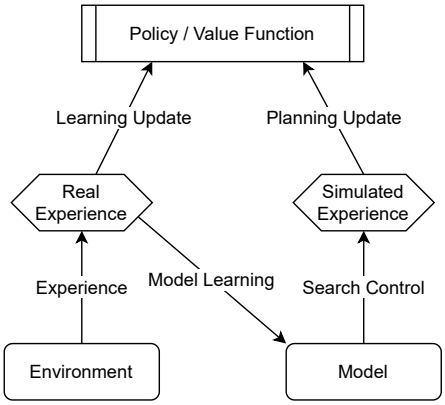


Figure 1: Dyna-Q Architecture

2. Related Works

Several improvements over Dyna-Q architecture have been proposed in the past. Moore and Atkeson[1] proposed updating state-action value functions based on the priority of the updates. Peng and Williams[2] came up with a similar architecture, namely Queue-Dyna. These methods improve training time by controlling the way the environment model is used to sample states for planning.

This, to our knowledge, is the first work which propose a schedule over number of planning steps.

3. Proposed Work

A naïve implementation of Dyna-Q would be to do fixed number of planning steps for each learning step as shown in algorithm 1.

Algorithm 1 Naïve implementation of Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in S$ and $a \in A(s)$
Initialize n (number of planning steps)
while not converged **do**
    $S \leftarrow$ current (non-terminal) state
    $A \leftarrow \varepsilon$-greedy(S,Q)
    Take action A; observe reward R and state S'
    $Q(S,A) \xleftarrow{+} \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    Add R, S' to $Model(S,A)$
    loop n-times
        $S \leftarrow$ random previously observed state
        $A \leftarrow$ random action previously taken in S
        $R, S' \leftarrow Model(S,A)$
        $Q(S,A) \xleftarrow{+} \alpha[R+\gamma \max_a Q(S',a) - Q(S,A)]$
    end loop
end while

Because the model of the environment will not be good initially, so the planning steps are not useful initially. Moreover they might result in a bias because the sampled states will not be a representative of true state distribution. Therefore, instead of keeping the number of planning steps (n_p) constant, we may vary them based on the number of learning steps (n_l) that our agent has seen so far.

$$n_p = f(n_l)$$

The function $f : \mathbb{Z} \to \mathbb{Z} \cup \{0\}$ is called the planning schedule. We expect the schedule to have the following properties

1. Non-decreasing function: The function should be non-decreasing because the model of environment gets better as more learning steps are taken. Therefore, the degree of planning should increase as the agent observes more state-action pairs.

2. Starts with a small value: The function should start with a small value because initially the model of environment is bad (contains only a handful of observations) and hence planning based on such a small number of observations will only be a waste of resources.

3. Does not return large values: The function should be clipped at a reasonable value. This is because even if the agent has seen enough observations, the model so created may still be biased and hence relying only on planning will be detrimental. Some amount of learning should always be a part of training.

Algorithm 2 Dyna-Q with varying planning steps

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in S$ and $a \in A(s)$
Initialize planning schedule f
$n_l \leftarrow 0$ (number of learning steps)
while not converged **do**
    $S \leftarrow$ current (non-terminal) state
    $A \leftarrow \varepsilon$-greedy(S,Q)
    Take action A; observe reward R and state S'
    $Q(S,A) \xleftarrow{+} \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    Add R, S' to $Model(S,A)$
    $n_l \leftarrow n_l + 1$
    $n_p \leftarrow f(n_l)$ (number of planning steps)
    loop n_p-times
        $S \leftarrow$ random previously observed state
        $A \leftarrow$ random action previously taken in S
        $R, S' \leftarrow Model(S,A)$
        $Q(S,A) \xleftarrow{+} \alpha[R+\gamma \max_a Q(S',a) - Q(S,A)]$
    end loop
end while

4. Experiment

We ran experiment for different schedules on `CliffWalking-v0` and `Taxi-v3` environment and studied the convergence and training properties of various planning schedules. For sake of brevity, we will be present results on the `CliffWalking-v0` environment in this paper. The experiments are run 10 times and the mean is observed to smooth irregularities.

4.1. Schedules

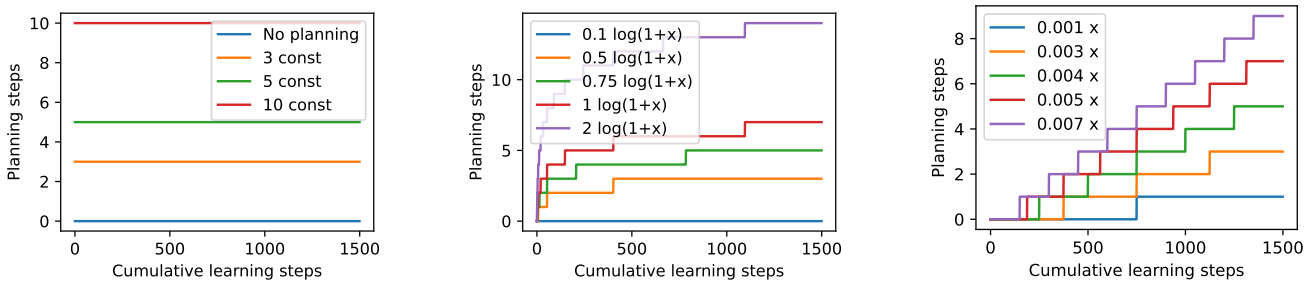
We evaluated 3 classes of schedules - constant, log, and linear. Exact schedules can be seen seen in fig. 2.

4.2. Agent's Performance Evaluation

To evaluate agent's performance, after every 50 learning steps, we used the current policy on the environment and collected the average reward over 100 episodes.

4.3. Schedule's Performance Evaluation

To quantify a schedule's performance, we calculated the cost incurred by the agent while training. Training is composed of learning and planning steps. Both types of steps have a non-zero cost associated with them. The cost may be financial, computational, or execution time. Let the ratio of cost of planning step to learning



(a) Constant schedules (b) Log schedules (c) Linear schedules

Figure 2: Different classes of schedules used for experiment

step be denoted by c. $c \neq 0$ because planning steps always incur cost in terms of execution time.

$$c = \frac{\text{Cost of planning step}}{\text{Cost of learning step}} \qquad 0 < c \leq 1$$

For different values of c, we observe the total relative cost incurred to train the agent.

We assume that the training has converged when the average reward is in 2% neighborhood of the best converged reward.

4.4. Results

Based on the convergence criteria the following schedule converge

Schedule	n_l	$\sum_t n_p$
5 const	1350	8100
10 const	1000	11000
$0.75 \log(1+x)$	1400	7342
$2 \log(1+x)$	1000	12330
$0.005x$	1450	6355

Table 1: Number of learning and planning steps taken by different schedules

One can observe from Table 1 that the schedules that have smaller n_l have larger $\sum_t n_p$ and vice-versa. This shows the trade-off between planning and learning as observed by Sutton [4].

Figure 3 clearly shows that agent without planning perform worse than agent with planning and learning. To further analyse how different schedules compare with each other, we plot relative cost of different schedules for cost ratios varying from 0 to 1.

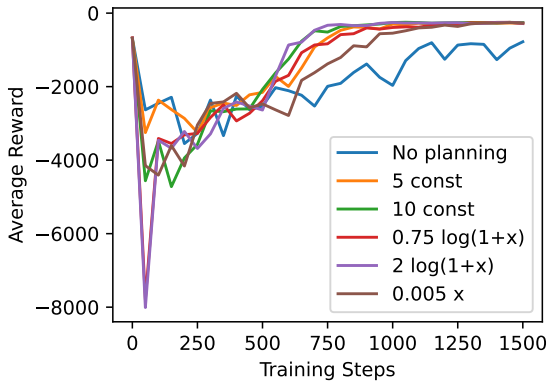


Figure 3: Performance of different schedules in `CliffWalking-v0` task

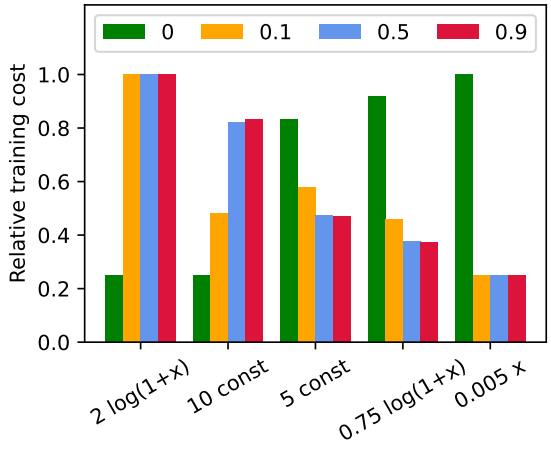


Figure 4: Relative training cost of schedules for different c

From fig. 4, we can see that for $c = 0$, constant schedules with large planning steps give lesser cost. This is because when $c = 0$, the planning cost is zero.

Therefore, doing more planning leads to earlier convergence without any planning cost.

But as the cost ratio increases, schedules $0.75 \log(1+x)$ and $0.005x$ start becoming cheaper. This happens because theses schedules saved cost when the environment model was not good enough by cutting on number of planning steps.

5. Conclusion and Future Work

Using dynamic number of planning steps, one can achieve faster training times than naïve implementations when planning steps have significant cost relative to learning steps. For larger environments, it might mean saving hours of training times and compute cost.

This technique of using dynamic number of planning steps can also be applied to advanced Dyna-Q algorithms like prioritized sweeping. The number of planning steps can also be changed based on number of distinct states observed, and the mean TD-error to make training even more cost-effective.

References

[1] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, Oct. 1993.

[2] J. Peng and R. J. Williams. Efficient learning and planning within the dyna framework. *Adaptive Behavior*, 1(4):437–454, Mar. 1993.

[3] C. R. Shelton. Reinforcement learning with partially known world dynamics, 2013.

[4] R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, July 1991.