Reinforcement Learning: Dyna-Q+

Mateo Guaman Castro

October 27, 2018

Abstract

In this homework assignment, two closely related versions of a Dyna-Q+ agent were implemented to learn the optimal policy for an agent navigating a maze. In this problem, the task is for the agent to get to the end of the maze, however, the maze changes after a certain number of time steps. Dyna-Q+ is a variant of Dyna methods, where direct reinforcement learning and planning are combined and used in parallel to get better results. The two implementations in this assignment differ in how the action values are modified to promote exploration. As shown in this homework assignment, combining direct reinforcement learning methods with planning achieves better results than either of these by themselves, and using Dyna-Q+ to encourage exploration is an effective way of making the agent more robust against changes in its environment. As the results of this assignment show, Dyna-Q+ methods perform better than Dyna-Q methods.

1 Background

In reinforcement learning, there are two types of learning an agent can do: learning with a model, and learning with experience. When the agent learns using a model, the states, actions, true rewards, and dynamics are known beforehand. These models are usually represented as Markov Decision Processes. Learning, in this case, is done simply by planning and estimating value functions exhaustively. So, in this context, the model is used to determine an optimal policy using planning. This type of learning is called model-based learning. The other type of learning is done using experience gathered from the agent's interactions with its environment. This learning can be done by updating action values directly from the experience received at each time-step using methods such as Q-Learning or Sarsa. In this context, real environment experience is used to determine action values and to learn the optimal policy. This type of learning is called model-free learning, as well as Direct RL. Although model-based and model-free learning may seem like contradictory ways of learning, they can be combined to lead to better results.

But how can model-based learning and model-free learning be combined if model-based assumes the existence of a model which might not be the case in the majority of situations? The answer is that the model can be learned from the experience the agent receives from the environment. As a bit of context to this question, a model is anything that the agent can use to predict how the environment will respond to the actions that the agent takes, mainly the

next state and the reward received from taking the available actions at each state. There exist two types of models, as mentioned by Sutton and Barto in [1]: distribution models, which exhaustively describe all possibilities and their probabilities, and sample models, which produce just one of the possibilities, sampled according to its probability. Coming back to the original question, the learned model will be a learned sample model, "sampled" from the real experience from the environment. This sample-based planning will then be used just to generate, or simulate, experience.

One architecture used to combine these model-based learning and model-free learning is Dyna. This architecture is introduced in [1] as a way to combine direct reinforcement learning, learn the model from real experience, and then learn and plan concurrently (or sequentially in a specified order) the value function or the optimal policy from both real and simulated experience. Figure 1 shows the Dyna architecture. In the planning stage, states and actions are sampled from previously encountered state-action pairs.

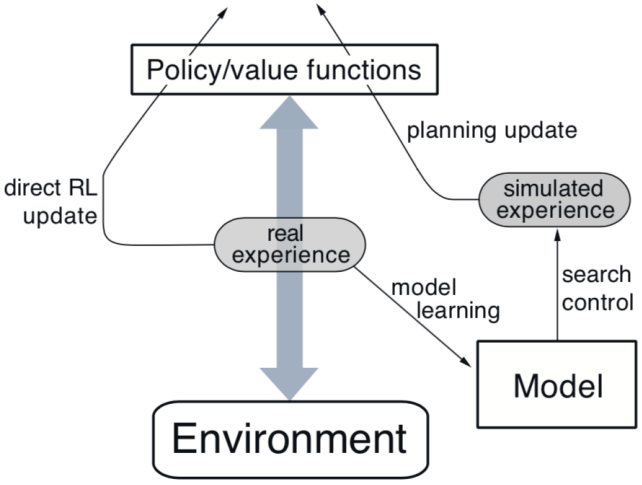

Figure 1: Dyna architecture. Model learning and simulation occur in parallel with direct learning. Search control refers to the way of selecting previously encountered actions in previously selected states, i.e. sampling from previous experience

Sutton and Barto show an algorithm for an implementation of the Dyna architecture called Dyna-Q, which can be shown in Figure 2. Dyna-Q uses Q-Learning to update value functions. This is a sequential implementation of Dyna, since planning and direct learning do not occur at the same time. One very important observation from this implementation is that both direct learning and planning utilize the same algorithm, namely one step Q-Learning, to update the value functions. The only difference is the source of the experience.

Since the model is learned on the fly, the agent may develop an incorrect model early on for a variety of reasons. The model may also be inaccurate if the environment changes over time, meaning the environment is non-stationary. The agent would then develop a sub-optimal policy. In order to encourage exploration to develop an accurate representation of the model, the Dyna architecture can be slightly altered by introducing an optimistic bias in the planning

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow \varepsilon$-greedy(S,Q)
- (c) Take 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) Loop repeat n times:
  - $S \leftarrow$ random previously observed state
  - $A \leftarrow$ random action previously taken in S
  - $R, S' \leftarrow Model(S,A)$
  - $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Figure 2: Algorithm for tabular, sequential Dyna

stage. That is, the agent predicts greater reward from certain actions than are actually possible in order to update the model to a more accurate version of itself as the agent gets more experience.

2 Motivation

Both planning and direct learning methods have advantages that make them favorable over each other, but instead of seeing them for their differences, these two types of methods can be combined to unite their strengths. Additionally, planning and direct learning can be run concurrently in the Dyna architecture, which could provide an efficient algorithm. In the big picture, combining planning with direct learning has lead to very impressive results, as is the case of Alpha Go Zero, developed by DeepMind [2]. In that case, Monte Carlo Tree Search is used. Therefore, combining direct learning with planning has been proven to be very powerful.

3 Experiment

3.1 Hypothesis

The first hypothesis is that Dyna-Q will do better than regular Q-Learning. That is, adding a certain number of planning steps to the agent will improve the performance of the algorithm. This is because the additional planning will increase the number of state-action pairs visited at any given time step, either from real experience or from simulated experience. Therefore, all of these visited state-action pairs will have updated value functions, instead of just one from direct learning. The agent will learn faster and will, then, obtain more reward. The second hypothesis is that Dyna-Q+ methods should perform comparably in non-stationary environments to Dyna-Q methods. However, in non-stationary environments, Dyna-Q+ will perform noticeably better than Dyna-Q methods.

3.2 Methodology

There are three parts to this experiment. The first one was demonstrating that Dyna-Q performs better than Q-Learning to motivate the exploration of the optimistic Dyna-Q+ methods. The second part of the experiment involved tuning the hyper-parameters of the agent to get a solid baseline to which the performance of the Dyna-Q+ models could be compared. Finally, the third part involved implementing two different versions of Dyna-Q+ to test their performance against regular Dyna-Q and to compare the performance between them.

The two Dyna-Q+ methods compared will be called in this assignment: Dyna-Q+ with bonus on reward, and Dyna-Q+ with bonus on action value. For Dyna-Q+ with bonus on reward, in the planning steps, the reward received from the model is given a "bonus" based on how long ago the action sampled at that planning step was selected in the real environment. This bonus takes a form of $r + \kappa\sqrt{(\tau)}$, where r is the reward obtained from the model, κ is a small number, and τ is the number of time steps since the action was last selected in the real environment. This corresponds to the third line under step (f) in Figure 2.

For Dyna-Q+ with bonus on action value, the action selected in step (b) of the algorithm in Figure 2 corresponds to the one with highest $Q(S_t, a) + \kappa\sqrt{(\tau(S_t, a))}$, while still being ϵ-greedy.

3.3 Implementation

The environment used for this homework assignment can be seen in Figure 3, from [1].

Figure 3: Baseline maze used to test single-step Q-Learning, Dyna-Q, and both Dyna-Q+ methods. S represents the starting position, and G represents the end of the maze. The darker squares represent a wall.

In all of these experiments, the states in the state space of the agent were represented as 2-element tuples:

$$S(t) = (row, column)$$

At any given state, the agent could select between four actions: *up*, *down*, $left$, *right*, each represented by a 2-element tuple. Therefore, the action space was, respectively:

$$A(t) = \{(1, 0), (-1, 0), (0, 1), (0, -1)\}$$

The agent obtained a reward of 0 for all states that were not the goal state, and a reward of 1 when it got to the goal state. As for the dynamics of the environment, if the agent hit a wall of the boundary of the maze, the agent would stay in the same state and receive a reward of 0. Otherwise, the agent would move according to the action selected.

For the first part of the experiment, regular one-step Q-learning was compared to Dyna-Q learning with 8 planning steps in each time step.

As for parameters used, the parameters used for Dyna-Q, Dyna-Q+ with bonus on reward, and Dyna-Q- with bonus on action value, were: $\epsilon = 0.3$, $\gamma = 0.95$, $\alpha = 0.7$, $\kappa = 0.001$, $n = 8$. In all of these experiments, the agent ran for 5000 steps, and the results were averaged over 30 runs.

Two more mazes were used on top of the baseline maze in order to compare Dyna-Q and Dyna-Q+ methods. Two different tests were performed. In the first one, the baseline maze was used for the first 2000 iterations, but after that, the maze changed to the one in Figure 4. This was done to test how the agent adapted to drastic changes in the environment. In the second test, the first 2000 iterations were run on the maze in Figure 4, but after that, the maze changed to the one in Figure 5, where a new opening shows up that gives a better possible path to the agent.

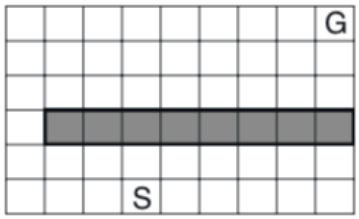

Figure 4: Maze used to test the adaptability of the algorithms to drastic changes in the environment

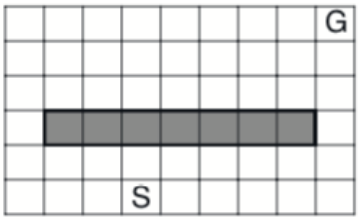

Figure 5: Maze used to test the adaptability to subtle changes that lead to better paths

4 Results and interpretation

4.1 Single-step Q-Learning vs Dyna-Q

In this section, the increase in performance of the Dyna-Q method over Q-Learning is very significant. The results can be seen in the plot in Figure 6. It

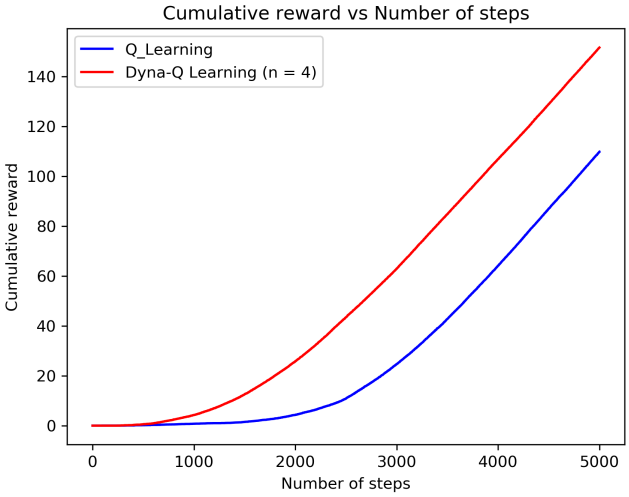

Figure 6: Q-Learning vs Dyna-Q

suffices to say that it is clear that having planning steps in the learning algorithm increases the performance of the algorithm. This is because any number of planning steps will result in the randomly (but previously-chosen) selected action's values to be updated, which will create a more accurate representation of the environment, and therefore the agent will learn faster.

4.2 Dyna-Q vs Dyna-Q+ (bonus on reward) vs Dyna-Q+ (bonus on action value)

4.2.1 Drastic changes in environment

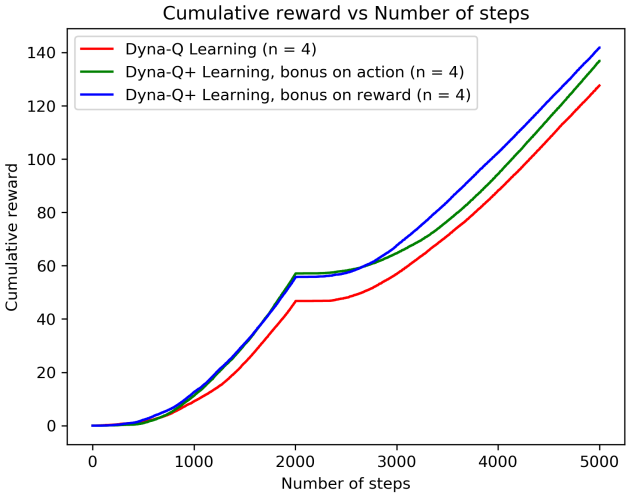

Figure 7: Drastic change in environment from Figure 3 before time step 2000, to Figure 4 until the end

In this case, the maze was changed after 2000 time steps from the one in Figure 3 to the one in Figure 4. The change in these mazes is pretty drastic, which means that the agent had to spend some time to find out another way to get to the end of the maze. As it can be seen in Figure 7, the two Dyna-Q+ algorithms do better than the regular Dyna-Q algorithm. This is understandable, since the boost in exploration of the Dyna-Q+ algorithms will lead to the agent discovering new paths faster. In this case, the Dyna-Q+ method with bonus on the reward performed better than the one with bonus on the action value. The difference in performance of the two Dyna-Q+ seems somewhat arbitrary, but this will be discussed later on in more detail.

4.2.2 Subtle changes in environment

In this experiment, the maze was changed after 2000 time steps from the one in Figure 4 to the one in Figure 5. The change in these mazes is rather subtle, in that a new opening appears in the wall, that will lead to a better path to the end of the maze. However, since the original path the agent learned does not disappear, there is not much motivation for the agent to change its policy in the Dyna-Q method. However, since the agent explores more using the two Dyna-Q+ methods, these methods eventually start diverging from the Dyna-Q performance and start to get a better reward, since the agent has learned about the new, more efficient, path. Again, the difference in performance between the two Dyna-Q+ methods is not very significant, but this will be explored more later on.

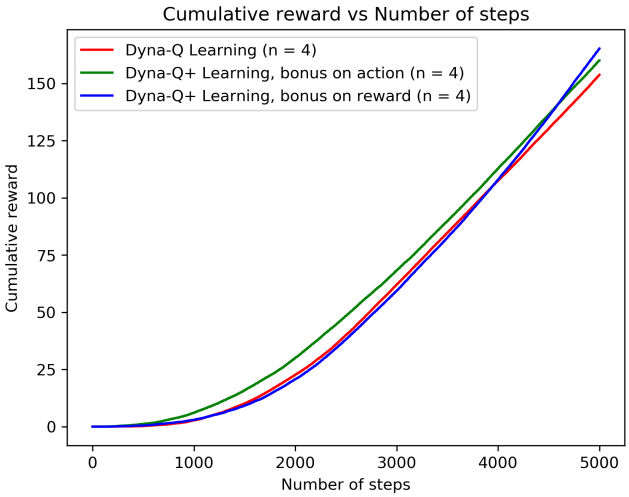

Figure 8: Subtle change in environment from Figure 4 before time step 2000, to Figure 5 until the end

4.2.3 Parameters

There are a few interesting results that were observed when looking for optimal parameters for the learning agents. To begin with, the parameters are very finicky. Very minute variations in some of these parameters would lead to drastic

changes in the performance of the algorithms. For instance, a difference in κ from 0.001 to 0.003 would cause the performance of one of the Dyna-Q+ methods to get significantly reduced.

Epsilon needs to be high enough for the agent to explore the environment in its initial stages, since no amount of planning will come up with values for states that have not been visited in this case, so epsilon controls how many new states are selected in the real environment. As for γ, it was kept to a value of 0.95 for all tests, but it just controls the reward discount in Q-Learning. Next, α, the learning rate, was chosen to be 0.7 empirically. Finally, κ, which is used to boost exploration in the Dyna-Q+ agents, must be low so that the exploration done is not overwhelming to the performance of the agent, but should not be too low as to become negligible and therefore just another version of Dyna-Q.

4.2.4 General Discussion

In general, from the results obtained in the previous experiments, it is clear that both Dyna-Q+ methods perform better than Dyna-Q. However, both methods also seem to have very comparable results. The way Dyna-Q+ with bonus on the reward works is that it actually boosts the action values of those actions the agent selects, in order to make the agent take this actions in the real environment to find out their real values. In the other hand, Dyna-Q+ with bonus on the action values does not modify the values of the actions, but rather it just updates the value-functions to a more recent estimate. Intuitively, the second Dyna-Q+ algorithm should perform better than the other one, but the evidence from the results do not firmly support this claim. It could have something to do with how much the maze itself and the hyperparameters affect the performance of these algorithms. What can be said, however, is that the two Dyna-Q+ methods perform comparably to each other, and, with the right parameters, they can perform better than Dyna-Q.

5 Conclusion

In this homework assignment, the convergence properties of Monte Carlo estimation methods were experimentally proven by using an on-policy Monte Carlo control method with ϵ-greedy as its soft policy. This was done by having an agent learn the fastest way to navigate a racetrack given some restrictions from the environment. Additionally, different configurations of the agent's parameters as well as the environment of the agent were explored in order to get a better intuition of the positives and negatives of Monte Carlo methods.

In this homework assignment, planning algorithms and direct learning algorithms were combined using an architecture called Dyna and a specific method called Dyna-Q to produce better results than stand alone direct learning methods. Additionally, two different versions of Dyna-Q that promote exploration, called Dyna-Q+, were explored to determine the difference of performance of these three different methods. In the end, it can be said that Dyna-Q+ methods perform better than Dyna-Q methods, and they perform comparably to each other.

References

[1] Sutton, Richard S., and Andrew G. Barto. *Reinforcement Learning: an Introduction.* 2nd ed., The MIT Press, 2018.

[2] Silver, David, et al. "Mastering the Game of Go without Human Knowledge." Nature, vol. 550, no. 7676, 2017, pp. 354–359., doi:10.1038/nature24270.