A SHORT TUTORIAL ON REINFORCEMENT LEARNING

Review and Applications

CHENGCHENG LI & LARRY PYEATT
Computer Scinece Department, Texas Tech University, Lubbock, Texas, 79401 USA

Abstract: Dynamic Programming (DP) has been widely used as an approach solving the Markov Decision Process problem. This paper takes a well-known gambler's problem as an example to compare different DP solutions to the problem, and uses a variety of parameters to explain the results in detail. Ten C++ programs were written to implement the algorithms. The numerical results from gamble's problem and graphical output from the tracking car problem support the conceptual definitions of RL methods.

Key words: Reinforcement Learning, Dynamic Programming, Monte Carlo method, Temporal Difference, Markov Decision Process

1. INTRODUCTION

The three fundamental classes of RL methods are dynamic programming (DP), Monte Carlo (MC) and temporal difference (TD) methods. Almost all the RL methods are evolved from or from the combinations of these fundamental RL methods.

Dynamic Programming (DP) is one of the major methods used to solve MDP(Markov Decision Process) problems (Puterman, p.81). The policy iteration method is the straightforward way to apply DP to an MDP problem. A random policy or any non-optimal policy is evaluated by calculating the state values using the Bellman equation. A new policy replaces the old policy and is regarded as a more optimal policy than the old one, based on the calculated state values. The loop then keeps going on until either policy or state values converge and do not change. Value iteration is actually an improved version of policy iteration. It truncates the policy improvement

process in policy improvement and still keeps a guaranteed convergence toward the optimal solution.

The basic idea of MC methods is based on averaging sample returns. If a state or state-action pair has been visited many times, the average value of the returns of these visits is supposed to converge to the true value of this state or state-action pair. Based on this assumption, MC methods only consider and use experience. MC methods do value evaluation and improvement simultaneously.

TD methods combine the advantages of both MC and DP methods. TD methods do not require a complete knowledge of the environment. However, TD methods use bootstrapping which is similar to DP methods. On the other hand, TD methods do not go through the whole episode. In stead, they use a number of steps in each episode. The most commonly used on-policy TD method is SARSA, and off-policy TD method, which evaluates one policy while following another, is Q-learning method.

2. GAMBLER'S PROBLEM

A classic Gambler's problem is used to show a DP solution to a MDP problem. The description of the problem is as followings: "A gambler bets on the outcomes of coin flips. He either wins the same amount of money as his bet or loses his bet. Game stops when he reaches 100 dollars, or loses by running out of money." (Sutton and Barto, p.101)

Figures 1 and 2 show the value function and optimal policy when p=0.4, where p is the winning rate. The state value function is no longer linear. There are some local maximums when the capital reaches 25, 50, and 75. Fifty is the state with the most abnormally high state value. The policy is to reach the high value states before reaching the goal.

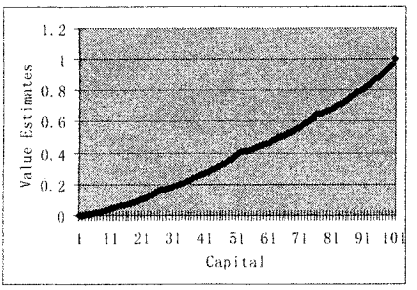

Figure 1. Optimal State Value Functions (p=0.4)

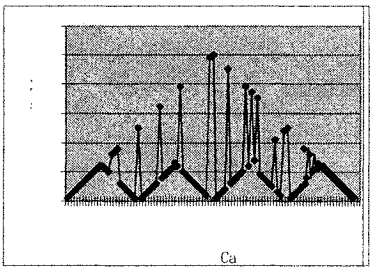

Figure 2. Optimal Policy (p=0.4)

There are many spikes in the graph shown in Figure 2, and they are considered noise. We found that it is caused by roundup errors of the floating point limitation of CPUs. Different errors are shown on AMD, Pentium, and

Sun CPUs. The computer makes the wrong decision due to the floating point limitation by truncating the number after the 16th digit. The solution to this problem is that an updated policy must increase the action value by at least the smallest decimal number which the CPU can handle, for example, 10^{-16} for double precision. By applying this concept, and changing the policy improvement rule from

" If (action_value > previous_max_value)
Then choose this action" to

" If (action_value > previous_max_value+10E-16)
Then choose this action ," all the noise is eliminated. The optimal policy graphs for $0<p<0.5$ are all the same. Another way to explain this is that because of the nonlinear property of the state value function, there is a certain amount of bet which can result in obtaining an optimal return for each state before betting all the capital. The exception to this is when current states are local maximums (25, 50, 75).

The local maximum is the smallest integer value divisible by a polynomial of two from the number of states. The reason is that the gambler problem is a discrete MDP problem, and every state has an integer value as capital. Keeping this in mind, let us further compare the results in Figures 3 and 4. Figure 3 has 1,024 states, which is a polynomial of two. So the local minimum is as small as 1. The result of this is that if a player sets a slightly different goal, for example, from 1,023 to 1,024, there may be completely different optimal policies for different goals, as can be seen in Figure 4, which has 1,023 states.

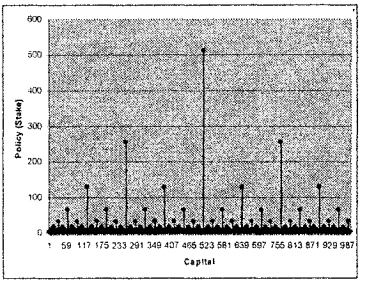

Figure 3. Optimal Policy ($0<p<0.5$), 1,024 States

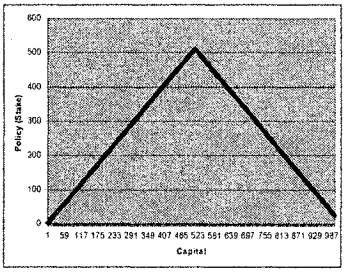

Figure 4. Optimal Policy ($0<p<0.5$), 1,023 States

3. TRACKING CAR PROBLEM

The tracking car problem is very special which could be solved by using MC, DP, and TD methods. Thus, the tracking car problem can be used to compare the three methods. The specification of tracking car problem is as follows. "Find the best strategy to accelerate or decelerate a racing car at a turning track. The maximum velocity of the car is 5. A random accelerate on either vertical or horizontal direction may occur." (Sutton and Barto, p.127)

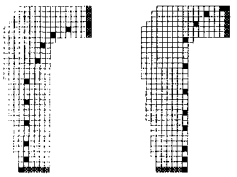

Figure 5. Optimal Tracks from On-Policy MC algorithm.

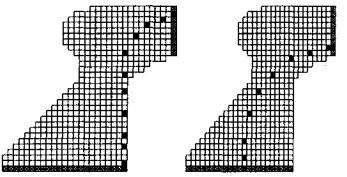

Figure 6. Optimal Tracks from DP Value Iteration algorithm.

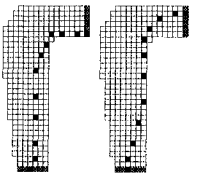

Figure 7. Optimal Track from SARAS and Q-learning algorithms.

There are more states defined in DP algorithm for this tracking car problem because car positions on track with a particular vertical and horizontal speed are counted as factors of states. There are four horizontal and four vertical speed, and they can not both be zero. So, the number of states is calculated by

$$NumberOfStates=OnTrackPositions*(4*4-1) \quad (1)$$

4. DISCUSSION ON RESULTS AND CONCLUSIONS

MC methods have a very close relationship to TD methods, which are also based on experience and sample episodes but bootstrap. If we increase the number of steps in the TD methods till the end of every episode, MC may be viewed as a special case of TD methods. MC methods are based on experience and sample episodes. They do not require a full definition of the environment, which is the major advantage of MC over DP methods. MC methods do not bootstrap and convergence of state or action values is guaranteed by a large number of visits on every state or action.

In practice, we found that correct setting of initial value of every state is critical for the correct convergence of state or action values. For example, we set that every state has been visited 45 times and every visit has a return of 1 before the loop starts. As a result, all states values are set much higher than their real values. The first several iterations of loop then would not set states to very low values. We then guarantee values of states slowly get close to their true values from only one side of their true values, so that large number of visits on every state are guaranteed.

Choosing greedy actions to update action values makes Q-learning an off-policy TD method, while SARSA is an on-policy TD method which uses e-greedy method. Figure 7 shows that SARSA intends to take a saver track and avoid being on the most left and the most top lanes of the track because the probability of being off the track is high on those lanes. Q-learning then chooses more greedy tracks than SARSA. If there is no or low-probability of random left and right accelerations, the results from Q-learning would be more close to the real optimal solution. If the probability of random accelerations is high, SARSA is the method to use. In most cases, greedy methods are more close to the real optimal solutions.

This study covers the major concepts of reinforcement learning and the Markov decision process. DP, MC, and TD methods are discussed and compared through solving problems. A noise reduction method and computer floating point limitations are also discussed. Ten C++ programs were written to implement the algorithms. The numerical results from gamble's problem and graphical output from the tracking car problem support the conceptual definitions of RL methods.

REFERENCES

1. Richard S. Sutton and Andrew G. Barto (1998), Reinforcement Learning: An Introduction, *MIT Press*, Cambridge, MA.
2. Martin L. Puterman (1994), Markov Decision Processes: Discrete Stochastic Dynamic Programming. *A Wiley-Interscience Publication*, New York.